Graph Mining
Graph Neural Networks

Clement Wang

2025 – 2026



INSTITUT
POLYTECHNIQUE
DE PARIS

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN

Spectral view of GNNs

Limitations and extensions

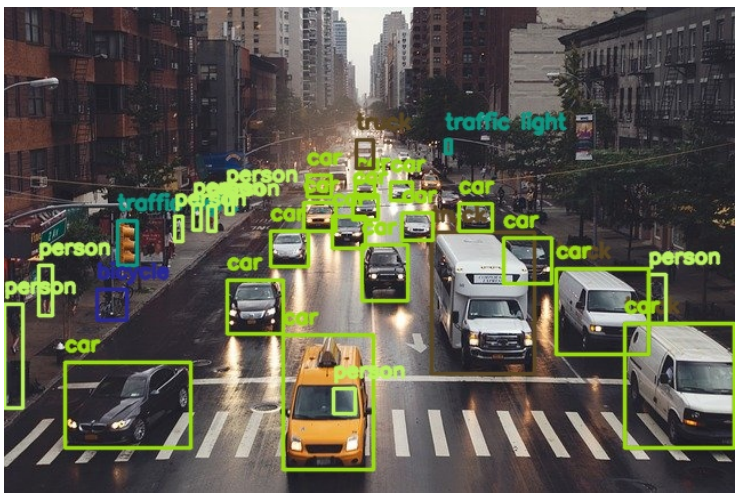
Some applications

Conclusion

Deep learning on structured data

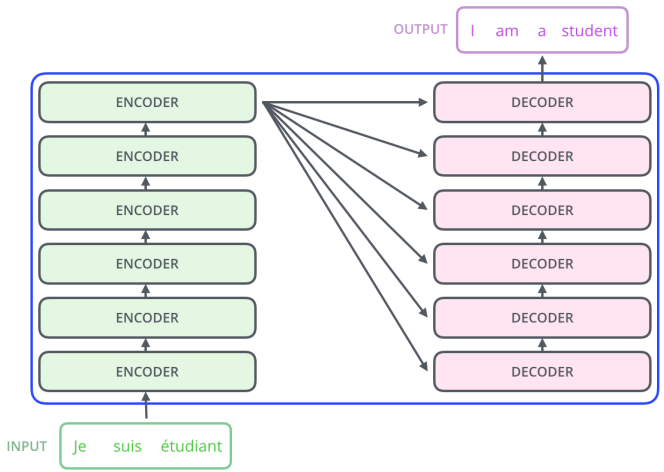
Deep learning has transformed many fields by learning useful representations directly from raw data.

Computer vision



Pixels $\longrightarrow$ objects, faces, scenes

Natural language processing

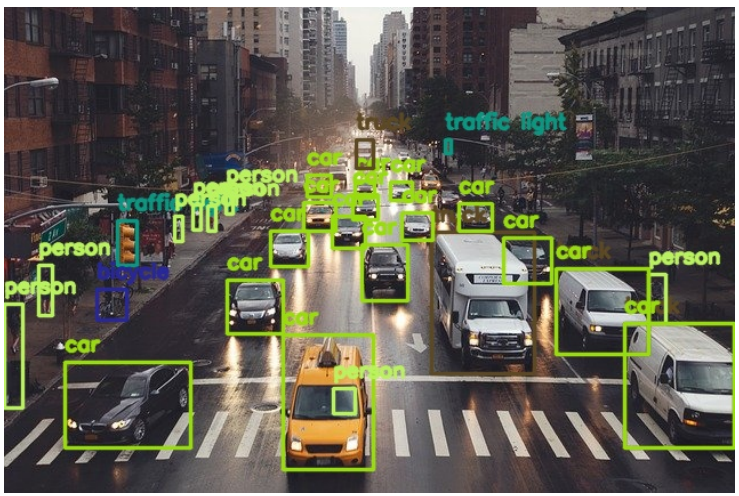


Text $\longrightarrow$ translation, summarization, assistants

Deep learning on structured data

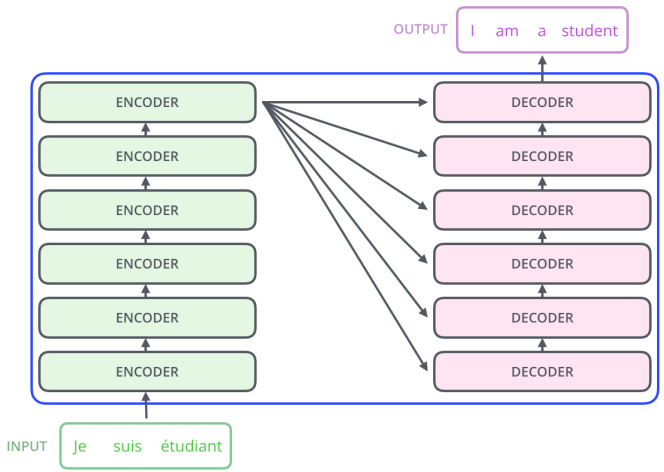
Deep learning has transformed many fields by learning useful representations directly from raw data.

Computer vision



Pixels $\longrightarrow$ objects, faces, scenes

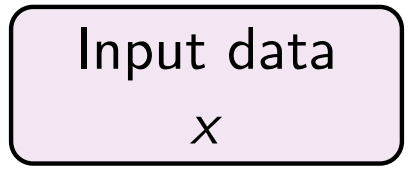
Natural language processing



Text $\longrightarrow$ translation, summarization, assistants

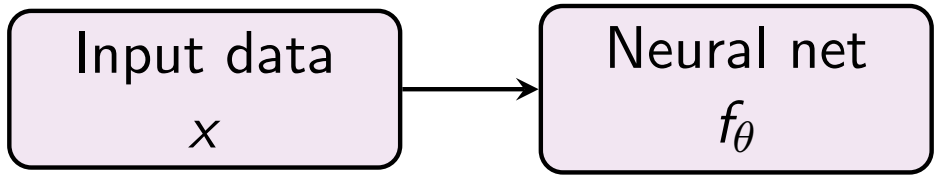
Can we learn directly from graphs too?

The deep learning recipe



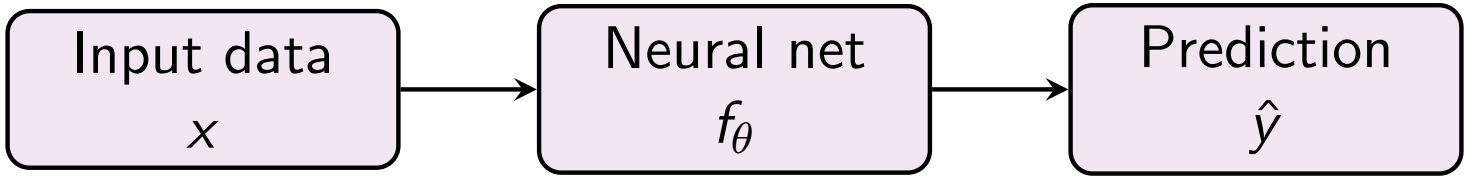
1. Represent the data as an input x

The deep learning recipe



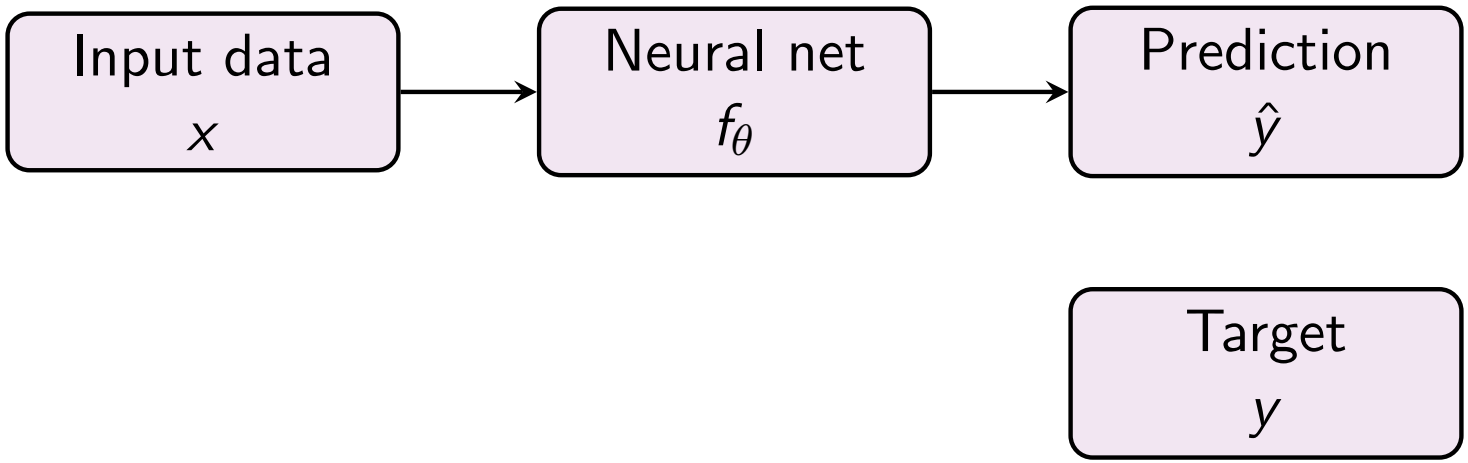
1. Represent the data as an input x
2. Choose a learnable function f_θ

The deep learning recipe



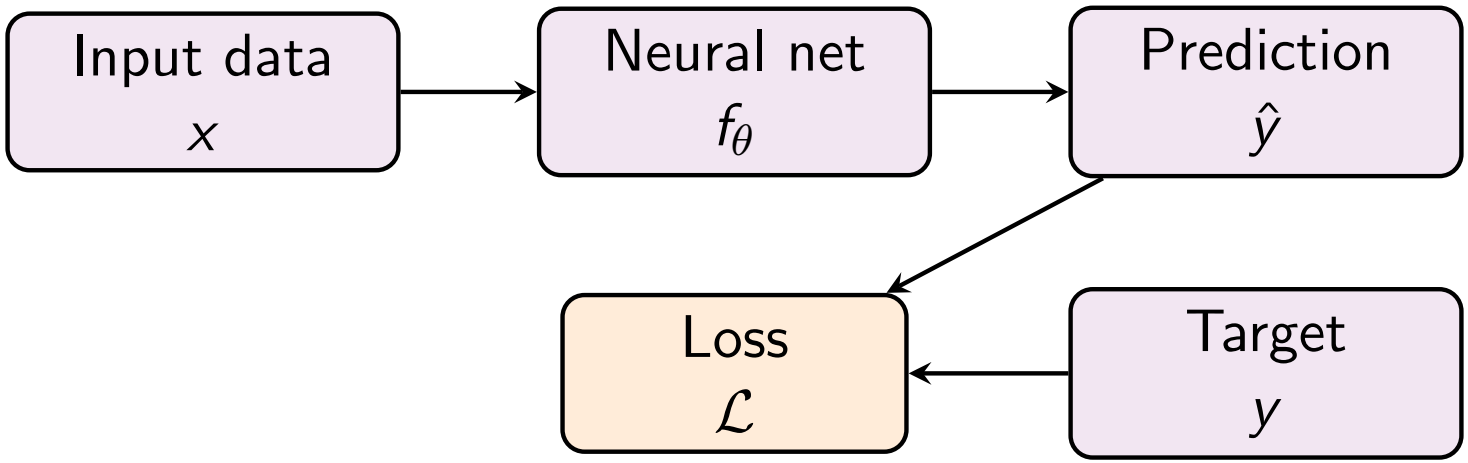
1. Represent the data as an input x
2. Choose a learnable function f_θ
3. Compute a prediction $\hat{y}$

