POLITECNICO MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

Graph Machine Learning

Leonardo De Grandis
leonardo.degrandis@polimi.it

Advanced Computer Architectures Course
T.1.1 - 12/5/2024

Wait, who are you?

BSc in Mechanical Engineering@Polimi

MSc in Automation and Control Engineering@Polimi

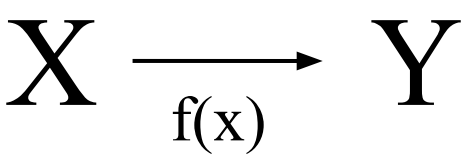
PhD in Information Technology@Polimi (1st year)

Interested in:

- Graph Analytics & Graph ML
- High Performance Computing
- Bioinformatics & Health Informatics
- Motorsport & Racing

What's Machine Learning?

Develop mathematical tools capable of **modeling the relationships** between **input** and **output** variables

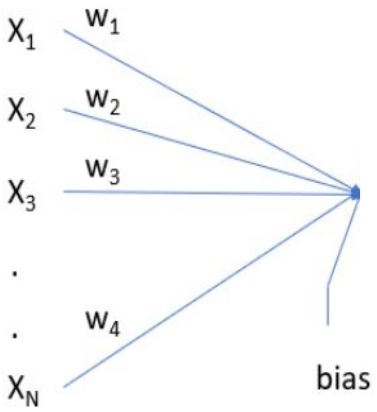


We **learn** f(x) from data

Neural Networks

A **neuron** combines input to produce an output signal:

- **weighted aggregation** of the input features



Neural Networks

- We can compose multiple **multiple neurons** to form a **layer**
- We can stack multiple layers to form a **deep architecture**, which usually is **more expressive**

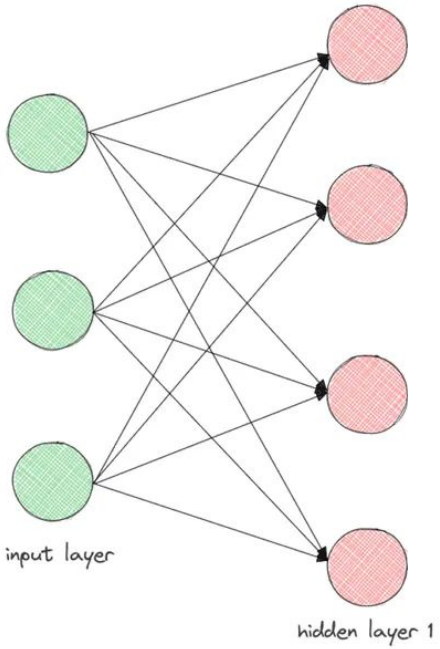
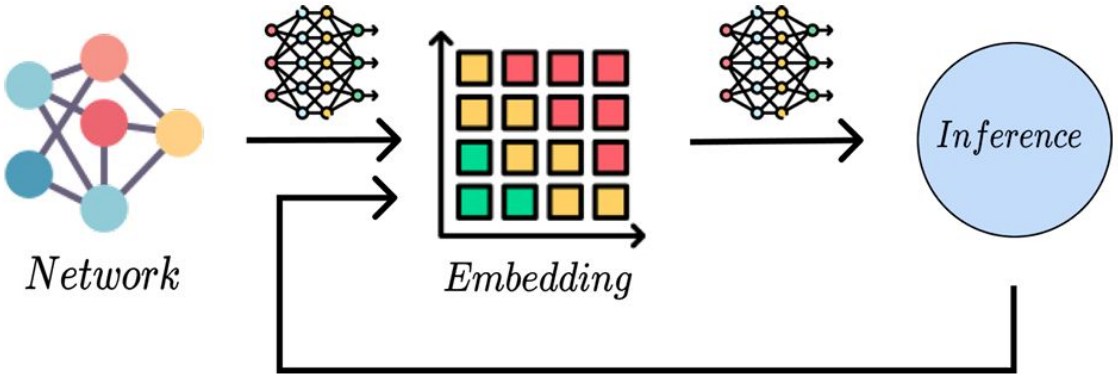


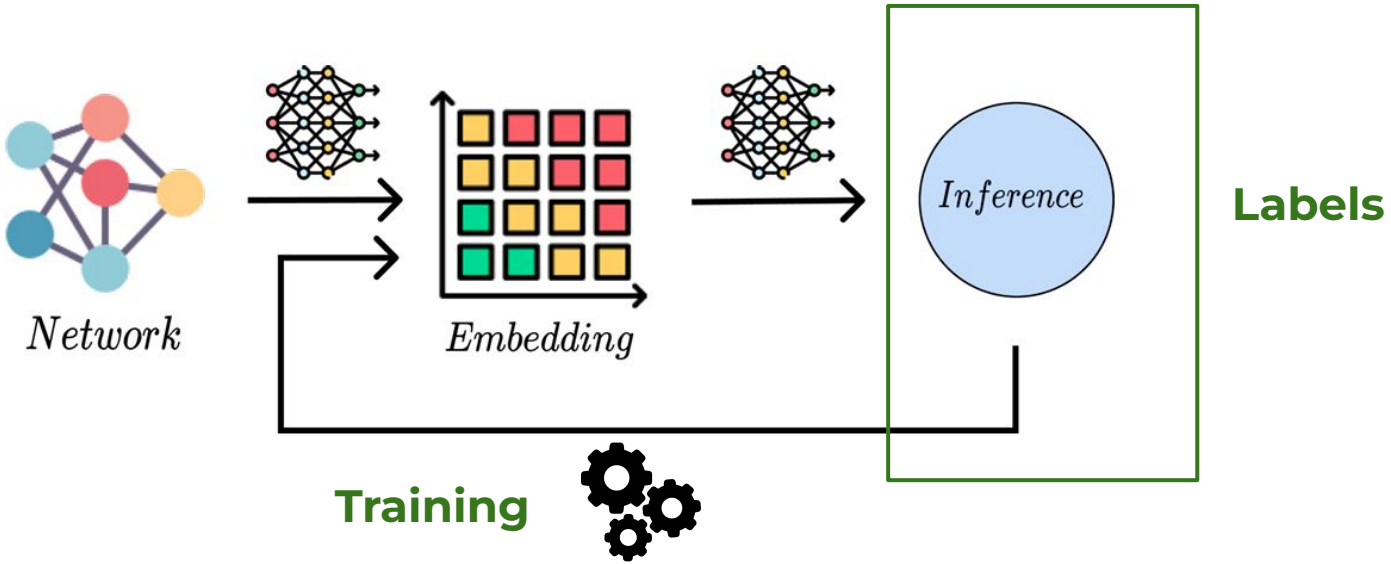
Image from https://www.baeldung.com/cs/hidden-layers-neural-network

The Learning Problem



Outputs of intermediate layers are the **embeddings**. They are **representations** how the model is elaborating the inputs and building its knowledge.

