Automated Machine Learning on Graph

Xin Wang, Ziwei Zhang, Wenwu Zhu
Tsinghua University
Network (Graph)

The general description of data and their relations.
Many types of data are networks

Social Networks

Biology Networks

Finance Networks

Internet of Things

Information Networks

Logistic Networks
Why network is important?

In few cases, you only care about a subject but not its relations with other subjects.
Many applications are intrinsically network problems

Recommendation Systems

Link prediction in bipartite graphs
Many applications are intrinsically network problems

Financial credit & risk management

Node importance & classification

Sample of the international financial network
Credits: ETH Zurich
Many applications are intrinsically network problems

New material discovery

Subgraph pattern discovery
Learning from networks

Network Embedding

GNN

Automated Graph Machine Learning
Learning from networks

Network Embedding
Network Embedding

$G = (V, E)$

- Easy to parallel
- Can apply classical ML methods
The ultimate goal

Network Inference

- Node importance
- Community detection
- Network distance
- Link prediction
- Node classification
- Network evolution
...
The goal of network embedding

**Goal** Support network inference in vector space

- Reflect network structure
- Maintain network properties

Transform network nodes into vectors that are fit for off-the-shelf machine learning models
Network Structures

Nodes & Links

Pair-wise Proximity

Community Structures

Hyper Edges

Global Structure

Dedicated Network Embedding Tutorial at KDD 2019:
High-Order Proximity

- Capturing the underlying structure of networks

- Advantages:
  - Solve the sparsity problem of network connections
  - Measure indirect relationship between nodes
DeepWalk

- Exploit truncated random walks to define neighborhoods of a node.

Random Walks on Graph

- $V_{26} - V_{25} - V_{32} - V_3 - V_{10}$...
- $V_5 - V_7 - V_{17} - V_6 - V_{11}$ ...
- $V_{31} - V_{33} - V_{21} - V_{33} - V_{15}$

LINE

LINE with First-order Proximity: local pairwise

\[ O_1 = - \sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j) \]

LINE with Second-order Proximity: neighborhood structures

\[ O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot | v_i), p_2(\cdot | v_i)) \]

GraRep

1-step
2-step
3-step
4-step

capturing different k-step information

Do not distinguish 1-step and 2-step.

maintaining different k-step information separately

Structural Deep Network Embedding

Idea

- solve non-linearity problem

- constrain

- preserve structure

Challenges

- How to represent network data in deep neural networks?
- How to preserve the first and second-order proximity in deep neural networks?

Framework of SDNE

Unsupervised Autoencoder
(preserve second-order proximity)

Supervised Constraint
(preserve first-order proximity)

Laplacian Eigenmaps

Unsupervised Autoencoder
(preserve second-order proximity)

Reconstruct the neighborhood structure of each vertex through deep autoencoder

\[ L_{2nd} = \| (\hat{X} - X) \odot B \|_F^2 \]

- Reconstruction of adjacency matrix
- Adjacency matrix
- Balancing weight

Preserve first-order proximity

- Incur penalty when connected vertexes are mapped far away in the embedding space

\[ L_{1st} = \sum_{i,j=1}^{n} a_{i,j} \| y_i - y_j \|_2^2 \]

Connection between the \( i \)-th and \( j \)-th vertex
The embedding of the \( i \)-th vertex
The embedding of the \( j \)-th vertex

Algorithm

Objective Function

- second-order loss
- first-order loss

\[ L = L_{2nd} + \alpha L_{1st} + \nu L_{reg} \]

Algorithm

Algorithm 1: Training Algorithm for the semi-supervised deep model of SDNE

**Input:** the network \( G = (V, E) \) with adjacency matrix \( S \), the parameters \( \alpha \) and \( \nu \)

**Output:** Network representations \( Y \) and updated Parameters: \( \theta \)

1. Pretrain the model through deep belief network to obtain the initialized parameters \( \theta = \{\theta^{(1)}, ..., \theta^{(K)}\} \)
2. \( X = S \)
3. repeat
4. Based on \( X \) and \( \theta \), apply Eq. 1 to obtain \( \hat{X} \) and \( Y = Y^{(K)} \).
5. \( \mathcal{L}_{mix}(X; \theta) = \| (\hat{X} - X) \odot B \|^2_F + 2\alpha \text{tr}(Y^TLY) + \nu L_{reg} \).
6. Based on Eq. 6, use \( \partial L_{mix} / \partial \theta \) to back-propagate through the entire network to get updated parameters \( \theta \).
7. until converge
8. Obtain the network representations \( Y = Y^{(K)} \)

Experimental Results

The precision keeps at least 0.9

Table 5: precision@$k$ on ARXIV GR-QC for link prediction

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>P@2</th>
<th>P@10</th>
<th>P@100</th>
<th>P@200</th>
<th>P@300</th>
<th>P@500</th>
<th>P@800</th>
<th>P@1000</th>
<th>P@10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDNE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1*</td>
<td>0.99**</td>
<td>0.97**</td>
<td>0.91**</td>
<td>0.257**</td>
</tr>
<tr>
<td>LINE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.936</td>
<td>0.74</td>
<td>0.79</td>
<td>0.2196</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>1</td>
<td>0.8</td>
<td>0.6</td>
<td>0.555</td>
<td>0.443</td>
<td>0.346</td>
<td>0.2988</td>
<td>0.293</td>
<td>0.1591</td>
</tr>
<tr>
<td>GraRep</td>
<td>1</td>
<td>0.2</td>
<td>0.04</td>
<td>0.035</td>
<td>0.033</td>
<td>0.038</td>
<td>0.035</td>
<td>0.035</td>
<td>0.019</td>
</tr>
<tr>
<td>Common Neighbor</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.96</td>
<td>0.9667</td>
<td>0.98</td>
<td>0.8775</td>
<td>0.798</td>
<td>0.192</td>
</tr>
<tr>
<td>LE</td>
<td>1</td>
<td>1</td>
<td>0.93</td>
<td>0.855</td>
<td>0.827</td>
<td>0.66</td>
<td>0.468</td>
<td>0.391</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Significantly outperforms Line at the: ** 0.01 and * 0.05 level, paired t-test.

Improve at least 15%

Different networks/tasks require different high-order proximities
- E.g., multi-scale classification (Bryan Perozzi, et al, 2017)

Examples of high-order proximities:
- E.g., equal weights, exponentially decayed weights (Katz)
- Proximities of different orders can also be arbitrarily weighted
- E.g., networks with different scales and sparsity
What is the right order?

- Existing methods can only preserve one fixed high-order proximity
- Different high-order proximities are calculated separately

How to preserve arbitrary-order proximity while guaranteeing accuracy and efficiency?

→ What is the underlying relationship between different proximities?
Problem Formulation

- High-order proximity: a polynomial function of the adjacency matrix
  \[ S = f(A) = w_1 A^1 + w_2 A^2 + \ldots + w_q A^q \]
  - \( q \): order; \( w_1 \ldots w_q \): weights, assuming to be non-negative
  - \( A \): could be replaced by other variations (such as the Laplacian matrix)

- Objective function: matrix factorization
  \[ \min_{U^*, V^*} \| S - U^* V^{*T} \|_F^2 \]
  - \( U^*, V^* \in \mathbb{R}^{N \times d} \): left/right embedding vectors
  - \( d \): dimensionality of the space

- Optimal solution: Singular Value Decomposition (SVD)
  - \([U, \Sigma, V]\): top-d SVD results
    \[ U^* = U \sqrt{\Sigma}, \quad V^* = V \sqrt{\Sigma} \]
Eigen-decomposition Reweighting

- Eigen-decomposition reweighting

**Theorem 4.2 (Eigen-Decomposition Reweighting).** If $[\lambda, x]$ is an eigen-pair of $A$, then $[F(\lambda), x]$ is an eigen-pair of $S = F(A)$.

\[
A \xrightarrow{\text{Eigen-decomposition}} \begin{bmatrix} \Lambda \\ X \end{bmatrix} \text{ Efficient!} \quad \begin{bmatrix} S \\ F(\Lambda) \end{bmatrix} \xrightarrow{\text{Eigen-decomposition}} \begin{bmatrix} F(X) \\ X \end{bmatrix} \text{ Efficient!} \\
\text{Time Consuming!} \quad \text{Polynomial } F(\cdot) \quad \text{Time Consuming!} \quad \text{Polynomial } F(\cdot)
\]

- **Insights:** high-order proximity is simply re-weighting dimensions!

\[
U^* = U\sqrt{\Sigma}, V^* = V\sqrt{\Sigma}
\]

Preserving Arbitrary-Order Proximity

- Shifting across different orders/weights:

- Preserving arbitrary-order proximity
- Low marginal cost
- Accurate and efficient

Experimental Results

- Link Prediction

Section Summary

Nodes & Links

Pair-wise Proximity

Community Structures

Hyper Edges

Global Structure

Network Characteristics

Application Characteristics

Dedicated Network Embedding Tutorial at KDD 2019:
A Survey on Network Embedding

Learning from networks

GNN
Graph Neural Networks

Can we design a learning mechanism to directly work on graphs?
The First Graph Neural Network

- Basic idea: a recursive definition of states

\[ s_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F}(s_i, s_j, F^V_i, F^V_j, F^E_{i,j}) \]

- A simple example: PageRank:

\[ s_i = \alpha \sum_{j \in \mathcal{N}(i)} \frac{s_j}{d_j} + \frac{(1-\alpha)}{n} \]

Many GNNs have emerged since then

"Spatial methods"

Original GNN
Gori et al. (2005)

GG-NN
Li et al. (ICLR 2016)

MoNet
Monti et al. (CVPR 2017)

Relation Nets
Santoro et al. (ICLR 2018)

GraphSAGE
Hamilton et al. (NIPS 2017)

Programs as Graphs
Allamanis et al. (ICLR 2017)

"DL on graph explosion"

GAT
Veličković et al. (ICLR 2018)

NRI
Kipf et al. (ICML 2018)

"Spectral methods"

GCN
Kipf & Welling (ICLR 2017)

Spectral
Graph CNN
Bruna et al. (ICLR 2015)

ChebNet
Defferrard et al. (NIPS 2016)

Other early work:
- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

(slide inspired by Alexander Gaunt's talk on GNNs)
How are GNNs compared with other NNs?

- Capture information from graph neighborhoods
- Capture information from nearby grids (i.e., a 2-D graph)
- Capture information from contexts (i.e., a 1-D graph)

We need to exchange information within neighborhoods
Message-passing Framework

- Formulation:
  \[
  m_i^{(l)} = \text{AGG}(\{h_j^{(l)}, \forall j \in \tilde{N}_i\})
  \]
  \[
  h_i^{(l+1)} = \text{UPDATE}(\{h_i^{(l)}, m_i^{(l)}\})
  \]

- \(h_i^{(l)}\): representation of node \(v_i\) in the \(l^{th}\) layer
- \(m_i^{(l)}\): messages for node \(v_i\) in the \(l^{th}\) layer by aggregating neighbor representations

Graph Convolutional Networks (GCN)

- Main idea: averaging messages from direct neighborhoods
  \[ H^{l+1} = \rho \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l \Theta^l \right) \]

- Stacking multiple layers like standard CNNs:
  - State-of-the-art results on node classification

Main idea: weight messages using attention mechanism

\[ h_i^{l+1} = \rho(\sum_{j \in N(i)} \alpha_{ij}^l h_j^l) \]
Some expected properties of GNNs:

- Trained end-to-end for downstream tasks
  - Vs. network embedding: unsupervised representation learning to handle various tasks
- Utilize node features and graph structures simultaneously
- Can handle real applications with data represented as graphs

Are existing GNNs good enough?
Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness
Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness
The intrinsic problem GCN is solving

$$H^{l+1} = \rho \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^l \Theta^l \right)$$

Fusing topology and features in the way of smoothing features with the assistance of topology.