The deep learning recipe



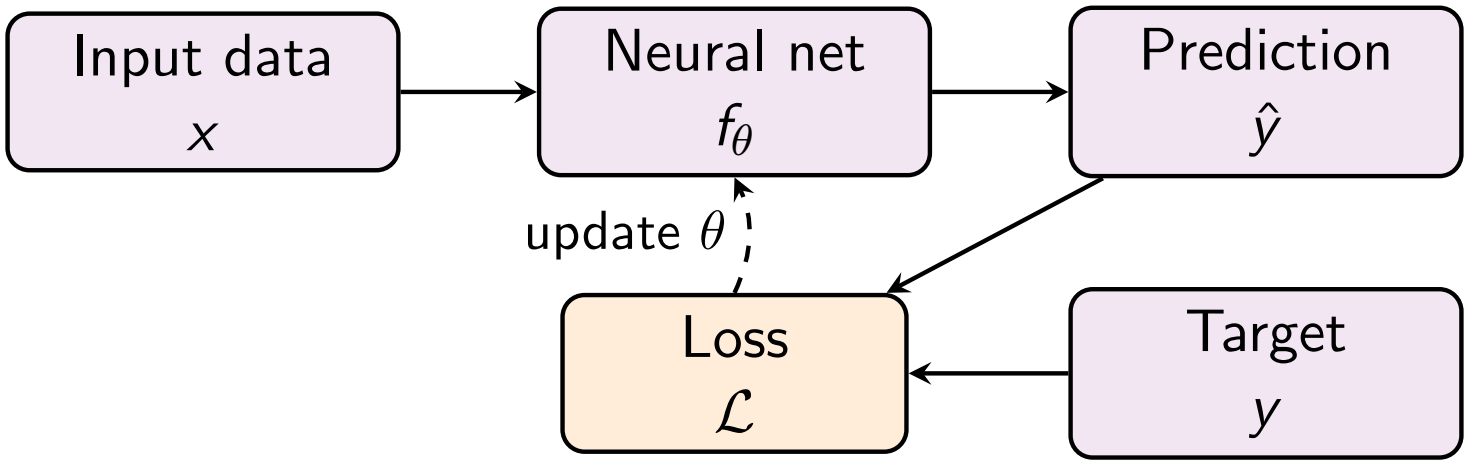
1. Represent the data as an input x
2. Choose a learnable function f_θ
3. Compute a prediction $\hat{y}$
4. Use a target label y

The deep learning recipe



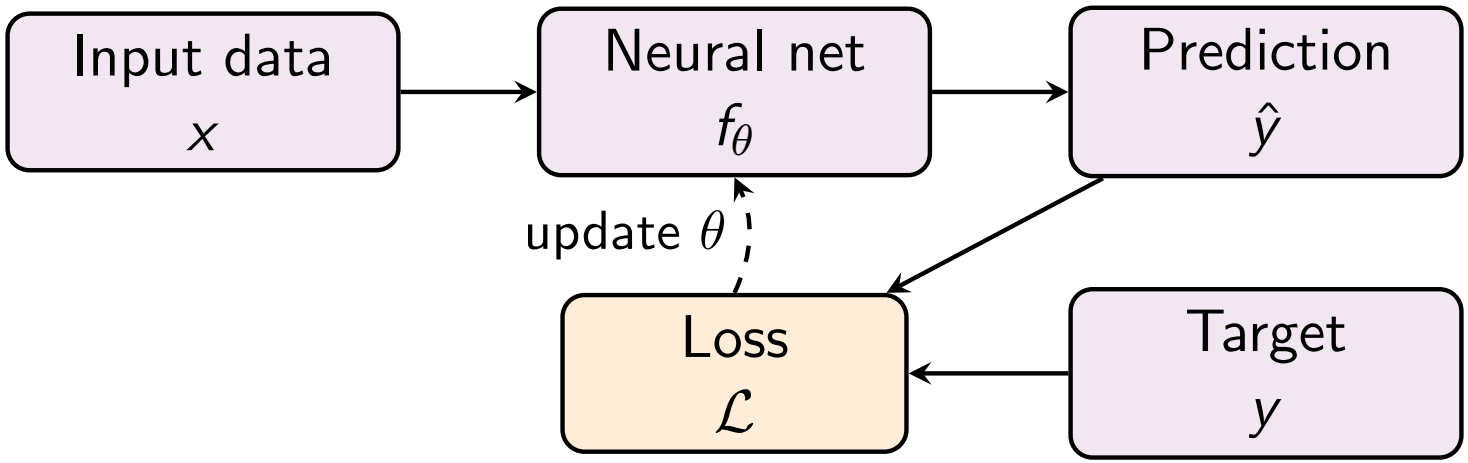
1. Represent the data as an input x
2. Choose a learnable function f_θ
3. Compute a prediction $\hat{y}$
4. Use a target label y
5. Measure the error with a loss $\mathcal{L}(\hat{y}, y)$

The deep learning recipe



1. Represent the data as an input x
2. Choose a learnable function f_θ
3. Compute a prediction $\hat{y}$
4. Use a target label y
5. Measure the error with a loss $\mathcal{L}(\hat{y}, y)$
6. Learn θ by gradient descent

The deep learning recipe



1. Represent the data as an input x
2. Choose a learnable function f_θ
3. Compute a prediction $\hat{y}$
4. Use a target label y
5. Measure the error with a loss $\mathcal{L}(\hat{y}, y)$
6. Learn θ by gradient descent

What is the right f_θ for graph data?

Why graph neural networks?

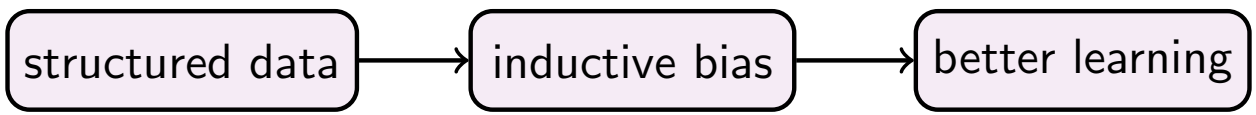
Deep learning is very effective when the architecture matches the structure of the data.

Data	**Structure**	**Typical architecture**
Images	regular grid	CNN
Text	sequence of tokens	Transformer
Graphs	nodes and edges	GNN

Why graph neural networks?

Deep learning is very effective when the architecture matches the structure of the data.

Data	Structure	Typical architecture
Images	regular grid	CNN
Text	sequence of tokens	Transformer
Graphs	nodes and edges	GNN



For graphs, the structure is the edge set.

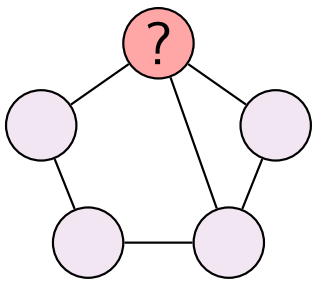
Prediction tasks on graphs

There are three standard prediction settings.

Prediction tasks on graphs

There are three standard prediction settings.

Node-level

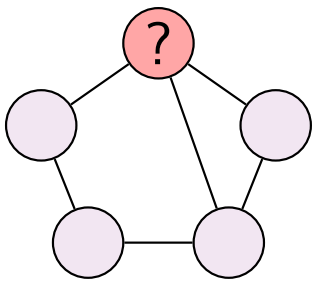


Predict a property of a node in a graph.

Prediction tasks on graphs

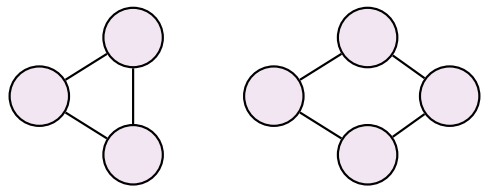
There are three standard prediction settings.

Node-level



Predict a property of a node in a graph.

Graph-level



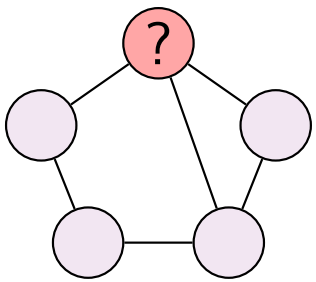
label?

Predict a property of an entire graph.

Prediction tasks on graphs

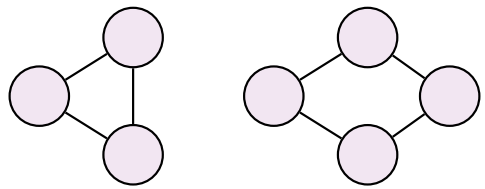
There are three standard prediction settings.

Node-level



Predict a property of a node in a graph.

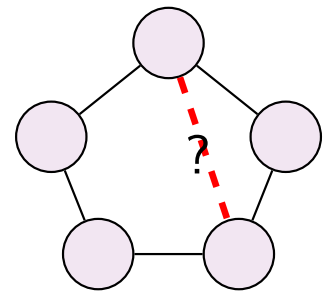
Graph-level



label?

Predict a property of an entire graph.

Edge-level

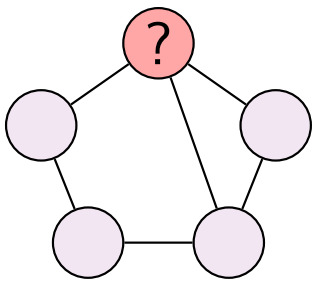


Predict a property or existence of an edge.

Prediction tasks on graphs

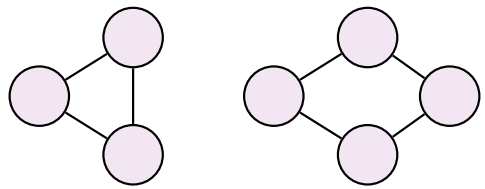
There are three standard prediction settings.

Node-level



Predict a property of a node in a graph.

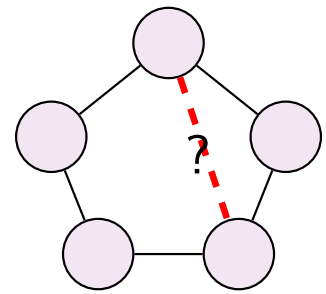
Graph-level



label?

Predict a property of an entire graph.

Edge-level



Predict a property or existence of an edge.

We will first focus on **node-level prediction**.

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN
- Notations
- Message passing layers
- Training setting