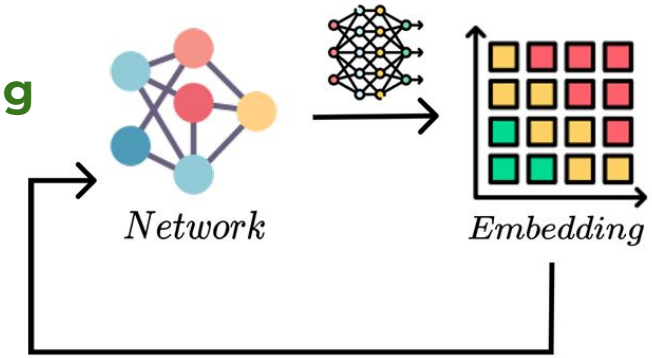
Supervised Training



We iterate the optimization procedure by predicting the output for a batch of samples and adjusting the weights to minimize the error with respect to the **labels**

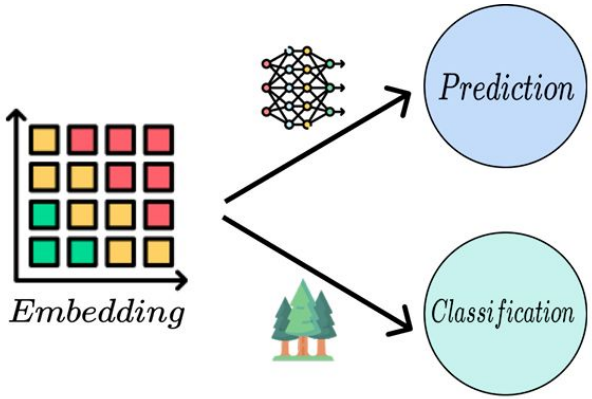
Unsupervised Pre-Training

Pre-Training



A pre-training phase is done **without the need of labels**. We leverage the **structure of the data itself** as a target for the optimization.

Transfer Learning - Fine Tuning



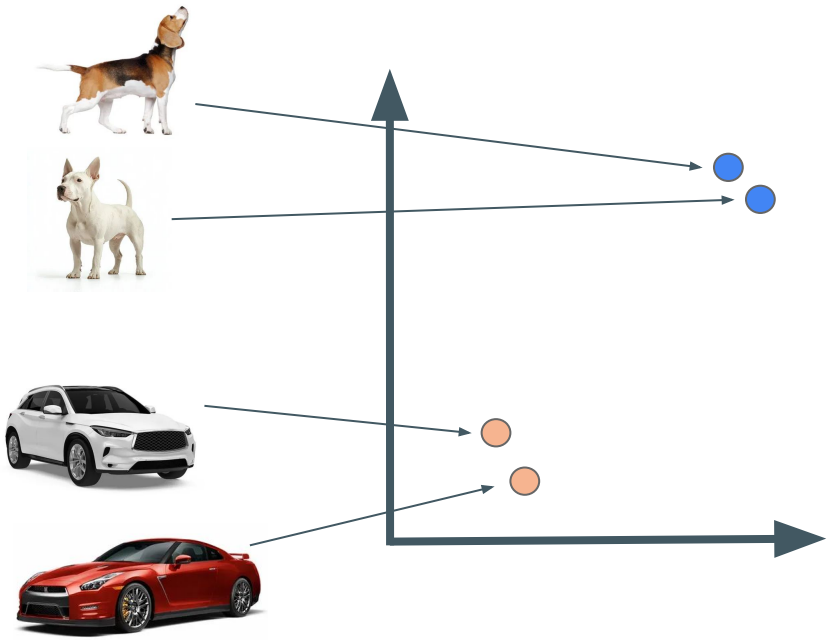
Few Lables
&&
Multi task

We leverage the knowledge learnt during the pre training as input to simpler classifiers / regressors. We can **learn from huge, wider and unlabeled data sources** and transfer its backbone to specific domain

About the Embeddings

Each sample is mapped into an high dimensionality **latent space:**

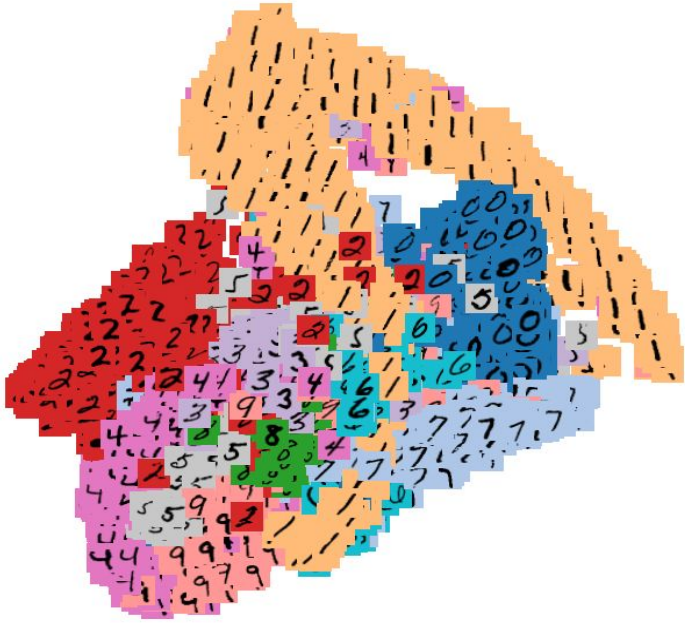
- Smoothness ($x \approx y \rightarrow f(x) \approx f(y)$)
- Multiple explanatory factors
- Hierarchical organization of factors (abstraction)
- Clustering
- Manifolds
- Temporal and spatial coherence
- Sparsity



Bengio, Yoshua, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013): 1798-1828.

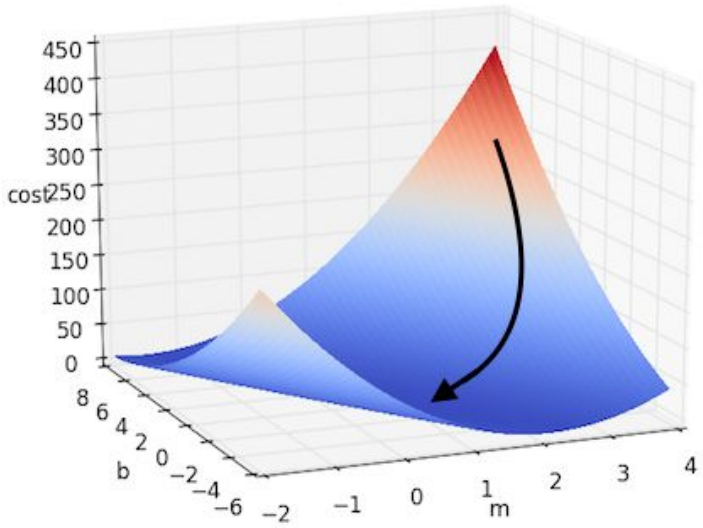
Embeddings, Not Only for Graphs

Manifold hypothesis: high dimensional data can be mapped in lower dimensionality manifold and the model tries to learn this geometry



How do we learn?

