Subgraph Federated Learning via Spectral Methods

Alexandre Graell i Amat

Chalmers University of Technology, Gothenburg, Sweden

in collaborarion with Javad Aliakbari and Johan Östman

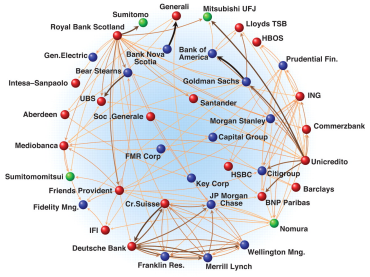
Workshop on Information Theory for Future Networks

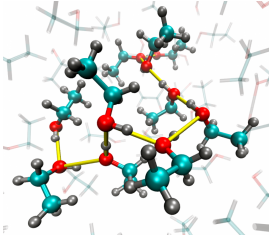
Karlsruhe Institute of Technology, Karlsruhe, Germany

March 3, 2026

CHALMERS

Graph-structured data

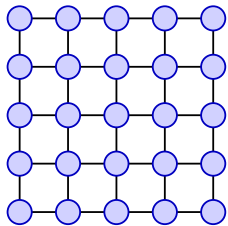




Graph neural networks

- Traditional ML designed for sequences (text, speech) or grid data (images)

fixed order, local context

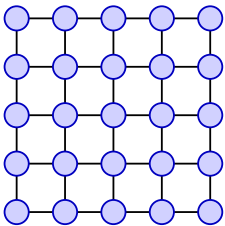


regular structure, spatial locality

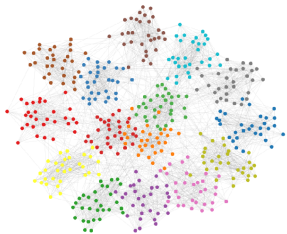
Graph neural networks

- Traditional ML designed for sequences (text, speech) or grid data (images)

fixed order, local context



regular structure, spatial locality



irregular
no canonical ordering

GNNs: Learn node representations that encode both features and structural information.

Applications

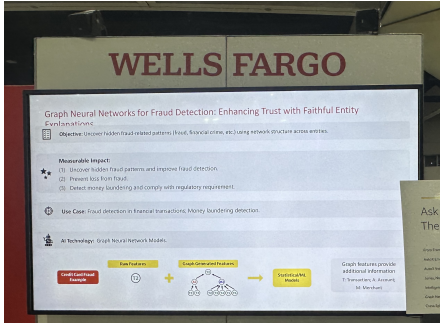
Node embeddings can be used for various machine learning tasks:

- Node-level tasks: Node classification
- Link-level tasks: Link prediction
- Graph-level tasks: Graph classification

Applications

Node embeddings can be used for various machine learning tasks:

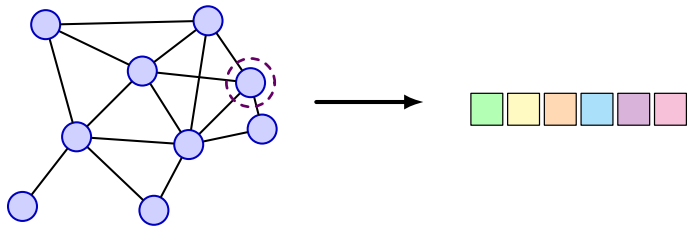
- Node-level tasks: Node classification
- Link-level tasks: Link prediction
- Graph-level tasks: Graph classification



NeurIPS 2025

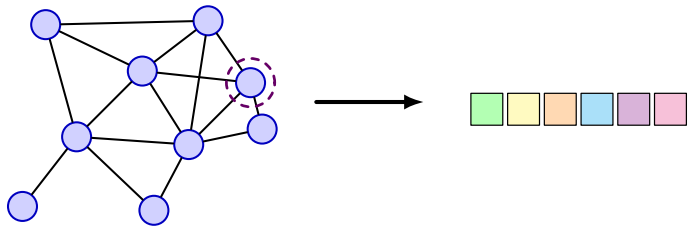
Node embeddings

Goal: Given $\mathcal{G}$, learn a low-dimensional embedding $\boldsymbol{z}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$



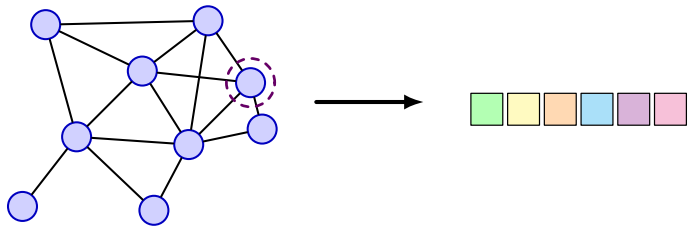
Node embeddings

Goal: Given $\mathcal{G}$, learn a low-dimensional embedding $\boldsymbol{z}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$



Node embeddings

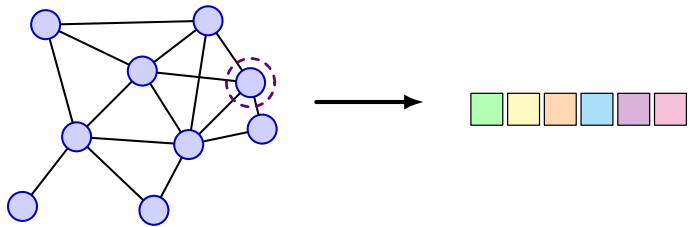
Goal: Given $\mathcal{G}$, learn a low-dimensional embedding $\boldsymbol{z}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$



How?

Node embeddings

Goal: Given $\mathcal{G}$, learn a low-dimensional embedding $\boldsymbol{z}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$

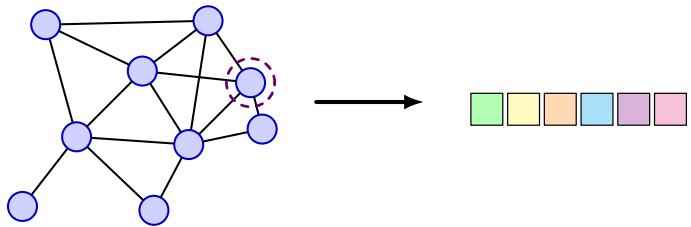


How?

- Node similarity in the graph $\longrightarrow$ similarity in embedding space

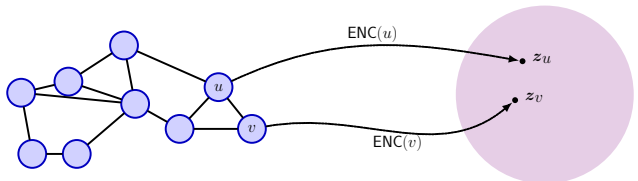
Node embeddings

Goal: Given $\mathcal{G}$, learn a low-dimensional embedding $\boldsymbol{z}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$

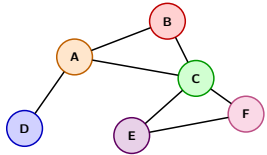


How?

- Node similarity in the graph $\longrightarrow$ similarity in embedding space



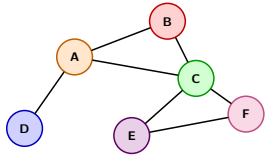
Graph neural networks

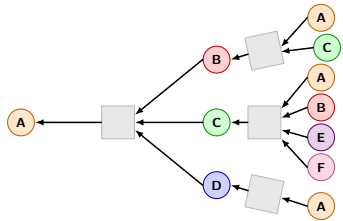


Core principle: Compute node representation by aggregating information from neighbors via message passing.

