How to Retrain Recommender System?
A Sequential Meta-Learning Method

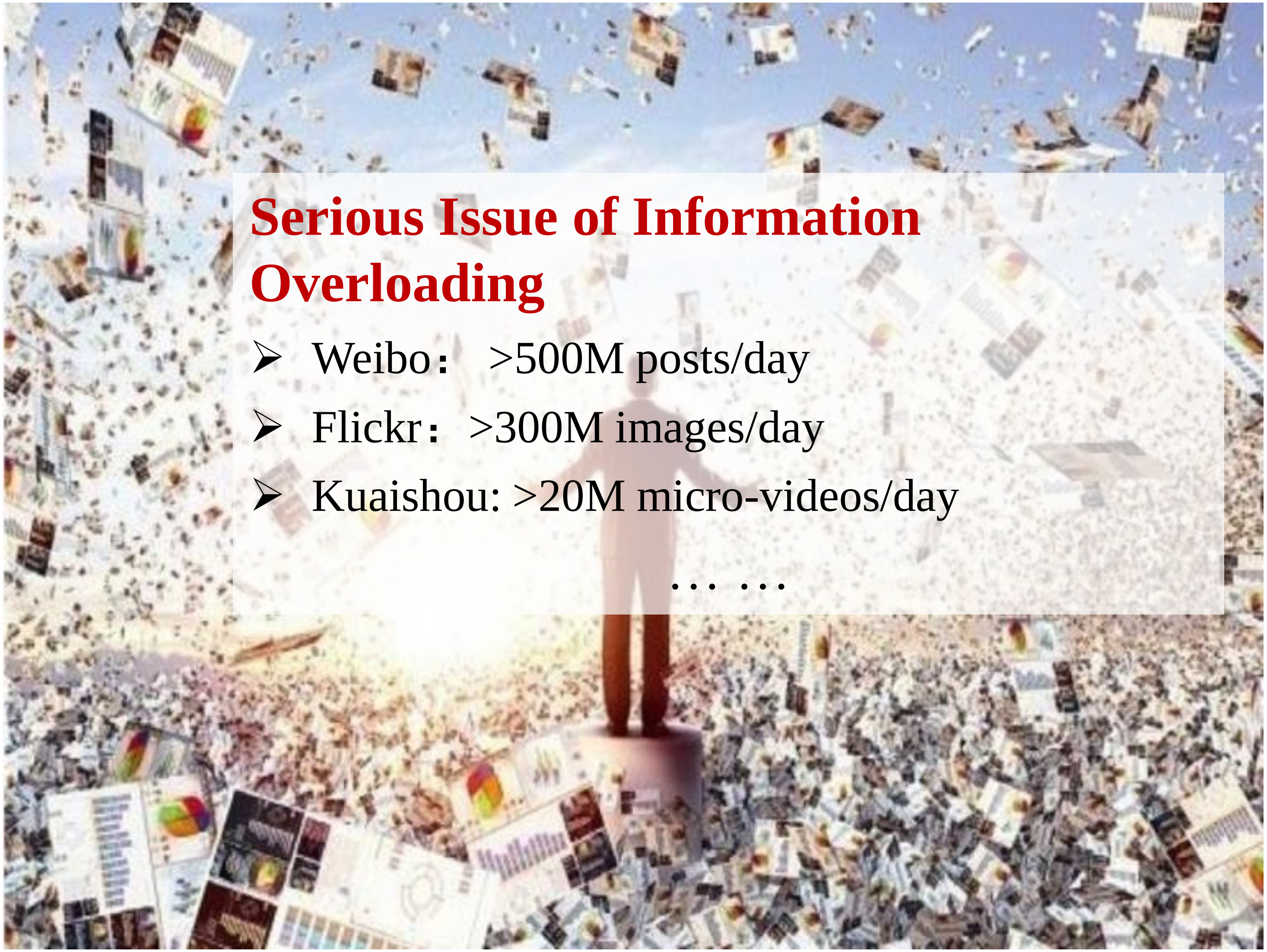
Yang Zhang, Fuli Feng, Chenxu Wang, Xiangnan He,
Meng Wang, Yan Li, Yongdong Zhang

University of Science and Technology of China， National University of Singapore
Hefei University of Technology, Beijing Kuaishou Technology Co., Ltd. Beijing, China

Outline

- Introduction
- Our solution
- Experiments
- Conclusion

Age of Information Explosion



Serious Issue of Information Overloading

- Weibo： >500M posts/day
- Flickr： >300M images/day
- Kuaishou: >20M micro-videos/day

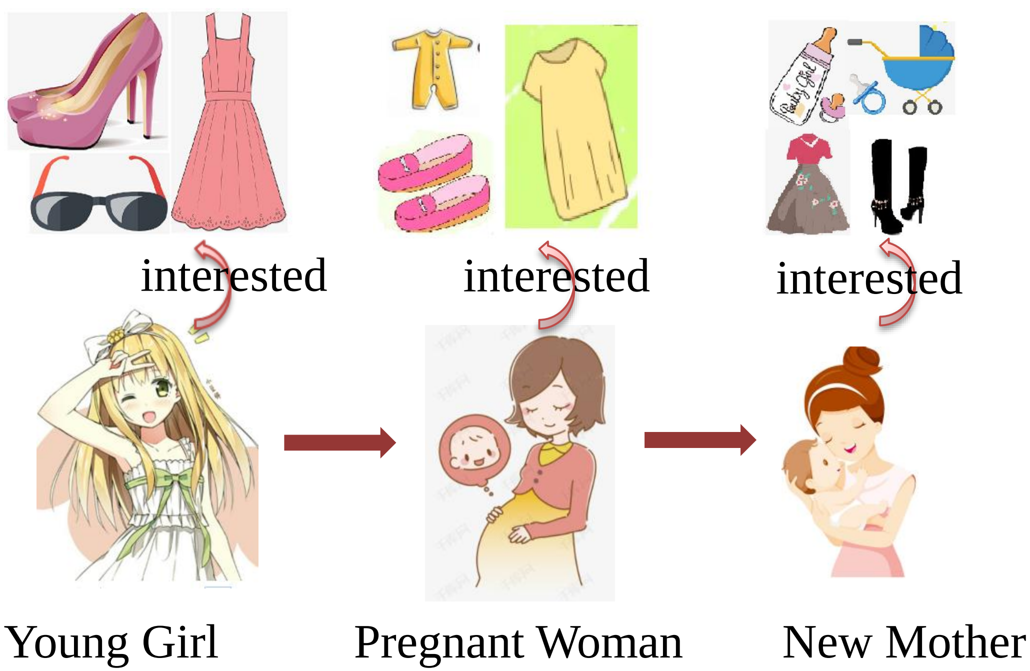
… …

How does a RecSys Work?

