Graph Representation Learning

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

http://cs224w.stanford.edu

Machine Learning in Networks

Node classification

Example: Link Prediction

Machine Learning Lifecycle

- **(Supervised) Machine Learning Lifecycle requires feature engineering every single time!**

Feature Learning in Graphs

Goal: Efficient task-independent feature learning for machine learning with graphs!

Why network embedding?

- **Task: We map each node in a network into a low-dimensional space**
  - Distributed representations for nodes
  - Similarity of embeddings between nodes indicates their network similarity
  - Encode network information and generate node representation

Example Node Embedding

- **2D embeddings of nodes of the Zachary's Karate Club network:**

Input

Output

Image from: Perozzi et al. DeepWalk: Online Learning of Social Representations. *KDD 2014.*

Why Is It Hard?

- Modern deep learning toolbox is designed for simple sequences or grids.
  - CNNs for fixed-size images/grids….

  - RNNs or word2vec for text/sequences…

Why Is It Hard?

- But networks are far more complex!
  - Complex topographical structure (i.e., no spatial locality like grids)

  - No fixed node ordering or reference point (i.e., the isomorphism problem)
  - Often dynamic and have multimodal features.

Embedding Nodes

Setup

- **Assume we have a graph G:**
  - V is the vertex set.
  - $\mathbf{A}$ 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: $\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$

in the original network

Similarity of the embedding

Need to define!

original network **embedding space**

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:**

$$\underbrace{\text{similarity}(u, v)}_{\text{in the original network}} \approx \underbrace{\mathbf{z}_v^\top \mathbf{z}_u}_{\text{Similarity of the embedding}}$$

Two Key Components

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

$$\text{ENC}(v) = \mathbf{z}_v$$

$\mathbf{d}$-dimensional embedding

node in the input graph

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

$$\text{similarity}(u, v) \approx \mathbf{z}_v^\top \mathbf{z}_u$$

Similarity of u and v in the original network

dot product between node embeddings

"Shallow" Encoding

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

$$\text{ENC}(v) = \mathbf{Z}\mathbf{v}$$

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

$\mathbf{v} \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node *v*

"Shallow" Encoding

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

"Shallow" Encoding

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

Each node is assigned to a unique embedding vector

Many methods: DeepWalk, node2vec, TransE

How to Define Node Similarity?

- Key choice of 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"?
  - …?

Random Walk Approaches to Node Embeddings

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

Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**, and move to this neighbor; then we select a neighbor of this point at random, and move to it, etc. The (random) sequence of points selected this way is a **random walk on the graph**.

Random-walk Embeddings

$$\mathbf{z}_u^\top \mathbf{z}_v \approx$$

probability that u and v 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**

$P_R(v|u)$

2. **Optimize embeddings to encode these random walk statistics:**

Similarity (here: dot product=cos(θ)) encodes random walk "similarity"

$\theta \propto P_R(v|u)$

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

Unsupervised Feature Learning

- Intuition: Find embedding of nodes to d-dimensions that preserves similarity
- Idea: Learn node embedding such that nearby nodes are close together in the network
- Given a node u, how do we define nearby nodes?
  - $N_R(u)$ … neighbourhood of u obtained by some strategy R

Feature Learning as Optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $z: u \to \mathbb{R}^d$.

- Log-likelihood objective:

$$\max_{\mathbf{z}} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) \mid z_u)$$

  - where $N_R(u)$ is neighborhood of node u by strategy R

- Given node u, we want to learn feature representations that are predictive of the nodes in its neighborhood $N_{\mathrm{R}}(u)$

Random Walk Optimization

1. Run **short fixed-length 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 according to: Given node u, predict its neighbors $N_{\mathrm{R}}(u)$

$$\max_{\mathrm{z}} \sum_{u \in V} \log \mathrm{P}(N_{\mathrm{R}}(u) \mid z_u)$$

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

Random Walk Optimization

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

- **Intuition:** Optimize embeddings to maximize likelihood of random walk co-occurrences
- **Parameterize $P(v|\mathbf{z}_u)$ using softmax:**

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

Why softmax?
We want node v to be most similar to node u (out of all nodes n).
Intuition: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

Random Walk Optimization

Putting it all together:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

Random Walk Optimization

But doing this naively is too expensive!!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

Nested sum over nodes gives $O(|V|^2)$ complexity!

