Automated Machine Learning for Recommender System

Chen Gao

Ph.D Candidate, Tsinghua University

gc16@mails.tsinghua.edu.cn

Outline

- AutoML for Collaborative Filtering Task
- AutoML for Click-through Rate Prediction Task
- AutoML for Tuning Hyper-parameters in RecSys

Outline

- AutoML for Collaborative Filtering Task

- AutoML for Click-through Rate Prediction Task

- AutoML for Tuning Hyper-parameters in RecSys

Collaborative Filtering – Problem Setup

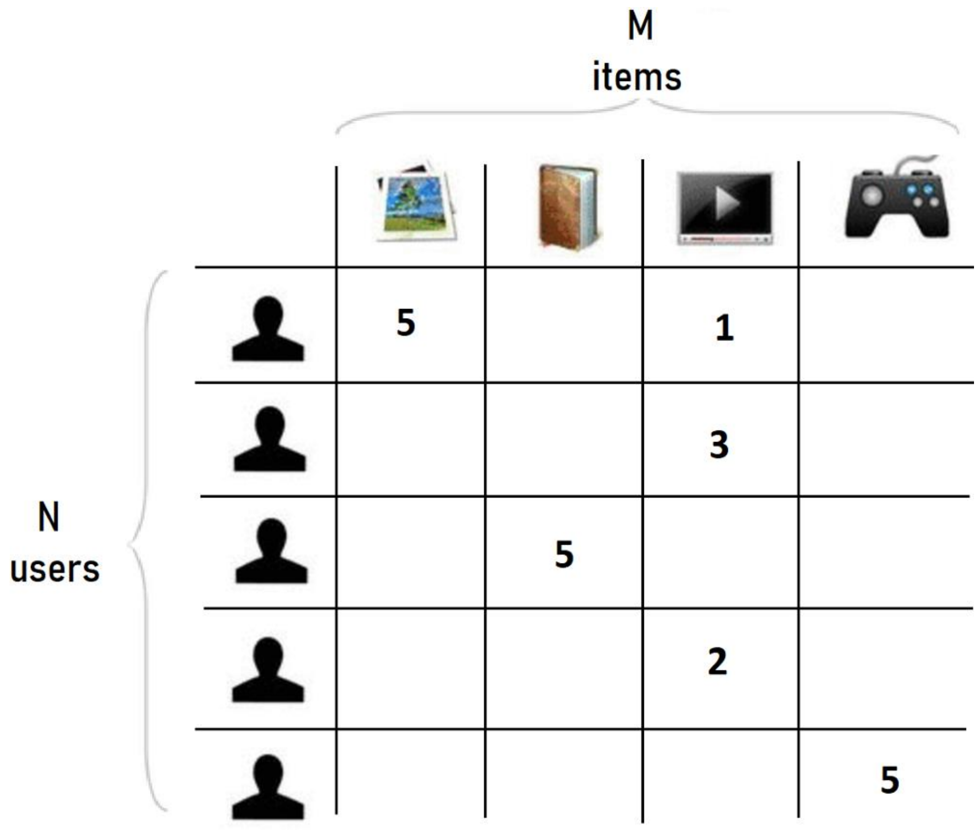


Image is from: https://mc.ai/deep-learning-for-collaborative-filtering-using-fastai/

Study the fundamental CF problem [1]

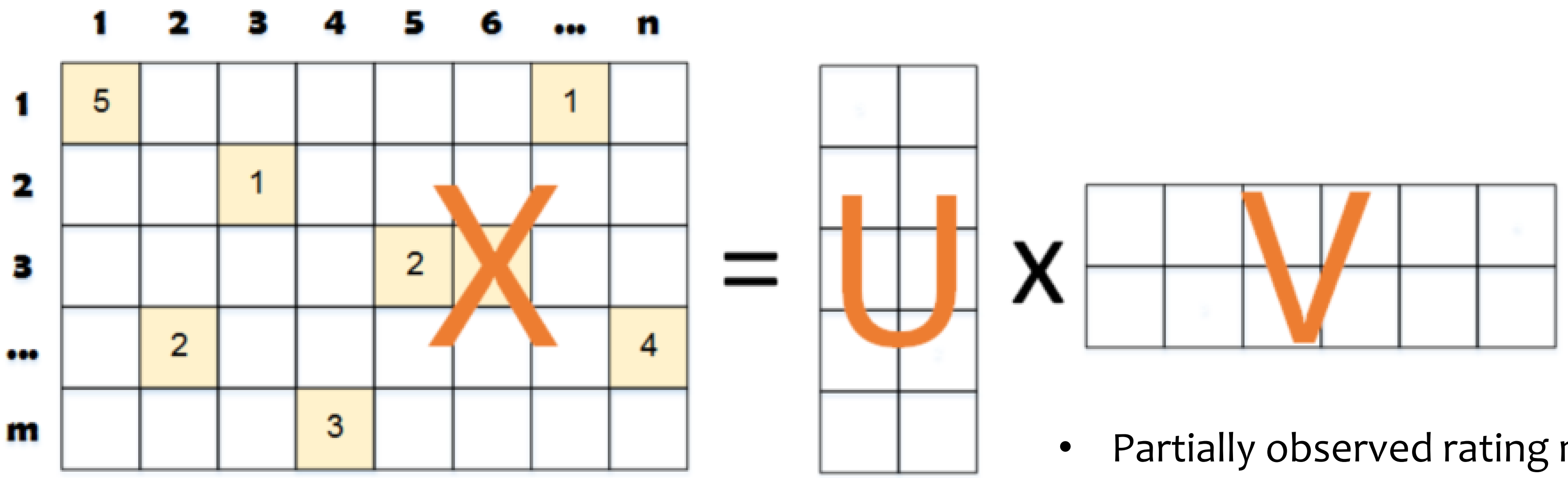
- Data: a rating matrix with many unknown positions
- Task: estimate ratings on unknown positions
- Measurement: RMSE on estimated ratings.

More clarifications

- Take CF as a regression task here
- Implicit feedbacks [2,3] are NOT considered
- Side-information is [4] (e.g. user/item features) NOT assumed

[1]. Exact Matrix Completion via Convex Optimization. Foundations of Computational Mathematics. 2008
[2]. Collaborative filtering for implicit feedback datasets. ICDE 2008
[3]. Neural Collaborative Filtering. WWW 2017
[4]. Meta-Graph Based Recommendation Fusion over Heterogeneous Information Networks. KDD 2017

Collaborative Filtering – Low-rank approach [1,2]



Prediction

$$\min_{U,V} \sum_{(i,j)\in\Omega} \ell\left(\boldsymbol{u}_i^\top \boldsymbol{v}_j, \boldsymbol{O}_{ij}\right) + \underbrace{\frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2}_{\text{Regularization}},$$

Observed Positions

Rating

- Partially observed rating matrix O can be well-approximated by a low-rank matrix X
- Matrix U: user embedding, matrix V: item embedding
- Rating prediction is given by an inner product of user embedding and item embedding

[1]. Factorization meets the neighborhood: a multifaceted collaborative filtering model. KDD 2008
[2]. Exact Matrix Completion via Convex Optimization. Foundations of Computational Mathematics. 2008

Collaborative Filtering – Interaction Function (IFC)

$$\min_{\boldsymbol{U},\boldsymbol{V}} \sum\nolimits_{(i,j)\in\Omega} \ell\left(\boldsymbol{u}_i^\top \boldsymbol{v}_j, \boldsymbol{O}_{ij}\right) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2,$$

1. Generate embedding vectors for users and items
2. Generate predictions by an inner product between embedding vectors
3. Evaluate predictions by a loss function on the training data set

Interaction function: how embedding vectors interact with each other

Low-rank approach: using inner product as interaction function

Collaborative Filtering – Alternative IFCs

Is low-rank approach good enough? NO, depends on tasks and datasets

- User **UA** likes item **IA** very much
- User **UB** likes item **IA** also very much

UA and UB should have very similar embeddings

Triangle inequality: $\|u_i - u_j\| \leq \|u_i - v_j\| + \|u_k - v_j\|$

Alternative IFC: minus/plus [1]

Failure of low-rank approach: $u_i^T v_j = u_k^T v_j$ does not mean u_i and u_k have small distance

[1]. Collaborative metric learning. WWW 2017

Collaborative Filtering – More Example IFCs

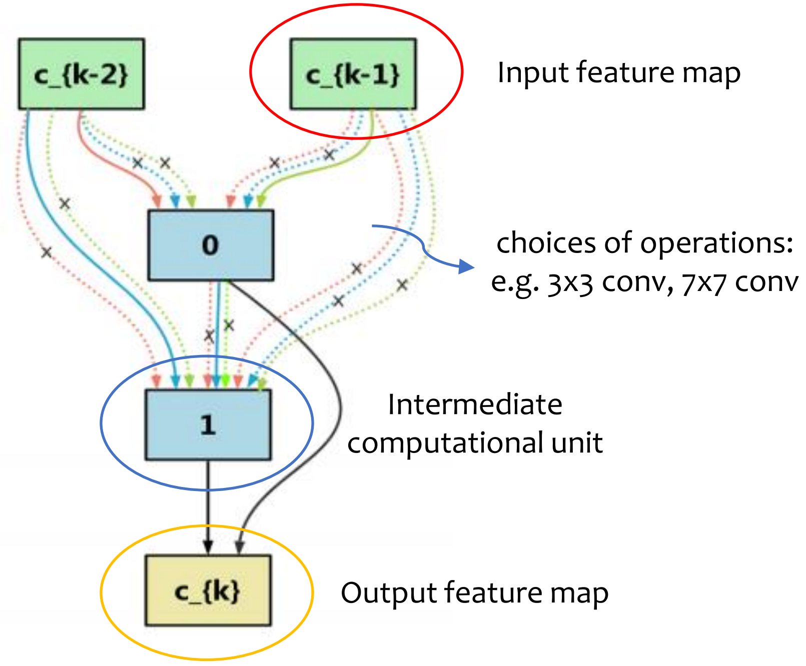
Table 1: Popular human-designed interaction functions (IFC) for CF, where H is a parameter to be trained. SIF searches a proper IFC from the validation set (i.e., by AutoML), while others are all designed by experts.