All the weights and the biases are learnt through an optimization process to **minimize a loss function** (e.g. minimize the prediction error)



The most effective method we have right now is to perform a **gradient-based optimization** (through automatic differentiation)

Image from https://ml4a.github.io/ml4a/how_neural_networks_are_trained/

Gradient Descent Method

1. We do a **forward pass** of the input to **get the prediction** of the model

2. The model tracks the **differentiable operations** performed by the model and computes the gradients of each parameter in the **backward pass with the chain rule**

$$\text{Repeat Until Convergence } \{$$
$$\quad \text{for i} = 1...\text{m } \{$$
$$\quad\quad \omega \leftarrow \omega - \alpha * \nabla_w L_m(w)$$
$$\quad \}$$
$$\}$$

w	learnable parameters
α	learning rate
∇L	gradient of parameters wrt loss

Gradient Descent Method

We do not know the shape of the loss function: potentially there are a lot of **local minima** with suboptimal performance.

In practice, **variants of SGD**, like AdamW, RMSprop..., are incredibly **robust** and work well in most of the cases.

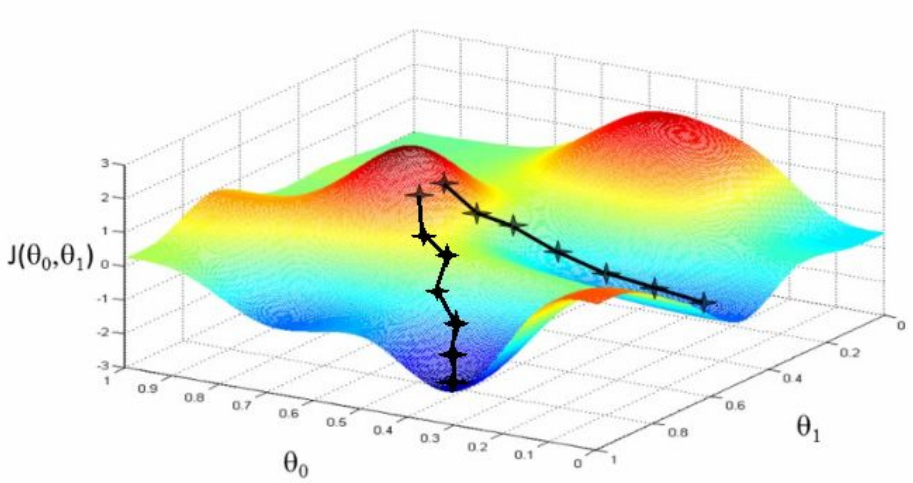
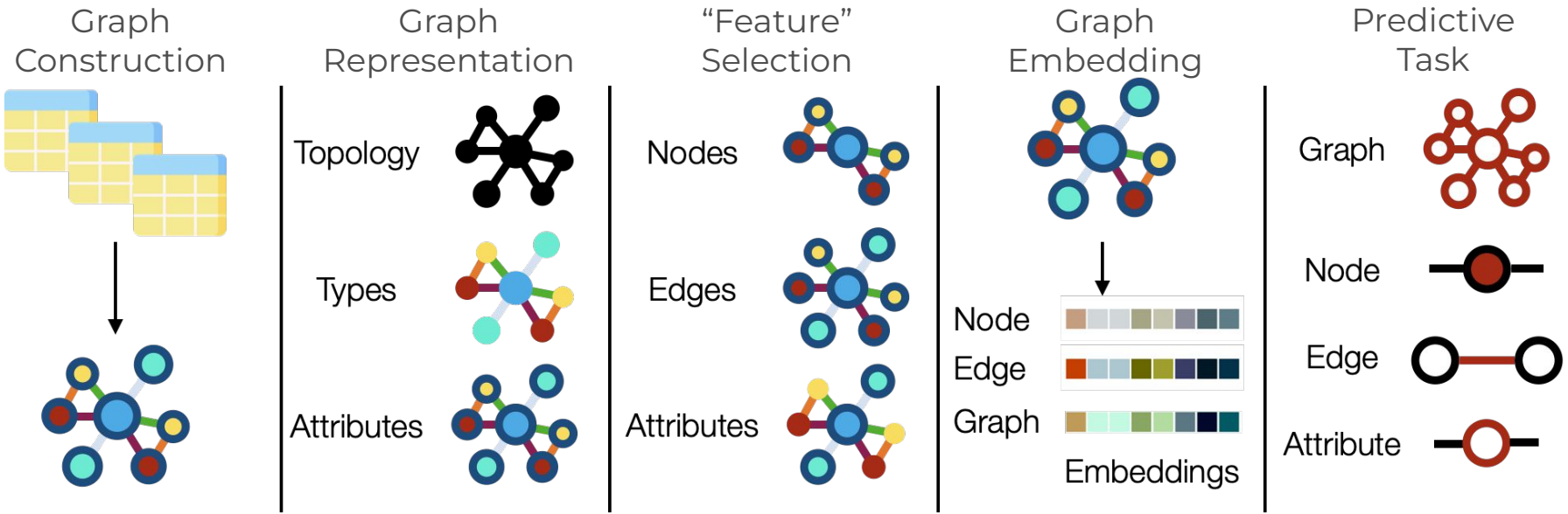
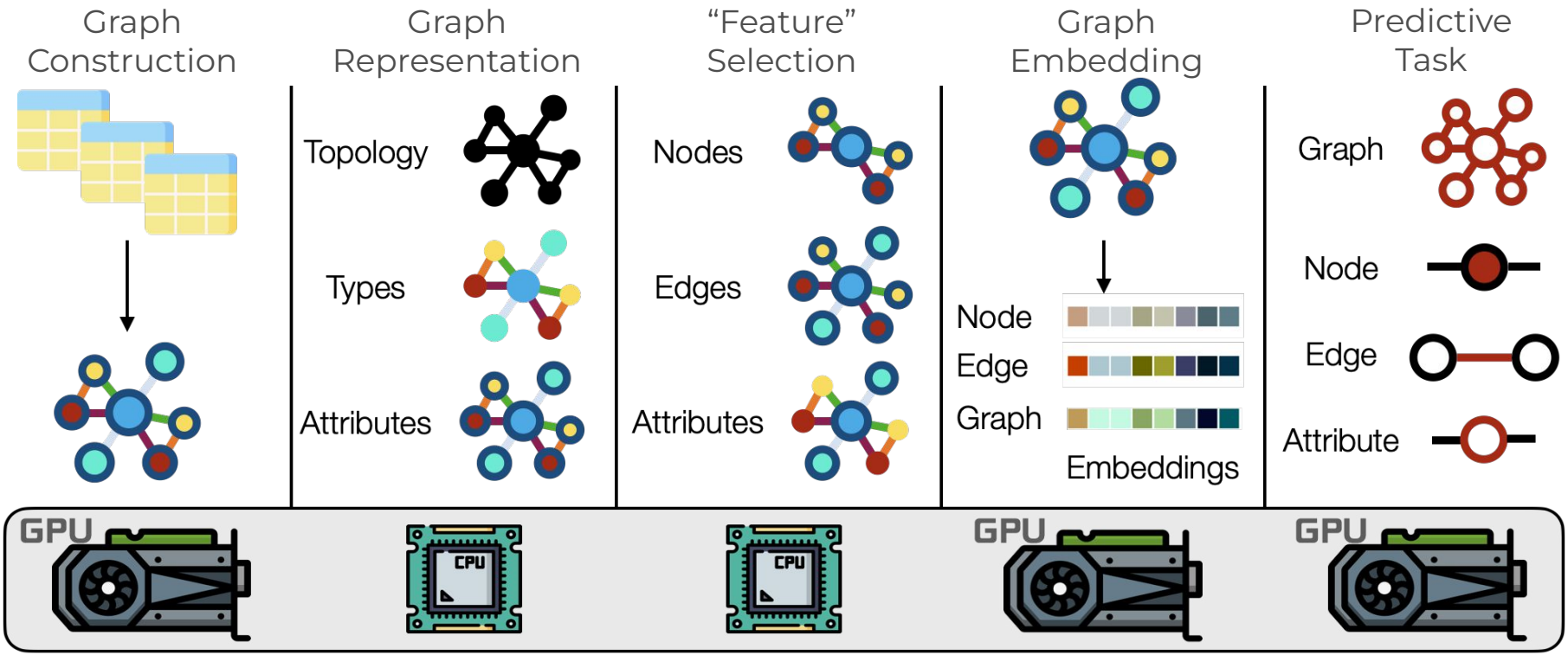


