Graph Representation Learning: An Overview

CSCI 699: ML4Know
Instructor: Xiang Ren
USC Computer Science
This Lecture

1) Node embeddings
   - Map nodes to low-dimensional embeddings.

2) Graph neural networks
   - Deep learning architectures for graph-structured data
Part 1: Node Embeddings
Embedding Nodes

Intuition: Find embedding of nodes to $d$-dimensions so that “similar” nodes in the graph have embeddings that are close together.
Setup

• Assume we have a graph $G$:
  • $V$ is the vertex set.
  • $A$ is the adjacency matrix (assume binary).
  • No node features or extra information is used!
Embedding Nodes

• Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the original network.
Embedding Nodes

Goal: \( \text{similarity}(u, v) \approx z_v^T z_u \)

Need to define!
Learning Node Embeddings

1. Define an encoder (i.e., a mapping from nodes to embeddings)

2. Define a node similarity function (i.e., a measure of similarity in the original network).

3. Optimize the parameters of the encoder so that:

\[
\text{similarity}(u, v) \approx z_v^T z_u
\]
Two Key Components

- **Encoder** maps each node to a low-dimensional vector.

\[
\text{ENC}(v) = z_v
\]

- **Similarity function** specifies how relationships in vector space map to relationships in the original network.

\[
\text{similarity}(u, v) \approx z_v^\top z_u
\]
“Shallow” Encoding

• Simplest encoding approach: encoder is just an embedding-lookup

\[ \text{ENC}(v) = Zv \]

\[ Z \in \mathbb{R}^{d \times |V|} \] matrix, each column is node embedding [what we learn!]

\[ v \in \mathbb{I}^{|V|} \] indicator vector, all zeroes except a one in column indicating node \( v \)
“Shallow” Encoding

• Simplest encoding approach: encoder is just an embedding-lookup

\[ Z = \text{embedding matrix} \]

embedding vector for a specific node

Dimension/size of embeddings

one column per node
“Shallow” Encoding

• Simplest encoding approach: **encoder is just an embedding-lookup.**

  i.e., each node is assigned a unique embedding vector.

• E.g., node2vec, DeepWalk, LINE
“Shallow” Encoding

• Simplest encoding approach: encoder is just an embedding-lookup.

• We will focus on shallow encoding in this section…

• In the next section we will discuss more encoders based on deep neural networks.
How to Define Node Similarity?

• Key distinction between “shallow” methods is how they define node similarity.
• E.g., should two nodes have similar embeddings if they….  
  • are connected?
  • share neighbors?
  • have similar “structural roles”? 
  • …?
Multi-hop Similarity

Material based on:
- Ou et al. 2016. *Asymmetric Transitivity Preserving Graph Embedding*. *KDD*.
Multi-hop Similarity

**Idea:** Consider k-hop node neighbors.
- E.g., two or three-hop neighbors.

- **Red:** Target node
- **Green:** 1-hop neighbors
  - \(A\) (i.e., adjacency matrix)
- **Blue:** 2-hop neighbors
  - \(A^2\)
- **Purple:** 3-hop neighbors
  - \(A^3\)
Multi-hop Similarity

• Basic idea:

\[
\mathcal{L} = \sum_{(u,v) \in V \times V} \|z_u^T z_v - A^k_{u,v}\|^2
\]

• Train embeddings to predict k-hop neighbors.

• In practice (GraRep from Cao et al, 2015):
  • Use log-transformed, probabilistic adjacency matrix:

\[
\tilde{A}^k_{i,j} = \max \left( \log \left( \frac{(A_{i,j} / d_i)^k}{\sum_{l \in V} (A_{l,j} / d_l)^k} \right) - \alpha, 0 \right)
\]

  • Train multiple different hop lengths and concatenate output.
Multi-hop Similarity

• Another option: Measure overlap between node neighborhoods.

• Example overlap functions:
  • Jaccard similarity
  • Adamic-Adar score
Multi-hop Similarity

\[ \mathcal{L} = \sum_{(u, v) \in V \times V} \| Z_u^\top Z_v - S_{u,v} \|^2 \]

- \( S_{u,v} \) is the neighborhood overlap between \( u \) and \( v \) (e.g., Jaccard overlap or Adamic-Adar score).
- This technique is known as \textbf{HOPE (Yan et al., 2016)}.
Summary so far

• Basic idea so far:
  • 1) Define pairwise node similarities.
  • 2) Optimize low-dimensional embeddings to approximate these pairwise similarities.

