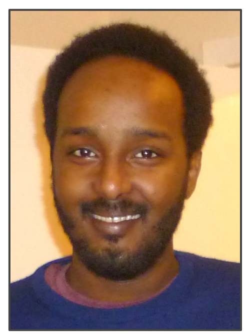
GA-653449

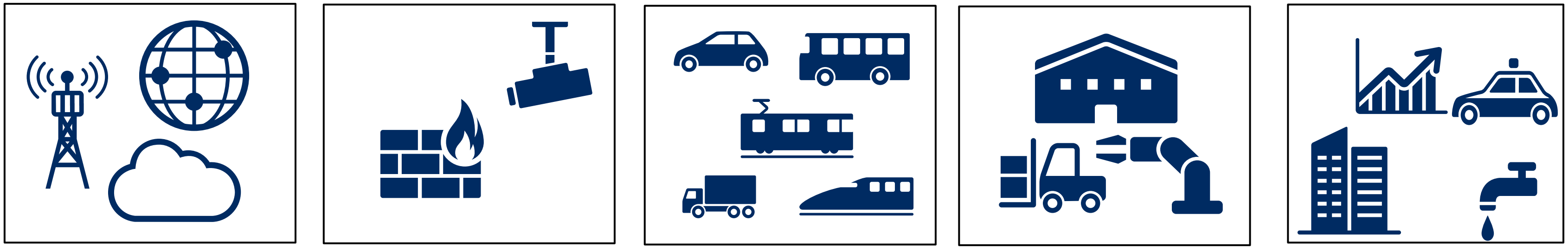
Learning Convolutional Neural Networks for Graphs

Mathias Niepert   Mohamed Ahmed   Konstantin Kutzkov


NEC Laboratories Europe

Representation Learning for Graphs



Telecom | Safety | Transportation | Industry | Smart cities

Intermediate Representation: **Graphs**

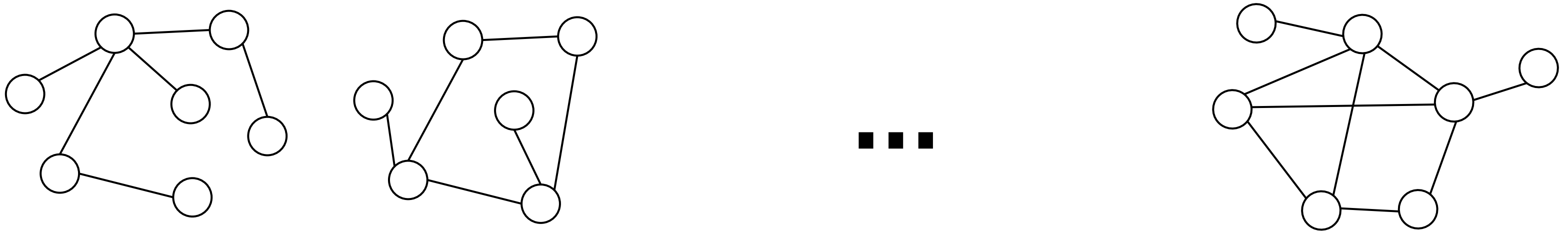
?

Deep Learning System

(Edge) deployment

Problem Definition

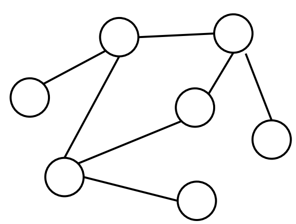
- **Input:** Finite collection of graphs



  - Nodes of any two graphs are ***not*** necessarily in correspondence
  - Nodes and edges may have attributes (discrete and continuous)

- **Problem:** Learn a representation for classification/regression

- **Example:** Graph classification problem

class[] = ?

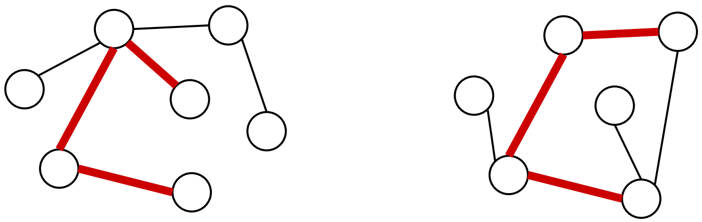
State of the Art: Graph Kernels

Define kernel based on substructures

- Shortest paths
- Random walks
- Subtrees
- ...

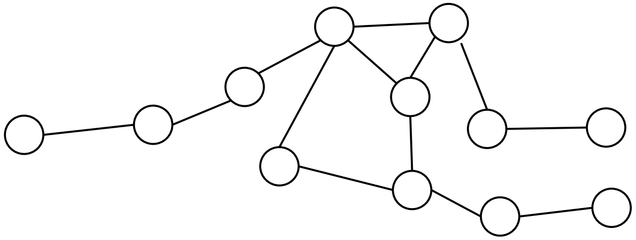
Kernel is similarity function on pairs of graphs

- Count the number of common substructures



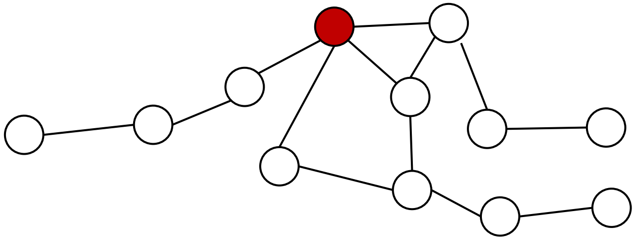
Use graph kernels with SVMs

Patchy: Learning CNNs for Graphs

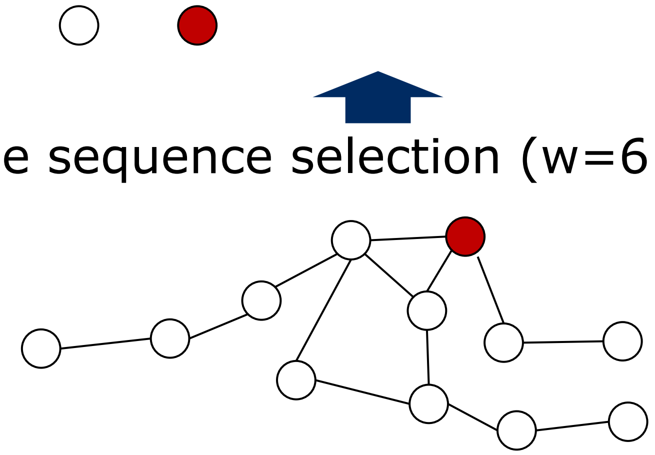


Patchy: Learning CNNs for Graphs

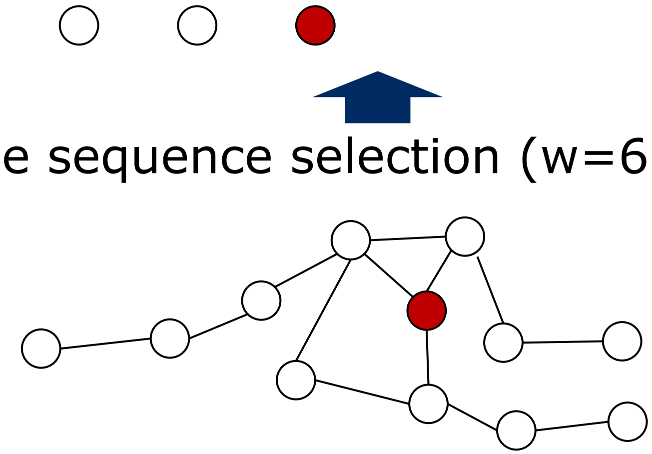
node sequence selection (w=6 nodes)



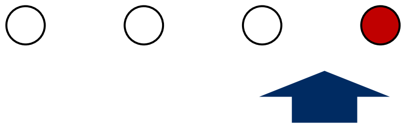
Patchy: Learning CNNs for Graphs



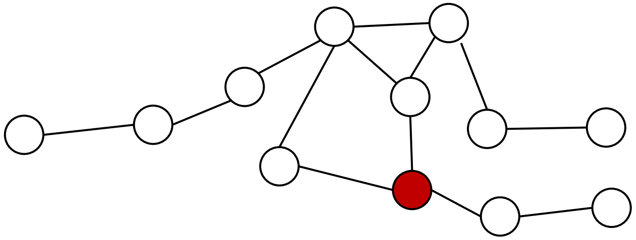
Patchy: Learning CNNs for Graphs



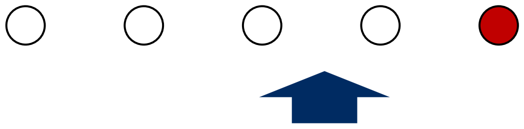
Patchy: Learning CNNs for Graphs



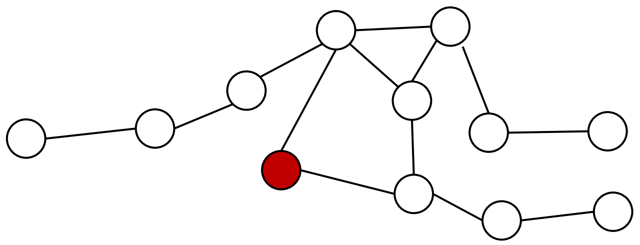
node sequence selection (w=6 nodes)