Spectral view of GNNs

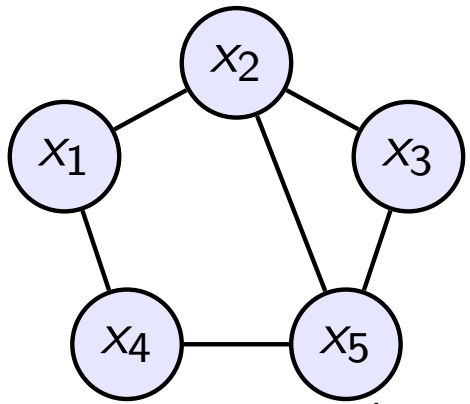
Limitations and extensions

Some applications

Conclusion

Node-level prediction and notations

Node-level prediction and notations

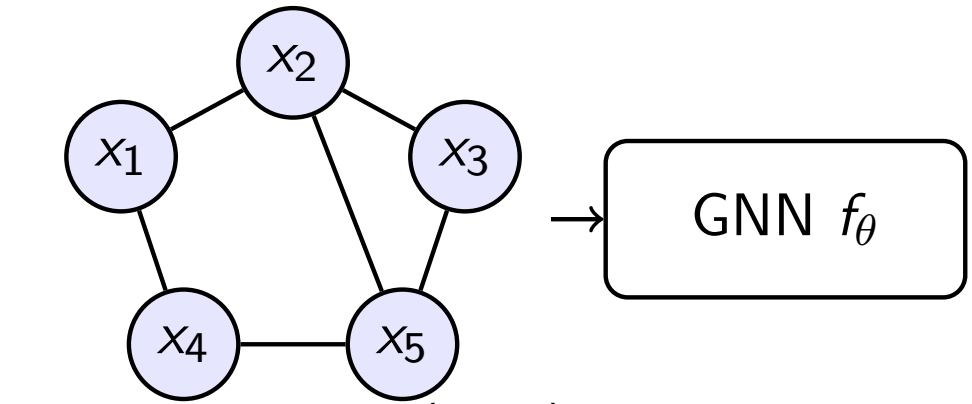


input graph $G = (V, E)$
node features $X \in \mathbb{R}^{n \times d_0}$

$$G = (V, E), \qquad n = |V|$$

$$x_v \in \mathbb{R}^{d_0}, \qquad X \in \mathbb{R}^{n \times d_0}$$

Node-level prediction and notations

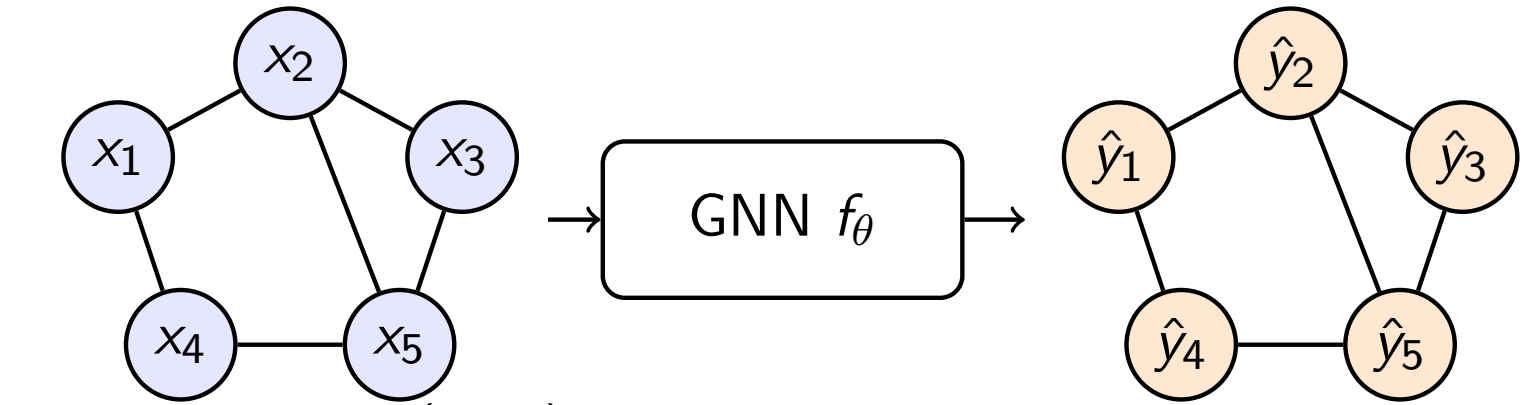


input graph $G = (V, E)$
node features $X \in \mathbb{R}^{n \times d_0}$

$$G = (V, E), \qquad n = |V|$$

$$x_v \in \mathbb{R}^{d_0}, \qquad X \in \mathbb{R}^{n \times d_0}$$

Node-level prediction and notations



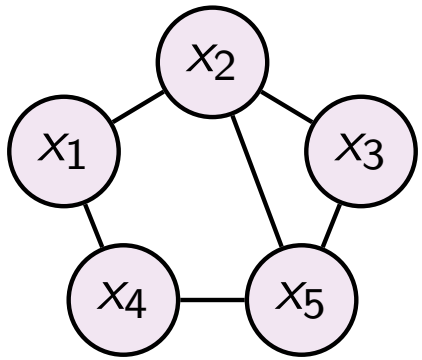
input graph $G = (V, E)$
node features $X \in \mathbb{R}^{n \times d_0}$

one prediction per node
$\hat{y}_v = f_\theta(G, X)_v$

$$G = (V, E), \qquad n = |V| \qquad \hat{y}_v = f_\theta(G, X)_v, \qquad v \in V$$

$$x_v \in \mathbb{R}^{d_0}, \qquad X \in \mathbb{R}^{n \times d_0} \qquad y_v \in \{1, \ldots, C\}, \qquad v \in V$$

Node representations

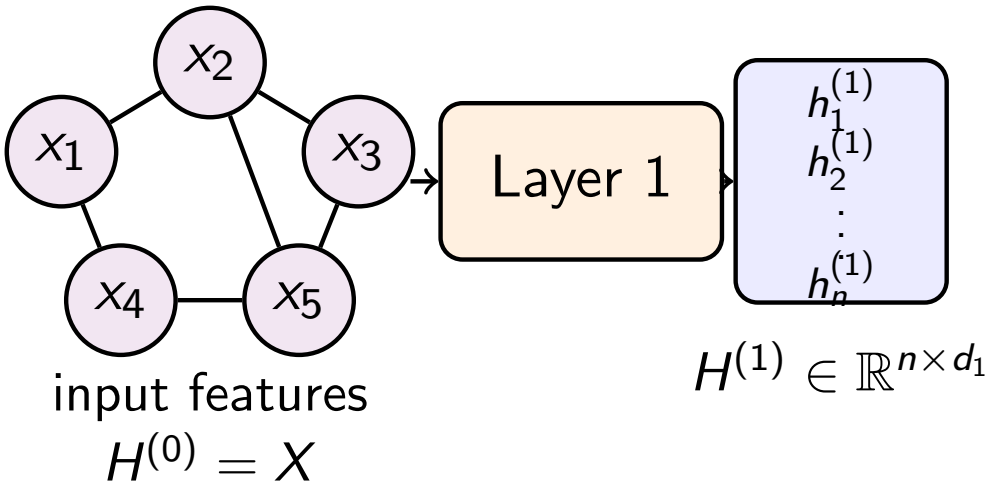


input features

$H^{(0)} = X$

A GNN starts from the input node features.

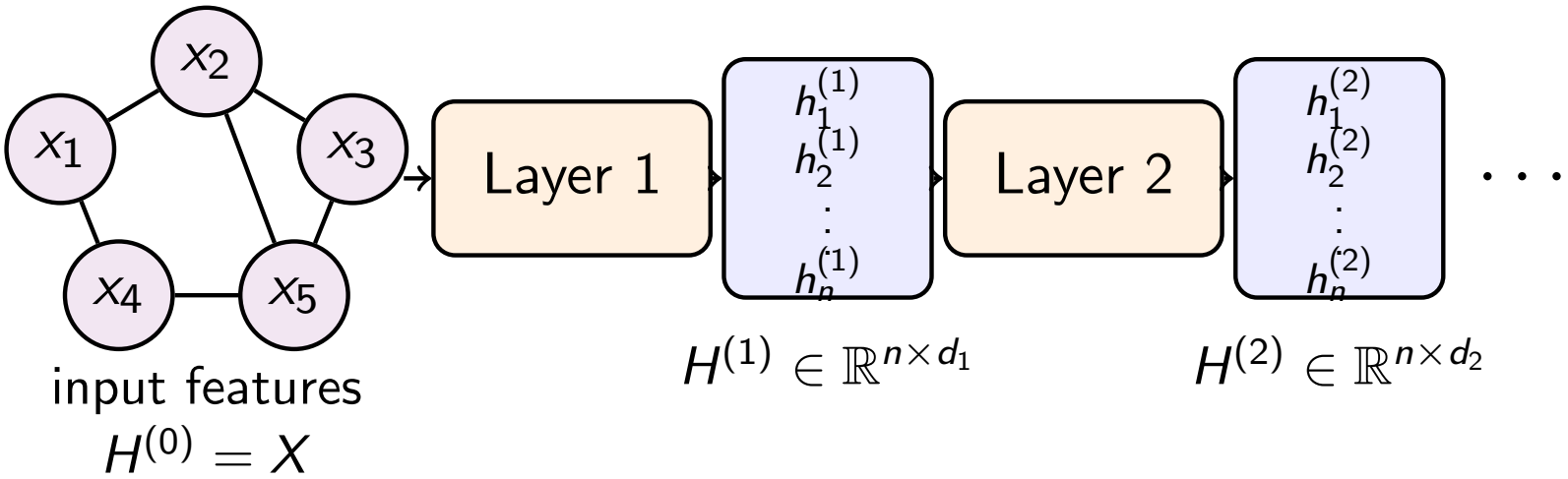
Node representations



Each layer outputs one representation per node.

$$h_v^{(1)} \in \mathbb{R}^{d_1}$$

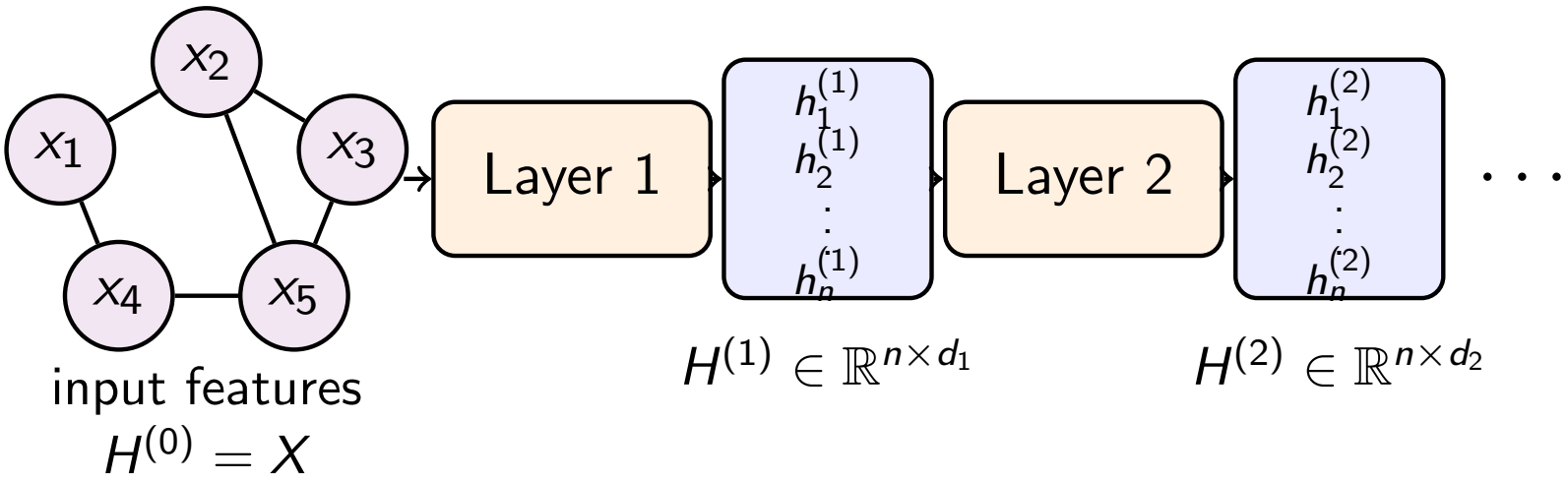
Node representations



Stacking layers gives a sequence of node representations.

