Graph Neural Networks
3D Deep Learning

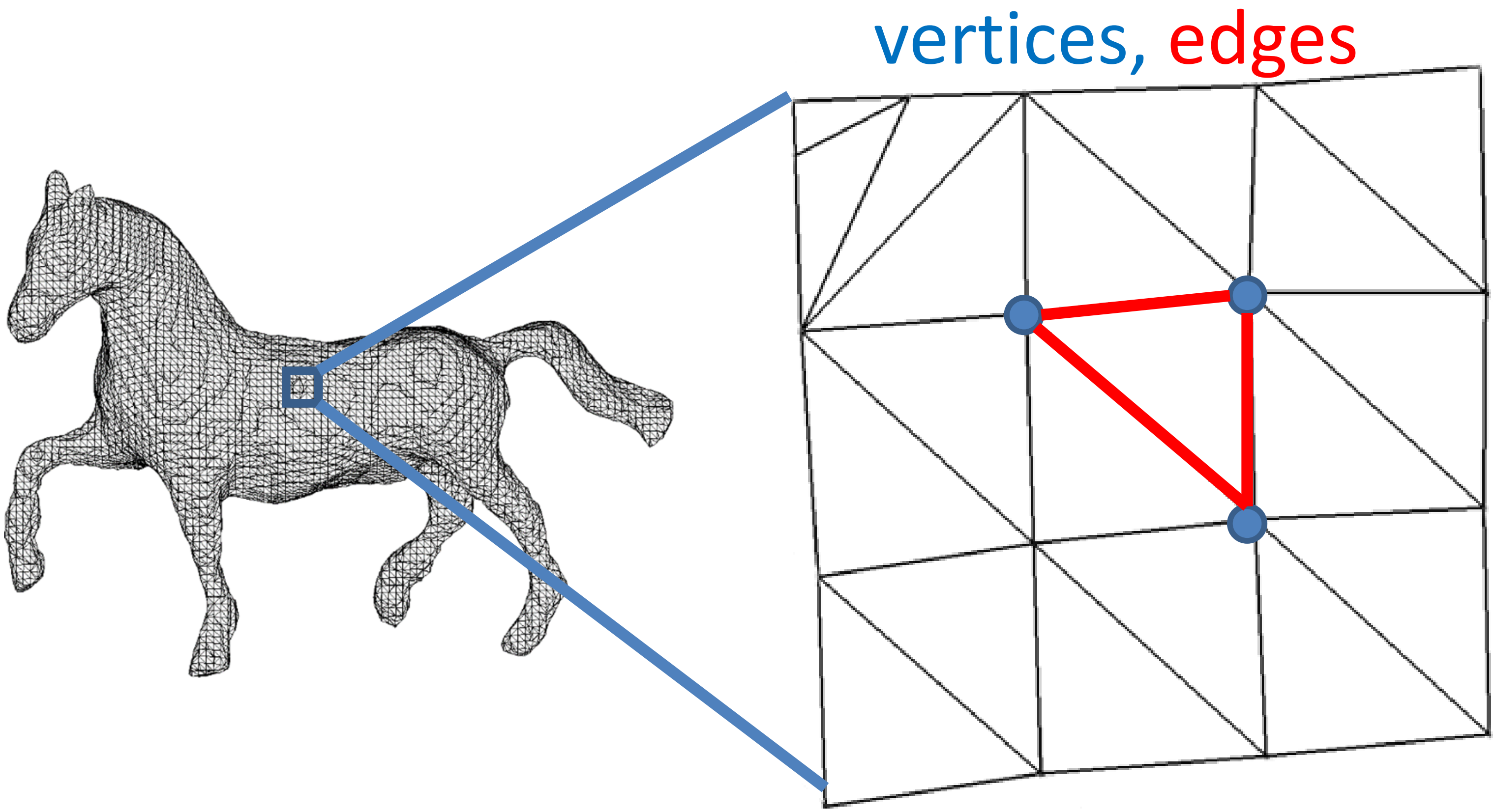
Intelligent Visual Computing
Evangelos Kalogerakis

3D Deep Learning approaches

- The Multi-View approach
- The Voxel approach
- The Point approach
- **The Graph approach**

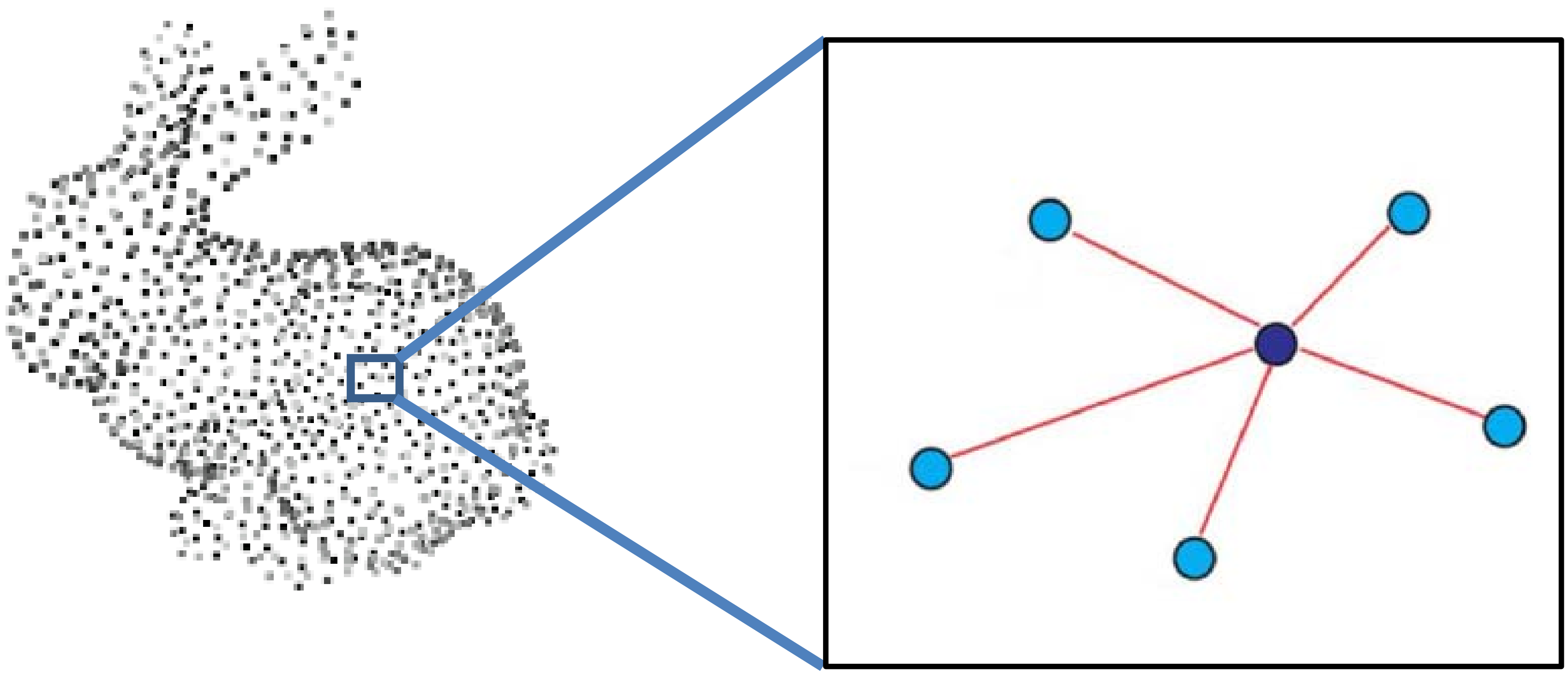
Meshes as Graph

Using the shape representation (mesh) directly



Point clouds as graphs

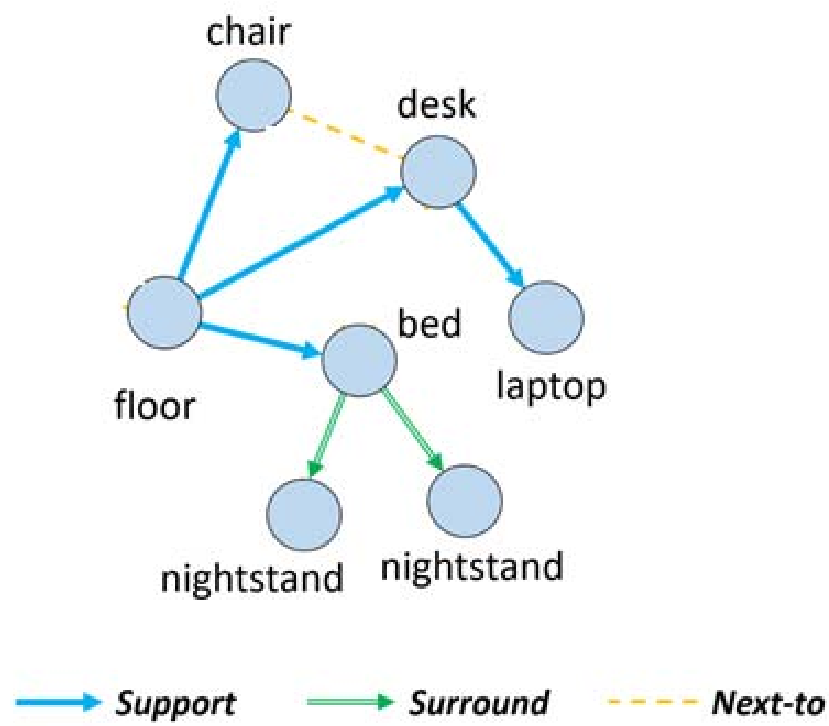
Connect each point to its K-nearest neighbors or all points within a Euclidean ball



Scenes as a Graph

Represent scene (an arrangement of shapes) as a graph

3D indoor scene example (bedroom)



Setup for graph nets

Input:

{Vertices V, Edges E}

$\boldsymbol{X} = \{ \boldsymbol{x_i} \}$ are per-vertex raw representations

$\boldsymbol{Y} = \{ \boldsymbol{y_{i,j}} \}$ are per-edge raw representations (optional)

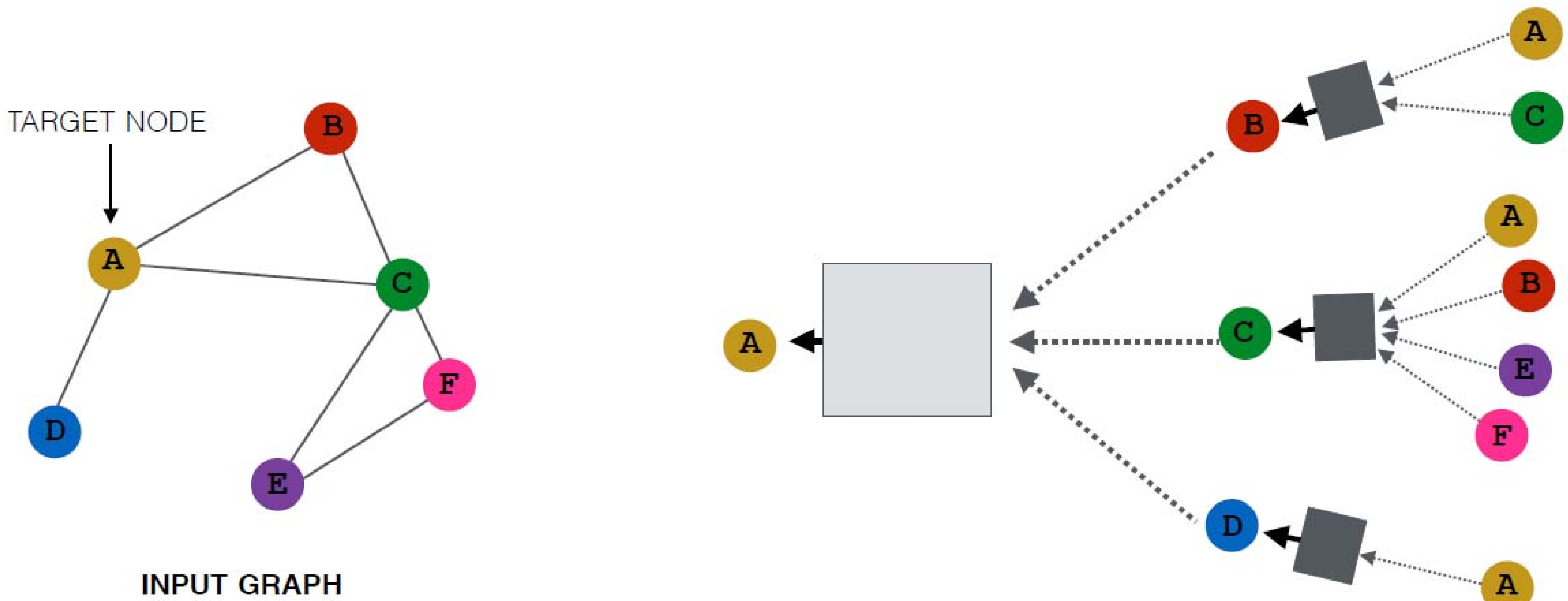
Output:

a) Learned node/edge representations – these can be used for node/edge tasks (e.g., node/edge classification)

b) Learned global graph representation -- this can be used for graph recognition tasks (e.g., classify a graph)

Key Idea

Infer node representations (embeddings) based on neighborhoods.

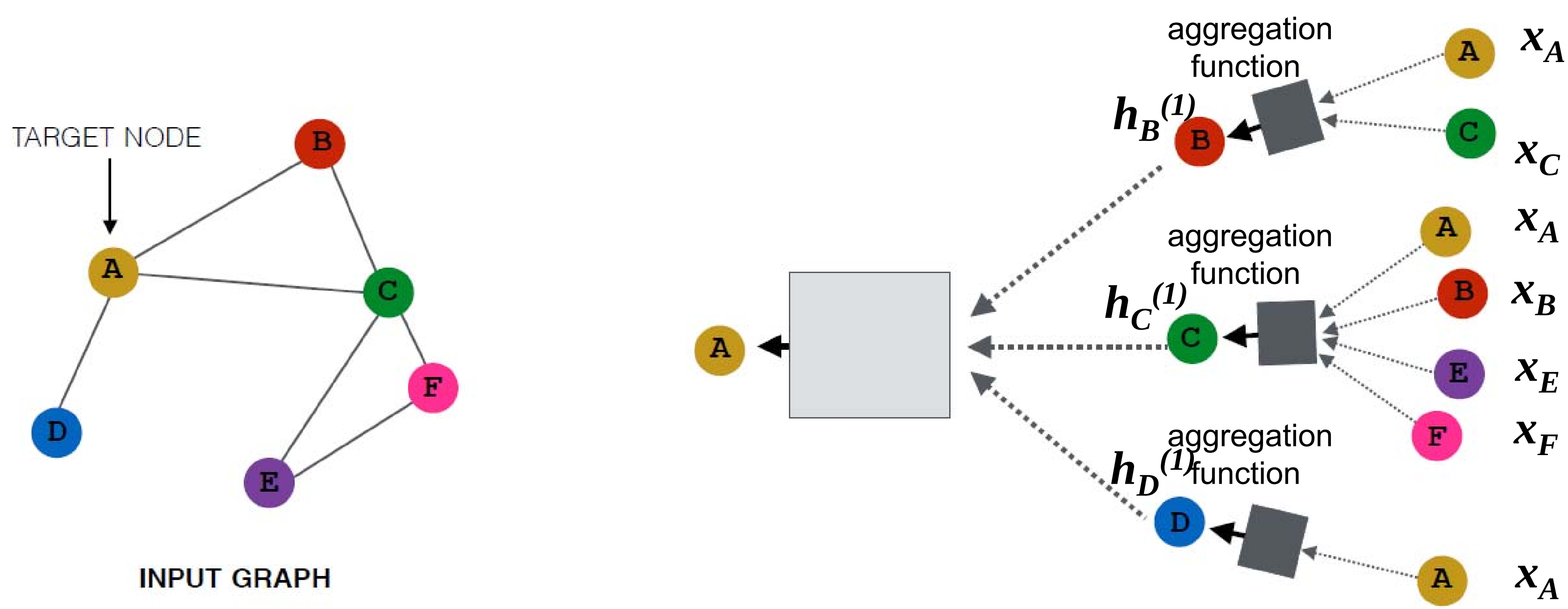


Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

Infer node representations (embeddings) based on neighborhoods.

