Automated Machine Learning for Recommendations: Fundamentals and Advances

Xiangyu Zhao
City University of Hong Kong
xianzhao@cityu.edu.hk

Wenqi Fan
The Hong Kong Polytechnic University
wenqifan03@gmail.com

Huifeng Guo
Huawei Noah's Ark Lab
huifeng.guo@huawei.com

Bo Chen
Huawei Noah's Ark Lab
chenbo116@huawei.com

Yejing Wang
City University of Hong Kong
adave631@gmail.com

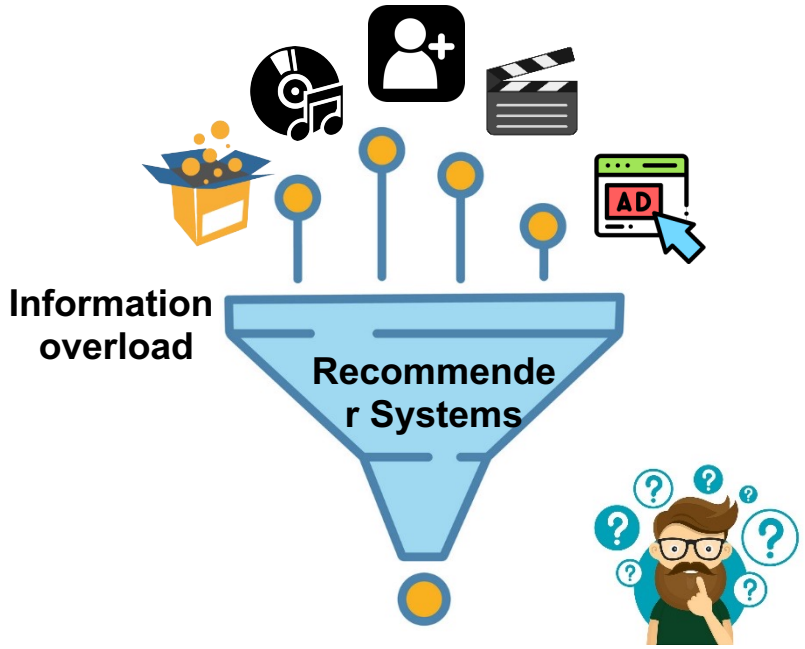
Ruiming Tang
Huawei Noah's Ark Lab
tangruiming@huawei.com

Homepage: https://advanced-recommender-systems.github.io/AutoML-Recommendations/

Table of Contents

Recommender Systems

Age of Information



Recommend item X to

Items can be: Products, News, Movies, Videos, Friends, etc.

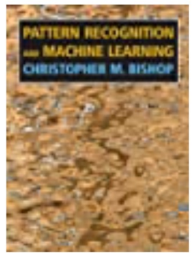
Recommender Systems

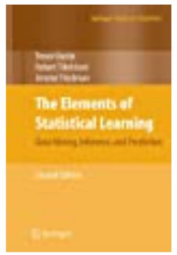
Recommendation has been widely applied in online services:
- **E-commerce**, Content Sharing, Social Networking ...

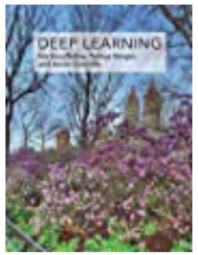
amazon ebay 淘宝网 Taobao.com JD.COM

Product Recommendation

Frequently bought together



A + 

B + 

C

Total price: $208.9

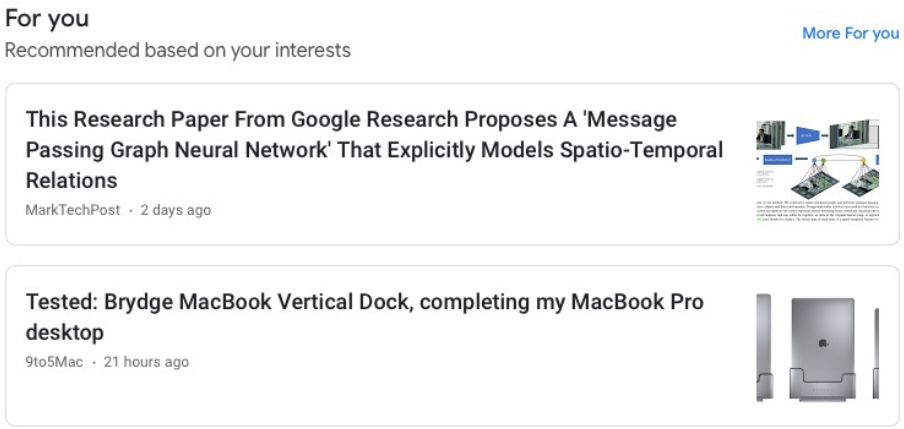
Add all three to Cart

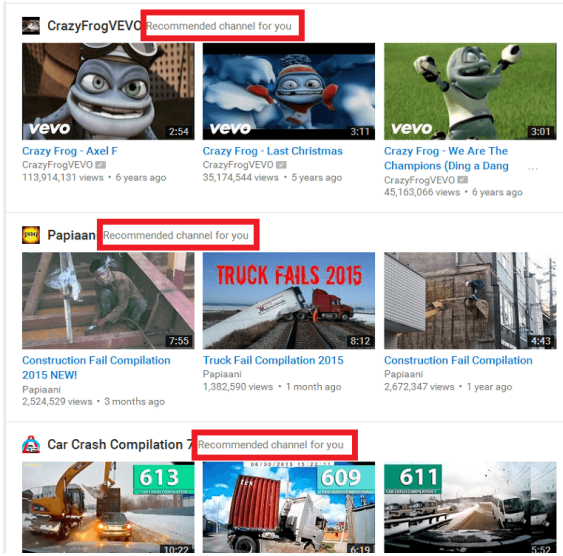
Add all three to List

Recommender Systems

Recommendation has been widely applied in online services:
- E-commerce, **Content Sharing**, Social Networking ...

News/Video/Image Recommendation





Recommender Systems

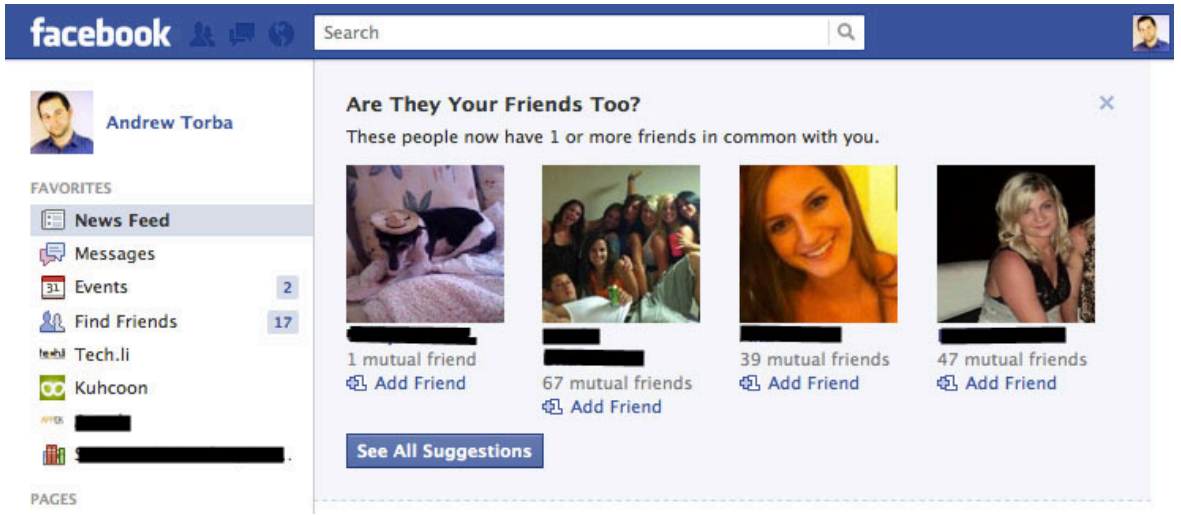
Recommendation has been widely applied in online services:
- E-commerce, Content Sharing, **Social Networking** ...

facebook

新浪微博 weibo.com

LinkedIn

Friend Recommendation



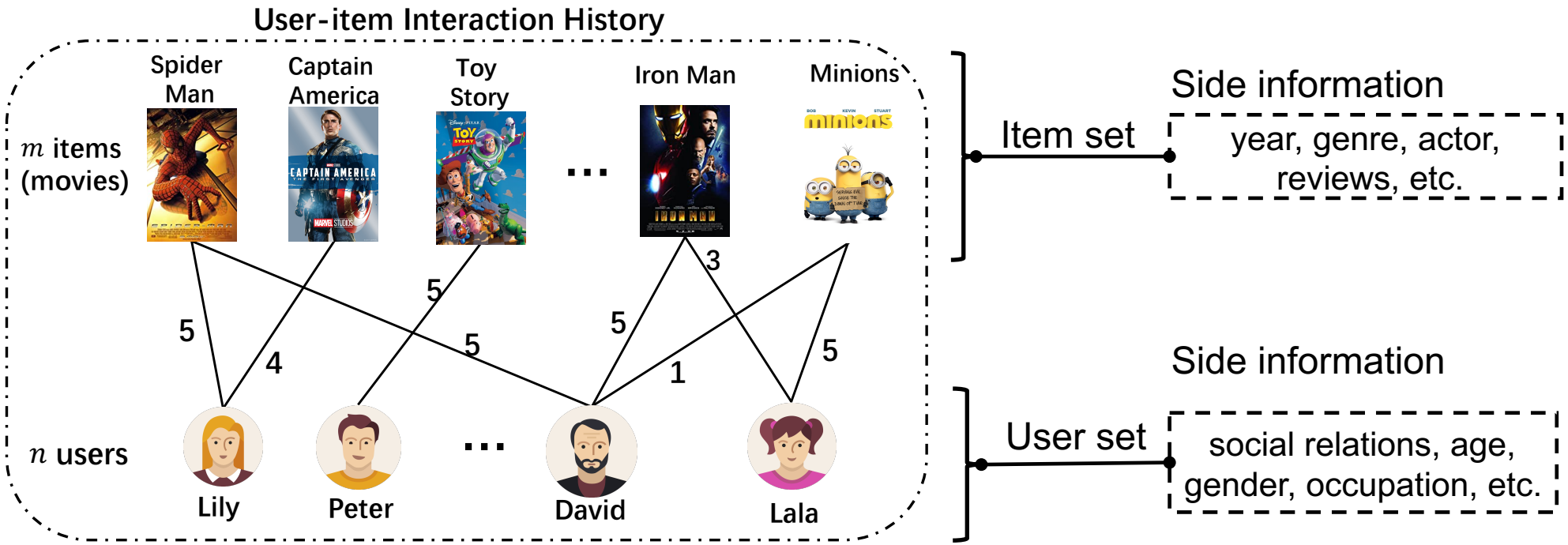
Problem Formulation

INPUT

Historical user-item interactions or additional side information (e.g., social relations, item's knowledge, etc.)

OUTPUT

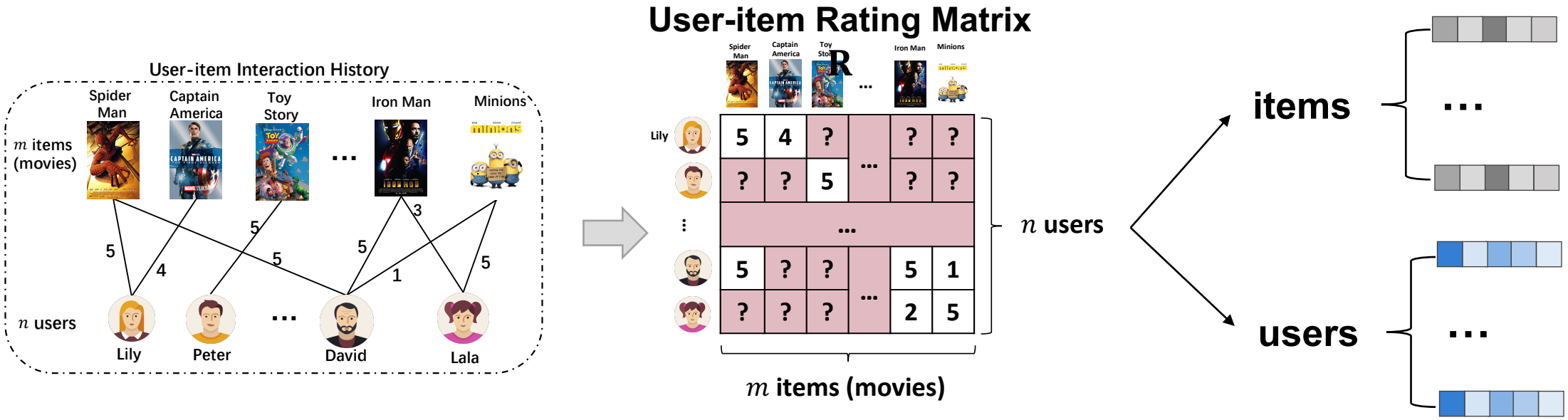
Predict how likely a user would interact with a target Item (e.g., click, view, or purchase)



Recommender Systems

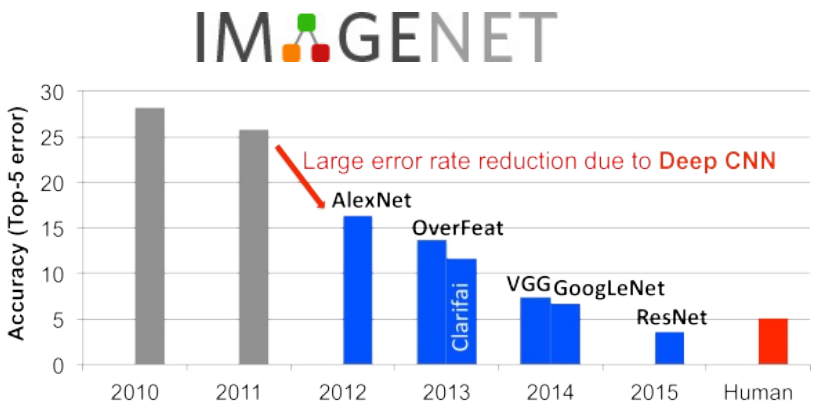
- Collaborative Filtering (CF) is the most well-known technique for recommendation.
  - Similar users (with respect to their historical interactions) have similar preferences.
  - Modelling users' preference on items based on their past interactions (e.g., ratings and clicks).

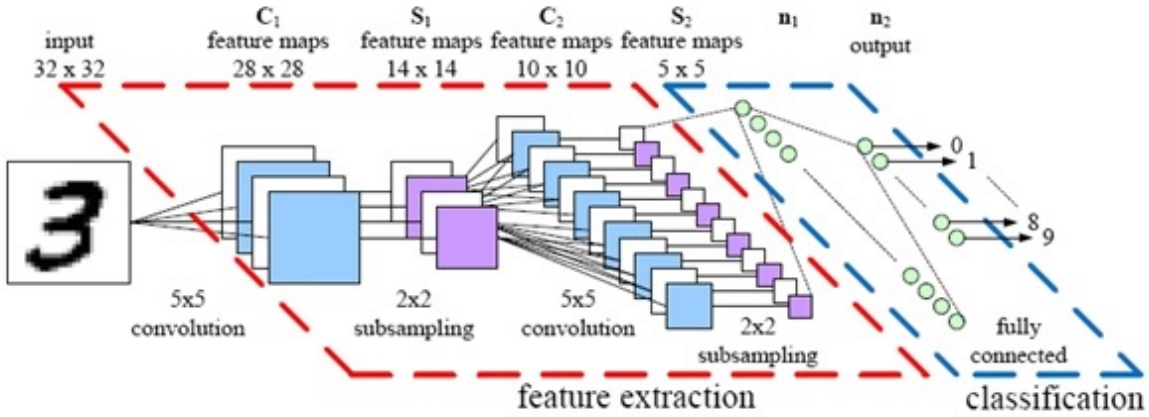
- Learning representations of users and items is the key of CF.



Task: predicting missing movie ratings in Netflix.

Deep Learning is Changing Our Lives





Recommender Systems Architecture

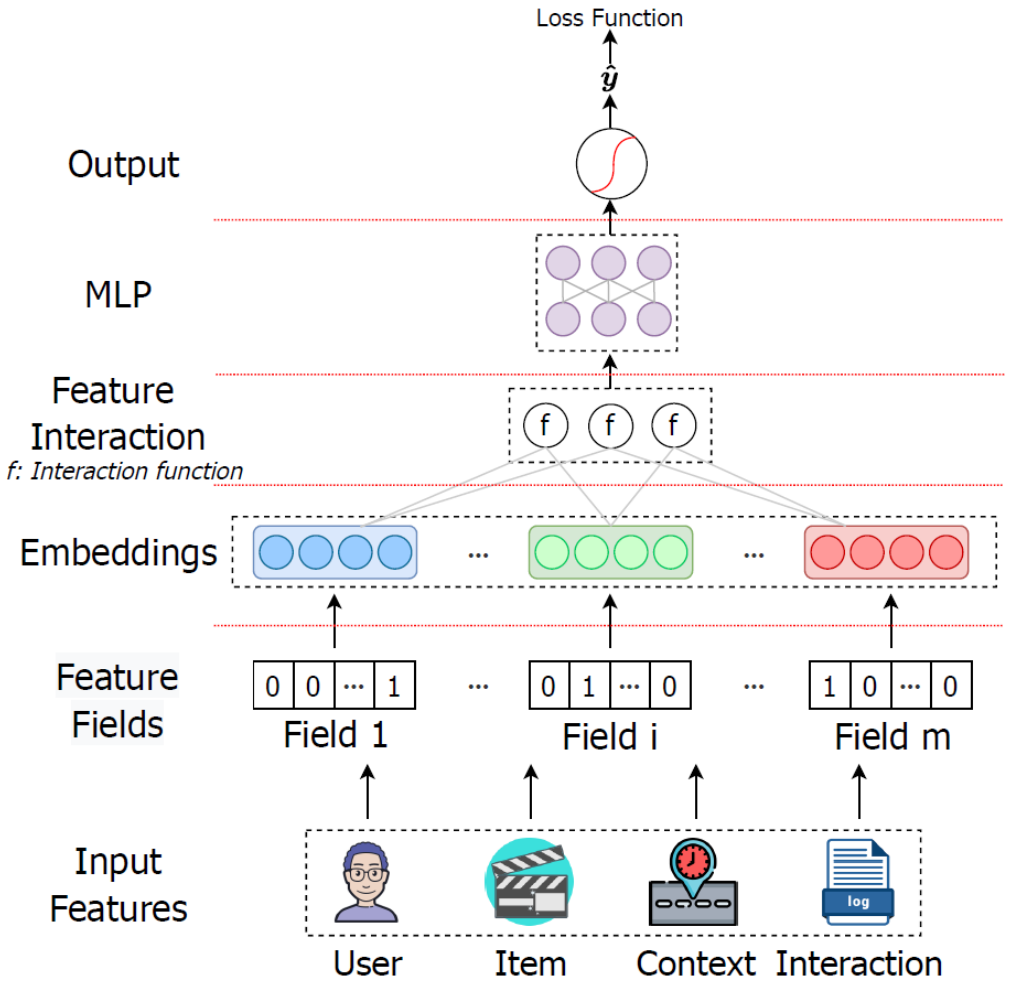
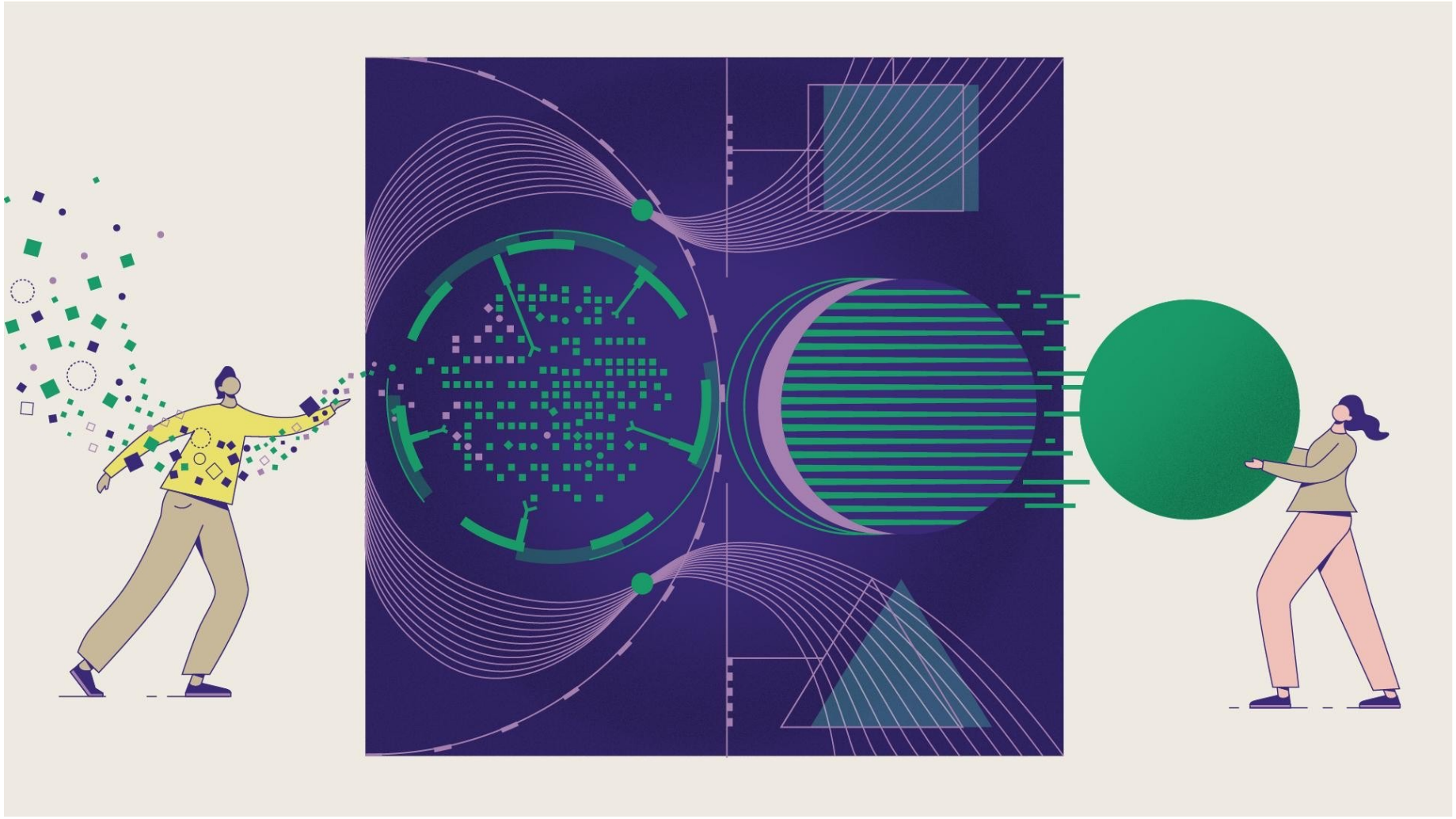


Table of Contents

Why AutoML?



