GRAPH EMBEDDING

AND

GCN

(GRAPH CONVOLUTIONAL NEURAL NETWORKS)

GRAPH EMBEDDING

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78-94.

Cai, H., Zheng, V. W., & Chang, K. C. C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, *30*(9), 1616-1637.

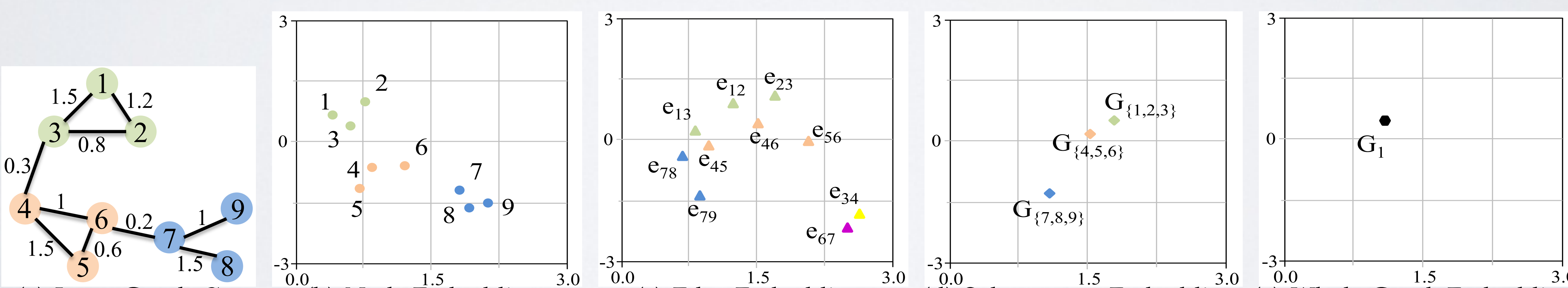
NAMES

- Graph embedding / Network embedding
- Representation learning on networks
  - **Representation learning = feature learning**, as opposed to **manual feature engineering (heuristics)**
- Embedding => Latent space

VARIANT

- We can differentiate:
  - Node embedding
  - Edge Embedding
  - Substructure embedding
  - Whole graph Embedding

- In this course, only *node embedding* (often called graph embedding)



(a) Input Graph G_1 (b) Node Embedding (c) Edge Embedding (d) Substructure Embedding (e) Whole-Graph Embedding

Cai, H., Zheng, V. W., & Chang, K. C. C. (2018). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, *30*(9), 1616-1637.

IN CONCRETE TERMS

- A graph is composed of
  - Nodes (possibly with labels)
  - Edges (possibly directed, weighted, with labels)

- A graph/node embedding technique in **d** dimensions will assign a vector of length **d** to each node, that will be useful for **what we want to do with the graph**.

- A vector can be assigned to an edge *(u,v)* by combining vectors of *u* and *v*

WHAT TO DO WITH EMBEDDINGS?

- Two possible ways to use an embedding:
  - Unsupervised learning:
    - The *distance* between vectors in the embedding is used for *something*
  - Supervised learning:
    - Algorithm learn to predict *something* from the features in the embedding

WHAT CAN WE DO WITH EMBEDDINGS ?

EMBEDDING TASKS

- Common tasks:
  - Link prediction (supervised)
  - Graph reconstruction (unsupervised link prediction ? / ad hoc)
  - Community detection (unsupervised)
  - Node classification (supervised community detection ?)
  - Role definition (Variant of node classification, can be unsupervised)
  - Visualisation (distances, like unsupervised)

OVERVIEW OF MOST POPULAR METHODS

HISTORIC METHODS

(PRE NEURAL NETWORKS)

LE: LAPLACIAN EIGENMAPS

- Introduced 2001
- Objective function:
  - $y* = \min \sum_{i \neq j} ||y_i - y_j||^2 W_{ij}$
    - $y*$: optimal embedding
    - y_i: embedding of node i
    - W_{ij}: weight between nodes *i* and *j*
- Nodes connected (close) in the graph should be close in the embedding, Highest weights = strongest influence

LE: LAPLACIAN EIGENMAPS

- $y* = \min \sum_{i \neq j} \|y_i - y_j\|^2 W_{ij}$
- Can be written in matrix form as:
  - $\min y^T L y$
  - L: Laplacian, D: Degree matrix
- To avoid trivial solution, we impose the constraint:
  - $y^T D y = I$
- Solution: d eigenvectors of lowest eigenvalues of $D^{-1/2} L D^{-1/2}$

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78-94.

HOPE: HIGHER-ORDER PROXIMITY PRESERVED EMBEDDING

- Preserve a proximity matrix
- $y* = \min \sum_{i,j} |W_{ij} - y_i y_j^T|$
- W can be the adjacency matrix, or number of common neighbors, Adamic Adar, etc.
- As similarity tends towards 0, associated embeddings should tend towards orthogonality

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78-94.