:

Graph neural networks

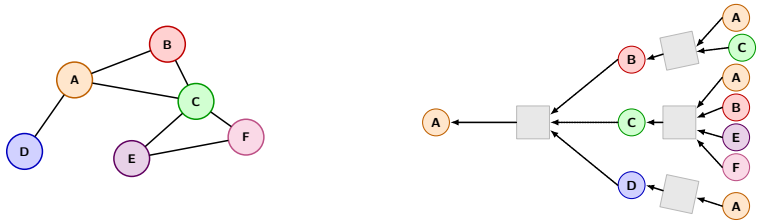




Core principle: Compute node representation by aggregating information from neighbors via message passing.

:

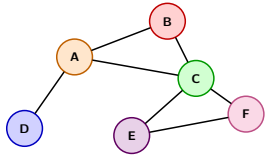
Graph neural networks

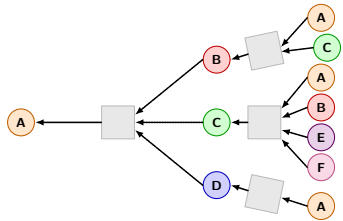


Core principle: Compute node representation by aggregating information from neighbors via message passing.

Two key operations at each layer:

Graph neural networks



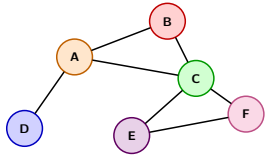


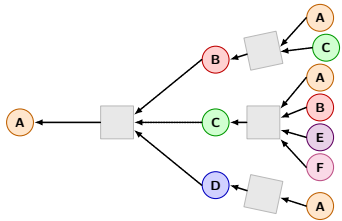
Core principle: Compute node representation by aggregating information from neighbors via message passing.

Two key operations at each layer:

- Aggregation: Nodes collect information from neighbors

Graph neural networks



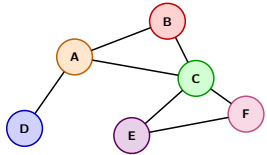


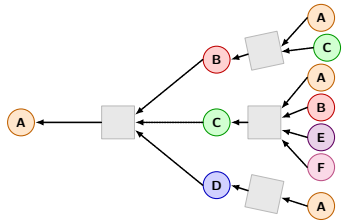
Core principle: Compute node representation by aggregating information from neighbors via message passing.

Two key operations at each layer:

- Aggregation: Nodes collect information from neighbors
- Update: Node embeddings revised based on aggregated message

Graph neural networks





Core principle: Compute node representation by aggregating information from neighbors via message passing.

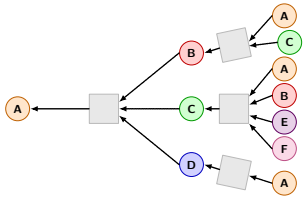
Two key operations at each layer:

- Aggregation: Nodes collect information from neighbors
- Update: Node embeddings revised based on aggregated message
- GNN depth determines receptive field: L layers $\rightarrow$ L-hop neighborhood

Neural message passing

Notation:

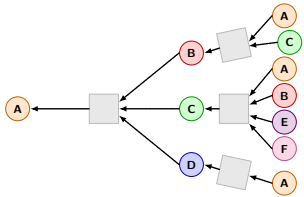
- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



At each layer ℓ, two steps:

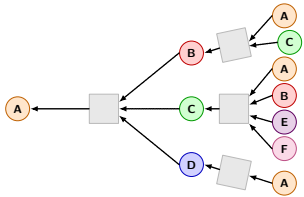
1. Aggregation step: Embeddings from neighbors passed to v and combined

$$\boldsymbol{m}_v^{(\ell)} = \text{Agg}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in \mathcal{N}(v)\right\}\right)$$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



At each layer ℓ, two steps:

1. Aggregation step: Embeddings from neighbors passed to v and combined

$$\boldsymbol{m}_v^{(\ell)} = \text{Agg}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in \mathcal{N}(v)\right\}\right)$$

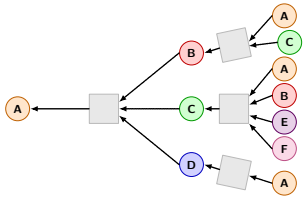
2. Update step: Embedding of node v is updated

$$\boldsymbol{h}_v^{(\ell+1)} = \text{Update}\left(\boldsymbol{h}_v^{(\ell)}, \boldsymbol{m}_v^{(\ell)}\right)$$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



At each layer ℓ, two steps:

1. Aggregation step: Embeddings from neighbors passed to v and combined

$$\boldsymbol{m}_v^{(\ell)} = \text{Agg}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in \mathcal{N}(v)\right\}\right)$$

2. Update step: Embedding of node v is updated

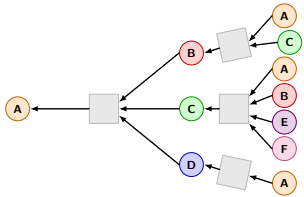
$$\boldsymbol{h}_v^{(\ell+1)} = \text{Update}\left(\boldsymbol{h}_v^{(\ell)}, \boldsymbol{m}_v^{(\ell)}\right)$$

Aggregate and update are arbitrary differentiable functions (neural networks).

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



At each layer ℓ, two steps:

1. Aggregation step: Embeddings from neighbors passed to v and combined

$$\boldsymbol{m}_v^{(\ell)} = \text{Agg}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in \mathcal{N}(v)\right\}\right) = \frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u^{(\ell)}$$

2. Update step: Embedding of node v is updated

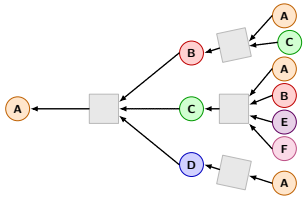
$$\boldsymbol{h}_v^{(\ell+1)} = \text{Update}\left(\boldsymbol{h}_v^{(\ell)}, \boldsymbol{m}_v^{(\ell)}\right)$$

Aggregate and update are arbitrary differentiable functions (neural networks).

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



At each layer ℓ, two steps:

1. Aggregation step: Embeddings from neighbors passed to v and combined

$$\boldsymbol{m}_v^{(\ell)} = \text{Agg}\left(\left\{\boldsymbol{h}_u^{(\ell)}, u \in \mathcal{N}(v)\right\}\right) = \frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} \boldsymbol{h}_u^{(\ell)}$$

2. Update step: Embedding of node v is updated

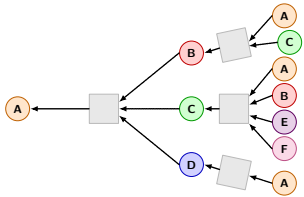
$$\boldsymbol{h}_v^{(\ell+1)} = \text{Update}\left(\boldsymbol{h}_v^{(\ell)}, \boldsymbol{m}_v^{(\ell)}\right) = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{m}_v^{(\ell)} + \boldsymbol{B}^{(\ell)}\boldsymbol{h}_v^{(\ell)}\right)$$

Aggregate and update are arbitrary differentiable functions (neural networks).

