A Tutorial on Geometric Deep Learning & Graph Neural Networks

Semi-Supervised Classification with Graph Convolutional Networks

Thomas N. Kipf, Max Welling (ICLR 2017)

By Prachi Garg, Vision for Mobility Reading Group
September 9, 2020

Euclidean vs non-Euclidean Geometry

- Euclidean geometry
    - Most commonly known and studied geometry
    - Geometry of flat surfaces
- **3 main properties that distinguish it from non-Euclidean geometry:**
    - Parallel Postulate - Defines the nature of parallel lines
        - Given a line and a point, there is only one other line that you can draw though the point that will be parallel to the original line. **Non-Euclidean geometry doesn't follow the parallel postulate**
    - In Euclidean geometry, the interior angles of triangles always add up to 180 degrees
    - Shortest distance between 2 points is a straight line joining the two points.
- Most common non-Euclidean geometries - spherical geometry, elliptic geometry and hyperbolic geometry
- Image, text, audio, video data fall under euclidean data
- Arbitrary structured graphs fall under the category of non-Euclidean data

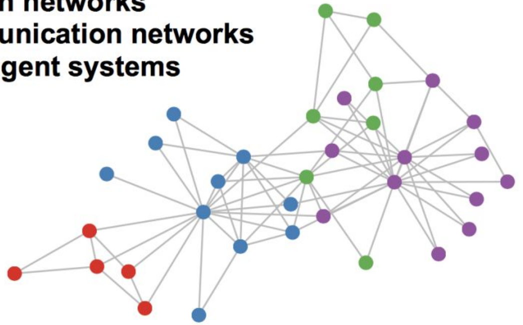
Example - A non-inflated balloon is a flat object governed by Euclidean geometry. Now inflate the same balloon, its surface is no longer flat, we need non-Euclidean geometry to define it.

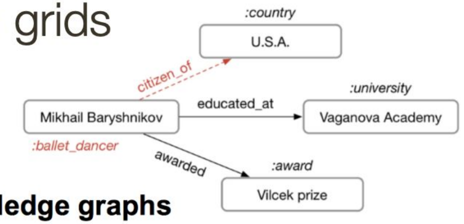
Graph-structured data

Arbitrary structured graphs and manifolds are ubiquitous and form the backbone of several real-world application

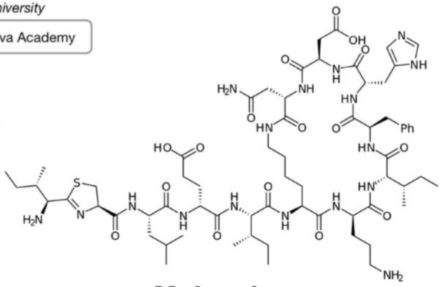
A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems

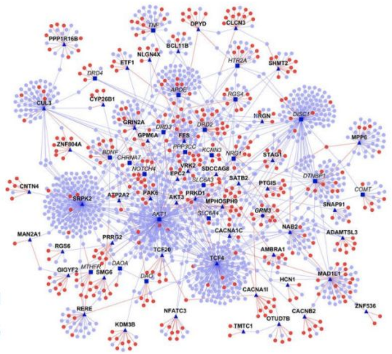




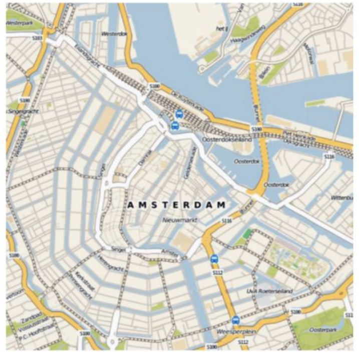
Knowledge graphs



Molecules



Protein interaction networks



Road maps

Traditional approaches for modeling graphs

Graph embedding techniques -

- Convert graphs into euclidean form and use traditional ML, DL models

- Employs a class of graph pre-processing techniques, turn a graph into a computationally digestible format for the ML model while trying to preserve graph structure and data

- Some of these techniques:
  - DeepWalk [link]
  - Node2Vec [link]
  - Graph2Vec [link]
  - Large-scale Information Network Embedding (LINE) [link]
  - Structural Deep Network embedding (SDNE) [link]

Pitfalls of traditional techniques

These approaches rely on summary graph statistics (*e.g.* degrees or clustering coefficients), kernel functions, or hand designed features to measure local neighborhood structures.

