From Neural Networks to Graph Neural Networks

Antonio Longa

April 10, 2026

UiT The Arctic University of Norway

antonio.longa@uit.no

Outline

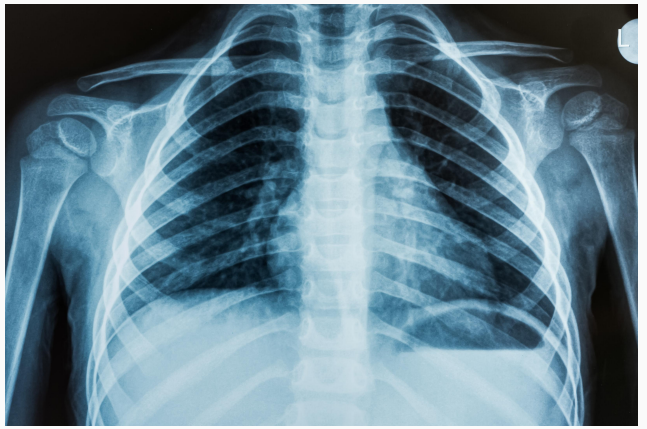
- **Motivation**
  - Supervised learning as function approximation
  - From feature engineering to deep learning
- **Multi-Layer Perceptrons (MLPs)**
  - Perceptron and multi-layer perceptrons
  - Activation functions and output layers
  - Training, generalization, and expressivity
- **Learning on Graphs**
  - Graphs as relational data
  - Node-, edge-, and graph-level tasks
  - Why MLPs are not enough
- **Graph Neural Networks**
  - Message passing
  - Graph Convolutional Networks (GCNs)
  - Limitations: over-smoothing, over-squashing, expressivity
- **Research directions in our group**

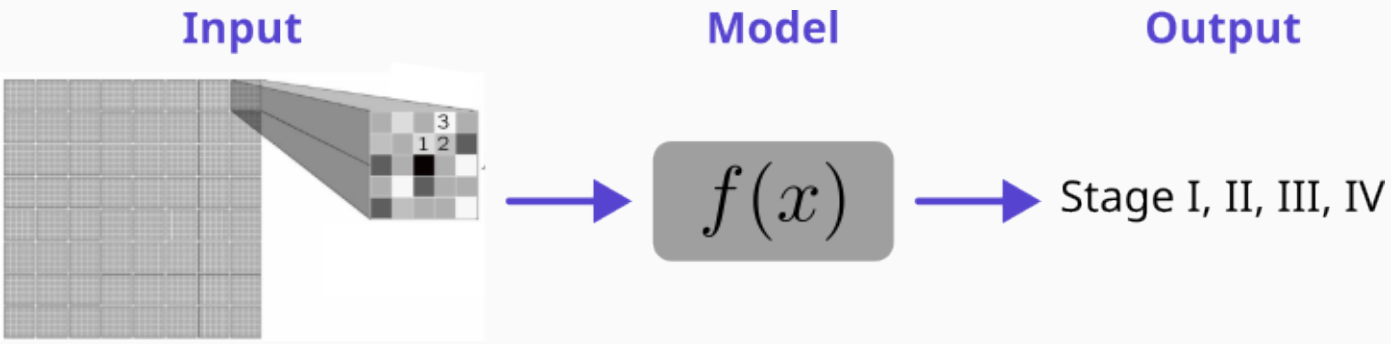
Motivation

A Real Problem [Motivation]

Given an X-ray image:

$\rightarrow$ determine the stage of a tumor





A Real Problem [Motivation]

Formally, we consider an input space $\mathcal{X}$ and an output space $\mathcal{Y}$:

$$x \in \mathcal{X} \qquad y \in \mathcal{Y}$$

We assume that there exists an unknown function:

$$f^\star : \mathcal{X} \to \mathcal{Y}$$

and we are given samples:

$$(x_i, y_i) \sim \mathcal{D}$$

Our goal is to learn a function f_θ such that:

$$f_\theta(x) \approx f^\star(x)$$

Learning Settings [Motivation]

Different learning paradigms arise depending on the available data.

- **Supervised learning**
  - data: $\{(x_i, y_i)\}_{i=1}^{n}$
  - goal: learn $f_\theta(x) \approx y$
  - e.g. tumor stage prediction, image classification

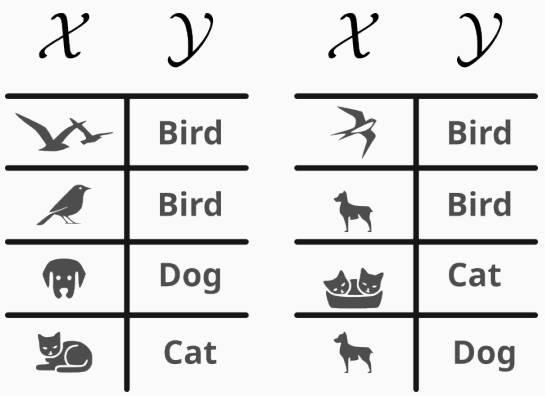
$\mathcal{X}$	$\mathcal{Y}$
	Bird
	Bird
	Dog
	Cat

$\mathcal{X}$	$\mathcal{Y}$
	Bird
	Bird
	Cat
	Dog

Learning Settings [Motivation]

Different learning paradigms arise depending on the available data.

- **Supervised learning**
  - data: $\{(x_i, y_i)\}_{i=1}^n$
  - goal: learn $f_\theta(x) \approx y$
  - e.g. tumor stage prediction, image classification



- **Unsupervised learning**
  - data: $\{x_i\}_{i=1}^n$
  - goal: discover hidden structure in the data
  - examples: clustering

In this talk, we focus on supervised learning.

The classical approach (pre deep learning) [Motivation]

In classical approaches, we decompose the problem as:

$$f(x) = g(\phi(x))$$

where:

- $\phi : \mathcal{X} \to \mathbb{R}^d$ is a feature map
- $g : \mathbb{R}^d \to \mathcal{Y}$ is a simple predictor

The classical approach (pre deep learning) [Motivation]

In classical approaches, we decompose the problem as:

$$f(x) = g(\phi(x))$$

where:

- $\phi : \mathcal{X} \to \mathbb{R}^d$ is a feature map
- $g : \mathbb{R}^d \to \mathcal{Y}$ is a simple predictor

How to design ϕ?

- Detect contours
- Contrast
- Irregularity of regions

The classical approach (pre deep learning) [Motivation]

In classical approaches, we decompose the problem as:

$$f(x) = g(\phi(x))$$

where:

- $\phi : \mathcal{X} \to \mathbb{R}^d$ is a feature map
- $g : \mathbb{R}^d \to \mathcal{Y}$ is a simple predictor

How to design ϕ?

- Detect contours
- Contrast
- Irregularity of regions

ϕ must be designed manually and this requires:

- domain knowledge
- trial and error

Moreover, the space of possible feature maps is extremely large, and there is no systematic way to choose a good one.

The deep learning approach [Motivation]

Deep learning changes this paradigm.

Instead of fixing ϕ, we parametrize both ϕ and g:

$$f_\theta(x) = g_{\theta_2}\big(\phi_{\theta_1}(x)\big)$$

where:

- ϕ_{θ_1} is a learned representation
- g_{θ_2} is a learned predictor

Equivalently, we consider a single parametric function:

$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

and we learn all parameters jointly from data.

The deep learning approach [Motivation]

Deep learning tells us that, instead of manually designing features, we should learn a parametric function directly from data.

This raises a fundamental question:

How do we choose the family of functions f_θ?

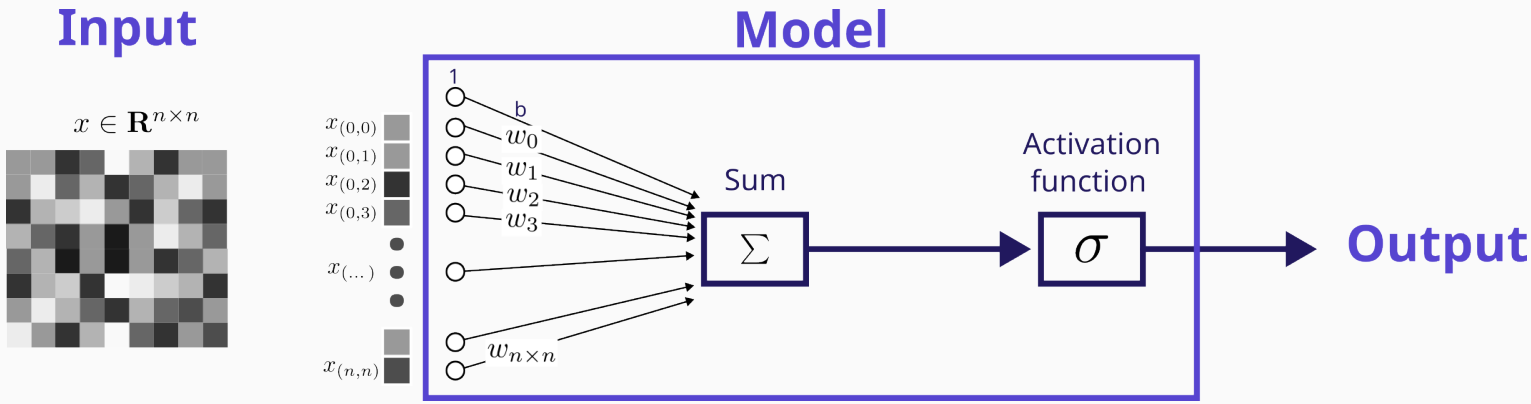
A natural answer is to consider **neural networks**, that is, highly flexible parametric models obtained by composing simple transformations.