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



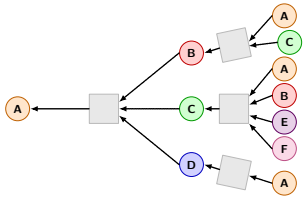
- Node embedding at layer $\ell + 1$:

$$\boldsymbol{h}_v^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)} \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(\ell)}}{|\mathcal{N}(v)|} + \boldsymbol{B}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



- Node embedding at layer $\ell + 1$:

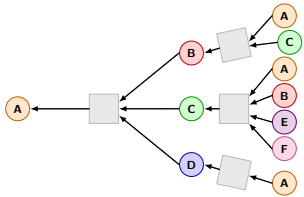
$$\boldsymbol{h}_v^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)} \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(\ell)}}{|\mathcal{N}(v)|} + \boldsymbol{B}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

- Final node embedding: $\boldsymbol{h}_v^{(L)}$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



- Node embedding at layer $\ell + 1$:

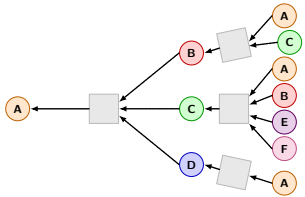
$$\boldsymbol{h}_v^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)} \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(\ell)}}{|\mathcal{N}(v)|} + \boldsymbol{B}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

- Final node embedding: $\boldsymbol{h}_v^{(L)}$
- Node classification: $\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)}\right)$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



- Node embedding at layer $\ell + 1$:

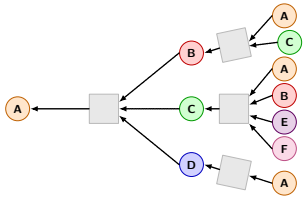
$$\boldsymbol{h}_v^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)} \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(\ell)}}{|\mathcal{N}(v)|} + \boldsymbol{B}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

- Final node embedding: $\boldsymbol{h}_v^{(L)}$
- Node classification: $\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)}\right)$ $\quad (\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{c}}(\boldsymbol{\theta}) \triangleq \sum_{v \in \mathcal{V}_{\text{labeled}}} \text{CE}\left(\boldsymbol{y}_v, \hat{\boldsymbol{y}}_v\right))$

Neural message passing

Notation:

- $\boldsymbol{x}_v$: feature vector of node v
- $\boldsymbol{h}_v^{(\ell)}$: embedding of node v at layer ℓ
- $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$



- Node embedding at layer $\ell + 1$:

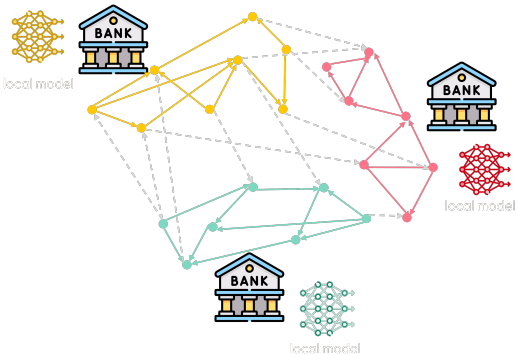
$$\boldsymbol{h}_v^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)} \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h}_u^{(\ell)}}{|\mathcal{N}(v)|} + \boldsymbol{B}^{(\ell)} \boldsymbol{h}_v^{(\ell)}\right)$$

- Final node embedding: $\boldsymbol{h}_v^{(L)}$
- Node classification: $\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)}\right)$ $\quad (\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{c}}(\boldsymbol{\theta}) \triangleq \sum_{v \in \mathcal{V}_{\text{labeled}}} \text{CE}\left(\boldsymbol{y}_v, \hat{\boldsymbol{y}}_v\right))$

Important: Same parameters $\boldsymbol{W}^{(\ell)}, \boldsymbol{B}^{(\ell)}$ shared across all nodes!

The need of federated learning

How do we train a machine learning model on data distributed across several clients while preserving data privacy?



Federated learning

central server

Federated learning [1]:

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

Federated learning

Federated learning [1]:

- Training a global model in a distributed fashion across clients

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

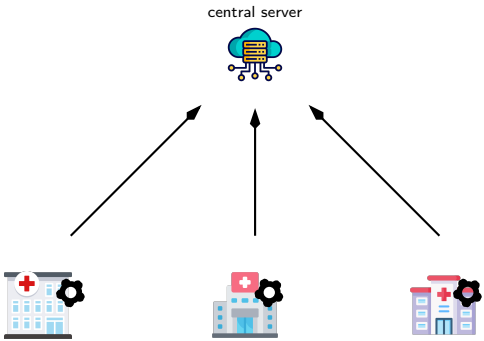
Federated learning

Federated learning [1]:

- Training a global model in a distributed fashion across clients
- Clients train models locally

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

Federated learning



Federated learning [1]:

- Training a global model in a distributed fashion across clients
- Clients train models locally
- Clients send model updates to central server

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

Federated learning

Federated learning [1]:

- Training a global model in a distributed fashion across clients
- Clients train models locally
- Clients send model updates to central server
- Central server aggregates received models and broadcasts result back to clients

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

Federated learning

Federated learning [1]:

- Training a global model in a distributed fashion across clients
- Clients train models locally
- Clients send model updates to central server
- Central server aggregates received models and broadcasts result back to clients

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

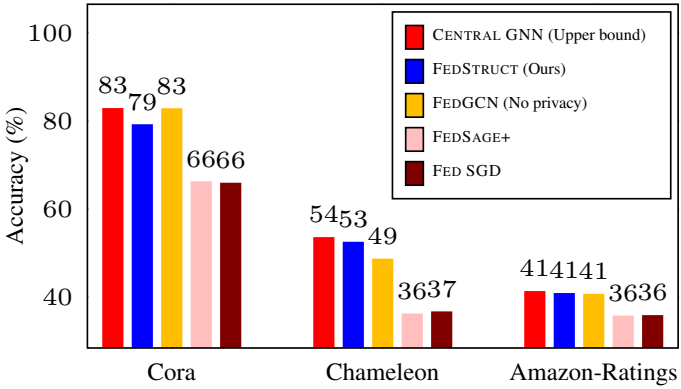
Federated learning

Federated learning [1]:

- Training a global model in a distributed fashion across clients
- Clients train models locally
- Clients send model updates to central server
- Central server aggregates received models and broadcasts result back to clients
- Multiple rounds

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication- efficient learning of deep networks from decentralized data," AISTATS, 2017.

Results



- 10 clients

Subgraph federated learning

Interconnected subgraphs: Interconnections play critical role!

Subgraph federated learning

Interconnected subgraphs: Interconnections play critical role!

- Standard FL does not work: Message passing requires access to node features and embeddings of neighbors

Subgraph federated learning

Interconnected subgraphs: Interconnections play critical role!

- Standard FL does not work: Message passing requires access to node features and embeddings of neighbors

Current approaches:

- In-painting: Estimates node features for neighboring nodes residing on other clients [Zhang *et al.*, NeurIPS 2021], [Peng *et al.*, IEEE Trans. MI], [Zhang *et al.*, FedGraph 2022], [Zhang *et al.*, SDM 2024]

Subgraph federated learning

Interconnected subgraphs: Interconnections play critical role!

- Standard FL does not work: Message passing requires access to node features and embeddings of neighbors

Current approaches:

- In-painting: Estimates node features for neighboring nodes residing on other clients [Zhang *et al.*, NeurIPS 2021], [Peng *et al.*, IEEE Trans. MI], [Zhang *et al.*, FedGraph 2022], [Zhang *et al.*, SDM 2024]
- Secure aggregation: Shares aggregated node features with other clients [Yao *et al.*, NeurIPS 2024]

Subgraph federated learning

Interconnected subgraphs: Interconnections play critical role!