1. **Expensive and Inflexible -** Use Hand designed features, can't dynamically adapt to heterogeneous graphs with different types and sizes

2. **Not generalisable** (transductive) across problem statements and application domains or even new graphs

3. **Scalability** - Many random walks approaches use shallow models that can't scale well to more complicated and large-scale graphs

4. Compromise on the rich topological relationships in the graph

5. Computationally inefficient - No parameter sharing between nodes

Why deep learning over graphs? Taking inspiration from CNNs

Representation learning methods like deep learning can overcome each pitfall

Main idea is to be able to process the graph as it is, in its original form along with its rich structural information without any conversions.

Desirable properties for a neural network layer that can process a graph:

- Computational and storage efficiency (requiring no more than O(V + E) time and memory);
- Fixed number of parameters (**independent of input graph size**);
- Localisation (acting on a local neighbourhood of a node);
- Ability to specify **arbitrary importance** to different neighbours;
- Applicability to **inductive problems** (arbitrary, unseen graph structures).

Geometric deep learning and Graph Neural Networks

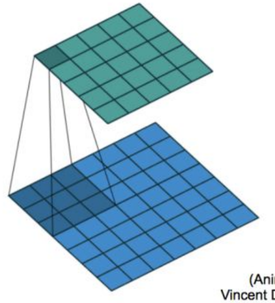
Geometric Deep Learning

Generalisation of deep learning to non-Euclidean data, using deep neural networks to model arbitrary structured graphs and manifolds

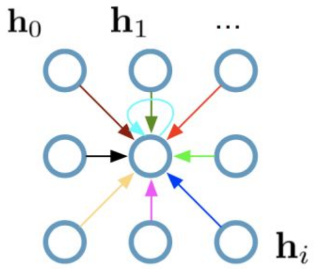
Layer wise forward propagation - in the form of neighbourhood aggregation for each node in the graph
Use some form of message passing to aggregate information from the neighbours and use the inter-node relationships

Similarity with CNNs

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

* slide from Thomas Kipf, **University of Amsterdam**

Graph Convolutional Network

Given Graph G = (V, E)
Input: Input feature matrix X (a N x D feature matrix)
Adjacency Matrix - $A \in R^{NxN}$ containing the graph structural information
Output: Z, NxF feature matrix where F is the number of output classes

$$H^{(l+1)} = f(H^{(l)}, A),$$

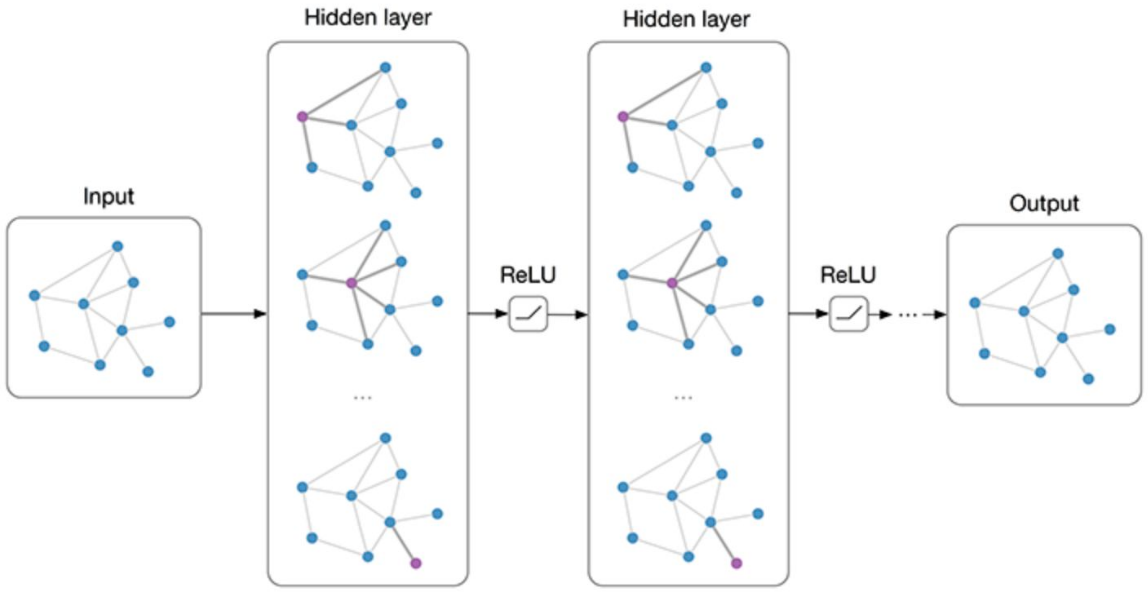
$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right),$$

