Representation Learning with Networks

Jure Leskovec (@jure)
Stanford University
Quickly about me

- Assoc. Prof. CS Stanford
- Chief Scientist at Pinterest
- Research group:
  - ML in graphs, social networks, data science, computational social science
  - 12 PhD students, 6 Postdocs
  - 2 Staff, 10 MS and BS students
- http://snap.stanford.edu
Why networks?

Networks are a general language for describing and modeling complex systems.
Network!
Media & Information
World economy
Roads
Human cell
Complex social, technological, and biological systems represented as networks of interconnected entities
Many Data are Networks

Social networks

Economic networks

Biomedical networks

Information networks: Web & citations

Internet

Networks of neurons

Figure 3: Higher-order cluster in the C. elegans neuronal network (28). A: The 4-node "bi-fan" motif, which is over-expressed in the neuronal networks (1). Intuitively, this motif describes a cooperative propagation of information from the nodes on the left to the nodes on the right.

B: The best higher-order cluster in the C. elegans frontal neuronal network based on the motif in (A). The cluster contains three ring motor neurons (RMEL/V/R; cyan) with many outgoing connections, serving as the source of information; six inner labial sensory neurons (IL2DL/VR/R/DR/VL; orange) with many incoming connections, serving as the destination of information; and four URA neurons (purple) acting as intermediaries. These RME neurons have been proposed as pioneers for the nerve ring (20), while the IL2 neurons are known regulators of nictation (21), and the higher-order cluster exposes their organization. The cluster also reveals that RIH serves as a critical intermediary of information processing. This neuron has incoming links from all three RME neurons, outgoing connections to five of the six IL2 neurons, and the largest total number of connections of any neuron in the cluster.

C: Illustration of the higher-order cluster in the context of the entire network. Node locations are the true two-dimensional spatial embedding of the neurons. Most information flows from left to right, and we see that RME/V/R/L and RIH serve as sources of information to the neurons on the right.
Networks are Mysterious

- Not just how to compute, predict, model network data
- But also, why are networks the way they are? What do networks reveal about the underlying system?
Why Networks? Why Now?

- Universal language for describing complex data
  - Networks from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Economics, Statistics, Biology
- Data availability (+computational challenges)
  - Web/mobile, bio, health, and medical
- Impact!
  - Social networking, Social media, Drug design
Networks: Why Now?

Age and size of networks

Jure Leskovec, Stanford
Detecting Fraud

Which transactions are likely to be fraudulent?

(thickness) purchase price
Social and Content Networks
Which link is missing?

(A) vanilla extract, celery  
(B) pepper, onion

Recipe recommendation using ingredient networks. Teng et al., 2012.
Knowledge on Wikipedia

All Wikipedia languages
2M geo-located articles

Montreal

Markus Krötzsch, TU Dresden

Jure Leskovec, Stanford
Can you detect a hoax?

Diffusion Cascades!
Modeling Epidemics

http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0040961
60-90% of LinkedIn users signed up due to an invitation from another user. Global Diffusion via Cascading Invitations: Structure, Growth, and Homophily. Anderson et al., WWW ‘15.
Facebook Cascades

Timeline Photos
Back to Album · I fucking love science's Photos · I fucking love science's Page

Thickness $a$ — Radius $z$

$V = \pi z^2 a$

$V = Pi(z*z)a$

I fucking love science
Seriously. If you have a pizza with radius "z" and thickness "a", its volume is $Pi(z*z)a$.

Likes: Lena Von Der Sten, Iman Khalaf, 明伦 and 73,191 others like this.

27,761 shares

Open Photo Viewer
Download
Embed Post

Jure Leskovec, Stanford
Time: Early (red), Late (blue)
Copies of the Same Post
Single Copy of a Post
This Tutorial: ML with Networks
Four Fundamental Problems

- How to compute over networks
  - Network manipulation system
- How to do ML on networks
  - Building ML features of nodes
- How to detect communities
  - Uncovering modular organization
- How to infer and build networks
  - From data to graphs
Problem 1: Computing with network data


Networks: Common Language

\[ |N| = 4 \]
\[ |E| = 4 \]
Network Representations

Email network $\Rightarrow$ directed multigraph with self-edges

Facebook friendships $\Rightarrow$ undirected, unweighted

Citation networks $\Rightarrow$ unweighted, directed, acyclic

Collaboration networks $\Rightarrow$ undirected multigraph or weighted graph

Mobile phone calls $\Rightarrow$ directed, (weighted?) multigraph

Protein Interactions $\Rightarrow$ undirected, unweighted with self-interactions
How do you define a network?