	IFC	operation	space	predict time	recent examples
human-designed	$\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$	inner product	$O((m+n)k)$	$O(k)$	MF [28], FM [37]
	$\boldsymbol{u}_i - \boldsymbol{v}_j$	plus (minus)	$O((m+n)k)$	$O(k)$	CML [19]
	$\max\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right)$	max, min	$O((m+n)k)$	$O(k)$	ConvMF [25]
	$\sigma\left([\boldsymbol{u}_i; \boldsymbol{v}_j]\right)$	concat	$O((m+n)k)$	$O(k)$	Deep&Wide [9]
	$\sigma\left(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}\left[\boldsymbol{u}_i; \boldsymbol{v}_j\right]\right)$	multi, concat	$O((m+n)k)$	$O(k^2)$	NCF [17]
	$\boldsymbol{u}_i * \boldsymbol{v}_j$	conv	$O((m+n)k)$	$O(k\log(k))$	ConvMF [25]
	$\boldsymbol{u}_i \otimes \boldsymbol{v}_j$	outer product	$O((m+n)k)$	$O(k^2)$	ConvNCF [16]
AutoML	SIF (proposed)	searched	$O((m+n)k)$	$O(k)$	——

Is there an absolute best IFC? : NO, depends on tasks and datasets [1]

[1]. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. Recsys 2019

NASP – Efficient NAS via Proximal Iterations[1]



Supernet for CNN (a computational cell)

Edge representation: $\bar{O}(X) = \sum_{k=1}^{d} a_k O_k(X)$ where $\alpha \in C_1 \cap C_2$,

(with sparse constraint) $C_1 = \{\alpha \mid \|\alpha\|_0 = 1\}$ and $C_2 = \{\alpha \mid 0 \le \alpha_k \le 1\}$.

Optimization Objective: $\min_{\alpha} \bar{\mathcal{L}}(w^*(\alpha), \alpha)$, s.t. $\begin{cases} w^*(\alpha) = \arg\min_w \mathcal{L}(w, \alpha) \\ \alpha \in C_1 \cap C_2 \end{cases}$

(bi-level optimization)

Validation set   *Training set*

α: architecture parameter, w: network parameter

Algorithm 1 Neural architecture search by proximal iterations (NASP) algorithm [47].

require: A mixture operation $\bar{O}$ parametrized by (2), parameter w and step-size η;
while not converged **do**
 Obtain *discrete* architecture representation $\bar{\alpha} = \text{prox}_{C_1}(\alpha)$;
 Update *continuous* architecture representation
$$\alpha = \text{prox}_{C_2}\left(\alpha - \nabla_{\bar{\alpha}} \bar{\mathcal{L}}(\bar{w}, \bar{\alpha})\right);$$
 where $\bar{w} = w - \eta \nabla_w \mathcal{L}(w, \bar{\alpha})$ (is an approximation to $w^*(\bar{\alpha})$);
 Get new *discrete* architecture $\bar{\alpha} = \text{prox}_{C_1}(\alpha)$;
 Update w using $\nabla_w \mathcal{L}(w, \bar{\alpha})$ with $\bar{\alpha}$;
end while
return Searched architecture $\bar{\alpha}$.

A proximal algorithm for NAS

- Space: Super-net
- Algorithm: Proximal Gradient descent
- Evaluation: Parameter-sharing

10+ times faster than DARTS[2]

[1] Efficient Neural Architecture Search via Proximal Iterations. AAAI 2020
[2] DARTS: Differentiable architecture search. ICLR 2019

Motivation

Is there an absolute best IFC? : NO, depends on tasks and datasets

Why not search (by NAS) an IFC from the data on the given task?

AutoML

- Search space
- Search Algorithm
- Evaluation Method

All these need to be carefully designed to be efficient and get better performance than existing IFCs.

HOW?

Search Space – Problem definition

Generalize standard CF objective:

$$\min F(\boldsymbol{U},\boldsymbol{V},\boldsymbol{w}) \equiv \sum\nolimits_{(i,j)\in\Omega} \ell\left(\boldsymbol{w}^\top f\left(\boldsymbol{u}_i,\boldsymbol{v}_j\right),\boldsymbol{O}_{ij}\right) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2, \text{ s.t. } \|\boldsymbol{w}\|_2 \le 1,$$

Interaction Function:
- take user / item embeddings as input and output a vector

What can be a good candidate space?

Definition 3.1 (AutoML problem). *Let $\mathcal{M}$ be a performance measure (the lower the better) defined on the validation set $\bar{\Omega}$ (disjoint from Ω), and $\mathcal{F}$ be a family of vector-valued functions with two vector inputs. The problem of searching for an interaction function (SIF), i.e., finding f^*, is defined as*

$$f^* = \arg\min_{f\in\mathcal{F}} \sum\nolimits_{(i,j)\in\bar{\Omega}} \mathcal{M}\left(f(\boldsymbol{u}_i^*,\boldsymbol{v}_j^*)^\top \boldsymbol{w}^*,\boldsymbol{O}_{ij}\right) \quad (6)$$
$$\text{s.t. } [\boldsymbol{U}^*,\boldsymbol{V}^*,\boldsymbol{w}^*] = \arg\min_{\boldsymbol{U},\boldsymbol{V},\boldsymbol{w}} F(\boldsymbol{U},\boldsymbol{V},\boldsymbol{w}),$$

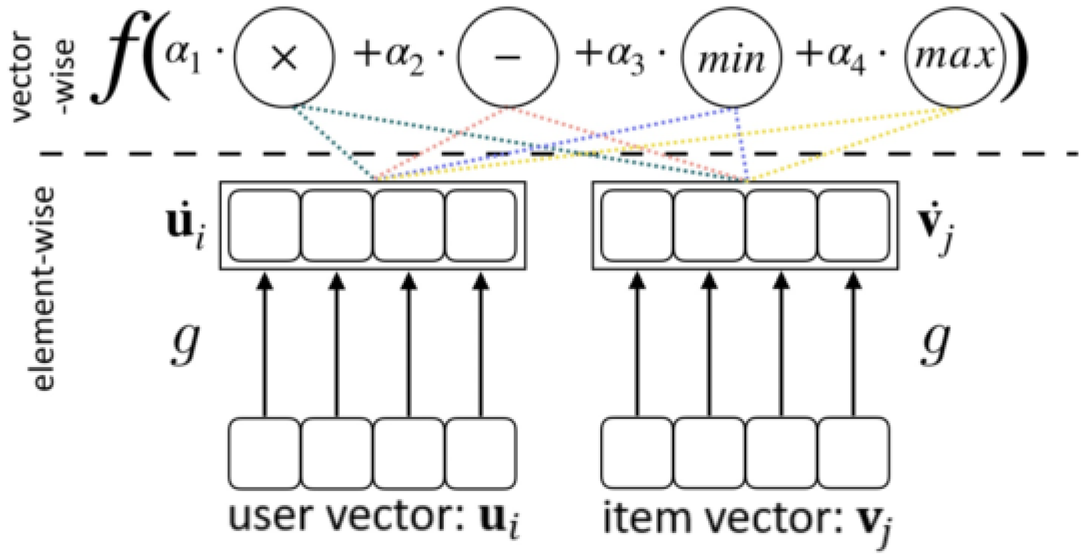
where $\boldsymbol{u}_i^*$ (resp. $\boldsymbol{v}_j^*$) is the ith column of $\boldsymbol{U}^*$ (resp. jth column of $\boldsymbol{V}^*$).

- Too large: lead to extremely huge computational cost for subsequent optimization algorithms
- Too small: no need to AutoML, worse performance than existing IFCs

Search Space – Learning from Existing IFCs

IFC	operation
$\langle \boldsymbol{u}_i, \boldsymbol{v}_j \rangle$	inner product
$\boldsymbol{u}_i - \boldsymbol{v}_j$	plus (minus)
$\max\left(\boldsymbol{u}_i, \boldsymbol{v}_j\right)$	max, min
$\sigma\left([\boldsymbol{u}_i; \boldsymbol{v}_j]\right)$	concat
$\sigma\left(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}\left[\boldsymbol{u}_i; \boldsymbol{v}_j\right]\right)$	multi, concat
$\boldsymbol{u}_i * \boldsymbol{v}_j$	conv
$\boldsymbol{u}_i \otimes \boldsymbol{v}_j$	outer product

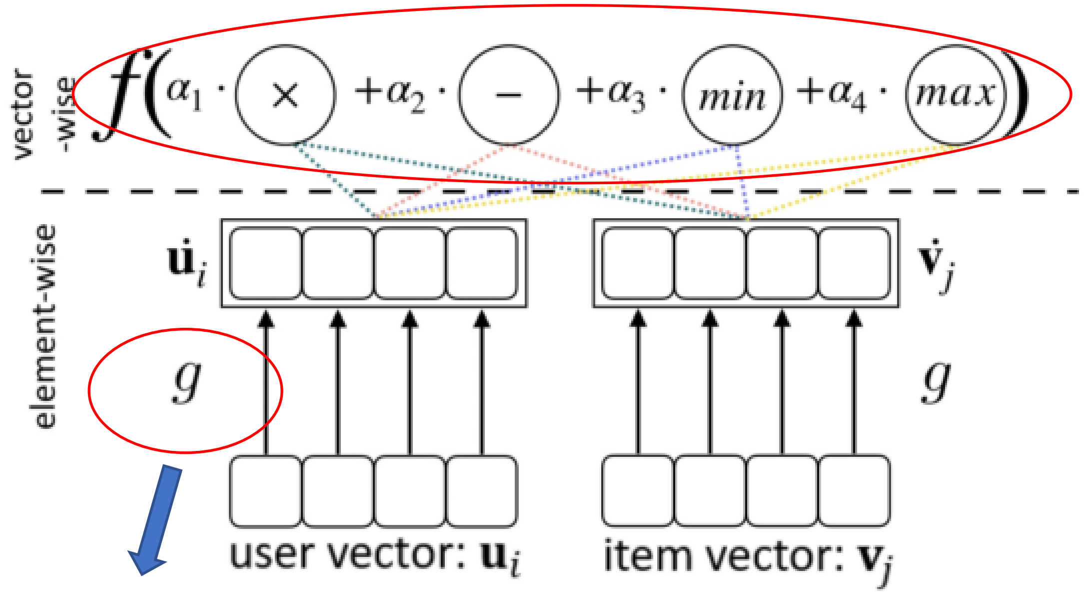
Cut the search space into two blocks



- Vector-level: simple linear algebra operations
- Elementwise: shared nonlinear transformation

Bilevel Optimization Objective

Can be seen as choices on operations



Implement using a small MLP

A Supernet Representation

S: architecture hyper-parameters

T: parameters

$$\begin{aligned} \min_S \quad & H(S,T) \equiv \sum\nolimits_{(i,j)\in\bar{\Omega}} \mathcal{M}(h_\alpha(\boldsymbol{u}_i^*, \boldsymbol{v}_j^*)^\top \boldsymbol{w}_\alpha^*, \boldsymbol{O}_{ij}) \qquad (9) \\ \text{s.t.} \quad & \boldsymbol{\alpha} \in \mathcal{C} \text{ and } \boldsymbol{T}^* \equiv \{\boldsymbol{U}^*, \boldsymbol{V}^*, \{\boldsymbol{w}_m^*\}\} = \arg\min_{\boldsymbol{T}} F_\alpha(\boldsymbol{T}; \boldsymbol{S}), \end{aligned}$$