Success of Machine Learning

Astronomy

Robotic

Teaching

Material Design

Creative Arts

Energy

Game Play

Search

Chemistry

Health Care

Physics

Image Recognition

Weather Prediction

Product Recommendation

Drug Discovery

Service

Financial Services

Manufacturing

Retail

Traffic Prediction

Credit Assignment

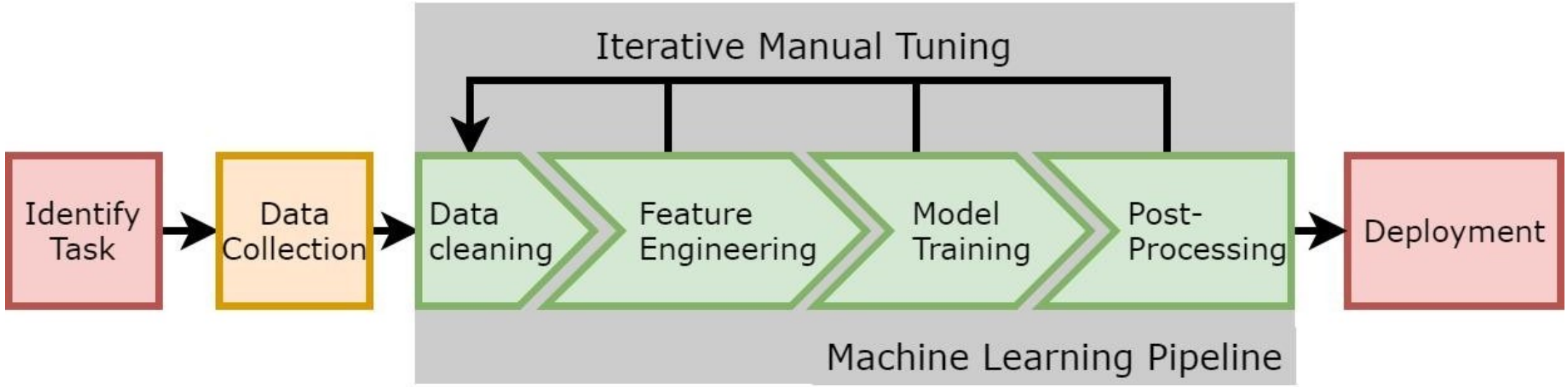
Media

Maintenance Prediction

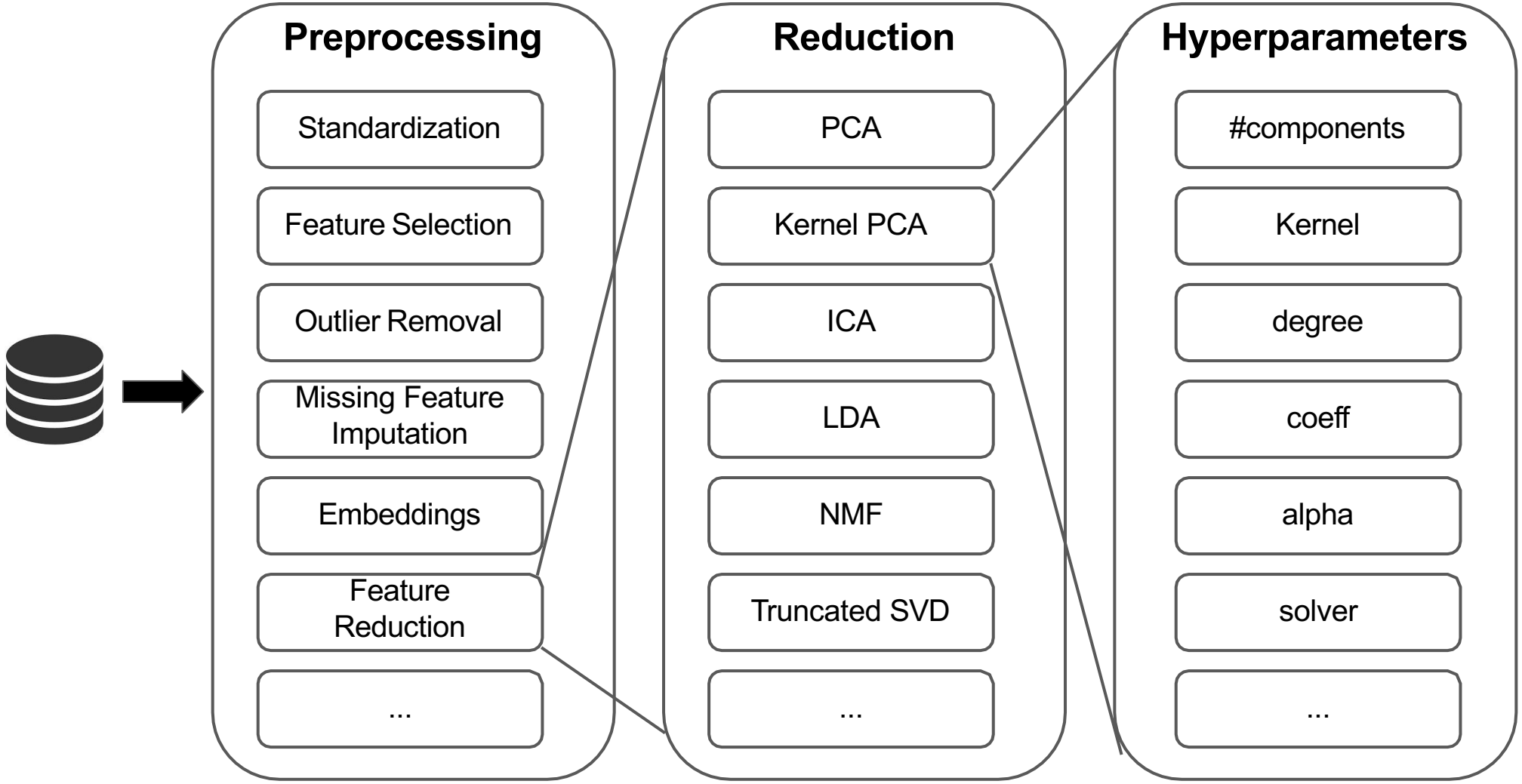
Social Media

Summary Generation

Machine Learning Pipeline

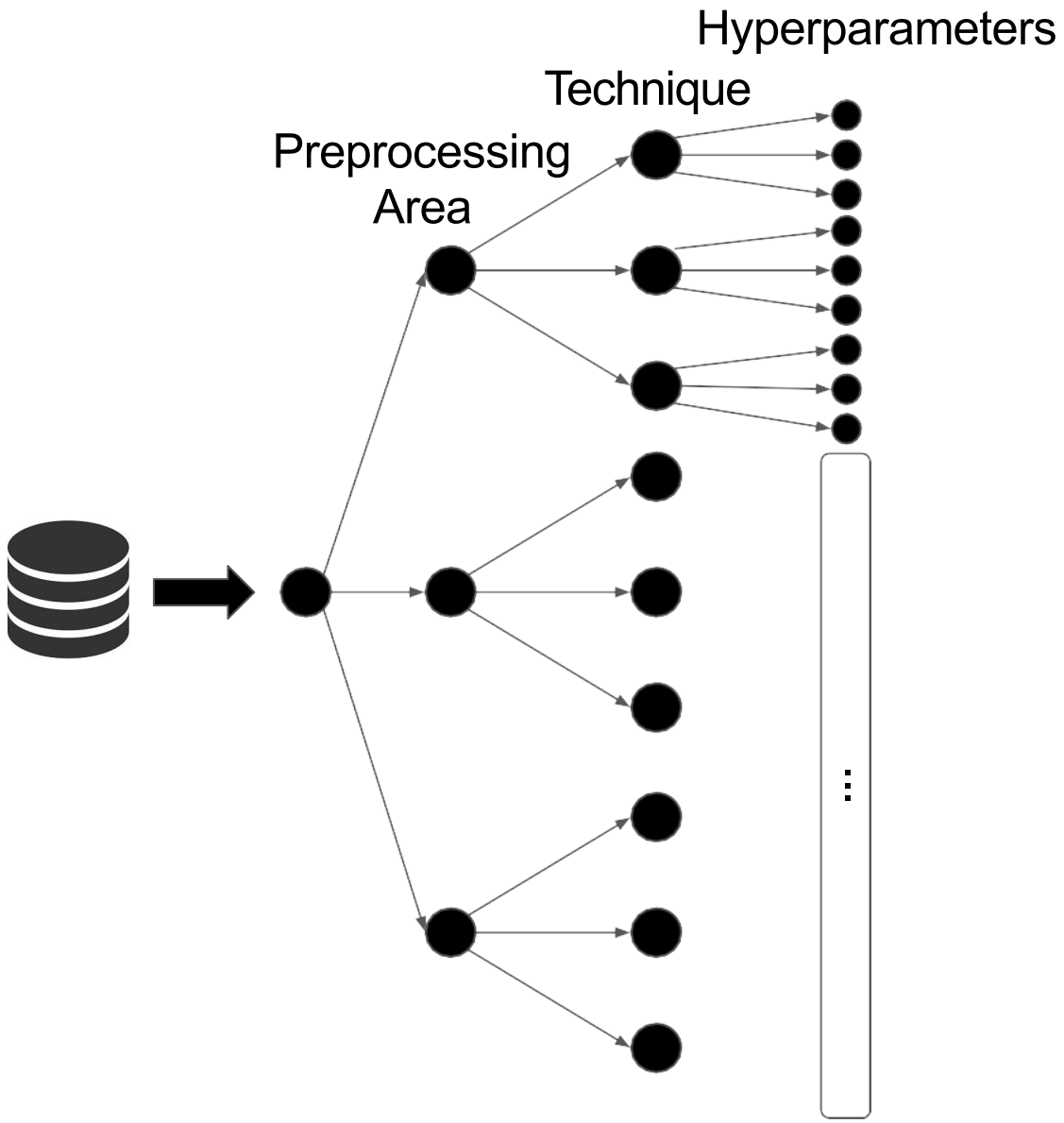


Preprocessing?



→ **We might want more than 1 data preprocessor!**

Complexity of the Preprocessing



- Naive Assumptions:
  only 3 decisions at each level
- **Possible options**: 3 x 3 x 3 = **27**

- More realistic assumption:
  at least 10 decisions at leach level
- **Possible options:** 10 x 10 x 10 = **1000**

- Choose 3 preprocessors instead of 1
  → 1000 x 1000 x 1000 =
  1 000 000 000

- Still naive!
  → Hyperparameters are often continuous and not discrete
  → **infinite amount of settings!**

Classification Algorithms

LARS

Least Squares

Kernel SVM

Elastic-net

Recurrent Network

Linear SVM

Least-Angle

Lasso

ResNet

Poly.SVM

Feedforward Network

Ridge

Bayesian Regression

Naive Bayes

Logistic Regression

SGD

LGB

Gaussian Process

Nearest Neighbor

Boosting Trees

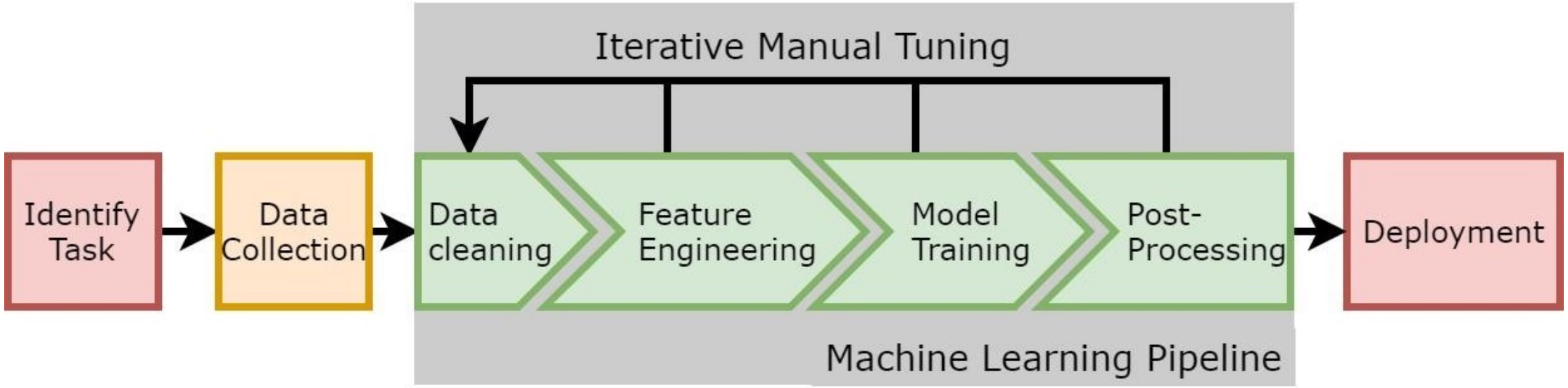
XGB

Decision Trees

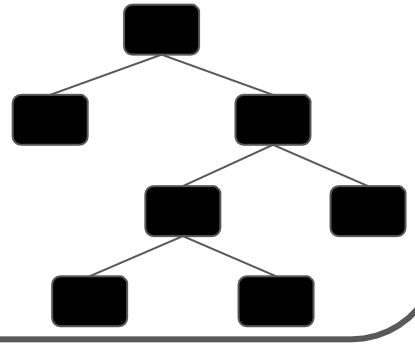
Random Forests

→ **There are more than 100 classification algorithms!**
→ **Each of these has 2-50 hyperparameters**

Challenges in Designing ML Pipelines



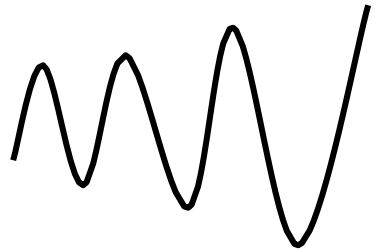
Complex Search Space



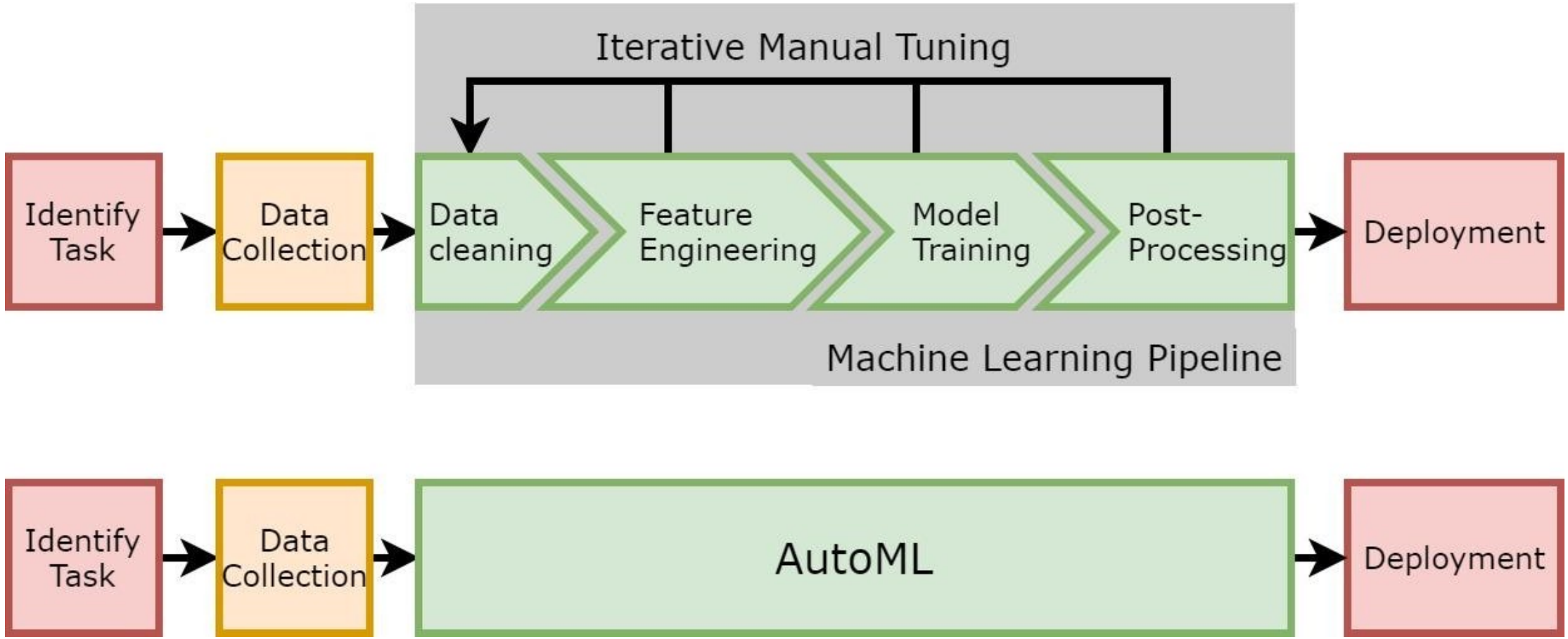
Black-Box Problem

Expensive Evaluations

Noise on observations



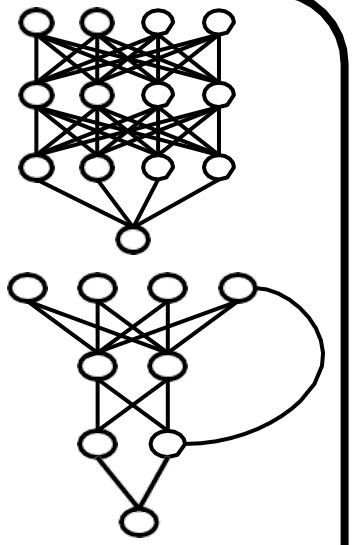
From Manual ML to Automated ML



Design Decisions by AutoML

classifier	#λ
AdaBoost (AB)	4
Bernoulli naïve Bayes	2
decision tree (DT)	4
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	4
linear SVM	4
kernel SVM	7
multinomial naïve Bayes	2
passive aggressive	3
QDA	2
random forest (RF)	5
Linear Class. (SGD)	10

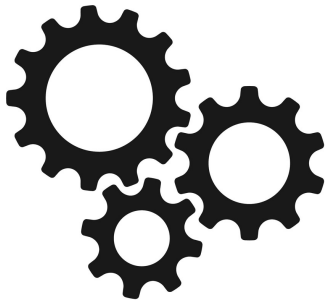
Algorithms



Architecture Design

preprocessor	#λ
extreml. rand. trees prepr.	5
fast ICA	4
feature agglomeration	4
kernel PCA	5
rand. kitchen sinks	2
linear SVM prepr.	3
no preprocessing	-
nystroem sampler	5
PCA	2
polynomial	3
random trees embed.	4
select percentile	2
select rates	3
one-hot encoding	2
imputation	1
balancing	1
rescaling	1

Pre-processing



Hyper-parameters

...

Neural Architecture Search (NAS)

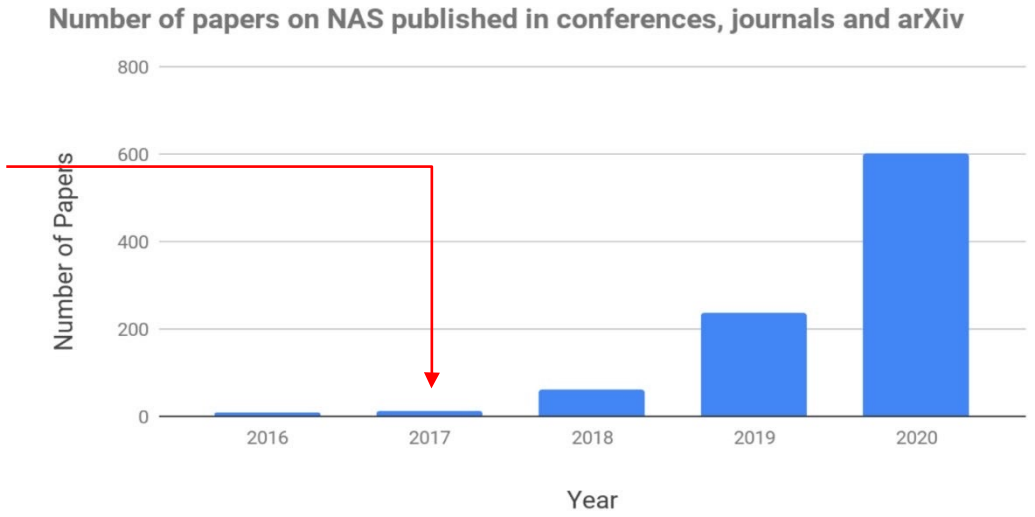
- Find neural architecture A such that deep learning works best for given data
  - Measured by validation error of architecture A with trained weights $w^*(A)$

$$\min_{A \in \mathcal{A}} \mathcal{L}_{\text{val}}(w^*(A), A)$$

$$\text{s.t. } w^*(A) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, A)$$

- Famously tackled by

  reinforcement learning [Zoph & Le, ICLR 2017]
  - 12.800 architectures trained fully
  - 800 GPUs for 2 weeks (about $60.000 USD)



[Zoph, 2017] **B. Zoph** *et al.*, Neural Architecture Search with Reinforcement Learning, *ICLR*, 2017.

NAS is Exploding!

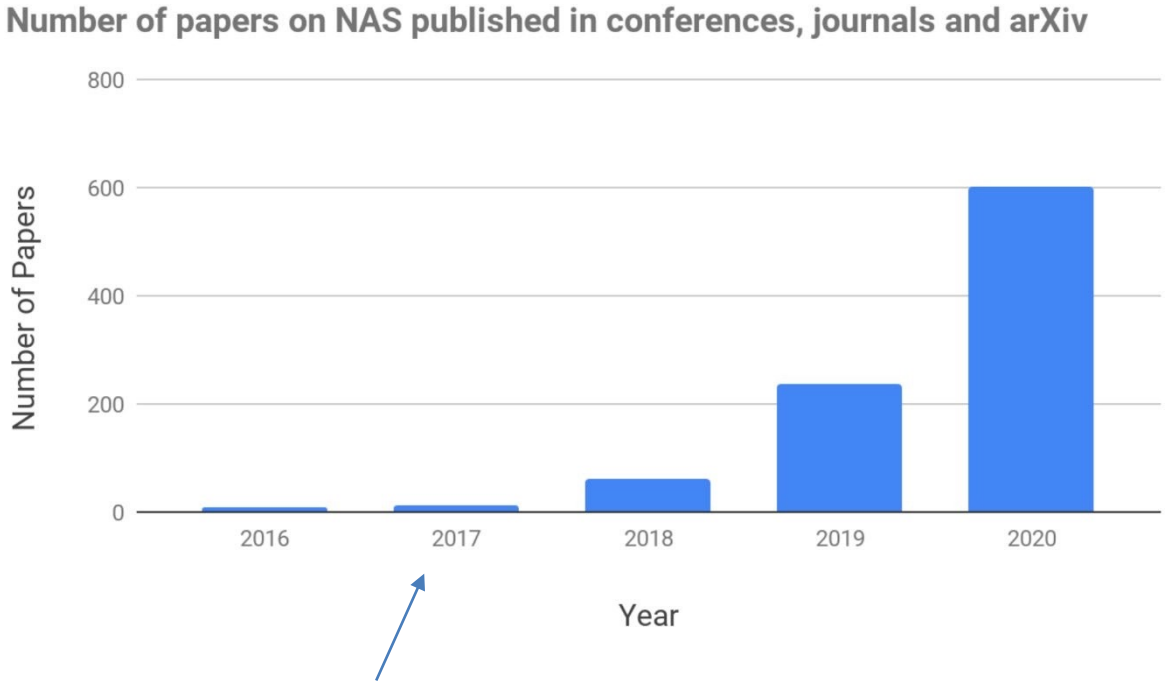
Journal of Machine Learning Research 20 (2019) 1-21 Submitted 9/18; Revised 3/19; Published 3/19

Neural Architecture Search: A Survey

Thomas Elsken THOMAS.ELSKEN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence
71272 Renningen, Germany
and University of Freiburg

Jan Hendrik Metzen JANHENDRIK.METZEN@DE.BOSCH.COM
Bosch Center for Artificial Intelligence
71272 Renningen, Germany

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg
79110 Freiburg, Germany



Neural Architecture Search with Reinforcement Learning [Zoph & Le, ICLR 2017]

Framework: Trial and Update

- Almost all NAS algorithms are based on the "trial and update" framework
  - Starting with a set of initial architectures (*e.g.*, manually defined) as individuals
  - Assuming that better architectures can be obtained by slight modification
  - Applying different operations on the existing architectures
  - Preserving the high-quality individuals and updating the individual pool
  - Iterating till the end

- Three fundamental requirements
  - The building blocks: defining the search space (dimensionality, complexity, *etc.*)
  - The representation: defining the transition between individuals
  - The evaluation method: determining if a generated individual is of high quality

Framework: Building Blocks

- Building blocks are like basic *genes* for these individuals
- Some examples here
  - Genetic CNN: only 3×3 convolution is allowed to be searched (followed by default BN and ReLU operations), 3×3 pooling is fixed
  - NASNet: 13 operations shown below
  - PNASNet: 8 operations, removing those never-used ones from NASNet
  - ENASNet: 6 operations
  - DARTS: 8 operations

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

Framework: Search

- Finding new individuals that have potentials to work better
  - Heuristic search in the large space
- Two mainly applied methods: the genetic algorithm and reinforcement learning
  - Both are heuristic algorithms applied to the scenarios of a large search space and limited ability to explore every single element in the space
  - A fundamental assumption: both of these heuristic algorithms can preserve good genes and based on which discover possible improvements
- Also, it is possible to integrate architecture search to network optimization
  - These algorithms are often much faster

[Real, 2017] **E. Real** *et al.*, Large-Scale Evolution of Image Classifiers, *ICML*, 2017.
[Xie, 2017] **L. Xie** *et al.*, Genetic CNN, *ICCV*, 2017.
[Zoph, 2018] **B. Zoph** *et al.*, Learning Transferable Architectures for Scalable Image Recognition, *CVPR*, 2018.
[Liu, 2018] **C. Liu** *et al.*, Progressive Neural Architecture Search, *ECCV*, 2018.
[Pham, 2018] **H. Pham** *et al.*, Efficient Neural Architecture Search via Parameter Sharing, *ICML*, 2018.
[Liu, 2019] **H. Liu** *et al.*, DARTS: Differentiable Architecture Search, *ICLR*, 2019.

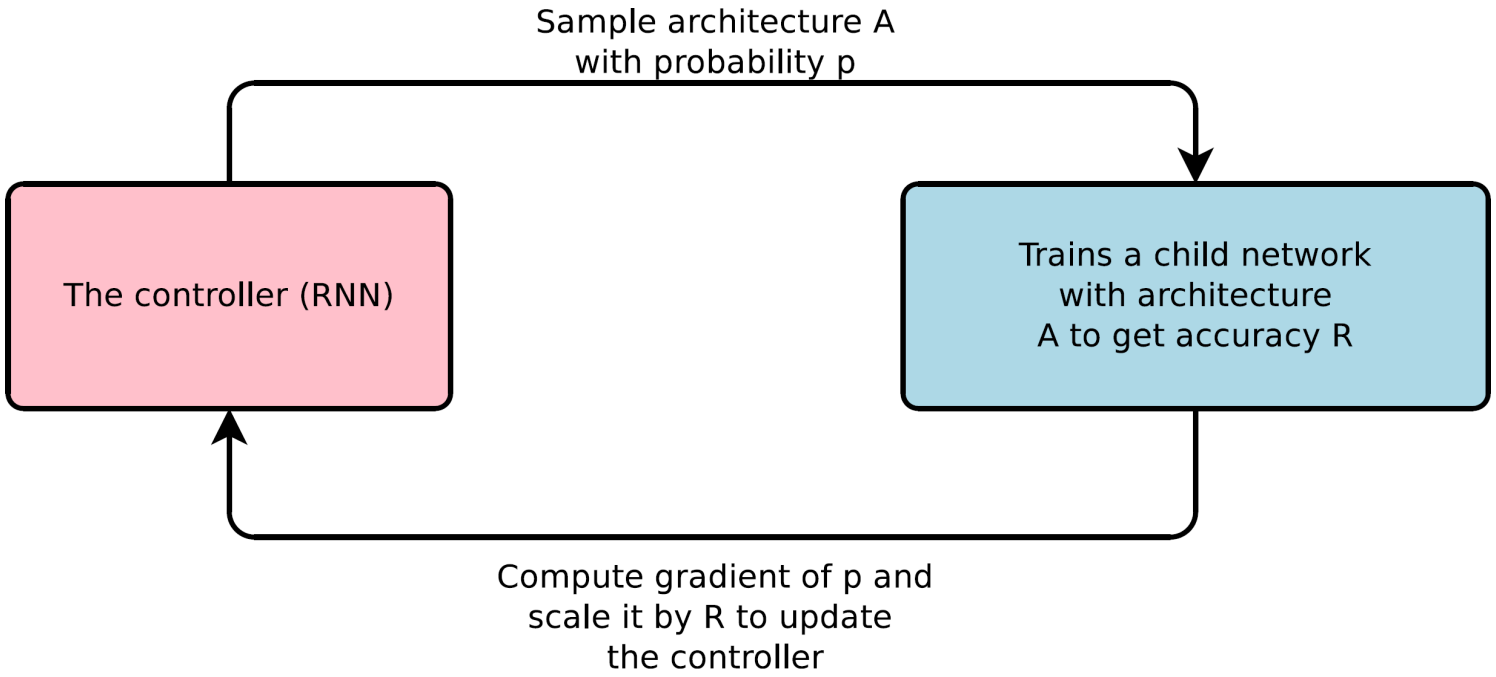
Framework: Evaluation

- Evaluation aims at determining which individuals are good and to be preserved
- Conventionally, this was often done by training a network from scratch
  - This is extremely time-consuming, so researchers often train NAS on a small dataset like CIFAR and then transfer the found architecture to larger datasets like ImageNet
  - Even in this way, the training process is really slow: Genetic-CNN requires 17 GPU-days for a single training process, and NAS-RL requires more than 20,000 GPU-days
- Efficient methods were proposed later
  - Ideas include parameter sharing (without the need of re-training everything for each new individual) and using a differentiable architecture (joint optimization)
  - Now, an efficient search process on CIFAR can be reduced to a few GPU-hours, though training the searched architecture on ImageNet is still time-consuming

NAS with Reinforcement Learning

- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands:

    800 GPUs for 3-4 weeks, 12.800 architectures trained

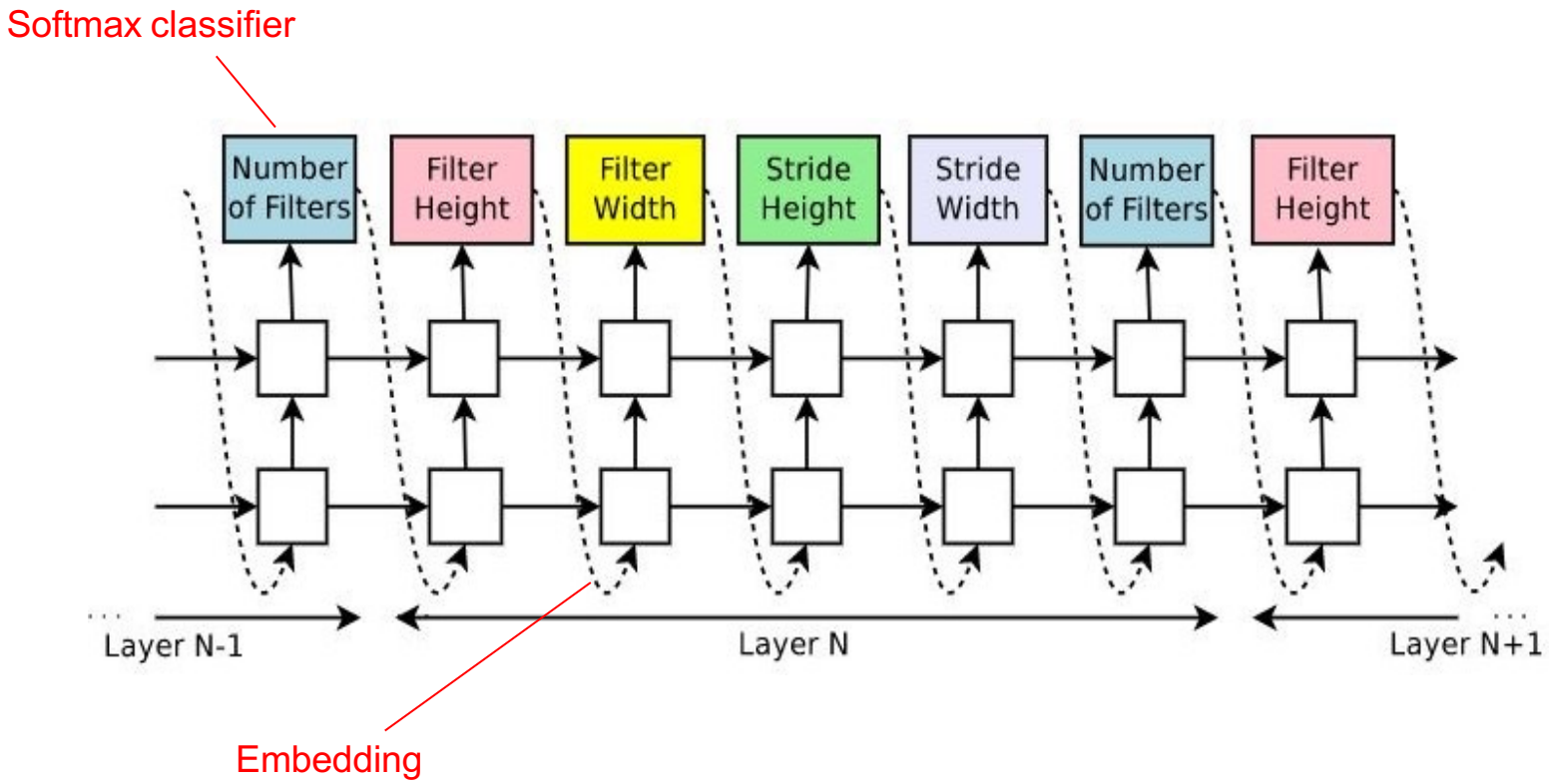


NAS with Reinforcement Learning

- Architecture of neural network represented as string e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”]

[Zoph & Le, ICLR 2017]

- Controller (RNN) generates string that represents architecture



Training with REINFORCE

Parameters of Controller RNN

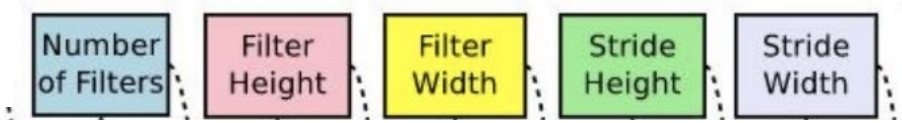
Accuracy of architecture on held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN viewed as a sequence of actions

NAS as Hyperparameter Optimization

- Architecture of neural network represented as string e.g., [“filter height: 5”, “filter width: 3”, “# of filters: 24”] [Zoph & Le, ICLR 2017]
- We can simply treat these as categorical parameters
  - E.g., 25 cat. parameters for each of the 2 cells in [Zoph et al, CVPR 2018]



[Zoph, 2018] **B. Zoph** *et al.*, Learning Transferable Architectures for Scalable Image Recognition, *CVPR*, 2018.

