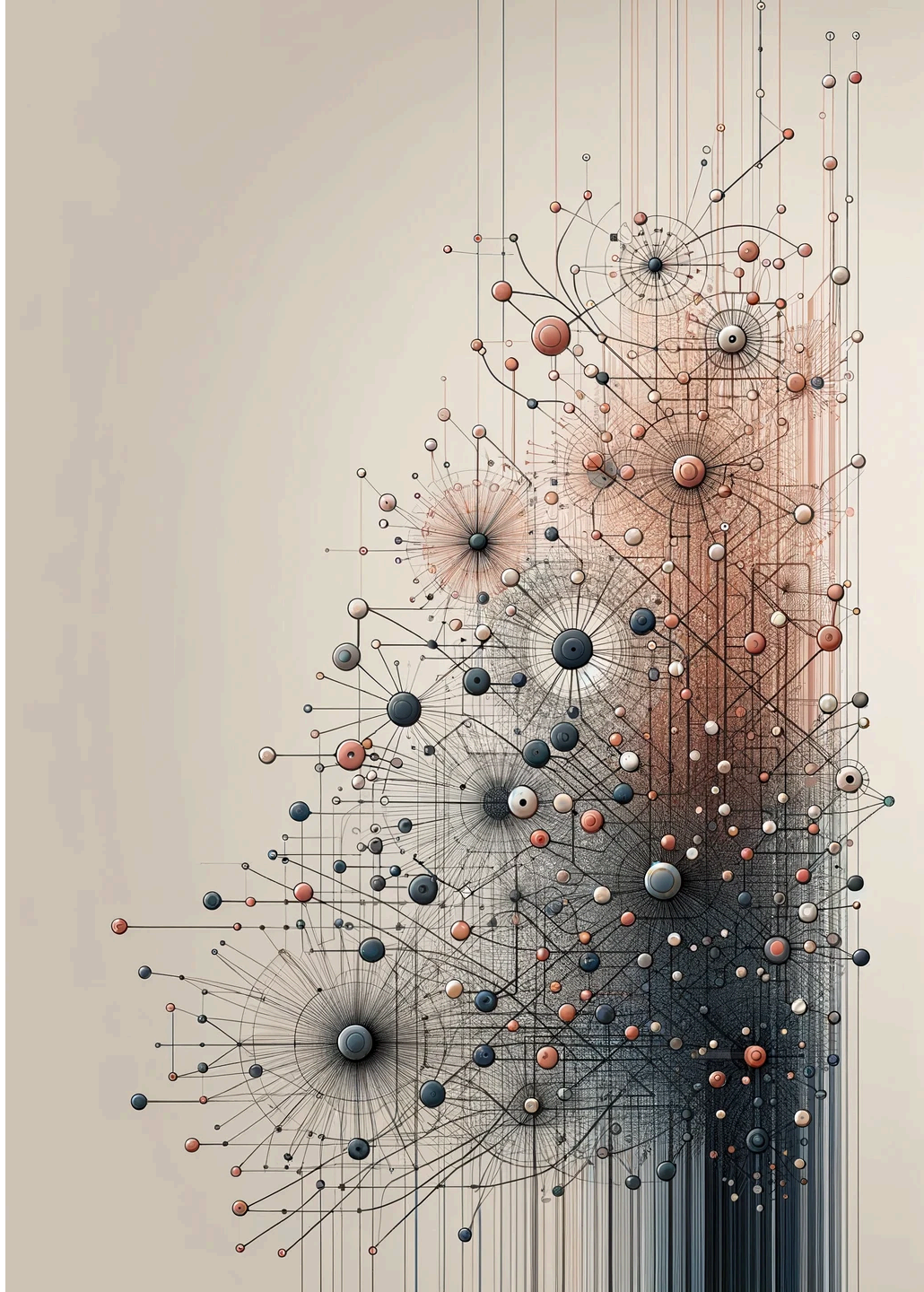
UNIVERSITÀ DI TORINO

Analisi e Visualizzazione delle Reti Complesse

NS21 - Representation Learning on Graphs

Prof. Rossano Schifanella



UNIVERSITÀ DI TORINO

Agenda

1. **Foundations**: Traditional approaches vs. representation learning
2. **Mathematical Framework**: Encoders, decoders, and objective functions
3. **Shallow Embedding Methods**: Detailed look at DeepWalk and node2vec
4. **Alternative Approaches**: Matrix factorization and spectral methods
5. **Applications**: From node classification to graph-level tasks
6. **Frontiers**: Current limitations and future directions

