Artificial Intelligence Applied to the Web

Chapter 3 Part 4 - Graph Neural Networks

Diego Cornejo, Felipe Hernández and Juan Velásquez

University of Chile
Departament of Industrial Engineering

Spring 2025

Outline

Graph Machine Learning

Neural Networks and Graphs

Unsupervised Graph Learning

Supervised Graph Learning

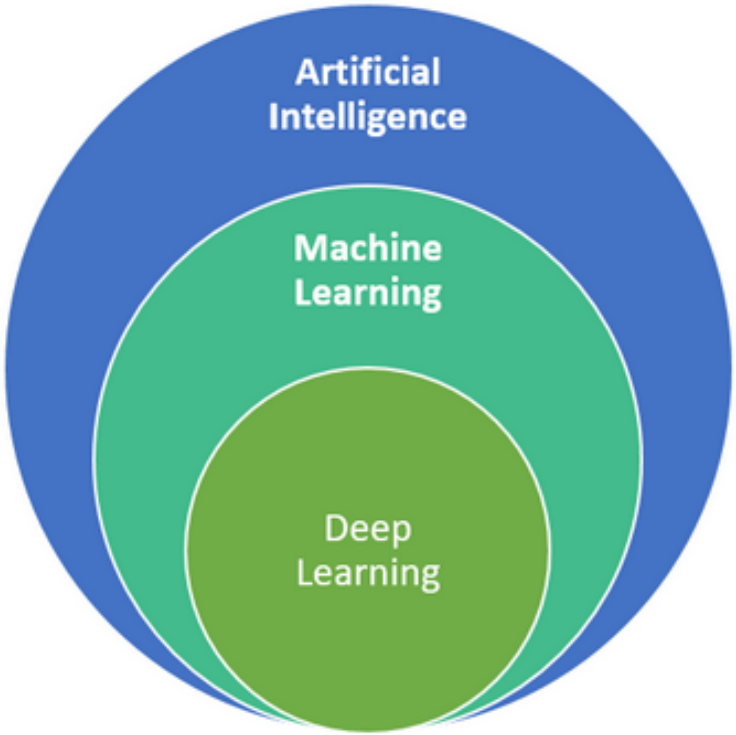
A Quick Recap

Machine Learning

- **Machine learning** is a subset of Artificial Intelligence that aims to provide systems with the ability to *learn*.
- It performance can be measured using a specific performance metric (also known as a **loss function**).
- According to T. Mitchell: *"An algorithm is said to learn from experience, E, if its performance at task T, measured by P, improves with experience E".*

A Quick Recap

Machine Learning



Machine Learning

Types of learning

- Four main learning types categories:
  - **Supervised**: The model learns from data where the correct answer is known, using labeled input-output pairs like.
  - **Unsupervised**: The model learns from unlabeled data, with the goal of discovering hidden structures, patterns, or similarities on its own.
  - **Semi-supervised**: A hybrid approach where the model is trained on a dataset containing both a small amount of labeled data and a large amount of unlabeled data.
  - **Reinforcement**: An agent learns to make a sequence of decisions by interacting with an environment, receiving rewards or penalties for its actions.

The Benefit of Machine Learning on Graphs

Why Use ML on Graphs?

- **Traditional ML has limitations**: Classic algorithms work best with grid-like data (e.g., tables, images) and often fail to capture the complex relationships inherent in graph structures.
- **Graphs model real-world systems**: Many systems, from social networks to biological interactions, are naturally represented as graphs.
- **Automatic pattern discovery**: The key advantage of Graph ML is its ability to **automatically detect and interpret complex, latent patterns** within graph-structured data.
- **Deeper insights**: These are patterns that are often too intricate for traditional models to identify, allowing for more effective modeling of relationships.