Image from https://ml4a.github.io/ml4a/how_neural_networks_are_trained/, Andrew Ng

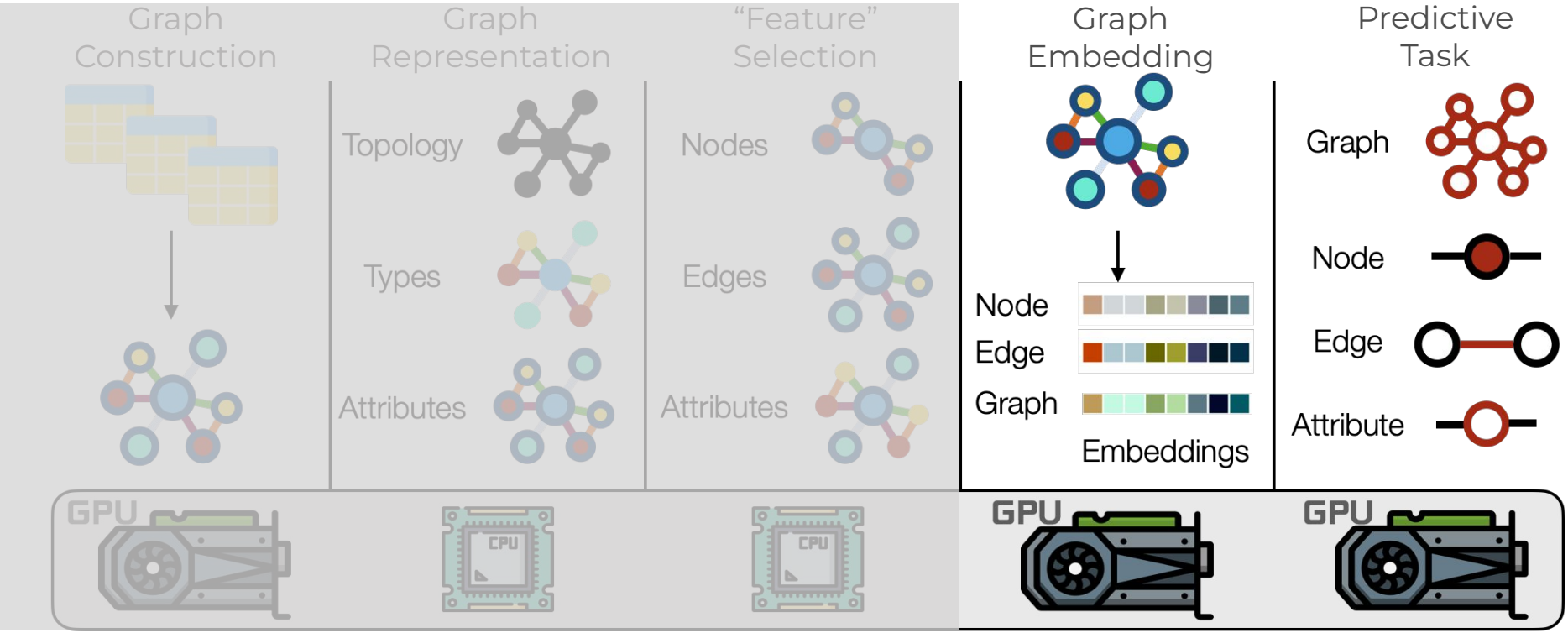
Generic Graph ML Pipeline



Generic Graph ML Pipeline



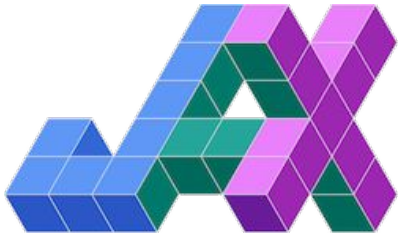
Generic Graph ML Pipeline



Main Frameworks

PyTorch
https://github.com/pytorch/pytorch

Tensorflow
https://github.com/tensorflow/tensorflow



Jax
https://github.com/jax-ml/jax

Tools

Developer

AmpliGraph [1]

STELLAR GRAPH [2]

PyG [3]

Spektral [4]

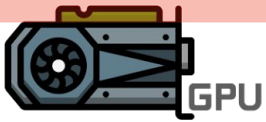
DGL DEEP GRAPH LIBRARY [5]

RAPIDS [10]

TensorFlow [6]

PyTorch [7]

cuDNN [8] cuBLAS [9] cuDF [11] cuML [12] cuGraph [13]

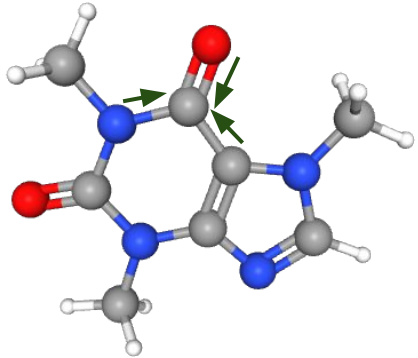


GPU

[1] https://github.com/Accenture/AmpliGraph/
[2] https://github.com/stellargraph/stellargraph
[3] https://github.com/pyg-team/pytorch_geometric
[4] https://github.com/danielegrattarola/spektral/
[5] https://github.com/dmlc/dgl
[6] https://www.tensorflow.org/
[7] https://pytorch.org/
[8] https://developer.nvidia.com/cudnn
[9] https://developer.nvidia.com/cublas
[10] https://rapids.ai/about.html
[11] https://github.com/rapidsai/cudf
[12] https://github.com/rapidsai/cuml
[13] https://github.com/rapidsai/cugraph

Why Graphs?

