Neural Architecture Search
- with a focus on GNN

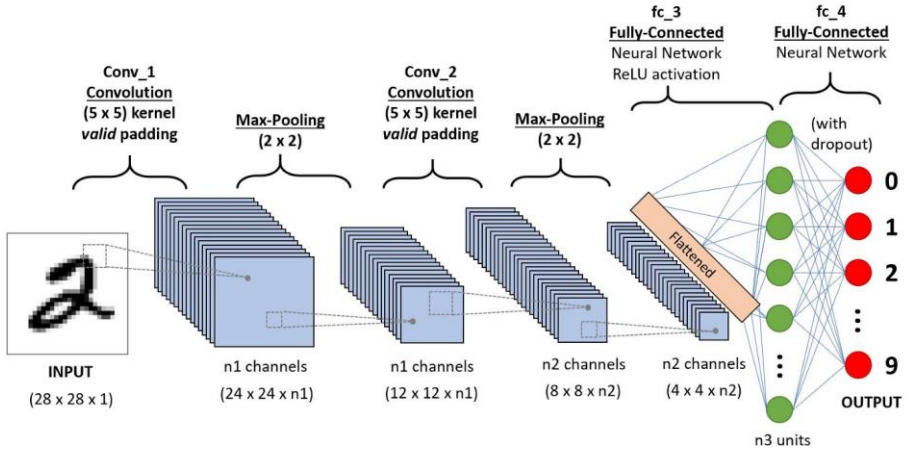
姚权铭

qyaoaa@tsinghua.edu.cn

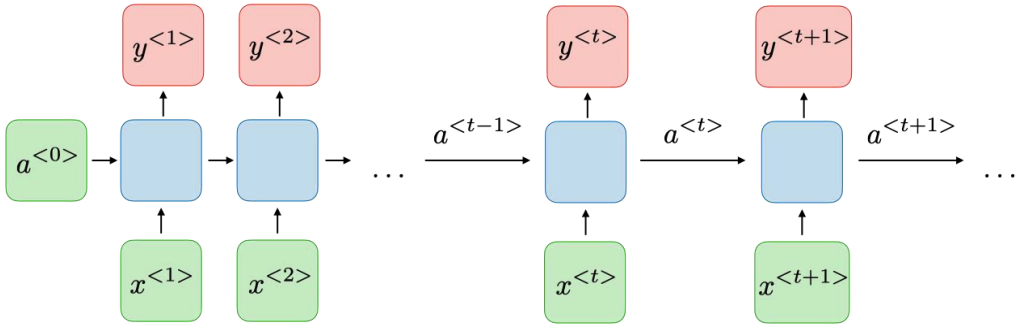
Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
- Summary

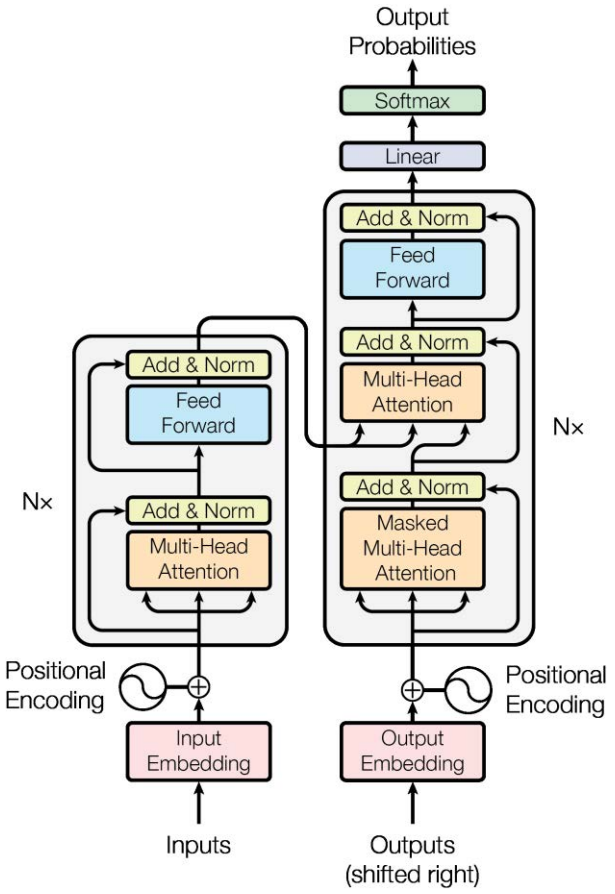
What are Neural Architectures?



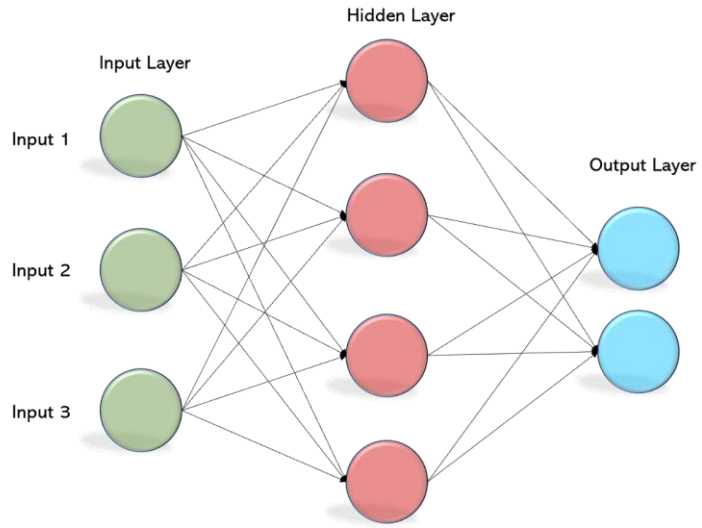
Convolutional neural network (CNN)



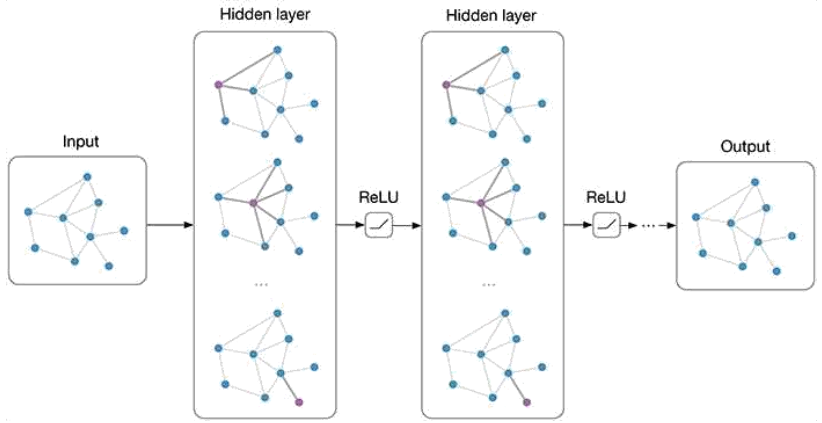
Recurrent Neural Networks (RNN)



Transformer



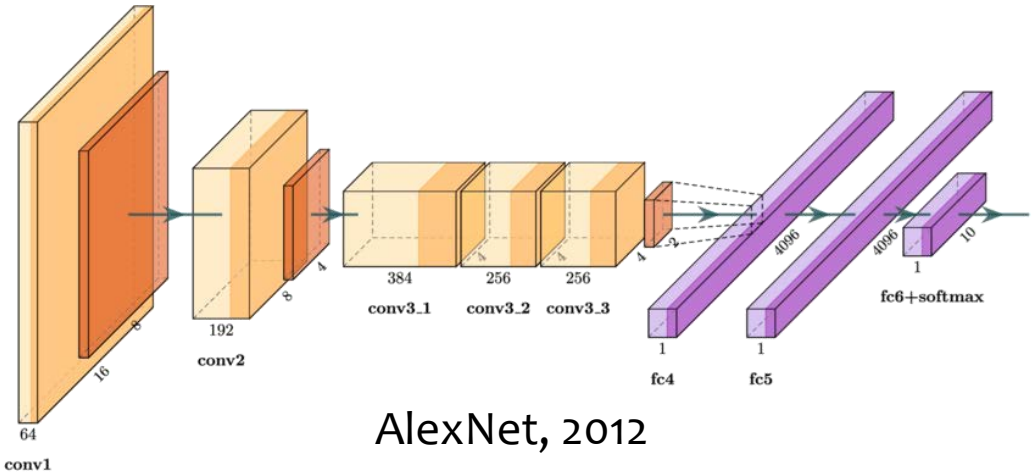
Multilayer perceptron (MLP)



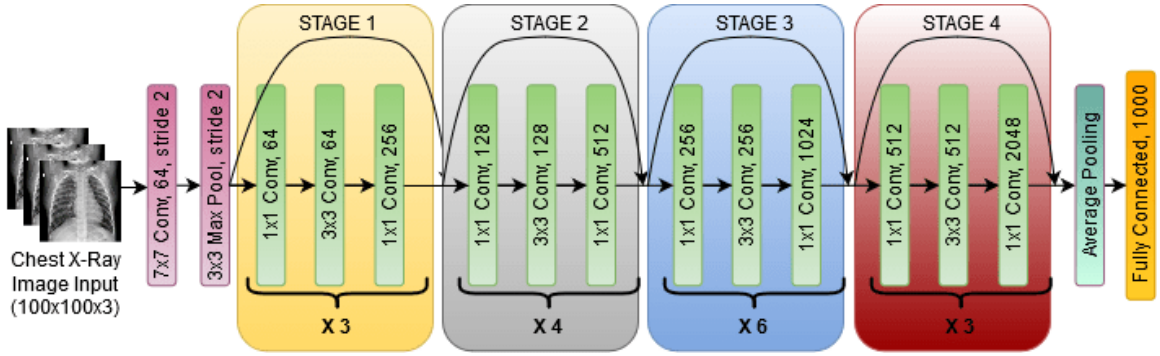
Graph Neural Network (GNN)

Evolving of Hand-design Architectures

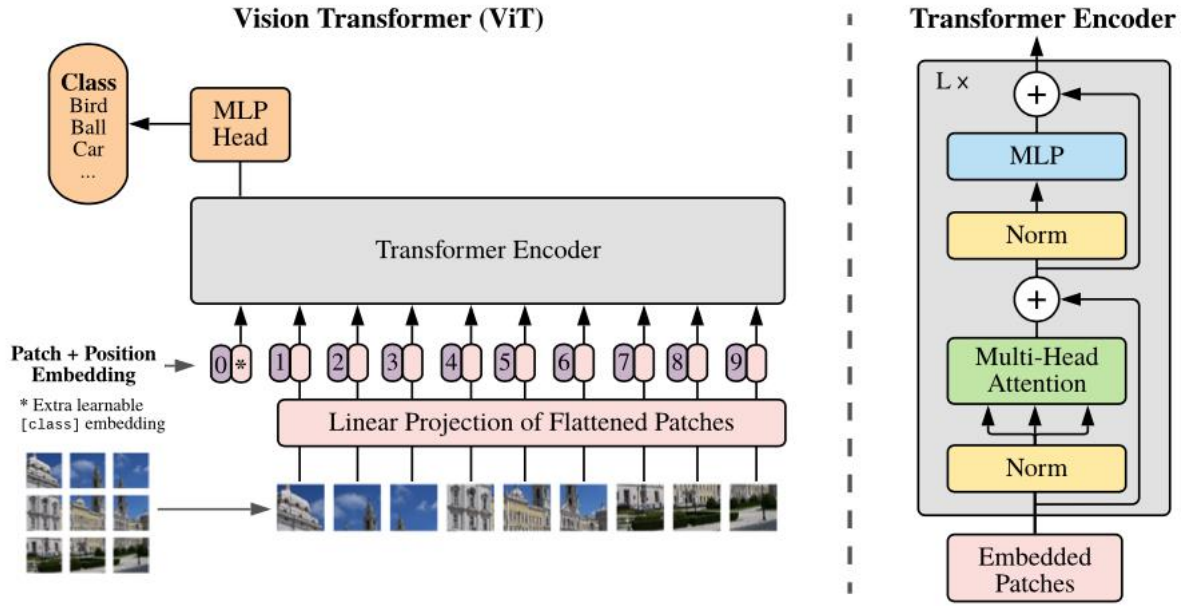
Architectures for image classification



AlexNet, 2012



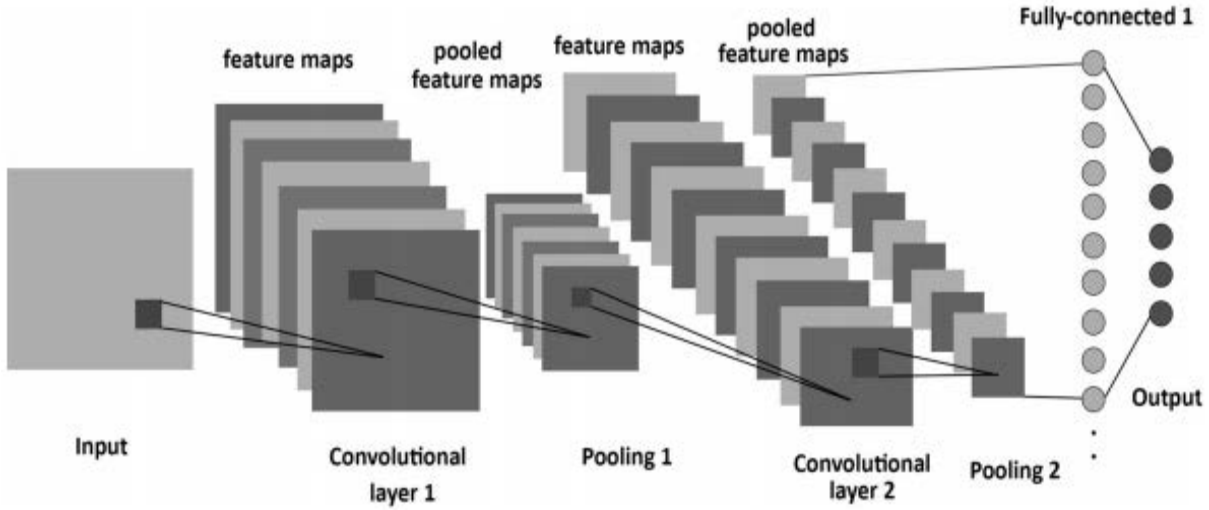
ResNet, 2015



ViT, 2020

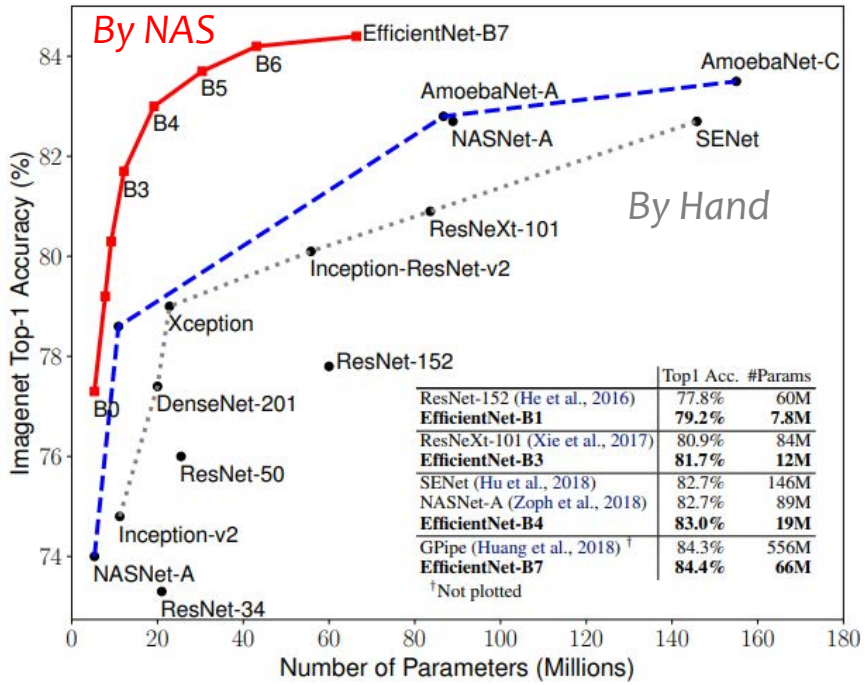
Why Need to Search Architectures?

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



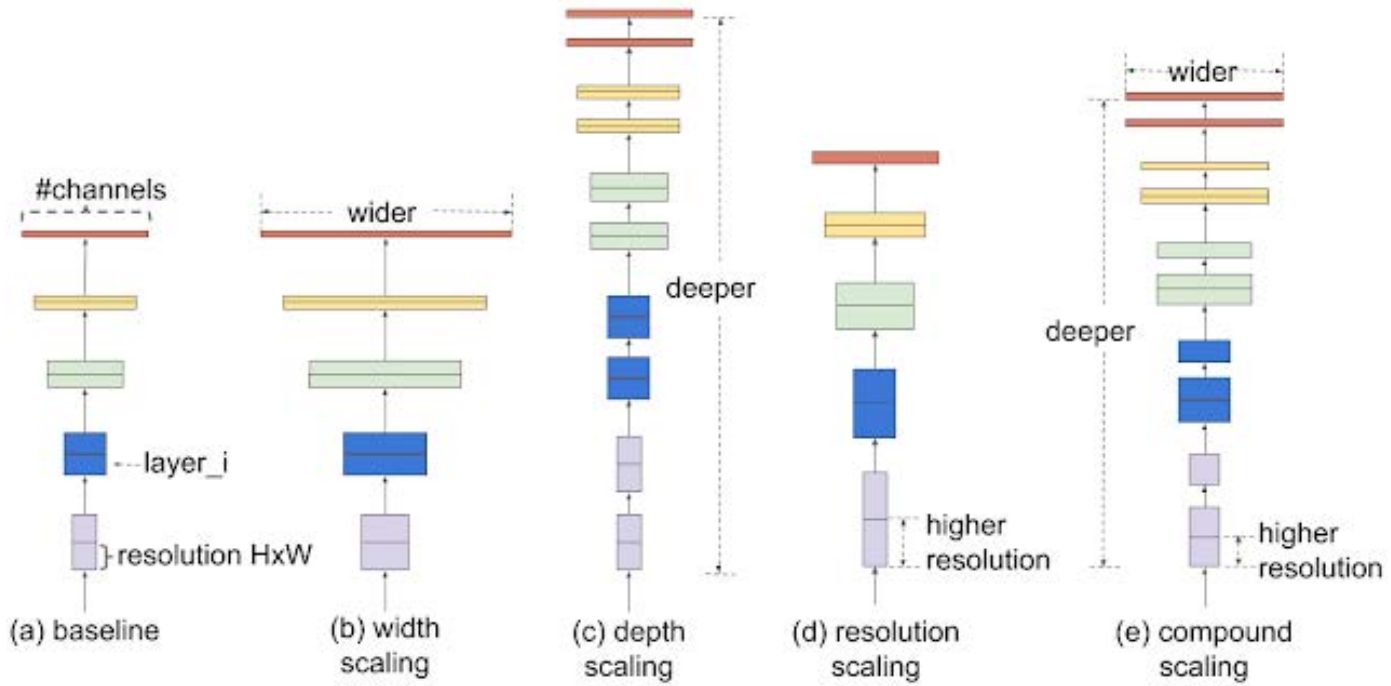
Much better than hand-designed ones

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

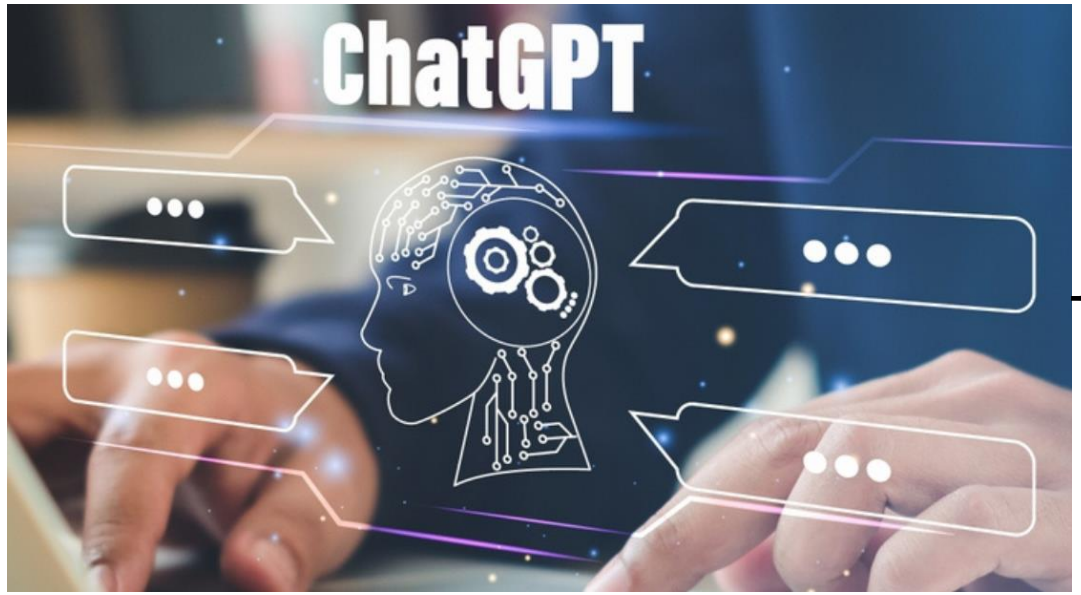
Any Changed with Large Language Model?

