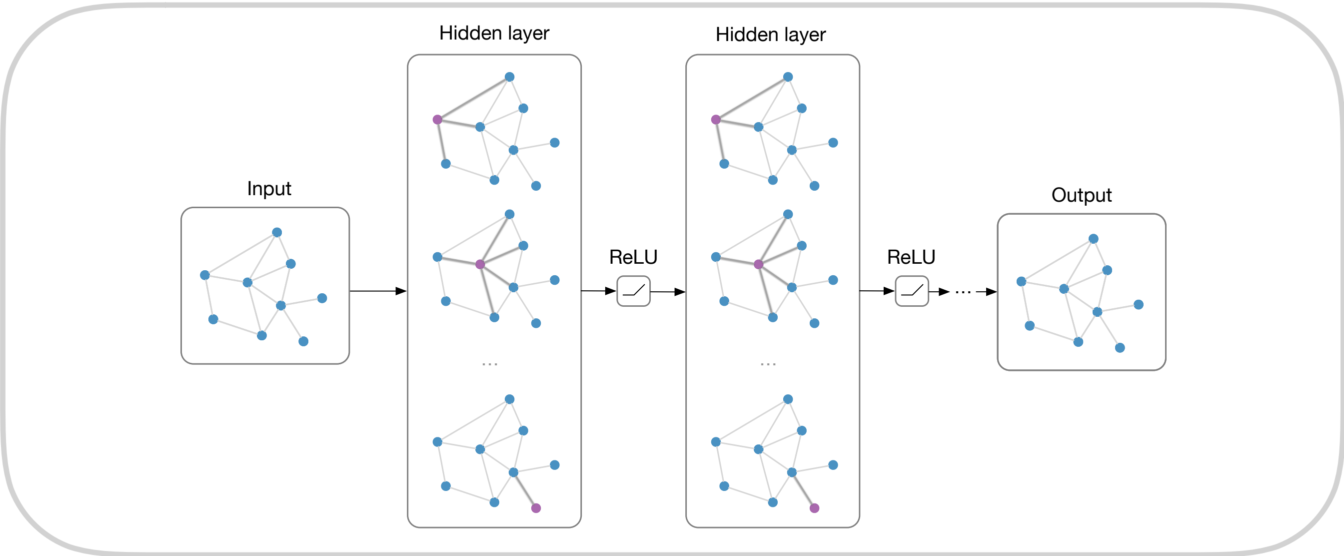
Deep Learning on Graphs with Graph Convolutional Networks



Thomas Kipf, 6 April 2017

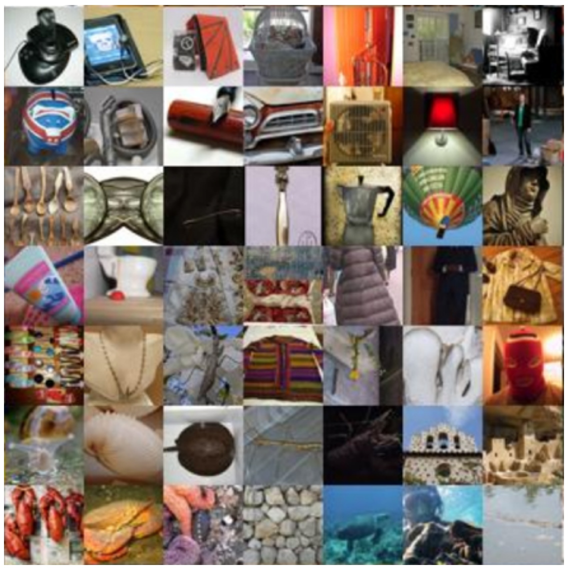
joint work with Max Welling (University of Amsterdam)

Universiteit van Amsterdam

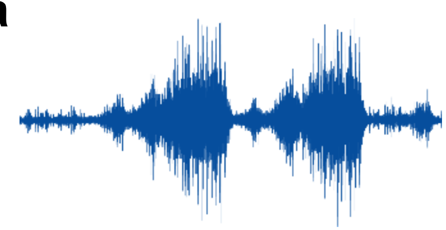
AMLAB Amsterdam Machine Learning Lab

The success story of deep learning

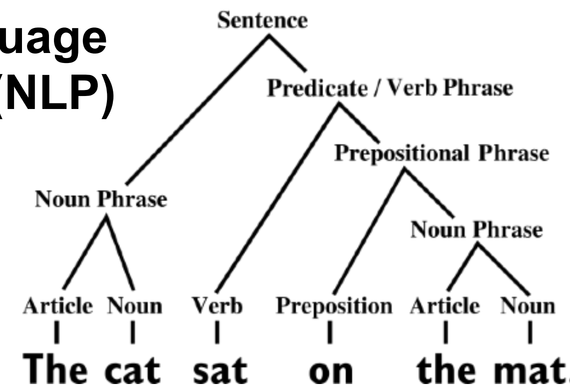
IMAGENET



Speech data



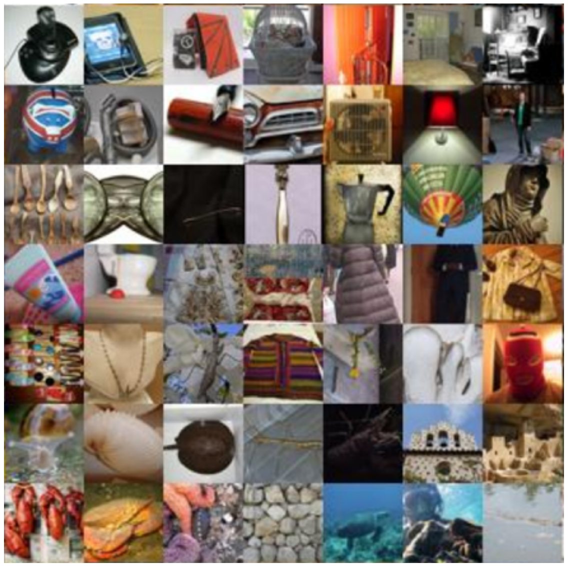
Natural language processing (NLP)



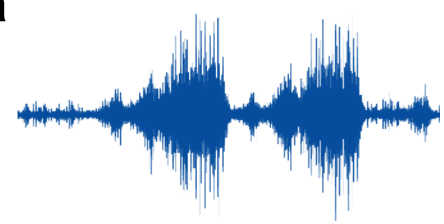
...

The success story of deep learning

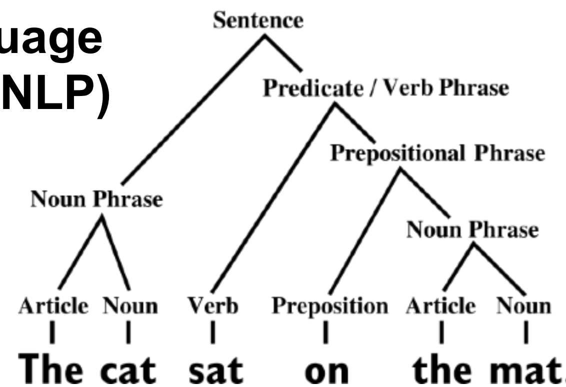
IMAGENET



Speech data



Natural language processing (NLP)



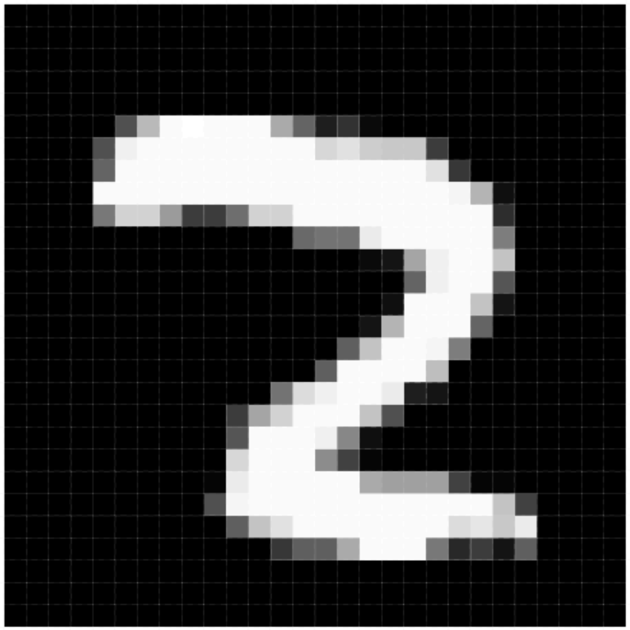
…

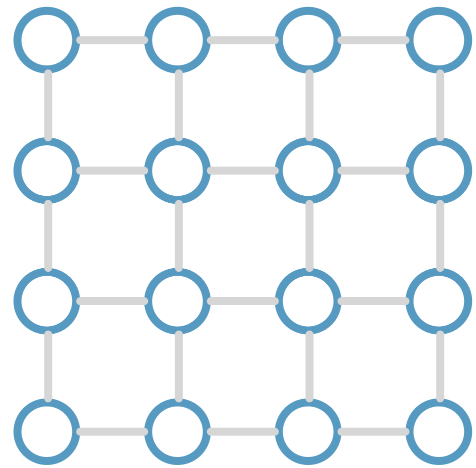
Deep neural nets that exploit:

- translation invariance (weight sharing)
- hierarchical compositionality

Recap: Deep learning on Euclidean data

Euclidean data: grids, sequences…

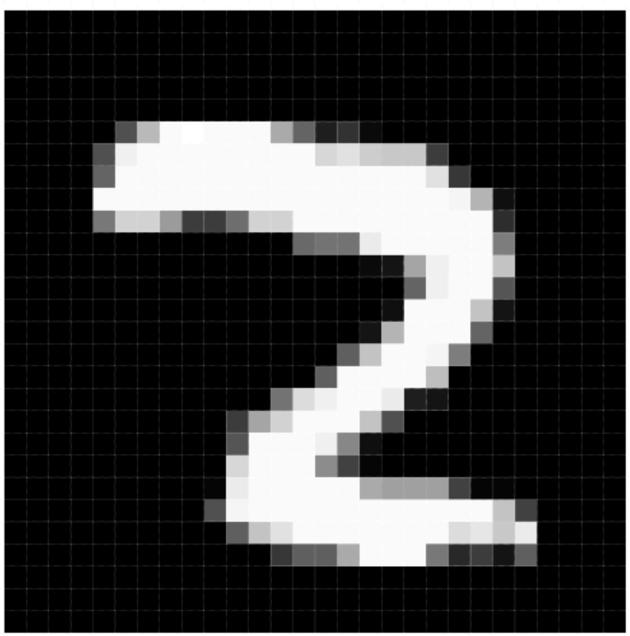


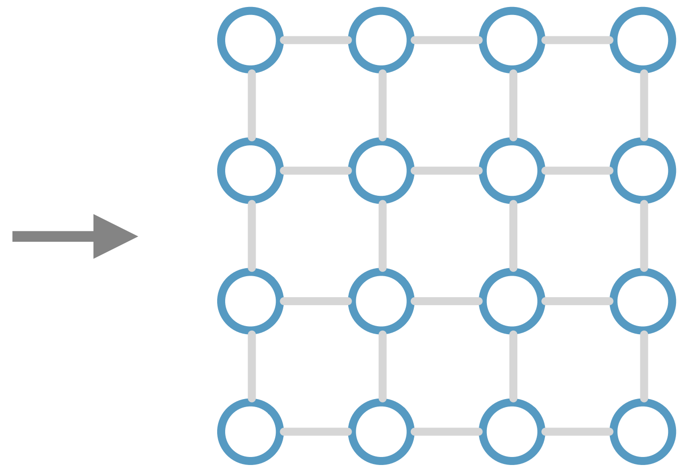


2D grid

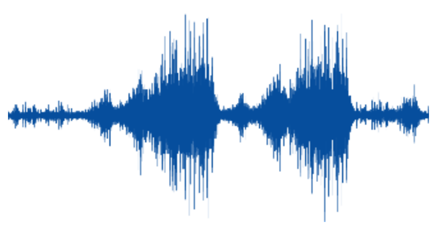
Recap: Deep learning on Euclidean data

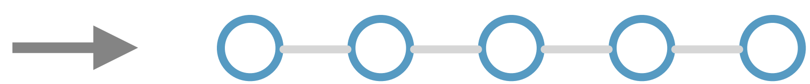
Euclidean data: grids, sequences…





2D grid

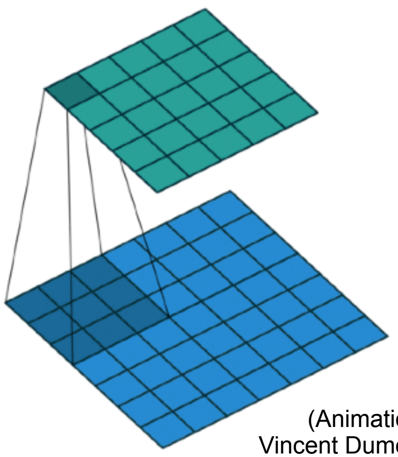




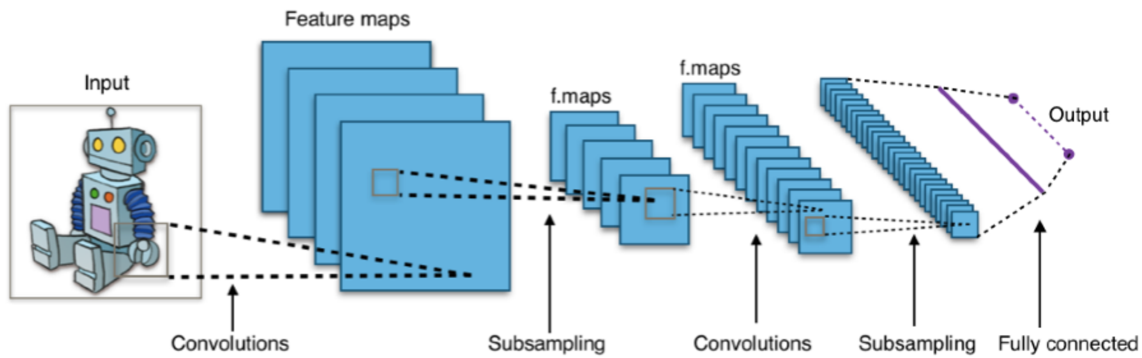
1D grid

Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)



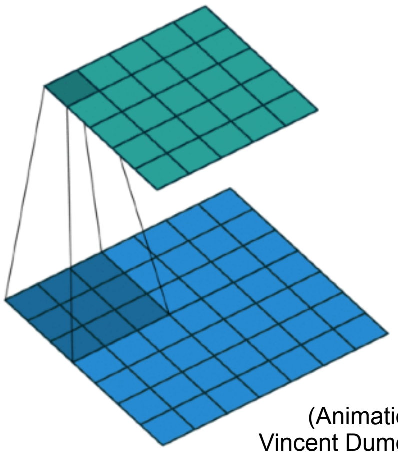
(Animation by
Vincent Dumoulin)



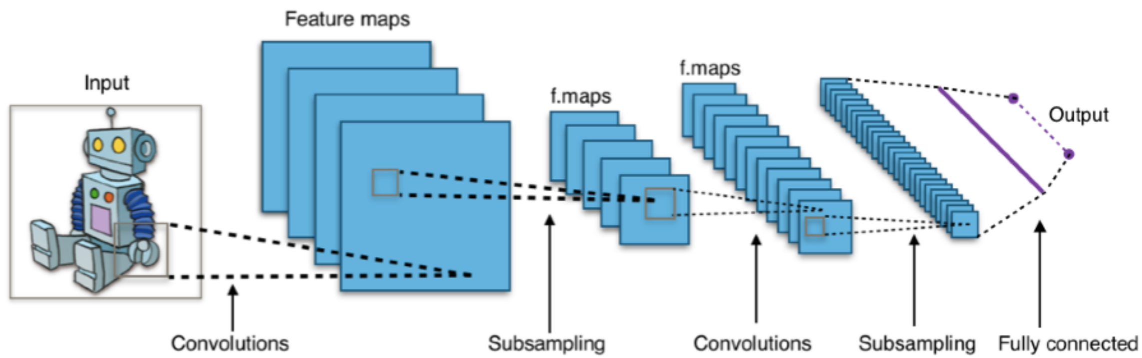
(Source: Wikipedia)

Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)



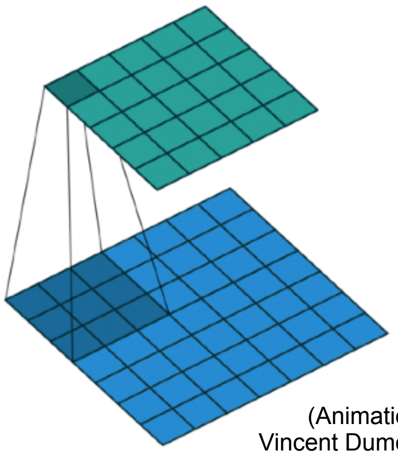
(Animation by Vincent Dumoulin)



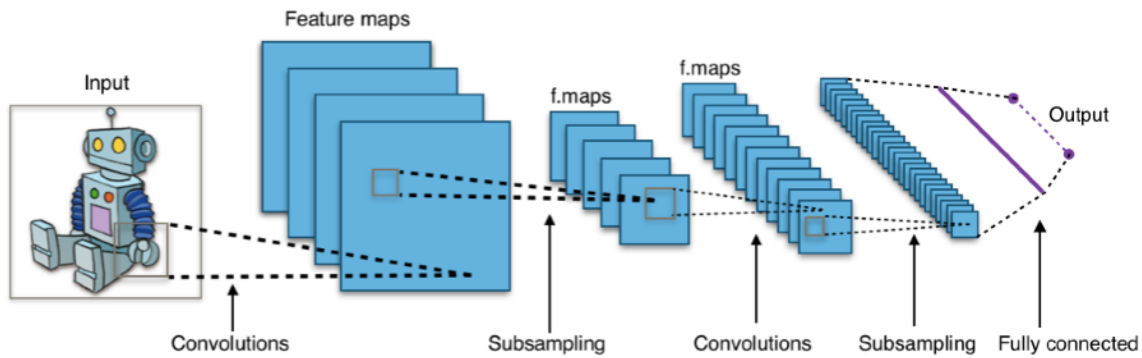
(Source: Wikipedia)

Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)

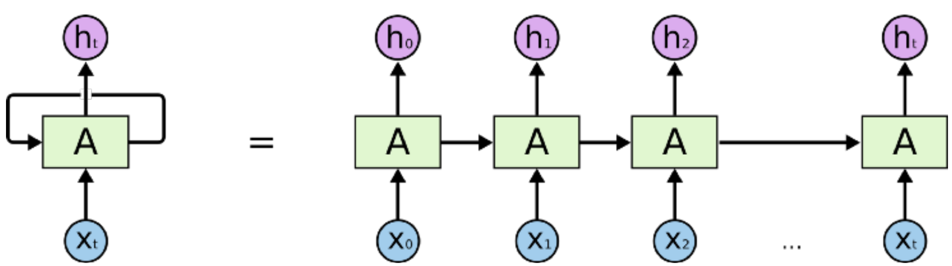


(Animation by Vincent Dumoulin)



(Source: Wikipedia)

Recurrent neural networks (RNNs)



(Source: Christopher Olah's blog)

