Deep Learning

9. Graph Convolutions and Graph Attention

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

Tue. 21.4.	(1)	1. Supervised Learning (Review 1)
Tue. 28.4.	(2)	2. Neural Networks (Review 2)
Tue. 5.5.	(3)	3. Regularization for Deep Learning
Tue. 12.5.	(4)	4. Optimization for Training Deep Models
Tue. 19.5.	(5)	5. Convolutional Neural Networks
Tue. 26.5.	(6)	6. Recurrent Neural Networks
Tue. 2.6.	—	— *Pentecoste Break* —
Tue. 9.6.	(7)	7. Autoencoders
Tue. 16.6.	(8)	ctd.
Tue. 23.6.	(9)	8. Attention Layers
Tue. 30.6.	(10)	9. Graph Convolutions and Graph Attention
Tue. 7.7.	(11)	10. Generative Adversarial Networks
Tue. 14.7.	(12)	Q & A

Outline

1. Supervised Learning for Graphs
2. Recurrent Graph Networks
3. Graph Convolutions
4. Graph Attention
5. Set-valued Inputs (DeepSet)

Outline

Graphs

- **directed graph** $G := (V, E)$:

  any set V (called **vertices**) together with
  $E \subseteq V \times V$ (called **edges**)

- **undirected graph** $G := (V, E)$:

  any set V (called **vertices**) together with
  $E \subseteq \text{subsets}(V, \text{card} = 2)$ (called **edges**)

  - equivalent to a directed graph with symmetric edges,
    i.e., with $E \overset{!}{=} \{(w, v) \mid (v, w) \in E\}$.

- **vertex attributes** $x^{\text{vrt}} \in \mathbb{R}^{V \times M^{\text{vrt}}}$

- **edge attributes** $x^{\text{edge}} \in \mathbb{R}^{E \times M^{\text{edge}}}$

Some Notation for Graphs

- **neighborhood** of a vertex $v \in V$ in an undirected graph:

$$\mathcal{N}(v) := \{w \in V \mid \{v, w\} \in E\}$$

- **fanin** and **fanout** neighborhood of a vertex $v \in V$ in a directed graph:

$$\mathcal{N}_{\text{in}}(v) := \{w \in V \mid (w, v) \in E\}$$
$$\mathcal{N}_{\text{out}}(v) := \{w \in V \mid (v, w) \in E\}$$

- Denote the set of graphs with M^{vrt} vertex features and M^{edge} edge features (both possibly none) by

$$\text{graphs}(M^{\text{vrt}}, M^{\text{edge}})$$

Graph Classification

- Given a set $\mathcal{D}^{\text{train}} \subset \text{graphs}(M^{\text{vrt}}, M^{\text{edge}}) \times \mathbb{R}^O$ of pairs of graphs $x_n = (V_n, E_n)$ and targets $y_n \in \mathbb{R}^O$ and a pairwise loss $\ell : \mathbb{R}^O \times \mathbb{R}^O \to \mathbb{R}$ on targets

  find a prediction model

$$\hat{y} : \text{graphs}(M^{\text{vrt}}, M^{\text{edge}}) \to \mathbb{R}^O$$

  with minimal loss for test data $\mathcal{D}^{\text{test}} \subset \text{graphs}(M^{\text{vrt}}, M^{\text{edge}}) \times \mathbb{R}^O$ (from the same distribution):

$$\ell(\hat{y}; \mathcal{D}^{\text{test}}) := \frac{1}{|\mathcal{D}^{\text{test}}|} \sum_{(V,E,y) \in \mathcal{D}^{\text{test}}} \ell(y, \hat{y}(V, E))$$

Vertex Classification (aka **Node Classification**)

- vertex features x^{vrt} are partitoned into two groups, **vertex predictors** and **vertex targets**:

$$x^{\text{vrt}} = \text{concat}_2(\tilde{x}, y), \quad \tilde{x} \in \mathbb{R}^{V \times M}, y \in \mathbb{R}^{V \times O}$$

- given the graph (V, E),
  all vertex predictors $\tilde{x}$,
  a random subset of vertex targets $y|_{V^{\text{train}}}$, $V^{\text{train}} \subset V$, and
  a pairwise loss $\ell : \mathbb{R}^O \times \mathbb{R}^O \to \mathbb{R}$ on targets,
  find a prediction $\hat{y} \in \mathbb{R}^{V^{\text{test}} \times O}$ of the remaining vertex targets $V^{\text{test}} := V \setminus V^{\text{train}}$ s.t. their loss w.r.t. the true targets is minimal:

$$\ell(\hat{y}; y) := \frac{1}{|V^{\text{test}}|} \sum_{v \in V^{\text{test}}} \ell(y_v, \hat{y}_v)$$

- called a **transductive** problem, because
  - no model function $\hat{y}$ is sought, that predicts targets individually for every instance/node,
  - but just the target values for a fixed set of predictors predicted collectively

A New Problem?