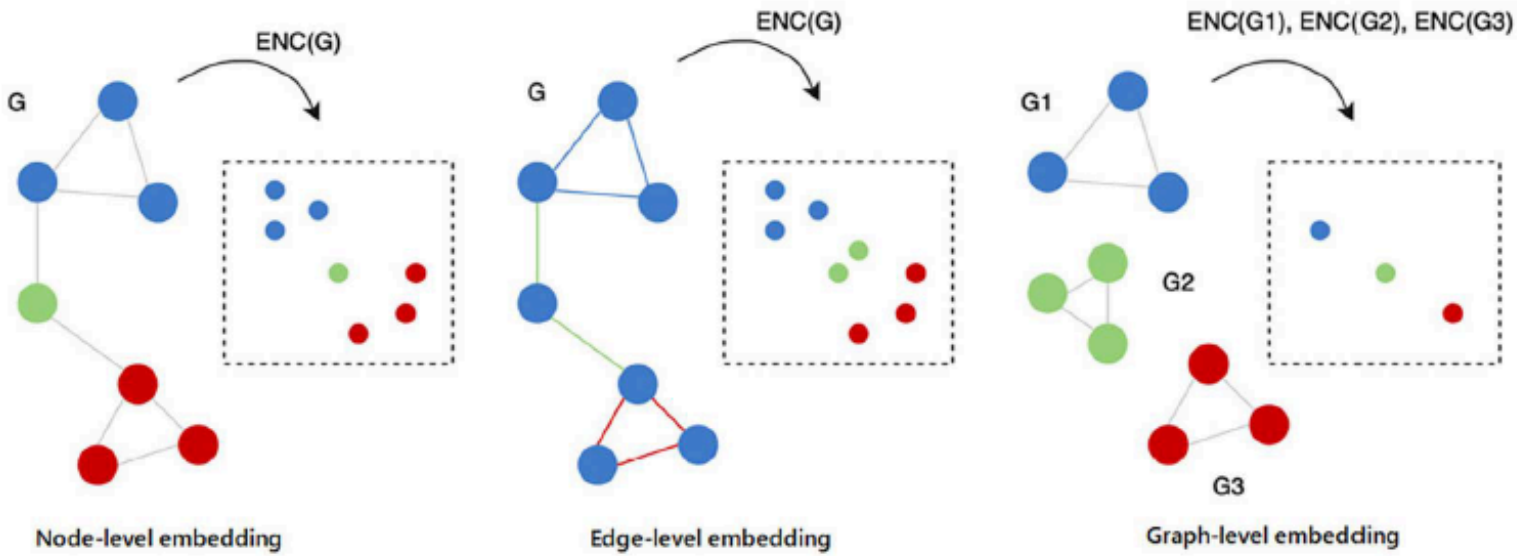
The Benefit of Machine Learning on Graphs

Why Use ML on Graphs?

- **Multi-level analysis**: Graph ML algorithms can extract and process features at different levels of granularity:
  - Node-level (e.g., classifying a user's role)
  - Edge-level (e.g., predicting a future friendship)
  - Graph-level (e.g., classifying an entire molecule)

The Benefit of Machine Learning on Graphs

Why Use ML on Graphs?



The Generalized Graph Embedding Problem

The Challenge with Graph Data

- **The Problem**: How do we feed a graph into a traditional machine learning algorithm that expects a fixed-size vector of numbers?
- **Classic Approach (Feature Engineering)**: Manually calculate graph metrics (like degree, efficiency, centrality) for each node or graph. This is time-consuming and may not capture all the important information.
- **The Modern Solution (Representation Learning)**: Instead of manual engineering, we learn a function that **automatically** maps the graph into a low-dimensional **vector space**. This process is called **network embedding**.
- **The Goal**: The embedding must preserve the structural properties of the original graph. Nodes that are "similar" in the graph should be close to each other in the new vector space.

What an Embedding Looks Like

From Structure to Space

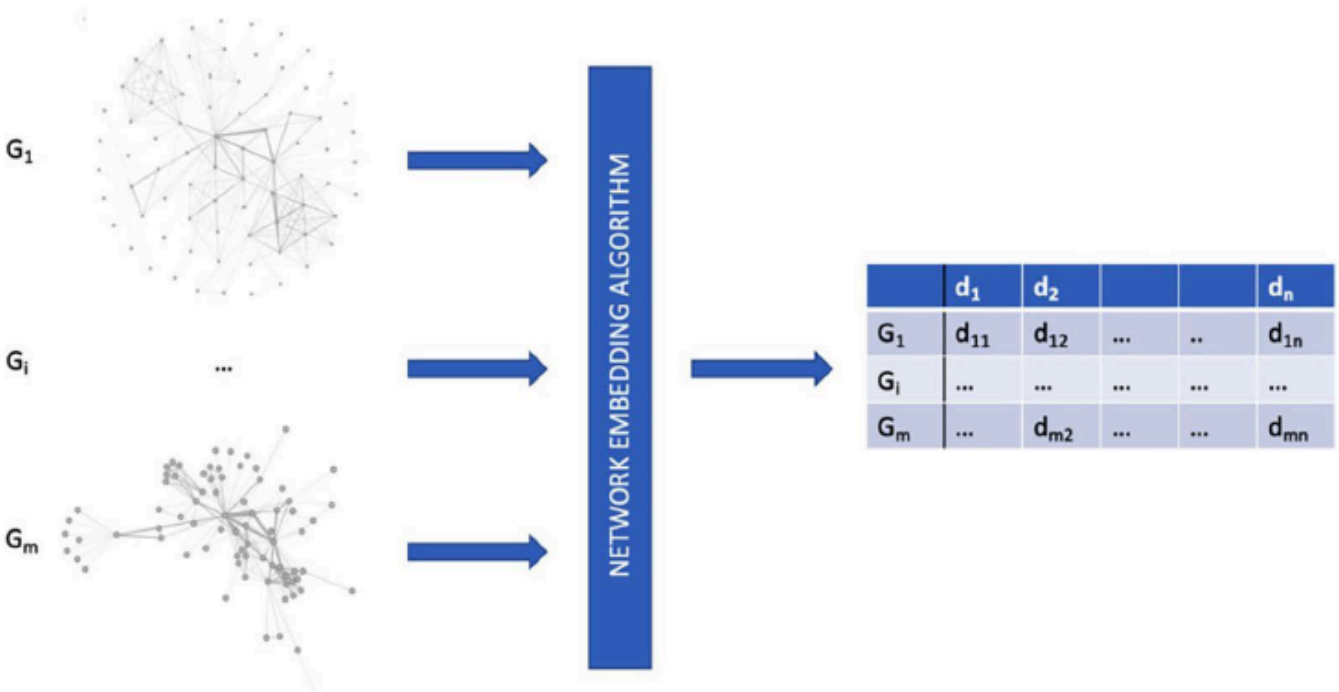
- An embedding algorithm transforms the graph's topology into geometric relationships.
- **Example**: In a barbell graph, there are three distinct structural groups: two dense clusters and one connecting path.
- After applying an embedding algorithm like `Node2Vec`, these three groups appear as clearly separated clusters of points in the new 2D vector space.
- This new vector representation can now be used as input for any standard ML model.