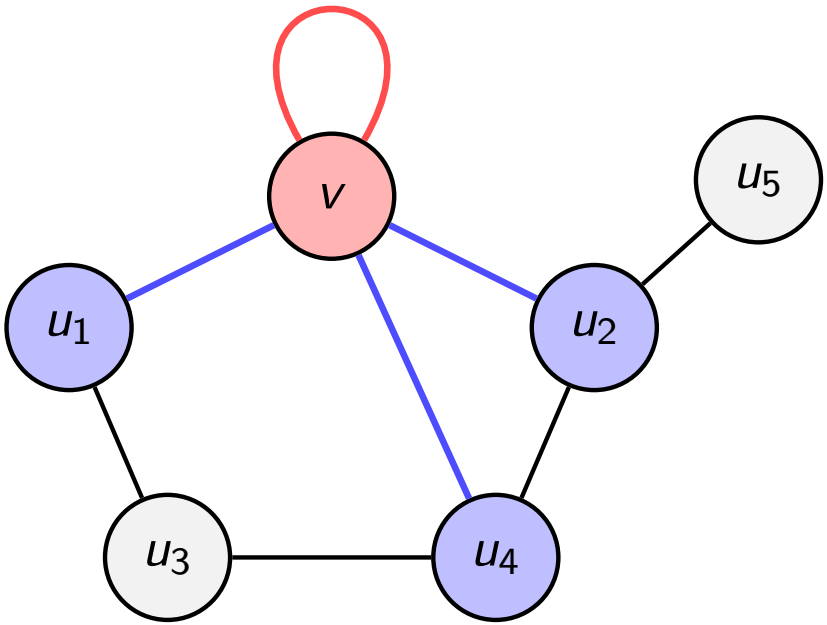
$$h_v^{(\ell)} \in \mathbb{R}^{d_\ell}$$

Node representations



$$H^{(\ell)} = \begin{bmatrix} -(h_1^{(\ell)})^\top - \\ \vdots \\ -(h_n^{(\ell)})^\top - \end{bmatrix} \in \mathbb{R}^{n \times d_\ell}, \qquad h_v^{(0)} = x_v, \qquad H^{(0)} = X.$$

Neighborhood notation



$$\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$$

$$\mathcal{N}(v) = \{u_1, u_2, u_4\}$$

$$\widetilde{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$$

A GNN layer updates $h_v^{(\ell)}$ from the representations of nodes in

$$\widetilde{\mathcal{N}}(v).$$

Multilayer perceptrons

An MLP takes a fixed-size vector $x \in \mathbb{R}^d$.

Layer

$$h^{(\ell+1)} = \sigma\left(W^{(\ell)} h^{(\ell)} + b^{(\ell)}\right).$$

Multilayer perceptrons

An MLP takes a fixed-size vector $x \in \mathbb{R}^d$.

Layer

$$h^{(\ell+1)} = \sigma\left(W^{(\ell)} h^{(\ell)} + b^{(\ell)}\right).$$

Problem for graphs:

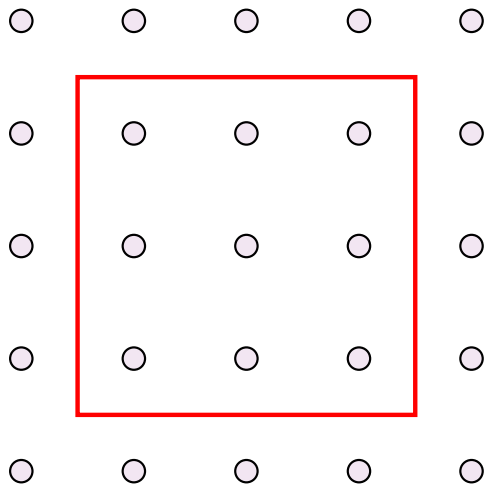
An MLP does not naturally exploit the edge structure.

From grids to graphs

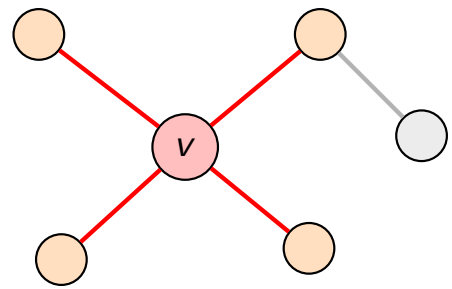
CNNs work well on images because local neighborhoods are fixed and reused everywhere.

From grids to graphs

CNNs work well on images because local neighborhoods are fixed and reused everywhere.



fixed 3×3 neighborhood



graph-defined neighborhood

GNNs generalize the idea of local aggregation to irregular neighborhoods.

Message passing

A GNN starts from node features:

$$h_v^{(0)} = x_v.$$

A message passing layer has two steps.

Message passing

A GNN starts from node features:

$$h_v^{(0)} = x_v.$$

A message passing layer has two steps.

1. Aggregate

$$m_v^{(\ell)} = \mathrm{AGG}^{(\ell)}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right).$$

$\mathrm{AGG}^{(\ell)} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}$ in many GNNs.

Message passing

A GNN starts from node features:

$$h_v^{(0)} = x_v.$$

A message passing layer has two steps.

1. Aggregate

$$m_v^{(\ell)} = \mathrm{AGG}^{(\ell)}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right).$$

$\mathrm{AGG}^{(\ell)} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}$ in many GNNs.

2. Update

$$h_v^{(\ell+1)} = \mathrm{UPDATE}^{(\ell)}\left(h_v^{(\ell)}, m_v^{(\ell)}\right).$$

Message passing

A GNN starts from node features:

$$h_v^{(0)} = x_v.$$

A message passing layer has two steps.

1. Aggregate

$$m_v^{(\ell)} = \mathrm{AGG}^{(\ell)}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right).$$

$\mathrm{AGG}^{(\ell)} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}$ in many GNNs.

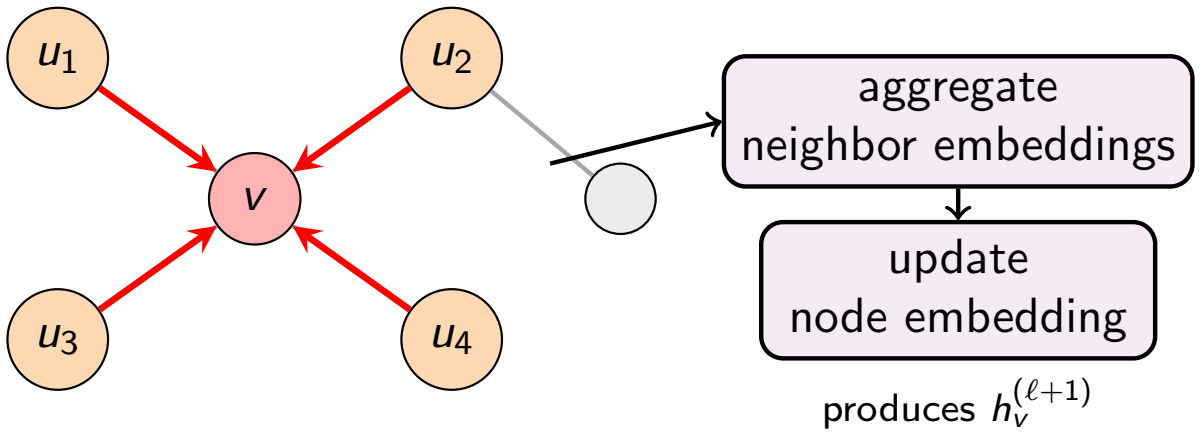
2. Update

$$h_v^{(\ell+1)} = \mathrm{UPDATE}^{(\ell)}\left(h_v^{(\ell)}, m_v^{(\ell)}\right).$$

neighbors send messages, nodes update states.

Message passing: visual view

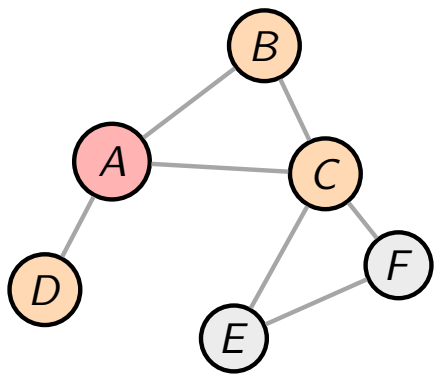
One layer updates each node using its 1-hop neighborhood.



The same aggregation and update functions are shared across all nodes.

Computation tree and receptive field

Stacking message-passing layers unfolds the graph into a computation tree.

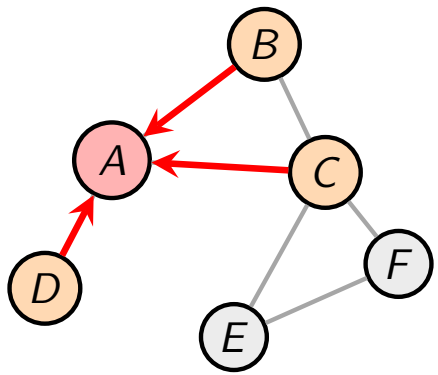


original graph, target A

We focus on a target node A.

Computation tree and receptive field

Stacking message-passing layers unfolds the graph into a computation tree.

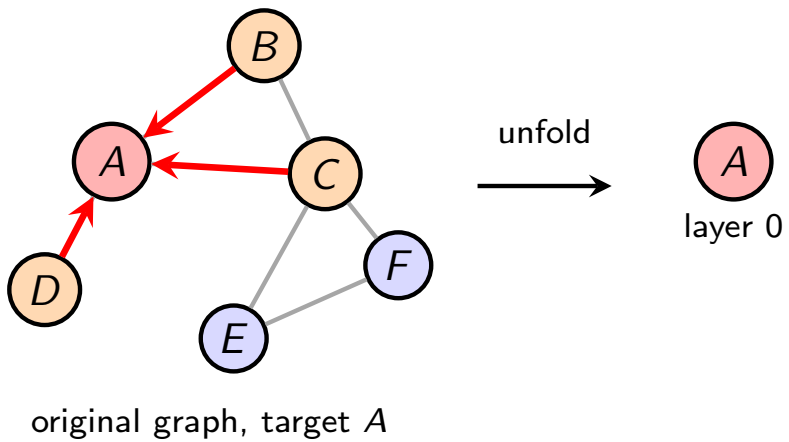


original graph, target A

After one message-passing layer, A receives information from its neighbors.

Computation tree and receptive field

Stacking message-passing layers unfolds the graph into a computation tree.

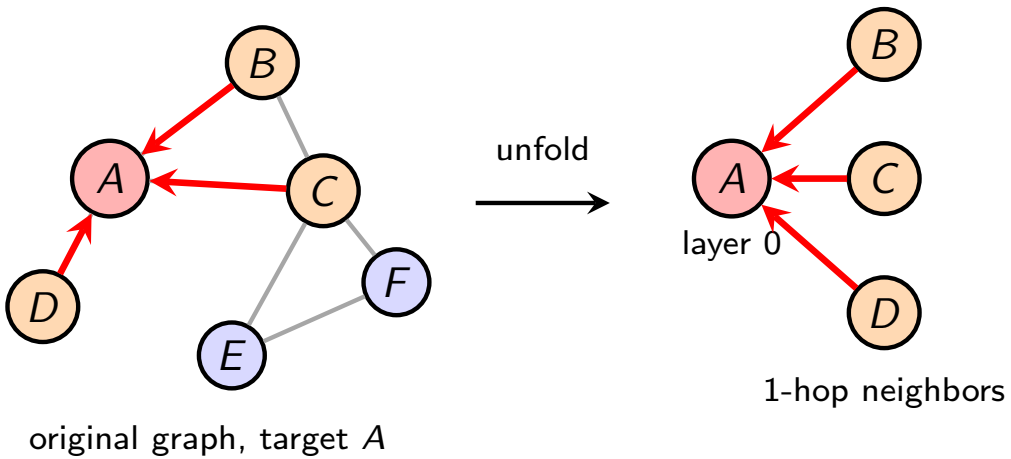


original graph, target A

The computation for A can be represented as a tree rooted at A.