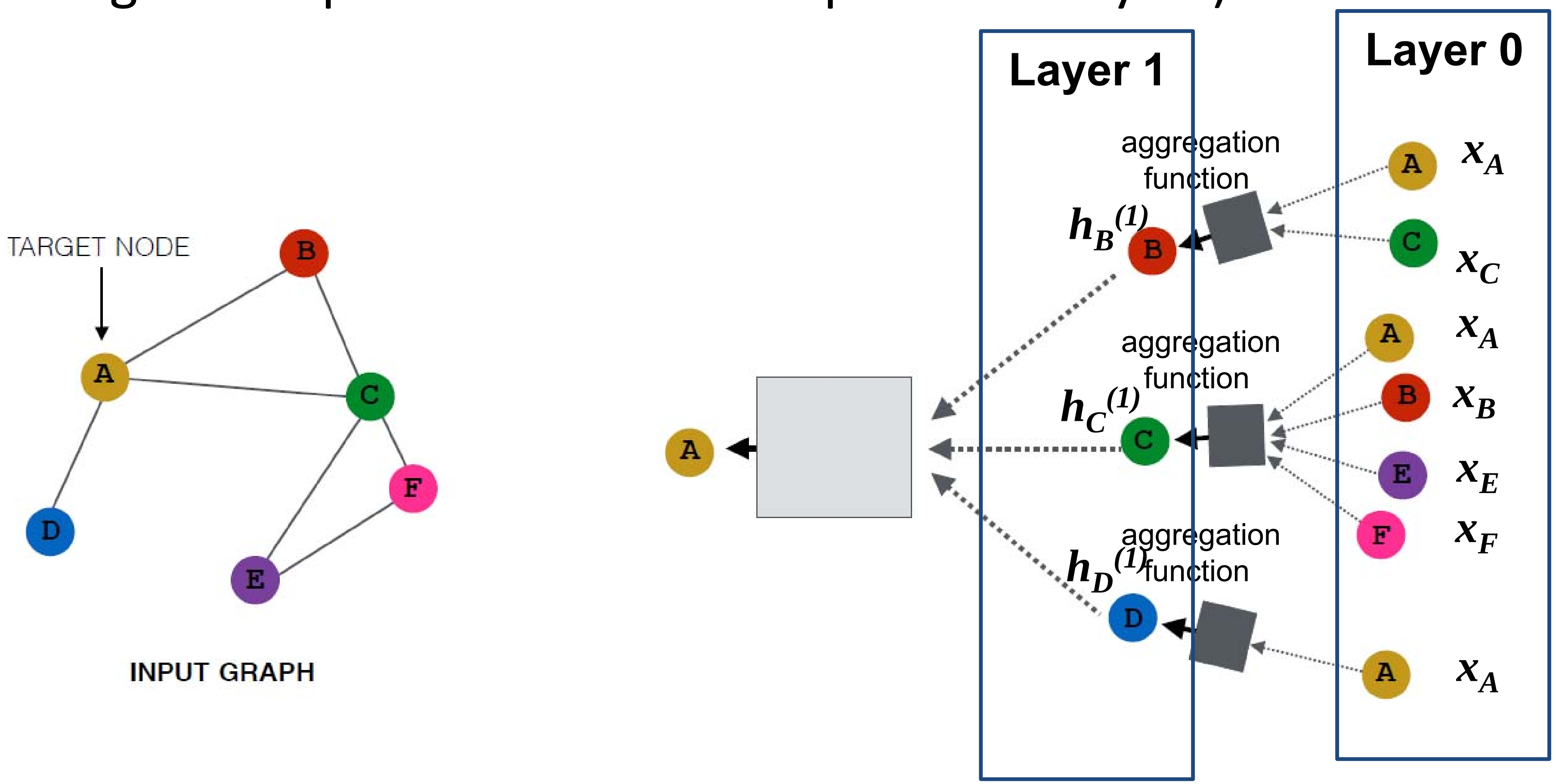
Nodes **aggregate information** (messages) from their neighbors.



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

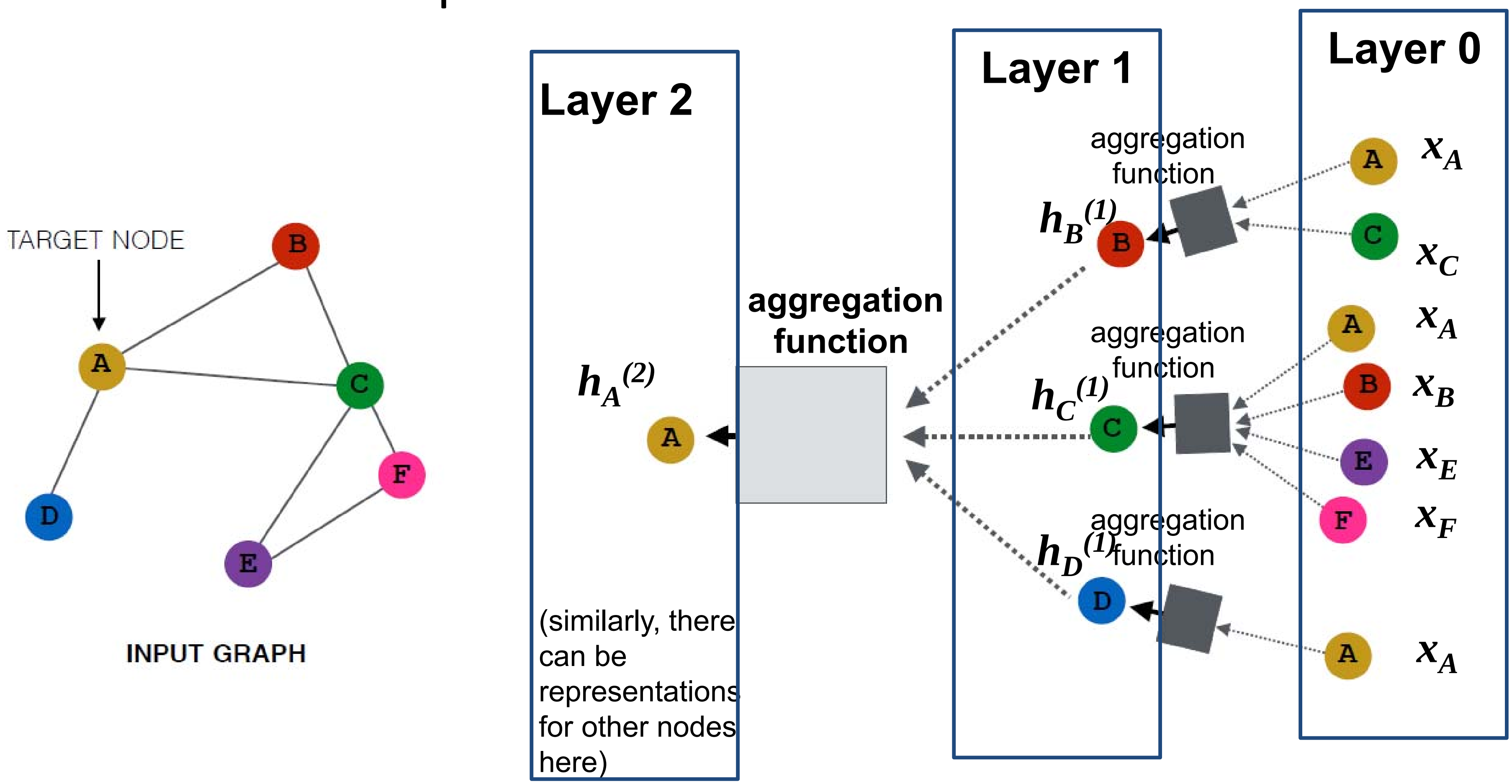
The result of aggregation is a new representations for each node.
(think again the process is a forward pass over layers)



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

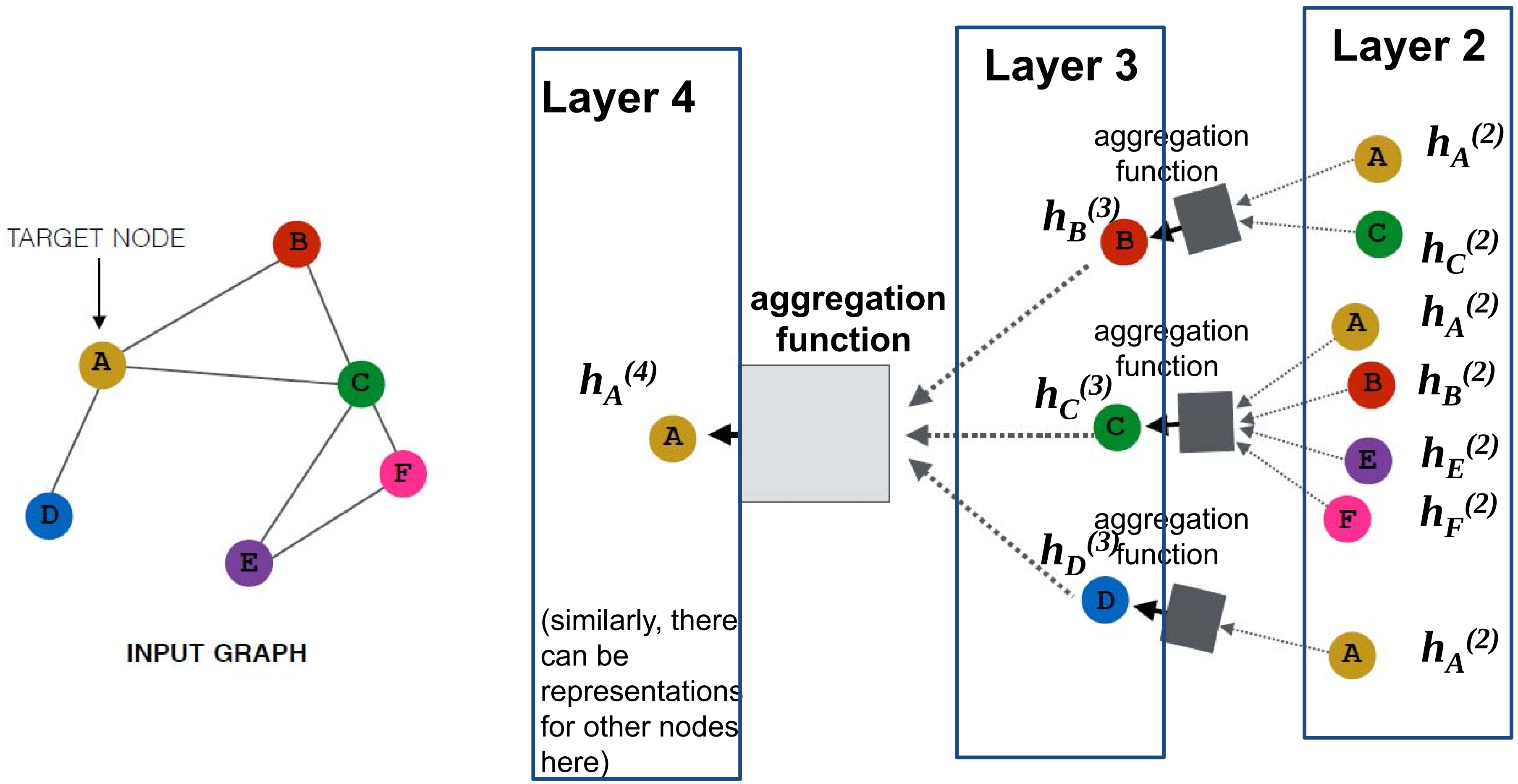
The resulting representations can be in turn aggregated again to produce new node representations.



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

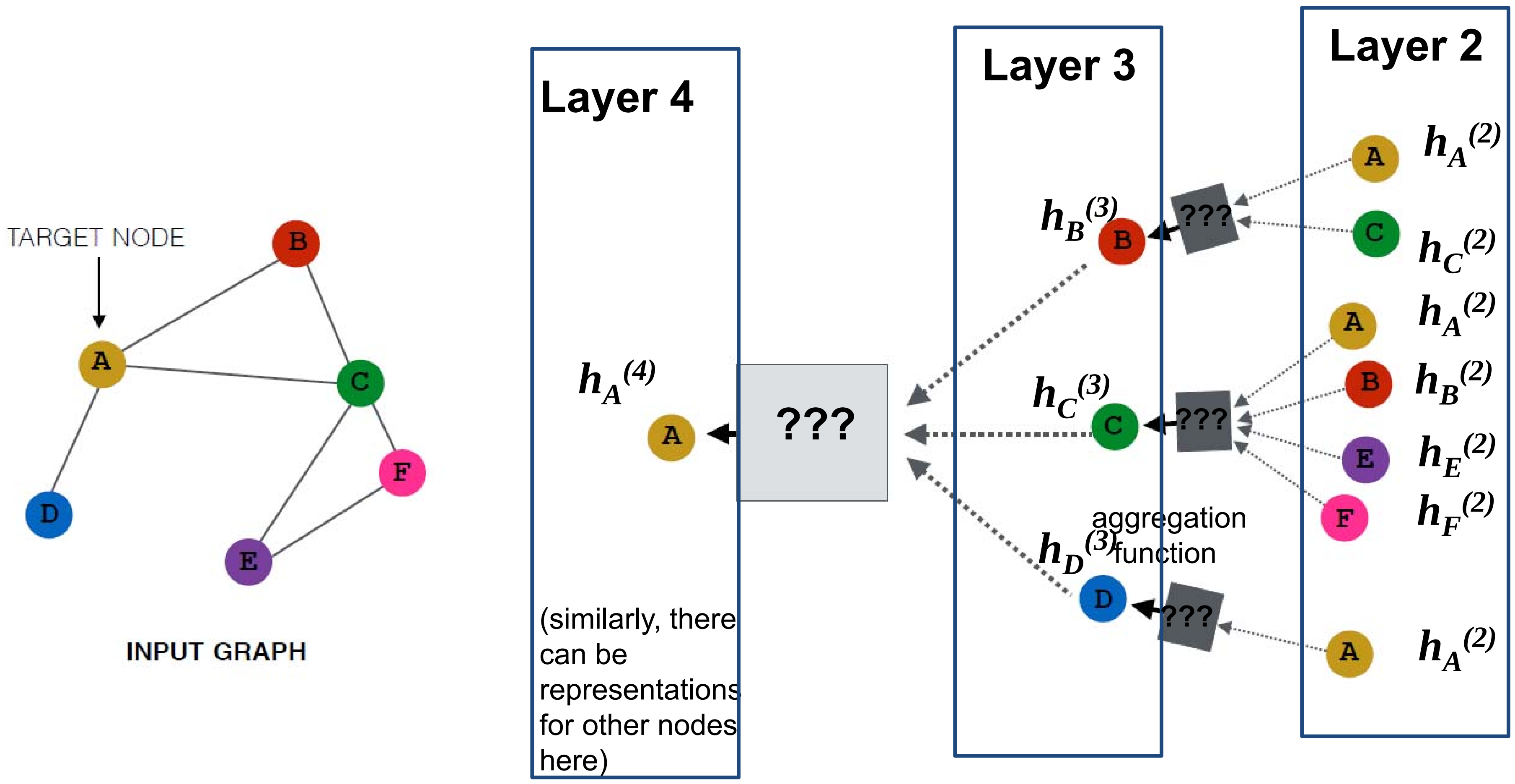
... which can in turn used in further aggregations, node updates and so on....