Key Distinction

- *Transductive* methods must be retrained to generate embeddings for new, unseen nodes.
- *Inductive* methods learn a general function that can create embeddings for new data without retraining.

What an Embedding Looks Like

From Structure to Space



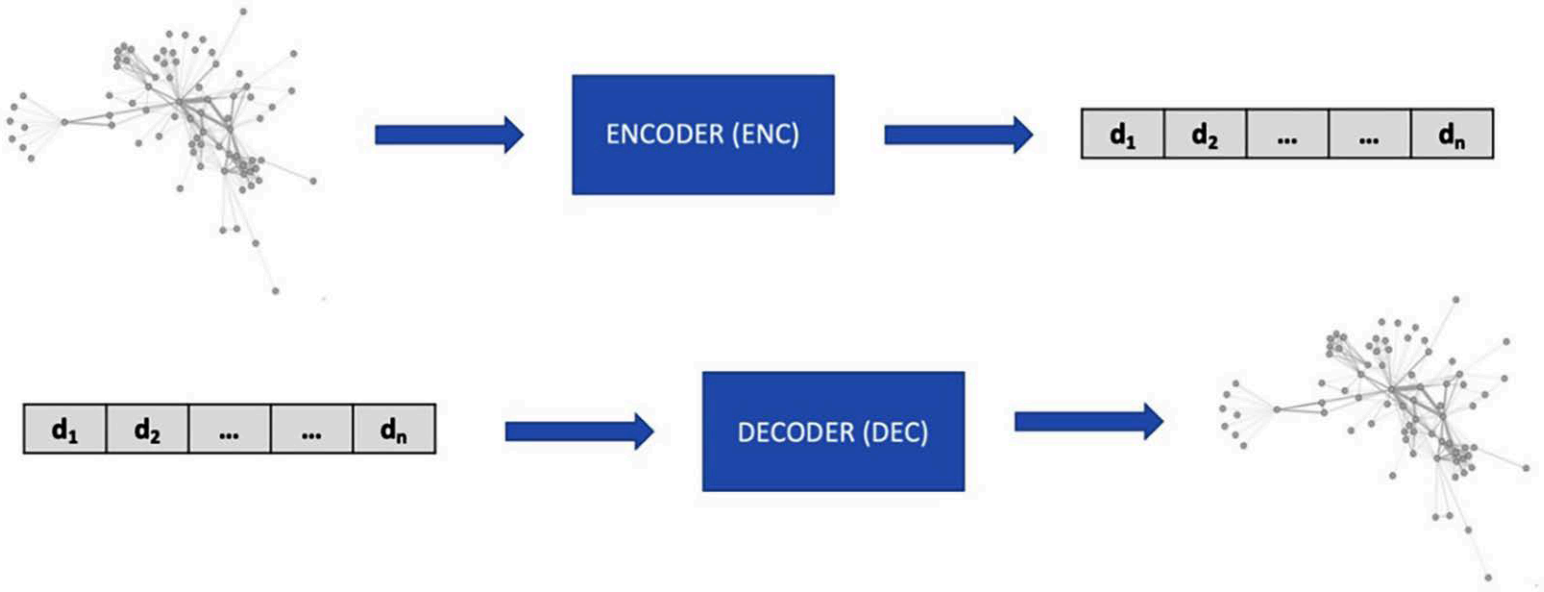
A Taxonomy of Embedding Algorithms

A Unifying Framework

- To organize the many different embedding algorithms, we can use a unified **Encoder-Decoder (ENC-DEC)** framework:
  - **The Encoder (ENC)**: This is a function that takes the input graph (or its nodes/edges) and maps it into the low-dimensional embedding space.
  - **The Decoder (DEC)**: This function takes the learned embedding and tries to decode structural information about the original graph from it (e.g., predict if two nodes are connected).
- **The Core Idea**: If an embedding is "good," it must contain enough compressed information for the decoder to successfully reconstruct the graph's key properties.