Traditional Feature Engineering

- **Manual feature design**: Degree, clustering coefficient, betweenness, ...
- **Example**: To predict influential users in a social network:
  - Engineer features: degree, PageRank, community membership
  - Feed these into classical ML models (SVM, Random Forest)
- **Limitations**:
  - Labor-intensive
  - Requires domain expertise
  - Features designed for one task often don't transfer to others

Why Traditional Methods Fall Short

- Networks have intricate dependencies that are hard to capture manually
- Complex patterns exist beyond simple statistics
- **Real example**: In protein-protein interaction networks, functional similarity depends on complex structural patterns that aren't captured by simple metrics

A New Paradigm: Learn the Features

- Automatically discover representations that capture network structure
- Use these representations for multiple downstream tasks
- **Key difference**: Features emerge from the data, not human intuition

What is an Embedding?

- An **embedding** is a mapping from a discrete object to a vector of real numbers
- In the vector space, similar objects should have similar vector representations
- These vectors can be used as input to machine learning algorithms

Types of Graph Embeddings

- **Node embeddings**: Map each node to a vector (focus of this lecture)
  - Applications: Node classification, link prediction, recommendation
- **Edge/relation embeddings**: Map relationships to vectors
  - Applications: Knowledge graphs, multi-relational networks
- **Graph embeddings**: Map entire graphs to vectors
  - Applications: Graph classification, similarity search between graphs

UNIVERSITÀ DI TORINO

Node Embedding: Mathematical Foundations

Embedding as Dimensionality Reduction

- Networks are inherently high-dimensional structures (each node is a dimension)
- Adjacency matrix $A \in \{0,1\}^{|V|\times|V|}$ has $O(|V|^2)$ entries
- We seek a low-dimensional representation $Z \in \mathbb{R}^{|V|\times d}$ where $d \ll |V|$
- This is analogous to classical dimensionality reduction, but preserving graph-specific structures

Objective Functions & Preservation Properties

- **First-order proximity**:
  - Directly connected nodes should be close in embedding space
  - Example: Friends in a social network → similar vectors
  - Preserves immediate connections
- **Second-order proximity**:
  - Nodes sharing many neighbors should be similar, even if not connected
  - Example: Two professors who collaborate with the same researchers
  - Captures "friends of friends" relationships
- **Global structure preservation**:
  - Maintains overall network distances and paths
  - Example: Preserving communities or hierarchical structures
  - Useful when analyzing network roles or positions

Different methods prioritize different types of proximity based on the application goals.

UNIVERSITÀ DI TORINO

Goal: Learning Meaningful Node Representations

The Embedding Function

We seek a mapping function $f : V \to \mathbb{R}^d$ that transforms each node $v \in V$ into a d-dimensional vector $\mathbf{z}_v$ such that:

- Similar nodes in the graph have similar embeddings
- The embedding space preserves meaningful graph properties
- $d \ll |V|$ (dimensionality reduction)

Formal Setup: The Graph Representation Problem

Input:

- Graph $G = (V, E)$ with nodes V and edges E
- Adjacency matrix A where $A_{ij} = 1$ if $(i, j) \in E$
- Initially, no node features or labels (unsupervised setting)

Output:

- Node embedding matrix $Z \in \mathbb{R}^{|V| \times d}$
- Each row $\mathbf{z}_v \in \mathbb{R}^d$ is the embedding for node v

Objective:

Ensure that $\mathbf{z}_u^T \mathbf{z}_v$ (or similar similarity function) reflects meaningful relationships between nodes u and v in the original graph