The simplest and most classical example is the **Multi-Layer Perceptron (MLP)**.

Multi-Layer Perceptron (MLP)

Perceptron

We start from the simplest parametric model: the **Perceptron**.



Given an input $x \in \mathbb{R}^d$, the perceptron computes:

$$f_\theta(x) = w^\top x + b$$

and produces a prediction:

$$\hat{y} = \sigma(w^\top x + b)$$

where:

- $w \in \mathbb{R}^d$ are the weights
- $b \in \mathbb{R}$ is the bias
- σ is an activation function

Perceptron

The perceptron defines a linear decision boundary:

$$w^\top x + b = 0$$

Therefore, it can only model **linearly separable** problems.

However, many real-world tasks require modeling **nonlinear relationships**, which a single perceptron cannot capture.

Key idea: increase expressivity by composing multiple perceptrons...

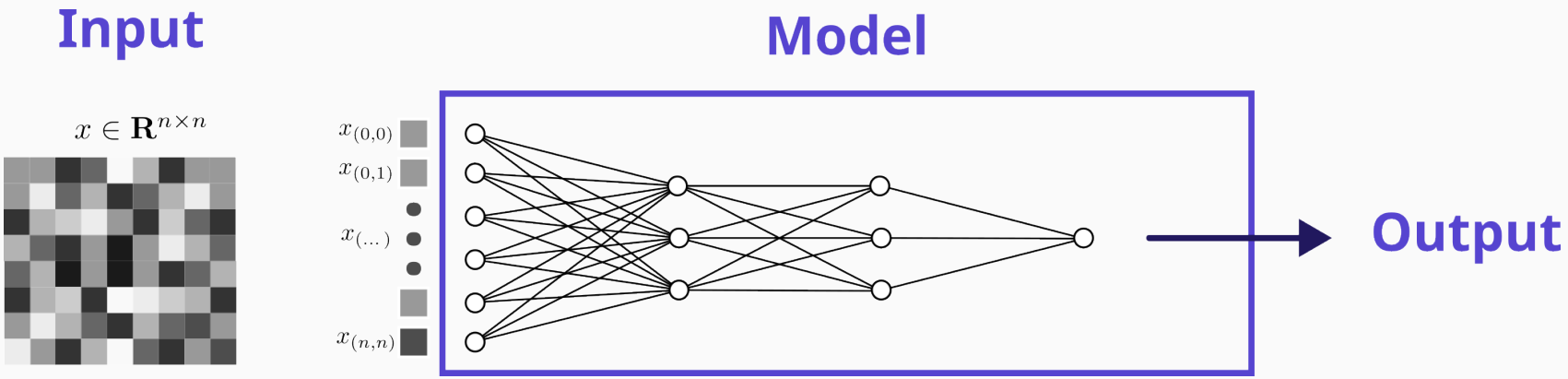
Multi-Layer Perceptron (MLP)

Multi-Layer Perceptron (MLP)

An MLP is obtained by stacking multiple perceptrons in layers.

Each layer takes as input the output of the previous one:

$$x \to h^{(1)} \to h^{(2)} \to \cdots \to \hat{y}$$

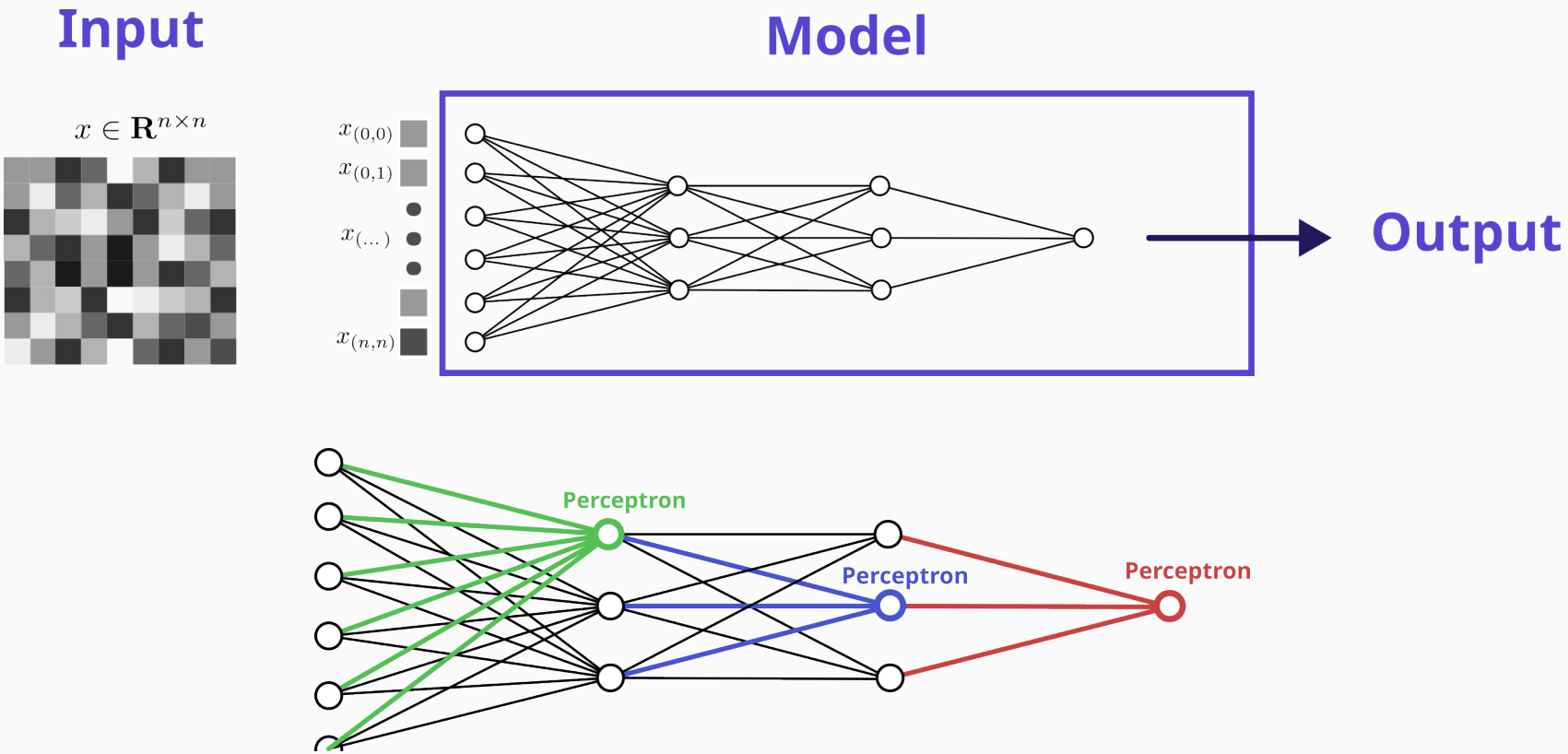


Multi-Layer Perceptron (MLP)

An MLP is obtained by stacking multiple perceptrons in layers.

Each layer takes as input the output of the previous one:

$$x \to h^{(1)} \to h^{(2)} \to \cdots \to \hat{y}$$



Multi-Layer Perceptron (MLP)

Formally, an MLP is a function defined by the composition of L layers.

$$
\begin{aligned}
h^{(0)} &= x \\
h^{(\ell)} &= \sigma\big(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}\big), \qquad \ell = 1, \dots, L-1 \\
f_\theta(x) &= W^{(L)} h^{(L-1)} + b^{(L)}
\end{aligned}
$$

The parameters are:

$$
\theta = \{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L}
$$

Multi-Layer Perceptron (MLP)

The MLP progressively builds internal representations of the input.

Each layer transforms the data into a more useful representation:

$$x \to h^{(1)} \to h^{(2)} \to \dots$$

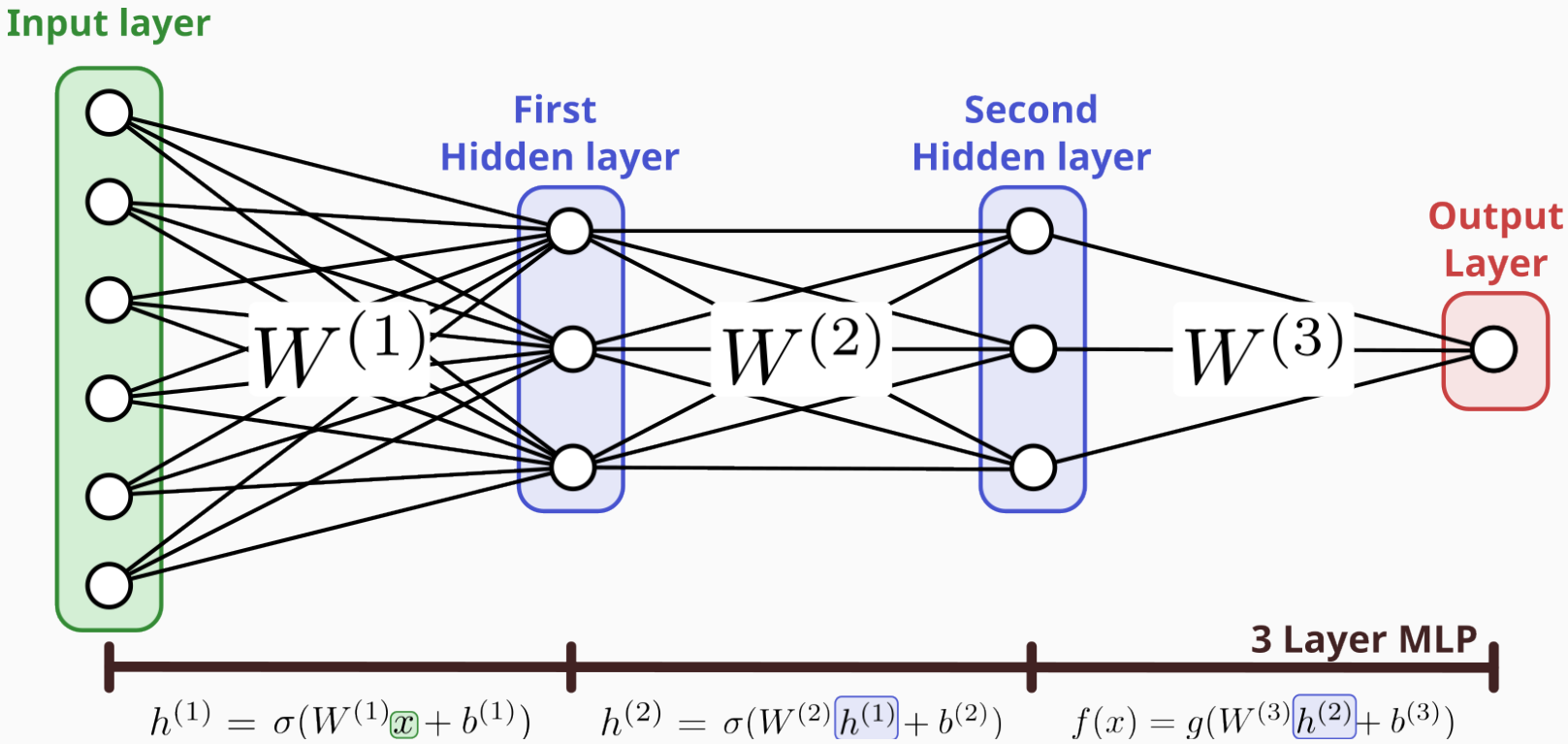
We can interpret:

$$\phi_\theta(x) = h^{(L-1)}$$

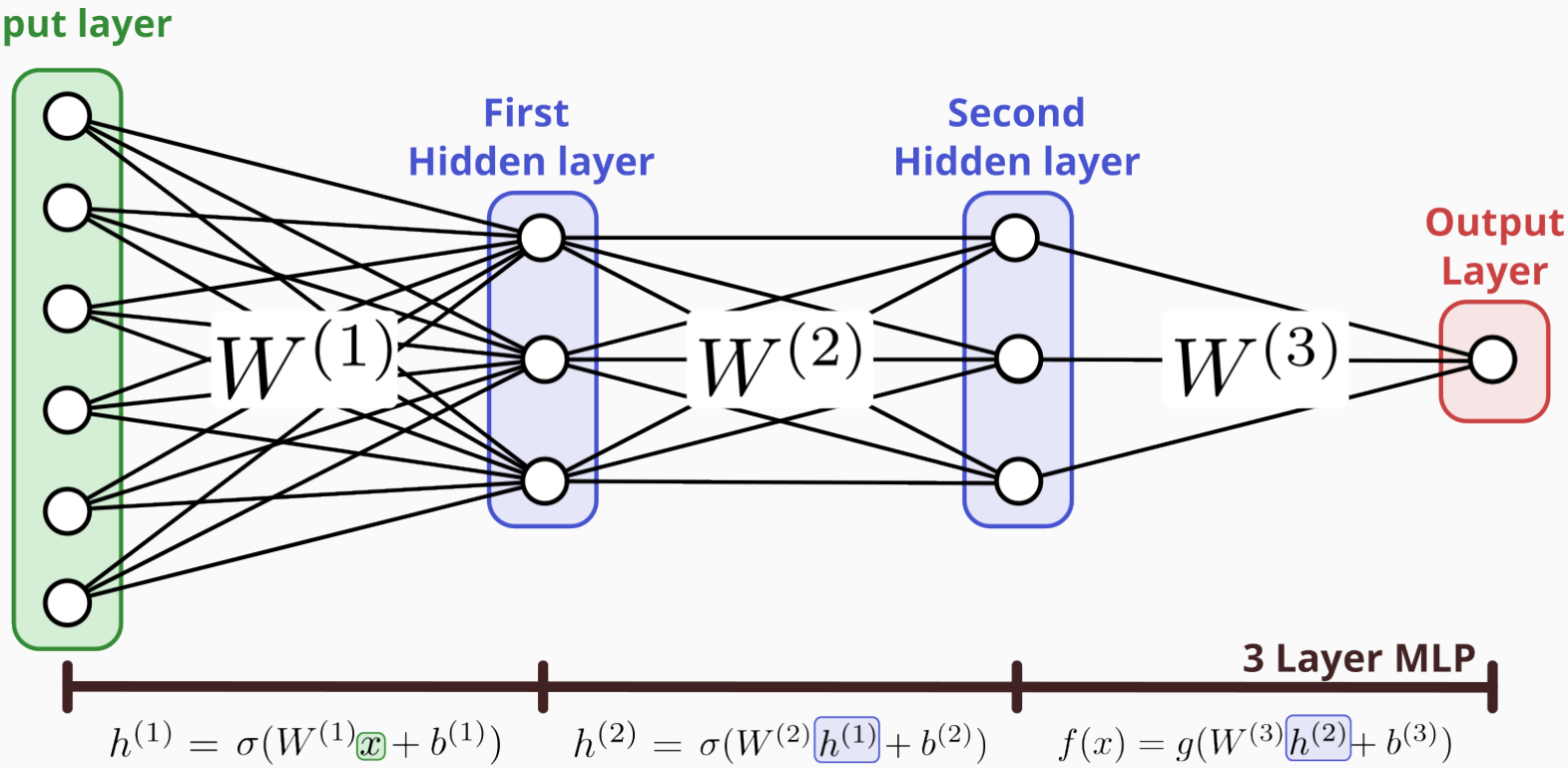
as a learned feature map, and the final layer as a predictor.

Thus, feature extraction and prediction are learned jointly from data.

Multi-Layer Perceptron (MLP) Visual



Multi-Layer Perceptron (MLP) Visual



$$f(x) = g(W^{(3)}\sigma(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + b^{(2)}) + b^{(3)})$$

"Feature map"

Simple classifier

MLP: Activation Functions and Output Layers

Nonlinear activation functions σ. Common examples include:

ReLU:	**Tanh:**	**Sigmoid:**
$\sigma(x) = \max(0, x)$	$\sigma(x) = \tanh(x)$	$\sigma(x) = \frac{1}{1+e^{-x}}$

They introduce nonlinearity and allow the network to model complex functions.

MLP: Activation Functions and Output Layers

Nonlinear activation functions σ. Common examples include:

ReLU:	**Tanh:**	**Sigmoid:**
$\sigma(x) = \max(0, x)$	$\sigma(x) = \tanh(x)$	$\sigma(x) = \frac{1}{1+e^{-x}}$

They introduce nonlinearity and allow the network to model complex functions.

Output (Readout) Layer

The final layer maps learned representations to the output space:

$$f(x) = g(h_L(x))$$

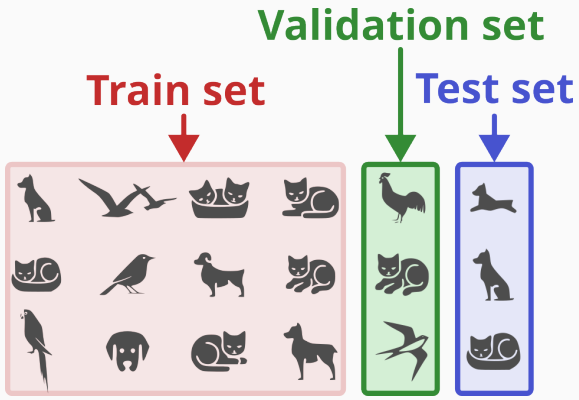
Regression	**Binary clas.**	**Multiclass clas.**
linear output	sigmoid	softmax
Predicting the price of a house.	Fraudulent or legitimate transaction.	Predicting the stage of the tumor.

Training, Generalization and UAT [short]

1. Training (Optimization)

Data is typically split into:

- **Training set**: fit the model
- **Validation set**: model selection/tuning
- **Test set**: final evaluation

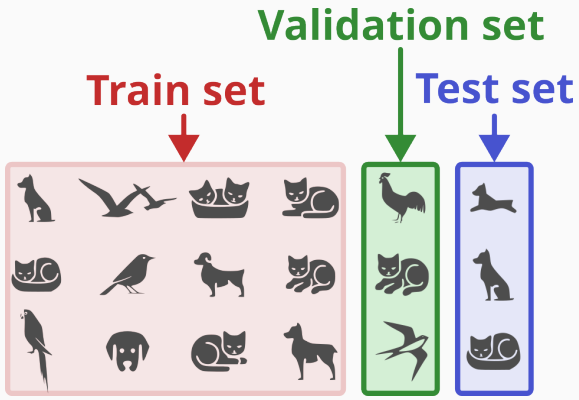


Training, Generalization and UAT [short]

1. Training (Optimization)

Data is typically split into:

- **Training set**: fit the model
- **Validation set**: model selection/tuning
- **Test set**: final evaluation



Given training data $\{(x_i, y_i)\}_{i=1}^n$, we define a loss function:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(f_\theta(x_i), y_i\big)$$

The goal is to find parameters that minimize the loss:

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta)$$

Training, Generalization and UAT [short]

The loss function measures how well the model fits the data.