- Standard FL does not work: Message passing requires access to node features and embeddings of neighbors

Current approaches:

- In-painting: Estimates node features for neighboring nodes residing on other clients [Zhang *et al.*, NeurIPS 2021], [Peng *et al.*, IEEE Trans. MI], [Zhang *et al.*, FedGraph 2022], [Zhang *et al.*, SDM 2024]
- Secure aggregation: Shares aggregated node features with other clients [Yao *et al.*, NeurIPS 2024]

But no privacy!

FedStruct

Observation: Graph structure is informative, less sensitive, and may be used to train strong classifiers.

FedStruct

Observation: Graph structure is informative, less sensitive, and may be used to train strong classifiers.

FedStruct: Harness explicit information about the global graph's structure to improve node label prediction while ensuring that neither the server nor the clients have access to the node features.

J. Aliakbari, J. Östman, and A. Graell i Amat, "Decoupled Subgraph Federated Learning," ICLR 2025.

FedStruct

Observation: Graph structure is informative, less sensitive, and may be used to train strong classifiers.

FedStruct: Harness explicit information about the global graph's structure to improve node label prediction while ensuring that neither the server nor the clients have access to the node features.

Node prediction:

$$\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)}\right)$$

J. Aliakbari, J. Östman, and A. Graell i Amat, "Decoupled Subgraph Federated Learning," ICLR 2025.

FedStruct

Observation: Graph structure is informative, less sensitive, and may be used to train strong classifiers.

FedStruct: Harness explicit information about the global graph's structure to improve node label prediction while ensuring that neither the server nor the clients have access to the node features.

Node prediction:

$$\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)}\right) \qquad \Longrightarrow \qquad \hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v^{(L)} + \boldsymbol{z}_v\right)$$

- $\boldsymbol{h}_v$: Node feature embeddings, computed locally, $\boldsymbol{h}_v = f_{\boldsymbol{\theta}_\mathsf{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v)$
- $\boldsymbol{z}_v$: Node structure embeddings (encode nodes' structural information; clients need to collaborate)

J. Aliakbari, J. Östman, and A. Graell i Amat, "Decoupled Subgraph Federated Learning," ICLR 2025.

Node structure embeddings

- To generate NSEs, FedStruct relies on node structure features (encapsulate structural information about the nodes)

$$\boldsymbol{s}_v = \boldsymbol{Q}_{\mathsf{NSF}}(v, \mathcal{E})$$

$\boldsymbol{Q}_{\mathsf{NSF}}$: Can be defined through various node embedding algorithms (one-hot degree vector, node2vec, ...)

Node structure embeddings

- To generate NSEs, FedStruct relies on node structure features (encapsulate structural information about the nodes)

$$\boldsymbol{s}_v = \boldsymbol{Q}_{\mathsf{NSF}}(v, \mathcal{E})$$

$\boldsymbol{Q}_{\mathsf{NSF}}$: Can be defined through various node embedding algorithms (one-hot degree vector, node2vec, ...)

Node structure features:

$$\boldsymbol{z}_v = \sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_{\mathsf{s}}}(\boldsymbol{s}_u)$$

Node structure embeddings

- To generate NSEs, FedStruct relies on node structure features (encapsulate structural information about the nodes)

$$\boldsymbol{s}_v = \boldsymbol{Q}_{\mathsf{NSF}}(v, \mathcal{E})$$

$\boldsymbol{Q}_{\mathsf{NSF}}$: Can be defined through various node embedding algorithms (one-hot degree vector, node2vec, ...)

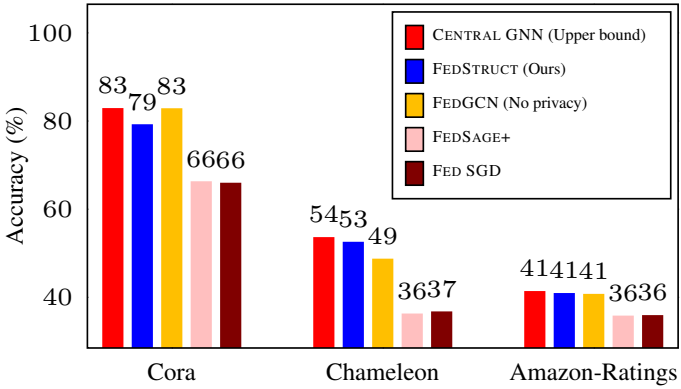
Node structure features:

$$\boldsymbol{z}_v = \sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_{\mathsf{s}}}(\boldsymbol{s}_u)$$

Node prediction:

$$\hat{\boldsymbol{y}}_v = \mathsf{softmax}\Big(\underbrace{f_{\boldsymbol{\theta}_{\mathsf{f}}}(\boldsymbol{X}_i, \mathcal{E}_i, v)}_{\boldsymbol{h}_v} + \underbrace{\sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_{\mathsf{s}}}(\boldsymbol{s}_u)}_{\boldsymbol{z}_v} \Big)$$

Results



- 10 clients

FedStruct's limitations

FedStruct requires:

FedStruct's limitations

FedStruct requires:

- $\bar{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$: Can be computed offline in a decentralized fashion, but requires significant communication between clients and potential privacy leakage

FedStruct's limitations

FedStruct requires:

- $\bar{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$: Can be computed offline in a decentralized fashion, but requires significant communication between clients and potential privacy leakage
- $\boldsymbol{S} \in \mathbb{R}^{n \times d_s}$ $(\boldsymbol{s}_v)$: Optimized during training $\longrightarrow$ communication intensive and exposes gradients of $\boldsymbol{S}$ to clients (potential privacy leakage)

FedStruct's limitations

FedStruct requires:

- $\bar{\boldsymbol{A}} \in \mathbb{R}^{n \times n}$: Can be computed offline in a decentralized fashion, but requires significant communication between clients and potential privacy leakage
- $\boldsymbol{S} \in \mathbb{R}^{n \times d_s}$ $(\boldsymbol{s}_v)$: Optimized during training $\longrightarrow$ communication intensive and exposes gradients of $\boldsymbol{S}$ to clients (potential privacy leakage)

communication cost, no formal privacy guarantees

FedLap: Subgraph federated learning via spectral methods

J. Aliakbari, J. Östman, A. Panahi, and A. Graell i Amat, "Subgraph Federated Learning via Spectral Methods," NeurIPS, 2025.

FedLap: Subgraph federated learning via spectral methods

Core idea: Leverage structural information through Laplacian smoothing.

J. Aliakbari, J. Östman, A. Panahi, and A. Graell i Amat, "Subgraph Federated Learning via Spectral Methods," NeurIPS, 2025.