- NAS for LLM: model scaling

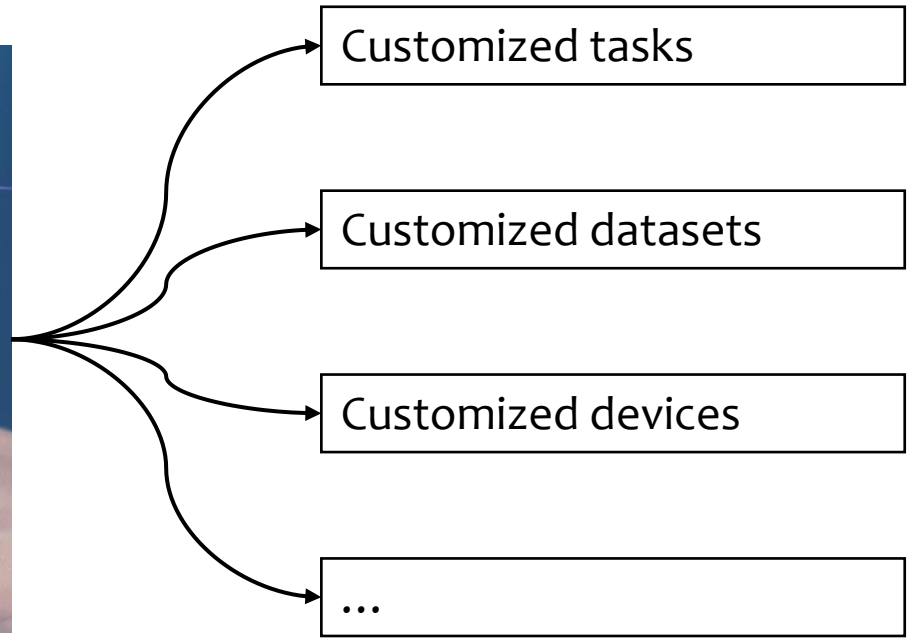


Any Changed with Large Language Model?

- LLM for NAS: domain customization



A pre-trained large model



Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
- Summary

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 Strategy

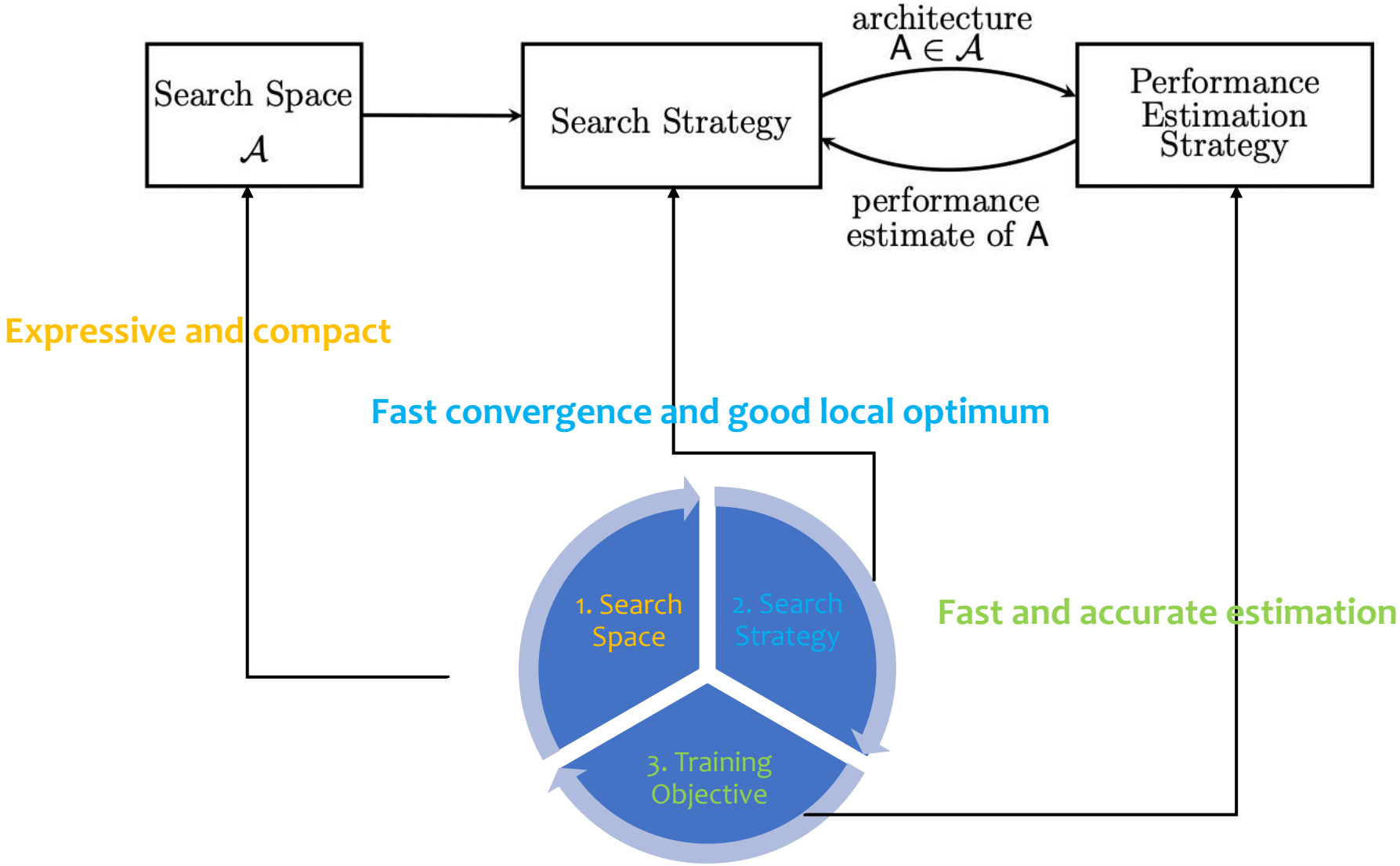
$$\text{s.t.} \quad \min_{w} L(F(w; \lambda), D_{\text{tra}})$$

Training Objective

2. Design or select proper search algorithm

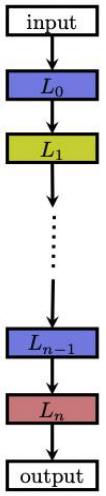
- Reduce model training cost (time to get w^*)

Three Components in NAS



Pioneer Work of NAS (RL-NAS)

1. Search space: the set of candidate architectures constructed based on the designed backbone. (5^L)



Filter height
Filter width
Stride height
Stride width
Filter number

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

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:

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

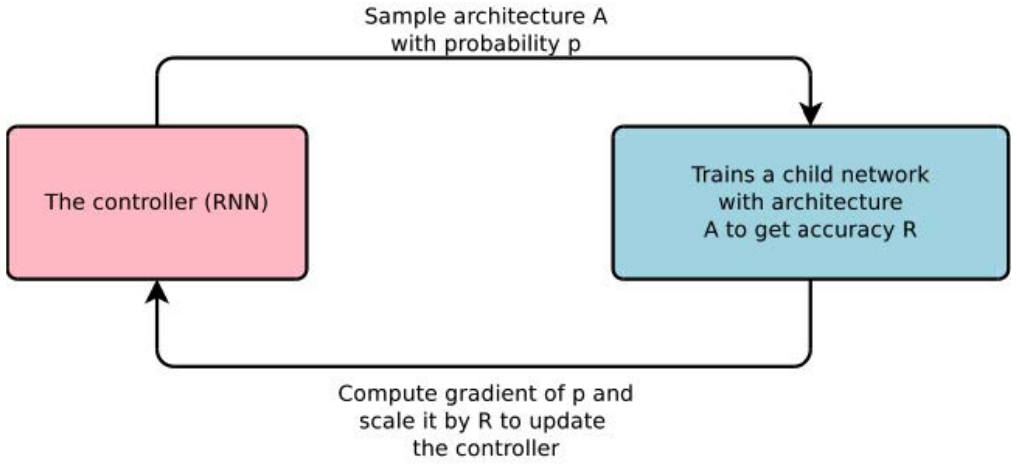
$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c) R_k$$

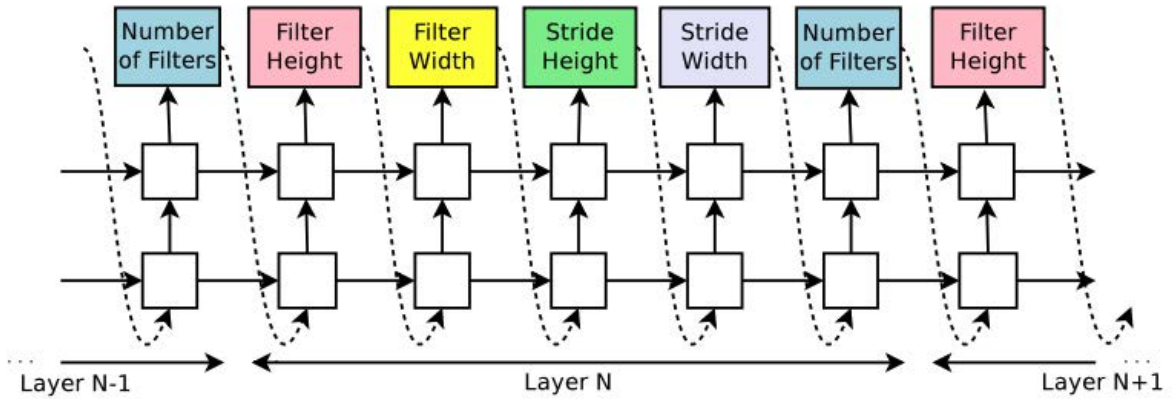
$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

3. Performance estimation: train from scratch to convergence.

Neural Architecture Search with Reinforcement Learning, 2017

RL-NAS Pipeline





Neural Architecture Search with Reinforcement Learning, 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

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

Neural Architecture Search with Reinforcement Learning, 2017

Recap

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

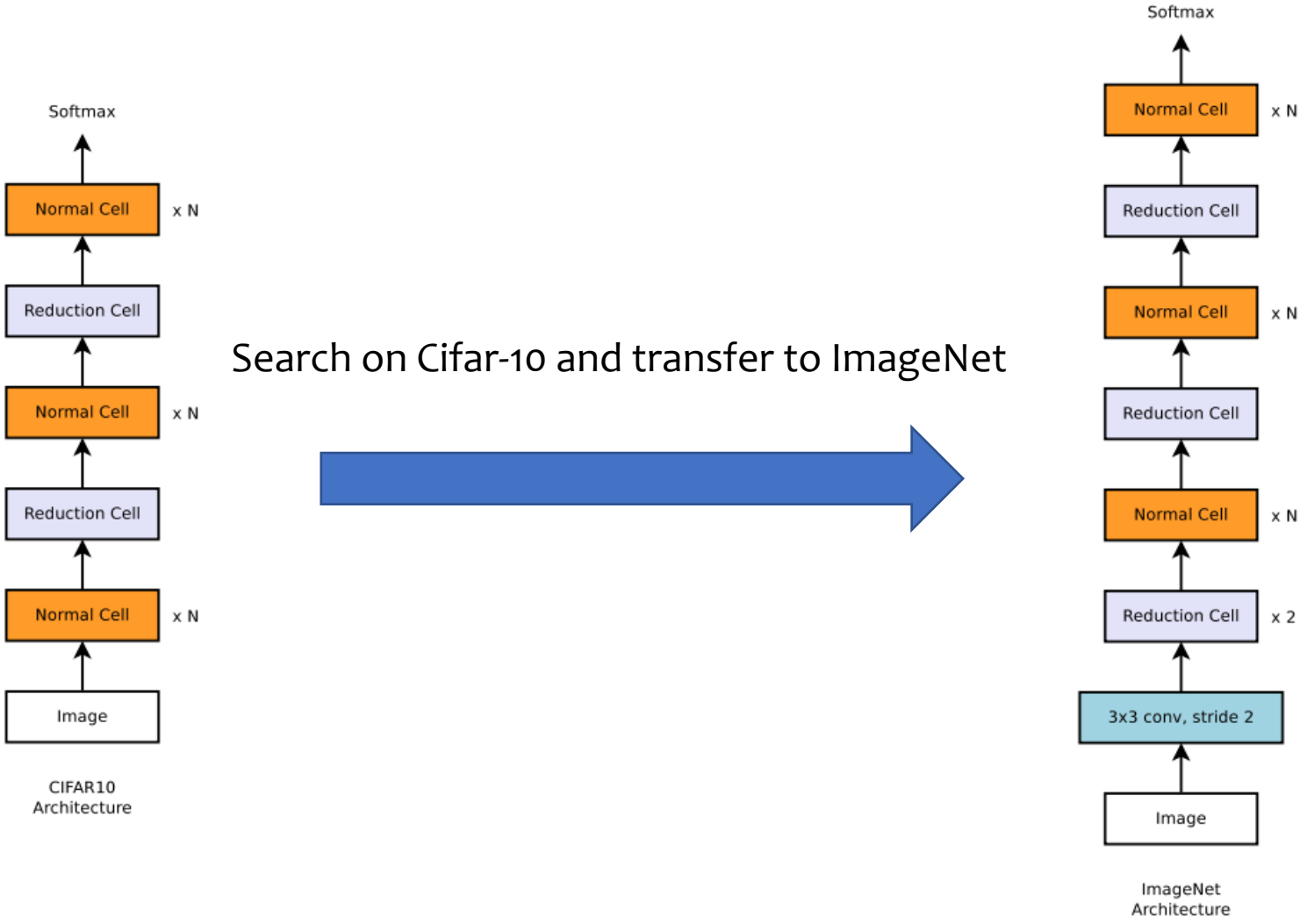
NASNet: Making NAS Possible on ImageNet

Compare to RL-NAS:

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

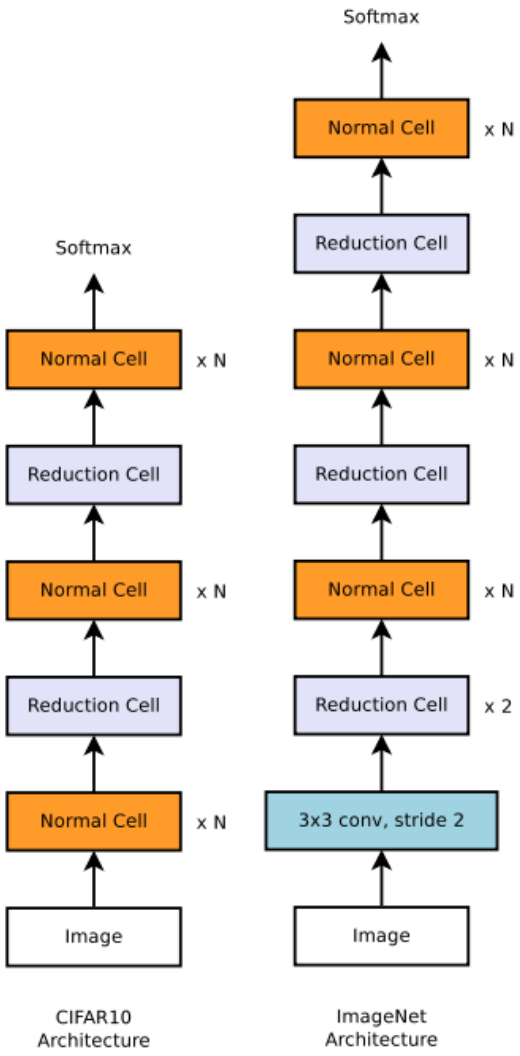
Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Transferable Search Space

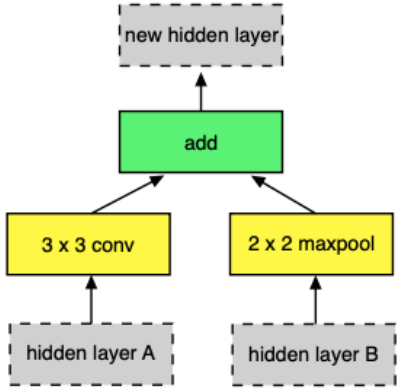


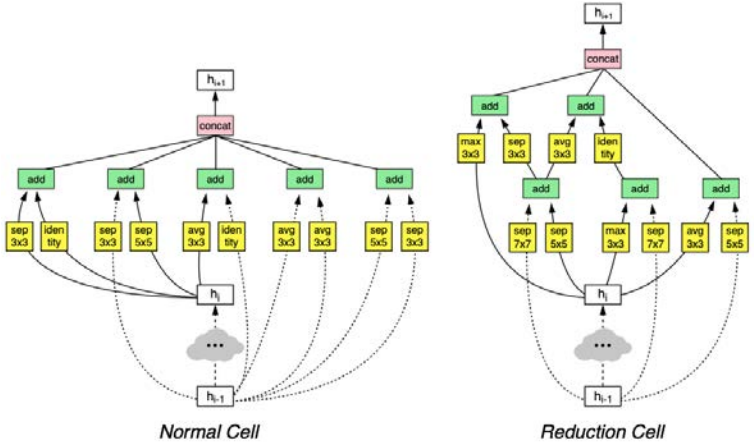
Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Improvements on Search Space



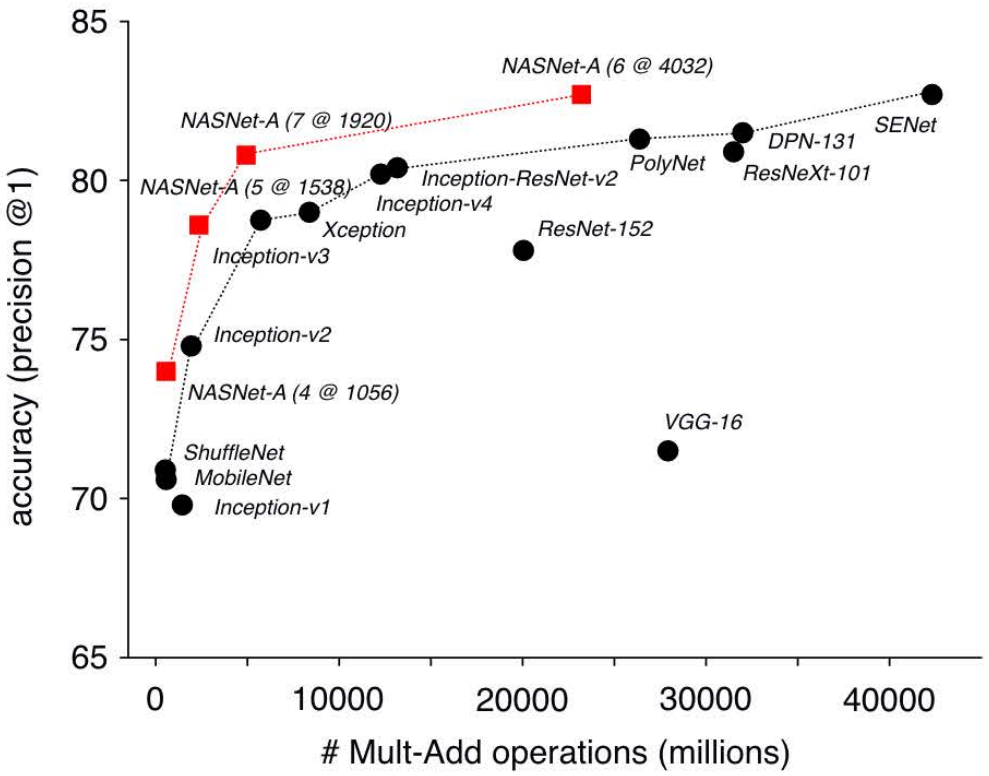
- Key idea: Search for an architectural building block and then transfer to a larger dataset.
  - Design the Normal and Reduction cell, and then stacking to obtain CNNs.
  - Each cell has B blocks, and each block has five operations
- Extracting and re-organizing the factors considering the effectiveness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

