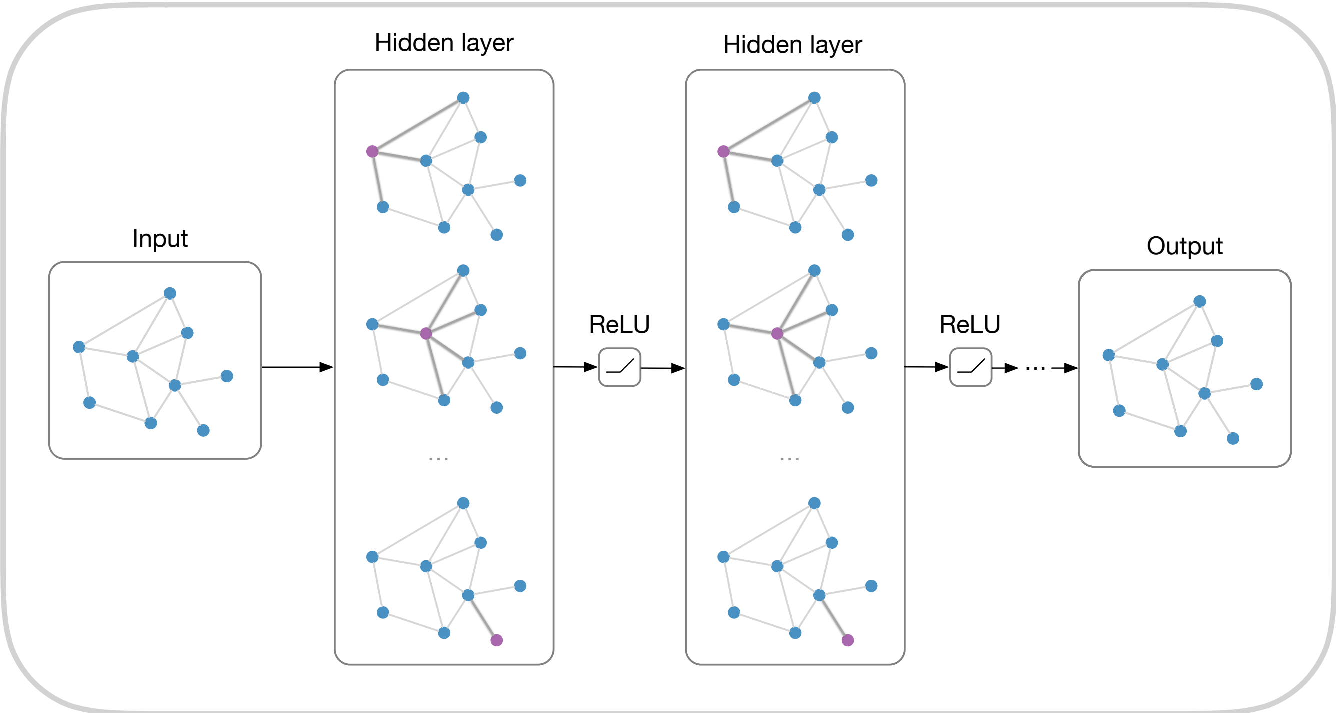
Deep Learning on Graph-Structured Data

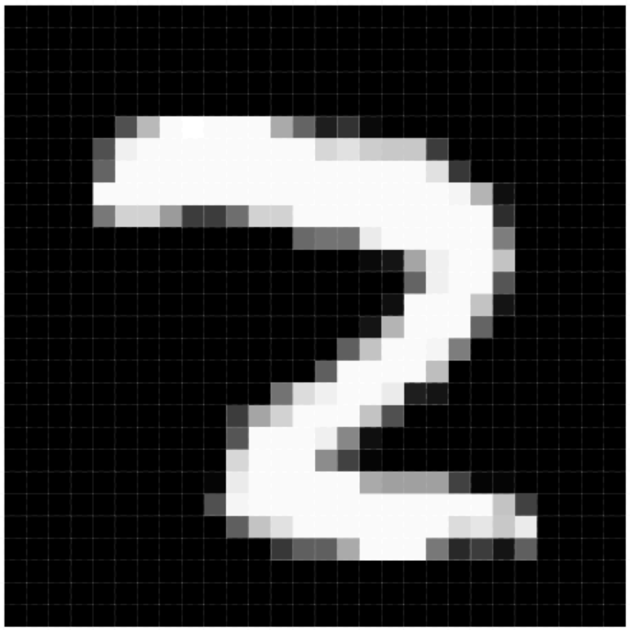


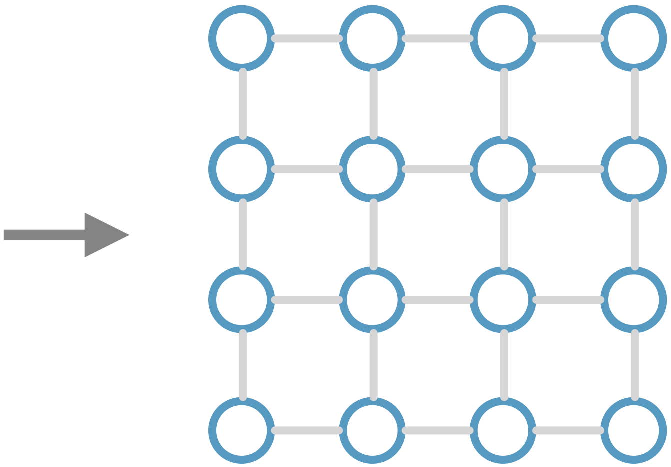
Thomas Kipf, 1 December 2016

Universiteit van Amsterdam

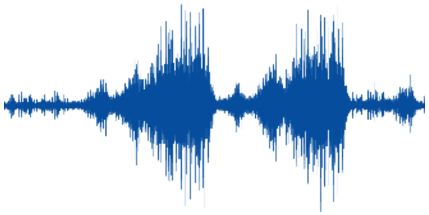