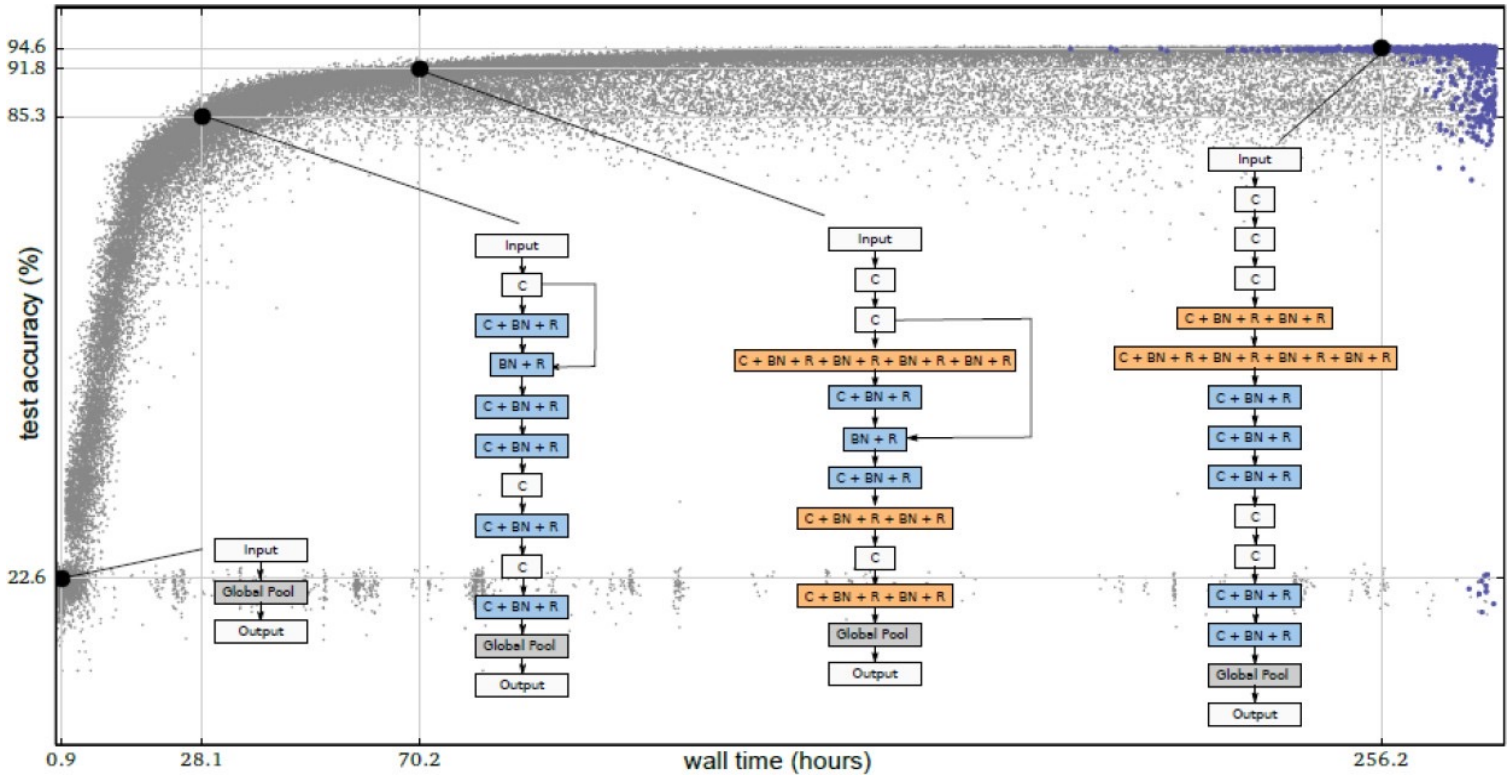
NAS with Evolution

- Neuroevolution

  (already since the 1990s [Angeline et al., 1994; Stanley and Miikkulainen, 2002])

  - Mutation steps, such as adding, changing or removing a layer

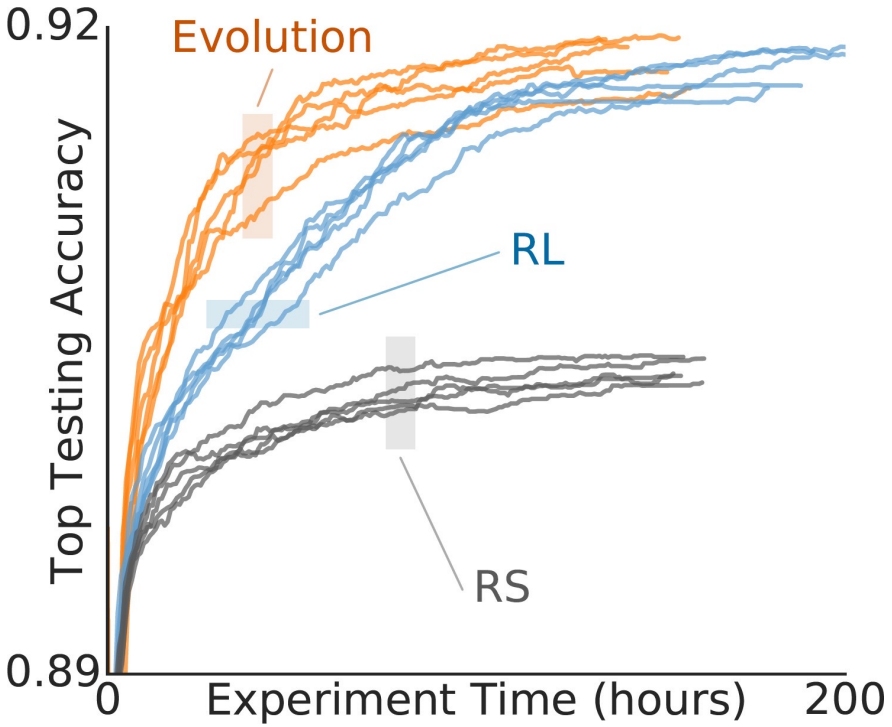
    [Real et al., ICML 2017; Miikkulainen et al., arXiv 2017]



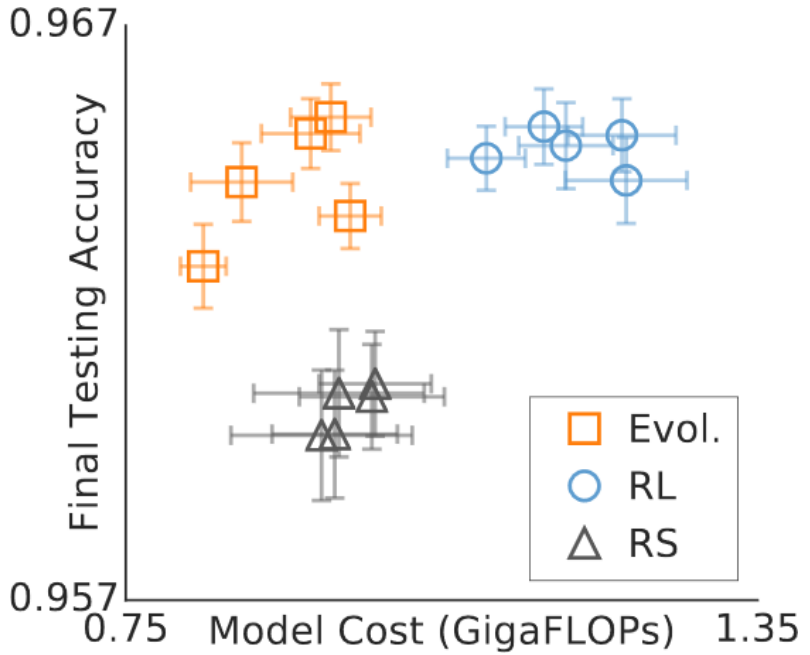
RL vs. Evolution vs. Random Search

[Real et al., AAAI 2019]

during architecture search



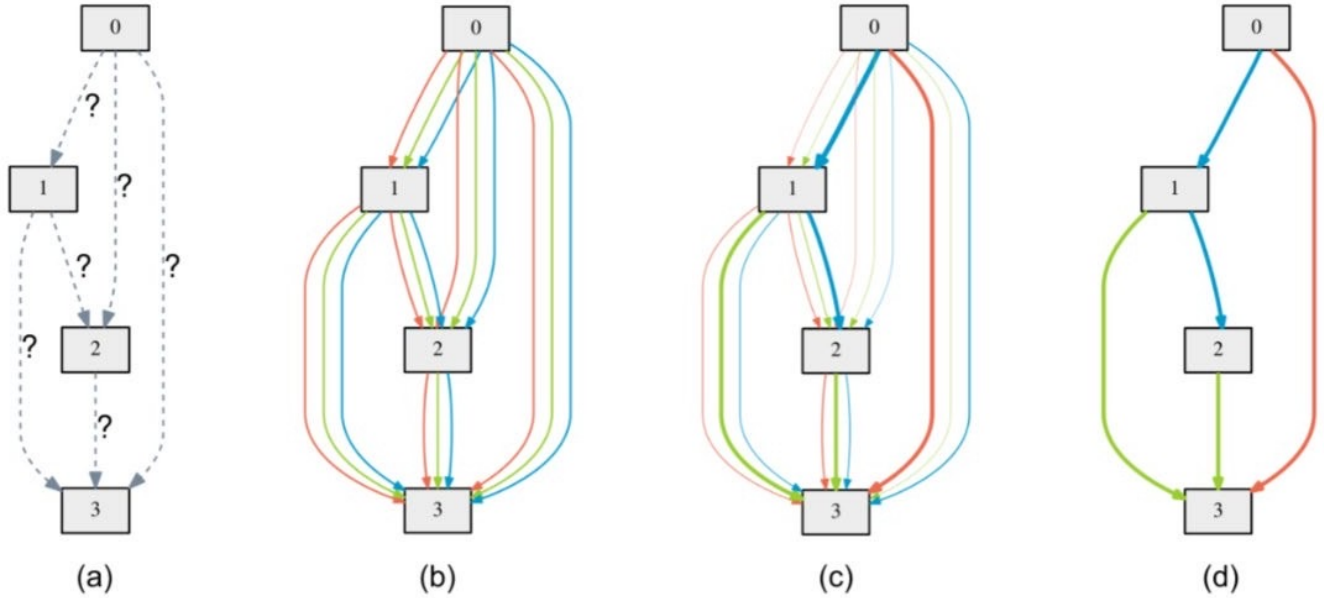
final evaluation



Overview of NAS Speedup Techniques

- **Weight Sharing & One-Shot Models**
  - ENAS [Pham et al, 2018], DARTS [Liu et al, 2019] and many follow-ups
- **Weight Inheritance & Network Morphisms**
  - Local changes in architecture, followed by fine-tuning steps
  - [Cai et al, 2018; Elsken et al, 2017; Cortes et al, 2017; Cai et al, 2018, Elsken et al, 2019]
- **Meta-Learning**
  - Learning across datasets
  - To initialize architectural weights of DARTS [Lian et al, 2020; Elsken et al, 2020]
  - Prior for blackbox optimization methods [Wong et al, 2018; Runge et al, 2019; Zimmer et al, 2020]
- **Multi-Fidelity Optimization**
  - Exploit cheaper proxy models for blackbox optimizers, in particular Bayesian optimization
  - [Jamieson & Talwalkar, 2016; Li et al, 2017; Falkner et al, 2018; Zela et al, 2018; White et al, 2021]

DARTS: Differentiable Architecture Search



- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - Mixed operator: $\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$
- Solve a bi-level optimization problem (c)

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

- In the end, discretize to obtain a single architecture (d)

Network Morphisms

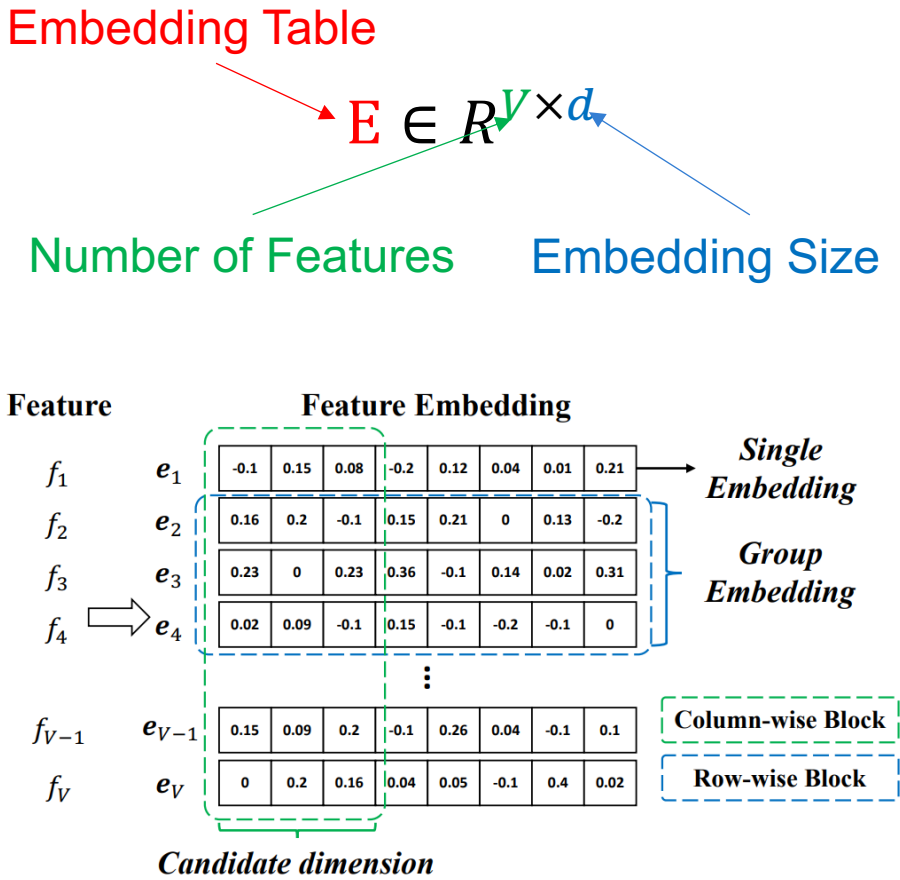
- Network morphisms [Chen et al., 2016; Wei et al., 2016]
  - Change the network structure, but not the modelled function (i.e., for every input, the network yields the same output
  - As before applying the network morphism)

  - Can use this in NAS algorithms as operations to generate new networks
  - Avoids costly training from scratch

Table of Contents

DRS Embedding Components

- The Embedding Table is used to map **the high-dimensional features** into a **low-dimensional latent space**.
- Feature embedding is **the cornerstone of the DRS** as the number of parameters in DRS is concentrated in the embedding table.
- To improve the **prediction accuracy, save storage space and reduce model capacity**, AutoML-based solutions are proposed for the learning of embedding table.
  - Single Embedding Search
  - Group Embedding Search



Single Embedding Search-AMTL

- AMTL leverages a twins-based architecture to avoid the unbalanced parameters update problem due to the different frequencies.
- AMTL designs an embedding search space with d^V size, where d is the embedding size.
- The twins-based architecture acts as a frequency-aware policy network to search the optimum dimension for each feature value→relaxed to a continuous space by temperated softmax.

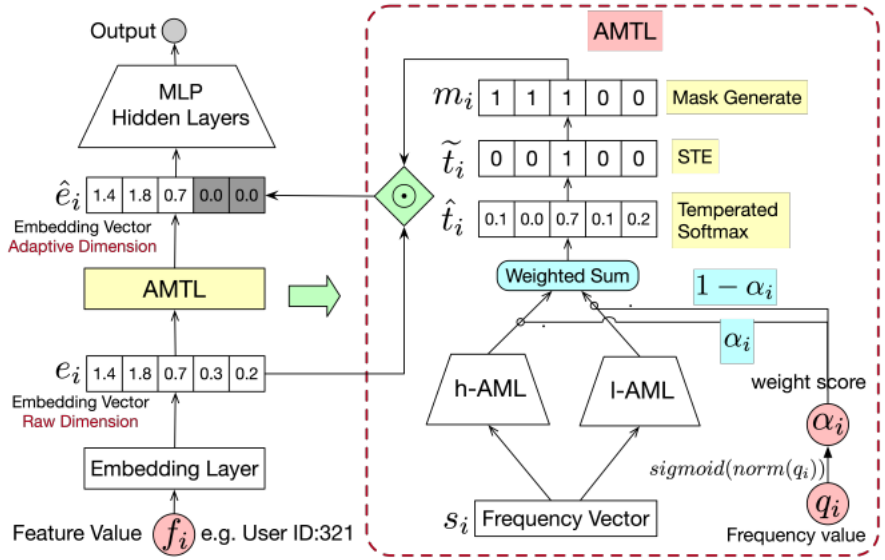


Figure 2: The framework of AMTL.

Learning effective and efficient embedding via an adaptively-masked twins-based layer. CIKM 2021.

Single Embedding Search-PEP

- Pruning-based Solution by enforcing column-wise sparsity on the Embedding Table with L_0 normalization.
- Search Space: 2^{Vd}, where V is numer of features and d is the embedding size for each feature.

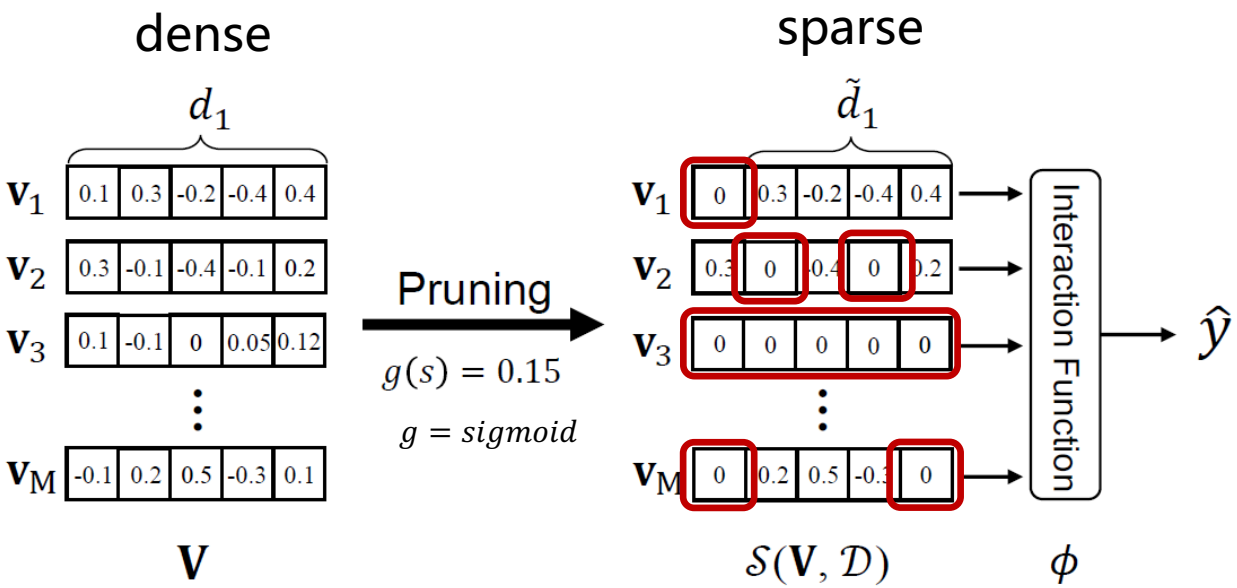


Figure 1: The basic idea of PEP.

$$\min \mathcal{L}, s.t. \ ||\mathbf{V}||_0 \leq k,$$ NP-hard

↓

Soft threshold re-parameterization :

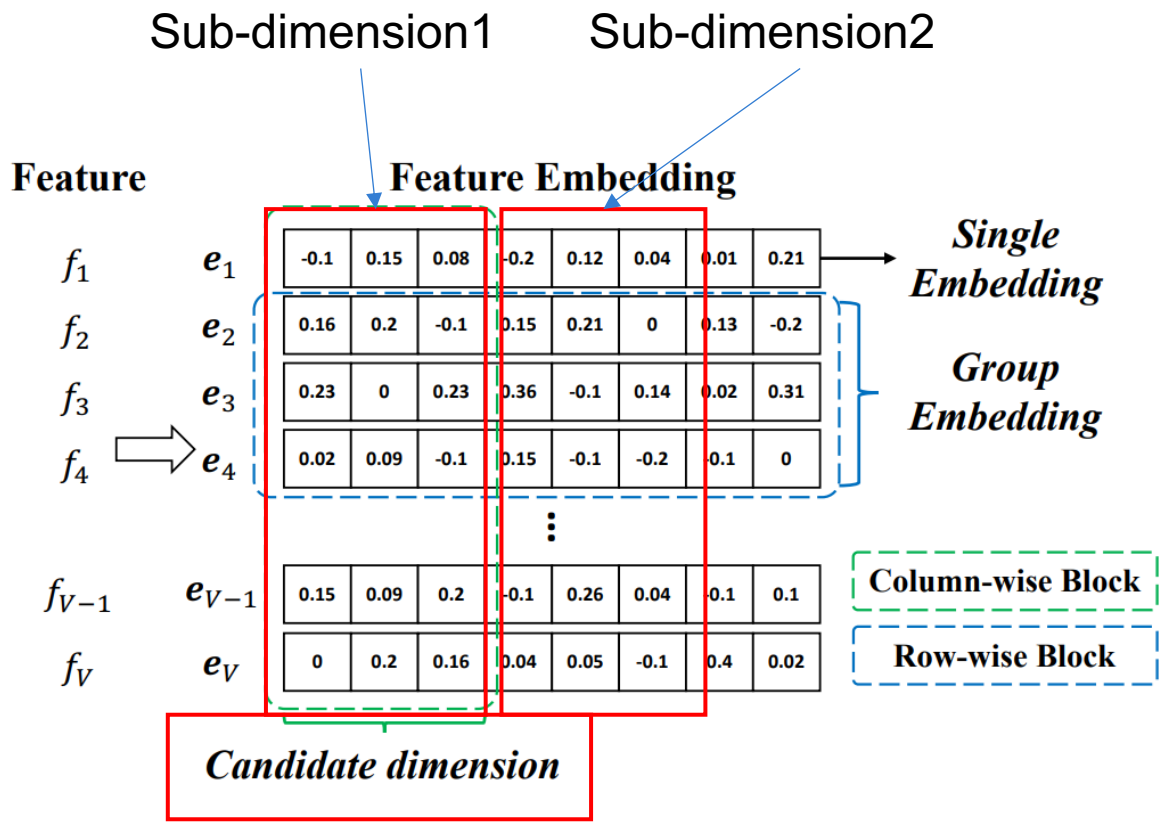
$$\hat{\mathbf{V}} = \mathcal{S}(\mathbf{V}, s) = sign(\mathbf{V})\text{ReLU}(|\mathbf{V}| - g(s)),$$

$$\min \mathcal{L}(\mathcal{S}(\mathbf{V}, s), \Theta, \mathcal{D}).$$

PEP: Learnable Embedding Sizes for Recommender Systems. ICLR 2021.

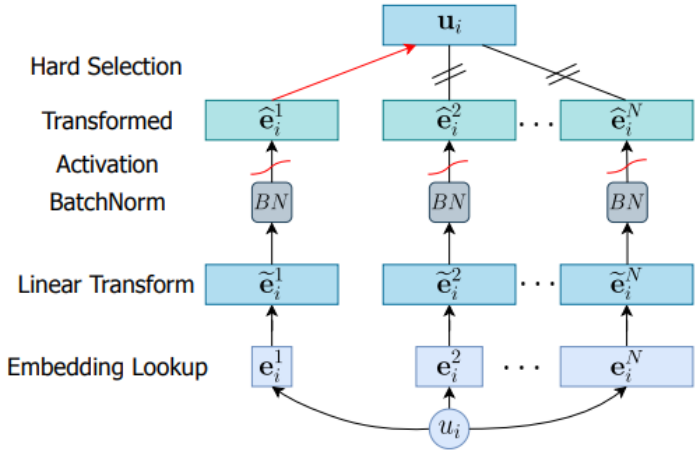
Single Embedding Search

- The search space of PEP and AMTL is highly related with the embedding size d.
- To reduce the search space, AutoEmb and ESPAN divide the embedding dimension into several columnwise sub-dimensions.

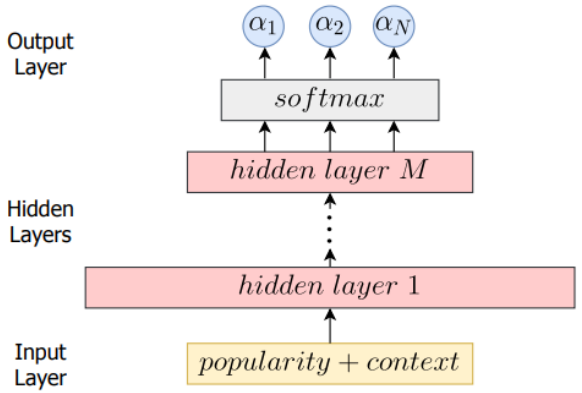


Single Embedding Search-AutoEmb

- AutoEmb leverages two controller networks to decide the embedding sizes for users and items via end-to-end differentiable soft selection.
- Summing over the candidate subdimensions with learnable weights.
- Search Space: from d^V to a^V, where V is nubmer of features and, d is the embedding size, and a is the numer of sub-dimensions for each feature.



AutoEmb

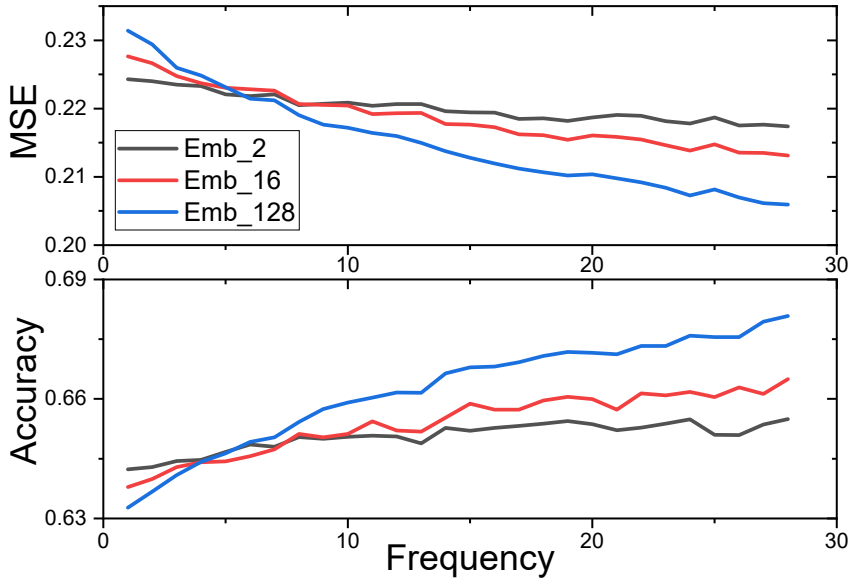


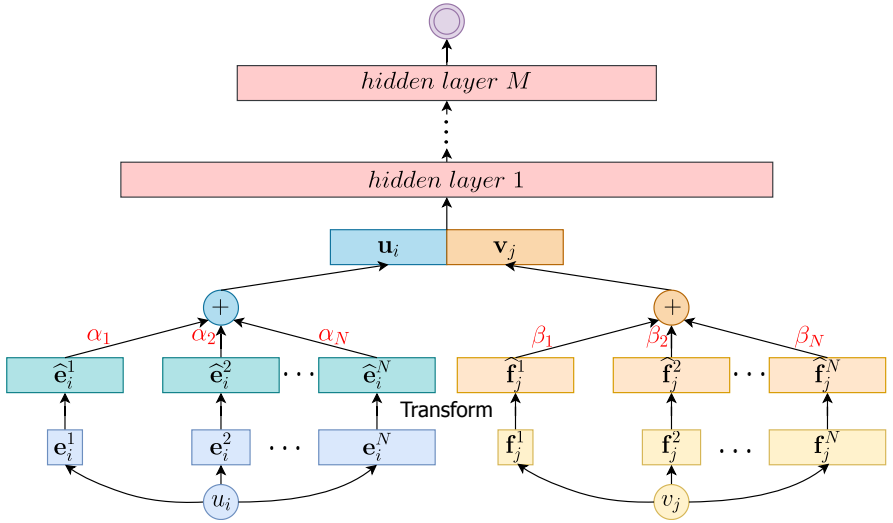
Controller network

AutoEmb: Automated Embedding Dimensionality Search in Streaming Recommendations. ICDM 2021.