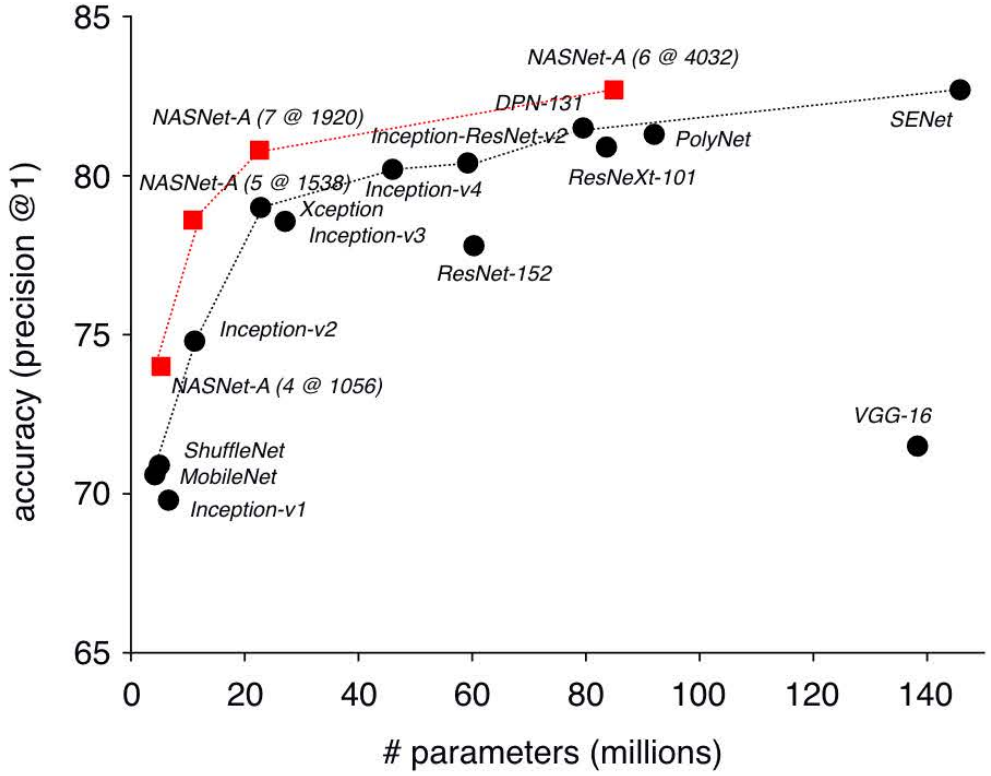




Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

NASNet: Making NAS Possible on ImageNet





Learning Transferable Architectures for Scalable Image Recognition. CVPR 2018

Recap

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

ENAS: very Fast Performance Estimation

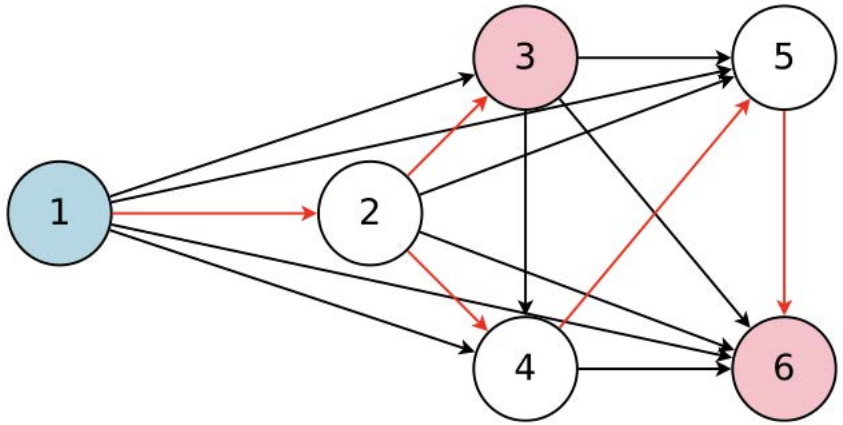
Compare to RL-NAS:

Search space: Directed Acyclic Graph
(modern choice until today)
Search strategy: Same
Training objective: 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 strategy	Evaluation
RL NAS	Variable-length string	Reinforcement learning	Train from scratch until convergence
NASNet	Block and cell		
ENAS			Weight sharing

DARTS: Differentiable NAS Starts Here

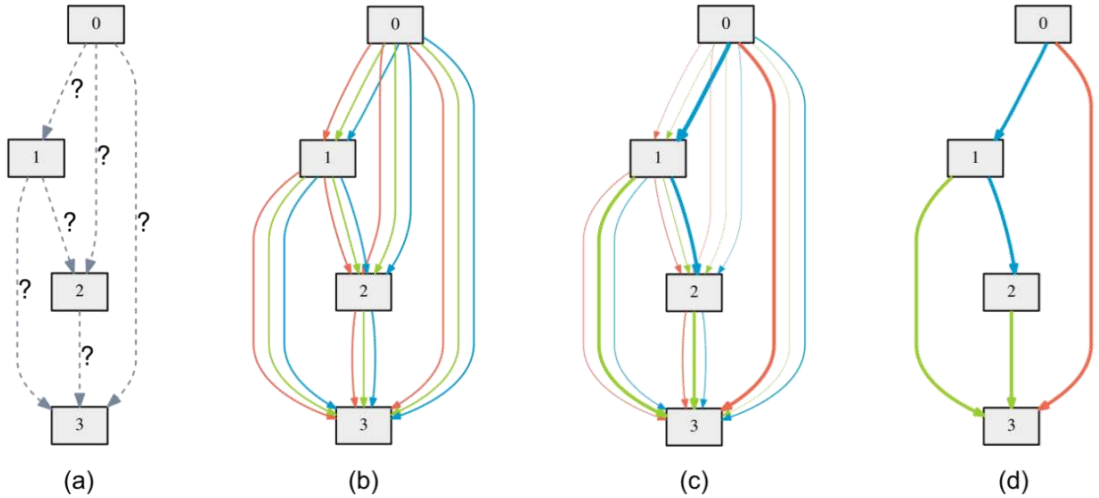
Compare to RL-NAS:

Search space: Directed Acyclic Graph with relaxed choices
Search strategy: **Differentiable is all you need**
Training objective (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)

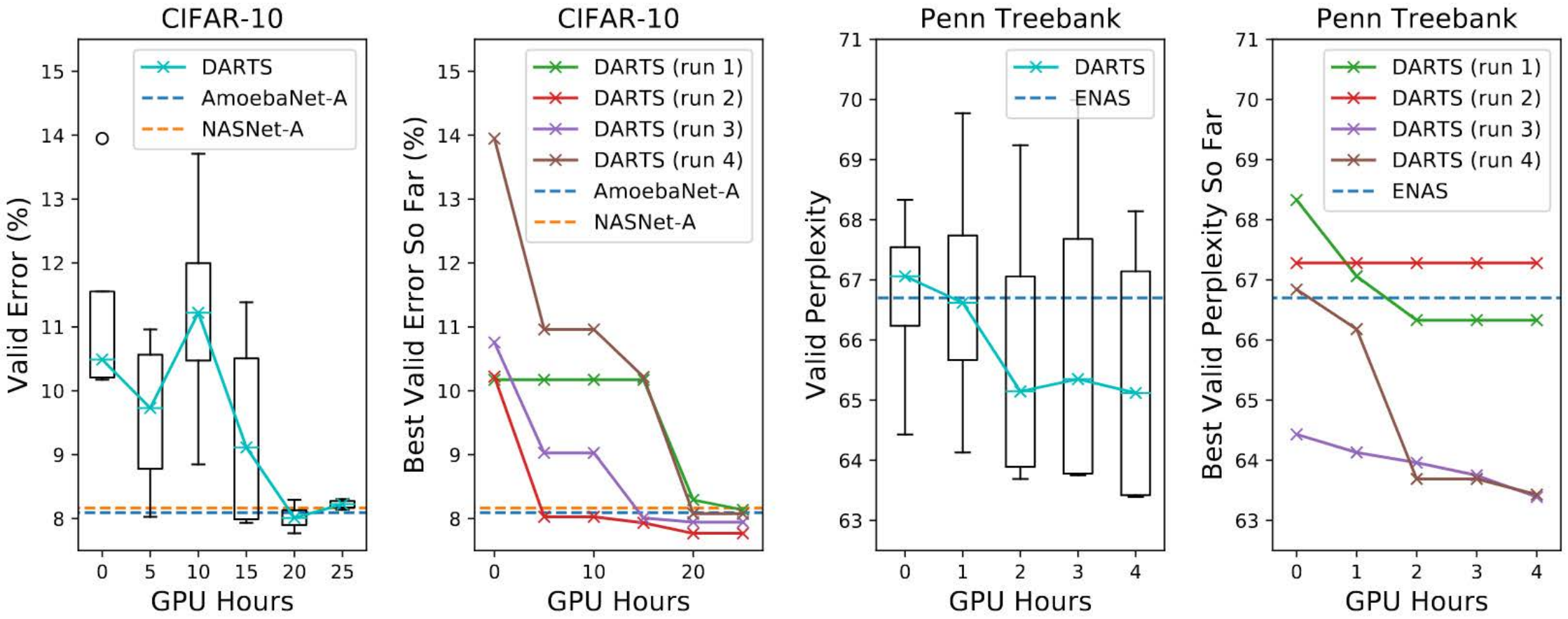
while *not converged* **do**

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: Search Convergence



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	–	$\sim$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	$\sim$225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

DARTS: Differentiable Architecture Search, ICLR 2019

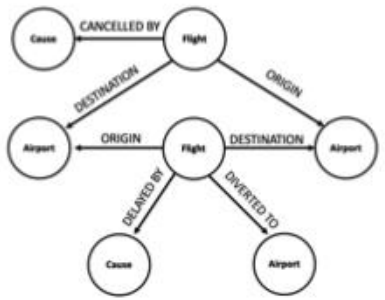
Summary

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

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
  - Background: Graph Representation Learning
  - Issue and Solution
  - Search Space: Design and Evaluation
  - Condensed Space: Construction and Evaluation
  - Summary
- NAS for GNN
- Summary

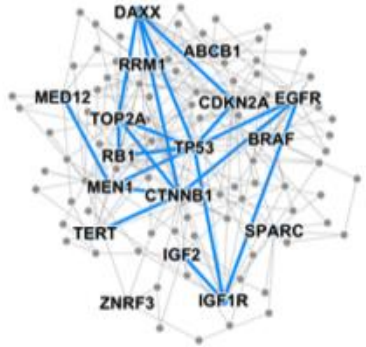
Real World Graph and Applications



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

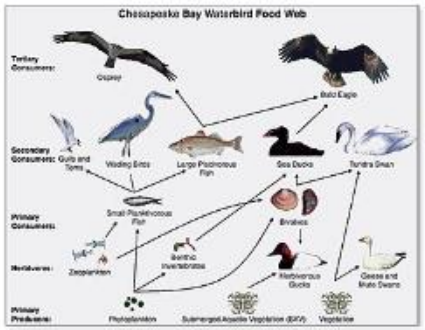


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

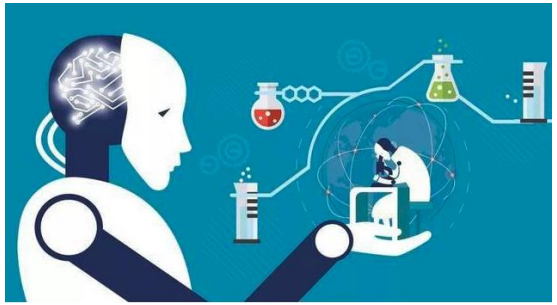
Image credit: visitlondon.com

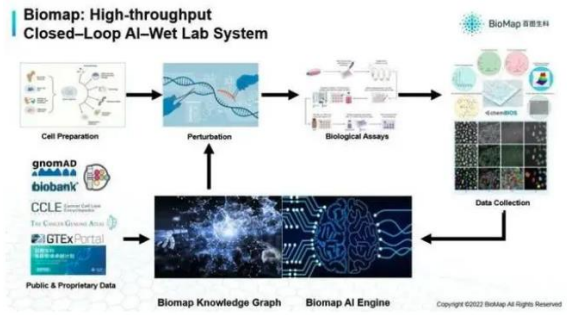
Underground Networks

Real World Graph and Applications

Different tasks on the graph

Node-level: Financial anti-fraud detection



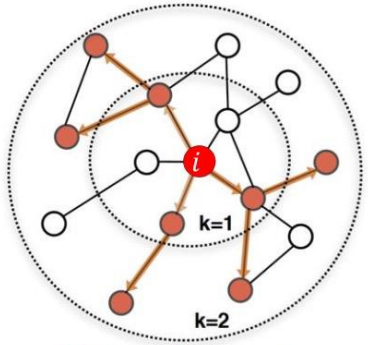


Graph-level: Molecular property prediction

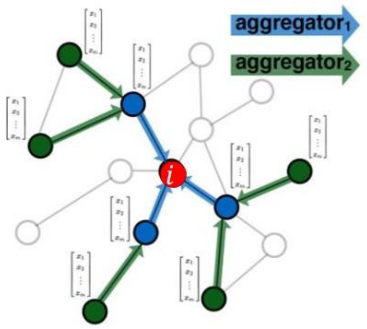
Edge-level: Recommendation system

Graph Neural Networks

- Key idea:
  - Node's neighborhood defines a computation graph
  - Learn how to **propagate information** across the graph to compute node features
- Generate node embeddings based on **local network neighborhoods, i.e.,** aggregate information from their neighbors using neural networks

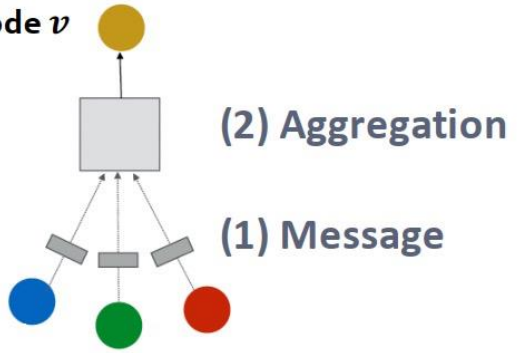


Determine node computation graph



Propagate and transform information

Graph Neural Networks



Message passing function

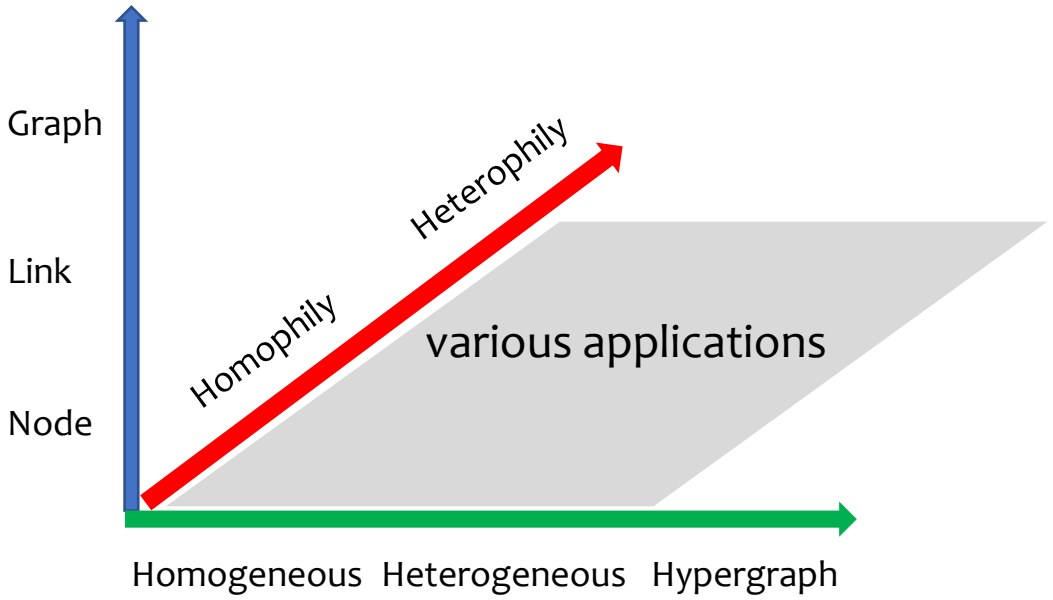
- Input: node features of v and its neighbors
- Message function: $m_u^{(l)} = \mathrm{MSG}^{(l)}\left(h_u^{(l-1)}\right)$, e.g., A linear layer $m_u^{(l)} = \mathrm{W}^{(l)} h_u^{(l-1)}$
- Aggregation function: $h_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{m_u^{(l)}, u \in N(v)\right\}\right)$, e.g., Sum/Mean/Max

Method	Message	Aggregate	Total
GCN	$m_u^{(l+1)} = \frac{1}{\lvert N(v)\rvert} W^{(l)} h_u^{(l)}$	Sum	$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \frac{1}{\lvert N(v)\rvert} h_u^{(l)}\right)$
GraphSAGE	Sum/Mean/ Max/LSTM	Sum/Mean/ Max/LSTM	$h_v^{(l+1)} = \sigma\left(W^{(l)} \mathrm{CONCAT}\left(h_v^{(l)}, \mathrm{AGG}\left(\left\{h_u^{(l)}, u \in N(v)\right\}\right)\right)\right)$
GAT	Attention weighted	Sum	$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in N(v)} \alpha_{vu} h_u^{(l)}\right)$

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
  - Background: Graph Representation Learning
  - Issue and Solution
  - Search Space: Design and Evaluation
  - Condensed Space: Construction and Evaluation
  - Summary
- NAS for GNN
- Summary

Challenges of GNN Architecture Design



Heterophily → Homophily

Table 5: Real data: mean accuracy ± stdev over different data splits. Best model per benchmark highlighted in gray. The "*" results are obtained from [26] and "N/A" denotes non-reported results.

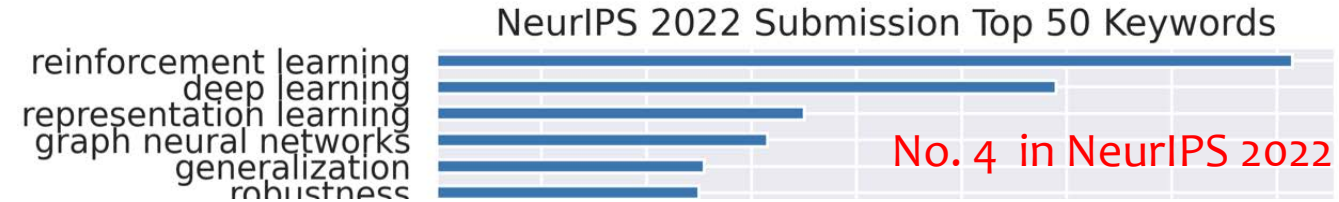
	Texas	Wisconsin	Actor	Squirrel	Chameleon	Cornell	Cora Full	Citeseer	Pubmed	Cora
Hom. ratio h	**0.11**	**0.21**	**0.22**	**0.22**	**0.23**	**0.3**	**0.57**	**0.74**	**0.8**	**0.81**
#Nodes $\|\mathcal{V}\|$	183	251	7,600	5,201	2,277	183	19,793	3,327	19,717	2,708
#Edges $\|\mathcal{E}\|$	295	466	26,752	198,493	31,421	280	63,421	4,676	44,327	5,278
#Classes $\|\mathcal{Y}\|$	5	5	5	5	5	5	70	7	3	6
H_2GCN-1	84.86±6.77	86.67±4.69	35.86±1.03	36.42±1.89	57.11±1.58	82.16±4.80	68.13±0.49	77.07±1.64	89.40±0.34	86.92±1.37
H_2GCN-2	82.16±5.28	85.88±4.22	35.62±1.30	37.90±2.02	59.39±1.98	82.16±6.00	69.05±0.37	76.88±1.77	89.59±0.33	87.81±1.35
GraphSAGE	82.43±6.14	81.18±5.56	34.23±0.99	41.61±0.74	58.73±1.68	75.95±5.01	65.14±0.75	76.04±1.30	88.45±0.50	86.90±1.04
GCN-Cheby	77.30±4.07	79.41±4.46	34.11±1.09	43.86±1.64	55.24±2.76	74.32±7.46	67.41±0.69	75.82±1.53	88.72±0.55	86.76±0.95
MixHop	77.84±7.73	75.88±4.90	32.22±2.34	43.80±1.48	60.50±2.53	73.51±6.34	65.59±0.34	76.26±1.33	85.31±0.61	87.61±0.85
GraphSAGE+JK	83.78±2.21	81.96±4.96	34.28±1.01	40.85±1.29	58.11±1.97	75.68±4.03	65.31±0.58	76.05±1.37	88.34±0.62	85.96±0.83
Cheby+JK	78.38±6.37	82.55±4.57	35.14±1.37	45.03±1.73	63.79±2.27	74.59±7.87	66.87±0.29	74.98±1.18	89.07±0.30	85.49±1.27
GCN+JK	66.49±6.64	74.31±6.43	34.18±0.85	40.45±1.61	63.42±2.00	64.59±8.68	66.72±0.61	74.51±1.75	88.41±0.45	85.79±0.92
GCN	59.46±5.25	59.80±6.99	30.26±0.79	36.89±1.34	59.82±2.58	57.03±4.67	68.39±0.32	76.68±1.64	87.38±0.66	87.28±1.26
GAT	58.38±4.45	55.29±8.71	26.28±1.73	30.62±2.11	54.69±1.95	58.92±3.32	59.81±0.92	75.46±1.72	84.68±0.44	82.68±1.80
GEOM-GCN*	67.57	64.12	31.63	38.14	60.90	60.81	N/A	77.99	90.05	85.27
MLP	81.89±4.78	85.29±3.61	35.76±0.98	29.68±1.81	46.36±2.52	81.08±6.37	58.76±0.50	72.41±2.18	86.65±0.35	74.75±2.22

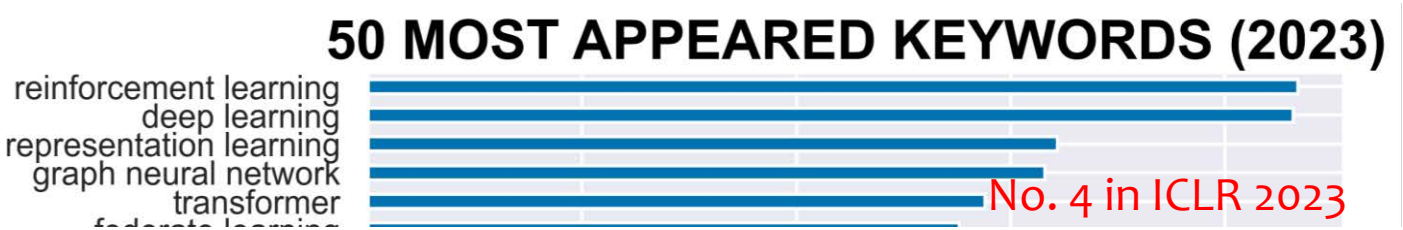
Simple MLP is even better than GNN

Due to the diversity and complexity of graph-based tasks, they have different preferences when designing GNNs.