- Working :

Collect/Clean … Data → Offline training & Saving Model → Serve Online

With time goes by, Model may serve more and more bad. Because:



(1) User Interests drifts.

long- and short- term interest!

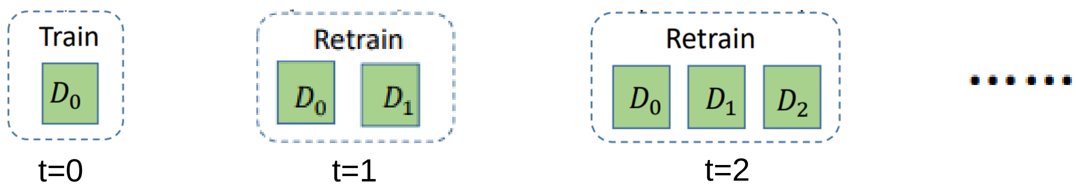
(2) New users/items are coming…

=> Solution: train the model again with new collected data.(i.e. retrain)

Full retraining and Fine tuning

- **method 1 -- Full retraining**

  Use all previous and new collected to retrain model. (initialed by previous model)



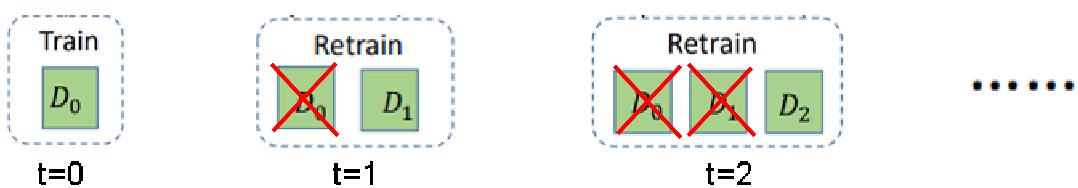
  pros: In some case, more data may reflect user interests more accurately

  cons: Cost highly (memory and computation) ;

  Overemphasis on previous data. (proportion of the last two datasets: t=1: 100% t=9: 20%)

- **Method 2 – Fine tuning:**

  In each period, only use new collected data to retrain/adjust previous model.



Pros: fast, low cost (memory, computation)

Cons: overfitting and forgetting issue (long-term interest)

Sample-based method

- Method 3 – sample-based methods:

  full-retraining: slow, high cost, ignore short-term interest

  Fine tuning: Fast, forgetting long-term interest

  trade-off : Sample previous data: long-term interest

  New data: short-term interest

  SPMF:

  each period:

  - Step 1: Use the sampled previous data and new data to retrain model
  - Step 2: Update the Reservoir (With some P)

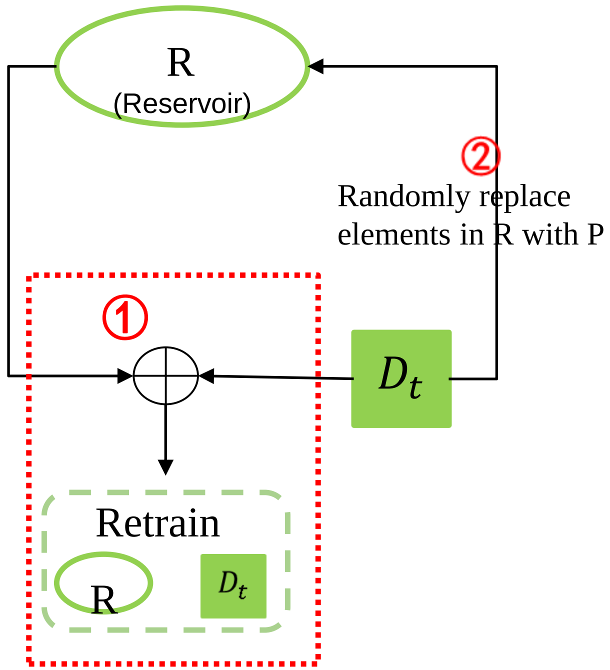
  Pros: A trade-off between Full-retraining and Fine-tuning.

  long- and short-term interest

  trade-off between cost and performance

  Cons: Not best performance

  Human-defined sampling strategies



- Other methods: memory-based methods

Motivation

- Common limitation of existing methods:

lack an explicit optimization towards the retraining objective — i.e., the retrained model should serve well for the recommendations of the next time period

Problems： one stagey only work well in one scenario, and bad in other scenarios
such as , one sample stagey can't be suitable for all recommendation scenarios.

This motivates us to design a new method can add the retraining objective optimization process. Meanwhile, we want to avoid to save previous to save long-term interest, in order to realize efficient retraining.

- PROBLEM FORMULATION:

Original form:

Previous model New model

$$(\{D_m : m \le t\}, W_{t-1}) \xrightarrow{get} W_t \xleftarrow{test} D_{t+1}.$$

Form with our goal: (constrained form)

$$(D_t, W_{t-1}) \xrightarrow{get} W_t \xleftarrow{test} D_{t+1},$$

Only new data can be used, and the goal is that perform well in the next period.

Outline

- Introduction
- Our solution
- Experiments
- Conclusion

Our solution -- key idea

- Previous model (parameters) has captured most information of previous data.