- Q: Can we solve vertex classification like any other classification problem?

A New Problem?

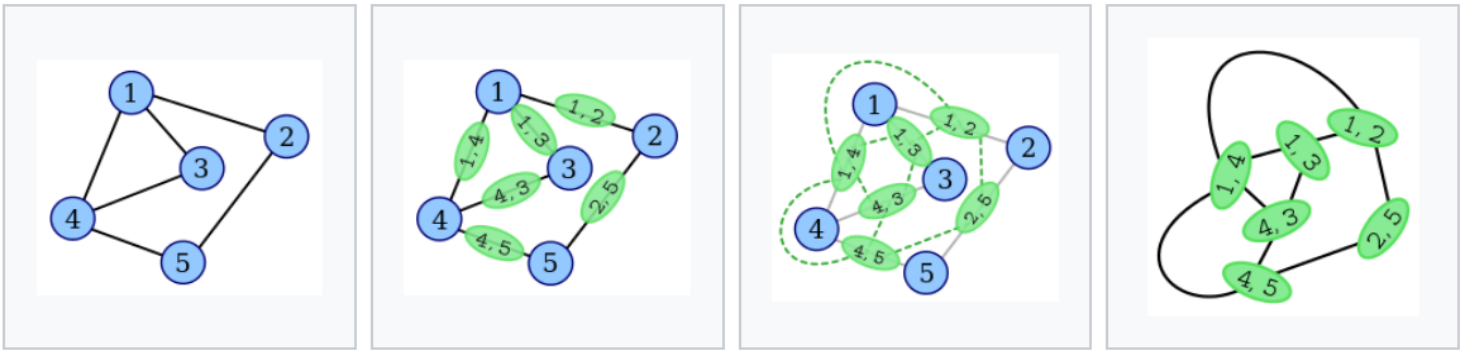
- Q: Can we solve vertex classification like any other classification problem?
- Is there also an edge classification problem?

Line Graphs

- **line graph** of a graph $G := (V, E)$:

$$\text{line}(V, E) := (E, \{\{e, f\} \subset E \mid |e \cap f| = 1\})$$

  - each original edge becomes a vertex.
  - two edges are connected by an edge, if originally they are connected by a vertex.



Graph G | Vertices in L(G) constructed from edges in G | Added edges in L(G) | The line graph L(G)

[source: wikipedia/line_graph]

Line Graphs / Features

- $G := (V, E), \tilde{G} := \text{line}(V, E)$
- edges become vertices $\rightsquigarrow$ edge features become vertex features:

$$\tilde{x}^{\text{vrt}}_e := x^{\text{edge}}_e, \quad e \in E$$

- vertices become edges $\rightsquigarrow$ vertex features become edge features:

$$\tilde{x}^{\text{edge}}_{e,f} := x^{\text{vrt}}_{e \cap f} \quad e, f \in E$$

  - beware, a vertex can become many edges, thus all the edges represented by the same original vertex have the same features.

Edge Classification?

- Is there also an edge classification problem?
- practically: sure.
- methododically: no, it is not a new problem.
  - edge classification is the same as vertex classification for the line graph.

Outline

Graph Neural Networks

- recurrent latent vertex features $z_v \in \mathbb{R}^K$:

$$z_v \stackrel{!}{=} \sum_{w \in \mathcal{N}(v)} h(z_v, z_w, x_v^{\text{vrt}}, x_w^{\text{vrt}}, x_{v,w}^{\text{edge}}) \tag{1}$$

$$y_v := g(z_v, x_v^{\text{vrt}})$$

  - g, h fully connected neural networks
  - Q: How can we compute the latent features?

- Scarselli et al. [2008]

Graph Neural Networks

- recurrent latent vertex features $z_v \in \mathbb{R}^K$:

$$z_v \stackrel{!}{=} \sum_{w \in \mathcal{N}(v)} h(z_v, z_w, x_v^{\text{vrt}}, x_w^{\text{vrt}}, x_{v,w}^{\text{edge}}) \tag{1}$$

$$y_v := g(z_v, x_v^{\text{vrt}})$$

  - g, h fully connected neural networks
  - z_v have to be computed as fixpoints of eq. 1
  - thus h needs to be chosen with care: to be contractive.