Single Embedding Search-ESPAN

- Embedding dimension often determines the capacity to encode information.
- Dynamically search the embedding sizes for different users and items
  - Optimal recommendation quality all the time
  - More efficient in memory

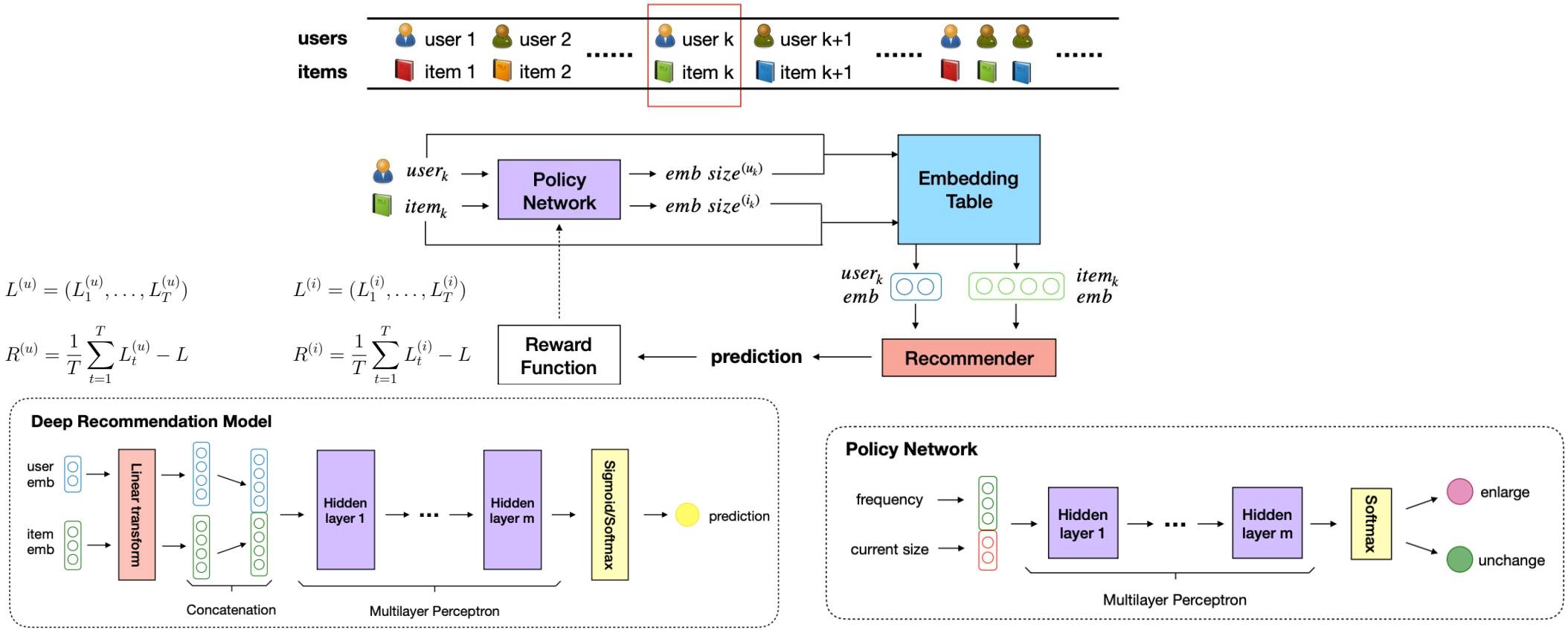




ESAPN: Automated Embedding Size Search in Deep Recommender Systems. SIGIR 2020.

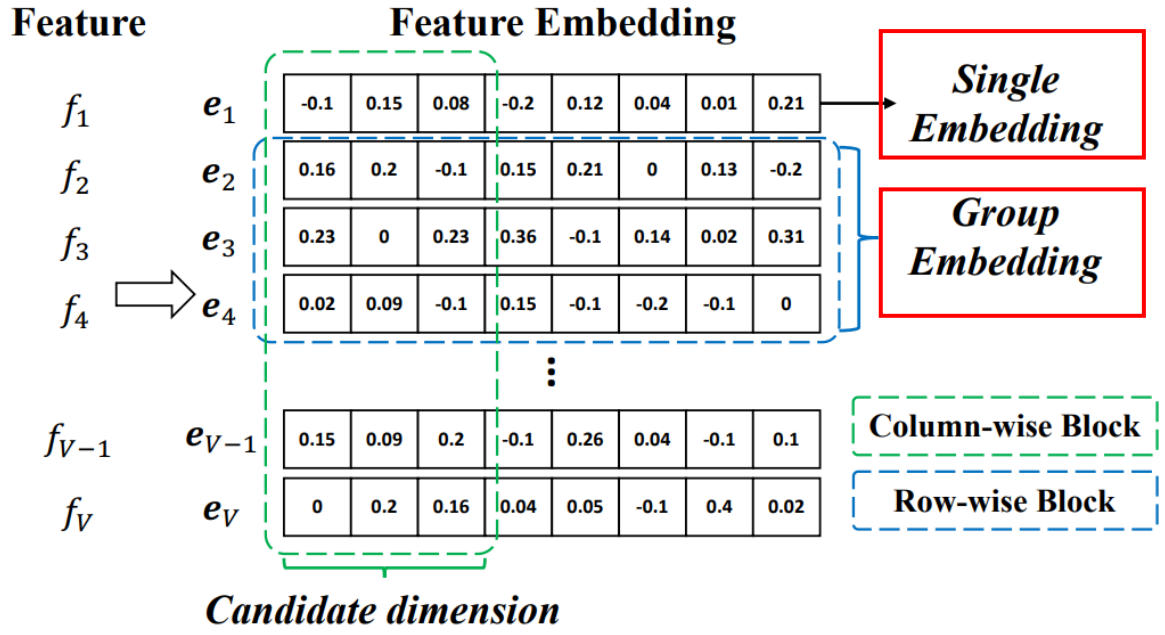
Single Embedding Search-ESPAN

- Two Components
  - Deep recommendation model
  - Embedding Size Adjustment Policy Network (ESAPN) - hard selection via RL



Group Embedding Search

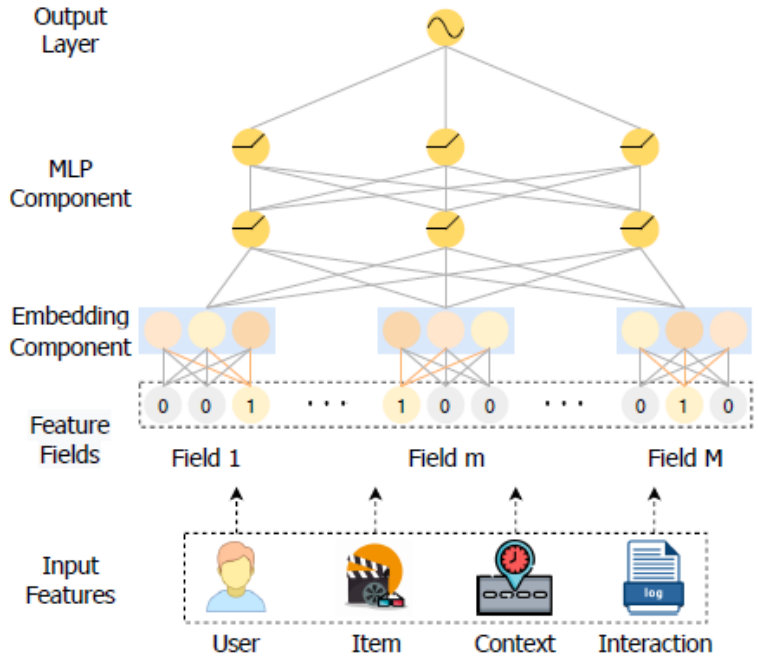
- AutoEmb and ESAPN shrink the search space by dividing the embedding dimension into candidate column-wise sub-dimensions.
- Another solution is to **group** the feature values of a field based on some indicators (e.g., frequencies) and assign a **row-wise group embedding dimension** for all the values within the group.



Group Embedding Search-AutoDim

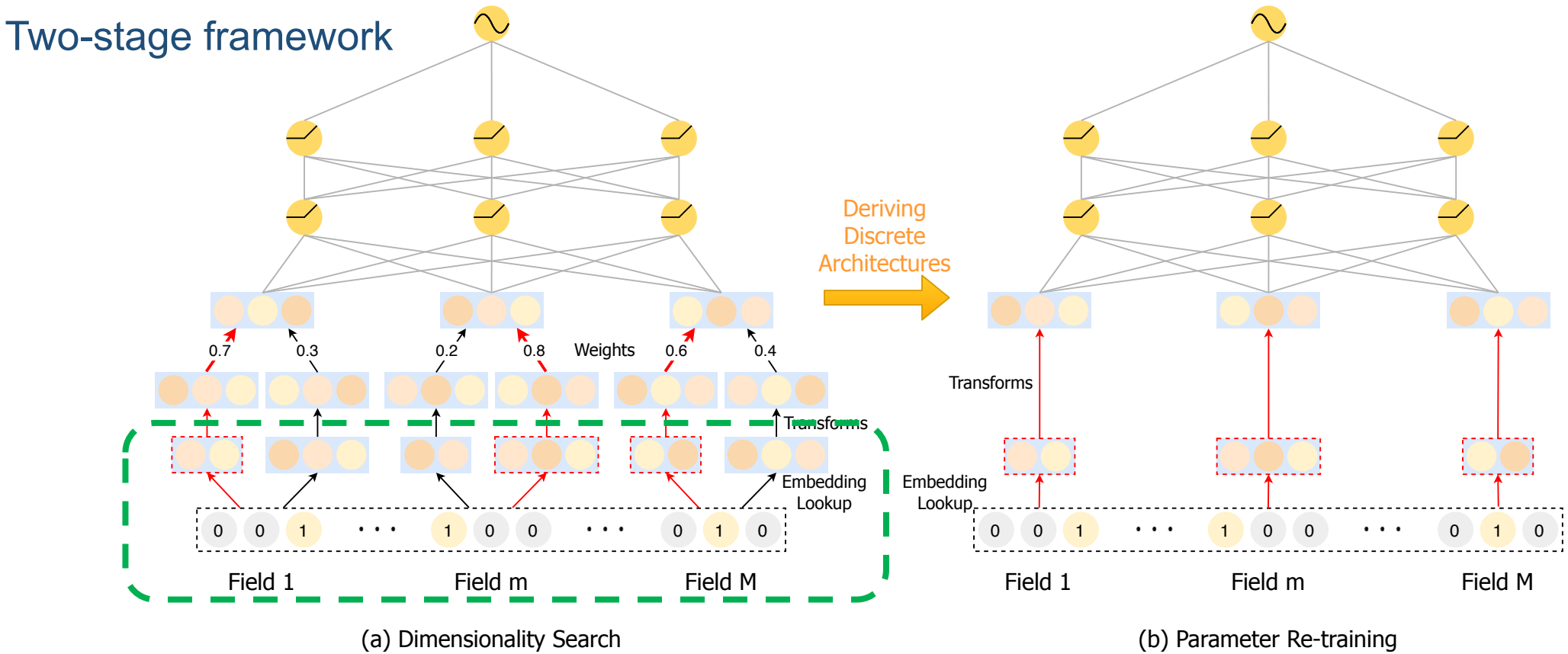
- Pre-defines several candidate sub-dimensions like AutoEmb.
- Setting the number of groups b = 1 and searching a global embedding dimension for all the feature values of a **field**.
- Search Space: from a^V to a^m, where m is the number of groups.

Goal:
Selecting embedding dimensions to different feature fields automatically in a data-driven manner.



AutoDim: Field-aware Embedding Dimension Search in Recommender Systems. WWW 2021.

Group Embedding Search-AutoDim

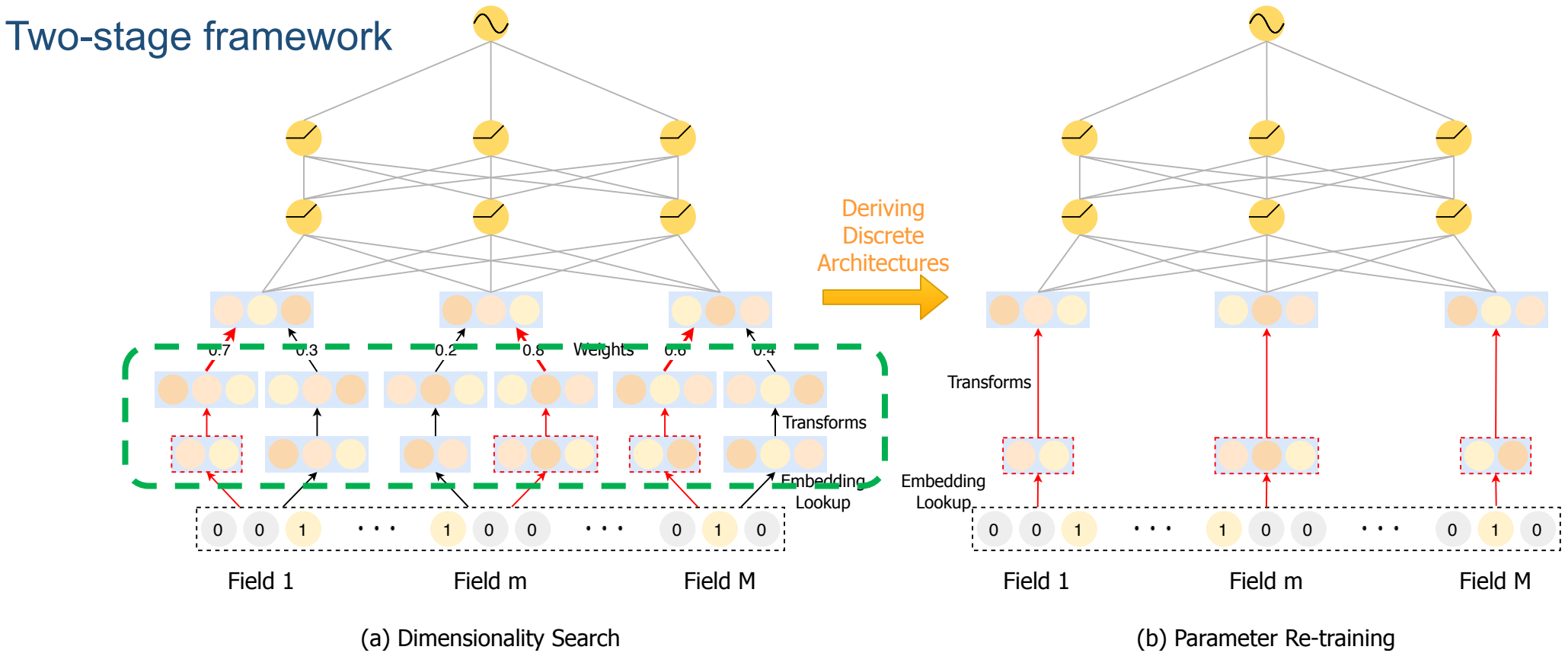


(a) Dimensionality Search

(b) Parameter Re-training

AutoDim searches dimensions for feature fields in a soft and continuous fashion via **Gumbel Softmax**, reducing to a smaller serach space: 5 candiate for each feature field.

Group Embedding Search-AutoDim

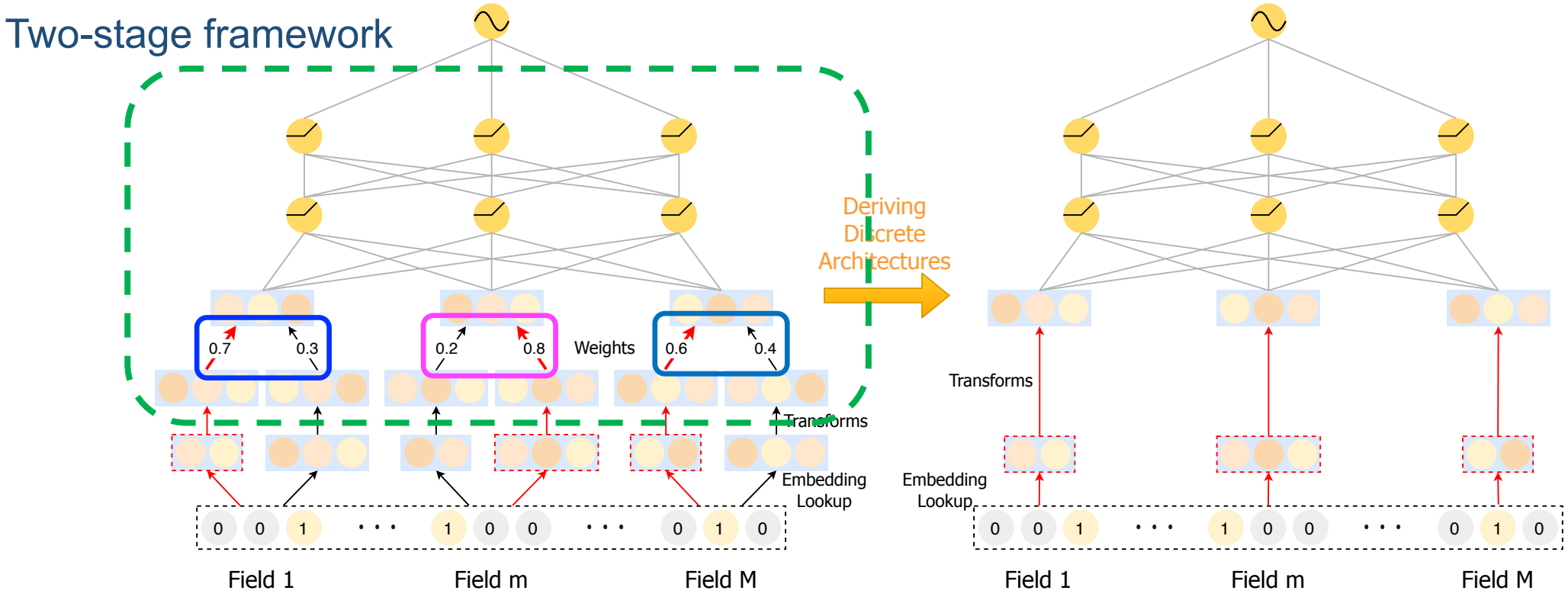


(a) Dimensionality Search

(b) Parameter Re-training

AutoDim searches dimensions for feature fields in a soft and continuous fashion via **Gumbel Softmax**, reducing to a smaller serach space: 5 candiate for each feature field.

Group Embedding Search-AutoDim

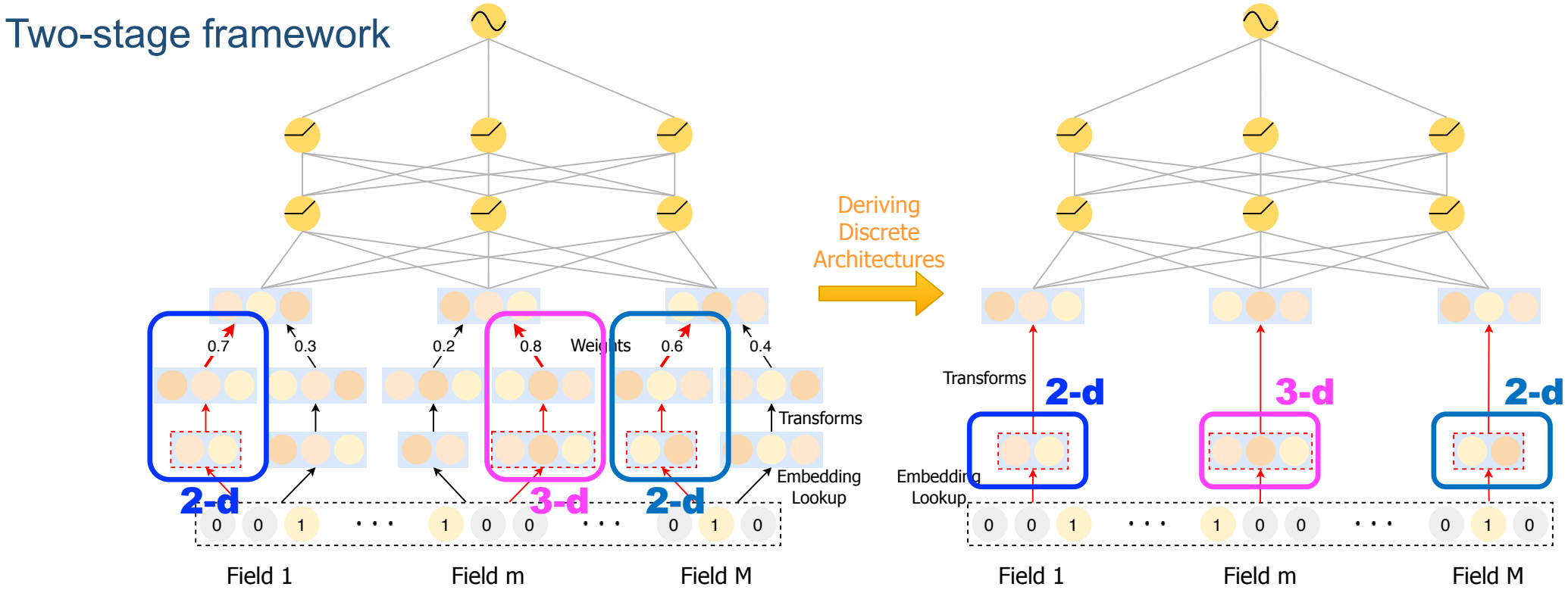


(a) Dimensionality Search

(b) Parameter Re-training

AutoDim searches dimensions for feature fields in a soft and continuous fashion via **Gumbel Softmax**, reducing to a smaller serach space: 5 candiate for each feature field.

Group Embedding Search-AutoDim



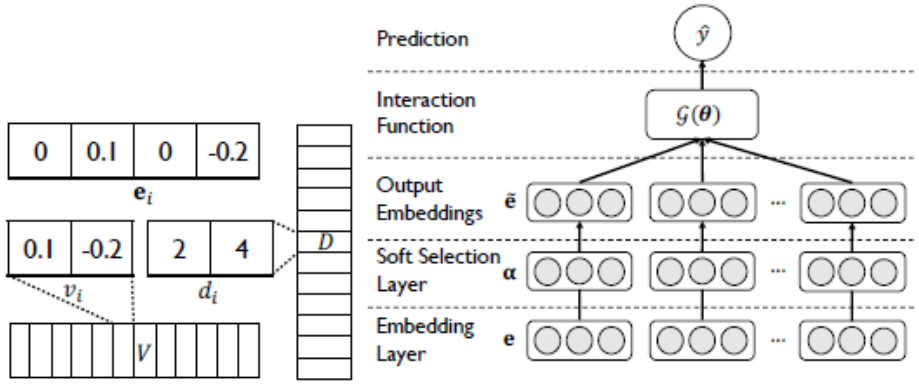
(a) Dimensionality Search

(b) Parameter Re-training

AutoDim searches dimensions for feature fields in a soft and continuous fashion via **Gumbel Softmax**, reducing to a smaller serach space: 5 candiate for each feature field.

Group Embedding Search-DNIS

- DNIS splits the feature into multi-groups based on the feature frequencies or clustering.
- Search space: from 2^{Vd} into 2^{bd}, where b is the number of groups.
- DNIS searches for mixed feature embedding dimensions in a more flexible space through continuous relaxation and differentiable optimization.



(a) Example of notations. (b) Model structure.

$$\tilde{\mathbf{e}}_i = \mathbf{e}_i \odot \boldsymbol{\alpha}_{l*}$$

$$\tilde{\mathbf{E}}_{i,j} = \begin{cases} 0, & \text{if } |\tilde{\mathbf{E}}_{i,j}| < \epsilon \\ \tilde{\mathbf{E}}_{i,j}, & \text{otherwise} \end{cases}$$

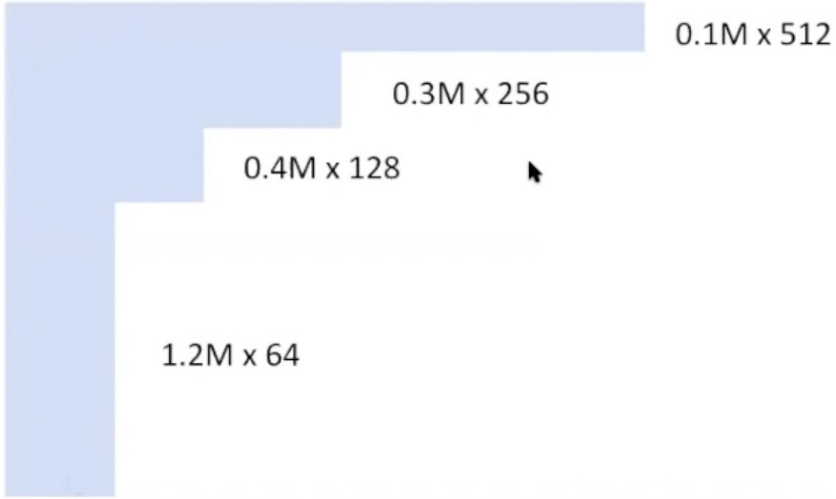
DNIS: Differentiable Neural Input Search for Recommender Systems. In Arxiv.

Group Embedding Search-NIS

- Input component assigns embedding vectors to each item of these discrete features, which dominate both the size and the inductive bias of the model.
- The vocabulary and embedding sizes for discrete features are often selected heuristically.

Head Feature
- More data, more information
- Needing larger embedding size

Tail Feature
- Less data, less information
- Small embedding size is enough



Single-size Embedding (SE) $\sum_{F \in \mathcal{F}} v_F \times d_F \leq \mathcal{C}$

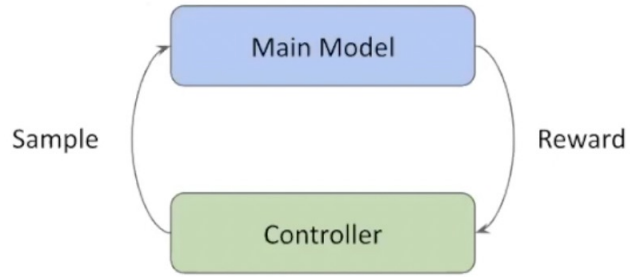
Multi-size Embedding (ME) $\sum_{F \in \mathcal{F}} \sum_{i=1}^{M_F} v_{F_i} \times d_{F_i} \leq \mathcal{C}$

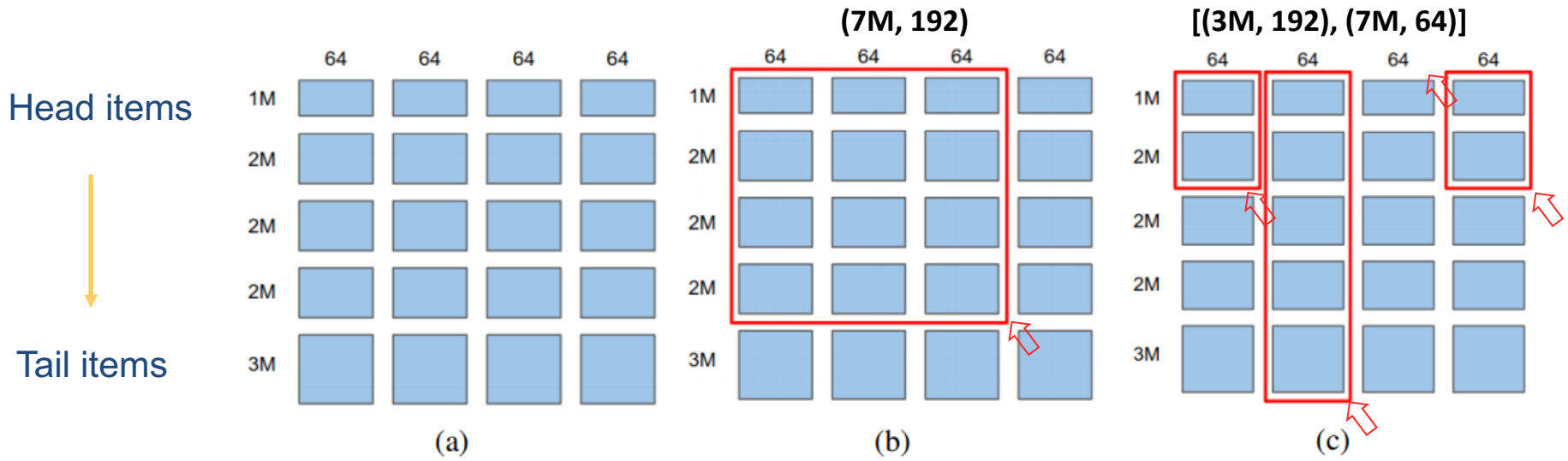
NIS: Neural Input Search for Large Scale Recommendation Models. KDD 2020.