Revisiting Graph Neural Networks All We Have is Low-Pass Filters, arXiv 1905.09550
Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology, NeurIPS 2019
The intrinsic problem GCN is solving

- Theoretical analysis: node features as “true signal”, GNNs as a low-pass filtering
  - Simplified GCN\[^{[1]}\]: removing all non-linearity
    \[
    H^{(l)} = S^l H^{(0)}
    \]
    \[
    S = \bar{D}^{-1/2} \bar{A} \bar{D}^{-1/2} = I - \bar{L}_{\text{Sym}}
    \]
  - From graph signal processing, \( f' = S^l f \) corresponds to a spectral filter\[^{[2]}\]
    \[
    g(\lambda) = (1 - \lambda)^l
    \]
  - GCNs are a special form of Laplacian smoothing for node features\[^{[3]}\]

1. Simplifying Graph Convolutional Networks, ICML 2019
2. Revisiting Graph Neural Networks All We Have is Low-Pass Filters, arXiv 1905.09550
3. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning, AAAI
Can GNNs Fully Preserve Graph Structures?

- When feature plays the key role, GNN performs good.
- How about the contrary?
- Synthesis data: stochastic block model + random features
  - DeepWalk greatly outperforms all the GCNs.
  - Recall the message-passing framework.
    \[
    \begin{align*}
    m_i^{(l)} &= \text{AGG}(\{h_j^{(l)}, \forall j \in N_i\}) \\
    h_i^{(l+1)} &= \text{UPDATE}([h_i^{(l)}, m_i^{(l)}])
    \end{align*}
    \]
    Graph structures only provide neighborhoods in aggregation.
- Initial node features provide important inductive bias!

<table>
<thead>
<tr>
<th>Method</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>10.0</td>
</tr>
<tr>
<td>GCN-1</td>
<td>29.7</td>
</tr>
<tr>
<td>GCN-2</td>
<td>48.4</td>
</tr>
<tr>
<td>GCN-3</td>
<td>56.2</td>
</tr>
<tr>
<td>GCN-5</td>
<td>53.3</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>99.9</td>
</tr>
</tbody>
</table>

GCN-X: X number of layers
A New Perspective to Understand GNNs

- A new perspective: treating GNNs as a type of (non-linear) dimensionality reduction

- A slightly modified framework:

  \[ H^{(l+1)} = \sigma \left( F(A) H^{(l)} W^{(l)} \right) \]

  - Non-linear mapping
  - Graph structures
  - Previous bases
  - Linear transform

- Three-steps
  1. Projecting graph structures into a subspace spanned by node representations in the last step
  2. The projected representations are linearly transformed followed by a non-linear mapping
  3. Repeat the process by using the new node representations as bases

- Why are the existing GNNs feature-centric?
  → The initial space is solely determined by node features!
Eigen-GNN: A Graph Structure Preserving Plug-in

- **Framework**

\[
\mathbf{H}^{(0)} = [X, f(Q)]
\]

union space of $X$ and $f(Q)$

- **Experimental results:**

<table>
<thead>
<tr>
<th>Node classification</th>
<th>Link Prediction</th>
<th>Graph Isomorphism Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Method</td>
<td>Harvard</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>A.Y</td>
<td>GCN\text{random}</td>
<td>74.6±0.5</td>
</tr>
<tr>
<td></td>
<td>GCN\text{degree}</td>
<td>74.4±2.0</td>
</tr>
<tr>
<td></td>
<td>GCN\text{DW}</td>
<td>82.5±1.0</td>
</tr>
<tr>
<td></td>
<td>Eigen-GCN</td>
<td>82.7±1.2</td>
</tr>
<tr>
<td>A.X.Y</td>
<td>GCN\text{feat}</td>
<td>70.6±1.3</td>
</tr>
<tr>
<td></td>
<td>GCN\text{feat+DW}</td>
<td>83.1±0.7</td>
</tr>
<tr>
<td></td>
<td>Eigen-GCN\text{feat+struc}</td>
<td>84.6±1.4</td>
</tr>
</tbody>
</table>

*Gain $\dagger$:

- $+1.9$
- $+0.4$
- $+3.3$

**Permutation-equivariance of GNNs**

- **Permutation-equivariance property**
  - If we randomly permute the IDs of nodes while maintaining the graph structure, the representations of nodes in GNNs should be permuted accordingly.

- **Pros:**
  - Guarantees that the representations of automorphic nodes are identical.
  - Automatically generalize to all the $O(N!)$ permutations when training with only one permutation.

- **Most of the existing message-passing GNNs satisfy permutation-equivariance.**

![Permutation-equivariant Node Embeddings](image-url)
Permutation-equivariance vs. Proximity-aware

- However, permutation-equivariance and proximity-aware are conflicting.

Structurally equivalent

But no proximity

Permutation-equivariant Node Representations

Conflict!

Proximity-preserving Node Representations


On the Equivalence between Positional Node Embeddings and Structural Graph Representations. ICLR 2020.
Unique Node Identifiers

- The key problem is we can only differentiate nodes with unique identifiers.

Theoretical analysis: unique node identifiers are one necessary condition for GNNs to be universal approximation.
Stochastic Message Passing (SMP)

- Assign stochastic features as node identifiers
- Gaussian features: associate with random projection literature
- A dual GNN architecture:

\[
H = \mathcal{F}_{\text{output}}([\tilde{\mathbf{E}}, \mathbf{H}^{(L)}])
\]

\[
\tilde{\mathbf{E}} = \mathcal{F}_{\text{GNN}}(\mathbf{A}, \mathbf{E}; \mathbf{W}), \quad \mathbf{H}^{(L)} = \mathcal{F}_{\text{GNN'}}(\mathbf{A}, \mathbf{F}; \mathbf{W}')
\]

Theoretical Guarantee

- SMP can preserve node proximities

  **Theorem 2.** An SMP in Eq. (9) with the message-passing matrix $\tilde{\mathbf{A}}$ and the number of propagation steps $K$ can preserve the walk-based proximity $\tilde{\mathbf{A}}^K (\tilde{\mathbf{A}}^K)^T$ with high probability if the dimensionality of the stochastic matrix $d$ is sufficiently large, where the superscript $T$ denotes matrix transpose. The theorem is regardless of whether $\mathbf{E}$ are fixed or resampled.

- SMP can recover the existing permutation-equivariant GNNs

  **Corollary 2.** For any task, Eq. (8) with the aforementioned linear $\mathcal{F}_{\text{output}}(\cdot)$ is at least as powerful as the permutation-equivariant $\mathcal{F}_{\text{GNN}'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$, i.e., the minimum training loss of using $\mathbf{H}$ in Eq. (8) is equal to or smaller than using $\mathbf{H}^{(L)} = \mathcal{F}_{\text{GNN}'}(\mathbf{A}, \mathbf{F}; \mathbf{W}')$.

- An adaptive GNN that maintains both proximity-awareness and permutation-equivariance

Experimental Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Community</th>
<th>Social Status</th>
<th>Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>10.0</td>
<td>50.0</td>
<td>5.0</td>
</tr>
<tr>
<td>SGC</td>
<td>10.0±0.0</td>
<td>51.0±1.4</td>
<td>5.0±0.0</td>
</tr>
<tr>
<td>GCN</td>
<td>8.7±1.1</td>
<td>91.6±1.8</td>
<td>8.9±0.9</td>
</tr>
<tr>
<td>GAT</td>
<td>10.0±0.0</td>
<td>73.4±12.9</td>
<td>5.0±0.0</td>
</tr>
<tr>
<td>P-GNN</td>
<td>64.1±4.8</td>
<td>54.9±9.8</td>
<td>5.6±1.2</td>
</tr>
<tr>
<td>SMP-Linear</td>
<td>98.8±0.6</td>
<td>93.9±0.9</td>
<td>93.8±1.6</td>
</tr>
</tbody>
</table>

Table 4: The results of link prediction on the PPA dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Hits@100</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGC</td>
<td>0.1187±0.0012</td>
</tr>
<tr>
<td>GCN</td>
<td>0.1867±0.0132</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>0.1655±0.0240</td>
</tr>
<tr>
<td>P-GNN</td>
<td>Out of Memory</td>
</tr>
<tr>
<td>Node2vec</td>
<td>0.2226±0.0083</td>
</tr>
<tr>
<td>Matrix Factorization</td>
<td>0.3229±0.0094</td>
</tr>
<tr>
<td>SMP-Identity</td>
<td>0.2018±0.0148</td>
</tr>
<tr>
<td>SMP-Linear</td>
<td>0.3582±0.0070</td>
</tr>
</tbody>
</table>

Outline

- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness
Technical challenges in real applications

Research

Robustness  Interpretability  Applicability

Application

Hot directions in computer vision:

Adversarial  Explainable  Scalable
Robustness in GNNs

- Adversarial attacks
  - Small perturbations in graph structures and node attributes lead to great changes
  - A serious concern for applying GNNs to real-world applications

Adversarial Attacks on Neural Networks for Graph Data, *KDD 2018*
Adversarial Attacks on GNNs

- **Categories**
  - **Targeted vs. Non-targeted**
    - Targeted: the attacker focuses on misclassifying some target nodes
    - Non-targeted: the attacker aims to reduce the overall model performance
  - **Direct vs. Influence**
    - Direct: the attacker can directly manipulate the edges/features of the target nodes
    - Influence: the attacker can only manipulate other nodes except the targets

- **Attacker knowledge:**

<table>
<thead>
<tr>
<th>Settings</th>
<th>Parameters</th>
<th>Predictions</th>
<th>Labels</th>
<th>Training Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>White-Box Attack (WBA)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Practical White-box Attack (PWA)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Restrict Black-box Attack (RBA)</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Robust Graph Convolutional Networks

- How to enhance the robustness of GNNs against adversarial attacks?
- Adversarial attacks in node classification
  - Connect nodes from different communities to confuse the classifier

- Distribution vs. plain vectors
  - Plain vectors cannot adapt to such changes
  - Variances can help to absorb the effects of adversarial changes
  - Gaussian distributions → Hidden representations of nodes

Gaussian based representations: variance terms absorb the effects of adversarial attacks

Variance-based Attention: Remedy the propagation of adversarial attacks

Sampling process: explicitly considers mathematical relevance between means and variances

Experimental Results

- Node Classification on Clean Datasets

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>81.5</td>
<td>70.9</td>
<td>79.0</td>
</tr>
<tr>
<td>GAT</td>
<td>83.0</td>
<td>72.5</td>
<td>79.0</td>
</tr>
<tr>
<td>RGCN</td>
<td>83.1</td>
<td>71.3</td>
<td>79.2</td>
</tr>
</tbody>
</table>

- Against Non-targeted Adversarial Attacks

Figure 2: Results of different methods when adopting Random Attack as the attack method.

Recap: Graph Neural Networks

- Message-passing framework of GNNs

Frontiers:
- Does GNN fuse *feature* and *topology* optimally?
- Technical challenges in real applications: robustness
Deep Learning has been shown to be successful in a number of domains, ranging from acoustics, images, to natural language processing. However, applying deep learning to the ubiquitous graph data is non-trivial because of the unique characteristics of graphs. Recently, substantial research efforts have been devoted to applying deep learning methods to graphs, resulting in beneficial advances in graph analysis techniques. In this survey, we comprehensively review the different types of deep learning methods on graphs. We divide the existing methods into five categories based on their model architectures and training strategies: graph recurrent neural networks, graph convolutional networks, graph autoencoders, graph reinforcement learning, and graph adversarial methods. We then provide a comprehensive overview of these methods in a systematic manner mainly by following their development history. We also analyze the differences and compositions of different methods. Finally, we briefly outline the applications in which they have been used and discuss potential future research directions.