Random Walk Optimization

But doing this naively is too expensive!!

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

The normalization term from the softmax is the culprit… can we approximate it?

Negative Sampling

- **Solution:** Negative sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

sigmoid function
(makes each term a "probability" between 0 and 1)

random distribution over all nodes

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_i sampled from background distribution P_v.

More at https://arxiv.org/pdf/1402.3722.pdf

Instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**" n_i

Negative Sampling

$$\log\left(\frac{\exp(\mathbf{z}_u^\top \mathbf{z}_v)}{\sum_{n\in V}\exp(\mathbf{z}_u^\top \mathbf{z}_n)}\right)$$

random distribution over all nodes

$$\approx \log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \sum_{i=1}^{k}\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_{n_i})), n_i \sim P_V$$

- Sample k negative nodes proportional to degree
- Two considerations for k (# negative samples):
  1. Higher k gives more robust estimates
  2. Higher k corresponds to higher bias on negative events

  In practice k =5-20

Random Walks: Stepping Back

1. Run **short fixed-length** random walks starting from each node on the graph using some strategy R.

2. For each node u collect $\mathrm{N}_R(u)$, the multiset of nodes visited on random walks starting from u

3. Optimize embeddings using Stochastic Gradient Descent:

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

We can efficiently approximate this using negative sampling!

How should we randomly walk?

- So far we have described how to optimize embeddings given random walk statistics
- **What strategies should we use to run these random walks?**
  - Simplest idea: **Just run fixed-length, unbiased random walks starting from each node** (i.e., DeepWalk from Perozzi et al., 2013).
    - The issue is that such notion of similarity is too constrained
  - How can we generalize this?

Overview of node2vec

- **Goal:** Embed nodes with similar network neighborhoods close in the feature space
- We frame this goal as a maximum likelihood optimization problem, independent to the downstream prediction task
- **Key observation:** Flexible notion of network neighborhood $N_R(u)$ of node u leads to rich node embeddings
- Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

node2vec: Biased Walks

Idea: use flexible, biased random walks that can trade off between **local** and **global** views of the network (Grover and Leskovec, 2016).

node2vec: Biased Walks

Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:

Walk of length 3 ($N_R(u)$ of size 3):

$N_{BFS}(u) = \{ s_1, s_2, s_3 \}$ Local microscopic view

$N_{DFS}(u) = \{ s_4, s_5, s_6 \}$ Global macroscopic view

BFS vs. DFS

BFS:
Micro-view of neighbourhood

DFS:
Macro-view of neighbourhood

Interpolating BFS and DFS

Biased fixed-length random walk R that given a node u generates neighborhood $N_R(u)$

- Two parameters:
  - **Return parameter p:**
    - Return back to the previous node
  - **In-out parameter q:**
    - Moving outwards (DFS) vs. inwards (BFS)
    - Intuitively, q is the "ratio" of BFS vs. DFS

Biased Random Walks

Biased 2nd-order random walks explore network neighborhoods:

- Rnd. walk just traversed edge (s_1, w) and is now at w
- **Insight:** Neighbors of w can only be:

Idea: Remember where that walk came from

Biased Random Walks

- Walker came over edge (s_1, w) and is at w. Where to go next?

$1/p, 1/q, 1$ are unnormalized probabilities

- p, q model transition probabilities
  - p … return parameter
  - q … "walk away" parameter

Biased Random Walks

- Walker came over edge (s_1, w) and is at w. Where to go next?