Zhu et al. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. NeurIPS 2020

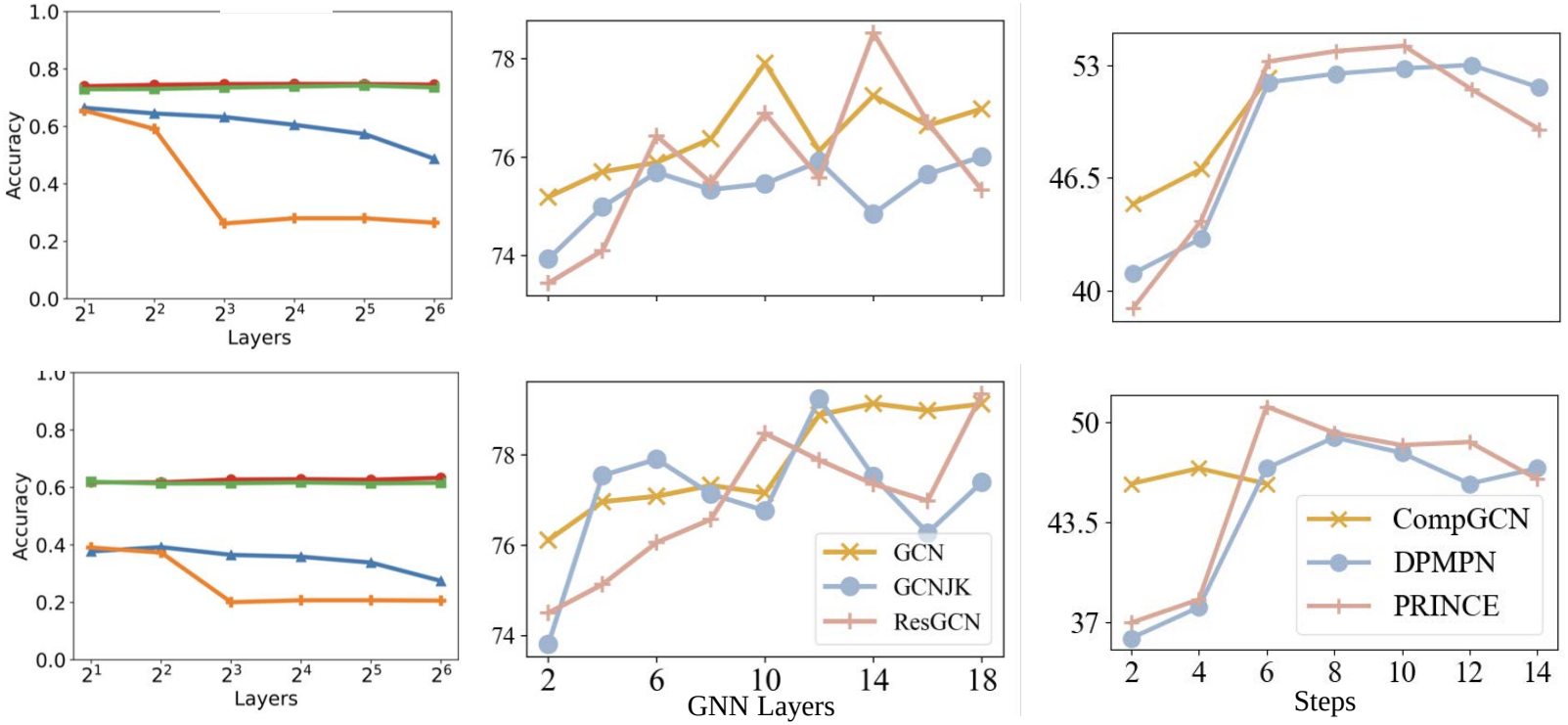
Challenges of GNN Architecture Design





- GNNs becomes popular in recent years.
- Diverse GNNs are provided.
- How to design suitable GNN when considering diverse data and task?

Challenges of GNN Architecture Design



(a) Node classification task (b) Graph classification task (c) Link Prediction task

- Going deeper not always with better performance.
- No (converged) winning architecture on all model depth.
- How to design expressive GNNs with limited layers?

Explore the GNN design space

Explore the GNN Design Space

Challenges in exploring the design space of GNN:

- How to construct a general search space for GNN?
- How to evaluating the search space on diverse graphs?
- Target to design compact and expressive search space.

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
  - Background: Graph Representation Learning
  - Issue and Solution
  - Search Space: Design and Evaluation
  - Condensed Space: Construction and Evaluation
  - Summary
- NAS for GNN
- Summary

GNN Design Space

The common senses when designing GNNs:

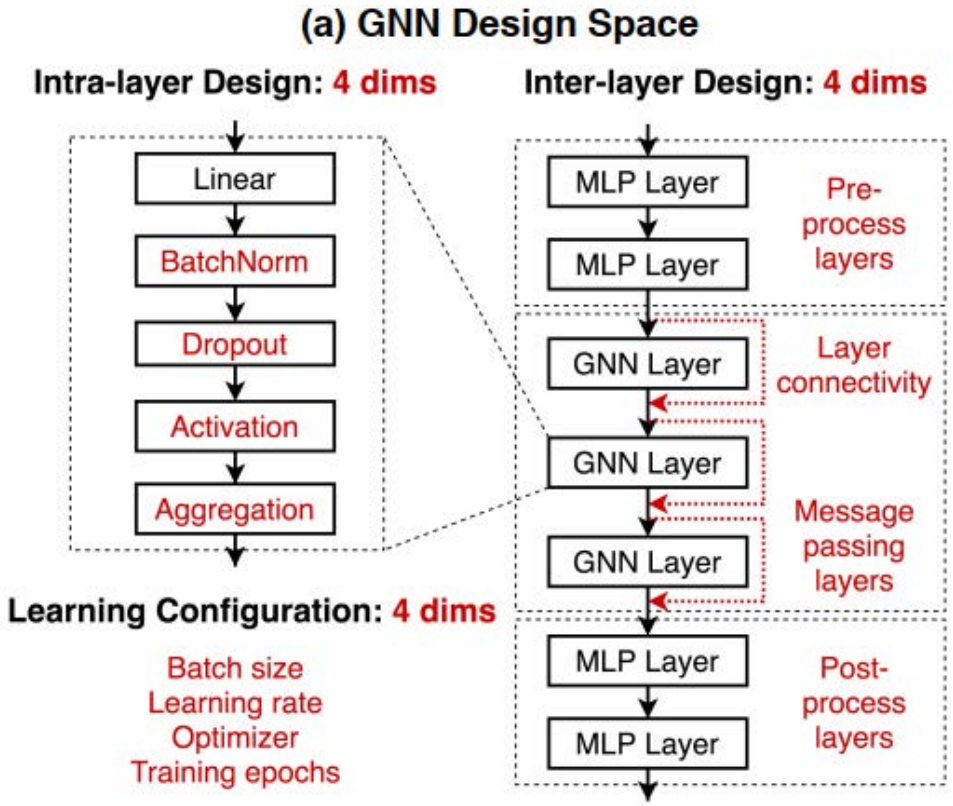
- Intra-layer: designed under the message passing scheme

$$m_u^{(l)} = \mathrm{MSG}^{(l)}\left(h_u^{(l-1)}\right)$$
$$h_v^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{m_u^{(l)}, u \in N(v)\right\}\right)$$

- Inter-layer: utilizing the representations in intermediate layers to improve effectiveness.
- Learning configuration: designing the hyper-parameters when training GNNs towards better performance.

GNN Design Space

Space size: 314,928



Pre-process Layer	1, 2, 3
MPNN Layer	2, 4, 6, 8
Layer connectivity	SUM, CAT, Stack
Post-process Layer	1, 2, 3
BN	True, False
Dropout	0, 0.3, 0.6
Activation	Relu, Prelu, Swish
Aggregation	Mean, Max, Sum
Batch size	16, 32, 64
Learning rate	0.1, 0.01, 0.001
Optimizer	SGD, ADAM
Train epochs	100, 200, 400

Architecture backbone

GNN Layer

Hyper-parameters

GNN Layer: $\mathbf{h}_v^{(k+1)} = \text{AGG}\left(\left\{\text{ACT}\left(\text{DROPOUT}\left(\text{BN}(\mathbf{W}^{(k)}\mathbf{h}_u^{(k)} + \mathbf{b}^{(k)})\right)\right), u \in \mathcal{N}(v)\right\}\right)$

Space Evaluation - Data Preparation

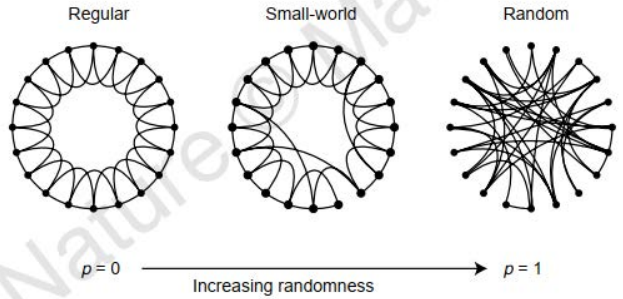
- Target: effectively evaluate the space with medium- sized, diverse and realistic datasets.
- Multiple tasks and diverse datasets from real-world and synthetic ones are selected.
  - 18 node classification tasks (12 synthetic + 6 real-world)
  - 14 graph classification tasks (8 synthetic + 6 real-world)

node-AmazonComputers-N/A-N/A	graph-PROTEINS-N/A-N/A
node-AmazonPhoto-N/A-N/A	graph-BZR-N/A-N/A
node-CiteSeer-N/A-N/A	graph-COX2-N/A-N/A
node-CoauthorCS-N/A-N/A	graph-DD-N/A-N/A
node-CoauthorPhysics-N/A-N/A	graph-ENZYMES-N/A-N/A
node-Cora-N/A-N/A	graph-IMDB-N/A-N/A
node-scalefree-clustering-pagerank	graph-scalefree-clustering-path
node-scalefree-const-clustering	graph-scalefree-const-path
node-scalefree-const-pagerank	graph-scalefree-onehot-path
node-scalefree-onehot-clustering	graph-scalefree-pagerank-path
node-scalefree-onehot-pagerank	graph-smallworld-clustering-path
node-scalefree-pagerank-clustering	graph-smallworld-const-path
node-smallworld-clustering-pagerank	graph-smallworld-onehot-path
node-smallworld-const-clustering	graph-smallworld-pagerank-path
node-smallworld-const-pagerank	
node-smallworld-onehot-clustering	
node-smallworld-onehot-pagerank	
node-smallworld-pagerank-clustering	

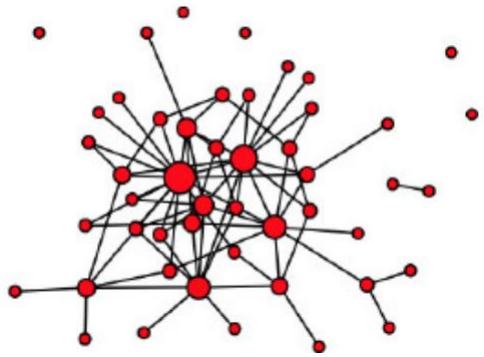
Space Evaluation - Data Preparation

- Synthetic tasks
  - Generate datasets with diverse graph structural properties, features, and labels.
  - Graph structures
    - Construct graphs with different Average Clustering Coefficient C and Average Path Length L.
    - 4 Small-world graphs[1] and scale-free graphs[2] in each grid. (Two families of graphs with diverse structural properties)
  - Features

    Constant scalar / One-hot / Clustering Coefficient / PageRank score
  - Labels
    - Node level: PageRank score / Clustering Coefficient
    - Graph level: Average Path Length



Small-world graphs: Designed based on social networks, and most neighboring nodes can be reached from every other node by a small number of hops or steps.



Scale-free graphs: Designed based on power law. Most nodes have very few neighbors, while a few important nodes have a huge number of connections.

[1] Collective dynamics of 'small-world'networks. Nature, 393(6684):440, 1998.
[2] Growing scale-free networks with tunable review E, 65(2):026107, 2002. clustering. Physical
[3] http://www.jsums.edu/nmeghanathan/files/2015/08/CSC-641-Fall2015-Module-5-Scale-free-Networks.pdf?x61976

Space Evaluation – Evaluation Strategy

- Controlled random search
  - Fix one dimension, draw $S = 96$ random designs on the other dimensions.
  - Rank all models over 96 setups.

(a) Controlled Random Search

GNN Design Space					GNN Task Space	
BatchNorm	Activation	…	Message layers	Layer Connectivity	Task level	dataset
True	relu	…	8	skip_sum	node	CiteSeer
False	relu	…	8	skip_sum	node	CiteSeer
True	relu	…	2	skip_cat	graph	BZR
False	relu	…	2	skip_cat	graph	BZR
…						
True	prelu	…	4	stack	graph	scale free
False	prelu	…	4	stack	graph	scale free

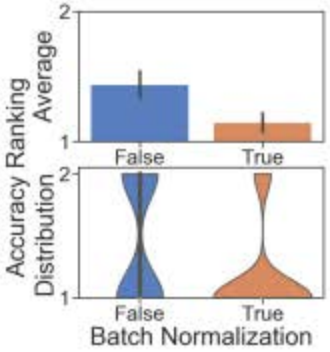
96×2

11 search dimensions

(b) Rank Design Choices by Performance

Experimental Results	
Val. Accuracy	Design Choice Ranking
0.75	1
0.54	2
0.88	1 (a tie)
0.86	1 (a tie)
0.89	1
0.36	2

(c) Ranking Analysis



Space Evaluation – Results

The performance rankings in each design dimension

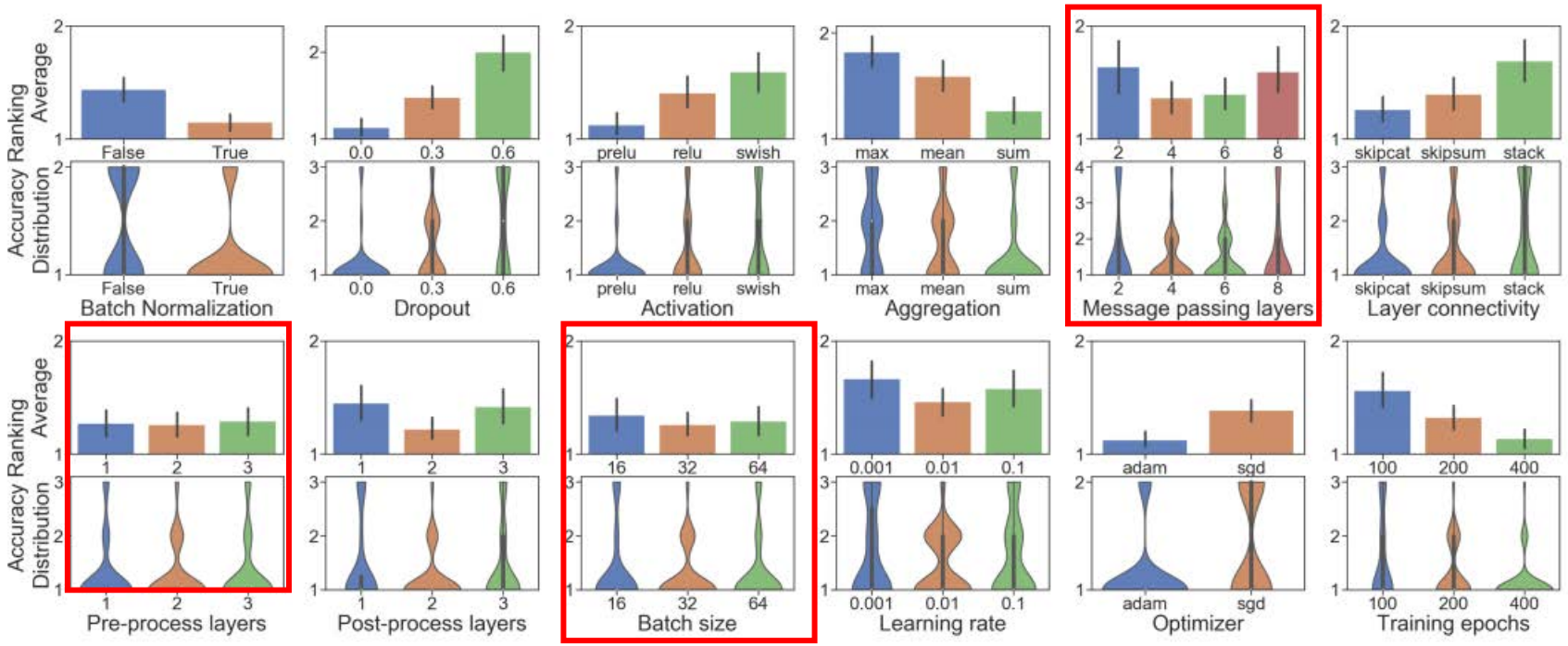


Figure 3: **Ranking analysis for GNN design choices in all 12 design dimensions**. Lower is better. A tie is reached if designs have accuracy / ROC AUC differences within $\epsilon = 0.02$.

- BN is helpful.
- Remove dropout
- Use PRelu activation
- Sum aggregator. (Ref to GIN.)
- Concatenate all layers embeddings.
- Adam optimizer.
- 400 training epochs.

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
  - Background: Graph Representation Learning
  - Issue and Solution
  - Search Space: Design and Evaluation
  - Condensed Space: Construction and Evaluation
  - Summary
- NAS for GNN
- Summary

Condensed Design Space

- Construct the condensed space which may generally perform well on all datasets.
  - The top-ranked operations are preserved.
  - The comparable operations are all preserved.

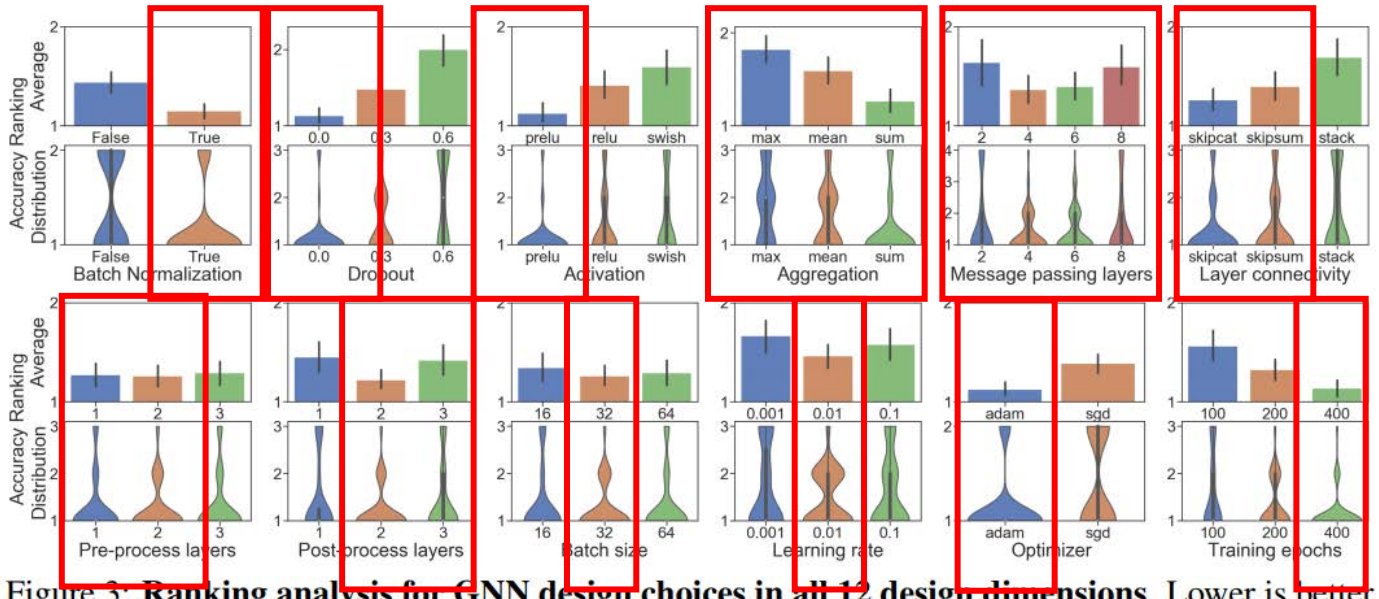
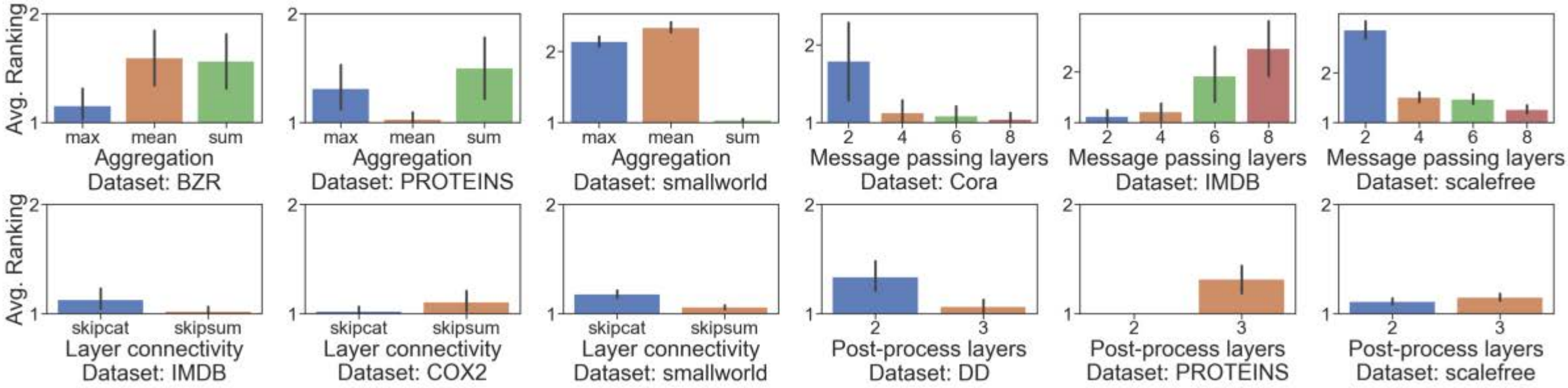


Figure 3: **Ranking analysis for GNN design choices in all 12 design dimensions**. Lower is better. A tie is reached if designs have accuracy / ROC AUC differences within $\epsilon = 0.02$.