High level

where F_α is the training objective:

$$\begin{aligned} F_\alpha(\boldsymbol{T};\boldsymbol{S}) \equiv & \sum\nolimits_{(i,j)\in\Omega} \ell(h_\alpha(\boldsymbol{u}_i, \boldsymbol{v}_j), \boldsymbol{O}_{ij}) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2, \\ & \text{s.t. } \|\boldsymbol{w}_m\|_2 \leq 1 \text{ for } m = 1, \dots, |\mathcal{O}|. \end{aligned}$$

Low level

- High level: optimize S
- Lew level: optimize T
- Bilevel programming is expensive to solve - T^* needs to be obtained from model training

Algorithm & Evaluation – Reusing NASP

Bilevel objective:

$$\min_S \quad H(S,T) \equiv \sum_{(i,j)\in\bar{\Omega}} \mathcal{M}(h_\alpha(\boldsymbol{u}_i^*, \boldsymbol{v}_j^*)^\top \boldsymbol{w}_\alpha^*, O_{ij}) \qquad (9)$$
$$\text{s.t.} \quad \boldsymbol{\alpha} \in \mathcal{C} \text{ and } \boldsymbol{T}^* \equiv \{\boldsymbol{U}^*, \boldsymbol{V}^*, \{\boldsymbol{w}_m^*\}\} = \arg\min_{\boldsymbol{T}} F_\alpha(\boldsymbol{T}; \boldsymbol{S}),$$

where F_α is the training objective:

$$F_\alpha(\boldsymbol{T};\boldsymbol{S}) \equiv \sum_{(i,j)\in\Omega} \ell(h_\alpha(\boldsymbol{u}_i, \boldsymbol{v}_j), O_{ij}) + \frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2,$$
$$\text{s.t. } \|\boldsymbol{w}_m\|_2 \le 1 \text{ for } m = 1, \ldots, |O|.$$

Reuse NASP for fast optimization

- **Effectiveness:** Maintain discrete architectures for S
- **Efficiency:** Update both S and T in an end-to-end and stochastic manner

Algorithm 2 Searching Interaction Function (SIF) algorithm.

1: Search space $\mathcal{F}$ represented by a structured MLP (Figure 1);
2: **while** epoch $t = 1, \cdots, T$ **do**
3: Select one operation $\bar{\boldsymbol{\alpha}} = \text{prox}_{\mathcal{C}_1}(\boldsymbol{\alpha})$;
4: *sample a mini-batch on validation data set*;
5: Update continuous $\boldsymbol{\alpha}$ for vector-wise operations

$$\boldsymbol{\alpha} = \text{prox}_{\mathcal{C}_2}(\boldsymbol{\alpha} - \eta\nabla_{\bar{\boldsymbol{\alpha}}} H(\boldsymbol{T}, \boldsymbol{S}));$$

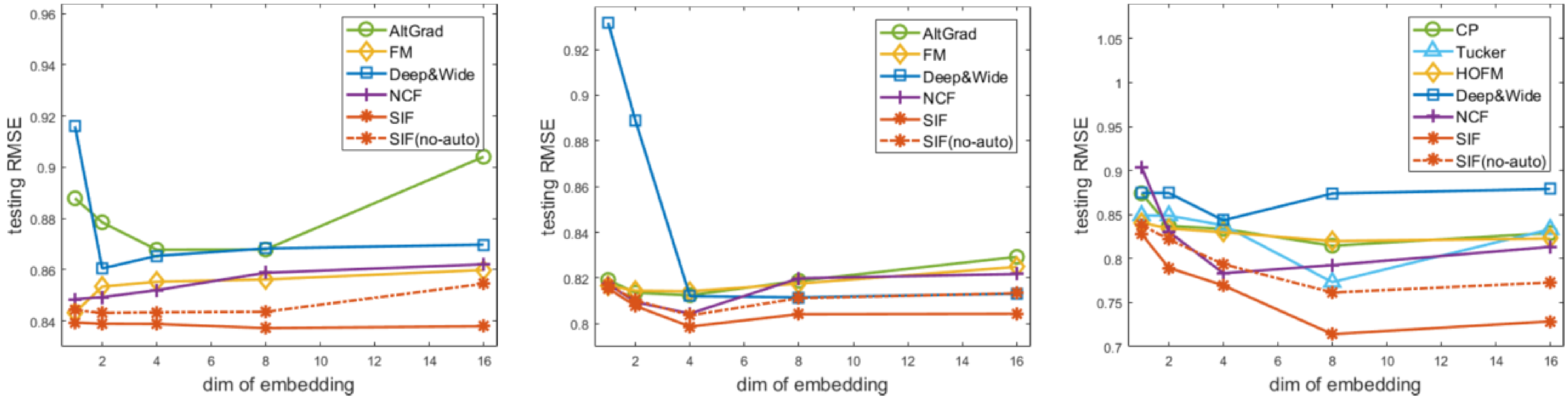
6: Update element-wise transformation

$$\boldsymbol{p} = \text{prox}_{\|\cdot\|_2\le 1}\left(\boldsymbol{p} - \eta\nabla_{\boldsymbol{p}} H(\boldsymbol{T}, \boldsymbol{S})\right),$$
$$\boldsymbol{q} = \text{prox}_{\|\cdot\|_2\le 1}\left(\boldsymbol{q} - \eta\nabla_{\boldsymbol{q}} H(\boldsymbol{T}, \boldsymbol{S})\right);$$

7: *sample a mini-batch on training data set*;
8: Get selected operation $\bar{\boldsymbol{\alpha}} = \text{prox}_{\mathcal{C}_1}(\boldsymbol{\alpha})$;
9: Update training parameters $\boldsymbol{T}$ with gradients on F_α;
10: **end while**
11: **return** Searched interaction function (parameterized by $\boldsymbol{\alpha}$, $\boldsymbol{p}$ and $\boldsymbol{q}$, see (7) and (8)).

Comparison with CF Approaches

(i) Alternating gradient descent (“***AltGrad***”); (ii) Factorization machine (“***FM***”); (iii) ***Deep&Wide***; (iv) Neural collaborative filtering (“***NCF***”); (v) ***SIF***; and (iv) ***SIF(no-auto)***, architecture is optimized with training data

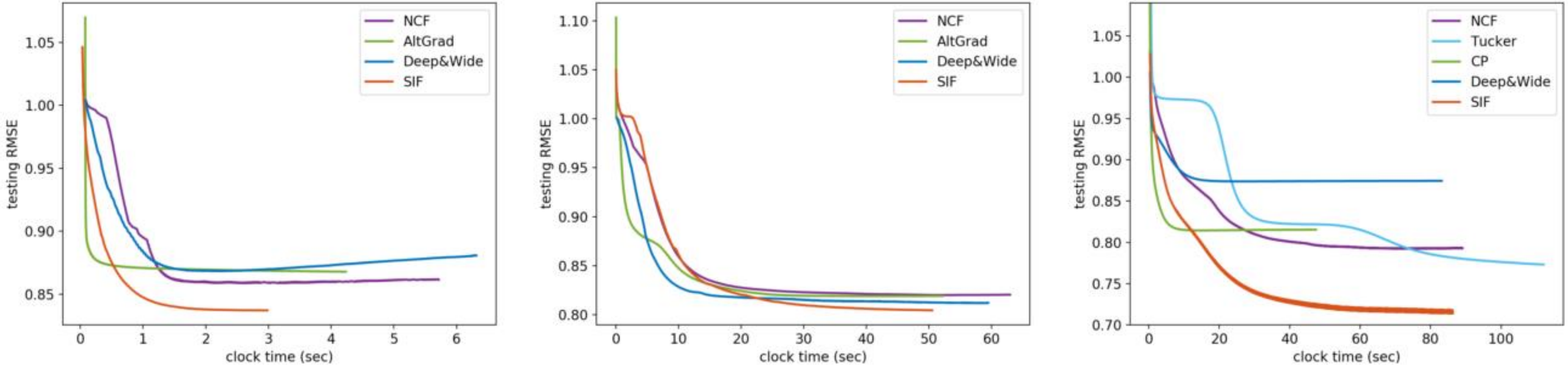


(a) MovieLens-100K. (b) MovieLens-1M. (c) Youtube.

Figure 2: Comparison of testing RMSEs between *SIF* and other CF approaches with different embedding dimension.

SIF is the best, and validation set helps architecture search

Comparison with CF Approaches



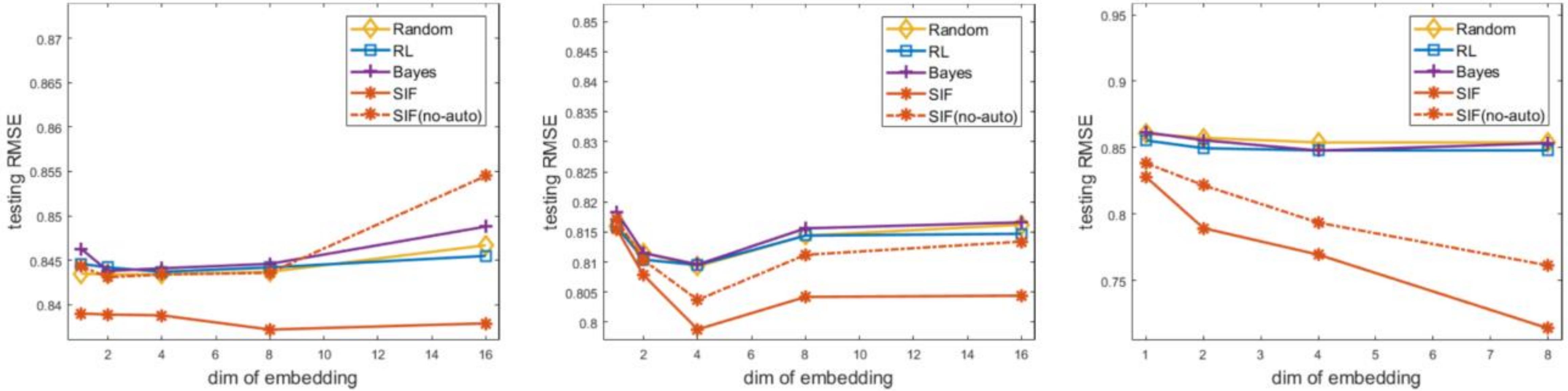
(a) MovieLens-100K. (b) MovieLens-1M. (c) Youtube.

Figure 3: Comparison of the convergence between *SIF* (with searched IFC) and other CF methods when embedded dimension is 8. *FM* and *HOFM* are not shown as their code donot support a callback to record testing performance.

