Kipf, T., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks

Radim Špetlík

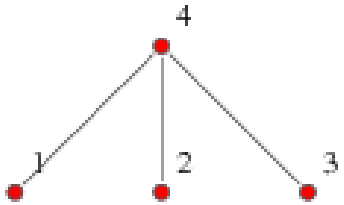
Czech Technical University in Prague

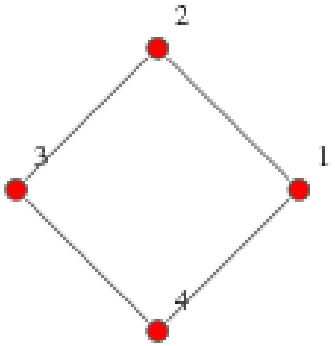
Overview

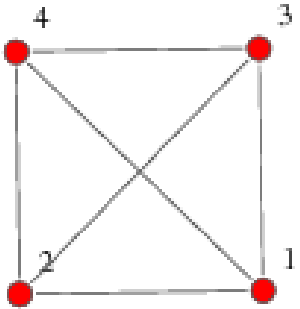
- **Kipf and Welling**
  - **use first order approximation in Fourier-domain to obtain an efficient linear-time graph-CNNs**
  - apply the approximation to the semi-supervised graph node classification problem

Graph Adjacency Matrix A

- symmetric, square matrix
- $A_{ij} = 1$ iff vertices v_i and v_j are incident
- $A_{ij} = 0$ otherwise







$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

http://mathworld.wolfram.com/AdjacencyMatrix.html

Graph Convolutional Network

- given a graph $G = (V, E)$, graph-CNN is a function which:
  - takes as input:
    - feature description $\boldsymbol{x_i} \in \mathbb{R}^D$ for every node i; summarized as $X \in \mathbb{R}^{N \times D}$, where N is number of nodes, D is number of input features
    - description of the graph structure in matrix form, typically an adjacency matrix A
  - produces:
    - node-level output $Z \in \mathbb{R}^{N \times F}$, where F is the number of output features per node