Traditional vs. “deep” learning

Traditional approach

Hand-designed feature extractor → Classifier “on top” → Output

Traditional vs. “deep” learning

Traditional approach

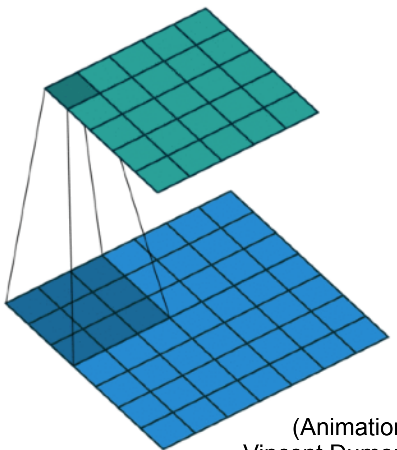
Hand-designed feature extractor → Classifier “on top” → Output

End-to-end learning

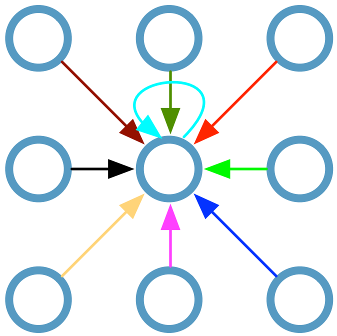
Deep neural network → Output

CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

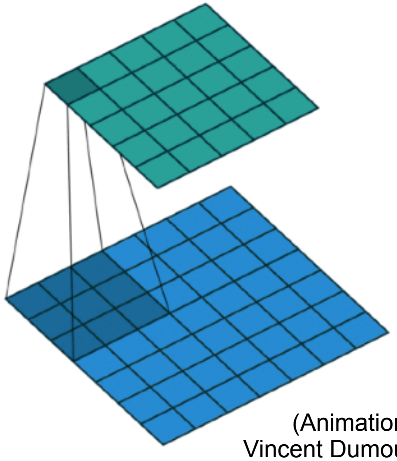


(Animation by
Vincent Dumoulin)

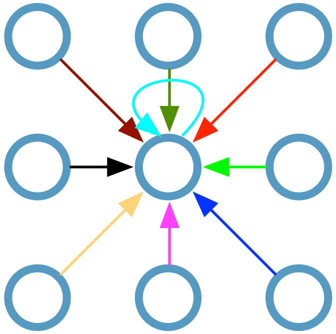


CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

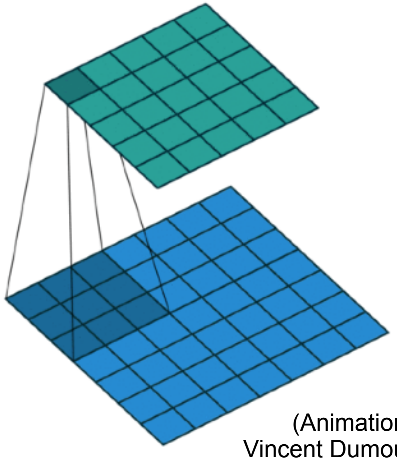


(Animation by
Vincent Dumoulin)

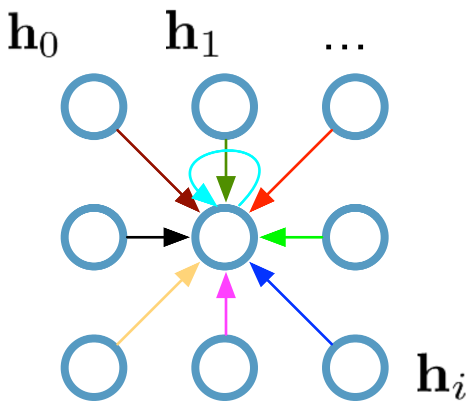


CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

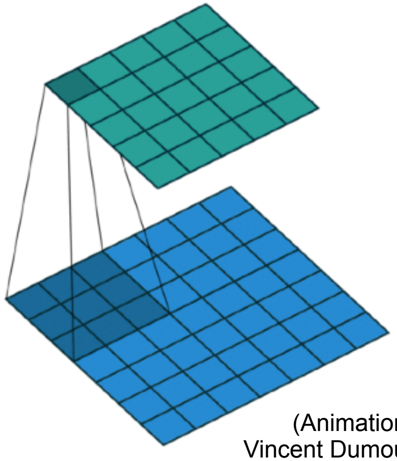


(Animation by
Vincent Dumoulin)

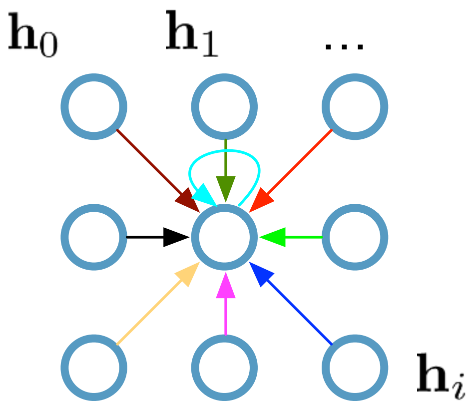


CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)

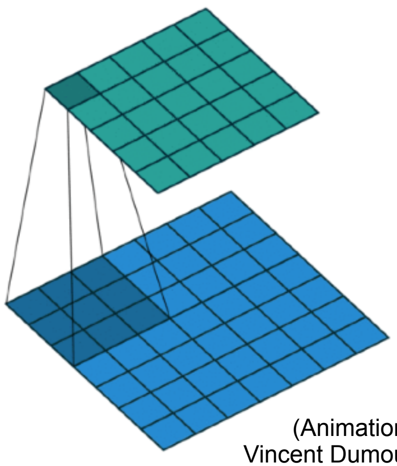


Update for a single pixel:

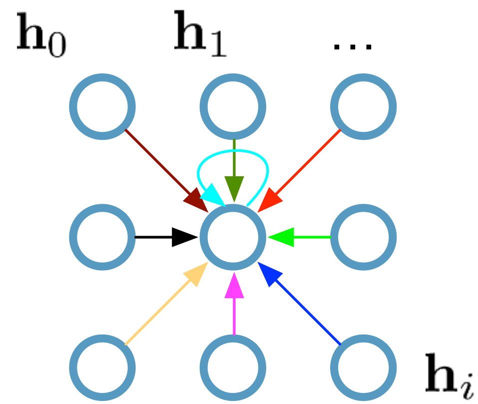
- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



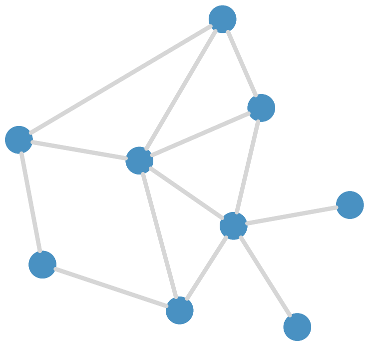
Update for a single pixel:

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

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

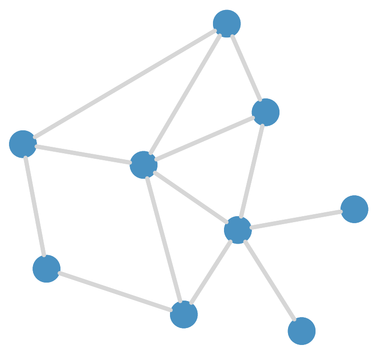
Graph-structured data

What if our data looks like this?

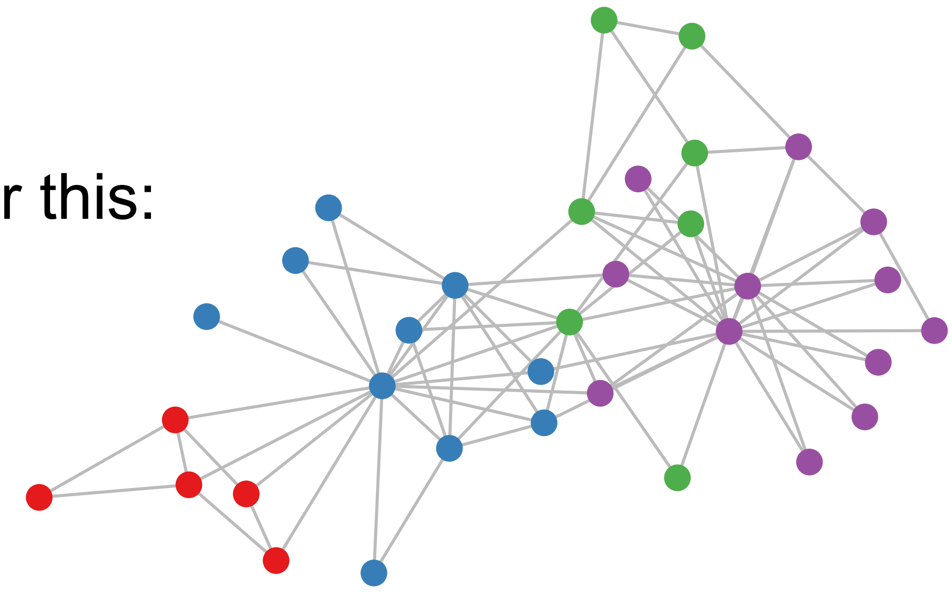


Graph-structured data

What if our data looks like this?