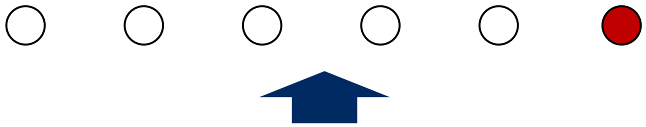
Patchy: Learning CNNs for Graphs



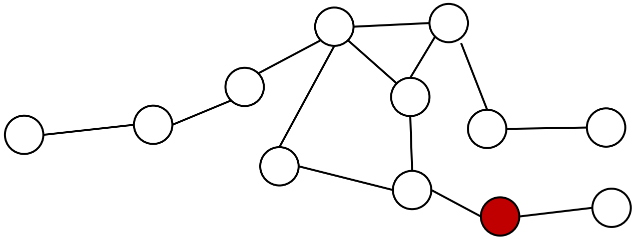
node sequence selection (w=6 nodes)



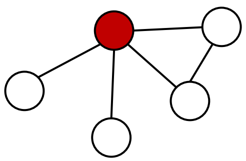
Patchy: Learning CNNs for Graphs



node sequence selection (w=6 nodes)

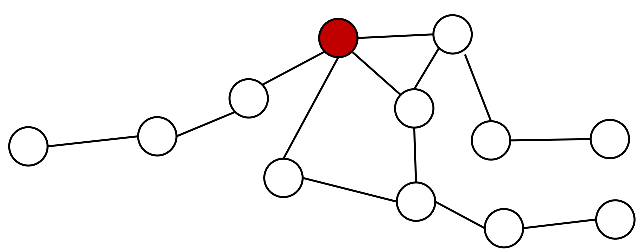


Patchy: Learning CNNs for Graphs

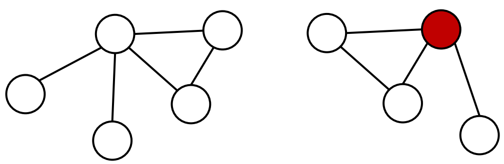


neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)

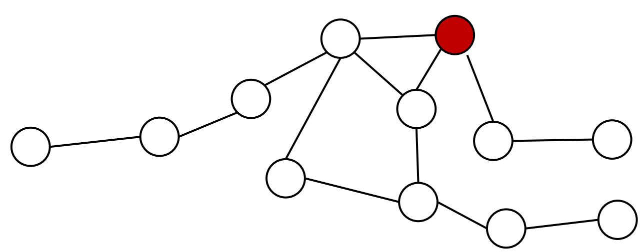


Patchy: Learning CNNs for Graphs

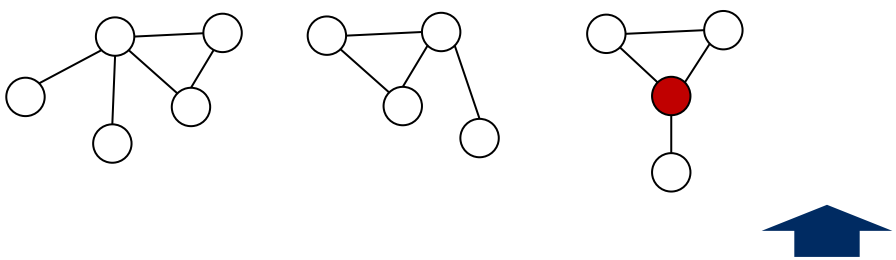


neighborhood assembly (at least k=4 nodes)

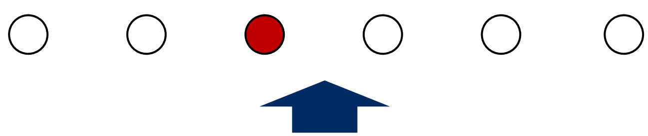
node sequence selection (w=6 nodes)



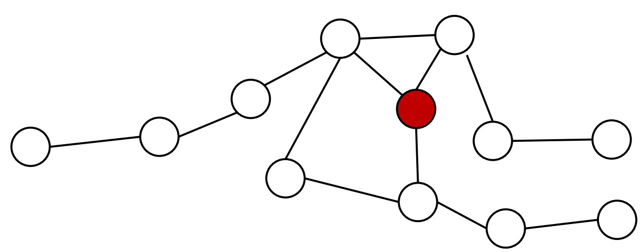
Patchy: Learning CNNs for Graphs



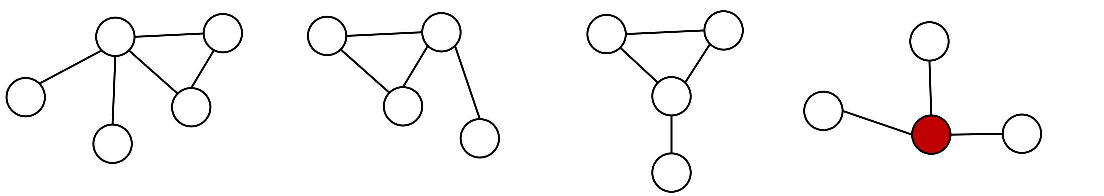
neighborhood assembly (at least k=4 nodes)



node sequence selection (w=6 nodes)

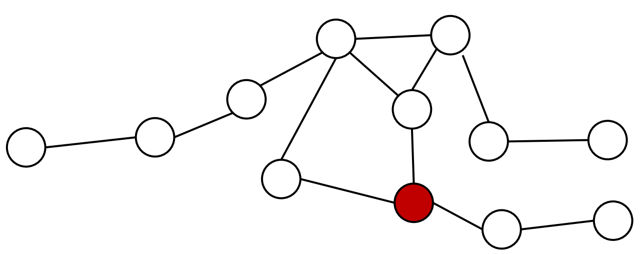


Patchy: Learning CNNs for Graphs

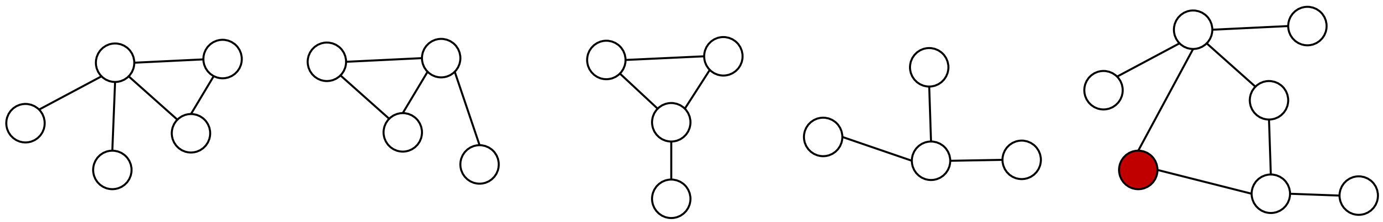


neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)

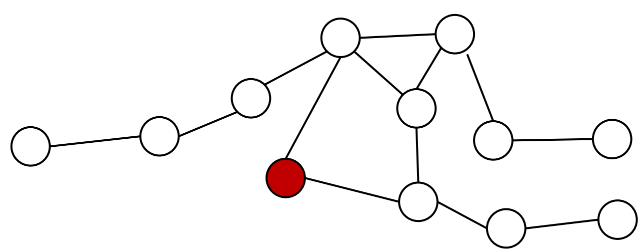


Patchy: Learning CNNs for Graphs

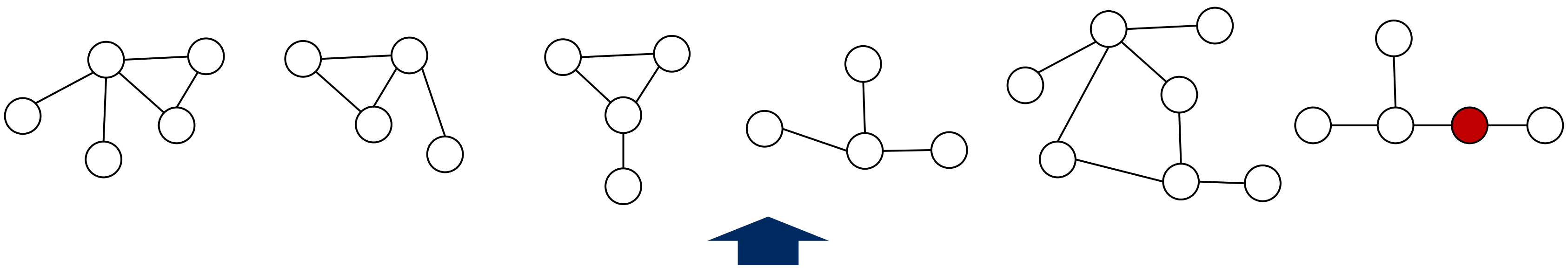


neighborhood assembly (at least k=4 nodes)

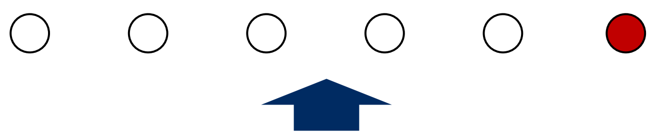
node sequence selection (w=6 nodes)