- How to build a graph:
  - What are nodes?
  - What are edges?

- Choice of the proper network representation of a given domain/problem determines our ability to use networks successfully:
  - In some cases there is a unique, unambiguous representation
  - In other cases, the representation is by no means unique
  - The way you assign links will determine the nature of the question you can study
End-to-End Graph Analytics

Need end-to-end graph analytics system that is flexible, scalable, and allows for easy implementation of new algorithms.
End-to-End Graph Analytics

- **Stanford Network Analysis Platform (SNAP)**
  General-purpose, high-performance system for analysis and manipulation of networks
  - C++, Python (BSD, open source)
  - [http://snap.stanford.edu](http://snap.stanford.edu)
- Scales to networks with hundreds of millions of nodes and billions of edges

New knowledge and insights
Availability of Hardware

Can networks fit in RAM of a single machine?

- Big machines are getting affordable:
  - Server 1TB RAM, 80 cores, $25K

- **Big RAM is eating big data:**
  - Yearly increase of dataset sizes: 20%
  - Yearly increase of RAM sizes: 50%

Bottom line: Want to do graph analytics? **Get a BIG machine!**
Graph Analytics Workflow

- **Input:** Structured data
- **Output:** Results of network analyses
  - Node, edge, network properties
  - Expanded relational tables
  - Networks
SNAP Overview

High-Level Language User Front-End

- Interface with Graph Processing Engine
- Metadata (Provenance)
- Provenance Script

SNAP: In-memory Graph Processing Engine

- Filters
- Graph Methods
- Graph Containers
- Graph, Table Conversions
- Table Objects
- Secondary Storage

Jure Leskovec, Stanford
SNAP supports several graph types
- Directed, Undirected, Multigraph
- >200 graph algorithms
- Any algorithm works on any container
Overview of Network Analytics

- **How to get a network**
  - From a **real-world dataset**
  - Generate a **synthetic network**
  - From an **existing network**

- **Calculate network properties**
  - Quick summary of network properties
  - **Global connectivity**: connected components
  - **Local connectivity**: node degrees
  - Key nodes in the network: node centrality
  - **Neighborhood connectivity**: triads, clustering coefficient
  - **Graph traversal**: breadth and depth first search
  - Groups of nodes: community detection
  - **Global graph properties**: spectral graph analysis
  - Core nodes: K-core decomposition
SNAP Implementation

- High-level front end
  - Python module
  - Uses SWIG for C++ interface
- High-performance graph engine
  - C++ based on SNAP
- Multi-core support
  - OpenMP to parallelize loops
  - Fast hash table, vector operations
## SNAP: Published Benchmarks

<table>
<thead>
<tr>
<th>System</th>
<th>Hosts</th>
<th>CPUs per host</th>
<th>Host Configuration</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GraphChi</td>
<td>1</td>
<td>4</td>
<td>8x core AMD, 64GB RAM</td>
<td>158s</td>
</tr>
<tr>
<td>TurboGraph</td>
<td>1</td>
<td>1</td>
<td>6x core Intel, 12GB RAM</td>
<td>30s</td>
</tr>
<tr>
<td>Spark</td>
<td>50</td>
<td>2</td>
<td></td>
<td>97s</td>
</tr>
<tr>
<td>GraphX</td>
<td>16</td>
<td>1</td>
<td>8X core Intel, 68GB RAM</td>
<td>15s</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>64</td>
<td>2</td>
<td>8x hyper Intel, 23GB RAM</td>
<td>3.6s</td>
</tr>
<tr>
<td><strong>SNAP</strong></td>
<td>1</td>
<td>4</td>
<td>20x hyper Intel, 32GB RAM</td>
<td>6.0s</td>
</tr>
</tbody>
</table>

Twitter2010, one iteration of PageRank
Snap.py Resources

- **Prebuilt packages** available for Mac, Windows, Linux

- **Snap.py documentation**:  

- **SNAP user mailing list**  
  [http://groups.google.com/group/snap-discuss](http://groups.google.com/group/snap-discuss)