- Scarselli et al. [2008]

Gated Graph Sequence Neural Networks

- two types of latent vertex features $z_v, h_v \in \mathbb{R}^K$
  - stacking several layers: $z_v^t, h_v^t,\ t = 1, \ldots, T$
    $h_v^0 := x_v^{\text{vrt}}$, padded with zeros.

- latent vertex features $z^t \in \mathbb{R}^{V \times K}$:

$$z_v^{t+1} := (h^t)^T (x_{v,.,.}^{\text{edge}} w^{\text{out}} + x_{.,v,.}^{\text{edge}} w^{\text{in}}), \quad w^{\text{out}}, w^{\text{in}} \in \mathbb{R}^{M^{\text{edge}}}$$

  - representing the edge features as array $x^{\text{edge}} \in \mathbb{R}^{V \times V \times M^{\text{edge}}}$ (being zero for non-edges)
  - based on **all** previous latent vertex features $h^t \in \mathbb{R}^{V \times K}$

- latent vertex features $h^t \in \mathbb{R}^{V \times K}$, computed via a GRU RNN:

$$h_v^t := \text{GRU}(z_v^t, h_v^{t-1})$$

  - using z^t instead of the sequence inputs x^t as done usually.
  - being applied to each vertex in isolation!

- Li et al. [2015]

Gated Graph Sequence Neural Networks

- latent vertex features $z_v \in \mathbb{R}^K$:

$$z_v^{t+1} := (h^t)^T (x_{v,.,.}^{\text{edge}} w^{\text{out}} + x_{.,v,.}^{\text{edge}} w^{\text{in}}), \quad w^{\text{out}}, w^{\text{in}} \in \mathbb{R}^{M^{\text{edge}}}$$

- make it simple: no edge features,
  i.e., $M^{\text{edge}} := 1$ and $x_{.,.,1}^{\text{edge}} = A$ is just the adjacency matrix.

$$z_v^{t+1} = w^{\text{out}} \sum_{u \in \mathcal{N}_{\text{out}}(v)} h_u^t + w^{\text{in}} \sum_{u \in \mathcal{N}_{\text{in}}(v)} h_u^t$$

  - i.e, this is DeepSet for the fanin and fanout neighborhoods.
    (before DeepSet was invented, that is.)

Outline

Papers on Graph Convolutions (source: Zhang et al. [2020])

Method	Type	Convolution	Readout	T.C.	M.G.	Other Characteristics
Bruna *et al.* [40]	Spectral	Interpolation kernel	Hierarchical clustering + FC	$O(N^3)$	No	-
Henaff *et al.* [41]	Spectral	Interpolation kernel	Hierarchical clustering + FC	$O(N^3)$	No	Constructing the graph
ChebNet [42]	Spectral/Spatial	Polynomial	Hierarchical clustering	$O(M)$	Yes	-
Kipf&Welling [43]	Spectral/Spatial	First-order	-	$O(M)$	-	-
CayletNet [44]	Spectral	Polynomial	Hierarchical clustering + FC	$O(M)$	No	-
GWNN [45]	Spectral	Wavelet transform	-	$O(M)$	No	-
Neural FPs [46]	Spatial	First-order	Sum	$O(M)$	Yes	-
PATCHY-SAN [47]	Spatial	Polynomial + an order	An order + pooling	$O(M \log N)$	Yes	A neighbor order
LGCN [48]	Spatial	First-order + an order	-	$O(M)$	Yes	A neighbor order
SortPooling [49]	Spatial	First-order	An order + pooling	$O(M)$	Yes	A node order
DCNN [50]	Spatial	Polynomial diffusion	Mean	$O(N^2)$	Yes	Edge features
DGCN [51]	Spatial	First-order + diffusion	-	$O(N^2)$	-	-
MPNNs [52]	Spatial	First-order	Set2set	$O(M)$	Yes	A general framework
GraphSAGE [53]	Spatial	First-order + sampling	-	$O(Ns^L)$	Yes	A general framework
MoNet [54]	Spatial	First-order	Hierarchical clustering	$O(M)$	Yes	A general framework
GNs [9]	Spatial	First-order	A graph representation	$O(M)$	Yes	A general framework
Kearnes *et al.* [55]	Spatial	Weave module	Fuzzy histogram	$O(M)$	Yes	Edge features
DiffPool [56]	Spatial	Various	Hierarchical clustering	$O(N^2)$	Yes	Differentiable pooling
GAT [57]	Spatial	First-order	-	$O(M)$	Yes	Attention
GaAN [58]	Spatial	First-order	-	$O(Ns^L)$	Yes	Attention
HAN [59]	Spatial	Meta-path neighbors	-	$O(M_\phi)$	Yes	Attention
CLN [60]	Spatial	First-order	-	$O(M)$	-	-
PPNP [61]	Spatial	First-order	-	$O(M)$	-	Teleportation connections
JK-Nets [62]	Spatial	Various	-	$O(M)$	Yes	Jumping connections
ECC [63]	Spatial	First-order	Hierarchical clustering	$O(M)$	Yes	Edge features
R-GCNs [64]	Spatial	First-order	-	$O(M)$	-	Edge features
LGNN [65]	Spatial	First-order + LINE graph	-	$O(M)$	-	Edge features
PinSage [66]	Spatial	Random walk	-	$O(Ns^L)$	-	Neighborhood sampling
StochasticGCN [67]	Spatial	First-order + sampling	-	$O(Ns^L)$	-	Neighborhood sampling
FastGCN [68]	Spatial	First-order + sampling	-	$O(NsL)$	Yes	Layer-wise sampling
Adapt [69]	Spatial	First-order + sampling	-	$O(NsL)$	Yes	Layer-wise sampling
Li *et al.* [70]	Spatial	First-order	-	$O(M)$	-	Theoretical analysis
SGC [71]	Spatial	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GFNN [72]	Spatial	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GIN [73]	Spatial	First-order	Sum + MLP	$O(M)$	Yes	Theoretical analysis
DGI [74]	Spatial	First-order	-	$O(M)$	Yes	Unsupervised training

