



# Relational Representation Learning with Graph Neural Networks

Matthias Fey

[matthias.fey@tu-dortmund.de](mailto:matthias.fey@tu-dortmund.de)

   /rusty1s

# Roadmap

1. Motivation & Introduction
2. Common Operators
3. Efficient Computation of Graph Neural Networks
4. Relation to the Weisfeiler-Lehman Algorithm
5. Recent Extensions
  - Hierarchical Models
  - Deep(er) Models
  - Unsupervised Models
  - Generative Models
6. Conclusion

# Motivation & Introduction

# Motivation

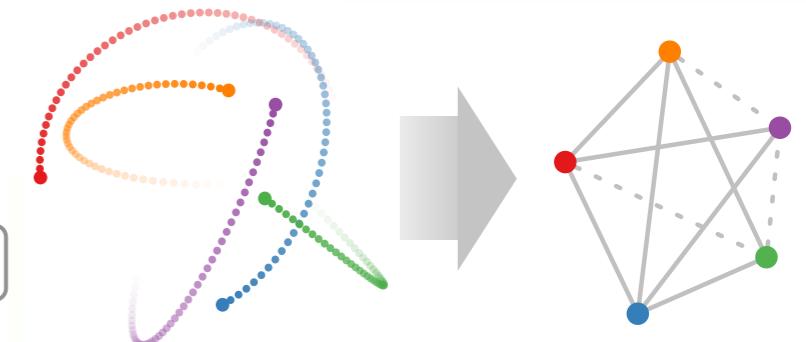
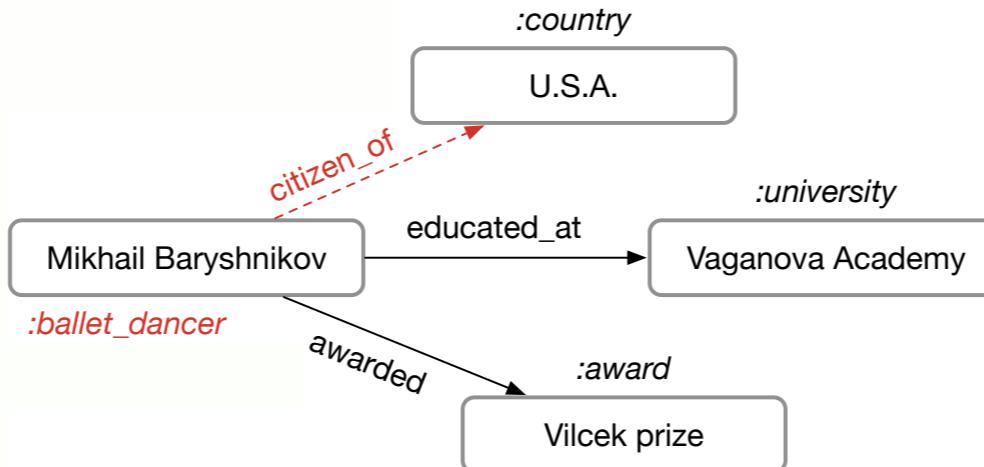
Enable learning on irregularly structured data, e.g., graphs, point clouds, and manifolds:

borrow from/generalize successful concepts on Euclidean domains, e.g., convolution

Enforce relational inductive biases into the model:  
enable learning about entities, relations, and rules for composing them

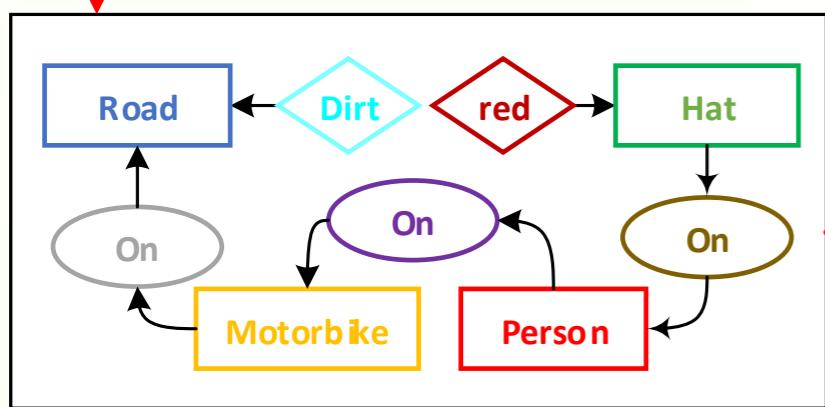
known as either **Geometric Deep Learning** or  
**Relational Representation Learning**

