Automated machine learning

Quanming Yao

Assistant Prof. EE. Tsinghua

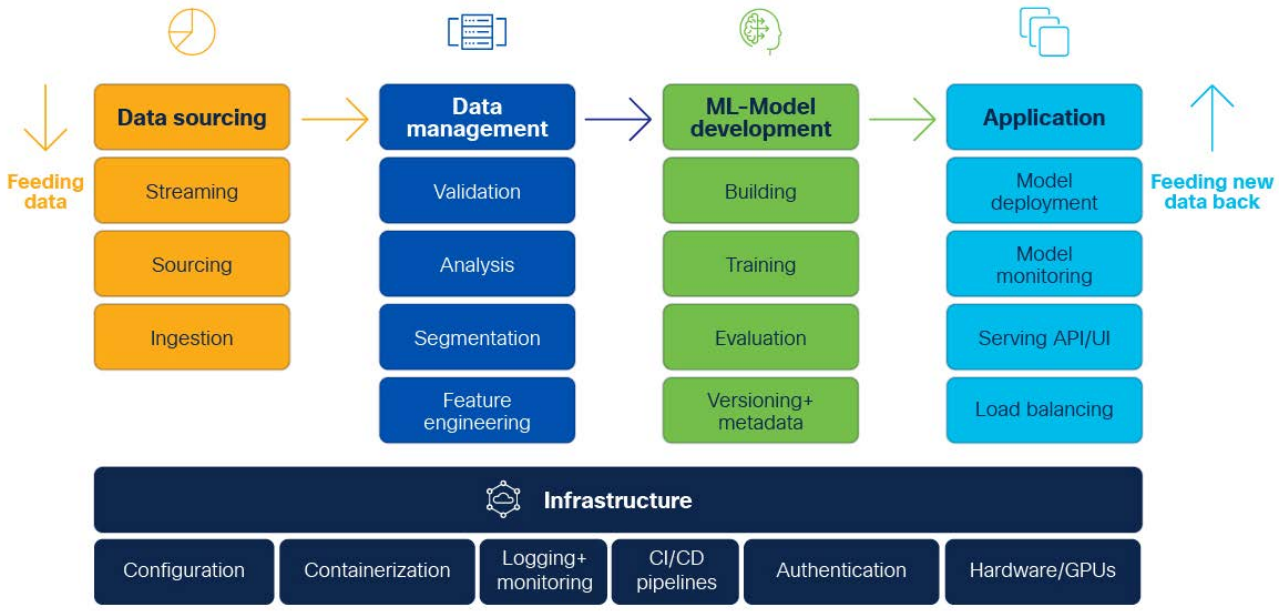
qyaoaa@tsinghua.edu.cn

Outline

- What is AutoML
- How to do AutoML
- Application examples of AutoML
- Theory insights

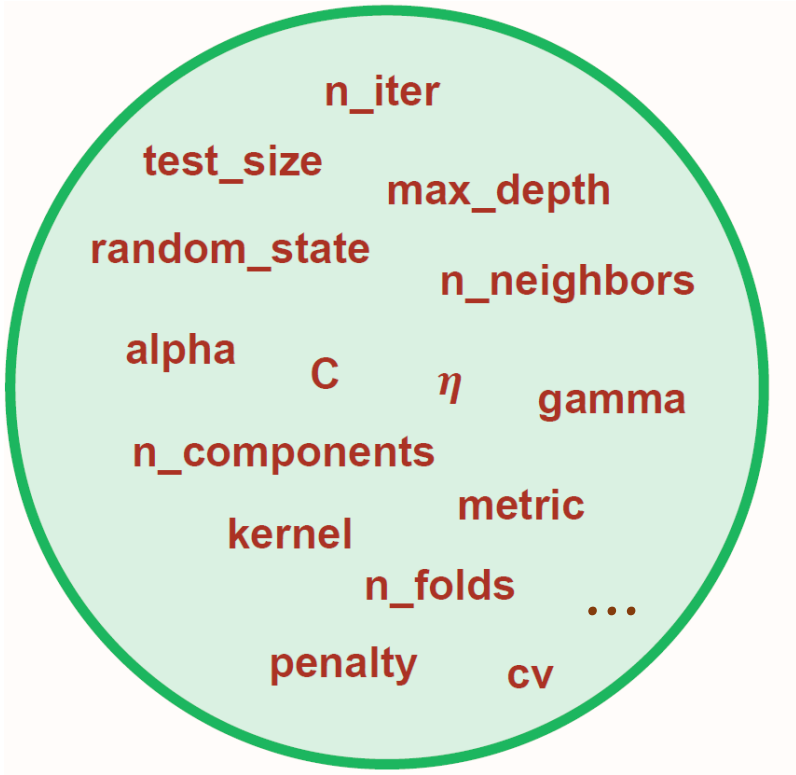
Common problem in machine learning

- For a certain machine learning problem, how to choose different components to form machine learning pipelines?
  - How to preprocess the data?
  - How to screen data feature?
  - Need to use pre-training model?
  - What kind of model to use?



Common problem in machine learning

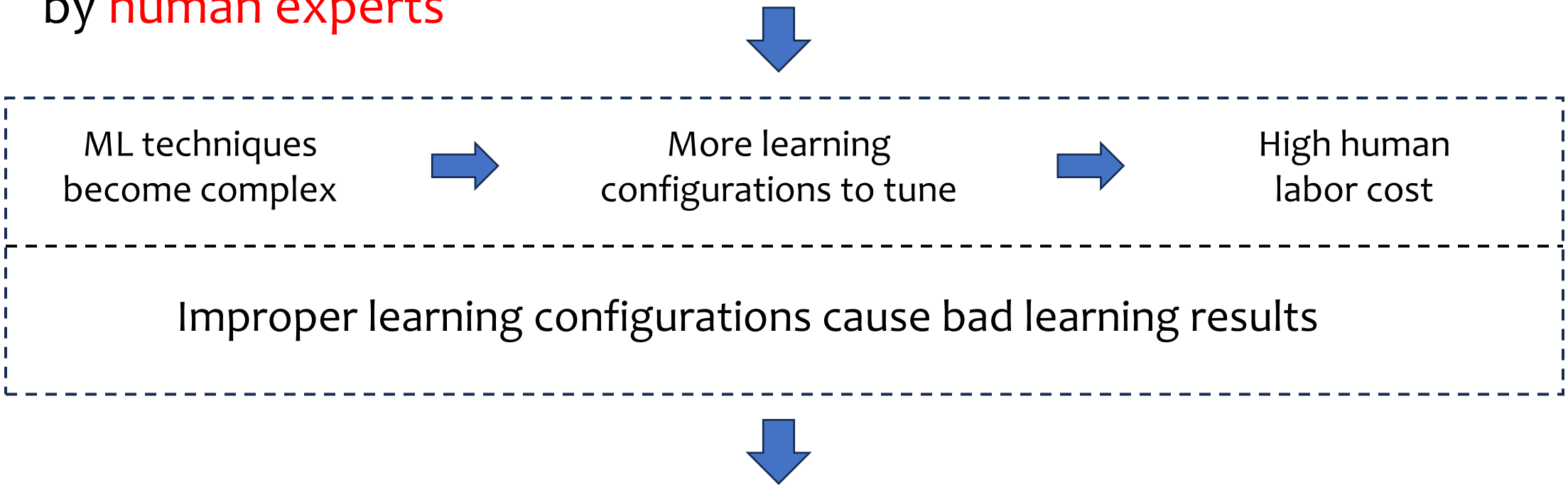
- For a specific model, which set of hyperparameters performs the best?
- Candidate hyper-parameters:
  - Learning rate
  - Optimization algorithm
  - Batch size
  - Dropout rate
  - …



Hyper-parameters

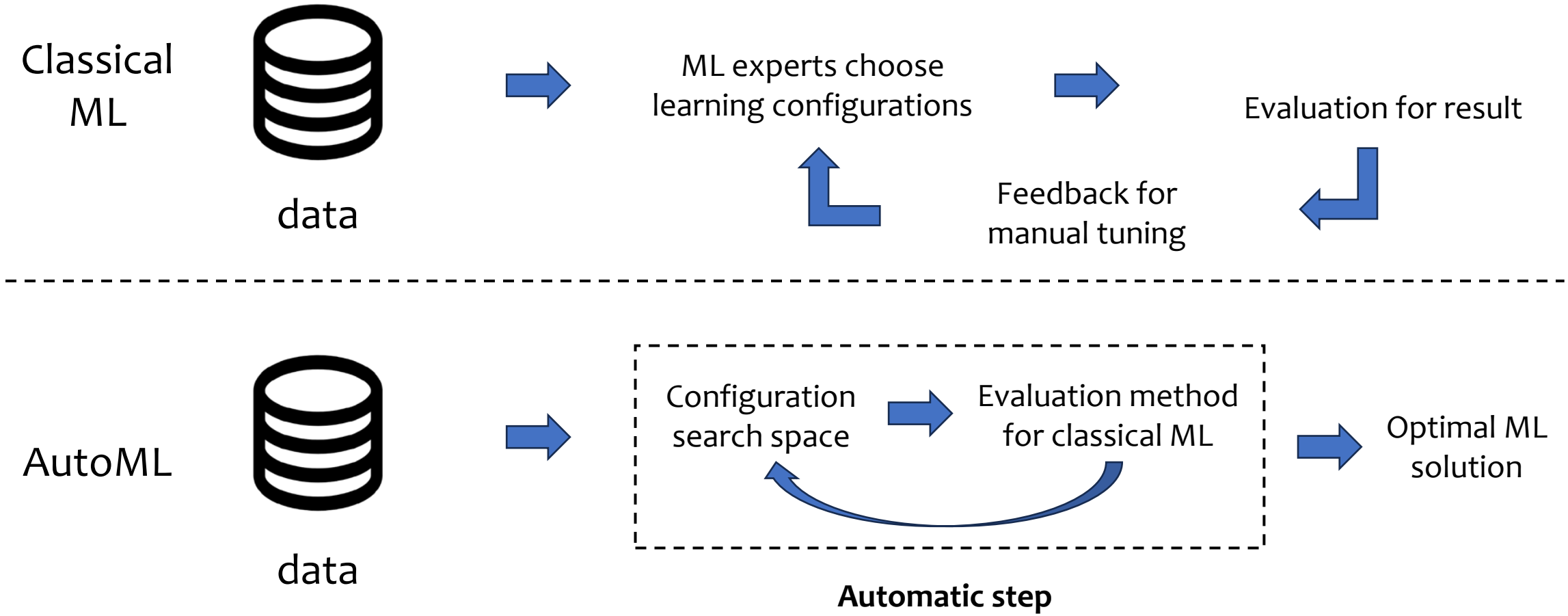
Problem for tuning learning configurations

- In classical machine learning, learning configurations are designed by human experts



Need for AutoML: let machine find optimal learning configurations automatically

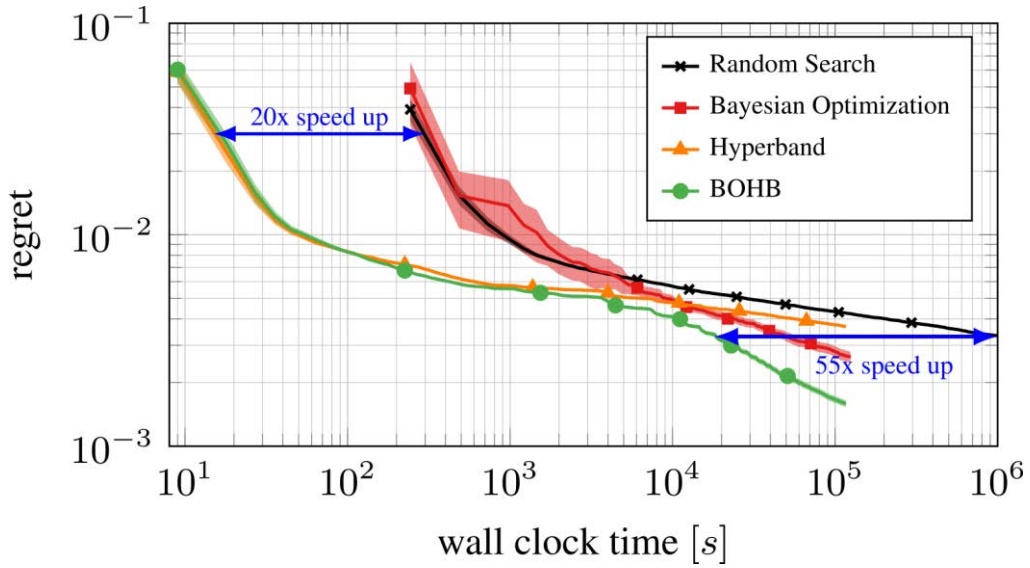
Classical machine learning - AutoML



Example: Hyper-parameter optimization

An example of hyper-parameter search space:

Hyper-parameter	Range
Learning rate	[0.1,0.01,0.001,0.0001]
Momentum	[0.99,0.97,0.95,0.90,0.80]
Dropout rate	[0,0.1,0.2,0.3,0.4,0.5]
Optimizer	[Adam,SGD,AdamW,Rmsprop]
Hidden dimension	[50,100,150,200,250,300]

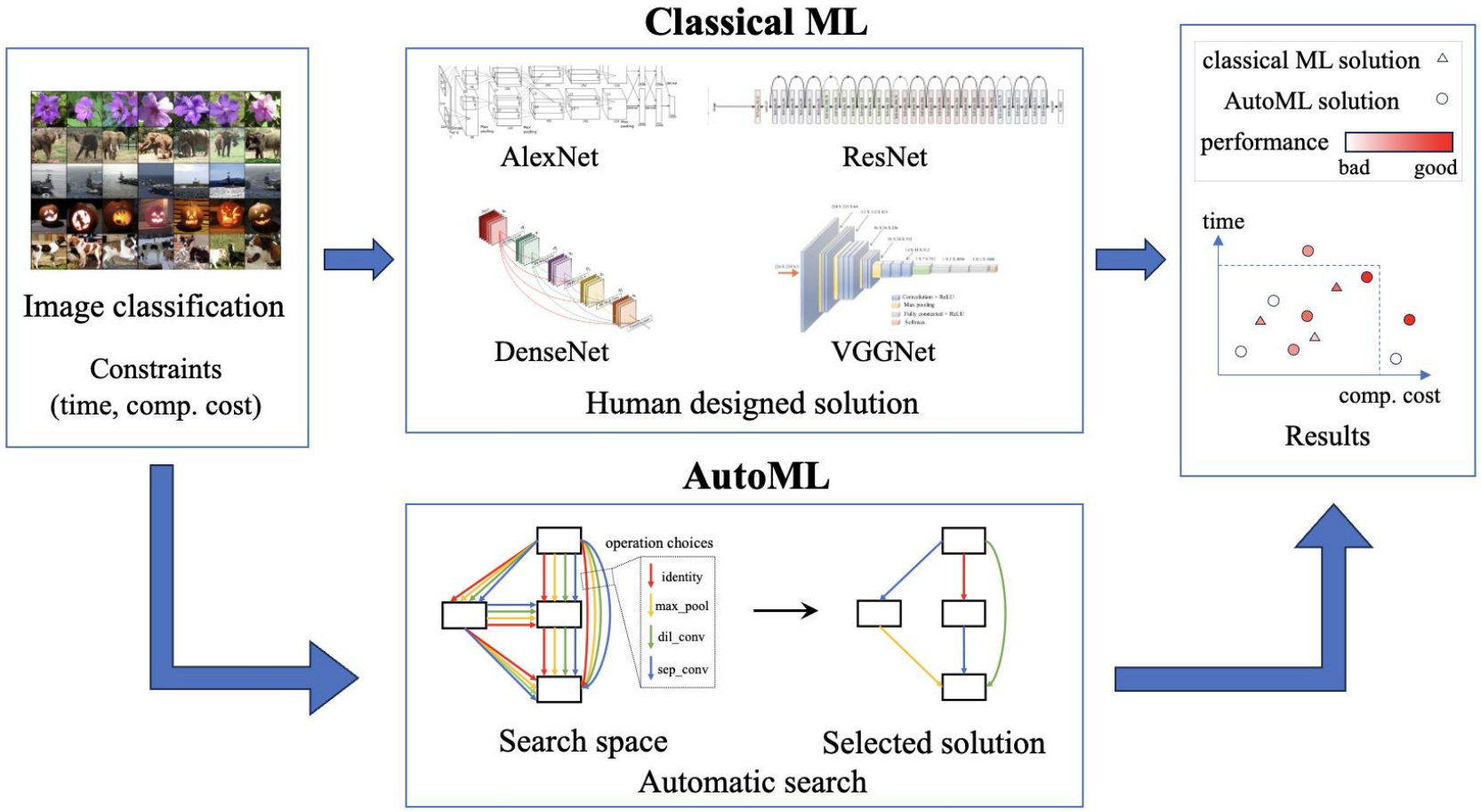


Common classical ML solution:
Manual search: high human labor cost
Exhaustive search: high evaluation cost
(2380 different configurations)

AutoML solution:
e.g. Bayesian optimization
Bring more than 20 * speed up

Example: Neural architecture search

- Classical ML manually design architecture based on experts' experience
- AutoML automatically search for the optimal architecture in the search space



Why need AutoML? Commerical applications

H2O AutoML

Typical applications

Data Preprocessing

- Imputation, one-hot encoding, standardization
- Feature selection and/or feature extraction
- Count/Label/Target encoding of categorical features

Model Generation

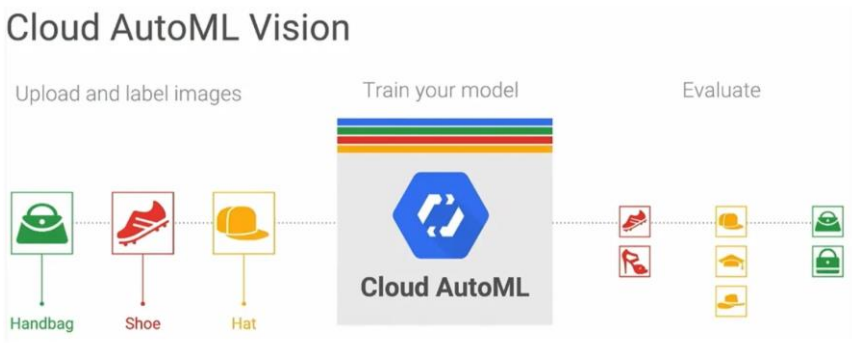
- Cartesian grid search or random grid search
- Bayesian hyperparameter optimization
- Individual models can be tuned using a validation set

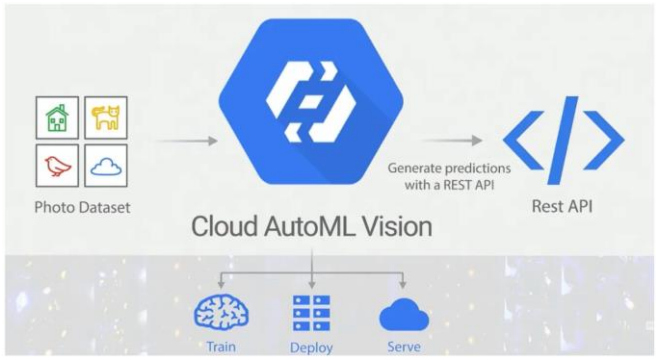
Ensembles

- Ensembles often out-perform individual models
- Stacking/Super Learning
- Ensemble Selection

Why need AutoML? Commerical applications

- Google cloud AutoML
  - Cloud AutoML Vision: help automating image classification and search optimization
  - Typical customers: Disney and Urban Outfitters.





Outline

- What is AutoML
- How to do AutoML
  - Problem definition and general learning strategy
  - Search space
  - Search algorithm
  - Evaluation strategy
- Application examples of AutoML
- Theory insights

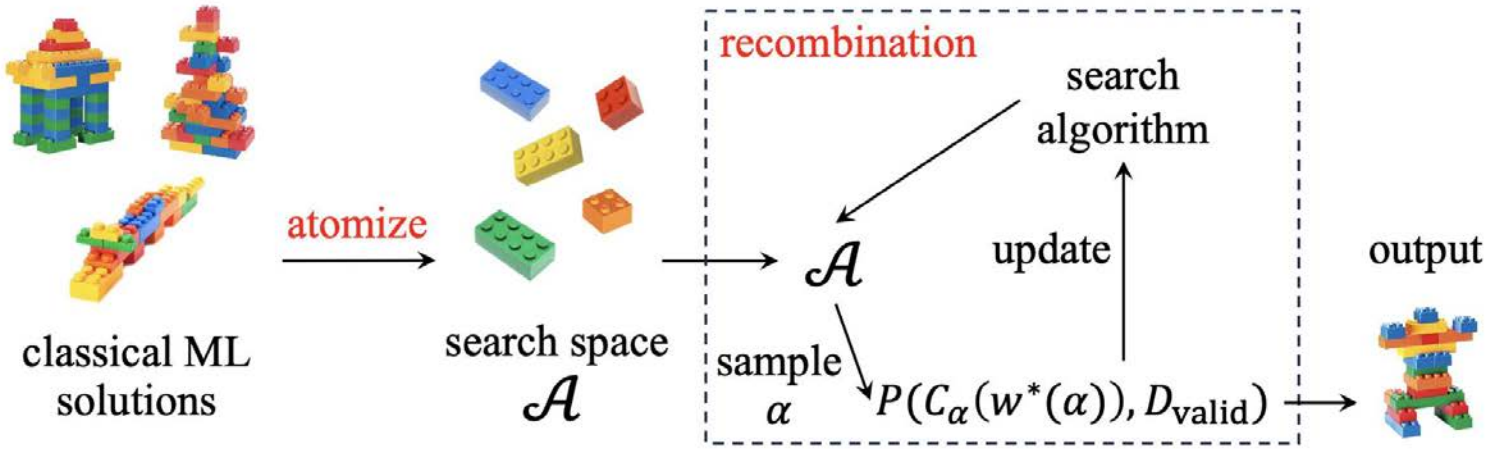
The definition of AutoML problems

- Define the AutoML problem as a bi-level optimization problem:

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} P(C_\alpha(w^*(\alpha)), \mathcal{D}_{valid}),$$
$$s.t. \begin{cases} w^*(\alpha) = \arg\min_w L(C_\alpha(w), \mathcal{D}_{train}) \\ G(\alpha) \leq 0 \end{cases},$$

- Here α is learning configuration, A is search space
- w is model parameter, C is the learning method
- D_{train}/D_{valid} is the train/validation dataset
- L is loss function, G is the constrain of the problem

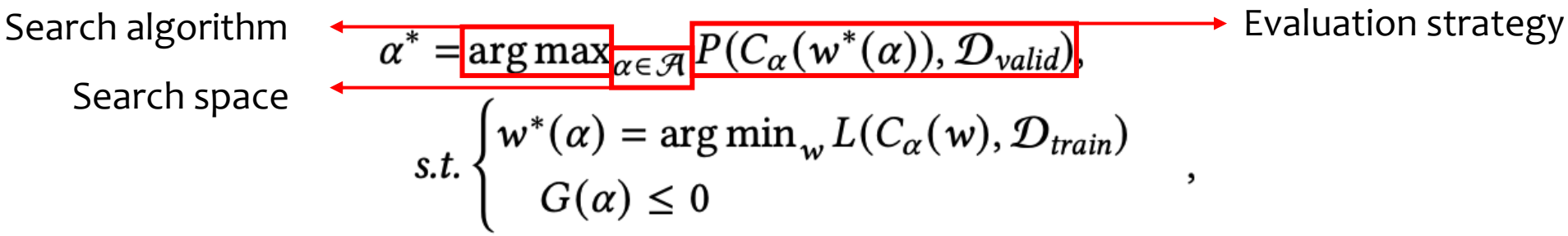
General learning strategy



- Key words: atomization and recombination
  - Atomizes the classical machine learning solutions to form the search space
  - Conducts the recombination step to obtain new learning method C
  - Evaluation the performance of C correspond to alpha

Three key components

- Corresponding place in definition:



- Three core issues:
  - how to design a good **search space**?
  - how to design an effective search **algorithm**?
  - how to evaluate model performance **efficiently**?