From Problem to Solution: Learning Framework

The Learning Challenge

- We now have a clear objective: Learn node embeddings where similarity in embedding space reflects graph relationships
- But how exactly do we learn these embeddings?
- We need a systematic approach that:
    i. Defines what relationships to preserve from the original graph
    ii. Maps nodes to embeddings efficiently
    iii. Ensures embeddings capture the desired relationships
    iv. Provides a training objective to optimize

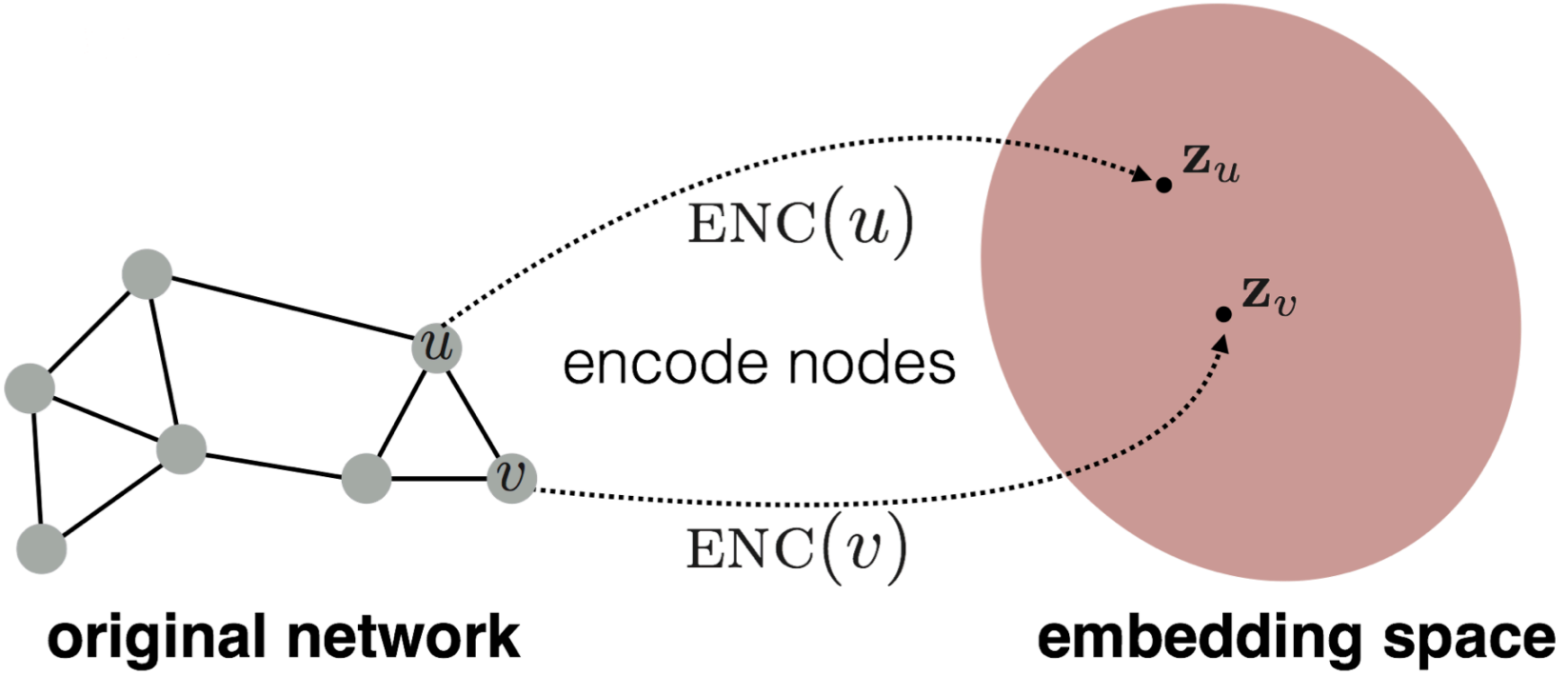
We need to understand:

- What **similarity** in the original graph means (e.g., direct connections, shared neighbors)
- How to map from nodes to vectors (**encoding**)
- How to predict similarity from embeddings (**decoding**):
  - Converting vector representations back to predicted graph similarities
- How to measure and minimize differences between true and predicted similarities:
  - Define a **loss function** that quantifies the error (e.g., mean squared error)
  - Use **optimization techniques** like stochastic gradient descent to find embeddings that minimize this loss

UNIVERSITÀ DI TORINO

A Unified View

1. **Encoder**: $ENC(v) = \mathbf{z}_v$
   - Maps nodes to vector representations
2. **Similarity function**: $s_G(u, v)$
   - Measures node similarity in the original graph
   - Examples: Adjacency, shortest path distance, common neighbors
3. **Decoder**: $DEC(\mathbf{z}_u, \mathbf{z}_v) \approx s_G(u, v)$
   - Reconstructs graph similarity from embeddings
   - Often implemented as vector dot product: $\mathbf{z}_u^T \mathbf{z}_v$
4. **Optimization objective**:
   - Minimize reconstruction error: $\sum_{u,v \in V} \|DEC(\mathbf{z}_u, \mathbf{z}_v) - s_G(u, v)\|^2$


$\mathbf{z}_u$
$\mathbf{z}_v$
$\mathrm{ENC}(u)$
encode nodes
$\mathrm{ENC}(v)$
u
v
original network
embedding space

Types of Encoders

Graph embedding methods differ in how they encode nodes into vectors:

1. **Shallow encoders (the focus on this class)**:
    - Direct mapping from node IDs to embedding vectors
    - Each node has its own independent embedding vector
    - No parameter sharing between nodes
    - Examples: DeepWalk, node2vec, LINE, HOPE
2. **Deep encoders**:
    - Use neural networks to compute embeddings
    - Process node features and structural information
    - Parameters are shared across nodes
    - Examples: Graph Neural Networks (GNNs), Graph Convolutional Networks (GCNs)

Comparative Analysis

Aspect	Shallow Encoders	Deep Encoders
Training	Faster, simpler	More complex, requires tuning
Parameters	$O(\|V\| \times d)$	$O(d^2)$ (independent of graph size)
Features	Cannot use node features	Can leverage node & edge attributes
Inductive	No (transductive only)	Yes (can generalize to new nodes)
Scalability	Limited by memory	Limited by computation
Applications	Static graphs, simple tasks	Dynamic graphs, complex tasks

Inductive vs. Transductive Node Embeddings

Transductive embeddings are learned specifically for nodes seen during training. They require retraining or fine-tuning to handle new nodes.
Example methods: DeepWalk, node2vec, spectral embeddings.

Inductive embeddings learn generalized embedding functions, enabling the generation of embeddings for new, unseen nodes or graphs without retraining.
Example methods: GraphSAGE, Graph Attention Networks (GAT), GIN.

- **Key difference:** Inductive methods generalize easily; transductive methods do not.

UNIVERSITÀ DI TORINO

Shallow Encoding: The Embedding Lookup

The Simplest Encoder

- Direct mapping from nodes to embeddings
- Represented as a matrix $Z \in \mathbb{R}^{|V| \times d}$
- $ENC(v) = Z[v]$ (lookup of row v in matrix Z)

Optimization Approach

- Learn the entries of Z directly during training
  - Unlike neural networks, we directly optimize the embedding values themselves
  - The matrix elements are the actual parameters we're learning
- Each node gets its own parameter vector that's updated independently
  - No parameter sharing between nodes
- **Advantage**: Simple, efficient for moderate-sized graphs

How to Define Node Similarity?