Interaction function obtained from SIF can be trained as fast as state-of-the-art

Comparison with AutoML Approaches

(i) “Random”; (ii) “RL”: reinforcement learning; (iii) “Bayes”: HyperOpt

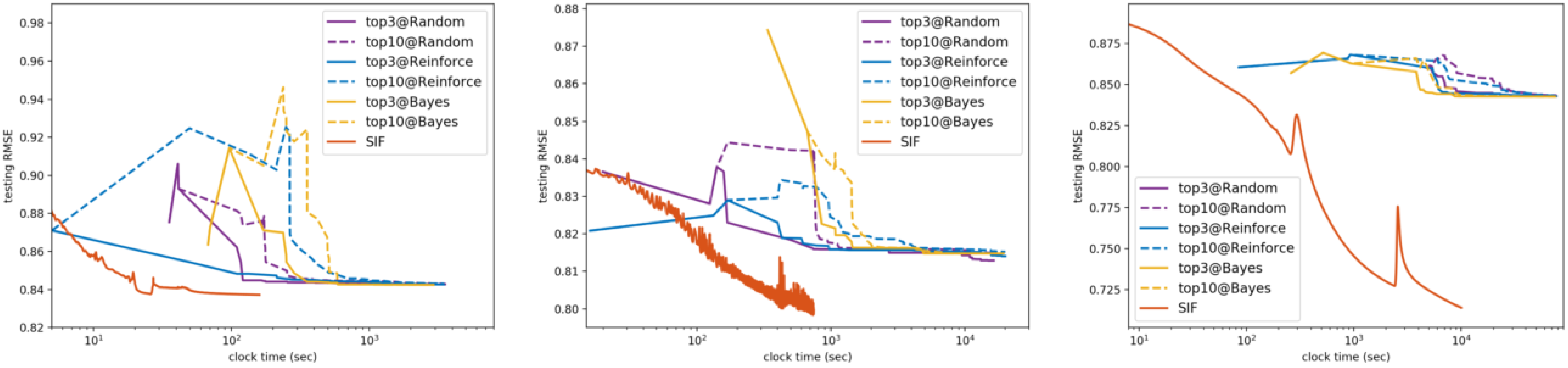


(a) MovieLens-100K. (b) MovieLens-1M. (c) Youtube.

Figure 4: Comparison of testing RMSEs between *SIF* and other AutoML approaches with different embedding dimensions. *Gen-approx* is slow with bad performance, thus is not run on Youtube.

SIF can find better architecture than other AutoML search algorithms

Comparison with AutoML Approaches



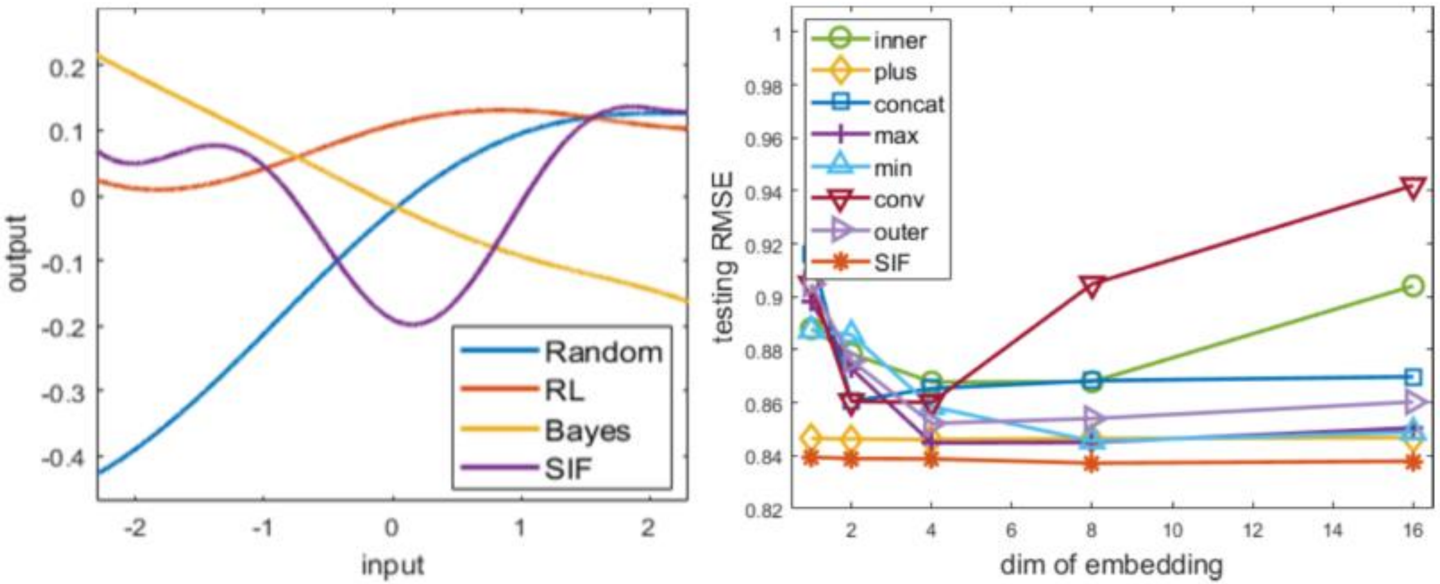
(a) MovieLens-100K. (b) MovieLens-1M. (c) Youtube.

Figure 5: Comparison of search efficiency among *SIF* and other AutoML approaches when embedded dimension is 8.

SIF is much faster than other AutoML search algorithms

Case Study - Searched Interaction Functions (IFCs)

dim	Random	RL	Bayes	SIF
1	concat	concat	min	inner
2	concat	plus	concat	min
4	concat	concat	concat	min
8	plus	plus	plus	inner
16	concat	plus	plus	plus

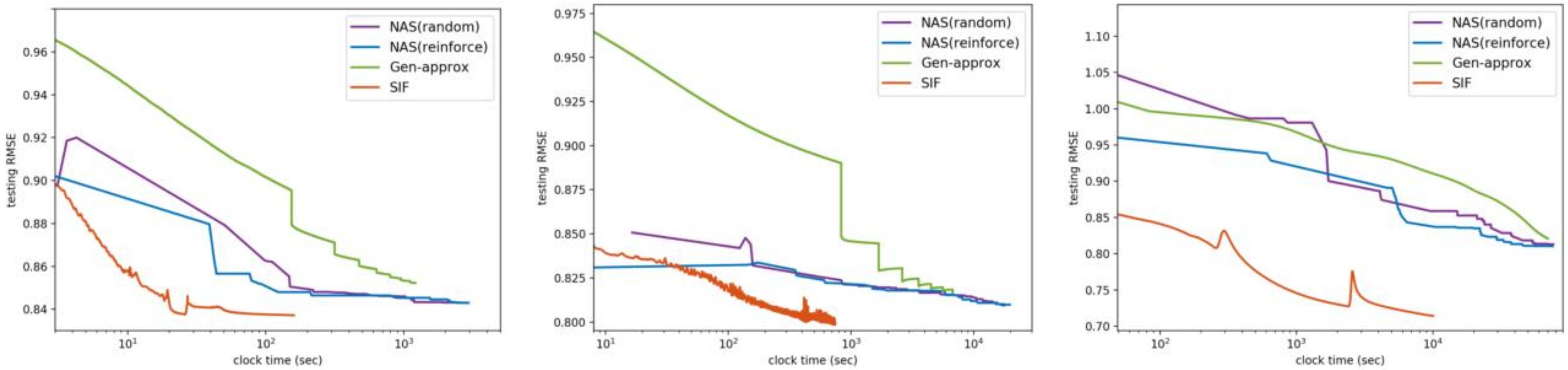


(a) Operations (vector-wise). (b) Non-linear transformation (element-wise). (c) Performance of each single operation.

Figure 6: (a-b). Searched IFCs on MivienLens-100K with embedded dimension equals 8. (c). Performance comparison between SIF and each single operation on MovieLens-100K.

SIF can find more complex transformation and give better performance than any single operation

Ablation Study – Different search space



(a) MovieLens-100K. (b) MovieLens-1M. (c) Youtube.

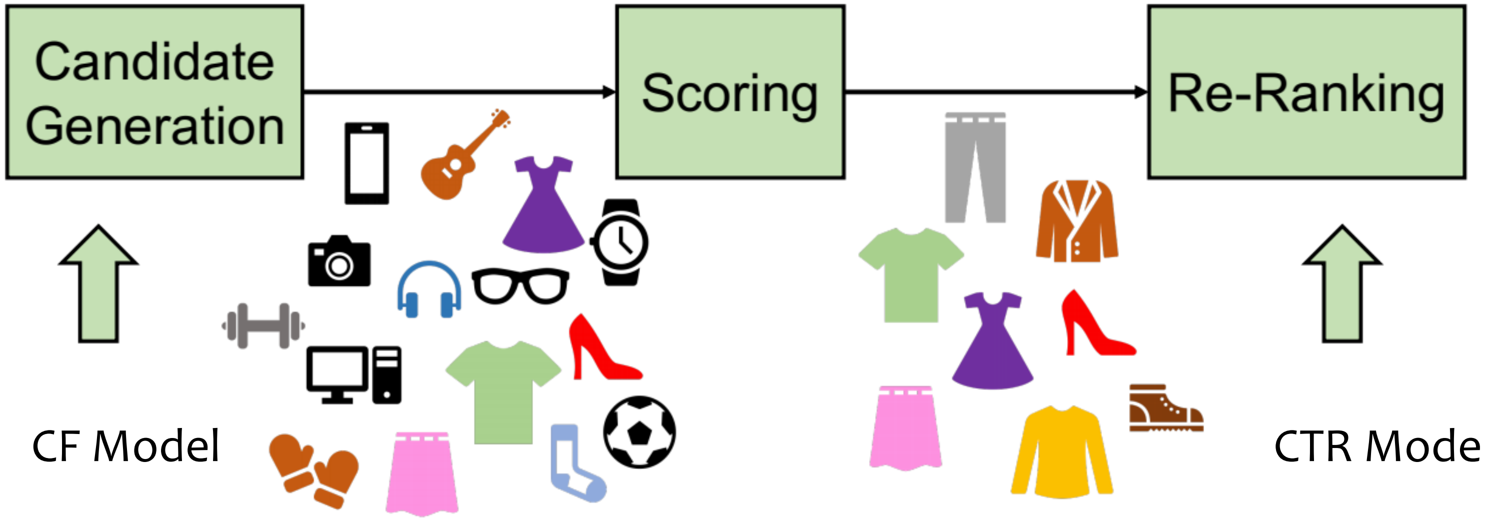
Figure 8: Comparison on different search space designs. Embedding dimension is 8.

NAS and Gen-approx are two general search space: inefficient to be searched

Outline

- AutoML for Collaborative Filtering Task
- AutoML for Click-through Rate Prediction Task
- AutoML for Tuning Hyper-parameters in RecSys

A pipeline of modern recommendation engine



Data Input : User-item interaction/rating data

Data Input : Rich attributes and context

Click-through rate prediction: tabular data

	age (n)	job (c)	marital (c)	education (c)	balance (n)	housing (c)
0	30	unemployed	married	primary	1787	no
1	33	services	married	secondary	4789	yes
2	35	management	single	tertiary	1350	yes
3	30	management	married	tertiary	1476	yes
4	59	blue-collar	married	secondary	0	yes
5	35	management	single	tertiary	747	no

An example of tabular data (UCI-Bank)

Cross-feature

- What is cross-features?
  - Taking cross-product of sparse features
- Why we need cross-features?
  - Capture the interaction among categorical features
  - Achieve great success in real-world business
- Traditional manner
  - LR+GBDT
  - DeepFM[1], xDeepFM[2], etc.