Outline

- What is AutoML
- How to do AutoML
  - Problem definition and general learning strategy
  - Search space
  - Search algorithm
  - Evaluation strategy
- Application examples of AutoML
- Theory insights

Search space

- The determination of the search space A.
- Trade-off to consider:
  - Expressive enough to achieve good generalization
  - Compact enough to alleviate the burden for search
- Search space categories:
  - General space
  - Structured space
  - Transformed space

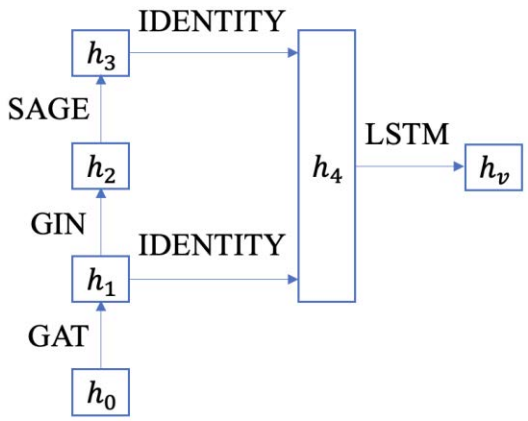
General space

- Search space defined based on requirements in specific machine learning problems
- Example: hyper-parameter setting for a specific model

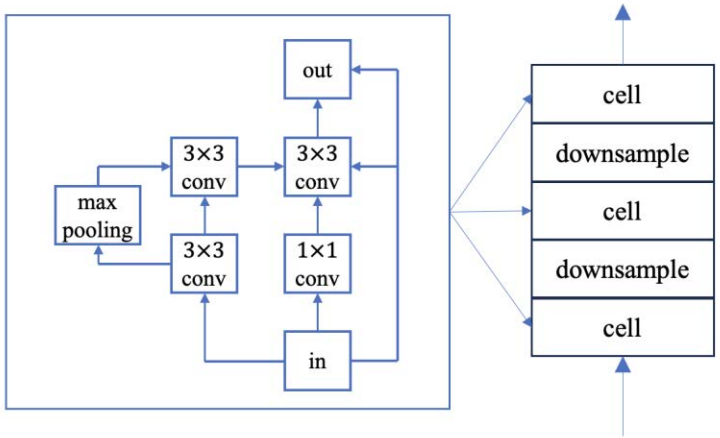
Hyper-parameter	Range
Learning rate	[0.1,0.01,0.001,0.0001]
Momentum	[0.99,0.97,0.95,0.90,0.80]
Dropout rate	[0,0.1,0.2,0.3,0.4,0.5]
Optimizer	[Adam,SGD,AdamW,Rmsprop]
Hidden dimension	[50,100,150,200,250,300]

Structured space

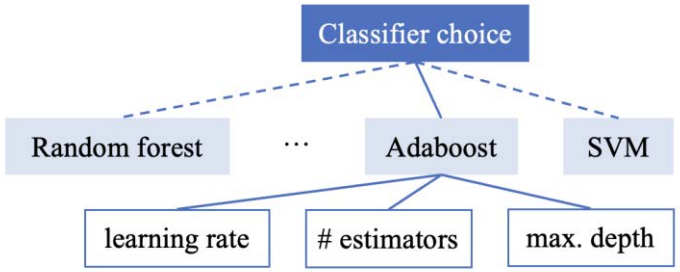
- Structure the initial space to a more condense search space



Directed acyclic graph



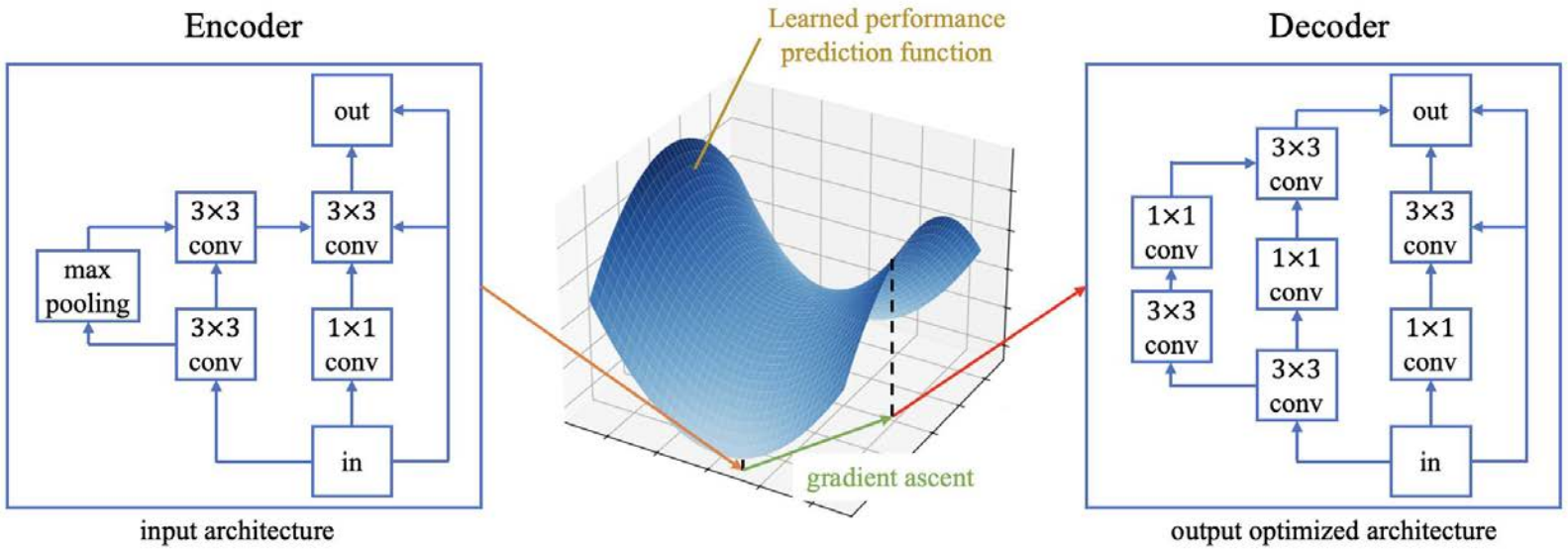
Cell



Hierarchical search space

Transformed space

- Introduce extra function to transform the search space A into a new space with better property (e.g. continuous, differentiable, etc.)
- Example: transformed space by autoencoder:



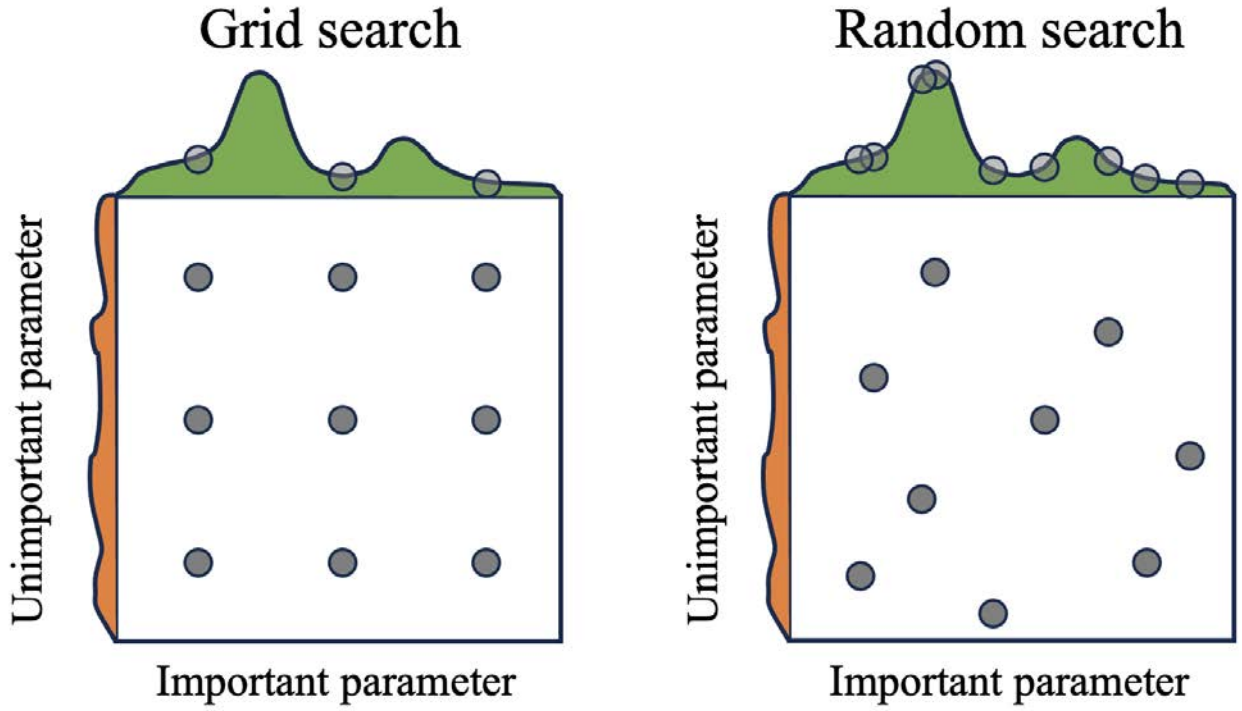
Outline

- What is AutoML
- How to do AutoML
  - Problem definition and general learning strategy
  - Search space
  - Search algorithm
  - Evaluation strategy
- Example 1: tuning knowledge graph (KG) learning hyper-parameters
- Example 2: Neural Architecture Search (NAS)
- Example 3: AutoML for foundation models

Search algorithm

- The search algorithm determines how to do recombination and finds a good α from search space A
- Trade-off to consider:
  - Conduct less iterations to be faster
  - Have precise search result to reduce optimization error
- Search algorithm categories
  - Grid search and random search
  - Evolutionary algorithm
  - Gradient-based method
  - Bayesian optimization method
  - Reinforcement learning

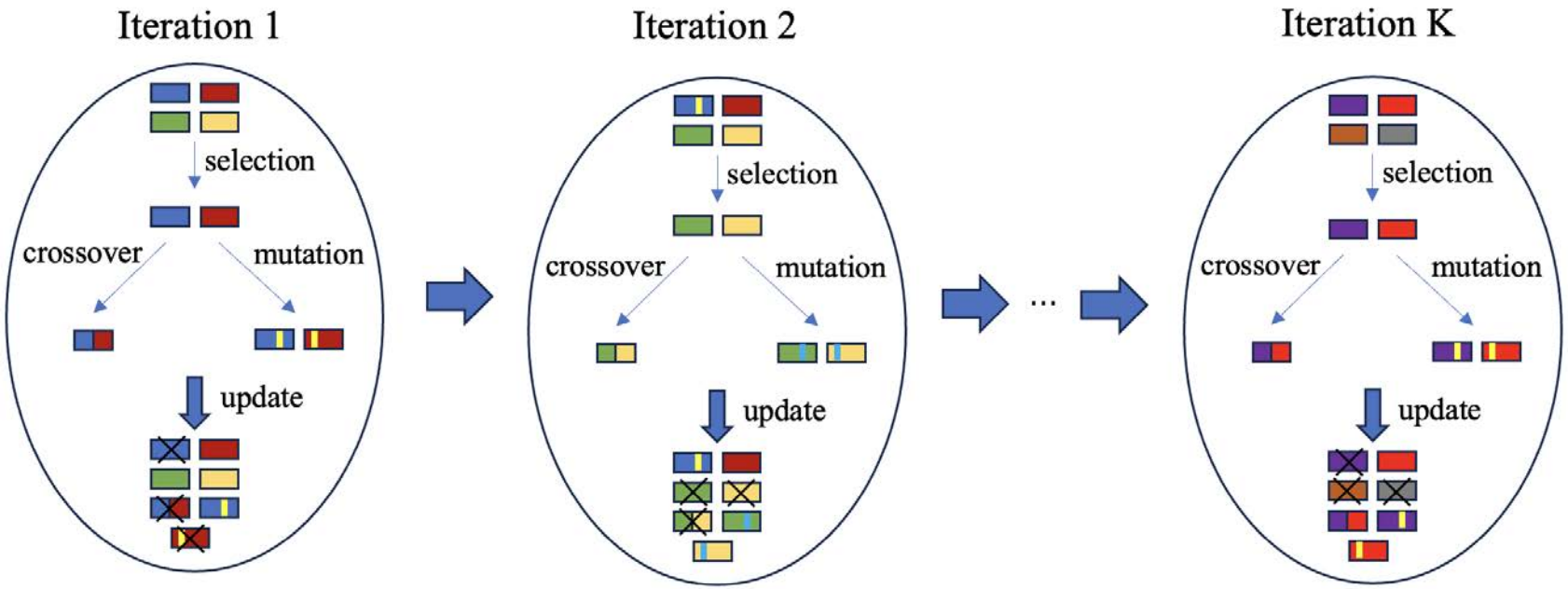
Grid search and random search



- Random search works better than grid search when some learning configurations are much more important than others
- Drawback of two methods: time consuming

Evolutionary algorithm

- Contain several steps: selection, mutation, crossover, update
- Population based, maintain a set of programs and improve the performance



Gradient-based method

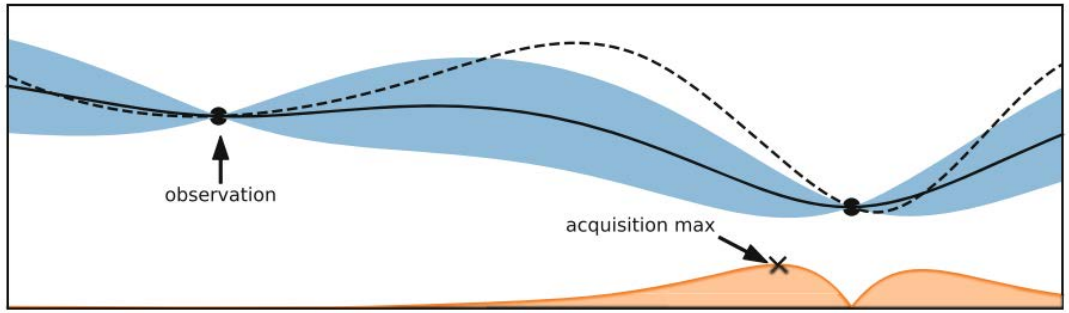
- Theoretically, the gradient of the performance w.r.t. learning configurations can be calculated:

$$\frac{P(C_\alpha(w^*), \mathcal{D}_{\text{valid}})}{\partial \alpha} = \frac{\partial P(C_\alpha(w^*), \mathcal{D}_{\text{valid}})}{\partial w^*} \frac{\partial w^*}{\partial \alpha} + \frac{\partial P(C_\alpha(w^*), \mathcal{D}_{\text{valid}})}{\partial \alpha},$$

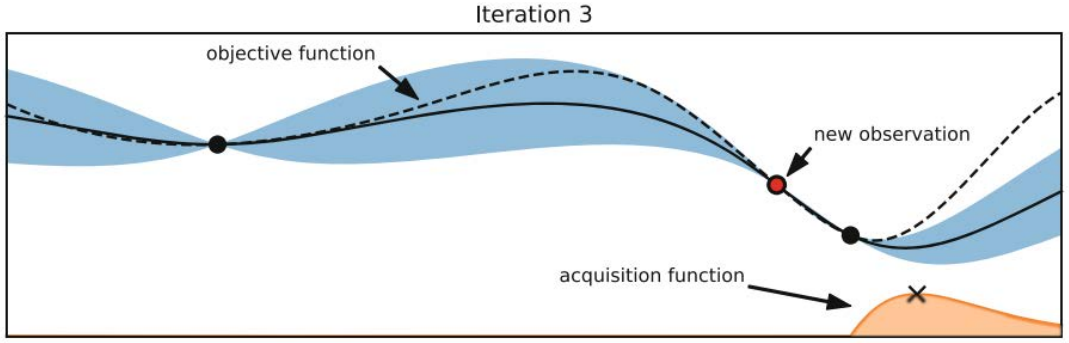
$$\text{where} \quad \frac{\partial w^*}{\partial \alpha} = -\left(\frac{\partial^2 L(C_\alpha(w^*), \mathcal{D}_{\text{train}})}{\partial w^{*2}}\right)^{-1} \frac{\partial^2 L(C_\alpha(w^*), \mathcal{D}_{\text{train}})}{\partial w^* \partial \alpha}.$$

- In practice, the calculation of the gradient usually involves different kinds of approximation

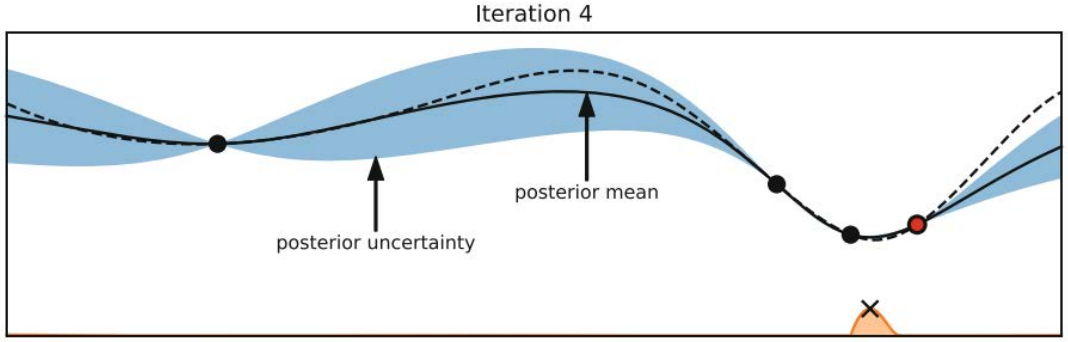
Exemplar Algorithm – Bayesian Optimization



a probabilistic surrogate model estimate M



acquisition function to decide which point to evaluate next

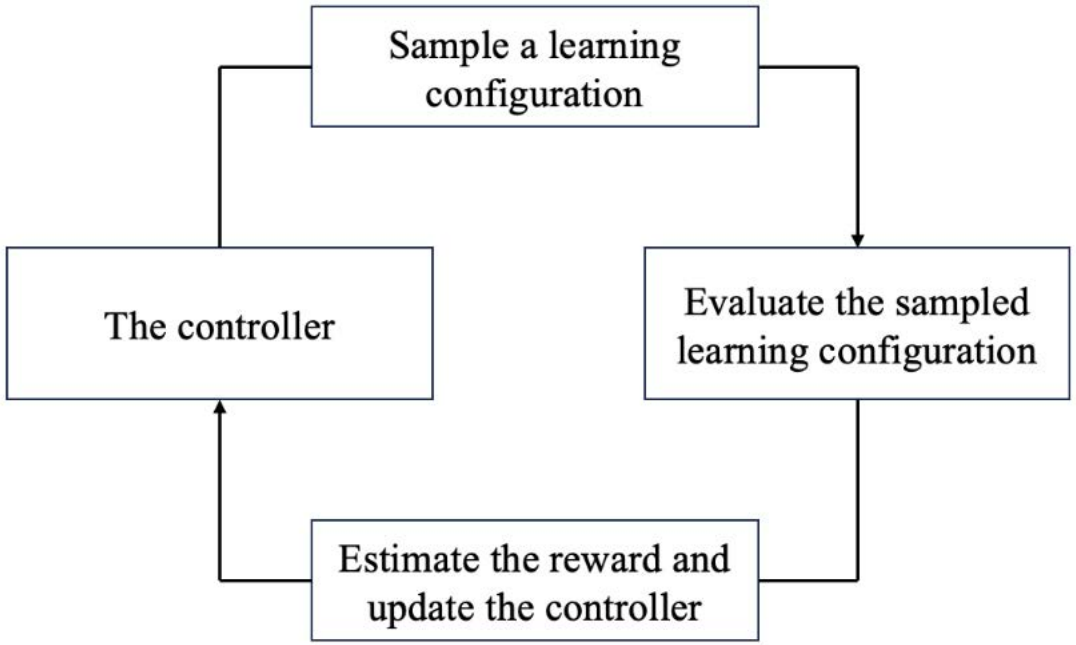


Gaussian process is typically used as surrogate model, good at dealing with smooth M.

Practical bayesian optimization of machine learning algorithms. NIPS 2012

Reinforcement learning

- Consider the AutoML problem as reinforcement learning problem
  - Trail and error process based on model evaluation results.