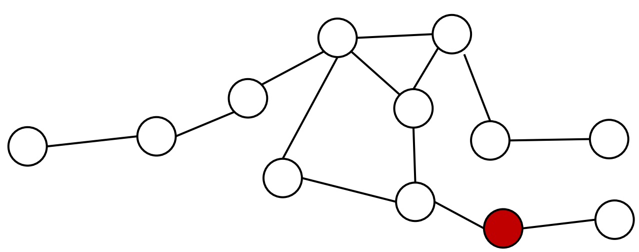
Patchy: Learning CNNs for Graphs



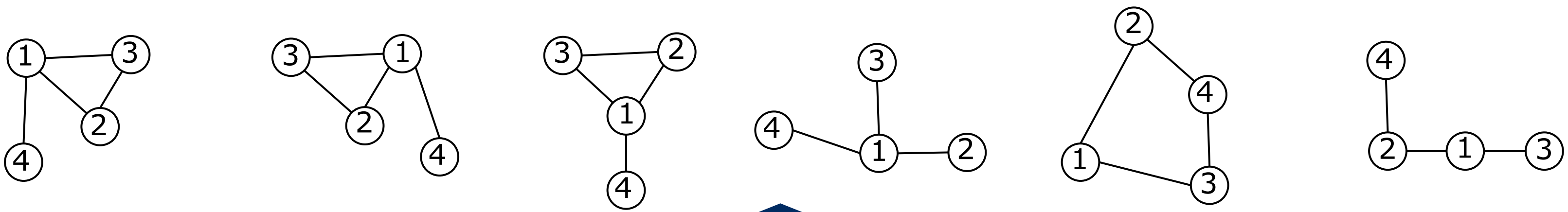
neighborhood assembly (at least k=4 nodes)



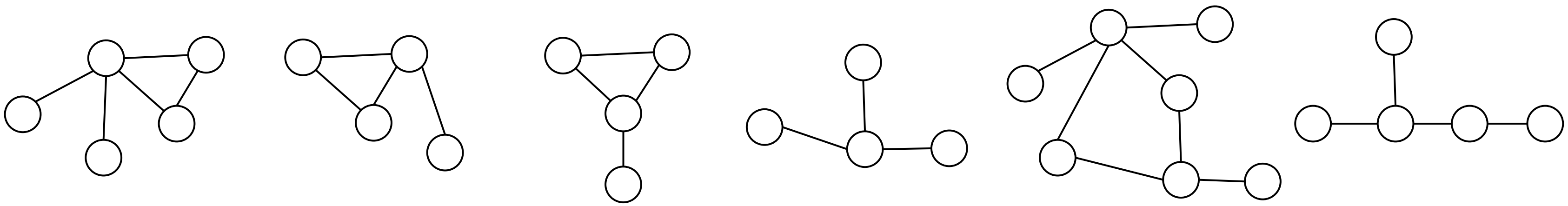
node sequence selection (w=6 nodes)



Patchy: Learning CNNs for Graphs

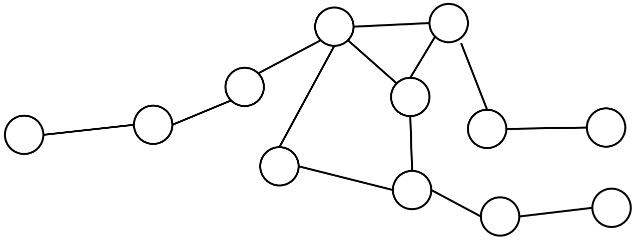


neighborhood normalization (exactly k=4 nodes)



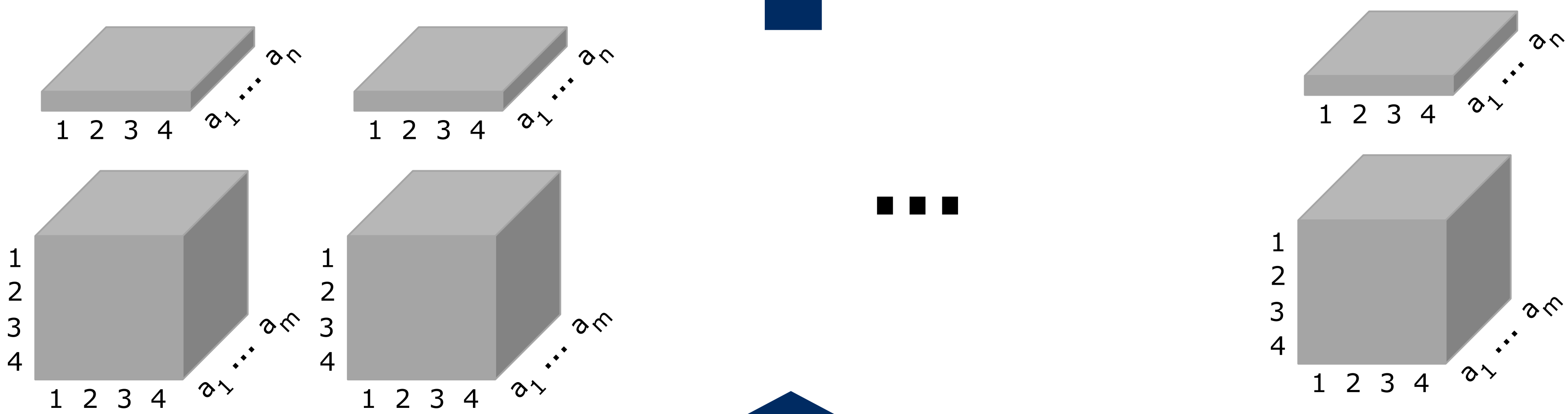
neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)

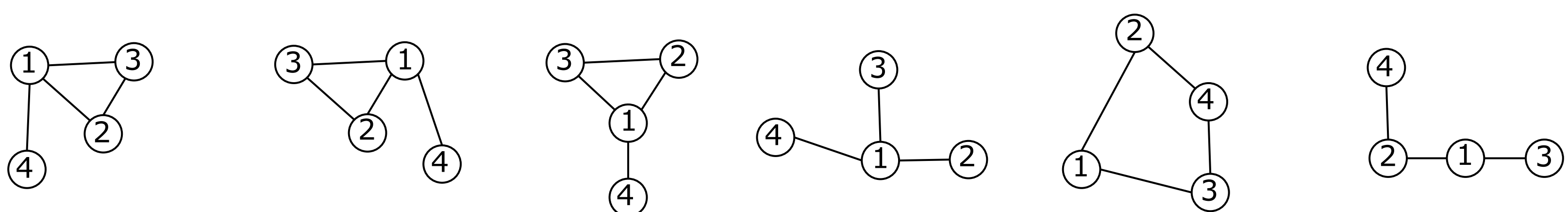


Patchy: Learning CNNs for Graphs

Convolutional architecture



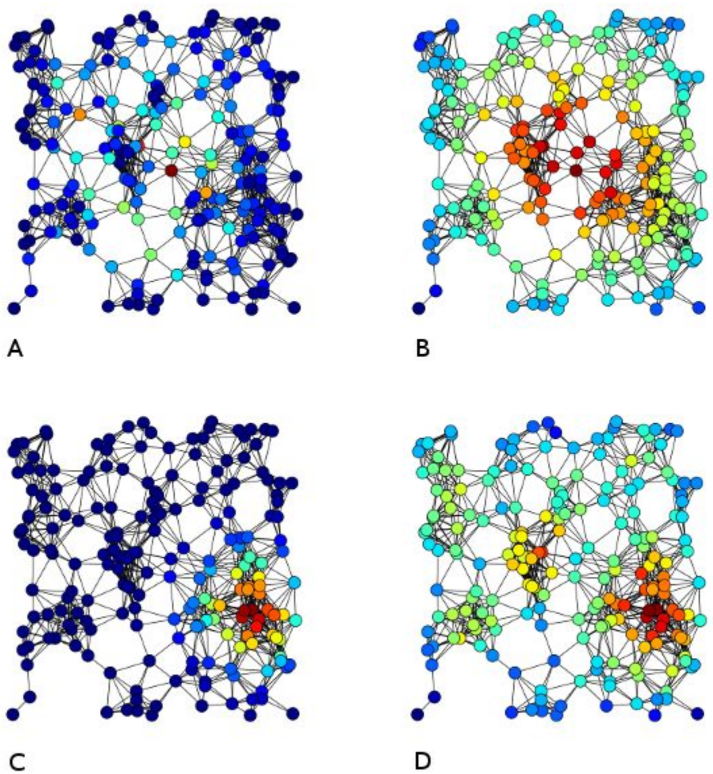
- normalized neighborhoods serve as **receptive fields**
- node and edge attributes correspond to **channels**