- Remember our goal: We want embeddings where **similar nodes** have **similar vectors**
- But what does "similar nodes" actually mean in a graph context?
- This is not just a technical detail; it's the fundamental question that shapes:
  - What patterns your embeddings will capture
  - Which downstream tasks they'll perform well on
  - What aspects of the complex system they'll represent

From Graph Theory to Learning Objective

- Node similarity is the bridge between graph structure and vector space
- The similarity function $s_G(u, v)$ becomes our **learning target**
- The encoder-decoder framework aims to make $\mathbf{z}_u^T \mathbf{z}_v \approx s_G(u, v)$
- Different definitions of $s_G(u, v)$ lead to different embedding methods

Graph-Based Similarity Measures for Nodes

Similarity Measure	Mathematical Form	What It Captures	Visual Example
First-order (Adjacency)	A_{uv} (direct connection)	Immediate neighbors	u -- v
Second-order (Common neighbors)	$\sum_w A_{uw}A_{wv}$ or A^2_{uv}	Shared neighborhood	u -- w -- v
Higher-order (k-step paths)	$(A^k)_{uv}$	Multi-hop connections	u -- ... -- v
Combined (Katz Index)	$\sum_{k=1}^{\infty} \beta^k (A^k)_{uv}$	All-path influence with decay	Weighted paths of all lengths

Key Insights

Node similarity measures are formalized ways to answer:

"How related are nodes u and v in the network?"

Different answers to this question lead to embeddings that preserve different aspects of network structure.

From Theory to Practice: The Computational Challenge

Recall our optimization objective:

- We want to minimize $\sum_{u,v \in V} \|DEC(\mathbf{z}_u, \mathbf{z}_v) - s_G(u,v)\|^2$
- This requires computing $s_G(u,v)$ for each node pair in the graph
- For large networks, this becomes computationally infeasible:
  - Many similarity measures (like Katz index) are themselves expensive to compute

The Scalability Problem:

- Computing similarity between all node pairs requires $O(|V|^2)$ operations
- Most graph mining tasks need to scale to millions of nodes
- **We need a more efficient approach than exhaustive computation**

Sampling-Based Solution

- Instead of computing all pairwise similarities:
  i. Sample meaningful node pairs that are likely similar (positive examples)
  ii. Sample random node pairs that are likely dissimilar (negative examples)
  iii. Learn embeddings that differentiate between them

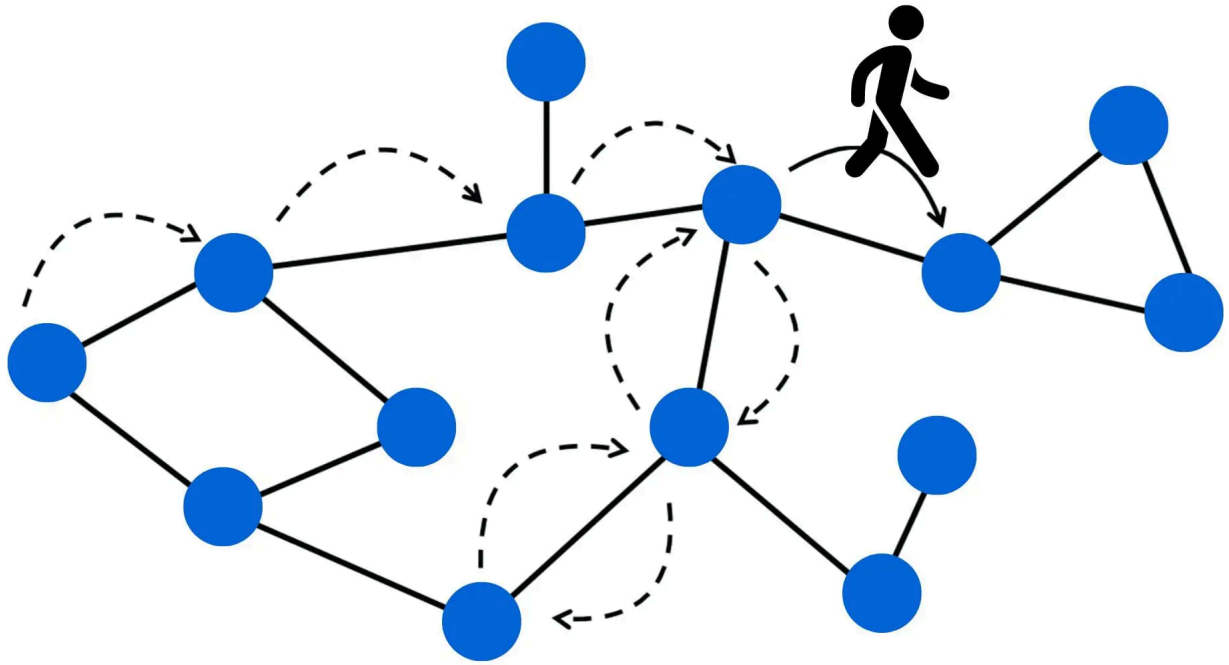
This is why random walks become important: they provide a systematic way to sample meaningful node pairs without computing all pairwise similarities explicitly.