Outline

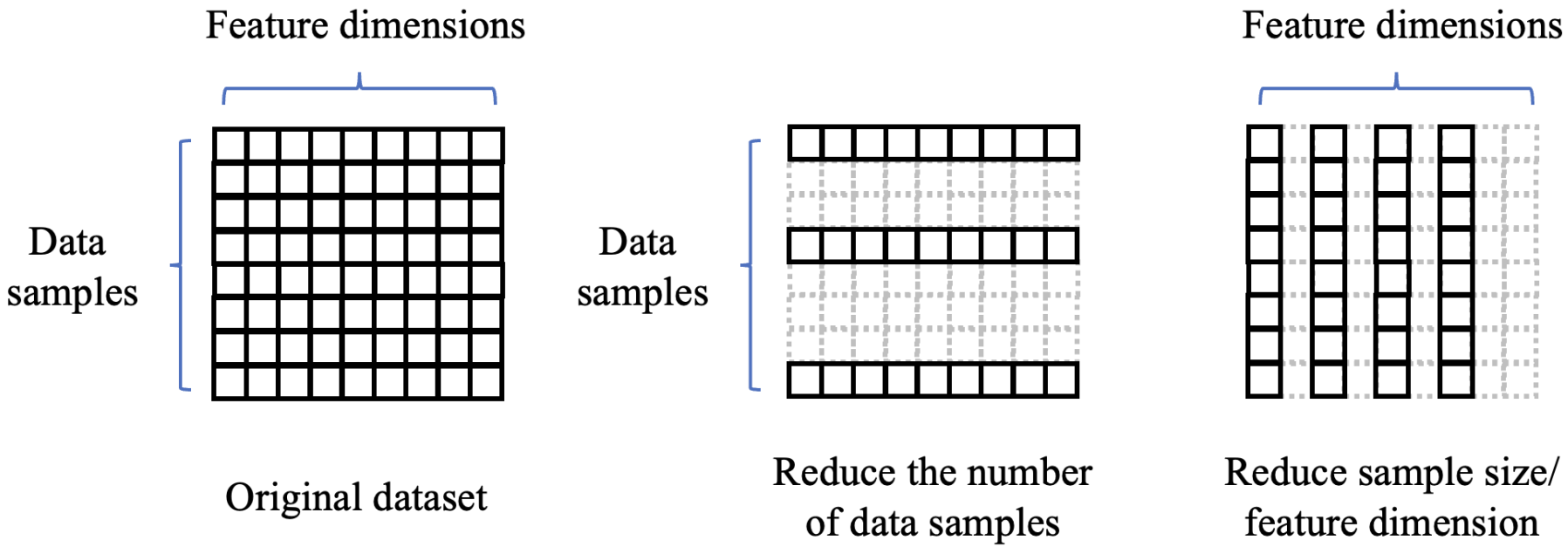
- What is AutoML
- How to do AutoML
  - Problem definition and general learning strategy
  - Search space
  - Search algorithm
  - Evaluation strategy
- Application examples of AutoML
- Theory insights

Evaluation strategy

- The model parameter for lower level objective should be trained for configuration evaluation.
- Trade-off to consider:
  - Most accurate evaluation: train the model from scratch to convergence, but too time-consuming
  - Fast evaluation methods are able to return the performance in a short time, but the evaluation result is usually inaccurate.
- Evaluation strategy categories
  - Reducing dataset quantity
  - Monitoring learning curves
  - Parameter reusing
  - Performance predictor

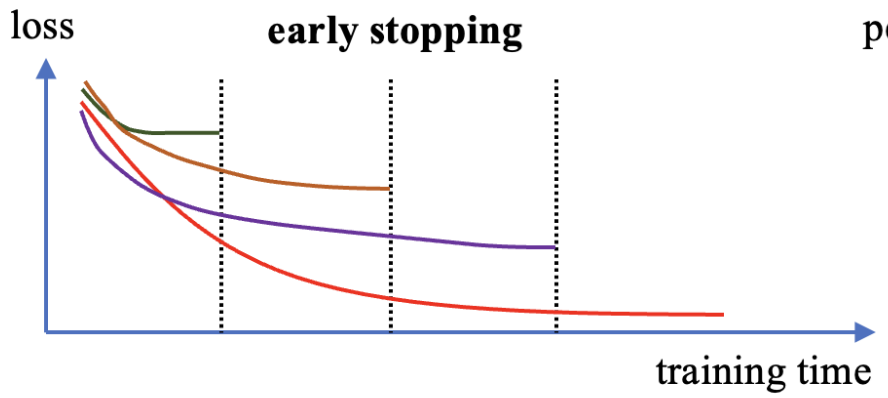
Reducing dataset quantity

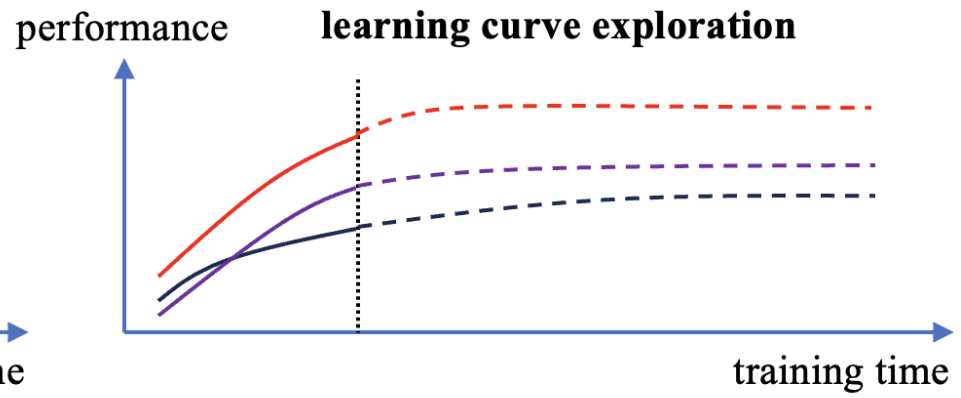
- Reducing the number of data samples
- Reducing sample size



Monitoring learning curves

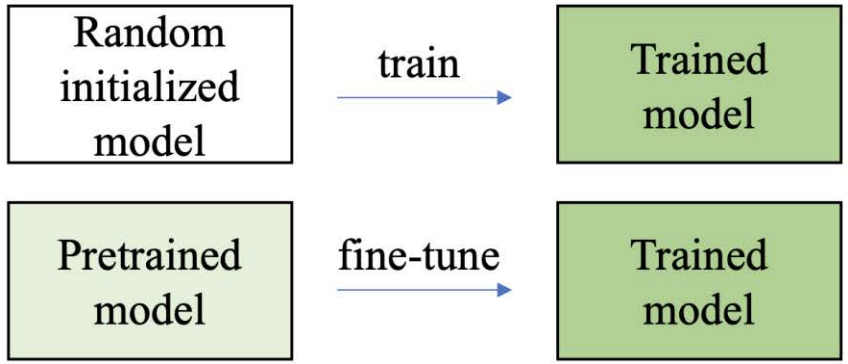
- Early stopping: terminates the training process in advance when performance in early training stage is poor or the performance does not get improved for a certain period
- Learning curve exploration: predict the performance based on partial learning curves.



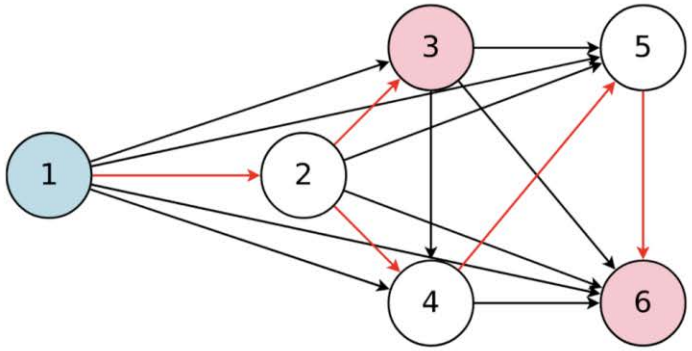


Parameter reusing

- Main intuition: give the model parameter a better initialization, so as to accelerate the evaluation process



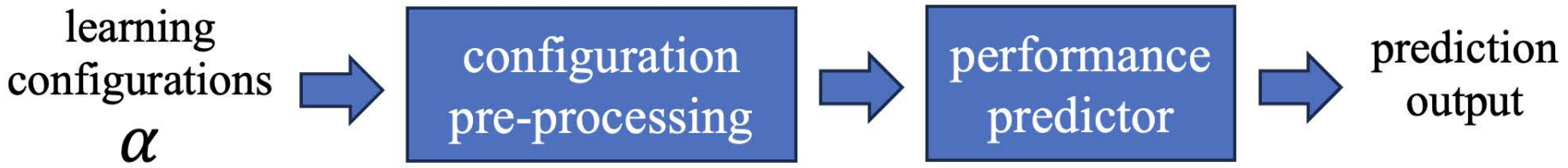
parameter warming up



supernet training
(red line for sampled model)

Performance predictor

- Learning configuration pre-processing:
  - collect the feature of the learning configurations without conducting the full training procedure
- Performance prediction
  - Design a surrogate model to map learning configuration to their actual performance



Summary

- Formulate AutoML as a bi-level optimization problem

$$\alpha^* = \arg\max_{\alpha \in \mathcal{A}} P(C_\alpha(w^*(\alpha)), \mathcal{D}_{valid}),$$
$$s.t. \begin{cases} w^*(\alpha) = \arg\min_w L(C_\alpha(w), \mathcal{D}_{train}) \\ G(\alpha) \leq 0 \end{cases},$$

- Two key words:
  - Atomization, recombination
- Three components
  - Search space, Search algorithm, Evaluation strategy

Reference

[1] Algorithms for hyper-parameter optimization. NeurIPS 2011

[2] Random search for hyper-parameter optimization. JMLR 2012

[3] Efficient neural architecture search via parameters sharing. ICML 2018

[4] Progressive neural architecture search. ECCV 2018

[5] DARTS: Differentiable Architecture Search. ICLR 2018

[6] Neural architecture optimization. NeurIPS 2018

[7] Regularized evolution for image classifier architecture search. AAAI 2019

[8] Neural architecture search with reinforcement learning. ICLR 2017

[9] Hyperband: A novel bandit-based approach to hyperparameter optimization. JMLR 2017

[10] Zero-Cost Proxies for Lightweight NAS. ICLR 2020

Outline

- What is AutoML
- How to do AutoML
- Application examples of AutoML
  - Example 1: AutoML for Hyper-Parameter Optimization (HPO)
  - Example 2: AutoML for Neural Architecture Search (NAS)
  - Example 3: AutoML for foundation models
- Theory insights

Tuning Knowledge Graph Learning Hyperparameters

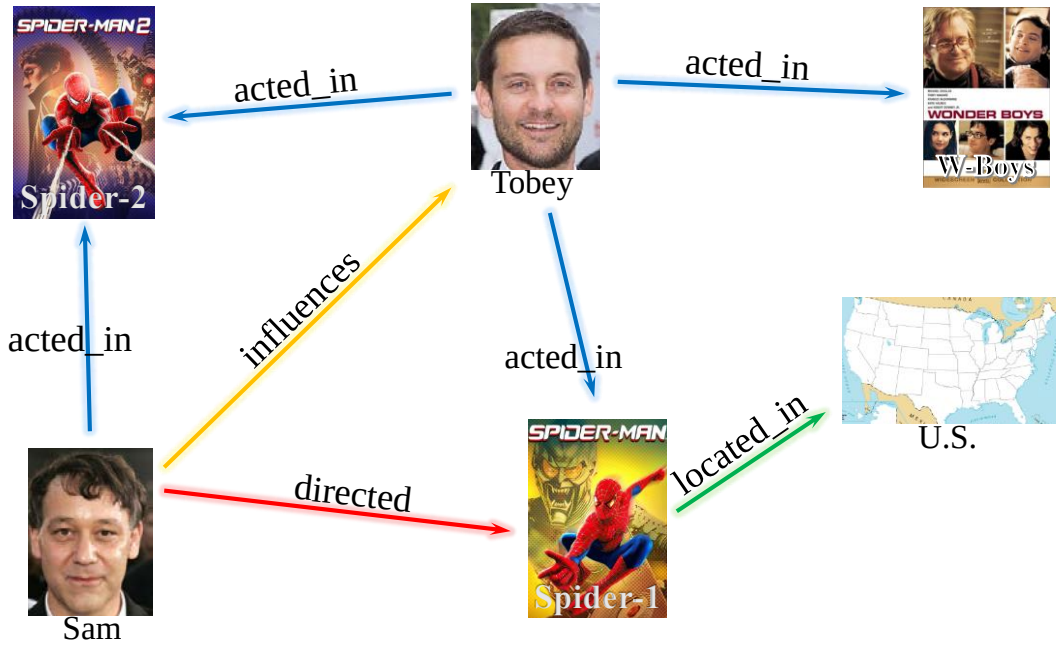
What is knowledge graph (KG) ?

Graph representation: $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{S})$.

Entities $\mathcal{E}$: real world objects or abstract concepts.

Relations $\mathcal{R}$: interactions between/among entities.

Fact/triples $\mathcal{S}$: the basic unit in form of (head entity, relation, tail entity), (h, r, t).



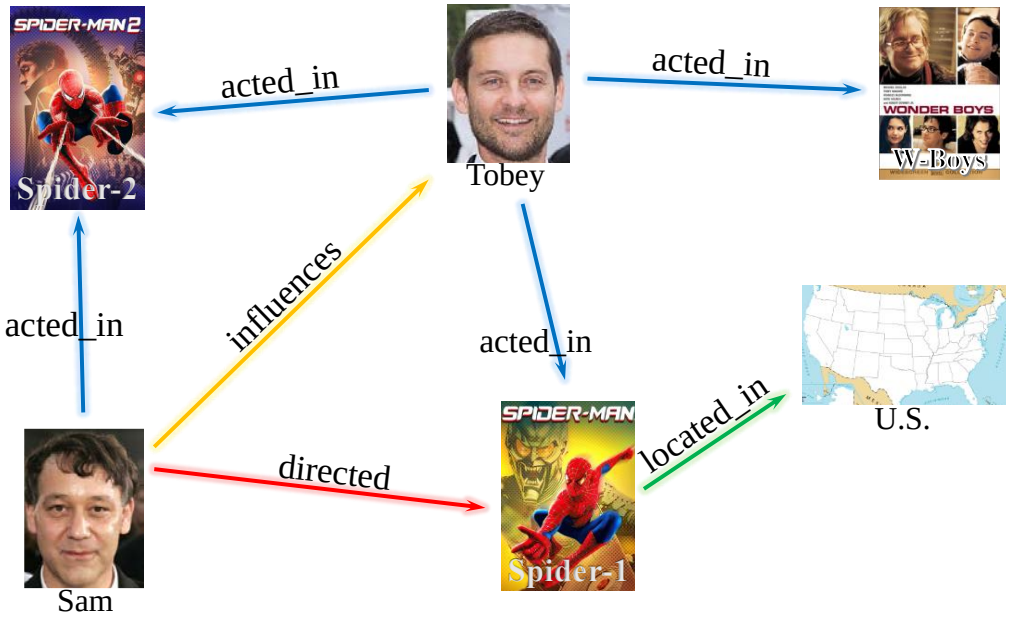
Google Knowledge Graph

Freebase

WIKIDATA

BIO2RDF

Representing learning in KG

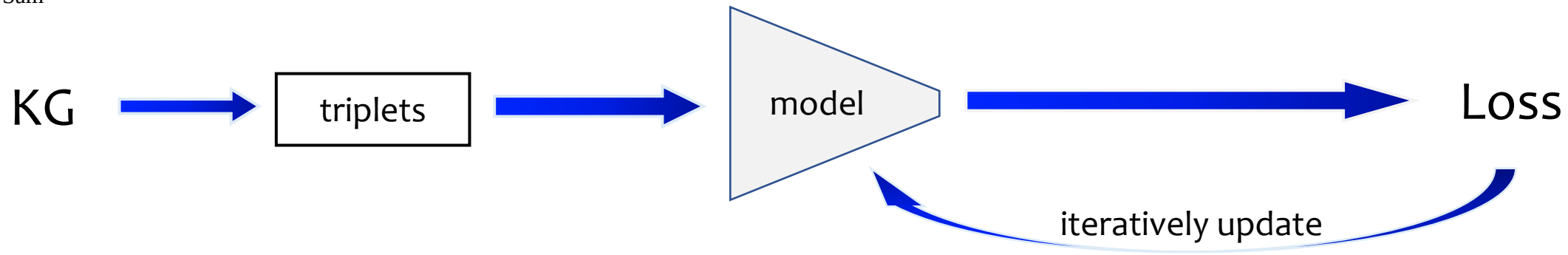


Observed triplets S^+:
maximize score

(Sam, directed, Spider-1)
(Tobey, acted_in, Spider-2)
(Spider-1, located_in, U.S.)
…

Unobserved triplets
S^-: minimize score

(Tobey, directed, Spider-1)
(Tobey, acted_in, U.S.)
(Spider-1, directed, U.S.)
…



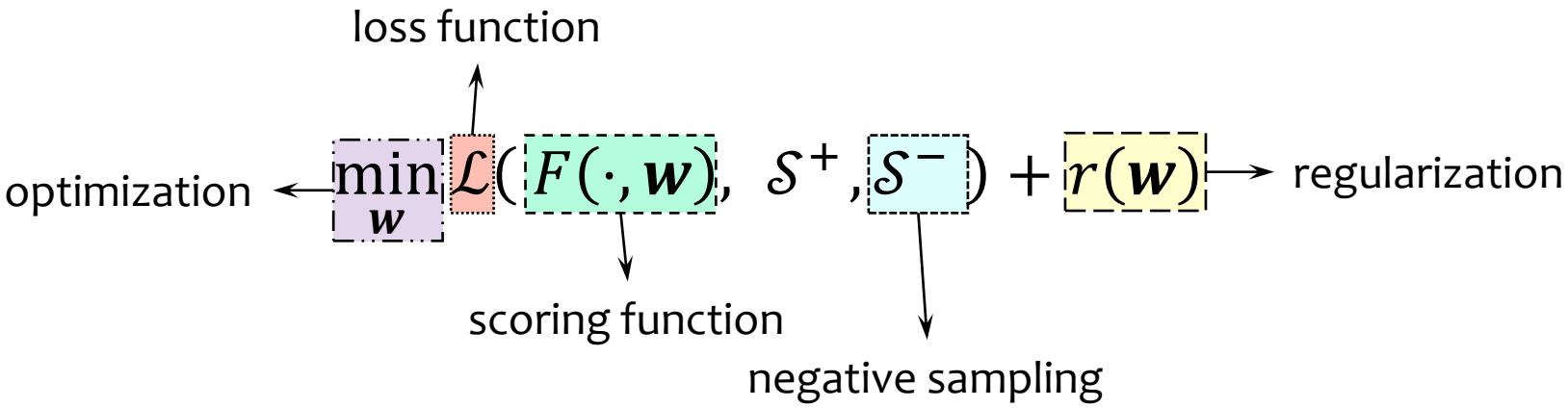
➢Objective: Preserve as much information on original graph as possible.

Recall: Common Learning Objective

For setting up a knowledge graph embedding system, we need

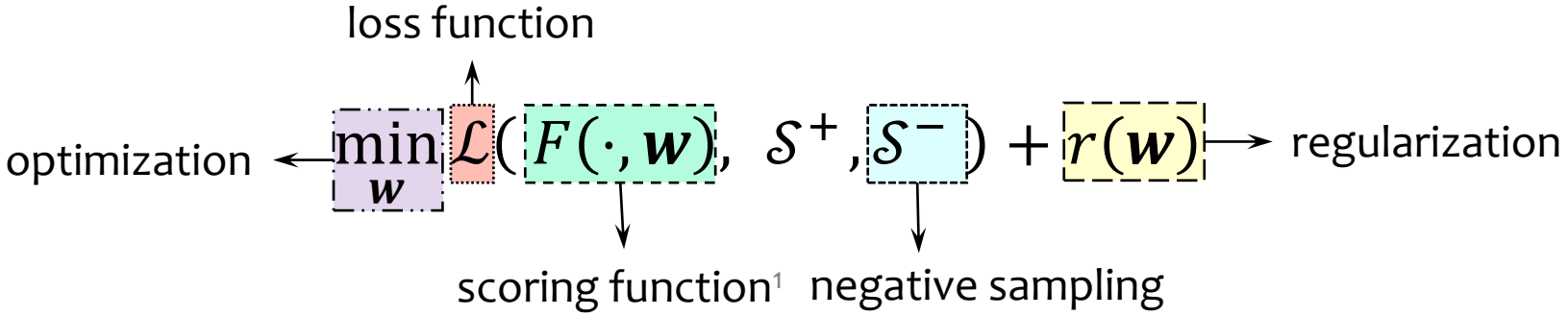
- A **scoring function F** to discriminate positive/negative triples
- A sampling scheme to generate **negative samples S^-**
- A **loss function L** and **regularization r** to define the learning problem
- An **optimization** strategy for convergence procedure

We can formulate the learning framework as:



Common Hyper-parameters

KGE components and related hyper-parameters (HPs)



component	name	type	range
negative sampling	# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	loss function	cat	{MR, BCE_(mean, sum, adv), CE}
	gamma (MR)	float	[1, 24]
	adv. weight (BCE_adv)	float	[0.5, 2.0]
regularization	regularizer	cat	{FRO, NUC, DURA, None}
	reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
	dropout rate	float	[0, 0.5]
optimization	optimizer	cat	{Adam, Adagrad, SGD}
	learning rate	float	$[10^{-5}, 10^{0}]$
	initializer	cat	{uniform, normal, xavier_uniform, xavier_norm}
	batch size	int	{128, 256, 512, 1024}
	dimension size	int	{100, 200, 500, 1000, 2000}
	inverse relation	bool	{True, False}

1: Note that HPs in SF are not covered here

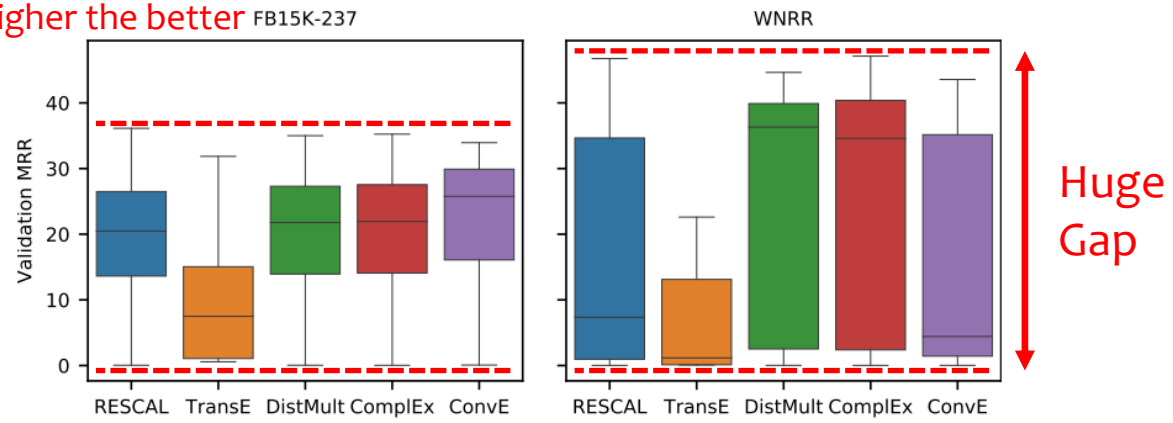
A best model or HPs for all problems ?