A Taxonomy of Embedding Algorithms

A Unifying Framework



The Four Main Categories

Algorithm Families

Based on the ENC-DEC framework, we can group algorithms into four main families:

- **Shallow Embedding Methods**: These are *transductive*. They learn a unique embedding for each node in the input graph and cannot generalize to new nodes without retraining. (e.g., `Node2Vec`).
- **Graph Autoencoding Methods**: These are *inductive*. They learn a general mapping function that can generate embeddings for new, unseen graph instances.

The Four Main Categories

Algorithm Families

- **Neighborhood Aggregation Methods**: Also *inductive*. These algorithms, like GNNs, learn a general function that creates embeddings by aggregating information from a node's local neighborhood and its features.
- **Graph Regularization Methods**: These methods use the graph structure to *regularize* a learning process on a set of features, enforcing that connected nodes should have similar predictions or embeddings.

The Generalized Loss Function

Training the Models

- These models are trained by optimizing a loss function that often combines two goals:

$$\text{Loss} = \alpha L_{\text{sup}}(y, \overline{y}) + L_{\text{rec}}\left(G, \overline{G}\right)$$

- The **Supervised Loss** (L_{sup}):
  - Used in supervised settings. It measures the difference between the model's prediction ($\overline{y}$) and the true label (y).
- The **Reconstruction Loss** (L_{rec}):
  - Used in unsupervised settings. It measures the error between the original graph (G) and the graph reconstructed by the decoder ($\overline{G}$).
- For purely unsupervised tasks, the coefficient α is set to 0. This framework elegantly unifies both learning paradigms.

Outline

Graph Machine Learning

Neural Networks and Graphs

Unsupervised Graph Learning

Supervised Graph Learning

A Quick Recap

What is an Artificial Neural Network?

- Artificial Neural Networks (ANNs) are the core of modern deep learning, and their popularity has exploded in recent years.
- They are versatile and powerful models that have been adopted across many domains, including graphs.
- Their success is thanks to recent advances in computing power (especially GPUs) and the availability of large datasets.

A Quick Recap

The Basic Building Block

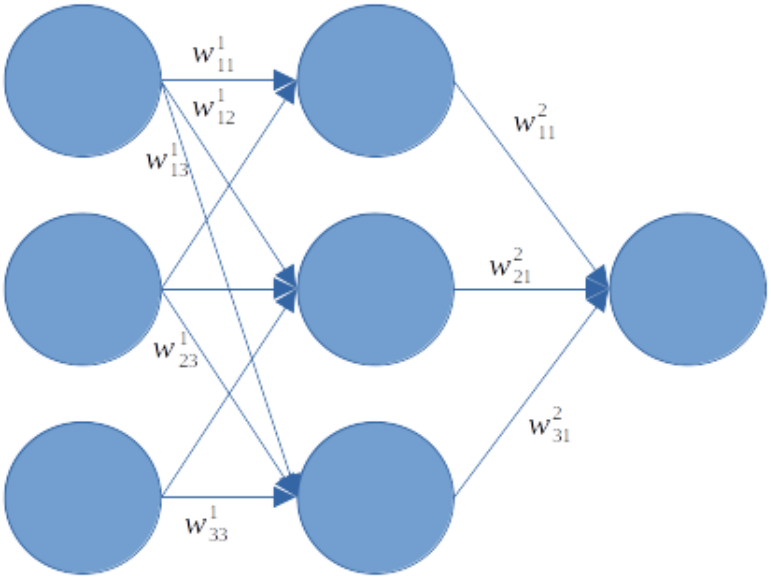
- An ANN is built from simple units called *neurons*.
- A neuron takes multiple inputs (x_i), combines them, and produces a single output (y).
- The relationship is defined by a simple mathematical formula:

$$y = f_a\left(b + \sum_i W_i x_i\right)$$

- This formula includes three key learnable components:
  - **Weights** (W_i): Determine the importance of each input.
  - **Bias** (b): An offset, similar to the intercept in a linear equation.
  - **Activation Function** (f_a): Introduces non-linearity.

A Quick Recap

Stacking Neurons into Layers



Introduction to GNNs

From Images to Graphs

- Graph Neural Networks (GNNs) are deep learning methods designed specifically to work on graph-structured data.
- They are also sometimes referred to as *geometric deep learning*.
- The core idea behind GNNs is a natural extension of **Convolutional Neural Networks (CNNs)**, which are incredibly successful on regular, grid-like data (e.g., images, text).

GNNs: The Core Concept

Neighborhood Aggregation

- A CNN works by applying a kernel that combines inputs from a pixel's local *neighborhood*.
- We can extend this same concept to graphs, which are non-Euclidean spaces.
- In a GNN, a node's "neighborhood" is not defined by physical distance, but by the **connections** (edges) embedded in the graph.
- A GNN layer works by aggregating information from a node's direct neighbors to compute a new, more complex feature representation for that node.

Variants of GNNs

The Two Main Approaches

- Many variations of the basic GNN have been proposed to improve their representation learning capabilities and to handle specific types of graphs (directed, dynamic, etc.).
- There are essentially two main types of convolutional operations for graph data:
  - **Spectral Approaches**: These methods define convolution in the *spectral domain* (by decomposing the graph into simpler elements).
  - **Spatial (Non-Spectral) Approaches**: These methods formulate convolution by directly *aggregating feature information* from a node's local neighbors.
- Other improvements also exist, such as adding new mechanisms to the propagation step, like *attention*, *gate mechanisms*, and *skip connections*.