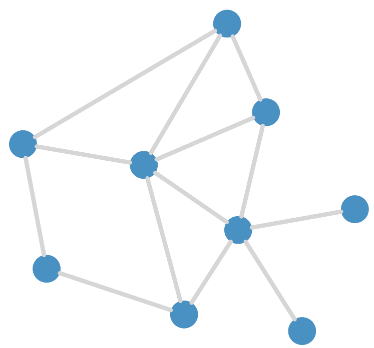


or this:

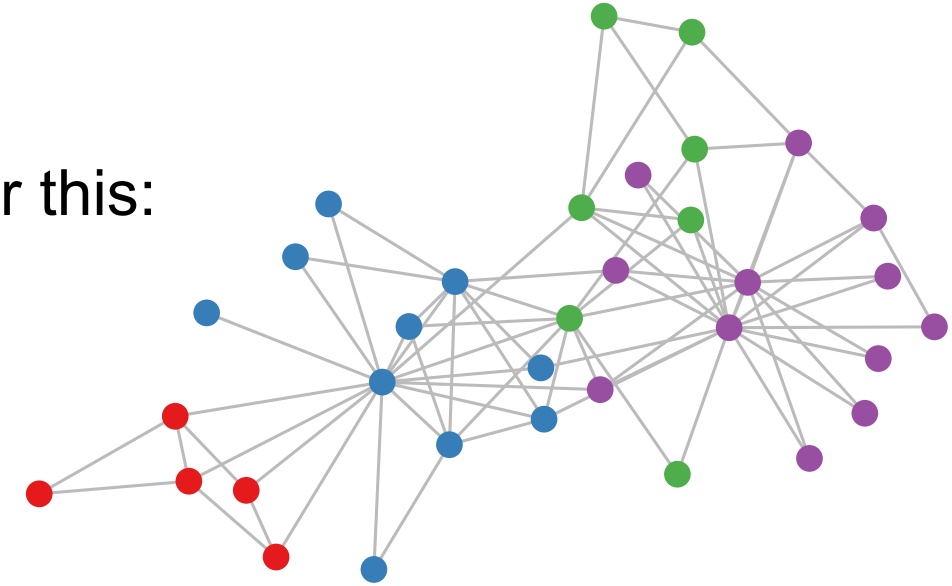


Graph-structured data

What if our data looks like this?



or this:

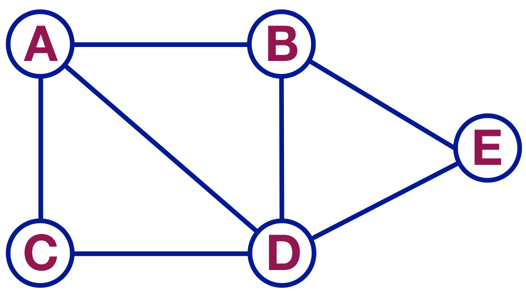


Real-world examples:

- Social networks
- World-wide-web
- Protein-interaction networks
- Telecommunication networks
- Knowledge graphs
- …

Graph-structured data

Graph: $G = (\mathcal{V}, \mathcal{E})$

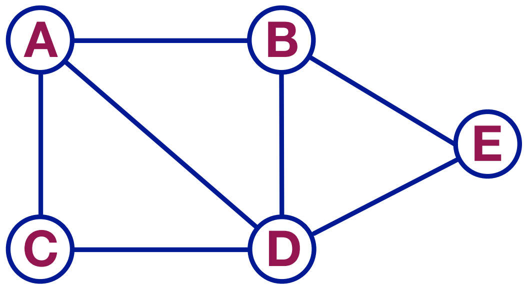


Adjacency matrix: $\mathbf{A}$

$$\begin{array}{cc} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Graph-structured data

Graph: $G = (\mathcal{V}, \mathcal{E})$



Adjacency matrix: $\mathbf{A}$

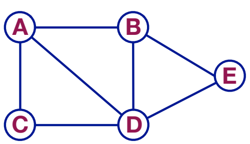
$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E
\end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Model wish list:

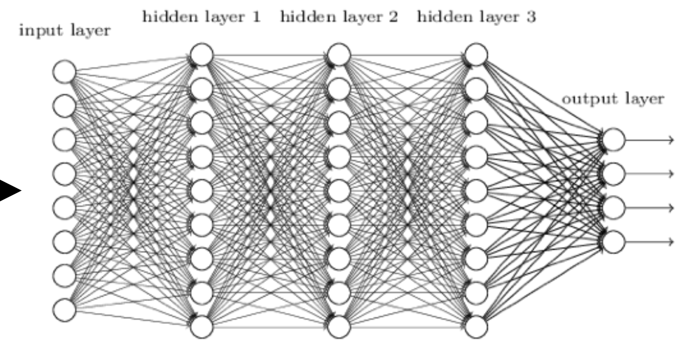
- Trainable in $\mathcal{O}(|\mathcal{E}|)$ time
- Applicable even if the input graph changes

A naïve approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{A}, \mathbf{X}]$
- Feed them into deep (fully connected) neural net
- Done?



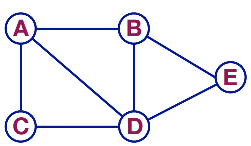
$$
\begin{array}{c}
\\
\text{A} \\ \text{B} \\ \text{C} \\ \text{D} \\ \text{E}
\end{array}
\begin{array}{c}
\begin{array}{cccccc} \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \quad\text{Feat} \end{array} \\
\left(\begin{array}{ccccc|cc}
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0
\end{array}\right)
\end{array}
\longrightarrow
$$



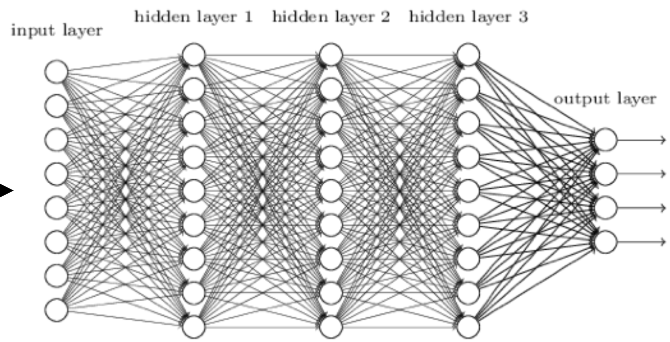
?

A naïve approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{A}, \mathbf{X}]$
- Feed them into deep (fully connected) neural net
- Done?



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & \text{Feat} & \end{array} \\ \left(\begin{array}{ccccccc} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{array} \right) \end{array} \longrightarrow$$



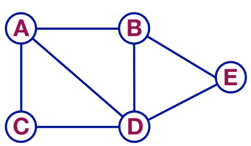
?

Problems:

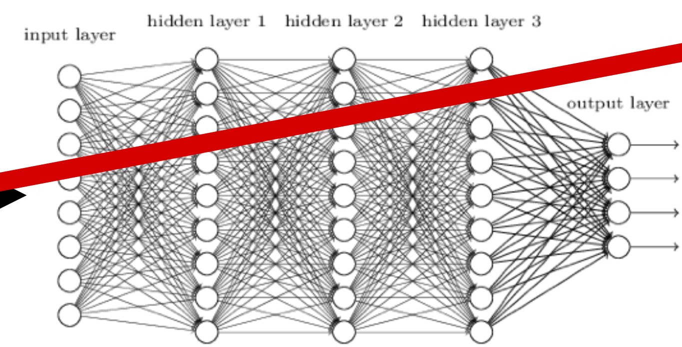
- Huge number of parameters $\mathcal{O}(N)$
- Re-train if graph changes

A naïve approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{A}, \mathbf{X}]$
- Feed them into deep (fully connected) neural net
- Done?



	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	[illegible]
D	1	1	[illegible]	0	1	1	1
E	0	1	0	1	0	1	0



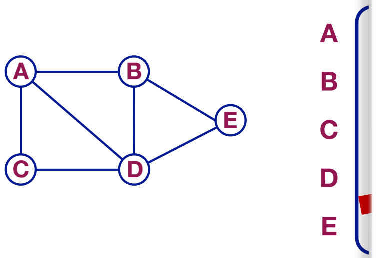
?

Problems:

- Huge number of parameters $\mathcal{O}(N)$
- Re-train if graph changes

A naïve approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{A}, \mathbf{X}]$
- Feed them into deep (fully connected) neural net
- Done?



We need weight sharing!

➡ CNNs on graphs or “Graph Convolutional Networks” (GCNs)

?

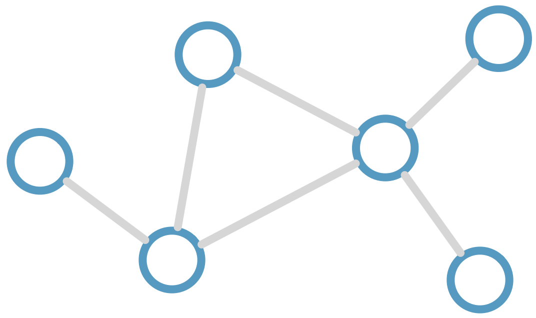
Problems:

- Huge number of parameters $\mathcal{O}(N)$
- Re-train if graph changes

GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

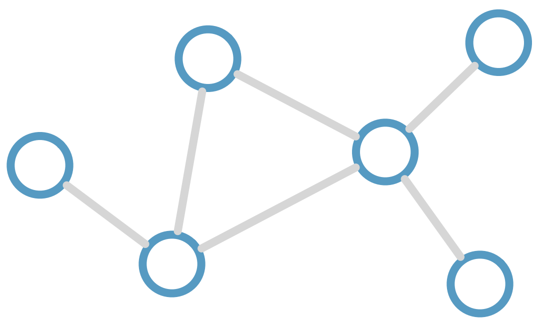
Consider this undirected graph:



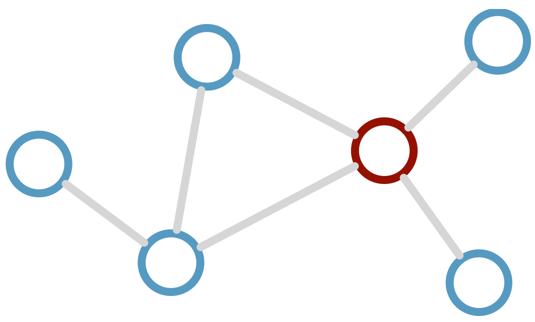
GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