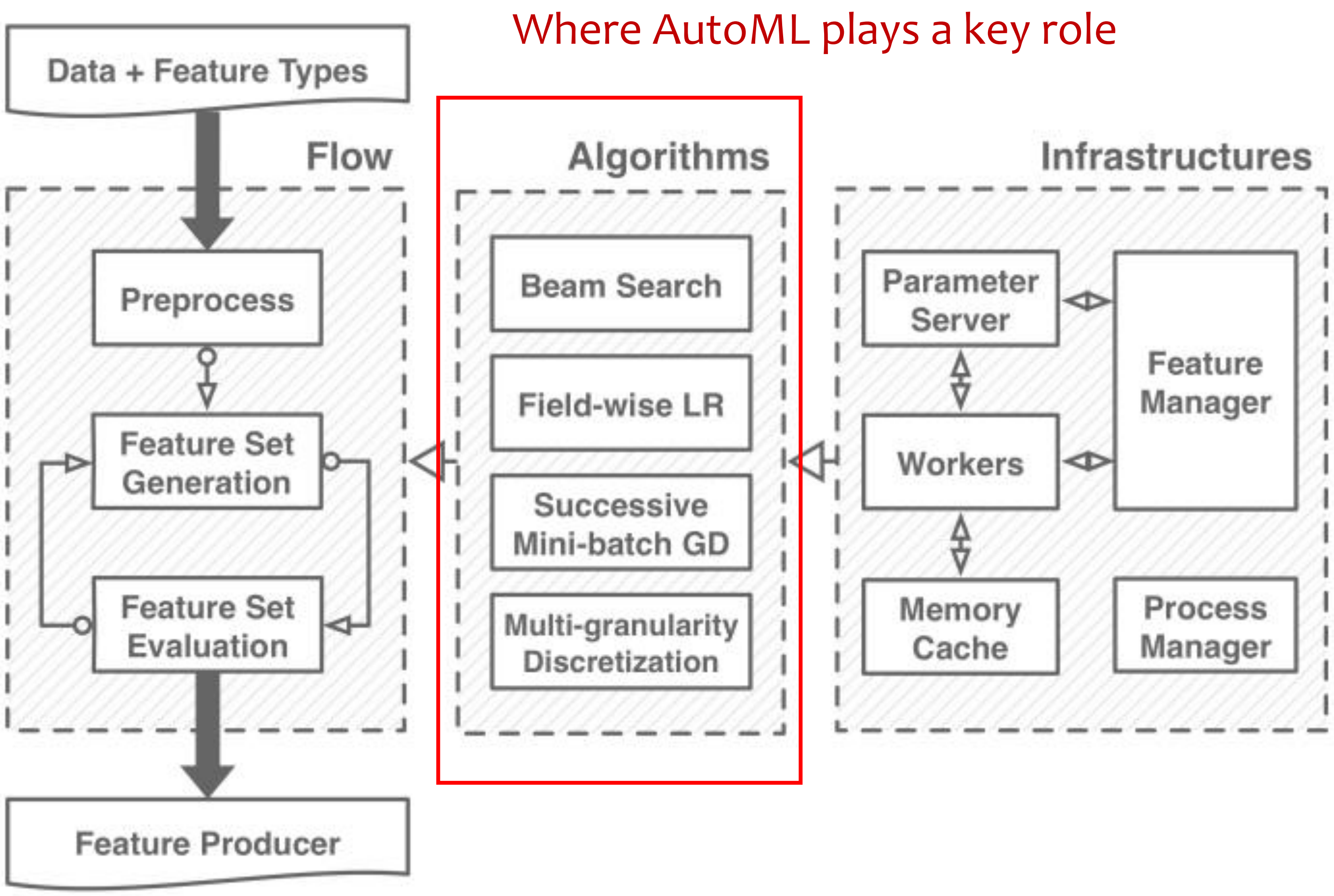
[1] Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." IJCAI 2017
[2] Lian, Jianxun, et al. "xdeepfm: Combining explicit and implicit feature interactions for recommender systems." KDD 2018

Motivation

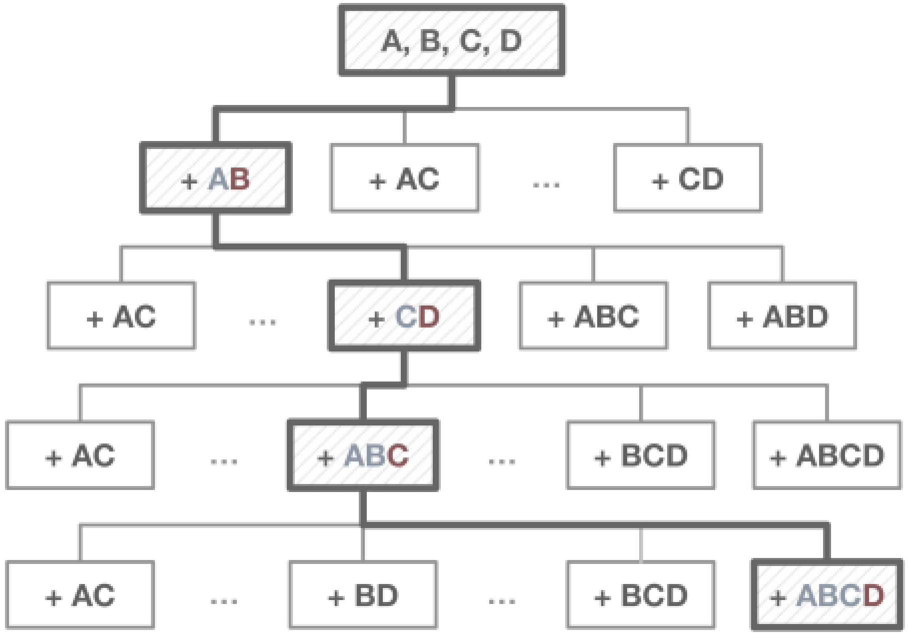
- Weaknesses of existing methods

Method	High-order Feature Crossing	Simplicity	Fast Inference	Interpretability
Search-based methods (e.g., [5, 34])	×	medium	√	√
Implicit deep-learning-based methods (e.g., [33, 42])	×	low	×	×
Explicit deep-learning-based methods (e.g., [26, 37])	×	low	×	√
AutoCross	√	high	√	√

Real-world System Framework of AutoCross



Method (feature search)



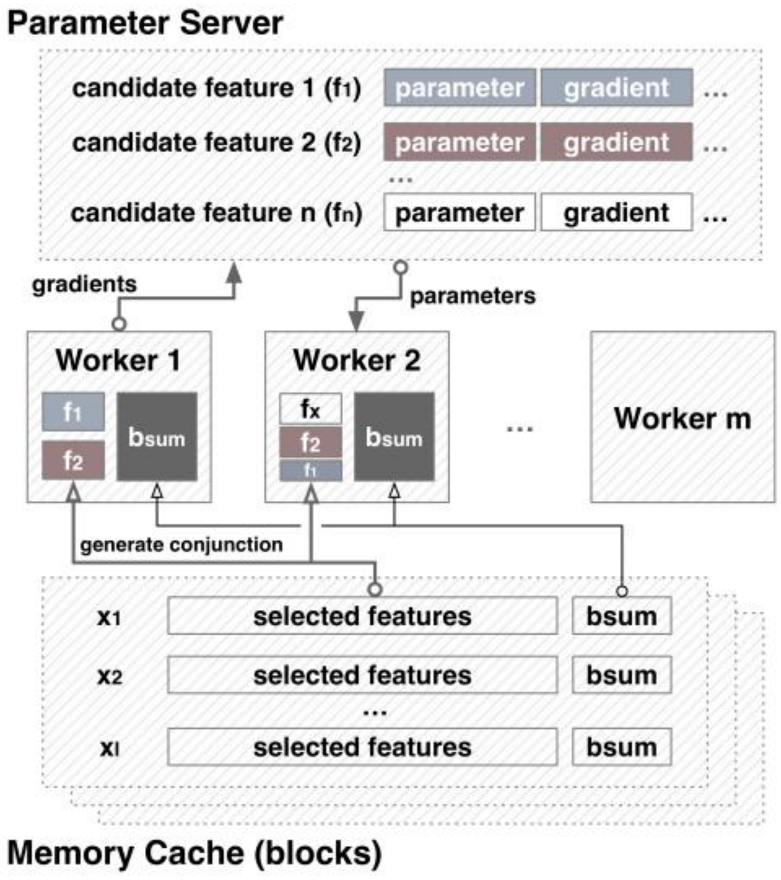
Search space and beam search strategy

Algorithm 1 Feature Set Search Strategy in AutoCross.

Require: original feature set $\mathcal{F}$.
Ensure: solution $\mathcal{S}^*$.

initialize current node $\mathcal{S}^* \leftarrow \mathcal{F}$;
while procedure not terminated **do**
  Feature Set Generation: expand $\mathcal{S}^*$, generate its children node set `children`($\mathcal{S}^*$) by adding to itself different pair-wise crossing of its elements;
  Feature Set Evaluation: evaluate all candidate feature sets in `children`($\mathcal{S}^*$) and identify the best child $\mathcal{S}'$;
  visit $\mathcal{S}'$: $\mathcal{S}^* \leftarrow \mathcal{S}'$
end while
return $\mathcal{S}^*$.

Method (feature evaluation)



field-wise logistic regression for feature evaluation based on PS architecture

$$P(y = 1|\mathbf{x}) = s(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = s(\mathbf{w}_s^{\mathrm{T}}\mathbf{x}_s + \mathbf{w}_c^{\mathrm{T}}\mathbf{x}_c) = s(\mathbf{w}_c^{\mathrm{T}}\mathbf{x}_c + b_{sum})$$

field-wise logistic regression (LR)

Algorithm 2 Successive Mini-batch Gradient Descent (SMBGD).

Require: set of candidate feature sets $\mathbb{S} = \{\mathcal{S}_i\}_{i=1}^{n}$, training data equally divided into $N \geq \sum_{k=0}^{\lceil \log_2 n \rceil - 1} 2^k$ data blocks.

Ensure: best candidate $\mathcal{S}'$.

for $k = 0, 1, \cdots, \lceil \log_2 n \rceil - 1$ **do**
 use additional 2^k data blocks to update the field-wise LR models of all $\mathcal{S} \in \mathbb{S}$, with warm-starting;
 evaluate the models of all $\mathcal{S}$'s with validation AUC;
 keep the top half of candidates in $\mathbb{S}$: $\mathbb{S} \leftarrow$ `top_half(`$\mathbb{S}$`)` (rounding down);
 break if $\mathbb{S}$ contains only one element;
end for
return $\mathcal{S}'$ (the singleton element of $\mathbb{S}$).

