Programming & Data Analytics & AI 1 — 2025/2026

Sant'Anna School of Advanced Studies, Pisa, Italy

Sant'Anna School of Advanced Studies – Pisa
Institute of Economics

Department of Excellence 2018 - 2022
EMbeDS
Economics and Management in the era of Data Science

Department of Excellence 2023 - 2027
LEMbeDS
Economics, Management and Law in the era of Data Science

DTU Danmarks Tekniske Universitet

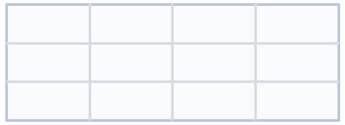
Module 2, Part 3 — Research-driven topics

From Neural Networks to Graph Neural Networks

Neural Networks • Convolutions • Transfer learning • Graph Convolution • Attention & GAT

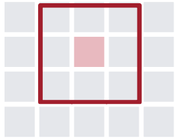
Tabular data

MLP / dense layers



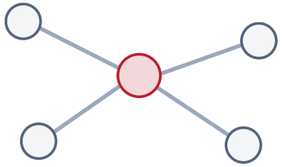
Images / grids

CNN / convolution



Graphs

GNN / message passing



Course responsible: Andrea Vandin • Lecturer: Riccardo Porcedda

AGENDA

Roadmap

Goal: intuition + practical pipeline (minimal theory)

- Neural Networks basics: forward/backpropagation, gradient descent and update rule
- Convolution: from simple filter to Convolutional Neural Networks (CNNs)
- Extending convolutions to graphs
- Attention mechanism: the core of LLMs and an application to graphs with Graph Attention Networks (GATs)

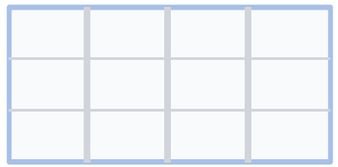
STRUCTURE MATTERS

Why different architectures?

Dense

Tabular → MLP

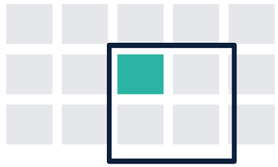
All features can interact
(no notion of locality)



Convolution

Images → CNN

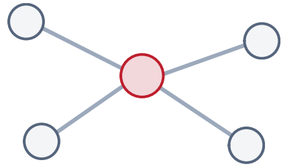
Local neighborhoods on a grid
translation equivariance



Message passing

Graphs → GNN

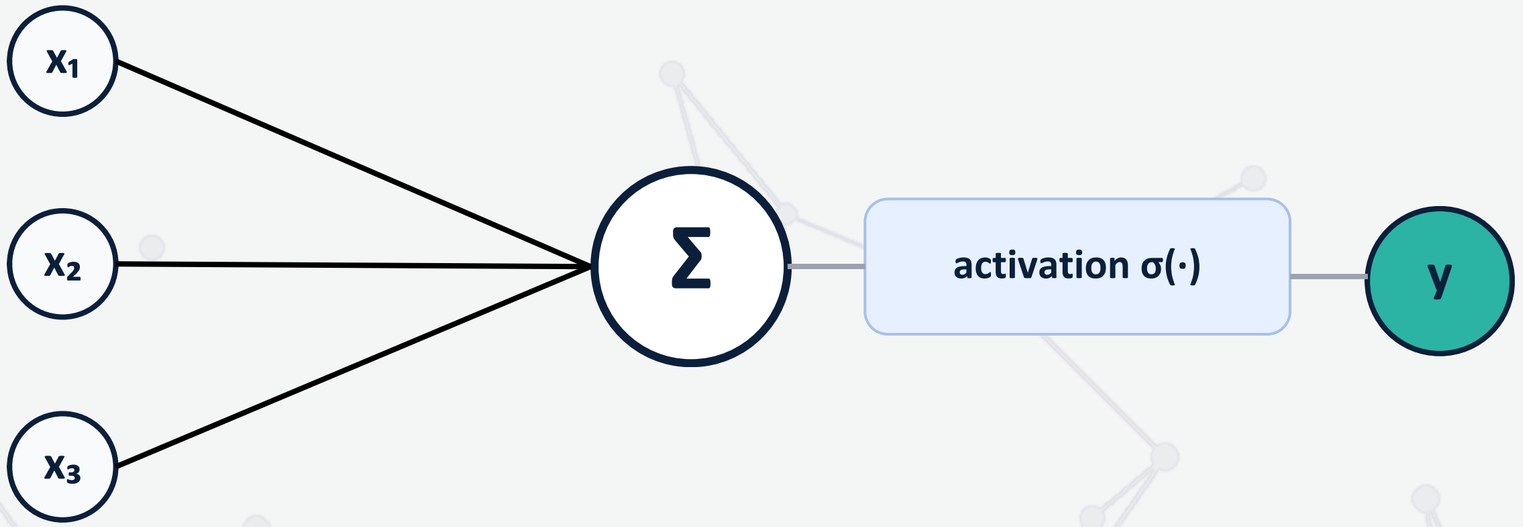
Local neighborhoods from
edges
permutation invariance



Neural Networks (MLP)