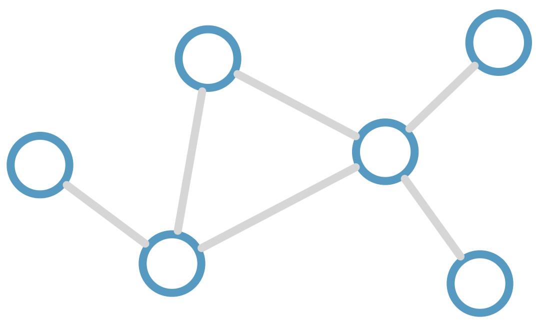
Calculate update for node in red:



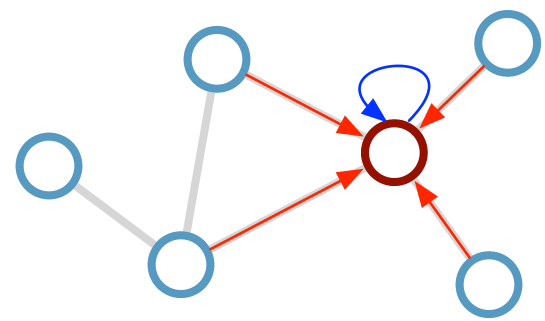
GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



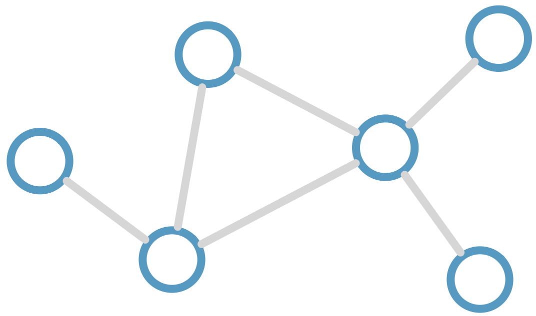
Calculate update for node in red:



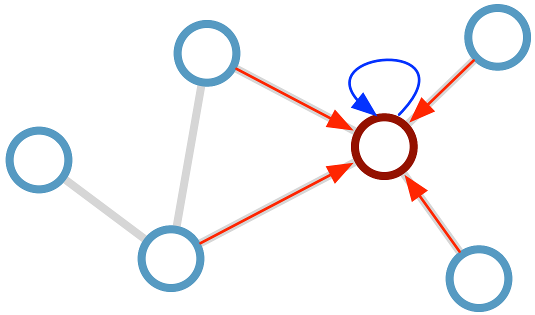
GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:



Calculate update for node in red:



Update rule: $$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

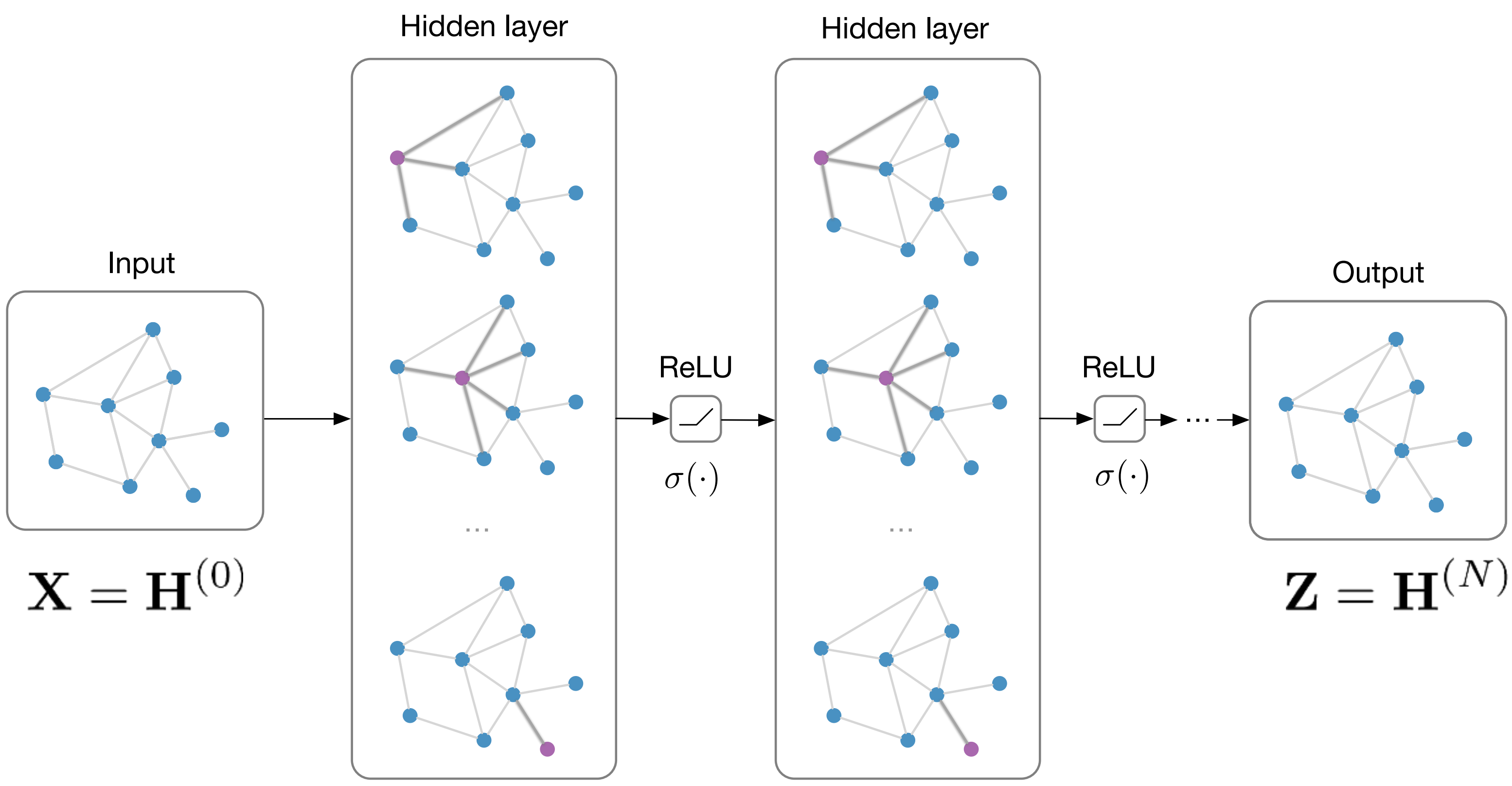
$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (per edge)

Note: We could also choose simpler or more general functions over the neighborhood

GCN model architecture

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

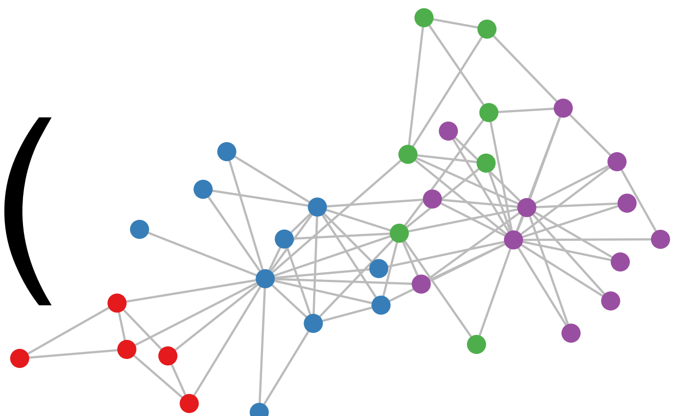
[Kipf & Welling, ICLR 2017]

What does it do? An example.

Forward pass through **untrained** 3-layer GCN model

Parameters initialized randomly

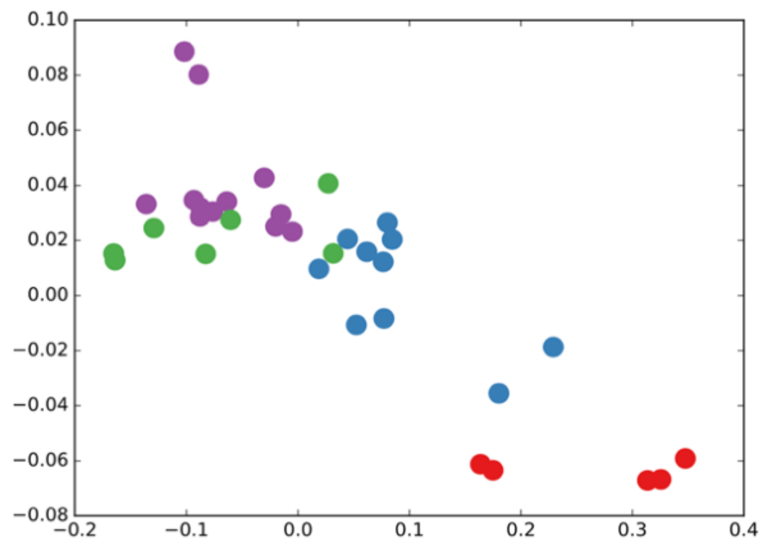
$f($



$) =$

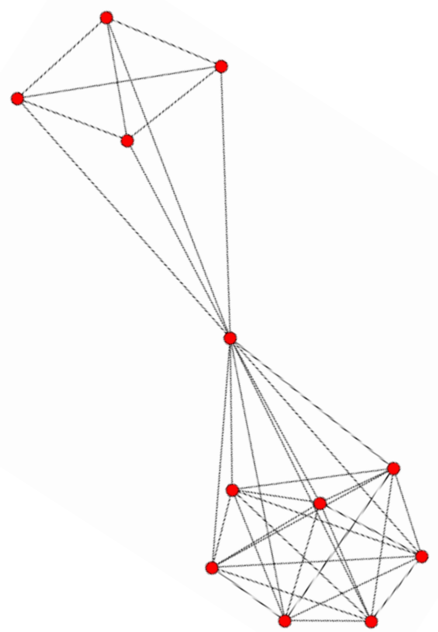
[Karate Club Network]