Deep Locally Connected Networks

- the vertices of the graph are partitioned into S clusters $C_1, \ldots, C_S$
- vertex cluster latent features $z_s \in \mathbb{R}^K$:

$$z^{\ell+1}_{s,k} := \sum_{v \in C_s} h(\sum_{j=1}^{K} W^{\ell+1}_{k,j} z^{\ell}_{v,j}) \quad s = 1, \ldots, S,\ j = 1, \ldots, K,\ W^{\ell+1} \in \mathbb{R}^{K \times K}$$

  - $h : \mathbb{R}^K \to \mathbb{R}^K$ is a suitable function (unspecified)
  - What actually does $h(\sum_{j=1}^{K} W^{\ell}_{k,j} z^{\ell}_{v,j})$?

- Bruna et al. [2013]

Deep Locally Connected Networks

- the vertices of the graph are partitioned into S clusters $C_1, \ldots, C_S$
- vertex cluster latent features $z_s \in \mathbb{R}^K$:

$$z_{s,k}^{\ell+1} := \sum_{v \in C_s} h(\sum_{j=1}^{K} W_{k,j}^{\ell+1} z_{v,j}^{\ell}) \quad s = 1, \ldots, S,\ j = 1, \ldots, K,\ W^{\ell+1} \in \mathbb{R}^{K \times K}$$

  - $h : \mathbb{R}^K \to \mathbb{R}^K$ is a suitable function (unspecified)
  - instead of summing over C_s, the paper talks just about "pooling".
- vertex clusters are assembled to a new graph with weighted edges, by summing and normalizing the weights between vertices of each two clusters.
- Bruna et al. [2013]

Spectral Networks

- vertex latent features $z \in \mathbb{R}^{V \times K}$:

$$z^{\ell+1}_{.,k} := a(\sum_{j=1}^{K} Q \operatorname{diag}(W^{\ell+1}_{k,j}) Q^T z^{\ell}_{.,j}), \quad W^{\ell-1}_{k,j} \in \mathbb{R}^{V}$$

  - $x^{\text{edge}} \in \mathbb{R}^{V \times V}$ symmetric, $x^{\text{edge}} \geq 0$ **edge weights**
  - $\text{Lap} := \operatorname{diag}(x^{\text{edge}} \mathbb{1}) - x^{\text{edge}}$ **Weighted Graph Laplacian**
  - $Q \in \mathbb{R}^{V \times V}$ the eigenvector matrix of the Laplacian:
    $\text{Lap} \stackrel{!}{=} Q \operatorname{diag}(\lambda) Q^T, \quad \lambda \in \mathbb{R}^{V}$

- Bruna et al. [2013]