We design the **aggregation** of information **from the neighbors**



Permutation invariance is guaranteed, making the learning more sample efficient

Message Passing

We want to **aggregate** information from **neighbors** v to to update the current embedding u

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right),$$

Message

Hamilton, William L. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

Message Passing

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v,$$

$$\text{UPDATE}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}\right) = \sigma\left(\mathbf{W}_{\text{self}}\mathbf{h}_u + \mathbf{W}_{\text{neigh}}\mathbf{m}_{\mathcal{N}(u)}\right)$$

Basic GNN message passing formulation:

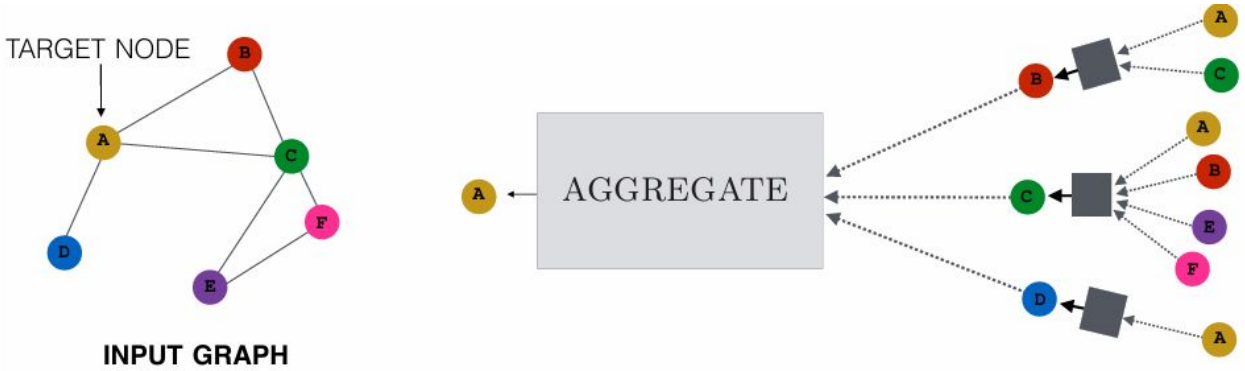
$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

Hamilton, William L. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

Message Passing

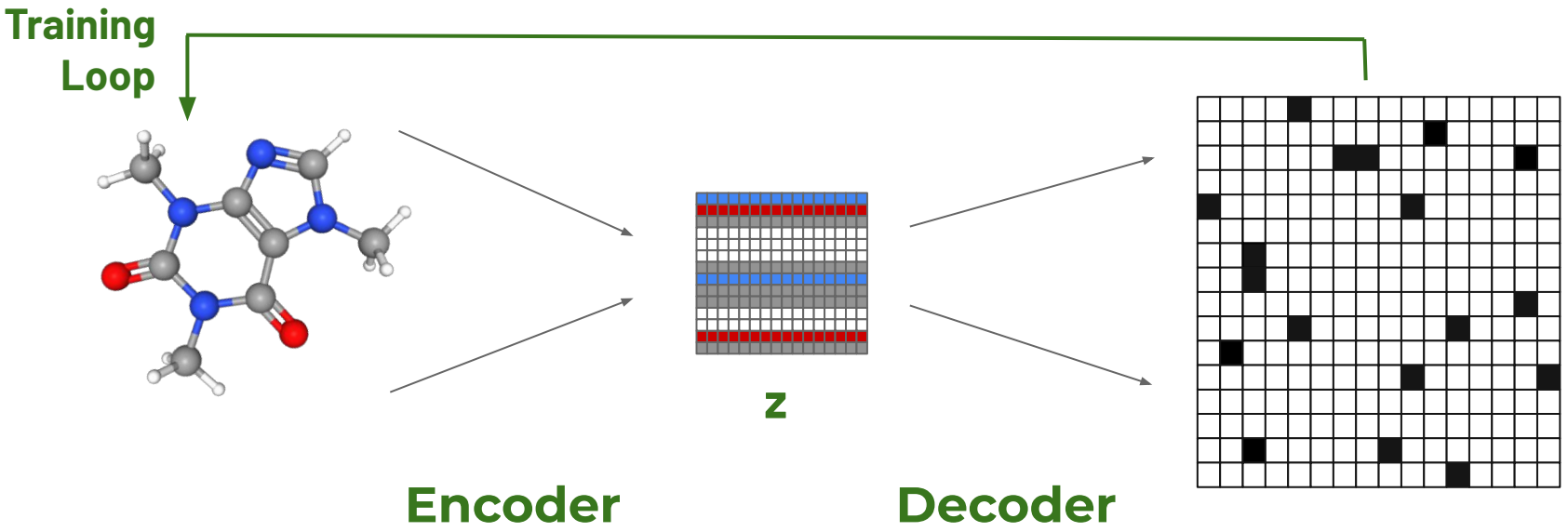
At each iteration we aggregate information from local neighborhood

After k iterations a node will contain information from its k-hop neighborhood



Hamilton, William L. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

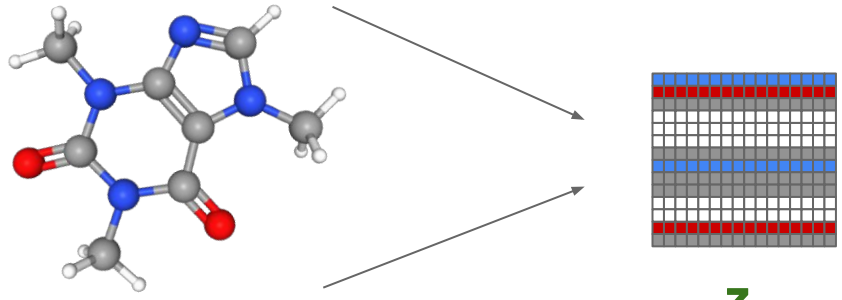
Learning Architecture - Auto Encoder



This unsupervised methodology is **not task specific**.

Hamilton, William L., Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications."

Learning Architecture - Auto Encoder



The encoder is **compressing knowledge**.

It produces an n-dimensional embedding for each node, where each feature describes a property of the input data.

