A Tutorial on Graph Neural Networks
Graph Convolution, Attention and Sample and Aggregate

Zhiming Xu
zhimingxu@smail.nju.edu.cn

Department of Computing
The Hong Kong Polytechnic University

October 15, 2020
Overview

Introduction

Graph Convolutional Networks

GraphSAGE

Graph Attention Network
Recap

- **Graph**
  
  A data structure consists of *Vertices* and *Edges*. Denoted by set $\mathcal{V}$ and $\mathcal{E}$, respectively, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

- **Neural Networks**
  
  An interconnected group of neurons performing a series of computations.

---

(a) A graph with six vertices and eight edges.

(b) A neural network with one hidden layer.

**Figure 2**: Example of graph and neural network.

---

$^1$The word "node" and "vertex" are used interchangeably in this tutorial.
Graph Neural Networks (GNNs)

- A type of neural networks operating directly on graphs [1].
- To learn a state representation which contains information of each vertex’s neighborhood.
- Notations in this tutorial

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{R}^m$</td>
<td>$m$-dimensional Euclidean space</td>
</tr>
<tr>
<td>$a, \bar{a}, A$</td>
<td>scalar, vector, matrix</td>
</tr>
<tr>
<td>$A$</td>
<td>adjacent matrix</td>
</tr>
<tr>
<td>$X$</td>
<td>(node) feature matrix</td>
</tr>
<tr>
<td>$D$</td>
<td>degree matrix, $D_{ii} = \sum_j A_{ij}$</td>
</tr>
<tr>
<td>$I_N$</td>
<td>$N$-dimensional identity matrix</td>
</tr>
<tr>
<td>$\vec{h}, H$</td>
<td>learned hidden vector, matrix</td>
</tr>
<tr>
<td>$W$</td>
<td>neural network weight matrix</td>
</tr>
<tr>
<td>$\sigma, \cdot^\top, \cdot|\cdot$</td>
<td>non-linear activation, transpose, concatenation</td>
</tr>
</tbody>
</table>

Table 1: Notations used in this tutorial
The Operation of Convolution

▶ Convolution
An operation on two functions $f$ and $g$ that produces a third function $f \ast g$.

▶ Convolutional Neural Network (CNN)
Neural networks with the operation of convolution, usually used on images where $g$ is grid and $f$ is called filter.

▶ Convolution on Graphs
Graphs are not as regular as grids. New methods are needed to generalize convolution to them.

(a) An example of 2D convolution.  
(b) Convolution on graphs?
Generalize Convolution to Graphs

- Spectral convolutions on graphs with signal $\vec{x} \in \mathbb{R}^n$ in the Fourier domain

$$g_\theta \ast \vec{x} = U g_\theta U^\top \vec{x} \quad (1)$$

where

1. Normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^\top$
2. $U$ is the matrix of eigenvectors of normalized graph Laplacian
3. $U^\top \vec{x}$ is the Fourier transformation on $\vec{x}$
4. $g_\theta$ is the spectral convolutional filter. Can be seen as a function $g_\theta(\Lambda)$ on eigenvalues of $L$

- Equation 1 is computationally expensive and thus needed an efficient approximation.
Approximations

1. $K^{\text{th}}$ order Chebyshev polynomial

   \[ g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \]  

   \[ \text{(2)} \]

2. In Equation 1, substitute $g_{\theta}$ with Equation 2

   \[ g_{\theta'} \ast \vec{x} = \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\vec{x}, \quad \tilde{L} = \frac{2}{\lambda_{\text{max}}} L - I_N \]  

   \[ \text{(3)} \]

3. Limit order $K$ to 1, round $\lambda_{\text{max}}$ to 2, and reduce parameters

   \[ g_{\theta'} \ast \vec{x} \approx \theta'_0 \vec{x} - \theta'_1 D^{-\frac{1}{2}} AD^{-\frac{1}{2}} \vec{x} \]

   \[ \approx \theta \left( I_N + D^{-\frac{1}{2}} AD^{-\frac{1}{2}} \right) \vec{x} \]  

   \[ \text{(4)} \]
Renormalization

- In Equation 4, the $I_N + D^{-\frac{1}{2}} AD^{-\frac{1}{2}}$ term's eigenvalues are in $[0, 2]$. Stacking layers with this operation might cause vanishing/exploding gradients.

- The renormalization trick is thus introduced to alleviate this problem

$$I_N + D^{-\frac{1}{2}} AD^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

where $\tilde{A} = A + I_N$, $\tilde{D}$ is $\tilde{A}$'s degree matrix
Graph Convolutional Network (GCN)

Fast Approximate Graph Convolution

- Generalize to vector signal nodes

\[ \theta \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) \vec{x} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \]

- Propagation rule

Multi-layer Graph Convolution Network [2]

\[ H^{(l+1)} = \sigma \left( \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)} \right) , H^{(0)} = X \]

- Two-layer example (Calculate \( \hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \) in advance)

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

- Distill high-dimensional information and reduce it to a dense vector.
- Low-dimensional vector embeddings of nodes in large graphs are very useful in various downstream tasks.

Problem with Using GCNs

- Whole graph is large and computationally prohibitive. Mini-batch is slow to train and hard to converge.
- Full graph is needed, training in a transductive way.
- Difficult to use on real-world dynamic graphs.
Sample neighborhood and aggregate the information.