Published in: IEEE Transactions on Knowledge and Data Engineering (Early Access)

Learning from networks

Automated Graph Machine Learning
Problems in Existing Graph Learning Methods

- Manually design architectures and hyper-parameters through trial-and-error
- Each task needs to be handled separately

Automated graph machine learning is critically needed!
A Glance of AutoML

Design ML methods → Design AutoML methods

Picture credit to Microsoft Azure Machine Learning AutoML
ML vs. AutoML

- Rely on **expert knowledge**
- **Tedious** trial-and-error
- **Low tuning efficiency**
- **Limited** by human design

- **Free human** out of the loop
- **High optimization efficiency**
- Discover & extract **patterns and combinations automatically**
Automated Graph Learning

- Automated Machine Learning on Graph
  - Graph Hyper-Parameter Optimization (HPO)
  - Graph Neural Architecture Search (NAS)

- The key: *Graph Structure!*

Various diverse graph structures may place complex impacts on graph HPO and graph NAS.
Challenge: Unique Graph ML Architecture

Data

NN architecture

Search Space
- zeroize
- skip-connect
- 1x1 conv
- 3x3 conv
- 3x3 avg pool

predefined operation set

Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017
NAS-Bench-201 Extending the Scope of Reproducible Neural Architecture Search, ICLR 2020
NAS-Bench-NLP Neural Architecture Search Benchmark for Natural Language Processing, arXiv 2020
Challenge: Complexity and diversity of graph tasks

- Link Prediction
- Community Detection
- Node Classification
- Network Distance
- Node Importance
- Graph Classification
- Graph Matching

Various graph properties

- High-order Proximity
- Permutation-equivariance
- Transitivity
- Non-transitivity

Various applications

- No single method can perfectly handle all scenarios

Various domains
Challenge: Scalability

Social Networks
- WeChat: 1.2 billion monthly active users (Sep 2020)
- Facebook: 2.8 billion active users (2020)

E-commerce Networks
- Millions of sellers, about 0.9 billion buyers, 10.6 trillion turnovers in China (2019)

Citation Networks
- 133 million authors, 277 million publications, 1.1 billion citations (AMiner, Feb 2021)

Challenge: how to handle billion-scale graphs?
Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries
Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries
Hyper-Parameter Optimization

- Goal: automatically find the optimal hyper-parameters

  
  Machine Learning Model

  ![Diagram](image)

  Optimal Hyper-parameter Configuration

  Data

  HPO

- Formulation: bi-level optimization

  \[
  \min_{\alpha \in \mathcal{A}} \mathcal{L}_{val} (W^* (\alpha), \alpha)
  \]

  s.t. \[ W^* (\alpha) = \arg \min_{W} (\mathcal{L}_{train} (W, \alpha)) \]

- Challenge: each trial of the inner loop on graph is computationally expensive, especially for large-scale graphs
AutoNE: Framework

Transfer the knowledge about optimal hyper-parameters from sampled subgraphs to the original massive graph

AutoNE: Sampling Module

**Goal**: sample representative subgraphs that share similar properties with the original large-scale graph

**Challenge**: preserve diversity of the origin graph

**Method**: multi-start random walk strategy

- Supervised: nodes with different labels
- Unsupervised: from different discovered communities, e.g., a greedy algorithm that maximizes modularity
AutoNE: Signature Extraction Module

- **Goal**: learn a vector representation for each subgraph so that knowledge can be transferred
- **Challenge**: learn comprehensive graph signatures
- **Method**: NetLSD [Tsitsulin et al. KDD18]

  - Based on spectral graph theory, heat diffusion process on a graph \( h_t(G) = tr(H_t) = tr(e^{-tL}) = \sum_j e^{-t\lambda_j} \)
**AutoNE: Meta-Learning Module**

- **Goal:** transfer knowledge about hyper-parameters of subgraphs to the original large-scale graph
- **Assumption:** two similar graphs have similar optimal hyper-parameter
- **Method:** Gaussian Process based meta-learner

\[
\ln p(f \mid X) = -\frac{1}{2} f^T K(X, X)^{-1} f - \frac{1}{2} \ln \det(K(X, X)) + \text{constant}.
\]
AutoNE: Experiments

Sampling-Based Graph ML

Factorization-Based Graph ML

Table 1: Results on a massive network with around thirty million edges, where we can only afford to run a NE algorithm on the whole network for a few times.

<table>
<thead>
<tr>
<th>Method</th>
<th>Trial 1 AUC</th>
<th>Trial 1 Time(s)</th>
<th>Trial 2 AUC</th>
<th>Trial 2 Time(s)</th>
<th>Trial 3 AUC</th>
<th>Trial 3 Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoNE</td>
<td>0.717</td>
<td>1067.9</td>
<td>0.726</td>
<td>1856.2</td>
<td>0.769</td>
<td>2641.9</td>
</tr>
<tr>
<td>Random</td>
<td>0.714</td>
<td>698.3</td>
<td>0.727</td>
<td>1426.3</td>
<td>0.715</td>
<td>2088.6</td>
</tr>
<tr>
<td>BayesOpt</td>
<td>0.715</td>
<td>702.5</td>
<td>0.714</td>
<td>1405.1</td>
<td>0.727</td>
<td>2307.7</td>
</tr>
</tbody>
</table>
Transfer the knowledge about optimal hyperparameters from the subgraphs to the original graph in an explainable way.

- **Goal**: sample representative subgraphs that share similar properties with the original large-scale graph, similar to AutoNE.
- **Challenge**: preserve diversity of the origin graph
- **Method**: multi-start random walk strategy
  - Supervised: nodes with different labels
  - Unsupervised: from different discovered communities, e.g., a greedy algorithm that maximizes modularity
Goal: Extract explainable graph features that can measure similarities.

Six explainable graph features

- Number of nodes: $|V|$
- Number of edges: $|E|$
- Number of triangles
- Global clustering coefficient: $3 \times \frac{\text{Number of triangles}}{\text{Number of triplets}}$
- Maximum total degree value
- Number of components

Similarity: Canberra Distance $g^i = d(f^i, f) = \Sigma_{k=1}^{6} \frac{|f^i_k - f_k|}{|f^i_k| + |f_k|}$
**e-AutoGR: Explainable Feature Extraction Module**

- **Goal:** Learn performance function on small sampled graphs and predict on the origin massive graph in an explainable manner

- **Method:**
  - Adopt **explainable** graph features
  - **Decorrelate the correlations** between different hyper-parameters given explainable graph features when learning the performance function
**Algorithm 1** Hyperparameter Decorrelation Weighing Regression (HyperDeco)

1: **Input:** Observed $X = [A, B]$ and performance $Y$, where $A$ denotes hyperparameters and $B$ denotes graph features.
2: **Output:** Updated parameters $\gamma$, $\Theta$.
3: Initialize parameters $\gamma^{(0)}$ and $\Theta^{(0)}$.
4: Calculate loss function with parameters $(\gamma^{(0)}, \Theta^{(0)})$.
5: Initialize the iteration variable $t \leftarrow 0$.
6: **repeat**
7: $t \leftarrow t + 1$.
8: Update $\gamma^{(t)}$ with gradient descent by fixing $\Theta$.
9: Update $\Theta^{(t)}$ with gradient descent by fixing $\gamma$.
10: Calculate loss function with parameters $(\gamma^{(t)}, \Theta^{(t)})$.
11: **until** Loss function converges or max iteration is reached.

**Algorithm 2** Explainable Automated Graph Representation (e-AutoGR)

1: **Input:** Graph $G$, Graph representation algorithm $R$.
2: **Output:** The optimal hyperparameter configuration $\Lambda^*$.
3: Sample $s$ subgraphs $G_i, i = 1, 2, \ldots, s$ from original graph $G$ according to Section 3.2.1.
4: Decide $t_i$ for each $G_i$ according to Section 3.2.2.
5: Execute algorithm $R$ on each subgraph $G_i$ for $t_i$ times and obtain hyperparameter matrix $A$ and graph feature matrix $B$ as well as the performance vector $Y$.
6: Initialize $Count = T$.
7: **repeat**
8: Execute **Step 1** to **Step 3** in Section 3.4.
9: $Count = Count - 1$.
10: **until** $Count == 0$
e-AutoGR: Experiments

Table 1. Best performance for each comparable automated graph representation approach on different datasets in terms of link prediction and node classification tasks over various graph representation algorithms. Bold font denotes the best approach.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Task</th>
<th>e-AutoGR</th>
<th>AutoNE</th>
<th>Random</th>
<th>Bayesian</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>Deepwalk</td>
<td>Link Prediction</td>
<td>0.871817</td>
<td>0.792662</td>
<td>0.803191</td>
<td>0.807158</td>
</tr>
<tr>
<td>BlogCatalog</td>
<td>Deepwalk</td>
<td>Classification</td>
<td>0.414682</td>
<td>0.414234</td>
<td>0.411551</td>
<td>0.407449</td>
</tr>
<tr>
<td>BlogCatalog</td>
<td>AROPE</td>
<td>Link Prediction</td>
<td>0.852612</td>
<td>0.851921</td>
<td>0.846578</td>
<td>0.851878</td>
</tr>
<tr>
<td>BlogCatalog</td>
<td>AROPE</td>
<td>Classification</td>
<td>0.326721</td>
<td>0.325060</td>
<td>0.326020</td>
<td>0.326428</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>Deepwalk</td>
<td>Link Prediction</td>
<td>0.729228</td>
<td>0.729330</td>
<td>0.696462</td>
<td>0.713133</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>Deepwalk</td>
<td>Classification</td>
<td>0.519657</td>
<td>0.509920</td>
<td>0.503319</td>
<td>0.502617</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>AROPE</td>
<td>Link Prediction</td>
<td>0.709392</td>
<td>0.709383</td>
<td>0.703443</td>
<td>0.707619</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>AROPE</td>
<td>Classification</td>
<td>0.529743</td>
<td>0.529011</td>
<td>0.530418</td>
<td>0.529732</td>
</tr>
<tr>
<td>Pubmed</td>
<td>Deepwalk</td>
<td>Link Prediction</td>
<td>0.873301</td>
<td>0.867633</td>
<td>0.853459</td>
<td>0.851824</td>
</tr>
<tr>
<td>Pubmed</td>
<td>Deepwalk</td>
<td>Classification</td>
<td>0.810916</td>
<td>0.810368</td>
<td>0.809199</td>
<td>0.810417</td>
</tr>
<tr>
<td>Pubmed</td>
<td>AROPE</td>
<td>Link Prediction</td>
<td>0.791435</td>
<td>0.796737</td>
<td>0.790228</td>
<td>0.790123</td>
</tr>
<tr>
<td>Pubmed</td>
<td>AROPE</td>
<td>Classification</td>
<td>0.727413</td>
<td>0.725412</td>
<td>0.725789</td>
<td>0.726411</td>
</tr>
<tr>
<td>Pubmed</td>
<td>GCN</td>
<td>Classification</td>
<td>0.708830</td>
<td>0.708712</td>
<td>0.708719</td>
<td>0.707918</td>
</tr>
</tbody>
</table>