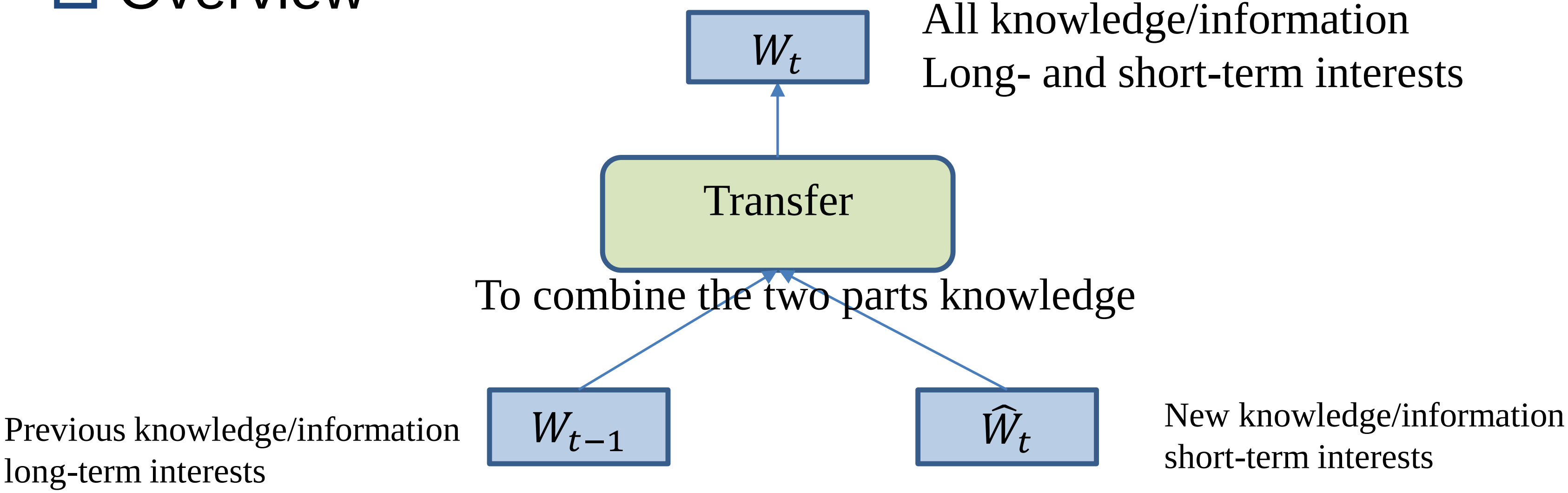
  So, save previous model instead of previous data.

- During training, consider utilize future data in some way!

  → Get a meta-model

Our solution -- framework

Overview



$\boldsymbol{W_{t-1}}$: Previous model, contains knowledge in the previous data

$\boldsymbol{\widehat{W}_t}$: A recommendation model to capture new information in the new collected data

Transfer: to combine the "knowledge" contained in W_{t-1} *and* $\widehat{W}_t$. Denote as f_Θ.

$\boldsymbol{W_t}$: new recommender model. $W_t = f_\Theta(W_{t-1}, \widehat{W}_t)$

We need:

A well-designed Transfer has the above ability.

A well-designed training process to make all the modules works.

Transfer

☐ Two simple methods

(1) pay different attentions to previous and current trained knowledge

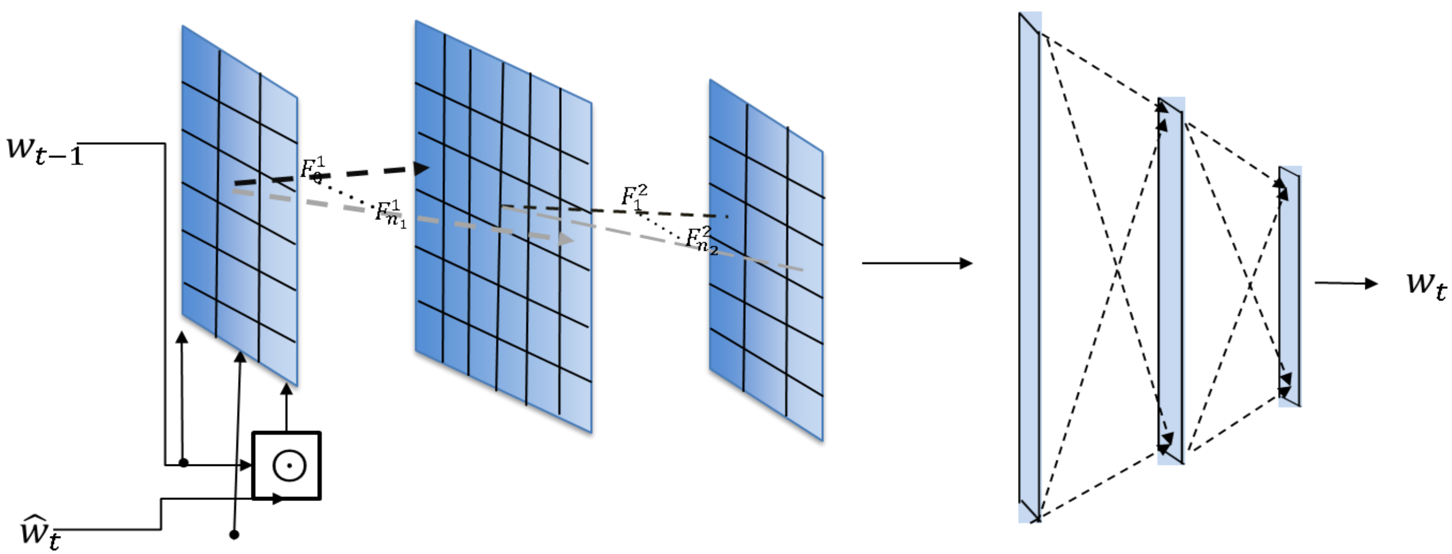
$$W_t = \alpha W_{t-1} + (1-\alpha)\,\widehat{W}$$

Cons : limited representation ability; the relations between different dimensions