A neuron = linear combination + bias + activation

$$y = \sigma(w^T x + b)$$

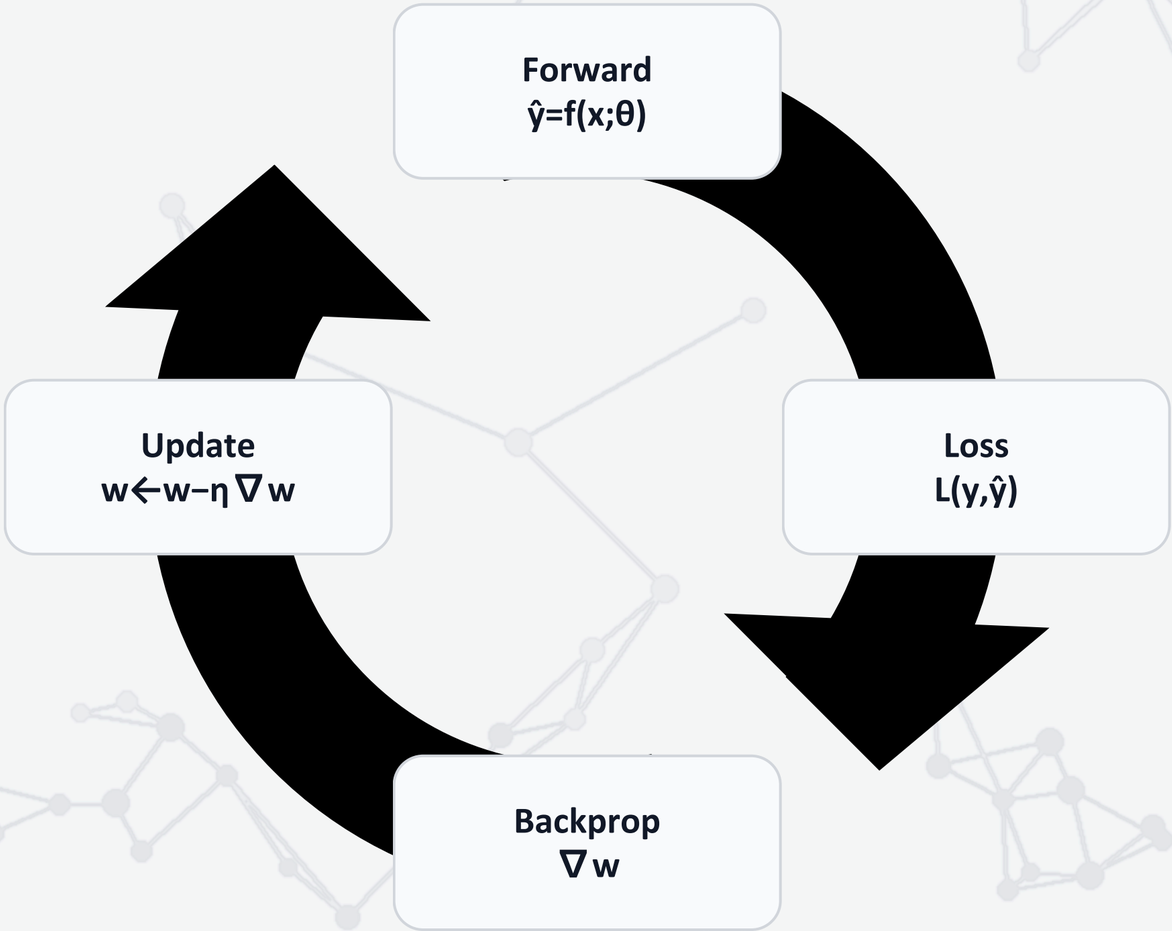


A Multi Layer Perceptron (MLP) stacks many layers:

$$h^{(1)} = \sigma(w_1^T x + b_1), \quad h^{(2)} = \sigma(w_2^T h^{(1)} + b_2), \quad \dots$$

Training loop

A training epoch consists of this loop:



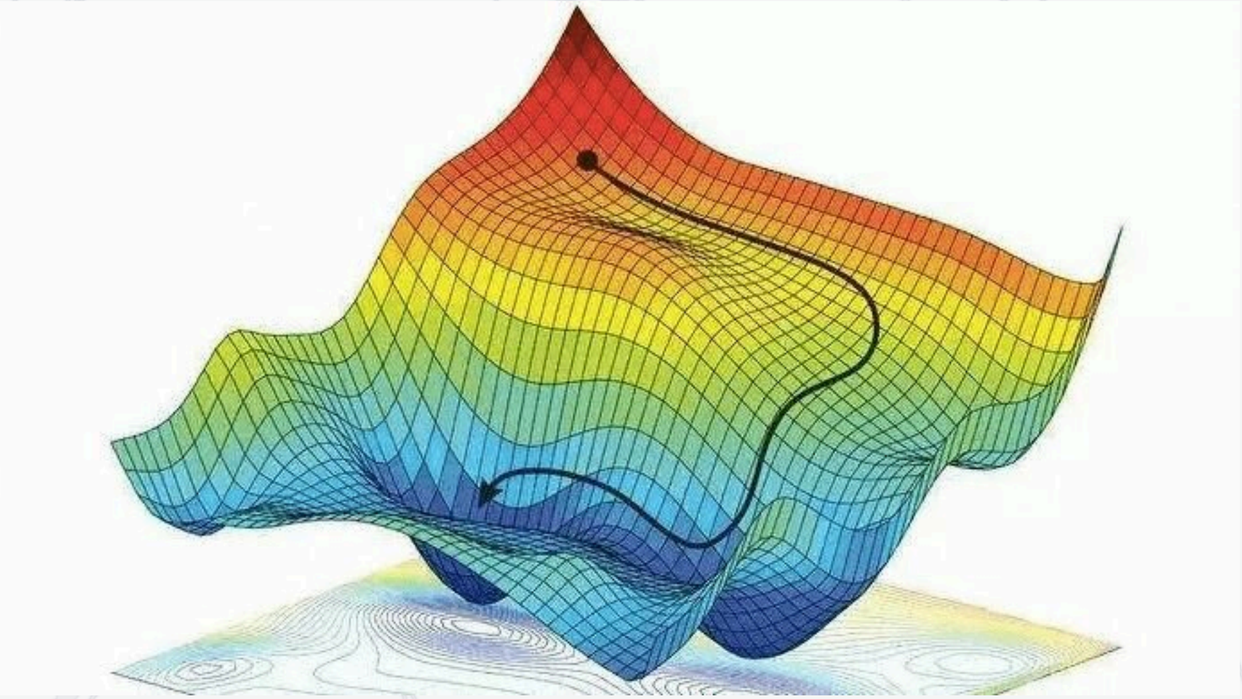
Gradient Descent

This landscape represents how the loss function changes by varying the parameters.

We want to find the minimum (the deepest well): how can we do it?

We use the gradient (i.e. the vector of partial derivatives) to obtain the right directions and we update the parameters accordingly:

$$w \leftarrow w - \eta \cdot \nabla L(w)$$



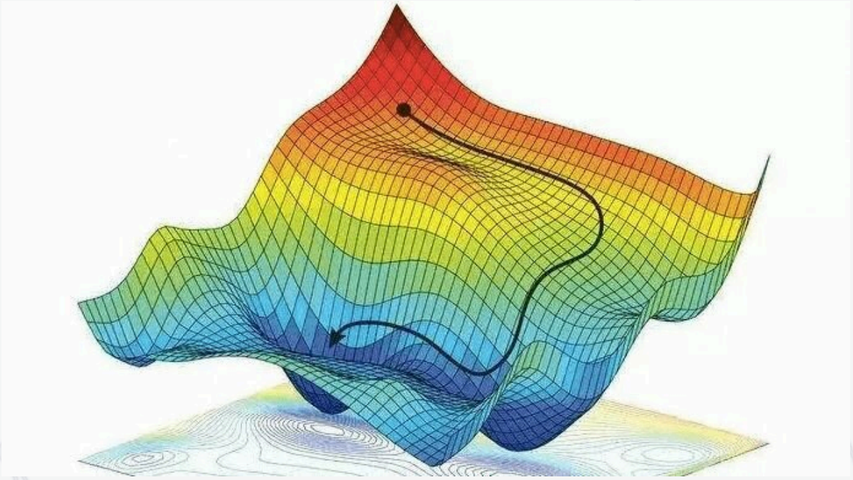
Stochastic Gradient Descent

In practice, computing the gradient on the entire training set at every step can be very expensive.

Instead, we approximate the true gradient using only a small random subset of data, called a mini-batch.

This makes training:

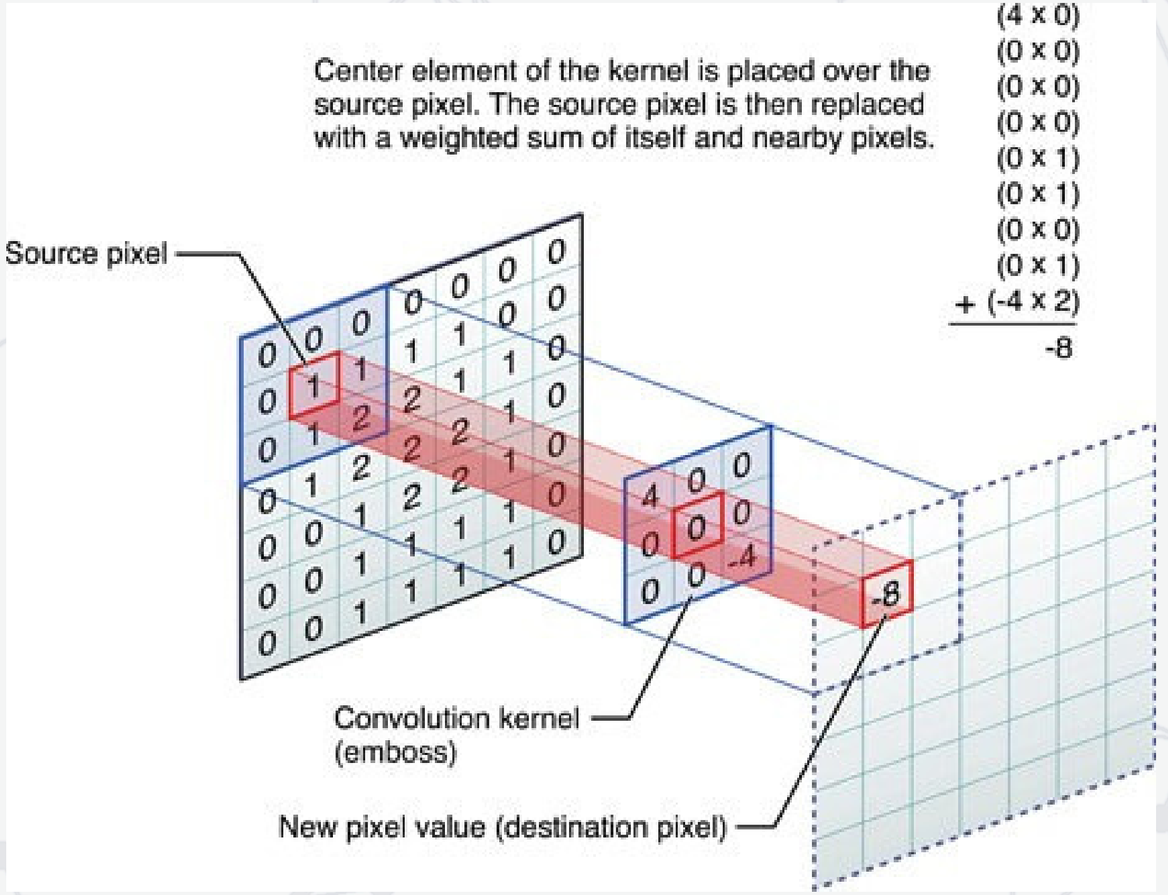
- **faster**
- **more memory efficient**
- **able to scale to large datasets**