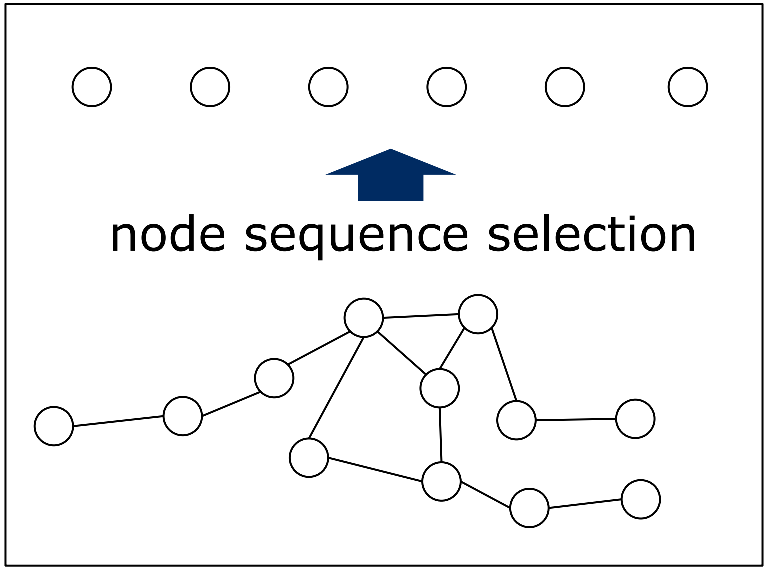


neighborhood normalization

Node Sequence Selection

- We use centrality measures to generate the node sequences
- Nodes with similar structural roles are aligned across graphs

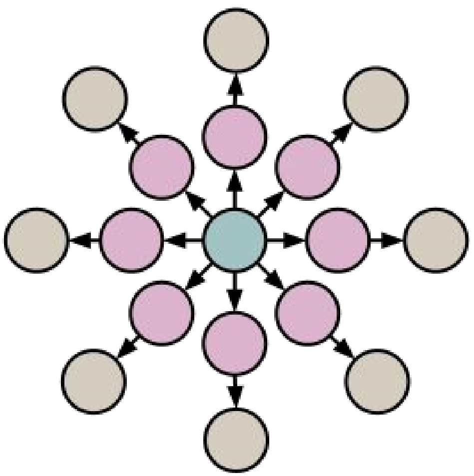


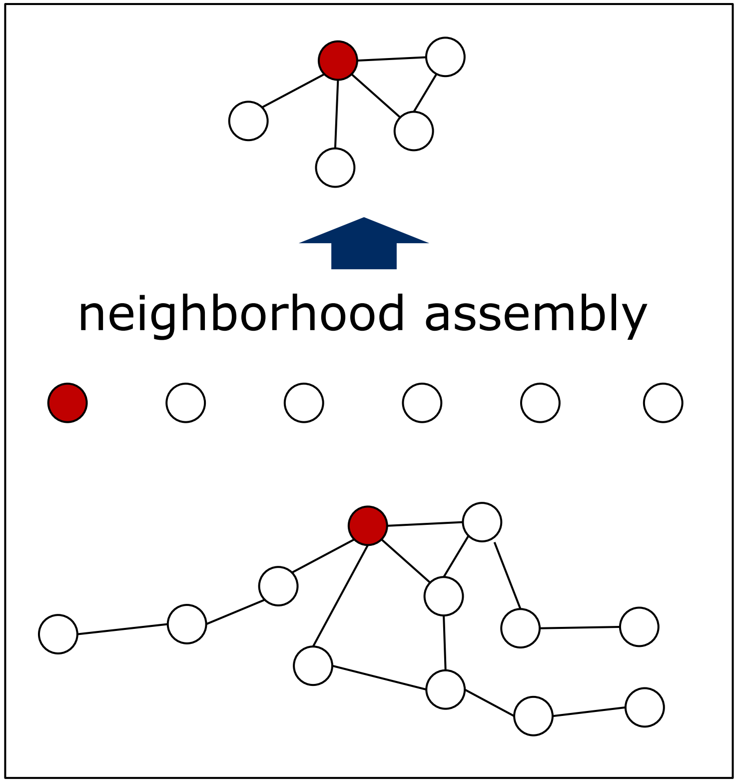


A: Betweenness centrality B: Closeness centrality
C: Eigenvector centrality D: Degree centrality

Neighborhood Assembly

- Simple breadth-first expansion until at least *k* nodes added, or no additional nodes to add





Graph Normalization Problem

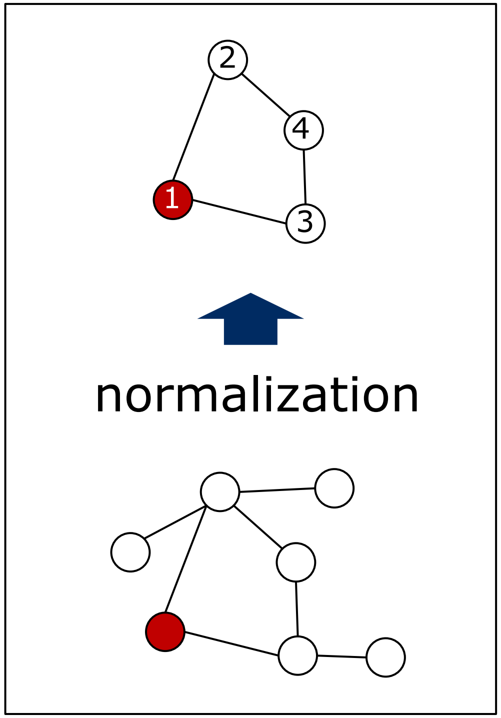
- Nodes of any two graphs should have similar position in the adjacency matrices iff their structural roles are similar

adjacency matrices under labeling

$$\arg\min_{\ell} \mathbb{E}_{\mathcal{G}} \left[\left| \mathbf{d}_{\mathbf{A}} \left(\mathbf{A}^{\ell}(G), \mathbf{A}^{\ell}(G') \right) - \mathbf{d}_{\mathbf{G}}(G, G') \right| \right]$$

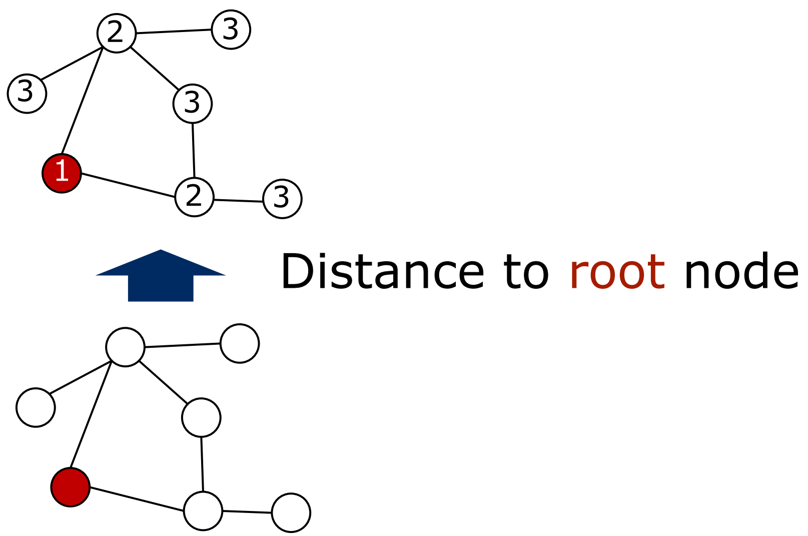
labeling method (centrality etc.)

distance measures in matrix and graph space

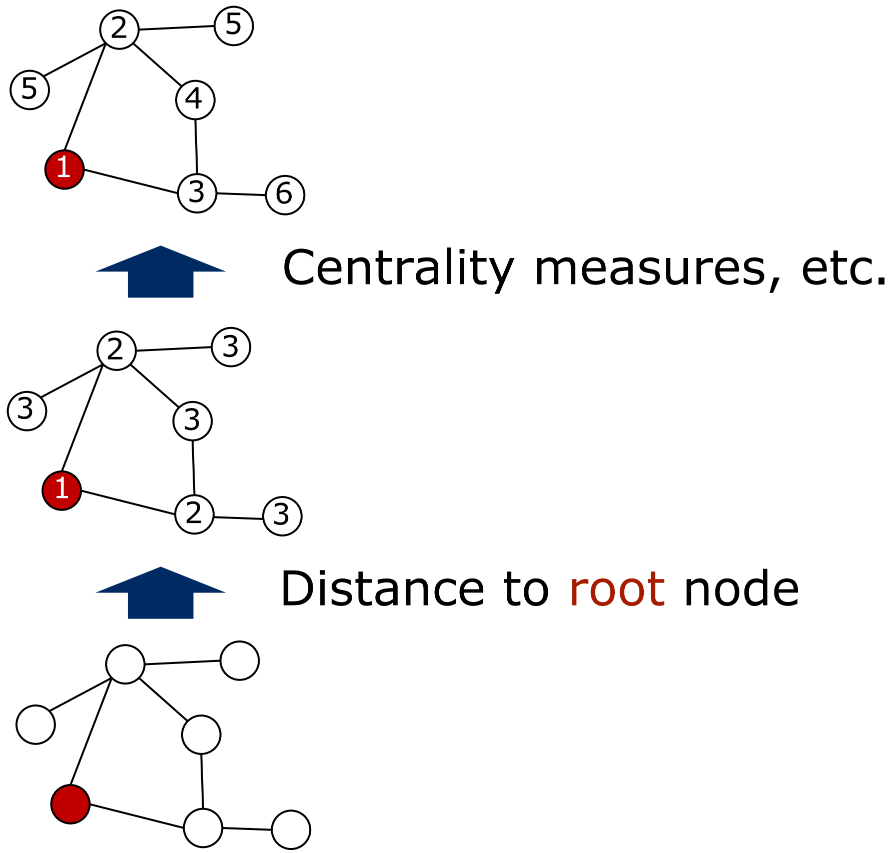


- **Result:** For several distance measure pairs it is possible to efficiently compare labeling methods without supervision
- **Example:** $||\mathbf{A} - \mathbf{A}'||_1$ and edit distance on graphs