Spectral Networks / Filter Parametrization

- how are the filters $W_{k,j}^{\ell-1} \in \mathbb{R}^V$ parametrized $(k, j = 1, \ldots, K)$?
  - let's use just $W \in \mathbb{R}^V$ to denote $W_{k,j}^{\ell-1}$ on this slide
- non-parametric filter (not used):

$$W = \theta \in \mathbb{R}^V$$

- interpolation from fixed size filter $\theta \in \mathbb{R}^L$, $L \ll V$ with a cubic spline:

$$W(\theta) = \mathrm{qsk}(V)\theta, \quad \mathrm{qsk}(V) \in \mathbb{R}^{V \times L} \text{ cubic spline kernel}$$

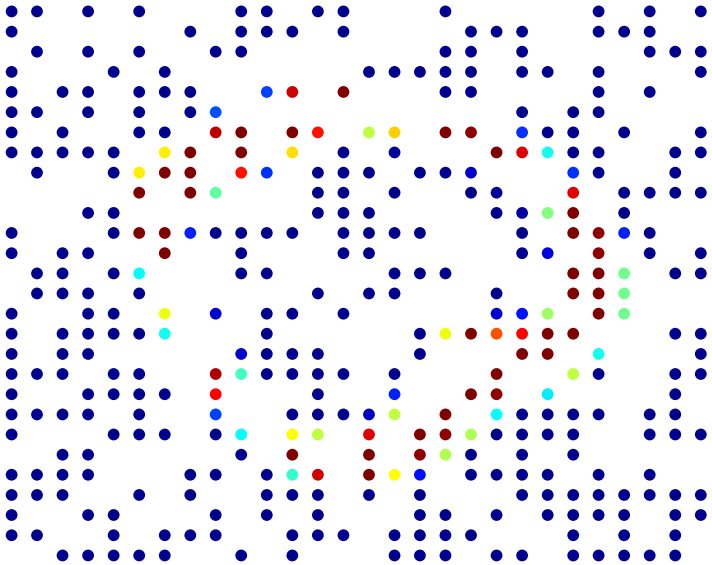
- polynom in the eigenvalues $\lambda \in \mathbb{R}^V$ of the Laplacian:

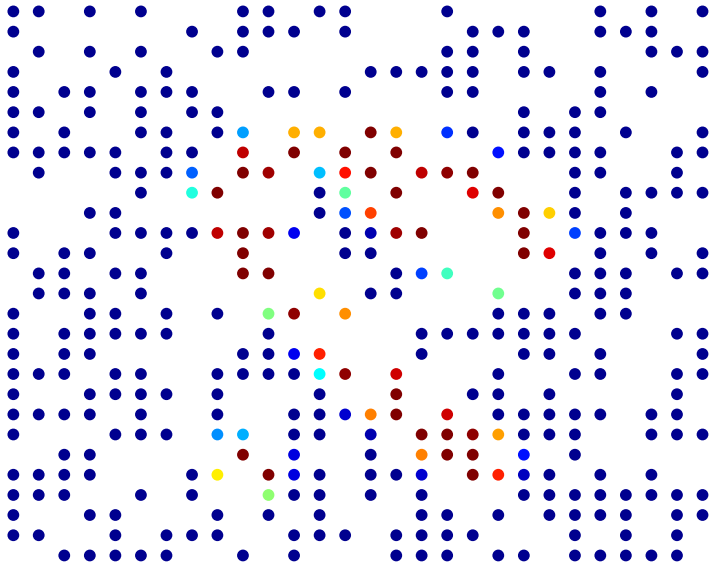
$$W(\lambda; \theta) = \sum_{d=0}^{D-1} \theta_d \lambda^d, \quad \theta \in \mathbb{R}^D, D \in \mathbb{N} \text{ degree}$$

  - Defferrard et al. [2016]

Note: Here λ^d denotes elementwise powers: $\lambda^d := (\lambda_v^d)_{v=1:V}$.

Experiments / Subsampled MNIST





[source: Bruna et al. [2013]]

Q: What you see?

Experiments / Subsampled MNIST

Table 1: Classification results on MNIST subsampled on 400 random locations, for different architectures. FCN stands for a fully connected layer with N outputs, LRFN denotes the locally connected construction from Section 2.3 with N outputs, MPN is a max-pooling layer with N outputs, and SPN stands for the spectral layer from Section 3.2.

method	Parameters	Error
Nearest Neighbors	N/A	4.11
400-FC800-FC50-10	$3.6 \cdot 10^5$	1.8
400-LRF1600-MP800-10	$7.2 \cdot 10^4$	1.8
400-LRF3200-MP800-LRF800-MP400-10	$1.6 \cdot 10^5$	**1.3**
400-SP1600-10 ($d_1 = 300$, $q = n$)	$3.2 \cdot 10^3$	2.6
400-SP1600-10 ($d_1 = 300$, $q = 32$)	$1.6 \cdot 10^3$	2.3
400-SP4800-10 ($d_1 = 300$, $q = 20$)	$5 \cdot 10^3$	1.8