Convolutions

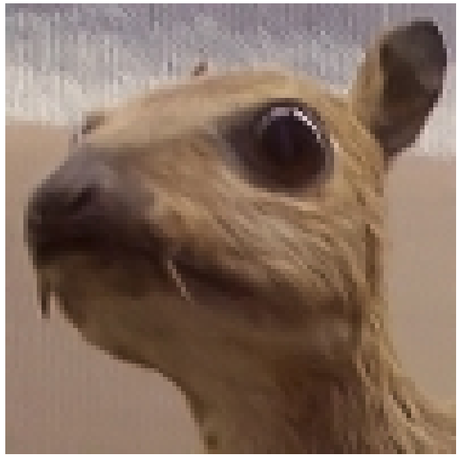
Convolution = local receptive field + weight sharing

- Convolution looks at a small neighborhood of pixels at a time.
- A kernel slides over the image and computes a weighted combination of local pixel values.
- Reusing the same kernel everywhere reduces the number of parameters (weight sharing).
- This makes CNNs efficient and especially good at detecting local visual patterns.



Example of simple convolutions

Input image



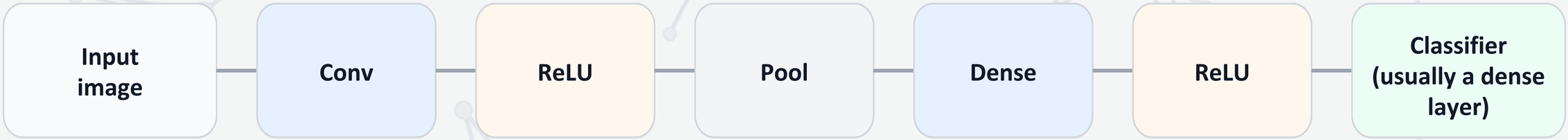
Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

CNN architecture

A typical CNN architecture



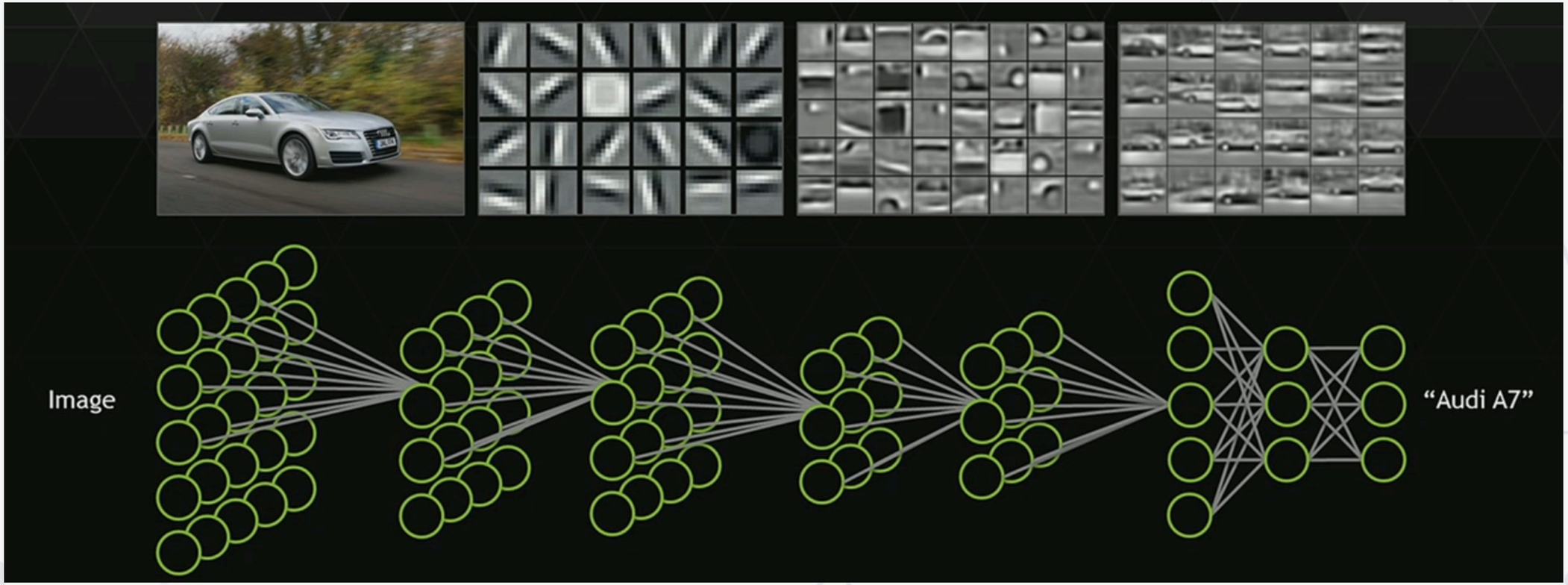
The idea is that:

- Early layers detect edges/textures
- Deeper layers combine patterns into objects
- The final layers classifies the objects

CNN architecture

The idea is that:

- Early layers detect edges/textures
- Deeper layers combine patterns into objects
- The final layers classifies the objects

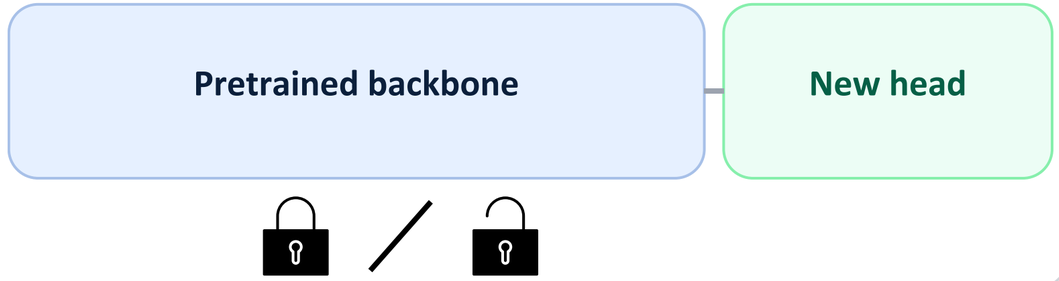


What if we don’t have enough resources (hardware for training or data)?

Reuse what another model has already learned!

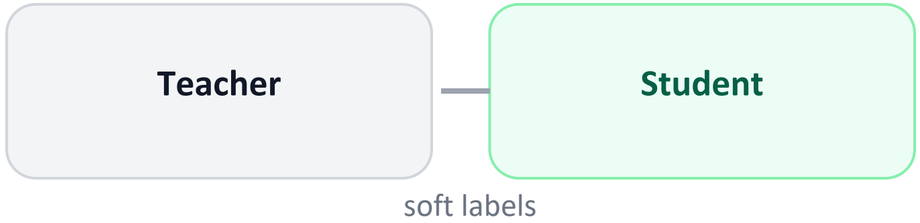
Transfer learning

- Freeze a pretrained backbone + train a new head
- Or fine-tune with a smaller learning rate
- Helps when data is limited



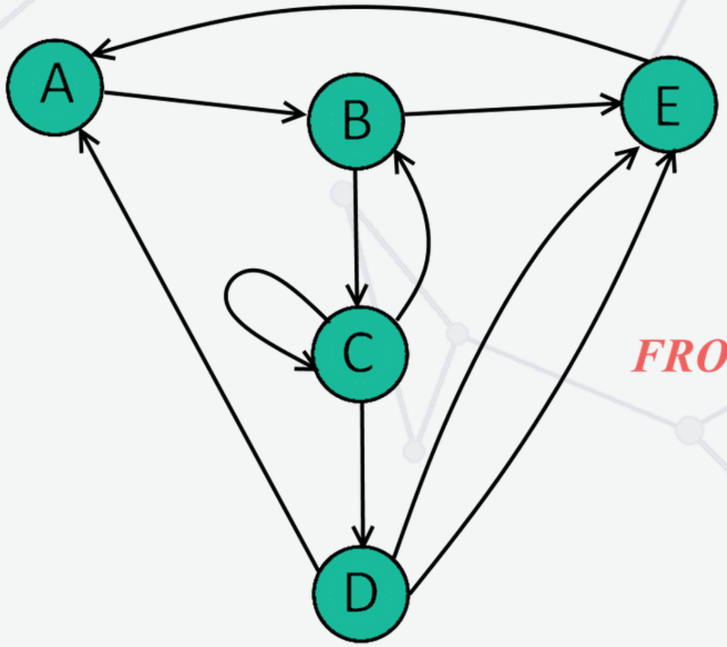
Knowledge distillation

- Train a compact student to match a strong teacher
- Use soft targets (probabilities)
- Useful for compression & deployment



Graphs

A graph = nodes + edges (+ features)