$w \rightarrow$

Target t	Prob.	Dist. (s_1, t)
s_1	$1/p$	0
s_2	1	1
s_3	$1/q$	2
s_4	$1/q$	2

Unnormalized transition prob. segmented based on distance from s_1

- **BFS-like** walk: Low value of p
- **DFS-like** walk: Low value of q

$N_R(u)$ are the nodes visited by the biased walk

node2vec algorithm

- 1) Compute random walk probabilities
- 2) Simulate r random walks of length l starting from each node u
- 3) Optimize the node2vec objective using Stochastic Gradient Descent

Linear-time complexity

All 3 steps are individually parallelizable

How to Use Embeddings

- **How to use embeddings z_i of nodes:**
  - **Clustering/community detection:** Cluster points z_i
  - **Node classification:** Predict label $f(z_i)$ of node i based on z_i
  - **Link prediction:** Predict edge (i, j) based on $f(z_i, z_j)$
    - Where we can: concatenate, avg, product, or take a difference between the embeddings:
      - Concatenate: $f(z_i, z_j) = g([z_i, z_j])$
      - Hadamard: $f(z_i, z_j) = g(z_i * z_j)$ (per coordinate product)
      - Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$
      - Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$

Summary so far

- **Basic idea:** Embed nodes so that distances in embedding space reflect node similarities in the original network.
- **Different notions of node similarity:**
  - Adjacency-based (i.e., similar if connected)
  - Multi-hop similarity definitions
  - Random walk approaches **(covered today)**

Summary so far

- **So what method should I use..?**
- No one method wins in all cases….
  - E.g., node2vec performs better on node classification while multi-hop methods performs better on link prediction (Goyal and Ferrara, 2017 survey)
- Random walk approaches are generally more efficient
- **In general:** Must choose definition of node similarity that matches your application!

An Application of Embeddings to the Knowledge Graph:

Translating Embeddings for Modeling Multi-relational Data

Bordes, Usunier, Garcia-Duran. NeurIPS 2013.

Knowledge Graph

Pierre-Yves Vandenbussche

A **knowledge graph** is composed of facts/statements about inter-related entities

In KGs, edges can be of many types!

Nodes are referred to as **entities**, edges as **relations**

KG Completion (Link Prediction)

KG **incompleteness** can substantially affect the efficiency of systems relying on it!

INTUITION: we want a link prediction model that learns from local and global connectivity patterns in the KG, taking into account entities and relationships of different types at the same time.

DOWNSTREAM TASK: relation predictions are performed by using the learned patterns to generalize observed relationships between an entity of interest and all the other entities.

Translating Embeddings

Translating Embeddings

- In **TransE**, relationships between entities are represented as triplets
  - $\boldsymbol{h}$ (head entity), $\boldsymbol{l}$ (relation), $\boldsymbol{t}$ (tail entity) => (h, l, t)
- Entities are first embedded in an entity space R^k
  - similarly to the previous methods
- Relations are represented as **translations**
  - $h + l \approx t$ if the given fact is true
  - else, $h + l \neq t$

TransE Learning Algorithm

Algorithm 1 Learning TransE

input Training set $S = \{(h, \ell, t)\}$, entities and rel. sets E and L, margin γ, embeddings dim. k.

initialize $\boldsymbol{\ell} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each $\ell \in L$
$\boldsymbol{\ell} \leftarrow \boldsymbol{\ell} / \|\boldsymbol{\ell}\|$ for each $\ell \in L$
$\mathbf{e} \leftarrow \text{uniform}(-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}})$ for each entity $e \in E$

Entities and relations are initialized uniformly, and normalized

loop
  $\mathbf{e} \leftarrow \mathbf{e} / \|\mathbf{e}\|$ for each entity $e \in E$
  $S_{batch} \leftarrow \text{sample}(S, b)$ // sample a minibatch of size b
  $T_{batch} \leftarrow \emptyset$ // initialize the set of pairs of triplets
  for $(h, \ell, t) \in S_{batch}$ **do**
    $(h', \ell, t') \leftarrow \text{sample}(S'_{(h,\ell,t)})$ // sample a corrupted triplet

Negative sampling with triplet that does not appear in the KG

    $T_{batch} \leftarrow T_{batch} \cup \{((h, \ell, t), (h', \ell, t'))\}$
  end for
  Update embeddings w.r.t. $\sum_{((h,\ell,t),(h',\ell,t')) \in T_{batch}} \nabla \left[\gamma + d(\boldsymbol{h} + \boldsymbol{\ell}, \boldsymbol{t}) - d(\boldsymbol{h'} + \boldsymbol{\ell}, \boldsymbol{t'})\right]_+$

positive sample

negative sample

end loop

Comparative loss: favors lower distance values for valid triplets, high distance values for corrupted ones