Condensed Design Space

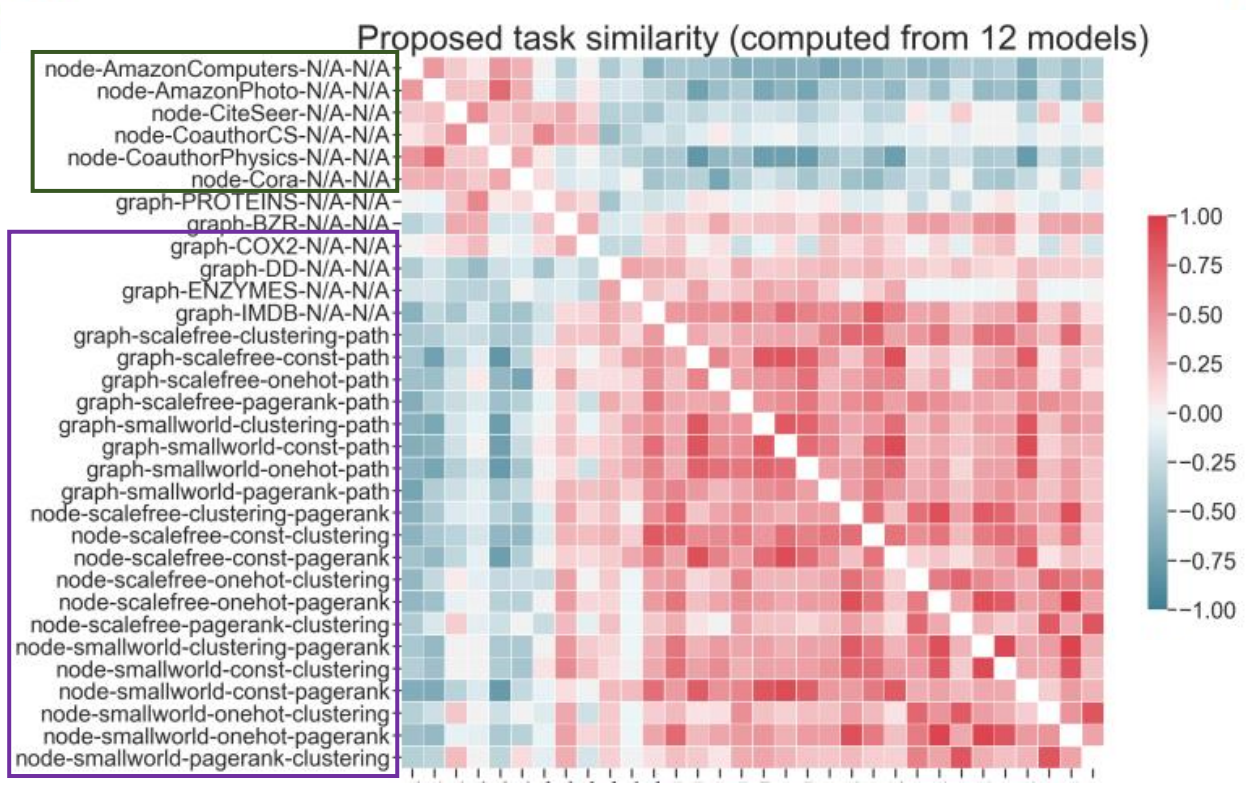
Evaluating the condensed space:

- Operation dimensions drastically vary across tasks. (NAS + data-specific.)
- The information gain from novel tasks learning may not work.



Evaluations on Dataset Similarity

- Select 12 architectures based on condensed space, and then calculate the rank on all datasets.
- Calculate Kendall rank correlations between each dataset pair.
- Format of tasks: level-dataset-feature-label.



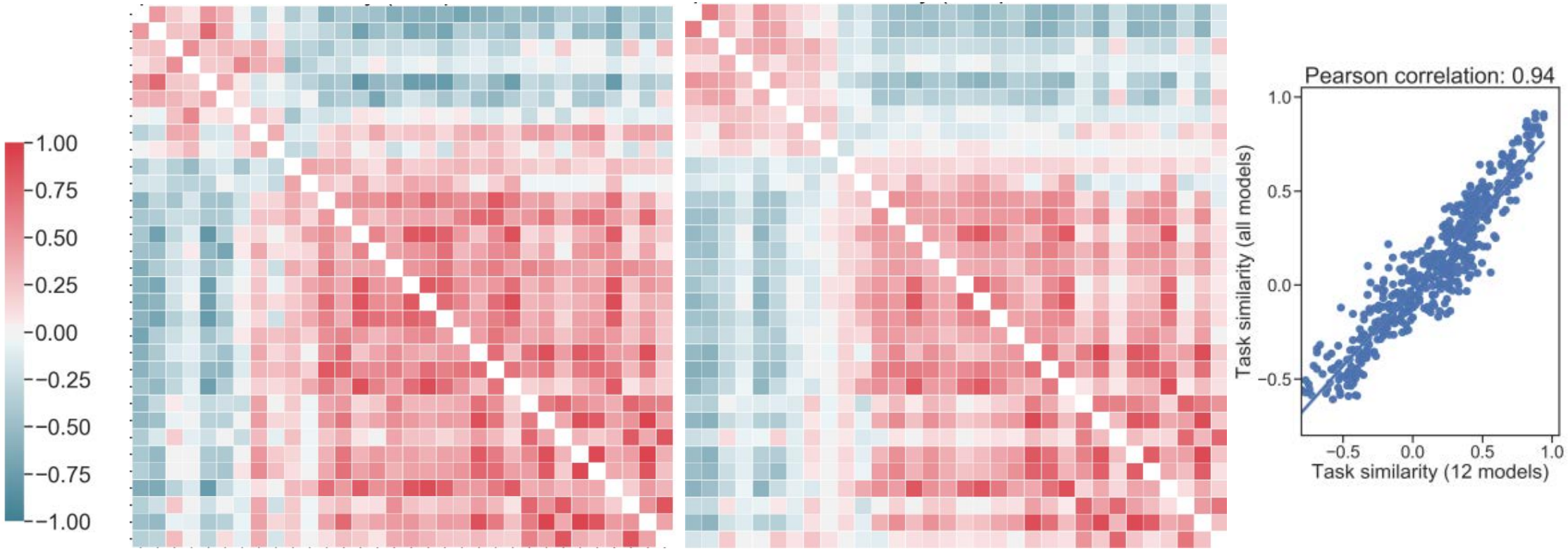
Datasets roughly clustered into 2 clusters:

- Real-world node classification datasets
- Synthetic node classification dataset and all graph classification dataset.

Evaluations on Dataset Similarity

Evaluate the influence of architecture size.

Using anchor architectures is efficient and sufficient.



a) Task similarity (12 anchors)

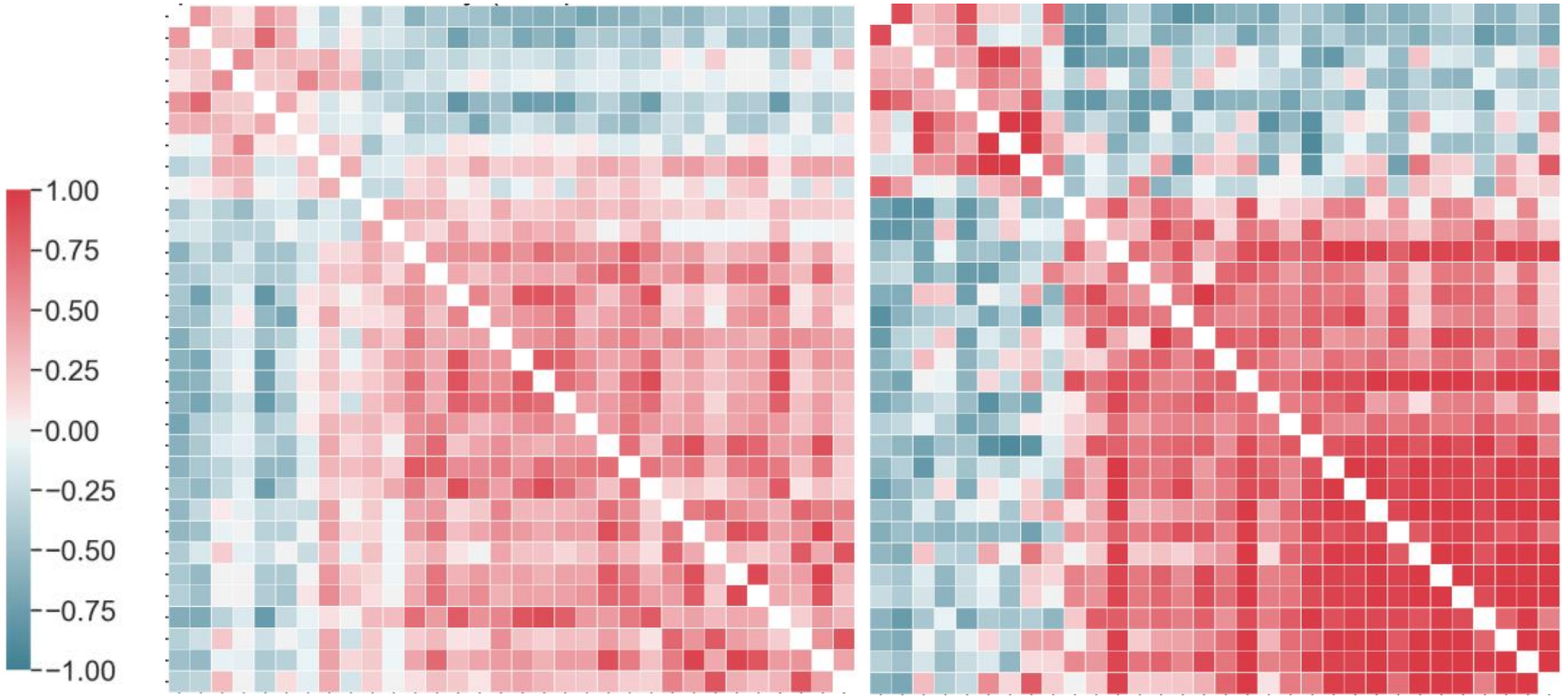
b) Task similarity (96 full models)

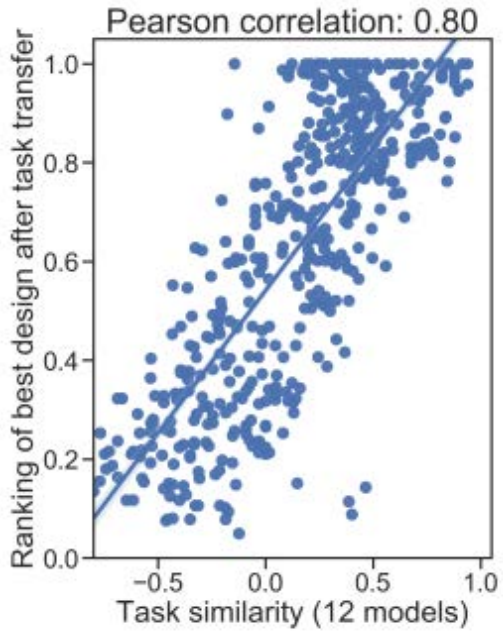
c) Similarity correlation

Evaluations on Transfer Learning

Evaluate the correlations between dataset similarity and transfer performance.

Transfer learning between similar datasets will obtain better performance.





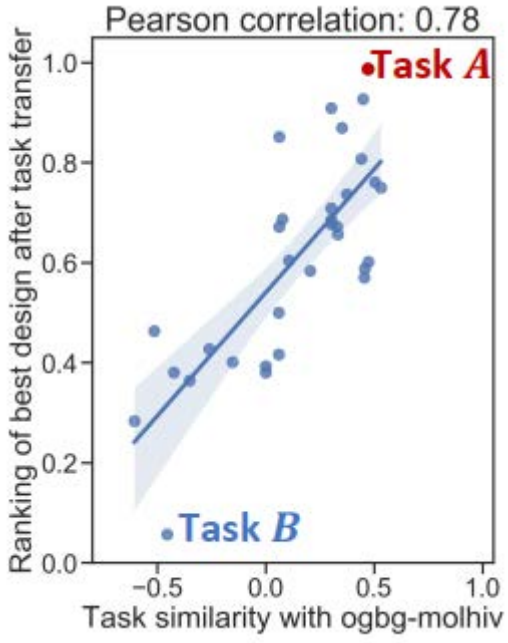
a) Task similarity (12 anchors)

b) Transfer performance(normalized) ranking

c) Similarity correlation

Transfer into OGB Dataset

- Medium-sized datasets employed in this paper. Try to transfer SOTA architectures to large-scale datasets.
- OGB is the large-scale real-world dataset benchmark and choose OGBG-molhiv (molecular property prediction dataset) as an example.
- Better performance is obtained based on condensed space (0.785>0.771)



	Task *A*: graph-scalefree-const-path	Task *B*: node-CoauthorPhysics	Target task: ogbg-molhiv
Best design in our design space	(1, 8, 3, skipcat, sum)	(1, 4, 2, skipcat, max)	(2, 6, 3, skipcat, add)
Best design's performance	0.865	0.968	***0.792***
Previously reported SOTA	N/A	0.930	**0.771**
Task Similarity with ogbg-molhiv	0.47	-0.61	1.0
Performance after transfer to ogbg-molhiv	***0.785***	0.736	N/A

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
  - Background: Graph Representation Learning
  - Issue and Solution
  - Search Space: Design and Evaluation
  - Condensed Space: Construction and Evaluation
  - Summary
- NAS for GNN
- Summary

Summary

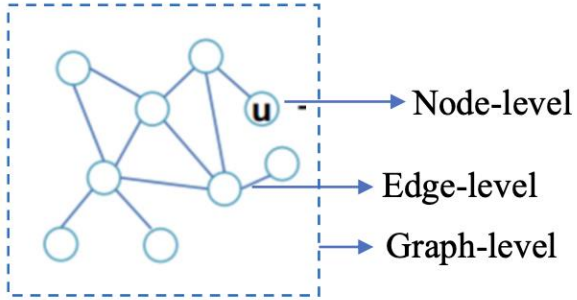
- Different data have different preferences when designing operations. Data-specific GNN is crucial given diverse real-world data.
- The search space needs to be expressive and compact, and it can be evaluated with performance ranking.
- Searching architectures on condensed space can efficiently obtain better architectures on new datasets.
- However, obtaining a general condensed search space need lots of evaluations.

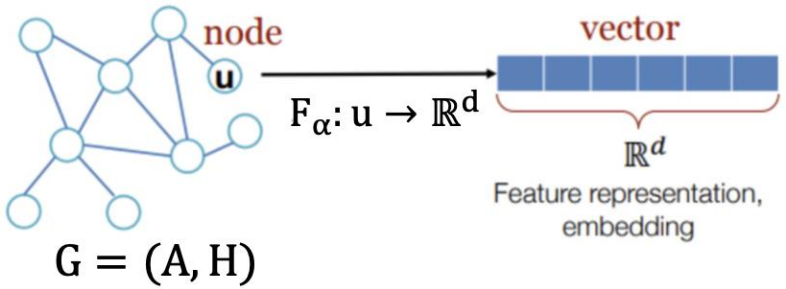
Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
  - Introduction: Graph Neural Architecture Search for Different Graph Learning Tasks
  - Node-level: GraphNAS
  - Graph-level: PAS
  - Link-level: GNAS-CF
  - Summary
- Summary

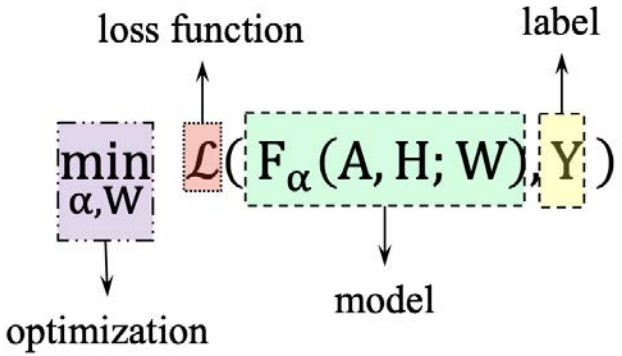
Recall: Different Graph Learning Tasks

- Graph representation learning framework with GNNs





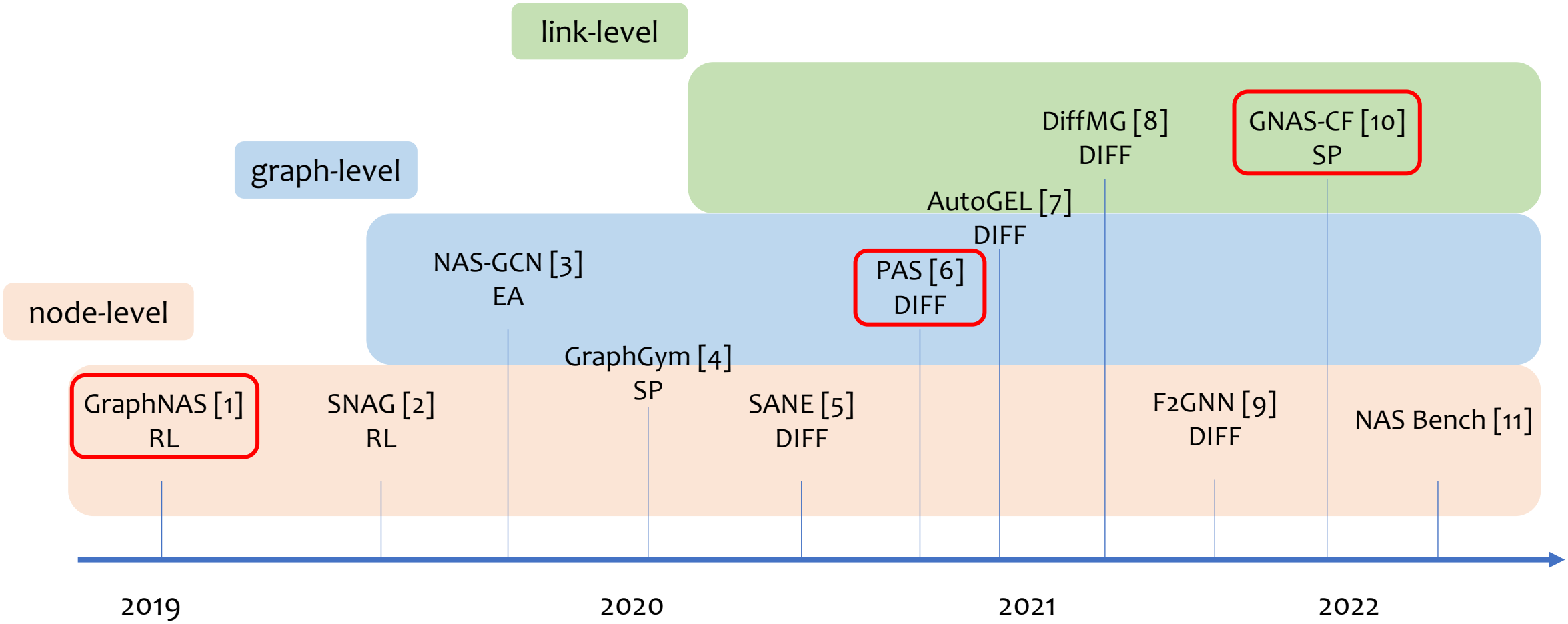
$h_u \in \mathbb{R}^d$



- **Node classification → node representation learning**
  - node label y_u
  - Mapping representations into label space $f: \mathbb{R}^d \rightarrow \mathbb{R}^C$
  - f: MLP network for example
- **Graph classification → graph pooling**
  - Graph label y_G
  - Learning the graph representation $f_{rd}: (A, H) \rightarrow \mathbb{R}^d$
  - Mapping representations into label space $f: \mathbb{R}^d \rightarrow \mathbb{R}^C$
  - f_{rd}: global sum for example $z = \sum_{u \in G} h_u$
- **Link Prediction → scoring function learning**
  - Edge label y_e
  - Learning the edge representation $f_{sf}: (h_u, h_v) \rightarrow \mathbb{R}^d$
  - Learning edge score $f: \mathbb{R}^d \rightarrow \mathbb{R}^1$
  - Predicting the existence of edge (u, v)
  - f_{sf}: concatenation for example $h = W(h_u || h_v)$