2-dim output per node



Relation to Weisfeiler-Lehman algorithm

A "classical" approach for node feature assignment

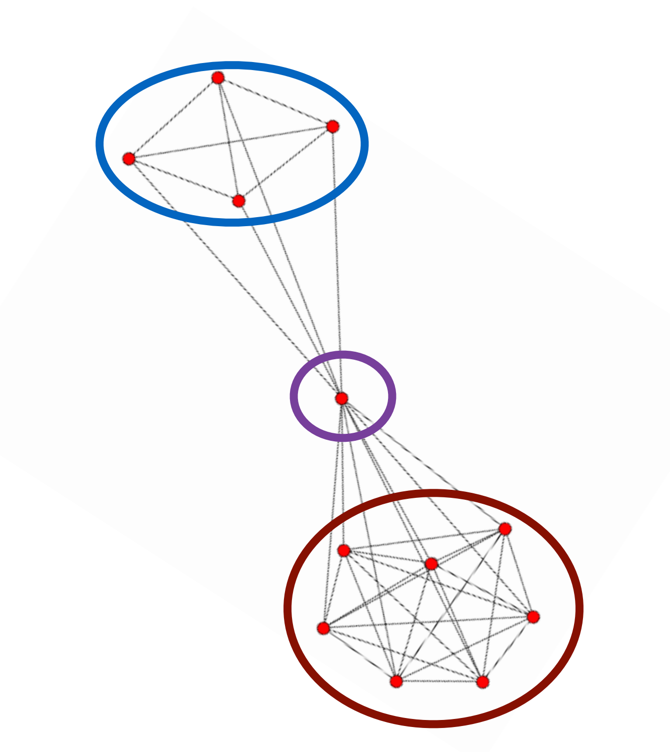


Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
t $\leftarrow$ 0;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
until *stable node coloring is reached*;

Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

t $\leftarrow$ 0;

repeat

  for $v_i \in \mathcal{V}$ **do**

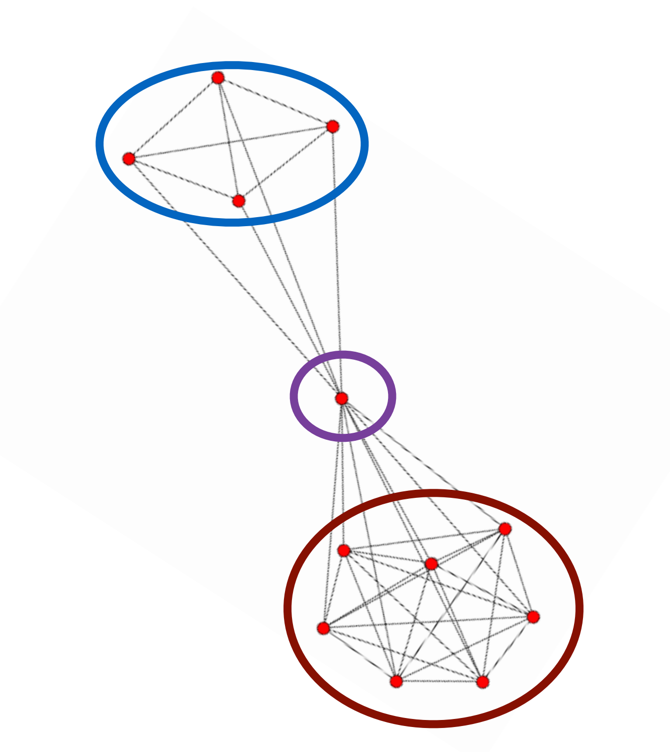
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;

  $t \leftarrow t + 1$;

until *stable node coloring is reached*;

Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

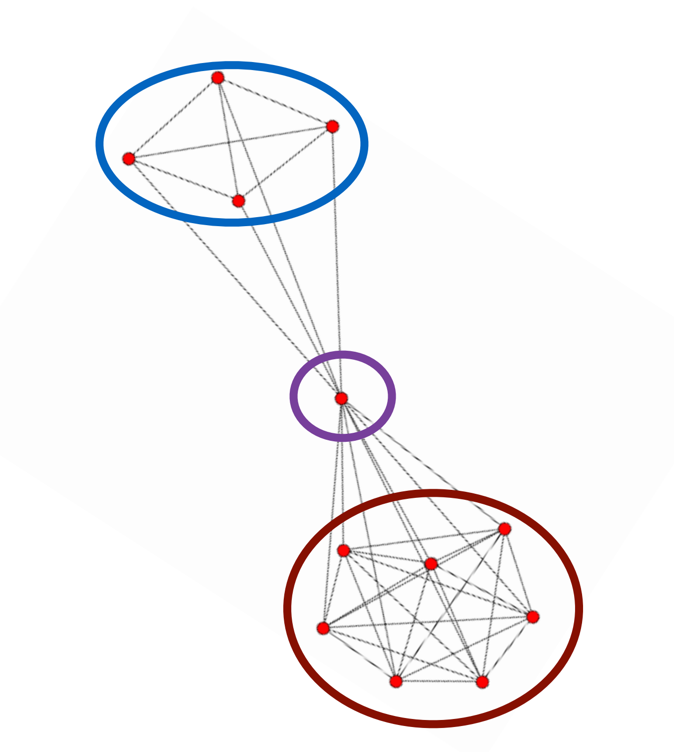
Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
t $\leftarrow$ 0;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)$;
  $t \leftarrow t + 1$;
until *stable node coloring is reached*;

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)

Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$

Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$

$t \leftarrow 0$;

repeat

  for $v_i \in \mathcal{V}$ **do**

    $\cancel{h_i^{(t+1)} \leftarrow \mathrm{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)}$;

  $t \leftarrow t + 1$;

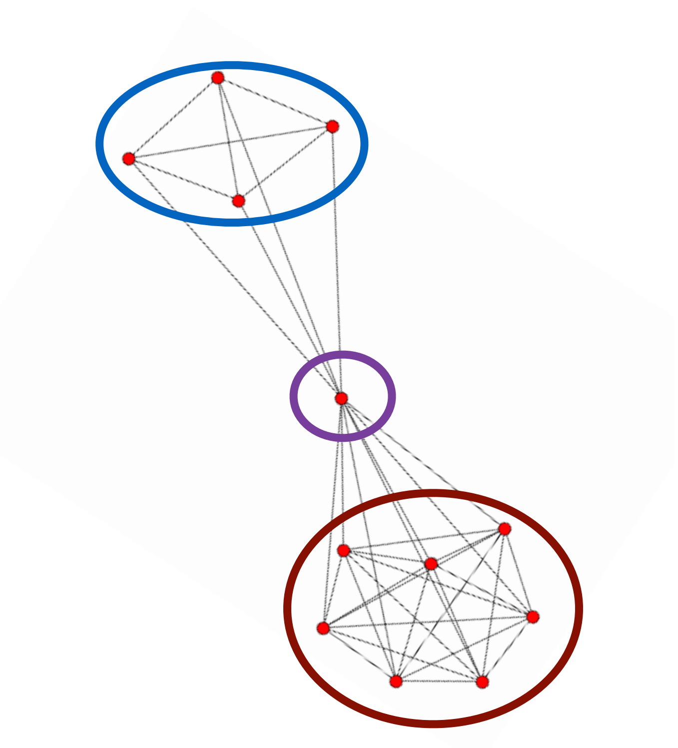
until *stable node coloring is reached*;

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)

Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment



Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring $(h_1^{(0)}, h_2^{(0)}, ..., h_N^{(0)})$
Output: Final node coloring $(h_1^{(T)}, h_2^{(T)}, ..., h_N^{(T)})$
$t \leftarrow 0$;
repeat
  for $v_i \in \mathcal{V}$ **do**
    $\cancel{h_i^{(t+1)} \leftarrow \text{hash}\left(\sum_{j \in \mathcal{N}_i} h_j^{(t)}\right)}$;
un

GCN: $$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)} \mathbf{W}_1^{(l)}\right)$$

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)

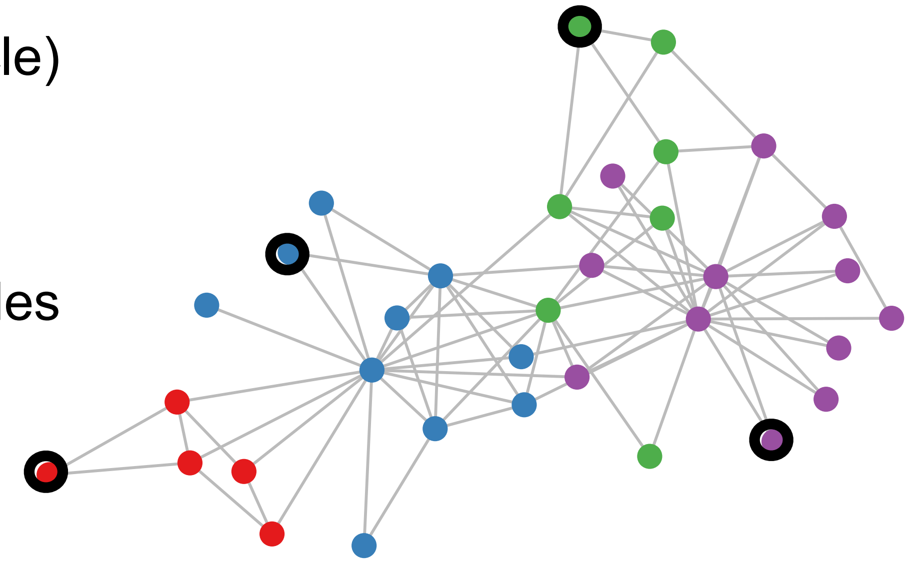
Semi-supervised classification on graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:

Predict node label of unlabeled nodes

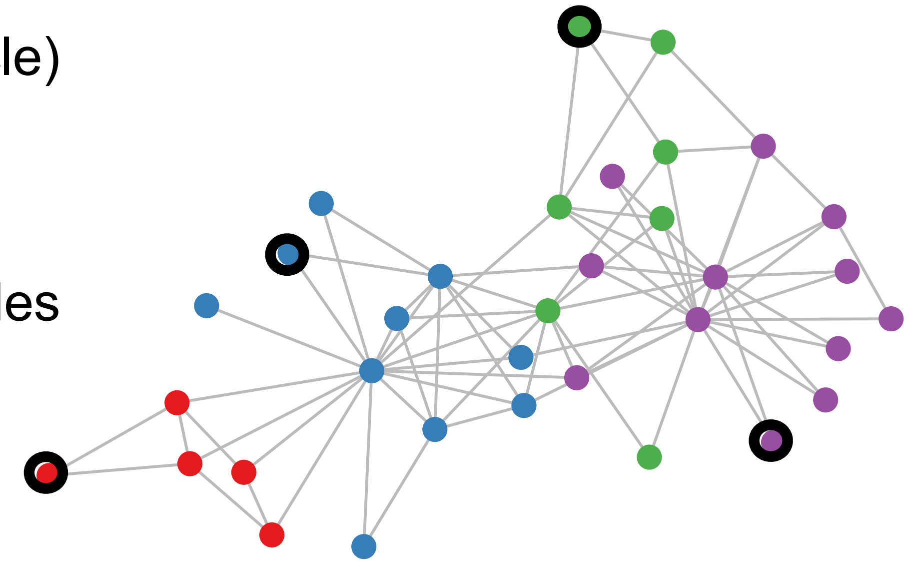


Semi-supervised classification on graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



Standard approach:

graph-based regularization [Zhu et al., 2003]

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2$$

assumes: connected nodes likely to share same label

Semi-supervised classification on graphs

Embedding-based approaches

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

Examples: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

Semi-supervised classification on graphs

Embedding-based approaches

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

Examples: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

Problem: Embeddings are not optimized for classification!

Semi-supervised classification on graphs

Embedding-based approaches

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

Examples: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

Problem: Embeddings are not optimized for classification!

Idea: Train graph-based classifier end-to-end using GCN

Evaluate loss on labeled nodes only:

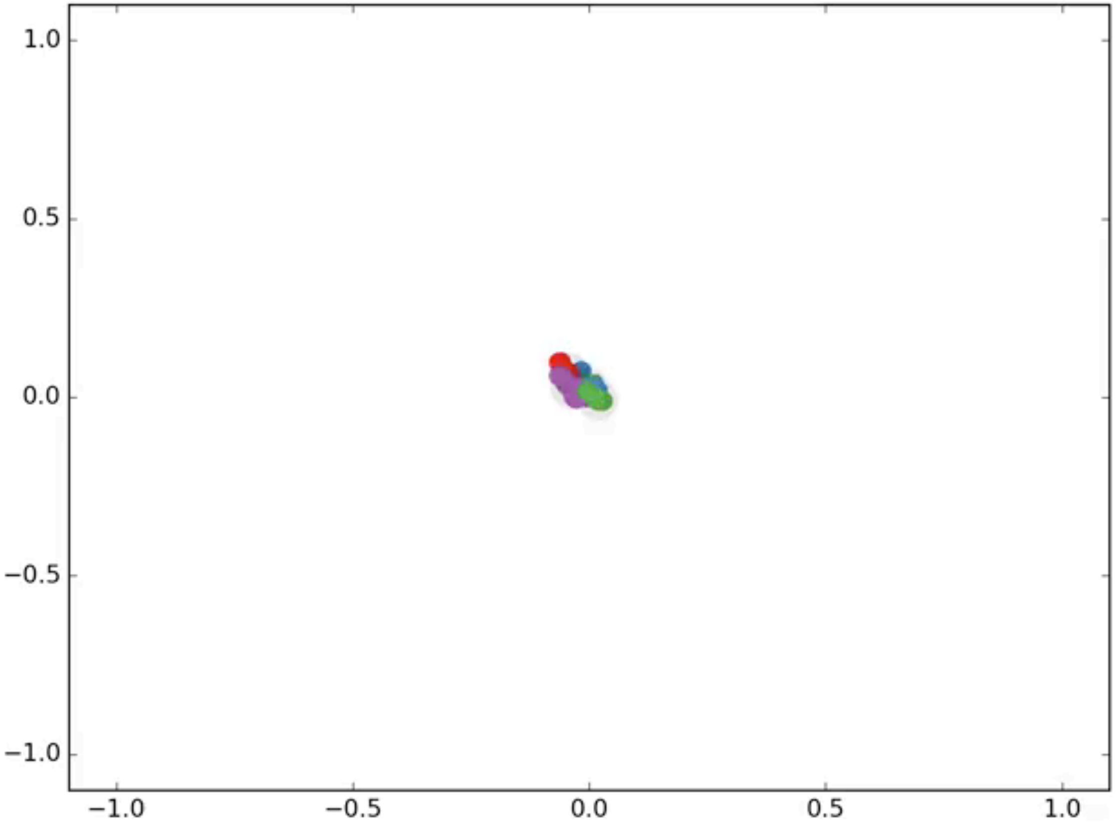
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

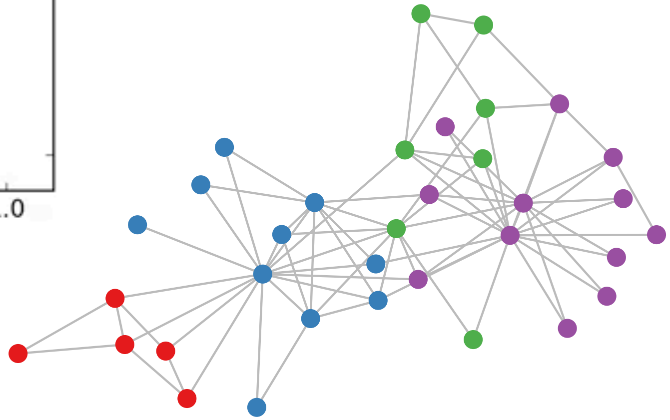
$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

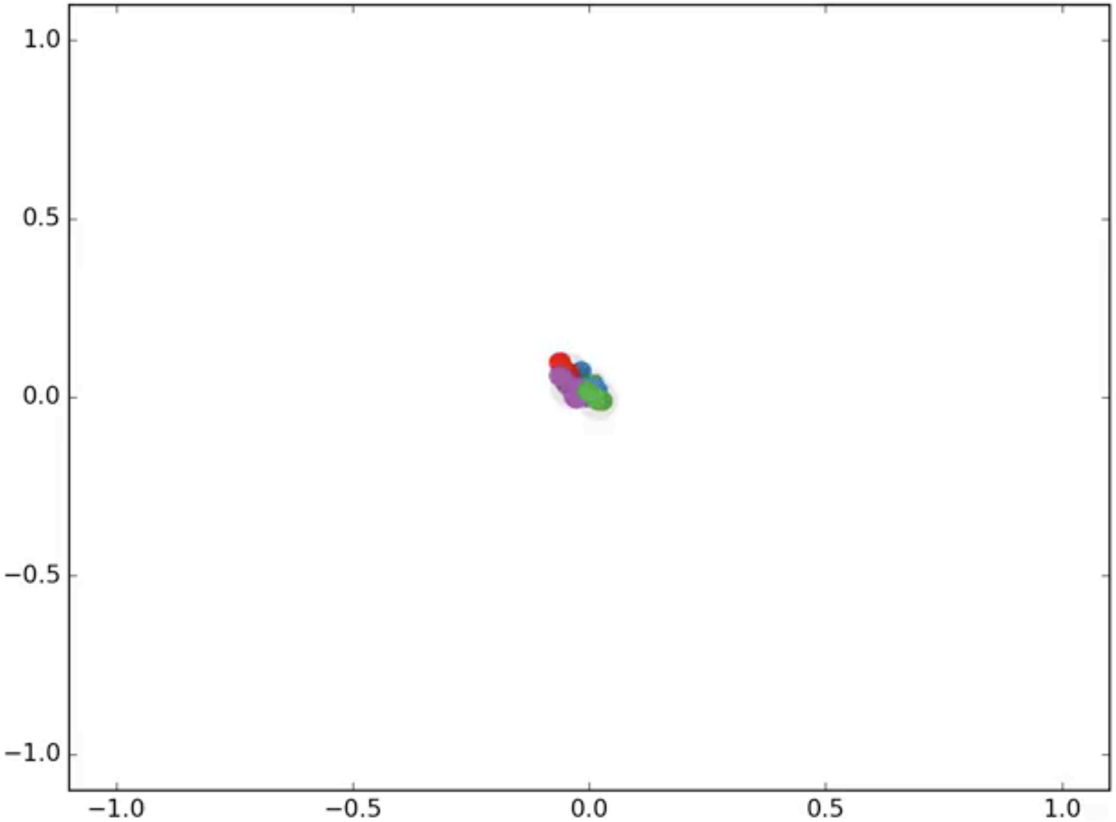
Toy example (semi-supervised learning)