Evaluations (Effectiveness)

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
LR (base)	0.9400	0.9169	0.8292	0.8655	0.7855
AC+LR	**0.9455**	**0.9280**	**0.8567**	**0.8942**	0.8034
AC+W&D	0.9420	**0.9260**	**0.8623**	**0.9033**	**0.8068**
CMI+LR	**0.9431**	0.9153	0.8336	0.8901	0.7844
Deep	0.9418	0.9130	0.8369	0.8745	0.7985
xDeepFM	0.9419	0.9131	0.8358	0.8746	**0.8059**
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
LR (base)	0.8368	0.8356	0.6960	0.6117	0.5992
AC+LR	**0.8545**	**0.8536**	**0.7065**	**0.6276**	0.6393
AC+W&D	**0.8531**	**0.8552**	**0.7026**	**0.6260**	**0.6547**
Deep	0.8479	0.8463	0.6936	0.6207	0.6509
xDeepFM	0.8504	0.8515	0.6936	0.6241	**0.6514**

Experimental results (test AUC) on benchmark and real-world business datasets.

AC+LR v.s. **LR (base)**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.585%	1.211%	3.316%	3.316%	2.279%	2.141%
Data1	Data2	Data3	Data4	Data5	**Average**
2.115%	2.154%	1.509%	2.599%	6.692%	3.014%
AC+W&D v.s. **LR (base)**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.213%	0.992%	3.992%	4.367%	2.712%	2.455%
Data1	Data2	Data3	Data4	Data5	**Average**
1.948%	2.346%	0.948%	2.338%	9.546%	3.368%
AC+W&D v.s. **Deep**					
Bank	Adult	Credit	Employee	Criteo	**Average**
0.021%	1.424%	3.035%	3.293%	1.039%	1.763%
Data1	Data2	Data3	Data4	Data5	**Average**
0.6133%	1.0516%	1.2976%	0.8539%	0.5361%	0.880%

Experimental results (test AUC) on benchmark and real-world business datasets.

Evaluations (Efficiency)

Benchmark Datasets				
Bank	Adult	Credit	Employee	Criteo
0.0267	0.0357	0.3144	0.0507	3.0817
Real-World Business Datasets				
Data1	Data2	Data3	Data4	Data5
0.9327	0.7973	1.5206	2.7572	5.1861

Cross feature generation time (unit: hour).

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
AC+LR	0.00048	0.00048	0.00062	0.00073	0.00156
AC+W&D	0.01697	0.01493	0.00974	0.02807	0.02698
Deep	0.01413	0.01142	0.00726	0.02166	0.01941
xDeepFM	0.08828	0.05522	0.04466	0.06467	0.18985
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
AC+LR	0.00367	0.00111	0.00185	0.00393	0.00279
AC+W&D	0.03537	0.01706	0.04042	0.02434	0.02582
Deep	0.02616	0.01348	0.03150	0.01414	0.01406
xDeepFM	0.32435	0.11415	0.40746	0.12467	0.13235

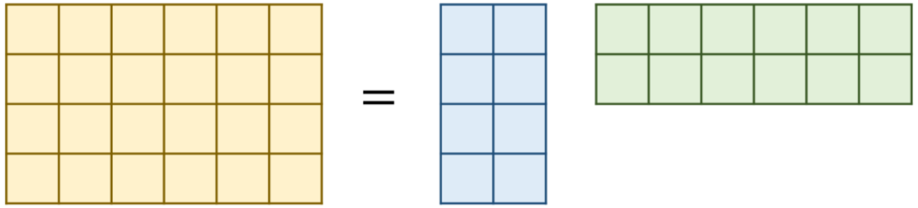
Inference latency comparison (unit: millisecond)

Outline

- AutoML for Collaborative Filtering Task
- AutoML for Click-through Rate Prediction Task
- AutoML for Tuning Hyper-parameters in RecSys

Regularization Tuning Headache

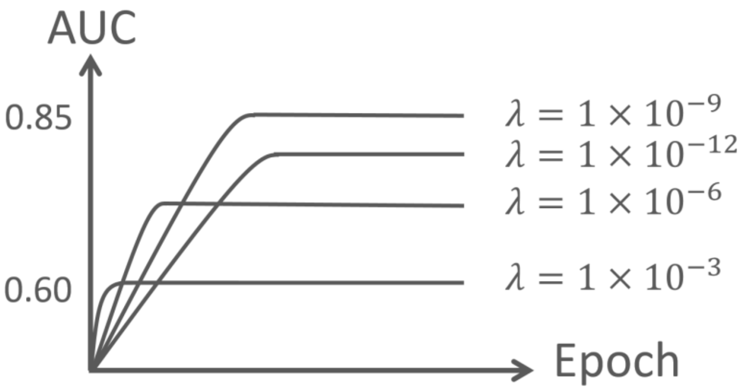
Bob



Model

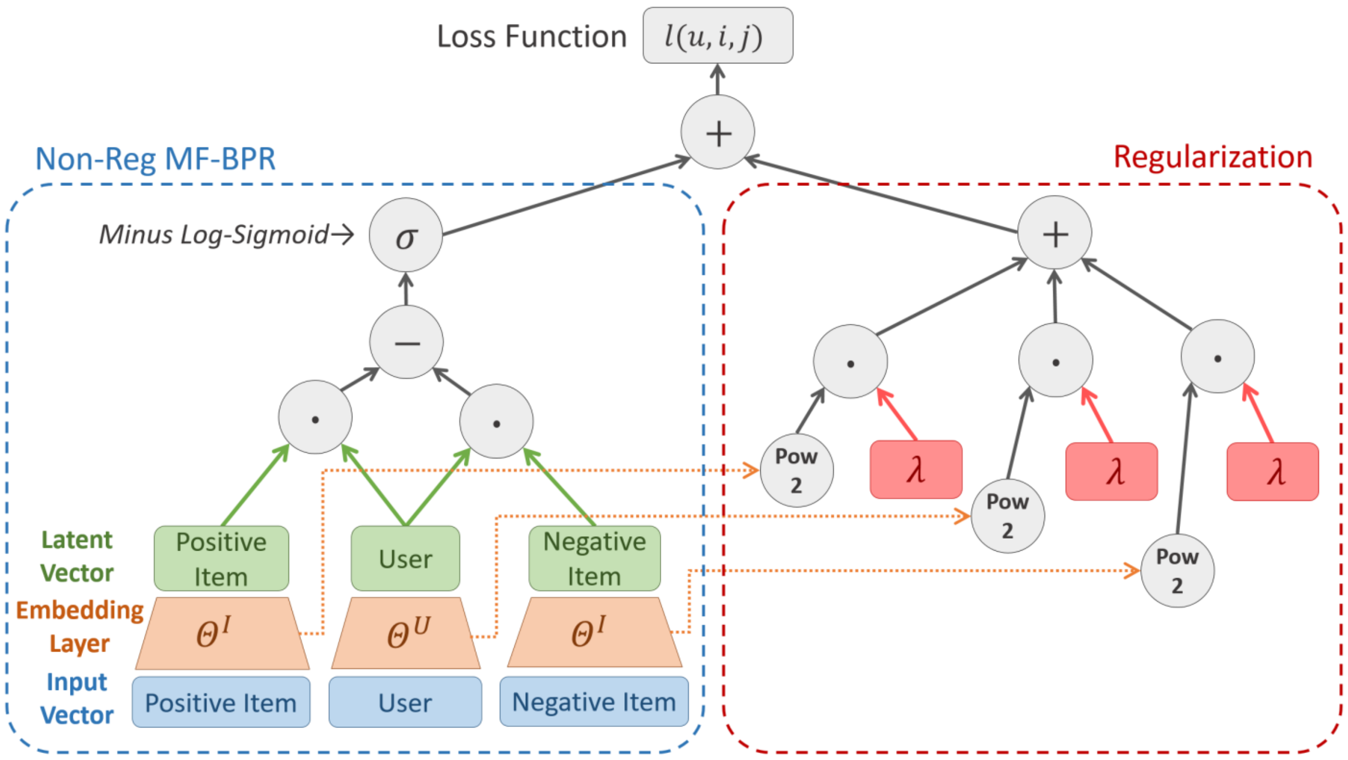
$$l = \tilde{l}(\theta) + \underbrace{\lambda||\theta||^2}_{\text{Penalty}}$$

Regularized Loss



What if we can do the regularization automatically?

Matrix Factorization with Bayesian Personalized Ranking criterion

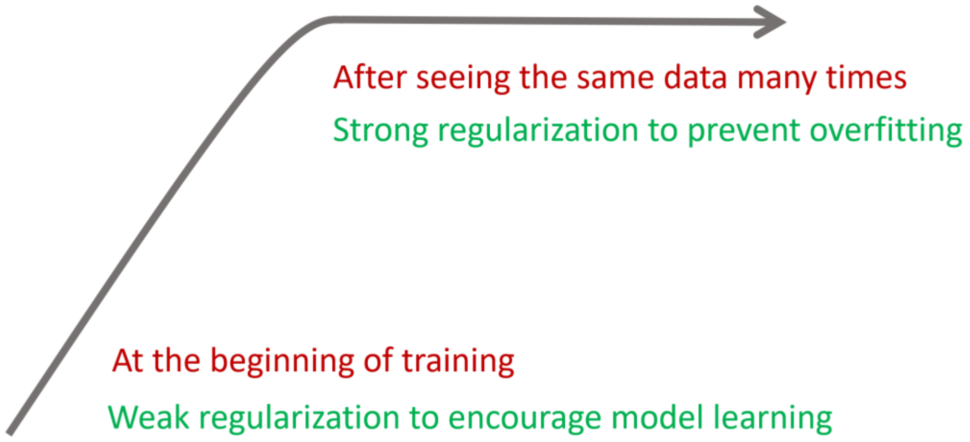


$$
\begin{aligned}
l_{S_T}(\Theta|\lambda) &= \tilde{l}_{S_T}(\Theta) + \Omega(\Theta|\lambda) \\
&= -\sum_{(u,i,j)\in S_T} \ln(\sigma(\hat{y}_{ui}(\Theta) - \hat{y}_{uj}(\Theta))) + \Omega(\Theta|\lambda)
\end{aligned}
$$

S_T: training set,
u: user,
i: positive item,
j: negative item,
$\hat{y}_{ui}$: score function parametrized by MF for (u, i) pair
$\hat{y}_{uj}$: score function parametrized by MF for (u, j) pair

Why hard to tune?

Hypothesis 1: fixed regularization strength throughout the process



Why hard to tune?

Hypothesis 2: compromise on regularization granularity

What we usually do to determine λ?