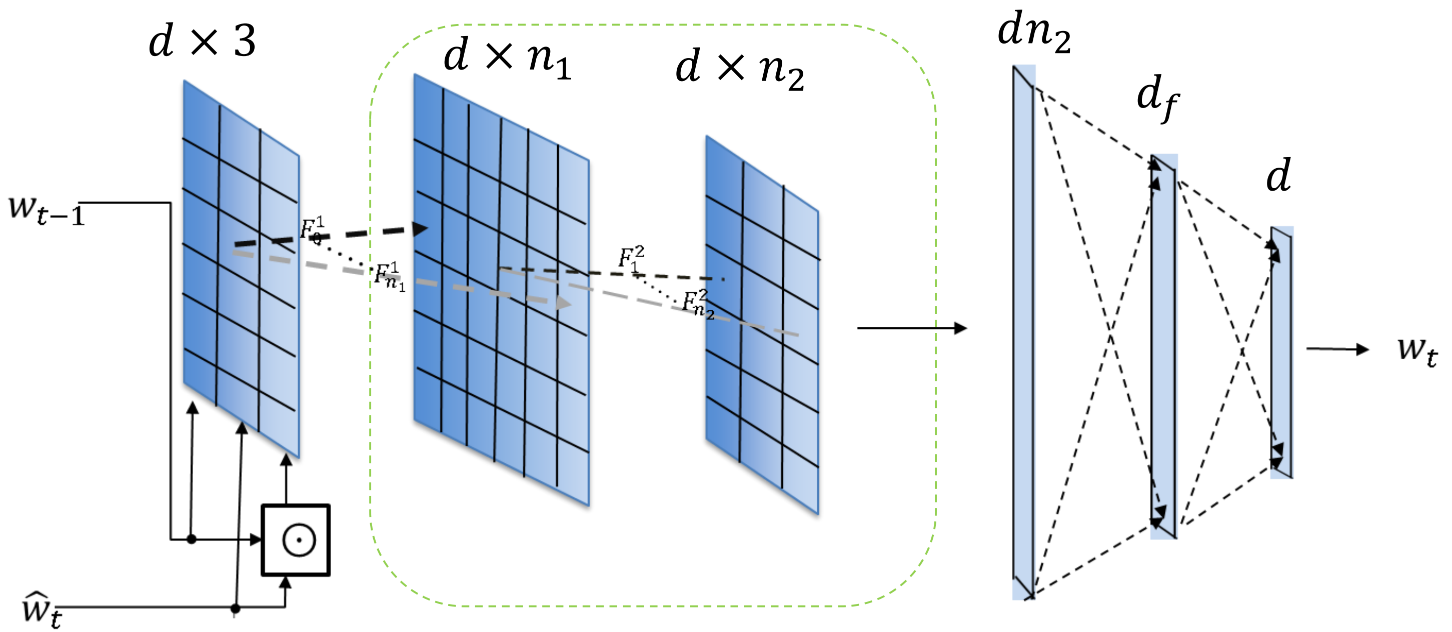
(2) MLP: $W_t = MLP(W_{t-1}||\widehat{W_t})$

Cons: not emphasize the interactions between the parameters of the same dimension

☐ Our solution -- A CNN-based Neural Networks



Transfer



Stack layer

CNN layers

FC and output layer

Stack $w_{t-1}, \widehat{w}_t$ and $\frac{w_{t-1} \odot \widehat{w}_t}{\alpha}$ to form a 2D picture

If w is not a vector, we can shape it to a vector.

- **1d and horizontal** convolution.
- Capture the relation between the parameters of the same dimension ---- **parameter evolution**

[-1,1,0] $\widehat{w}_t - w_{t-1}$ $(MF: interest\ drif)$
[0,0,1] interest vanish or appear
With the second layer, stronger ability

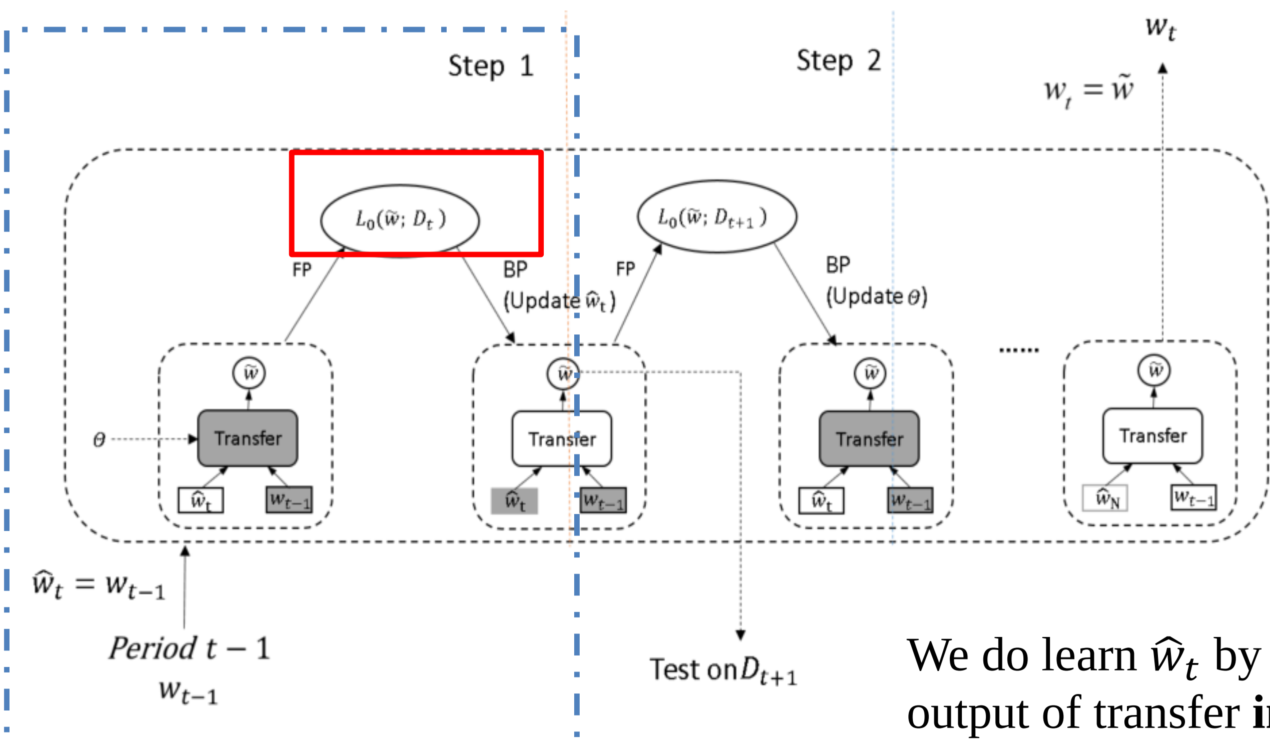
- Capture the relation between the parameters of the same dimension
- Recover the shape

Sequential Meta Learning

- Directly train our model, only make fit to current period, and even only make transfer focus on $\hat{w}_t$.

  -- to make transfer works, we proposed **Sequential Meta Learning(SML)**

Each period t, alternately update transfer and current model $\hat{w}_t$. *Each round, there are two main step*



Black blocks are fixed!