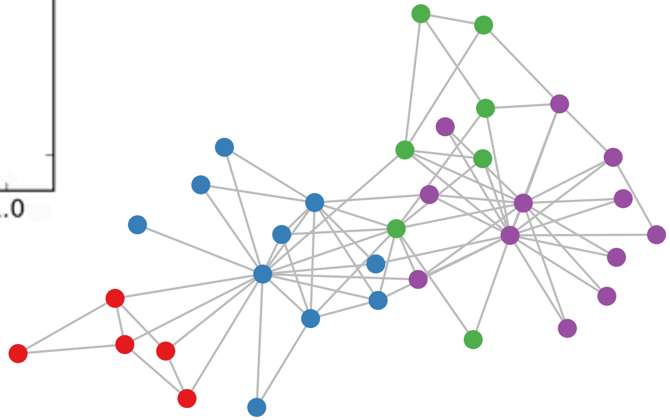




Video also available here:
http://tkipf.github.io/graph-convolutional-networks

Toy example (semi-supervised learning)



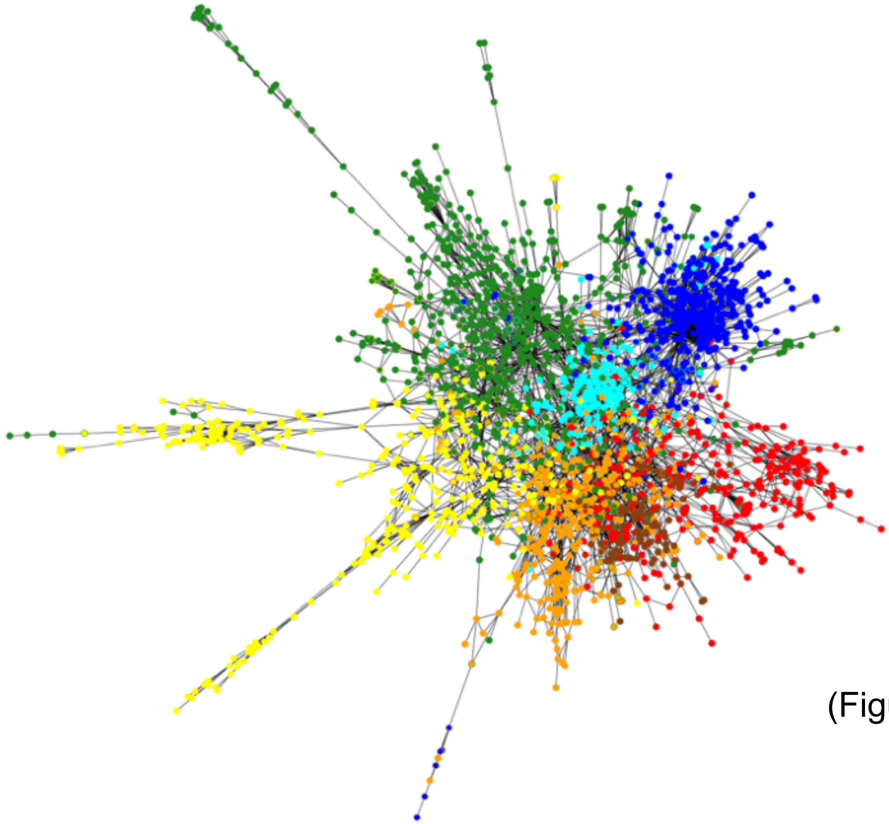


Video also available here:
http://tkipf.github.io/graph-convolutional-networks

Application: Classification on citation networks

Input: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

Target: Paper category (e.g. stat.ML, cs.LG, …)



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Experiments and results

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\, W^{(1)}\Big)$

Dataset statistics

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

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)

Experiments and results

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$

Dataset statistics

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

Classification results (accuracy)

no input features

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [24]	59.6	59.0	71.1	26.7
LP [27]	45.3	68.0	63.0	26.5
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	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

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)

Experiments and results

Model: 2-layer GCN $Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$

Datas	...	l rate
Citese		0.036
Cora		0.052
Pubme		0.003
NELL		0.001

Learned representations
(t-SNE embedding of hidden layer activations)

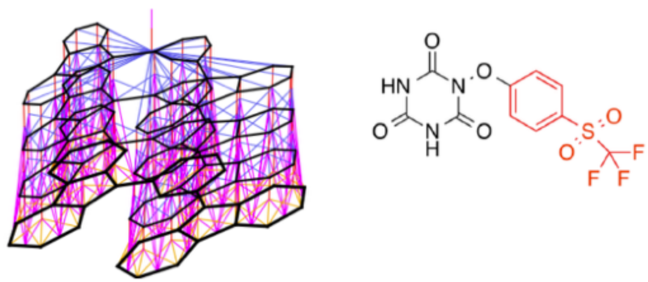
no input features

				LL
				8
				7
				[illegible]
DeepWalk [18]	43.2	67.2	65.3	58.1
Planetoid* [25]	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

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)

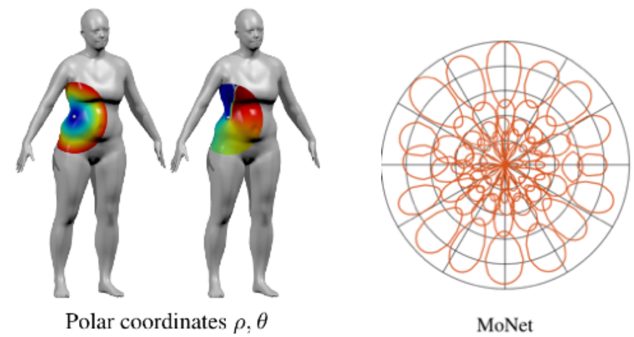
Other recent applications

Molecules



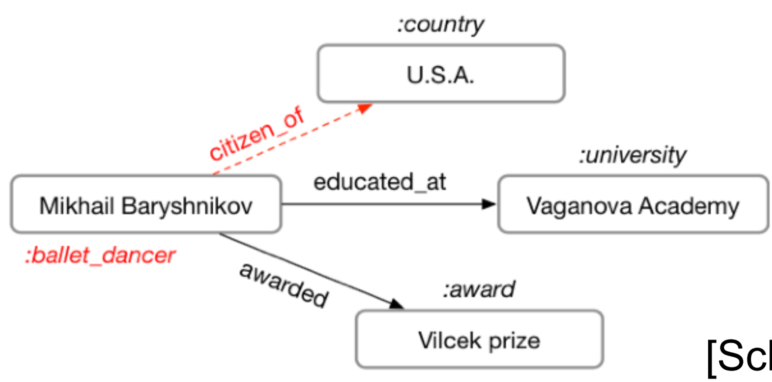
[Duvenaud et al., NIPS 2015]

Shapes



[Monti et al., 2016]

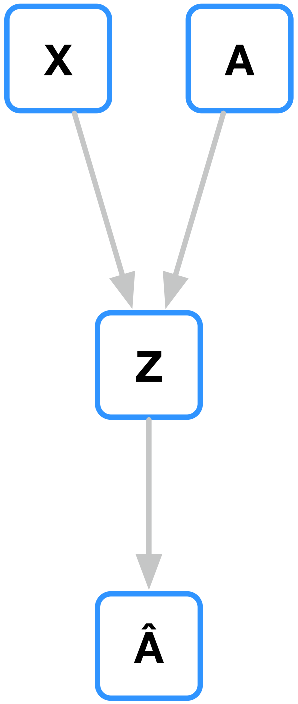
Knowledge Graphs



[Schlichtkrull et al., 2017]

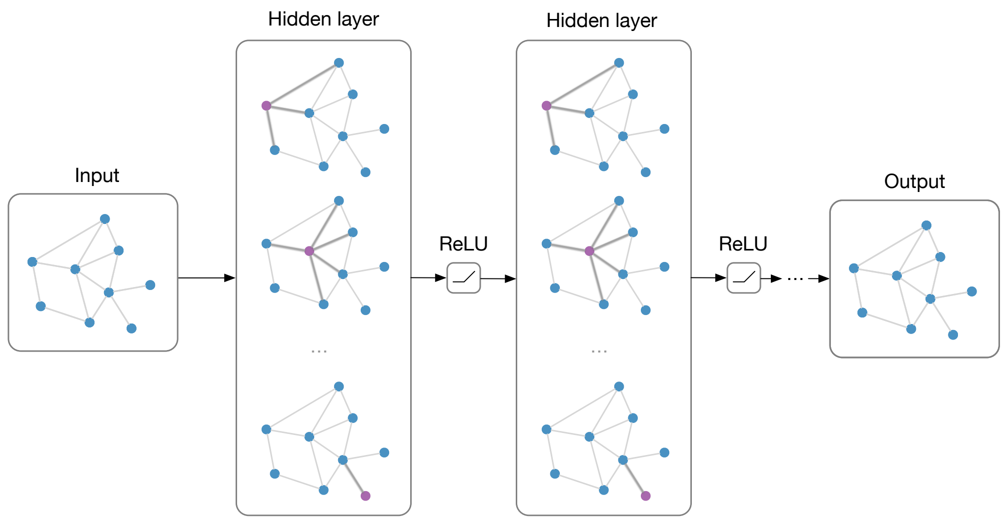
Link prediction with Graph Auto-Encoders

Kipf & Welling, NIPS Bayesian Deep Learning Workshop, 2016



GAE

Encoder $\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A})$



Decoder $\hat{\mathbf{A}} = \sigma\left(\mathbf{Z}\mathbf{Z}^\top\right)$

Further reading

Blog post Graph Convolutional Networks:
http://tkipf.github.io/graph-convolutional-networks

Code on Github:
http://github.com/tkipf/gcn

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017:
https://arxiv.org/abs/1609.02907

Kipf & Welling, Variational Graph Auto-Encoders, NIPS BDL Workshop, 2016:
https://arxiv.org/abs/1611.07308

You can get in touch with me via:

- E-Mail: T.N.Kipf@uva.nl
- Twitter: @thomaskipf
- Web: http://tkipf.github.io

Project funded by SAP