Figure 3. Explaining hyperparameter importance in affecting performance.
e-AutoGR: Experiments
Hierarchical Synopsis: multi-level hierarchical architecture of network

Vs. sampling in AutoNE and eAutoGR:
- Better time-constrained hyperparameter tuning
- Direct transfer without the meta-learning process

HESGA-TSM

- Optimization: genetic algorithm
- Hierarchical evaluation
  - Fast evaluation to select candidates
    - Reduce training epochs
  - Full evaluation: fully train candidates models and test on validation set
- Tradeoff between efficiency and effectiveness

A Genetic Algorithm with Tree-structured Mutation for Hyperparameter Optimisation of Graph Neural Networks. arXiv 2021.
Tree-structure Mutation

**Algorithm 1 TSM with An Given Individual**

1: **Input** an individual $s$, tree, $t = []$
2: $p = \text{tree}(s)$ \hspace{1cm} \triangleright \text{pathway} p \text{ identification in tree}
3: **for** $j = 1 \rightarrow n_h$ **do** \hspace{1cm} \triangleright \text{in pathway} p, \text{the maximum depth of tree without leaf nodes} n_h
4: \hspace{1cm} $t_{total} = f(t_j)$ \hspace{1cm} \triangleright t_j \text{ denotes the times recorded in node} j, f \text{ is reciprocal function}
5: \hspace{1cm} $t$.append($f(t_j)$)
6: **end for**
7: \hspace{1cm} $t'$ \leftarrow \text{each element in} t \text{ is divided by} t_{total}$
8: $i = rws(n_h, t')$ \hspace{1cm} \triangleright \text{node} i, rws \text{ roulette wheel selection}
9: \hspace{1cm} $r \leftarrow \text{the defined value range in node} i$
10: $v = uniform(r)$ \hspace{1cm} \triangleright \text{mutated value} v$
11: \hspace{1cm} $s' \leftarrow s_i$ is replaced with $binary(v)$ \hspace{1cm} \triangleright \text{for} s, s_i \text{ the fragment of binary coding} \text{for mutation}$
12: **Output** \text{individual} $s'$

---

A Genetic Algorithm with Tree-structured Mutation for Hyperparameter Optimisation of Graph Neural Networks. arXiv 2021.
A unified framework for graph learning algorithms

\[ A_{\text{samp}} = \text{Sample}(A) \]
\[ A_{\text{agg}} = \text{Aggregate}(A_{\text{samp}}) \]
\[ X_k = f_k(X_{k-1}) = \phi(A_{\text{agg}}X_{k-1}W_k) \]
\[ = f_k(f_{k-1}(\ldots f_1(X_0))) \]

Five hyper-parameters:.
- Dimension \(d\): The dimensionality of the messages
- Length \(k\): The size of neighborhood
- Width \(w\): The number of message passing steps.
- Non-linearity \(l\): The nonlinearity in the message passing
- Aggregation strategy \(a\): How to aggregate messages

Algorithm: Bayesian optimization

Spoiler: very similar to search space of graph NAS (using HPO methods)
Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries
Neural Architecture Search (NAS)

- Goal: automatically learn the best neural architecture

Categorization

FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search, CVPR 2019
Neural Architecture Search A Survey, JMLR 2019
Graph NAS Search Space: Message-passing Framework

- Message-passing framework of GNNs

\[ m_i^{(l)} = \text{AGG}^{(l)} \left( \{ a_{ij}^{(l)} W^{(l)} h_j^{(l)}, \forall j \in \mathcal{N}(i) \} \right) \]

\[ h_i^{(l+1)} = \sigma \left( \text{COMBINE}^{(l)} \left[ m_i^{(l)}, h_i^{(l)} \right] \right) , \]

- \( h_i^{(l)} \): the representation of node \( v_i \) in the \( l^{th} \) layer
- \( m_i^{(l)} \): the received message of node \( v_i \) in the \( l^{th} \) layer

- All these choices can be searched

Graph NAS Search Space: Micro

\[ m_i^{(l)} = \text{AGG}^{(l)} \left( \left\{ a_{ij}^{(l)} W_h^{(l)} h_i^{(l)}, \forall j \in \mathcal{N}(i) \right\} \right) \]

\[ h_i^{(l+1)} = \sigma \left( \text{COMBINE}^{(l)} \left[ m_i^{(l)}, h_i^{(l)} \right] \right), \]

- Micro search space:
  - Aggregation function \( \text{AGG}(\cdot) \): how to aggregate information from neighbors
    - Requirement: not depending on orders (i.e., neighbors are regarded a set instead of a sequence)
    - Common choices: mean, max, sum, etc.
  - Aggregation weights \( a_{ij} \): the importance of different neighbors
  - Combining function \( \text{COMBINE}(\cdot) \): how to update representation
    - Common choices: CONCAT, SUM, MLP, etc.
  - Non-linearity \( \sigma(\cdot) \): Sigmoid, ReLU, tanh, etc.
  - Dimensionality of \( h_i^{(l)} \), the number of attention heads (when using attention)

Graph Neural Architecture Search, IJCAI 2020.
Graph NAS Search Space: Macro

- Macro search space: how to arrange different layers
  - Residual connection, dense connection, etc.

- Formulation:
  \[ H^{(l)} = \sum_{j<l} F_{jl} \left( H^{(j)} \right) \]
  - \( F_{jl} \): connectivity pattern from \( j^{th} \) to the \( l^{th} \) layer
  - ZERO (not connecting), IDENTITY (residual connection), MLP, etc.
Graph NAS Search Space: Pooling

- Other search spaces
  - Pooling methods: \( h_G = \text{POOL} (H) \)
    - Aggregate node-level representation into graph-level representation
  - Hyper-parameters: similar to HPO for graphs
    - Number of layers, number of epochs, optimizer, dropout rate, etc.

- Spaces for specific tasks:
  - E.g., spatial-temporal graph operators
Most previous general NAS search strategies can be directly applied
- Controller (e.g., RNN) + Reinforcement learning (RL)
- Evolutionary
- Differentiable

Controller samples architecture (e.g., as a sequence)
RL feedback rewards (e.g., validation performance) to update the controller
Most previous general NAS search strategies can be directly applied
- Controller (e.g., RNN) + Reinforcement learning (RL)
- Evolutionary
- Differentiable

Need to define how to sample parents, generate offspring, and update populations
- E.g., remove the worst individual (Real, et al., 2017), remove the oldest individual (Real, et al., 2018), or no remove (Liu, et al., 2018)
Most previous general NAS search strategies can be directly applied

- Controller (e.g., RNN) + Reinforcement learning (RL)
- Evolutionary
- Differentiable

Generate a super-network to combine operations of the search space

Continuous relaxation to make the model differentiable

Graph NAS Search Strategy

DARTS: Differentiable Architecture Search, ICLR 2019
Graph NAS Performance Estimation

- Low-fidelity training
  - Reduce number of epochs
  - Reduce training data: sample subgraphs as in HPO

- Inheriting weights
  - Challenge: parameters in graph ML (e.g., GNNs) are unlike other NNs
  - E.g., constraints by AGNN (Zhou et al., 2019)
    - Same weight shapes
    - Same attention and activation functions

- Weight sharing in differentiable NAS with one-shot model
NAS for Graph Machine Learning

- Summary of NAS for graph ML

<table>
<thead>
<tr>
<th>Method</th>
<th>Search space</th>
<th>Tasks</th>
<th>Search Strategy</th>
<th>Performance Estimation</th>
<th>Other Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphNAS [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>RNN controller + RL</td>
<td>Inherit weights</td>
</tr>
<tr>
<td>AGNN [2019]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Self-designed controller + RL</td>
<td>Inherit weights</td>
</tr>
<tr>
<td>SNAG [2020a]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>RNN controller + RL</td>
<td>Single-path one-shot</td>
</tr>
<tr>
<td>PDNAS [2020c]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Simplify the micro search space</td>
</tr>
<tr>
<td>FSE [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Support heterogeneous graphs</td>
</tr>
<tr>
<td>NAS-GNN [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Evolutionary algorithm</td>
<td>-</td>
</tr>
<tr>
<td>AutoGraph [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Evolutionary algorithm</td>
<td>-</td>
</tr>
<tr>
<td>GeneticGNN [2020b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Evolutionary algorithm</td>
<td>-</td>
</tr>
<tr>
<td>EGAN [2021a]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Sample small graphs for efficiency</td>
</tr>
<tr>
<td>NAS-GCN [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Evolutionary algorithm</td>
<td>Handle edge features</td>
</tr>
<tr>
<td>LPGNAS [2020b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Search for quantisation options</td>
</tr>
<tr>
<td>You et al. [2020b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Random search</td>
<td>Transfer across datasets and tasks</td>
</tr>
<tr>
<td>SAGS [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Self-designed algorithm</td>
<td>-</td>
</tr>
<tr>
<td>Peng et al. [2020]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>CEM-RL [2019]</td>
<td>-</td>
</tr>
<tr>
<td>GNAS[2021]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>-</td>
</tr>
<tr>
<td>AutoSTG [2021]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Search spatial-temporal modules</td>
</tr>
<tr>
<td>DSS [2021]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>-</td>
</tr>
<tr>
<td>SANE [2021b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Differentiable</td>
<td>Dynamically update search space</td>
</tr>
<tr>
<td>AutoAttend [2021b]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Evolutionary algorithm</td>
<td>Cross-layer attention</td>
</tr>
</tbody>
</table>

Table 1: A summary of different NAS methods for graph machine learnings.
Graph NAS Example: GraphNAS

- Search space: micro + macro
- Search strategy: RNN controller + RL
  - GNN architecture description: a sequence of choices
- No parameter sharing

Graph Neural Architecture Search, *IJCAI 2020*
## Graph NAS Example: GraphNAS

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

- Outperforms the existing manually designed architectures

---

Graph Neural Architecture Search, *IJCAI 2020*
The optimal architectures differ across different datasets

Graph Neural Architecture Search, IJCAI 2020
Graph NAS Example: AutoGNN

- Search space: micro
- Search strategy: reinforced conservative controller + RL
  - Conservative Explorer: maintain the best neural architecture found so far
  - Guided Architecture Modifier: modify the best architecture found so far via selecting and mutating the action classes

Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*
Performance estimation: constrained parameter sharing

Three constraints:
- Same tensor shapes
- Same attention and activation functions
- Skip connections and batch normalization do not share

Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*
# Graph NAS Example: AutoGNN

<table>
<thead>
<tr>
<th>Baseline Class</th>
<th>Model</th>
<th>#Layers</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>#Params</td>
<td>Accuracy</td>
<td>#Params</td>
</tr>
<tr>
<td>Handcrafted</td>
<td>Chebyshev</td>
<td>2</td>
<td>0.09M</td>
<td>81.2%</td>
<td>0.09M</td>
</tr>
<tr>
<td>Architectures</td>
<td>GCN</td>
<td>2</td>
<td>0.02M</td>
<td>81.5%</td>
<td>0.05M</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>2</td>
<td>0.09M</td>
<td>83.0 ± 0.7%</td>
<td>0.23M</td>
</tr>
<tr>
<td></td>
<td>LGCN</td>
<td>3 ~ 4</td>
<td>0.06M</td>
<td>83.3 ± 0.5%</td>
<td>0.05M</td>
</tr>
<tr>
<td>NAS Baselines</td>
<td>GraphNAS-w/o share</td>
<td>2</td>
<td>0.09M</td>
<td>82.7 ± 0.4%</td>
<td>0.23M</td>
</tr>
<tr>
<td></td>
<td>GraphNAS-with share</td>
<td>2</td>
<td>0.07M</td>
<td>83.3 ± 0.6%</td>
<td>1.91M</td>
</tr>
<tr>
<td></td>
<td>Random-w/o share</td>
<td>2</td>
<td>0.37M</td>
<td>81.4 ± 1.1%</td>
<td>0.95M</td>
</tr>
<tr>
<td></td>
<td>Random-with share</td>
<td>2</td>
<td>2.95M</td>
<td>82.3 ± 0.5%</td>
<td>0.95M</td>
</tr>
<tr>
<td>AGNN</td>
<td>AGNN-w/o share</td>
<td>2</td>
<td>0.05M</td>
<td>83.6 ± 0.3%</td>
<td>0.71M</td>
</tr>
<tr>
<td></td>
<td>AGNN-with share</td>
<td>2</td>
<td>0.37M</td>
<td>82.7 ± 0.6%</td>
<td>1.90M</td>
</tr>
</tbody>
</table>