• Issues:
  • Expensive: Generally $O(|V|^2)$, since we need to iterate over all pairs of nodes.
  • Brittle: Must hand-design deterministic node similarity measures.
  • Massive parameter space: $O(|V|)$ parameters
Random Walk Approaches

Material based on:

- Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. KDD.
- Grover et al. 2016. node2vec: Scalable Feature Learning for Networks. KDD.
Random-walk Embeddings

\[ z_u^T z_v \approx \text{probability that } u \text{ and } v \text{ co-occur on a random walk over the network} \]
Random-walk Embeddings

1. Estimate probability of visiting node \( v \) on a random walk starting from node \( u \) using some random walk strategy \( R \).

2. Optimize embeddings to encode these random walk statistics.
Why Random Walks?

1. **Expressivity**: Flexible stochastic definition of node similarity that incorporates both local and higher-order neighborhood information.

2. **Efficiency**: Do not need to consider all node pairs when training; only need to consider pairs that co-occur on random walks.
Random Walk Optimization

1. Run short random walks starting from each node on the graph using some strategy $R$.

2. For each node $u$ collect $N_R(u)$, the multiset* of nodes visited on random walks starting from $u$.

3. Optimize embeddings to according to:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} - \log(P(v | z_u))$$

* $N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks.
Random Walk Optimization

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} - \log(P(v|z_u))$$

- **Intuition:** Optimize embeddings to maximize likelihood of random walk co-occurrences.

- **Parameterize** $P(v|z_u)$ using **softmax**:

$$P(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}$$
Random Walk Optimization

Putting things together:

\[ \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} - \log \left( \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \right) \]

- Sum over all nodes \( u \)
- Sum over nodes \( v \) seen on random walks starting from \( u \)
- Predicted probability of \( u \) and \( v \) co-occurring on random walk

Optimizing random walk embeddings = Finding embeddings \( z_u \) that minimize \( \mathcal{L} \)
Random Walk Optimization

But doing this naively is too expensive!!

\[
\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} - \log \left( \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_n^T z_n)} \right)
\]

Nested sum over nodes gives \(O(|V|^2)\) complexity!!
Random Walk Optimization

But doing this naively is too expensive!!

\[ \mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} - \log \left( \frac{\exp(z_u^\top z_v)}{\sum_{n \in V} \exp(z_u^\top z_n)} \right) \]

The normalization term from the softmax is the culprit... can we approximate it?
Negative Sampling

Solution: Negative sampling

\[
\log \left( \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \right)
\]

\[
\approx \log(\sigma(z_u^T z_v)) - \sum_{i=1}^{k} \log(\sigma(z_u^T z_{n_i})), n_i \sim P_V
\]

i.e., instead of normalizing w.r.t. all nodes, just normalize against k random “negative samples”
Part 2: Graph Neural Networks
From “Shallow” to “Deep”

- Limitations of shallow encoding:
  - \(O(|V|)\) parameters are needed: there is no parameter sharing and every node has its own unique embedding vector.
  - Inherently “transductive”: It is impossible to generate embeddings for nodes that were not seen during training.
  - Do not incorporate node features: Many graphs have features that we can and should leverage.
From “Shallow” to “Deep”

• We will now discuss “deeper” methods based on graph neural networks.

\[ \text{ENC}(v) = \text{complex function that depends on graph structure.} \]

• In general, all of these more complex encoders can be combined with the similarity functions from the previous section.
The Basics: Graph Neural Networks

Based on material from:

Setup

• Assume we have a graph $G$:
  • $V$ is the vertex set.
  • $A$ is the adjacency matrix (assume binary).
  • $X \in \mathbb{R}^{m \times |V|}$ is a matrix of node features.
    • Categorical attributes, text, image data
      • E.g., profile information in a social network.
    • Node degrees, clustering coefficients, etc.
    • Indicator vectors (i.e., one-hot encoding of each node)
Neighborhood Aggregation

- **Key idea:** Generate node embeddings based on local neighborhoods.
Neighborhood Aggregation

• **Intuition:** Nodes aggregate information from their neighbors using neural networks
Neighborhood Aggregation

- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- “layer-0” embedding of node $u$ is its input feature, i.e. $x_u$. 

```
INPUT GRAPH
```

```
TARGET NODE
```

```
Layer-0
```

```
Layer-1
```

```
Layer-2
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Neighborhood Aggregation

• Key distinctions are in how different approaches aggregate information across the layers.

what’s in the box?
Neighborhood Aggregation

- **Basic approach:** Average neighbor information and apply a neural network.

1) average messages from neighbors

2) apply neural network
The Math

- **Basic approach:** Average neighbor messages and apply a neural network.

\[
 h_v^0 = x_v
\]

Initial “layer 0” embeddings are equal to node features

\[
 h_v^k = \sigma \left( \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + B_k h_v^{k-1} \right), \quad \forall k > 0
\]

- \(k\)th layer embedding of \(v\)
- non-linearity (e.g., ReLU or tanh)
- average of neighbor’s previous layer embeddings
- previous layer embedding of \(v\)
Training the Model

- How do we train the model to generate “high-quality” embeddings?

Need to define a loss function on the embeddings, $\mathcal{L}(z_u)$!
Training the Model

After $K$-layers of neighborhood aggregation, we get output embeddings for each node.

We can feed these embeddings into any loss function and run stochastic gradient descent to train the aggregation parameters.
Training the Model

• Train in an **unsupervised manner** using only the graph structure.

• Unsupervised loss function can be anything from the last section, e.g., based on
  • Random walks (node2vec, DeepWalk)
  • Graph factorization
  • i.e., train the model so that “similar” nodes have similar embeddings.
Training the Model

• **Alternative**: Directly train the model for a supervised task (e.g., node classification):

  - e.g., an online social network
Training the Model

• **Alternative**: Directly train the model for a supervised task (e.g., node classification):

\[ \mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v \theta)) + (1 - y_v) \log(1 - \sigma(z_v \theta)) \]
Overview of Model Design

1) Define a neighborhood aggregation function.

2) Define a loss function on the embeddings, $\mathcal{L}(z_u)$
Overview of Model Design

3) Train on a set of nodes, i.e., a batch of compute graphs
Overview of Model

4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!
Inductive Capability

- The same aggregation parameters are shared for all nodes.
- The number of model parameters is sublinear in $|V|$ and we can generalize to unseen nodes!
Inductive Capability

Inductive node embedding          generalize to entirely unseen graphs

train on one graph

generalize to new graph

e.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B
Graph Convolutional Networks

Based on material from:
• Kipf et al., 2017. Semisupervised Classification with Graph Convolutional Networks. ICLR.
Graph Convolutional Networks

• Kipf et al.’s **Graph Convolutional Networks (GCNs)** are a slight variation on the neighborhood aggregation idea:

\[
    h^k_v = \sigma \left( W_k \sum_{u \in N(v) \cup v} \frac{h^{k-1}_u}{\sqrt{|N(u)||N(v)|}} \right)
\]
Graph Convolutional Networks

**Basic Neighborhood Aggregation**

\[
\mathbf{h}_v^k = \sigma \left( \mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1} \right)
\]

**GCN Neighborhood Aggregation**

\[
\mathbf{h}_v^k = \sigma \left( \mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}} \right)
\]

- same matrix for self and neighbor embeddings
- per-neighbor normalization
Graph Convolutional Networks

• Empirically, they found this configuration to give the best results.
  • More parameter sharing.
  • Down-weights high degree neighbors.

\[
h^k_v = \sigma \left( W^k \sum_{u \in N(v) \cup v} \frac{h^{k-1}_u}{\sqrt{|N(u)||N(v)|}} \right)
\]

use the same transformation matrix for self and neighbor embeddings

instead of simple average, normalization varies across neighbors
Summary

• **Graph convolutional networks**
  • Average neighborhood information and stack neural networks.

*(Not covered in lecture)*

• **GraphSAGE**
  • Generalized neighborhood aggregation.

• **Gated Graph Neural Networks**
  • Neighborhood aggregation + RNNs
Recent advances in graph neural nets

- **Attention-based neighborhood aggregation:**
  - Graph Attention Networks ([Velickovic et al., 2018](#))
  - GeniePath ([Liu et al., 2018](#))

- **Generalizations based on spectral convolutions:**
  - Geometric Deep Learning ([Bronstein et al., 2017](#))
  - Mixture Model CNNs ([Monti et al., 2017](#))

- **Speed improvements via subsampling:**
  - FastGCNs ([Chen et al., 2018](#))
  - Stochastic GCNs ([Chen et al., 2017](#))