Figure 4: Illustration of GraphSAGE forward propagation.²

²http://snap.stanford.edu/graphsage/
GraphSAGE Forward Propagation \[3\]

**Result:** Node $i$’s representation $z_i$ after $K$ iterations

\[
\vec{h}_i^0 \leftarrow \vec{h}_i, \forall i \in \mathcal{V};
\]

for $k = 1 \ldots K$ do

for $i \in \mathcal{V}$ do

\[
\begin{align*}
\vec{h}_{N_i}^k & \leftarrow \text{AGGREGATE}_k \left( \{\vec{h}_j, \forall j \in \mathcal{N}_i\} \right); \\
\vec{h}_i^k & \leftarrow \sigma \left( W^k \cdot \left[ \vec{h}_i^{k-1} \| \vec{h}_{N_i}^k \right] \right);
\end{align*}
\]

end

\[
\vec{h}_i^k \leftarrow \frac{\vec{h}_i^k}{\|\vec{h}_i^k\|_2}, \forall i \in \mathcal{V};
\]

end

\[
\vec{z}_i \leftarrow \vec{h}_i^K, \forall i \in \mathcal{V}
\]
Parameter Learning of GraphSAGE

Graph-Based Loss Function \([3]\)

\[
L_G(\vec{h}_i) = - \log \left( \sigma \left( \vec{h}_i^\top \vec{h}_j \right) \right) - Q \cdot \left( \mathbb{E}_{v_i \sim P_{n(i)}} \log \left( \sigma \left( -\vec{h}_i^\top \vec{h}_{v_i} \right) \right) \right)
\]

- \(j\) is a node that co-occurs near \(i\) on fixed-length random walk.
- \(\sigma\) is the sigmoid function, \(\sigma(x) = \frac{1}{1 + \exp(-x)}\)
- \(P_n\) is a negative sampling distribution, \(Q\) is \# of negative samples.

Based on loss \(L_G\), the parameters in Algorithm 1 are optimized with stochastic gradient descend.
Choice of Aggregator Functions

▶ Mean Aggregator

\[ \vec{h}_i^k \leftarrow \sigma \left( W \cdot \text{MEAN} \left( \left\{ \vec{h}_i^{k-1} \right\} \cup \left\{ \vec{h}_j^{k-1}, \forall j \in \mathcal{N}_i \right\} \right) \right) \]

▶ LSTM Aggregator

\[ \vec{h}_i^k \leftarrow \text{LSTM} \left( \pi \left\{ \vec{h}_j, \forall j \in \mathcal{N}_i \right\} \right) \]

▶ Pooling Aggregator

\[ \vec{h}_i^k \leftarrow \max \left( \left\{ \sigma \left( W_{pool} \vec{h}_j^k + \vec{b} \right), \forall j \in \mathcal{N}_i \right\} \right) \]
Attention Mechanism

- Attention mechanism achieves great successes in sequence-based tasks.
- They can be used to deal with variable size inputs, and focus on the most relevant parts by assigning different weights.
- Attention used on a single sequence is called self-attention.

![Attention visualization, generated by bertviz](image)

**Figure 5**: Attention visualization, generated by bertviz
(Self-)Attention Mechanism on Graphs

- **Input**: Set of node features \( H = \{ \vec{h}_1, \vec{h}_2, \cdots, \vec{h}_N \} \), \( h_i \in \mathbb{R}^d \). \( N \) is the number of nodes and \( d \) is the feature dimension.

- **Output**: A new set of node features 
  \( H' = \{ \vec{h}'_1, \vec{h}'_2, \cdots, \vec{h}'_N \} \), \( h'_i \in \mathbb{R}^d \).

- **Attention** \( a : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \) with *weight matrix* \( W \)
  
  \[ e_{ij} = a(W\vec{h}_i, W\vec{h}_j) \]

  \( e_{ij} \) is called attention coefficients.
Graph Attentional Layer (cont.)

Details of Attention $\alpha$

- Masked: Calculate $e_{ij}$ for $j \in \mathcal{N}_i$, i.e., $i$’s neighborhood.
- Normalized: Use softmax to normalize across all $j$’s

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

- Attention $\alpha$’s implementation

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}(\vec{a}^\top [W\vec{h}_i || W\vec{h}_j])\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}(\vec{a}^\top [W\vec{h}_i || W\vec{h}_j])\right)}$$

**LeakyReLU**

$$\text{LeakyReLU} = \begin{cases} 
\alpha \cdot x, & x < 0 \\
x, & x > 0 
\end{cases}$$
Graph Attentional Layer (cont.)

Attention Acts on Hidden Representations

- Linear combination and activation

\[ \tilde{h}'_i = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} W \tilde{h}_j \right) \]

- Multi-head attention
  
  • Concatenation

\[ \tilde{h}'_i = \left\|_{k=1}^{K} \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} W^k \tilde{h}_j \right) \right\| \]

  • Average

\[ \tilde{h}'_i = \sigma \left( \frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij} W^k \tilde{h}_j \right) \]
Graph Attention Network (GAT)

Graph Attention Network Propagation Rule and Illustration \([4]\]

\[
\vec{h}'_i = \sigma \left( \sum_{j \in N_i} \frac{\exp \left( \text{LeakyReLU} \left( \vec{a}^\top \left[ W\vec{h}_i || W\vec{h}_j \right] \right) \right)}{\sum_{k \in N_i} \exp \left( \text{LeakyReLU} \left( \vec{a}^\top \left[ W\vec{h}_i || W\vec{h}_j \right] \right) \right)} W\vec{h}_j \right)
\]

**Figure 6:** Left: Attention mechanism \(a\). Right: Multi-head attention on a graph.
Q & A