# A Non-Exhaustive List of Applications



Dynamics in  
Physical Systems

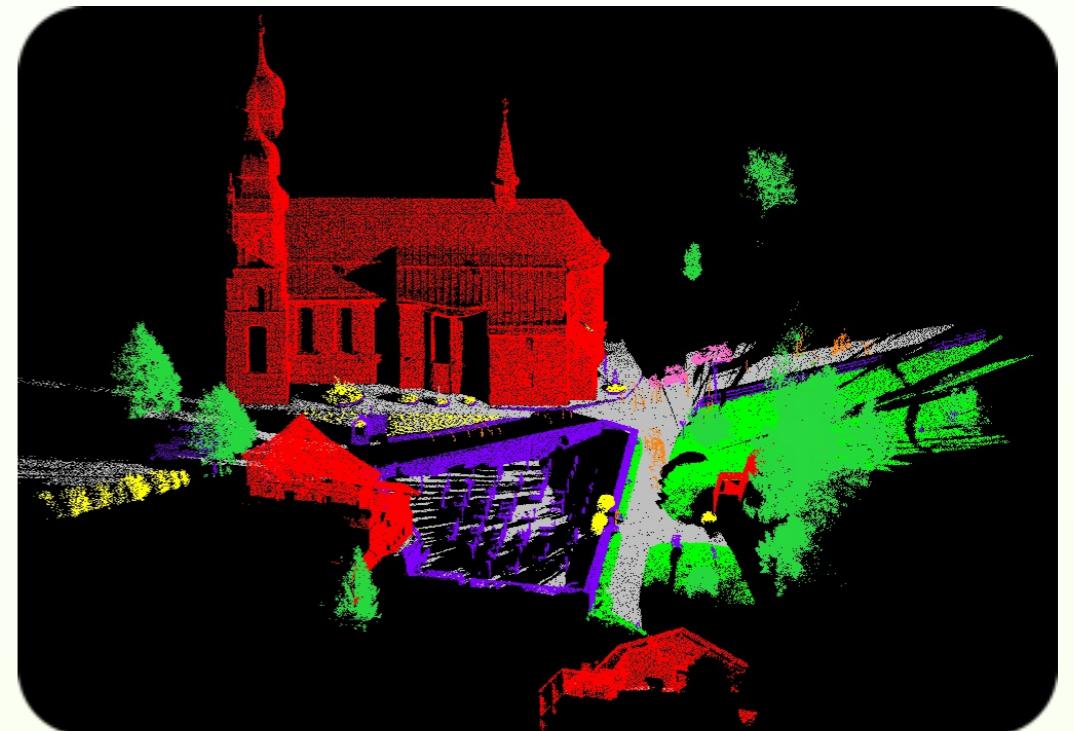
Reasoning in  
Knowledge Graphs



Visual Scene Understanding



Autonomous Driving

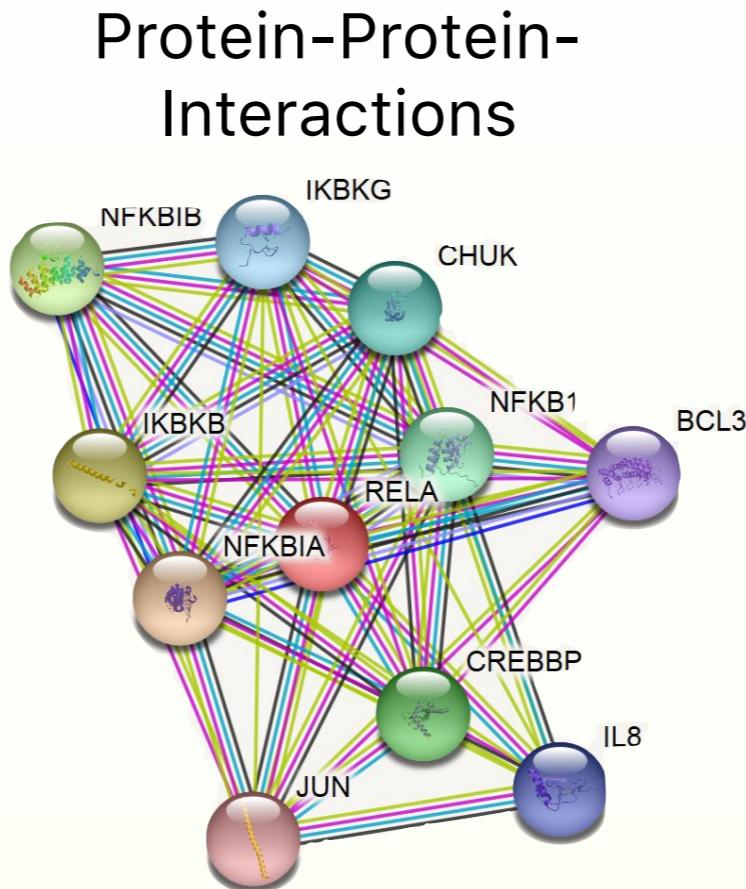


Point Cloud Analysis

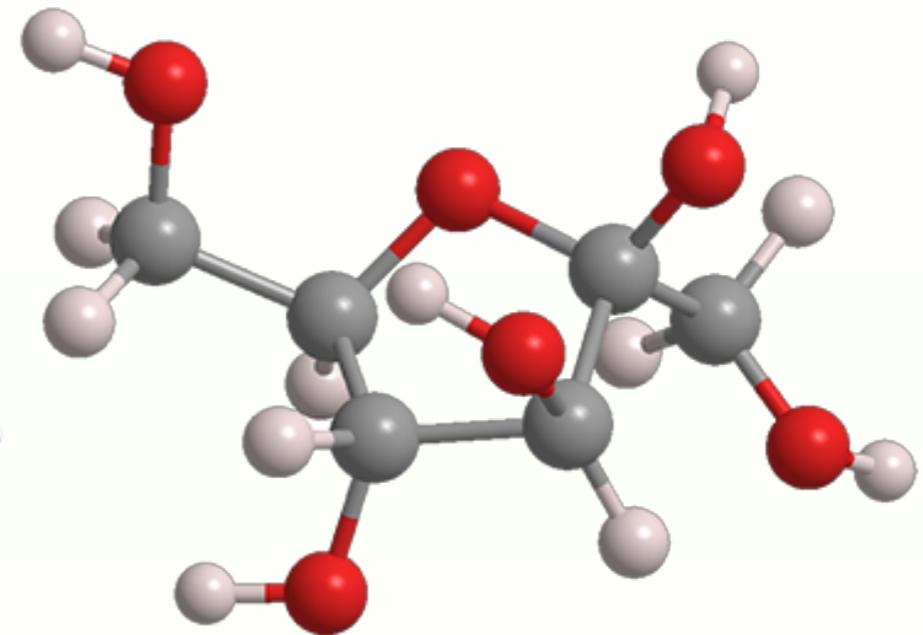
# A Non-Exhaustive List of Applications



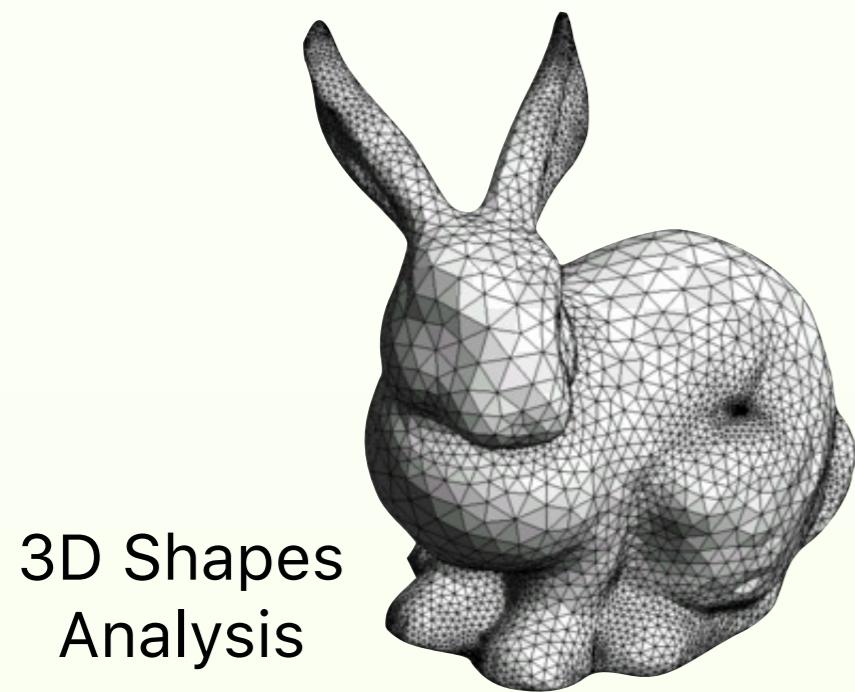
Social Networks



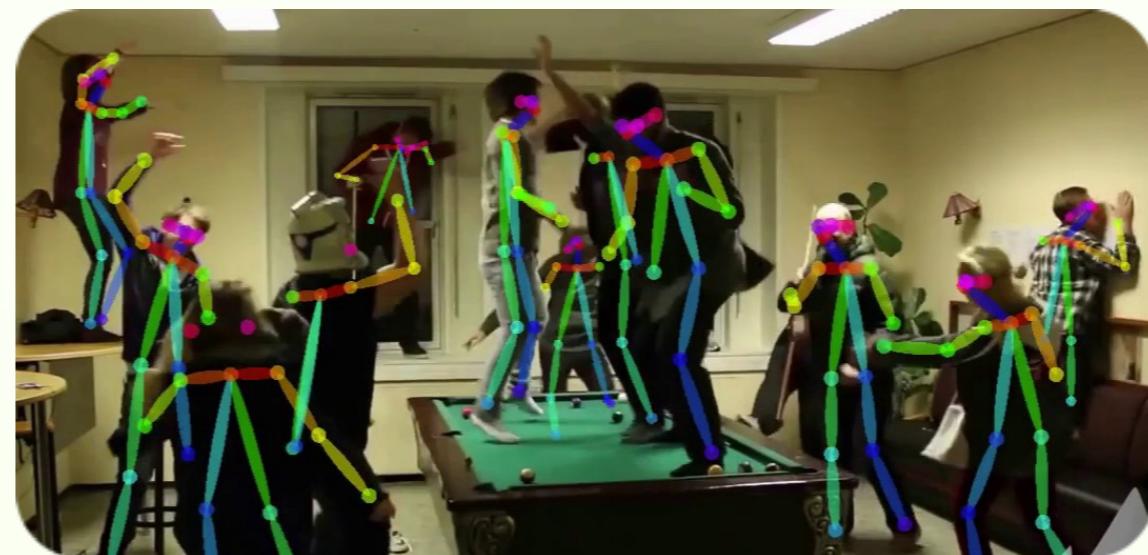
Protein-Protein-Interactions



Prediction of molecular properties  
and drug discovery



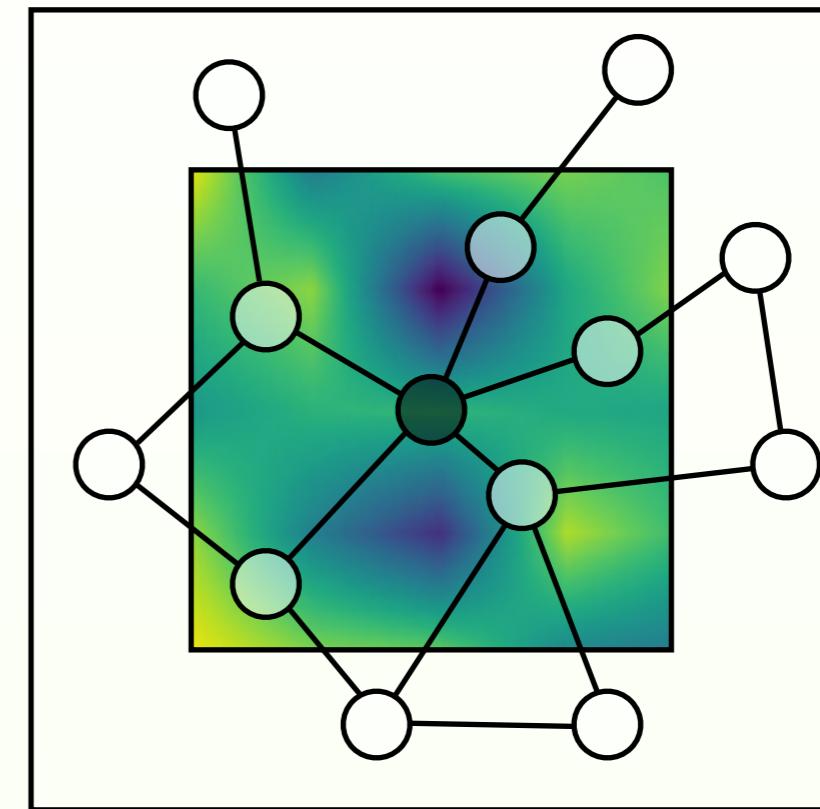
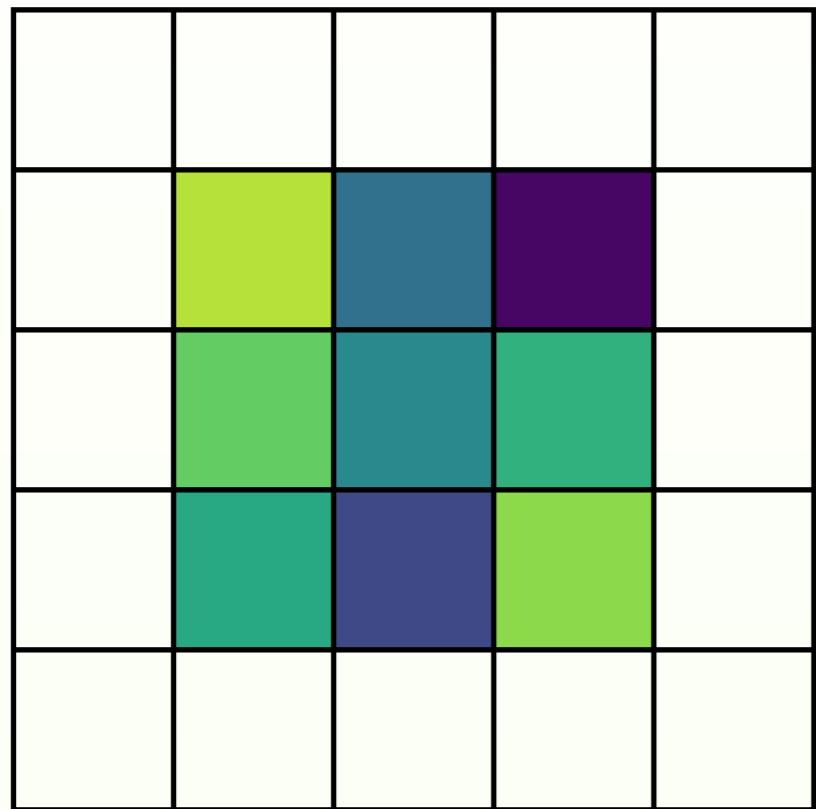
3D Shapes  
Analysis



Human Pose Estimation

# Graph Neural Networks (GNN)

Generalization of the convolution operator to graphs



dynamic positions  
dynamic number of neighbors

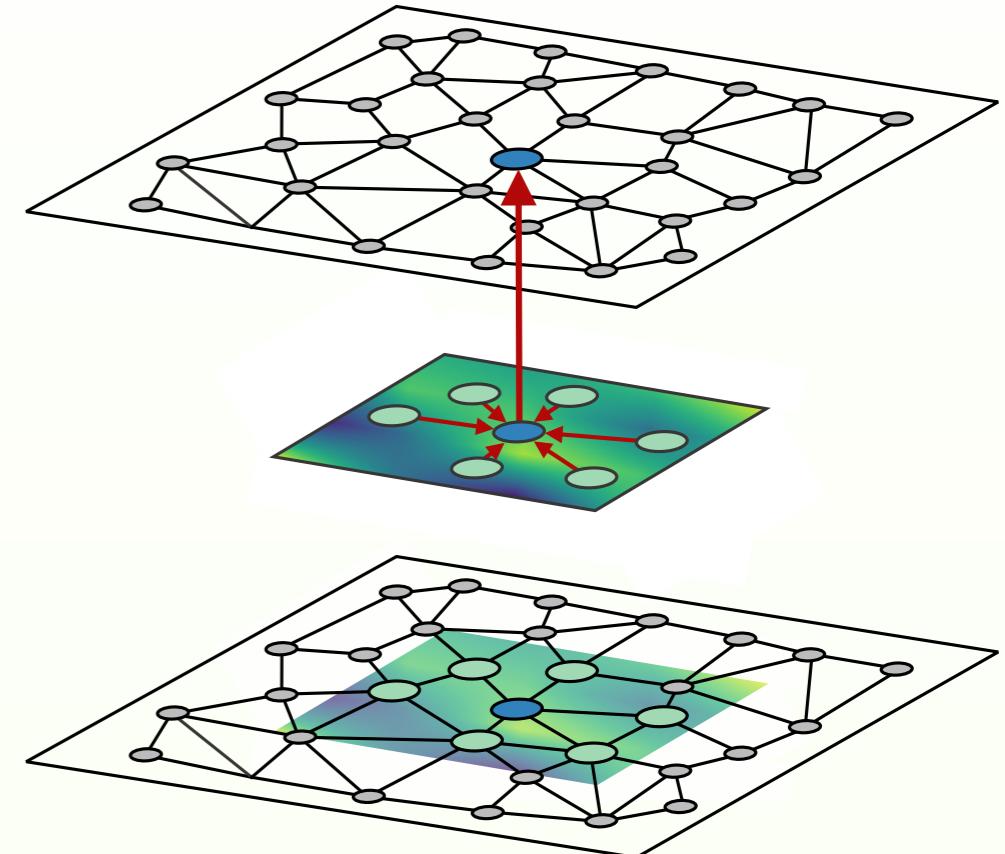
→ From discrete to continuous filters

# Graph Neural Networks (GNNs)

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

Node features  $\vec{h}_v^{(t)} \in \mathbb{R}^F$  in layer  $t$

GNNs can be expressed as a neighborhood aggregation or message passing scheme:



$$\vec{m}_v^{(t)} = \text{AGGREGATE}^{(t)} \left( \left\{ \vec{h}_w^{(t-1)} : w \in \mathcal{N}(v) \right\} \right)$$

permutation-  
invariant

$$\vec{h}_v^{(t)} = \text{COMBINE}^{(t)} \left( \vec{h}_v^{(t-1)}, \vec{m}_v^{(t)} \right)$$

# Common Operators

# Operators for Learning on Graphs

$$\vec{h}_v^{(t)} = \sigma \left( \Theta^{(t)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} C_{v,w}^{(t-1)} \vec{h}_w^{(t-1)} \right)$$

Non-  
Linearity      Trainable  
parameters      Added  
self-loops      Normalization  
coefficient

Normalization can be either ...