Timeline

Algorithm abbreviation
RL: reinforcement learning
DIFF: differentiable search algorithm
EA: evolutionary algorithm
SP: search space pruning



Reference

- [1] Graph Neural Architecture Search. IJCAI 2019
- [2] Simplifying Architecture Search for Graph Neural Network. Arxiv 2020
- [3] Graph neural network architecture search for molecular property prediction. Big Data 2020
- [4] Design space for graph neural networks. NeurIPS 2020
- [5] Search to aggregate neighborhood for graph neural network. ICDE 2021
- [6] Pooling Architecture Search for Graph Classification. CIKM 2021
- [7] AutoGEL: An automated graph neural network with explicit link information. NeurIPS 2021
- [8] DiffMG: Differentiable Meta Graph Search for Heterogeneous Graph Neural Networks. KDD 2021
- [9] Designing the Topology of Graph Neural Networks: A Novel Feature Fusion Perspective. WWW 2022
- [10] Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022
- [11] NAS-Bench-Graph: Benchmarking Graph Neural Architecture Search. Arxiv 2022

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
  - Introduction: Graph Neural Architecture Search for Different Graph Learning Tasks
  - Node-level: GraphNAS
  - Graph-level: PAS
  - Link-level: GNAS-CF
  - Summary
- Summary

GraphNAS

- The first graph neural architecture search method
  - Search for combinations of GNN layers
- A customized search space and RL based search algorithm.
- Outstanding performance in various settings of node classification
  - Semi-supervised
  - Supervised

Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Search Space

- Operations in the search space
  - Neighbor sampling $Smpl$
  - Message computation $Func$
  - Message aggregation $Aggr$
  - Multi-head and readout $Read$
  - Activation function σ
  - Number of multi-head k
  - Output dimension d
- An exemplar GNN layer

$$[first\_order, gat, sum, concat, 8, 16, elu]$$

Operators	Values
$Smpl$	$first\_order$
$Func$	$e_{uv}h_u$
$Aggr$	sum, $mean$, max, mlp
$Read$	avg, for the last layer $concat$, otherwise
activate function σ	$sigmoid$, $tanh$, $relu$, $identity$, $softplus$, $leaky\_relu$, $relu6$, elu
multi-head k	1, 2, 4, 6, 8, 16
output dimension d	8, 16, 32, 64, 128, 256, 512

operators in search space

e_{uv}	Values
const	$e_{uv}^{con} = 1$
gcn	$e_{uv}^{gcn} = 1/\sqrt{d_u d_v}$
gat	$e_{uv}^{gat} = leaky\_relu((W_l * h_u + W_r * h_v))$
sym-gat	$e_{uv}^{sym} = e_{vu}^{gat} + e_{uv}^{gat}$
cos	$e_{uv}^{cos} =< W_l * h_u, W_r * h_v >$
linear	$e_{uv}^{lin} = tanh(sum(W_l * h_u))$
gene-linear	$e_{uv}^{gan} = W_a * tanh(W_l * h_u + W_r * h_v)$

attention functions

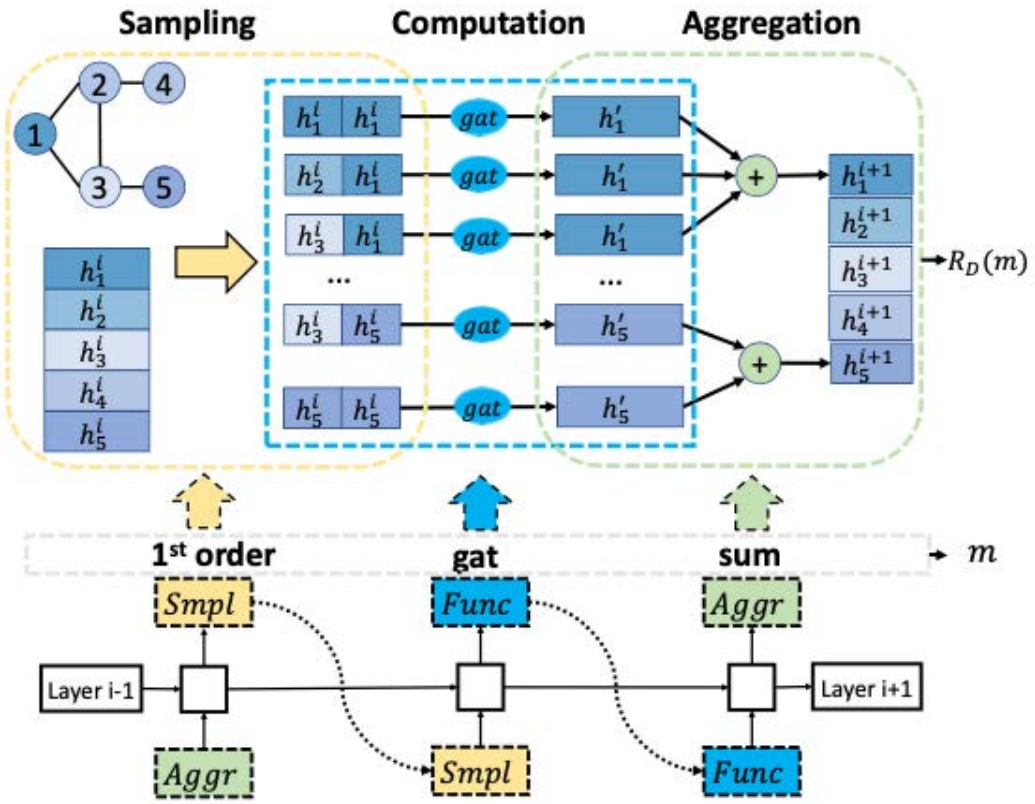
Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Search Space Complexity

- K-layer GNN as the model
  - Example: list of strings

$$[\ first\_order, gcn, sum, concat, 1, 16, relu,$$
$$first\_order, gat, sum, avg, 8, 16, elu\ ].$$

- Size of the search space
  - $9{,}408^L$ (L as number of GNN layers)
  - 8×10^7 When $L = 2$



Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Search Strategy

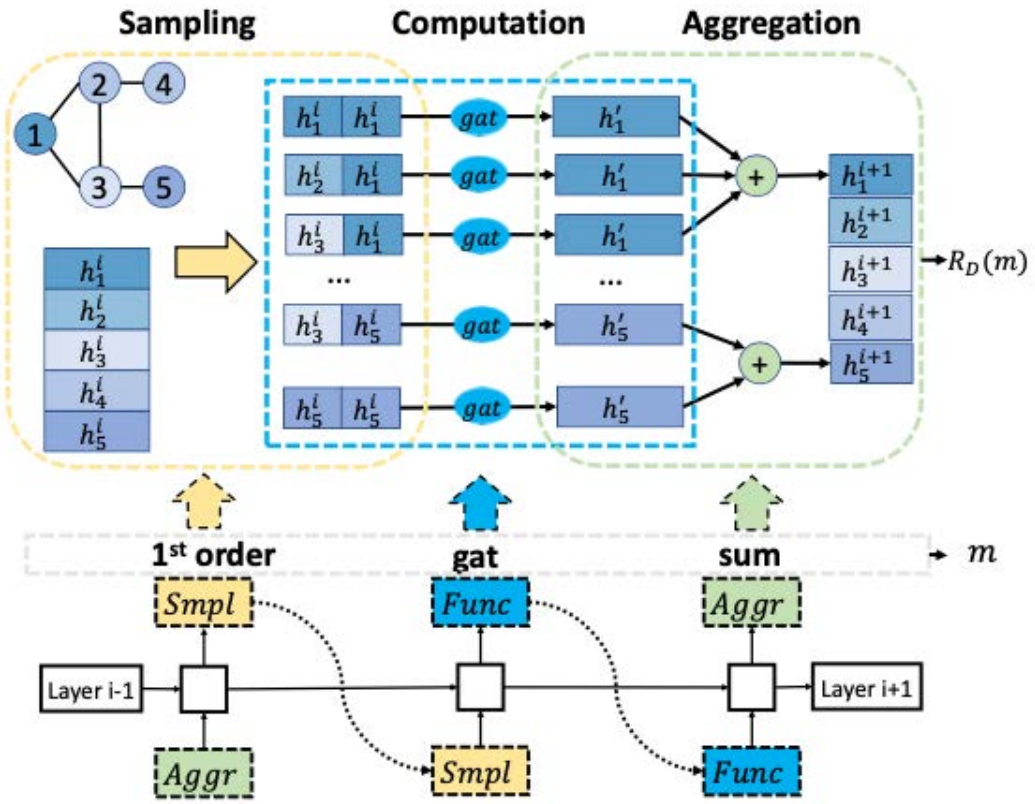
- Search algorithm
  - A RNN controller samples an architecture
  - Training till convergency
  - Update RNN controller based on the validation accuracy
- Search Objective

$$m^* = \arg\max_{m \in \mathcal{M}} \ \mathbb{E}[R_D(m)].$$

- Policy gradient

$$\nabla_\theta \mathbb{E}_{P(m_{1:T};\theta)}[R] = \sum_{t=1}^{T} \mathbb{E}_{P(m_{1:T};\theta)}[\nabla_\theta log P(m_t | m_{t-1:1}; \theta)(R - b)]$$

  - $m_{1:T}$: a list of operators with length T
  - θ : the parameters of the RNN controller



Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

GraphNAS Algorithm

Algorithm 1 GraphNAS search algorithm

Require: search space $\mathcal{M}$; controller RNN parameterized by θ; graph G; validation set D; # of operators T; # of models K; # of repeats N; # of samples S
Ensure: the best architecture m^*
// policy gradient
while the number of samples S is not met **do**
  $h_0 = \mathbf{0}$ // initial hidden state of RNN
  // sample operators $m_{1:T}$
  for $i = 1, ..., T$ **do**
    $x_i \leftarrow h_{i-1}$ // input of RNN
    $h_i \leftarrow RNN_\theta(x_i, h_{i-1})$;
    $P_i \leftarrow Softmax(h_{i-1})$;
    Sample m_i from $\mathcal{M}$ under P_i
  end for
  Design architecture m using operators $m_{1:T}$
  Train m on a given graph G
  Calcuate reward $R_D(m)$ on validation set D
  Update parameter θ *w.r.t.* $R_D(m)$
end while
// model selection
Select top K models *w.r.t.* validation accuracy
Re-train the K models for N times, select the best m^*
return m^*

search space and other hyper-parameter settings

use an RNN model parameterized by θ to generate the operators

use policy gradient to update θ

evaluate for the best model

Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Experiments

- Datasets
  - Three benchmark ones.
- Task
  - Node classification
- Settings
  - Semi-supervised
    - 20 training, 500 validation, 100 test
  - Supervised
    - 500 validation, 500 test, rest training
  - Supervised with randomly split
  - Transfer learning

N, E, F and C denote the number of "Nodes", "Edges", "Features" and "Classes", respectively.

Dataset	N	E	F	C
Cora	2,708	5,278	1,433	7
CiteSeer	3,327	4,552	3,703	6
PubMed	19,717	44,324	500	3

Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Experiments

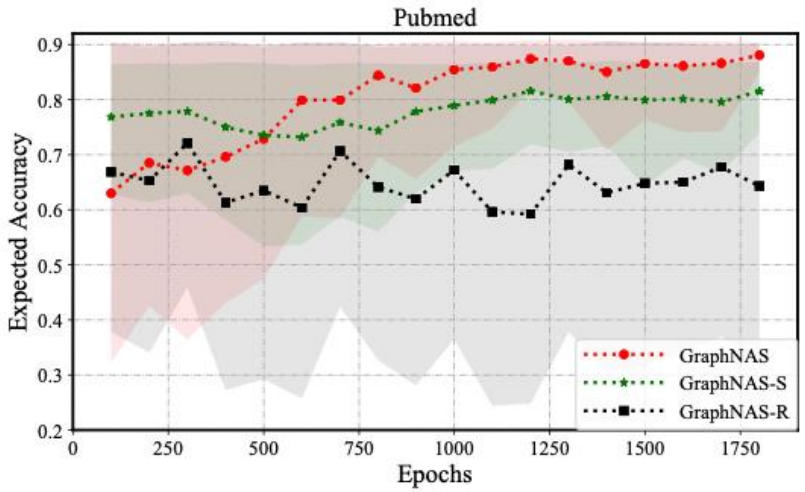
- Results
  - Outperform all non-NAS methods
    - GraphNAS-R: random search algorithm
    - GraphNAS-S: simple variant with only one computational node in each layer.

	Cora			Citeseer			Pubmed		
	semi	sup	rand	semi	sup	rand	semi	sup	rand
GCN	81.4±0.5	90.2±0.0	88.3±1.3	70.9±0.5	80.0±0.3	77.2±1.7	79.0±0.4	87.8±0.2	88.1±1.4
GAT	83.0±0.7	89.5±0.3	87.2±1.1	72.5±0.7	78.6±0.3	77.1±1.3	79.0±0.3	86.5±0.6	87.8±1.4
ARMA	82.8±0.6	89.8±0.1	88.2±1.0	72.3±1.1	79.9±0.6	76.7±1.5	78.8±0.3	88.1±0.2	88.7±1.0
APPNP	83.3±0.1	90.4±0.2	87.5±1.4	71.8±0.4	79.2±0.4	77.3±1.6	80.2±0.2	87.4±0.3	88.2±1.1
HGCN	79.8±1.2	89.7±0.4	87.7±1.1	70.0±1.3	79.2±0.5	76.9±1.3	78.4±0.6	88.0±0.5	88.0±1.6
GraphNAS-R	83.3±0.4	90.0±0.3	88.5±1.0	73.4±0.4	81.1±0.3	76.5±1.3	79.0±0.4	90.7±0.6	90.3±0.8
GraphNAS-S	81.4±0.6	90.1±0.3	88.5±1.0	71.7±0.6	79.6±0.5	77.5±2.3	79.5±0.5	88.5±0.2	88.5±1.1
GraphNAS	**83.7±0.4**	**90.6±0.3**	**88.9±1.2**	**73.5±0.3**	**81.2±0.5**	**77.6±1.5**	**80.5±0.3**	**91.2±0.3**	**91.1±1.0**

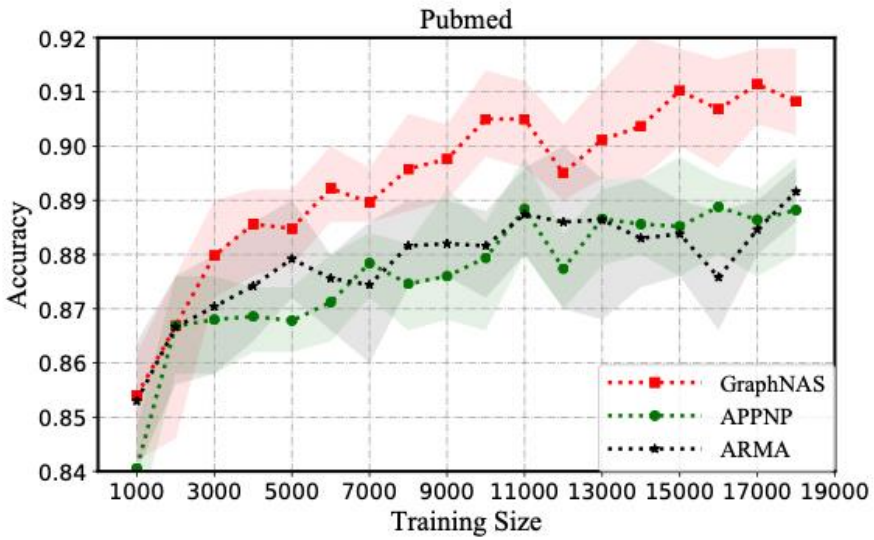
Table 3: Node classification results *w.r.t.* accuracy, where "semi" stands for semi-supervised learning experiments, "sup" for supervised learning experiments and "rand" for supervised learning experiments with randomly split data.

Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Experiments



Convergence curve: compared with GraphNAS-R and GraphNAS-S



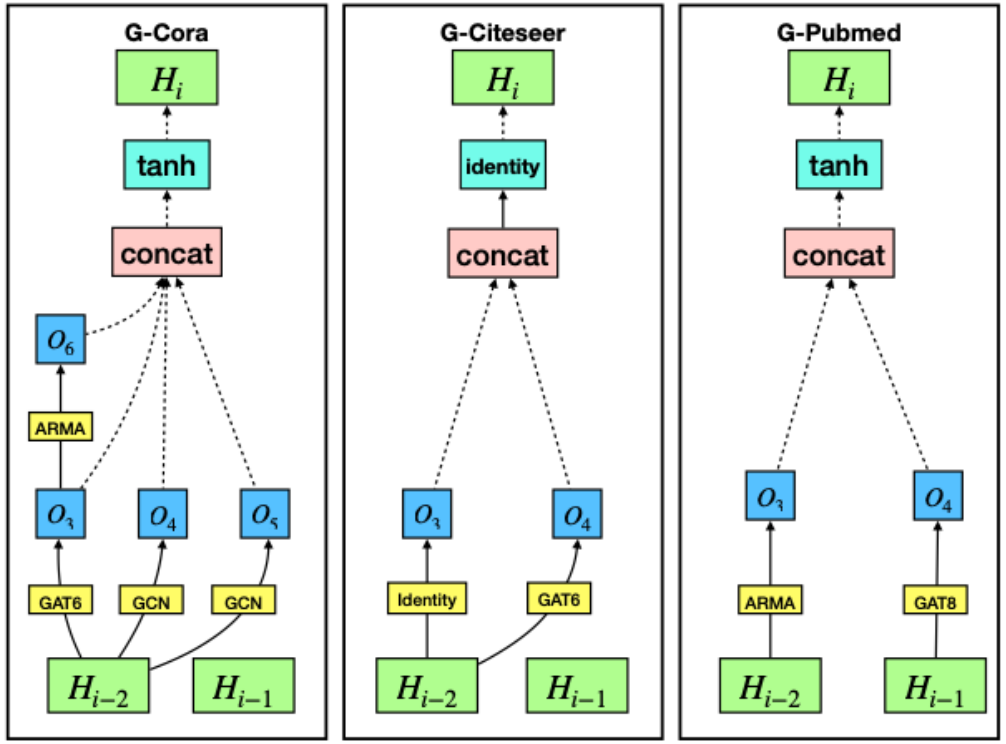
Performance with different number of training samples: compared with other GNN models

Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Experiments

- Searched architectures

Figure 5: An example of the graph neural architectures designed by GraphNAS on the supervised learning task. The architecture *G-Cora* designed by GraphNAS on Cora is $[1, gat\_6, 1, gcn, 1, gcn, 3, arma, tanh, concat]$, the architecture *G-Citeseer* designed by GraphNAS on Citeseer is $[1, identity, 1, gat\_6, identity, concat]$, and the architecture *G-Pubmed* designed by GraphNAS on Pubmed is $[2, gat\_8, 1, arma, tanh, concat]$.



Gao et al. GraphNAS: Graph Neural Architecture Search with Reinforcement Learning. IJCAI 2020

Quick Summary

- A first graph neural network architecture search by RL
- Mainly designed for node-level task (graph node classification)
- Effective but expensive due to "trial-and-error" manner
  - Stand alone evaluation strategy (trains the model from scratch to convergence for performance evaluation)
  - Using continuous relaxation strategy can be more effective

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
  - Introduction: Graph Neural Architecture Search for Different Graph Learning Tasks
  - Node-level: GraphNAS
  - Graph-level: PAS
  - Link-level: GNAS-CF
  - Summary
- Summary

PAS

- Pooling Architecture Search (PAS) for Graph Classification
  - The first method to learn data-specific pooling architectures.

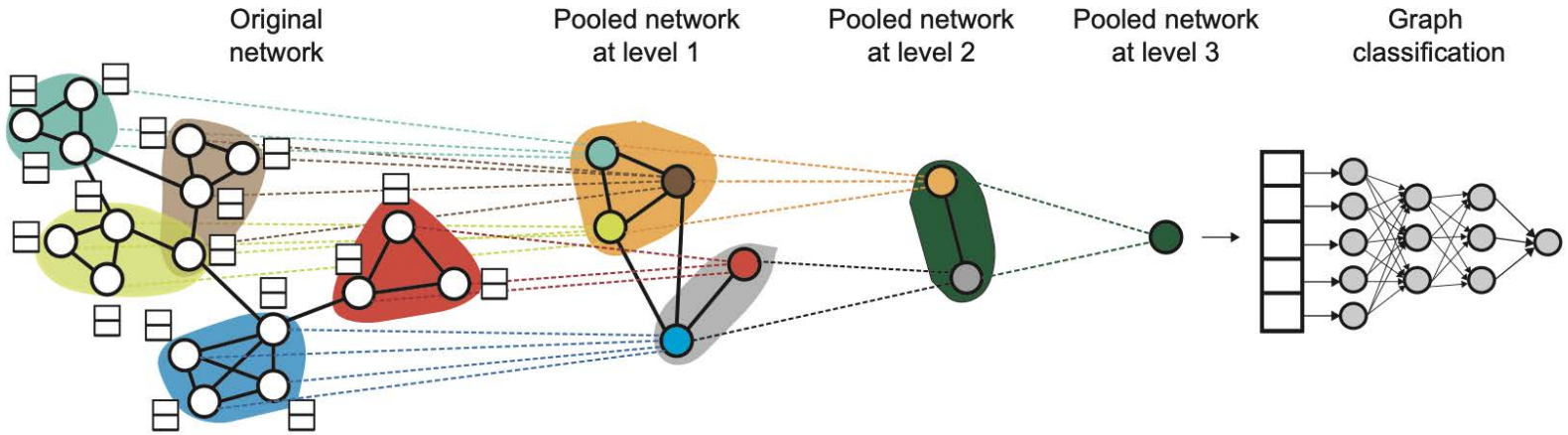
- Differentiable pooling architecture search
  - The hierarchical pooling method is not always superior to the global one. [Diego et al. 2020, Federico et al. 2020]
  - A unified framework which can design the global and hierarchical pooling methods adaptively.
  - A coarsening strategy is designed to relax the search space and enable the usage of the differentiable search algorithm.