1. Self Loop $\tilde{A} = A + I_N$
2. Normalization - each node will have different number of neighbours, So normalise A by taking (D^{-1} A) where D is the diagonal node degree matrix. $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

Layer wise propagation rule:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right).$$

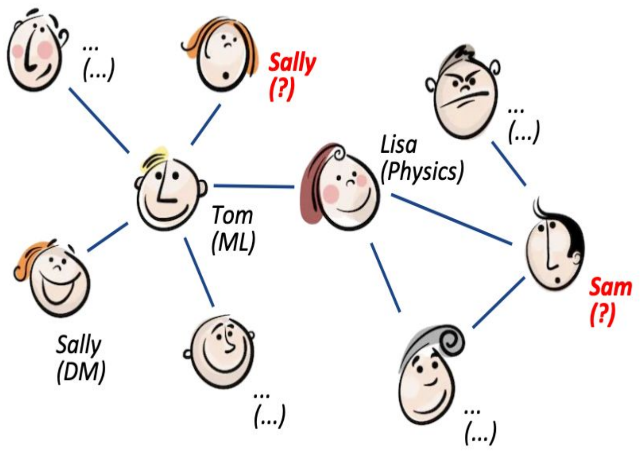
Graph Convolutional Network



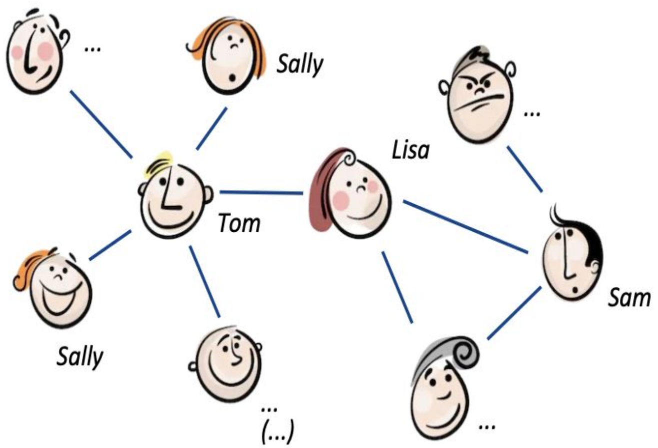
Multi-layer Graph Convolutional Network (GCN) with first-order filters.

Successive application of filters of this form then effectively convolve the Kth-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model.

- **Node Classification:** (Semi-supervised Learning)
    - Predict research area of unlabeled authors



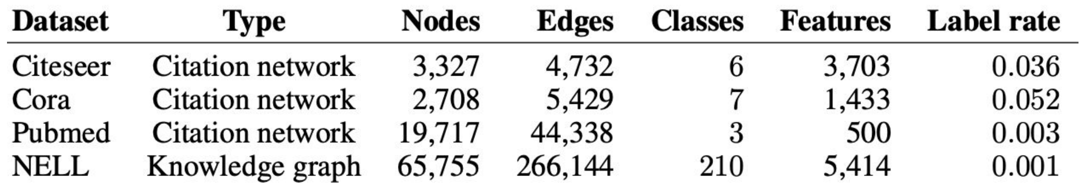
- **Co-authorship Network**
    - **Nodes**: Authors, **Edges**: Co-authorship