Graph Convolutional Network

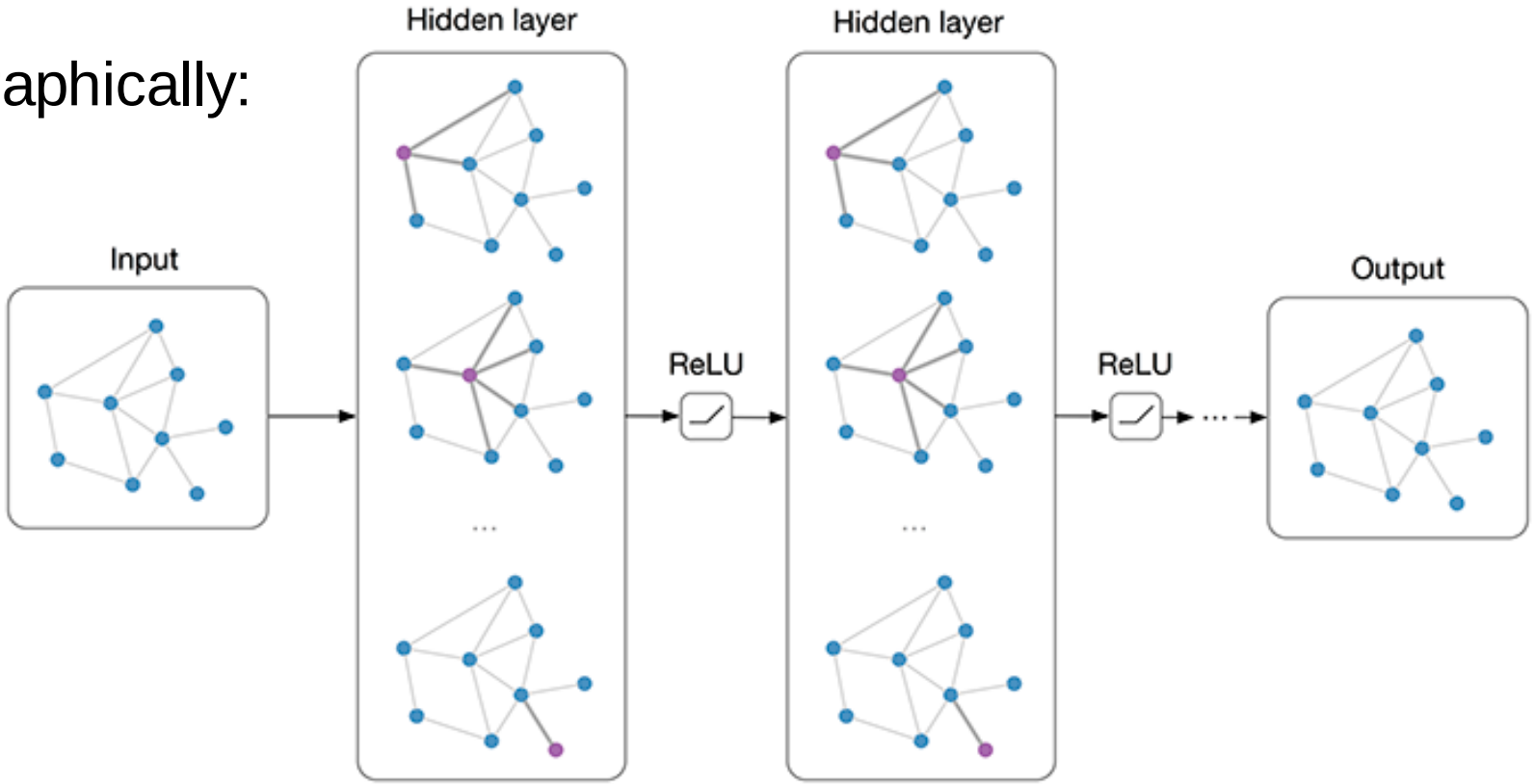
- is composed of non-linear functions

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

where $H^{(0)} = X$, and $H^{(L)} = Z$, and L is the number of layers.

Graph Convolutional Network

- graphically:



https://tkipf.github.io/graph-convolutional-networks/

Graph Convolutional Network

Let's start with a simple layer-wise propagation rule

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

where $W^{(l)} \in \mathbb{R}^{D_l \times D_{l+1}}$ is a weight matrix for the l-th neural network layer, $\sigma(\cdot)$ is a non-linear activation function, $A \in \mathbb{R}^{N \times N}$ is adjacency matrix, N is the number of nodes, $H^{(l)} \in \mathbb{R}^{N \times D_l}$

https://samidavies.wordpress.com/2016/09/20/whats-up-with-the-graph-laplacian/

Graph Convolutional Network

multiplication with A not enough, we're missing the node itself

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

we fix it by

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

where $\hat{A} = A + I$, I is the identity matrix

Graph Convolutional Network

$\hat{A}$ is typically not normalized; this multiplication

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

would change the scale of features $H^{(l)}$

we fix that by symmetric normalization, i.e. $\hat{D}^{-\frac{1}{2}}A\hat{D}^{-\frac{1}{2}}$, where $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$, $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$, producing

$$f\left(H^{(l)}, A\right) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}),$$

Graph Convolutional Network

$$f\left(H^{(l)}, A\right) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

Examining a single layer, single filter $\theta \in \mathbb{R}$, and a single node feature vector $\boldsymbol{x} \in \mathbb{R}^{D}$

$$\theta\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}\boldsymbol{x}$$

Graph Convolutional Network

$$\theta \widehat{D}^{-\frac{1}{2}} \hat{A} \widehat{D}^{-\frac{1}{2}} \boldsymbol{x}$$

$$\hat{A} = A + I,\ \widehat{D}_{ii} = \sum_j \hat{A}_{ij}$$

$$\theta \left(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) \boldsymbol{x}$$

… *renormalization trick*

Graph Convolutional Network

$$\theta\left(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)\boldsymbol{x}$$

$\theta = \theta_0' = -\theta_1'$

$$\theta_0'\boldsymbol{x} - \theta_1' D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\boldsymbol{x}$$