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

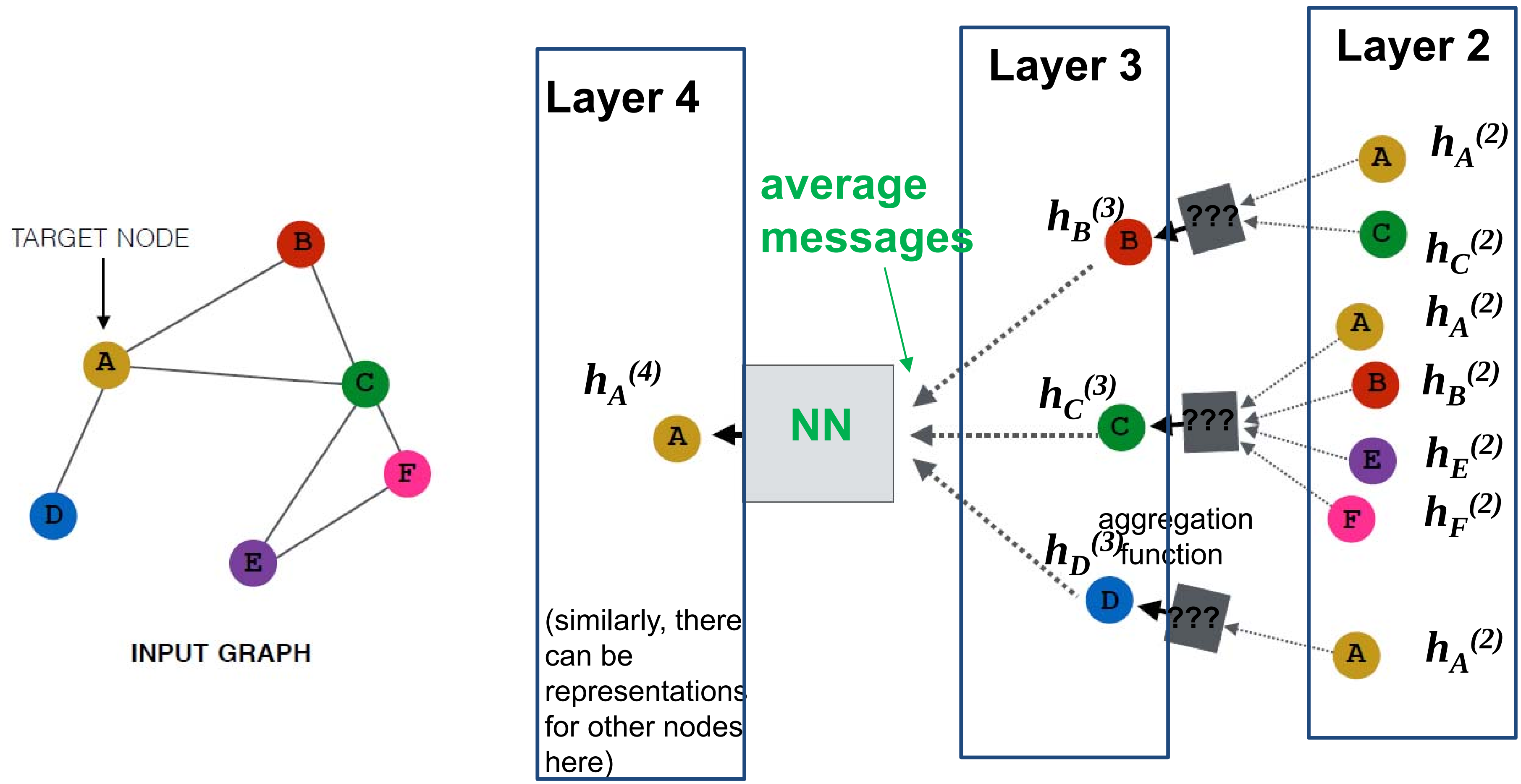
What are these "aggregation functions"?



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Key Idea

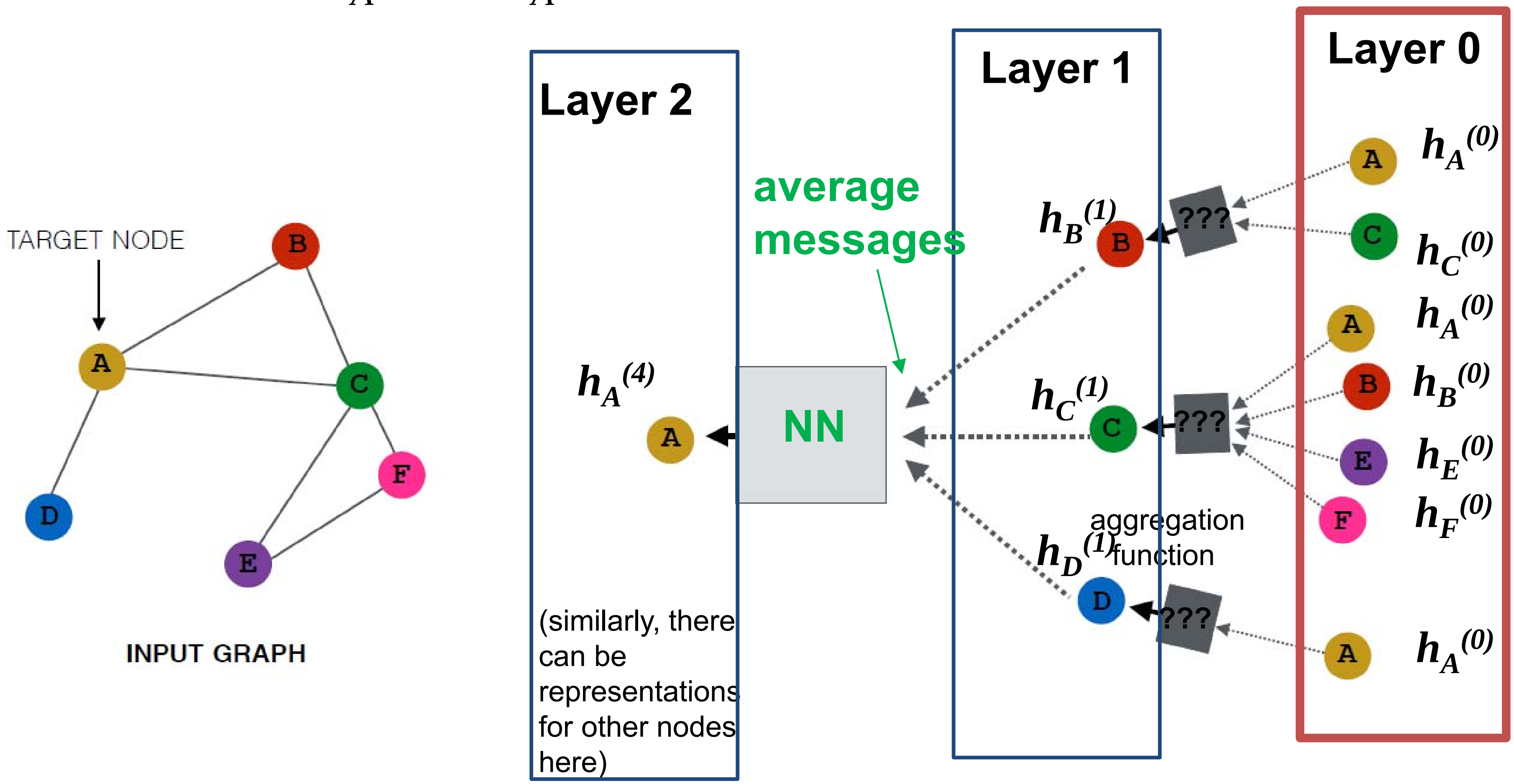
One idea: average messages, then NN



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Simplest approach

$$Initialization: \quad \mathbf{h}_A^{(0)} = \mathbf{x}_A$$

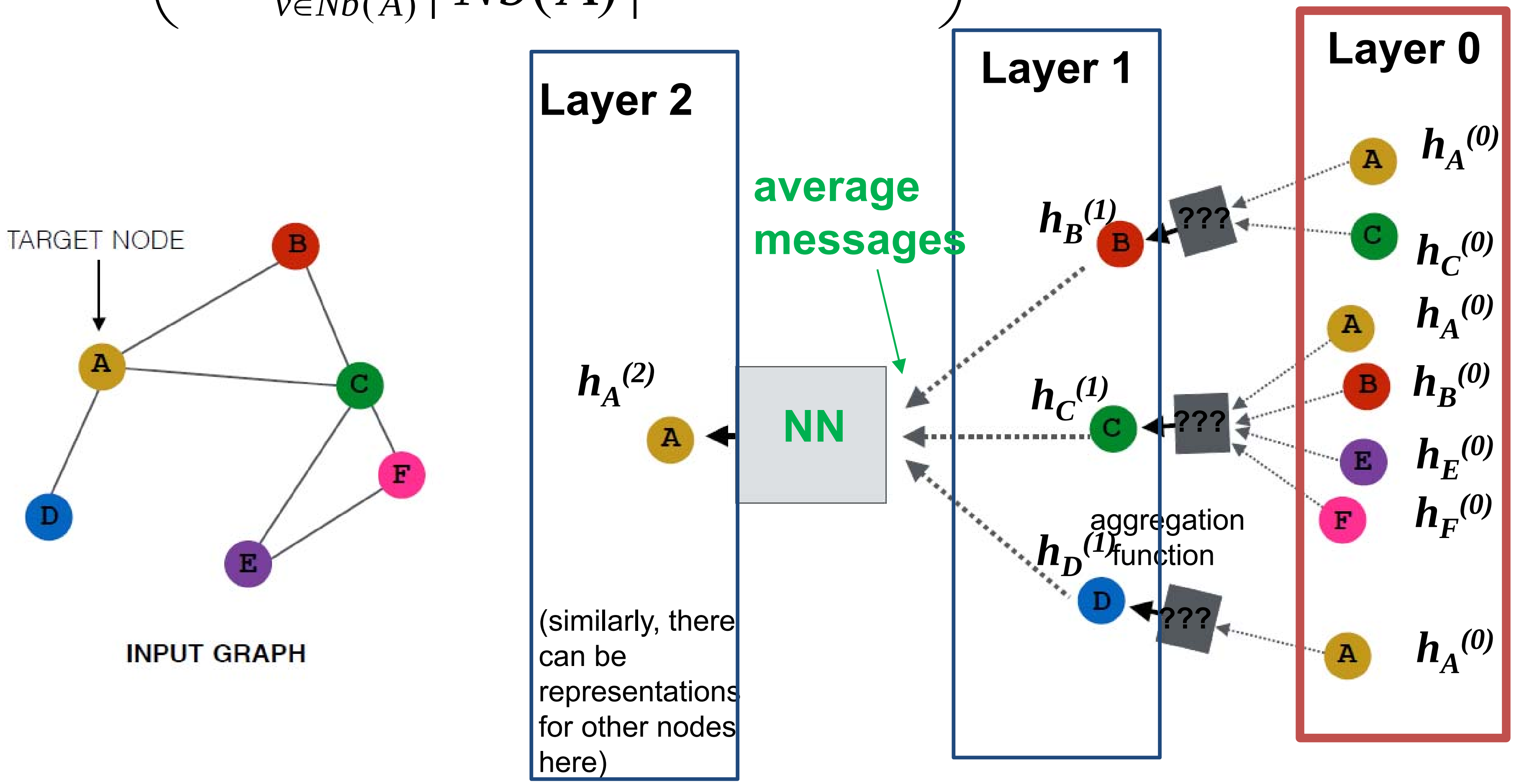


Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Simplest approach

Updates :

$$\mathbf{h}_A^{(k)} = g\left(\mathbf{W_k} \sum_{v \in Nb(A)} \frac{\mathbf{h}_v^{(k-1)}}{|Nb(A)|} + \mathbf{B_k} \mathbf{h}_A^{(k-1)}\right)$$

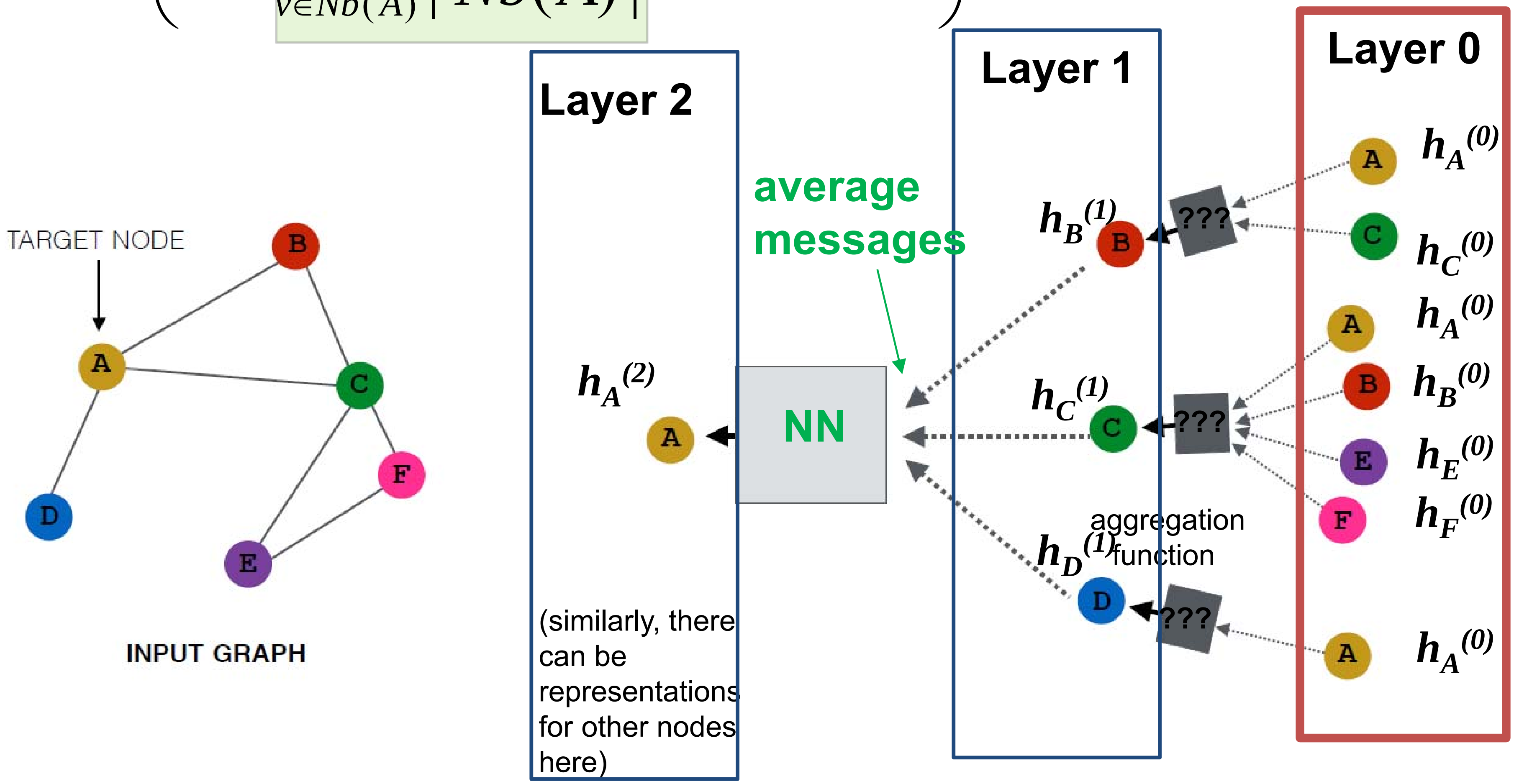


Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

Updates :