		FB15k				WN18				FB15k-237				WN18RR				YAGO3-10			
		H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR	H@1	H@10	MR	MRR
Tensor Decomposition Models	DistMult	73.61	86.32	173	0.784	72.60	94.61	675	0.824	22.44	49.01	199	0.313	39.68	50.22	5913	0.433	41.26	66.12	1107	0.501
	ComplEx	81.56	90.53	34	0.848	94.53	95.50	3623	0.949	25.72	52.97	202	0.349	42.55	52.12	4907	0.458	50.48	70.35	1112	0.576
	ANALOGY	65.59	83.74	126	0.726	92.61	94.42	808	0.934	12.59	35.38	476	0.202	35.82	38.00	9266	0.366	19.21	45.65	2423	0.283
	SimplE	66.13	83.63	138	0.726	93.25	94.58	759	0.938	10.03	34.35	651	0.179	38.27	42.65	8764	0.398	35.76	63.16	2849	0.453
	HolE	75.85	86.78	211	0.800	93.11	94.94	650	0.938	21.37	47.64	186	0.303	40.28	48.79	8401	0.432	41.84	65.19	6489	0.502
	TuckER	72.89	88.88	39	0.788	94.64	95.80	510	0.951	25.90	53.61	162	0.352	42.95	51.40	6239	0.459	46.56	68.09	2417	0.544
Geometric Models	TransE	49.36	84.73	45	0.628	40.56	94.87	279	0.646	21.72	49.65	209	0.31	2.79	49.52	3936	0.206	40.57	67.39	1187	0.501
	STransE	39.77	79.60	69	0.543	43.12	93.45	208	0.656	22.48	49.56	357	0.315	10.13	42.21	5172	0.226	3.28	7.35	5797	0.049
	CrossE	60.08	86.23	136	0.702	73.28	95.03	441	0.834	21.21	47.05	227	0.298	38.07	44.99	5212	0.405	33.09	65.45	3839	0.446
	TorusE	68.85	83.98	143	0.746	94.33	95.44	525	0.947	19.62	44.71	211	0.281	42.68	53.35	4873	0.463	27.43	47.44	19455	0.342
	RotatE	73.93	88.10	42	0.791	94.30	96.02	274	0.949	23.83	53.06	178	0.336	42.60	57.35	3318	0.475	40.52	67.07	1827	0.498
Deep Learning Models	ConvE	59.46	84.94	51	0.688	93.89	95.68	413	0.945	21.90	47.62	281	0.305	38.99	50.75	4944	0.427	39.93	65.75	2429	0.488
	ConvKB	11.44	40.83	324	0.211	52.89	94.89	202	0.709	13.98	41.46	309	0.230	5.63	52.50	3429	0.249	32.16	60.47	1683	0.420
	ConvR	70.57	88.55	70	0.773	94.56	95.85	471	0.950	25.56	52.63	251	0.346	43.73	52.68	5646	0.467	44.62	67.33	2582	0.527
	CapsE	1.93	21.78	610	0.087	84.55	95.08	233	0.890	7.34	35.60	405	0.160	33.69	55.98	720	0.415	0.00	0.00	60676	0.000
	RSN	72.34	87.01	51	0.777	91.23	95.10	346	0.928	19.84	44.44	248	0.280	34.59	48.34	4210	0.395	42.65	66.43	1339	0.511
	AnyBURL	81.09	87.86	288	0.835	94.63	95.96	233	0.951	24.03	48.93	480	0.324	44.93	55.97	2530	0.485	45.83	66.07	815	0.528

No best models

		RESCAL	TransE	DistMult	ComplEx	ConvE
FB15K-237	Valid. MRR	*36.1*	*31.5*	*35.0*	*35.3*	*34.3*
	Emb. size	128 *(-0.5)*	512 *(-3.7)*	256 *(-0.2)*	256 *(-0.3)*	256 *(-0.4)*
	Batch size	512 *(-0.5)*	128 *(-7.1)*	1024 *(-0.2)*	1024 *(-0.3)*	1024 *(-0.4)*
	Train type	1vsAll *(-0.8)*	NegSamp –	NegSamp *(-0.2)*	NegSamp *(-0.3)*	1vsAll *(-0.4)*
	Loss	CE *(-0.9)*	CE *(-7.1)*	CE *(-3.1)*	CE *(-3.8)*	CE *(-0.4)*
	Optimizer	Adam *(-0.5)*	Adagrad *(-3.7)*	Adagrad *(-0.2)*	Adagrad *(-0.5)*	Adagrad *(-1.5)*
	Initializer	Normal *(-0.8)*	XvNorm *(-3.7)*	Unif. *(-0.2)*	Unif. *(-0.5)*	XvNorm *(-0.4)*
	Regularizer	None *(-0.5)*	L2 *(-3.7)*	L3 *(-0.2)*	L3 *(-0.3)*	L3 *(-0.4)*
	Reciprocal	No *(-0.5)*	Yes *(-9.5)*	Yes *(-0.3)*	Yes *(-0.3)*	Yes –
WNRR	Valid. MRR	*46.8*	*22.6*	*45.4*	*47.6*	*44.3*
	Emb. size	128 *(-1.0)*	512 *(-5.1)*	512 *(-1.1)*	128 *(-1.0)*	512 *(-1.2)*
	Batch size	128 *(-1.0)*	128 *(-5.1)*	1024 *(-1.1)*	512 *(-1.0)*	1024 *(-1.3)*
	Train type	KvsAll *(-1.0)*	NegSamp –	KvsAll *(-1.1)*	1vsAll *(-1.0)*	KvsAll *(-1.2)*
	Loss	CE *(-2.0)*	CE *(-5.1)*	CE *(-2.4)*	CE *(-3.5)*	CE *(-1.4)*
	Optimizer	Adam *(-1.2)*	Adagrad *(-5.8)*	Adagrad *(-1.5)*	Adagrad *(-1.5)*	Adam *(-1.4)*
	Initializer	Unif. *(-1.0)*	XvNorm *(-5.1)*	Unif. *(-1.3)*	Unif. *(-1.5)*	XvNorm *(-1.4)*
	Regularizer	L3 *(-1.2)*	L2 *(-5.1)*	L3 *(-1.1)*	L2 *(-1.0)*	L1 *(-1.2)*
	Reciprocal	Yes *(-1.0)*	Yes *(-5.9)*	Yes *(-1.1)*	No *(-1.0)*	Yes –

The higher the better



No best hyper-parameters

solution: search for best hyper-parameter for specific problem 🤔

Problem Definition

HP searching problem setup

Definition 1 (Hyper-parameter search for KG embedding). *The problem of HP search for KG embedding model is formulated as*

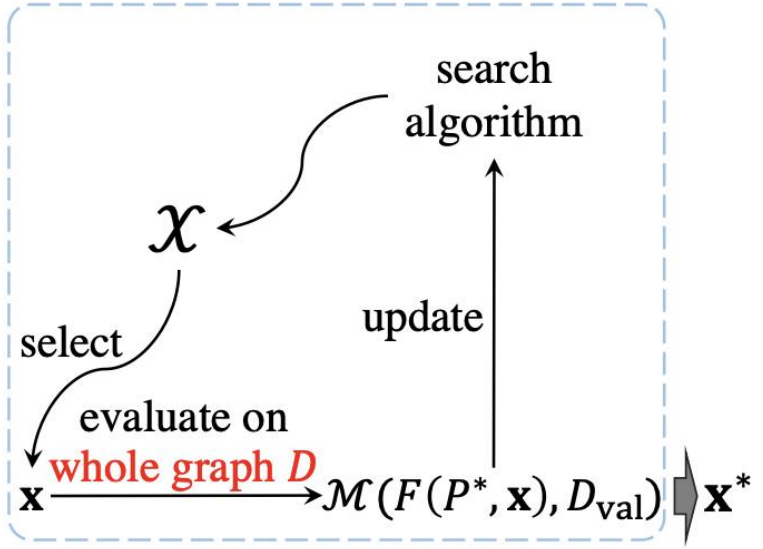
$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathcal{M}\big(F(\boldsymbol{P}^*, \mathbf{x}), D_{val}\big), \quad (2)$$

$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}} \mathcal{L}\big(F(\boldsymbol{P}, \mathbf{x}), D_{tra}\big). \quad (3)$$

Performance measurement

Model parameters

HP configuration



Three major aspects for efficiency in Def. 1

the **size** of search space χ	make the space smaller	**Search space**
the validation **curvature** of $\mathcal{M}$	capture the curvature better	**Search algorithm**
the evaluation **cost** in solving $argmin_{\mathcal{P}}$	obtain cheap and accurate evaluations	**Evaluation strategy**

Understand HP in KG Learning

Search space

name	type	range
# negative samples	cat	{32, 128, 512, 2048, 1VsAll, kVsAll}
loss function	cat	{MR, BCE_(mean, sum, adv), CE}
gamma (MR)	float	[1, 24]
adv. weight (BCE_adv)	float	[0.5, 2.0]
regularizer	cat	{FRO, NUC, DURA, None}
reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
dropout rate	float	$[0, 0.5]$
optimizer	cat	{Adam, Adagrad, SGD}
learning rate	float	$[10^{-5}, 10^{0}]$
initializer	cat	{uniform, normal, xavier_uniform, xavier_norm}
batch size	int	{128, 256, 512, 1024}
dimension size	int	{100, 200, 500, 1000, 2000}
inverse relation	bool	{True, False}

Three major aspects for efficiency in problem definition:

1. the size of search space χ
2. the validation curvature of $\mathcal{M}$
3. the evaluation cost in solving $argmin_{\mathcal{P}}$

Questions to be answered

- What are the properties of each HP?
  - ranking distribution
  - consistency
  - computing cost
- Can we decrease the range for each HP?
- Can we decouple some HPs?

Design Space for Graph Neural Networks. NIPS 2020

Understand HP in KG Learning

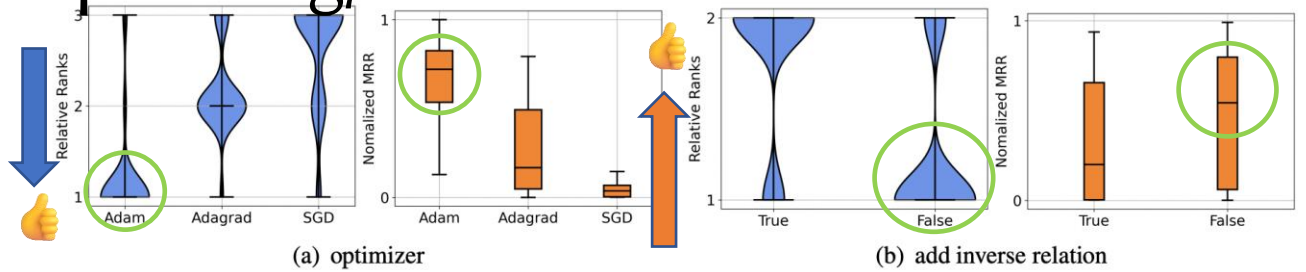
Excavating properties of HPs | Ranking/Performance distribution

Core: control variates

The HPs can be classified into 4 groups

1. fixed choice
2. limited range
3. monotonously related
4. no obvious patterns

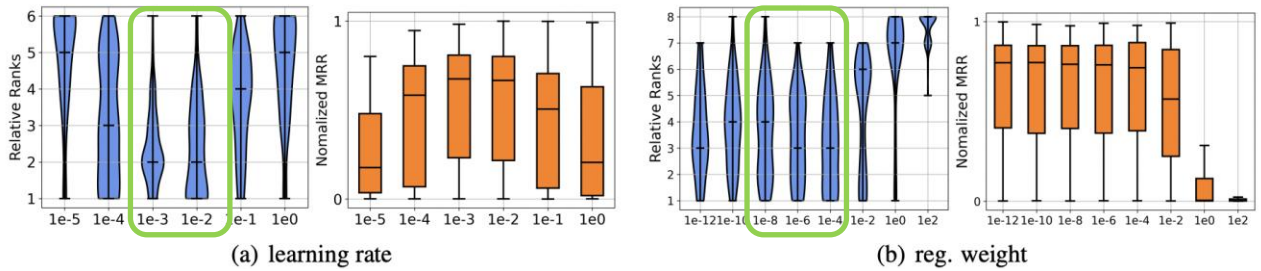
1.



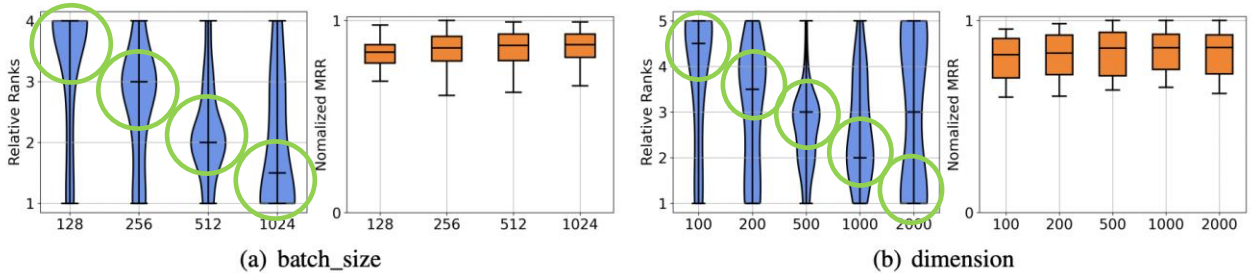
✗ lower rankings

✓ higher rankings

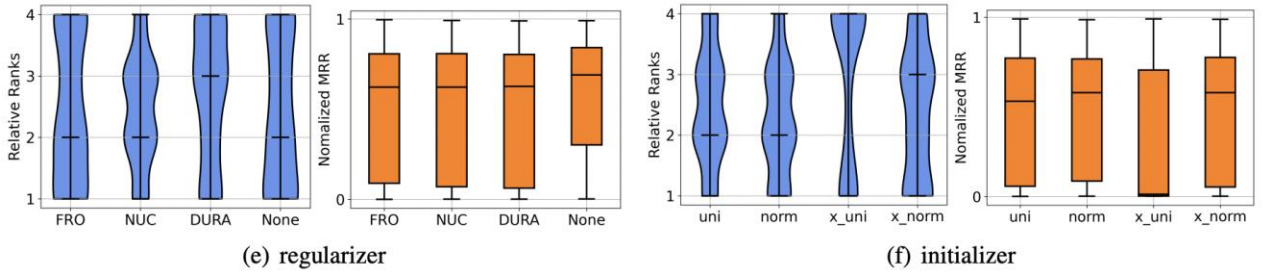
2.



3.



4.



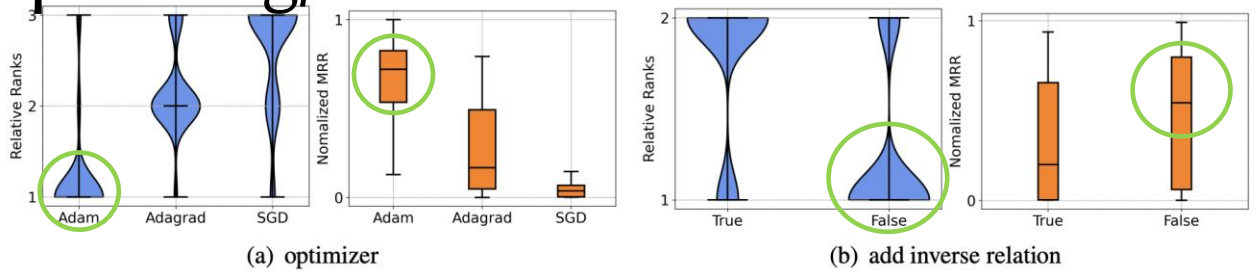
Understand HP in KG Learning

Excavating properties of HPs | Ranking/Performance distribution

The HPs can be classified into 4 groups

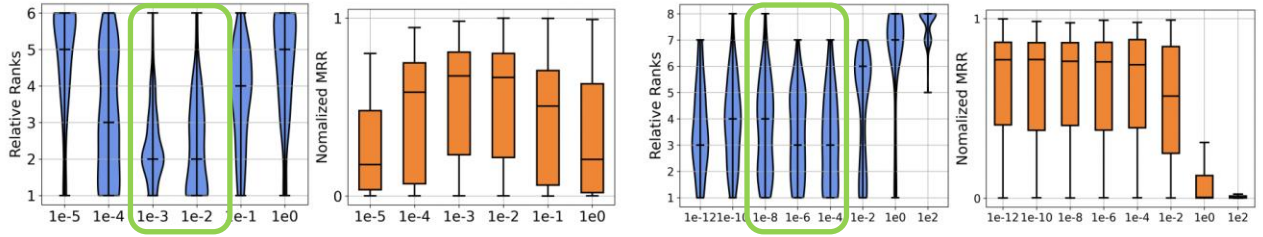
1. fixed choice
2. limited range
3. monotonously related
4. no obvious patterns

1.



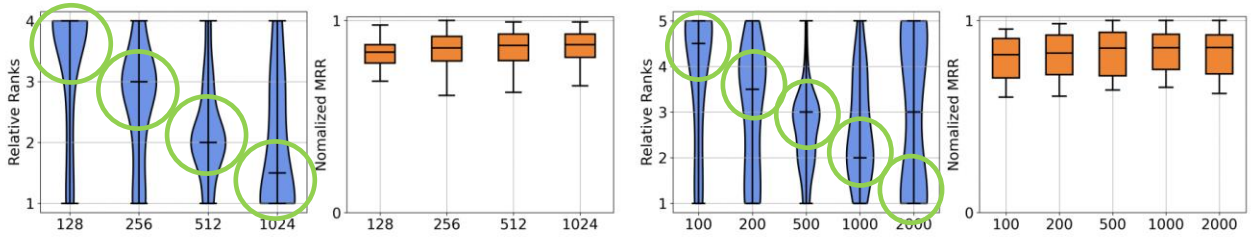
(a) optimizer (b) add inverse relation

2.



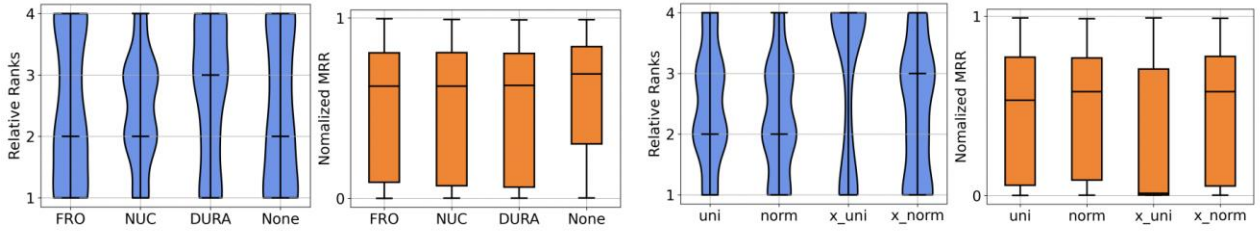
(a) learning rate (b) reg. weight

3.



(a) batch_size (b) dimension

4.



(e) regularizer (f) initializer

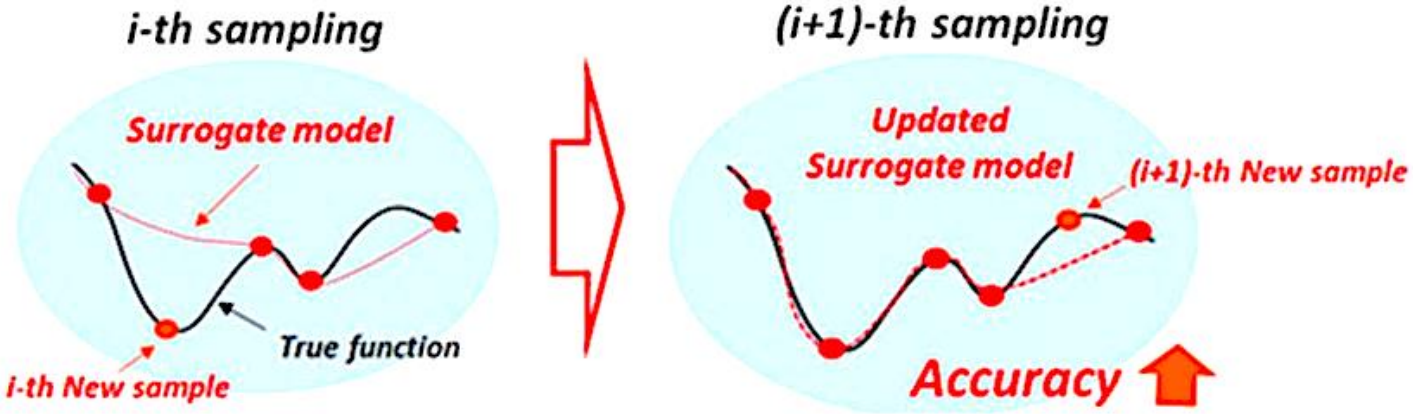
✗ lower rankings

✓ higher rankings

Understand HP in KG Learning

Search algorithm | from validation curvature

Surrogate: estimate the ground-truth function with given observations



Three major aspects for efficiency in Def. 1

1. the size of search space χ
2. the validation curvature of $\mathcal{M}$
3. the evaluation cost in solving $argmin_{\mathcal{P}}$

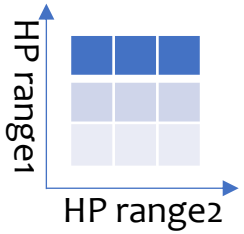
Requirements

- Accurate approximation to the ground-truth
- Easy to be updated with new observations

What can be a bad surrogate?

Understand HP in KG Learning

Search algorithm | from the aspect of predictor



Predictors

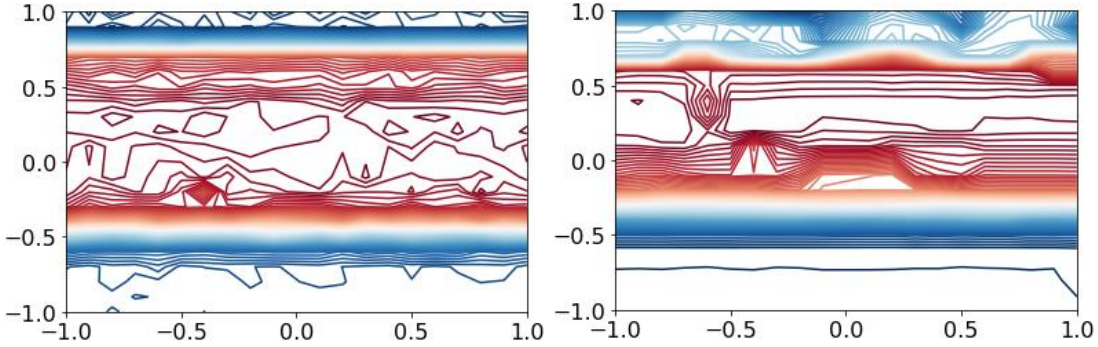
- Gaussian process (GP)
- Multi layer perceptron (MLP)
- Random forest (RF)

Observation:
RF is better in approximating the curvature

# train configurations	10	20	30
GP	0.0693±0.02	0.029±0.01	0.019±0.01
MLP	2.121±0.4	2.052±0.3	0.584±0.1
RF	**0.003±0.002**	**0.002±0.001**	**0.001±0.001**

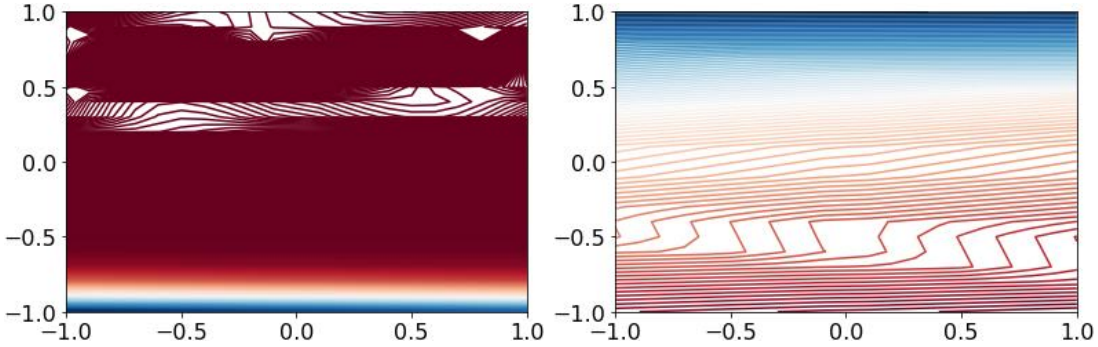
MSE results

Validation curvature



Ground truth

RF prediction



GP prediction

MLP prediction

Understand HP in KG Learning

Evaluation strategy | Time cost

Three major aspects for efficiency in Def. 1
1. the size of search space χ
2. the validation curvature of $\mathcal{M}$
3. the evaluation cost in solving $argmin_{\mathcal{P}}$

dataset	#entity	#relation	#train	#validate	#test
WN18RR (Dettmers et al., 2017)	41k	11	87k	3k	3k
FB15k-237 (Toutanova and Chen, 2015)	15k	237	272k	18k	20k
ogbl-biokg (Hu et al., 2020)	94k	51	4,763k	163k	163k
ogbl-wikikg2 (Hu et al., 2020)	2,500k	535	16,109k	429k	598k

Average evaluation time cost:

~2.1h
~3.5h
~17.3h
~21.7h

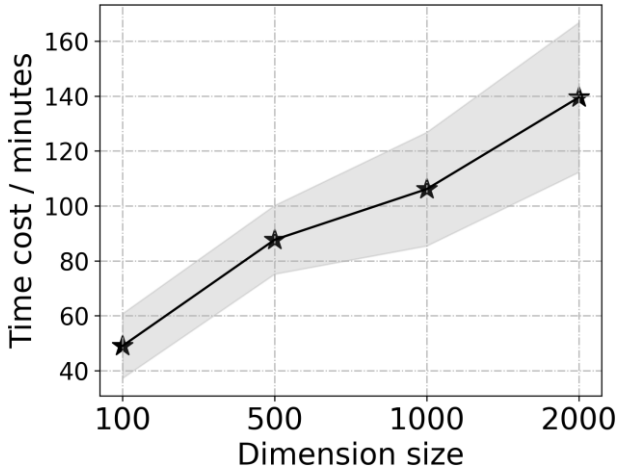
Expensive

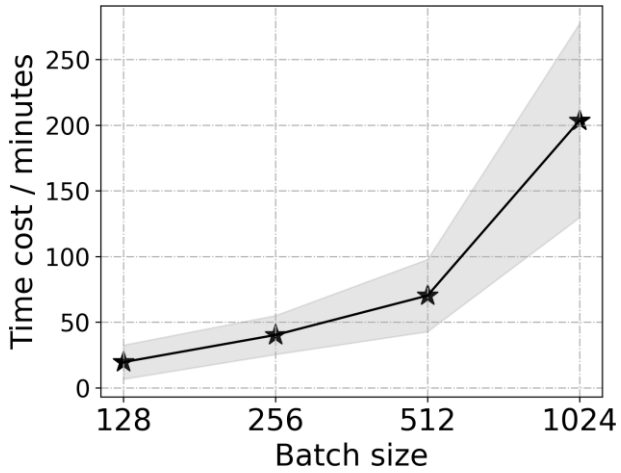
How to reduce the evaluation cost?