Common choices depend on the task:

- **Regression:** Mean Squared Error

$$\ell(y, f_\theta(x_i)) = (y - f_\theta(x_i))^2$$

- **Binary classification:** Binary Cross-Entropy

$$\ell(y, f_\theta(x_i)) = -y \log f_\theta(x_i) - (1 - y) \log(1 - f_\theta(x_i))$$

- **Multiclass classification:** Cross-Entropy

$$\ell(y, f_\theta(x_i)) = -\sum_{c=1}^{C} y_c \log f_\theta(x_i)_c$$

Key idea: learning = minimizing a suitable loss function.

Training, Generalization and UAT [short]

2. Generalization Our goal is not only to fit the training data, but to perform well on unseen (test) data. We distinguish between:

- **Training error:** performance on observed samples
- **Test error:** performance on new data

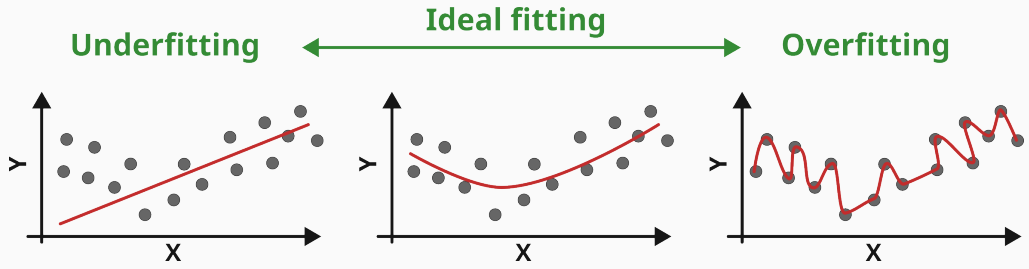
Training, Generalization and UAT [short]

2. Generalization Our goal is not only to fit the training data, but to perform well on unseen (test) data. We distinguish between:

- **Training error:** performance on observed samples
- **Test error:** performance on new data

However, minimizing training error alone may lead to:

- **Underfitting:** model is too simple
- **Overfitting:** model memorizes the data



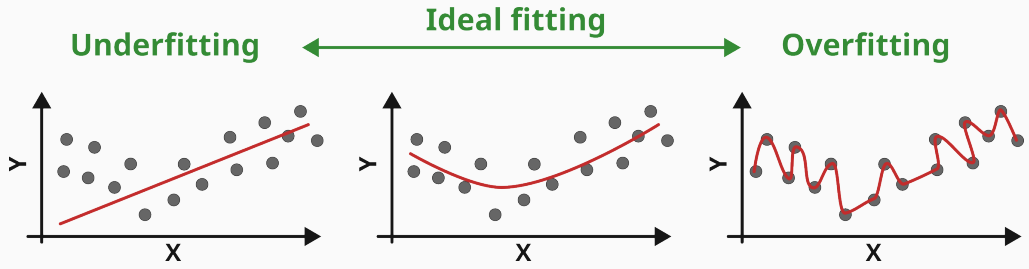
Training, Generalization and UAT [short]

2. Generalization Our goal is not only to fit the training data, but to perform well on unseen (test) data. We distinguish between:

- **Training error:** performance on observed samples
- **Test error:** performance on new data

However, minimizing training error alone may lead to:

- **Underfitting:** model is too simple
- **Overfitting:** model memorizes the data



Evaluation metrics To measure performance on unseen data, we use task-dependent metrics:

- **Regression:** i.e. Mean Squared Error
- **Classification:** i.e. Accuracy

Training, Generalization and UAT [short]

3. Expressivity: Universal Approximation Theorem [2]

- A sufficiently wide MLP can approximate any continuous function on a compact domain
- In principle, even a single hidden layer is enough
- However, the theorem is only an **existence result**:
  - it does **not** say how many neurons are needed
  - it does **not** say whether the network can be trained efficiently

Take-home message: MLPs are expressive, but expressivity does not imply practical learnability.

Recap: From Data to Neural Networks

We started from a supervised learning problem:

$$(x, y) \sim \mathcal{D}, \quad f^\star : \mathcal{X} \to \mathcal{Y}$$

Classical approach:

$$f(x) = g(\phi(x)) \quad \text{with handcrafted features}$$

Deep learning:

$$f_\theta(x) = g_{\theta_2}(\phi_{\theta_1}(x)) \quad \text{learned from data}$$

MLPs:

- composition of linear maps + nonlinear activations
- learn representations and predictions jointly

Learning process:

- minimize a loss function
- aim for good generalization on unseen data

Learning on graphs

Limitations of MLPs

So far, we considered inputs as vectors:

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

MLPs operate on fixed-size, independent inputs.

Limitation:

- They do not model relationships between different data points
- They ignore interactions and dependencies

However, in many real-world problems, data are inherently **relational**.

Relational data can be modeled as a graph.

Graphs

Relational data can be modeled as a graph:

A graph is defined as:

$$G = (V, E, X_V, X_E)$$

where:

- V nodes
- $E \subseteq V \times V$ edges
- $X_V \in \mathbb{R}^{|V| \times d}$ node features
- $X_E \in \mathbb{R}^{|E| \times d'}$ edge features

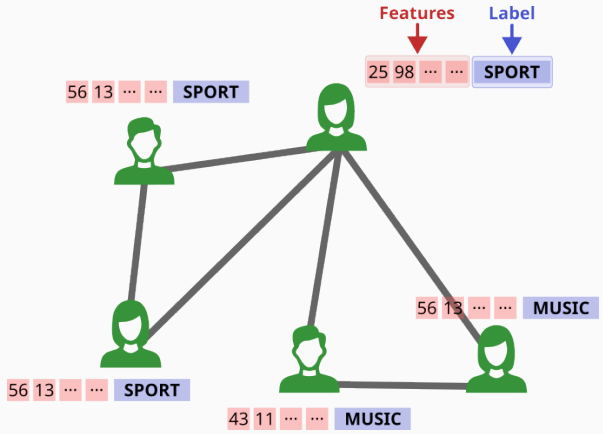
Labels can be associated to different elements:

$$\text{nodes: } y_v, \quad \text{edges: } y_{(u,v)}, \quad \text{graph: } y_G$$

Tasks on Graphs

Node-level tasks. Predict a label for each node:

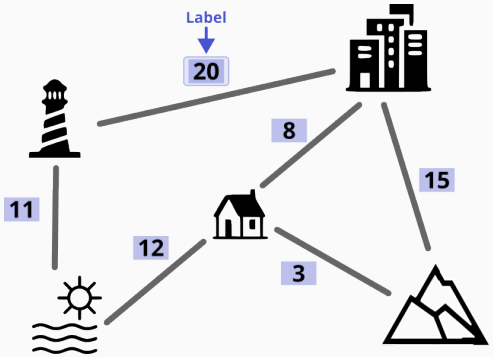
$$y_v \in \mathcal{Y}, \quad v \in V$$

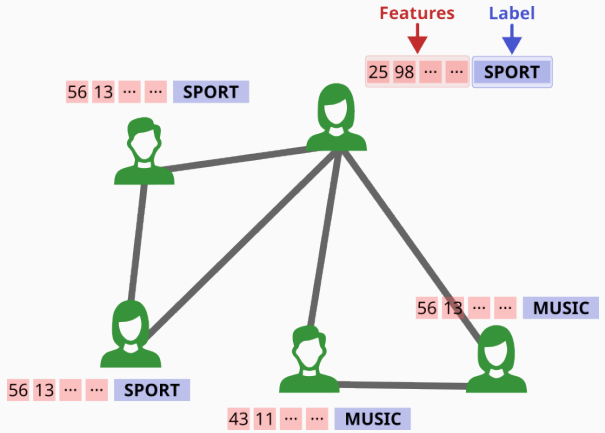


Tasks on Graphs

Node-level tasks. Predict a label for each node:

$$y_v \in \mathcal{Y}, \quad v \in V$$





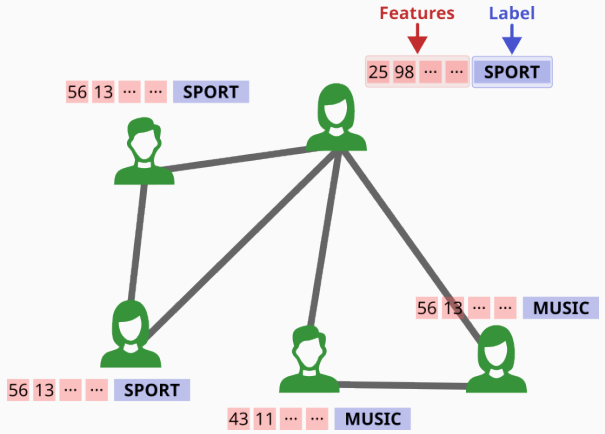
Edge-level tasks. Predict properties of edges or missing links:

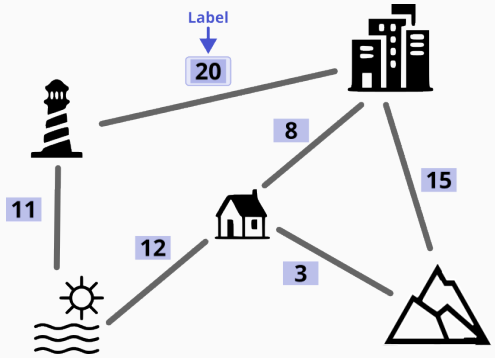
$$y_{(u,v)} \in \mathcal{Y}, \quad (u,v) \in E$$