What is a Random Walk?

- Start at node u
- Repeatedly move to a random neighbor
- Record the sequence of visited nodes: $(u, v_1, v_2, ..., v_k)$

Why Random Walks Define Good Similarity

1. **Flexible definition of context**: Nodes appearing in same walks are related
2. **Balance of local & global**: Can capture both community and structural roles
3. **Computational efficiency**: No need to compute similarities for all node pairs



From Walks to Node Embeddings

For each node u, define its "network neighborhood" $N_R(u)$ as nodes that appear near u in random walks.

Optimization Problem

Maximize the probability of observing the neighborhood given the node:

$$\max_f \sum_{u \in V} \log P(N_R(u)|f(u))$$

Where $f(u) = \mathbf{z}_u$ is the embedding function.

In Practice: Skip-Gram Objective

The Word-Node Analogy:

- **Language & Networks**: Just as sentences are sequences of related words, random walks are sequences of related nodes
- **Context in Language**: Words appearing near "bank" could be "money," "loan," or "river" (defining its meaning)
- **Context in Networks**: Nodes appearing near a person could be friends, colleagues, or family (defining their social role)
- **Distributional Hypothesis**: In both domains, meaning/function comes from context
  - "You shall know a word by the company it keeps" (Firth, 1957)
  - "You shall know a node by the walks it participates in"

Learning from Context:

- We learn node embeddings by predicting which nodes appear together in random walks
- Nodes with similar contexts (appear in similar walks) develop similar embeddings
- This naturally captures community structure and functional similarity

Beyond Random Walk Methods (out of scope)

While random walk approaches are popular, several other major classes of embedding methods exist:

1. **Matrix Factorization-Based**:
    - Directly decompose graph matrices (adjacency, Laplacian, etc.)
    - Example approach: Graph Factorization, HOPE, GraRep
    - Key idea: Learn embeddings such that $ZZ^T \approx S$, where S is a similarity matrix
2. **Spectral Methods**:
    - Based on eigendecomposition of graph Laplacian matrices
    - Preserve global graph properties
    - Key idea: Embeddings are eigenvectors corresponding to smallest non-zero eigenvalues

3. **Deep Autoencoder Approaches**:
    - Use autoencoders to compress adjacency or feature information
    - Learn embeddings by reconstructing neighborhood information
    - Examples: SDNE (Structural Deep Network Embedding), DNGR
4. **Direct Optimization Methods**:
    - Optimize embedding objectives directly without random sampling
    - Often involve explicit preservation of first/second-order proximities
    - Example: LINE (Large-scale Information Network Embedding)

These methods present alternative perspectives on capturing graph structure in embeddings.

Random-Walk Based Methods: Practical Approaches

Evolution of Random Walk Embedding Algorithms

- The theory of using random walks for embeddings led to several practical algorithms
- These methods differ primarily in **how they generate walks** and define neighborhoods
- Major approaches include:
  - **DeepWalk** (Perozzi et al., 2014): First method to use random walks + skip-gram
  - **node2vec** (Grover & Leskovec, 2016): Advanced walks with controllable exploration strategy
  - Others: LINE, HARP, VERSE (each with different neighborhood definitions)