Step 1: learning $\hat{w}_t$

Goal : learn the knowledge in new collected data D_t

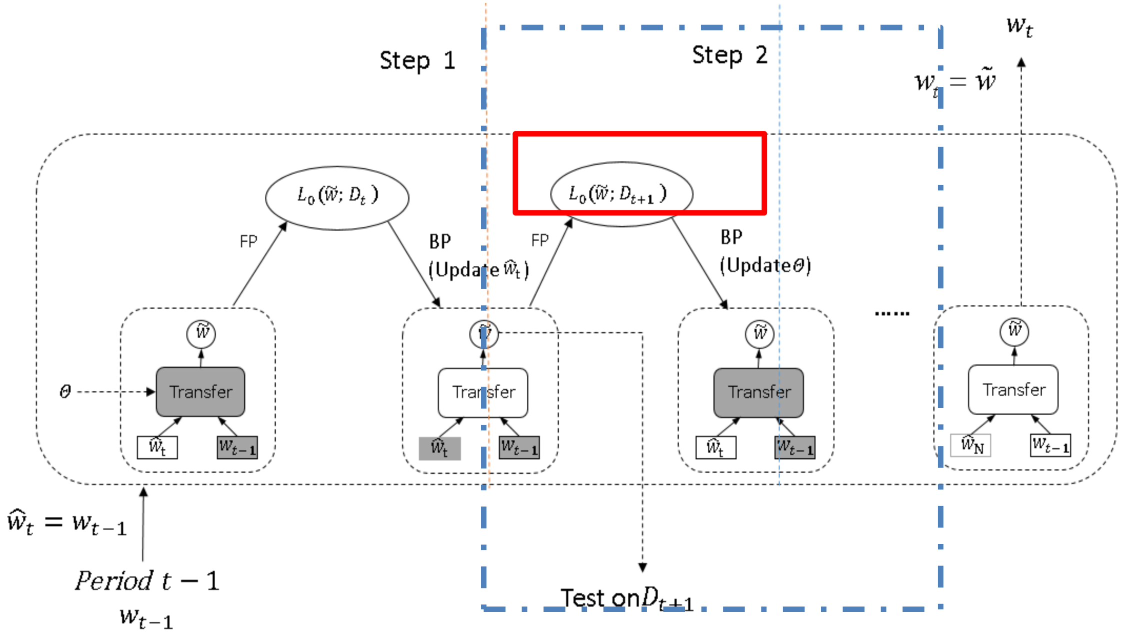
Fix the transfer, minimize:

$$L_r(\hat{W}_t|D_t) = L_0(f_\Theta(W_{t-1}, \hat{W}_t)|D_t) + \lambda_1||\hat{W}_t||^2$$

We do learn $\hat{w}_t$ by recommendation loss of the output of transfer **instead of itself loss** to make $\hat{w}_t$ suitable **to as the input of transfer**.

Sequential Meta Learning

Black blocks are fixed!



Step 2: learning the transfer

Θ : *shared across all tasks*

capture some task-invariant patterns

Goal: obtain such patterns that are tailored for the next-period recommendations

So, fixed $\hat{w}_t$, minimize :

$$L_s(\Theta|D_{t+1}) = L_0(f_\Theta(W_{t-1}, \hat{W}_t)|D_{t+1}) + \lambda_2||\Theta||^2,$$

i.e. recommendation loss on D_{t+1}

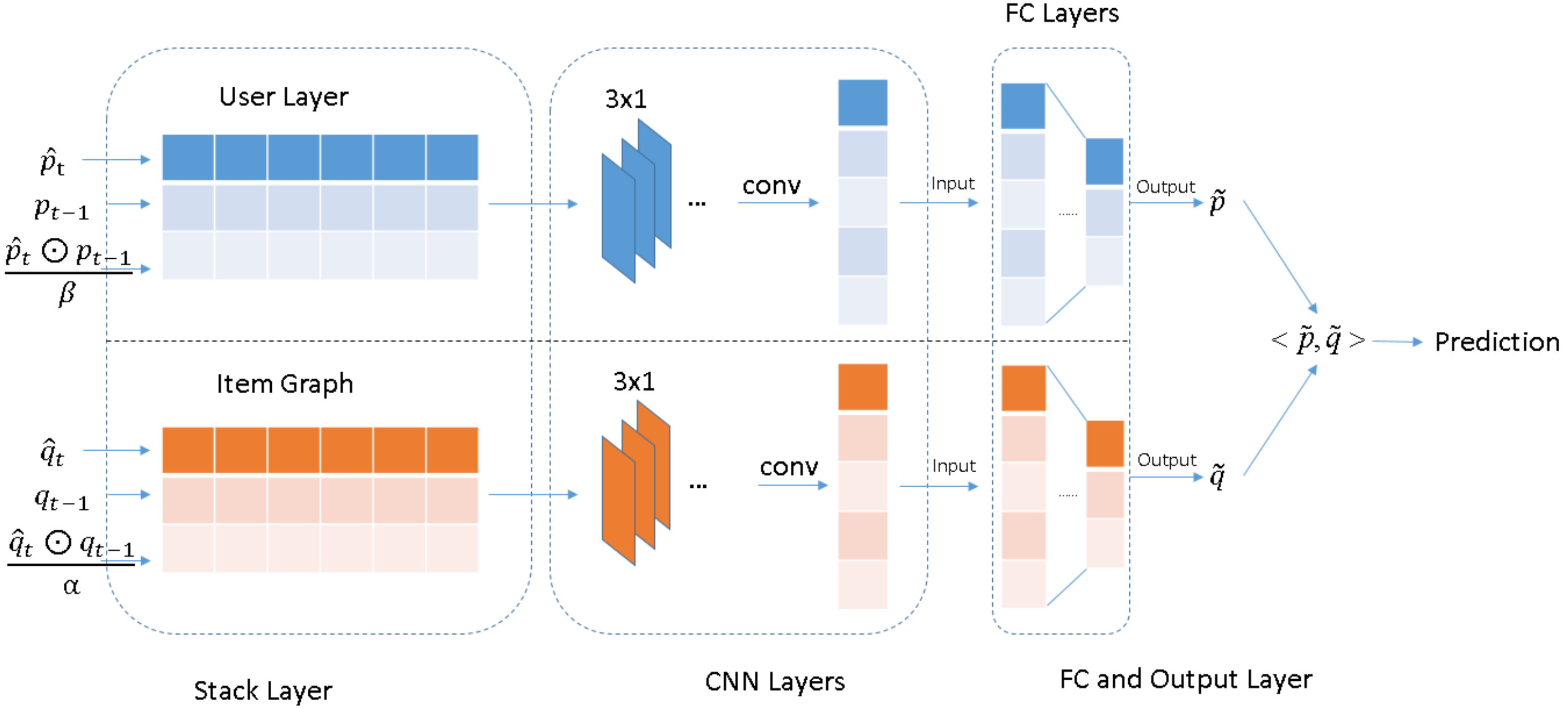
Then, the above two steps are iterated until convergence or a maximum number of iterations is reached.