- **Developer resources**  
  - Software available as open source under BSD license  
  - GitHub repository  
    [https://github.com/snap-stanford/snap-python](https://github.com/snap-stanford/snap-python)
SNAP C++ Resources

- **Source code** available for Mac, Windows, Linux

- **SNAP documentation**
  - Quick Introduction, User Reference Manual
  - Source code, see [tutorials](http://snap.stanford.edu/snap/doc.html)

- **SNAP user mailing list**
  [http://groups.google.com/group/snap-discuss](http://groups.google.com/group/snap-discuss)

- **Developer resources**
  - Software available as open source under BSD license
  - GitHub repository
    [https://github.com/snap-stanford/snap](https://github.com/snap-stanford/snap)
  - SNAP C++ Programming Guide
SNAP Network Datasets

Collection of over 70 web and social network datasets:
http://snap.stanford.edu/data

Mailing list: http://groups.google.com/group/snap-datasets

- Social networks: online social networks, edges represent interactions between people
- Twitter and Memetracker: Memetracker phrases, links and 467 million Tweets
- Citation networks: nodes represent papers, edges represent citations
- Collaboration networks: nodes represent scientists, edges represent collaborations (co-authoring a paper)
- Amazon networks: nodes represent products and edges link commonly co-purchased products
Problem 2: Automatic Feature Learning in Graphs
Machine Learning in Networks

Node classification

Machine Learning

Jure Leskovec, Stanford
Machine Learning Lifecycle

- (Supervised) Machine Learning Lifecycle: This feature, that feature. Every single time!

- Raw Data
- Structured Data
- Learning Algorithm
- Model

- Automatically learn the features
- Downstream prediction task

Jure Leskovec, Stanford
Feature Learning in Graphs

Goal: Efficient task-independent feature learning for machine learning in networks!
Why Is It Hard?

Graph representations learning is hard:

- Images are fixed size
  - GCN
  - Text is linear
    - word2vec
  - Graphs are neither of these!
    - Node numbering is arbitrary (node isomorphism problem)
    - Much more complicated structure
Two Ideas, Three Algorithms

1) “Linearizing” the graph
   - Create a “sentence” for each node using random walks
     - node2vec, OhmNet

2) Graph convolution networks
   - Propagate information between the nodes of the graph
     - GraphSage
node2vec: Scalable Feature Learning for Networks
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

- **Given a node** \( u \), how do we define nearby nodes?
  - \( N_S(u) \) ... neighbourhood of \( u \) obtained by some strategy \( S \)
Unsupervised Feature Learning

- **Goal:** Find embedding \( f(u) \) that predicts nearby nodes \( N_S(u) \):

\[
\max_f \sum_{u \in V} \log Pr(N_S(u) | f(u))
\]

- Make independence assumption:

\[
Pr(N_S(u) | f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i | f(u))
\]

- Then softmax:

\[
Pr(n_i | f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}
\]

Estimate \( f(u) \) using stochastic gradient descent.
How to determine $N_S(u)$

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

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

$$N_{DFS}(u) = \{ s_4, s_5, s_6 \}$$  \quad \text{Global macroscopic view}
BFS vs. DFS

**BFS:**
Micro-view of neighbourhood

**DFS:**
Macro-view of neighbourhood
Interpolating BFS and DFS

Biased random walk $S$ that given a node $u$ generates neighborhood $N_S(u)$

- Two parameters:
  - Return parameter $p$:
    - Return back to the previous node
  - In-out parameter $q$:
    - Moving outwards (DFS) vs. inwards (BFS)
Biased Random Walks

Biased 2\textsuperscript{nd}-order random walks explore network neighborhoods:

- BFS-like: low value of $p$
- DFS-like: low value of $q$

$p, q$ can be learned in a semi-supervised way.
node2vec algorithm

1) Compute random walk probs.
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.
Experiments: Micro vs. Macro

Interactions of characters in a novel:

\[
p=1, \ q=2
\]
Microscopic view of the network neighbourhood

\[
p=1, \ q=0.5
\]
Macroscopic view of the network neighbourhood
Scalability of node2vec

Scalability on Erdos-Renyi graphs with average degree 10

- Red dots: with SGD
- Blue dots: without SGD

Log10 time (in seconds) vs. Log10 nodes.
Incomplete Network Data (PPI)