- SOTA performance

Pooling Architecture Search for Graph Classification. CIKM 2021
Rethinking pooling in graph neural networks. NeurIPS 2020
A fair comparison of graph neural networks for graph classification. ICLR 2020

Background: Graph Pooling

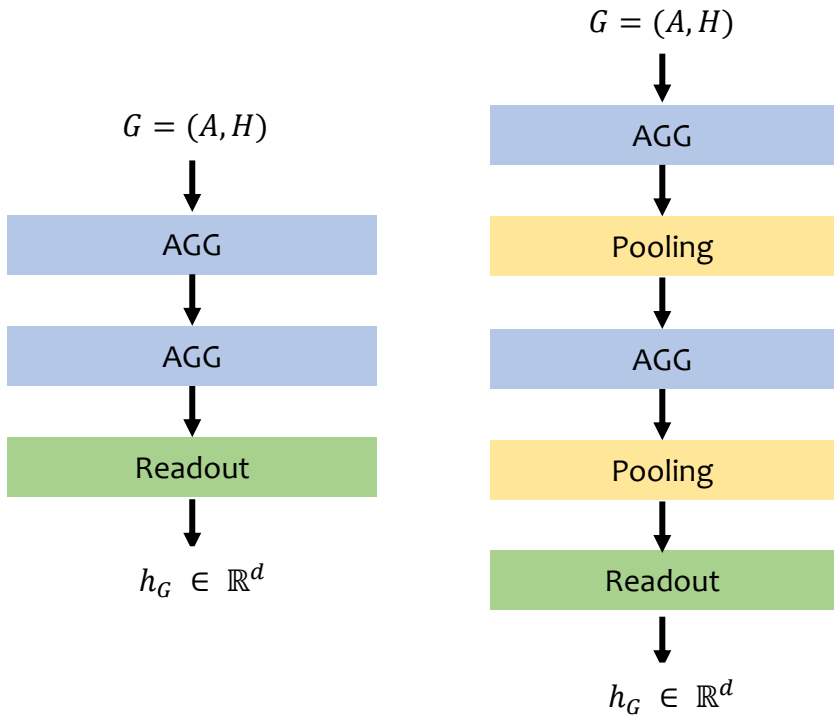
- Graph pooling: a technique used in GNNs to reduce the size and complexity of graphs
  - Easier to obtain graph representations
  - Simplify complex graphs
  - Capturing important graph features



an example of graph pooling

Graph Pooling Methods

- Global pooling
  - Pool nodes at the end of GNN.
  - Representative works: GIN[1] / DGCNN[2]
  - Learn flat graph-level representation.
- Hierarchical pooling
  - Learn hierarchical information by stacking aggregation layers and pooling operations.
  - Representative works: DiffPool [3] / SAGPool [4] / ASAP [5]



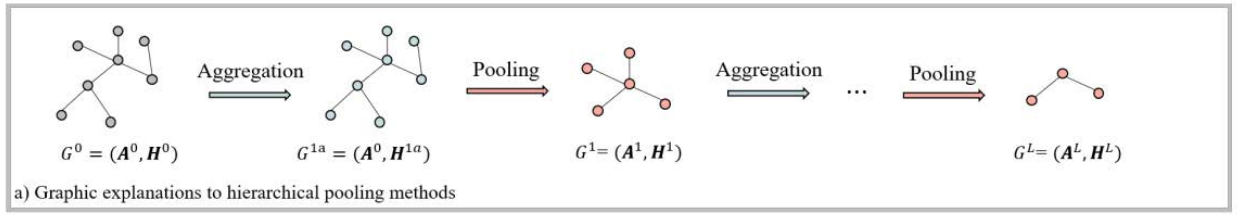
Global pooling

Hierarchical pooling

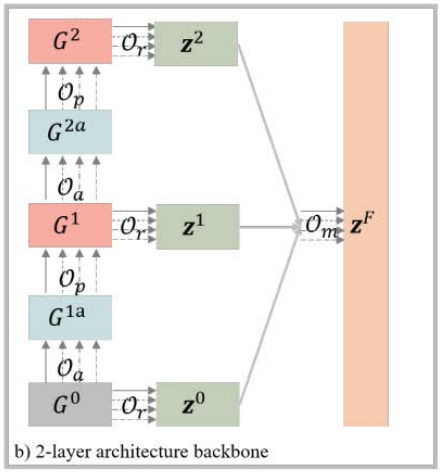
[1] How powerful are graph neural networks? ICLR 2019
[2] An end-to-end deep learning architecture for graph classification. AAAI 2018
[3] Hierarchical graph representation learning with differentiable pooling. NeurIPS 2018
[4] Self-Attention Graph Pooling. ICML 2019
[5] ASAP: Adaptive Structure Aware Pooling for Learning Hierarchical Graph Representations. AAAI 2020

The Unified Framework

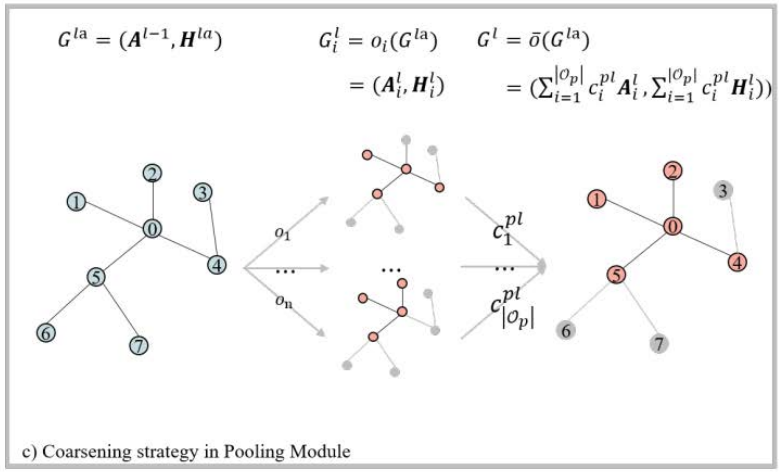
- Four key modules derived from existing pooling methods.
- One novel search space which can cover both the global and hierarchical pooling architectures.
- One coarsening strategy to relax the search space.



a) Graphic explanations to hierarchical pooling methods



b) 2-layer architecture backbone



c) Coarsening strategy in Pooling Module

	Methods	Task	Search Space				Search
			A	P	R	M	Algorithm
GNNs	GraphSAGE [13]	Node	√	-	-	×	-
	JK-Net [42]	Node	√	-	-	√	-
	GIN [41]	Graph	√	×	√	×	-
	DiffPool [44]	Graph	√	√	√	×	-
	SAGPool [20]	Graph	√	√	√	√	-
NAS	GraphNAS [10]	Node	√	-	-	×	RL
	SNAG [50]	Node	√	-	-	√	RL
	RE-MPNN [16]	Graph	√	×	√	√	EA
	AutoGM [45]	Node	√	-	-	×	Bayesian
	SANE [51]	Node	√	-	-	√	Gradient Descent
	DSS [23]	Node	√	-	-	×	Gradient Descent
	GNAS [3]	Node	√	-	-	×	Gradient Descent
	PAS (proposed)	Graph	√	√	√	√	Gradient Descent

A: aggregation P: pooling
R: readout M: merge

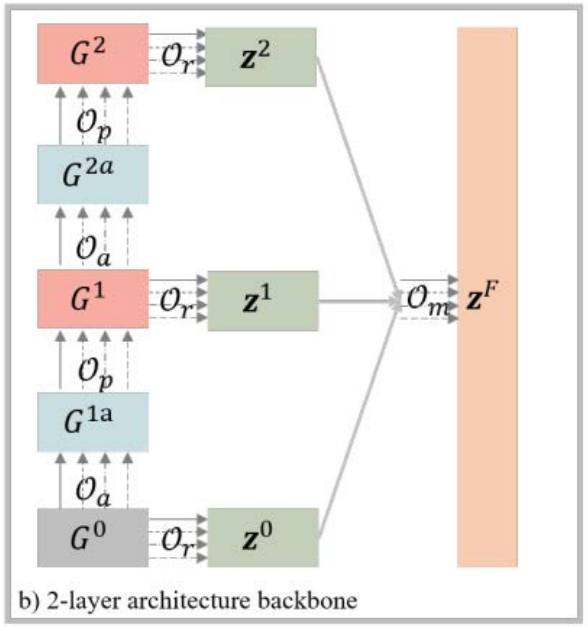
Search Space

Pooling Module:

- Calculate the node scores and generate the coarsen graphs by selecting Top-K nodes.

$$\mathbf{S} = f_s(\mathbf{A}, \mathbf{H}), idx = \mathtt{TOP}_k(\mathbf{S}),$$
$$\mathbf{A}' = \mathbf{A}(idx, idx), \mathbf{H}' = \mathbf{H}(idx, :).$$

- NONE operation: no pooling operation in this layer.



b) 2-layer architecture backbone

Module name	Operations
Aggregation $\mathcal{O}_a$	GCN, GAT, SAGE, GIN, GRAPHCONV, MLP
Pooling $\mathcal{O}_p$	TOPKPOOL, SAGPOOL, ASAP, HOPPOOL_1, HOPPOOL_2, HOPPOOL_3, MLPPOOL, GCPOOL, GAPPOOL, NONE
Readout $\mathcal{O}_r$	GLOBAL_SORT, GLOBAL_ATT, SET2SET, GLOBAL_MEAN, GLOBAL_MAX, GLOBAL_SUM, ZERO
Merge $\mathcal{O}_m$	M_LSTM, M_CONCAT, M_MAX, M_MEAN, M_SUM

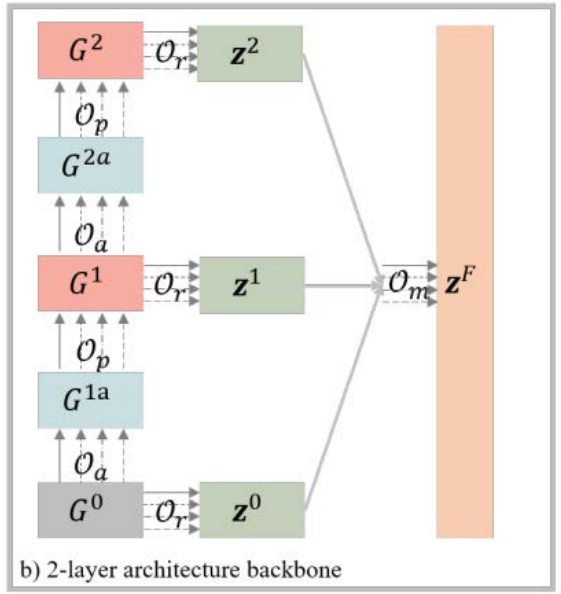
Search space size: $6^2 \times 10^2 \times 7^3 \times 5 = 6{,}174{,}000$

Differentiable Search Algorithm

- Differentiable search algorithm[1] is proposed to address the search efficiency problem.
- It relaxes the discrete search space into a continuous one by mixing the output of all candidate operations.
  - Relaxation function

$$\bar{o}(x) = \sum_{k=1}^{|\mathcal{O}|} c_k o_k(x) \qquad c_k \epsilon (0,1)$$

  - Mixed results in PAS
    - Aggregation: $\mathbf{H}^{la} = \sum_{i=1}^{|O_a|} c_i^{al} o_i(\mathbf{A}^{l-1}, \mathbf{H}^{l-1})$
    - Readout: $\mathbf{z}^{l} = \sum_{i=1}^{|O_r|} c_i^{rl} o_i(\mathbf{A}^{l}, \mathbf{H}^{l})$
    - Merge: $\mathbf{z}^{F} = \sum_{i=1}^{|O_m|} c_i^{m} o_i(\mathbf{z}^{0}, \mathbf{z}^{1}, \cdots, \mathbf{z}^{L})$



b) 2-layer architecture backbone

DARTS: Differentiable Architecture Search, ICLR 2019

Differentiable Search Algorithm

- It is unachievable to relax the pooling module since different coarse graphs contain diverse nodes and edges.

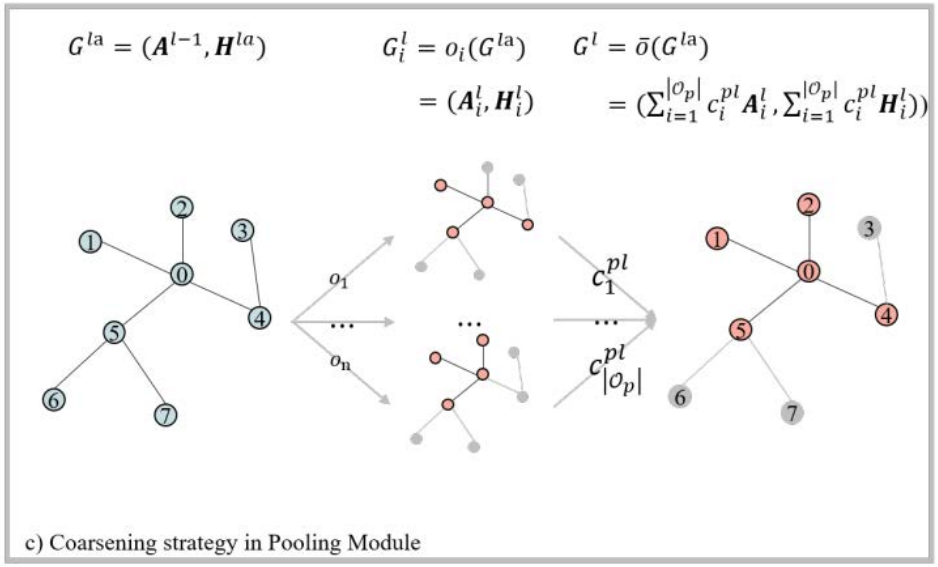
$V_1 = \{0,1,4,5\},\ A_1 \epsilon \mathbb{R}^{4\times4}, H_1 \epsilon \mathbb{R}^{4\times d}$

$V_{\mathrm{n}} = \{0,1,2,5\},\ A_n \epsilon \mathbb{R}^{4\times4}, H_n \epsilon \mathbb{R}^{4\times d}$

Mismatched nodes

- Coarsening strategy
  - In pooling operations, keep the same "shape" when formulating the coarsen graphs.
  - For unselected nodes and edges (in grey), we set the features and weights to 0.

$$G^l = (\boldsymbol{A}^l, \boldsymbol{H}^l) = (\sum_{i=1}^{|O_p|} c_i^{pl} \boldsymbol{A}_i^l, \sum_{i=1}^{|O_p|} c_i^{pl} \boldsymbol{H}_i^l)$$



c) Coarsening strategy in Pooling Module

Experiments

- Datasets
  - 6 real-world datasets from 3 domains
- Baselines
  - Global pooling methods
  - Hierarchical pooling methods
  - NAS methods:
    - RL based: GraphNAS[1] / SNAG[2]
    - EA / Random / Bayesian

Dataset	# of Graphs	# of Feature	# of Classes	Avg.# of Nodes	Avg.# of Edges	Domain
D&D	1,178	89	2	384.3	715.7	Bioinfo
PRO	1,113	3	2	39.1	72.8	Bioinfo
IMDB-B	1,000	0	2	19.8	96.5	Social
IMDB-M	1,500	0	3	13	65.9	Social
COX2	467	3	2	41.2	43.5	Chemistry
NCI109	4127	0	2	29.69	32.13	Chemistry

[1] Gao et al. Graphnas: Graph neural architecture search with reinforcement learning. IJCAI 2020
[2] Zhao et al. Simplifying Architecture Search for Graph Neural Network. CIKM-CSSA 2020

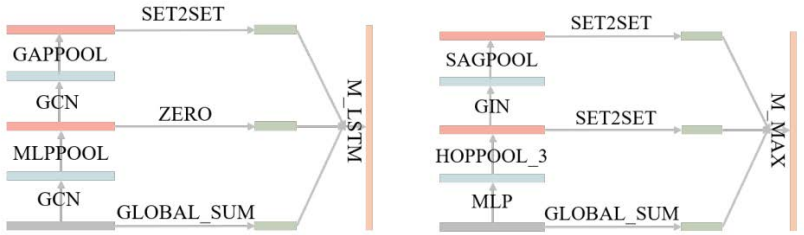
Experiments

- No absolute winner from human-designed models and PAS consistently outperforms all baselines.
- Compared with NAS baselines, the proposed differentiable search algorithm provide performance gain.

Table 4: Performance comparisons of PAS and all baselines. We report the mean test accuracy and the standard deviation by 10-fold cross-validation. The best results in different groups of baselines are underlined, and the best result on each dataset is in boldface.

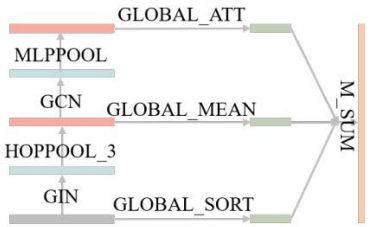
	Method	D&D	PROTEINS	IMDB-BINARY	IMDB-MULTI	COX2	NCI109
Global pooling	GCN	0.7812±0.0433	0.7484±0.0282	0.7267±0.0642	0.5040±0.0302	0.7923±0.0219	0.7344 ± 0.0192
	GAT	0.7556±0.0372	0.7530±0.0372	0.7407±0.0453	0.4967±0.0430	0.8156±0.0417	0.7410 ± 0.0245
	SAGE	0.7727±0.0406	0.7375±0.0297	0.7217±0.0529	0.4853±0.0543	0.8031±0.0594	0.7553 ± 0.0164
	GIN	0.7540±0.0368	0.7448±0.0278	0.7167±0.0277	0.4980±0.0250	0.8309±0.0417	0.7456 ± 0.0210
	GCN-JK	0.7769±0.0235	0.7539±0.0517	0.7347±0.0445	0.4886±0.0496	0.7966±0.0226	0.7383 ± 0.0188
	GAT-JK	0.7809±0.0618	0.7457±0.0405	0.7327±0.0468	0.4953±0.0384	0.8072±0.0287	0.7172 ± 0.0292
	SAGE-JK	0.7735±0.0420	0.7494±0.0355	0.7287±0.0618	0.4973±0.0340	0.7990±0.0415	0.7325 ± 0.0356
	GIN-JK	0.7513±0.0395	0.7312±0.0451	0.7207±0.0486	0.5080±0.0302	0.8117±0.0467	0.7441 ±0.0220
	DGCNN	0.7666±0.0403	0.7357±0.0469	0.7367±0.0570	0.4900±0.0356	0.7985±0.0264	0.7506 ± 0.0165
	MLP-Baseline	0.7752±0.0390	0.7239±0.0353	0.7287±0.0520	0.4980±0.0428	0.7838±0.0104	0.6445 ± 0.0160
Hierarchical pooling	ASAP	0.7735±0.0415	0.7493±0.0357	0.7427±0.0397	0.5013±0.0344	0.8095±0.0320	0.7376 ± 0.0224
	SAGPool	0.7506±0.0506	0.7312±0.0447	0.7487±0.0409	0.4933±0.0490	0.7945±0.0298	0.6489 ± 0.0315
	Graph U-Net	0.7710±0.0517	0.7440±0.0349	0.7317±0.0484	0.4880±0.0319	0.8030±0.0421	0.7279 ± 0.0229
	DiffPool	0.7775±0.0400	0.7355±0.0322	0.7186±0.0563	0.4953±0.0398	0.7966±0.0264	0.7315 ± 0.0214
	MinCutPool	0.7803±0.0363	0.7575±0.0380	0.7077±0.0489	0.4900±0.0283	0.8007±0.0385	0.7405±0.0248
NAS	GraphNAS	0.7198±0.0454	0.7251±0.0336	0.7110±0.0230	0.4693±0.0364	0.7773±0.0140	0.7228 ± 0.0228
	GraphNAS-WS	0.7674±0.0455	0.7520±0.0251	0.7360±0.0463	0.4827±0.0350	0.7816±0.0058	0.7049 ± 0.0192
	SNAG	0.7223±0.0386	0.7053±0.0311	0.7250±0.0461	0.4933±0.0289	0.7903±0.0212	0.7090±0.0224
	SNAG-WS	0.7351±0.0303	0.7233±0.0244	0.7360±0.0516	0.5000±0.0248	0.8054±0.0381	0.7063±0.0160
	EA	0.7514±0.0301	0.7341±0.0298	0.7400±0.0412	0.4860±0.0405	0.7945±0.0159	0.7324±0.0126
	Random	0.7792±0.0482	0.7394±0.0423	0.7210±0.0554	0.4980±0.0398	0.8073±0.0231	0.7306±0.0241
	Bayesian	0.7555±0.0321	0.7314±0.0239	0.7270±0.0335	0.4980±0.0397	0.8029±0.0172	0.7204±0.0114
	PAS	**0.7896±0.0368**	**0.7664±0.0329**	**0.7510±0.0532**	**0.5220±0.0373**	**0.8344±0.0633**	**0.7684±0.0272**