FedLap

Recall (FedStruct):

$$\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v + \boldsymbol{z}_v\right) = \text{softmax}\Big(f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v) + \sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_u)\Big)$$

- $\boldsymbol{h}_v$: Node feature embeddings (local)
- $\boldsymbol{z}_v$: Node structure embeddings (clients share some structural information)

FedLap

Recall (FedStruct):

$$\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v + \boldsymbol{z}_v\right) = \text{softmax}\Big(f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v) + \sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_u)\Big)$$

- $\boldsymbol{h}_v$: Node feature embeddings (local)
- $\boldsymbol{z}_v$: Node structure embeddings (clients share some structural information)

FedLap's node prediction:

$$\hat{\boldsymbol{y}}_v = \text{softmax}\Big(\underbrace{f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v)}_{\boldsymbol{h}_v} + \underbrace{\boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_v)}_{\boldsymbol{z}_v}\Big)$$

with $\boldsymbol{\theta} = (\boldsymbol{\theta}_\text{f}, \boldsymbol{\theta}_\text{s}, \boldsymbol{S})$ optimized during training

FedLap

Recall (FedStruct):

$$\hat{\boldsymbol{y}}_v = \text{softmax}\left(\boldsymbol{h}_v + \boldsymbol{z}_v\right) = \text{softmax}\Big(f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v) + \sum_{u \in \mathcal{V}} \bar{\boldsymbol{A}}_{vu} \boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_u)\Big)$$

- $\boldsymbol{h}_v$: Node feature embeddings (local)
- $\boldsymbol{z}_v$: Node structure embeddings (clients share some structural information)

FedLap's node prediction:

$$\hat{\boldsymbol{y}}_v = \text{softmax}\Big(\underbrace{f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v)}_{\boldsymbol{h}_v} + \underbrace{\boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_v)}_{\boldsymbol{z}_v}\Big)$$

with $\boldsymbol{\theta} = (\boldsymbol{\theta}_\text{f}, \boldsymbol{\theta}_\text{s}, \boldsymbol{S})$ optimized during training (doesn't depend on $\bar{\boldsymbol{A}}$)

FedLap

$$\hat{\boldsymbol{y}}_v = \text{softmax}\Big(f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v) + \boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_u)\Big) \qquad \text{with} \qquad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L}_\text{c}(\boldsymbol{\theta}), \ \ \boldsymbol{\theta} = (\boldsymbol{\theta}_\text{f}, \boldsymbol{\theta}_\text{s}, \boldsymbol{S})$$

- But doesn't work! (without any constraint, $\boldsymbol{S}$ optimized only for labeled training nodes $\longrightarrow$ overfitting!)

FedStruct: $\bar{\boldsymbol{A}}_{uv}$ puts constraints between neighboring nodes

FedLap

$$\hat{\boldsymbol{y}}_v = \text{softmax}\Big(f_{\boldsymbol{\theta}_\text{f}}(\boldsymbol{X}_i, \mathcal{E}_i, v) + \boldsymbol{g}_{\boldsymbol{\theta}_\text{s}}(\boldsymbol{s}_u)\Big) \qquad \text{with} \qquad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L}_\text{c}(\boldsymbol{\theta}), \ \ \boldsymbol{\theta} = (\boldsymbol{\theta}_\text{f}, \boldsymbol{\theta}_\text{s}, \boldsymbol{S})$$

- But doesn't work! (without any constraint, $\boldsymbol{S}$ optimized only for labeled training nodes $\longrightarrow$ overfitting!)

FedStruct: $\bar{\boldsymbol{A}}_{uv}$ puts constraints between neighboring nodes

FedLap: Constrains $\boldsymbol{S}$ introducing a Laplacian smoothing regularizer:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_\text{c}(\boldsymbol{\theta}) + \lambda_\text{reg} \frac{\text{Tr}(\boldsymbol{S}^\top \boldsymbol{L}_\mathcal{G} \boldsymbol{S})}{\text{Tr}(\boldsymbol{S}^\top \boldsymbol{S})}, \qquad \text{where} \quad \boldsymbol{L}_\mathcal{G} = \boldsymbol{D} - \boldsymbol{A}$$

Laplacian smoothing

Observation: Neighboring nodes tend to behave similarly.

Laplacian smoothing

Observation: Neighboring nodes tend to behave similarly.

Laplacian Smoothing: Graph-based regularization that encourages similar representations for neighboring nodes.

Laplacian smoothing

Observation: Neighboring nodes tend to behave similarly.

Laplacian Smoothing: Graph-based regularization that encourages similar representations for neighboring nodes.

$$\mathcal{L}_{\text{reg}} = \sum_{u,v} \boldsymbol{A}_{uv} \| f_{\boldsymbol{\theta}}(\boldsymbol{x}_u) - f_{\boldsymbol{\theta}}(\boldsymbol{x}_v) \|^2$$

$f_{\boldsymbol{\theta}}(\cdot)$: NN-based differentiable function
$\boldsymbol{A}_{uv}$: connection weight between node u and v

Laplacian smoothing

Observation: Neighboring nodes tend to behave similarly.

Laplacian Smoothing: Graph-based regularization that encourages similar representations for neighboring nodes.

$$\mathcal{L}_{\text{reg}} = \sum_{u,v} \boldsymbol{A}_{uv} \| f_{\boldsymbol{\theta}}(\boldsymbol{x}_u) - f_{\boldsymbol{\theta}}(\boldsymbol{x}_v) \|^2 = \mathsf{Tr}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}) \boldsymbol{L}_{\mathcal{G}}^{\mathsf{T}} f_{\boldsymbol{\theta}}(\boldsymbol{X}) \right)$$

$f_{\boldsymbol{\theta}}(\cdot)$: NN-based differentiable function
$\boldsymbol{A}_{uv}$: connection weight between node u and v

FedLap

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathsf{reg}} \frac{\sum_{(u,v)\in\mathcal{E}} \|\boldsymbol{s}_u - \boldsymbol{s}_v\|^2}{\sum_{v\in\mathcal{V}} \|\boldsymbol{s}_v\|^2}$$

- The regularizer encourages neighboring nodes to have similar structure embeddings
- Avoids overfitting on training nodes

Lower communication, better privacy.

FedLap

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathsf{reg}} \frac{\sum_{(u,v)\in\mathcal{E}} \|\boldsymbol{s}_u - \boldsymbol{s}_v\|^2}{\sum_{v\in\mathcal{V}} \|\boldsymbol{s}_v\|^2}$$

- The regularizer encourages neighboring nodes to have similar structure embeddings
- Avoids overfitting on training nodes

Lower communication, better privacy.

- But requires exchanging $\boldsymbol{s}_v \in \mathcal{V}_i^*$ $\longrightarrow$ potential privacy leakage

FedLap+

Idea: Move to the spectral domain.

FedLap+

Idea: Move to the spectral domain.

- $\boldsymbol{L}_{\mathcal{G}}$ is symmetric and positive semidefinite. Can be decomposed as $\boldsymbol{L}_G = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$

FedLap+

Idea: Move to the spectral domain.

- $\boldsymbol{L}_{\mathcal{G}}$ is symmetric and positive semidefinite. Can be decomposed as $\boldsymbol{L}_G = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$
- Can rewrite loss function as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\mathsf{Tr}(\boldsymbol{W}^\top \boldsymbol{\Lambda} \boldsymbol{W})}{\mathsf{Tr}(\boldsymbol{W}^\top \boldsymbol{W})}, \qquad \text{where} \quad \boldsymbol{W} = \boldsymbol{U}^\top \boldsymbol{S}$$

FedLap+

Idea: Move to the spectral domain.

- $\boldsymbol{L}_{\mathcal{G}}$ is symmetric and positive semidefinite. Can be decomposed as $\boldsymbol{L}_G = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$
- Can rewrite loss function as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{\Lambda} \boldsymbol{W})}{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{W})}, \qquad \text{where} \quad \boldsymbol{W} = \boldsymbol{U}^\top \boldsymbol{S}$$

Global structure captured by $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1 \leq \cdots \leq \lambda_n)$

FedLap+

Idea: Move to the spectral domain.