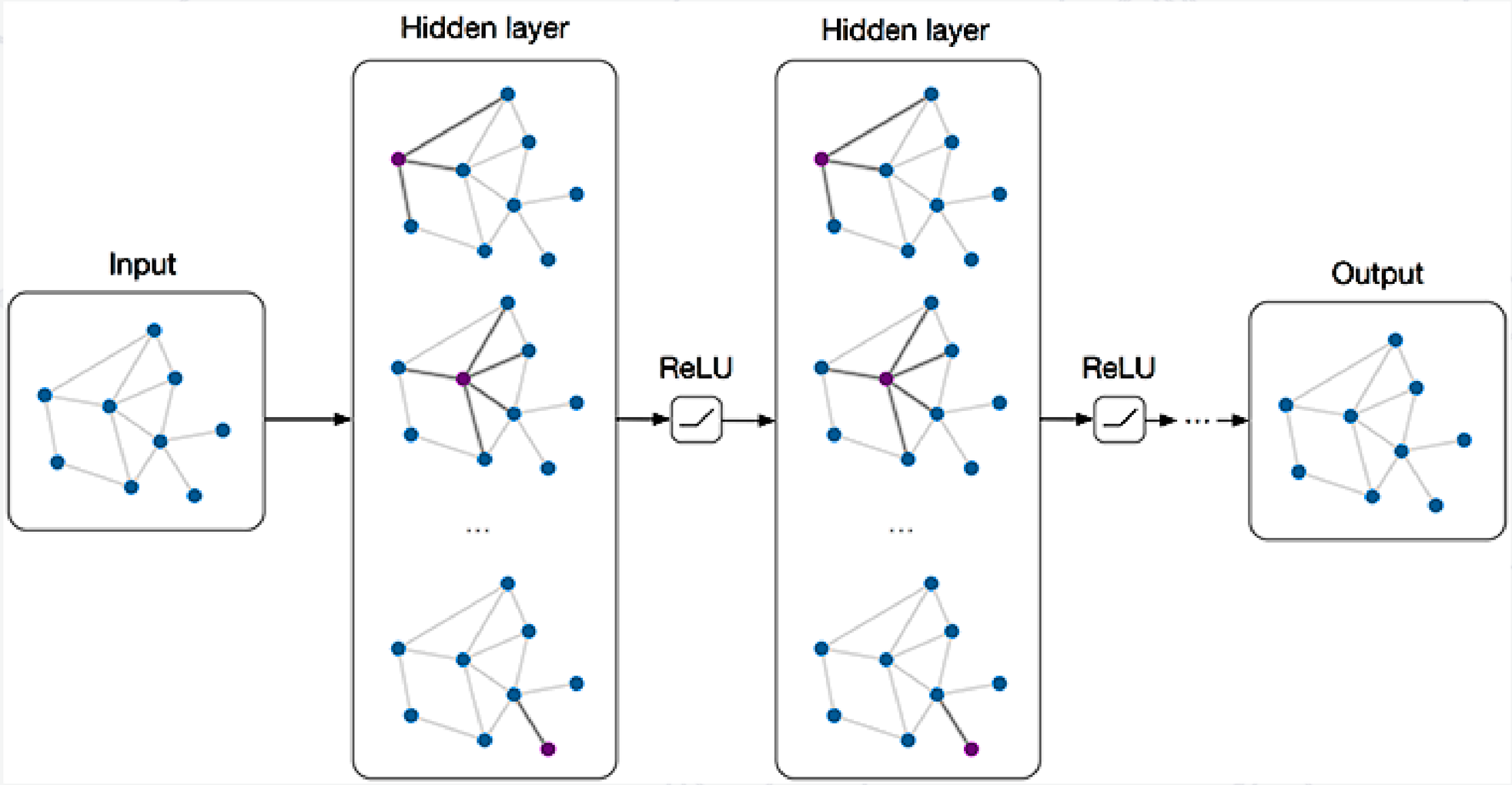
$$\begin{array}{c} \\ \\ \text{FROM} \end{array} \overbrace{\begin{array}{c} \\ \\ \end{array}}^{\text{TO}} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{l} A \\ B \\ C \\ D \\ E \end{array}$$

Common tasks with graphs

- Node classification (label nodes)
- Link prediction (missing edges)
- Graph classification (label the whole graph)