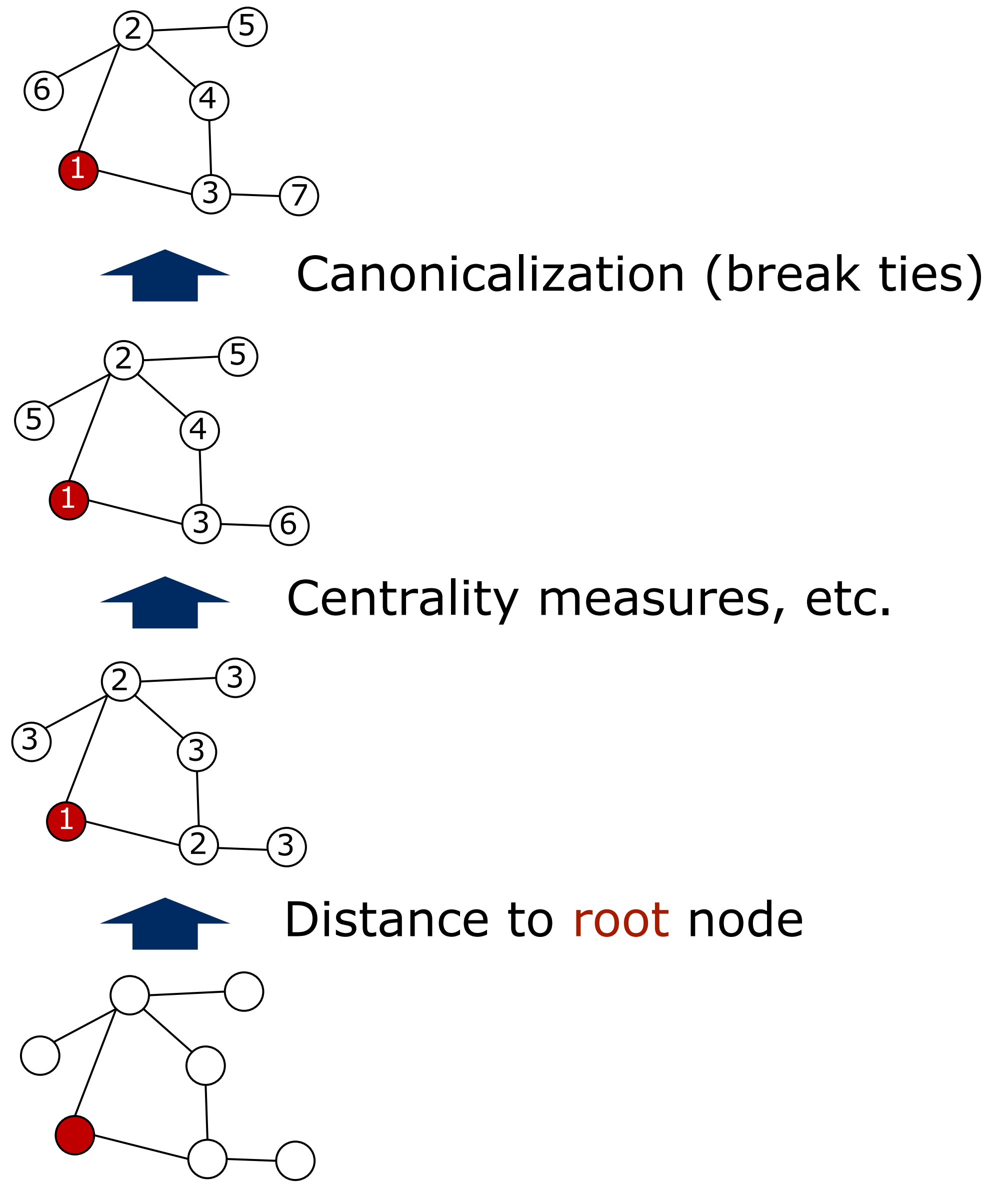
Graph Normalization



Graph Normalization

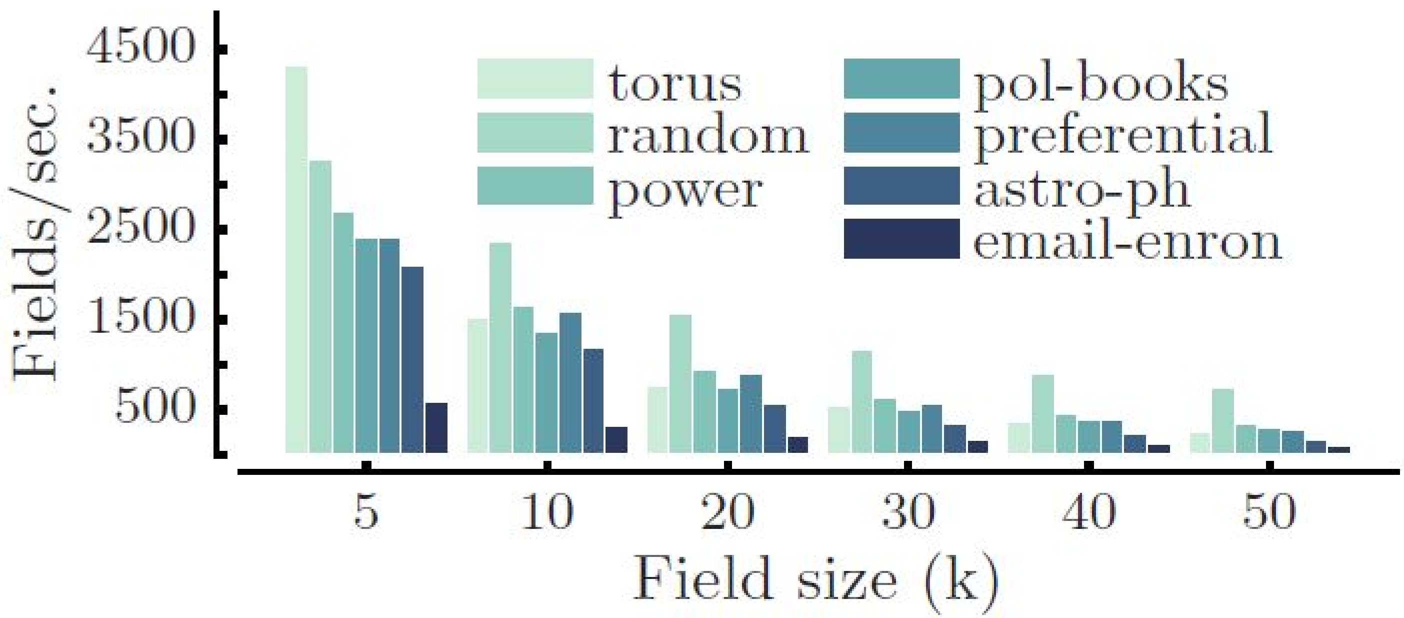


Graph Normalization

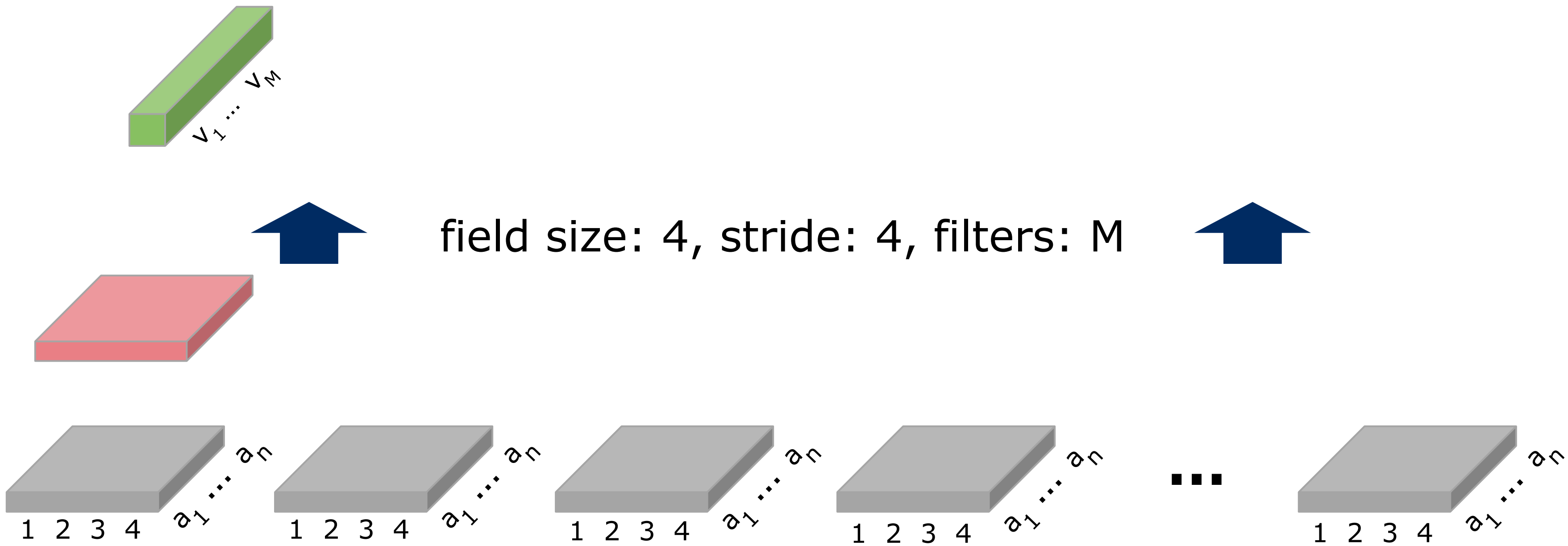


Computational Complexity

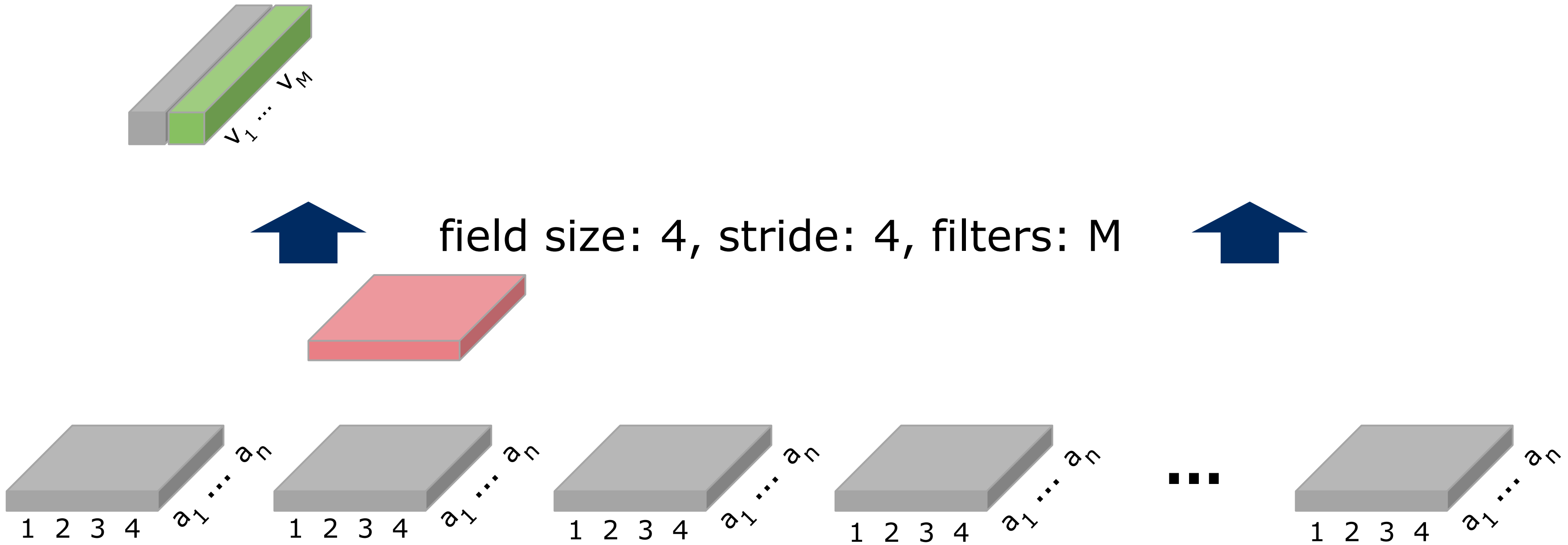
- At most *linear* in number of input graphs
- At most *quadratic* in number of nodes for each graph (depends on maximal node degree and centrality measure)