Tasks on Graphs

Node-level tasks. Predict a label for each node:

$$y_v \in \mathcal{Y}, \quad v \in V$$



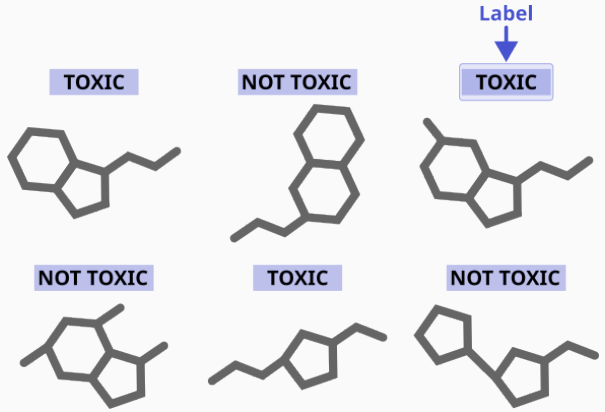


Edge-level tasks. Predict properties of edges or missing links:

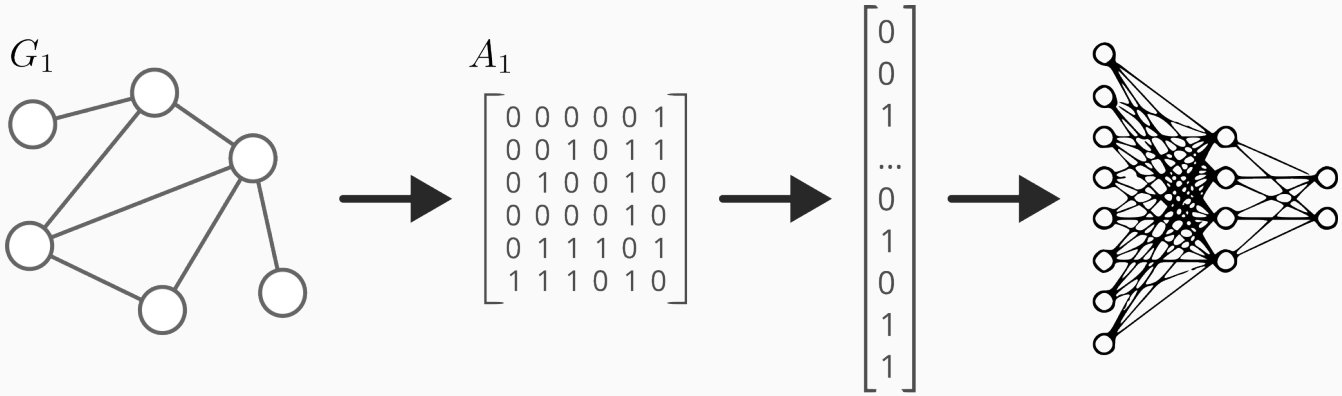
$$y_{(u,v)} \in \mathcal{Y}, \quad (u,v) \in E$$

Graph-level tasks. Predict a label for the entire graph:

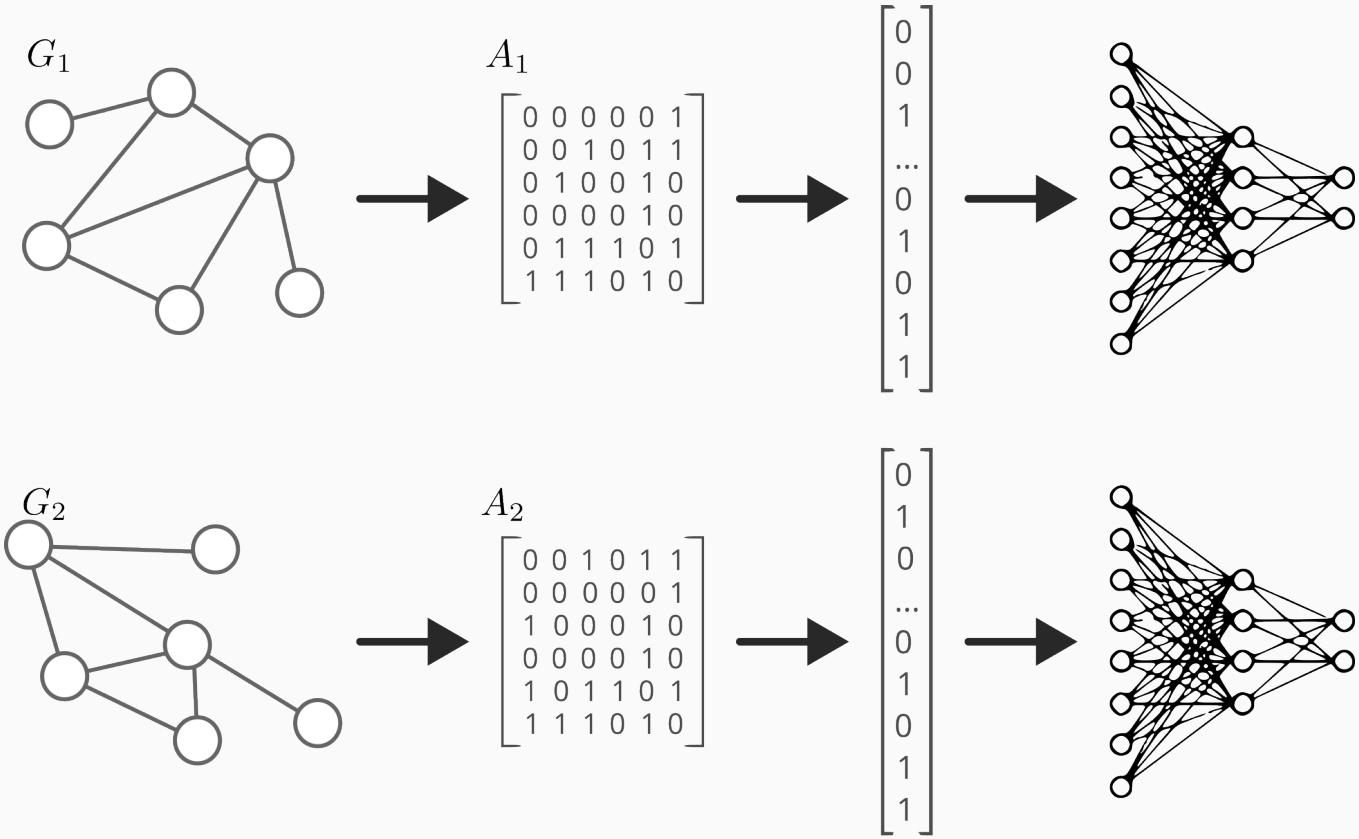
$$y_G \in \mathcal{Y}$$



Why not use an MLP on graphs?



Why not use an MLP on graphs?



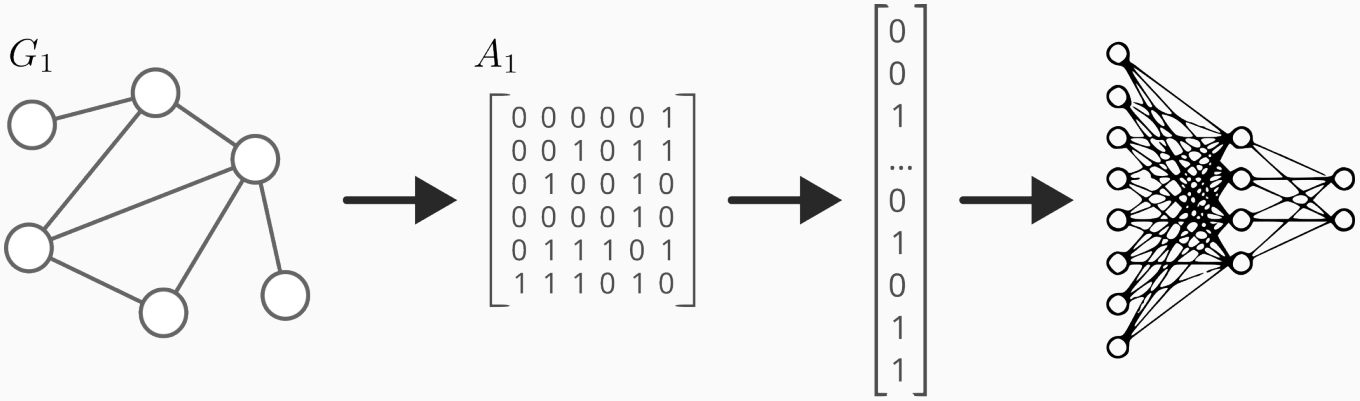
$$G_1 \cong G_2 \quad \text{but} \quad A_1 \neq A_2$$

Two graphs G_1 and G_2 are **isomorphic** ($G_1 \cong G_2$) if there exists a permutation $\pi : V \to V$ such that: $(i, j) \in E_1 \iff (\pi(i), \pi(j)) \in E_2$