Graph Convolutional Network

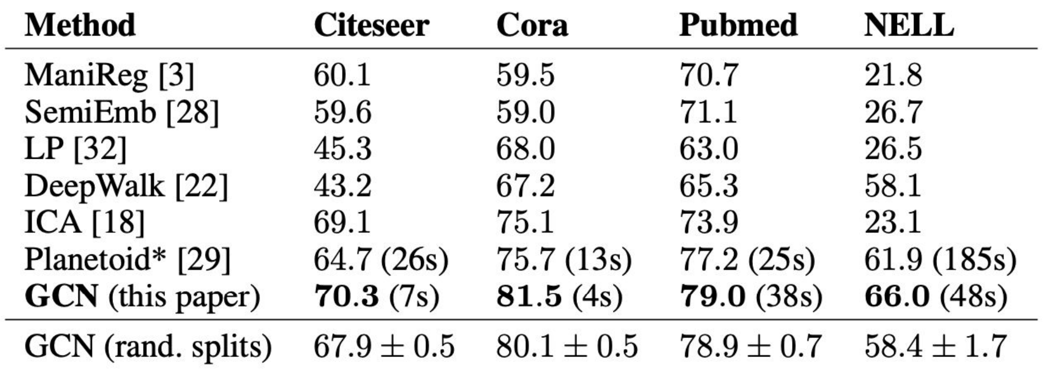
Datasets and Results

Table 1: Dataset statistics, as reported in Yang et al. (2016).



Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Table 2: Summary of results in terms of classification accuracy (in percent).



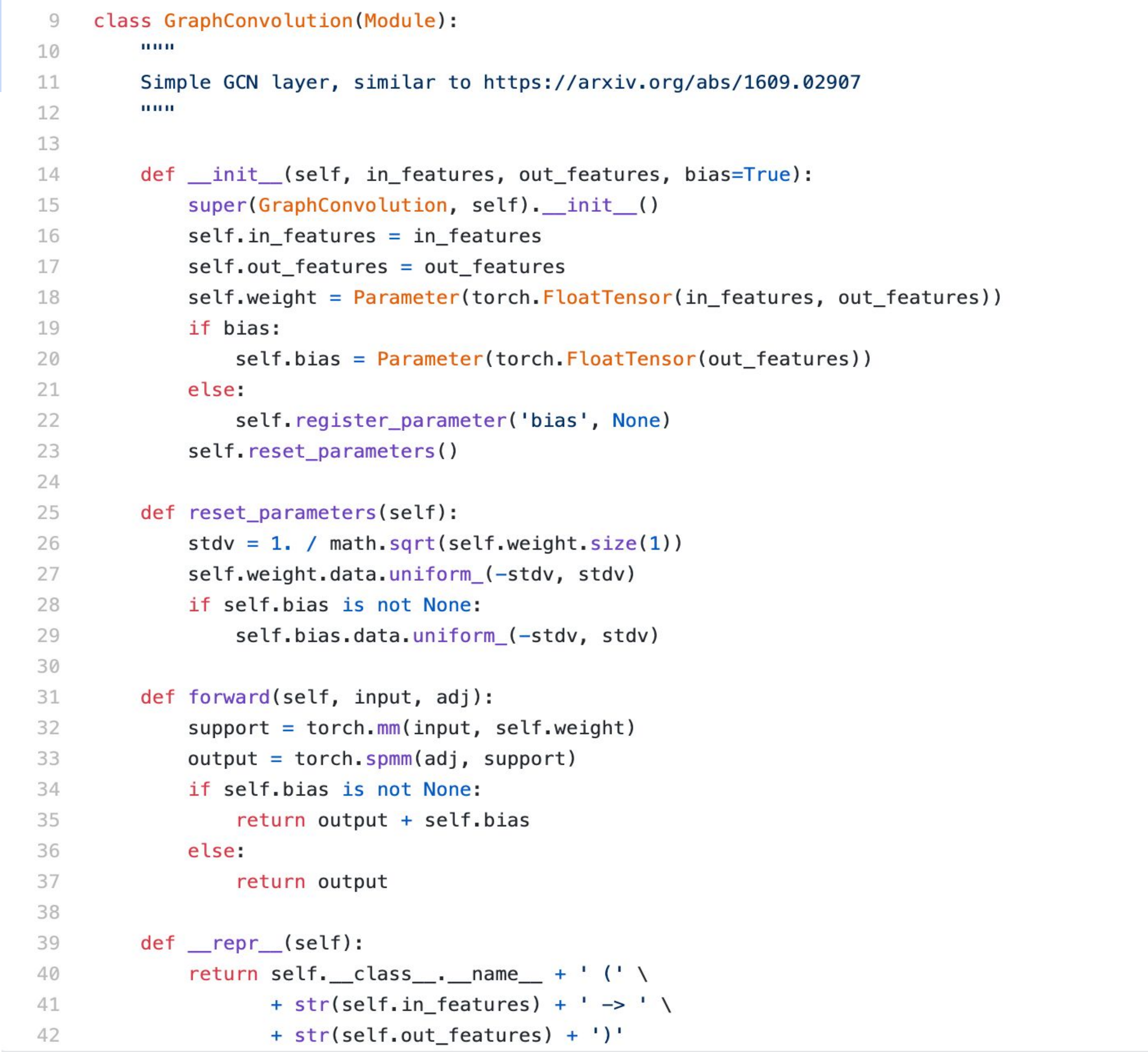
Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Graph Convolutional Network