average messages

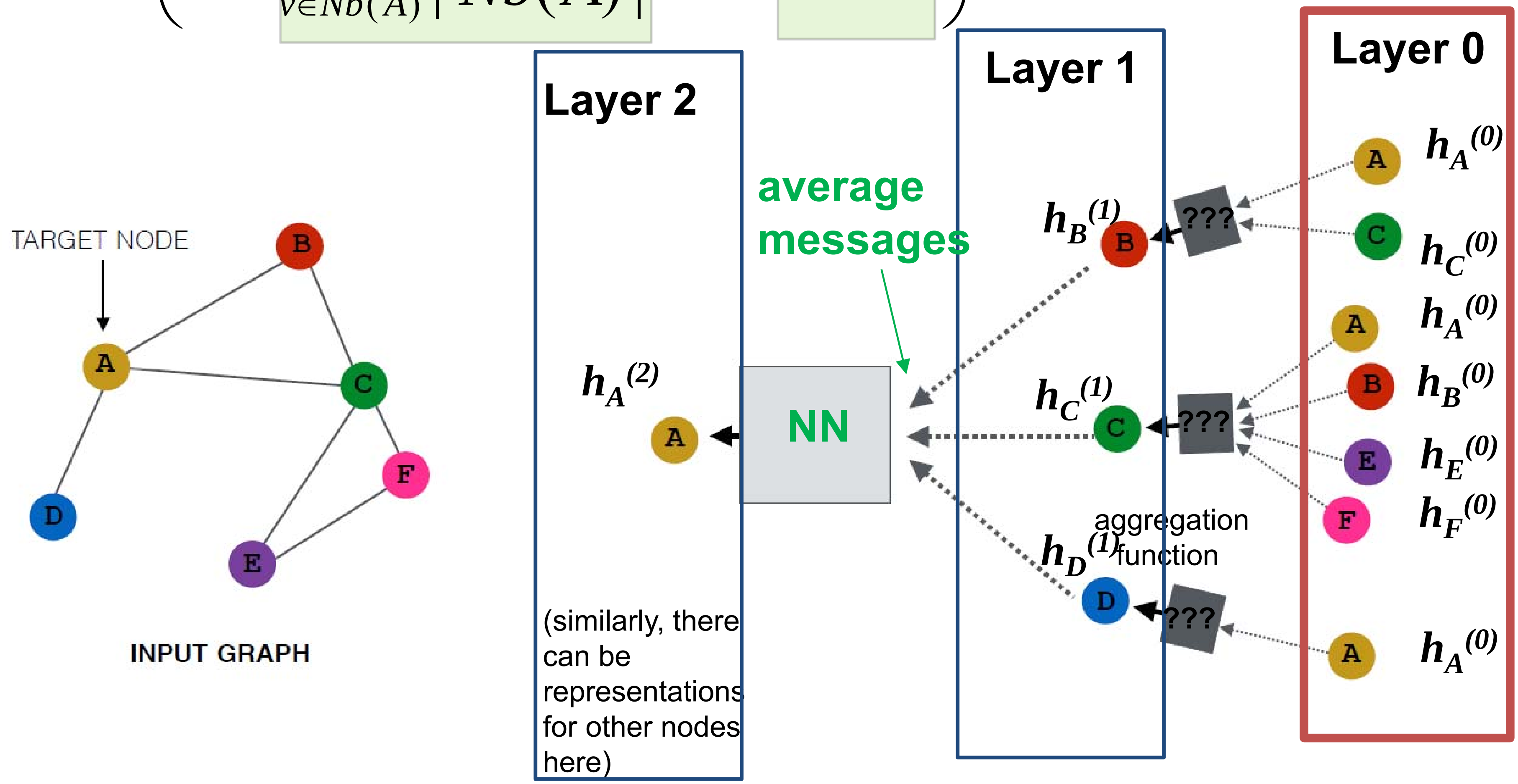
$$\mathbf{h}_A^{(k)} = g\left(\mathbf{W}_\mathbf{k} \boxed{\sum_{v \in Nb(A)} \frac{\mathbf{h}_v^{(k-1)}}{|Nb(A)|}} + \mathbf{B}_\mathbf{k} \mathbf{h}_A^{(k-1)} \right)$$



Transform
aggregated message
+ previous node representation

Updates :

$$\mathbf{h}_A^{(k)} = g\left(\mathbf{W}_\mathbf{k} \sum_{v \in Nb(A)} \frac{\mathbf{h}_v^{(k-1)}}{|Nb(A)|} + \mathbf{B}_\mathbf{k} \mathbf{h}_A^{(k-1)} \right)$$



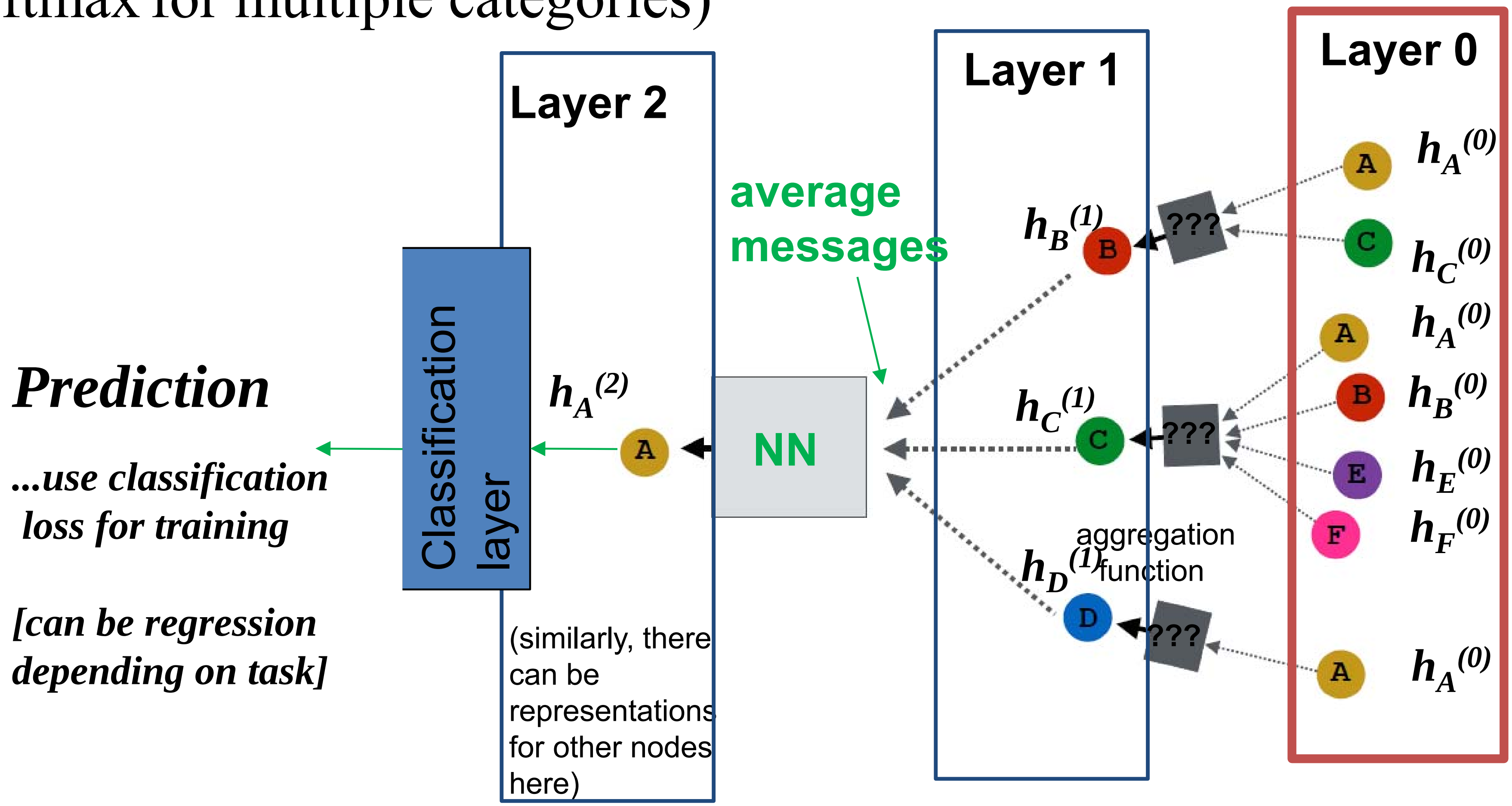
Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

How to train?

$Classification\ (binary):$

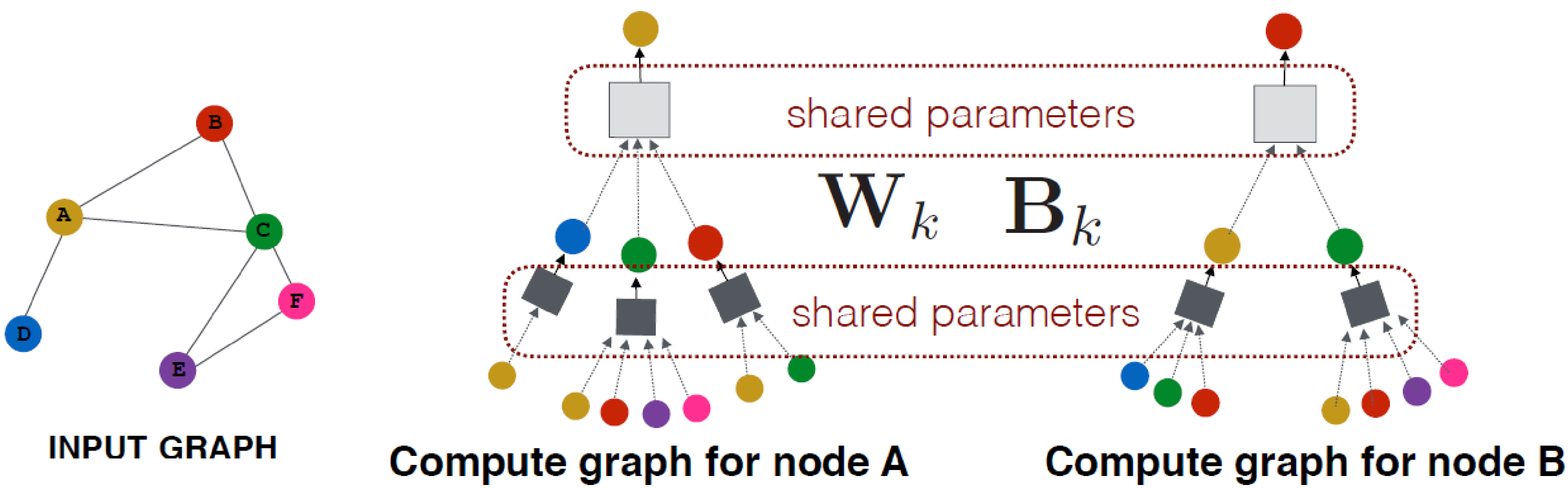
$$P(C=1\,|\,\mathbf{h}_A^{(k)})=\sigma\left(\mathbf{W}_C\mathbf{h}_A^{(k)}\right)$$

$(\ldots\text{or softmax for multiple categories})$



Important for generalization to new graphs

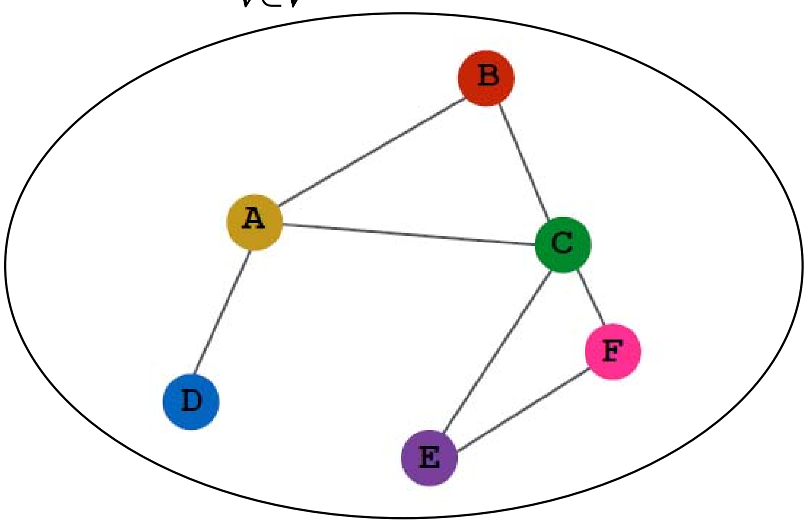
Aggregation function should be the same for all nodes in the same layer (otherwise it would be specific to each particular node)



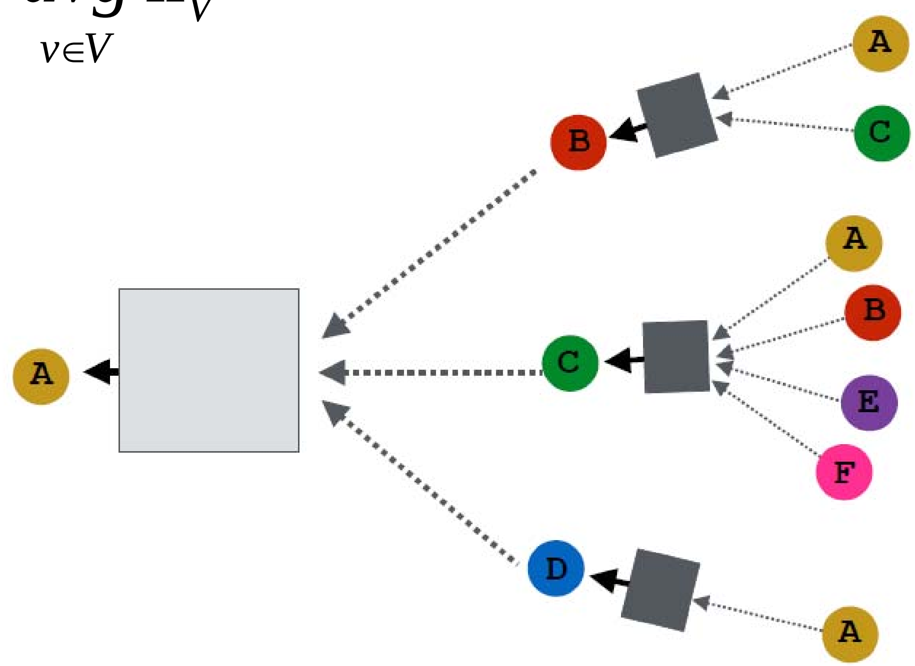
(Whole) Graph Classification

A graph representation can be extracted by pooling all node representations ... then can be passed through a classifier

$$\mathbf{h}_{graph} = \sum_{v \in V} \mathbf{h}_V^{(k)} \quad or \quad \mathbf{h}_{graph} = \underset{v \in V}{avg}\, \mathbf{h}_V^{(k)}$$