Auto-GNN Neural Architecture Search of Graph Neural Networks, *arXiv 2019*
Graph NAS Example: SANG

- Search space: micro + macro
- Search strategy: RNN + RL

Simplifying Architecture Search for Graph Neural Network, CIKM 2020 workshop
Graph NAS Example: SANG

- A caution of search space: trade-off between effectiveness and efficiency
- Alternate the search space using domain knowledge and trial-and-errors

<table>
<thead>
<tr>
<th></th>
<th>Node aggregators</th>
<th>Layer aggregators</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphNAS/ Auto-GNN</td>
<td>GCN, SAGE-SUM/-MEAN/-MAX, MLP, GAT, GAT-SYM/-COS/-LINEAR/-GEN-LINEAR,</td>
<td>-</td>
<td>Hidden Embedding Size, Attention Head, Activation Function</td>
</tr>
<tr>
<td>Ours</td>
<td>All above plus SAGE-LSTM and GeniePath</td>
<td>CONCAT, MAX, LSTM</td>
<td>IDENTITY, ZERO</td>
</tr>
</tbody>
</table>
Graph NAS Example: AutoGraph

- Previous RL based NAS: a fixed number of layers

- Search space: micro + macro
- Search strategy: evolution
  - A special “Layer Add” operation
  - A flexible number of layers

AutoGraph: Automated Graph Neural Network, ICONIP 2020
Graph NAS Example: AutoGraph

Table 2. Experiment results on Cora, Citeseer and Pubmed

<table>
<thead>
<tr>
<th>Models</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chebyshev</td>
<td>81.2%</td>
<td>69.8%</td>
<td>74.4%</td>
</tr>
<tr>
<td>GCN</td>
<td>81.5%</td>
<td>70.3%</td>
<td>79.0%</td>
</tr>
<tr>
<td>GAT</td>
<td>83.0 ± 0.7%</td>
<td>72.5 ± 0.7%</td>
<td>79.0 ± 0.3%</td>
</tr>
<tr>
<td>LGCN</td>
<td>83.3 ± 0.5%</td>
<td>73.0 ± 0.6%</td>
<td>79.5 ± 0.2%</td>
</tr>
<tr>
<td>GraphNAS</td>
<td>83.3 ± 0.6%</td>
<td>73.5 ± 1.0%</td>
<td>78.8 ± 0.5%</td>
</tr>
<tr>
<td>AutoGraph</td>
<td>83.5 ± 0.4%</td>
<td>74.4 ± 0.4%</td>
<td>80.3 ± 0.3%</td>
</tr>
</tbody>
</table>

Table 3. Experiment results on PPI

<table>
<thead>
<tr>
<th>Models</th>
<th>micro-F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphSAGE (lstm)</td>
<td>0.612</td>
</tr>
<tr>
<td>GeniePath</td>
<td>0.979</td>
</tr>
<tr>
<td>GAT</td>
<td>0.973 ± 0.002</td>
</tr>
<tr>
<td>LGCN</td>
<td>0.772 ± 0.002</td>
</tr>
<tr>
<td>GraphNAS</td>
<td>0.985 ± 0.004</td>
</tr>
<tr>
<td>AutoGraph</td>
<td>0.987 ± 0.003</td>
</tr>
</tbody>
</table>
Graph NAS Example: AutoGraph

![Performance of GNNs with different layers](image)

AutoGraph: Automated Graph Neural Network, *ICONIP 2020*
Graph NAS Example: NASGNN

- Search space: micro + macro
- Search strategy: aging evolution (Real et al., AAAI 2019), RL, random search

<table>
<thead>
<tr>
<th></th>
<th>Macro</th>
<th>Micro</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Time</td>
<td>Accuracy</td>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COR</td>
<td>0.83 ± 0.007</td>
<td>0.75 ± 0.16</td>
<td>0.82 ± 0.005</td>
<td>1.73 ± 0.53</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.83 ± 0.003</td>
<td>1.45 ± 0.38</td>
<td>0.81 ± 0.001</td>
<td>2.42 ± 0.62</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.82 ± 0.003</td>
<td>0.96 ± 0.02</td>
<td>0.80 ± 0.009</td>
<td>1.20 ± 0.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIT</td>
<td>0.75 ± 0.002</td>
<td>1.18 ± 0.10</td>
<td>0.71 ± 0.007</td>
<td>2.80 ± 0.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.73 ± 0.004</td>
<td>1.52 ± 0.42</td>
<td>0.68 ± 0.006</td>
<td>2.24 ± 0.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.73 ± 0.005</td>
<td>1.05 ± 0.03</td>
<td>0.69 ± 0.006</td>
<td>1.29 ± 0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MED</td>
<td>0.82 ± 0.003</td>
<td>1.40 ± 0.37</td>
<td>0.82 ± 0.009</td>
<td>1.40 ± 0.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.80 ± 0.003</td>
<td>2.10 ± 0.14</td>
<td>0.76 ± 0.017</td>
<td>2.58 ± 0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.85 ± 0.045</td>
<td>1.31 ± 0.02</td>
<td>0.80 ± 0.009</td>
<td>1.10 ± 0.18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>0.98 ± 0.001</td>
<td>3.35 ± 0.78</td>
<td>0.99 ± 0.002</td>
<td>2.05 ± 0.48</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.95 ± 0.001</td>
<td>3.13 ± 0.11</td>
<td>0.97 ± 0.002</td>
<td>2.00 ± 0.34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.97 ± 0.001</td>
<td>1.50 ± 0.03</td>
<td>0.99 ± 0.001</td>
<td>1.58 ± 0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHY</td>
<td>0.99 ± 0.002</td>
<td>4.21 ± 0.85</td>
<td>0.99 ± 0.000</td>
<td>1.53 ± 0.15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.98 ± 0.001</td>
<td>3.34 ± 0.27</td>
<td>0.98 ± 0.001</td>
<td>2.01 ± 0.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.98 ± 0.001</td>
<td>2.08 ± 0.07</td>
<td>0.99 ± 0.001</td>
<td>1.11 ± 0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>0.91 ± 0.005</td>
<td>3.09 ± 0.49</td>
<td>0.93 ± 0.004</td>
<td>4.02 ± 1.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.90 ± 0.010</td>
<td>3.43 ± 0.21</td>
<td>0.92 ± 0.008</td>
<td>3.68 ± 0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.89 ± 0.004</td>
<td>1.69 ± 0.07</td>
<td>0.92 ± 0.002</td>
<td>2.05 ± 0.07</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHO</td>
<td>0.97 ± 0.002</td>
<td>2.48 ± 0.22</td>
<td>0.98 ± 0.004</td>
<td>1.66 ± 0.41</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.96 ± 0.005</td>
<td>3.65 ± 0.19</td>
<td>0.97 ± 0.002</td>
<td>1.88 ± 0.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.96 ± 0.002</td>
<td>1.82 ± 0.04</td>
<td>0.97 ± 0.002</td>
<td>1.08 ± 0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Graph NAS Example: Genetic-GNN

- Search space: micro + hyper-parameters
- Search strategy: alternating evolution process between GNN architectures and learning hyper-parameter

Evolutionary Architecture Search for Graph Neural Networks, *arXiv 2020*
Graph NAS Example: Stacked MPNN

- Previous graph NAS: focus on node classification
- Molecular property prediction: graph-level tasks
- Search space: micro + macro + pooling
  - Choices: global pool, global gather, global attention pool, global attention sum pool, flatten
- Search strategy: regularized evolution

Experiments:

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Stacked MPNN</th>
<th>MoleculeNet GNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>QM7 (MAE)</td>
<td>48.0±0.7</td>
<td>77.9±2.1</td>
</tr>
<tr>
<td>ESOL (RMSE)</td>
<td>0.54±0.01</td>
<td>0.58±0.03</td>
</tr>
<tr>
<td>FreeSolv (RMSE)</td>
<td>1.21±0.03</td>
<td>1.15±0.12</td>
</tr>
<tr>
<td>Lipophilicity (RMSE)</td>
<td>0.598±0.043</td>
<td>0.715±0.035</td>
</tr>
</tbody>
</table>

Graph Neural Network Architecture Search for Molecular Property Prediction, IEEE BigData 2020
Graph NAS Example: DSS

- Search space: micro
- Search strategy: differentiable

\[ I^{(j)} = \sum_{\text{ii}} o^{(i,j)} \left( x^{(i)} \right), \quad 2 \leq j \leq N - 2. \]

\[ \bar{o}^{(i,j)}(x) = \sum_{o \in O_{i,j}} \frac{\exp \left( \alpha_o^{(i,j)} \right)}{\sum_{o' \in O_{i,j}} \exp \left( \alpha_o^{(i,j)} \right)} o(x). \]

One-shot Graph Neural Architecture Search with Dynamic Search Space, AAAI 2021
Graph NAS Example: DSS

- Dynamic search space: only the top-K operations are kept after an iteration
- Basic idea: if an operation is ranked lower than the other in a subset, it is ranked lower in the universe.

Algorithm 1: Search with dynamic search space

**Input:** a set of operation candidates $\mathcal{O}$, the number of nodes $N$ in a cell, the maximum size $M$ of the candidates subset, the maximum epoch $\text{max} \_\text{epoch}$ of inner-loop to optimize a hyper-network, the number $K$ of top candidates to be remained after each iteration.