Macro-F1 score vs Fraction of missing edges

Macro-F1 score vs Fraction of additional edges
node2vec: Discussion

General-purpose feature learning in networks:

- An explicit locality preserving objective for feature learning.
- Biased random walks capture diversity of network patterns.
- Scalable and robust algorithm with excellent empirical performance.
- Future extensions would involve designing random walk strategies entailed to network with specific structure such as heterogeneous networks and signed networks.
node2vec: Discussion

- There are many other methods in class of walk-based methods:
  - LINE
  - DeepWalk
  - Structural Deep Network Embedding
OhmNet: Extension to Hierarchical Networks

Multilayer Networks

Let’s generalize node2vec to multilayer networks!
Multi-Layer Networks

- Each network is a layer $G_i = (V_i, E_i)$
- Similarities between layers are given in hierarchy $\mathcal{M}$, map $\pi$ encodes parent-child relationships
The Approach

- Computational framework that learns features of every node and at every scale based on:
  - Edges within each layer
  - Inter-layer relationships between nodes active on different layers
Features in Multi-Layer Network

- **Given:** Layers $\{G_i\}$, hierarchy $\mathcal{M}$
  - Layers $\{G_i\}_{i=1..T}$ are in leaves of $\mathcal{M}$
- **Goal:** Learn functions: $f_i: V_i \rightarrow \mathbb{R}^d$

Jure Leskovec, Stanford
Features in Multi-Layer Network

- Approach has two components:
  - Per-layer objectives: Nodes with similar network neighborhoods in each layer are embedded close together
  - Hierarchical dependency objectives: Nodes in nearby layers in hierarchy are encouraged to share similar features
Per-Layer Objective: node2vec

- **Intuition**: For each layer, find a mapping of nodes to $d$ dimensions that preserves node similarity

- **Approach**: Similarity of nodes $u$ and $v$ is defined based on similarity of their network neighborhoods

- **Given** node $u$ in layer $i$ we define nearby nodes $N_i(u)$ based on random walks starting at node $u$
Per-Layer Objective: node2vec

- Given node $u$ in layer $i$, learn $u$’s representation such that it predicts nearby nodes $N_i(u)$:
  \[
  \omega_i(u) = \log Pr(N_i(u) | f_i(u))
  \]
  [Grover et al. 2016]

- Given $T$ layers, maximize:
  \[
  \Omega_i = \sum_{u \in V_i} \omega_i(u), \quad \text{for } i = 1, 2, \ldots, T
  \]
Interdependent Layers

- So far, we did not consider hierarchy $\mathcal{M}$
- Node representations in different layers are learned independently of each other

How to model dependencies between layers when learning node features?
Interdependent Layers

- Given node \( u \), learn \( u \)’s representation in layer \( i \) to be close to \( u \)’s representation in parent \( \pi(i) \):
  \[
  c_i(u) = \frac{1}{2} \| f_i(u) - f_{\pi(i)}(u) \|^2
  \]

- **Multi-scale**: Repeat at every level of \( \mathcal{M} \)

\[
C_i = \sum_{u \in L_i} c_i(u)
\]

\( L_i \) has all layers appearing in sub-hierarchy rooted at \( i \)
OhmNet: Final Model

Learning node features in multi-layer networks

Solve maximum likelihood problem:

$$\max_{f_1, f_2, \ldots, f_{|M|}} \sum_{i \in T} \Omega_i - \lambda \sum_{j \in M} C_j.$$
Experiments: Biological Nets

- Proteins are worker molecules
- Understanding protein function has great biomedical and pharmaceutical implications

107 genome-wide tissue-specific protein interaction networks

- 584 tissue-specific cellular functions
- Examples (tissue, cellular function):
  - (renal cortex, cortex development)
  - (artery, pulmonary artery morphogenesis)
Brain Tissues

9 brain tissue PPI networks in two-level hierarchy
Meaningful Node Embeddings

- Cerebellum
- Medulla oblongata
- Substantia nigra
- Frontal lobe
- Temporal lobe
- Pons
- Parietal lobe
- Occipital lobe
- Midbrain