Recap: Deep learning on Euclidean data

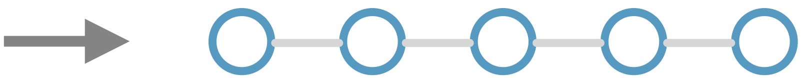
Euclidean data: grids, sequences…





2D grid



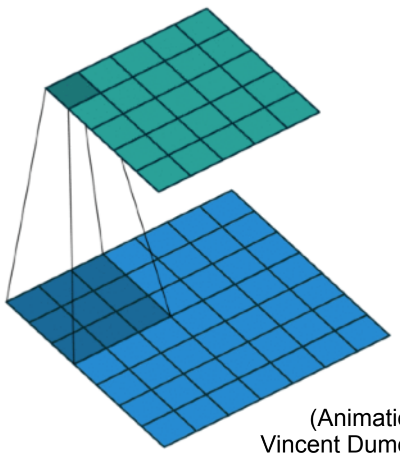


1D grid

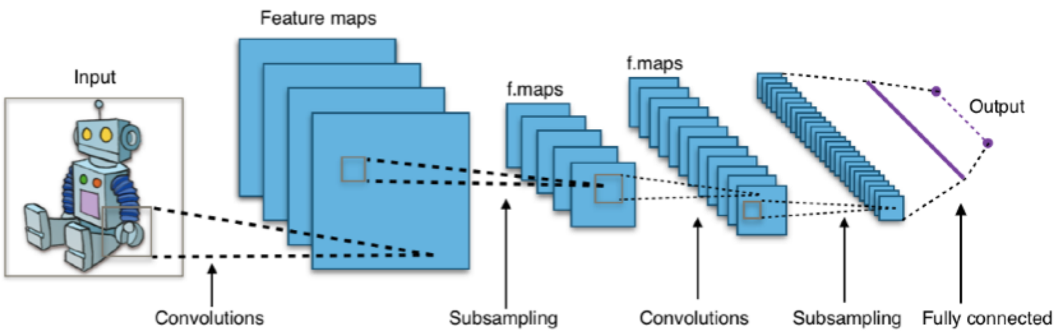
Recap: Deep learning on Euclidean data

We know how to deal with this:

Convolutional neural networks (CNNs)

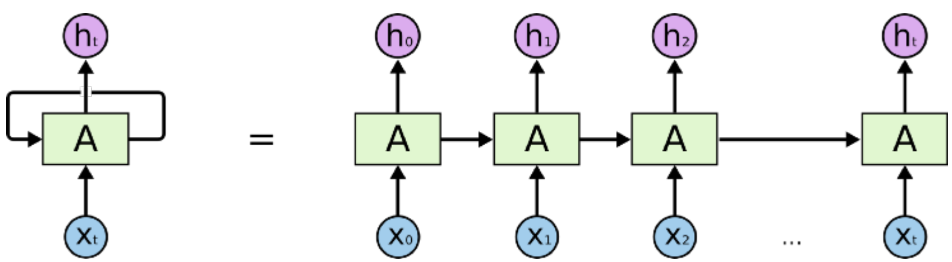


(Animation by Vincent Dumoulin)



(Source: Wikipedia)

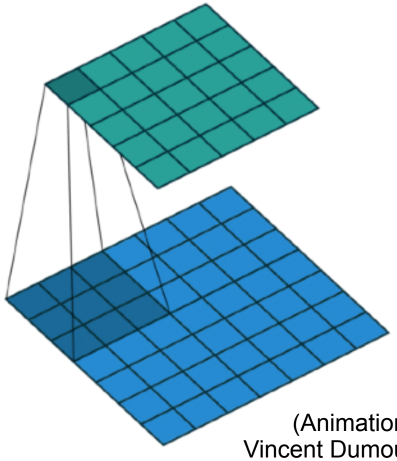
or recurrent neural networks (RNNs)



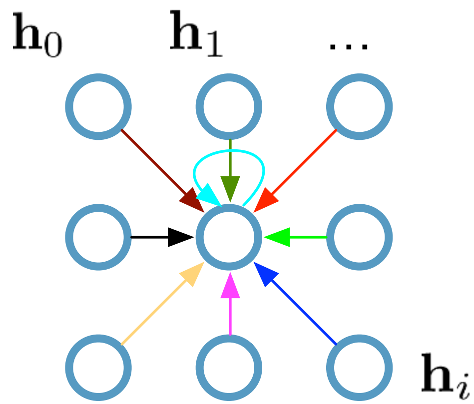
(Source: Christopher Olah's blog)

Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



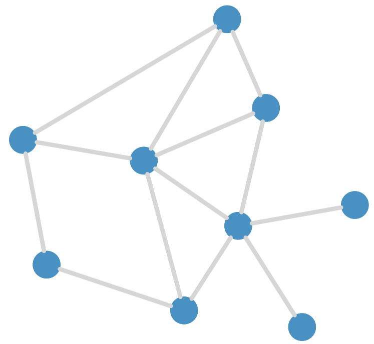
Update for a single pixel:

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

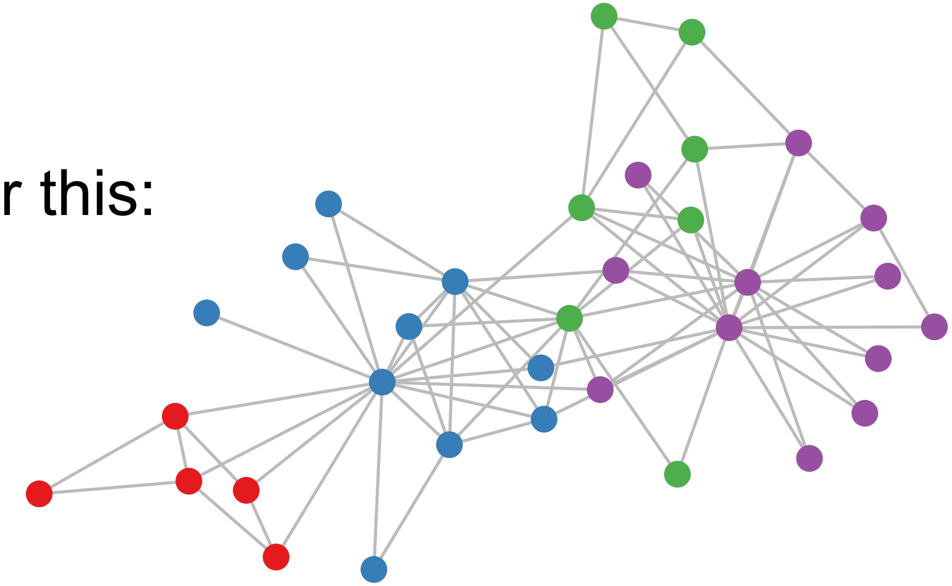
Full update: $\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$

Graph-structured data

What if our data looks like this?



or this:

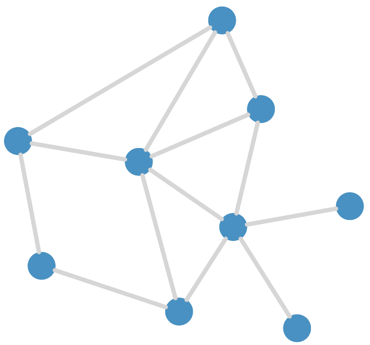


Real-world examples:

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

Graphs: Definitions

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



$\mathcal{V}$: Set of nodes $\{v_i\}$, $|\mathcal{V}| = N$

$\mathcal{E}$: Set of edges $\{(v_i, v_j)\}$

We can define:

$\mathbf{A}$ (adjacency matrix): $A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$

(can also be weighted)

Model wish list:

- Set of trainable parameters $\{\mathbf{W}^{(l)}\}$
- Trainable in $\mathcal{O}(|\mathcal{E}|)$ time
- Applicable even if the input graph changes

Spectral graph convolutions

Main idea:

Use **convolution theorem** to generalize convolution to graphs.

Loosely speaking:

A convolution corresponds to a multiplication in the Fourier domain.

Graph Fourier transform: [Hammond, Vandergheynst, Gribonval, 2009]

$\mathcal{F}_G[\mathbf{x}] = \mathbf{U}^T\mathbf{x}$ $\mathbf{U}$: eigenvectors of *graph Laplacian* $\mathbf{L}$

with $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ (normalized graph Laplacian)

and $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ (its eigen-decomposition)

D: degree matrix
$D_{ii} = \sum_j A_{ij}$

Spectral graph convolutional networks

Graph convolution: $\mathbf{g}, \mathbf{x} \in \mathbb{R}^N$

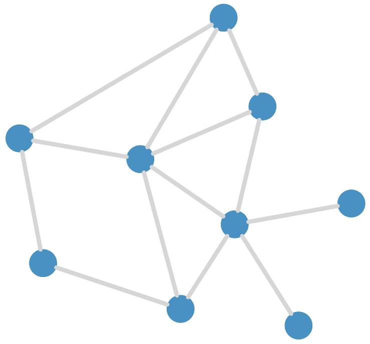
$$\mathbf{x} *_G \mathbf{g} = \mathcal{F}_G^{-1}\left[\mathcal{F}_G[\mathbf{g}] \odot \mathcal{F}_G[\mathbf{x}]\right] = \mathbf{U}\left(\mathbf{U}^T\mathbf{g} \odot \mathbf{U}^T\mathbf{x}\right)$$

or: $\mathbf{x} *_G \mathbf{g} = \mathbf{U}\,\mathrm{diag}(\hat{\mathbf{g}})\mathbf{U}^T\mathbf{x}$ with $\hat{\mathbf{g}} = \mathbf{U}^T\mathbf{g}$