Computation tree and receptive field

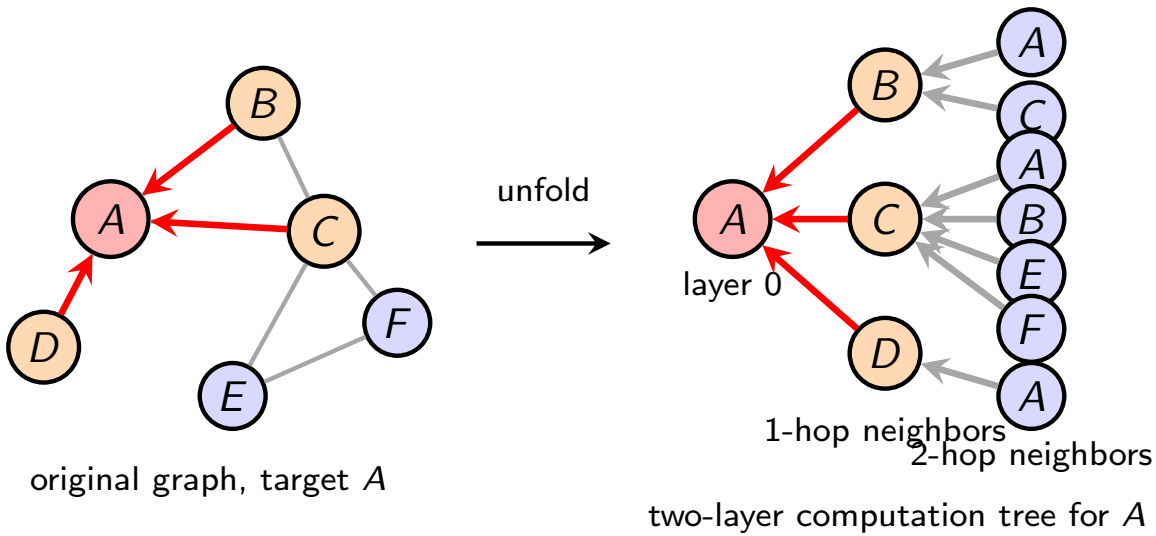
Stacking message-passing layers unfolds the graph into a computation tree.



The computation for A can be represented as a tree rooted at A.

Computation tree and receptive field

Stacking message-passing layers unfolds the graph into a computation tree.



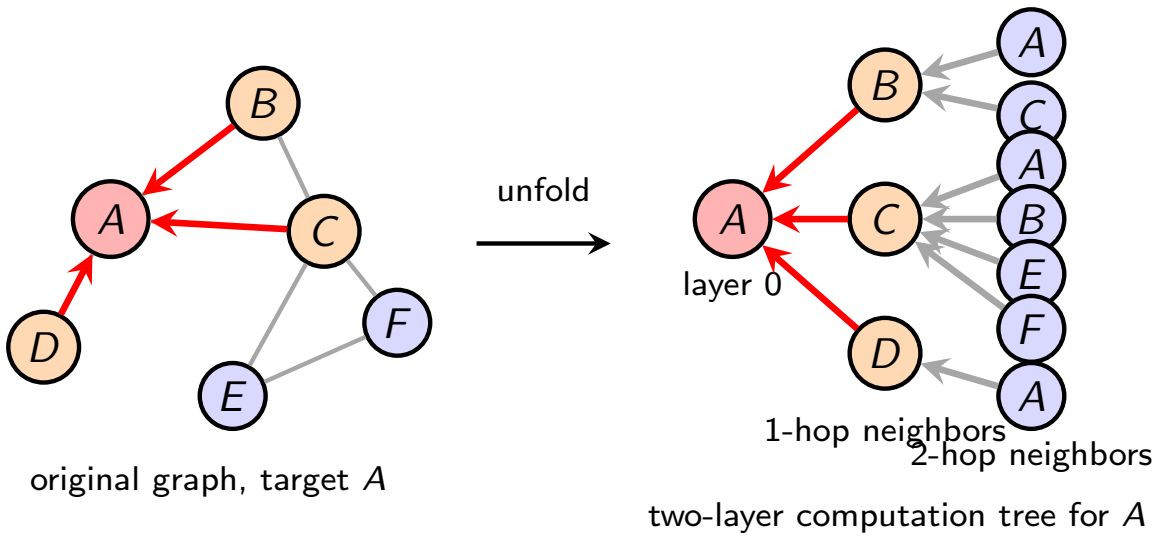
original graph, target A

two-layer computation tree for A

After two layers, information can arrive from nodes at distance two.

Computation tree and receptive field

Stacking message-passing layers unfolds the graph into a computation tree.



original graph, target A

two-layer computation tree for A

After k layers, node A can use information from its k-hop receptive field.

Graph convolutional networks

A Graph Convolutional Network (GCN) is a specific message passing layer.

Graph convolutional networks

A Graph Convolutional Network (GCN) is a specific message passing layer.

Notation

$$\widetilde{A} = A + I, \qquad \widetilde{D}_{vv} = \sum_{u=1}^{n} \widetilde{A}_{vu}, \qquad \widehat{A} = \widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2}.$$

Graph convolutional networks

A Graph Convolutional Network (GCN) is a specific message passing layer.

Notation

$$\widetilde{A} = A + I, \qquad \widetilde{D}_{vv} = \sum_{u=1}^{n} \widetilde{A}_{vu}, \qquad \widehat{A} = \widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2}.$$

Global matrix view

$$H^{(\ell+1)} = \sigma\left(\widehat{A} H^{(\ell)} W^{(\ell)}\right).$$

Graph convolutional networks

A Graph Convolutional Network (GCN) is a specific message passing layer.

Notation

$$\widetilde{A} = A + I, \qquad \widetilde{D}_{vv} = \sum_{u=1}^{n} \widetilde{A}_{vu}, \qquad \widehat{A} = \widetilde{D}^{-1/2} \widetilde{A} \widetilde{D}^{-1/2}.$$

Global matrix view

$$H^{(\ell+1)} = \sigma\left(\widehat{A} H^{(\ell)} W^{(\ell)}\right).$$

Equivalent local node view

$$h_v^{(\ell+1)} = \sigma\left(\sum_{u \in \widetilde{\mathcal{N}}(v)} \frac{1}{\sqrt{\widetilde{d}_v \widetilde{d}_u}} h_u^{(\ell)} W^{(\ell)}\right).$$

Bias terms are omitted for simplicity.

A two-layer GCN

Normalize the graph adjacency

$$\widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$$

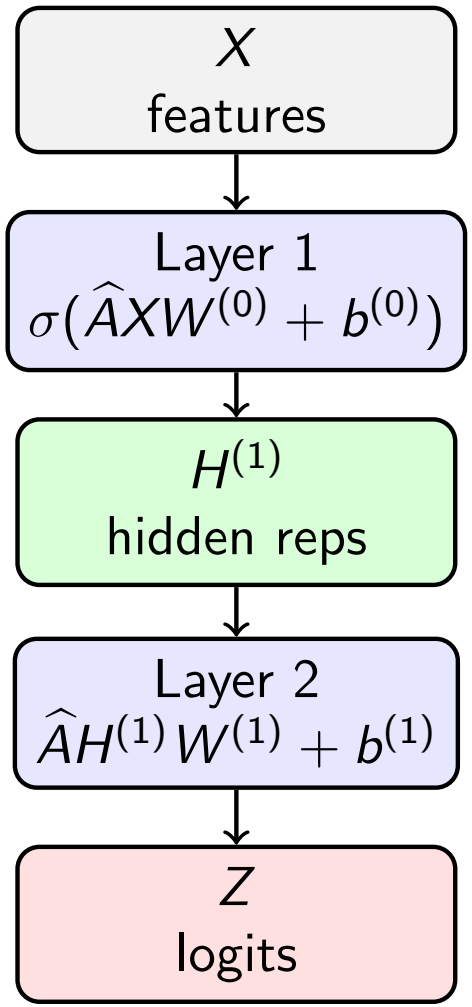
Layer 1

$$H^{(1)} = \sigma\left(\widehat{A}XW^{(0)} + b^{(0)}\right)$$

Layer 2

$$Z = \widehat{A}H^{(1)}W^{(1)} + b^{(1)}$$

Z contains the output scores, or logits, for all nodes.



GraphSAGE: aggregate and update

Another type of message passing layer.

GraphSAGE layer

$$m_v^{(\ell)} = \mathrm{AGG}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right), \qquad \mathrm{AGG} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}.$$

$$h_v^{(\ell+1)} = \sigma\left(\mathrm{CONCAT}(h_v^{(\ell)}, m_v^{(\ell)})W^{(\ell)}\right).$$

GraphSAGE: aggregate and update

Another type of message passing layer.

GraphSAGE layer

$$m_v^{(\ell)} = \mathrm{AGG}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right), \qquad \mathrm{AGG} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}.$$

$$h_v^{(\ell+1)} = \sigma\left(\mathrm{CONCAT}(h_v^{(\ell)}, m_v^{(\ell)})W^{(\ell)}\right).$$

For the mean aggregator,

$$m_v^{(\ell)} = \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} h_u^{(\ell)}.$$

GraphSAGE: aggregate and update

Another type of message passing layer.

GraphSAGE layer

$$m_v^{(\ell)} = \mathrm{AGG}\left(\{h_u^{(\ell)} : u \in \mathcal{N}(v)\}\right), \qquad \mathrm{AGG} \in \{\mathrm{mean}, \mathrm{max}, \mathrm{sum}\}.$$

$$h_v^{(\ell+1)} = \sigma\left(\mathrm{CONCAT}(h_v^{(\ell)}, m_v^{(\ell)})W^{(\ell)}\right).$$

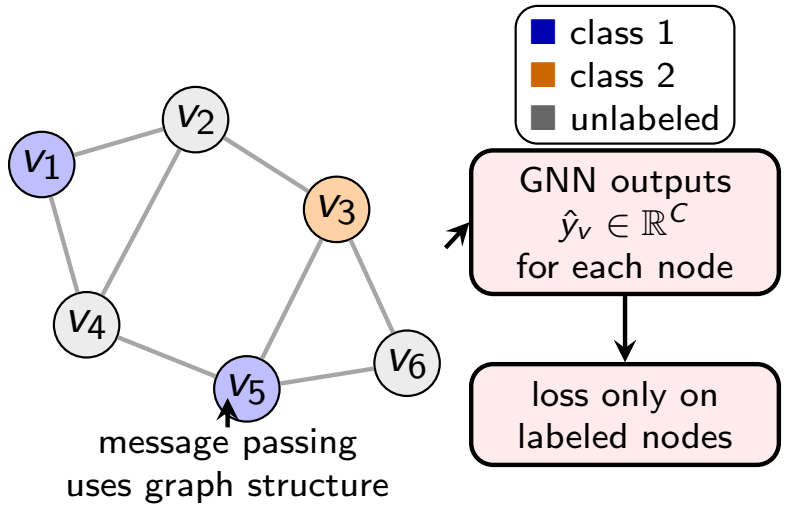
For the mean aggregator,

$$m_v^{(\ell)} = \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} h_u^{(\ell)}.$$

Splitting the weight matrix according to the concatenation gives:

$$h_v^{(\ell+1)} = \sigma\left(h_v^{(\ell)} W_{\mathrm{self}}^{(\ell)} + \frac{1}{d_v} \sum_{u \in \mathcal{N}(v)} h_u^{(\ell)} W_{\mathrm{neigh}}^{(\ell)}\right).$$

Training a GNN for node classification



The model predicts a class distribution for each node:

$$\hat{y}_v \in \mathbb{R}^C.$$

The loss is computed only on labeled training nodes:

$$\mathcal{L} = \sum_{v \in V_{\text{train}}} \ell(\hat{y}_v, y_v).$$

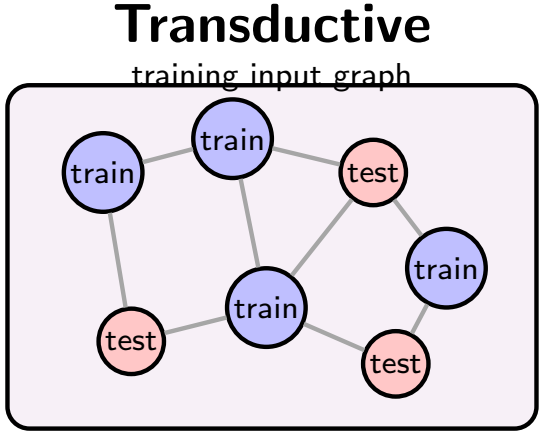
Unlabeled nodes can still be used during message passing.

Transductive vs. inductive learning

Same task, different training-time input graphs.

Transductive vs. inductive learning

Same task, different training-time input graphs.