... static:  $C_{v,w}^{(t)} := 1$

... structure-dependent:  $C_{v,w}^{(t)} := |\mathcal{N}(v)|^{-1}$

... data-dependent, aka Attention

Xu et al.: How Powerful are Graph Neural Networks? (ICLR 2019)

Hamilton et al.: Inductive Representation Learning on Large Graphs (NIPS 2017)

Kipf and Welling: Semi-Supervised Classification with Graph Convolutional Networks (ICLR 2017)

Velickovic et al.: Graph Attention Networks (ICLR 2018)

# Operators for Learning on Point Clouds

PointNet(++):

$$\vec{h}_v^{(t)} = \max_{w \in \mathcal{N}(v) \cup \{v\}} \phi_{\Theta}^{(t)} \left( \vec{h}_w^{(t-1)}, \vec{p}_w - \vec{p}_v \right)$$

↑  
trainable  
function, e.g., MLP

↑  
relative Cartesian  
coordinates

Neighborhood is typically given by ball query or  
 $K$  nearest neighbor search.

Formulation can be easily extended to handle  
arbitrary edge features!

Qi et al.: Deep Learning on Point Sets for 3D Classification and Segmentation (CVPR 2017)

Qi et al.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space (NIPS 2017)

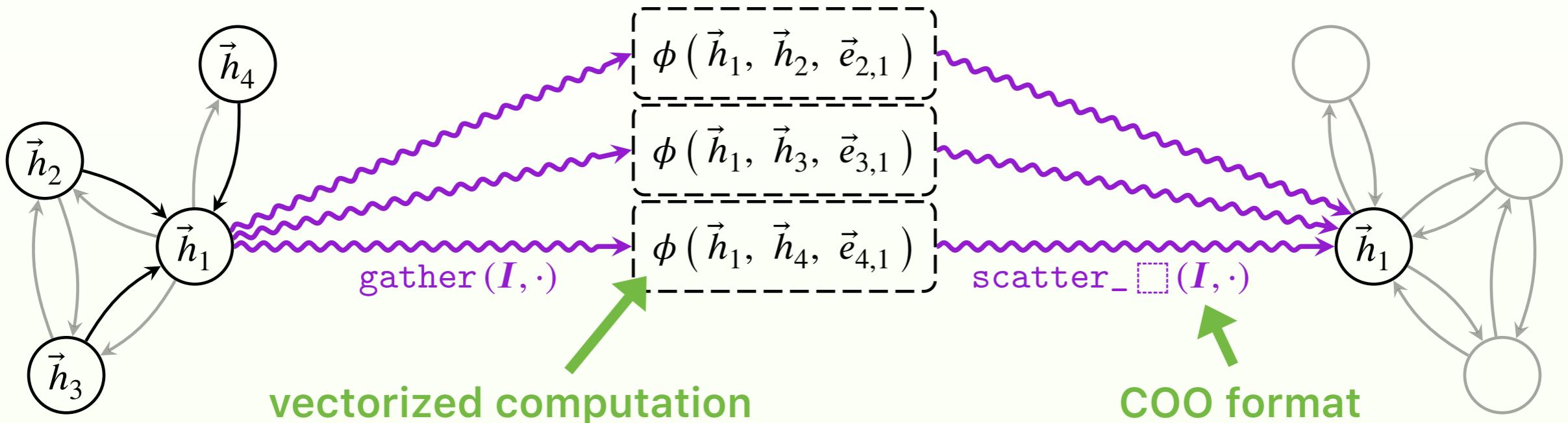
Fey et al.: SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels (CVPR 2018)

# **Efficient Computation of Graph Neural Networks**

# Efficient Parallelization of GNNs

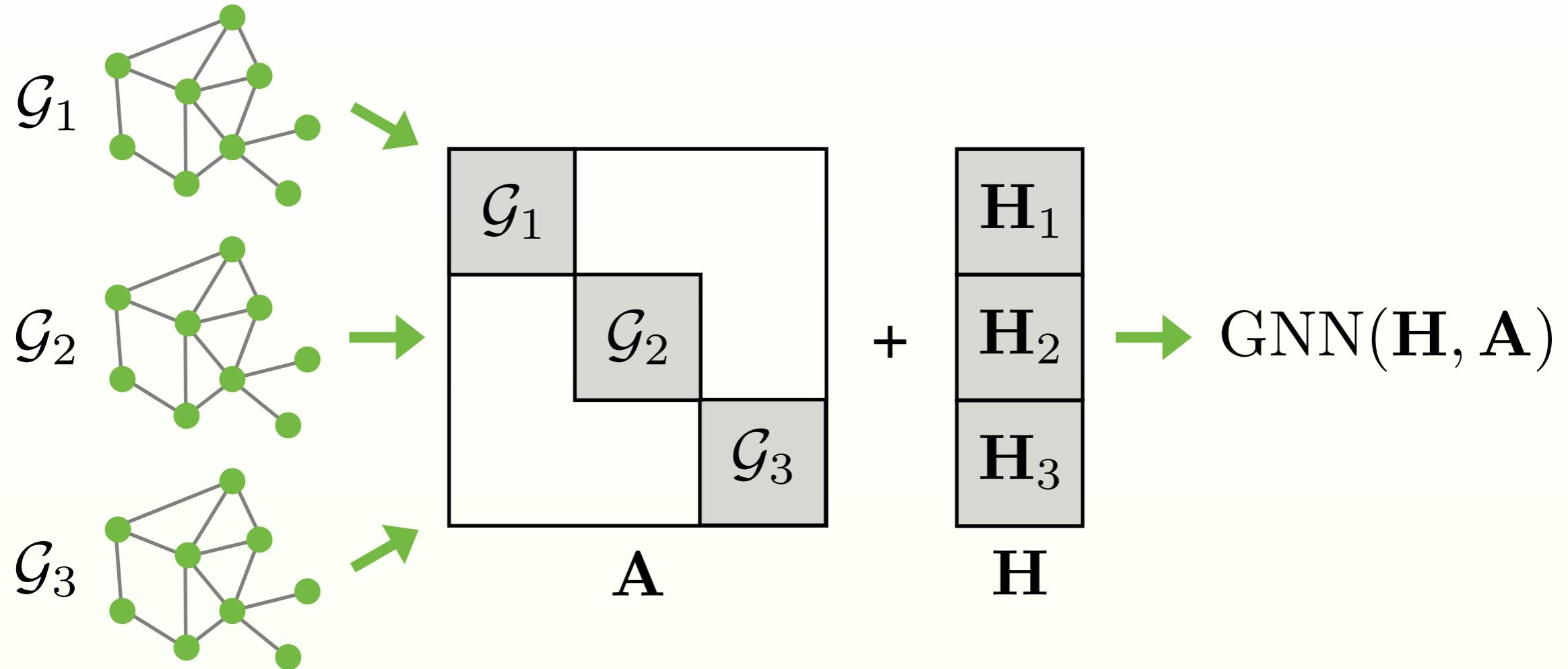
$$\text{AGGREGATE}^{(t)}(v) = \square_{w \in \mathcal{N}(v)} \phi_{\Theta}^{(t)} \left( \vec{h}_v^{(t-1)}, \vec{h}_w^{(t-1)}, \vec{e}_{w,v}^{(t-1)} \right)$$

$\square$  is placeholder for either sum, mean, or max



1. Gather node information into edge parallel space
2. Scatter edge information into node parallel space

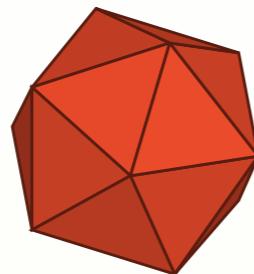
# Mini-Batching in GNNs



No additional overhead due to sparse layout!