Jure Leskovec, Stanford
Protein Function Prediction

42% improvement over state-of-the-art on the same dataset
Transfer Learning

- **Transfer functions to unannotated tissues**
- **Task:** Predict functions in target tissue without access to any annotation/label in that tissue

<table>
<thead>
<tr>
<th>Target tissue</th>
<th>OhmNet</th>
<th>Tissue non-specific</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placenta</td>
<td>0.758</td>
<td>0.684</td>
<td>11%</td>
</tr>
<tr>
<td>Spleen</td>
<td>0.779</td>
<td>0.712</td>
<td>10%</td>
</tr>
<tr>
<td>Liver</td>
<td>0.741</td>
<td>0.553</td>
<td>34%</td>
</tr>
<tr>
<td>Forebrain</td>
<td>0.755</td>
<td>0.632</td>
<td>20%</td>
</tr>
<tr>
<td>Blood plasma</td>
<td>0.703</td>
<td>0.540</td>
<td>40%</td>
</tr>
<tr>
<td>Smooth muscle</td>
<td>0.729</td>
<td>0.583</td>
<td>25%</td>
</tr>
<tr>
<td>Average</td>
<td>0.746</td>
<td>0.617</td>
<td>21%</td>
</tr>
</tbody>
</table>

Reported are AUC values

Jure Leskovec, Stanford
Inductive Representation Learning on Large Graphs.
So far...

Node2vec and OhmNet are:

- Unsupervised
- There is no model
  - Only the training (but no prediction phase)
- They directly learn node coordinates
  - Can’t generalize to unseen nodes. Entire embedding has to be retrained
  - Too many parameters to scale to large graphs
Inductive Feature Learning

Desiderata:

- Scale to large networks
- Generalize to new nodes
- Build a model of node embedding
  - Training phase can be slow but prediction phase should be fast
Inductive Feature Learning

Inductive node embedding → generalize to entirely unseen graphs

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

Jure Leskovec, Stanford
Inductive Feature Learning

Many application settings constantly encounter previously unseen nodes. e.g., Reddit, YouTube, GoogleScholar, ....

Need to generate new embeddings “on the fly”
What makes it hard?

- Generalizing to unseen subgraphs requires "aligning" the new subgraph to data we’ve seen before
  - Closely related to the subgraph isomorphism problem
- Node neighborhoods have no canonical ordering
  - No straightforward way to "linearize" neighborhood features
Graph convolutional networks

- CNN on an image:

  Goal is to generalize convolutions beyond simple lattices. Leverage node features/attributes (e.g., text, degrees)

Graph convolutional networks

Graph Convolutional Networks:

- convolutional architecture
- graph normalization
- neighborhood graph construction
- node sequence selection

Problem: For a subgraph how to come with canonical node ordering

Graph convolutional networks

- Graph Convolutional Networks:
  - Has to also crop excess nodes, etc.

New Idea: GCNs+Aggregation

- Kipf et al. (2017) proposed GCNs for semi-supervised learning on graphs:

\[ f(X, A) = \text{softmax}\left( \hat{A} \ \text{ReLU}\left( \hat{A}XW^{(0)} \right) W^{(1)} \right) \]

- \( X \) … Node feature vector
- \( A \) … Graph adjacency matrix
- \( W^{(0)}, W^{(1)} \) … Model parameters

- More generally:

\[ H^{(k+1)} = \sigma(AW^kH^{(k)}) \text{, and } H^{(0)} = X \]

GCNs + Aggregation

\[ H^{(k+1)} = \sigma(AW^k H^{(k)}) \], and \( H^{(0)} = X \)

- Notice this avoids coming up with a node ordering for a given subgraph
- Essentially, a fixed, center-surround filter that “averages” the features of immediate neighbors

Jure Leskovec, Stanford
GraphSAGE

- Adapt the GCN idea to inductive node embedding
- Generalize beyond simple convolutions
- Demonstrate that this generalization
  - Leads to significant performance gains
  - Allows the model to learn about local structures
GraphSAGE: Sample & Aggregate

- $H^{(k+1)} = \sigma(AW^k H^{(k)})$, and $H^{(0)} = X$

**Important:** Directly leverage input node features (e.g., attributes, degrees)
GraphSAGE: Sample & Aggregate

- Sample and aggregate:
  \[ h_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}); \]
  \[ h_v^k \leftarrow \sigma \left( W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k) \right) \]