DeepWalk

- **Origin**: Published by Perozzi et al. in 2014, the first major graph embedding approach
- **Core idea**: Nodes that appear in similar contexts in random walks should have similar embeddings
- **Inspiration**: Adapts techniques from natural language processing (Word2Vec) to graph data
- **Analogy**: Treats random walks on graphs as "sentences" and nodes as "words"
- **Reference**: DeepWalk: Online Learning of Social Representations (KDD 2014)

DeepWalk: Detailed Step-by-Step Process

Step 1: Generate Random Walks

- For each node v in the graph:
  - Start a walk from node v
  - For L steps, randomly move to a neighbor with equal probability $\frac{1}{degree(v)}$
  - Repeat this process r times per node (typically $r = 10$ walks of length $L = 80$)
- Result: Collection of node sequences $(v_0, v_1, v_2, ..., v_L)$ for each starting node

Step 2: Define Context Windows

- For each position i in each walk:
  - Define a window of size w around position i
  - Consider nodes v_{i-w} to v_{i+w} as the "context" of node v_i
  - This creates node-context pairs used for training

UNIVERSITÀ DI TORINO

DeepWalk: The Learning Process

Step 3: Apply Skip-gram Model

- For each node v and its context nodes in the random walks:
  - Initialize random embedding vectors for all nodes
  - Update embeddings to maximize the probability of predicting context nodes:

$$\max_{Z} \sum_{u \in V} \sum_{v \in N_R(u)} \log P(v|\mathbf{z}_u)$$

  - Where $P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_v^T \mathbf{z}_u)}{\sum_{n \in V} \exp(\mathbf{z}_n^T \mathbf{z}_u)}$

Step 4: Optimization in Practice

- Use stochastic gradient descent to update embeddings
- Apply negative sampling to make computation feasible
- Final result: Each node has a d-dimensional embedding vector

DeepWalk: Strengths and Limitations

Advantages:

- **Simplicity**: Easy to understand and implement
- **Scalability**: Works well on large graphs through sampling
- **No feature engineering**: Learns representations directly from graph structure
- **Versatility**: Performs reasonably well across different types of networks
- **Efficient training**: Random walks + negative sampling make training feasible

Limitations:

- **Uniform exploration only**: No control over how the network is explored
- **Cannot distinguish edge types**: Treats all edges as equivalent
- **No use of node/edge features**: Only uses the graph structure
- **Transductive learning**: Cannot generalize to unseen nodes
- **Static embeddings**: Not designed for dynamic/evolving graphs
- **Limited control**: Few parameters to tune for different graph structures

UNIVERSITÀ DI TORINO

node2vec

Key Innovations

- **Origin**: Developed by Grover & Leskovec (2016) to address DeepWalk's limitations
- **Core innovation**: Biased random walks with controllable exploration parameters
- **Reference**: node2vec: Scalable Feature Learning for Networks (KDD 2016)

Two Network Properties node2vec Can Capture

1. **Homophily (Community Structure)**
   - Nodes in the same community should be similar (like social groups)
   - Dense clusters of connections
   - Captured by BFS-like, local exploration

UNIVERSITÀ DI TORINO

node2vec (2)

2. **Structural Equivalence (Node Roles)**
    - Nodes with similar roles should be similar, regardless of distance
    - Example: Hub airports in different countries have similar connection patterns
    - Captured by DFS-like, broader exploration

node2vec: The Algorithm

Walk Control Parameters:

- **Return parameter** p: Controls likelihood of revisiting nodes
  - $p < 1$: Favors revisiting (stays in communities)
  - $p > 1$: Avoids backtracking (explores outward)
- **In-out parameter** q: Controls exploration strategy
  - $q < 1$: Favors outward exploration (good for structural roles)
  - $q > 1$: Keeps walks local (good for communities)

Process:

1. Define parameters p and q based on your task
2. For each node, run biased random walks using transition probabilities
3. Apply Skip-gram model to walks (same as DeepWalk)