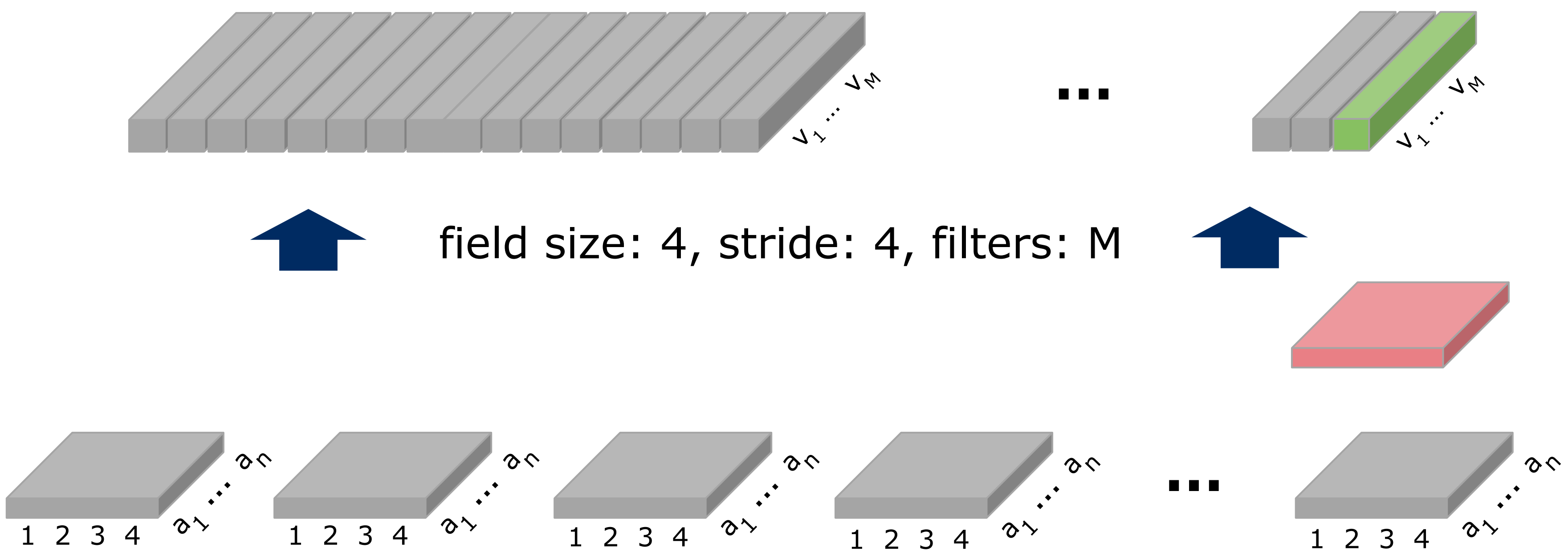
Convolutional Architecture



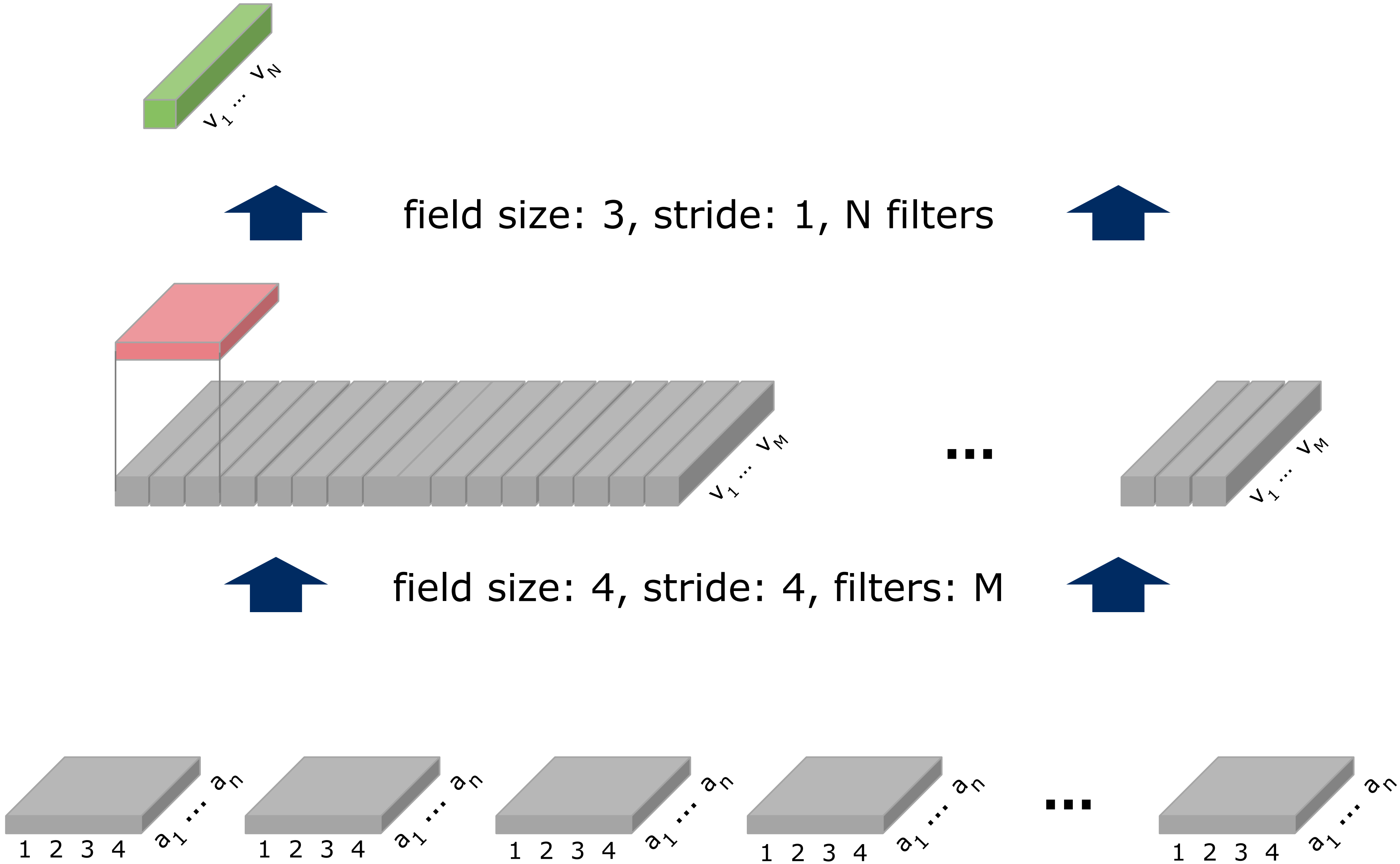
Convolutional Architecture



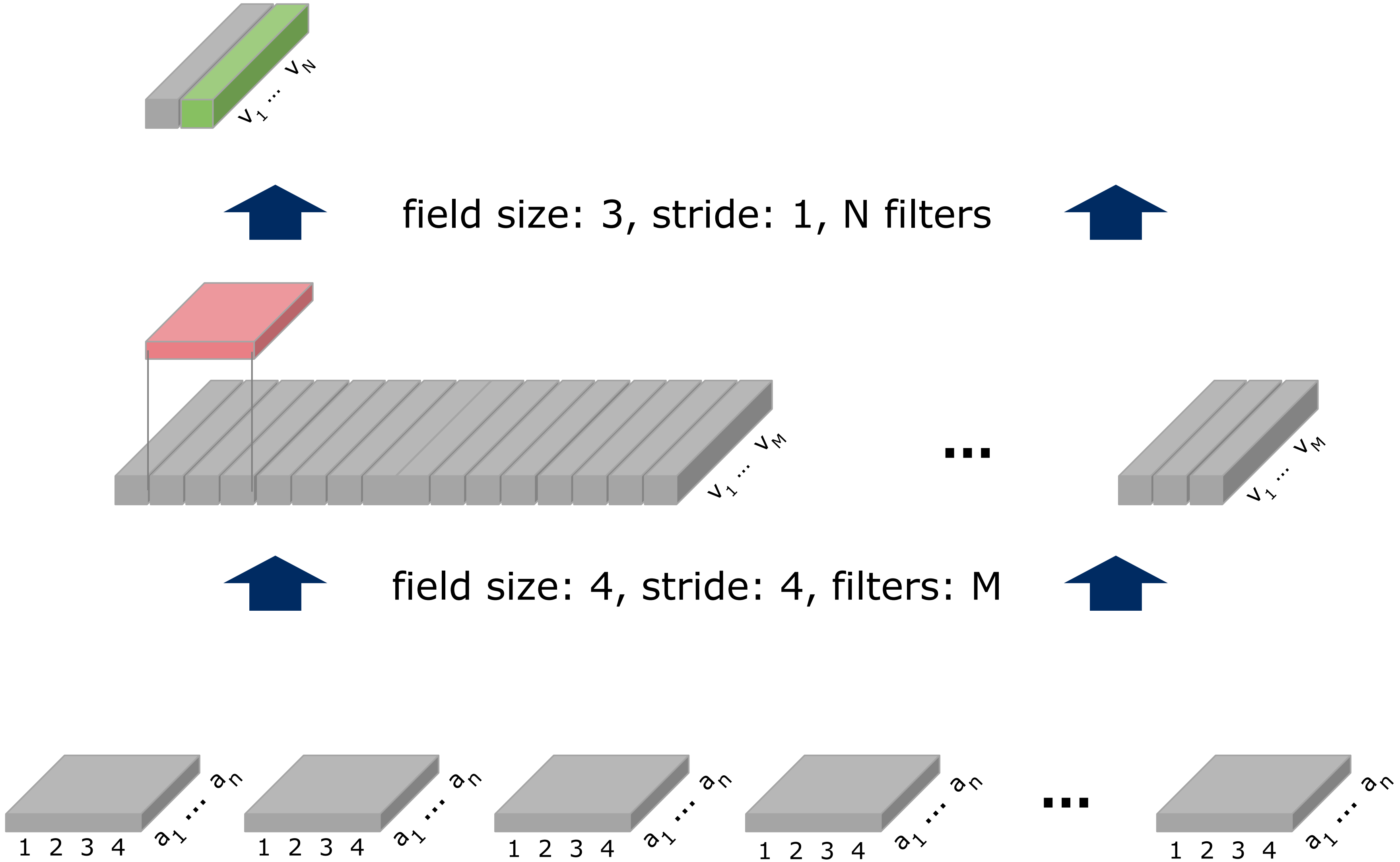
Convolutional Architecture



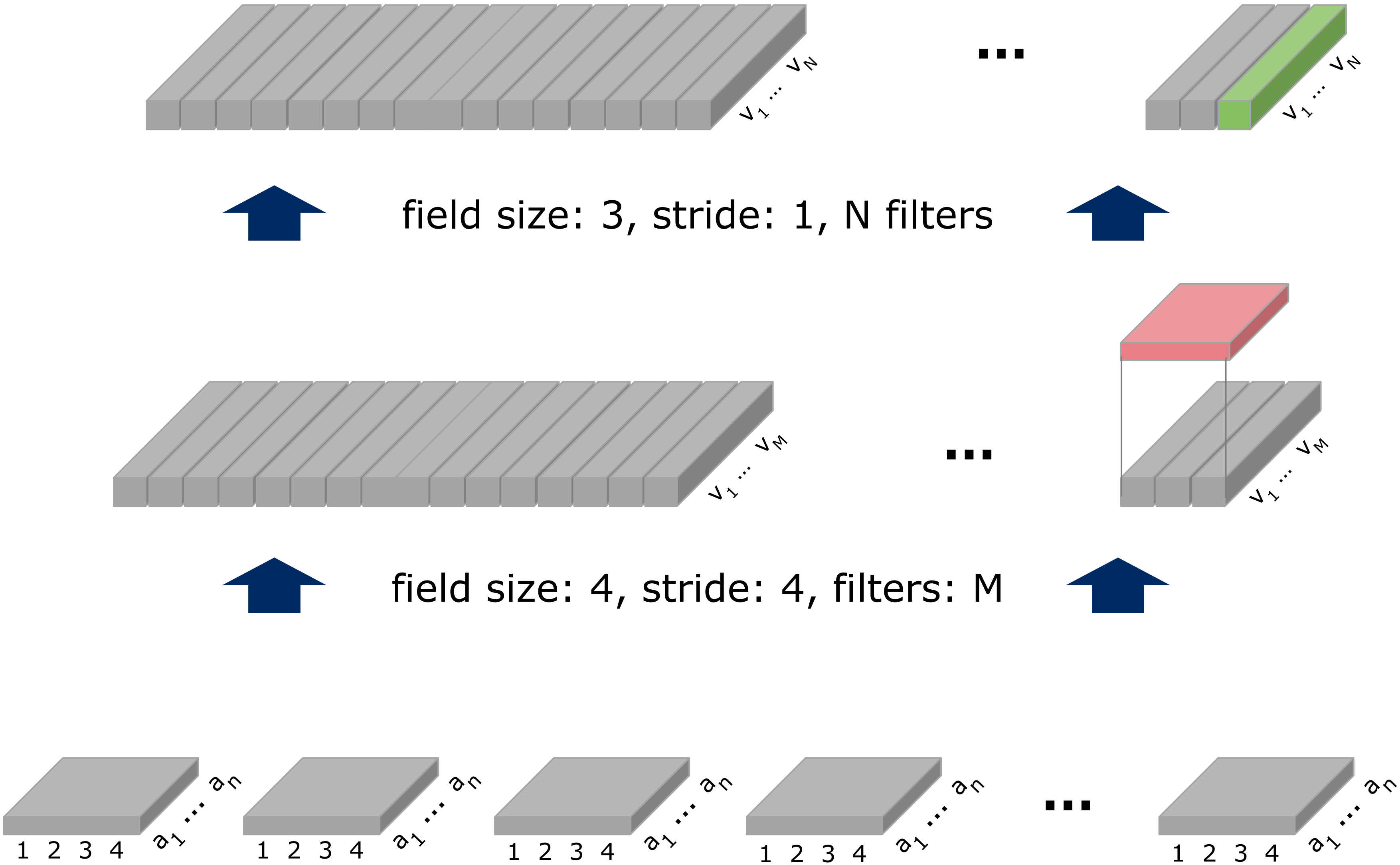
Convolutional Architecture



Convolutional Architecture



Convolutional Architecture



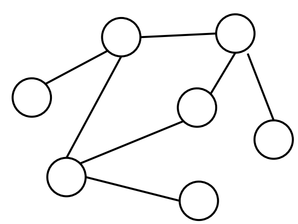
Experiments - Graph Classification

Finite collection of graphs and their class labels

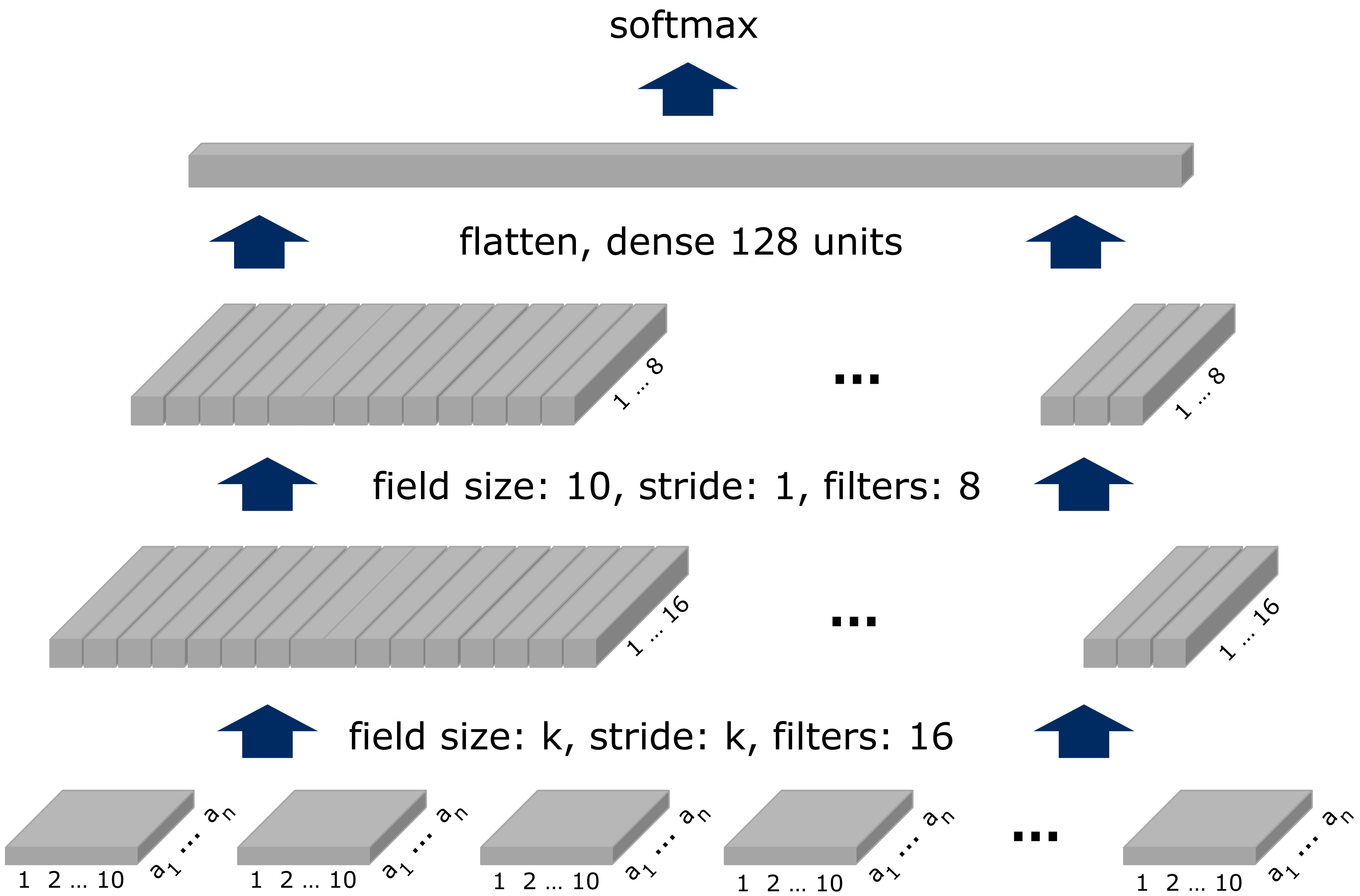
class = **1** class = **0** ... class = **1**

- Nodes of any two graphs are ***not*** necessarily in correspondence
- Nodes and edges may have attributes (discrete and continuous)

Learn a function from graphs to class labels

class[] = ?

Experiments - Convolutional Architecture



Classification Datasets

- **MUTAG:** Nitro compounds where classes indicate mutagenic effect on a bacterium (Salmonella Typhimurium)
- **PTC:** Chemical compounds where classes indicate carcinogenicity for male and female rats