Note that:

(1) we can run multiple passes of such sequential training on $\{D_t\}_{t=0}^T$ when offline training , to learn a better uniform transfer.

(2) When serving/testing , use the model gained before the first step 2.

Instantiation on MF



All users share a transfer, all item share another transfer.

Loss:

$$L_0(P, Q|D_t) = -\sum_{(u,i)\in D_t} \log(\sigma(\hat{y}_{ui})) - \sum_{(u,j)\in D_t^-} \log(1 - \sigma(\hat{y}_{uj})),$$

Outline

- Introduction
- Our solution
- Experiments
- Conclusion

Setting

- Datasets:

Dataset	Interactions	users	item	time span	Total periods
Adressa	3,664,225	478,612	20,875	three weeks	63
Yelp	3,014,421	59,082	122,816	> 10 years	40

Adressa[1]: (1) News clicked data. time-sensitive, choose recent news , short-term interests.
(2) split each day into three periods:
morning (0:00-10:00), afternoon (10:00-17:00) evening (17:00-24:00)

Yelp[2]: (1)users and businesses like restaurants. inherent (long-term) interest
(2)split it into 40 periods with an equal number of interactions

- Data splits: offline-training/validation/testing periods:
  Adressa: 48/5/10 Yelp: 30/3/7
- Evaluation: (1) done on each interaction basis[3].
  (2) sample 999 non-interacted items of a user as candidates
  (3) Recall@K and NDCG@K (K=5,10,20)

[1]. Jon Atle Gulla et.al. 2017. The Adressa dataset for news recommendation. In WI
[2] https://www.yelp.com/dataset/
[3] Xiangnan He et.al. 2017. Neural collaborative filtering. In WWW

Performance

- Average performance of testing periods

Datasets	Methods	recall@5	recall@10	recall@20	RI	NDCG@5	NDCG@10	NDCG@20	RI
Adressa	Full-retrain	0.0495	0.0915	0.1631	319.7%	0.0303	0.0437	0.0616	393.1%
	Fine-tune	0.1085	0.2235	0.3776	82.8%	0.0594	0.0962	0.1351	135.5%
	SPMF	0.1047	0.2183	0.3647	87.3%	0.0572	0.0935	0.1306	143.6%
	GRU4Rec	0.0213	0.0430	0.0860	809.0%	0.0125	0.0194	0.0302	1018.4%
	Caser	0.2658	0.3516	0.4259	6.5%	0.1817	0.2096	0.2285	2.1%
	SML	**0.2815**	**0.3794**	**0.4498**	-	**0.1838**	**0.2156**	**0.2336**	-
Yelp	Full-retrain	0.1849	0.2876	0.4139	18.0%	0.1178	0.1514	0.1829	22.7%
	Fine-tune	0.1507	0.2386	0.3534	41.7%	0.0963	0.1246	0.1535	48.5%
	SPMF	0.1664	0.2591	0.3749	30.7%	0.1072	0.1370	0.1662	35.1%
	GRU4Rec	0.1706	0.2764	0.4158	22.8%	0.1080	0.1420	0.1771	30.5%
	Caser	0.2195	0.3320	0.4565	2.8%	0.1440	0.1802	0.2117	3.12%
	SML	**0.2251**	**0.3380**	**0.4748**	-	**0.1485**	**0.1849**	**0.2194**	-

(1) Our method which only based on MF get best performance, even compared with SOTA methods

(2) Our method can get good performance on all datasets. Full-retrain and Fine-tune can only perform well one datasets respectively.

(3) sample-based retraining method SPMF performs better than Fine-tune on Yelp, but not on Adressa. Drawback of heuristically designed method.

-- **wonderful ability that automatically adapt to different scenarios**.

-- **historical data can be discarded during retraining, as long as the previous model can be properly utilized**

Performance

□ Each period -- recommendation and speed-up

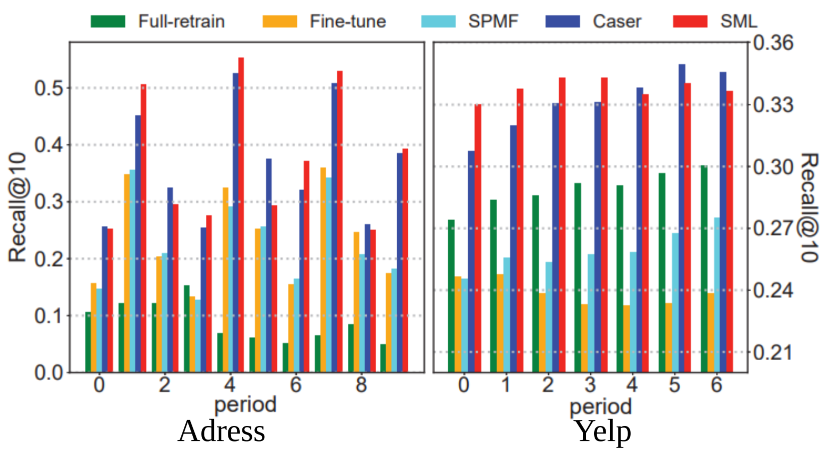


Table 2: Retraining time (seconds) at each testing period on Yelp. SML-S is the variant that disables transfer update.

period	0	1	2	3	4	5	6
Full-retrain	1,458	1,492	1,546	1,599	1,634	1,701	1,749
SML	90	91	89	89	89	89	90
Fine-tune	34	34	35	34	34	35	34
SML-S	8	8	8	8	8	8	8

(1) SML achieves the best performance in most cases
(2) the fluctuations on Adressa are larger than Yelp. Strong timeliness of the news domain

(1) SML is about 18 times faster than Full-retrain
(2) SML is stable
(3) SML-S (disabling the update of the transfer) SML-S is even faster than Fine-tune

Our method is efficient

How do the components of SML affect its effectiveness?