The Biased Walk Strategy

For current node v, previous node t, and potential next node x:

$$P(x|v,t) \propto \begin{cases} \frac{1}{p} \cdot w_{vx} & \text{if } d_{tx} = 0 \text{ (return to previous node)} \\ 1 \cdot w_{vx} & \text{if } d_{tx} = 1 \text{ (move to common neighbor)} \\ \frac{1}{q} \cdot w_{vx} & \text{if } d_{tx} = 2 \text{ (explore outward)} \end{cases}$$

Special cases:

- When $p = q = 1$: Equivalent to DeepWalk (uniform walks)
- When $p < 1, q > 1$: Favors homophily/community structure
- When $p > 1, q < 1$: Favors structural equivalence

Choosing Strategy Based on Task

Task	Parameter Setting	Walk Behavior
Community detection	$p = 0.25, q = 2$	Stay local within communities
Role discovery	$p = 2, q = 0.25$	Explore broadly across network
Link prediction	Task-dependent	Use grid search to optimize

Advantage over DeepWalk:

The flexibility to tune exploration strategy makes node2vec adaptable to different graph structures and tasks, addressing DeepWalk's "one-size-fits-all" limitation.

Beyond Node Embeddings: Representing Entire Graphs

Use Case: Molecule Classification

- Each molecule is a graph
- Need to classify as toxic/non-toxic, reactive/non-reactive, etc.
- **Challenge**: How to create fixed-size representations for variable-sized graphs?

Beyond Node Embeddings: Representing Entire Graphs (2)

Method 1: Aggregate Node Embeddings

1. Compute node embeddings for all atoms
2. Pool them: $\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$ or $\mathbf{z}_G = \frac{1}{|V|} \sum_{v \in G} \mathbf{z}_v$

Method 2: Virtual Supernode

1. Add a dummy node connected to all graph nodes
2. Learn embedding for this supernode
3. Use the supernode's embedding as the graph representation

Advanced Graph-Level Embedding Methods (out of scope)

Beyond simple aggregation, several methods capture graph-level properties:

1. **Hierarchical Pooling**:
   - Progressively coarsen the graph while computing embeddings
   - $Z^{(l+1)} = POOL(Z^{(l)}, A^{(l)})$
   - Creates a multi-resolution representation of the graph
2. **Graph Kernels**:
   - Compare substructures (graphlets, random walks, subtrees)
   - Example: Weisfeiler-Lehman kernel counts labeled subtrees
   - Creates implicit embeddings via kernel matrix
3. **Self-Supervised Methods**:
   - Learn graph-level representations by predicting graph properties
   - Examples: predicting graph statistics, graph reconstruction
   - Can leverage both structure and node features

Limitations

1. Transductive Learning Only

- Cannot generalize to unseen nodes
- Need to retrain for new nodes

2. Limited Structural Information

- Shallow encoders miss some structural patterns
- Hard to capture higher-order network properties

3. Ignore Node/Edge Features

- Many networks have rich metadata
- Pure structural embeddings waste this information

Graph Neural Networks: Beyond Shallow Embeddings

- **Richer information processing**: Can leverage both graph structure AND node/edge features
  - Example: In social networks, combine friendship links WITH user profiles and content
  - Critical for domains with complex attributes (molecules, knowledge graphs)
- **Adaptability to dynamic environments**: Can handle evolving graphs and new nodes
  - No need to retrain the entire model when the network changes
  - Essential for real-time applications (recommendation systems, fraud detection)
- **Better performance on complex patterns**:
  - Capture higher-order dependencies through message-passing architecture
  - Multiple layers allow learning hierarchical patterns in the data
  - Similar to how ConvNets revolutionized image analysis
- **Scalability to massive networks**: Many architectures designed specifically for web-scale graphs
  - Companies like Pinterest, Twitter, and Google use GNNs in production systems

Summary: The Power of Representation Learning

Key Takeaways

1. Representation learning automates feature extraction from graphs
2. Random walks provide an efficient way to define node similarity
3. The encoder-decoder framework unifies various embedding approaches
4. Learned embeddings outperform hand-crafted features in many tasks

UNIVERSITÀ DI TORINO

Q&A