LLE: LOCALLY LINEAR EMBEDDING

- Introduced 2000
- A node features can be represented as a linear combination of its neighbors'
  - $Y_i = \sum_j A_{ij} Y_j$
- Objective function:
  - $y* = \min \sum_i \|Y_i - \sum_j A_{ij} Y_j\|^2$

Goyal, P., & Ferrara, E. (2018). Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, *151*, 78-94.

RANDOM WALKS BASED

DEEPWALK

- The first "modern" graph embedding method
- Adaptation of **word2vec**/**skipgram** to graphs

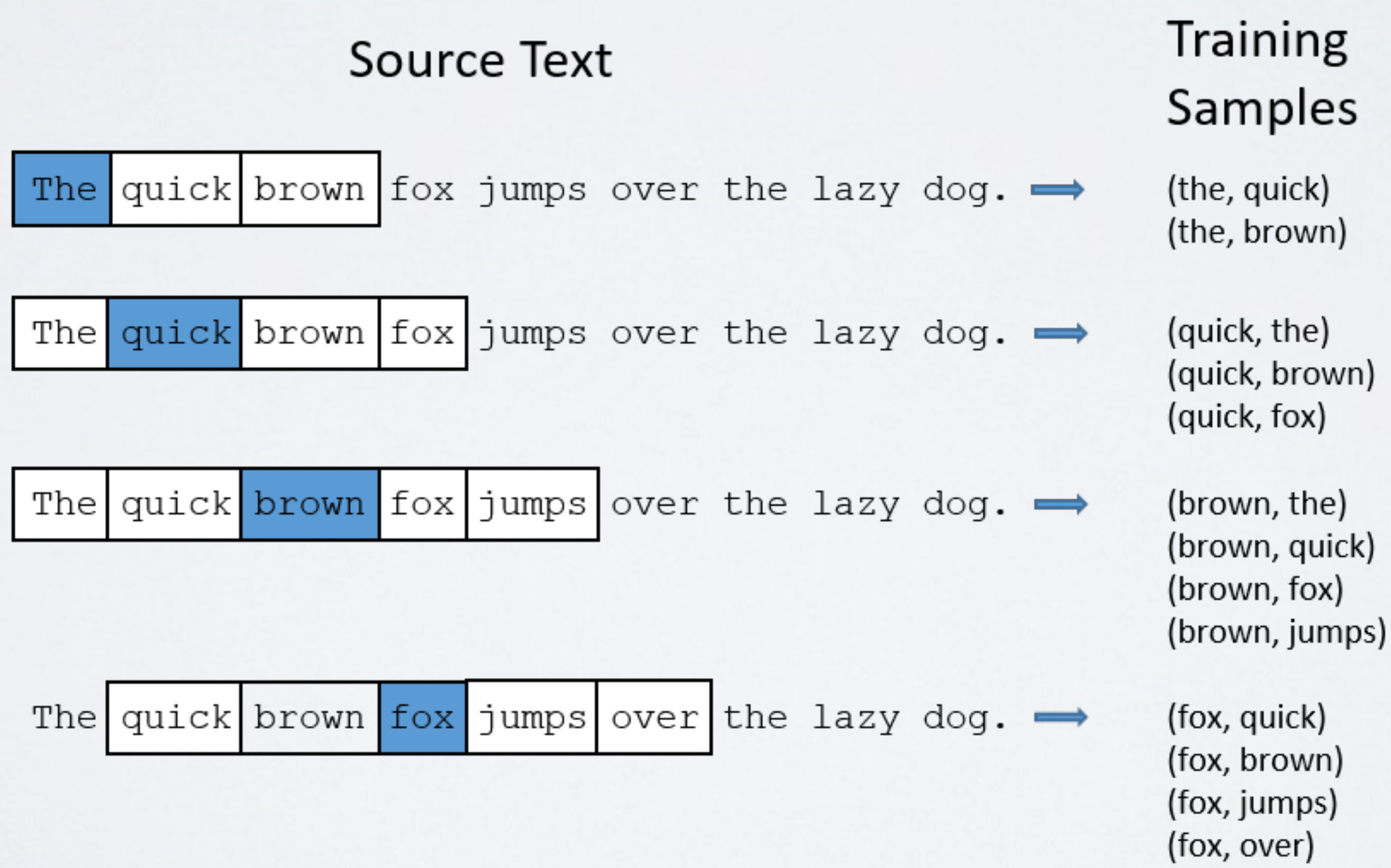
Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701-710). ACM.