Group Embedding Search-NIS

RL-based AutoML approach (ENAS)

- Main model is the deep recommendation model
- Controller learns to sample embedding dimensions that generate higher reward.
- Reward: $R = R_Q - \lambda * C_M$





Single-size Embedding (SE) **Multi-size Embedding (ME)**

Embedding Blocks: discretizing an embedding matrix of size v × d into S × T sub-matrices

Group Embedding Search-AutoDis

Existing methods for numerical feature representation have some limitations:

1. Category 1: No Embedding

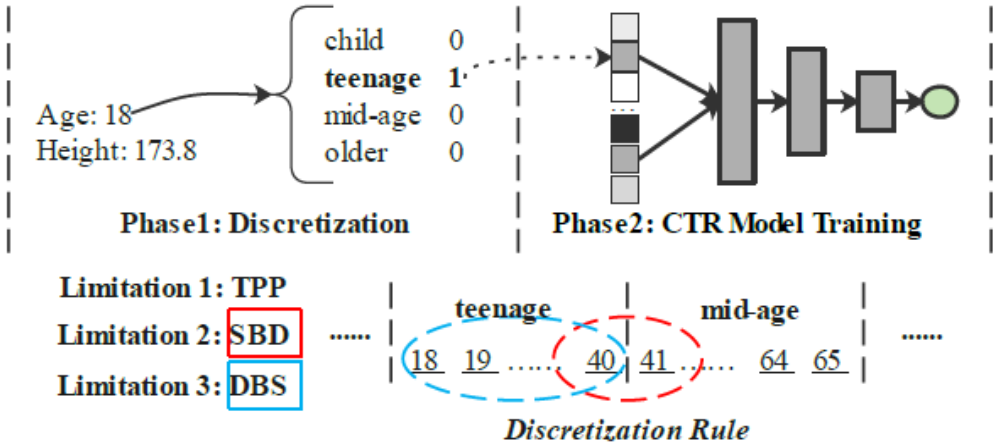
- **Low capacity**: difficult to capture informative knowledge of numerical fields.
- **Poor compatibility**: difficult to adapt to some models (e.g., FM).

2. Category 2: Field Embedding

- **Low capacity**: single shared field-specific embedding.

3. Category 3: Discretization

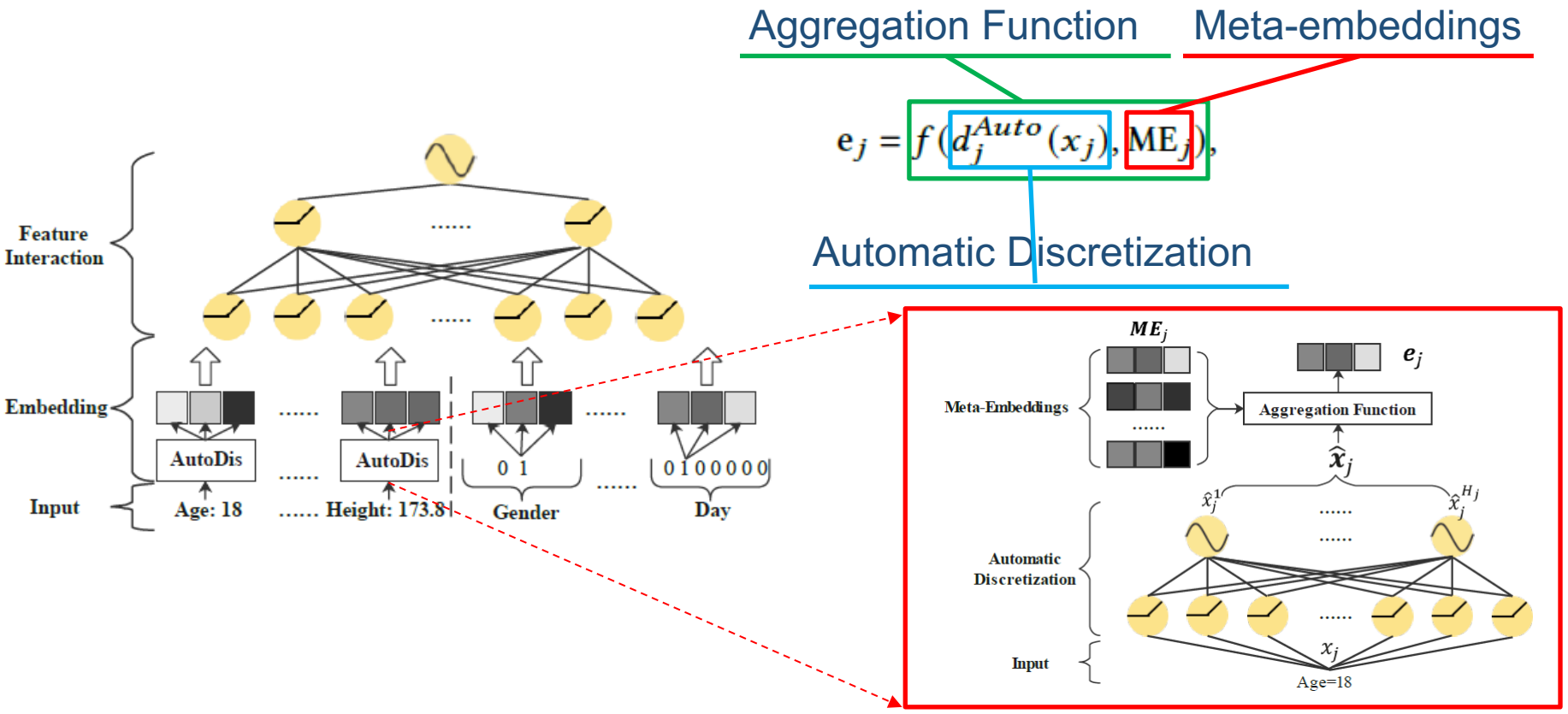
- **TPP** (Two-Phase Problem)
- **SBD** (Similar value But Dis-similar embedding)
- **DBS** (Dis-similar value But Same embedding)



AutoDis: An Embedding Learning Framework for Numerical Features in CTR Prediction. KDD 2021.

Group Embedding Search-AutoDis

AutoDis is a numerical features embedding learning framework with **high model capacity, end-to-end training and unique representation** properties preserved.



Summarize DRS Embedding

	Approach	Search Space	Search Type
AMTL	DARTS	d^V	Single Embedding Search
PEP	L0	2^{Vd}	Single Embedding Search
AutoEmb	DARTS	a^V	Single Embedding Search
ESPAN	RL	a^V	Single Embedding Search
AutoDim	DARTS	a^b	Group Embedding Search
NIS	DARTS	b^a or ab	Group Embedding Search
DNIS	darts	2^{bd}	Group Embedding Search
AutoDis	Softmax+MLP	--	Group Embedding Search

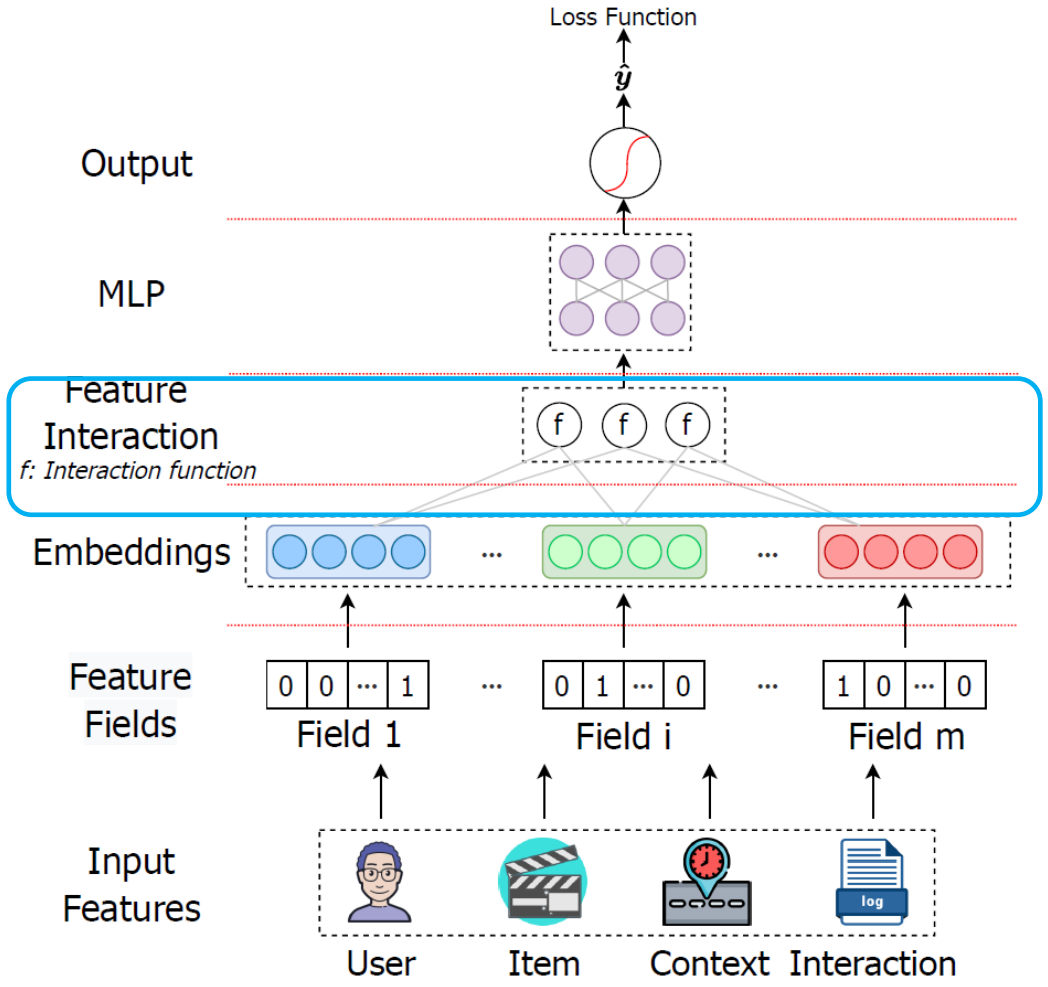
*We do not show the search space as it is the embedding approach for numerical feature.
** d is the embedding size, b is the number of feature groups, V is the number of features, a is the candidate embedding sizes, m is ? a < d, b<<V.

- DARTS is more popular as it has higher efficiency.
- The Search Space of Group Embedding Search is less than Single Embedding Search.
- Limited approaches for embedding learning of numerical features,

Table of Contents

Background

Effectively modelling **feature interactions** is important.



- Both low-order and high-order feature interactions play important roles to model user preference.
  - People like to download popular apps → id of an app may be a signal
  - People often download apps for food delivery at meal time → interaction between app category and time-stamp may be a signal
  - Male teenagers like shooting game → interaction of app category, user gender and age may be a signal
- Most feature interactions are hidden in data and difficult to identify (e.g., “diaper and beer" rule)

Background

The challenges of modelling **feature interactions**:

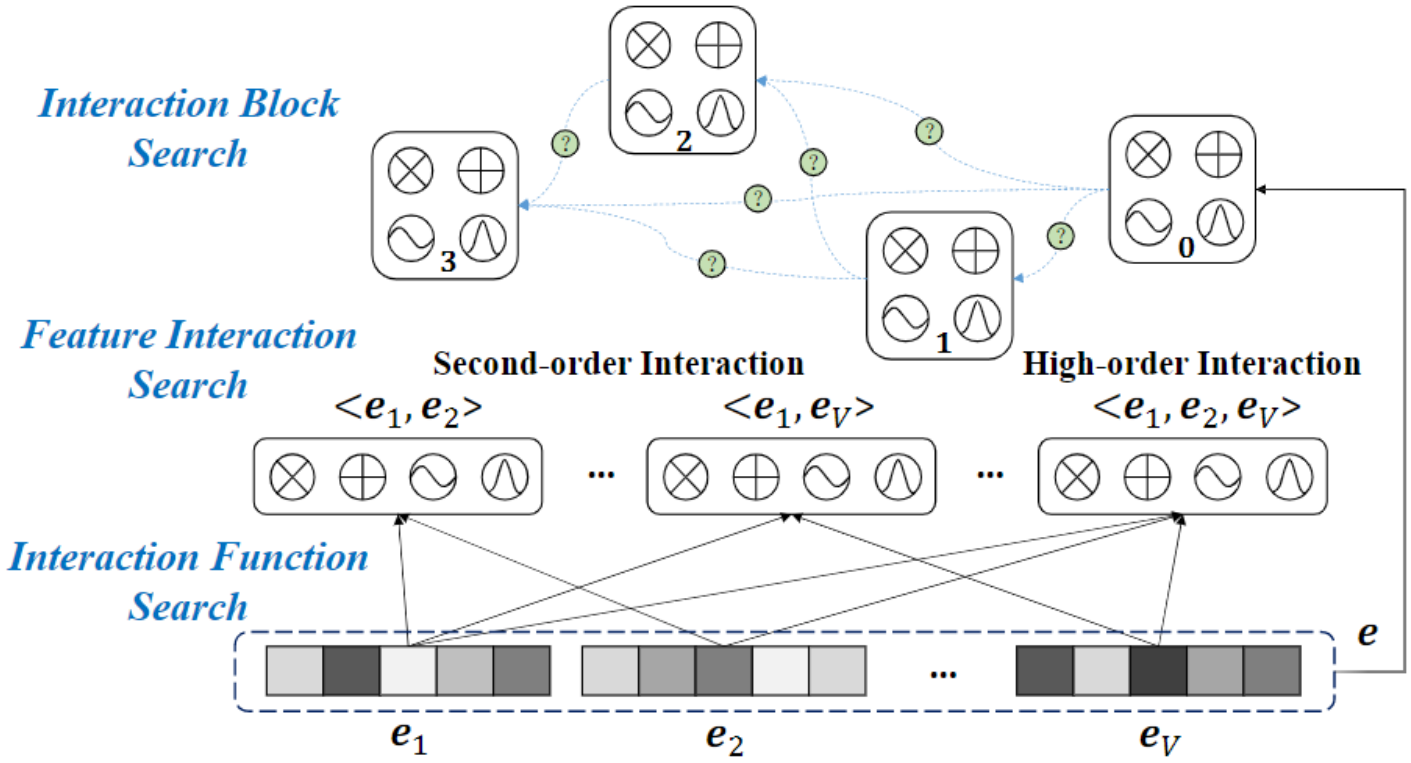
1) Enumerate all feature interactions
   - Large memory and computation cost
   - Difficult to be extended into high-order interactions
   - Useless interaction

2) Require human efforts to identify important feature interactions
   - High labor cost
   - Risks missing some counterintuitive (but important) interactions

3) Require human efforts to select appropriate interaction functions
   - Human expert knowledge
   - Global interaction function for all the feature interactions

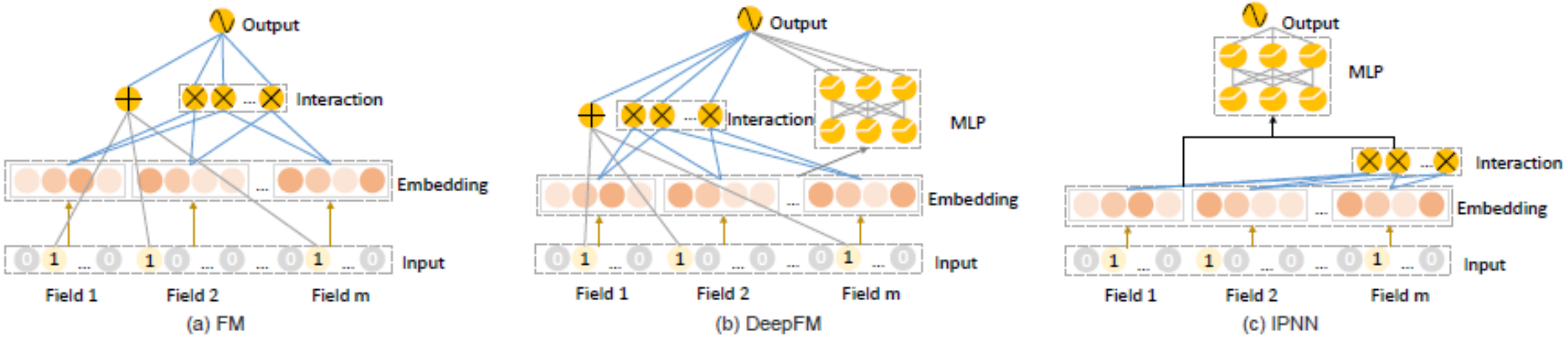
Background

Automatically **select important feature interactions with appropriate interaction functions**



Feature Interaction Search-AutoFIS

- Not all the feature interactions are useful.
- We apply AutoML techniques to identify such noisy feature interactions and filter them.
- Attention is not enough, as the learning of useful feature interactions is still diluted by many useless ones.



AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. KDD 2020

Feature Interaction Search-AutoFIS

- Search Stage
  - Detect useful feature interactions
- Retrain Stage
  - Retrain model with selected feature interactions

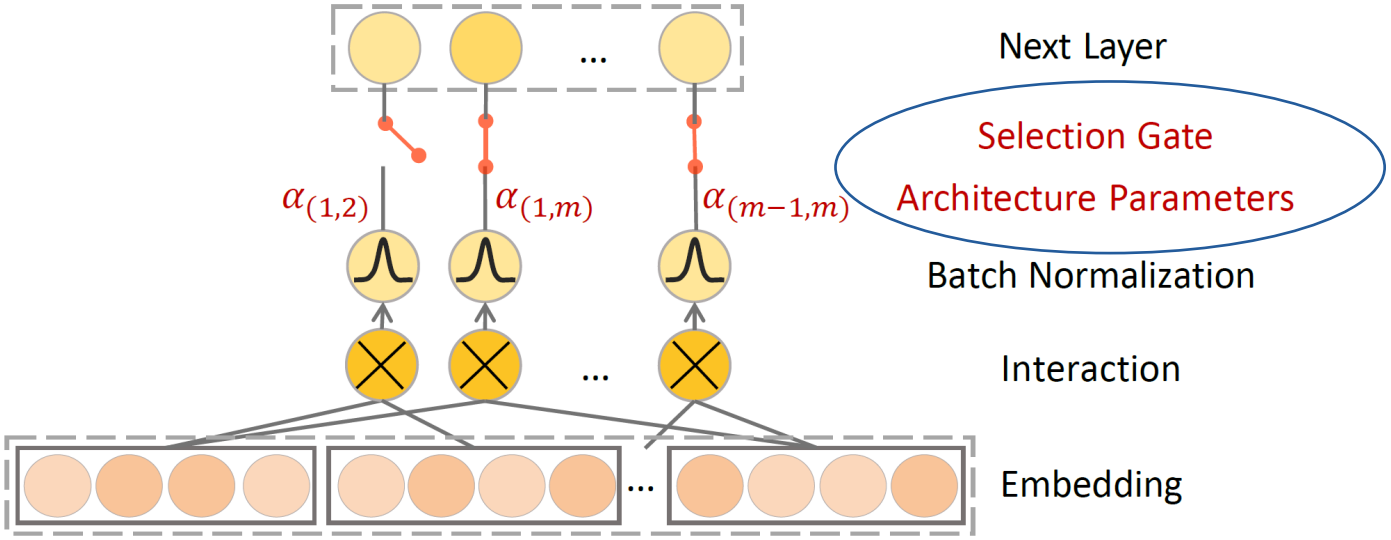
Feature Interaction Search-AutoFIS

Search Stage:

- Gate for each feature interaction
  - Huge search space $2^{C_m^2}$
- To make such identification differentiable, we relax the discrete search space to be continuous, by defining architecture parameters α.
- We use Batch Normalization and GRDA Optimizer to train stable and sparse architecture parameters.

$$l_{\text{AutoFIS}} = \langle w, x \rangle + \sum_{i=1}^{m} \sum_{j>i}^{m} \alpha_{(i,j)} \langle \boldsymbol{e}_i, \boldsymbol{e}_j \rangle$$

Indicator α = 0 or 1



Feature Interaction Search-AutoFIS

Retrain Stage:

- Abandon unimportant feature interactions
- Retrain model

Feature Interaction Search-AutoGroup

The limitation of AutoFIS:

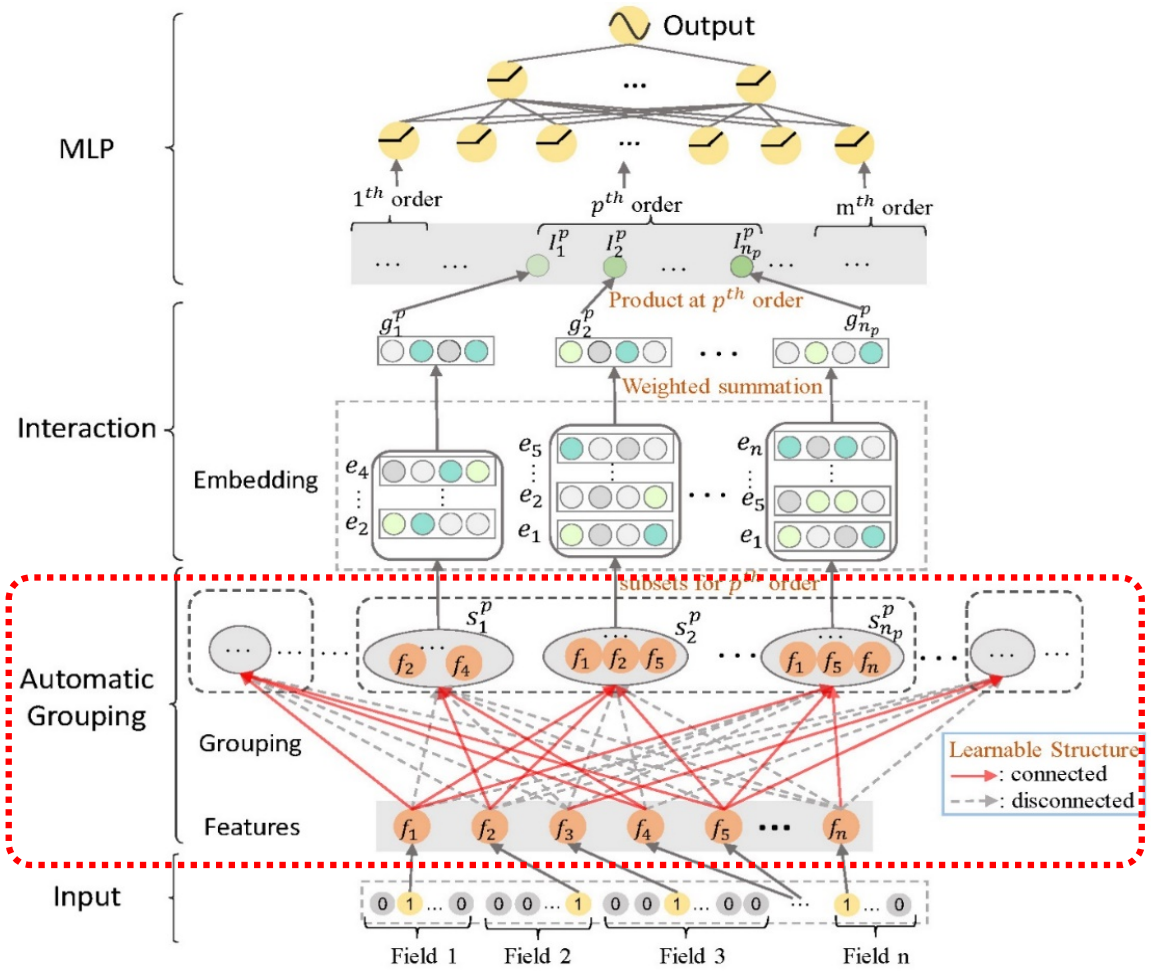
- When searching **high-order feature interactions**, the search space of AutoFIS is huge, resulting in low search efficiency.

Solution of AutoGroup:

- To solve the ***efficiency-accuracy dilemma***, AutoGroup proposes automatic feature grouping, reducing the pth-order search space from $2^{\mathrm{C}_{\mathrm{m}}^{\mathrm{p}}}$ to 2^{gm}, where g is the number of pre-defined groups.

Feature Interaction Search-AutoGroup

Feature Grouping Stage:



Each feature is possible to be selected into the feature sets of each order.

- $\Pi_{i,j}^{p} \in \{0,1\}$: whether select feature f_i into the j^{th} set of order-p.