Let P be the permutation matrix associated to π. Then: $A_2 = PA_1P^\top$

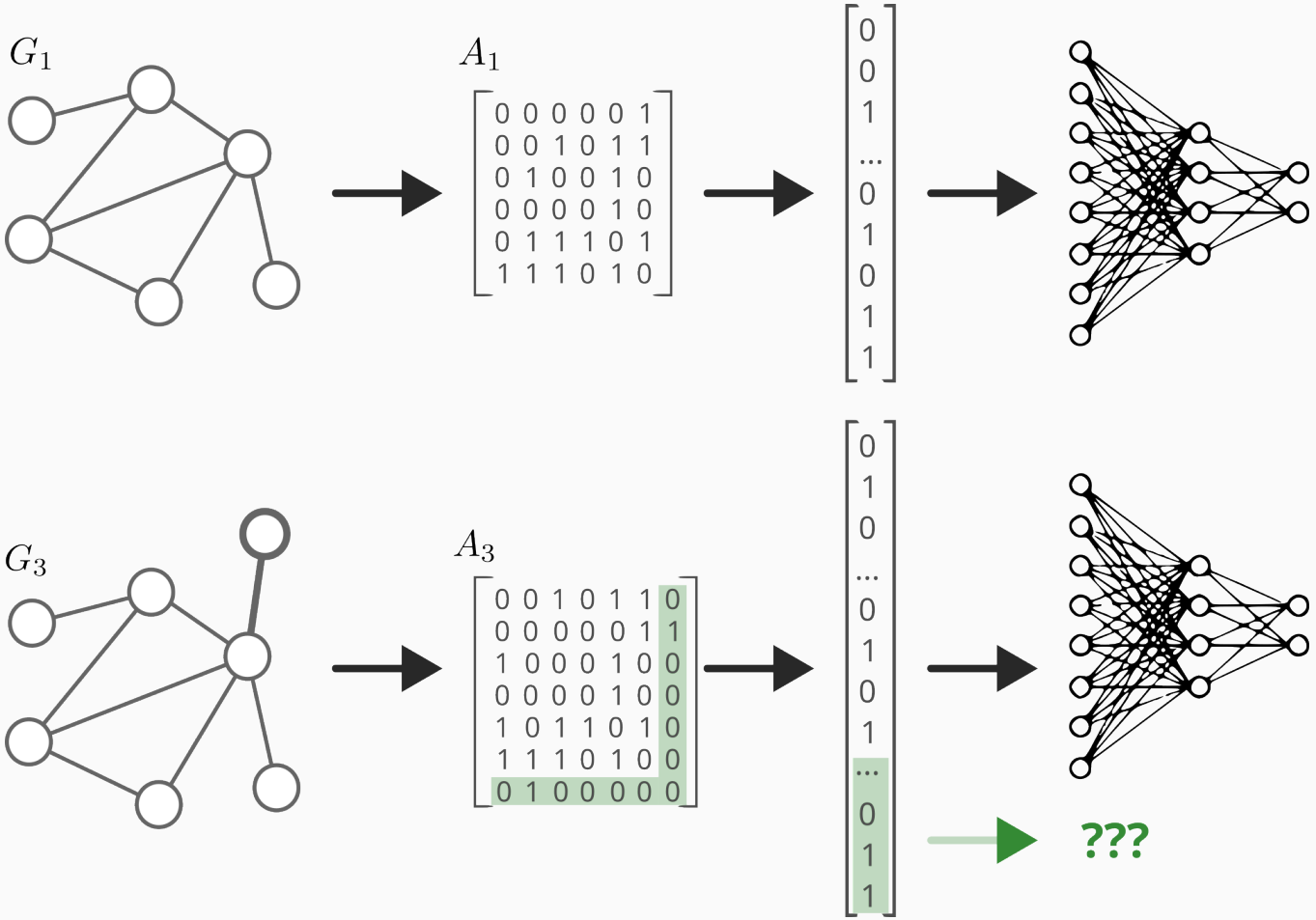
Why not use an MLP on graphs?

2. Different sizes



Why not use an MLP on graphs?

2. Different sizes



What do we want from a model on graphs?

A good model on graphs should:

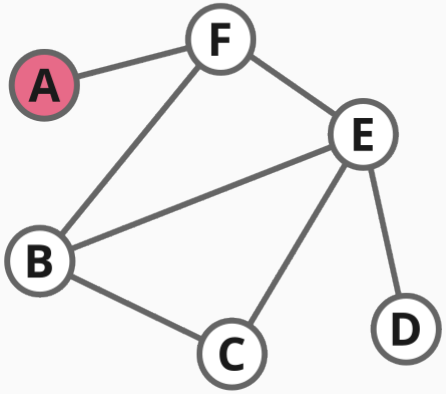
- Be **permutation invariant**
- Handle **variable-size graphs**
- Exploit **local structure** (neighbors matter)
- Share parameters across nodes (scalability)

Key idea: learn representations by aggregating information from neighbors.

Graph Neural Networks

Intuition

Input Graph

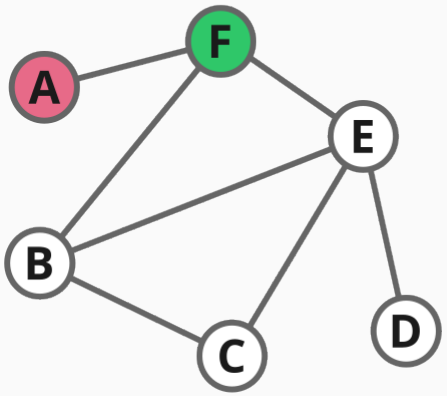


Computation Graph

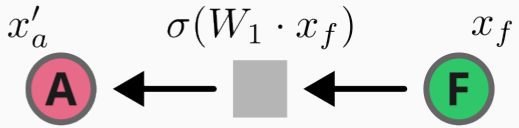
x'_a

Intuition

Input Graph

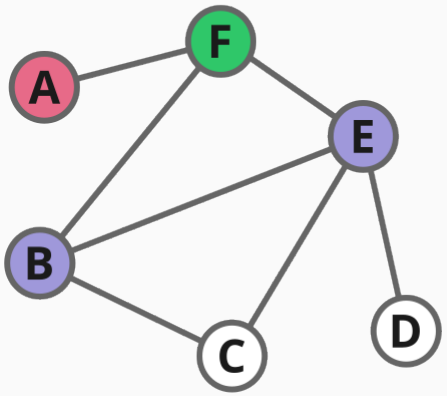


Computation Graph

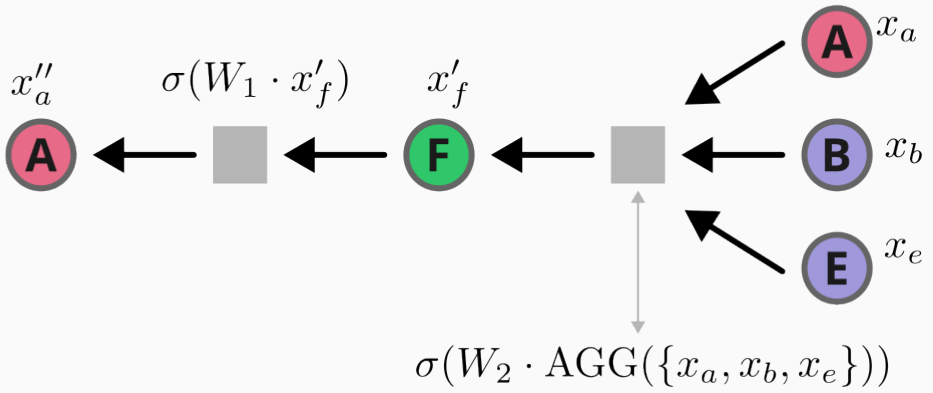


Intuition

Input Graph

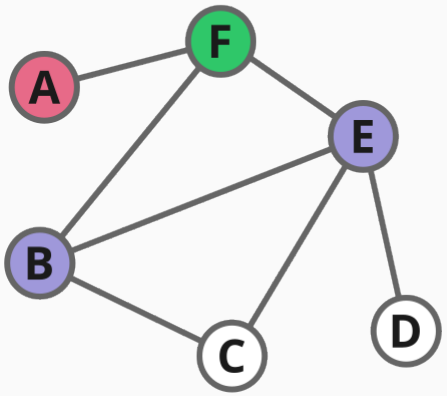


Computation Graph

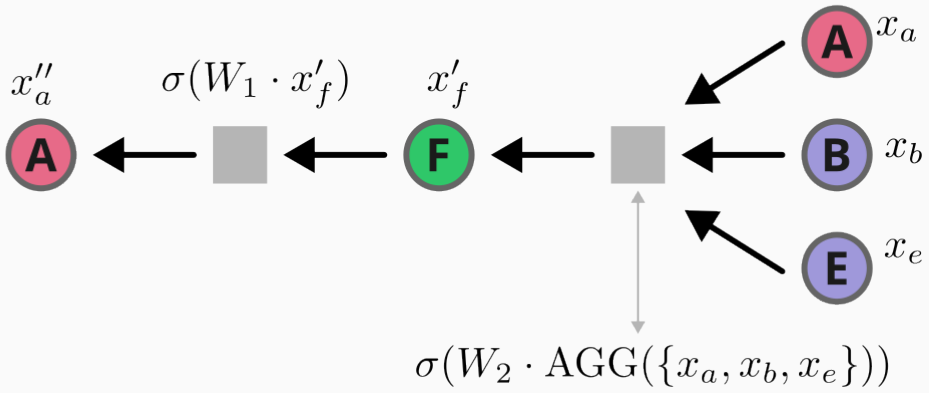


Intuition

Input Graph



Computation Graph



Each node representation is updated by aggregating information from its neighbors.

This process can be decomposed into three steps:

- **Message:** each neighbor sends information
- **Aggregation:** messages are combined
- **Update:** the node updates its feature

Why does message passing work?

Key assumption: local dependency structure

In many real-world graphs, neighboring nodes tend to be **similar** or **correlated**. **Examples:**

- Social networks: connected users share interests
- Citation networks: linked papers discuss related topics
- Molecules: bonded atoms influence each other

Key idea:

- propagate information locally
- progressively enlarge the receptive field

After k layers, each node has access to its k-hop neighborhood.