Spectral CNN on graphs:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{U}\,\mathrm{diag}(\mathbf{w}^{(l)})\mathbf{U}^T\mathbf{h}_i^{(l)}\right)$$

[Bruna et al., ICLR 2014]

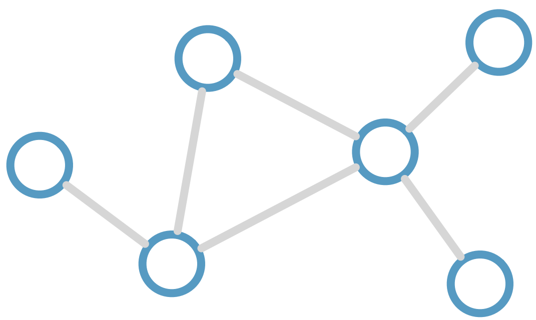


Limitations:

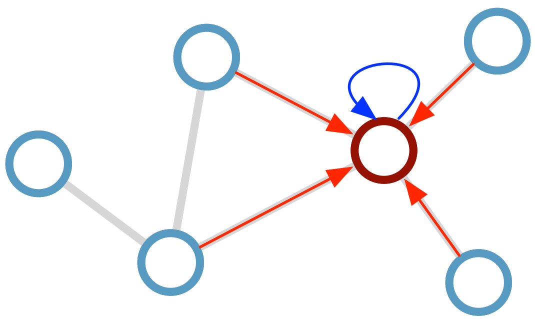
- Calculating $\mathbf{U}$ is expensive $\mathcal{O}(N^3)$
- Evaluating $\mathbf{U}^T\mathbf{x}$ is $\mathcal{O}(N^2)$
- Graph structure has to be fixed

Spatial graph convolutional networks (GCNs)

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

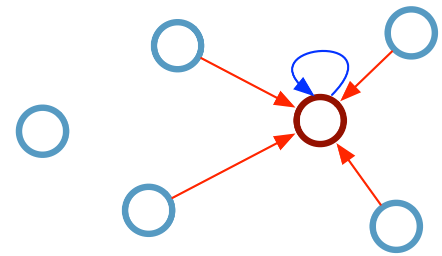
c_{ij} : norm. constant (per edge)

How is this related to spectral CNNs on graphs?

➡ Localized 1st-order approximation of spectral filters [Kipf & Welling, 2016]

Fully vectorized GCNs

$$\mathbf{H}^{(l+1)} = \sigma\left(\color{blue}{\mathbf{H}^{(l)}\mathbf{W}_0^{(l)}} + \color{red}{\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}}\right)$$

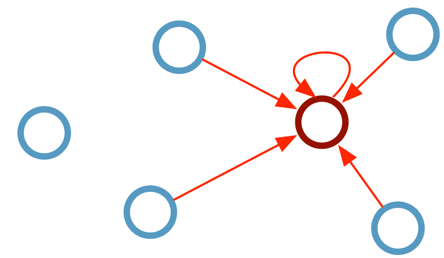


with $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ or $\tilde{\mathbf{A}} = \mathbf{D}^{-1}\mathbf{A}$

$$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)^T}, \ldots, \mathbf{h}_N^{(l)^T}]^T$$

Or treat self-connection in the same way:

$$\mathbf{H}^{(l+1)} = \sigma\left(\color{red}{\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}}\right)$$



with $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_N)\tilde{\mathbf{D}}^{-\frac{1}{2}}$ or $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}(\mathbf{A} + \mathbf{I}_N)$