- $\boldsymbol{L}_{\mathcal{G}}$ is symmetric and positive semidefinite. Can be decomposed as $\boldsymbol{L}_G = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$
- Can rewrite loss function as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{\Lambda} \boldsymbol{W})}{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{W})}, \qquad \text{where} \quad \boldsymbol{W} = \boldsymbol{U}^\top \boldsymbol{S}$$

Global structure captured by $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1 \leq \cdots \leq \lambda_n)$

$\mathcal{L}(\boldsymbol{\theta})$ can be written as

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\sum_{j=1}^{n} \lambda_j \|\boldsymbol{w}_j\|^2}{\sum_{j=1}^{n} \|\boldsymbol{w}_j\|^2}$$

FedLap+

Idea: Move to the spectral domain.

- $\boldsymbol{L}_{\mathcal{G}}$ is symmetric and positive semidefinite. Can be decomposed as $\boldsymbol{L}_G = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$
- Can rewrite loss function as:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{\Lambda} \boldsymbol{W})}{\mathrm{Tr}(\boldsymbol{W}^\top \boldsymbol{W})}, \qquad \text{where} \quad \boldsymbol{W} = \boldsymbol{U}^\top \boldsymbol{S}$$

Global structure captured by $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n]$ and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1 \leq \cdots \leq \lambda_n)$

$\mathcal{L}(\boldsymbol{\theta})$ can be written as

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{c}}(\boldsymbol{\theta}) + \lambda_{\mathrm{reg}} \frac{\sum_{j=1}^{n} \lambda_j \|\boldsymbol{w}_j\|^2}{\sum_{j=1}^{n} \|\boldsymbol{w}_j\|^2}$$

Intuition: Large λ_j (high-frequency components) are penalized more $\Rightarrow$ energy moves to small λ (smooth) components

FedLap+

Idea: Retain the r most informative spectral components

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

- Graph Laplacian approximated as:

$$\boldsymbol{L}_G \approx \boldsymbol{U}_{:,[r]} \boldsymbol{\Lambda}_{[r],[r]} \boldsymbol{U}_{:,[r]}^\top \quad \text{and} \quad \boldsymbol{S} \approx \boldsymbol{U}_{:,[r]} \boldsymbol{W}_{[r],:}$$

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

- Graph Laplacian approximated as:

$$\boldsymbol{L}_G \approx \boldsymbol{U}_{:,[r]} \boldsymbol{\Lambda}_{[r],[r]} \boldsymbol{U}_{:,[r]}^\top \quad \text{and} \quad \boldsymbol{S} \approx \boldsymbol{U}_{:,[r]} \boldsymbol{W}_{[r],:}$$

Principles:

- Explicit low-pass filter: Keep low-frequency eigenvectors, capturing smooth variations across the graph

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

- Graph Laplacian approximated as:

$$\boldsymbol{L}_G \approx \boldsymbol{U}_{:,[r]} \boldsymbol{\Lambda}_{[r],[r]} \boldsymbol{U}_{:,[r]}^\top \quad \text{and} \quad \boldsymbol{S} \approx \boldsymbol{U}_{:,[r]} \boldsymbol{W}_{[r],:}$$

Principles:

- Explicit low-pass filter: Keep low-frequency eigenvectors, capturing smooth variations across the graph
- Dimensionality reduction: Learn $\boldsymbol{W}_{[r],:} \in \mathbb{R}^{r \times d_s}$ instead of $\boldsymbol{S} \in \mathbb{R}^{n \times d_s}$

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

- Graph Laplacian approximated as:

$$\boldsymbol{L}_G \approx \boldsymbol{U}_{:,[r]} \boldsymbol{\Lambda}_{[r],[r]} \boldsymbol{U}_{:,[r]}^\top \quad \text{and} \quad \boldsymbol{S} \approx \boldsymbol{U}_{:,[r]} \boldsymbol{W}_{[r],:}$$

Principles:

- Explicit low-pass filter: Keep low-frequency eigenvectors, capturing smooth variations across the graph
- Dimensionality reduction: Learn $\boldsymbol{W}_{[r],:} \in \mathbb{R}^{r \times d_s}$ instead of $\boldsymbol{S} \in \mathbb{R}^{n \times d_s}$
- Each client requires only $\boldsymbol{U}_{v,:}$ from $\boldsymbol{U}_{:,[r]}$

FedLap+

Idea: Retain the r most informative spectral components $\longrightarrow$ first $r \ll n$ columns of $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ (first r rows of $\boldsymbol{W}$)

$$\boldsymbol{U}_{:,[r]} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r], \quad \boldsymbol{\Lambda}_{[r],[r]} = \operatorname{diag}(\lambda_1, \ldots, \lambda_r), \quad \boldsymbol{W}_{[r],:}$$

- Graph Laplacian approximated as:

$$\boldsymbol{L}_G \approx \boldsymbol{U}_{:,[r]} \boldsymbol{\Lambda}_{[r],[r]} \boldsymbol{U}_{:,[r]}^\top \quad \text{and} \quad \boldsymbol{S} \approx \boldsymbol{U}_{:,[r]} \boldsymbol{W}_{[r],:}$$

Principles:

- Explicit low-pass filter: Keep low-frequency eigenvectors, capturing smooth variations across the graph
- Dimensionality reduction: Learn $\boldsymbol{W}_{[r],:} \in \mathbb{R}^{r \times d_s}$ instead of $\boldsymbol{S} \in \mathbb{R}^{n \times d_s}$
- Each client requires only $\boldsymbol{U}_{v,:}$ from $\boldsymbol{U}_{:,[r]}$

Lower communication, faster training.

FedLap+: Decentralized Arnoldi iteration

- Need to compute the top-r eigenvectors and eigenvalues (full eigendecomposition $\mathcal{O}(n^3)$)

FedLap+: Decentralized Arnoldi iteration

- Need to compute the top-r eigenvectors and eigenvalues (full eigendecomposition $\mathcal{O}(n^3)$)

Idea: A decentralized version of the Arnoldi iteration (an efficient method to approximate the top-r eigenpairs).

FedLap+: Decentralized Arnoldi iteration

- Need to compute the top-r eigenvectors and eigenvalues (full eigendecomposition $\mathcal{O}(n^3)$)

Idea: A decentralized version of the Arnoldi iteration (an efficient method to approximate the top-r eigenpairs).

Arnoldi iteration:

- It computes an orthonormal basis $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m\}$ for the Krylov subspace $\mathcal{K}_m(\boldsymbol{M}, \boldsymbol{x}) = \text{span}\{\boldsymbol{x}, \boldsymbol{M}\boldsymbol{x}, \ldots, \boldsymbol{M}^{m-1}\boldsymbol{x}\}$ iteratively

FedLap+: Decentralized Arnoldi iteration

- Need to compute the top-r eigenvectors and eigenvalues (full eigendecomposition $\mathcal{O}(n^3)$)

Idea: A decentralized version of the Arnoldi iteration (an efficient method to approximate the top-r eigenpairs).