Outline

Graph Machine Learning

Neural Networks and Graphs

Unsupervised Graph Learning

Supervised Graph Learning

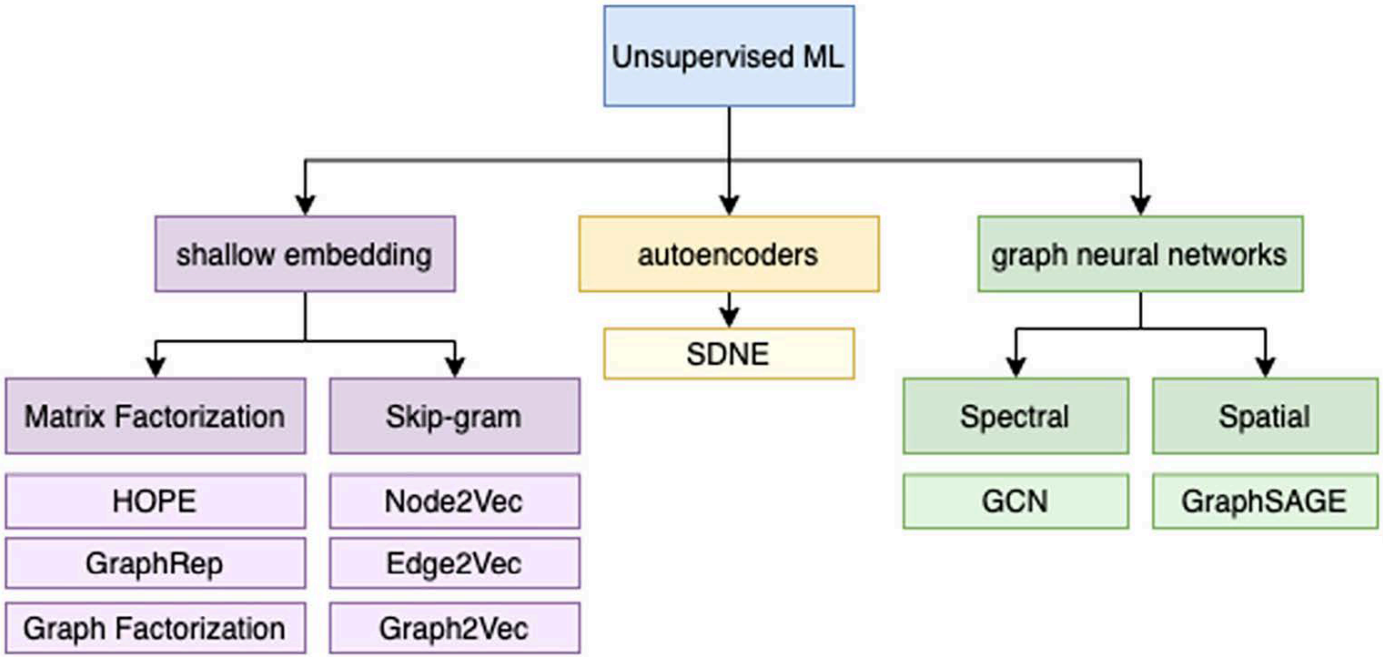
Unsupervised Graph Learning

What it is?

- The goal is to learn meaningful, low-dimensional representations (embeddings) of nodes, edges, or entire graphs **without using any labels**.
- The model learns embeddings that preserve the graph's structure by relying only on the adjacency matrix and node features.
- The learned representation is optimized so that it can be used to **reconstruct pair-wise node similarity** (e.g., rebuild the adjacency matrix).
- This process encodes latent relationships and allows us to discover novel, hidden patterns in the data.

Unsupervised Graph Learning

The Roadmap



Unsupervised: Shallow Embedding Methods

Learning Direct Representations

- Shallow embedding methods are a family of algorithms that learn a unique, low-dimensional vector for each node in a given graph.
- They are *transductive*. This means they only learn embeddings for the nodes they see during training. They cannot generate embeddings for new, unseen nodes without being completely retrained.
- We will explore two main categories of these methods:
  - **Matrix Factorization** based approaches.
  - **Skip-gram** based approaches, inspired by NLP.

Shallow Embedding: Matrix Factorization

Decomposing the Graph

- **The Core Idea**: These methods take a matrix representation of the graph—typically the adjacency matrix (A)—and decompose it into the product of two or more lower-dimensional matrices.
- The rows of these resulting low-dimensional matrices serve as the embeddings for the nodes.
- This process discovers the "latent factors" that explain the graph's structure.
- **Example Algorithm: Graph Factorization (GF)**
  - One of the earliest methods, which directly factorizes the adjacency matrix A.
  - It works well for undirected graphs where the adjacency matrix is symmetric.