**Output:** architecture parameter $\alpha$

1. Random sample $\mathcal{O}_{i,j} \in \mathcal{O}$ of size $M$
2. while $\mathcal{O} \neq \emptyset$ do
3. Let $\mathcal{O} \leftarrow \mathcal{O} - \mathcal{O}_{i,j}$
4. Random initialize $\alpha^{(i,j)}$
5. for $\text{epoch} \in \{1, \ldots, \text{max} \_\text{epoch}\}$ do
6. Update weights $w$ with $\mathcal{L}_{\text{train}}(w, \alpha)$
7. Update architecture $\alpha$ with $\mathcal{L}_{\text{valid}}(w, \alpha)$
8. end for
9. Select top $K$ operations $\mathcal{O}_{i,j}^* \in \mathcal{O}_{i,j}$
10. Random sample $\mathcal{O}'_{i,j} \in \mathcal{O}$ of size $M - K$
11. Let $\mathcal{O}_{i,j} \leftarrow \mathcal{O}_{i,j}^* \cup \mathcal{O}'_{i,j}$
12. end while

One-shot Graph Neural Architecture Search with Dynamic Search Space, AAAI 2021
Graph NAS Example: DSS

<table>
<thead>
<tr>
<th>Model</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>semi</td>
<td>full</td>
<td>semi</td>
</tr>
<tr>
<td>GCN (Kipf and Welling 2016)</td>
<td>81.4 ± 0.5%</td>
<td>90.2 ± 0.0%</td>
<td>70.9 ± 0.5%</td>
</tr>
<tr>
<td>GAT (Veličković et al. 2017)</td>
<td>83.0 ± 0.7%</td>
<td>89.5 ± 0.3%</td>
<td>72.5 ± 0.7%</td>
</tr>
<tr>
<td>ARMA (Bianchi et al. 2019)</td>
<td>82.8 ± 0.6%</td>
<td>89.8 ± 0.1%</td>
<td>72.3 ± 1.1%</td>
</tr>
<tr>
<td>APPNP (Klicpera et al. 2018)</td>
<td>83.3 ± 0.1%</td>
<td>90.4 ± 0.2%</td>
<td>71.8 ± 0.4%</td>
</tr>
<tr>
<td>H-GCN (Hu et al. 2019)</td>
<td>79.8 ± 1.2%</td>
<td>89.7 ± 0.4%</td>
<td>70.0 ± 1.3%</td>
</tr>
<tr>
<td>AGNN (Zhou et al. 2019)</td>
<td>83.6 ± 0.3%</td>
<td>-</td>
<td>73.8 ± 0.7%</td>
</tr>
<tr>
<td>GraphNAS (Gao et al. 2020)</td>
<td>83.7 ± 0.4%</td>
<td>-</td>
<td>73.5 ± 0.3%</td>
</tr>
<tr>
<td>GraphNAS (Gao et al. 2020)</td>
<td>-</td>
<td>90.6 ± 0.3%</td>
<td>81.2 ± 0.5%</td>
</tr>
<tr>
<td>DSS (Ours)</td>
<td>83.9 ± 0.3%</td>
<td>91.0 ± 0.2%</td>
<td>73.3 ± 0.3%</td>
</tr>
</tbody>
</table>

- Competitive performance to existing GNN NAS approaches with up to 10x speedup

One-shot Graph Neural Architecture Search with Dynamic Search Space, AAAI 2021
Graph NAS Example: PDNAS

- Search space: micro + macro
- Search algorithm: differentiable

Probabilistic Dual Network Architecture Search on Graphs, *arXiv 2021*
Graph NAS Example: EGAN

- Search space: micro + macro
- Search strategy: one-shot differentiable
- Large-scale graphs: sample subgraphs as proxies (similar to AutoNE and eAutoGR)

Efficient Graph Neural Architecture Search, OpenReview 2020
Graph NAS Example: SANE

- Node aggregator: similar to the micro space
  - Search how to aggregate neighborhoods
    \[
    h_v^l = \sigma(W^l \cdot \text{AGG}_{\text{node}}(\{h_u^{l-1}, \forall u \in \tilde{N}(v)\}))
    \]

- Layer aggregator: similar to the macro space
  - Search how to aggregate different layers
    \[
    z_v = \text{AGG}_{\text{layer}}(h_v^1, \cdots, h_v^K)
    \]

- Candidate operations

| \(O_n\) | SAGE-SUM, SAGE-MEAN, SAGE-MAX, GCN, GAT, GAT-SYM, GAT-COS, GAT-LINEAR, GAT-GEN-LINEAR, GIN, GeniePath |
| \(O_l\) | CONCAT, MAX, LSTM |
| \(O_s\) | IDENTITY, ZERO |

Search to aggregate neighborhood for graph neural network, ICDE 2021
**Graph NAS Example: SANE**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>PPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>0.8811 (0.0101)</td>
<td>0.7666 (0.0202)</td>
<td>0.8858 (0.0030)</td>
<td>0.6500 (0.0000)</td>
</tr>
<tr>
<td>GCN-JK</td>
<td>0.8820 (0.0118)</td>
<td>0.7763 (0.0136)</td>
<td>0.8927 (0.0037)</td>
<td>0.8078 (0.0000)</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>0.8741 (0.0159)</td>
<td>0.7599 (0.0094)</td>
<td>0.8834 (0.0044)</td>
<td>0.6504 (0.0000)</td>
</tr>
<tr>
<td>GraphSAGE-JK</td>
<td>0.8841 (0.0015)</td>
<td>0.7654 (0.0054)</td>
<td>0.8942 (0.0066)</td>
<td>0.8019 (0.0000)</td>
</tr>
<tr>
<td>GAT</td>
<td>0.8719 (0.0163)</td>
<td>0.7518 (0.0145)</td>
<td>0.8573 (0.0066)</td>
<td>0.9414 (0.0000)</td>
</tr>
<tr>
<td>GAT-JK</td>
<td>0.8726 (0.0086)</td>
<td>0.7527 (0.0128)</td>
<td>0.8674 (0.0055)</td>
<td>0.9749 (0.0000)</td>
</tr>
<tr>
<td>GIN</td>
<td>0.8600 (0.0083)</td>
<td>0.7340 (0.0139)</td>
<td>0.8799 (0.0046)</td>
<td>0.8724 (0.0002)</td>
</tr>
<tr>
<td>GIN-JK</td>
<td>0.8699 (0.0103)</td>
<td>0.7651 (0.0133)</td>
<td>0.8878 (0.0054)</td>
<td>0.9467 (0.0000)</td>
</tr>
<tr>
<td>GeniePath</td>
<td>0.8670 (0.0123)</td>
<td>0.7594 (0.0137)</td>
<td>0.8846 (0.0039)</td>
<td>0.7138 (0.0000)</td>
</tr>
<tr>
<td>GeniePath-JK</td>
<td>0.8776 (0.0117)</td>
<td>0.7591 (0.0116)</td>
<td>0.8868 (0.0037)</td>
<td>0.9694 (0.0000)</td>
</tr>
<tr>
<td>LGCN</td>
<td>0.8687 (0.0075)</td>
<td>0.7543 (0.0221)</td>
<td>0.8753 (0.0012)</td>
<td>0.7720 (0.0020)</td>
</tr>
</tbody>
</table>

**Human-designed architectures**

**NAS approaches**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
<th>PPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.8594 (0.0072)</td>
<td>0.7062 (0.0042)</td>
<td>0.8866 (0.0010)</td>
<td>0.9517 (0.0032)</td>
</tr>
<tr>
<td>Bayesian</td>
<td>0.8835 (0.0072)</td>
<td>0.7335 (0.0006)</td>
<td>0.8801 (0.0033)</td>
<td>0.9583 (0.0082)</td>
</tr>
<tr>
<td>GraphNAS</td>
<td>0.8840 (0.0071)</td>
<td>0.7762 (0.0061)</td>
<td>0.8896 (0.0024)</td>
<td>0.9692 (0.0128)</td>
</tr>
<tr>
<td>GraphNAS-WS</td>
<td>0.8808 (0.0101)</td>
<td>0.7613 (0.0156)</td>
<td>0.8842 (0.0103)</td>
<td>0.9584 (0.0415)</td>
</tr>
</tbody>
</table>

**one-shot NAS**

| SANE           | 0.8926 (0.0123) | 0.7859 (0.0108) | 0.9047 (0.0091) | 0.9856 (0.0120) |

Search to aggregate neighborhood for graph neural network, *ICDE 2021*
Graph NAS Example: SANE

Search to aggregate neighborhood for graph neural network, *ICDE 2021*
Graph NAS Example: AutoSTG

- Tasks: NAS for spatial-temporal graphs
- Typical example: traffic prediction, time-series prediction, etc.

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, WWW 2021
Graph NAS Example: AutoSTG

- Search space: spatial convolution (SC) and temporal convolution (TC)
  - Spatial convolution: diffusion convolution
  - Temporal convolution: \( H_i' = H_i \star \mathcal{K}_i \)
  - Zero: not connection
  - Identity: for residual connections

\[
DC (H, A, W) = \sum_{k=1}^{K} \sum_{p=1}^{P} (A_k)^p H W_{kp},
\]
Graph NAS Example: AutoSTG

- Search strategy: differentiable + meta learning to generate weight parameters of SC and TC

- Graph Meta Knowledge Learner: improves upon a previous work (Pan et al., KDD 2019)
Graph NAS Example: AutoSTG

Table 3: Predictive performance on PEMS-BAY and METR-LA datasets.

<table>
<thead>
<tr>
<th></th>
<th>MAE (↓)</th>
<th>RMSE (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Overall</td>
<td>15 min</td>
</tr>
<tr>
<td>PEMSBAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HA</td>
<td>3.84±0.00</td>
<td>3.84±0.00</td>
</tr>
<tr>
<td>GBRT</td>
<td>1.96±0.02</td>
<td>1.49±0.01</td>
</tr>
<tr>
<td>GAT-Seq2Seq</td>
<td>1.74±0.00</td>
<td>1.38±0.01</td>
</tr>
<tr>
<td>DCRNN</td>
<td>1.59±0.00</td>
<td>1.31±0.00</td>
</tr>
<tr>
<td>Graph WaveNet</td>
<td>1.59±0.00</td>
<td>1.31±0.01</td>
</tr>
<tr>
<td>ST-MetaNet+</td>
<td>1.60±0.01</td>
<td>1.31±0.00</td>
</tr>
<tr>
<td><strong>AutoSTG</strong></td>
<td><strong>1.56±0.01</strong></td>
<td><strong>1.31±0.00</strong></td>
</tr>
<tr>
<td>METR-LA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HA</td>
<td>4.79±0.00</td>
<td>4.79±0.00</td>
</tr>
<tr>
<td>GBRT</td>
<td>3.86±0.01</td>
<td>3.16±0.00</td>
</tr>
<tr>
<td>GAT-Seq2Seq</td>
<td>3.28±0.00</td>
<td>2.83±0.01</td>
</tr>
<tr>
<td>DCRNN</td>
<td>3.04±0.01</td>
<td>2.67±0.00</td>
</tr>
<tr>
<td>Graph WaveNet</td>
<td>3.05±0.01</td>
<td>2.70±0.01</td>
</tr>
<tr>
<td>ST-MetaNet+</td>
<td><strong>3.00±0.01</strong></td>
<td><strong>2.65±0.01</strong></td>
</tr>
<tr>
<td><strong>AutoSTG</strong></td>
<td>3.02±0.00</td>
<td>2.70±0.01</td>
</tr>
</tbody>
</table>

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, WWW 2021
Graph NAS Example: AutoSTG

Figure 10: The number of SC in the learned architectures.