$L = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$

$$\theta_0'\boldsymbol{x} + \theta_1'(L - I)\boldsymbol{x}$$

Graph Convolutional Network

$$\boldsymbol{g_{\theta'}} \star \boldsymbol{x} \approx \theta'_0 \boldsymbol{x} + \theta'_1 (L - I)\boldsymbol{x}$$

$$\boldsymbol{g_{\theta'}} \star \boldsymbol{x} \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})\boldsymbol{x}$$

$\tilde{L} = c\,L - I \quad , c \in \mathbb{R}$

$$\boldsymbol{g_\theta} \star \boldsymbol{x} = U \boldsymbol{g_\theta} U^\top \boldsymbol{x}$$

Inverse Fourier transform – filtering – Fourier transform

Graph Convolutional Network

An efficient graph convolution approximation was performed when the multiplication

$$\widehat{D}^{-\frac{1}{2}}\hat{A}\widehat{D}^{-\frac{1}{2}}X\Theta$$

was interpreted as approximation of convolution in Fourier domain using Chebyshev polynomials.

$$O(N^2) \rightarrow O(|E|D_l D_{l+1})$$

where N is number of nodes, $|\mathrm{E}|$ is number of edges, D_l is number of input channels, D_{l+1} is number of output channels.

Overview

- **Kipf and Welling**
  - use first order approximation in Fourier-domain to obtain an efficient linear-time graph-CNNs
  - **apply the approximation to the semi-supervised graph node classification problem**

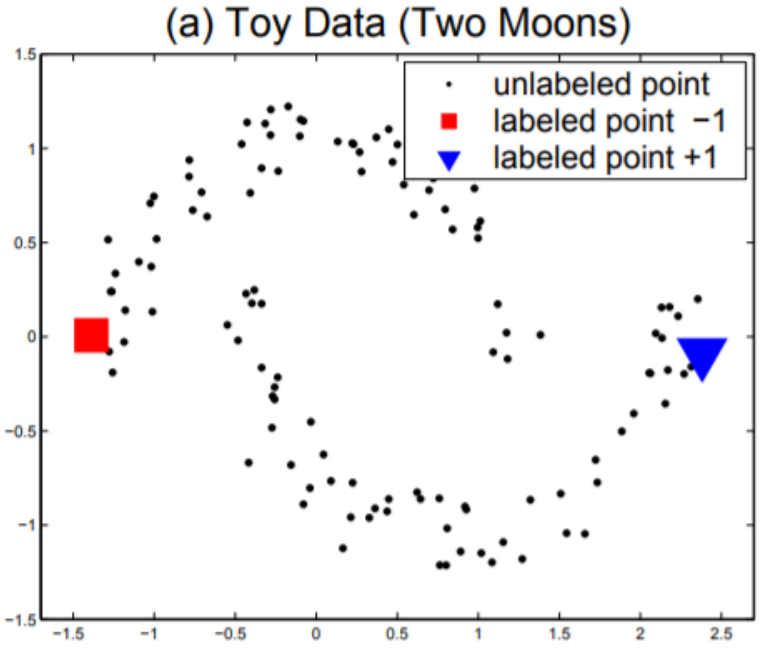
Semi-supervised Classification Task

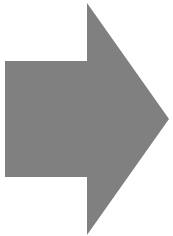
- given a point set $X = \{x_1, \dots, x_l, x_{l+1}, \dots, x_n\}$
- and a label set $L = \{1, \dots c\}$, where
  - first l points have labels $\{y_1, \dots, y_l\} \in L$
  - remaining points are unlabeled
  - c is the number of classes