Matrix Factorization: Higher-Order Proximity

Looking Beyond Direct Neighbors

- Simple factorization often only captures **first-order proximity** (direct connections).
- More advanced methods aim to preserve **higher-order proximity**, which captures relationships between nodes that are 2, 3, or more hops away.
- **Example Algorithm: HOPE (Higher-Order Proximity Preserved Embedding)**
  - Explicitly designed to preserve these longer-range relationships.
  - It uses two separate embedding matrices (one for "source" roles, one for "target" roles), making it effective for capturing asymmetric relationships in *directed graphs*.

Shallow Embedding: Skip-gram

The Word2Vec Analogy

- This family of methods is inspired by the famous **Word2Vec** algorithm from Natural Language Processing.
- It re-frames the graph embedding problem as a language modeling problem.
- **The Process**:
  1. Treat nodes as "words" and generate sequences of nodes ("sentences") by performing many **random walks** on the graph.
  2. Train a skip-gram neural network model on these "sentences". The model's goal is to predict a node's context (its neighbors in the walk) given the node itself.
  3. The learned weights of the network's hidden layer become the final node embeddings.

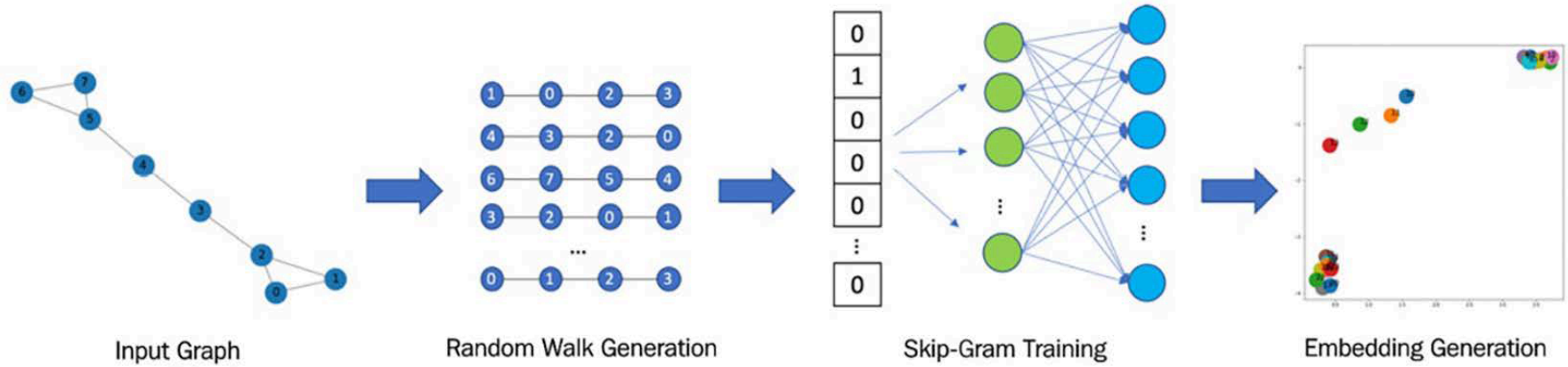
Skip-gram: DeepWalk vs. Node2Vec

Controlling the Walk

- **DeepWalk**:
  - The pioneering algorithm in this category.
  - It uses simple, completely **unbiased** random walks to explore the graph.
- **Node2Vec**:
  - The key innovation is the use of **biased** random walks.
  - It introduces two parameters, p and q, that control the walking strategy.
  - This allows the walk to balance between exploring a node's immediate neighborhood (Breadth-First-Search-like behavior) and exploring distant parts of the graph (Depth-First-Search-like behavior). This flexibility often leads to richer embeddings.

Skip-gram: DeepWalk vs. Node2Vec

Controlling the Walk



Unsupervised: Autoencoders

Learning to Reconstruct

- Autoencoders are a powerful type of neural network used for unsupervised learning and dimensionality reduction.
- Unlike methods like PCA, they can learn complex **non-linear** transformations.
- A key advantage is that they are *inductive*, meaning they learn a general function that can create embeddings for new, unseen data.
- **The Architecture**:
  - An **Encoder** network compresses the high-dimensional input into a low-dimensional latent vector (the embedding).
  - A **Decoder** network takes this embedding and tries to reconstruct the original input.

Autoencoders: A Key Variation

Denoising for Robustness

- A standard autoencoder can sometimes learn to just copy the input, a trivial "identity function," especially if the embedding space is large.
- To prevent this and learn more meaningful features, we use **Denoising Autoencoders**.
- **The Core Idea**:
  1. Intentionally add noise to (corrupt) the input data.
  2. Train the autoencoder to reconstruct the original, *clean* data from the noisy version.
- This forces the model to learn robust representations that capture the essential characteristics of the data while ignoring the noise.

Graph Autoencoders

Applying to Graph Structures

- We can apply the autoencoder concept to graphs by using the graph's **adjacency matrix** as the input and the target for reconstruction.
- However, this presents two critical challenges:
  - **Sparsity**: Most real-world graphs are very sparse, meaning the adjacency matrix is mostly zeros. A naive model will just learn to predict zero everywhere.
  - **Ambiguity**: The absence of a link (a zero in the matrix) does not necessarily mean two nodes are dissimilar; they might just be disconnected.