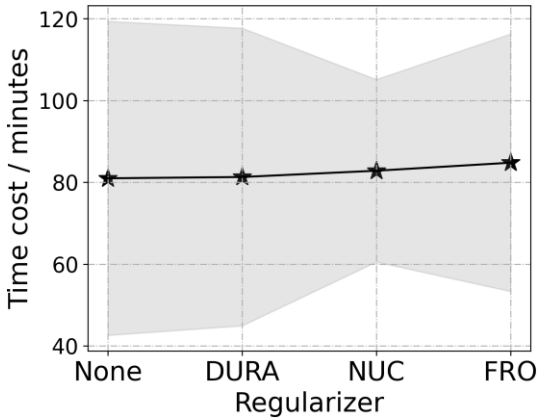
Understand HP in KG Learning

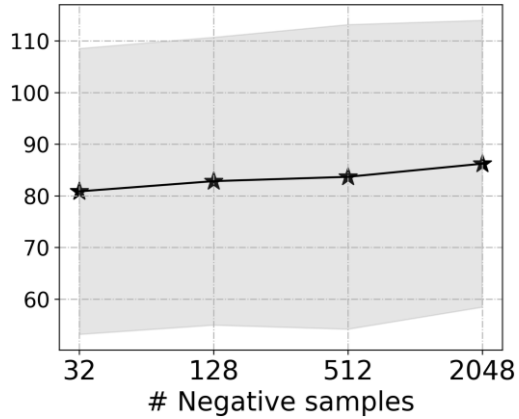
Evaluation strategy | Time cost[1]

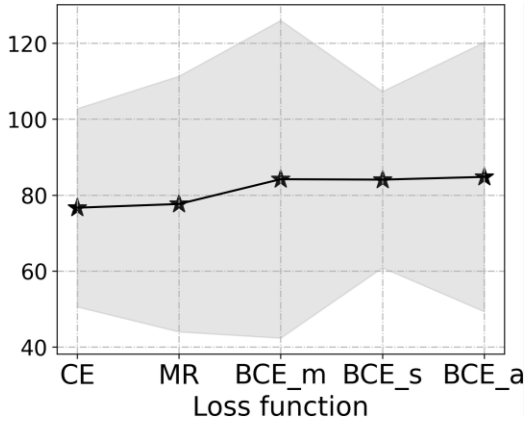
Observation:
batchsize↑ or dimension↑
⇒ time cost↑







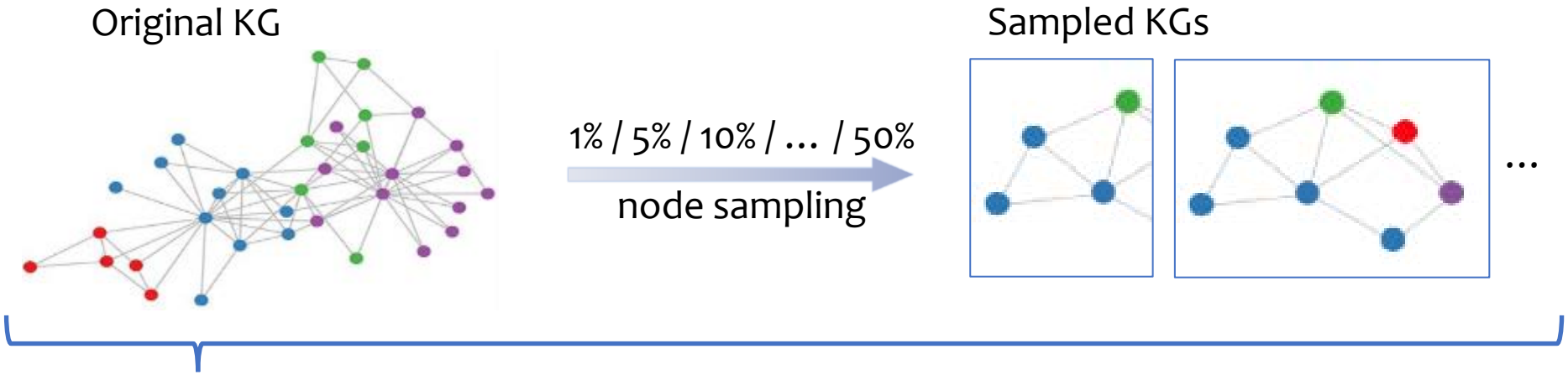


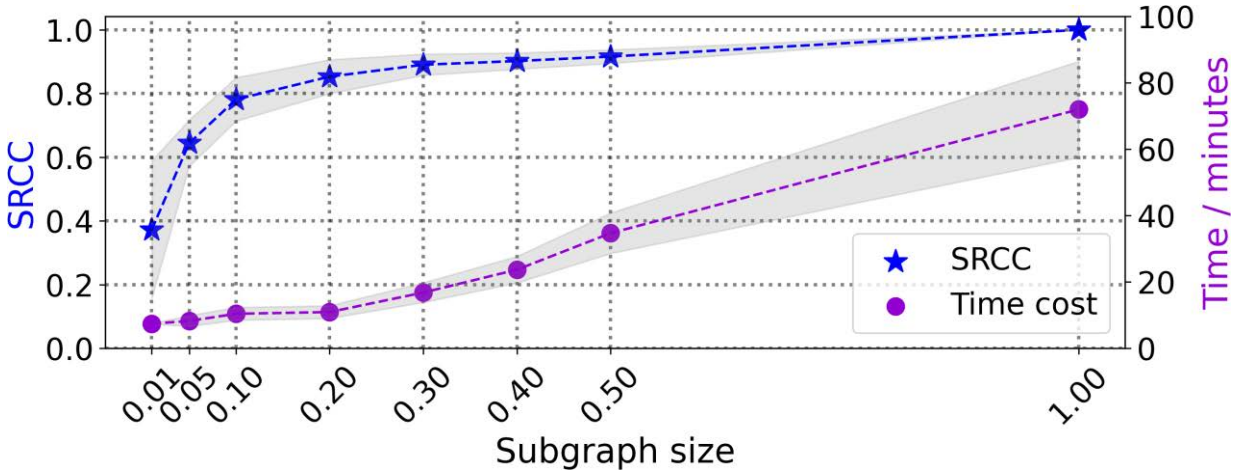


1: The experiments are implemented with PyTorch framework,
on a machine with Intel Xeon 6230R CPUs, 754 GB memory and RTX 3090 GPUs with 24 GB.

Understand HP in KG Learning

Evaluation strategy | Transferability of subgraphs





Observation:

- When the subgraph size↑, the consistency↑ and the cost↑
- To balance the consistency and cost, the subgraphs with 20% nodes are the better choices

Understand HP in KG Learning

Summary of the observations

- Ranking distribution/consistency for each HP's values
  - dimension/batchsize
- Full HP range can be reduced and decoupled

- The validation curvature is pretty complex
- RF is better than GP/MLP as the predictor

- Sampling with multi-start random walk can reduce cost while possessing high performance consistency

Three major aspects for efficiency in Def. 1

1. the size of search space χ
2. the validation curvature of $\mathcal{M}$
3. the evaluation cost in solving $argmin_{\mathcal{P}}$

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathcal{X}} \mathcal{M}\big(F(\boldsymbol{P}^*, \mathbf{x}), D_{val}\big),$$
$$\boldsymbol{P}^* = \arg\min_{\boldsymbol{P}} \mathcal{L}\big(F(\boldsymbol{P}, \mathbf{x}), D_{tra}\big).$$

How to design algorithm based on the above observations? 🤔

Motivation and Objective

Major disadvantages of existing works

Low efficiency in searching for HP configuration

- usually in a time-consuming trial-and-error way
- interaction, importance, and tunability of HPs are unclear
- lacking understanding of KG learning components

Objective of KGTuner:

- *Design a new searching algorithm,*
- *for any given dataset and embedding model with limited budget,*
- *to **efficiently** search for the hyper-parameter configuration.*

KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning. ACL 2022

Efficient Two-stage Search Algorithm

Pre-processing step – reduce search space

name	ranges in the whole space	revised ranges
optimizer	{Adam, Adagrad, SGD}	Adam
learning rate	$[10^{-5}, 10^{0}]$	$[10^{-4}, 10^{-1}]$
reg. weight	$[10^{-12}, 10^{2}]$	$[10^{-8}, 10^{-2}]$
dropout rate	[0, 0.5]	[0, 0.3]
inverse relation	{True, False}	{False}
batch size	{128, 256, 512, 1024}	128
dimension size	{100, 200, 500, 1000, 2000}	100

shrink range:

- Fixed choice
- Limited range

decoupled HPs:

- Monotonously related

Reduced space = shrinkage range HPs + decoupled HPs

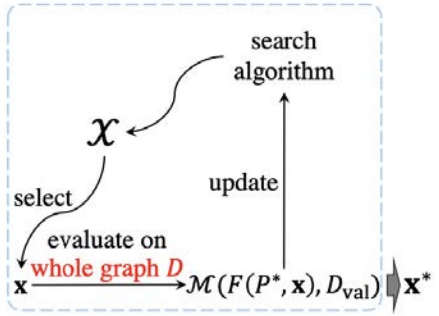
The reduced space is about **1/700** size of the full space

Efficient two-stage HP search algorithm

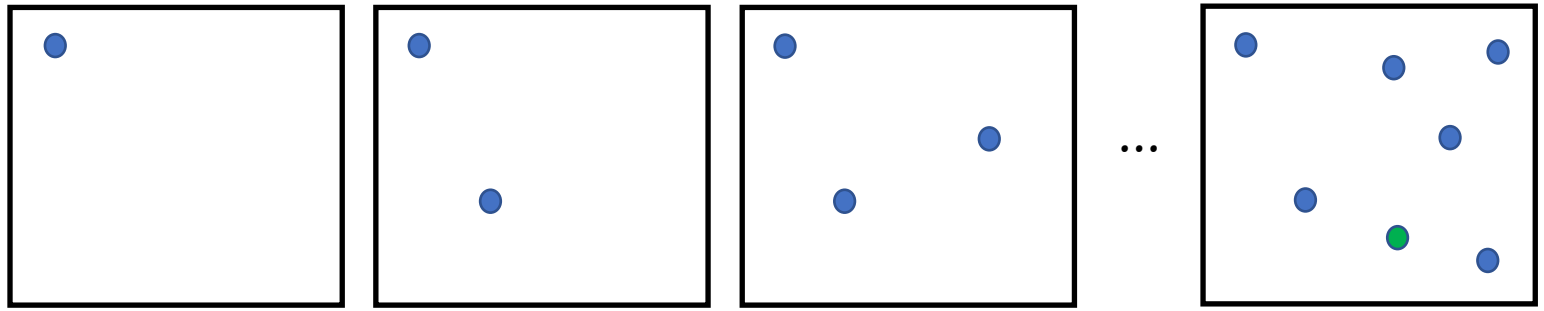
Searching process diagram

Limited time budget

One-stage method



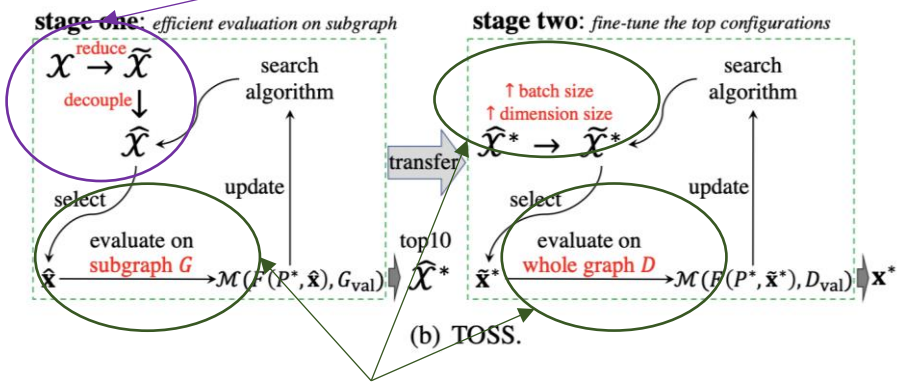
(a) Conventional methods.



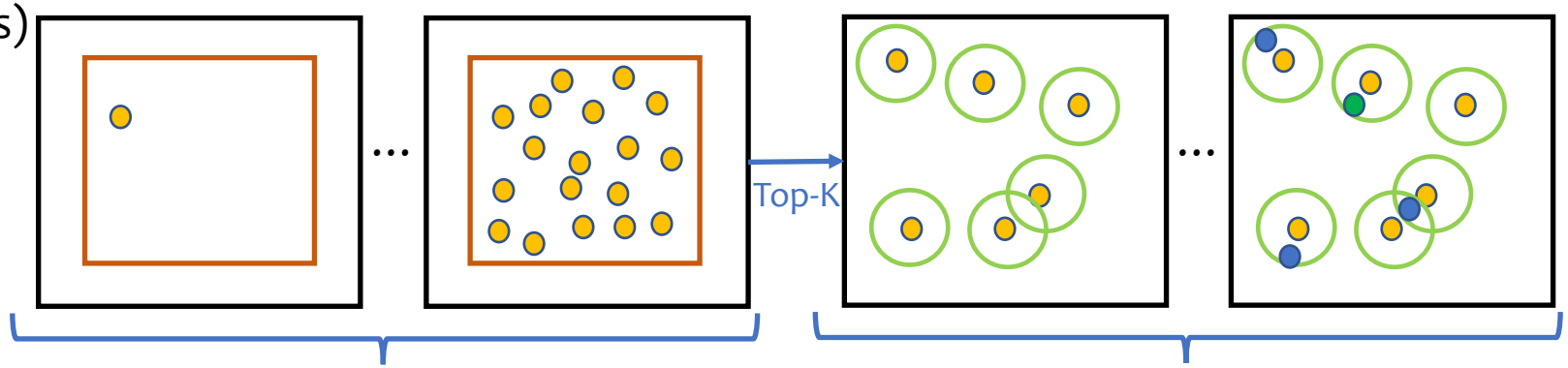
● / ● evaluation on original KG / sampled KG ● optimal configuration

Search space

Two-stage method (ours)



(b) TOSS.



Evaluation strategy

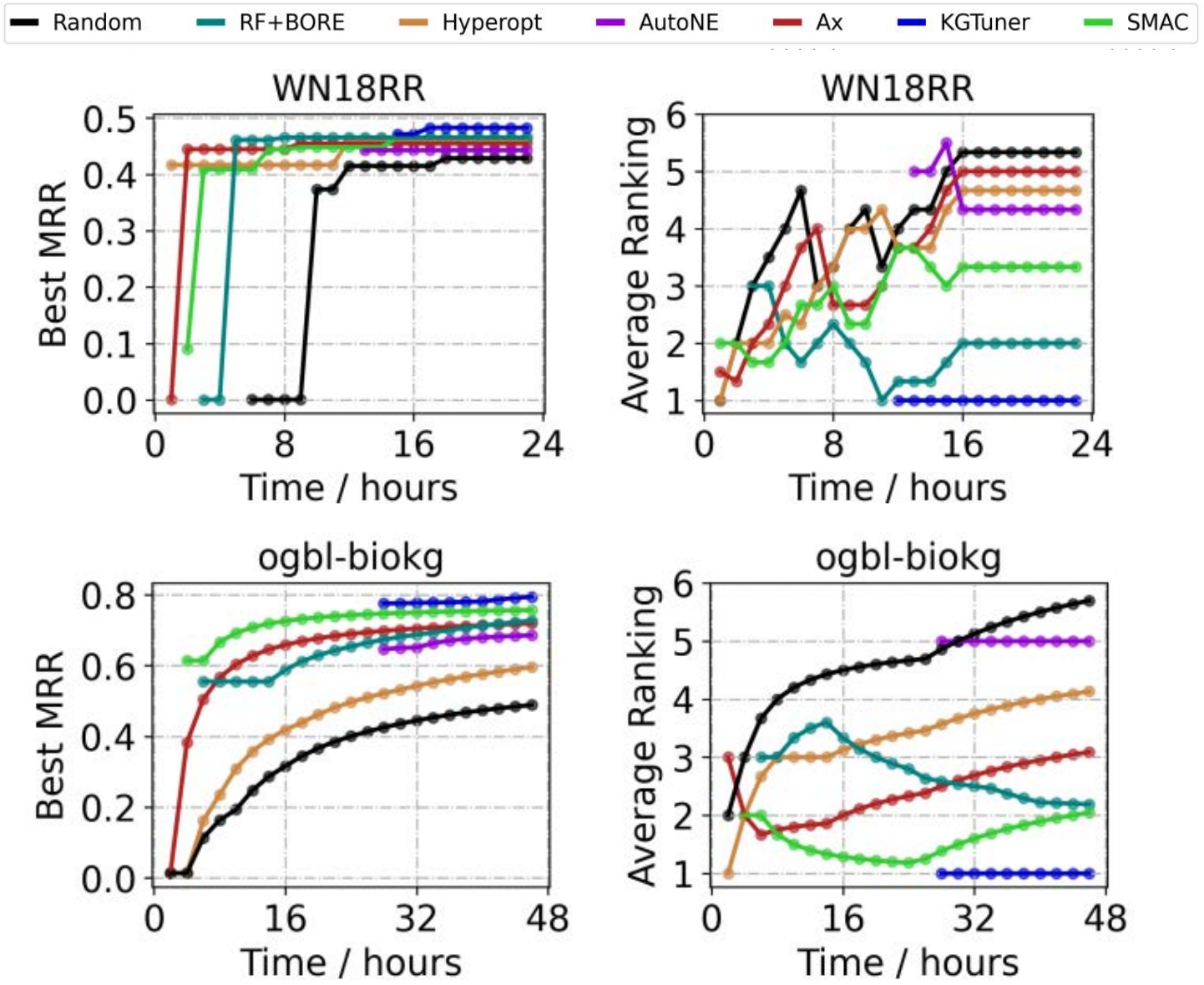
Experiment

Search algorithm comparison

Observation

- Random search is the worst due to the full randomness.
- SMAC and RF+BORE achieve better performance than Hyperopt and Ax since RF can fit the space better than TPE and GP.
- Due to the weak approximation and transferability, AutoNE also performs bad.
- TOSS is much better than all the baselines

	search space		surrogate	fast
	reduce	decouple	model	evaluation
Random	×	×	×	×
Hyperopt	×	×	TPE	×
Ax	×	×	GP	×
SMAC	×	×	RF	×
RF+BORE	×	×	RF	×
AutoNE	×	×	GP	√
TOSS	√	√	RF	√



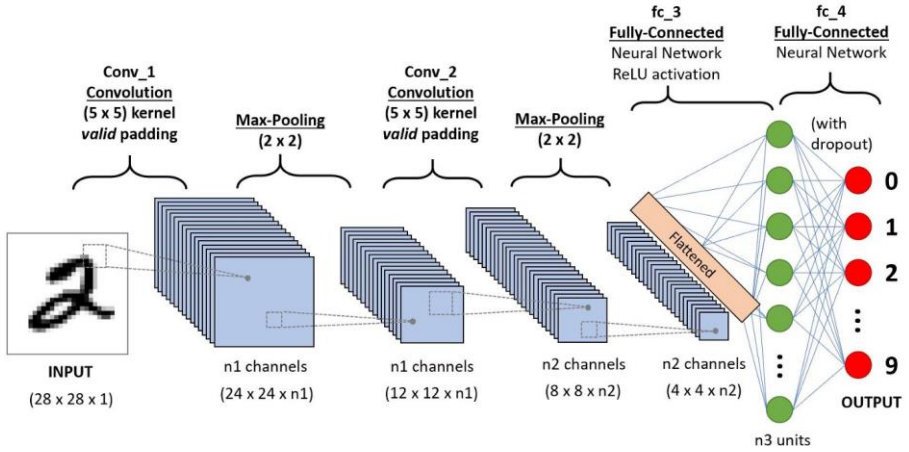
Summary

- Why?
  - No best hyper-parameters for all problems
  - Hyper-parameters have a great impact on the performance
- What?
  - Search for best hyper-parameter for specific problem
- How?
  - Search space: reduce space (shrink range, decouple HPs etc.)
  - Search algorithm: e.g. KGTuner (efficient two-stage HP search algorithm)
  - Evaluation strategy: e.g. subgraphs to the whole graph

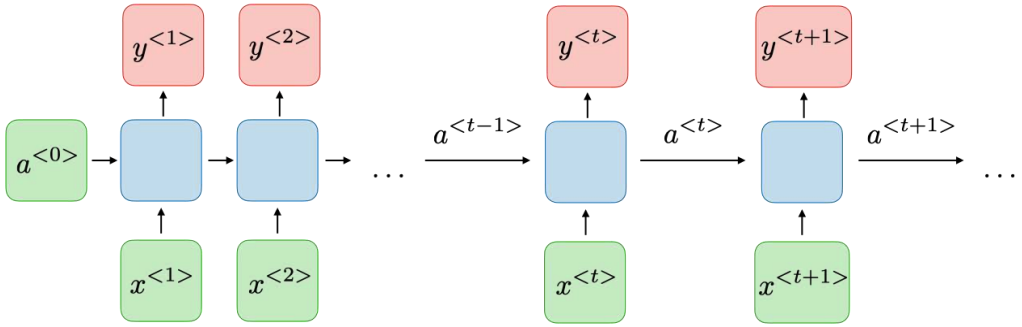
Outline

- What is AutoML
- How to do AutoML
- Application examples of AutoML
  - Example 1: AutoML for Hyper-Parameter Optimization (HPO)
  - Example 2: AutoML for Neural Architecture Search (NAS)
  - Example 3: AutoML for foundation models
- Theory insights

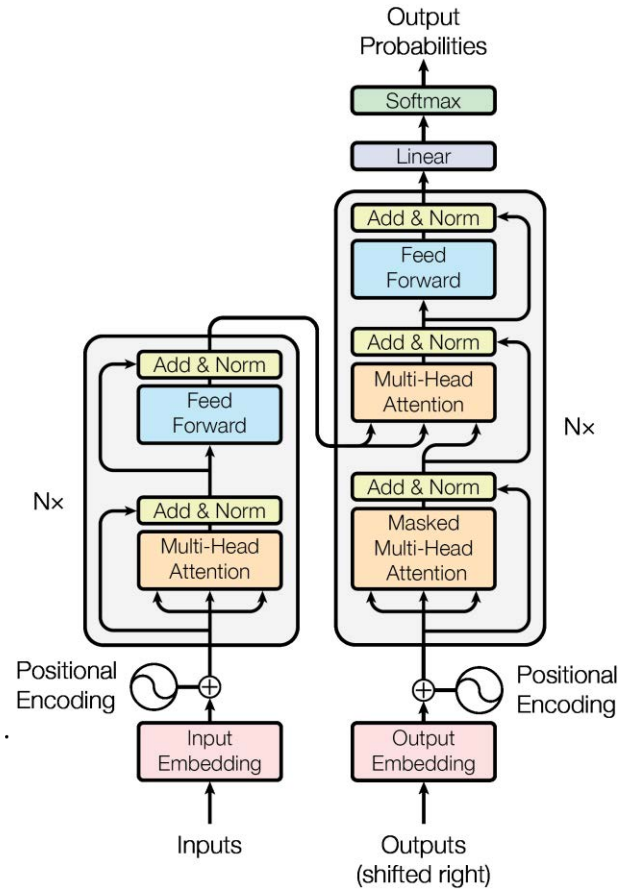
What are Neural Architectures?