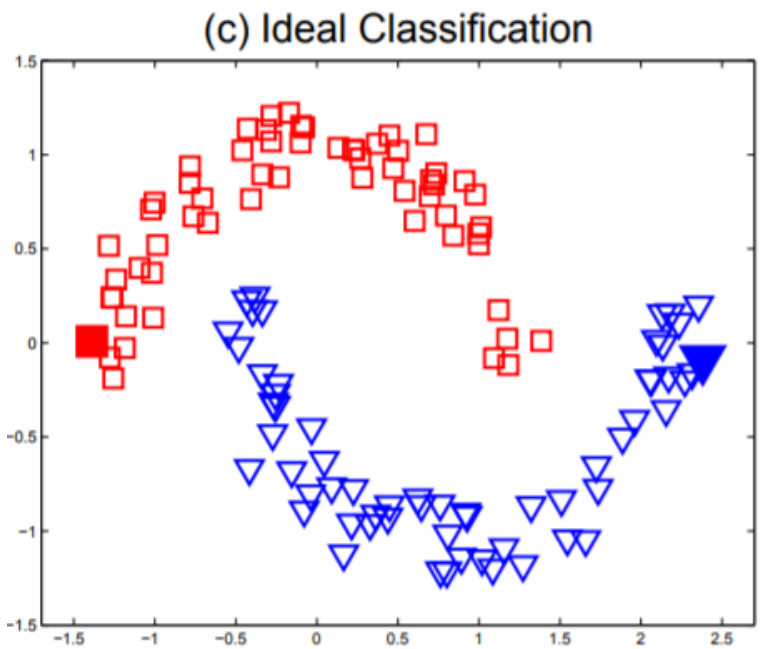
- the goal is to
  - predict the labels of the unlabeled points

Semi-supervised Classification Task

- graphically:







https://papers.nips.cc/paper/2506-learning-with-local-and-global-consistency.pdf

graph-CNN EXAMPLE

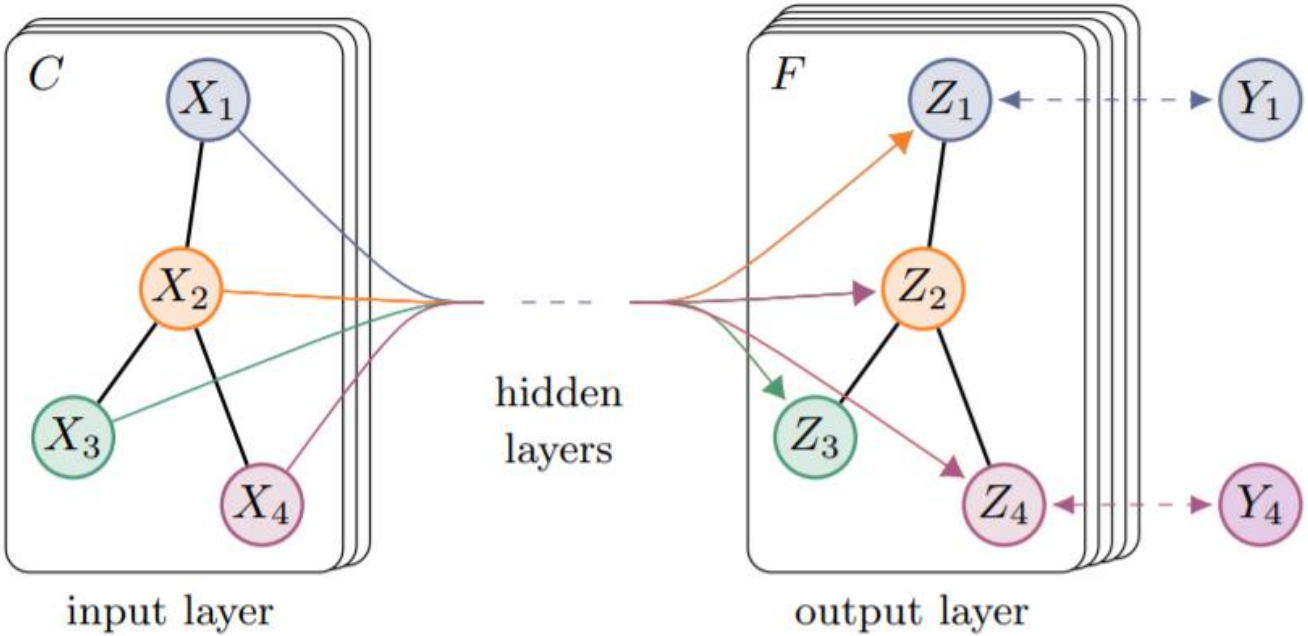
- example:
  - two-layer graph-CNN

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

where $W^{(0)} \in \mathbb{R}^{C\times H}$ with C input channels and H features maps, $W^{(1)} \in \mathbb{R}^{H\times F}$ with F output features per node