To make the selection differentiable, we relax the binary discrete value to a softmax over the two possibilities:

$$\overline{\Pi}_{i,j}^{p} = \frac{1}{1+\exp(-\alpha_{i,j}^{p})}\Pi_{i,j}^{p} + \frac{\exp\left(-\alpha_{i,j}^{p}\right)}{1+\exp\left(-\alpha_{i,j}^{p}\right)}(1-\Pi_{i,j}^{p}).$$

To learn a less-biased selection probability, we use Gumbel-Softmax:

$$\left(\overline{\Pi}_{i,j}^{p}\right)_{o} = \frac{\exp(\frac{\log \alpha_o + G_o}{\tau})}{\sum_{o' \in \{0,1\}} \exp(\frac{\log \alpha_{o'} + G_{o'}}{\tau})} \text{ where } o \in \{0,1\}.$$

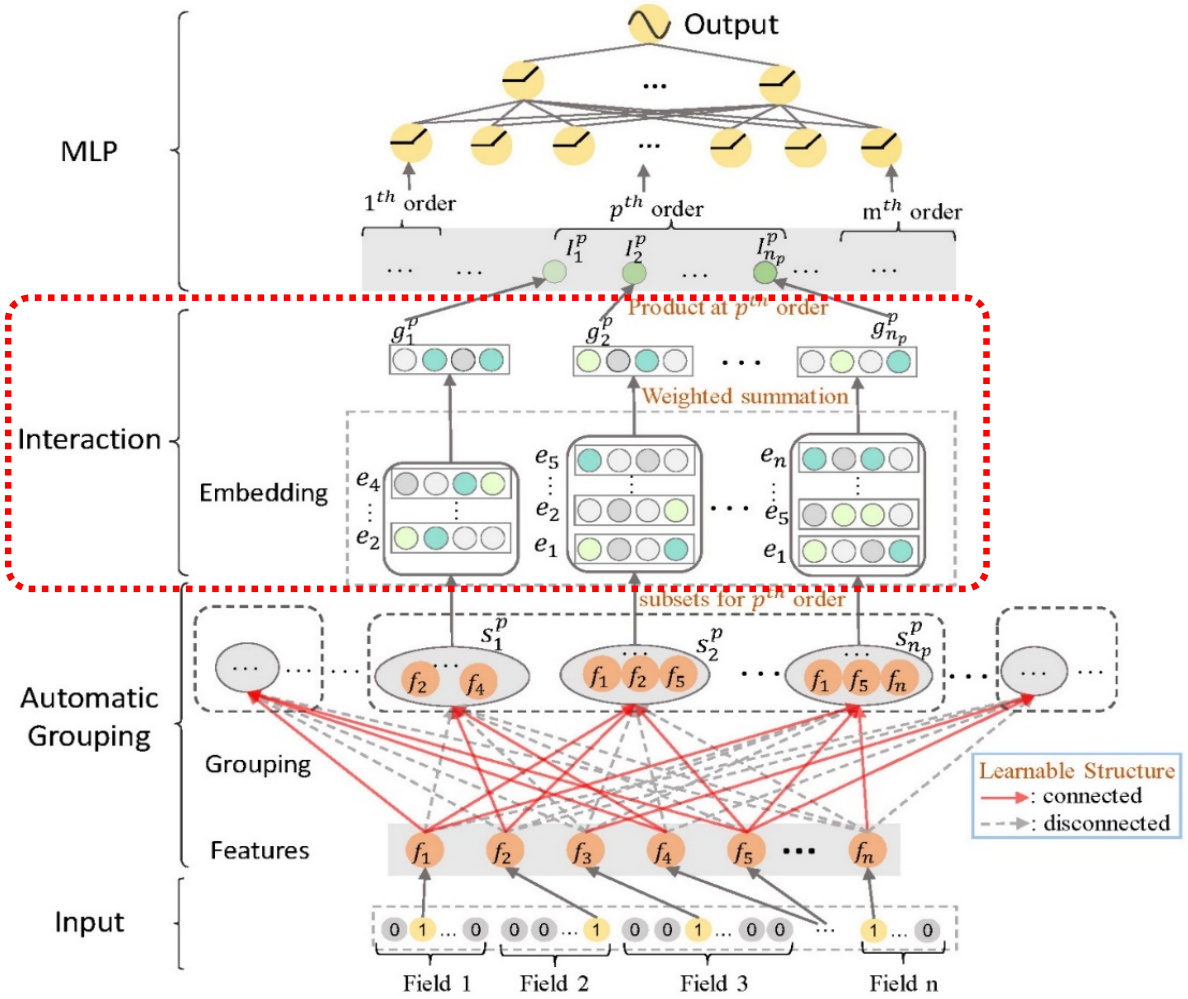
$$\alpha_0 = \frac{1}{1+\exp(-\alpha_{i,j}^{p})} \qquad \alpha_1 = \frac{\exp\left(-\alpha_{i,j}^{p}\right)}{1+\exp\left(-\alpha_{i,j}^{p}\right)}$$

$$G_o = -\log(-\log u) \text{ where } u \sim Uniform(0,1)$$

Trainable Parameters: $\{\alpha_{i,j}^{p}\}$

Feature Interaction Search-AutoGroup

Interaction Stage:



Feature set representation:

$$g_j^p = \sum_{f_i \in s_j^p} w_i^p \, e_i$$

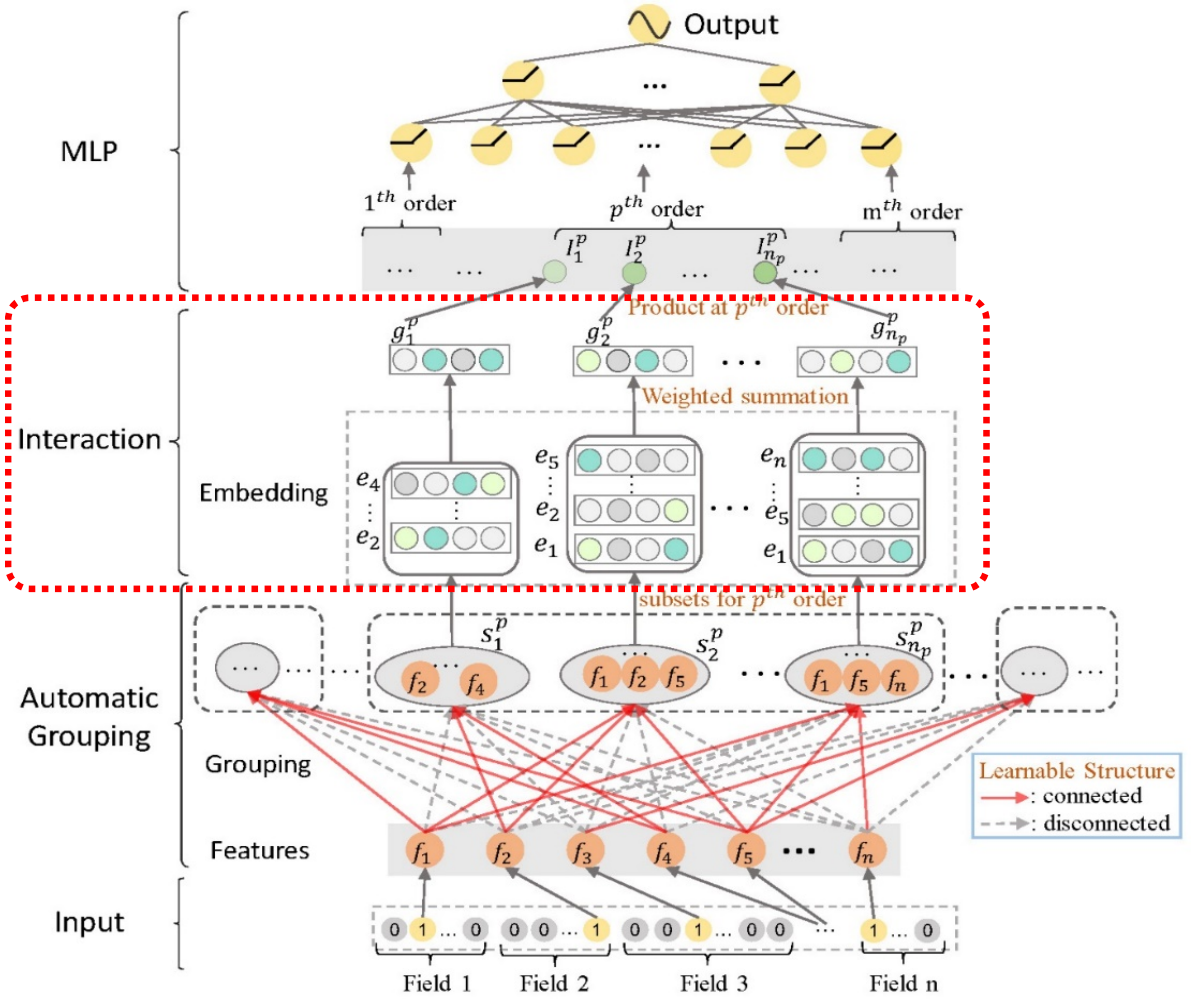
s_j^p: the j^{th} feature set for order-p feature interactions.

e_i: embedding for feature f_i

w_i^p: weights of embeddings in feature set s_j^p.

Feature Interaction Search-AutoGroup

Interaction Stage:



Interaction at a given order:

- Inspired by the reformulation of FM:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < e_i, e_j > x_i x_j$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\left((\sum_{i=1}^{n} x_i e_i)^2 - \sum_{i=1}^{n} (x_i e_i)^2 \right)$$

- The order-p interaction in a given set s_j^p is defined as:

$$I_j^p = \begin{cases} \left(g_j^p\right)^p - \sum_{f_i \in s_j^k} \left(w_i^p e_i\right)^p \in R, p \geq 2 \\ g_j^p \in R^\kappa, \quad p = 1 \end{cases}$$

Feature Interaction Search-FIVES

The limitation of AutoGroup:

- Although AutoGroup solve the ***efficiency-accuracy dilemma*** via feature grouping, it ignores the ***order-priority property***.
- ***Order-priority property***: The higher-order feature interactions quality can be relevant to their de-generated low-order ones, and lower-order feature interactions are likely to be more vital compared with higher-order ones.

Solution of FIVES:

- To solve the *efficiency-accuracy dilemma*, FIVES regards the original features as a feature graph conceptually and models the high-order feature interactions by a GNN with layer-wise adjacency matrix, reducing the pth-order search space from $2^{\mathrm{C}_{\mathrm{m}}^{\mathrm{p}}}$ to 2^{m^2}.
- To keep the *order-priority property*, FIVES parameterizes the adjacency matrix and makes them depend on the previous layer.

FIVES: Feature Interaction via Edge Search for Large-Scale Tabular Data. KDD 2021

Feature Interaction Search-FIVES

- With an adjacency tensor A, the dedicated graph convolutional operator produces the node representations layer-by-layer.. For the step k:

$$n_i^{(k)} = p_i^{(k)} \odot n_i^{(k-1)}$$
$$\text{where } p_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1}\{W_j n_j^{(0)}\} \quad (1)$$

- The node representation at k-th layer corresponds to the generated (k + 1)-order interactive features:

$$n_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1}\left\{W_j n_j^{(0)}\right\} \odot n_i^{(0)} \quad (2)$$
$$\approx \text{MEAN}_{(c_1,\ldots,c_k)|A_{i,c_j}^{(j)}=1,j=1,\ldots,k}\{f_{c_1} \otimes \ldots \otimes f_{c_k} \otimes f_i\}$$

Feature Interaction Search-FIVES

- The task of generating useful interactive features is equivalent to learning an optimal adjacency tensor A, namely **edge search**.
- The edge search task could be formulated as a bi-level optimization problem:

$$\begin{aligned} &\min_{A} \mathcal{L}(\mathcal{D}_{\text{val}}|A, \Theta(A)) \\ \text{s.t.} \quad &\Theta(A) = \arg\min_{\Theta} \mathcal{L}\left(\mathcal{D}_{\text{train}}|A, \Theta\right) \end{aligned} \tag{3}$$

- To make the optimization more efficient, we allow to use a soft $A^{(k)}$ for propagation at the k-th layer, while the calculation of $A^{(k)}$still depends on a binarized $A^{(k-1)}$:

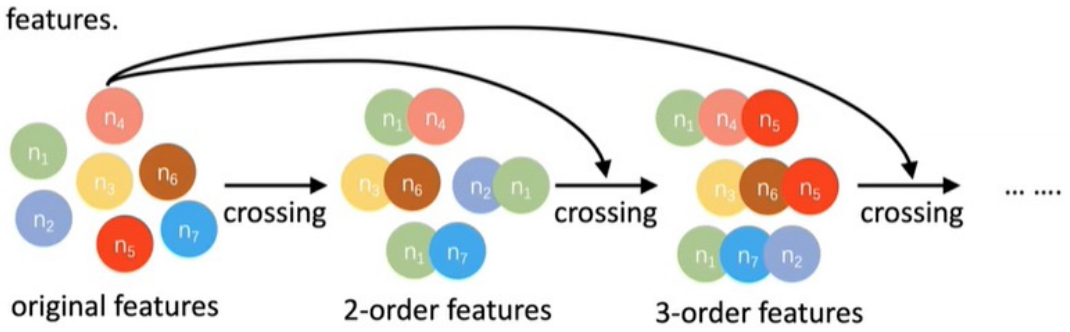
$$A^{(k)} \triangleq \left(D^{(k-1)}\right)^{-1} \varphi\left(A^{(k-1)}\right) \sigma\left(H^{(k)}\right) \tag{4}$$

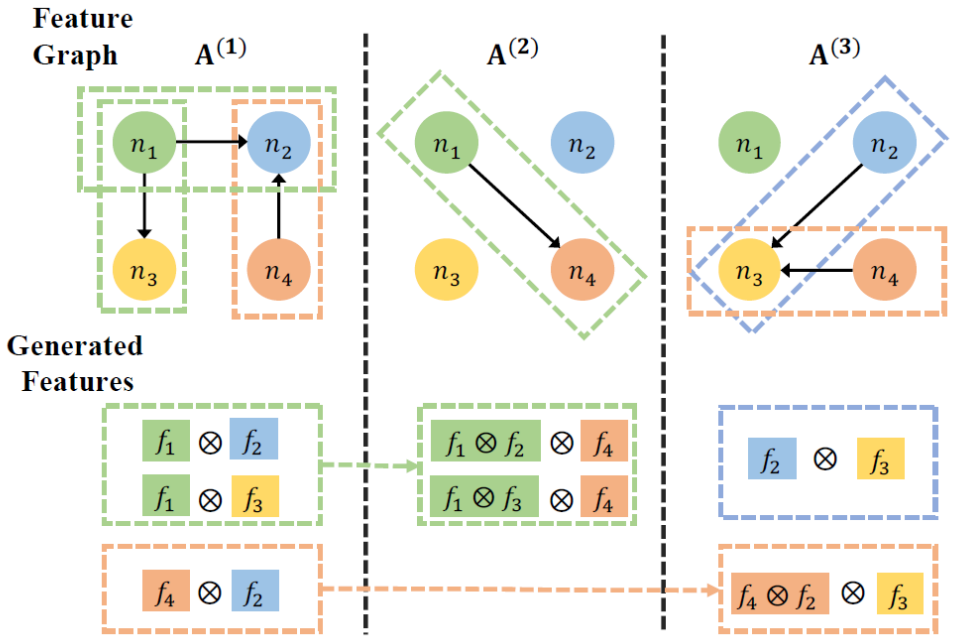
Degree matrix of $A^{(k-1)}$

Binarize the soft $A^{(k-1)}$

Interactions at k-th layer

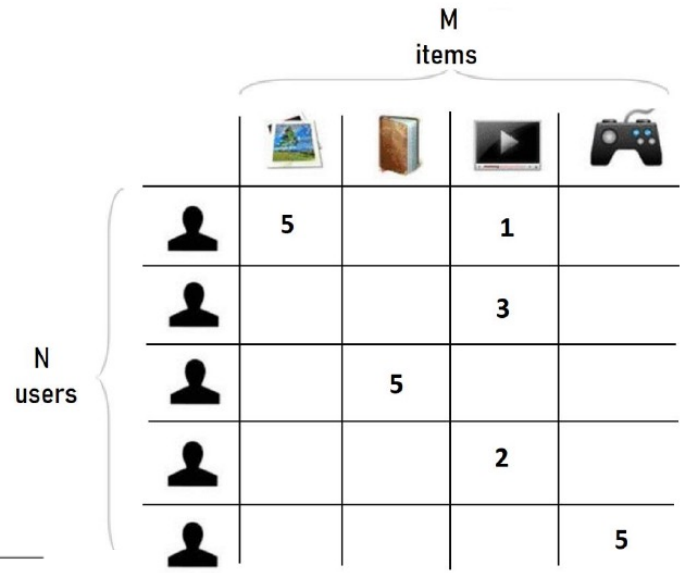
Feature Interaction Search-FIVES





Interaction Function Search-SIF

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



Interaction function:
how embedding vectors interact with each other

	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]
	$\max(\boldsymbol{u}_i, \boldsymbol{v}_j)$	max, min	$O((m+n)k)$	$O(k)$	ConvMF [25]
	$\sigma([\boldsymbol{u}_i; \boldsymbol{v}_j])$	concat	$O((m+n)k)$	$O(k)$	Deep&Wide [9]
	$\sigma(\boldsymbol{u}_i \odot \boldsymbol{v}_j + \boldsymbol{H}[\boldsymbol{u}_i; \boldsymbol{v}_j])$	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]

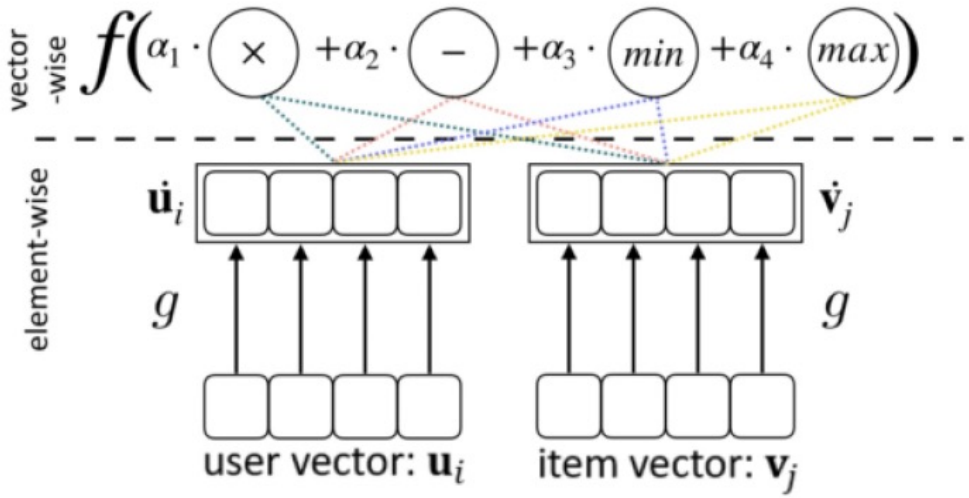
Efficient Neural Interaction Function Search for Collaborative Filtering. WWW 2020

Interaction Function Search-SIF

- SIF selects different interaction functions across different datasets.

IFC	operation
$\langle u_i, v_j \rangle$	inner product
$u_i - v_j$	plus (minus)
$\max(u_i, v_j)$	max, min
$\sigma([u_i; v_j])$	concat
$\sigma(u_i \odot v_j + H[u_i; v_j])$	multi, concat
$u_i * v_j$	conv
$u_i \otimes v_j$	outer product

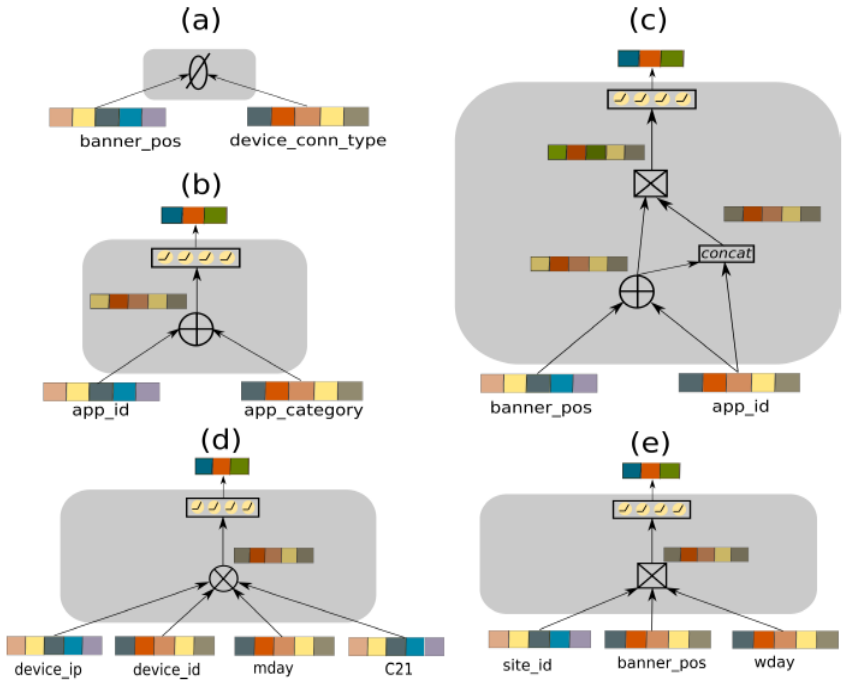
Cut the search space into two blocks

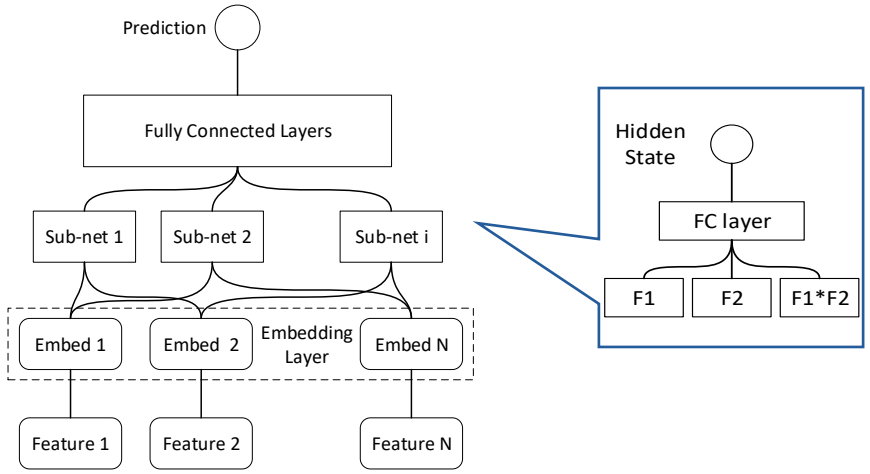


- Vector-level: simple linear algebra operations
- Elementwise: shared nonlinear transformation

Interaction Function Search-AutoFeature

- Not all the feature interactions between each pair of fields need to be modeled.
- Not all the useful feature interactions can be modeled by the same interaction functions

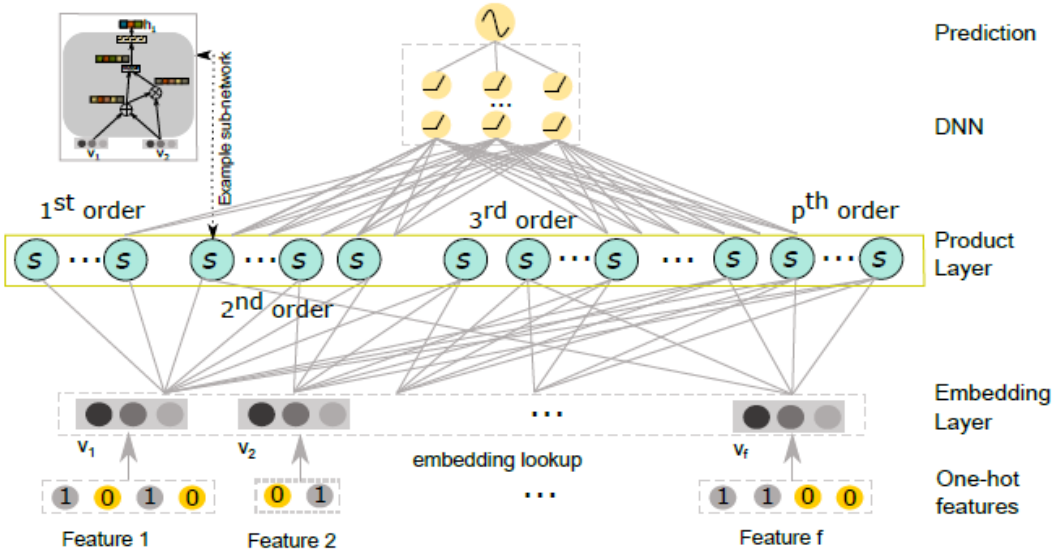


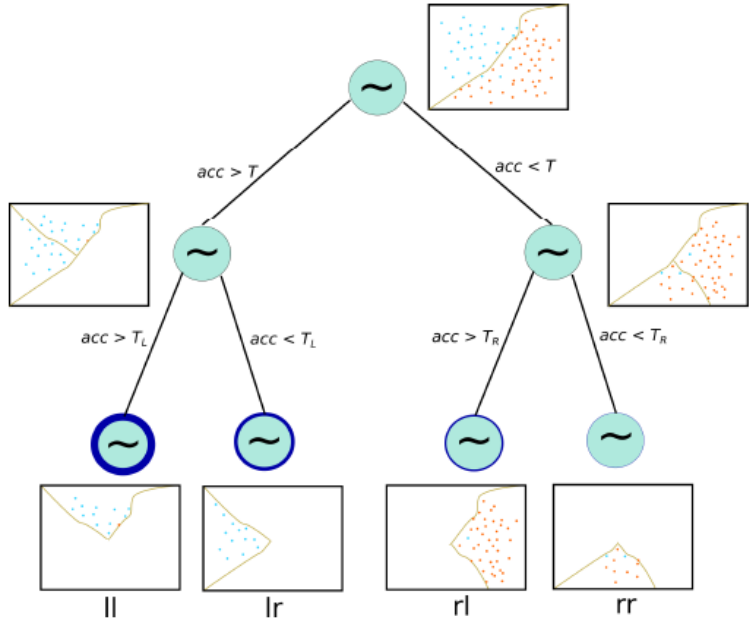


AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. CIKM 2020

Interaction Function Search-AutoFeature

- AutoFeature automatically designs a different sub-net for each pair of fields.
- Train a Naïve Bayes Tree to classify different network structures, where the tree tends to classify well-performed network structures into several leaves, such that the next generation can be more effective.

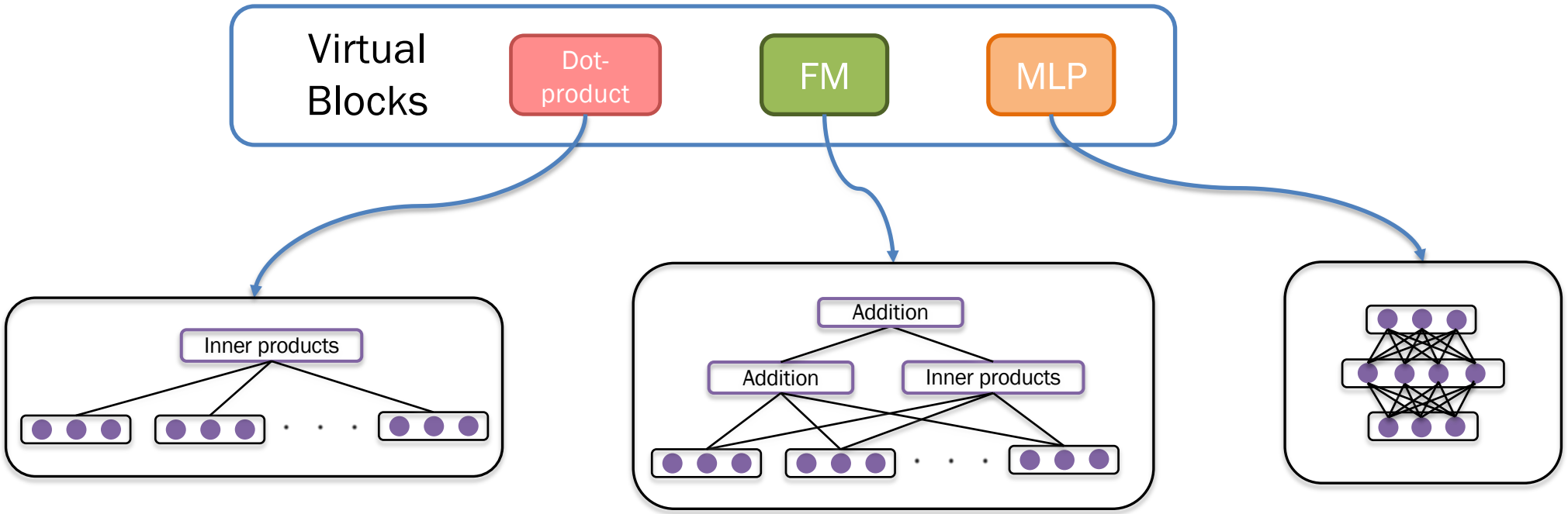




Interaction Block Search-AutoCTR

Hierarchical Search Space

- Properties: functionality complementary, complexity aware, …
- Examples: MLP block, dot-product block, factorization-machine block, …

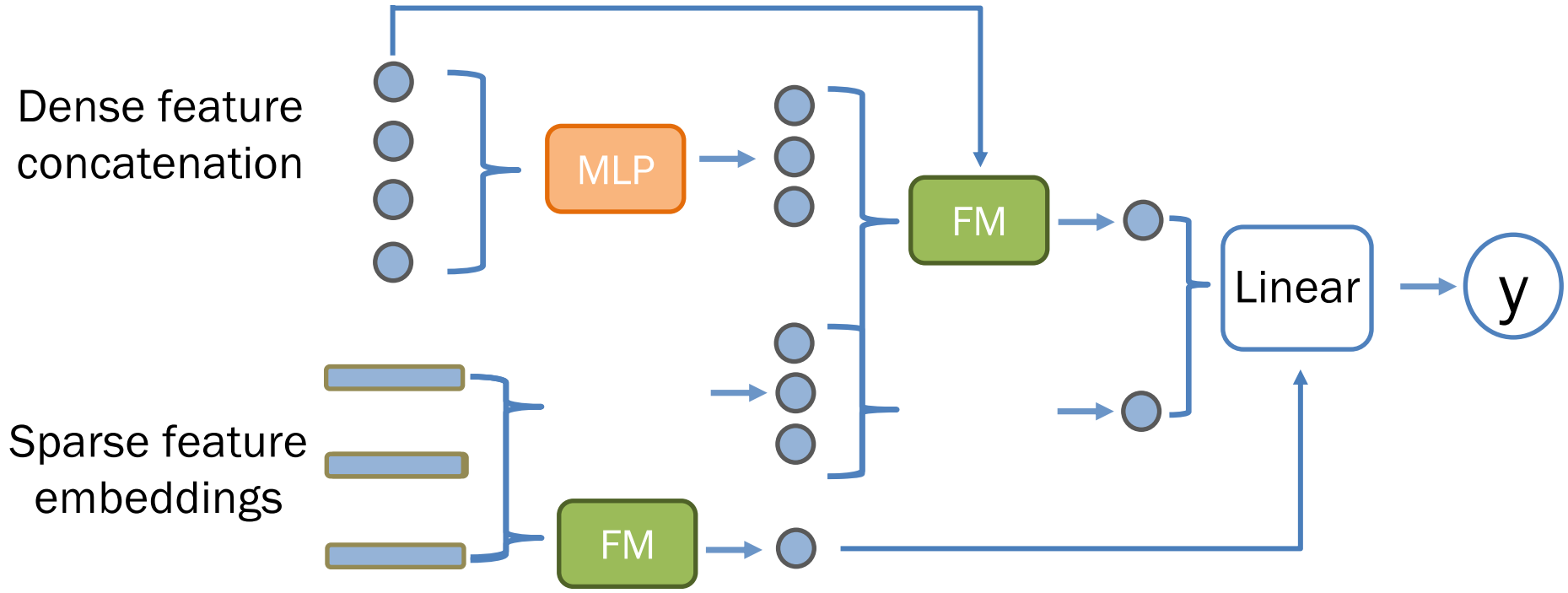


Towards Automated Neural Interaction Discovery for Click-Through Rate Prediction. KDD 2020