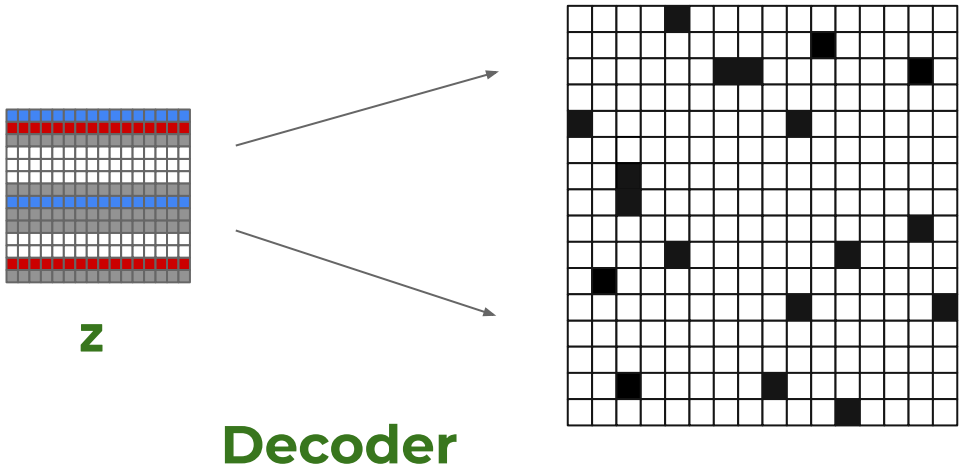
Hamilton, William L., Rex Ying, and Jure Leskovec. “Representation learning on graphs: Methods and applications.”

Learning Architecture - Auto Encoder

The decoder has to **reconstruct** the **adjacency** matrix:

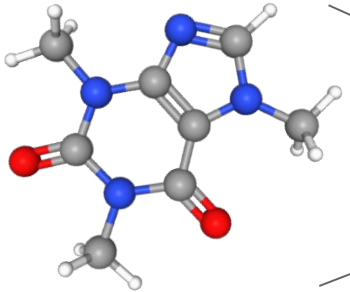
$$adj = z \cdot z^T$$

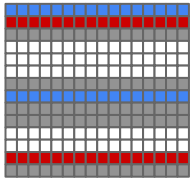
The representations of the nodes will be informative to perform this task.



Hamilton, William L., Rex Ying, and Jure Leskovec. "Representation learning on graphs: Methods and applications."

Whole-Graph Embedding





z

Encoder

Readout

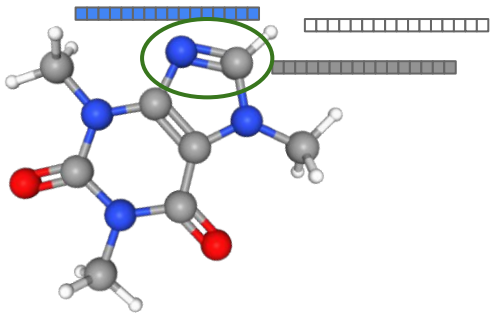
Pooling operations: get the max/mean of each feature

max (z) --> 

Gated aggregation: a learnable module select relevant node with gating.

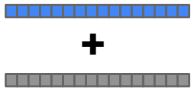
Attentional aggregation: ranked or weighted aggregation of nodes

Edge Embedding

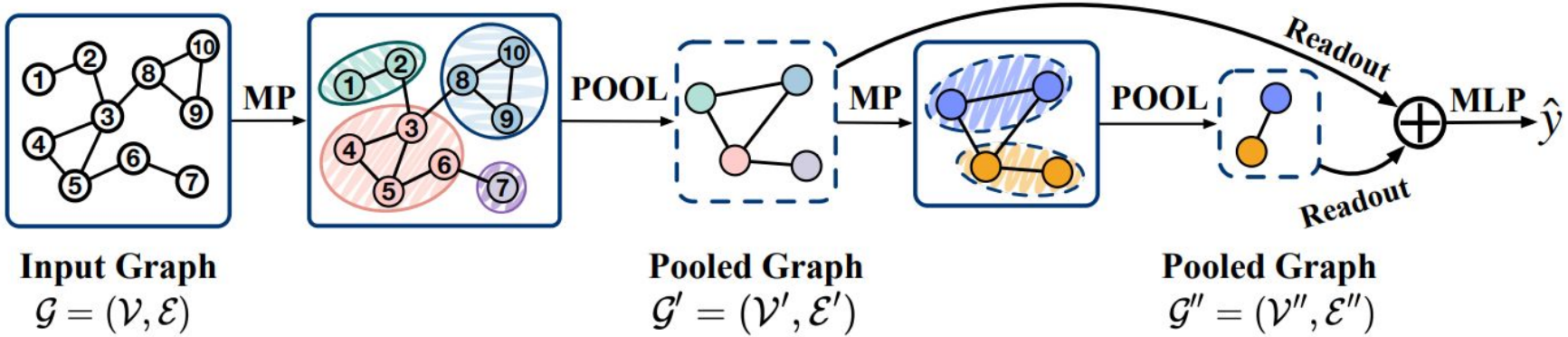


Concatenate start and end node embeddings

Sum start and end node embeddings



Local Pooling

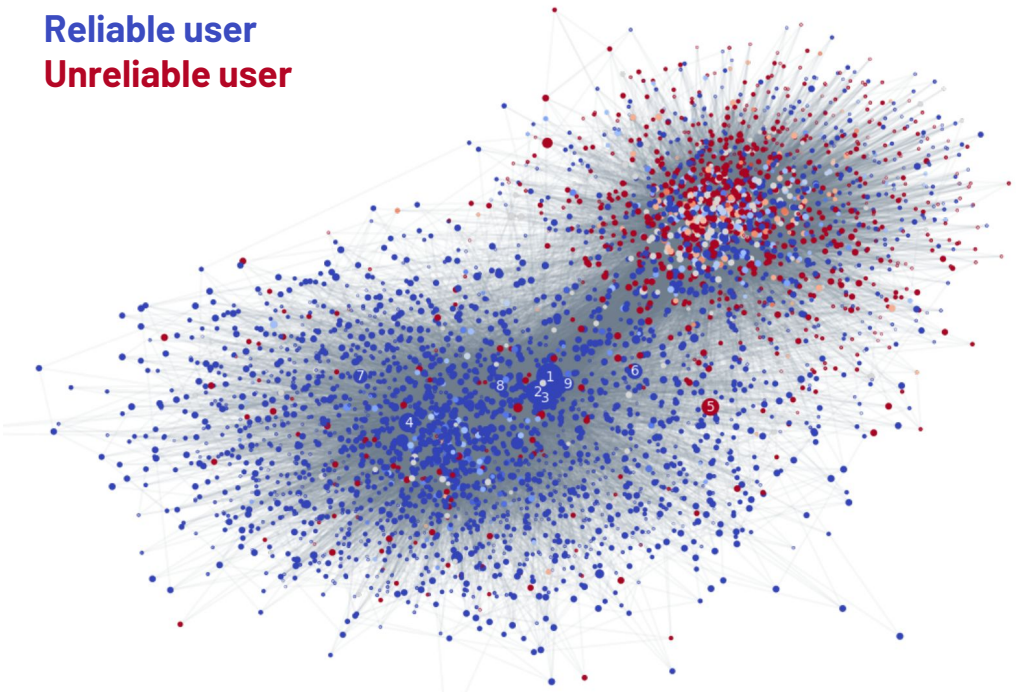


We can adopt **graph coarsening** and combine intermediate embeddings to get the final representations

Image from Liu, Chuang, et al. "Graph pooling for graph neural networks: Progress, challenges, and opportunities." *arXiv preprint arXiv:2204.07321* (2022).