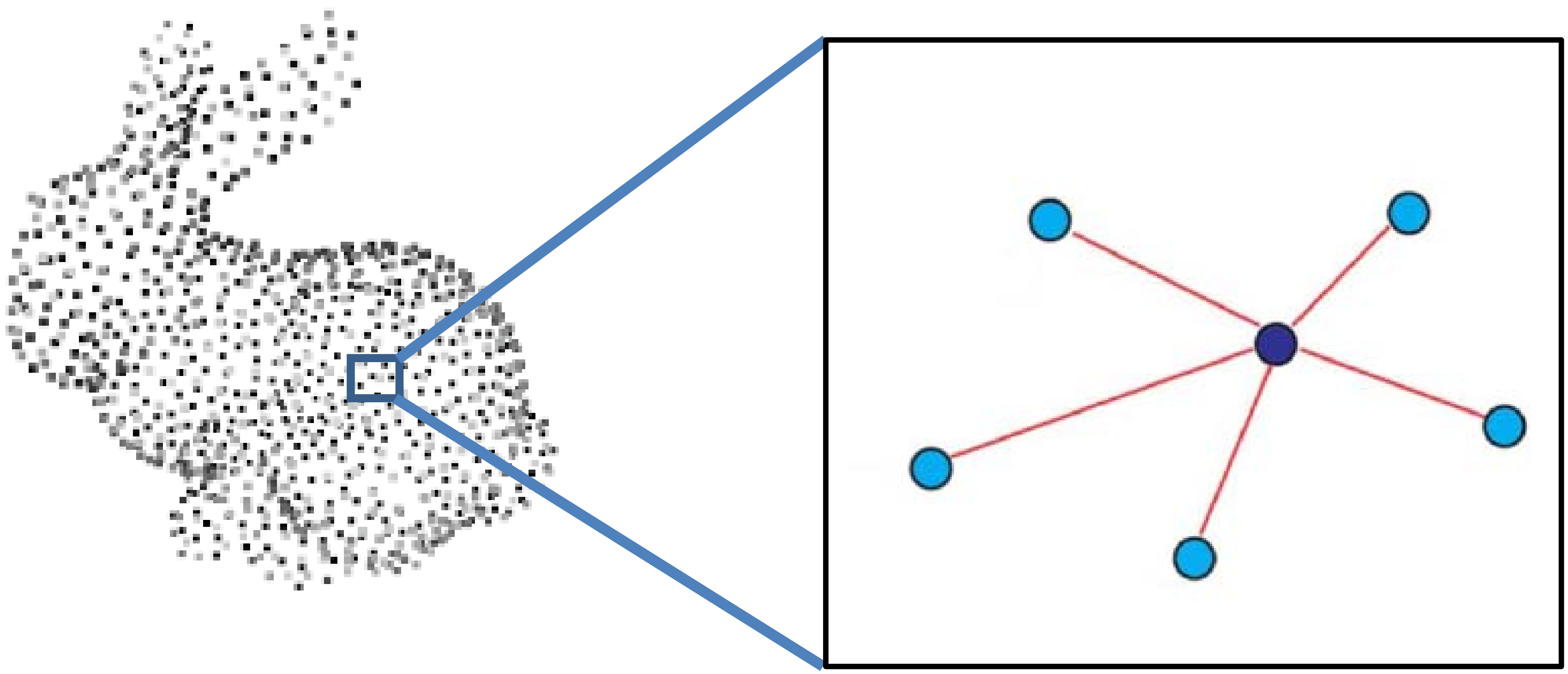
INPUT GRAPH



Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

DGCNN - first layer

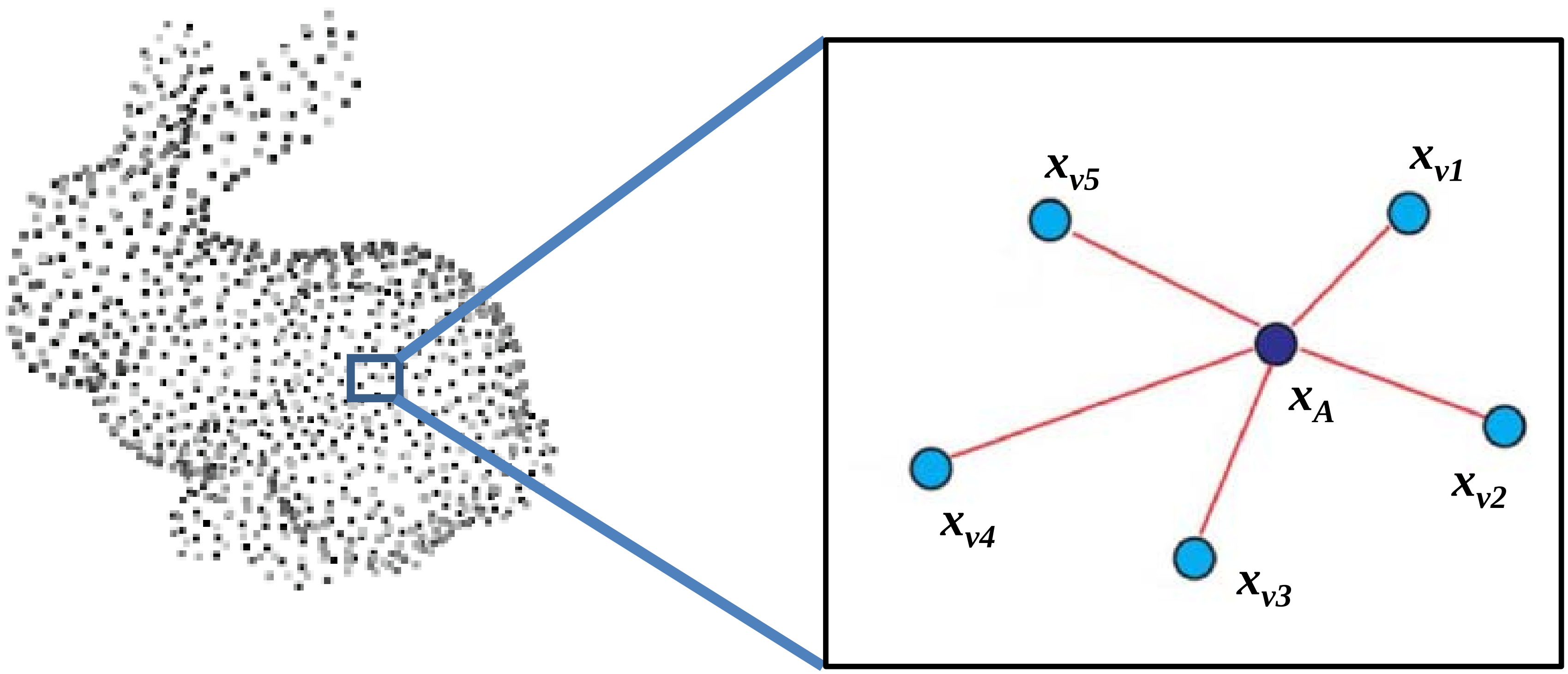
First, connect each point to its K-nearest neighbors



Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - first layer

Each point has an input raw representation $\boldsymbol{x_i}$
(3D point position)

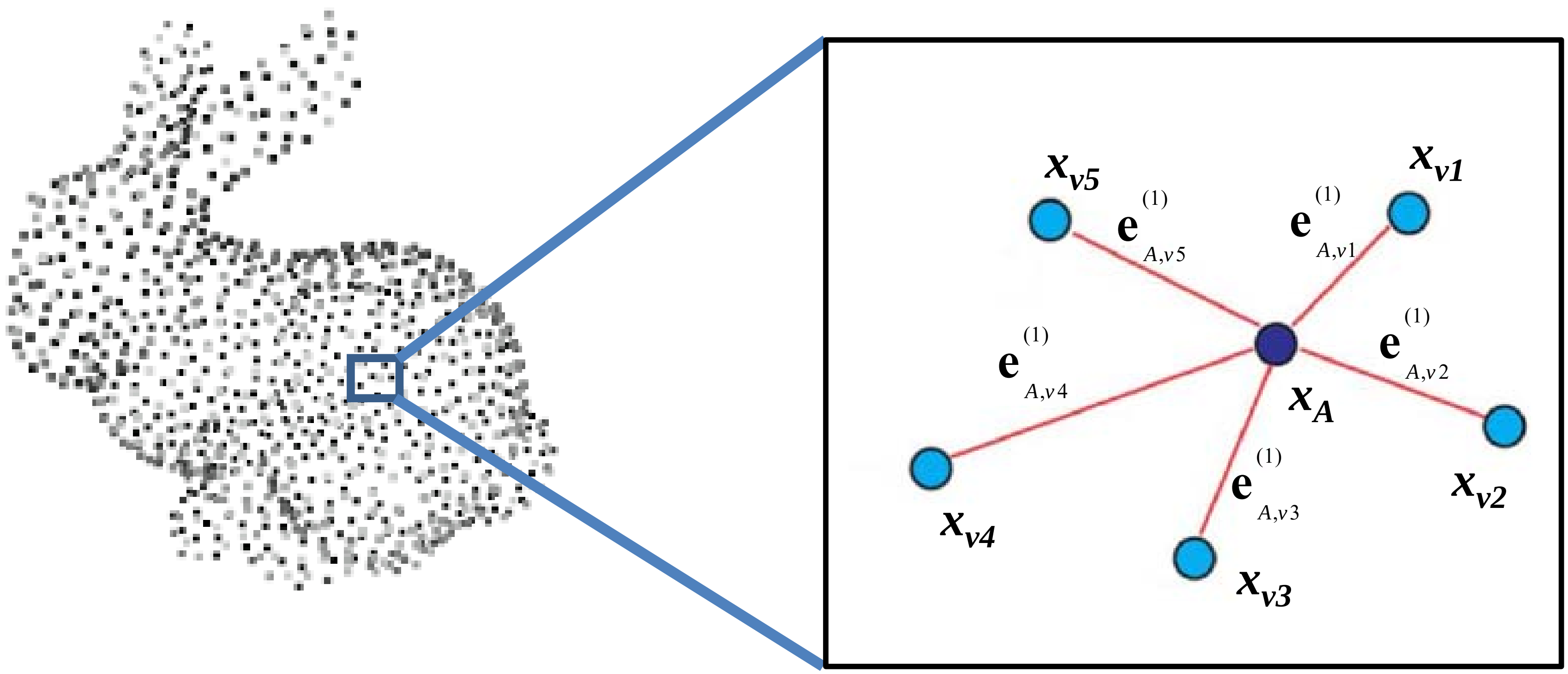


Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - first layer

Compute for each edge a feature representation (EdgeConv):

$$\mathbf{e}_{A,v}^{(1)} = \mathrm{ReLU}(\ MLP(\mathbf{x}_A, \mathbf{x}_v - \mathbf{x}_A)\ )$$