- **NCI:** Chemical compounds where classes indicate activity against non-small cell lung cancer and ovarian cancer cell lines
- **D&D:** Protein structures where classes indicate whether structure is an enzyme or not
- ...

Experiments - Graph Classification

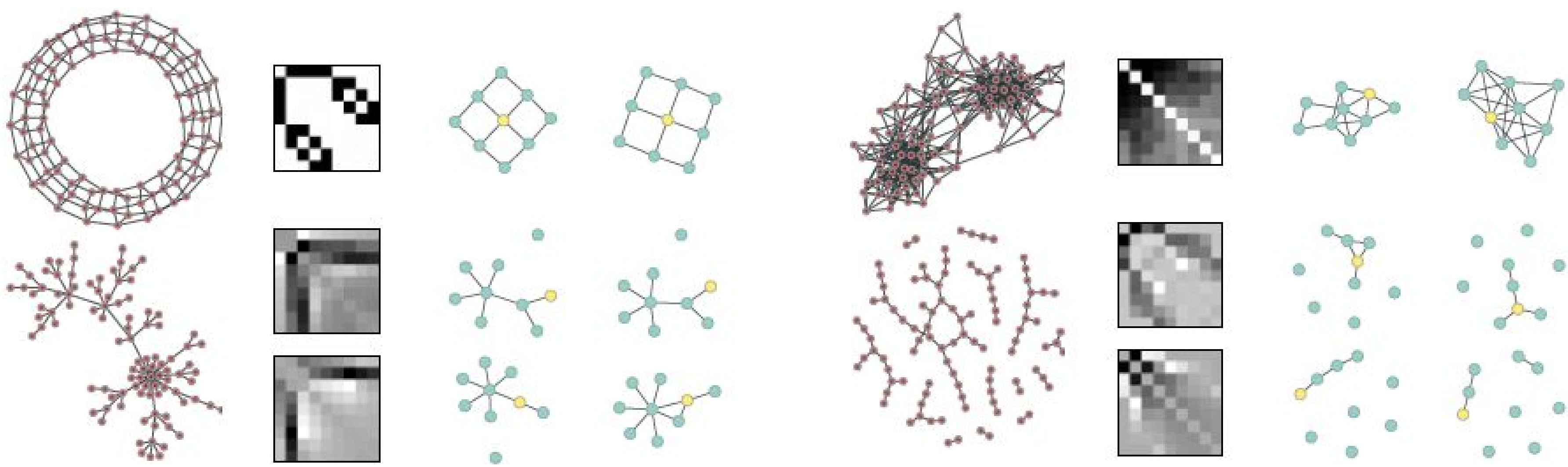
- **Q:** How efficient and effective compared to graph kernels?
- Apply Patchy to typical graph classification benchmark data

Data set	MUTAG	PCT	NCI1	PROTEIN
Max	28	109	111	620
Avg	17.93	25.56	29.87	39.06
Graphs	188	344	4110	1113
SP [7]	85.79 ± 2.51	58.53 ± 2.55	73.00 ± 0.51	75.07 ± 0.54
RW [17]	83.68 ± 1.66	57.26 ± 1.30	$>$ 3 days	74.22 ± 0.42
GK [38]	81.58 ± 2.11	57.32 ± 1.13	62.28 ± 0.29	71.67 ± 0.55
WL [39]	80.72 ± 3.00 ($5s$)	56.97 ± 2.01 ($30s$)	80.22 ± 0.51 ($375s$)	72.92 ± 0.56 ($143s$)
PSCN k=5	91.58 ± 5.86 ($2s$)	59.43 ± 3.14 ($4s$)	72.80 ± 2.06 ($59s$)	74.10 ± 1.72 ($22s$)
PSCN k=10	88.95 ± 4.37 ($3s$)	62.29 ± 5.68 ($6s$)	76.34 ± 1.68 ($76s$)	75.00 ± 2.51 ($30s$)