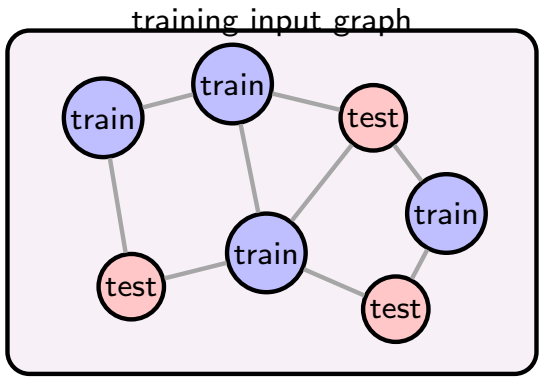


Full graph observed during training.
Test labels are hidden.

Transductive vs. inductive learning

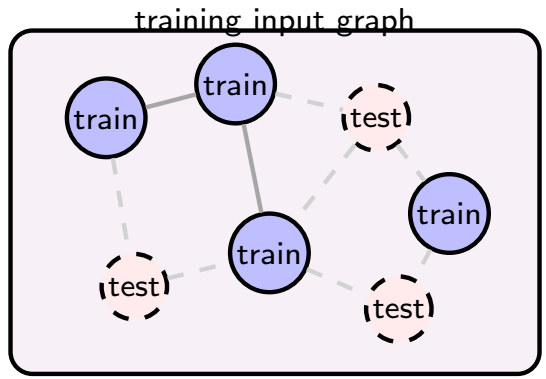
Same task, different training-time input graphs.

Transductive



Full graph observed during training.
Test labels are hidden.

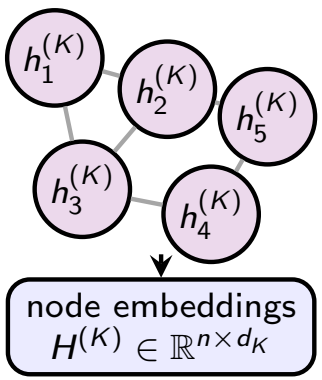
Inductive



Only training subgraph observed.
Test nodes appear at inference.

Key difference: are the test nodes part of the training-time input graph?

Graph-level prediction

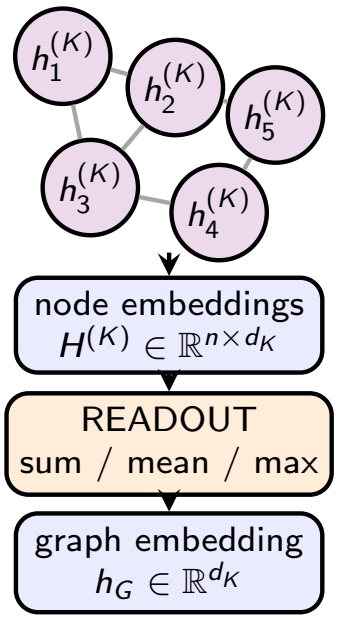


1. Node embeddings

After K message-passing layers:

$$h_v^{(K)} \in \mathbb{R}^{d_K}, \qquad H^{(K)} \in \mathbb{R}^{n \times d_K}.$$

Graph-level prediction



1. Node embeddings

After K message-passing layers:

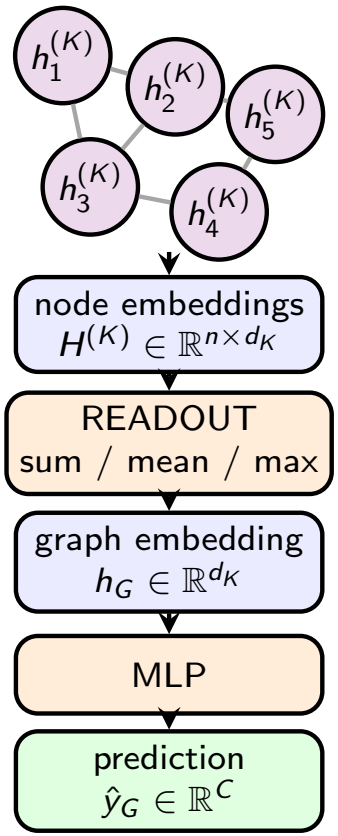
$$h_v^{(K)} \in \mathbb{R}^{d_K}, \qquad H^{(K)} \in \mathbb{R}^{n \times d_K}.$$

2. Readout

Aggregate all node embeddings into one graph embedding:

$$h_G = \mathrm{READOUT}\left(\{h_v^{(K)} : v \in V\}\right) \in \mathbb{R}^{d_K}.$$

Graph-level prediction



1. Node embeddings

After K message-passing layers:

$$h_v^{(K)} \in \mathbb{R}^{d_K}, \qquad H^{(K)} \in \mathbb{R}^{n \times d_K}.$$

2. Readout

Aggregate all node embeddings into one graph embedding:

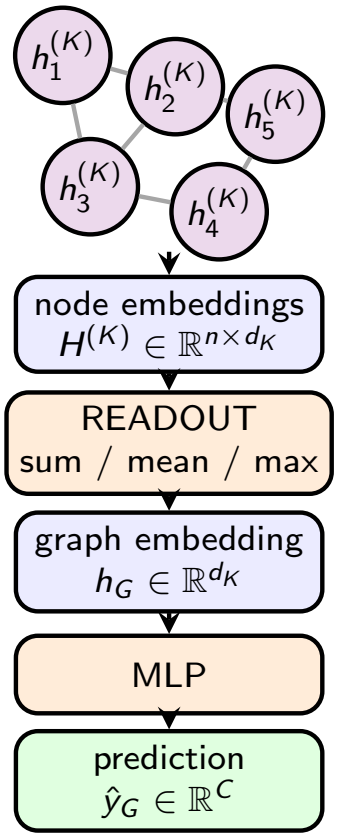
$$h_G = \mathrm{READOUT}\left(\{h_v^{(K)} : v \in V\}\right) \in \mathbb{R}^{d_K}.$$

3. Prediction

Predict a graph-level target:

$$\hat{y}_G = \mathrm{MLP}(h_G) \in \mathbb{R}^C.$$

Graph-level prediction



1. Node embeddings

After K message-passing layers:

$$h_v^{(K)} \in \mathbb{R}^{d_K}, \qquad H^{(K)} \in \mathbb{R}^{n \times d_K}.$$

2. Readout

Aggregate all node embeddings into one graph embedding:

$$h_G = \mathrm{READOUT}\left(\{h_v^{(K)} : v \in V\}\right) \in \mathbb{R}^{d_K}.$$

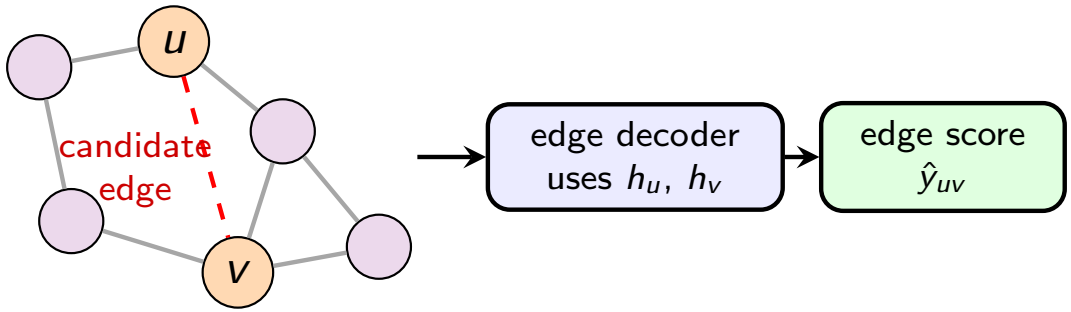
3. Prediction

Predict a graph-level target:

$$\hat{y}_G = \mathrm{MLP}(h_G) \in \mathbb{R}^C.$$

Examples: molecule property prediction, graph classification, transaction graph scoring.

Link prediction



Given two node embeddings, score whether an edge should exist:

$$\hat{y}_{uv} = \sigma(h_u^\top h_v).$$

More generally, use a neural decoder:

$$\hat{y}_{uv} = \mathrm{MLP}([h_u, h_v]).$$

Examples: recommendation, knowledge graph completion, interaction prediction.

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN

Spectral view of GNNs

Limitations and extensions

Some applications

Conclusion

Node features as graph signals

A scalar node feature can be viewed as a signal on the graph:

$$x : V \to \mathbb{R}, \qquad x \in \mathbb{R}^n.$$

For several features:

$$X \in \mathbb{R}^{n \times d_0}.$$

Node features as graph signals

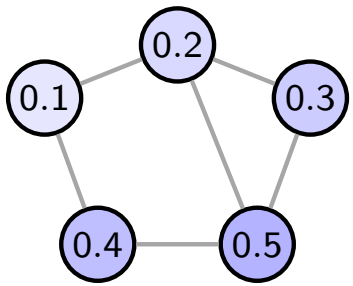
A scalar node feature can be viewed as a signal on the graph:

$$x : V \to \mathbb{R}, \qquad x \in \mathbb{R}^n.$$

For several features:

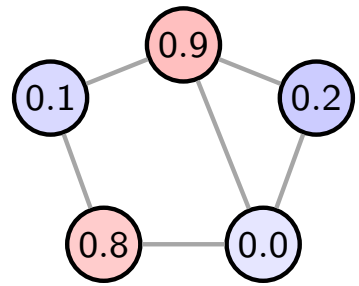
$$X \in \mathbb{R}^{n \times d_0}.$$

Smooth signal



Neighboring nodes have similar values.

Oscillatory signal



Neighboring nodes can have very different values.

Node features as graph signals

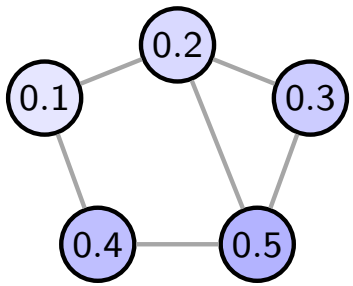
A scalar node feature can be viewed as a signal on the graph:

$$x : V \to \mathbb{R}, \qquad x \in \mathbb{R}^n.$$

For several features:

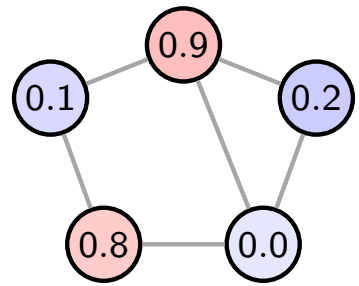
$$X \in \mathbb{R}^{n \times d_0}.$$

Smooth signal



Neighboring nodes have similar values.

Oscillatory signal



Neighboring nodes can have very different values.

Spectral graph methods formalize smooth and oscillatory graph signals.

Laplacian and smoothness

Let A be the adjacency matrix and D the degree matrix. The graph Laplacian is

$$L = D - A.$$

For a graph signal $x \in \mathbb{R}^n$,

$$x^\top L x = \sum_{(u,v)\in E} (x_u - x_v)^2,$$

up to a constant factor. Thus, $x^\top L x$ measures variation along edges.

Laplacian and smoothness

Let A be the adjacency matrix and D the degree matrix. The graph Laplacian is

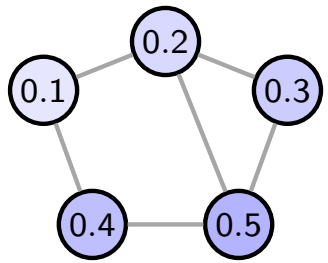
$$L = D - A.$$

For a graph signal $x \in \mathbb{R}^n$,

$$x^\top L x = \sum_{(u,v)\in E} (x_u - x_v)^2,$$

up to a constant factor. Thus, $x^\top L x$ measures variation along edges.

Low variation



$x^\top L x$ small

Smooth across edges.