Pytorch Implementation

https://github.com/tkipf/pygcn

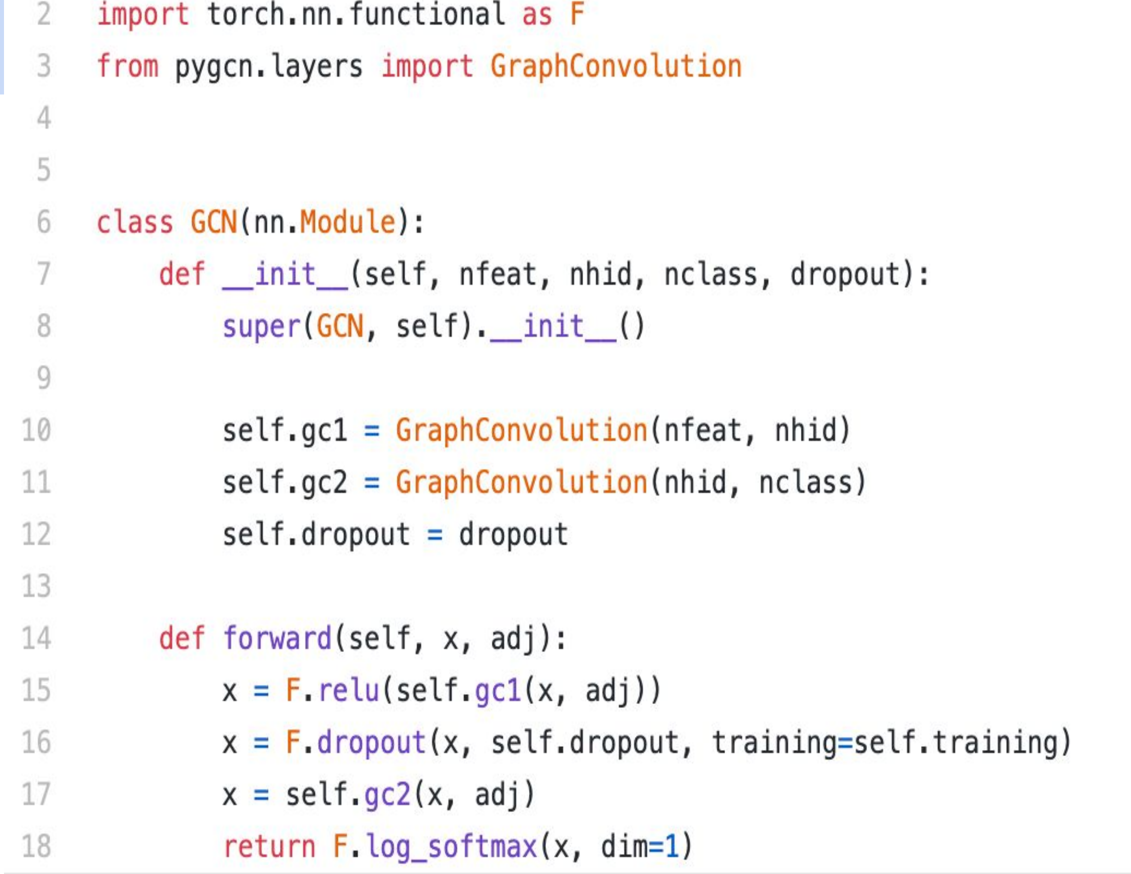
Layers.py



```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)
        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' (' \
               + str(self.in_features) + ' -> ' \
               + str(self.out_features) + ')'
```

Graph Convolutional Network

models.py



```python
import torch.nn.functional as F
from pygcn.layers import GraphConvolution


class GCN(nn.Module):
    def __init__(self, nfeat, nhid, nclass, dropout):
        super(GCN, self).__init__()

        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)
        self.dropout = dropout

    def forward(self, x, adj):
        x = F.relu(self.gc1(x, adj))
        x = F.dropout(x, self.dropout, training=self.training)
        x = self.gc2(x, adj)
        return F.log_softmax(x, dim=1)
```

$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) .$$

Graph Convolutional Network

Illustrative example of power of GNNs - Zachary's karate club network

This graph contains 34 nodes, connected by 154 (undirected and unweighted) edges. Every node is labeled by one of four classes.