Graph Convolutional Network

- graphically:



https://arxiv.org/pdf/1609.02907.pdf

graph-CNN EXAMPLE

- objective function:
  - cross-entropy

$$L = -\sum_{l \in Y_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} ,$$

where Y_L is a set of node indices that have labels,
Z_{lf} is the element in the l-th row, f-th column of matrix Z,
ground truth: Y_{lf} is 1 if instance l comes from a class f.

graph-CNN EXAMPLE - RESULTS

- weights trained with gradient descent

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-CNN EXAMPLE - RESULTS

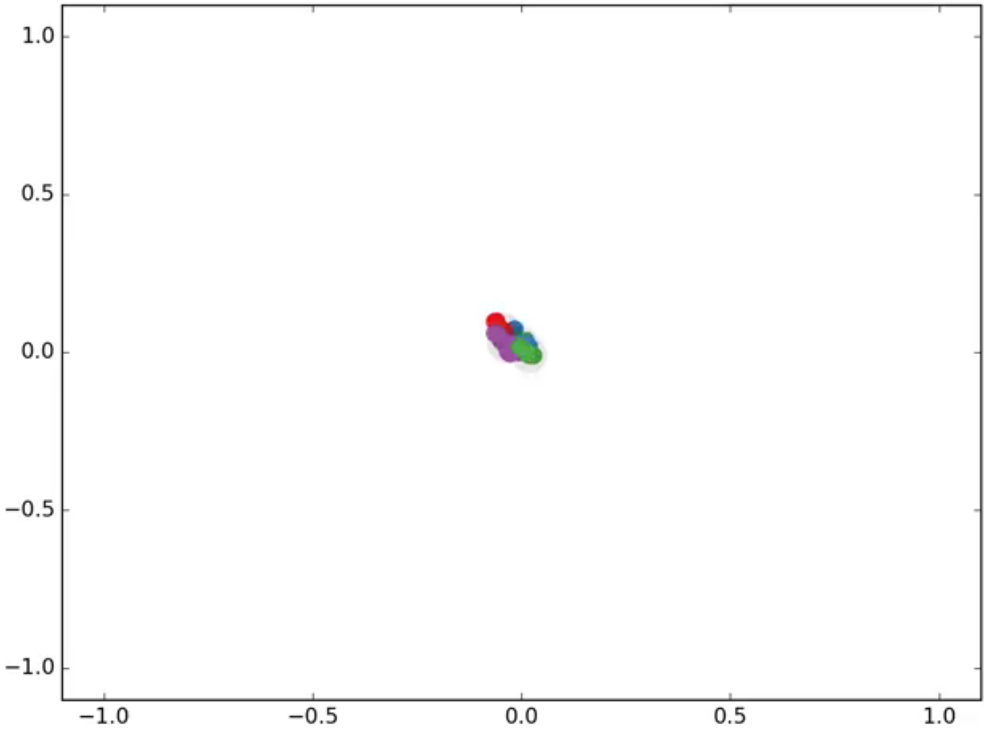
- different variants of propagation models

Table 3: Comparison of propagation models.

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K = 3$	$\sum_{k=0}^{K} T_k(\tilde{L}) X \Theta_k$	69.8	79.5	74.4
	$K = 2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) X \Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}} A D^{-\frac{1}{2}} X \Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

graph-CNN another EXAMPLE

- 3-layer GCN, “karate-club” problem, one labeled example per class:



300 training iterations

Limitations

- Memory grows linearly with data
- only works with undirected graph
- assumption of locality
- assumption of equal importance of self-connections vs. edges to neighboring nodes

$$\hat{A} = A + \lambda I$$

where λ is a learnable parameter.

Summary

- **Kipf and Welling**
  - use first order approximation in Fourier-domain to obtain an efficient linear-time graph-CNNs
  - apply the approximation to the semi-supervised graph node classification problem

Thank you very much for your time…

Answers to Questions

$$\tilde{A} = A + \lambda I_N$$

- The lambda parameter would control the influence of neighbouring edges vs. self-connections.
- How (or why) would the lambda parameter trade-off also between supervised and unsupervised learning?