Graph Autoencoders in Practice

The SDNE Model

- To solve these challenges, models like **Structural Deep Network Embedding (SDNE)** use a specialized loss function.
- The loss function heavily penalizes reconstruction errors on **non-zero** elements (existing edges) more than on zero elements.
- It combines two objectives to preserve both:
  - **Second-order proximity**: The main autoencoder loss ensures that nodes with similar neighborhoods have similar embeddings (by reconstructing the adjacency vector).
  - **First-order proximity**: An additional loss term explicitly forces directly connected nodes to have very close embeddings in the latent space.

Unsupervised Learning: GNNs

The Inductive Approach

- Graph Neural Networks (GNNs) are deep learning methods that operate directly on graphs.
- A key advantage of GNNs is that they are generally *inductive*, meaning they learn a function that can generate embeddings for new, unseen data.
- In an unsupervised setting, GNNs are trained to create embeddings that capture the graph's structural properties without any external labels.
- Two main approaches: *spectral* and *spatial* convolution.

Unsupervised GNNs: Spectral Convolution

Operating in the Spectral Domain

- Spectral methods are based on *spectral graph theory* and define convolution in the Fourier domain, using the graph's **Laplacian matrix**.
- While mathematically elegant, they can be computationally expensive.
- A well-known and simplified propagation rule (from GCNs by Kipf & Welling) is a key example:

$$H^{l+1} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{l}W^{l}\right)$$

- **Limitations**:
  - These methods often require processing the entire graph at once.
  - They typically assume a fixed graph, which can make it difficult to generalize to new graph structures.

Unsupervised GNNs: Spatial Convolution

Aggregating Neighbors Directly

- Spatial methods, like **GraphSAGE**, perform convolution by directly aggregating feature information from a node's local neighborhood.
- This approach is more scalable and flexible.
- **How it works for unsupervised learning**:
  - The model is trained using a loss function based on node similarity, often derived from random walks.
  - The goal is to learn embeddings that ensure:
    - Nodes that are "close" (e.g., appear in the same random walk) have **similar** vector representations.
    - Nodes that are "distant" have **dissimilar** vector representations.

Unsupervised GNNs in Practice

A Self-Supervised Example

- A common way to train a GNN without labels is to create a **self-supervised** task.
- The model learns to predict some intrinsic property of the graph itself.
- **Example: Learning Graph-Level Embeddings**
  1. Take a dataset of many individual graphs (e.g., the PROTEINS dataset).
  2. Define a "ground-truth" distance between any pair of graphs (e.g., based on their Laplacian spectrum).
  3. Train a GNN-based Siamese network to predict this distance.
- By learning to predict the distance, the GNN is forced to learn a meaningful embedding for each graph that encodes its unique structural properties.

Outline

Graph Machine Learning

Neural Networks and Graphs

Unsupervised Graph Learning

Supervised Graph Learning

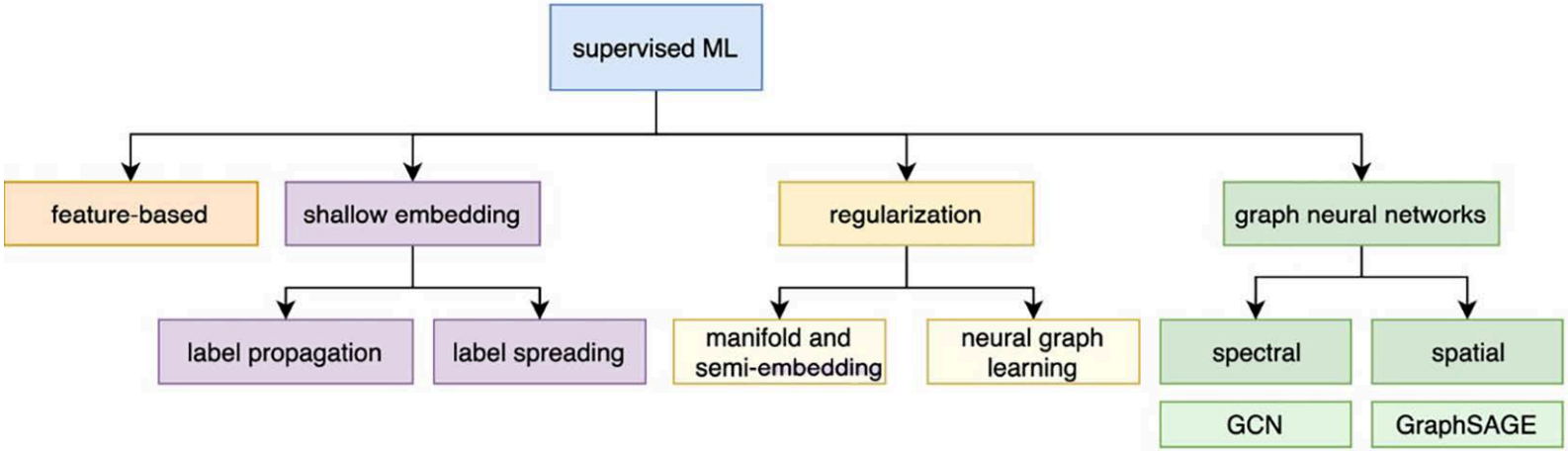
Supervised Graph Learning

What it is?