Training setting

- Semi-supervised learning, taking 1 labeled sample per class
- Use a 3 layer GCN, randomly initialised weights

Figure 4 shows how the GCN can learn powerful feature embeddings for the nodes.

It separates the 4 communities using minimal supervision and the graph structure alone.

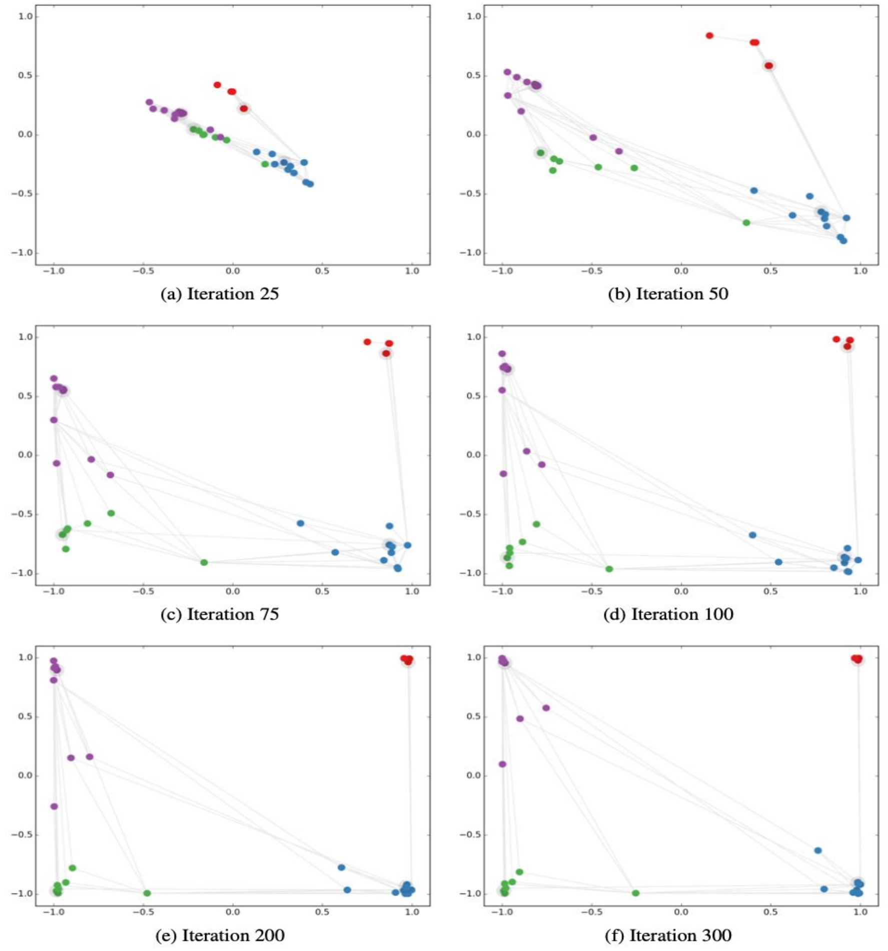


Figure 4: Evolution of karate club network node embeddings obtained from a GCN model after a number of semi-supervised training iterations. Colors denote class. Nodes of which labels were provided during training (one per class) are highlighted (grey outline). Grey links between nodes denote graph edges. Best viewed on a computer screen.

Inductive vs Transductive learning

Induction

- “Induce” rules, characteristics and patterns from the data to learn a model that will work on new inputs
- Result of inductive learning is a function that maps inputs to outputs
- Get good generalisation on unseen samples

Transduction

- Use the training data to make accurate predictions on the <u>specified set of unlabelled instances</u>
- It streamlines and focuses the process for a small set of target instances instead of trying to build a ‘general’ or universal model.

“When solving a problem of interest, do not solve a more general problem as an intermediate step. Try to get the answer that you really need but not a more general one.” - Vladimir Vapnik

GCN - Transductive approach, train-test data are parts of the same graph
GAT - Can be used for Inductive learning, Eg. on the PPI dataset: train-test data are 2 separate graphs

GCN - can’t have directed edges and edge features
GAT - can