- Usually Grid Search or Babysitting $\rightarrow$ global λ

Fine-grained regularization works better

- But unaffordable if we use grid-search!
- Resort to automatic methods!

Model Characteristic
Different importance of each latent dimension

Dataset Characteristic
Diverse frequencies among users/items

Alternating Optimization

$$\min_{\Lambda} \sum_{\{(u',i',j') \in S_V\}} l(u',i',j' \mid \arg\min_{\Theta} \sum_{\{(u,i,j) \in S_T\}} l(u,i,j \mid \Theta, \Lambda))$$

At iteration t

Train the wheel!

- Fix Λ, Optimize Θ

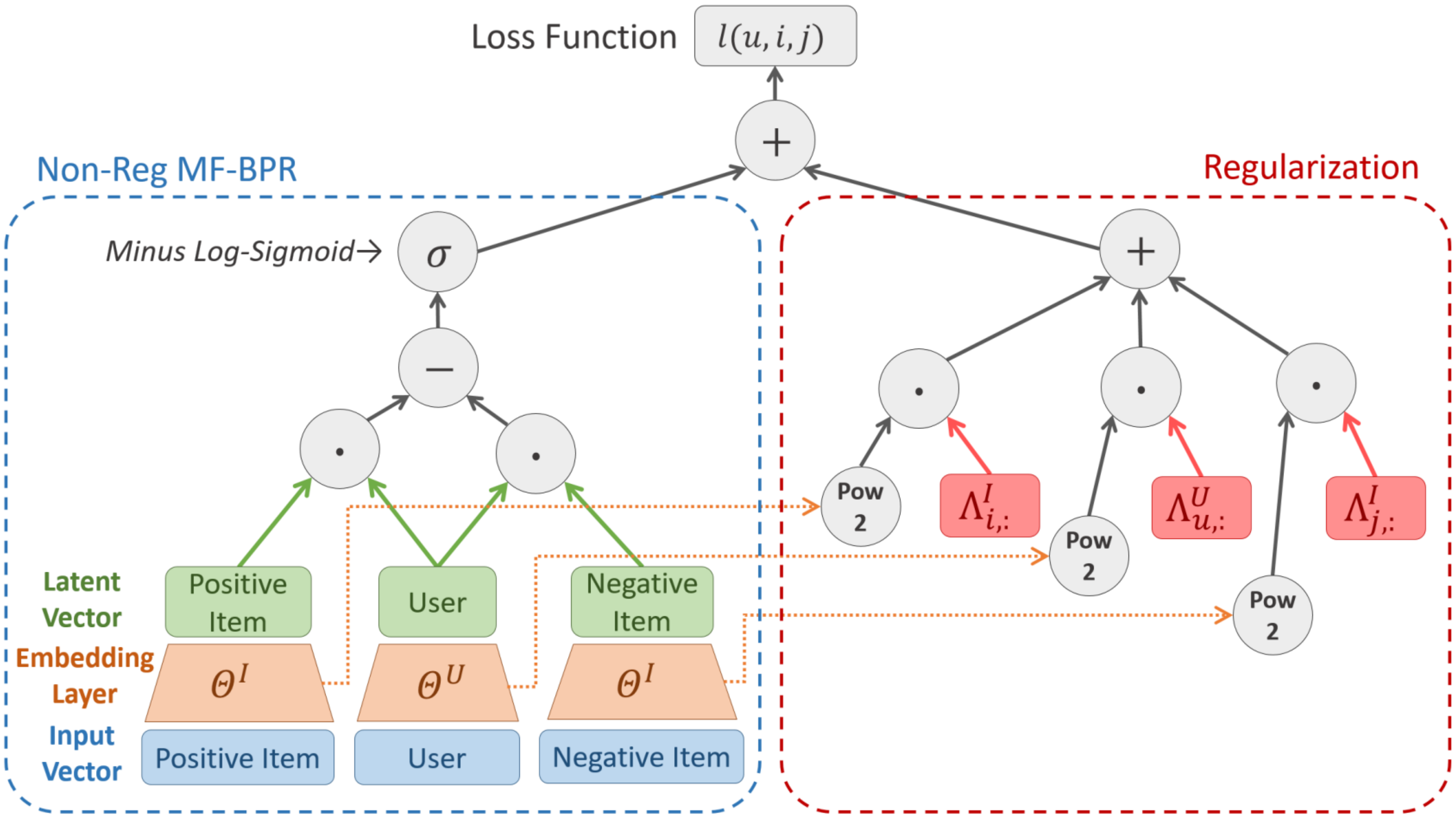
  $\rightarrow$ Conventional MF-BPR except λ is fine-grained now

Train the brake!

- Fix Θ, Optimize Λ

  $\rightarrow$ Find Λ which achieve the smallest validation loss

MF-BPR with fine-grained regularization



Fix Θ, Optimize Λ

Taking a greedy perspective, we look for Λ which can minimize the next-step validation loss

- If we keep using current Λ for next step, we would obtain $\bar{\Theta}_{t+1}$
- Given $\bar{\Theta}_{t+1}$, our aim is $\min_{\Lambda} l_{S_V}(\bar{\Theta}_{t+1})$ with the constraint of non-negative Λ

But how to obtain $\bar{\Theta}_{t+1}$ without influencing the normal Θ update?

- Simulate* the MF update!
  - Obtain the gradients by combining the non-regularized part and penalty part

$$\overline{\frac{\partial l_{S_T}}{\partial \Theta_t}} = \overline{\frac{\partial \tilde{l}_{S_T}}{\partial \Theta_t}} + \frac{\partial \Omega}{\partial \Theta_t}$$

Λ is the only variable here

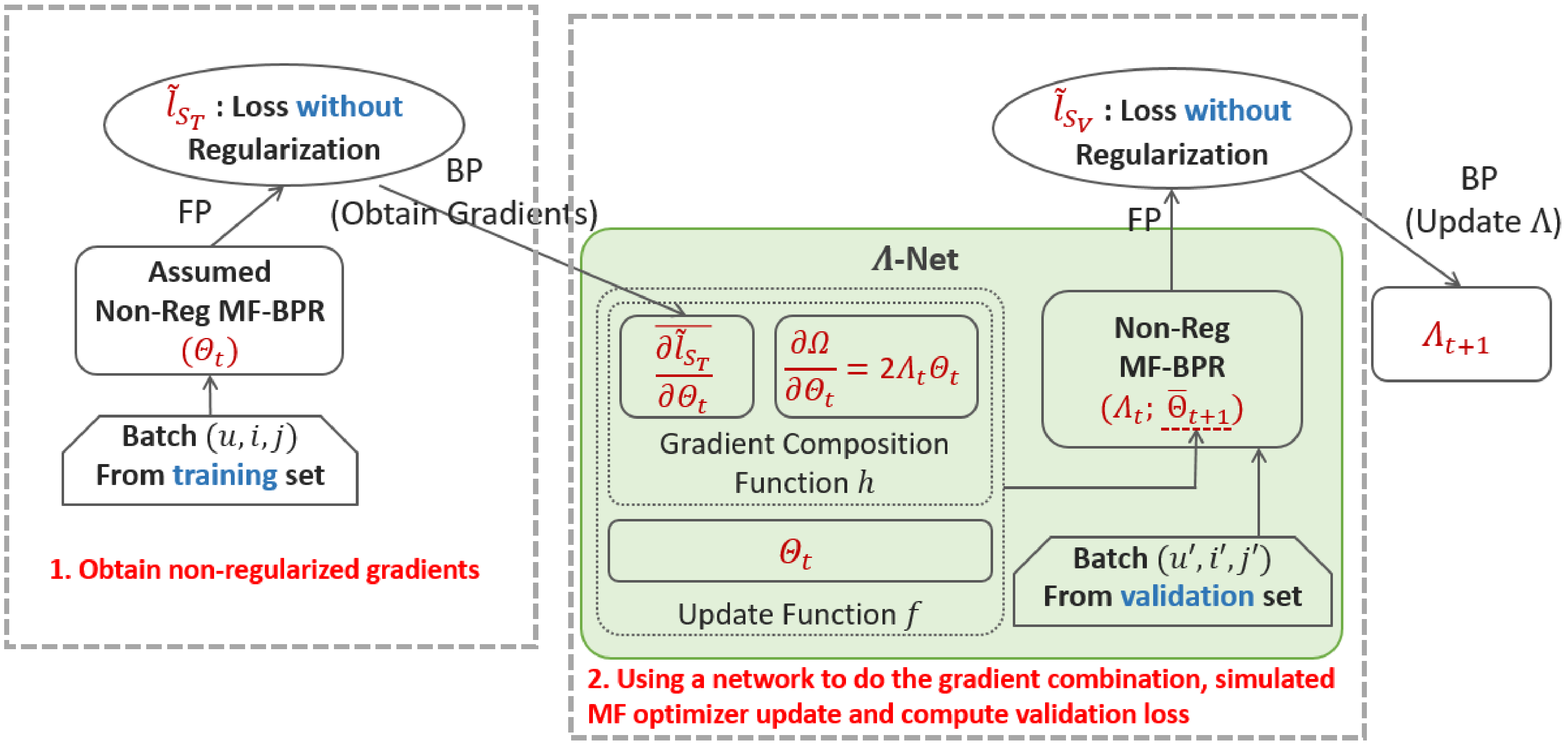
  - Simulate the operations that the MF optimizer would take

$$\bar{\Theta}_{t+1} = f(\Theta_t, \overline{\frac{\partial l_{S_T}}{\partial \Theta_t}})$$

f denotes the MF update function

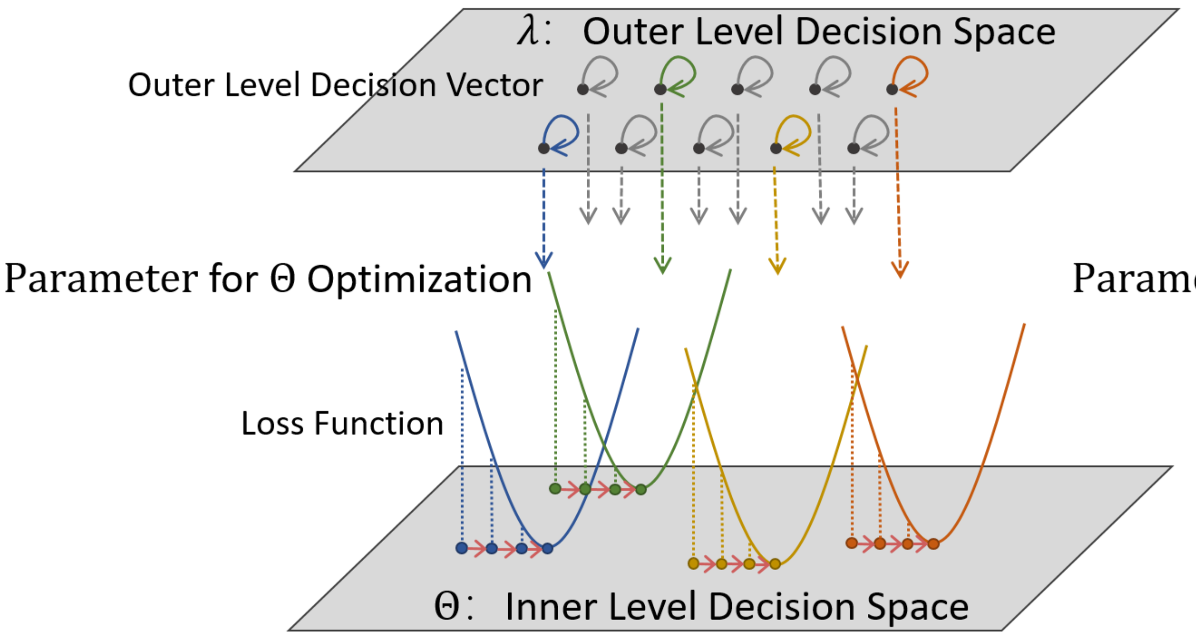
*: Using – over the letters to distinguish the simulated ones with normal ones