- Aggregate information from neighbors
- Concatenate it with previous features
- Pass through a single layer network
  - \( W^k \) and the activation function \( \sigma \)
GraphSAGE Algorithm

1. **Initialize representations as features**
   \[ h_0^v \leftarrow x_v, \forall v \in V; \]

2. **For** \( k = 1 \ldots K \)** do
   - **For** \( u \in V \) do
     - ** Aggregate information from neighbors**
       \[ h_{N(v)}^k \leftarrow \text{AGGREGATE}_k(\{h_{u}^{k-1}, \forall u \in N(v)\}); \]
     - **Concatenate neighborhood info with current representation and propagate**
       \[ h_v^k \leftarrow \sigma \left( W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{N(v)}^k) \right) \]
   - \[ h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in V \]

3. **End**
   \[ z_v \leftarrow h_v^K, \forall v \in V \]
WL isomorphism test

- The classic Weisfeiler-Lehman graph isomorphism test is a special case of GraphSAGE
- We replace the hash function with trainable neural nets:

\[
\begin{align*}
    h_v^0 &\leftarrow x_v, \forall v \in V; \\
    \text{for } k = 1 \ldots K \text{ do} \\
    \quad \text{for } v \in V \text{ do} \\
    \quad \quad h_{\mathcal{N}(v)}^k &\leftarrow \text{AGGREGATE}_k\left(\left\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\right\}\right); \\
    \quad \quad h_v^k &\leftarrow \otimes \left(V^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right) \\
    \quad \text{end} \\
    \quad h_v^k &\leftarrow h_v^k / \|h_v^k\|_2, \forall v \in V \\
    \text{end} \\
    z_v &\leftarrow h_v^K, \forall v \in V
\end{align*}
\]

Aggregation

\[
\begin{align*}
    h_v^0 &\leftarrow x_v, \forall v \in \mathcal{V}; \\
    \text{for } k = 1 \ldots K &\text{ do} \\
    &\quad \text{for } u \in \mathcal{V} \text{ do} \\
    &\quad \quad h_u^{k-1} \leftarrow \text{AGGREGATE}_k \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}; \\
    &\quad \quad h_v^k \leftarrow \sigma \left( W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k) \right) \\
    &\quad \text{end} \\
    &\quad h_v^k \leftarrow h_v^k / \| h_v^k \|_2, \forall v \in \mathcal{V} \\
    \text{end} \\
    z_v &\leftarrow h_v^K, \forall v \in \mathcal{V}
\end{align*}
\]
Aggregation

- Need a function that can aggregate over a set of points:
  - i.e., should be permutation invariant

*mean*: simple, element-wise mean

*LSTM*: apply LSTM to random permutation of the points

*max-pooling*: \( \max(\{\sigma(Wh_u + b), \forall u \in \mathcal{N}(v)\}) \)

*element-wise max*
How do we learn the model?

- The essential part of the model:
  \[ h^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{ h^{k-1}_u, \forall u \in \mathcal{N}(v) \}) ; \]
  \[ h^k_v \leftarrow \sigma \left( W^k \cdot \text{CONCAT}(h^{k-1}_v, h^k_{\mathcal{N}(v)}) \right) \]

- Two types of parameters:
  - Aggregate function can have params.
  - Matrix \( W^k \)

- How do we learn them?
  - Gradient descent. We can use supervised or unsupervised objective function:
    \[ J = -\log (\sigma(z^T_u z_v)) - \frac{1}{|Q|} \cdot \sum_{q=1}^Q \mathbb{E}_{v_n \sim P_n(v)} \log (-\sigma(z^T_u z_{v_n})) \]
    classification (cross-entropy) loss
GraphSAGE vs. node2vec

- Directly leverage feature information
- Instead of optimizing embeddings directly, optimize a neural network that aggregates neighborhood features.

Jure Leskovec, Stanford
GraphSAGE vs. GCN

- Adapt to inductive setting (e.g., unsupervised loss, neighborhood sampling, minibatch optimization)
- Generalized notion of “aggregating neighborhood”
References


**Code:**
- [http://snap.stanford.edu/node2vec/](http://snap.stanford.edu/node2vec/)
- [http://snap.stanford.edu/graphsage/](http://snap.stanford.edu/graphsage/)