$$\tilde{D}_{ii} = \sum_j (A_{ij} + \delta_{ij})$$

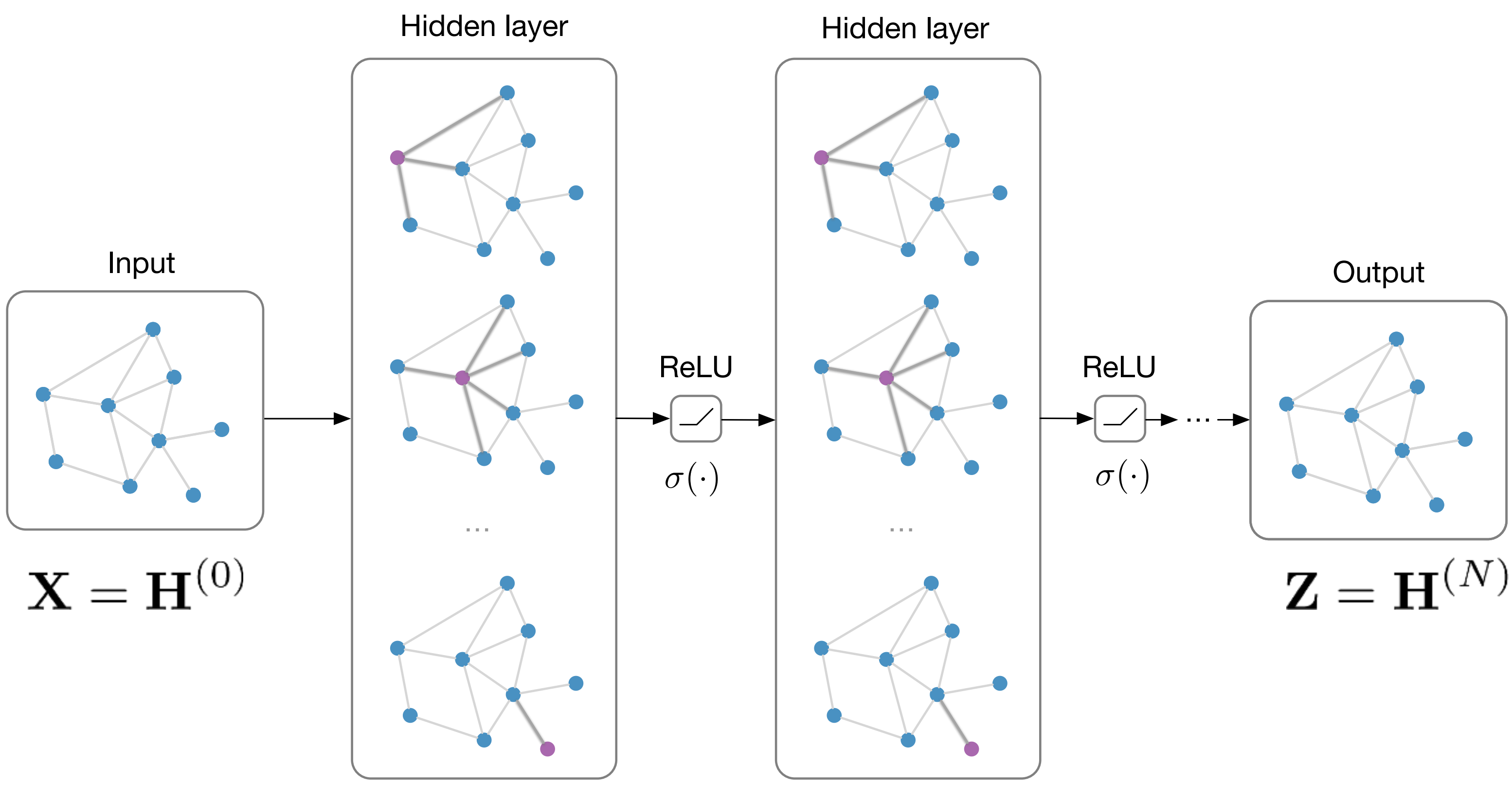
$\mathbf{A}$ is typically **sparse**

➡ We can use sparse matrix multiplications!