Data set	GK [38]	DGK [45]	PSCN k=10
COLLAB	72.84 ± 0.28	73.09 ± 0.25	72.60 ± 2.15
IMDB-B	65.87 ± 0.98	66.96 ± 0.56	71.00 ± 2.29
IMDB-M	43.89 ± 0.38	44.55 ± 0.52	45.23 ± 2.84
RE-B	77.34 ± 0.18	78.04 ± 0.39	86.30 ± 1.58
RE-M5k	41.01 ± 0.17	41.27 ± 0.18	49.10 ± 0.70
RE-M10k	31.82 ± 0.08	32.22 ± 0.10	41.32 ± 0.42

Experiments - Visualization

- **Q:** What do learned *edge filters* look like?
- Restricted Boltzmann machine applied to graphs
- Receptive field size of hidden layer: 9



graphs *sampled* from RBM

small instances of graphs

weights of hidden nodes

Discussion

Pros:

- Graph kernel design not required
- Outperforms graph kernels on several datasets (speed and accuracy)
- Incorporates node and edge features (discrete and continuous)
- Supports visualizations (graph motifs, etc.)

Cons:

- Prone to overfitting on smaller data sets (graph kernel benchmarks)
- Shift from designing graph kernels to tuning hyperparameters
- Graph normalization not part of learning

code to be released: patchy.neclab.eu