Message Passing Neural Networks (MPNN)

Let $x_i^{(k)}$ be the feature of node i at layer k.

A general GNN can be written as [3]:

$$m_i^{(k)} = \mathrm{AGG}\Big(\{M(x_i^{(k)}, x_j^{(k)}, e_{ij}) : j \in \mathcal{N}(i)\}\Big)$$

$$x_i^{(k+1)} = U\big(x_i^{(k)}, m_i^{(k)}\big)$$

Components:

- M: message function
- AGG: permutation-invariant aggregation (sum, mean, max)
- U: update function

Key idea: this defines a general class of Graph Neural Networks.

From general to specific models

The MPNN framework defines a large family of models.

Question: how should we choose M and U?

Different choices lead to different GNN architectures.

Example: Graph Convolutional Networks (GCN) [5]

Graph Convolutional Networks (GCN)

A simple and widely used instantiation of message passing is given by [5]:

$$X^{(k+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}X^{(k)}W^{(k)}\right)$$

where:

- $\tilde{A} = A + I$ is the adjacency matrix with self-loops
- $\tilde{D}$ is the degree matrix of $\tilde{A}$
- $X^{(k)}$ is the node feature matrix at layer k
- $W^{(k)}$ is a learnable weight matrix

GCN corresponds to choosing a very simple message function: a normalized linear aggregation.

A spectral perspective on graphs

So far, we defined GNNs in a **spatial way**, i.e. nodes aggregate information from neighbors.

An alternative viewpoint comes from **spectral graph theory**.

Let G be a graph with normalized Laplacian:

$$L = I - D^{-1/2} A D^{-1/2}$$

Since L is symmetric, it admits an eigendecomposition:

$$L = U \Lambda U^{\top}$$

Interpretation:

- U: graph Fourier basis
- Λ: frequencies

This allows us to define convolution on graphs in the spectral domain.

From spectral convolution to GCN

A spectral graph convolution can be defined as:

$$x \mapsto U\, g(\Lambda)\, U^\top x$$

where $g(\Lambda)$ is a filter in the spectral domain.

However:

- requires eigen-decomposition ($\mathcal{O}(n^3)$)
- not scalable/localized

Key idea: approximate $g(\Lambda)$ with a low-degree polynomial.

This leads to **localized filters** that depend only on neighbors.

From spectral convolution to GCN

A spectral graph convolution can be defined as:

$$x \mapsto U\, g(\Lambda)\, U^\top x$$

where $g(\Lambda)$ is a filter in the spectral domain.

However:

- requires eigen-decomposition ($\mathcal{O}(n^3)$)
- not scalable/localized

Key idea: approximate $g(\Lambda)$ with a low-degree polynomial.

This leads to **localized filters** that depend only on neighbors.

Result (Kipf & Welling)[5]:

$$X^{(k+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\,\tilde{D}^{-1/2}X^{(k)}W^{(k)}\right)$$

Conclusion: GCNs can be seen as a **first-order approximation** of spectral graph convolutions.

Spatial vs Spectral View

Spectral view:

- convolution = filtering in the graph Fourier domain
- based on the Laplacian spectrum

Spatial view:

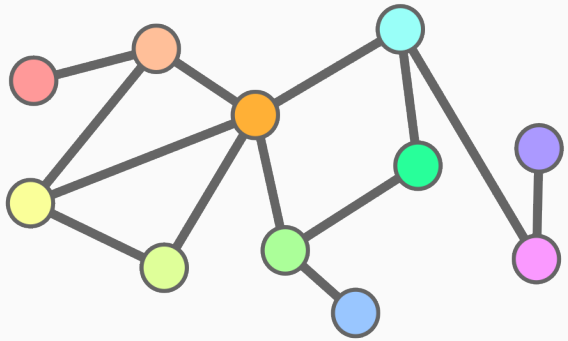
- convolution = aggregation from neighbors
- local, permutation-invariant operations

Key insight: many spatial GNNs can be interpreted as approximations of spectral filters.

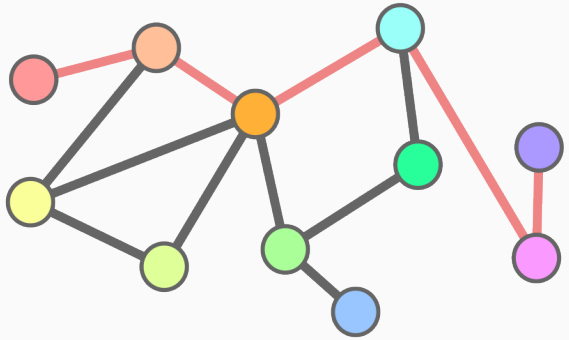
GNN Limitations. Over-smoothing: how many layers do we need?

How many message passing layers should we use

Input graph

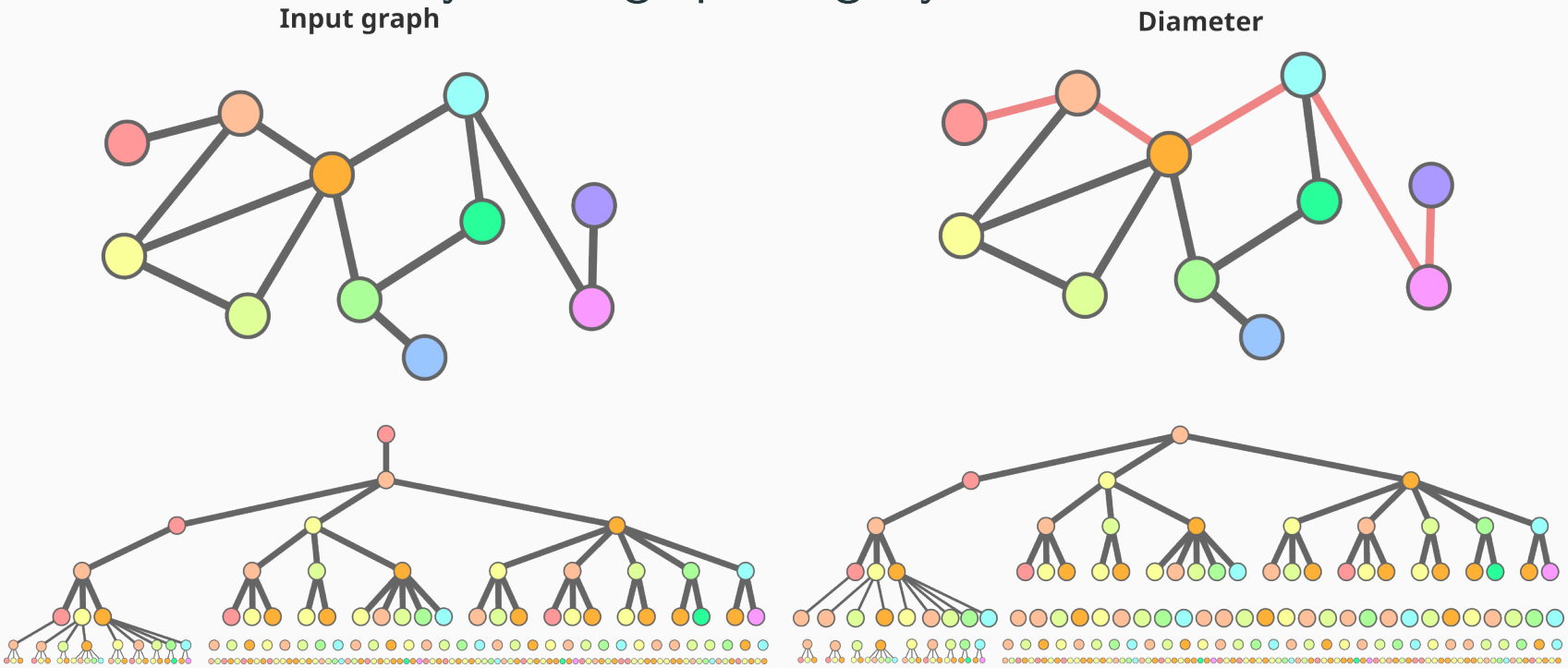


Diameter



GNN Limitations. Over-smoothing: how many layers do we need?

How many message passing layers should we use



Do we really want to propagate information across the entire graph?