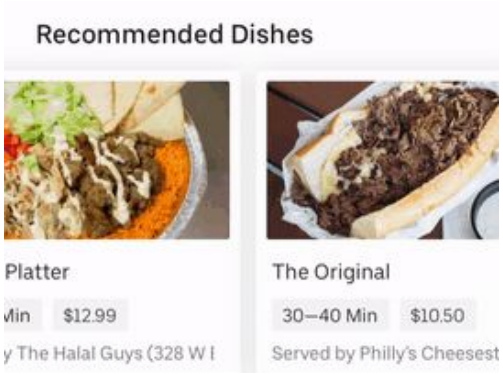
Fraud Detection

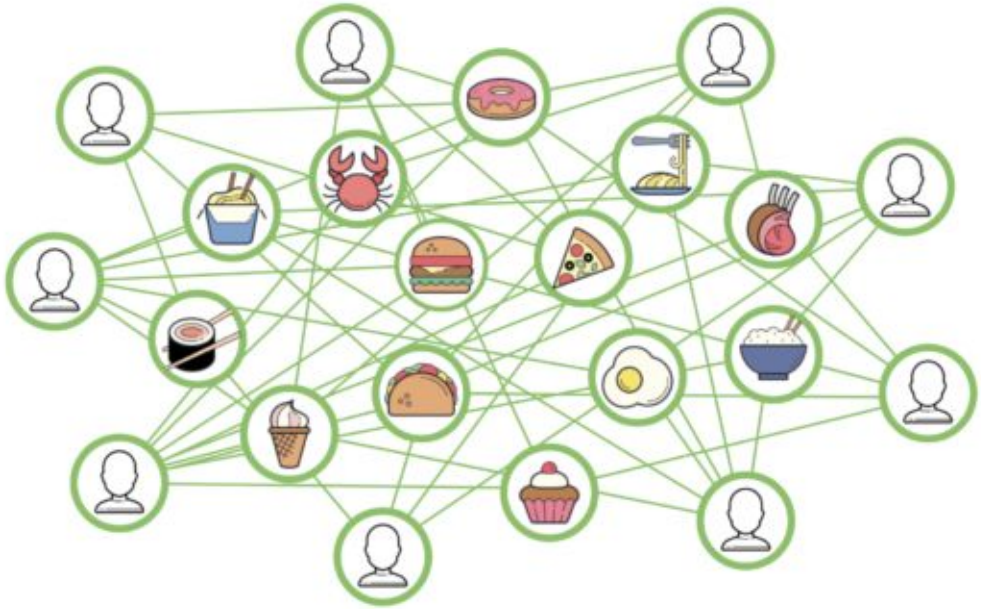
Fake News Detected on Social Media Using AI