Interaction Block Search-AutoCTR

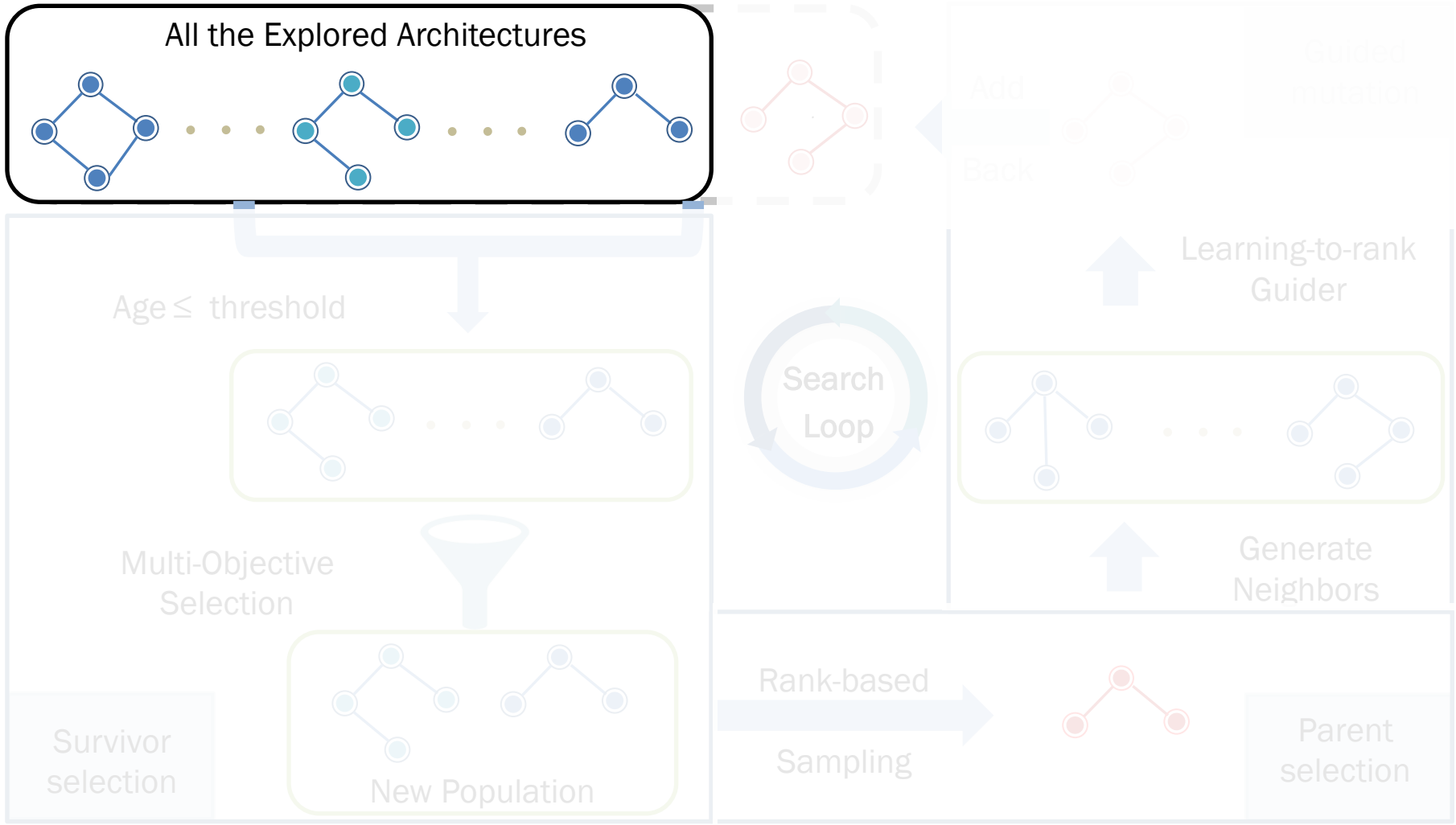
Search space construction

- DAG of virtual blocks and grouped feature embeddings
- Both block hyper-parameters and connection among blocks are to be searched



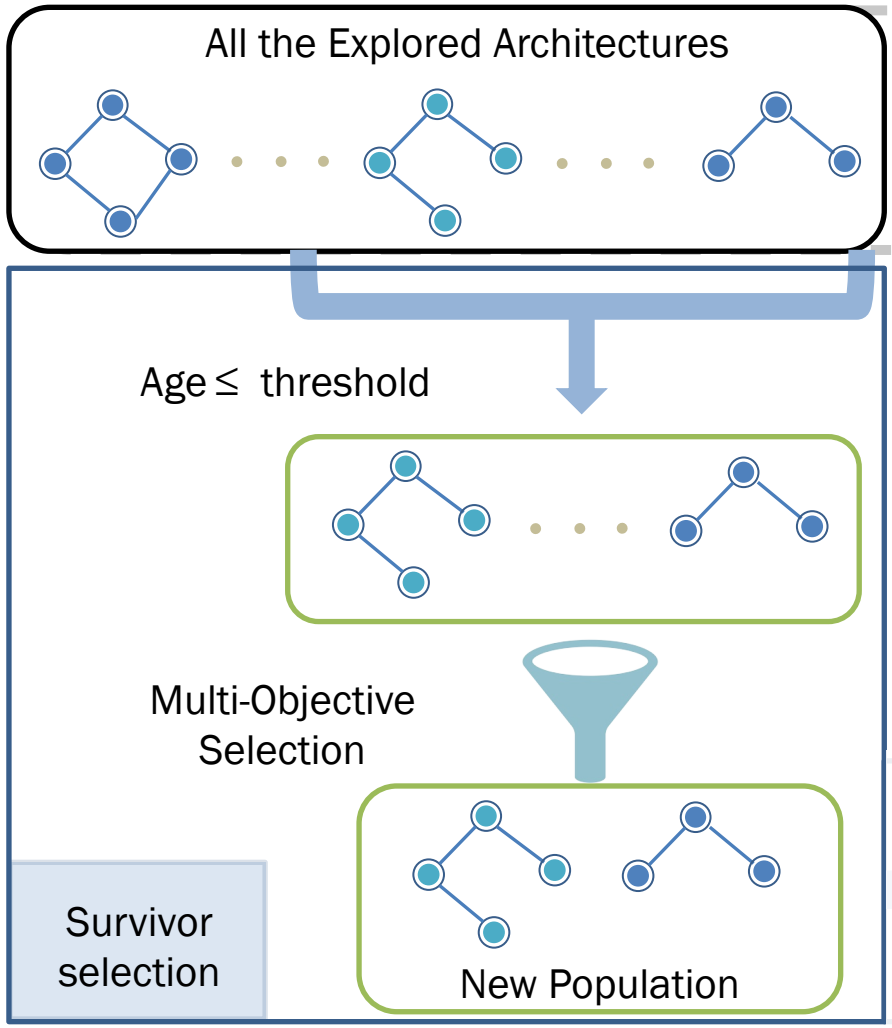
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



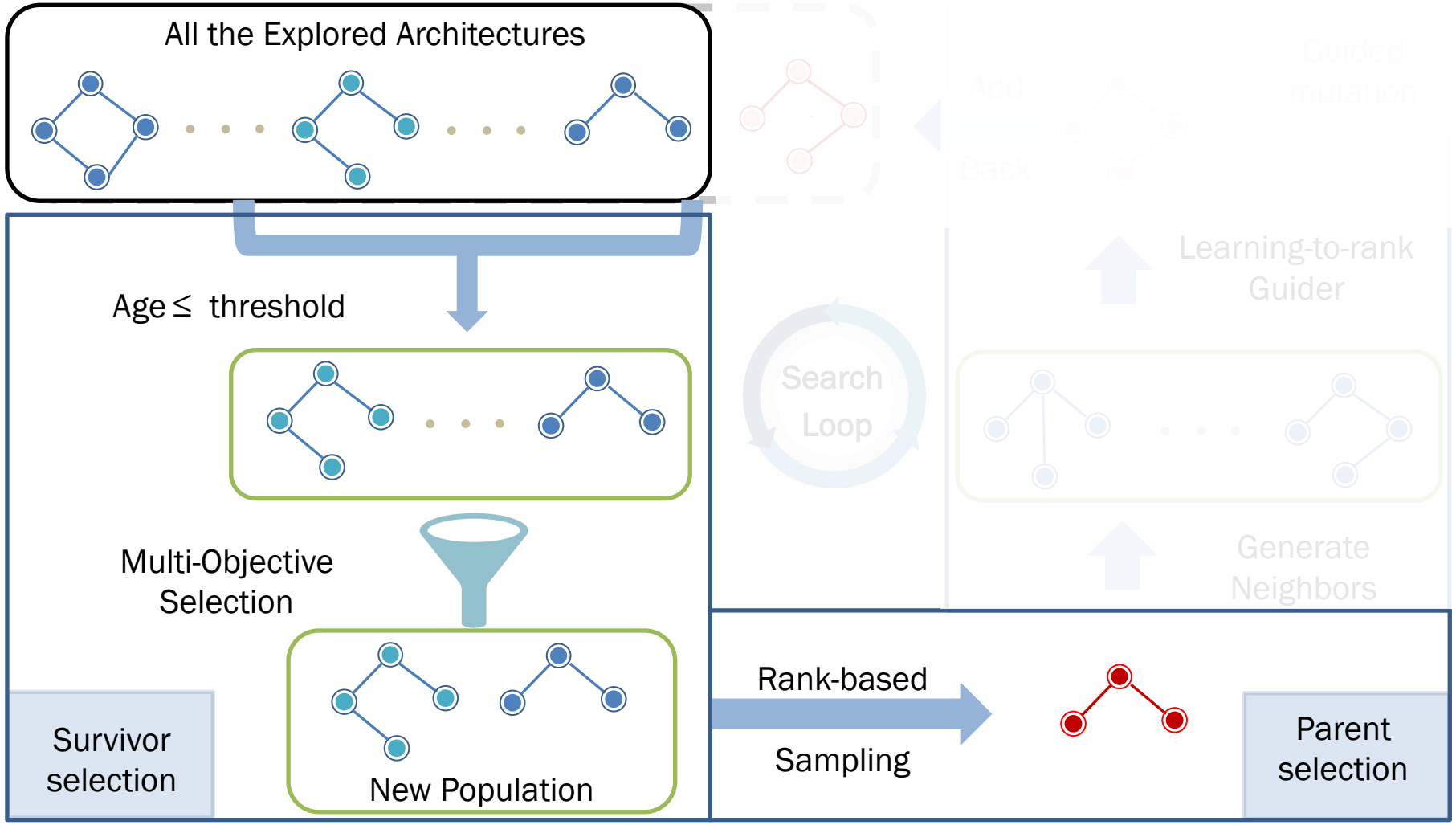
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



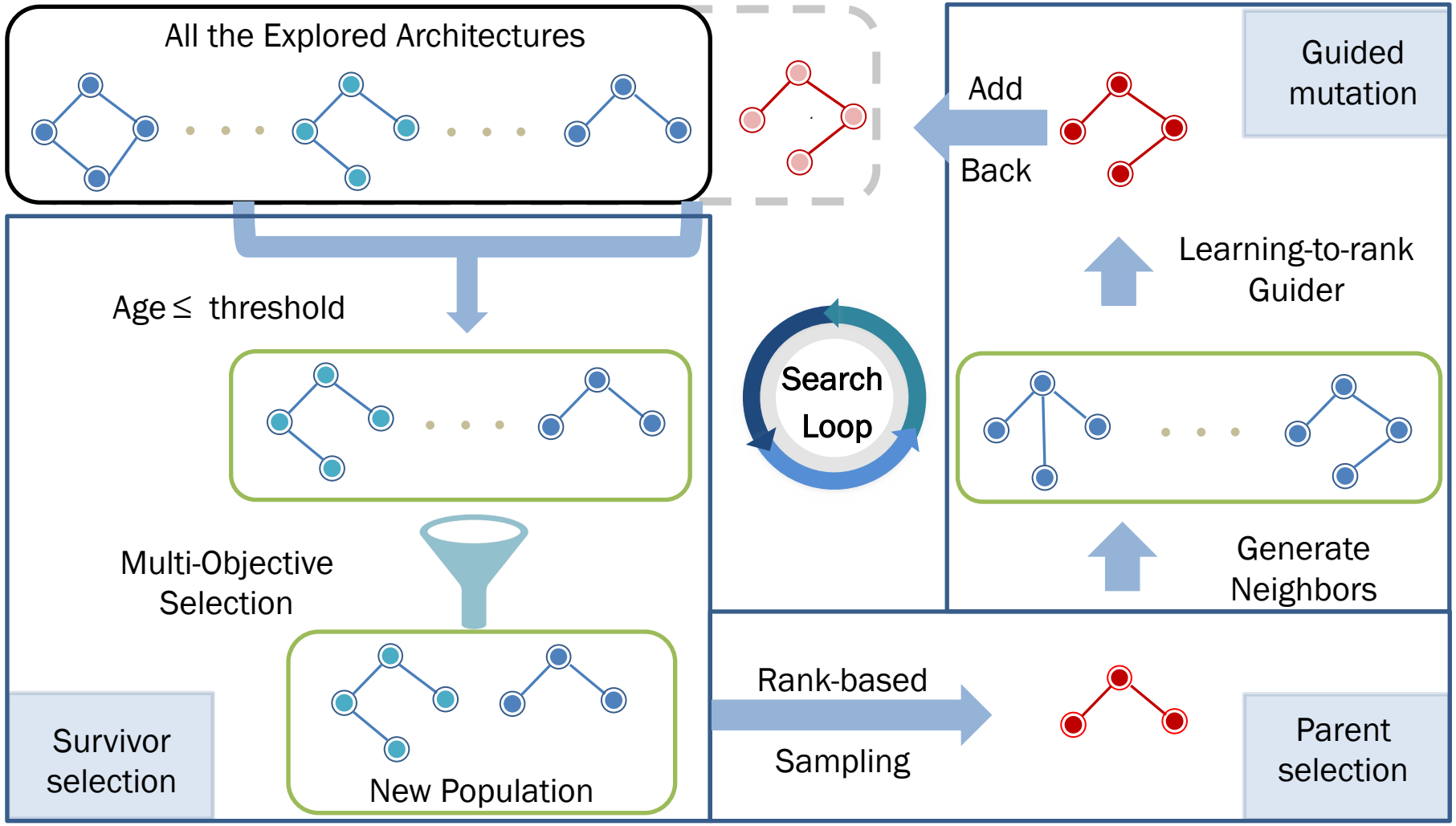
Interaction Block Search-AutoCTR

Multi-Objective Evolutionary Search Algorithm



Interaction Block Search-AutoCTR

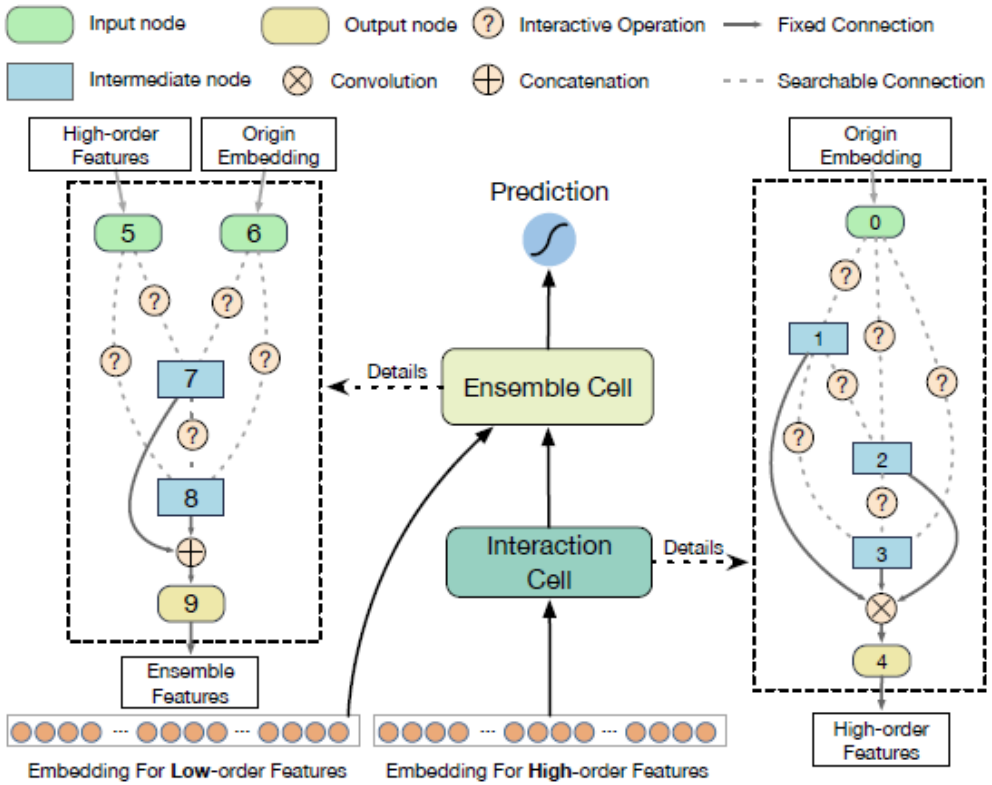
Multi-Objective Evolutionary Search Algorithm



Interaction Block Search-AutoPI

Search Space

- The interaction cell formulates the higher-order feature interactions
- The ensemble cell formulates the ensemble of lower-order and higher-order interactions



- Skip-connection[1]

$$\boldsymbol{E}' = o_{\text{skip}}(\boldsymbol{E}) = \boldsymbol{E} \in \mathbb{R}^{m \times k}$$

- SENET Layer[2]

$$\boldsymbol{E}' = \left[a'_1 \cdot \boldsymbol{x}_1, a'_2 \cdot \boldsymbol{x}_2, \cdots, a'_m \cdot \boldsymbol{x}_m\right] \in \mathbb{R}^{m \times k}$$

- Self-attention[3]

$$\boldsymbol{E}' = \left[\boldsymbol{e}_1^{\text{Res}}, \boldsymbol{e}_2^{\text{Res}}, \cdots, \boldsymbol{e}_m^{\text{Res}}\right] \in \mathbb{R}^{m \times k}$$

- FM Layer

$$\boldsymbol{p} = \{(\boldsymbol{x}_j \cdot \boldsymbol{x}_j)\}_{(i,j) \in R_x}$$
$$\text{s.t. } \boldsymbol{R}_x = \{(i,j)\}_{i \in \{1,\cdots,m\}, j \in \{1,\cdots,m\}, i<j}$$

- FC Layer

$$\boldsymbol{E}' = \boldsymbol{e} \cdot \boldsymbol{W}_{\text{FC}}$$

- 1D Conv

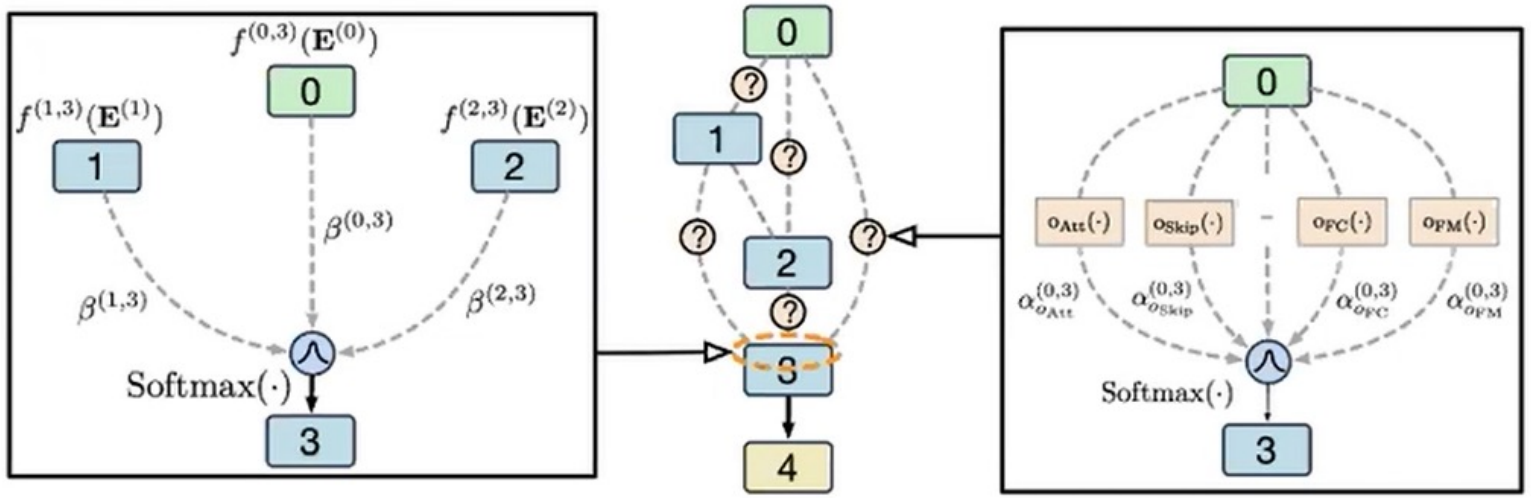
$$\left\{\boldsymbol{C}^{(i)} \in \mathbb{R}^{1 \times 1 \times m}\right\}_{i \in \{1,\cdots,m\}}$$

A General Method For Automatic Discovery of Powerful Interactions In Click-Through Rate Prediction. SIGIR 2021

Interaction Block Search-AutoPI

Search Strategy

- Continuous relaxation



Continuous relaxation visualization

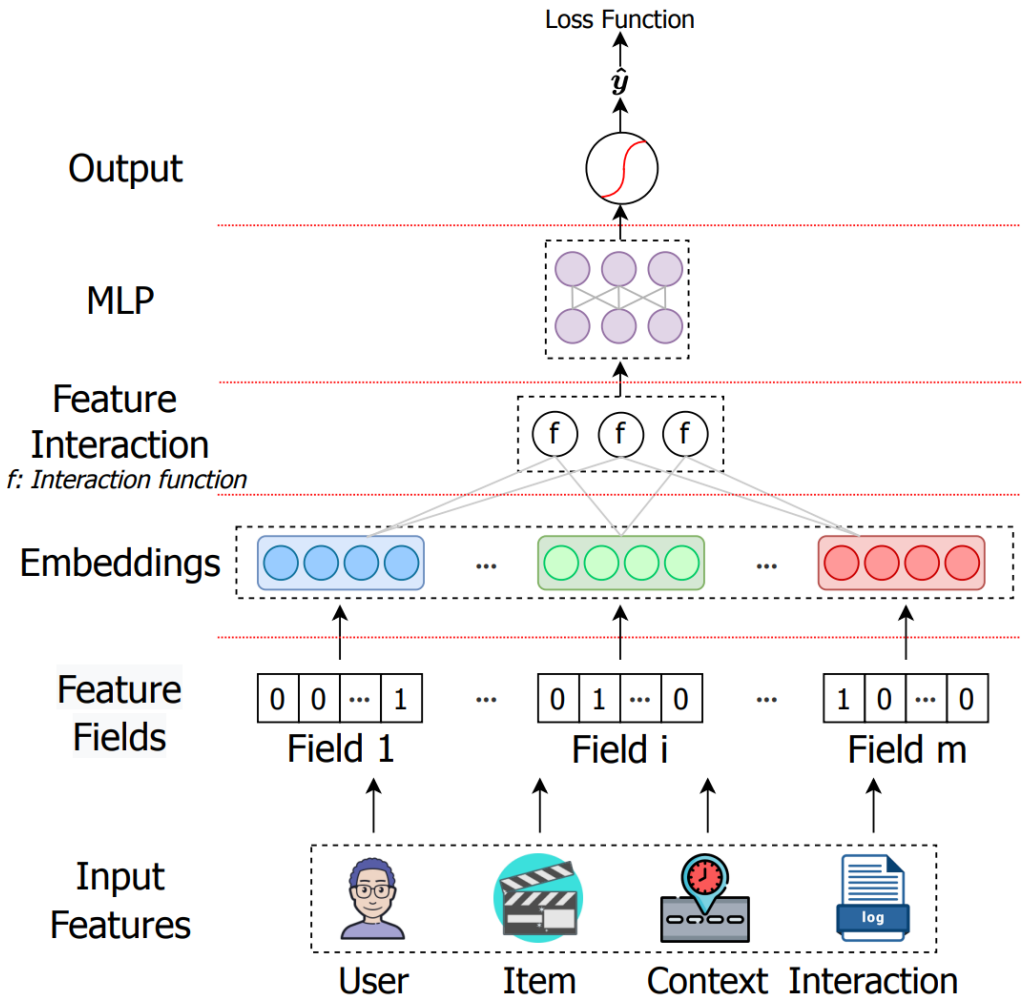
By introducing the **operator-level** and **edge-level architecture parameters** for continuous relaxation, a **differentiable objective function** will be obtained:

$$\min_{\alpha,\beta} \mathcal{L}_{val}\left(w^*(\alpha,\beta),\alpha,\beta\right)$$

$$\text{s.t. } w^*(\alpha,\beta) = \text{argmin}_w \mathcal{L}_{\text{train}}(w,\alpha,\beta)$$

Table of Contents

Background



- Comprehensive Search:
  - Search for several parts of DRS
  - E.g. Embedding size & Feature Interaction function
- System Design:
  - Search for architectures other than aforementioned parts
  - E.g. Loss funcion

Comprehensive Search & System

	Model Name	Search Strategy	Search Space
Comprehensive Search	**AIM**	Gradient	Embedding Dimension, Intearaction Function, Feature Interaction
	AutoIAS	Reinforcement Learning	Embedding Dimension, Projection Dimension, Interaction Function, Feature Interaction, MLP Layer & Dimension
	AMEIR	One-shot Random Search	Sequential model, Feature interaction, MLP Dimension
System Design	**AutoLoss**	Gradient	Optimization: Loss Function
	AutoGSR	Gradient	Structure Design: GNN Architecture
	AutoFT	Gradient	Parameter Tuning: Fine-Tune or Not (For pre-trained models)

AMEIR

- Motivation:
  - RS could be divided into 3 parts: Behavior modeling, feature interaction, MLP
  - Hard to find a unified model for all scenarios
  - Existing NAS methods for Recommendation are restricted