[source: Bruna et al. [2013]]

Outline

Graph Attention Networks (GAT)

- restricted multi-head attention over the vertex neighborhood:

$$\text{attnres}(X; \mathcal{N})_{v,u} := \mathbb{I}(u \in \mathcal{N}(v))\ \text{attn}(X)_{v,u}, \quad u, v \in V$$

- using the vanilla attention mechanism:

$$\begin{aligned}
\text{attn}(X; w, v) &:= a(XW^T VX^T) \\
\text{sha}(X; w, v, u) &:= \text{attnres}(X; w, v)\,(XU^T) \\
\text{mha}(X) &:= \text{concat}_2((\text{sha}(X_{1:V,\text{slice}(h)}; W^h, V^h, U^h))_{h=1:H})Q^T, \\
& \qquad Q \in \mathbb{R}^{K^{\text{out}} \times HK}
\end{aligned}$$

- with $X := x^{\text{edge}} \in \mathbb{R}^{V \times M^{\text{edge}}}$ the vertex features.
- Veličković et al. [2017]

Model Configuration

- 2 GAT layers
  1. 8 heads à 8 features, exponential linear unit
  2. 1 head à O features, softmax

- L2 regularization $\lambda = 0.0005$,

  Dropout $p = 0.6$ for both layers' inputs and
  both layers' attention weights.
  (= random sub-neighborhood)

- has been optimized for first dataset (Cora),
  used unchanged for the other two.

- # parameters: Cora: 276k
  Citeseer: 712k
  Pubmed: 97k

Note: Exponential linear unit (ELU): $\mathrm{elu}(z) := \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{else} \end{cases}$.

Experiments / Datasets

	Cora	Citeseer	Pubmed	PPI
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

[source: Veličković et al. [2017]]