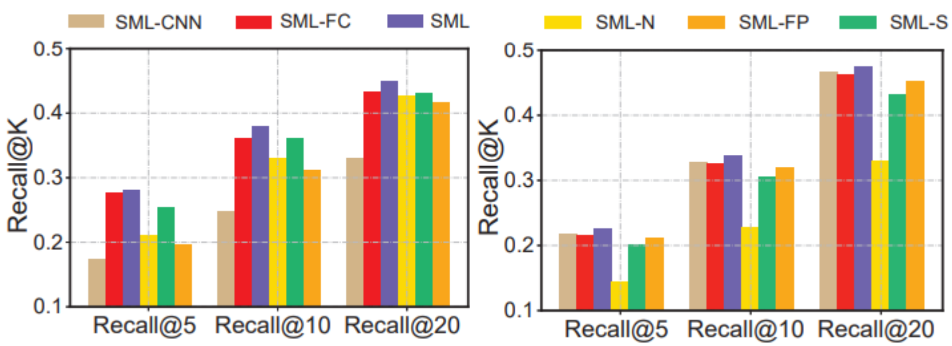
- Some variants:

SML-CNN: remove CNN **SML-FC**: remove FC layer

SML-N: disables the optimization of the transfer towards the next-period performance

SML-S: disabling the update of the transfer during testing

SML-FP: learns the $\widehat{W}_t$ directly based on itself recommendation loss on D_t



(a) Adressa (b) Yelp

CNN and FC layer: both dimension-wise relations and cross-dimension relations between $\widehat{W}_t$ and W_{t-1}

SML-N: worse than SML by 18.81% and 34.53% on average, optimizing towards future performance is important

SML-S: drops by 7.87% and 9.43%. The mechanism for transfer may need be changed with times goes by.

SML-FP: fails to achieve a comparable performance as SML on both datasets.

Where does improvements come from?

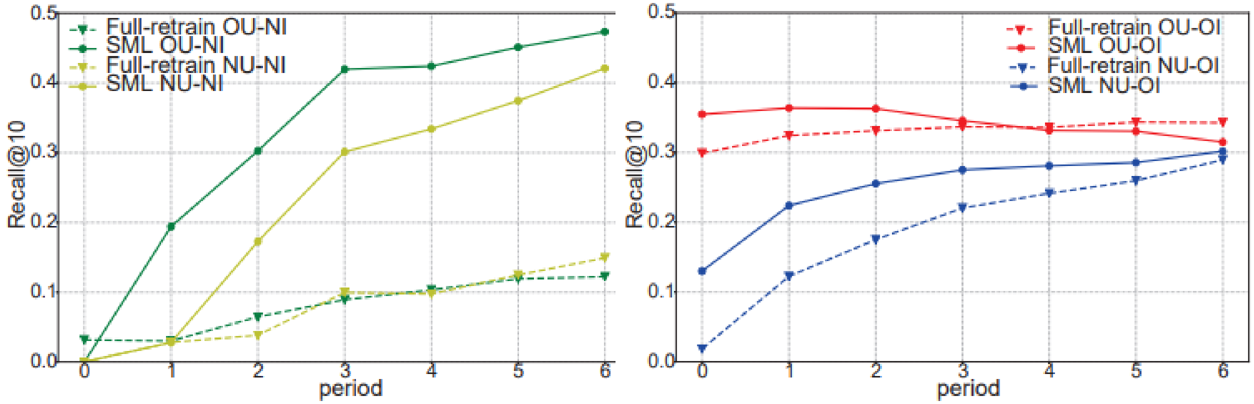
Compared with Full-retrain on Yelp

User(item): new users(items): only occur in the testing data.

old user(items): otherwise

interactions: old user-new item (OU-NI), new user-new item (NU-NI),

old user-old item (OU-OI), and new user-old item (NU-OI)



(a) Two types of new items (b) Two types of old items

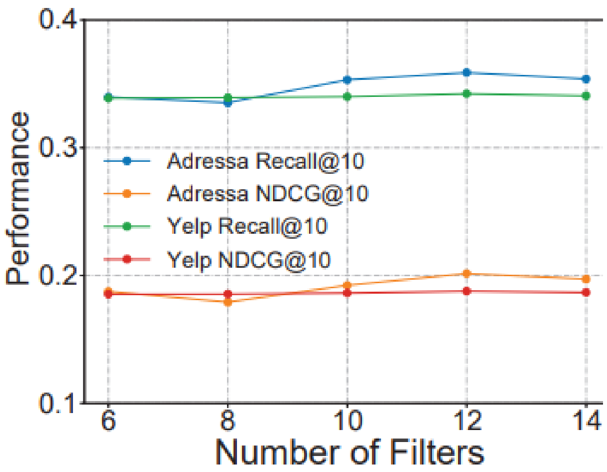
(1) improvements of SML over Full-retrain are mainly from the recommendations for **new users and new items**.

(2) strong ability of SML in quickly adapting to new data

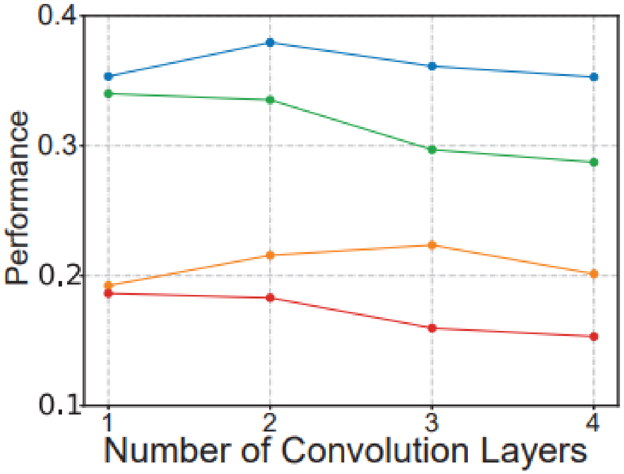
(3) performance on the interaction type of old user-old item is nearly not degraded

Influence of hyper-parameters

- Focus on hyper-parameters of CNN component



(a) Impact of filter number



(b) Impact of convolution layers

In some range of hyper-parameters, the performance is stable in some degree. There are better hyper-parameters.