Input examples can have different size!

# PyTorch Geometric: A Fast GNN Library

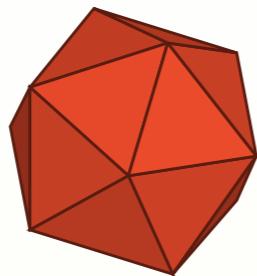


PyTorch  
geometric

- ✓ provides over 25 GNN implementations
- ✓ access to over 100 benchmark datasets
- ✓ extendable via a simple Message Passing API
- ✓ leverages dedicated CUDA kernels
- ✓ supports multi-GPUs
- ✓ thoroughly documented

[https://github.com/rusty1s/  
pytorch\\_geometric](https://github.com/rusty1s/pytorch_geometric)

# PyTorch Geometric: A Fast GNN Library



PyTorch  
geometric

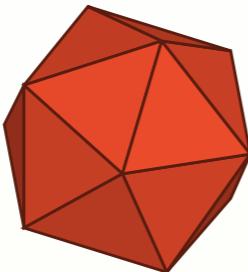
```
class MyOwnNet(Module):
    def __init__(self, in_channels, out_channels):
        self.conv1 = GCNConv(in_channels, 16)
        self.conv2 = GCNConv(16, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = relu(x)
        x = self.conv2(x, edge_index)
        x = softmax(x, dim=1)
        return x
```

layer  
initialization

network  
execution flow

# PyTorch Geometric: A Fast GNN Library



# PyTorch geometric

```
class MyOwnConv(MessagePassing):
    def __init__(self, ...):
        super(MyOwnConv, self).__init__('add')  
  
    def forward(self, x, edge_index):
        return self.propagate(edge_index, x=x)  
  
    def message(self, x_i, x_j):
        # Construct messages to aggregate.  
        ↑  
        Features get automatically mapped to  
        respective source and target nodes
```

add, mean or max  
aggregation

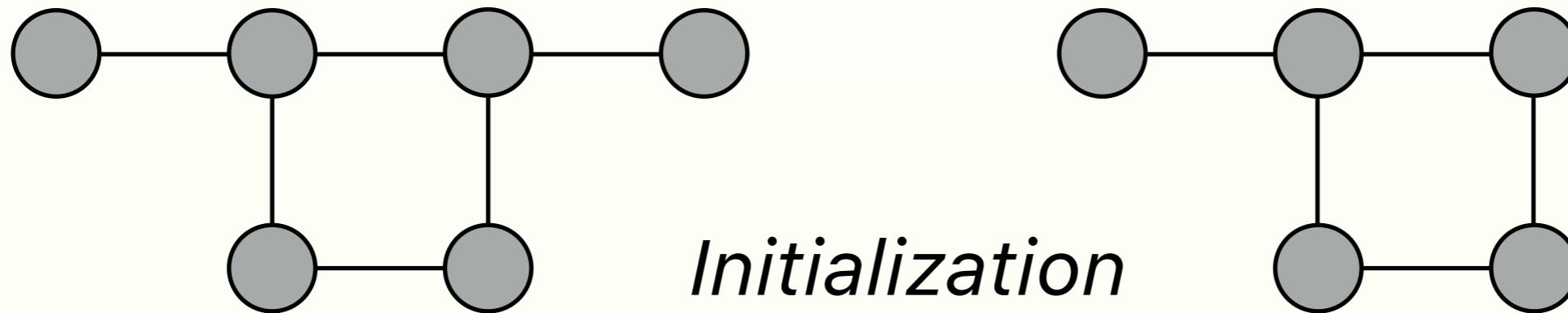
pass every-  
thing needed to  
construct messages

# Relation to the Weisfeiler-Lehman Algorithm

# How Powerful are Graph Neural Networks?

WL-Test - A Graph Isomorphism Heuristic via iterative color refinement:

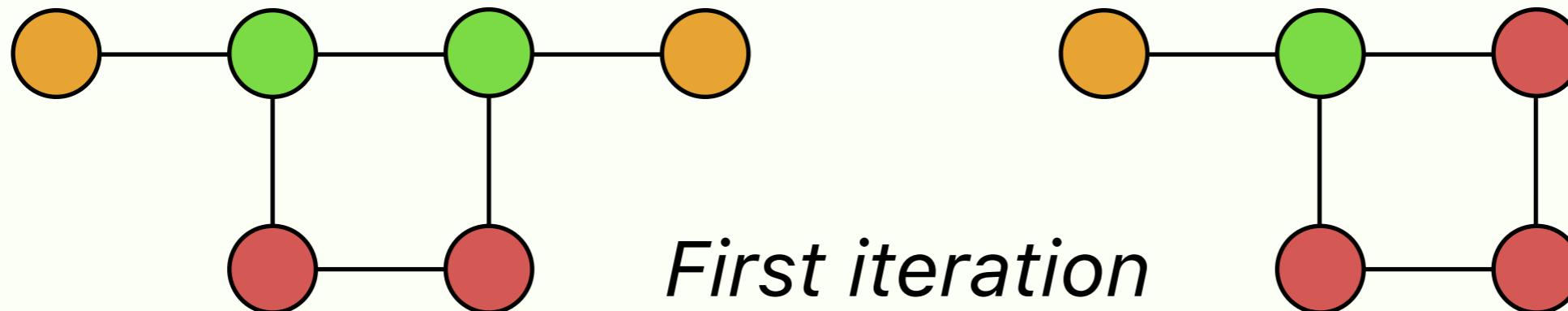
1. Aggregate colors from neighboring nodes
2. Hash aggregated colors into unique new colors



# How Powerful are Graph Neural Networks?

WL-Test - A Graph Isomorphism Heuristic via iterative color refinement:

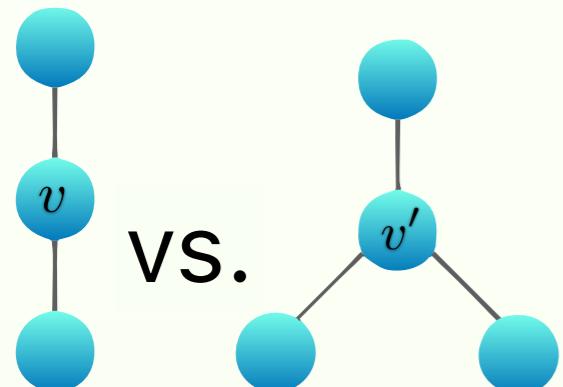
1. Aggregate colors from neighboring nodes
2. Hash aggregated colors into unique new colors