AutoSTG Neural Architecture Search for Predictions of Spatio-Temporal Graphs, *WWW 2021*
Graph NAS Example: Skeleton-based Action Recognition

- GNNs are widely used in skeleton-based action recognition
- However, all the existing methods are manually designed
- A general framework
  \[ A_D(i, j) = \frac{e^{\phi(h(x_i)) \otimes \psi(h(x_j))}}{\sum_{j=1}^{n} e^{\phi(h(x_i)) \otimes \psi(h(x_j))}} \]
- Search space:
  - Spatial convolution: \(\phi, \Psi\) as channel-wise convolution filters
  - Temporal convolution: \(\phi, \Psi\) as temporal convolution filters
- Search algorithm: modified from CEM-RL (Pourchot and Sigaud, ICLR 2019)

Learning Graph Convolutional Network for Skeleton-based Human Action Recognition by Neural Searching, AAAI 2020
Graph NAS Example: GNAS

- A new GNN paradigm: feature filtering + neighbor aggregation
  - Feature filtering: gating mechanism to control the information flow
    - Sparse filter: $\mathcal{F}_s(H) = QH, Q = \text{diag} \left( \mathcal{M}_Q([H, H_{in}]) \right)$
    - Dense filter: $\mathcal{F}_d(H) = Z \odot H, Z = \mathcal{M}_Z([H, H_{in}])$
    - Identity filter: $\mathcal{I}(H) = H$
  - Neighborhood aggregation: mean, max, sum

<table>
<thead>
<tr>
<th>GNNs</th>
<th>Approximation Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>$H_{out} \approx \mathcal{M}(L_{mean}(H_{in}))$</td>
</tr>
<tr>
<td>GIN</td>
<td>$H_{out} \approx \mathcal{M}([I(H_{in}) | F_s(H_{in}) | L_{sum}(H_{in})])$</td>
</tr>
<tr>
<td>GraphSage</td>
<td>$H_{out} \approx \mathcal{M}([I(H_{in}) | L_{mean}(H_{in})])$</td>
</tr>
<tr>
<td>GAT</td>
<td>$H_{out} \approx \mathcal{M}(F_s(L_{sum}(F_s(H_{in}))))$</td>
</tr>
<tr>
<td>GatedGCN</td>
<td>$H_{out} \approx \mathcal{M}([I(H_{in}) | F_d(L_{sum}(F_d(H_{in}))))$</td>
</tr>
</tbody>
</table>

Rethinking Graph Neural Architecture Search from Message-passing, CVPR 2021
Graph NAS Example: GNAS

- Search space:
  - Atomic operations: feature filtering and neighbor aggregation
  - Cell architecture: DAG + only one neighbor aggregation per path
    - Nodes only exchange information with first-order neighborhoods
    - Three-level search space: DAG + neighborhood aggregation + DAG

Rethinking Graph Neural Architecture Search from Message-passing, CVPR 2021
Overall framework

Graph NAS Example: GNAS

Rethinking Graph Neural Architecture Search from Message-passing, CVPR 2021
Algorithm: adaptively select depth

**Algorithm 1** Search Efficient GNN with Optimal message-passing Depth

<table>
<thead>
<tr>
<th>Input:</th>
<th>dataset $S$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>graph neural network $\mathcal{N}$</td>
</tr>
</tbody>
</table>

1. Initialize $D_o$ as half of average graph diameter of $S$
2. repeat
3. Initialize $\mathcal{N}_s$ as a search network with $D_o$-layer graph architectures
4. Optimize the architectures of $\mathcal{N}_s$ with GNAS on $S$
5. Derive a discrete sub-network of $\mathcal{N}_d$ from $\mathcal{N}_s$
6. $D_i = D_o$
7. Update $D_o$ as the number of graph architectures with at least one neighbor aggregation in $\mathcal{N}_d$
8. until $D_i = D_o$
9. return $\mathcal{N}_d$
Graph NAS Example: LPGNAS

Consider quantisation in GNNs

- Quantisation: reduce computation and memory cost

\[ h^k = \text{Act}(\text{Aggr}(\text{Atten}(a^k, \text{Linear}(w^k, h^{k-1})))) \]

\[ h^k_{\text{linear}} = Q_l(\text{Linear}(Q_w(w^k), Q_h(h^{k-1}))) \]

\[ h^k_{\text{atten}} = Q_{at}(\text{Atten}(Q_a(a^k), h^k_{\text{linear}})) \]

\[ h^k = \text{Act}(Q_{ag}(\text{Aggr}(h^k_{\text{atten}}))) \]

Learned low precision graph neural networks, 2021 *EuroMLSys workshop*
Graph NAS Example: LPGNAS

- Search space: micro + macro + quantisation

<table>
<thead>
<tr>
<th>Weights</th>
<th>Activations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantisation</td>
<td>Frac Bits</td>
</tr>
<tr>
<td>BINARY</td>
<td>0</td>
</tr>
<tr>
<td>BINARY</td>
<td>0</td>
</tr>
<tr>
<td>TERNARY</td>
<td>0</td>
</tr>
<tr>
<td>TERNARY</td>
<td>0</td>
</tr>
<tr>
<td>TERNARY</td>
<td>0</td>
</tr>
<tr>
<td>FIX1.3</td>
<td>3</td>
</tr>
<tr>
<td>FIX2.2</td>
<td>2</td>
</tr>
<tr>
<td>FIX1.5</td>
<td>5</td>
</tr>
<tr>
<td>FIX3.3</td>
<td>3</td>
</tr>
<tr>
<td>FIX2.4</td>
<td>4</td>
</tr>
<tr>
<td>FIX4.4</td>
<td>4</td>
</tr>
<tr>
<td>FIX4.4</td>
<td>4</td>
</tr>
<tr>
<td>FIX4.4</td>
<td>4</td>
</tr>
<tr>
<td>FIX4.8</td>
<td>8</td>
</tr>
<tr>
<td>FIX4.12</td>
<td>12</td>
</tr>
<tr>
<td>FIX4.12</td>
<td>12</td>
</tr>
<tr>
<td>FIX4.12</td>
<td>12</td>
</tr>
</tbody>
</table>

- Search strategy: differentiable

Learned low precision graph neural networks, 2021 EuroMLSys workshop
## Graph NAS Example: LPGNAS

<table>
<thead>
<tr>
<th>Method</th>
<th>QUAN</th>
<th>CORA Accuracy</th>
<th>CORA Size</th>
<th>CiteSeer Accuracy</th>
<th>CiteSeer Size</th>
<th>PubMed Accuracy</th>
<th>PubMed Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphSAGE</td>
<td>FLOAT</td>
<td>74.5 ± 0.0%</td>
<td>92.3KB</td>
<td>75.3 ± 0.0%</td>
<td>237.5KB</td>
<td>85.3 ± 0.1%</td>
<td>32.2KB</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>w10A12</td>
<td>74.3 ± 0.1%</td>
<td>28.8KB</td>
<td>75.1 ± 0.1%</td>
<td>74.2KB</td>
<td>85.0 ± 0.0%</td>
<td>10.1KB</td>
</tr>
<tr>
<td>GAT</td>
<td>FLOAT</td>
<td>88.9 ± 0.0%</td>
<td>369.5KB</td>
<td>75.9 ± 0.0%</td>
<td>950.3KB</td>
<td>86.1 ± 0.0%</td>
<td>129.6KB</td>
</tr>
<tr>
<td>GAT</td>
<td>W4A8</td>
<td>88.8 ± 0.1%</td>
<td>46.2KB</td>
<td>68.0 ± 0.1%</td>
<td>118.8KB</td>
<td>82.0 ± 0.0%</td>
<td>16.2KB</td>
</tr>
<tr>
<td>JKNET</td>
<td>FLOAT</td>
<td>88.7 ± 0.0%</td>
<td>214.9KB</td>
<td>75.5 ± 0.0%</td>
<td>505.2KB</td>
<td>87.6 ± 0.0%</td>
<td>94.5KB</td>
</tr>
<tr>
<td>JKNET</td>
<td>W6A8</td>
<td>88.7 ± 0.1%</td>
<td>40.3KB</td>
<td>73.2 ± 0.1%</td>
<td>94.7KB</td>
<td>86.1 ± 0.1%</td>
<td>17.7KB</td>
</tr>
<tr>
<td>PDNAS-2</td>
<td>FLOAT</td>
<td>89.3 ± 0.1%</td>
<td>192.2KB</td>
<td>76.3 ± 0.3%</td>
<td>478.6KB</td>
<td>89.1 ± 0.2%</td>
<td>72.8KB</td>
</tr>
<tr>
<td>PDNAS-3</td>
<td>FLOAT</td>
<td>89.3 ± 0.1%</td>
<td>200.0KB</td>
<td>75.5 ± 0.3%</td>
<td>494.4KB</td>
<td>89.1 ± 0.2%</td>
<td>81.4KB</td>
</tr>
<tr>
<td>PDNAS-4</td>
<td>FLOAT</td>
<td>89.8 ± 0.3%</td>
<td>205.0KB</td>
<td>75.6 ± 0.2%</td>
<td>500.0KB</td>
<td>89.2 ± 0.1%</td>
<td>102.7KB</td>
</tr>
<tr>
<td>LPGNAS</td>
<td>MIXED</td>
<td><strong>89.8 ± 0.0%</strong></td>
<td><strong>67.3KB</strong></td>
<td><strong>76.2 ± 0.1%</strong></td>
<td><strong>56.5KB</strong></td>
<td><strong>89.6 ± 0.1%</strong></td>
<td><strong>45.6KB</strong></td>
</tr>
</tbody>
</table>

Learned low precision graph neural networks, 2021 *EuroMLSys workshop*
Graph NAS Example: Design Space

- A systematic study of GNN design space and task space

(a) GNN Design Space
- Intra-layer Design: 4 dims
  - Linear
  - BatchNorm
  - Dropout
  - Activation
  - Aggregation
- Inter-layer Design: 4 dims
  - Pre-process layers
  - Layer connectivity
  - Message passing layers
  - Post-process layers

Learning Configuration: 4 dims
- Batch size
- Learning rate
- Optimizer
- Training epochs

(b) GNN Task Space
- Task Similarity Task Space
  - Anchor Model Metric
    - Performance ranking
    - Similarity to Task A

(c) Best GNN Designs Found in Different Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-process layers</th>
<th>Message passing layers</th>
<th>Post-process layers</th>
<th>Layer connectivity</th>
<th>Aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A: graph-IMDB</td>
<td>2</td>
<td>8</td>
<td>2</td>
<td>skip-sum</td>
<td>sum</td>
</tr>
<tr>
<td>Task B: node-smallworld</td>
<td>1</td>
<td>8</td>
<td>2</td>
<td>skip-sum</td>
<td>sum</td>
</tr>
<tr>
<td>Task C: node-CiteSeer</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>skip-cat</td>
<td>mean</td>
</tr>
</tbody>
</table>
Graph NAS Example: Design Space

Key results:

- Accuracy Ranking Distribution:
  - Batch Normalization: False, True
  - Dropout: 0.0, 0.3, 0.6
  - Activation: prelu, relu, swish
  - Aggregation: max, mean, sum
  - Message passing layers: 2, 4, 6, 8
  - Layer connectivity: skipcat, skipsum, stack
  - Pre-process layers: 1, 2, 3
  - Post-process layers: 1, 2, 3
  - Batch size: 16, 32, 64
  - Learning rate: 0.001, 0.01, 0.1
  - Optimizer: adam, sgd
  - Training epochs: 100, 200, 400
Graph NAS Example: Design Space

- Key results:

![Graph NAS Diagram]

- **Task A**: graph-scalefree-const-path
- **Task B**: node-CoauthorPhysics
- **Target task**: ogbg-molhiv

<table>
<thead>
<tr>
<th>Best design in our design space</th>
<th>Task A: graph-scalefree-const-path</th>
<th>Task B: node-CoauthorPhysics</th>
<th>Target task: ogbg-molhiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 8, 3, skipcat, sum)</td>
<td>0.885</td>
<td>0.968</td>
<td>0.792</td>
</tr>
<tr>
<td>Best design’s performance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Previously reported SOTA</td>
<td>N/A</td>
<td>0.930</td>
<td>0.771</td>
</tr>
<tr>
<td>Task Similarity with ogbg-molhiv</td>
<td>0.47</td>
<td>-0.61</td>
<td>1.0</td>
</tr>
<tr>
<td>Performance after transfer to ogbg-molhiv</td>
<td><strong>0.785</strong></td>
<td>0.736</td>
<td>N/A</td>
</tr>
</tbody>
</table>
AutoAttend: Overview

Design **self-attention models** automatically by searching for the best attention representations