Laplacian and smoothness

Let A be the adjacency matrix and D the degree matrix. The graph Laplacian is

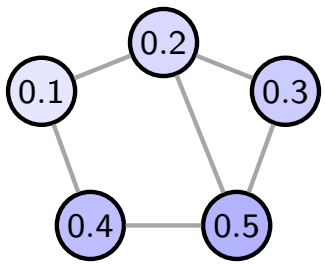
$$L = D - A.$$

For a graph signal $x \in \mathbb{R}^n$,

$$x^\top L x = \sum_{(u,v)\in E} (x_u - x_v)^2,$$

up to a constant factor. Thus, $x^\top L x$ measures variation along edges.

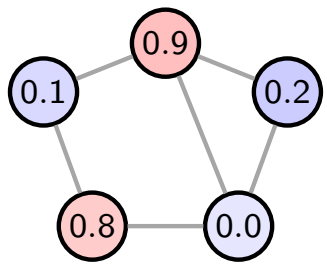
Low variation



$x^\top L x$ small

Smooth across edges.

High variation



$x^\top L x$ large

Rapid changes across edges.

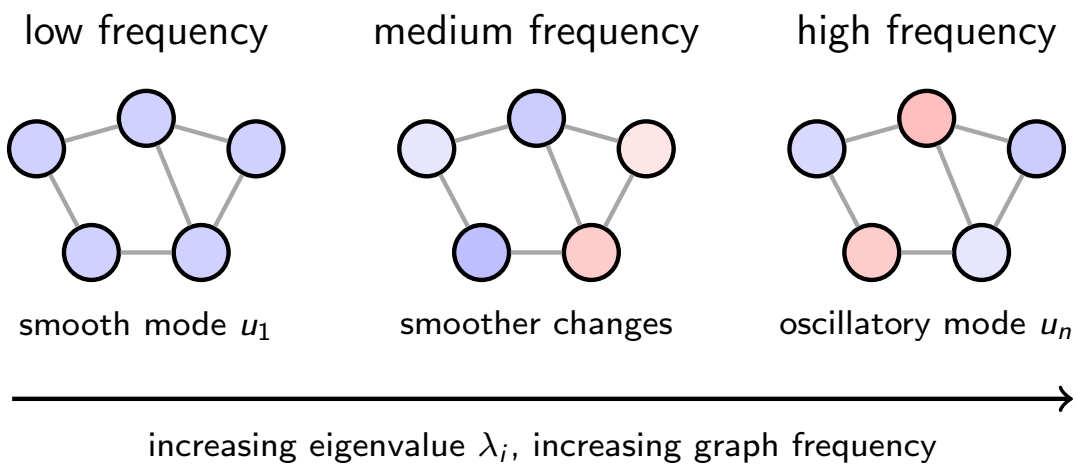
Graph Fourier modes

The Laplacian eigendecomposition is

$$L = U\Lambda U^\top, \qquad U = [u_1, \ldots, u_n].$$

The eigenvectors u_i play the role of Fourier modes on the graph:

$$0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

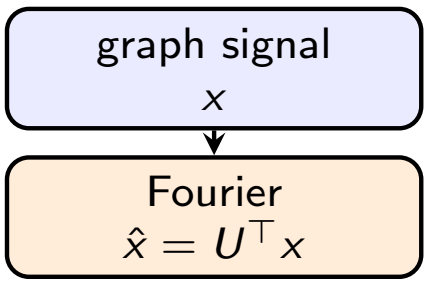


Small eigenvalues correspond to smooth modes; large eigenvalues to oscillatory modes.

Spectral graph convolution

Fourier coordinates

$$\hat{x} = U^\top x, \qquad x = U\hat{x}.$$



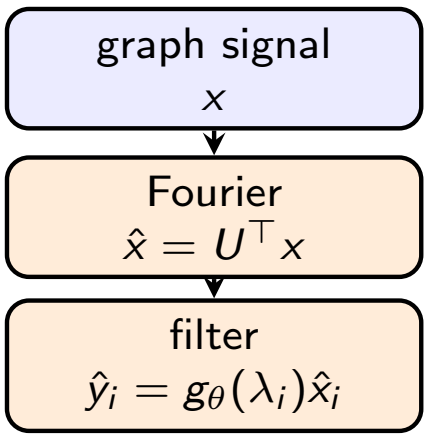
Spectral graph convolution

Fourier coordinates

$$\hat{x} = U^\top x, \qquad x = U\hat{x}.$$

Frequency filter

$$g_\theta : \lambda \mapsto g_\theta(\lambda), \qquad \hat{y}_i = g_\theta(\lambda_i)\hat{x}_i.$$



Spectral graph convolution

Fourier coordinates

$$\hat{x} = U^\top x, \qquad x = U\hat{x}.$$

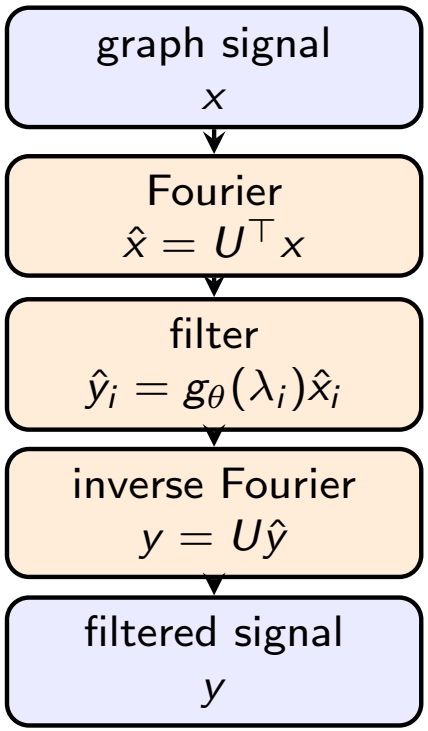
Frequency filter

$$g_\theta : \lambda \mapsto g_\theta(\lambda), \qquad \hat{y}_i = g_\theta(\lambda_i)\hat{x}_i.$$

Matrix form

$$g_\theta(\Lambda) = \operatorname{diag}\left(g_\theta(\lambda_1), \ldots, g_\theta(\lambda_n)\right),$$

$$y = U g_\theta(\Lambda) U^\top x.$$



Spectral graph convolution

Fourier coordinates

$$\hat{x} = U^\top x, \qquad x = U\hat{x}.$$

Frequency filter

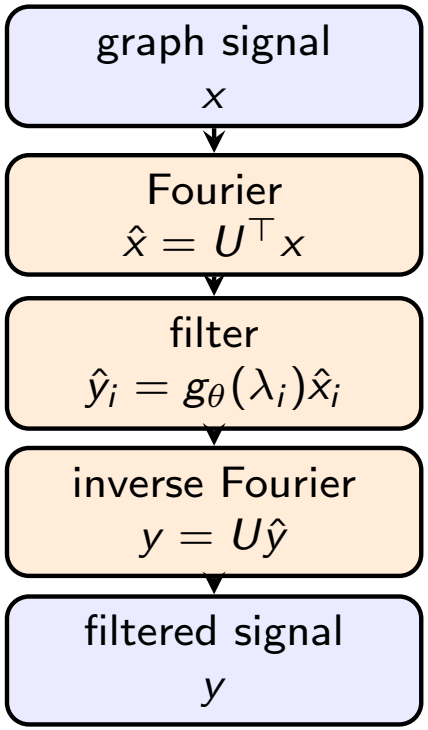
$$g_\theta : \lambda \mapsto g_\theta(\lambda), \qquad \hat{y}_i = g_\theta(\lambda_i)\hat{x}_i.$$

Matrix form

$$g_\theta(\Lambda) = \operatorname{diag}\left(g_\theta(\lambda_1), \ldots, g_\theta(\lambda_n)\right),$$

$$y = U g_\theta(\Lambda) U^\top x.$$

Learnable filtering in the graph Fourier basis.



From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

require the eigenvectors of L and can be global.

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

require the eigenvectors of L and can be global.
A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

require the eigenvectors of L and can be global.
A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

Then

$$g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x.$$

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

require the eigenvectors of L and can be global.
A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

Then

$$g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x.$$



$L^0 x = x$ keeps the node itself.

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

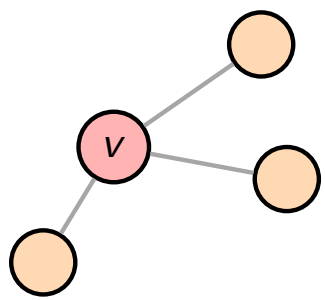
require the eigenvectors of L and can be global.

A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

Then

$$g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x.$$



$L^0 x = x$ keeps the node itself.

Lx mixes nearby nodes.

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

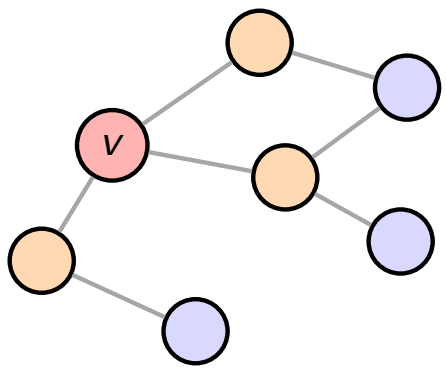
require the eigenvectors of L and can be global.

A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

Then

$$g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x.$$



K-hop receptive field

$L^0 x = x$ keeps the node itself.

Lx mixes nearby nodes.

$L^2 x$ mixes up to two hops.

From spectral filters to local filters

Direct spectral filters

$$y = U g_\theta(\Lambda) U^\top x$$

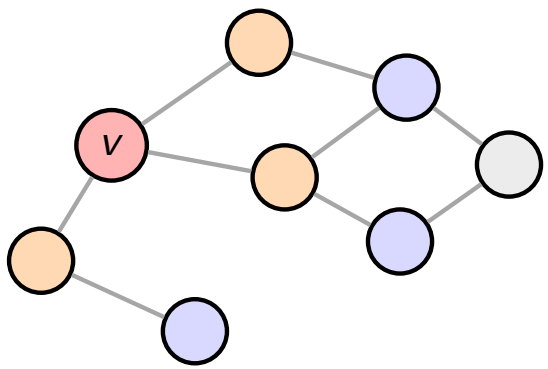
require the eigenvectors of L and can be global.

A common idea is to use polynomial filters:

$$g_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k.$$

Then

$$g_\theta(L)x = \sum_{k=0}^{K} \theta_k L^k x.$$



K-hop receptive field

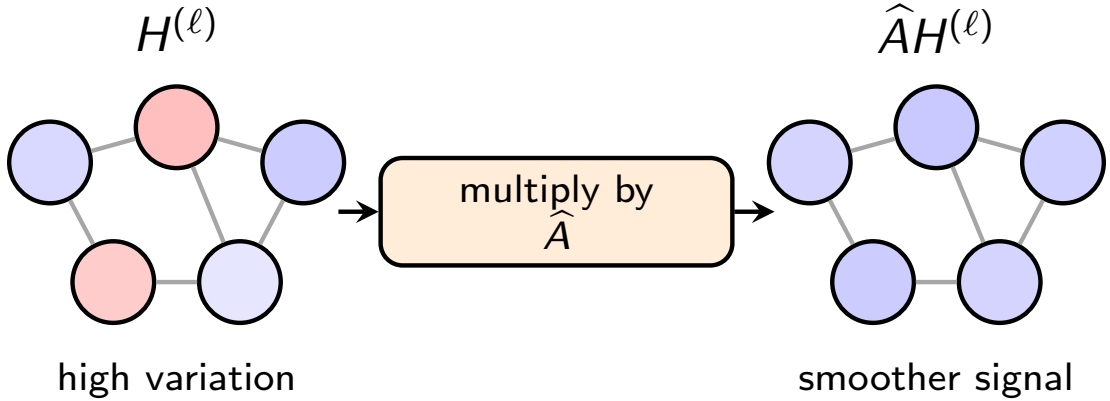
$L^0 x = x$ keeps the node itself.

Lx mixes nearby nodes.

$L^2 x$ mixes up to two hops.