Arnoldi iteration:

- It computes an orthonormal basis $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_m\}$ for the Krylov subspace $\mathcal{K}_m(\boldsymbol{M}, \boldsymbol{x}) = \text{span}\{\boldsymbol{x}, \boldsymbol{M}\boldsymbol{x}, \dots, \boldsymbol{M}^{m-1}\boldsymbol{x}\}$ iteratively
- It approximates eigendecomposition of $\boldsymbol{M}$ as

$$\boldsymbol{M} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U},^{\mathsf{T}}$$

where $\boldsymbol{U} = \boldsymbol{Q}_m\boldsymbol{V}$ and $\boldsymbol{U}\boldsymbol{U}^{\mathsf{T}} \approx \boldsymbol{I}$, with $\boldsymbol{Q}_m = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_m]$, and $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$ the matrix of eigenvectors and eigenvalues of an upper Hessenberg matrix

FedLap+: Decentralized Arnoldi iteration

- Need to compute the top-r eigenvectors and eigenvalues (full eigendecomposition $\mathcal{O}(n^3)$)

Idea: A decentralized version of the Arnoldi iteration (an efficient method to approximate the top-r eigenpairs).

Arnoldi iteration:

- It computes an orthonormal basis $\{\boldsymbol{q}_1, \dots, \boldsymbol{q}_m\}$ for the Krylov subspace $\mathcal{K}_m(\boldsymbol{M}, \boldsymbol{x}) = \text{span}\{\boldsymbol{x}, \boldsymbol{M}\boldsymbol{x}, \dots, \boldsymbol{M}^{m-1}\boldsymbol{x}\}$ iteratively
- It approximates eigendecomposition of $\boldsymbol{M}$ as

$$\boldsymbol{M} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U},^{\mathsf{T}}$$

where $\boldsymbol{U} = \boldsymbol{Q}_m\boldsymbol{V}$ and $\boldsymbol{U}\boldsymbol{U}^{\mathsf{T}} \approx \boldsymbol{I}$, with $\boldsymbol{Q}_m = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_m]$, and $\boldsymbol{V}$ and $\boldsymbol{\Sigma}$ the matrix of eigenvectors and eigenvalues of an upper Hessenberg matrix

FedLap+: Computationally efficient ($\mathcal{O}(nr^2)$), scalable (for large, sparse graphs), privacy-preserving.

FedLap+: Privacy analysis

FedLap+: Privacy analysis

FedLap+ comprises two phases:

FedLap+: Privacy analysis

FedLap+ comprises two phases:

- Offline Phase:
  - Once, prior to training
  - Clients only share information about graph interconnections to compute $\boldsymbol{L}_{\mathcal{G}}$'s top eigenvectors

FedLap+: Privacy analysis

FedLap+ comprises two phases:

- Offline Phase:
  - Once, prior to training
  - Clients only share information about graph interconnections to compute $\boldsymbol{L}_{\mathcal{G}}$'s top eigenvectors
- Online Phase:
  - Same as standard FL
  - Conventional privacy-enhancing mechanisms apply

FedLap+: Privacy analysis

FedLap+ comprises two phases:

- Offline Phase:
  - Once, prior to training
  - Clients only share information about graph interconnections to compute $\boldsymbol{L}_{\mathcal{G}}$'s top eigenvectors
- Online Phase:
  - Same as standard FL
  - Conventional privacy-enhancing mechanisms apply

Offline phase: structural privacy (existence of edges between nodes)

FedLap+: Privacy analysis

FedLap+ comprises two phases:

- Offline Phase:
  - Once, prior to training
  - Clients only share information about graph interconnections to compute $\boldsymbol{L}_{\mathcal{G}}$'s top eigenvectors
- Online Phase:
  - Same as standard FL
  - Conventional privacy-enhancing mechanisms apply

Offline phase: structural privacy (existence of edges between nodes)

Approach: A membership inference formulation.

FedLap+: Privacy analysis

Threat Model: A strong attacker that has access to $\boldsymbol{Q}$ and $\boldsymbol{U}$ and attempts to infer whether a specific edge (u, v) exists

FedLap+: Privacy analysis

Threat Model: A strong attacker that has access to $\boldsymbol{Q}$ and $\boldsymbol{U}$ and attempts to infer whether a specific edge (u, v) exists

Membership inference formulation:

- The attacker performs a binary hypothesis test:

$$H_0 : \boldsymbol{A}_{uv} = 0 \quad \text{vs.} \quad H_1 : \boldsymbol{A}_{uv} = 1$$

- Applies the likelihood ratio test:

$$\text{LLR}_{u,v} = \log \frac{P(\boldsymbol{U}|\boldsymbol{A}_{uv} = 1)}{P(\boldsymbol{U}|\boldsymbol{A}_{uv} = 0)}$$

(optimal by Neyman–Pearson lemma)

FedLap+: Privacy analysis

Threat Model: A strong attacker that has access to $\boldsymbol{Q}$ and $\boldsymbol{U}$ and attempts to infer whether a specific edge (u, v) exists

Membership inference formulation:

- The attacker performs a binary hypothesis test:

$$H_0 : \boldsymbol{A}_{uv} = 0 \quad \text{vs.} \quad H_1 : \boldsymbol{A}_{uv} = 1$$

- Applies the likelihood ratio test:

$$\text{LLR}_{u,v} = \log \frac{P(\boldsymbol{U}|\boldsymbol{A}_{uv} = 1)}{P(\boldsymbol{U}|\boldsymbol{A}_{uv} = 0)}$$

(optimal by Neyman–Pearson lemma)

Theorem:

$$H_1 : \quad \text{LLR}_{u,v} \sim \mathcal{N}\left(\frac{1}{2}\alpha, \alpha\right), \quad H_0 : \quad \text{LLR}_{u,v} \sim \mathcal{N}\left(-\frac{1}{2}\alpha, \alpha\right)$$

$$D_{\text{KL}}\left(\Pr(\text{LLR}_{u,v} \mid H_1) \,\middle\|\, \Pr(\text{LLR}_{u,v} \mid H_0)\right) \approx \alpha/2, \qquad \alpha = \frac{r}{np(1-p)}$$

FedLap+: Privacy analysis

- For practical values of r, n, and p,

$$\text{precision}(\gamma) + \text{recall}(\gamma) \leq 1 \qquad \forall \gamma$$

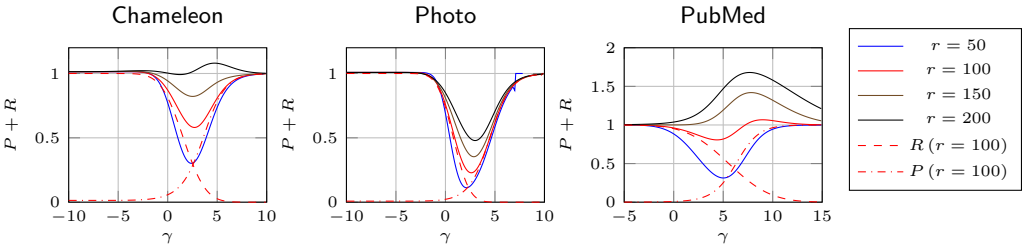
where precision$= \text{TP}/(\text{TP} + \text{FP})$, recall$= \text{TP}/(\text{TP} + \text{FN})$, γ is the decision threshold

FedLap+: Privacy analysis

- For practical values of r, n, and p,

$$\text{precision}(\gamma) + \text{recall}(\gamma) \leq 1 \qquad \forall \gamma$$

where precision$= \text{TP}/(\text{TP} + \text{FP})$, recall$= \text{TP}/(\text{TP} + \text{FN})$, γ is the decision threshold