Embedding Entire Graphs

Embedding Entire Graphs

- **Goal:** Want to embed an entire graph G

- **Tasks:**
  - Classifying toxic vs. non-toxic molecules
  - Identifying anomalous graphs

Approach 1

Simple idea:

- Run a standard graph embedding technique *on* the (sub)graph G
- Then just sum (or average) the node embeddings in the (sub)graph G

$$z_G = \sum_{v \in G} z_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure

Approach 2

- **Idea:** Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique

- Proposed by Li et al., 2016 as a general technique for subgraph embedding

Approach 3: Anonymous Walk Embeddings

States in **anonymous walk** correspond to the index of the first time we visited the node in a random walk

Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

Number of Walks Grows

Number of anonymous walks grows exponentially:

- There are 5 anon. walks a_i of length 3:
  a_1=111, a_2=112, a_3= 121, a_4= 122, a_5= 123

Idea 1 of Anonymous Walks

- Enumerate all possible anonymous walks a_i of l steps and record their counts
- Represent the graph as a probability distribution over these walks

- **For example:**
  - Set $l = 3$
  - Then we can represent the graph as a 5-dim vector
    - Since there are 5 anonymous walks a_i of length 3: 111, 112, 121, 122, 123
  - $Z_G[i]$ = probability of anonymous walk a_i in G

Idea 2: Sample Anonymous Walks

- Complete counting of all anonymous walks in a large graph may be infeasible
- **Sampling approach to approximating the true distribution:** Generate independently a set of m random walks and calculate its corresponding empirical distribution of anonymous walks
- How many random walks m do we need?
  - We want the distribution to have error of more than ε with prob. less than δ:

$$m = \left\lceil \frac{2}{\varepsilon^2}\left(\log(2^\eta - 2) - \log(\delta)\right) \right\rceil$$

where: η is the total number of anon. walks of length l.

For example:
There are $\eta = 877$ anonymous walks of length $l = 7$. If we set $\varepsilon = 0.1$ and $\delta = 0.01$ then we need to generate m=122500 random walks

Idea 3: Learn Walk Embeddings

Learn embedding z_i of every anonymous walk a_i

- The embedding of a graph G is then sum/avg/concatenation of walk embeddings z_i

How to embed walks?

- **Idea:** Embed walks s.t. next walk can be predicted
  - Set z_i s.t. we maximize $P(w_t^u | w_{t-\Delta}^u, \dots, w_t^u) = f(z)$
    - Where w_t^u is a t-th random walk starting at node u

Idea 3: Learn Walk Embeddings

- **Run T different random walks from u each of length l:**
$$N_R(u) = \{w_1^u, w_2^u \dots w_T^u\}$$
  - Let a_i be its anonymous version of walk w_i

- **Learn to predict walks that co-occur in Δ-size window**
- Estimate embedding z_i of anonymous walk a_i of w_i:
$$\max \frac{1}{T} \sum_{t=\Delta}^{T} \log P(w_t | w_{t-\Delta}, \dots, w_{t-1})$$

where: Δ… context window size

- $P(w_t | w_{t-\Delta}, \dots, w_{t-1}) = \frac{\exp(y(w_t))}{\sum_i^{\eta} \exp(y(w_i))}$
- $y(w_t) = b + U \cdot \left(\frac{1}{\Delta} \sum_{i=1}^{\Delta} z_i\right)$
  - where $b \in \mathbb{R}$, $U \in \mathbb{R}^D$, z_i is the embedding of a_i (anonymized version of walk w_i)

Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

Summary

We discussed 3 ideas to graph embeddings

- **Approach 1:** Embed nodes and sum/avg them
- **Approach 2:** Create super-node that spans the (sub) graph and then embed that node
- **Approach 3:** Anonymous Walk Embeddings
  - Idea 1: Represent the graph via the distribution over all the anonymous walks
  - Idea 2: Sample the walks to approximate the distribution
  - Idea 3: Embed anonymous walks