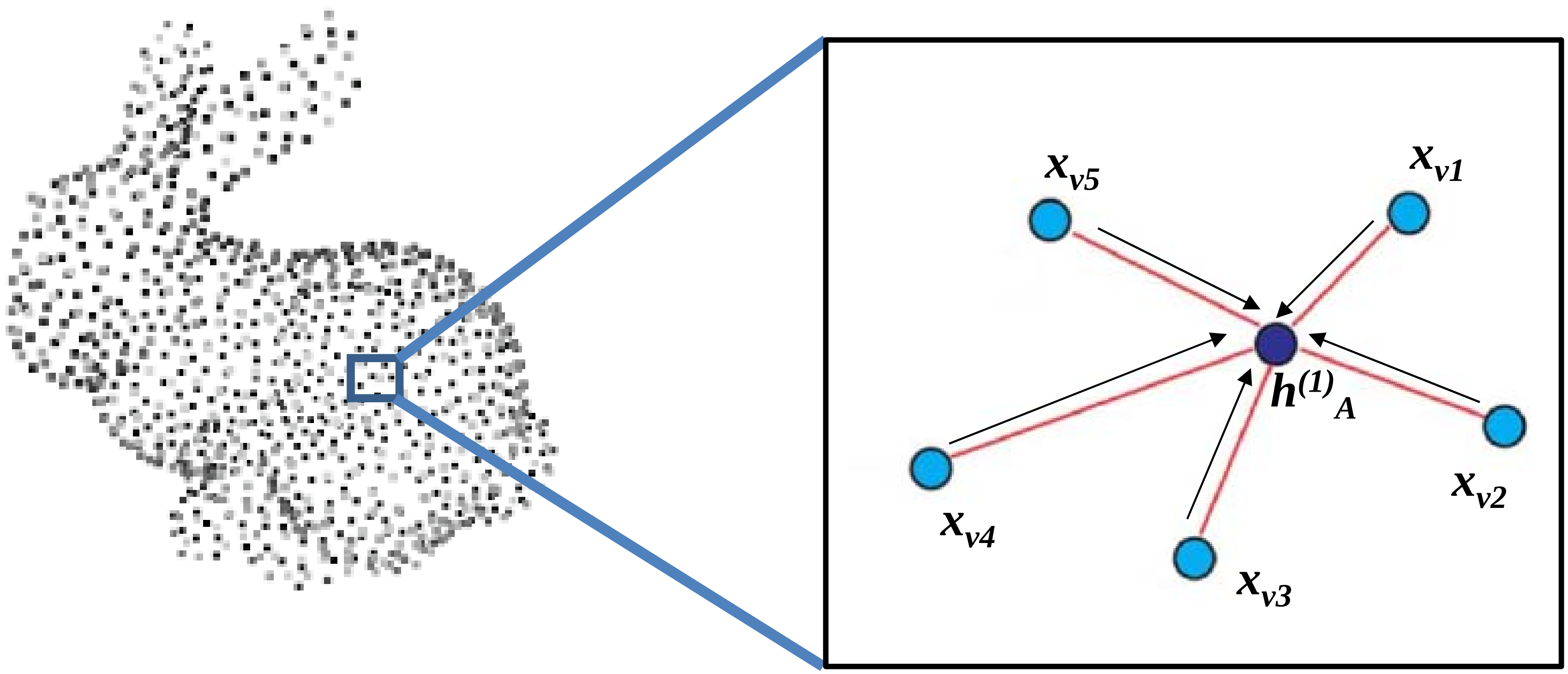


Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - first layer

Aggregate edge representations using max pooling on edges

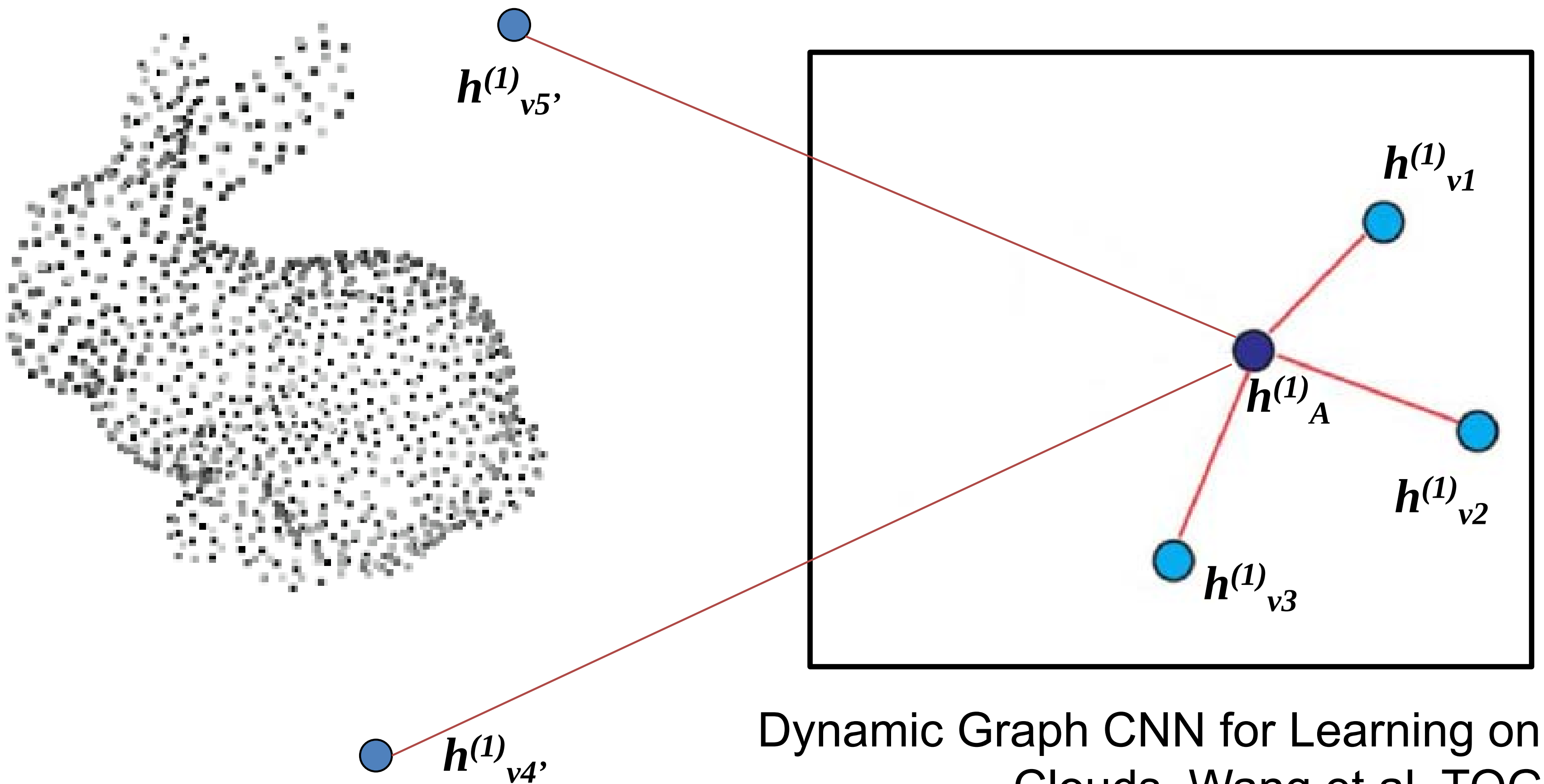
$$\mathbf{h}_A^{(1)} = \max_{v \in Nb(A)} \mathbf{e}_{A,v}^{(1)}$$



Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - second layer

First, connect each point to its K-nearest neighbors based on the node **feature representation of the first layer**
[this means new neighbors for each node]

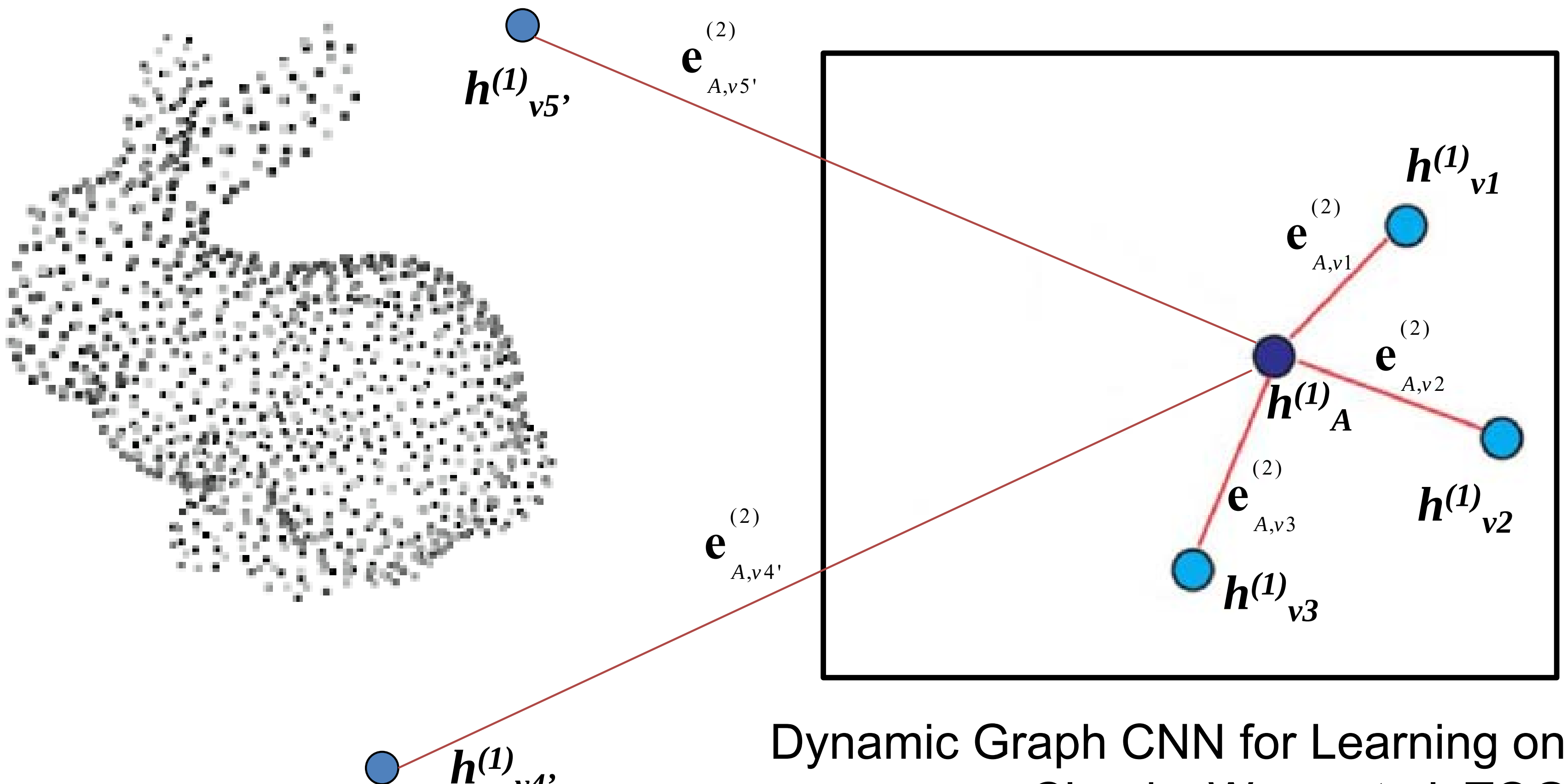


Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - second layer

Compute for each edge a feature representation (EdgeConv):

$$\mathbf{e}_{A,v}^{(2)} = \mathrm{ReLU}(\ MLP(\mathbf{h}_A^{(1)}, \mathbf{h}_v^{(1)} - \mathbf{h}_A^{(1)})\ )$$

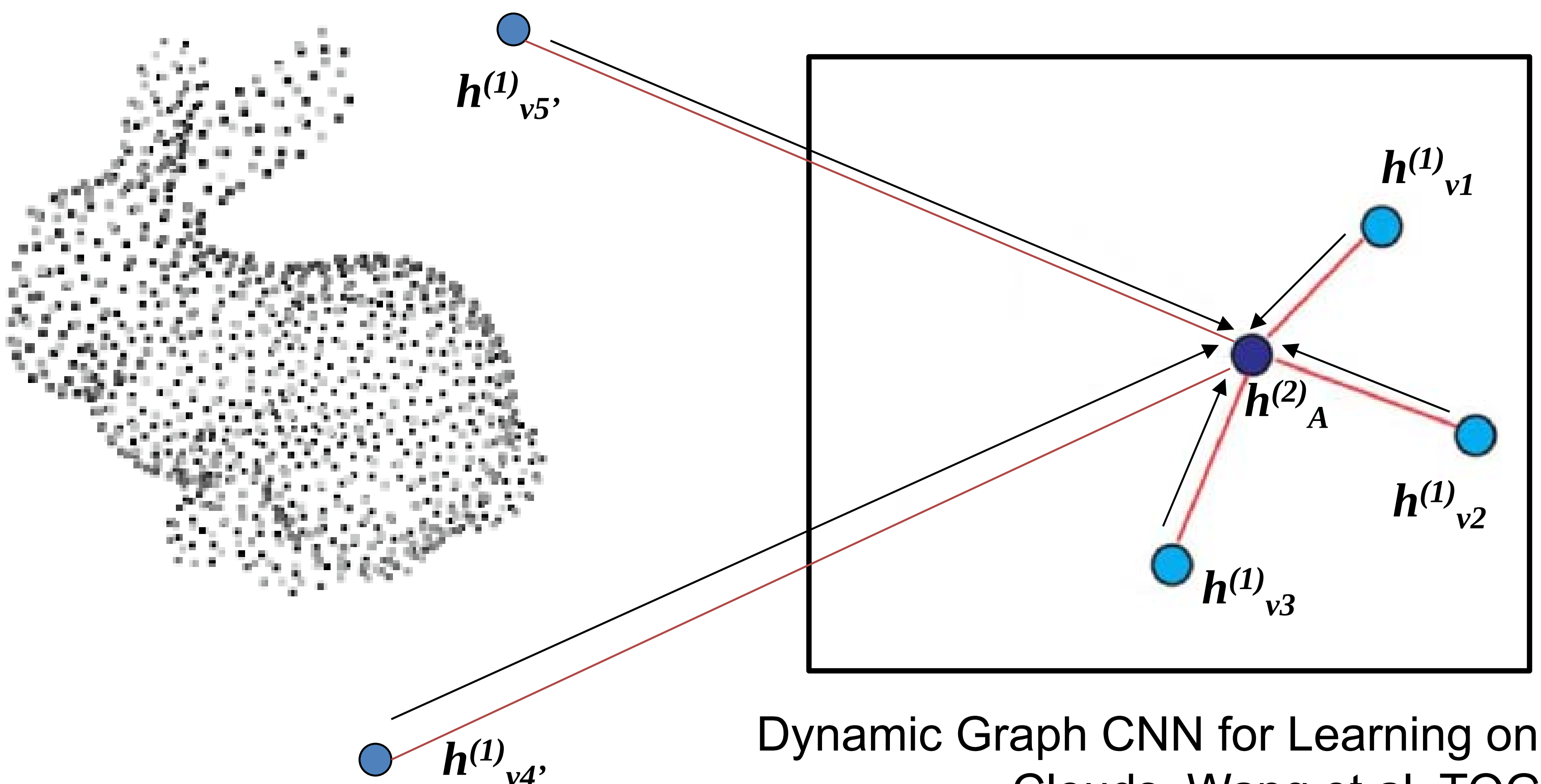


Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN - second layer

Aggregate edge representations using max pooling on edges

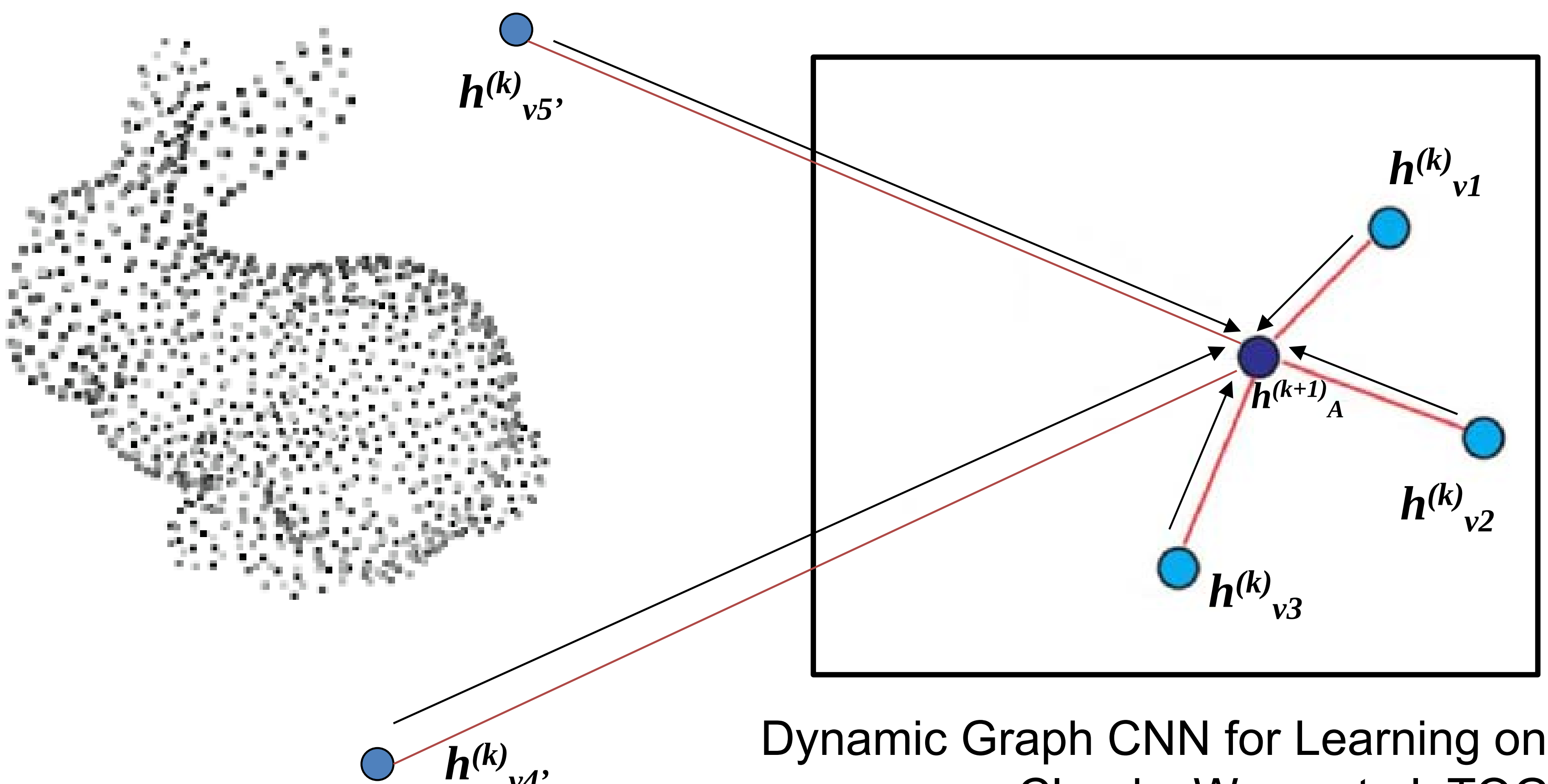
$$\mathbf{h}_A^{(2)} = \max_{v \in Nb(A)} \mathbf{e}_{A,v}^{(2)}$$



Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

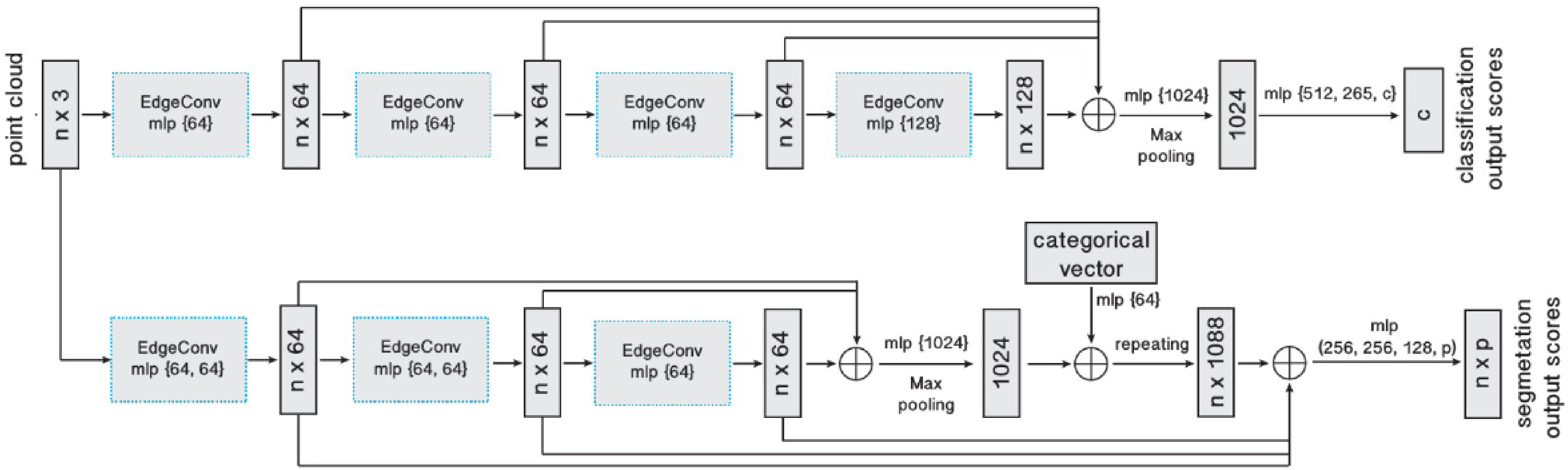
DGCNN - next layers

Same procedure for updating nearest neighbors in feature space, computing edge representations, then max pooling...



Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

DGCNN architecture



Dynamic Graph CNN for Learning on Point Clouds, Wang et al, TOG 2019

Classification results

	Mean Class Accuracy	Overall Accuracy
3DShapeNets [Wu et al. 2015]	77.3	84.7
VoxNet [Maturana and Scherer 2015]	83.0	85.9
Subvolume [Qi et al. 2016]	86.0	89.2
VRN (single view) [Brock et al. 2016]	88.98	-
VRN (multiple views) [Brock et al. 2016]	91.33	-
ECC [Simonovsky and Komodakis 2017]	83.2	87.4
PointNet [Qi et al. 2017b]	86.0	89.2
PointNet++ [Qi et al. 2017c]	-	90.7
Kd-net [Klokov and Lempitsky 2017]	-	90.6
PointCNN [Li et al. 2018a]	88.1	92.2
PCNN [Atzmon et al. 2018]	-	92.3
Ours (baseline)	88.9	91.7
Ours	**90.2**	**92.9**
Ours (2048 points)	**90.7**	**93.5**

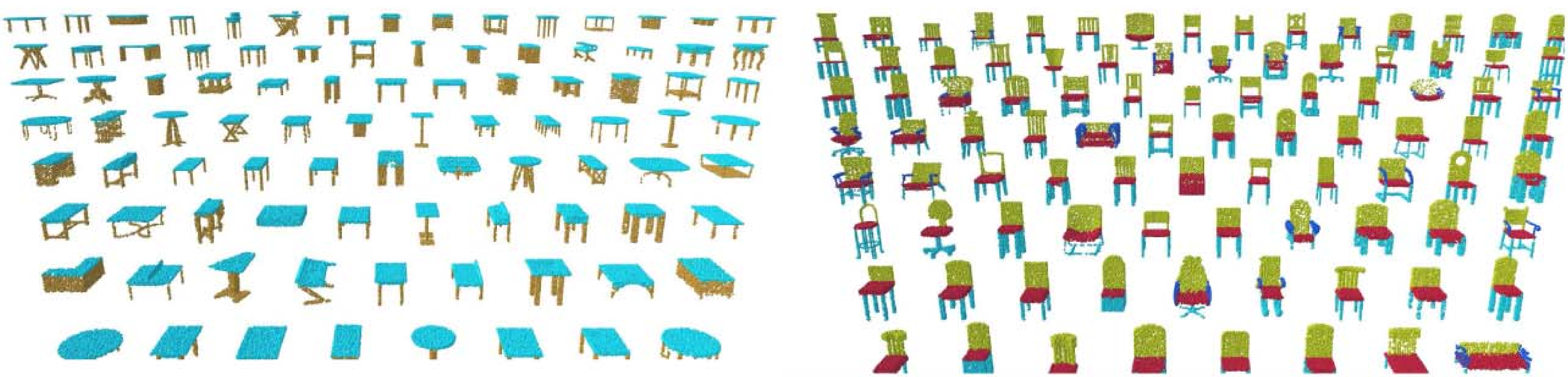
Table 2. Classification results on ModelNet40.

Segmentation results

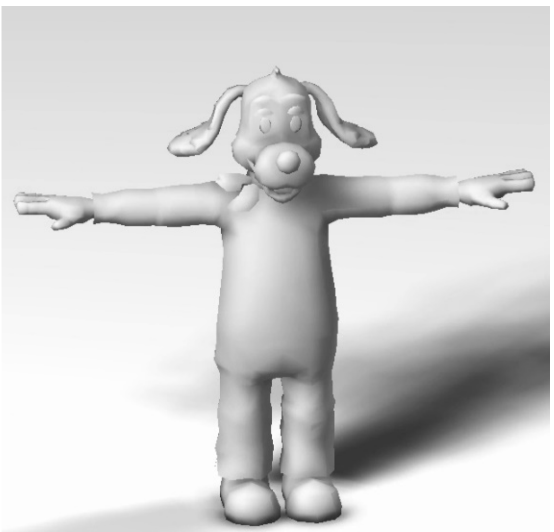
See also: *DeepGCNs: Can GCNs Go as Deep as CNNs?*
https://sites.google.com/view/deep-gcns

	MEAN	AERO	BAG	CAP	CAR	CHAIR	EAR PHONE	GUITAR	KNIFE	LAMP	LAPTOP	MOTOR	MUG	PISTOL	ROCKET	SKATE BOARD	TABLE
# SHAPES		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
PointNet	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PointNet++	85.1	82.4	79.0	**87.7**	77.3	**90.8**	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
Kd-Net	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
LocalFeatureNet	84.3	**86.1**	73.0	54.9	77.4	88.8	55.0	90.6	86.5	75.2	**96.1**	57.3	91.7	83.1	53.9	72.5	**83.8**
PCNN	85.1	82.4	80.1	85.5	79.5	**90.8**	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
PointCNN	**86.1**	84.1	**86.45**	86.0	**80.8**	90.6	**79.7**	**92.3**	**88.4**	**85.3**	**96.1**	**77.2**	**95.3**	**84.2**	**64.2**	**80.0**	83.0
Ours	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6

Table 6. Part segmentation results on ShapeNet part dataset. Metric is mIoU(%) on points.