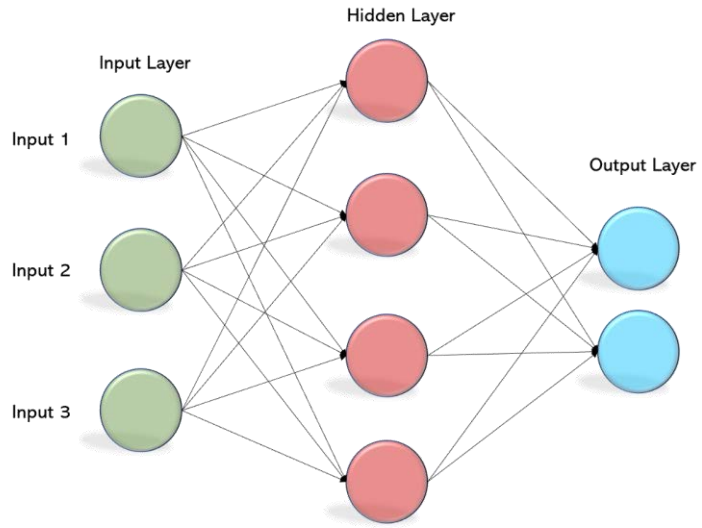
Convolutional neural network (CNN)



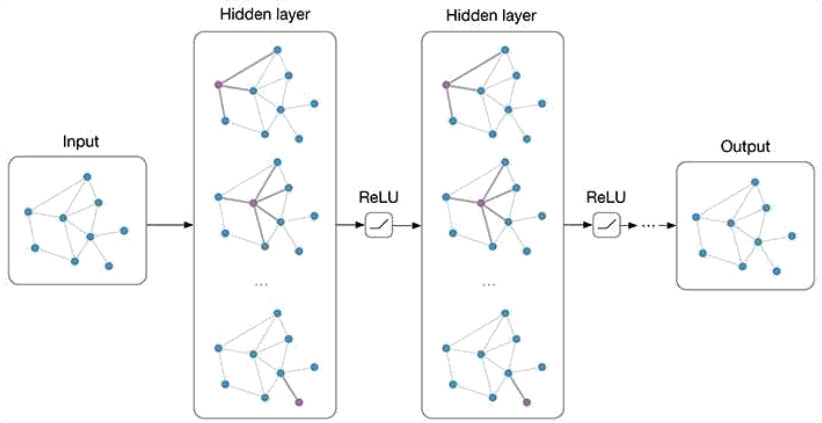
Recurrent Neural Networks (RNN)



Transformer



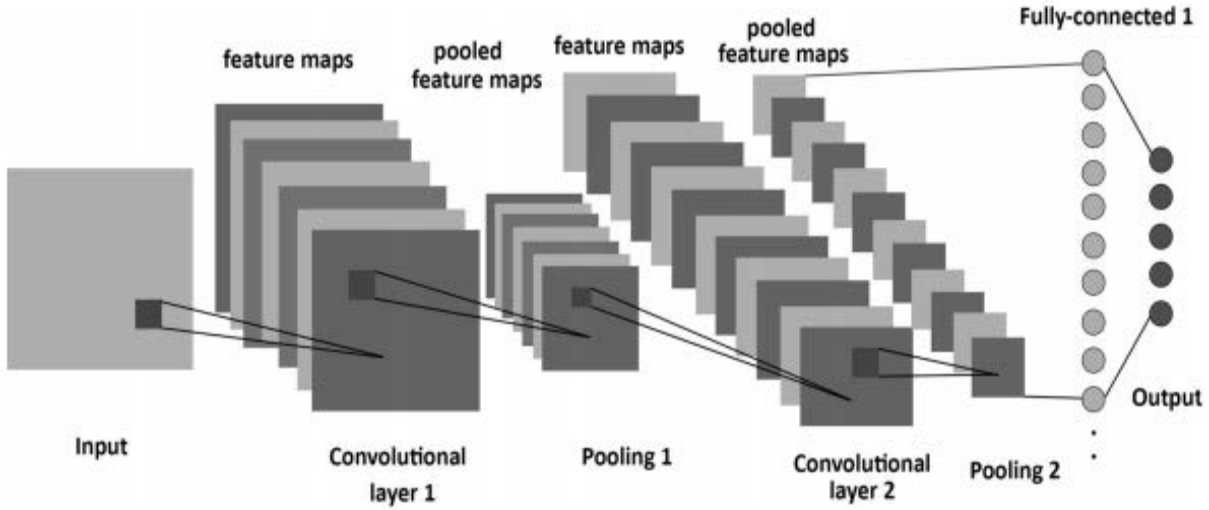
Multilayer perceptron (MLP)



Graph Neural Network (GNN)

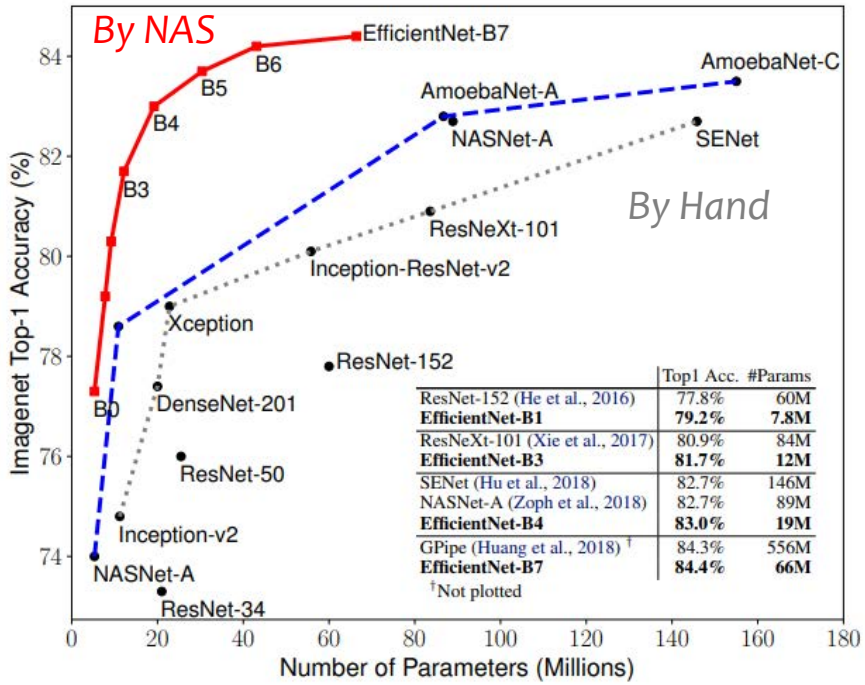
Why Need to Search Architectures?

Architecture of networks are critical to deep learning's performance but hard to fine-tune



Design choice in each layer

- number of filters
- filter height
- filter width
- stride height
- stride width
- skip connections

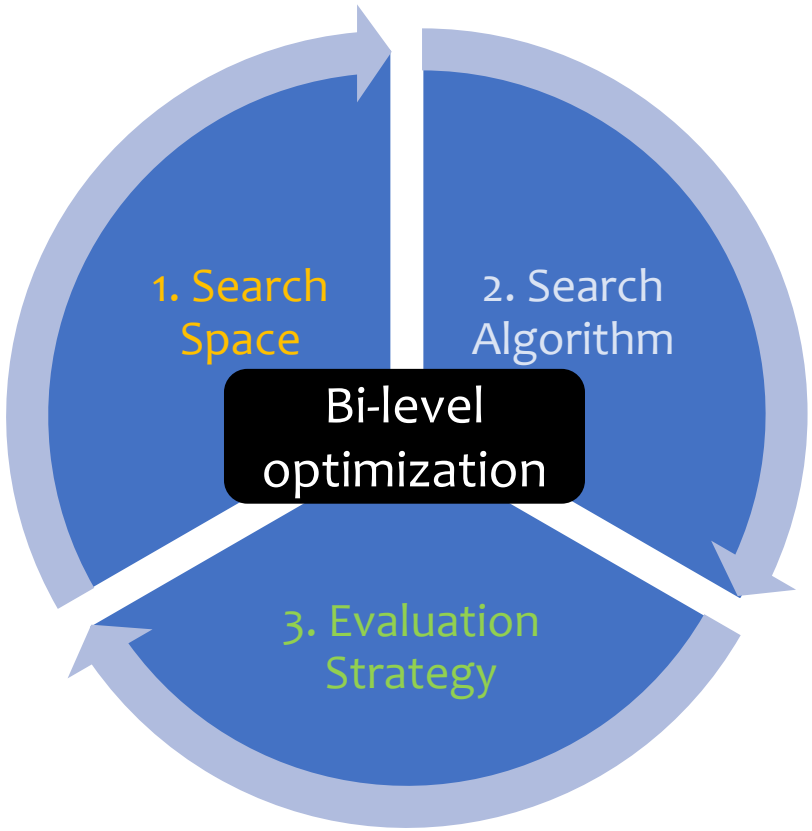


Much better than hand-designed ones

Neural Architecture Search (**NAS**) tries to directly optimize network architecture using validation data sets

Right figure: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019
Left figure: Gradient-based learning applied to document recognition. 1998

Math Formulation for NAS



1. Define an NAS problem

- Derive a search space from insights in specific domains
- Search objective is usually validation performance
- Search constraint is usually resource budgets
- Training objective usually comes from classical learning models

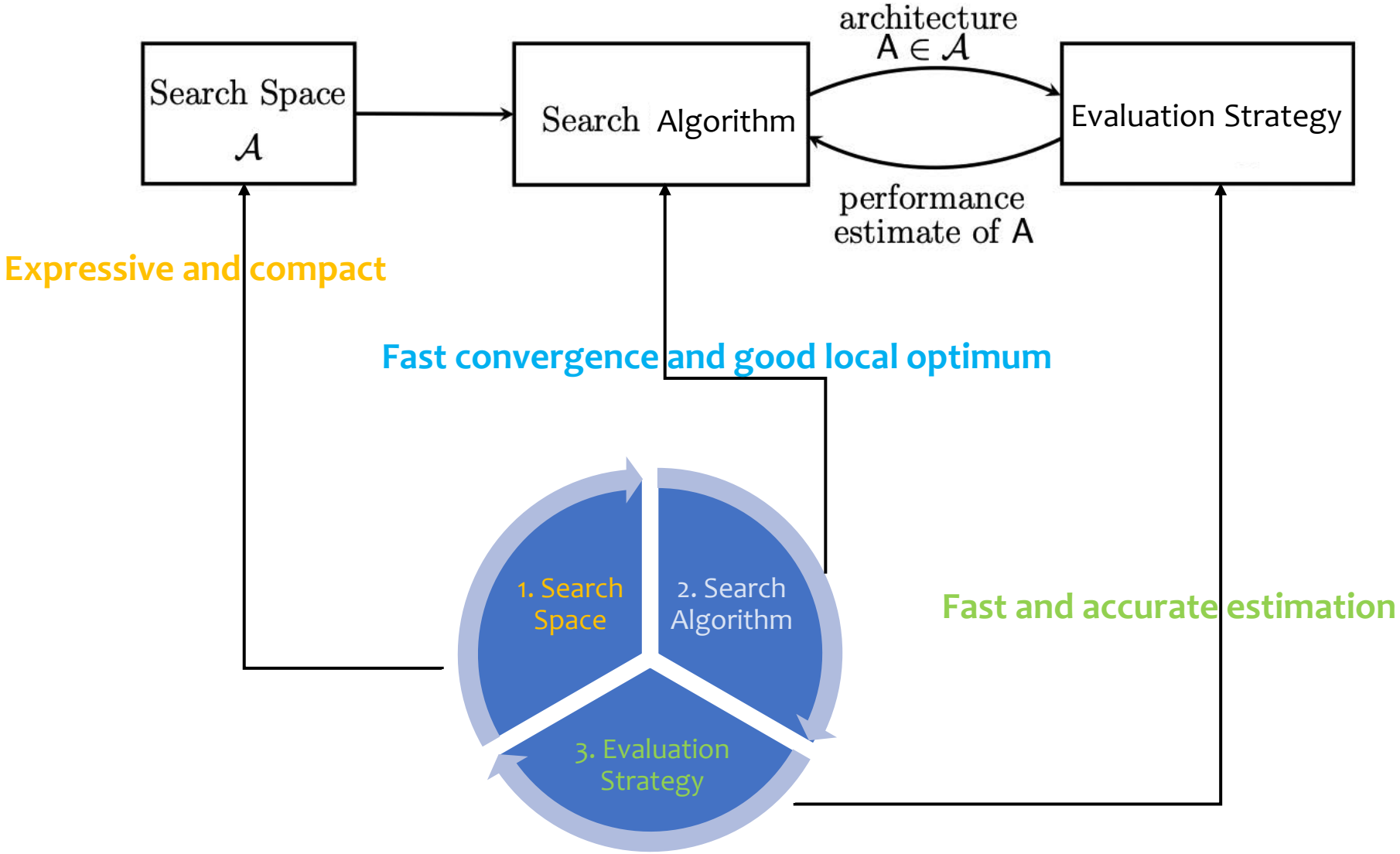
Search Space → $\min_{\lambda \in \mathcal{S}} M(F(w^*; \lambda), D_{\text{val}})$ ← Search Algorithm

s. t. $\min_{w} L(F(w; \lambda), D_{\text{tra}})$ ← Training Objective

2. Design or select proper search algorithm

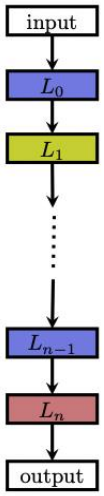
- Reduce model training cost (time to get w^*)

Three Components in NAS



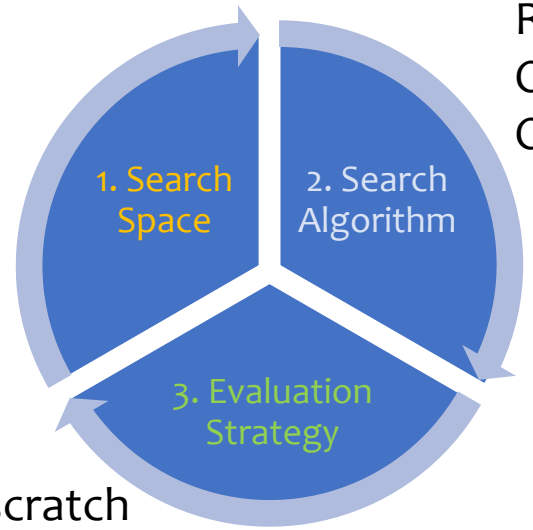
Pioneer Work of NAS (RL-NAS)

1. Search space: the set of candidate architectures constructed based on the designed backbone.



Filter height
Filter width
Stride height
Stride width
Filter number

5^L



3. Evaluation strategy: train from scratch to convergence.

2.Search algorithm: Explore the search space with RNN, and optimize the RNN with reinforcement learning to maximize the expected validation accuracy of the proposed architectures .

Reinforcement learning:
Action: the generated sequential operations $a_{1:T}$
Reward: accuracy R
Objective: $\max J(\theta_c) = \max E_{P(a_{1:T};\theta_c)}[R]$
Optimization:

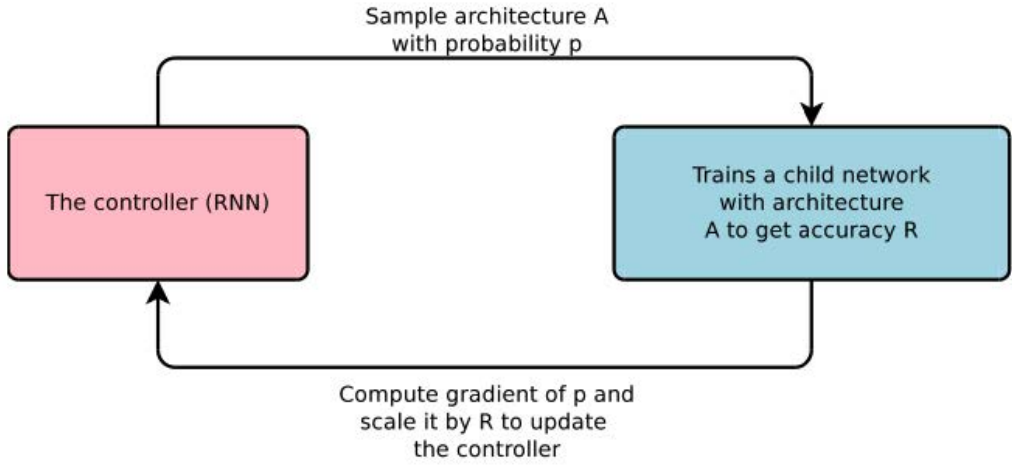
$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}\left[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\right]$$

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R_k$$

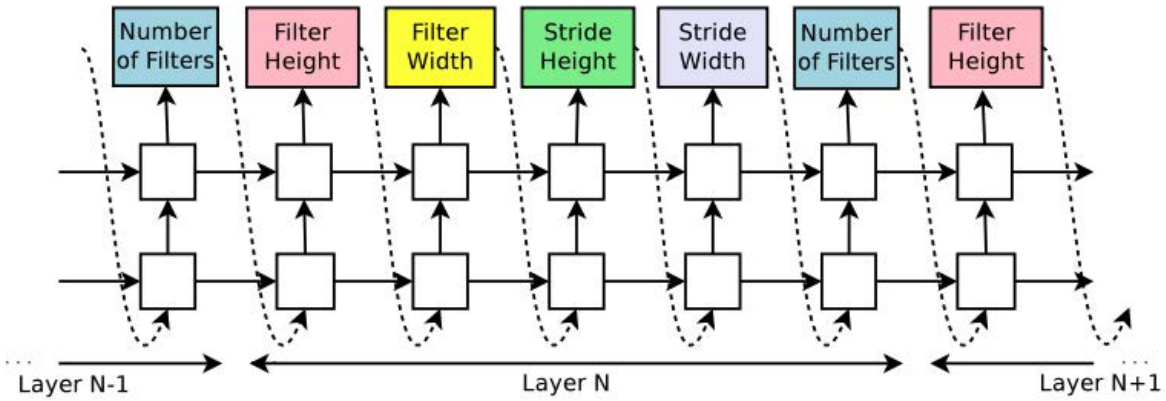
$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

Neural Architecture Search with Reinforcement Learning. ICLR 2017

RL-NAS Pipeline



An overview of Neural Architecture Search (search algorithm)



The controller to generate architectural hyperparameters of neural networks (search space)

Neural Architecture Search with Reinforcement Learning. ICLR 2017

RL-NAS Results

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

The lower the better

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

The lower the better

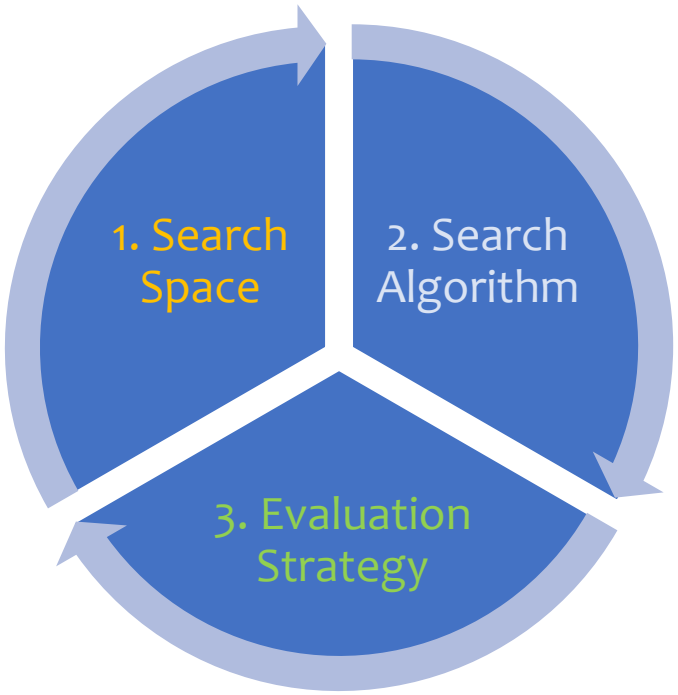
Neural Architecture Search with Reinforcement Learning. ICLR 2017

Recap

	Search space	Search algorithm	Evaluation strategy
RL-NAS	Variable-length string	Reinforcement learning	Train from scratch until convergence

NASNet: Making NAS Possible on ImageNet

An important large-scale image dataset

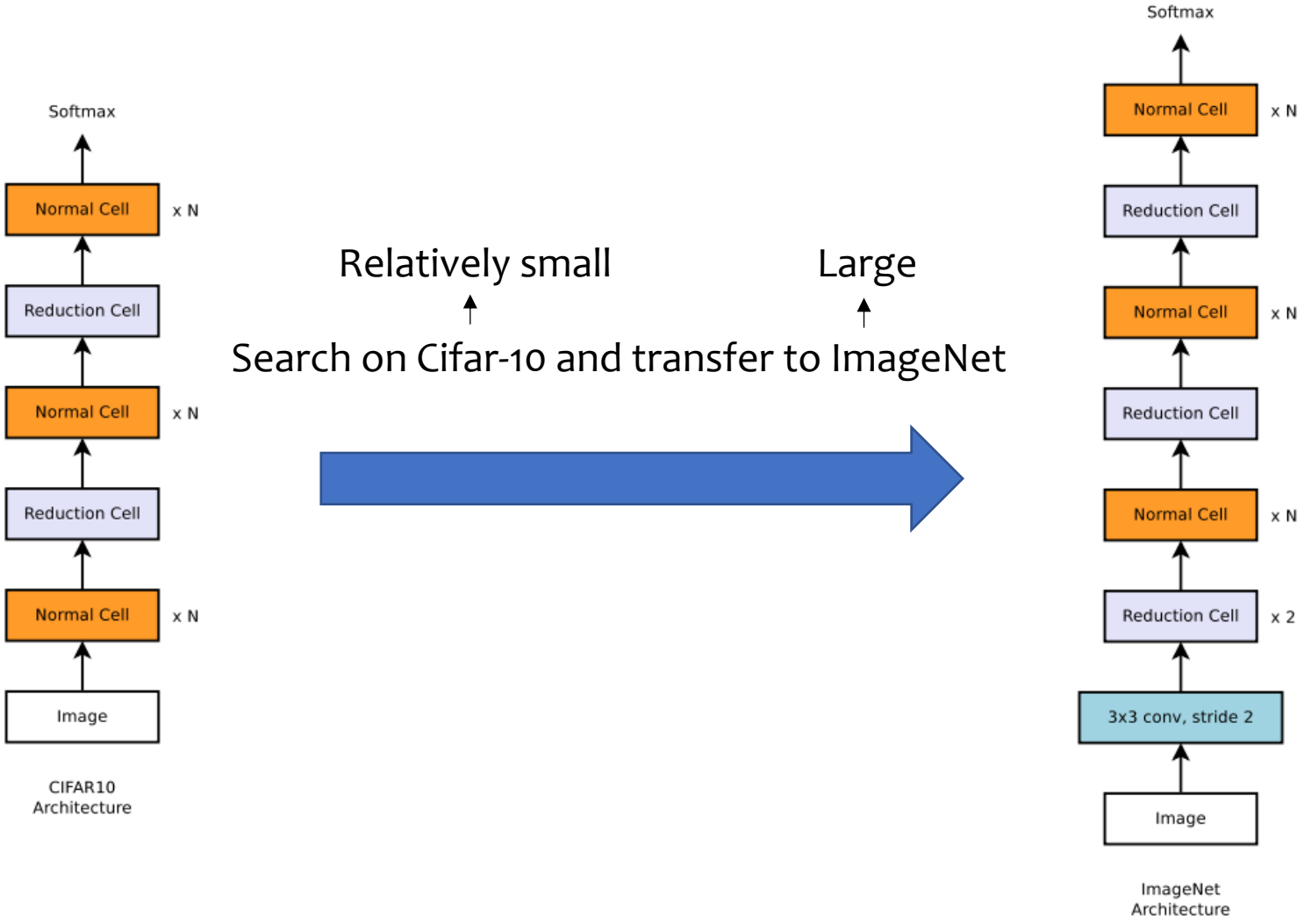


Compare to RL-NAS:

Search space: Details next
(requirement: **Expressive and compact**)
Search algorithm: Same
Evaluation strategy (performance estimation): Same

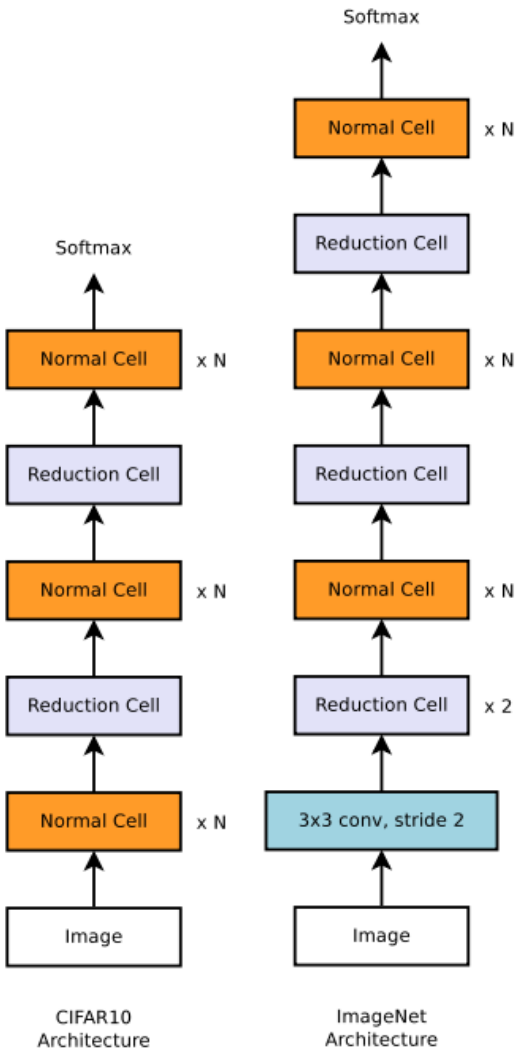
Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Transferable Search Space

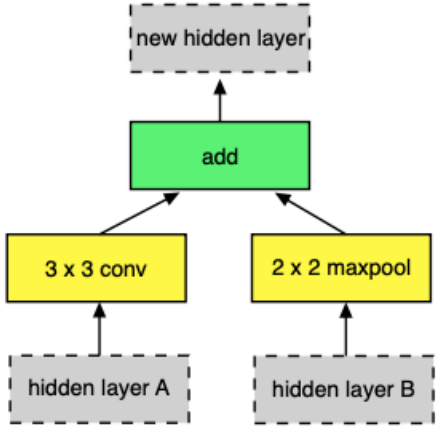


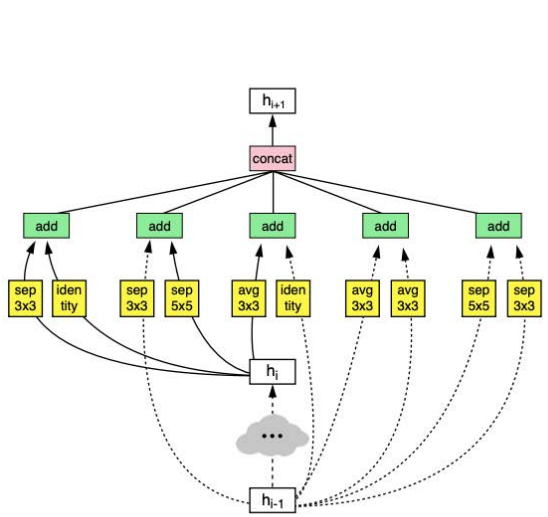
Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Improvements on Search Space

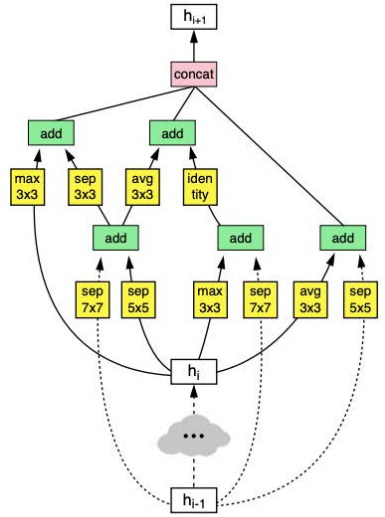


- Key idea: Search for an architectural building block and then transfer to a larger dataset.
  - Design the Normal and Reduction cell, and then stacking to obtain CNNs.
  - Each cell has B blocks, and each block has five operations
- Extracting and re-organizing the factors considering the effectiveness and compactness.





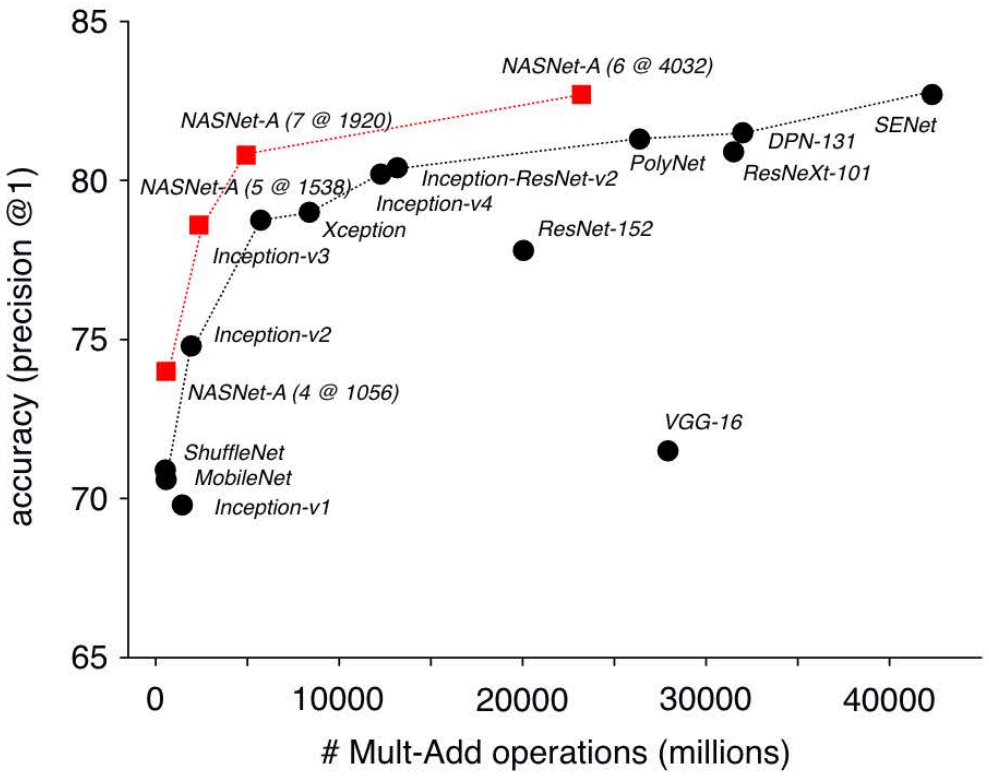
Normal Cell

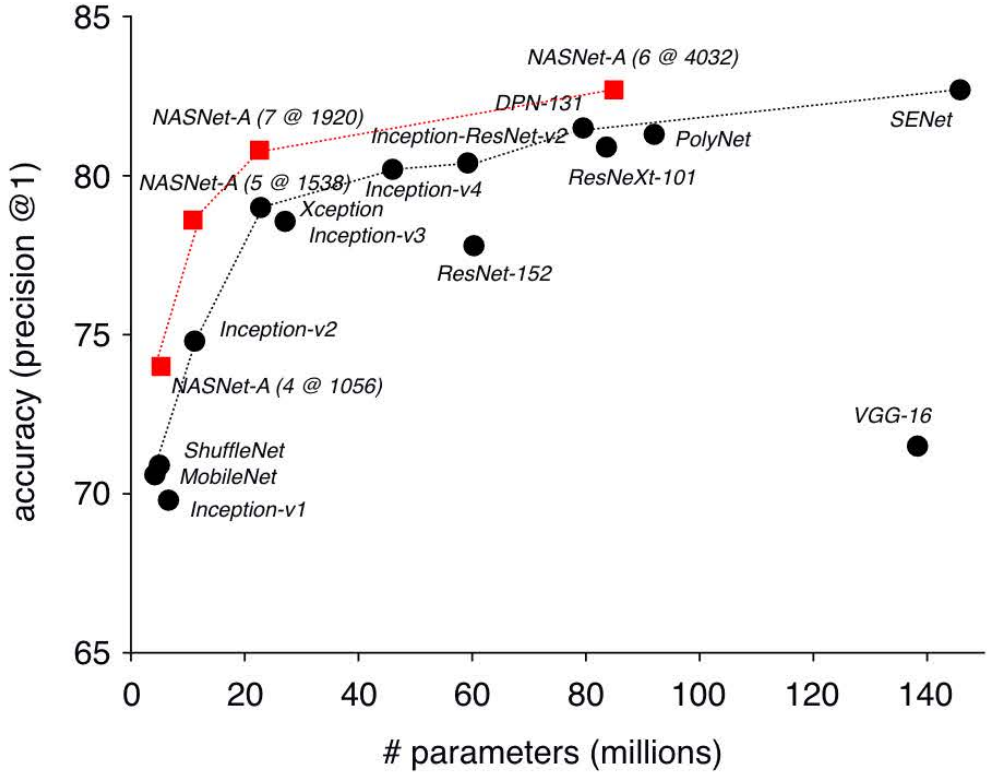


Reduction Cell

Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Making NAS Possible on ImageNet



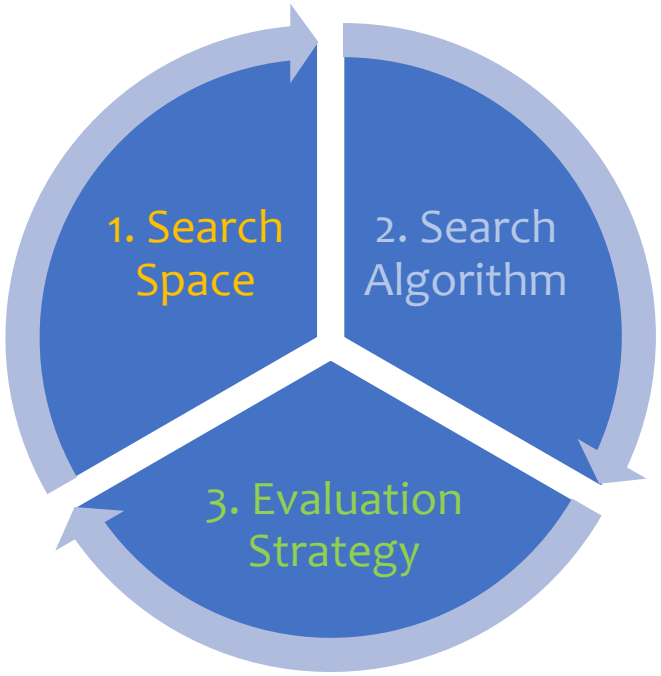


Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

Recap

	Search space	Search algorithm	Evaluation strategy
RL-NAS	Variable-length string	Reinforcement learning	Train from scratch until convergence
NASNet	Block and cell	Same as RL-NAS	Same as RL-NAS

ENAS: very Fast Performance Estimation



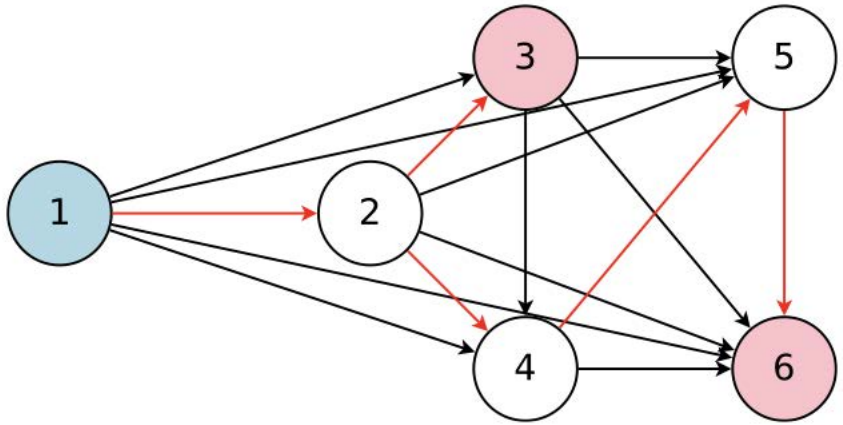
Compare to RL-NAS:

Search space: Directed Acyclic Graph
(modern choice until today)
Search algorithm: Same
Evaluation strategy: Details next
(requirement: **Fast and accurate estimation**)

Efficient Neural Architecture Search via Parameter Sharing, ICML 2018

ENAS: Weight Sharing

- Reuse the architecture parameters and reduce the training cost
- Sharing the parameters among the same child models between different architectures.



Node: child models
Edge: define a model

Arch 1: 1-2-3-6
Arch 2: 1-3-4-6
Arch 3: 1-3-5-6

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

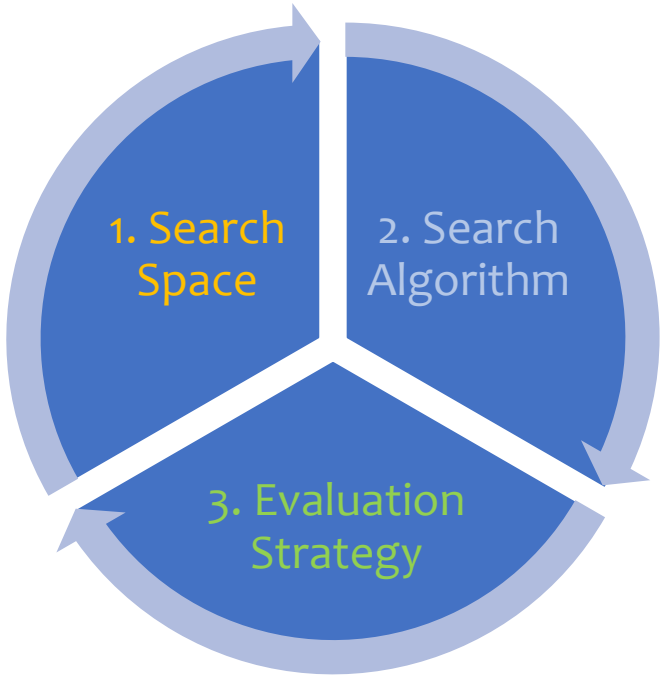
performance comparison

Efficient Neural Architecture Search via Parameter Sharing, ICML 2018

Recap

	Search space	Search algorithm	Evaluation strategy
RL-NAS	Variable-length string	Reinforcement learning	Train from scratch until convergence
NASNet	Block and cell	Same as RL-NAS	Same as RL-NAS
ENAS	Same as RL-NAS	Same as RL-NAS	Weight sharing

DARTS: Differentiable NAS Starts Here



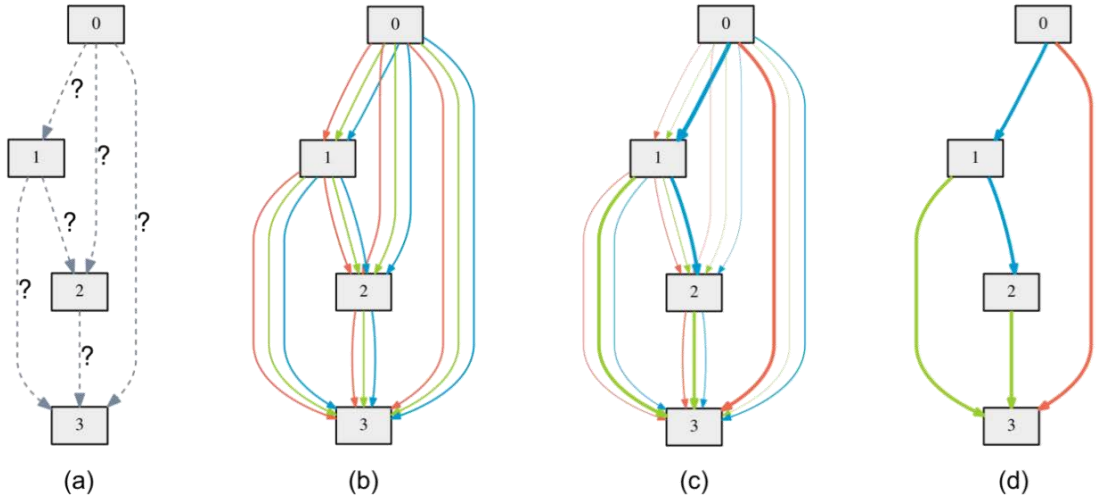
Compare to RL-NAS:

Search space: Directed Acyclic Graph with relaxed choices
Search algorithm: **Differentiable is all you need**
Evaluation strategy (performance estimation): One step gradient descent

All three components have been improved.

DARTS: Differentiable Architecture Search, ICLR 2019

DARTS: Pipeline



Overview of DARTS

(a). designing operations on edges

(b). **continuous relaxation** of the search space

(c). joint optimization of the mixing probabilities and the network weights

(d). inducing the final architecture

(b). continuous relaxation

learnable parameter

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

candidate operations

(c). joint optimization

objective

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

algorithm: SGD

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$$

(d). final architecture: most likely operation

$$o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \ \alpha_o^{(i,j)}$$

Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)

while *not converged* **do**

1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

DARTS: Differentiable Architecture Search, ICLR 2019

DARTS: Results on Cifar-10

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

DARTS: Differentiable Architecture Search, ICLR 2019

DARTS: Results on ImageNet

Architecture	Test Error (%)		Params	$+\times$	Search Cost	Search
	top-1	top-5	(M)	(M)	(GPU days)	Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet $2\times$ ($g = 3$) (Zhang et al., 2017)	26.3	–	~5	524	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	488	2000	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	558	2000	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	588	~225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

DARTS: Differentiable Architecture Search, ICLR 2019

Summary

	Search space	Search algorithm	Evaluation strategy
RL-NAS	Treat network as variable-length string	Reinforcement learning	train from scratch to convergence
NASNet	Block and cell	Same as RL-NAS	Same as RL-NAS
ENAS	Same as RL-NAS	Same as RL-NAS	Weight sharing
DARTS	Continuous relaxation	Differentiable	One step GD

Outline

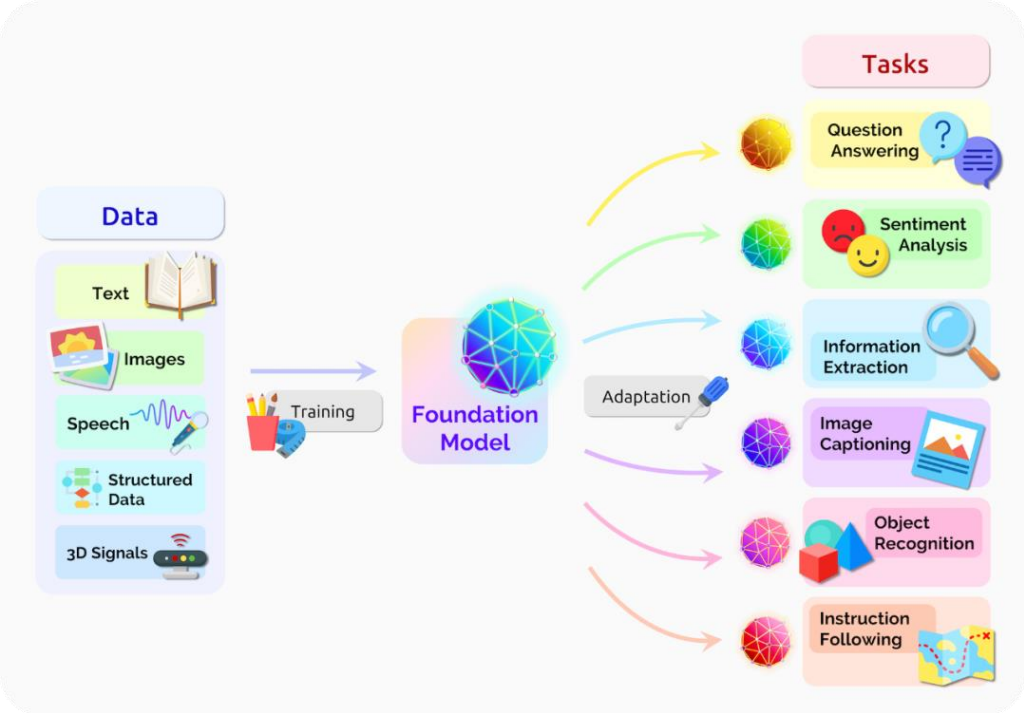
- What is AutoML
- How to do AutoML
- Application examples of AutoML
  - Example 1: AutoML for Hyper-Parameter Optimization (HPO)
  - Example 2: AutoML for Neural Architecture Search (NAS)
  - Example 3: AutoML for foundation models
- Theory insights

What is foundation model ?

“

Any model that is trained on broad data (generally using self-supervision at scale) that can be adapted (e.g., fine-tuned) to a wide range of downstream tasks. ”

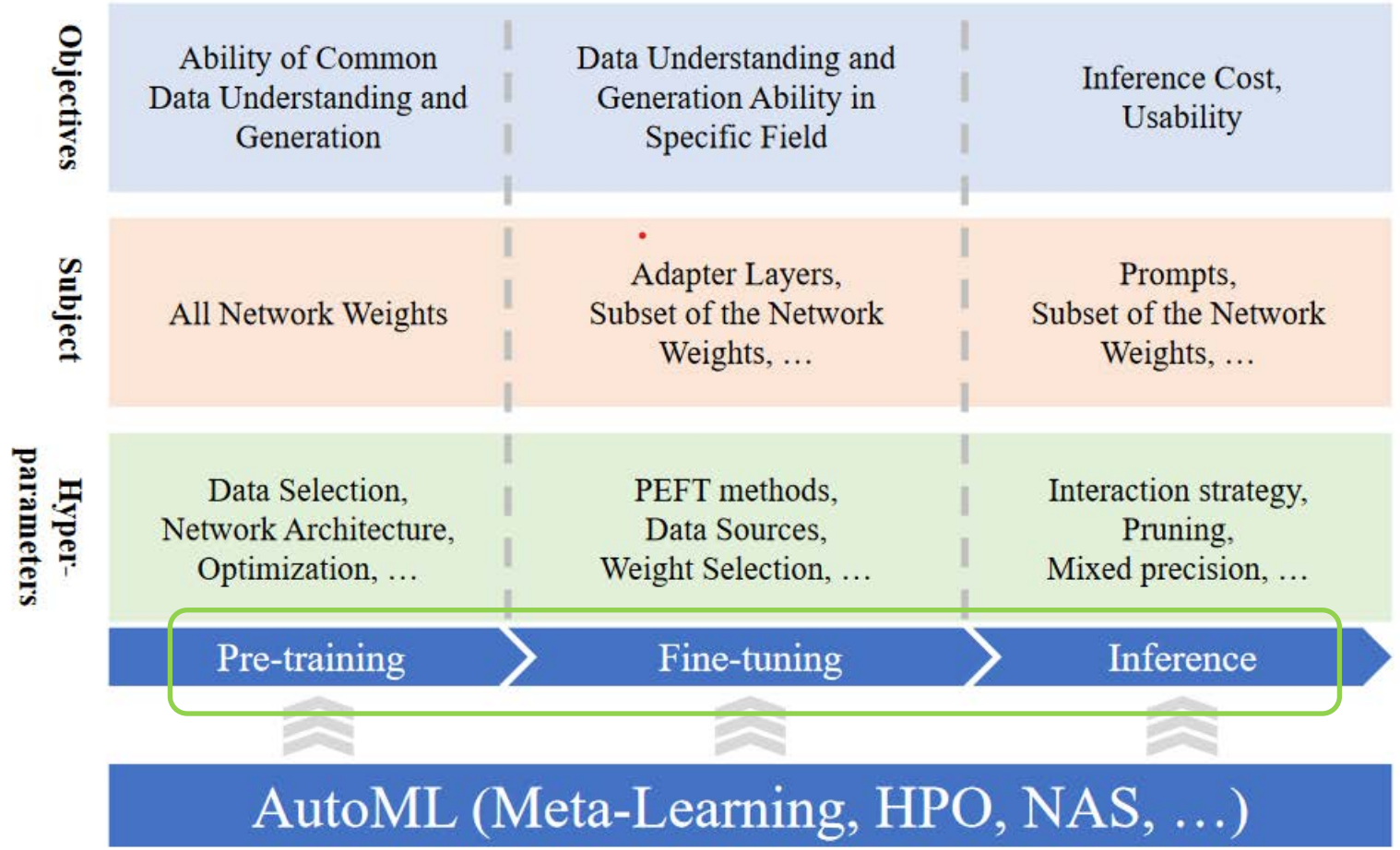
——the Stanford Institute for Human-Centered Artificial Intelligence’s (HAI) Center



- Large language model “GPT-x”
- The text-to-image generation model “Stable Diffusion”
- The graph learning model “GROVER” etc.