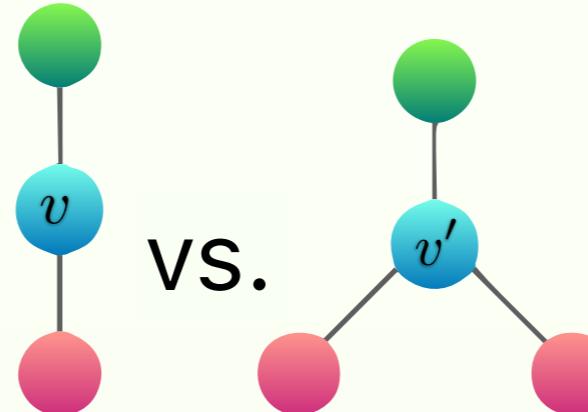
3. Graphs are non-isomorphic if color histograms differ after  $T$  rounds

# How Powerful are Graph Neural Networks?

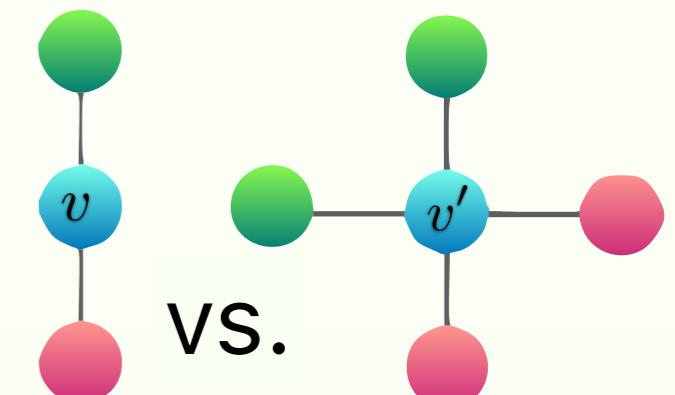
- GNNs are at most as powerful as the WL-Test in distinguishing graph structures
- GNNs are as powerful as the WL-Test *iff* AGGREGATE and COMBINE are injective
- Use sum instead of mean or max aggregation



mean/max fail



max fails



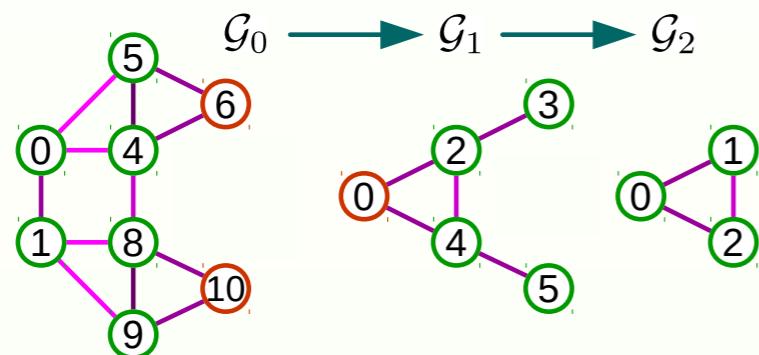
mean/max fail

# Recent GNN Extensions

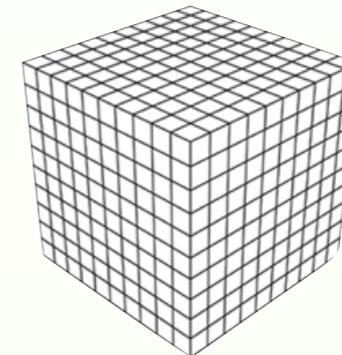
# Hierarchical Models

Either via deterministic coarsening methods...

**Graclus**  
(greedy pair matching)



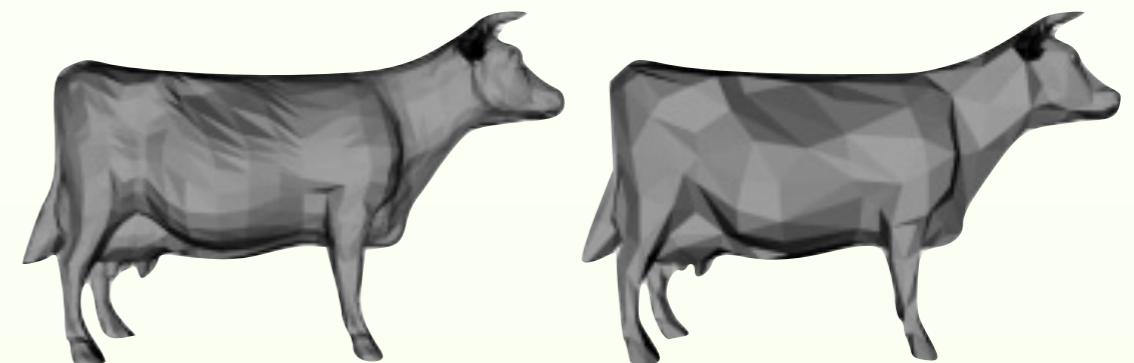
**Voxel Grids**  
(pool\_2d generalisation)



**Iterative Sampling**  
of the most distant points



**Mesh Downsampling**  
via quadric matrices



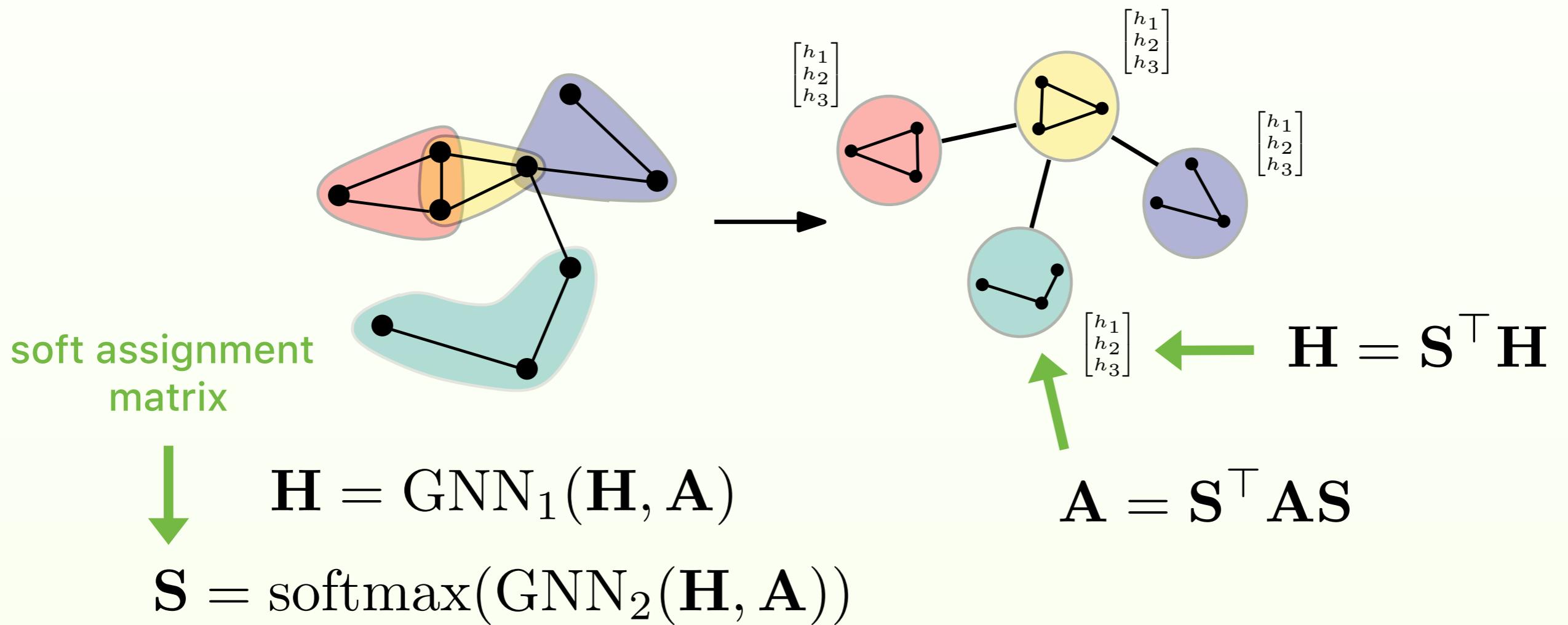
- Defferrard et al.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)  
Simonovsky and Komodakis: Dynamic Edge-Conditioned Filters in CNNs on Graphs (CVPR 2017)  
Qi et al.: PointNet++ Deep Hierarchical Feature Learning on Point Sets in a Metric Space (NIPS 2017)  
Ranjan et al.: Generating 3D Faces using Convolutional Mesh Autoencoders (ECCV 2018)