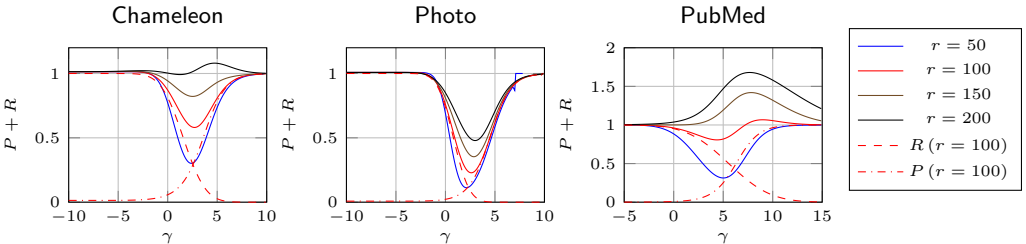


FedLap+: Privacy analysis

- For practical values of r, n, and p,

$$\text{precision}(\gamma) + \text{recall}(\gamma) \leq 1 \qquad \forall \gamma$$

where precision$= \text{TP}/(\text{TP} + \text{FP})$, recall$= \text{TP}/(\text{TP} + \text{FN})$, γ is the decision threshold



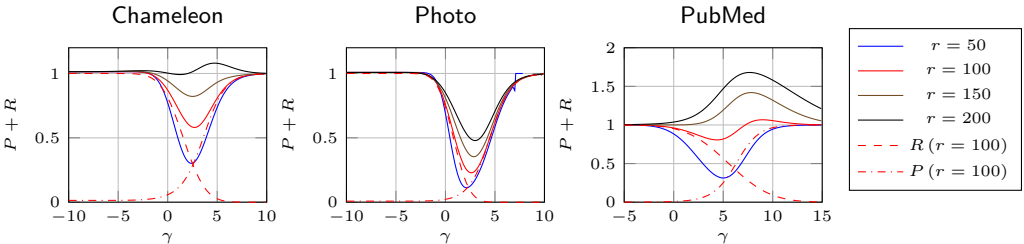
- $\text{P} + \text{R} \leq 1$: No better than trivial guessing

FedLap+: Privacy analysis

- For practical values of r, n, and p,

$$\text{precision}(\gamma) + \text{recall}(\gamma) \leq 1 \qquad \forall \gamma$$

where precision$= \text{TP}/(\text{TP} + \text{FP})$, recall$= \text{TP}/(\text{TP} + \text{FN})$, γ is the decision threshold



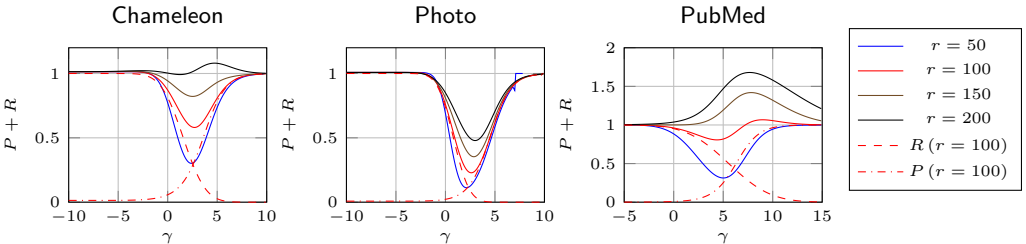
- $\text{P} + \text{R} \leq 1$: No better than trivial guessing
- KL divergence between H_0 and H_1 extremely small $\longrightarrow$ nearly indistinguishable edges

FedLap+: Privacy analysis

- For practical values of r, n, and p,

$$\text{precision}(\gamma) + \text{recall}(\gamma) \leq 1 \qquad \forall \gamma$$

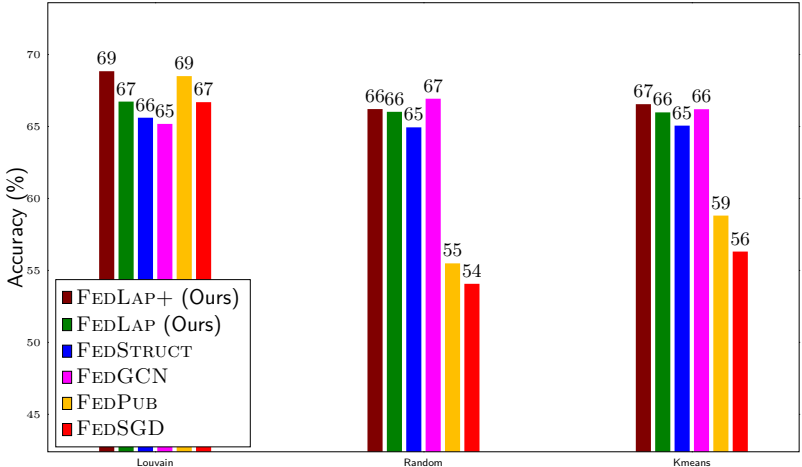
where precision$= \text{TP}/(\text{TP} + \text{FP})$, recall$= \text{TP}/(\text{TP} + \text{FN})$, γ is the decision threshold



- $\text{P} + \text{R} \leq 1$: No better than trivial guessing
- KL divergence between H_0 and H_1 extremely small $\longrightarrow$ nearly indistinguishable edges

An unrealistically strong adversary cannot reliably recover structural information.

Experimental results



- Dataset: ovbn-arxiv (169.343 nodes, 1.166.243 edges). 10% training ratio

Conclusion

FedLap: The state-of-the-art method for subgraph federated learning.

- Performance close to centralized setting
- Reduced the communication compared to FedStruct
- Strong privacy guarantees (membership inference-based analysis)
- Scalable and robust across different graph topologies, including heterophilic graphs

Conclusion

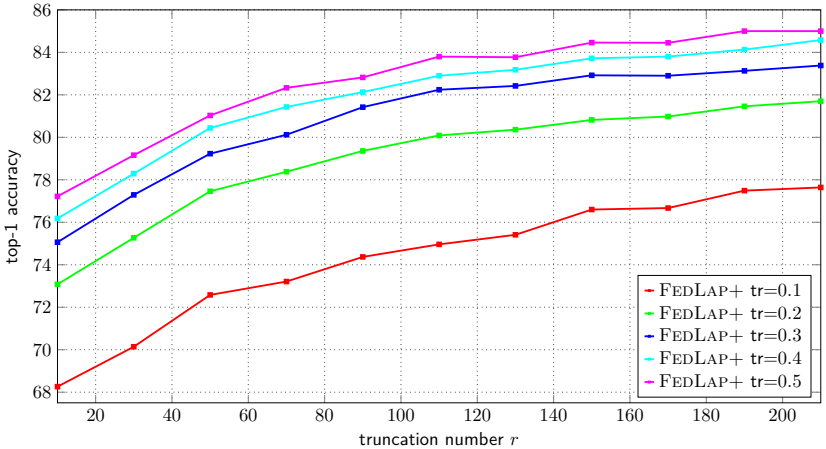
FedLap: The state-of-the-art method for subgraph federated learning.

- Performance close to centralized setting
- Reduced the communication compared to FedStruct
- Strong privacy guarantees (membership inference-based analysis)
- Scalable and robust across different graph topologies, including heterophilic graphs

What's next?

- Extension to dynamic graphs
- Gang detection

Experimental results: Truncation parameter r



- Dataset: Cora (2708 nodes, 10556 edges)

FedLap framework