On the Opportunities and Risks of Foundation Models

AutoML for all Stages of Foundation Models

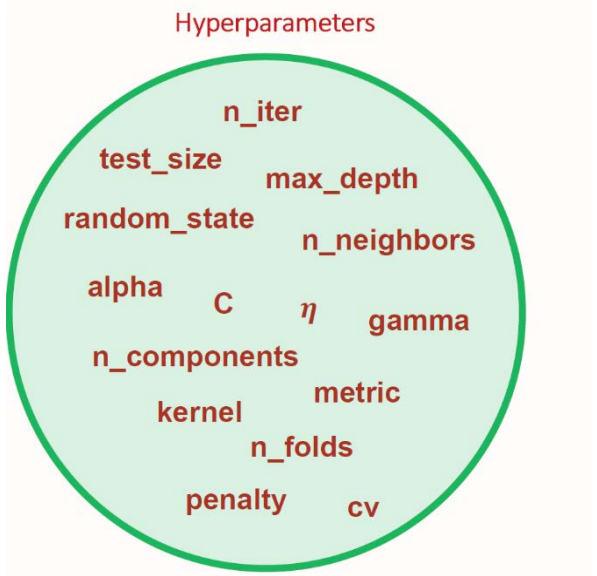


- In each stage, AutoML optimizes both subject and associated hyper-parameters to achieve corresponding objective

An Example: AutoML for Large Language Model

- Pre-training stage:
  - Estimated: GPT-4 used 25,000 A100s almost 100 days!
  - A large amount of training hyperparameters ← HPO

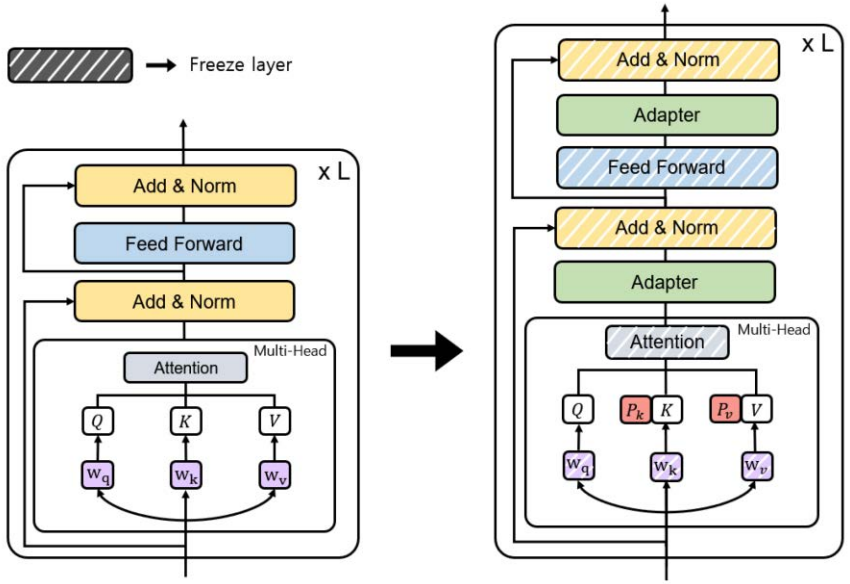
$10,000!



  - More powerful LLM architectures? ← NAS, evolutionary algorithm

An Example: AutoML for Large Language Model

- Fine-tuning stage (trained on specific data):
  - Four representative Parameter-Efficient Fine-Tuning (PEFT) methods
    - Prompt-tuning
    - Prefix-tuning
    - LoRA
    - Adapter

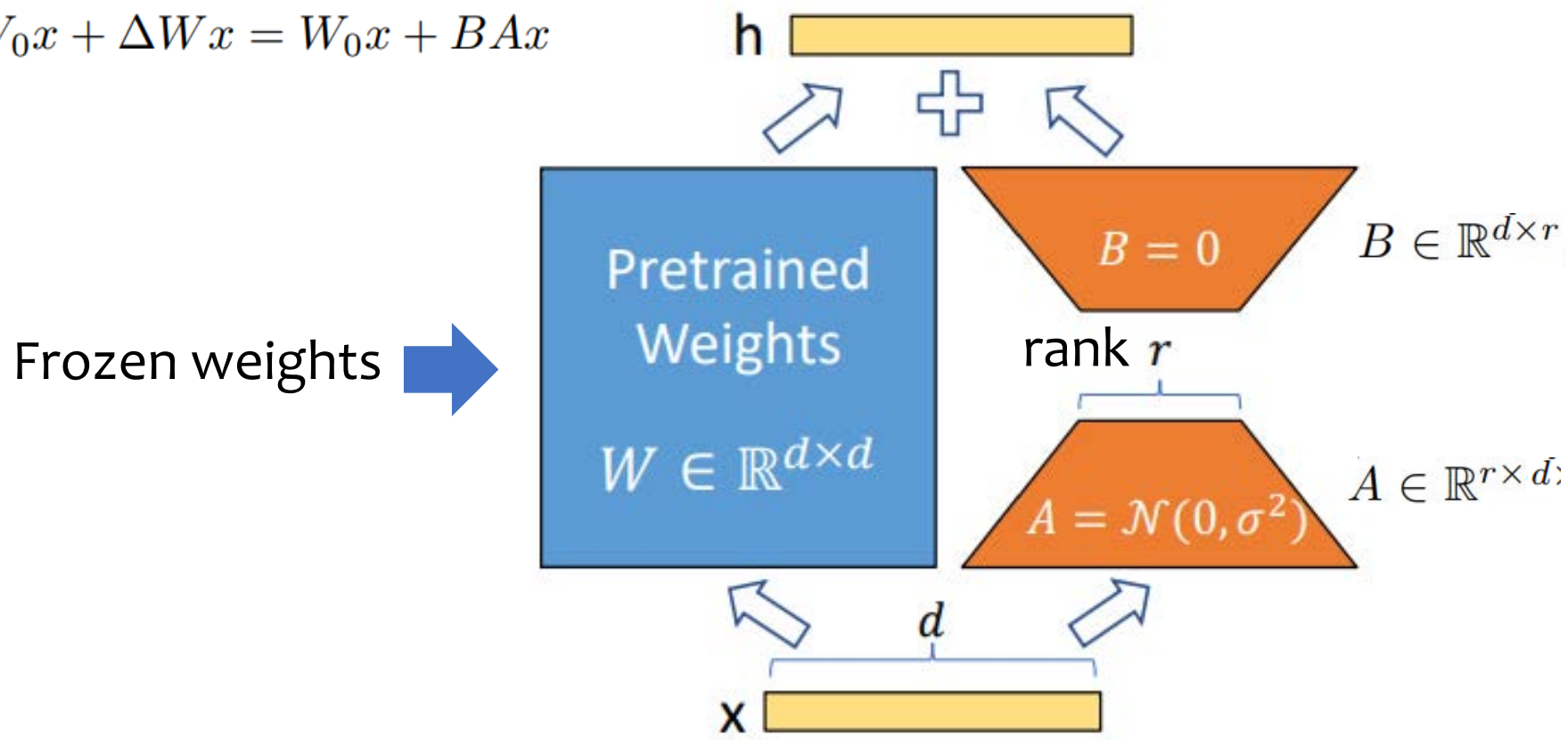


  - AutoML for searching PEFT methods and parameters for downstream tasks

E.g. Hyperformer++, AutoPEFT, S2PGNN etc.

LoRA: Low-Rank Adaptation of LLMs

$$h = W_0 x + \Delta W x = W_0 x + BAx$$



Lora: Low-rank adaptation of large language models. ICLR 2022

Performance on GPT-3

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

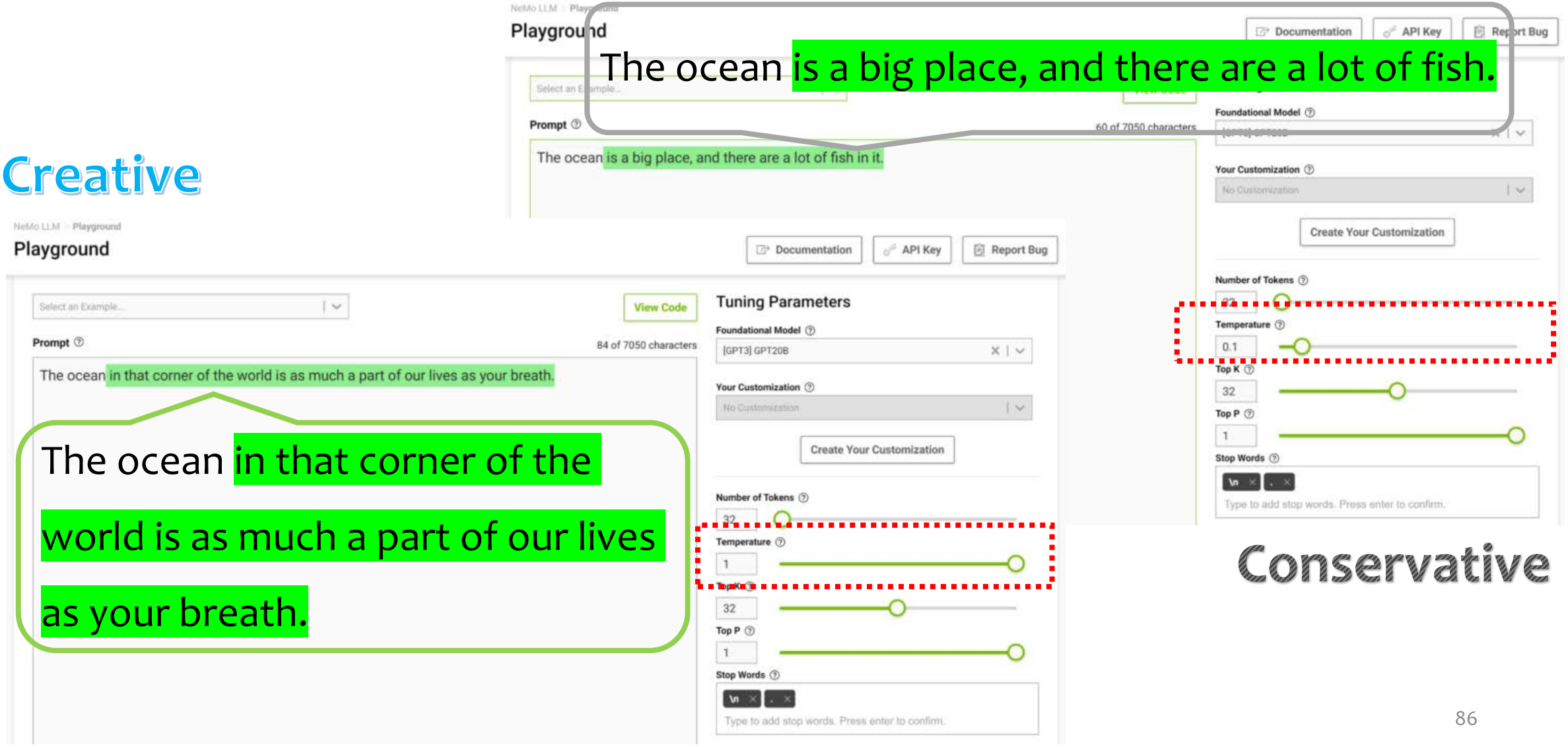
Lora: Low-rank adaptation of large language models. ICLR 2022

An Example: AutoML for Large Language Model

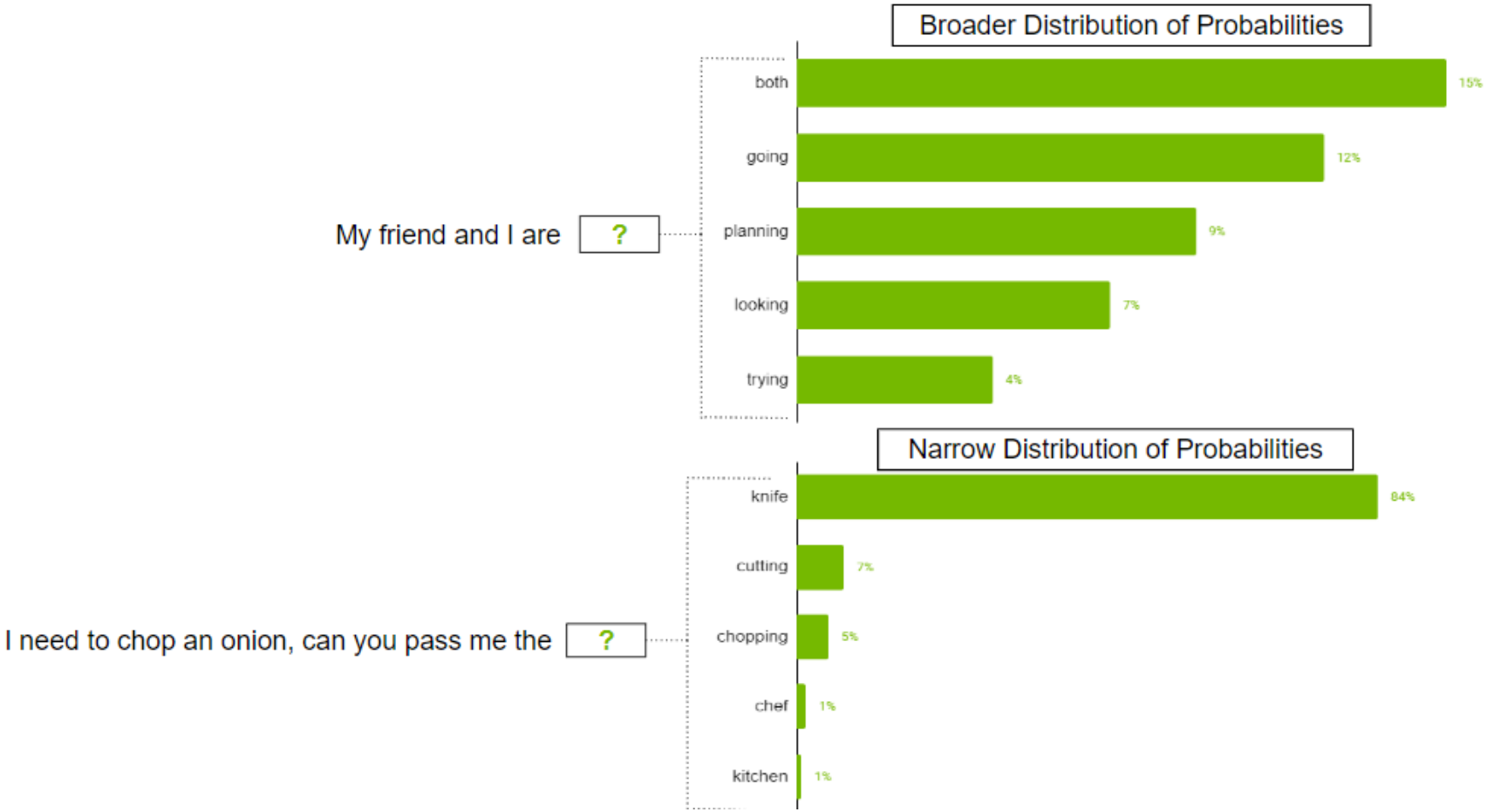
- Inference stage:
  - How to make inference more cost-effective and efficient ?
  - NAS: compress a well-trained foundation model
  - HPO: Hyperparameters to tune, such as “temperature” , “top-k” and “top-p”

inference performance  operational costs 

Hyper-parameter: Temperature



Hyper-parameter: Top-k and Top-p



Top-k: select the top K most probable words at each timestep.

Top-p: selecting words within a set whose cumulative probability exceeds a certain threshold p.

Summary

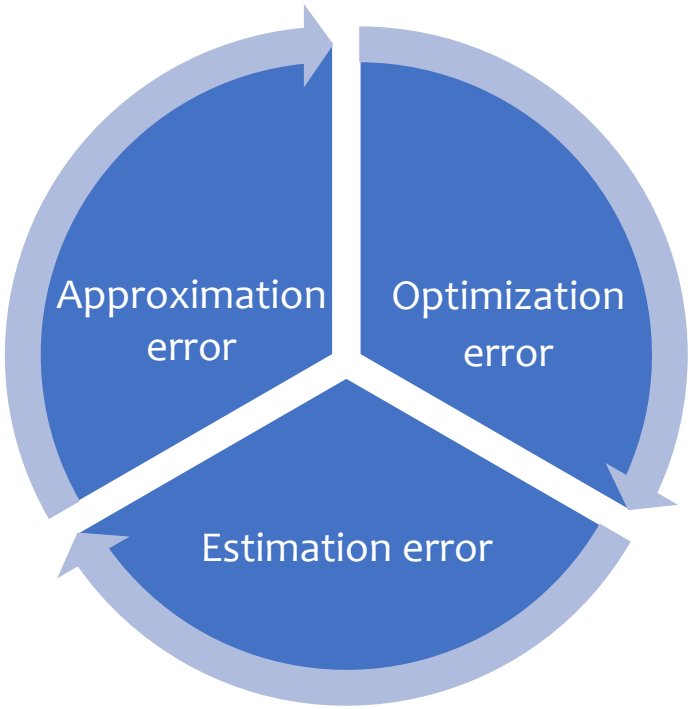
- AutoML can be used in all three stages of foundation models: pre-training, fine-tuning, inference.
- Future and challenges:
  - Cost of pre-Training base models
  - The multitude of performance indicators
  - Combination of different learning paradigms and modalities

Outline

- What is AutoML
- How to do AutoML
- Application examples of AutoML
- Theory insights

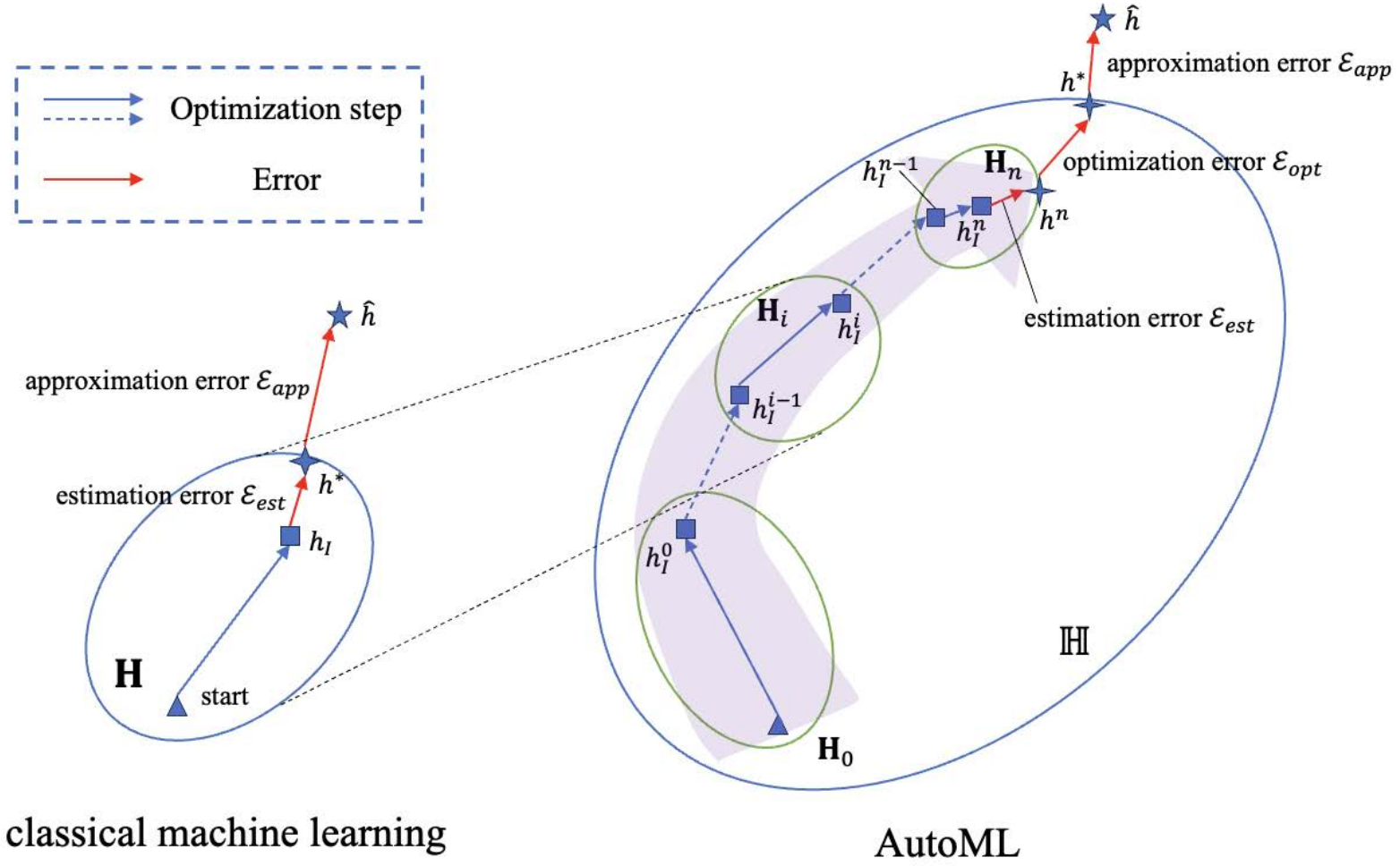
Recall: Machine learning error decomposion

- Approximation error:
  - Error caused by mis-specified/small function space
- Optimization error:
  - Error caused by imperfect algorithm
- Estimation error:
  - Error caused by insufficient data



Theoretical guarantee

- Comparison with classical machine learning
- Larger search space for smaller approximation error



Variations for three key components

- Reflect the trade-off for the design of three parts
  - Smaller error or shorter computation time

	search space (larger)	search algorithm (more iterations)	evaluation strategy (more per-iteration cost)
approximation error	↘	-	-
optimization error	↗	↘	↘
estimation error	-	-	-
computation time	↗	↗	↗

Class Summary

- Target of AutoML: let machine find optimal learning configurations automatically
- Three main components:
  - Search space, Search algorithm, Evaluation stragety
- Three examples:
  - Example 1: tuning knowledge graph (KG) learning hyper-parameters
  - Example 2: Neural Architecture Search (NAS)
  - Example 3: AutoML for foundation models

Thanks!

Q & A