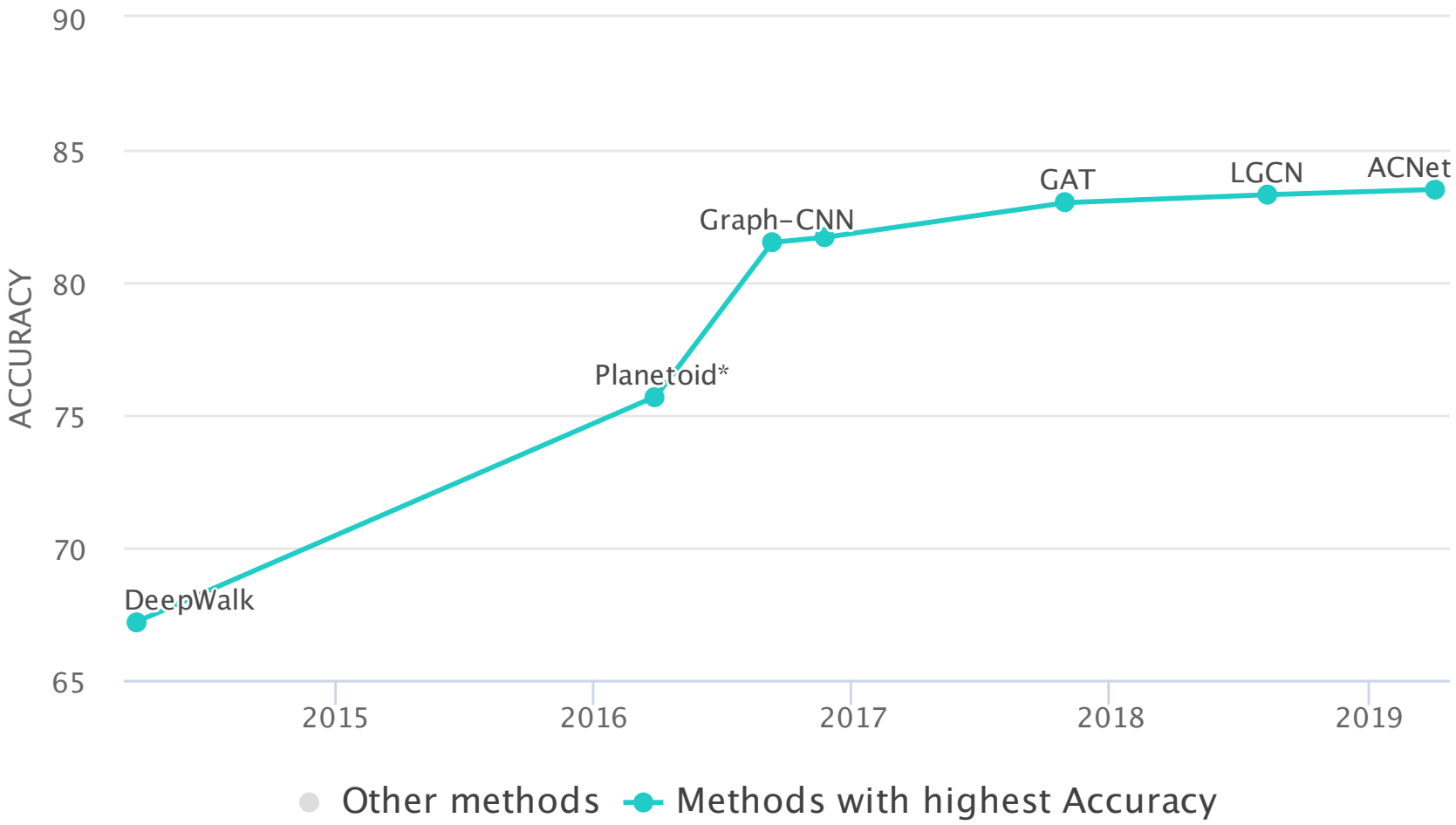
Experiments / Results

	Transductive		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	**79.0** $\pm 0.3\%$
GAT (ours)	**83.0** $\pm 0.7\%$	**72.5** $\pm 0.7\%$	**79.0** $\pm 0.3\%$

[source: Veličković et al. [2017]]

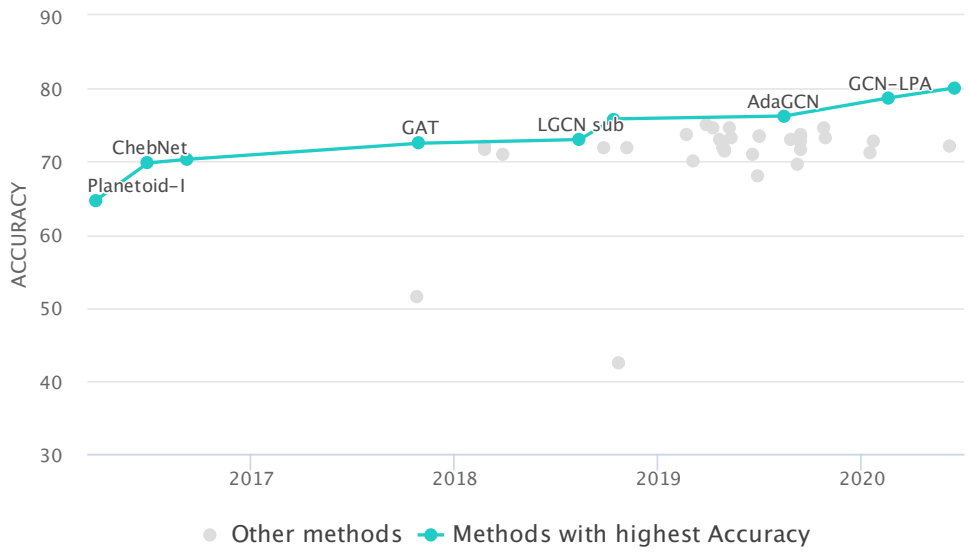
Note: shown is microaveraged F1.

Experiments / Results

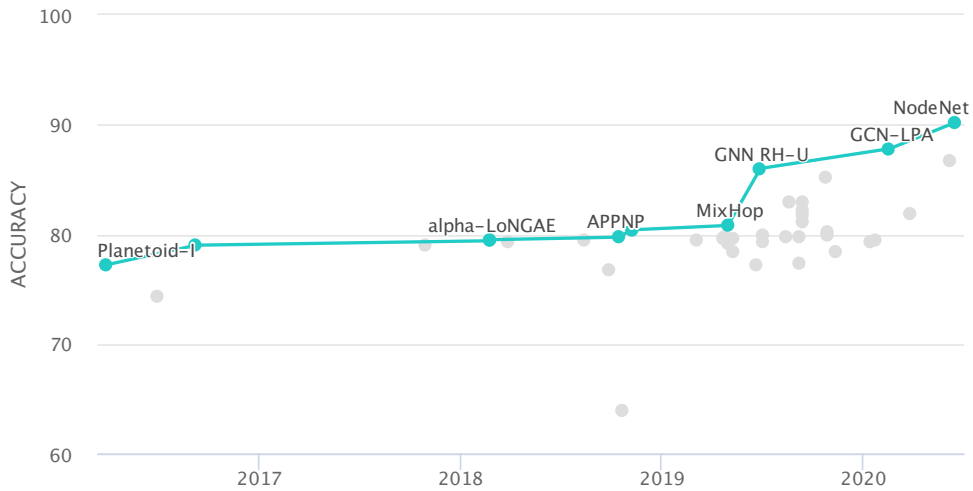


[source: paperswithcode.com]