➡ Efficient $\mathcal{O}(|\mathcal{E}|)$ implementation in Theano or TensorFlow

GCN model architecture

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



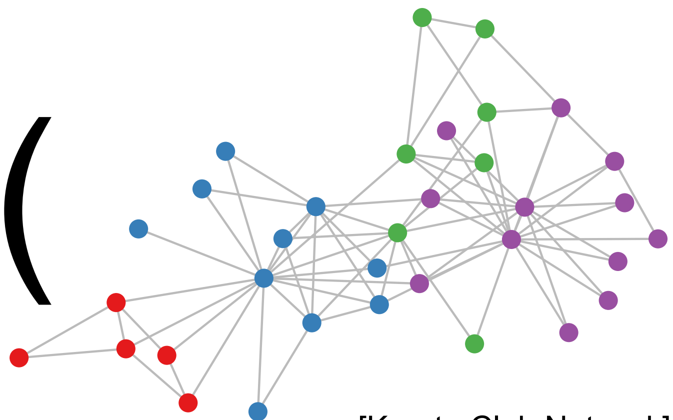
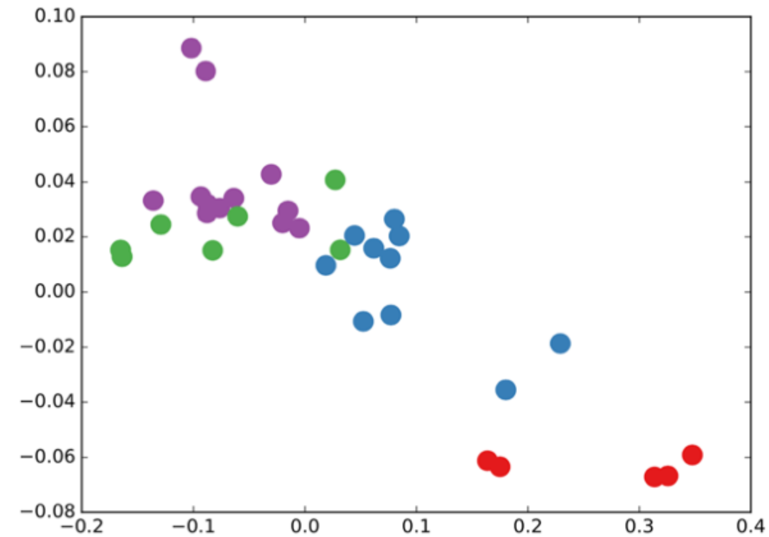
$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

What does it do? An example.