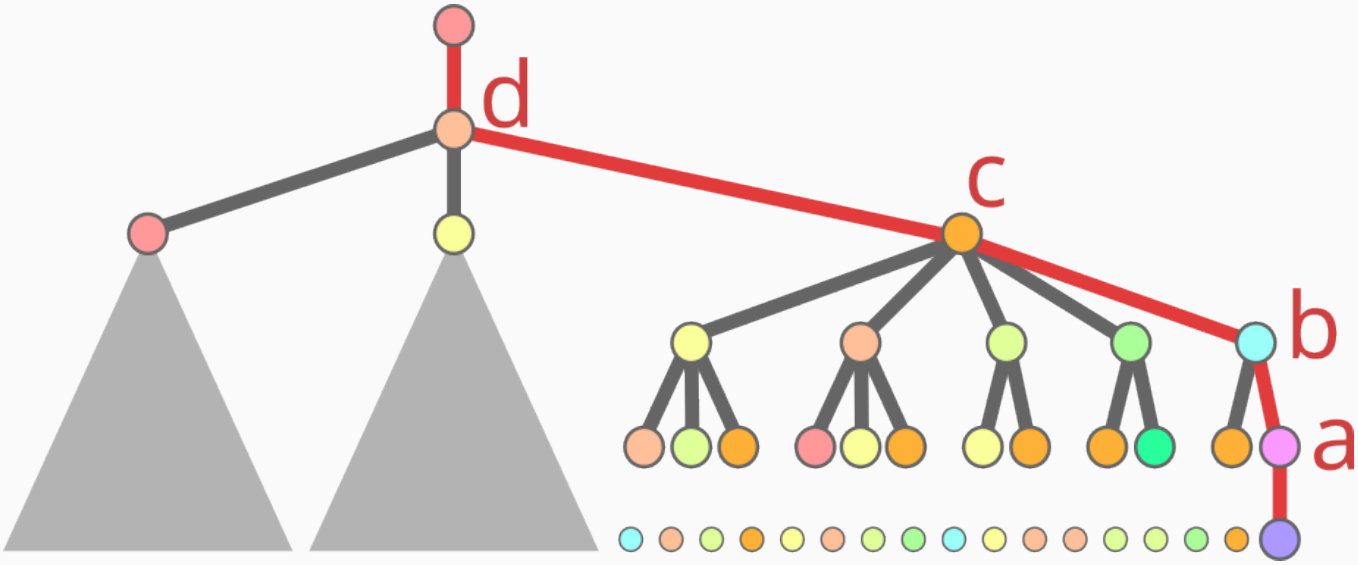
Over-smoothing: node representations become indistinguishable [11]

GNN Limitations. Over-squashing: long-range dependencies

can GNNs capture long-range interactions?

Example: consider a graph where the label of a node depends on a distant node. (information must travel across many intermediate nodes)

Over-squashing: long-range information is **compressed** and becomes hard to recover.[10]



GNN Limitations. Expressivity

Question: how expressive are GNNs?

Weisfeiler–Lehman (WL) test: an iterative procedure that updates node labels by aggregating the labels of neighboring nodes.

At each iteration:

$$\text{label}_i^{(k+1)} = \text{hash}\left(\text{label}_i^{(k)}, \{\text{label}_j^{(k)} : j \in \mathcal{N}(i)\}\right)$$

Key fact: many GNNs are at most as powerful as WL.

Implication: if WL cannot distinguish two graphs, neither can these GNNs.[7]

What the NGML group is doing?

Explainability in GNNs [6]

Problem: a GNN makes a prediction, but *why*?

We want to understand:

- which **nodes / edges** matter
- which **features** matter
- which **subgraph** supports the decision

Why important?

- trust
- debugging
- detecting spurious patterns

Main techniques:

- Subgraph explanations
- Counterfactuals

Uncertainty in NNs [9]

Question: how confident is the prediction?

Why?

- detect unreliable predictions
- decision making under uncertainty
- robustness to distribution shifts

Techniques:

conformal predition ensembles stochastic regularization

Roberto, Andrea, Filippo

Pooling in GNNs [1]

Problem: how do we obtain a **global representation** of a graph?

Goal:

- map variable-size graphs $\rightarrow$ fixed-size representation
- preserve relevant **structure**

Why is it non-trivial?

- must be **permutation invariant**
- must avoid losing important information

Main approaches:

global pooling hierarchical pooling learned clustering

Spatio-temporal Graphs [4]

Problem: data evolve over both **structure** and **time**

Setting:

$$G(t) = (V(t), E(t), X(t))$$

- nodes / edges may change over time
- features evolve dynamically

Goal:

- model **temporal dependencies**
- capture interaction between **space (graph)** and **time**

Main approaches:

temporal message passing recurrent models attention over time

Expressivity: beyond WL [8]

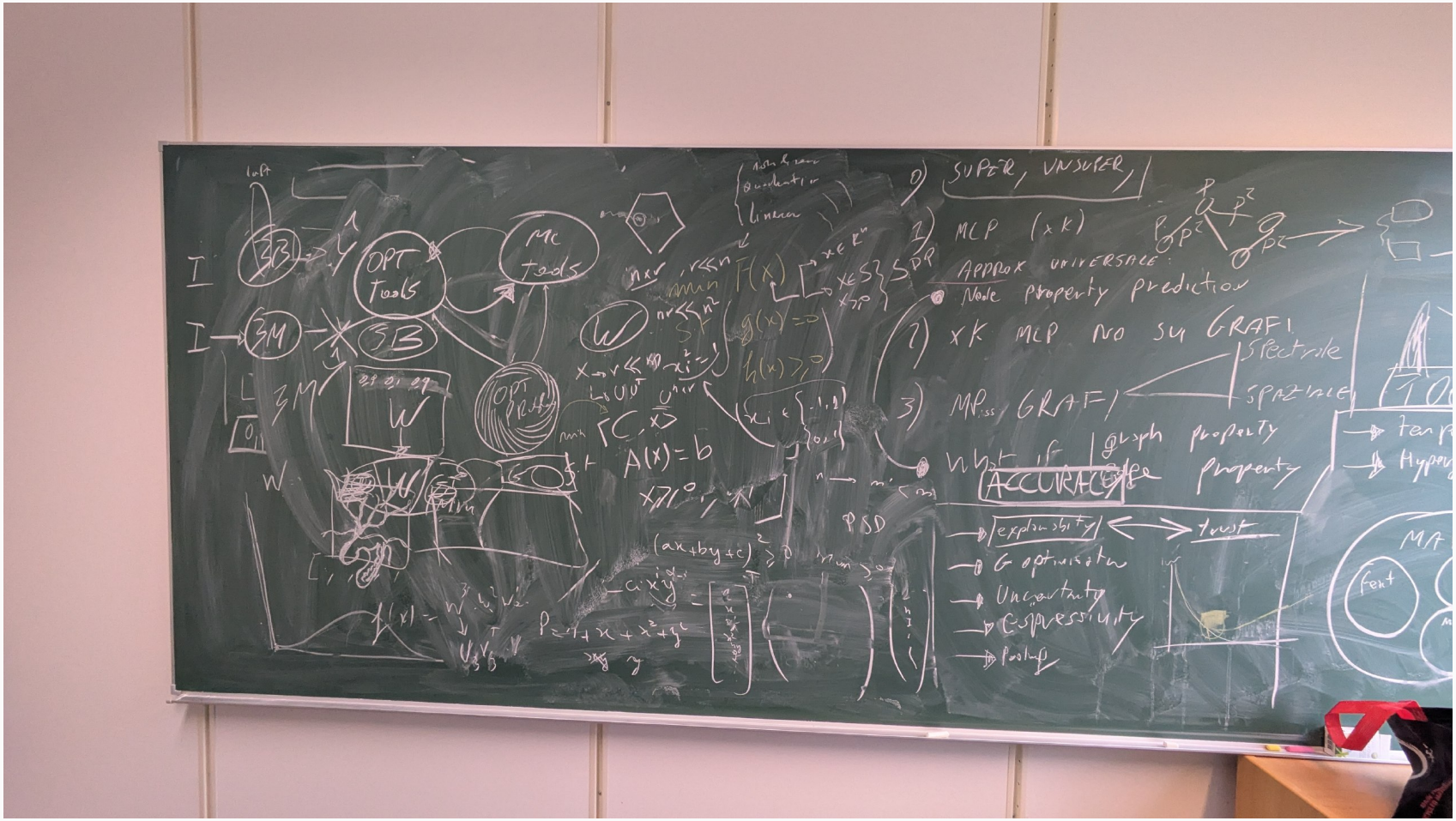
Limitation: standard GNNs $\leq$ Weisfeiler–Lehman (WL)

Research question: how can we go **beyond local aggregation**?

Main directions:

- **Higher-order methods**
  - k-WL, subgraph-based GNNs
- **Richer structures**
  - simplicial complexes, hypergraphs, hyperbolic embeddings
- **Global information**
  - positional encodings, spectral features
- **Breaking symmetry**
  - identifiers, random features

10 Minutes Discussion



Yassine Koubaa

Thank you!

Questions or discussion?

References

[1] Filippo Maria Bianchi and Veronica Lachi. **"The expressive power of pooling in Graph Neural Networks".** In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=zqyVjCjhYD.

[2] George Cybenko. **"Approximation by superpositions of a sigmoidal function".** In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

References ii

[3] Justin Gilmer et al. **"Neural message passing for quantum chemistry".** In: *International conference on machine learning*. Pmlr. 2017, pp. 1263–1272.

[4] Michele Guerra, Simone Scardapane, and Filippo Maria Bianchi. **"Interpreting Temporal Graph Neural Networks with Koopman Theory".** In: *arXiv preprint arXiv:2410.13469* (2024).

[5] Thomas N. Kipf and Max Welling. **"Semi-Supervised Classification with Graph Convolutional Networks".** In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

References iii

[6] Antonio Longa et al. **"Explaining the explainers in graph neural networks: a comparative study".** In: *ACM Computing Surveys* 57.5 (2025), pp. 1–37.

[7] Christopher Morris et al. **"Weisfeiler and leman go neural: Higher-order graph neural networks".** In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4602–4609.

[8] Dionisia Naddeo et al. **"Hyperbolic Graph Neural Networks Under the Microscope: The Role of Geometry-Task Alignment".** In: *arXiv preprint arXiv:2602.01828* (2026).

References iv

[9] Roberto Neglia et al. **"ResCP: Reservoir Conformal Prediction for Time Series Forecasting".** In: *arXiv preprint arXiv:2510.05060* (2025).

[10] Kenta Oono and Taiji Suzuki. **"Graph Neural Networks Exponentially Lose Expressive Power for Node Classification".** In: *International Conference on Learning Representations*.

[11] Jake Topping et al. **"Understanding over-squashing and bottlenecks on graphs via curvature".** In: *International Conference on Learning Representations*.