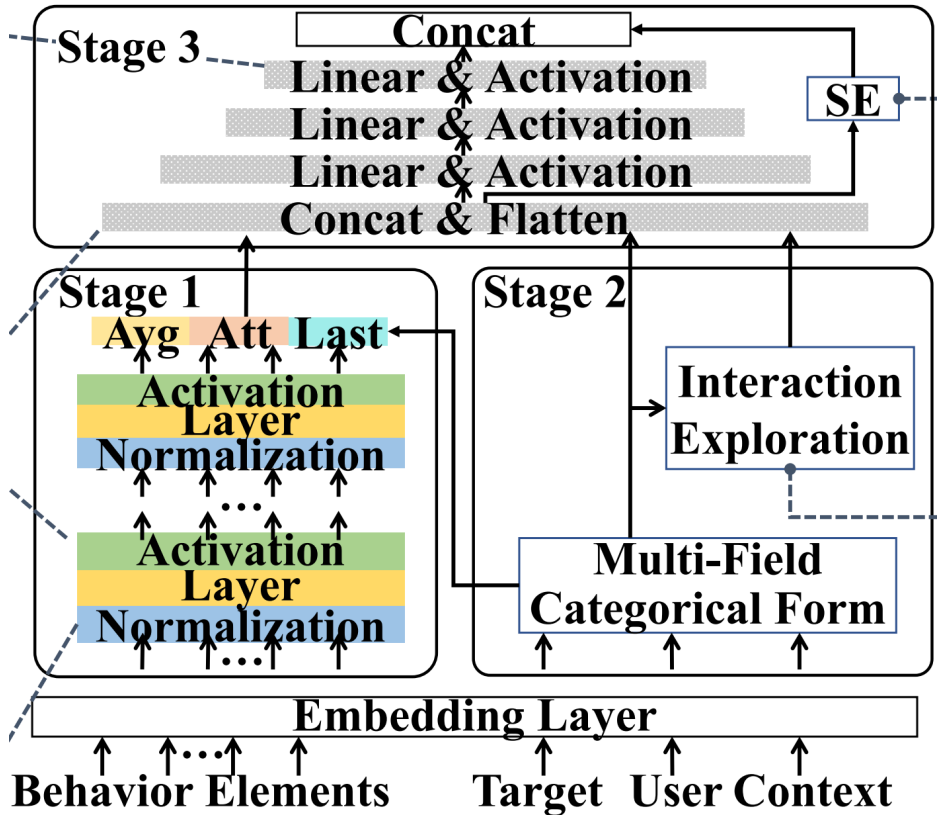
- Problem:
  - Automatically find the **complete** recommendation models for adapting **various** recommender tasks

- Solution:
  - Three-step one-shot random search, progressively construct the architecture

AMEIR: Automatic Behavior Modeling, Interaction Exploration and MLP Investigation in the Recommender System. IJCAI 2021

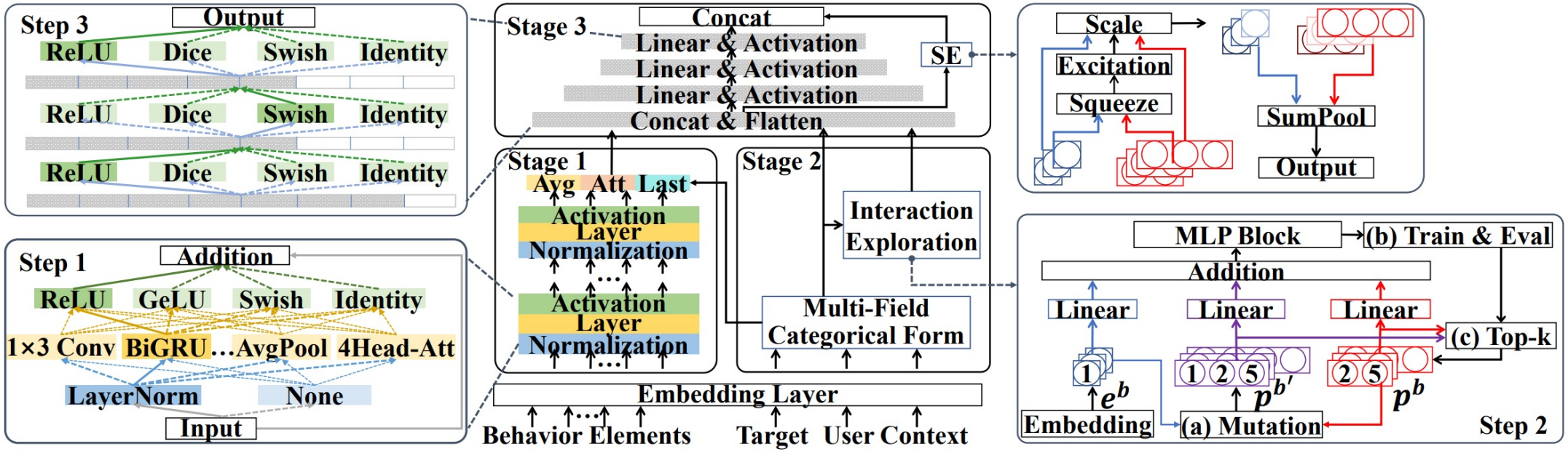
AMEIR – Search Space

- Stage 1: (Behavior Modeling for Sequential features)
  - Search for a fixed number of layers (L)
  - Normalization: {Layer Norm, None}
  - Layer: {Conv, Recur, Pooling, Attention} & Zero
  - Activation: {ReLU, GeLU, Swish, Identity}
  - Output: (Average, Attn, Last) please refer to the article
- Stage 2: (Interaction Exploration for non-Sequential features)
  - Interaction function: Hadamard Product (Fixed)
  - Only Search for feature combination for interaction
- Stage 3: (MLP investigation)
  - MLP Dimension: {0.1, 0.2, ..., 1.0} of input size
  - Activation: {ReLU, Swish, Identity, Dice}

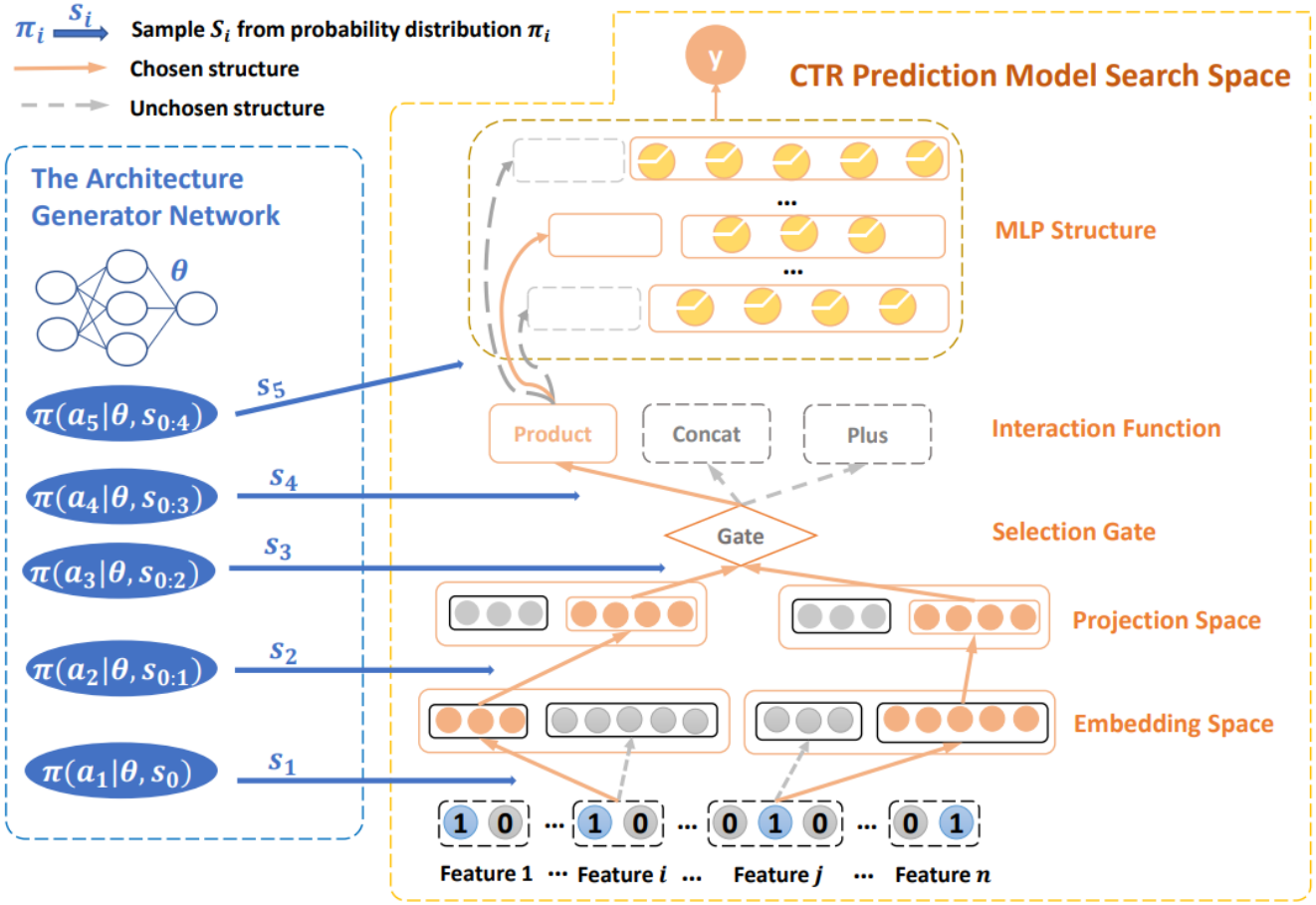


AMEIR

- Overall Search Strategy: One-shot random search
- Step 1: Using a predefined MLP, search for the optimal architecture.
- Step 2: Combined with SMBO, progressively expand the interaction sets, also use a predefined MLP.
- Step 3: Design a weight matrix of maximal dimension to realize one-shot search



AutoIAS



- Search Space:
  - Embedding: Embedding Size
  - Projection: Unified Size for interaction
  - Feature selection: Pairs of feature
  - Interaction Function: {Product, …}
  - MLP:
    - The number of layers
    - Layer dimensions
- Strategy: Reinforcement Learning

AutoIAS: Automatic Integrated Architecture Searcher for Click-Trough Rate Prediction. CIKM 2021

AutoLoss

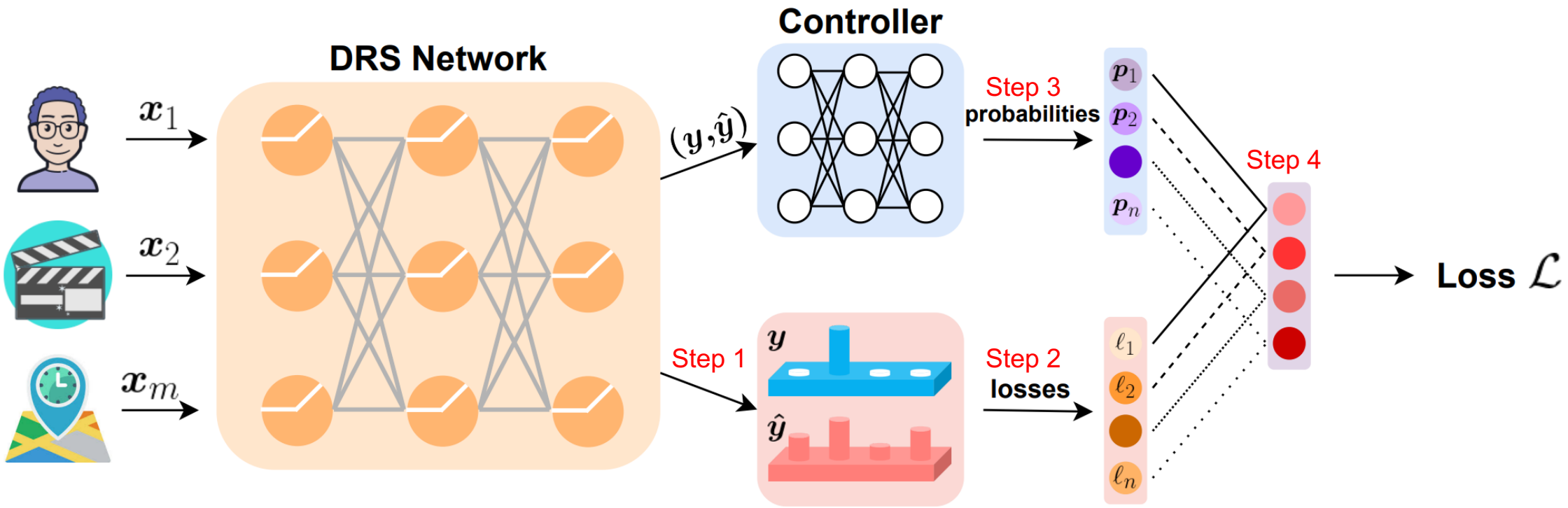
- Motivation:
  - Traditional Setting: Predefined and fixed loss function, sub-optimal
  - Exhaustively or manually searched fused loss: Costly, neglecting the difference between data examples

- Problem:
  - automatically and adaptively assign the appropriate loss function for each data ex

- Solution:
  - Design a controller network to adaptively adjust the probabilities over multiple loss functions for different data examples

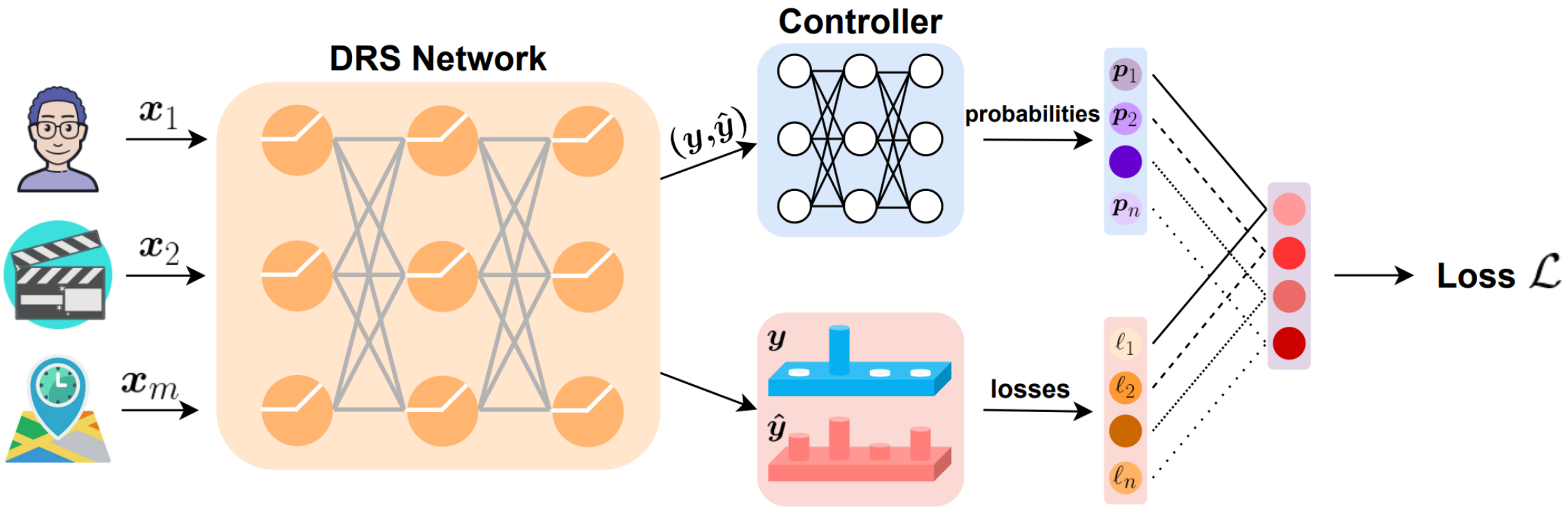
AutoLoss – Forward-propagation

- Step 1: The DRS makes predictions
- Step 2: Calculate candidate losses
- Step 3: The controller generate weights(probabilities) according to predictions.
- Step 4: Calculate the overall Loss (Weighted sum)



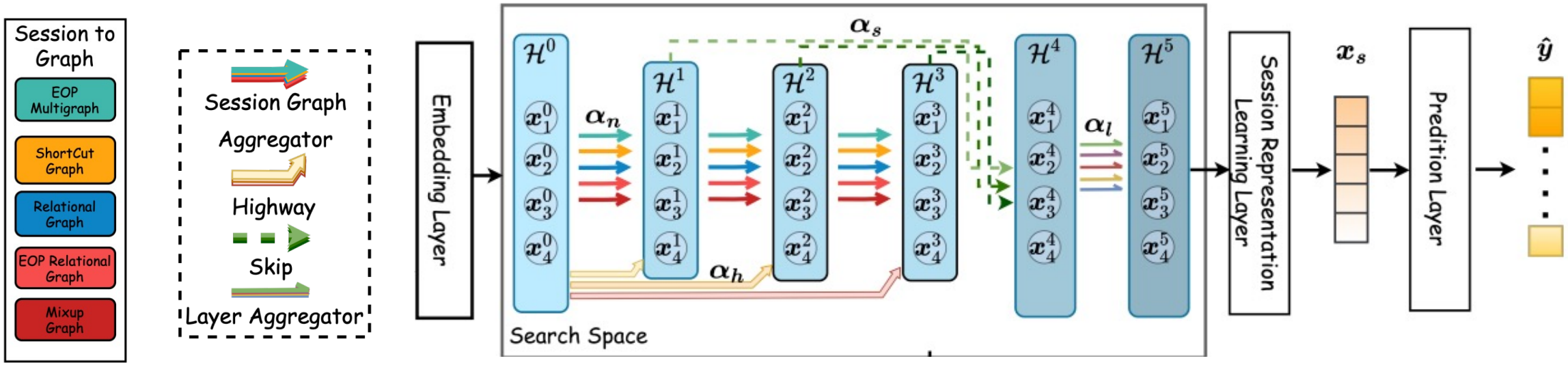
AutoLoss – Backward-propagation

- Parameters of the DRS network: Updated based on Training Error
- Parameters of the Controller: Updated based on Validation Error



AutoGSR

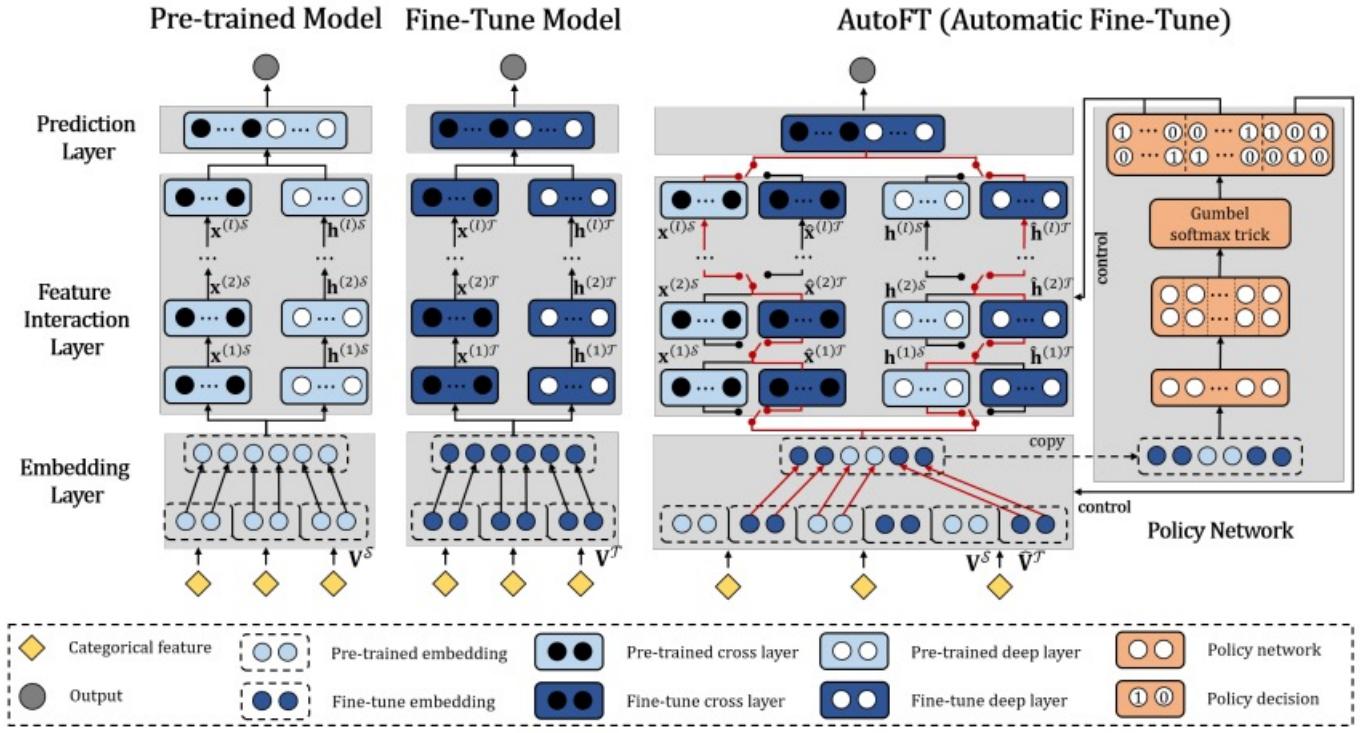
- Target: Search GNN architectures for Session-based Recommendation
- Methodology:
  - Search Space:
    - Session Aggregation: 5 popular graph types.
    - Layer Aggregation: mean, max, concat, sum & highway & skip.
  - Strategy: Continuous relaxion & Gradient



AutoGSR: Neural Architecture Search for Graph-based Session Recommendation. SIGIR 2022

AutoFT

- Target: Automatically which parts of pre-trained model should be fine-tuned
- Search Space:
  - Field-wise transfer: use fine-tuned embedding or not
  - Layer-wise transfer: Whether to fine-tune cross & deep layers
- Strategy: Continuous relaxion & Gradient



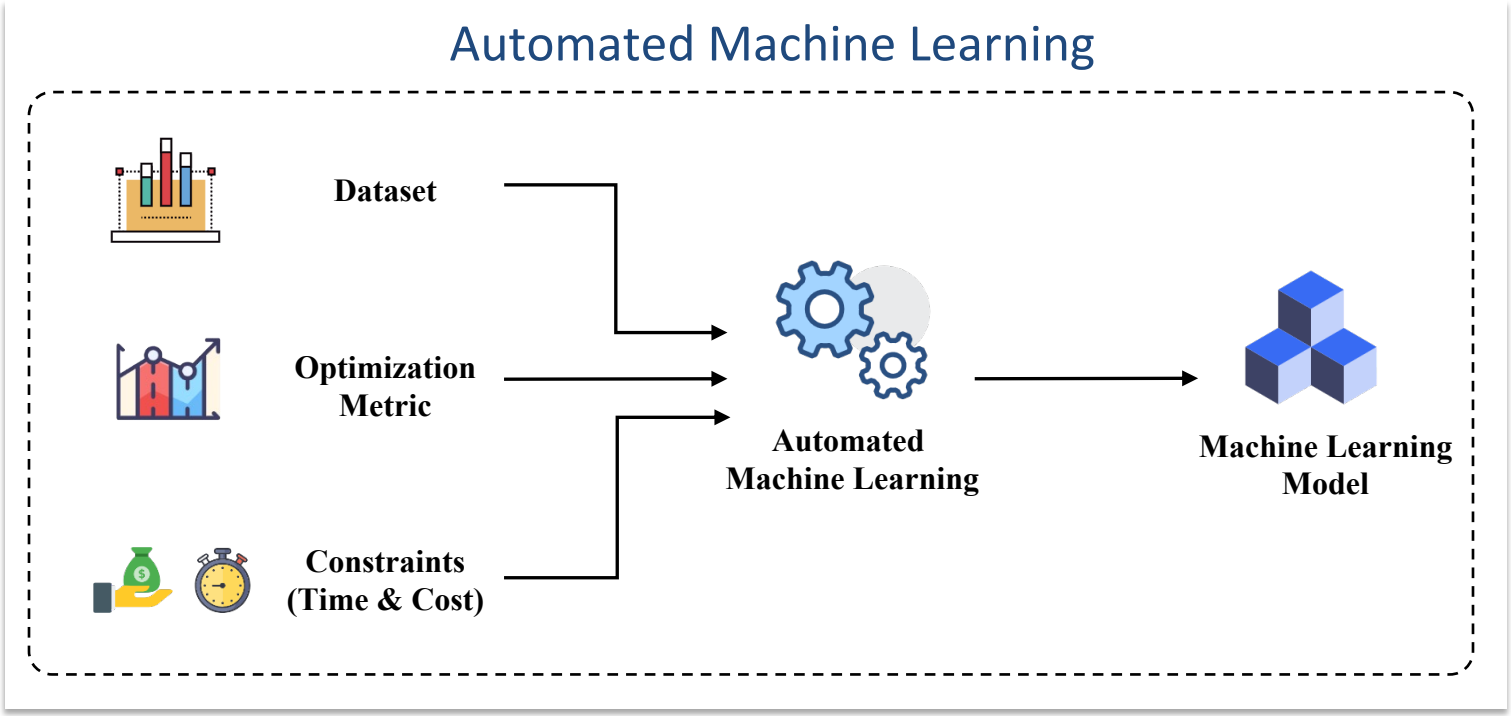
AutoFT: Automatic Fine-Tune for Parameters Transfer Learning in Click-Through Rate Prediction. In arxiv

Table of Contents

Conclusion

Automated Machine Learning contribute to improving the performance of recommender systems in a data-driven manner.

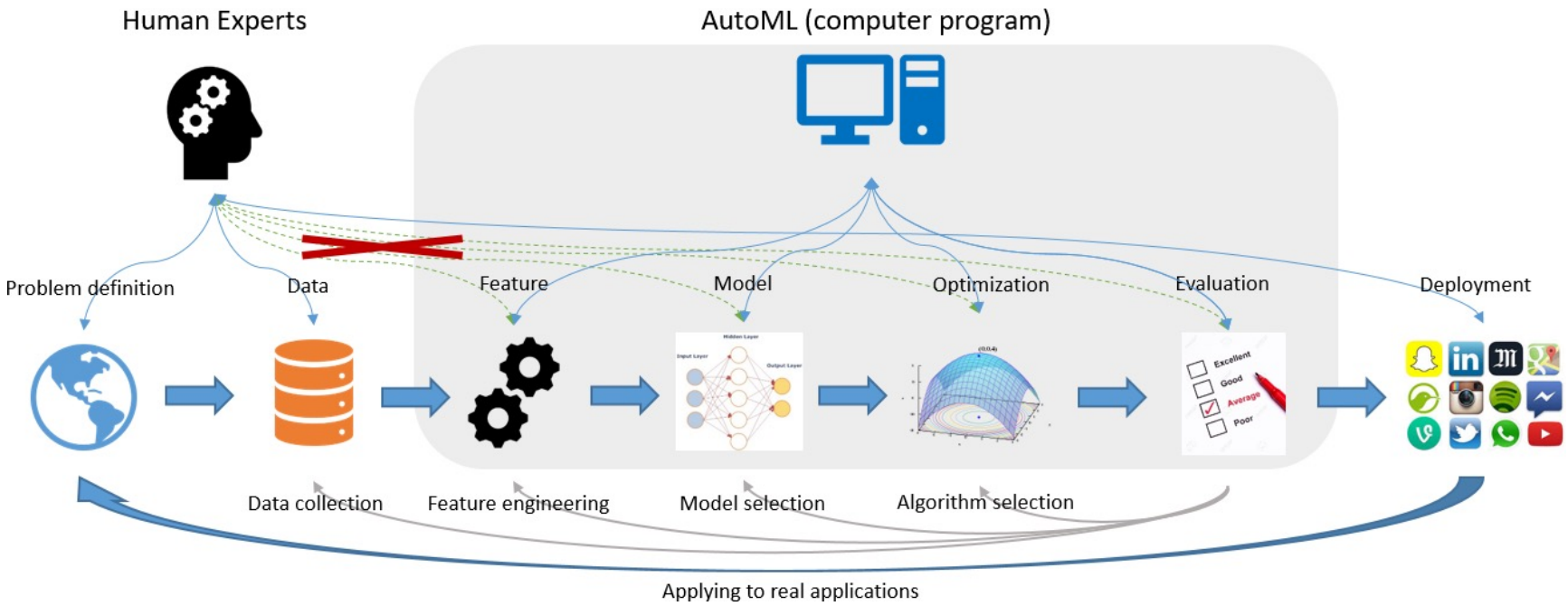
- Automatically search embedding dimensions to better model **feature representations**
- Automatically design deep networks to better capture **feature interactions**
- Automatically design **comprehensive system architectures** to better improve performance



Conclusion

AutoML advantages:

- Less expert knowledge
- Saving time and efforts
- Different data → different architectures



Future Directions

1) Feature Selection

- Recommendation scenario contains a plenty of features (user, item, context, and cross features)
- Automatically search optimal feature sub-set under hardware and latency constraints

2) Model Selection

- Search different sub-models/sub-architectures according to different requests adaptively

3) User Behavior Modeling

- User history behaviors contain different dimensions of interests
- Automatically retrieve beneficial history behaviors for modeling user preference

Future Directions

4) Muti-task learning

- Multi-task learning is one of the most important techniques in industry recommendation for considering different revenue targets (e.g., ctr, cvr, vv)
- Designing an automatic algorithm for recommendation based on muti-task learning

5) GNNs-based Recommendations

- Exploring the combination of AutoML and GNNs provides great opportunities to further boost the performance of GNNs-based recommendation methods.

6) Multi-Modality Recommendations

- Designing an optimal algorithm via automated machine learning techniques to advance deep multimodal learning for recommendation.

Thank you!