https://arxiv.org/abs/1902.06673

Recommender Systems

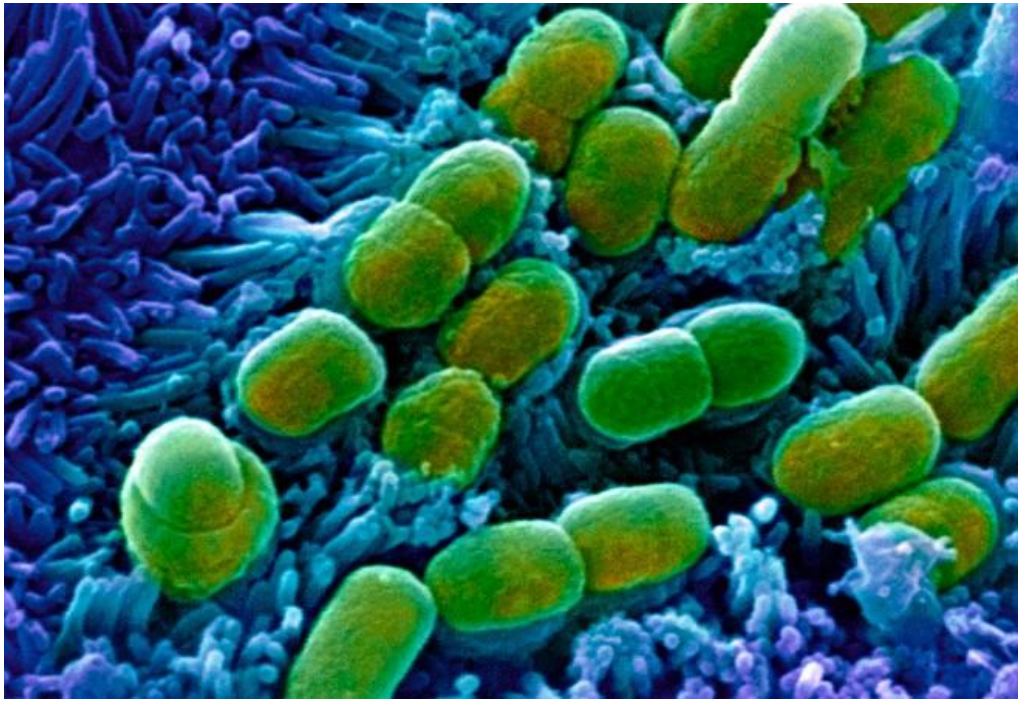




https://eng.uber.com/uber-eats-graph-learning/

Drug Repurposing

Powerful Antibiotics
Discovered Using AI



https://www.cell.com/cell/fulltext/S0092-8674(20)30102-1

Available Biomedical Data

Proteins

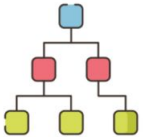
Diseases

Drugs

Gene Ontology

Genes

Human Phenotype

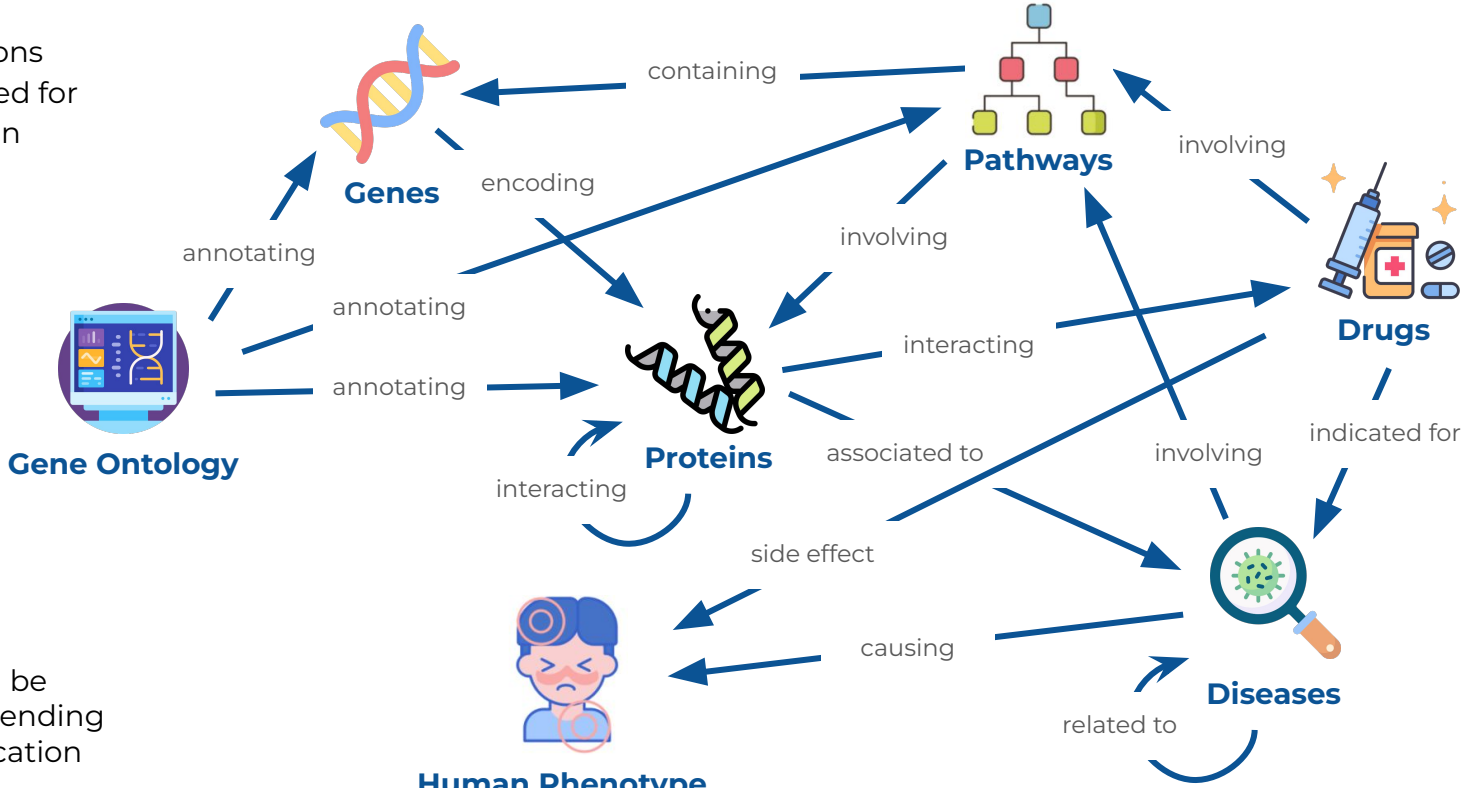


Pathways

Other data

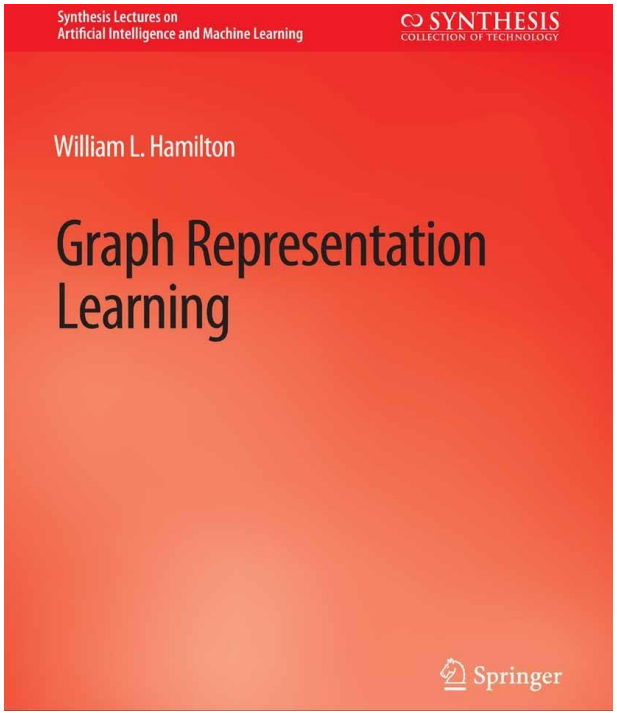
How Do We Organize the Knowledge

Note: some relations (edges) are omitted for better visualization



Other data can be integrated, depending on target application

The Ultimate Resource



- AKA “The Bible of Graph ML”
- Extends *"Representation learning on graphs: Methods and applications”*, using the same 4-blocks framework
- Freely available at:
  https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf

Challenge Rules

- Deadline: **30/06/25**
- Deliver **report (max 10 pages)** + **code/repository** to marco.santambrogio@polimi.it with leonardo.degrandis@polimi.it in CC
- Teams of max 2 people
- Oral exam will be scheduled after your delivery and before 24/07/25

Challenge 1 - ZINC

- About 250,000 molecular graphs with up to 38 heavy atoms: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.ZINC.html#torch_geometric.datasets.ZINC

- The task is to regress the penalized logP (also called constrained solubility)

Challenge 2 - DD

- D&D is a dataset of 1178 protein structures: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.TUDataset.html#torch_geometric.datasets.TUDataset

- Each protein is represented by a graph, in which the nodes are amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart

- The prediction task is to classify the protein structures into enzymes and non-enzymes

https://chrsmrrs.github.io/datasets/docs/datasets/

Icon from flaticon.com

Evaluation

Detailed report on architecture, results and training process:

- MAE (ZINC) / Accuracy (DD)
- Model size (# trainable parameters)
- Peak memory during training and inference
- Inference latency (time/sample)
- Training time (time/epoch)

Of the whole GML model if supervised, of the main encoder used to obtain the embeddings if unsupervised strategy

Strategies

Colab or Kaggle offer training hardware.

You may want to test different approaches:

- Global poolings (to get graph-level representations)
- Local poolings (to coarsen progressively the graphs)
- Mixed-precision / Quantization
- Pruning

Resources

https://github.com/leonardodegrandis/ACA-2025-GraphMachineLearning.git

- DeepFindr https://www.youtube.com/@DeepFindr
- Antonio Longa https://www.youtube.com/@94longa2112