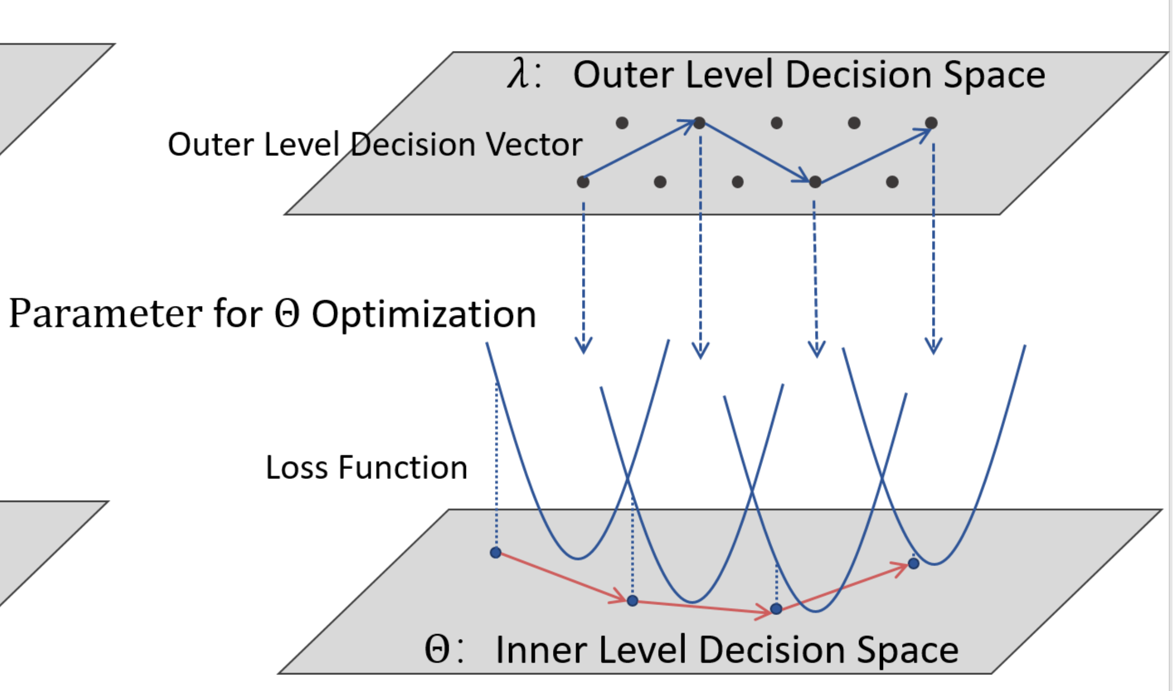
Fix Θ, Optimize Λ in Auto-Differentiation



Another Perspective on Regularization Tuning

$\Lambda - \text{trajectory}$





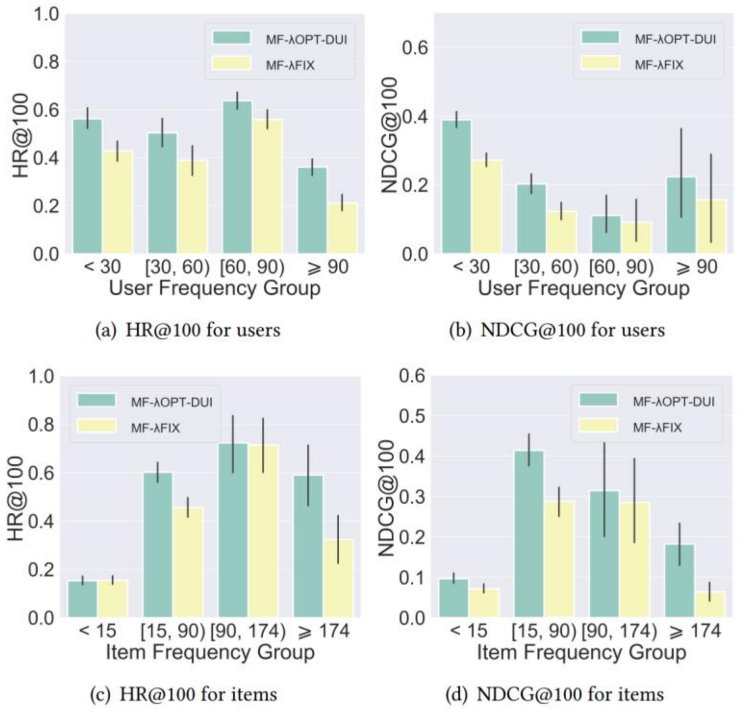
Result #1 Performance Comparison

Method	Amazon Food Review					MovieLens 10M				
	AUC	HR@50	HR@100	NDCG@50	NDCG@100	AUC	HR@50	HR@100	NDCG@50	NDCG@100
SGDA [26]	0.8130	0.1786	0.3857	0.1002	0.1413	0.9497	0.2401	0.3706	0.0715	0.0934
AMF [15]	0.8197	0.3541	0.4200	0.2646	0.2552	0.9495	0.2625	0.3847	0.0787	0.0985
NeuMF [16]	0.8103	0.3537	0.4127	0.2481	0.2218	0.9435	0.2524	0.3507	0.0760	0.0865
MF-λFix	0.8052	0.3482	0.4163	0.2251	0.2217	0.9497	0.2487	0.3779	0.0727	0.0943
MF-λOpt -D	0.8109	0.2134	0.3910	0.1292	0.1543	0.9501	0.2365	0.3556	0.0715	0.0909
-DU	0.8200	0.3694	0.4814	0.2049	0.2570	0.9554	0.2743	0.4109	0.0809	0.1031
-DI	0.8501	0.2966	0.4476	0.1642	0.2039	0.9516	0.2648	0.3952	0.0804	0.1013
-DUI	**0.8743**	**0.4470**	**0.5251**	**0.2946**	**0.2920**	**0.9575**	**0.3027**	**0.4367**	**0.0942**	**0.1158**

1. **Overall:** MF-λ**Opt**-DUI achieves the best performance, demonstrating the effect of fine-grained adaptive regularization. (approx. 10%-20% gain over baselines)
2. **Dataset:** Performance improvement on *Amazon Food Review* is larger than that on MovieLens 10M. This might due to the dataset size and density. *Amazon Food Review* has a smaller number of interactions. Complex models like NeuMF or AMF wouldn't be at their best condition. Also, smart regularization is necessary for different users/items, explaining why SGDA and MF-λ**Opt**-DUI performs worse. In our experiments, we also observe more fluctuation of training curves on *Amazon Food Review* for the adaptive λ methods.
3. **Variants of regularization granularity:** Although MF-λ**Opt**-DUI consistently performs best, MF-λ**Opt**-DU/ or MF-λ**Opt**-DU doesn't provide as much gain over the baselines, which might be due to merely addressing the regularization for partial model parameters.

Result #2: Sparseness & Activeness

Does the performance improvement come from addressing different users/items?



(a) HR@100 for users (b) NDCG@100 for users

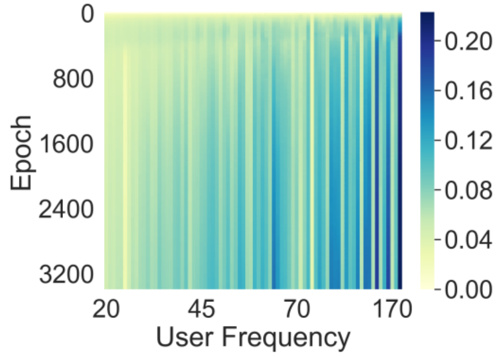
(c) HR@100 for items (d) NDCG@100 for items

Group users/items according to their frequencies and check the recommendation performance of each group, using *Amazon Food Review* as an example; black line indicates variance

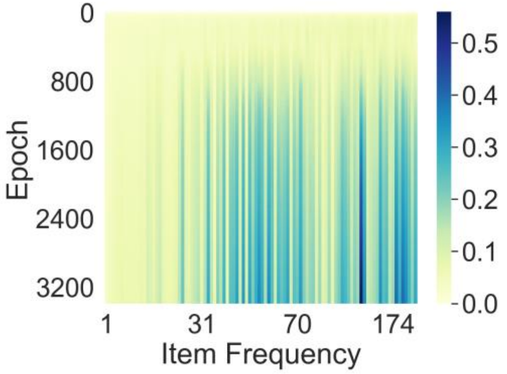
1. **User with varied frequencies:** For users, MF-λ**Opt**-DUI lifts HR@100 and NDCG@100. Compared to global MF-λ**Opt**-DUI , fine-grained regularization addressing users of different frequencies better.
2. **Item with varied frequencies:** For items, similar lift can be observed except that only slight lift for HR@100 of the <15 group and [90, 174) group.
3. **Variance within the same group:** Although the average lift can be observed across groups, the variance demonstrate that there are factors other than frequency which influence the recommendation performance.

Result #3: Analysis of λ-trajectory

How does MF-λ**Opt**-DUI address different users/items?



(a) For users on Amazon Food Review



For each user/item, we cache the λ from Epoch 0 to Epoch 3200 (almost converged). λs of users/items with the same frequency are averaged. The darker colors indicates larger λ.

1. **λ vs. user frequency:** At the same training stage, Users with higher frequencies are allocated larger λ. Active users have more data and the model learns from the data so quickly that it might get overfitting to them, making strong regularization necessary. A global λ , either small or large, would fail to satisfy both active users and sparse users.
2. **It vs. item frequency:** Similar as the analysis of users though not so obvious. Items with higher frequencies are allocated larger λ.
3. **λ vs. training progress:** As training goes on, λs gets larger gradually. Hence stronger regularization strengths are enforced at the late stage of training while the model is allowed to learn sufficiently at the beginning.

Conclusion

- AutoML can
  - help choosing models
  - select or generate data/feature
  - and even help tune hyper-parameters.