Experiments



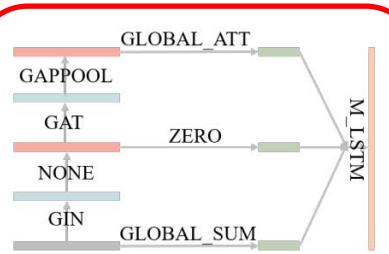
(a) D&D

(b) PROTEINS

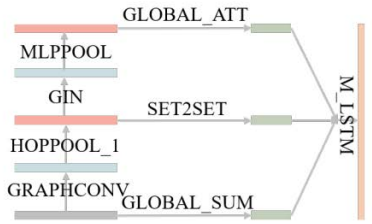


(c) IMDB-BINARY

(d) IMDB-MULTI

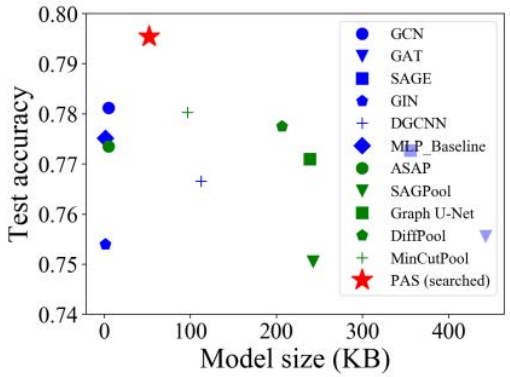


(e) COX2

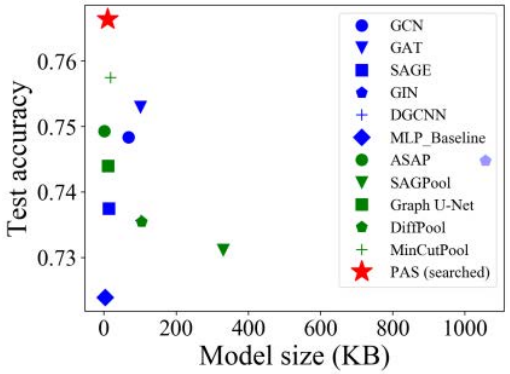


(f) NCI109

data-specific architectures



(a) D&D



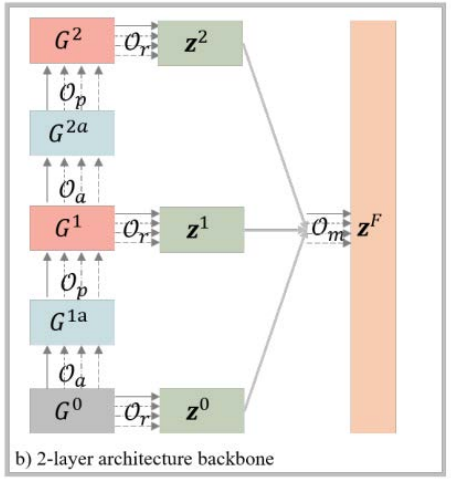
(b) PROTEINS

- Data-specific pooling architectures are obtained.
- Global pooling architecture on COX2 is obtained by PAS.
- These searched architectures can achieve SOTA performance with moderate size in terms of the model parameters.

Ablation Study

- Ablation studies on 4 modules:
  - Aggregation Module: fixed aggregators
  - Pooling Module: fixed NONE operation
  - Readout Module: fixed GLOBAL_MEAN operation
  - Merge Module: remove Merge Module

Fixed		D&D	IMDB-MULTI
Aggregation	PAS-GCN	0.7835±0.0407	0.5027±0.0409
	PAS-GAT	0.7878±0.0376	0.5087±0.0417
Pooling	PAS-Global	0.7708±0.0330	0.5173±0.0447
Readout	PAS-FR	0.7436±0.0472	0.5033±0.0436
Merge	PAS-RM	0.7682±0.0336	0.5047±0.0380
	PAS	**0.7896±0.0368**	**0.5220±0.0373**



b) 2-layer architecture backbone

Module name	Operations
Aggregation $\mathcal{O}_a$	GCN, GAT, SAGE, GIN, GRAPHCONV, MLP
Pooling $\mathcal{O}_p$	TOPKPOOL, SAGPOOL, ASAP, HOPPOOL_1, HOPPOOL_2, HOPPOOL_3, MLPPOOL, GCPOOL, GAPPOOL, NONE
Readout $\mathcal{O}_r$	GLOBAL_SORT, GLOBAL_ATT, SET2SET, GLOBAL_MEAN, GLOBAL_MAX, GLOBAL_SUM, ZERO
Merge $\mathcal{O}_m$	M_LSTM, M_CONCAT, M_MAX, M_MEAN, M_SUM

Quick Summary

- PAS is the first method to learn pooling architectures for graph classification.

- Contributions
  - Search space
    - Design pooling architectures on the global and hierarchical methods adaptively.
  - Search algorithm
    - Design a coarsening strategy to enable the usage of gradient descent.

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
  - Introduction: Graph Neural Architecture Search for Different Graph Learning Tasks
  - Node-level: GraphNAS
  - Graph-level: PAS
  - Link-level: GNAS-CF
  - Summary
- Summary

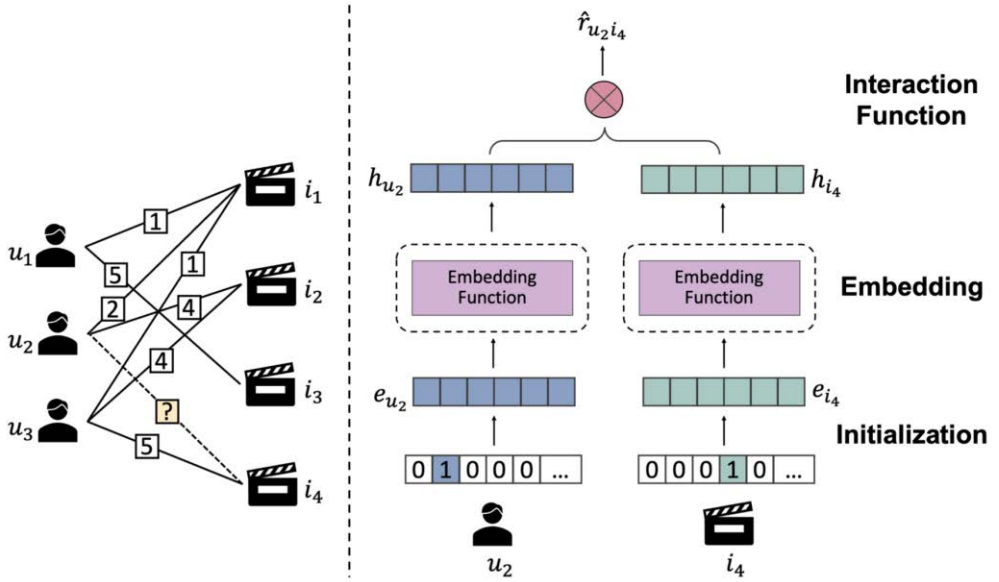
GNAS-CF

- A first graph neural architecture search method for collaborative filtering (CF)
- Design a unified search space that contain the key design in most of existing CF methods
- Prune the proposed search space and obtain outstanding performance in CF tasks

Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022

Background: Collaborative Filtering (CF)

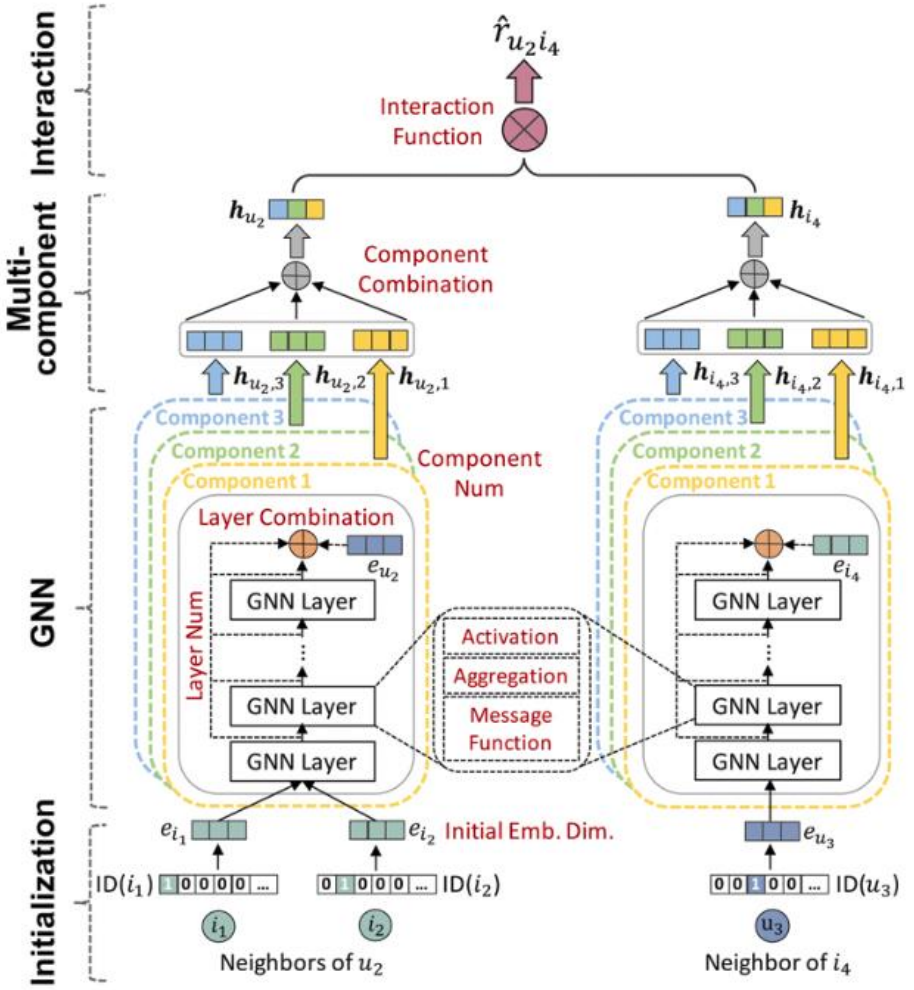
- **Collaborative Filtering (CF) :** popular method in recommender systems
- **Graph Neural Networks (GNNs)** have been incorporated for the CF tasks, since the user-item interactions can be naturally modeled as a bipartite graph (formulated as a **link prediction task**)



Framework

4 key modules in unified CF framework:

- Initialization: project user/item ID to embedding
- GNN: refine the embeddings
- Multi-component: model diverse user interests from different aspects
- Interaction: performs the user-item matching



unified CF framework

Search Space

Design Dimension	Choices
Initial Embedding Dimension d	64, 128, 256
Message Function $m(\cdot)$	Identity, Hadamard
Aggregation $f(\cdot)$	None, GCN, GAT, GIN, GraphSAGE
Activation $\sigma(\cdot)$	Identity, Sigmoid, Tanh, ReLU, PReLU, LeakyReLU
Layer Number L	1, 2, 3, 4
Layer Combination $g(\cdot)$	Stack, Concat, Sum, Mean
Component Number K	1, 2, 3, 4
Component Combination $c(\cdot)$	Concat, Mean, Att
Interaction Function $p(\cdot)$	Dot Product, Concat+MLP, Sum+MLP

choices in search space

$$\mathbf{h}_u^{(l+1)} = \sigma\left(f\left[\left\{\mathbf{m}_{u\leftarrow i}^{(l)}, \forall i \in \mathcal{N}(u)\right\}, \mathbf{h}_u^{(l)}\right]\right)$$

- 4 GNN aggregators and *None* to enlarge the capacity of the design space to include those non-GNN-based models

- To explore the impacts of different design dimensions, propose to profile the **design space,** defined as the Cartesian product of search space dimensions

Relationship with Existing CF Methods

- The table below shows the existing CF method
  - the framework unifies the key design dimensions in popular CF models
  - the search space is comprehensive enough to include a broad spectrum of model instantiations

Category	Model	$m(\cdot)$	$f(\cdot)$	$\sigma(\cdot)$	$g(\cdot)$	$c(\cdot)$	$p(\cdot)$	Single-/Multi-component
Classic	MF [16, 24]	Identity	None	Identity	Stack	-	Dot Product	Single
	LLORMA [18, 46]	Identity	None	Identity	Stack	Att	Dot Product	Multiple
MLP-based	NCF [11]	Identity	None	ReLU	Stack	-	Concat+MLP	Single
GNN-based	NGCF [36]	Hadamard	GCN	LeakyReLU	Concat	-	Dot Product	Single
	LightGCN [10]	Identity	GCN	Identity	Mean	-	Dot Product	Single
	LR-GCCF [2]	Identity	GCN	Identity	Concat	-	Dot Product	Single
	SMOG-CF [43]	Hadamard	GCN	ReLU	Concat	-	Dot Product	Single
	PinSage [41]	Identity	GraphSAGE	ReLU	Stack	-	Dot Product	Single
	MCCF [38]	Identity	GAT	ReLU	Stack	Att	Concat+MLP	Multiple
	DGCF [37]	Identity	GCN	Tanh	Sum	Concat	Dot Product	Multiple

Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022

Evaluation of the Search Space

- Dataset
  - 9 real-world dataset
  - diverse in scale and density
- Evaluation technique
  - **controlled random search**
  - first randomly sample S configurations
  - then only change one of the dimensions to analyze the performance difference
- Loss and evaluation metric
  - Mean Square Error(MSE) loss

$$L = \frac{\sum_{(u,i)\in O_t} (\hat{r}_{ui} - r_{ui})^2}{|O_t|} + \lambda \|\Theta\|^2$$

model parameter

  - Evaluation metric: Rooted Mean Square Error

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in O_e} (\hat{r}_{ui} - r_{ui})^2}{|O_e|}}$$

Dataset	# of Users	# of Items	# of Interactions	Rating Scale	Density
Yelp[1]	58,069	31,721	1,160,605	[1,5]	0.063%
Amazon-CDs [9]	31,296	24,379	622,163	[1,5]	0.082%
Amazon-Movies [9]	44,439	25,047	1,070,860	[1,5]	0.096%
YahooMusic [5, 25]	1,357	1,363	5,335	[1,100]	0.28%
Amazon-Beauty [9]	7,068	3,570	79,506	[1,5]	0.32%
Flixster[13, 25]	2,341	2,956	26,173	[0.5,5]	0.38%
Douban [23, 25]	2,999	3,000	136,891	[1,5]	1.52%
MovieLens-1M[2]	6,040	3,706	1,000,209	[1,5]	4.47%
MovieLens-100K[3]	943	1,682	100,000	[1,5]	6.31%

Group No.	Configuration Space					Experimental Results	
	Message Function	Activation	...	Interaction Function	Dataset	Performance	Ranking
1	Identity	ReLU	...	Dot Product	Yelp	0.8996	2
	Hadamard					0.8665	1
2	Identity	Sigmoid	...	Concat+MLP	Amazon-CDs	0.7636	1
	Hadamard					0.7812	2
...							
S	Identity	Tanh	...	Sum+MLP	MovieLens-1M	0.8231	1(tie)
	Hadamard					0.8231	1(tie)

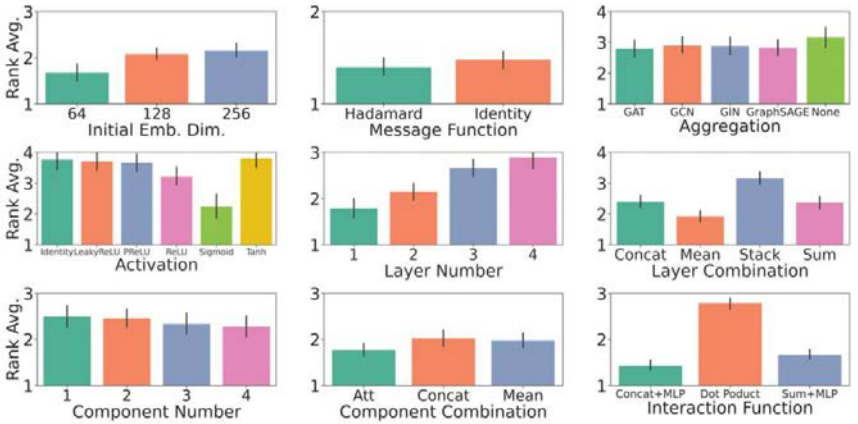
randomly sampled configurations

dimension of interest

Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022

Evaluation Results and Pruned Search Space

- Evaluation results
  - For several search space dimensions, there are choices that consistently perform better



Evaluation results on 9 search space dimensions

- Pruned search space
  - search space complexity: 103,680 → 96
  - consistently shows high quality in different recommendation settings (strong generalization ability)

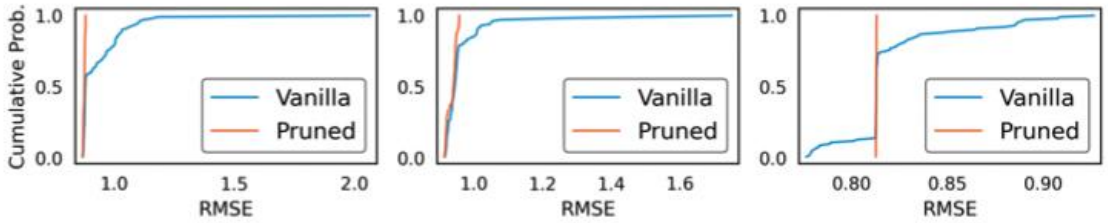
Design Dimension	Choices
Initial Embedding Dimension d	64
Message Function $m(\cdot)$	Identity, Hadamard
Aggregation $f(\cdot)$	None, GraphSAGE
Activation $\sigma(\cdot)$	Identity, Sigmoid, ReLU
Layer Number L	1, 2
Layer Combination $g(\cdot)$	Mean
Component Number K	1, 4
Component Combination $c(\cdot)$	Att
Interaction Function $h(\cdot)$	Concat+MLP, Sum+MLP

Pruned search space

Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022

Experiments

- Comparison of pruned space with initial search space
  - pruned space have better and stabler results
  - strong generalization to various settings
- Comparison with baselines
  - Randomly searched model from the pruned search space perform better than other CF baselines



Yelp Amazon-Beauty MovieLens-1M

Vanilla: initial search space
Pruned: pruned search space

Model	Epinions	Amazon-Sports
MF [16]	0.9945 ± 0.0000	0.9882 ± 0.0007
NCF [11]	1.0070 ± 0.0055	<u>0.9342 ± 0.0008</u>
NGCF [36]	1.1437 ± 0.0240	1.0668 ± 0.0038
LightGCN [10]	<u>0.9926 ± 0.0001</u>	0.9705 ± 0.0003
DGCF [37]	1.6800 ± 0.2272	0.9894 ± 0.0000
RS-10	**0.8729 ± 0.0014**	**0.9327 ± 0.0006**

Profiling the Design Space for Graph Neural Networks based Collaborative Filtering. WSDM 2022

Quick Summary

- Problem
  - The first attempt to profile the search space of GNN-based CF
- Evaluation of the Search Space
  - Propose a unified framework covering popular GNN-based CF models
  - Develop a search space and evaluate it by extensive experiments
- Evaluation of the Pruned search Space
  - Prune the search space for a higher concentration of top-performing models
  - Empirical studies demonstrate its high quality and strong generalization ability

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
  - Introduction: Graph Neural Architecture Search for Different Graph Learning Tasks
  - Node-level: GraphNAS
  - Graph-level: PAS
  - Link-level: GNAS-CF
  - Summary
- Summary

Summary: Relation with Three Components in NAS

	Search space	Search strategy	Evaluation
GraphNAS (node level)	customized search space for GNN	reinforcement learning	weight sharing
PAS (graph level)	add graph pooling operation	coarsening strategy for continuous relaxation	weight sharing
GNAS-CF (link level)	unified CF framework	controlled random search	stand alone

Summary

- Different graph learning problems have different suitable models
  - node level: node representation learning
  - graph level: graph pooling
  - link level: scoring function learning
- Different graph NAS problems has different solutions:
  - GraphNAS: reinforcement learning
  - PAS: differentiable search algorithm (continuous relaxation)
  - GNAS-CF: search space pruning
- Find suitable technique for a specific problem

Code Link

- RL NAS: https://github.com/titu1994/neural-architecture-search
- NASNet: https://github.com/MarSaKi/nasnet
- ENAS: https://github.com/carpedm20/ENAS-pytorch
- DARTS: https://github.com/khanrc/pt.darts
- GraphGym: https://github.com/snap-stanford/GraphGym
- GraphNAS: https://github.com/GraphNAS/GraphNAS
- PAS: https://github.com/LARS-research/PAS
- GNAS-CF: https://github.com/BUPT-GAMMA/Design-Space-for-GNN-based-CF

Outline

- A Brief Introduction
- Key Components and Classical Works
- Understanding GNN Architectures
- NAS for GNN
- Summary

Summary

- NAS has great benefit to model performance
- NAS has three key components: search space, search strategy, performance estimation strategy
- Graph NAS examples
  - Design general GNN with condensed space: GraphGym
  - Design specific GNN
    - Node-level: GraphNAS
    - Graph-level: PAS
    - Link-level: GNAS-CF

Thanks.