GCN as a low-pass graph filter

$$H^{(\ell+1)} = \sigma\left(\widehat{A}H^{(\ell)}W^{(\ell)}\right) \qquad \text{with} \qquad \widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}.$$



$\widehat{A}$ averages neighbors: it preserves smooth components and attenuates oscillatory ones.

Thus, a GCN layer behaves like a low-pass graph filter.

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN

Spectral view of GNNs

Limitations and extensions

Some applications

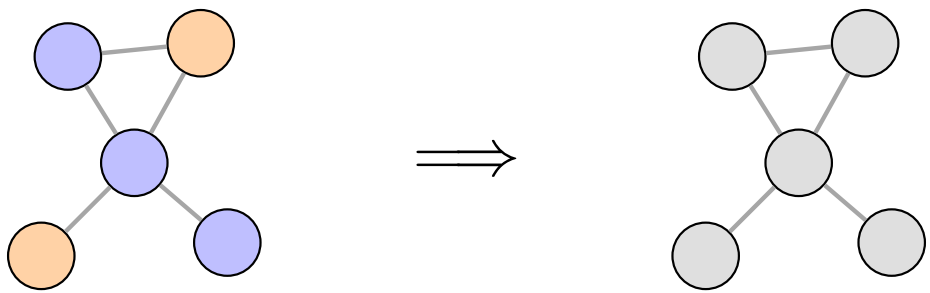
Conclusion

Oversmoothing

Each layer averages information with neighbors.

With many layers, node embeddings can become too similar:

$$h_u^{(k)} \approx h_v^{(k)} \quad \text{for many nodes } u, v.$$



early layers many layers

This is one reason many practical GNNs are shallow (3-4 layers usually).

Limitations of message passing

Message passing is simple and scalable, but it has limitations.

The graph is a useful inductive bias, but not always the right one.

Limitations of message passing

Message passing is simple and scalable, but it has limitations.

Very large graphs
hard to keep 10B nodes
and features in memory

The graph is a useful inductive bias, but not always the right one.

Limitations of message passing

Message passing is simple and scalable, but it has limitations.

Very large graphs
hard to keep 10B nodes and features in memory

Oversmoothing
limits the expressiveness of deep GNNs

The graph is a useful inductive bias, but not always the right one.

Limitations of message passing

Message passing is simple and scalable, but it has limitations.

Very large graphs
hard to keep 10B nodes and features in memory

Oversmoothing
limits the expressiveness of deep GNNs

Long-range signals
distant information is hard to use

The graph is a useful inductive bias, but not always the right one.

Limitations of message passing

Message passing is simple and scalable, but it has limitations.

Very large graphs
hard to keep 10B nodes and features in memory

Oversmoothing
limits the expressiveness of deep GNNs

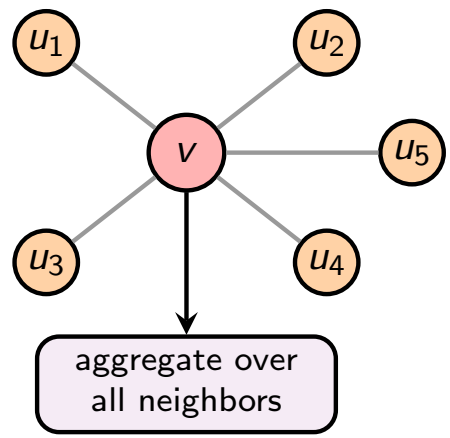
Long-range signals
distant information is hard to use

Heterophily
neighbors may have different labels

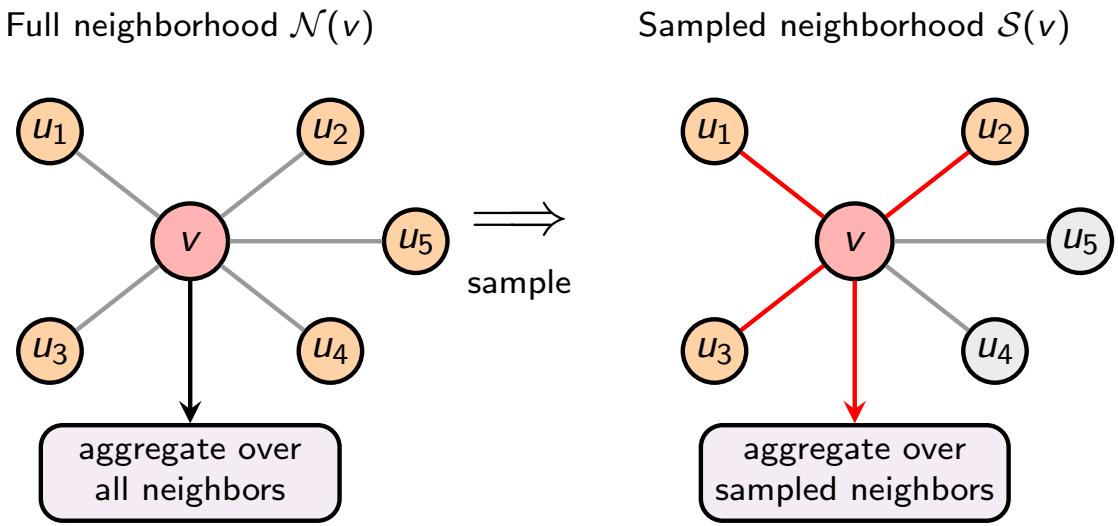
The graph is a useful inductive bias, but not always the right one.

Neighborhood sampling

Full neighborhood $\mathcal{N}(v)$



Neighborhood sampling



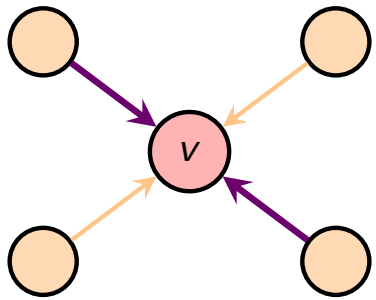
Instead of aggregating over all neighbors, sample $\mathcal{S}(v) \subseteq \mathcal{N}(v)$:

$$m_v^{(\ell)} = \mathrm{AGG}\left(\{h_u^{(\ell)} : u \in \mathcal{S}(v)\}\right).$$

Is attention all you need?

Two ways to add attention to graph learning:

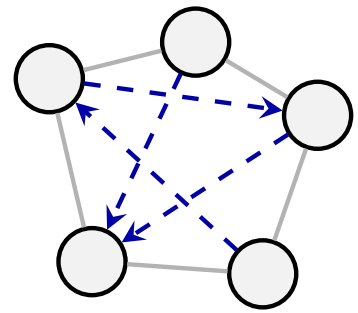
Graph Attention network (GAT): local attention



Learn weights on neighbors instead of using mean/sum.

$$h'_v = \sum_{u \in \mathcal{N}(v)} \alpha_{vu} W h_u$$

Graph Transformer: global attention



Let nodes attend beyond local neighbors.

Need tricks for graph structure and quadratic cost.

Attention is promising, but not a free lunch on large graphs.

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN

Spectral view of GNNs

Limitations and extensions

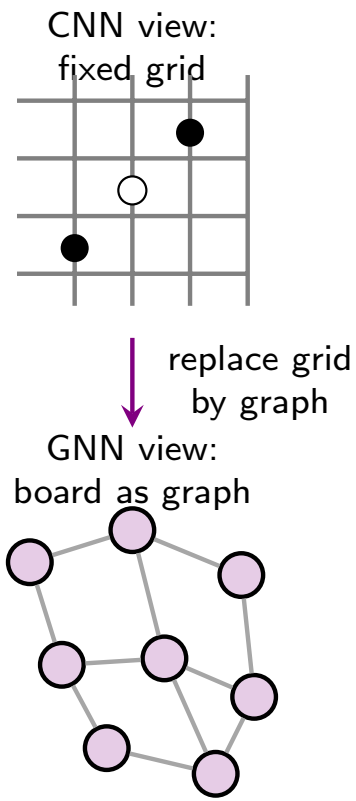
Some applications

Conclusion

Board games: AlphaZero with GNNs

AlphaZero-style agents evaluate a position with a neural network:

$$\text{board state} \longrightarrow (\text{policy}, \text{value}).$$



Goal. Evaluate a game position and choose strong moves.

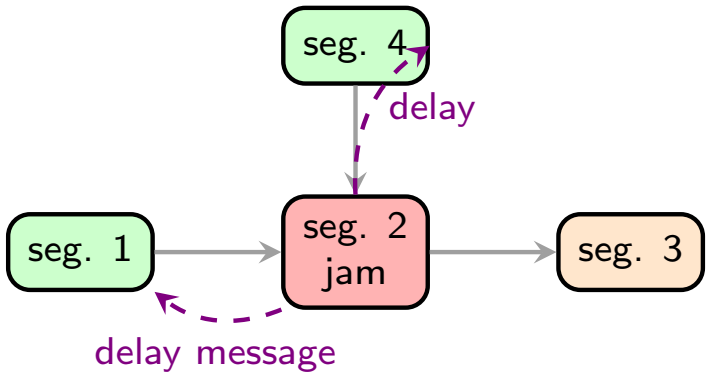
Classical AlphaZero backbone. CNNs work well on regular board grids, such as Go or chess.

GNN idea. Represent the board as a graph and pass messages between related positions.

Why useful? The same architecture can handle irregular board structure and transfer better across board sizes.

Reference: Ben-Assayag and El-Yaniv, *Train on Small, Play the Large: Scaling Up Board Games with AlphaZero and GNN*, 2021.

Logistics: Google Maps ETA



road segments as graph nodes

Goal. Predict estimated time of arrival in real time.

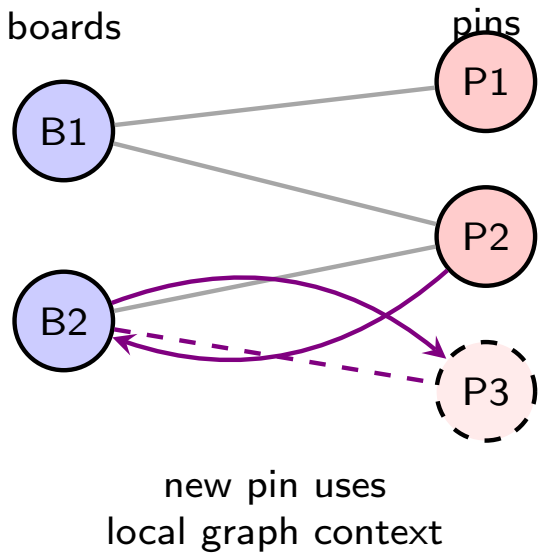
Graph view. Road segments are nodes; intersections define edges.

Why useful? Congestion propagates through the road network. A GNN can pass delay information to nearby segments.

Impact. Deployed in Google Maps; reported large ETA accuracy gains, up to 50% in some cities.

Reference: Derrow-Pinion et al., *ETA Prediction with Graph Neural Networks in Google Maps*, CIKM 2021.

Recommender systems: PinSage



Goal. Recommend related pins at Pinterest scale.

Graph view. Boards and pins form a large bipartite graph.

Why useful? A new or sparse item can still get an embedding from its neighborhood.

Impact. PinSage learns embeddings for billions of pins for large-scale recommendation.

Reference: Ying et al., *Graph Convolutional Neural Networks for Web-Scale Recommender Systems*, KDD 2018.

Where are we?

Introduction: Motivation and Deep learning reminder

From CNN to GNN

Spectral view of GNNs

Limitations and extensions

Some applications

Conclusion

From graphs to GNNs

A GNN combines the graph structure with node features to make predictions.



Big picture

The graph tells the model **where information can flow**; features tell it **what information flows**.

Graphs are useful only when edges bring predictive information.

Takeaways

1. GNNs learn representations by passing messages along graph edges.
2. A k-layer GNN uses information from the k-hop neighborhood.
3. GCN and GraphSAGE are two standard message-passing architectures.
4. GNNs are useful when edges encode predictive signal.