Guan Chaoyu, Xin Wang and Wenwu Zhu.
AutoAttend: Challenges

- How to define the most suitable search space?
  - Joint optimization of attention representation and other functional components
  - The search space should be flexible and expressive
  - Relatively low complexity and high feasible architecture density
- How to consider the special characteristics of each sub-architecture in parameter sharing?
  - Parameters of key, query, value, and common feature extraction operations have different functionalities
Attention layer is defined to allow model to have attention aggregation

A set of layers with optional connections between any two layers
AutoAttend: Improve Density

- Skeleton constraint: each layer must have one connection to its previous layer
- Key-Value Constraint: the key and value should have the same input layer
- Non-Zero Constraint: the important connections should not be zero
AutoAttend: Context-aware Parameter Sharing

- One-shot super-net based optimization relaxation

$$a^* = \arg\min_{a \in A} L_{val}(a, w^*),$$

$$\text{s.t. } w^* = \arg\min_{w \in W} E_{a \sim \Gamma(A)} L_{train}(a, w),$$

- Share parameters according to their contexts

- Evolutionary search for best architectures
Considerable improvement for natural language processing and graph representation learning tasks
AutoAttend: Ablation Studies

- Ablation studies on the attention layer and context-aware parameter sharing strategies

<table>
<thead>
<tr>
<th>SPACE</th>
<th>SST</th>
<th>CORA</th>
<th>CiteSeer</th>
<th>PUBMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE</td>
<td>81.15</td>
<td>81.80</td>
<td>72.18</td>
<td>81.04</td>
</tr>
<tr>
<td>FULL</td>
<td>81.68</td>
<td>82.96</td>
<td>72.90</td>
<td>81.04</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONTEXT</th>
<th>SST</th>
<th>CORA</th>
<th>CiteSeer</th>
<th>PUBMED</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>68.68</td>
<td>77.09</td>
<td>63.68</td>
<td>72.72</td>
</tr>
<tr>
<td>SC</td>
<td>68.96</td>
<td>77.81</td>
<td>63.62</td>
<td>73.31</td>
</tr>
<tr>
<td>TC</td>
<td>69.38</td>
<td>78.50</td>
<td>64.22</td>
<td>77.54</td>
</tr>
<tr>
<td>FC</td>
<td>69.40</td>
<td>78.61</td>
<td>64.23</td>
<td>77.72</td>
</tr>
</tbody>
</table>

- Using attention layer and considering both the input layer and output layer can increase performance
Outline

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries
AutoML library on Graph

- Graph related
  - PyTorch
  - DGL
  - Paddle Graph Learning
  - BigGraph
  - graph-learn

- AutoML related
  - Avi
  - AutoKeras
  - Hyperopt
  - H2O AutoML
  - TPOT
Introduction – AutoGL

- We design the first autoML framework & toolkit for machine learning on graphs

Open source  Easy to use  Flexible to be extended

https://mn.cs.Tsinghua.edu.cn/AutoGL
https://github.com/THUMNLab/AutoGL
Modular Design

- Key modules:
  - AutoGL Dataset: manage graph datasets
  - AutoGL Solver: a high-level API to control the overall pipeline
- Five functional modules:
  - Auto Feature Engineering,
  - Neural Architecture Search,
  - Hyper-parameter Optimization
  - Model Training
  - Auto Ensemble
Feature Engineering

Data → AutoGL Dataset → Auto Feature Engineering → Neural Architecture Search → Hyper-Parameter Optimization → Model Training → Auto Ensemble

Auto Feature Engineering

Node-level
- Generators
  - Graphlet, EigenGNN, Pagerank, Onehot, ...
  - Pyg.transform
  - Filter, GBDT, ...
- Selectors
- Netlsd
- NetworkX

Graph-level

AutoGL Solver
Neural Architecture Search

Data → AutoGL Dataset → Auto Feature Engineering → Neural Architecture Search → Hyper-Parameter Optimization → Model Training → Auto Ensemble

Neural Architecture Search

Algorithms:
- Random
- One-Shot
- RL

Search Space:
- GraphNAS
- Single Path

ENAS
- Darts
- Vanilla RL
- GraphNAS
- Macro
- Micro
Hyper-Parameter Optimization

Data → AutoGL Dataset

Auto Feature Engineering → Neural Architecture Search

Hyper-Parameter Optimization

Model Training → Auto Ensemble

AutoGL Solver

Hyper-Parameter Optimization

General-Purpose

Random Grid CAMES

Bayes

Anneal TPE

Graph Aware → AutoNE
Model Training

Trainer
- Learning rate
- Epochs
- Optimizer
- Loss
- Early Stopping

Model
- Forward
- Ops & Architectures
- Dropout & Hidden
...

Currently supported models
- Node classification
- Link Prediction
- Graph classification
Ensemble

Data → AutoGL Dataset → Auto Feature Engineering → Neural Architecture Search → Hyper-Parameter Optimization → Model Training → AutoGL Solver

voting

stacking

Meta-learner
## Example Results

### Table 1: The results of node classification

<table>
<thead>
<tr>
<th>Model</th>
<th>Cora</th>
<th>CiteSeer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>80.9 ± 0.7</td>
<td>70.9 ± 0.7</td>
<td>78.7 ± 0.6</td>
</tr>
<tr>
<td>GAT</td>
<td>82.3 ± 0.7</td>
<td>71.9 ± 0.6</td>
<td>77.9 ± 0.4</td>
</tr>
<tr>
<td>GraphSAGE</td>
<td>74.5 ± 1.8</td>
<td>67.2 ± 0.9</td>
<td>76.8 ± 0.6</td>
</tr>
<tr>
<td>AutoGL</td>
<td>83.2 ± 0.6</td>
<td>72.4 ± 0.6</td>
<td>79.3 ± 0.4</td>
</tr>
</tbody>
</table>

### Table 2: The results of graph classification

<table>
<thead>
<tr>
<th>Model</th>
<th>MUTAG</th>
<th>PROTEINS</th>
<th>IMDB-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-K Pooling</td>
<td>80.8 ± 7.1</td>
<td>69.5 ± 4.4</td>
<td>71.0 ± 5.5</td>
</tr>
<tr>
<td>GIN</td>
<td>82.7 ± 6.9</td>
<td>66.5 ± 3.9</td>
<td>69.1 ± 3.7</td>
</tr>
<tr>
<td>AutoGL</td>
<td>87.6 ± 6.0</td>
<td>73.3 ± 4.4</td>
<td>72.1 ± 5.0</td>
</tr>
</tbody>
</table>

### Table 3: The results of different HPO methods for node classification

<table>
<thead>
<tr>
<th>Method</th>
<th>Trials</th>
<th>GCN</th>
<th>Cora</th>
<th>GAT</th>
<th>GAT</th>
<th>CiteSeer</th>
<th>GAT</th>
<th>GAT</th>
<th>GAT</th>
<th>GAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>80.9 ± 0.7</td>
<td>82.3 ± 0.7</td>
<td>70.9 ± 0.7</td>
<td>71.9 ± 0.6</td>
<td>78.7 ± 0.6</td>
<td>77.9 ± 0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>random</td>
<td>1</td>
<td>81.0 ± 0.6</td>
<td>81.4 ± 1.1</td>
<td>70.4 ± 0.7</td>
<td>70.1 ± 1.1</td>
<td>78.3 ± 0.8</td>
<td>76.9 ± 0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>82.0 ± 0.6</td>
<td>82.5 ± 0.7</td>
<td>71.5 ± 0.6</td>
<td>72.2 ± 0.7</td>
<td>79.1 ± 0.3</td>
<td>78.2 ± 0.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>81.8 ± 1.1</td>
<td><strong>83.2 ± 0.7</strong></td>
<td>71.1 ± 1.0</td>
<td>72.1 ± 1.0</td>
<td><strong>79.2 ± 0.4</strong></td>
<td>78.2 ± 0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPE</td>
<td>1</td>
<td>81.8 ± 0.6</td>
<td>81.9 ± 1.0</td>
<td>70.1 ± 1.2</td>
<td>71.0 ± 1.2</td>
<td>78.7 ± 0.6</td>
<td>77.7 ± 0.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>82.0 ± 0.7</td>
<td>82.3 ± 1.2</td>
<td>71.2 ± 0.6</td>
<td>72.1 ± 0.7</td>
<td>79.0 ± 0.4</td>
<td><strong>78.3 ± 0.4</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td><strong>82.1 ± 1.0</strong></td>
<td>83.2 ± 0.8</td>
<td><strong>72.4 ± 0.6</strong></td>
<td>71.6 ± 0.8</td>
<td>79.1 ± 0.6</td>
<td>78.1 ± 0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
AutoGL Plans

Incoming new features:

- DGL backend
- More large-scale graph support
  - E.g., sampling, distributed, etc.
- More graph tasks
  - E.g., heterogenous graphs, spatial-temporal graphs, etc.

Warmly welcome all feedbacks and suggestions!

Contact: autogl@tsinghua.edu.cn
Section Summary

- Graph Hyper-parameter Optimization
- Graph Neural Architecture Search
- Automated Graph Learning Libraries

Open Problems:
- Graph models for AutoML
  - E.g., regard NN as Directed Acyclic Graph (DAG)
  - E.g., using GNNs as surrogate models
- Robustness and explainability
- Hardware-aware models
- Comprehensive evaluation protocols
Automated Graph Learning Survey

Automated Machine Learning on Graphs: A Survey

Ziwei Zhang∗, Xin Wang∗ and Wenwu Zhu†
Tsinghua University, Beijing, China
zw-zhang16@mails.tsinghua.edu.cn, {xin.wang,wwzhu}@tsinghua.edu.cn

Abstract
Machine learning on graphs has been extensively studied in both academic and industry. However, as the literature on graph learning booms with a vast number of emerging methods and techniques, it becomes increasingly difficult to manually design the optimal machine learning algorithm for different graph-related tasks. To solve this critical challenge, automated machine learning (AutoML) on graphs which combines the strength of graph machine learning and AutoML together, is gaining attention from the research community. Therefore, we comprehensively survey AutoML on graphs in this paper, primarily focusing on hyper-parameter optimization (HPO) and neural architecture search.

Zitnik and Leskovec, 2017, physical simulation [Kipf et al., 2018], traffic forecasting [Li et al., 2018b; Yu et al., 2018], knowledge representation [Wang et al., 2017], drug re-purposing [Ioannidis et al., 2020; Gysi et al., 2020] and pandemic prediction [Kapoor et al., 2020] for Covid-19.

Despite the popularity of graph machine learning algorithms, the existing literature heavily relies on manual hyper-parameter or architecture design to achieve the best performance, resulting in costly human efforts when a vast number of models emerge for various graph tasks. Take GNNs as an example. At least one hundred new general-purpose architectures have been published in top-tier machine learning and data mining conferences in the year 2020 alone, not to mention cross-disciplinary researches of task-specific designs. More and more human efforts are inevitably needed if we want to have a comprehensive understanding of all the existing models.

Ziwei Zhang, Xin Wang, Wenwu Zhu.

Paper collection: https://github.com/THUMNLab/awesome-auto-graph-learning
Summary

- Learn vectorized representation of nodes/graphs
  - Preserve structures and properties
- End-to-end learning paradigms on graphs
  - Balance structures and attributes/features

Network Embedding

GNN

Automated Graph Machine Learning

- The automation of designing learning algorithms on graphs
  - Handle large-scale and complicated graph structures
Thanks!

Xin Wang
xin_wang@tsinghua.edu.cn
http://mn.cs.tsinghua.edu.cn/xinwang/

Ziwei Zhang
zwzhang@tsinghua.edu.cn
https://zw-zhang.github.io/

Wenwu Zhu
wwzhu@tsinghua.edu.cn