Bridge: CNN → GNN

Let’s apply convolutions to graphs

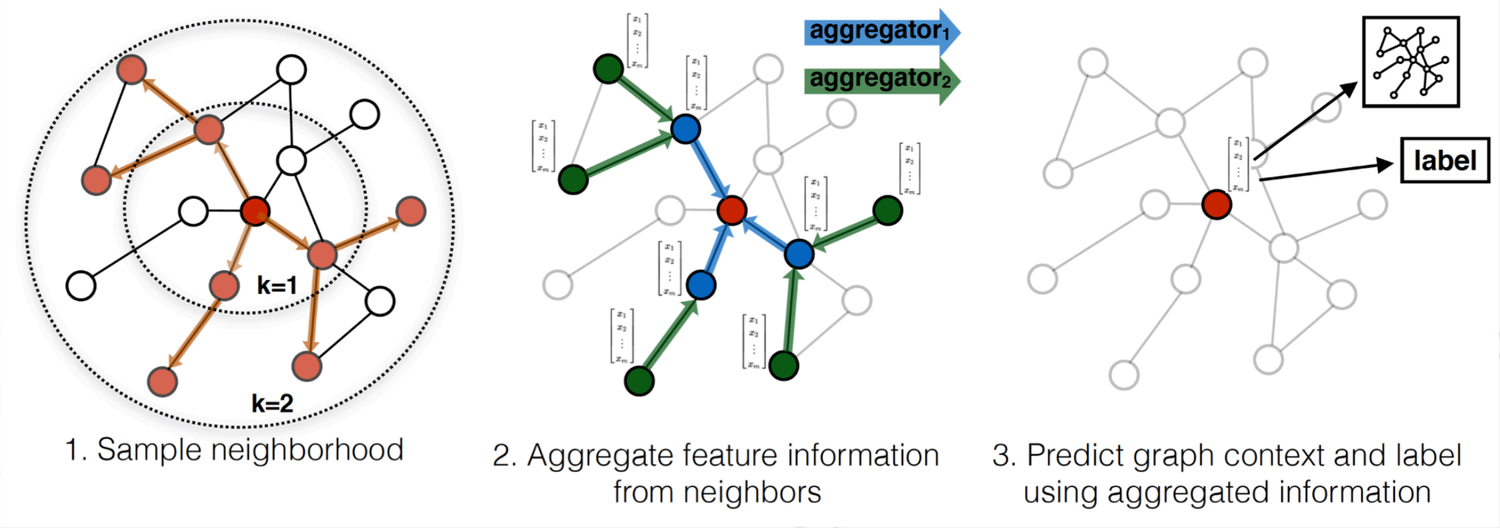


Message-Passing and Graph Convolution (GCN)

Each node updates its embedding by aggregating messages from neighbors.

Generic MP: $$h_i^{(l+1)} = \phi^{(l)}\left(h_i^{(l)}, \bigoplus_{j \in \mathcal{N}(i)} \psi^{(l)}\left(h_i^{(l)}, h_j^{(l)}, e_{ij}\right)\right)$$

GCN: $$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right)$$



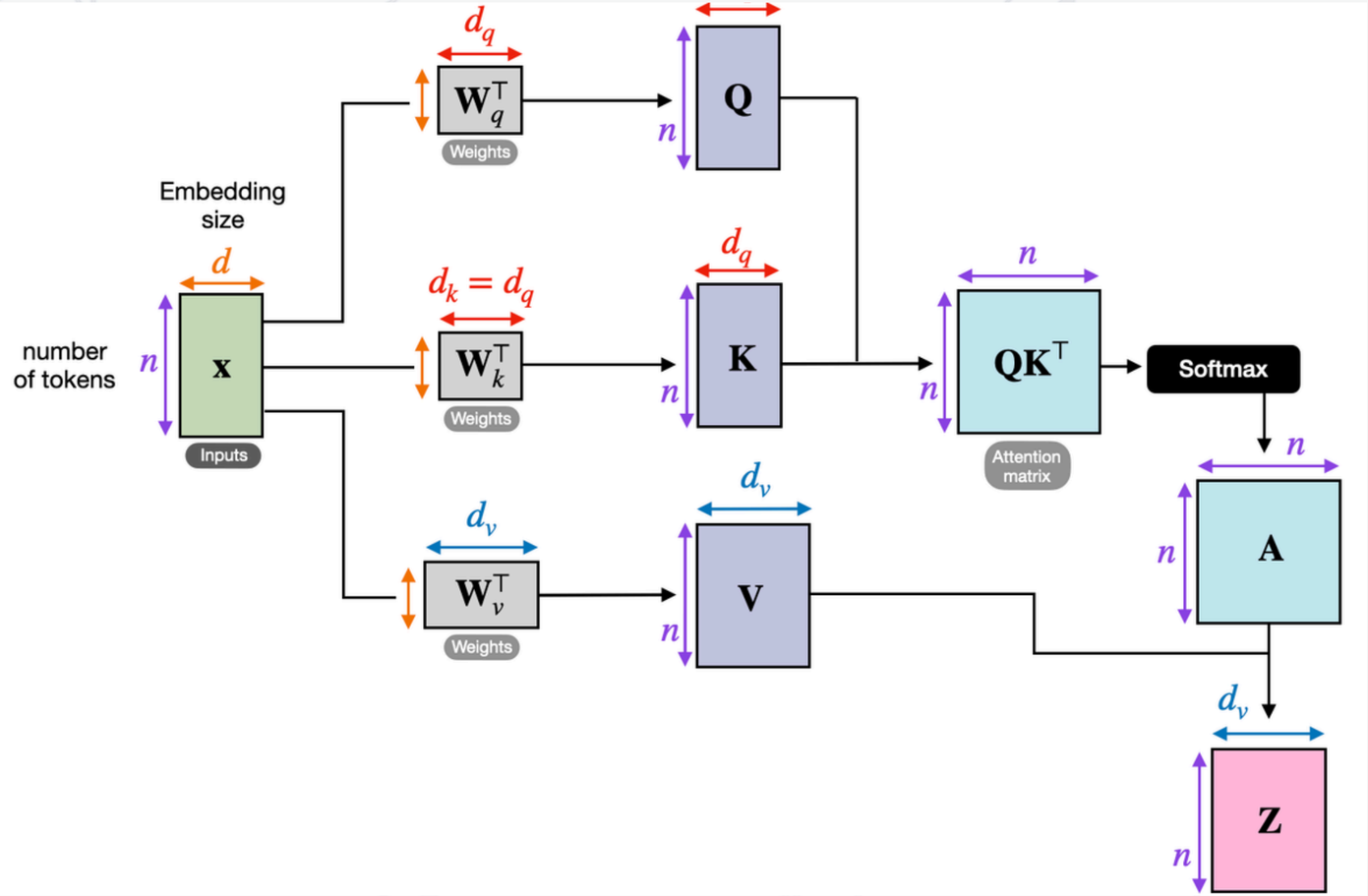
1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

Attention mechanism

Not all information is equally relevant: learn data-dependent weights.



