Graph Representation
Learning

Nikita Yadav, Rashi Verma, Rishabh Gupta,
Swyam Prakash Singh
Feature Learning in Graphs

Goal: Map each node into a low dimensional space which encodes network information.

Motivation: Various applications such as clustering, link prediction, label classification, ...
Feature Learning in Networks

- ‘Linearizing’ the graph
  - Create a sentence for each node using random walks
  - Node2vec

- Graph Convolutional Networks
  - Uses Convolutional architecture with deep network.
  - GCN
Node2vec: Unsupervised Feature Learning

● Let G = (V, E) be any (un)directed, (un)weighted network. Learn node embeddings $f : V \rightarrow \mathbb{R}^d$ such that nearby nodes are close together.

● Objective: $\max_f \sum_{u \in V} \log Pr(N_S(u) | f(u))$

● Assume Conditional independence: $Pr(N_S(u) | f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i | f(u))$

● Then Softmax: $Pr(n_i | f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$

How to determine $N_s(u)$?

- Classic strategies to define neighborhood $N_s(u)$ of a node $u$.
  - BFS captures homophily.
  - DFS captures structural equivalence.

- Interpolating BFS and DFS
  - Return parameter $p$: return back to the previous node.
  - In-out parameter $q$: Moving outward (DFS) vs inwards (BFS)
Random walk procedure

Sampling of nodes uses probability distribution given by

\[ P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases} \]

\[ \pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx} \]

\[ \alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases} \]
The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Node2vec extends the skip-gram architecture to networks.

Source: Efficient Estimation of Word Representations in Vector Space, T Mikolov et. al.,
Link Prediction

- It is the task to estimate the probability of links between nodes in a graph.
- Link prediction heuristics scores:
  - Common Neighbors: $|N(u) \cap N(v)|$
  - Jaccard's coefficient: $|N(u) \cap N(v)| / |N(u) \cup N(v)|$
  - ...
- How we did link prediction using node2vec?

![Original graph](image1)

![Residual graph](image2)

Sampled edges for training/testing
Datasets

- Zachary's karate club Network
  - Contains 34 nodes and 78 edges, where each link is between pair who interacted outside the club.
- Facebook dataset
  - Network between facebook friends. It contains 4,039 nodes and 88,234 edges.
- BlogCatalog dataset
  - Network of social relationships of the bloggers listed on the BlogCatalog website. It has 10,312 nodes, 333,983 edges.
Results - Zachary’s Karate Club Network

Original Graph

Subgraph

Predictions
Results on Facebook and BlogCatalog Dataset

Table 1: Mean Auc scores

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Hadamard</th>
<th>Average</th>
<th>L2</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook</td>
<td>0.9551</td>
<td>0.6443</td>
<td>0.986</td>
<td>0.9858</td>
</tr>
<tr>
<td>BlogCatalog</td>
<td>0.5829</td>
<td>0.6396</td>
<td>0.9032</td>
<td>0.8982</td>
</tr>
</tbody>
</table>

Fig. Grid search over parameters p and q, with Weighted L2

Fig. Parameter Sensitivity plots
Node2vec on weighted signed network

- Future extensions of node2vec:
  - Signed edge networks
  - Heterogeneous information networks
  - ...
- We explored ways to extend node2vec for weighted signed network
- Dataset: Bitcoin OTC trust weighted signed network
  - Edge weight range: -10 to 10
  - Contains 5881 nodes and 35592 edges.

Contd.

- Convert edge weight in range 0 to 1.

\[
\frac{1}{1 + \exp\left(-\frac{x}{3}\right)}
\]

- Result of link prediction: 0.88 AUC score with weighted L2 edge function.
OBJECTIVE:

Tuning of hyperparameters

Obtaining GCN embeddings

Unsupervised clustering on the graph using DEC technique
Graph Convolutional Network

Formulation of CNN in context of spectral theory.

Two approaches:

- **Spatial** - considers local receptive fields upto neighborhoods only.
  - e.g. locality on W, deep locally connected graph
- **Spectral** - exploiting global structure of graph by graph laplacian to generalize convolutional operator
  - e.g. GCN
Background

- Main aim is faster training and higher accuracy.
- Requirement is learning fast localized spectral filters
- Laplacian $L = D - W$ where $D$ is degree matrix and $W$ is adjacent matrix.
- For easiness we use convolution in fourier domain
- $x^* g y = U((U^T x) \odot (U^T y))$ where $\odot$ is element wise product and $U$ is fourier basis and obtained from eigen decomposition.
- $U$ is some function of eigen value of $L$ which can be well approximated by chebyshev polynomial:

$$g_{\theta'} \ast x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x$$
● Taking upto $K^{th}$ order of chebyshev polynomial will give $K$-localized expression
● If $K=1$, stacking multiple such layer will give rich class of convolutional filters
● Forward model -

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

where,

\[
\hat{A} = \bar{D}^{-\frac{1}{2}} \hat{A} \bar{D}^{-\frac{1}{2}}
\]
Experiments

Classification with chebyshev polynomial

Datasets used: Pubmed, Cora, Citeseer

<table>
<thead>
<tr>
<th>No of layers</th>
<th>Maximum degree</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>79.70 %</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>80.50 %</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>78.70 %</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>80.40 %</td>
</tr>
</tbody>
</table>

Table 1: Cora
<table>
<thead>
<tr>
<th>No of layers</th>
<th>Maximum degree</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>79.17 %</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>79.30 %</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>79.30 %</td>
</tr>
</tbody>
</table>

Table 2: Pubmed

<table>
<thead>
<tr>
<th>No of layers</th>
<th>Maximum degree</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>71.50 %</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>71.50 %</td>
</tr>
</tbody>
</table>

Table 3: Citeseer
Tuning hyperparameters

![Graph 1: Accuracy vs Hidden units](image1)

![Graph 2: Accuracy vs Epoch](image2)
Deep embedding Clustering

Method which simultaneously learn:

- Unsupervised Clustering using deep neural network
- Non linear mapping high dimension data space to lower dimension data space

Proposed by Xie et. al
DEC has two phases:

- Parameter initialization with deep autoencoders
- Parameter optimization/clustering based on minimizing KL divergence with help of auxiliary target distribution computation.
Implementation

- Determining hyperparameters by cross validation is not an option
- Commonly used parameters are used

Experiments

- Dataset- MNIST
- Clustering Accuracy- 83.42 %
- Number of encoder layers- 3
- Number of decoder layers- 3
- Dropout- 0.2
Finally we tried connecting both these models...

- We obtained graph embeddings from GCN
- Next we used DEC model but instead of encoder we used the embeddings obtained from GCN
- In DEC we used autoencoder where training is done using reconstruction loss
- But, in our model we used original feature vector instead of the gcn embeddings to train on reconstruction loss
References