Experiments / Results



[source: paperswithcode.com]



Outline

Set Function Representation Theorem

- let $\mathcal{X}, \mathcal{Y}$ be any sets, i.e., $\mathcal{X} := \mathbb{R}^M$, $\mathcal{Y} := \mathbb{R}^O$.
- any function on (finite/countable) sets of elements from $\mathcal{X}$ to $\mathcal{Y}$,

$$f : \text{sets}(\mathcal{X}) \to \mathcal{Y}$$

can be written as

$$f(X) = g(\sum_{x \in X} h(x))$$

for two suitable functions $h : \mathcal{X} \to \mathbb{R}^K, g : \mathbb{R}^K \to \mathcal{Y}$
and a suitable $K \in \mathbb{N}$.

Set Function Representation Theorem / Proof

$\Leftarrow$:

$$f(\pi(X)) = g(\sum_{x \in \pi(X)} h(x)) = g(\sum_{x \in X} h(x)) = f(X)$$

$\Rightarrow$: here only for the case that $\mathcal{X}$ is countable,
i.e., there exists a injective map $c : \mathcal{X} \to \mathbb{N}$:

$$h(x) := 4^{-c(x)}, \quad K := 1$$

$$\rightsquigarrow \quad z(X) := \sum_{x \in X} h(x) \text{ is injective (4-ary code of } X)$$

$$g(z) := f(z^{-1}(z))$$

and then:

$$g(\sum_{x \in \pi(X)} h(x)) = g(z(X)) = f(z^{-1}(z(X))) = f(X)$$

DeepSet

- invariant layer:
  - input: $x \in \mathbb{R}^{*\times M}$ ($\hat{=}\{x_1, x_2, \ldots, x_T\}, x_t \in \mathbb{R}^M$, number T of elements variable)
  - output: $z \in \mathbb{R}^O$

$$z(x) := g(\sum_{t=1}^{T} h(x_{t,.}))$$

$$h : \mathbb{R}^M \to \mathbb{R}^K \quad \textbf{element encoder}$$

$$g : \mathbb{R}^K \to \mathbb{R}^O \quad \textbf{decoder}$$

  - aggregation of encoded elements by sum, mean, or max.
- Zaheer et al. [2017]

Equivariant Layer

- equivariant layer:
  - input: $x \in \mathbb{R}^*$ ($\hat{=}\{x_1, x_2, \ldots, x_T\}, x_t \in \mathbb{R}$, number T of elements variable)
  - output: $z \in \mathbb{R}^*$ with same length

$$z(x) := a(cx + d(\max x)\mathbb{1}), \quad c, d \in \mathbb{R}$$

  - formulated for $x \in \mathbb{R}^M$, i.e., for M scalar elements
  - also $a(cx + d\mathbb{1}\mathbb{1}^T x)$ (sum) or $a(cx + d\frac{1}{|x|}\mathbb{1}\mathbb{1}^T x)$ (mean)

Summary

- **(Transductive) vertex classification** in a graph:
  given some labeled vertices, predict the targets/labels of the others.
- Early attempts to build neural networks for graphs used **recurrent approaches**
- **Graph Convolutions** aggregate vertex features
  - in the **spectral domain**:
    over vertex neighborhoods in the **Eigenspace of the graph Laplacian**.
  - in the **spatial domain**: over **vertex neighborhoods** in the graph directly.
- **Graph Attention** applies vanilla multi-head attention to the vertices, **restricting it to their neighborhoods**.
- Both, graph convolution networks and graph attention networks can be learned with **vanilla backpropagation**.
- Graph Attention Networks provide some of the best models currently for vertex classification.

Further Readings

- graph convolutions are not yet covered by the textbooks.
- see the referenced papers for details.
- surveys: Zhang et al. [2020]; Wu et al. [2020]; Kinderkhedia [2019]

References

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

Mital Kinderkhedia. Learning Representations of Graph Data–A Survey. *arXiv preprint arXiv:1906.02989*, 2019.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep Sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc., 2017.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 2020.