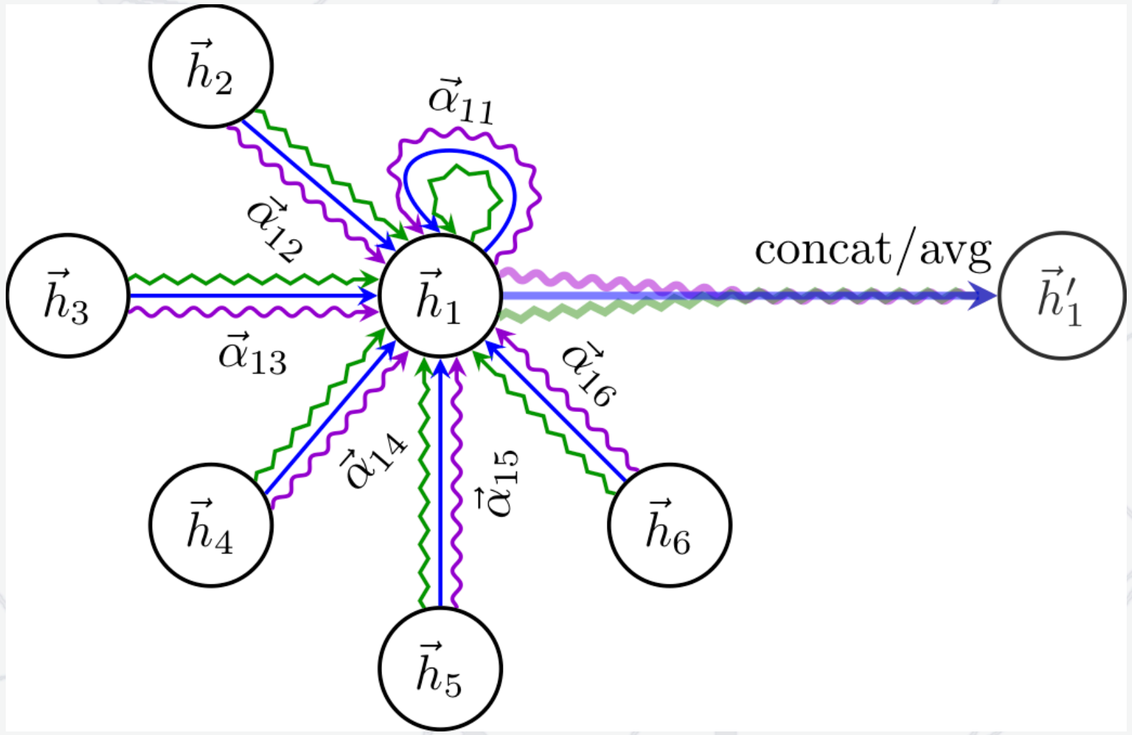
- Inputs X → linear projections: Q, K, V
- Scores: QK^T (similarity between queries and keys)
- Softmax → attention matrix A (weights)
- Output: Z = A · V (weighted combination of values)

Intuition

Query “asks” what it needs.
Keys say who is relevant.
Values are what we aggregate.

Graph Attention Networks (GAT)

GAT = message passing where neighbor contributions are learned via attention.



$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)\cup\{i\}} \alpha_{ij}^{(l)}\, W^{(l)} h_j^{(l)}\right)$$

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\mathrm{LeakyReLU}\left(a^{(l)\top}\left[\, W^{(l)} h_i^{(l)} \,\|\, W^{(l)} h_j^{(l)} \,\right]\right)\right)}{\sum_{k \in \mathcal{N}(i)\cup\{i\}} \exp\left(\mathrm{LeakyReLU}\left(a^{(l)\top}\left[\, W^{(l)} h_i^{(l)} \,\|\, W^{(l)} h_k^{(l)} \,\right]\right)\right)}$$

Why it helps

- Adaptive neighbor weighting
- Multi-head attention improves stability
- Good when neighbor importance is heterogeneous

RECAP

Wrap-up

Takeaways

MLP

Tabular / dense
No locality by default

CNN

Grid locality
Shared filters

GCN

Graph locality
Normalized aggregation

GAT

Graph locality
Learned attention weights

Demo idea (from the notebook)

- MLP on MNIST
- CNN on MNIST
- Node classification on Cora
- Compare MLP vs GCN vs GAT (accuracy + training time)

Useful links

3Blue1Brown — Neural networks

3Blue1Brown — Gradient descent

3Blue1Brown — Backprop

3Blue1Brown — Convolution

3Blue1Brown — Attention

Distill — GNN intro

Distill — Understanding GNNs