Conclusion & future works

- main contributions：
  - formulate the sequential retraining process as an optimizable problem
  - new retraining approach：
    - Recover knowledge of previous data by previous model instead of data. **It is efficient.**
    - **Effective** by optimizing for the future recommendation performance
- Future works:
  - Implement SML based on other models such as $\boldsymbol{LigthGCN}$[1] verify its **generality**
  - Task/category-aware transfer designed.
    different users/items may need different mechanism of transfer。

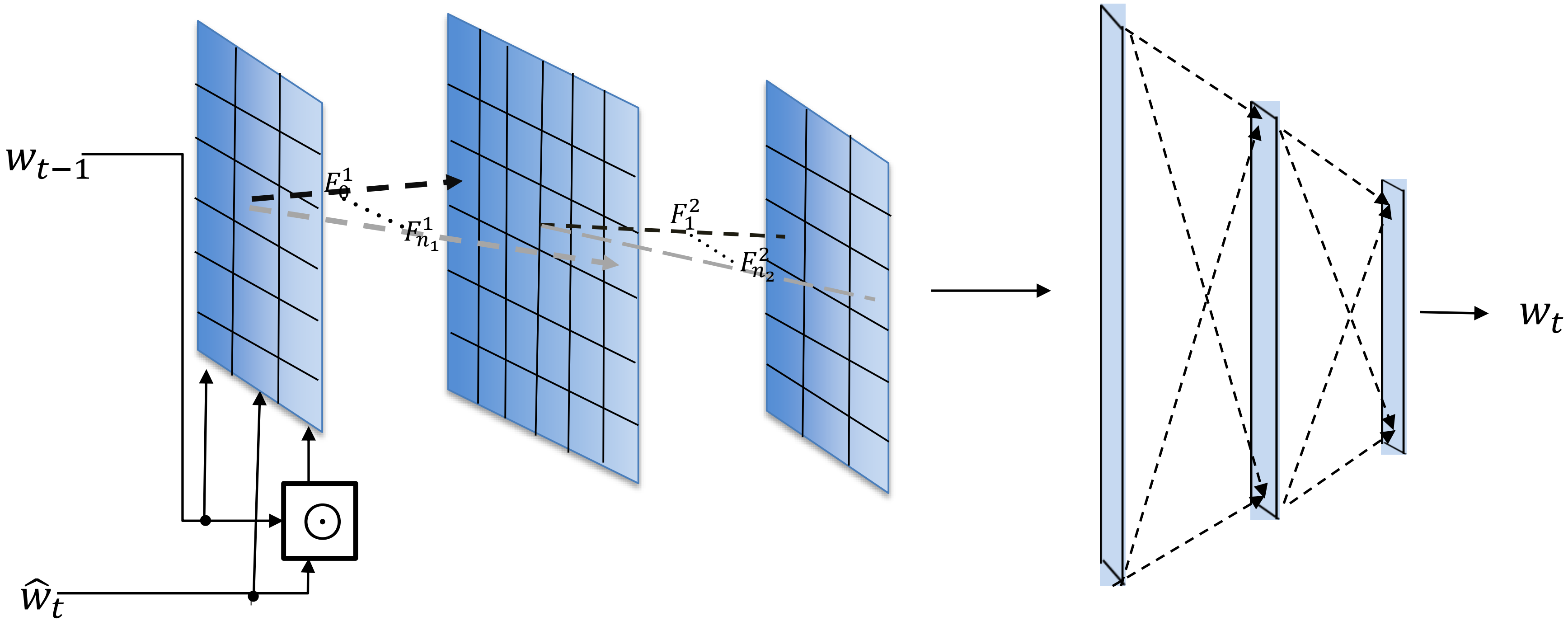
[1]. Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In SIGIR

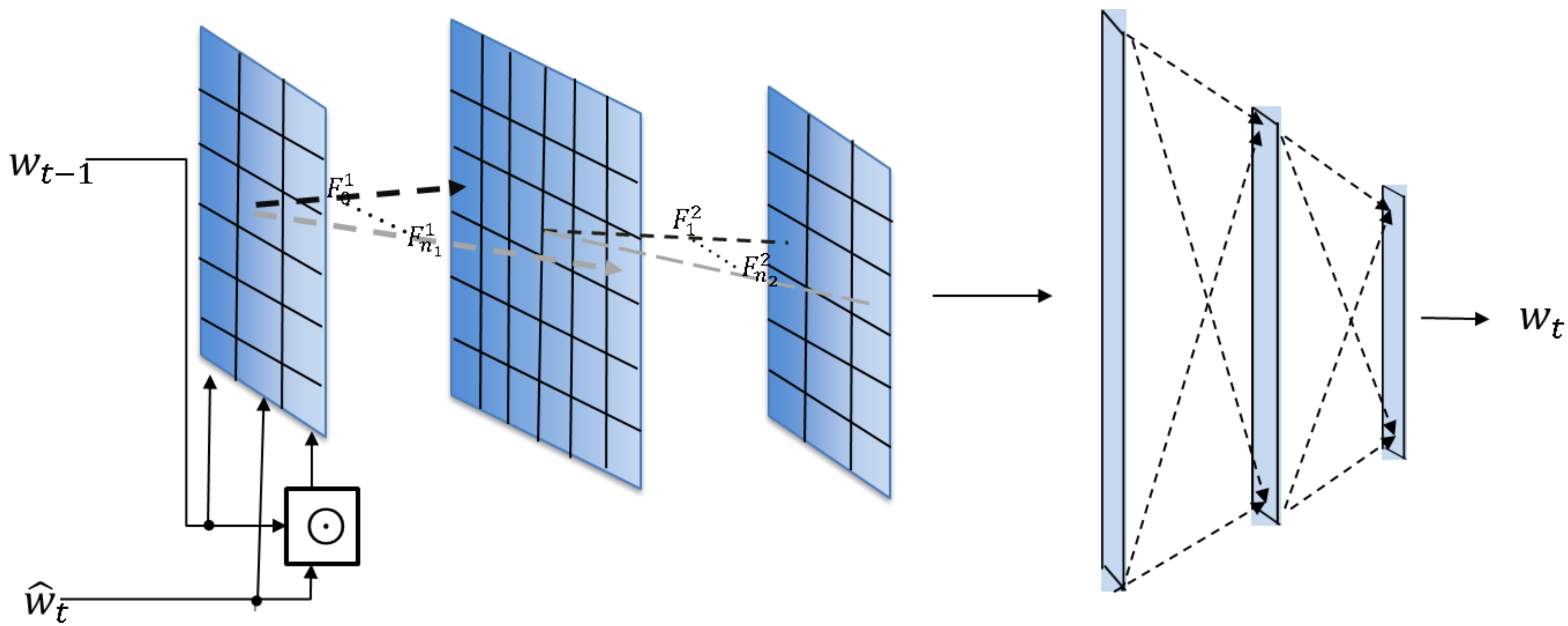
Thank You

Q&A

Transfer

model





w_{t-1}
$\hat{w}_t$
F_0^1
$F_{n_1}^1$
F_1^2
$F_{n_2}^2$
w_t