# Hierarchical Models

... or differentiable pooling modules

DiffPool:

One GNN learns node embeddings  
Another GNN learns cluster assignments

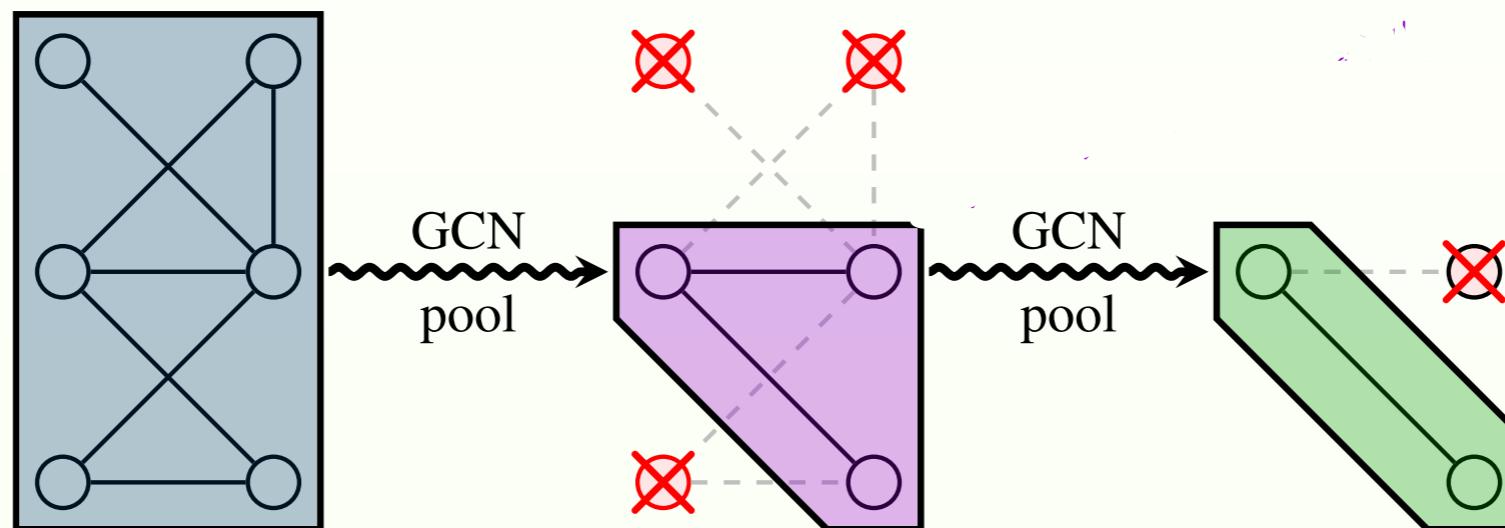


# Hierarchical Models

... or differentiable pooling modules

Top-k Pooling:

Filter nodes based on scoring function



$$\vec{y} = \tanh(\mathbf{H} \cdot \vec{\theta})$$

ensure gradient computation

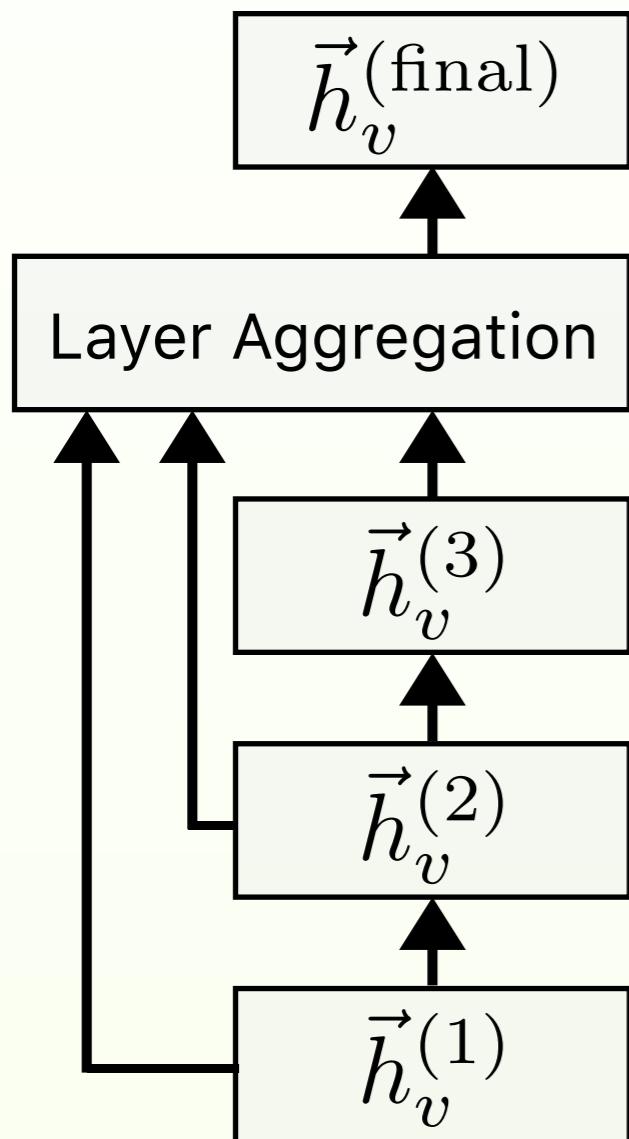
$$\vec{i} = \text{top}_k(\vec{y})$$

$$\mathbf{H} = (\mathbf{H} \odot \vec{y})_{\vec{i}}$$

# Deep(er) Models

"Washed out" representations after just a few layers

Idea: "Jump back" to earlier representations



Concatenation:  $\vec{h}_v^{(1)} \parallel \dots \parallel \vec{h}_v^{(T)}$

Pooling:

$$\max\left(\vec{h}_v^{(1)}, \dots, \vec{h}_v^{(T)}\right)$$

Weighted Sum:

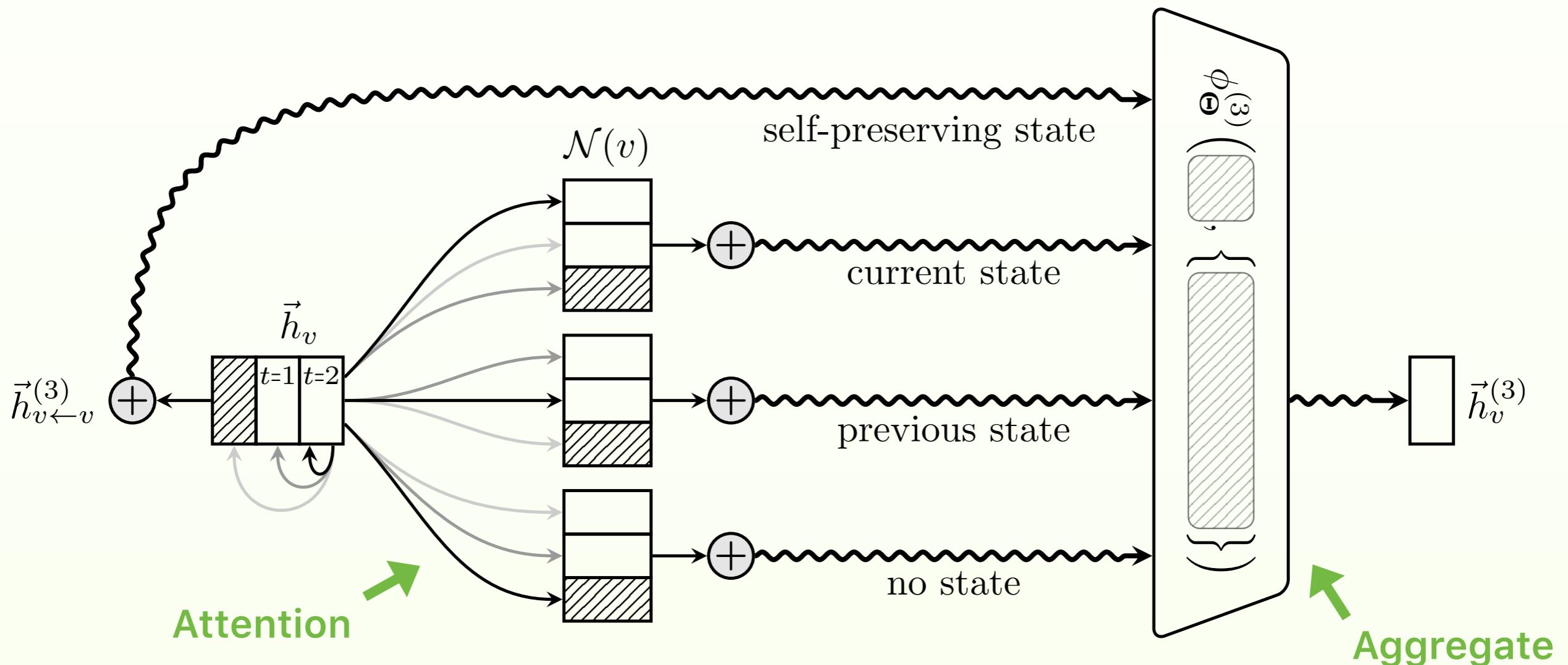
$$\sum_{t=1}^T \alpha_v^{(t)} \vec{h}_v^{(t)}$$

Attention

# Deep(er) Models

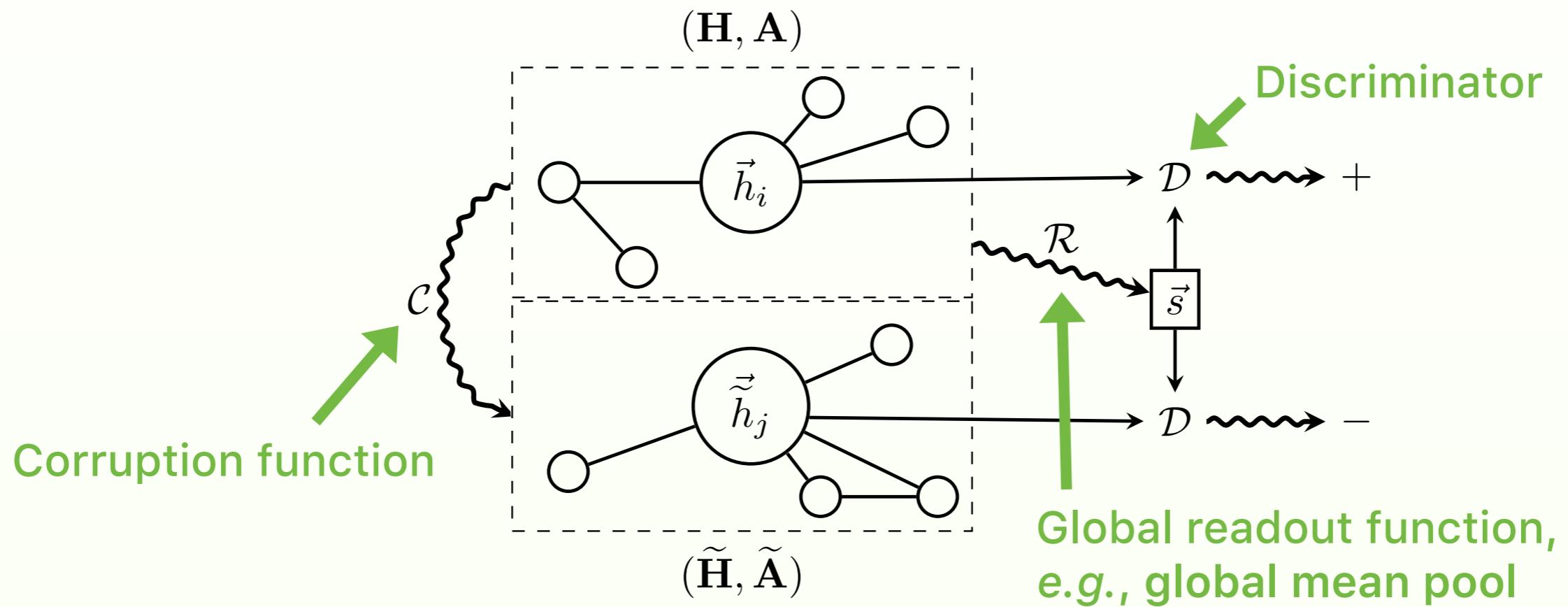
**Extension:** Allow jumps directly *while aggregating*

- Aggregate more global information in one branch while falling back to more local information in others



# Unsupervised Models

based on Message Passing-based models



Maximize mutual information:

$$\sum_{v \in \mathcal{V}} \mathbb{E}_{(\mathbf{H}, \mathbf{A})} \left[ \log \mathcal{D} \left( \vec{h}_v, \vec{s} \right) \right] + \mathbb{E}_{(\tilde{\mathbf{H}}, \tilde{\mathbf{A}})} \left[ \log \left( 1 - \mathcal{D} \left( \tilde{\vec{h}}_v, \vec{s} \right) \right) \right]$$

# Generative Models

(Variational/Adversarially Regularized) Autoencoder:  
GNN encoder and inner product/MLP decoder

→ expensive graph matching procedure, prefixed size

Generative Adversarial Nets for Graphs:  
MLP generator and GNN discriminator

→ likelihood-free, but still prefixed in size

Auto-regressive Graph Generation:  
Decomposing of graph generation into a sequence  
of node and edge formations

→ not prefixed in size, but inherently sequential

Kipf and Welling: Variational Graph Auto-Encoders (NIPS-W 2016)

Pan et al.: Adversarially Regularized Graph Autoencoder for Graph Embedding (IJCAI 2018)

Simonovsky and Komodakis: GraphVAE: Towards Generation of Small Graphs Using VAEs (arXiv/1802.03480)

Cao and Kipf: MolGAN: An Implicit Generative Model for Small Molecular Graphs (ICML-W 2018)

You et al.: Generating Realistic Graphs with Deep Auto-regressive Models (ICML 2018)

# Conclusion

GNNs have interesting applications in many different areas

GNNs follow a simple message passing scheme to learn *localized* embeddings

GNNs can be very efficiently implemented

GNNs can be enhanced by and used for various concepts

GNNs are really hot right now 🔥🔥🔥