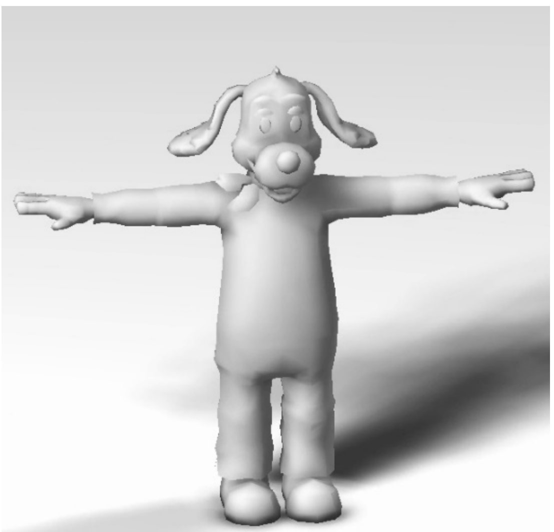


A mesh GNN for Character Rigging

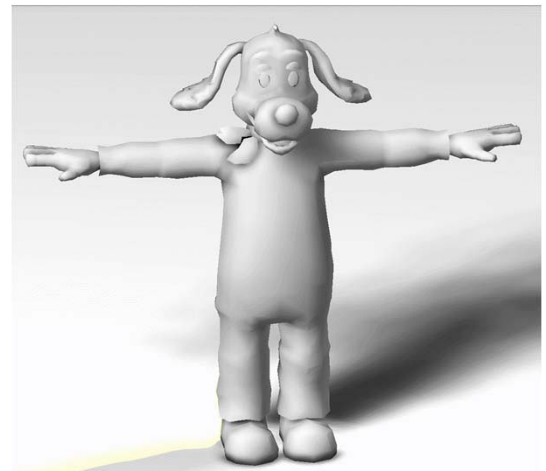


Input 3D model

Goal: Automatic Rigging

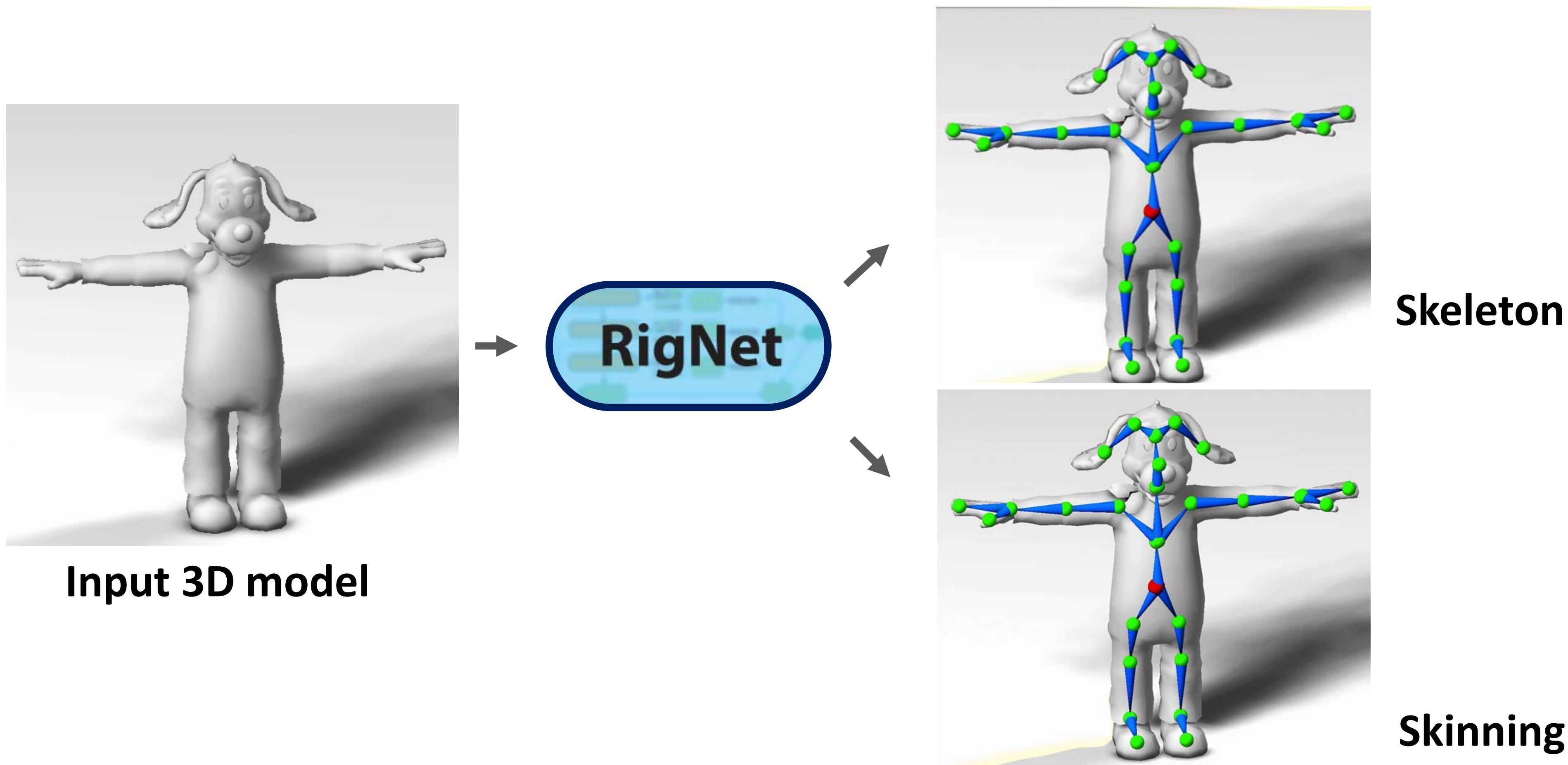


Input 3D model

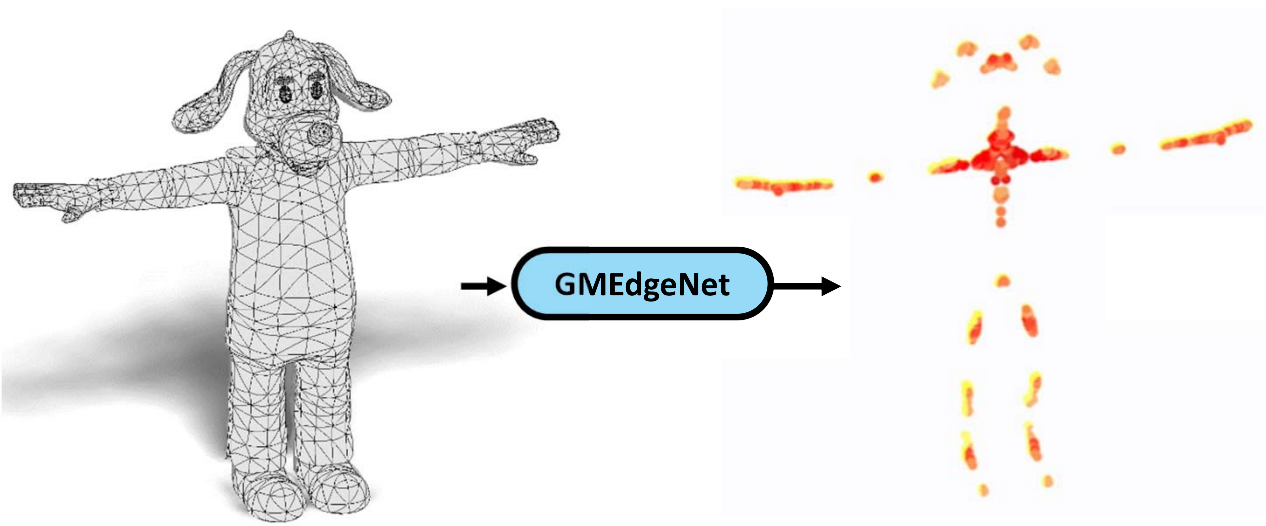


Skeleton

Goal: Automatic Rigging



Joint Prediction Module

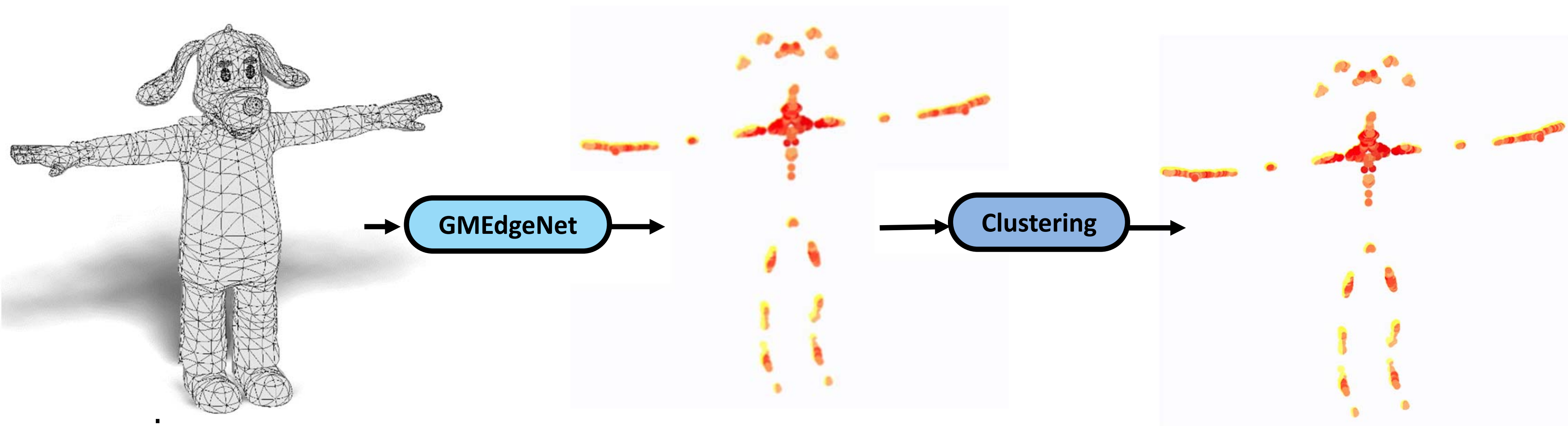


Input mesh M

perturbed vertices

$\mathbf{q} = \mathbf{v} + f_d(M)$

Joint Prediction Module

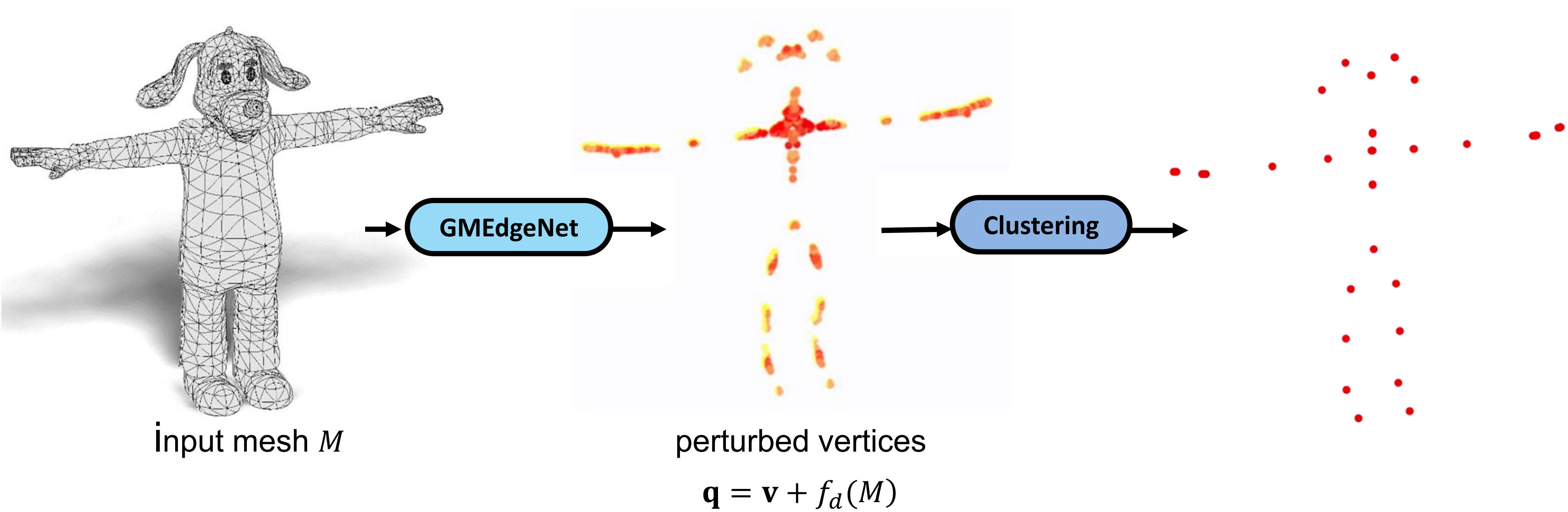


input mesh M

perturbed vertices

$\mathbf{q} = \mathbf{v} + f_d(M)$

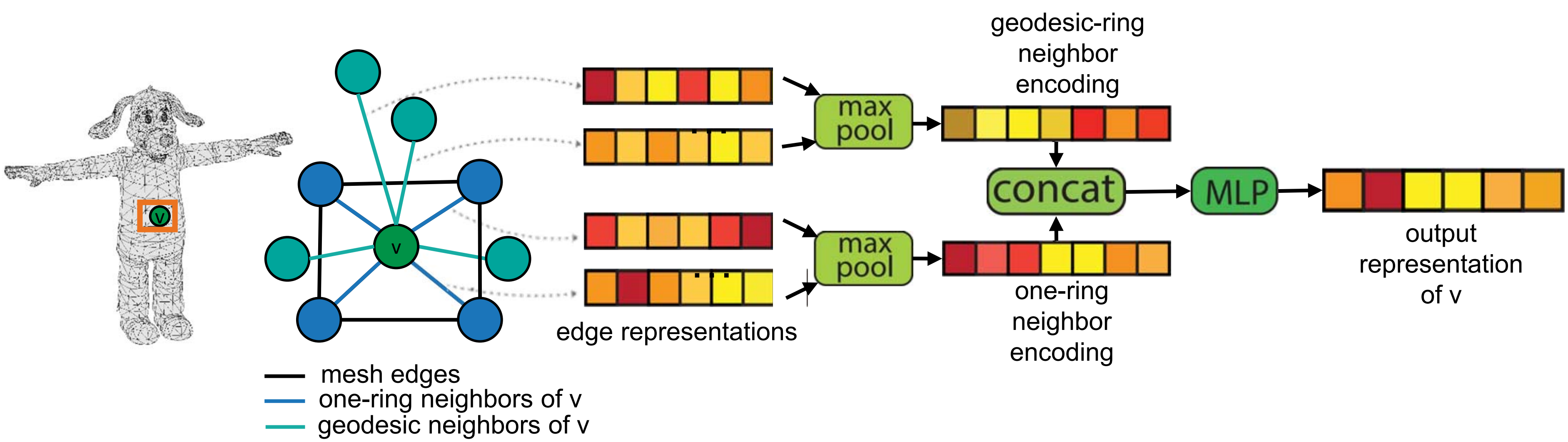
Joint Prediction Module



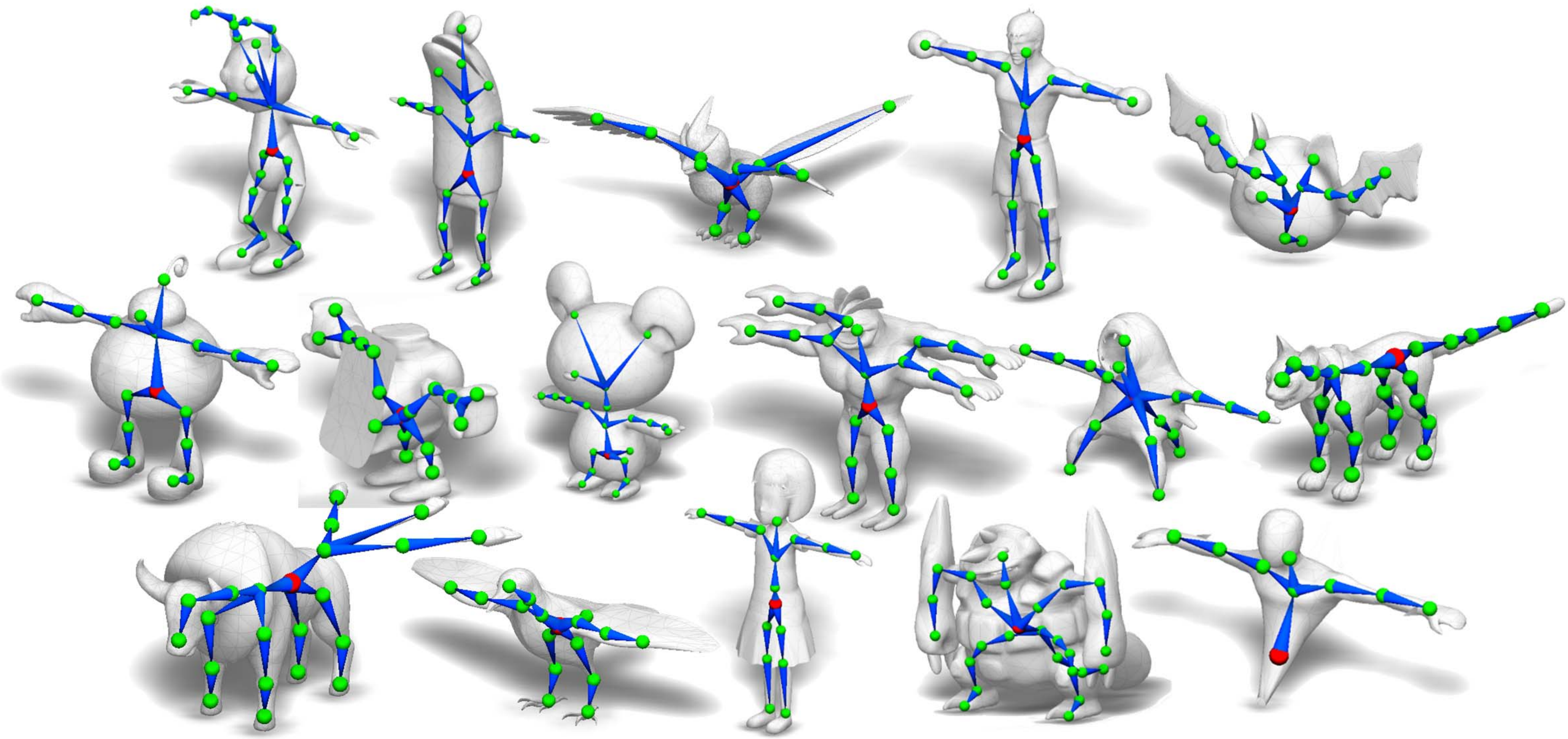
GMEdgeNet graph convolution

Edge features computed as in DGCNN:

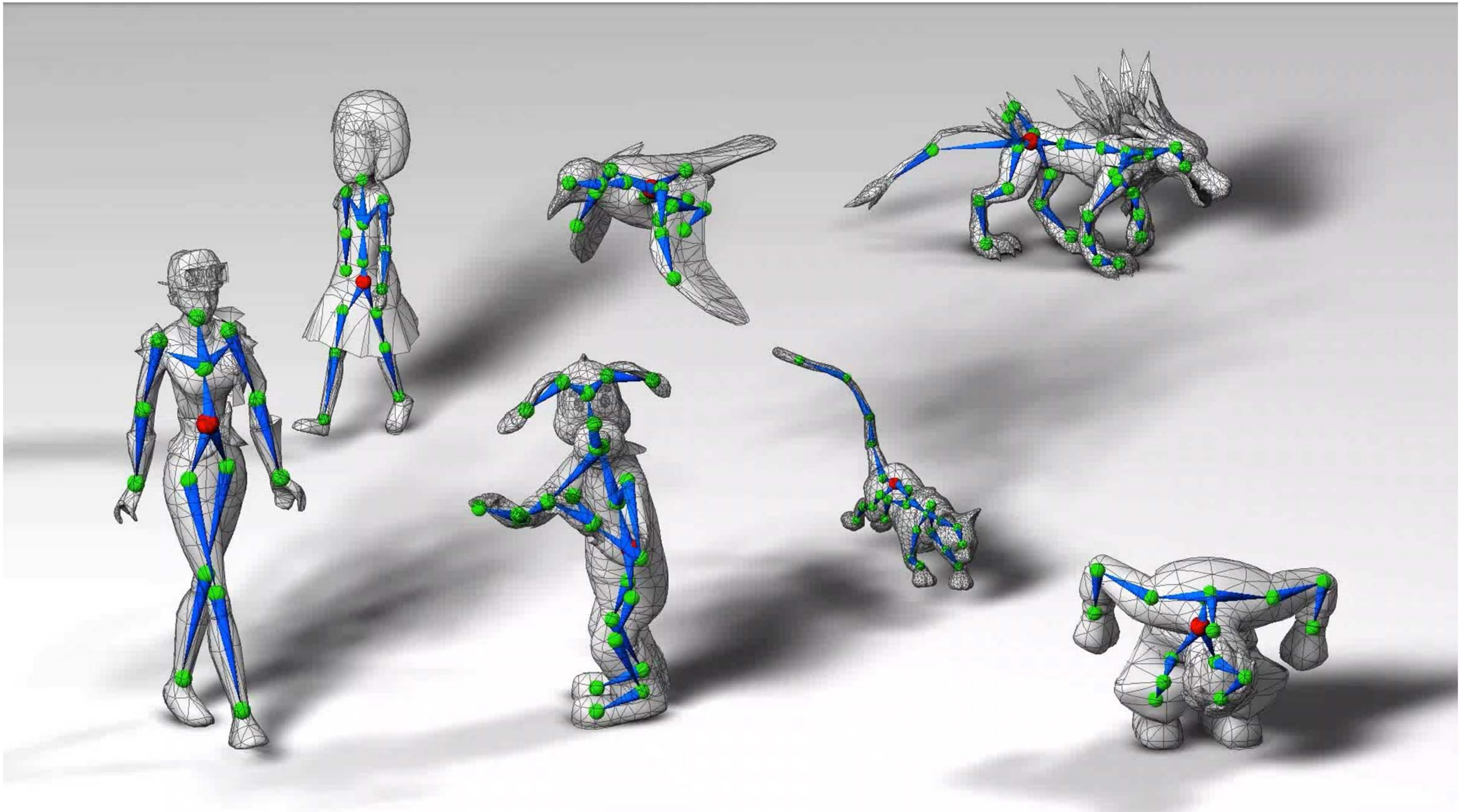
$$\mathbf{e} = \mathrm{ReLU}(\ MLP(\mathbf{x}_{vertex}, \mathbf{x}_{vertex} - \mathbf{x}_{neighbor})\ )$$



Qualitative Results



Qualitative Results



Graph-based 3D Deep Learning Advantages

- **Well-suited to analyze graphs**
  (graphs: meshes, shape and scene abstractions based on their objects and parts)

- **Effective in encoding local graph neighborhoods**

Graph-based 3D Deep Learning Disadvantages

- **Robustness to irregular and non-uniform sampling**
  (existing networks are not that robust, require training with augmentation on different graph connectivity)

- **Performance is not that good compared to sparse tensor nets/octrees/transformers for 3D point clouds of scenes/shapes**

- **When graph edges are not available (point clouds), they are derived from ad hoc heuristics**