- In supervised learning, our training set consists of pairs of `(x, y)`, where `x` is the graph input and `y` is a known output label.
- We aim to learn the function that maps inputs to these labels.
- Four main approaches:
  - Feature-Based Methods
  - Shallow Embedding Methods
  - Graph Regularization Methods
  - Graph Neural Networks (GNNs)

Supervised Graph Learning

The Roadmap



Supervised: Feature-Based Methods

The Classic Approach

- This is one of the simplest, yet often powerful, methods for applying machine learning to graphs.
- It's a straightforward two-step process:
  1. **Feature Extraction**: For each node or graph, manually compute a set of descriptive structural properties. These could be any of the related metrics, like:
     - Number of edges
     - Average clustering coefficient
     - Centrality scores
  2. **Train a Classic Classifier**: Use these computed properties as a standard feature vector to train a traditional ML algorithm like an SVM, Random Forest, or Logistic Regression.

Feature-Based Methods

Pros and Cons

- **Strengths**:
  - Easy to implement and understand.
  - The features are highly interpretable (e.g., "number of edges is predictive of class X").
  - Leverages well-established and trusted ML techniques.
- **Weaknesses**:
  - Relies heavily on manual **feature engineering**, which can be time-consuming and requires domain expertise.
  - The chosen features might not be optimal and could fail to capture the complex, latent relationships within the graph structure.

Supervised: Shallow Embedding Methods

Propagating Information

- These are semi-supervised, *transductive* algorithms designed for **node classification**.
- They are ideal for scenarios where you have a small number of labeled nodes and a large number of unlabeled ones.
- The core idea is that labels from the few known nodes are iteratively "propagated" to their unlabeled neighbors through the graph's edge structure, based on the assumption that "nearby nodes likely have the same label."

Shallow Embedding: Label Propagation

A Simple Iterative Method

- This is a well-known algorithm that works as follows:
  1. Each unlabeled node is initialized with a probability distribution over the labels.
  2. In each iteration, every node updates its label distribution to match the average of its neighbors' distributions.
  3. This process repeats until the labels in the network stabilize.
- The labels of the initially known nodes are considered "ground truth" and are **fixed**—they are reset to their original value at every iteration.
- Main limitation: This makes the algorithm very sensitive to noise or errors in the initial labels.

Shallow Embedding: Label Spreading

A More Flexible Approach

- This algorithm is an evolution of Label Propagation that addresses its main limitation.
- It **relaxes** the constraint on the initial labels, allowing their distributions to change during the iterative process.
- It introduces a regularization parameter (α) that balances the influence of two forces:
  - Adhering to the graph structure (information from neighbors).
  - Sticking to the initial "ground truth" label assignment.
- This makes the algorithm more **robust to noisy initial labels**, as it can "correct" a potentially wrong initial label if its neighbors strongly disagree.

Supervised: Graph Regularization

Using the Graph as a Constraint

- This is a powerful technique that integrates the graph structure directly into a model's **loss function**.
- **Core Assumption (Manifold Hypothesis)**: Connected nodes in the graph should have similar predictions or embeddings.
- The total loss function combines two terms:

$$\text{Loss} = \text{Loss}_{\text{supervised}} + \text{Loss}_{\text{graph}}$$

  - The **supervised loss** ensures the model fits the labeled data.
  - The **graph loss** penalizes the model if connected nodes have dissimilar predictions, enforcing "smoothness" and improving generalization.

Supervised: Graph Neural Networks

The End-to-End Approach

- GNNs provide the most powerful and flexible approach for supervised graph learning.
- The GNN acts as an **encoder**, learning rich feature representations of nodes or graphs that are specifically tailored to the supervised task.
- The final embeddings from the GNN are then passed to a standard classification or regression head (e.g., a Dense layer) to make the final prediction.
- The entire model, from the graph structure to the final prediction, is trained **end-to-end**, so the GNN learns the most predictive features automatically.

Supervised GNNs in Practice

Examples

GNNs can be applied to any supervised graph task. For example:

- Graph Classification with GCNs:
  - **Task**: Classify entire graphs (e.g., is a protein an enzyme?).
  - **Method**: A GCN backbone learns a representation for the entire graph, which is then fed into a classifier.
- Node Classification with GraphSAGE:
  - **Task**: Classify individual nodes within a larger graph (e.g., what is the research topic of a paper?).
  - **Method**: A GraphSAGE model learns embeddings for each node based on its local neighborhood, and these embeddings are then used to make a classification.

Artificial Intelligence Applied to the Web

Chapter 3 Part 4 - Graph Neural Networks

Diego Cornejo, Felipe Hernández and Juan Velásquez

University of Chile
Departament of Industrial Engineering

Spring 2025