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

Parameters initialized randomly

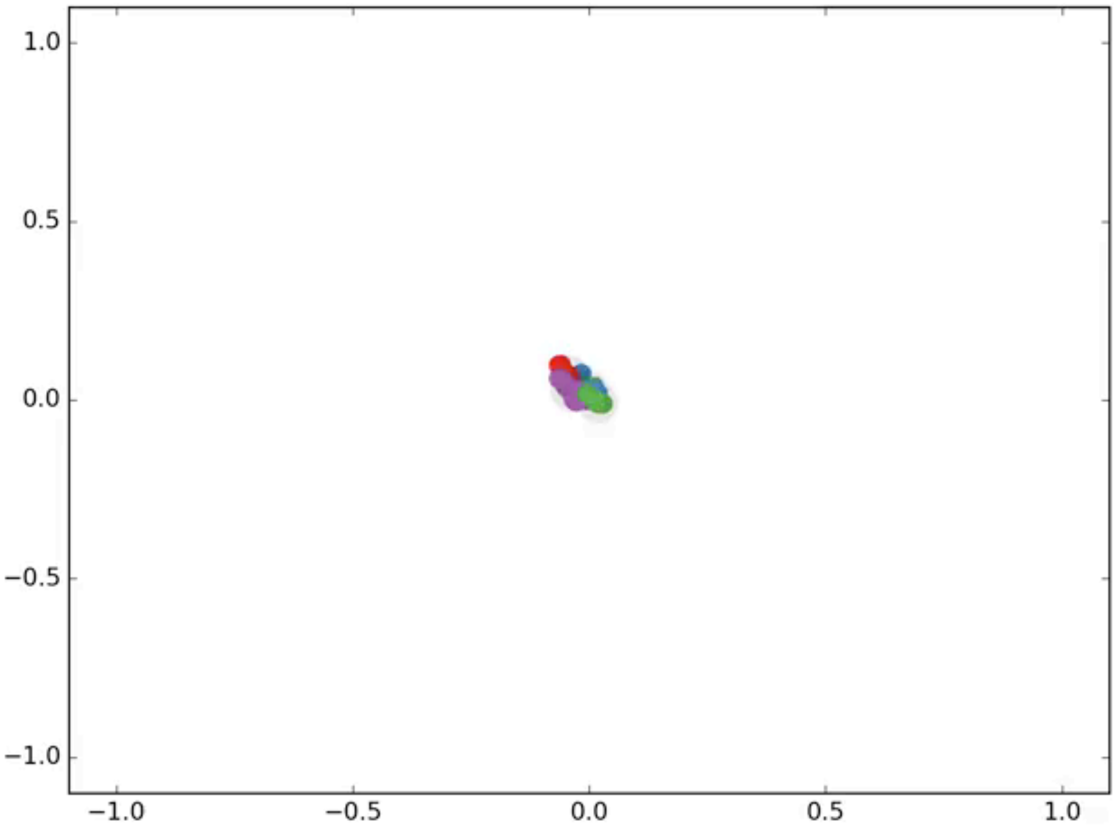
2-dim output per node

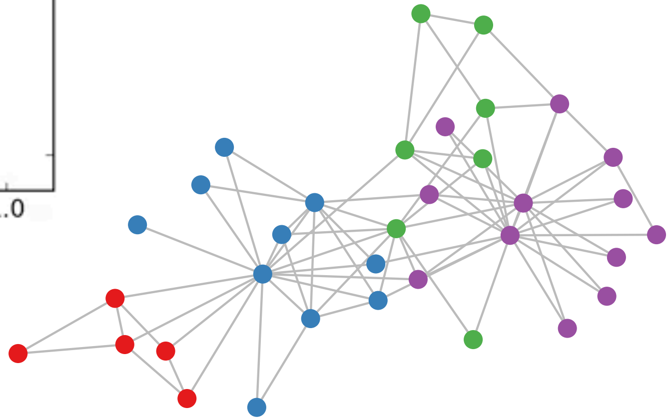
f() = 

[Karate Club Network]

Produces (useful?) random embeddings!

Add labels and train (semi-supervised)





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

Further reading

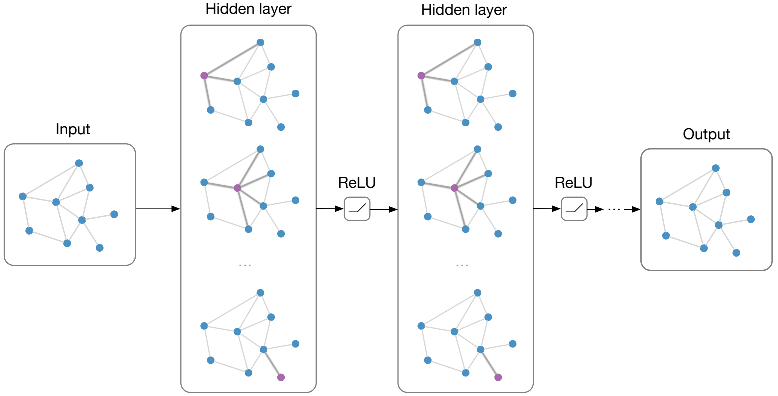
Blog post Graph Convolutional Networks:
http://tkipf.github.io/graph-convolutional-networks

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

Paper (Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, 2016):
https://arxiv.org/abs/1609.02907

Questions? You can get in touch with me via:

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



Interested in thesis projects? Get in touch!