SKIPGRAM

Word embedding

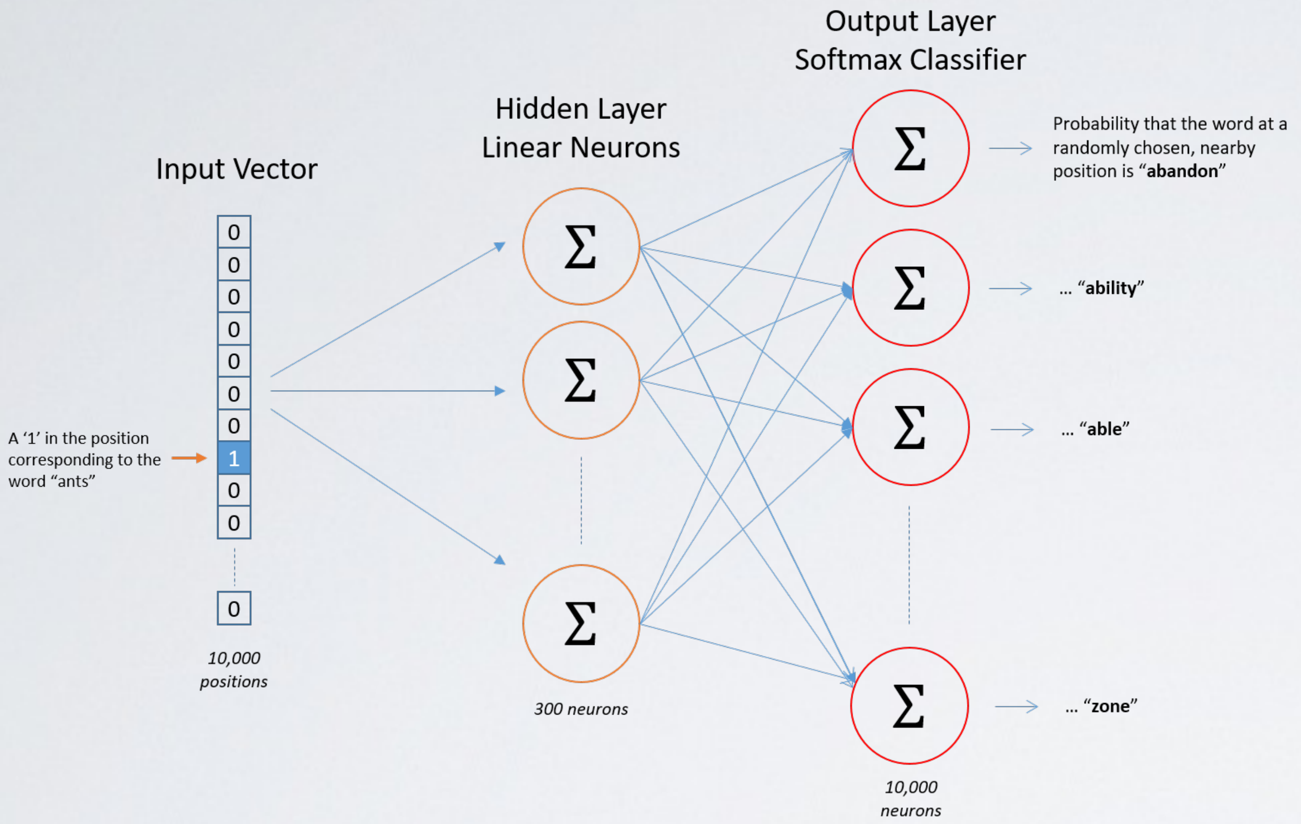
Corpus => Word = vectors

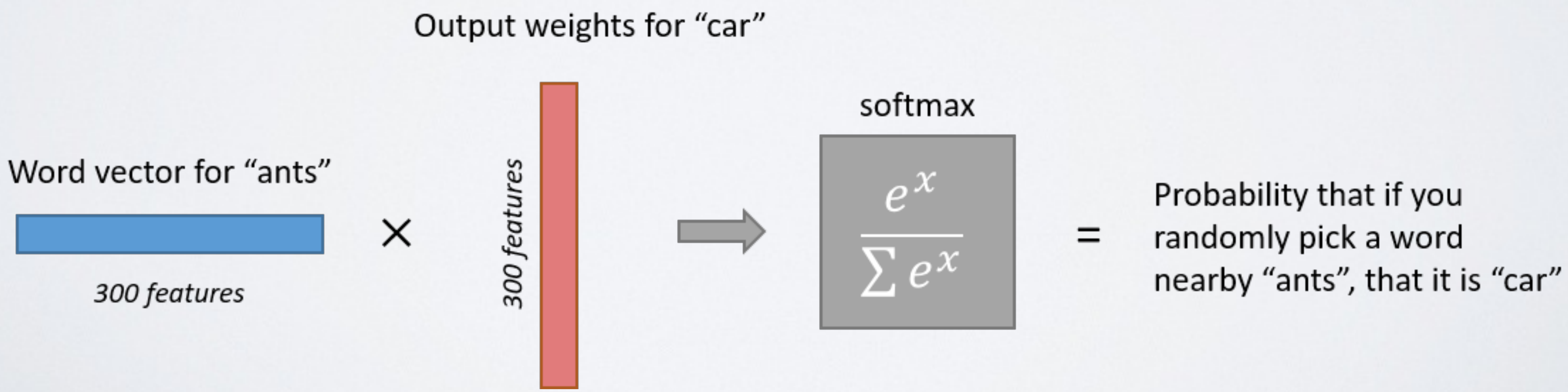
Similar embedding= similar **context**



[http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/]

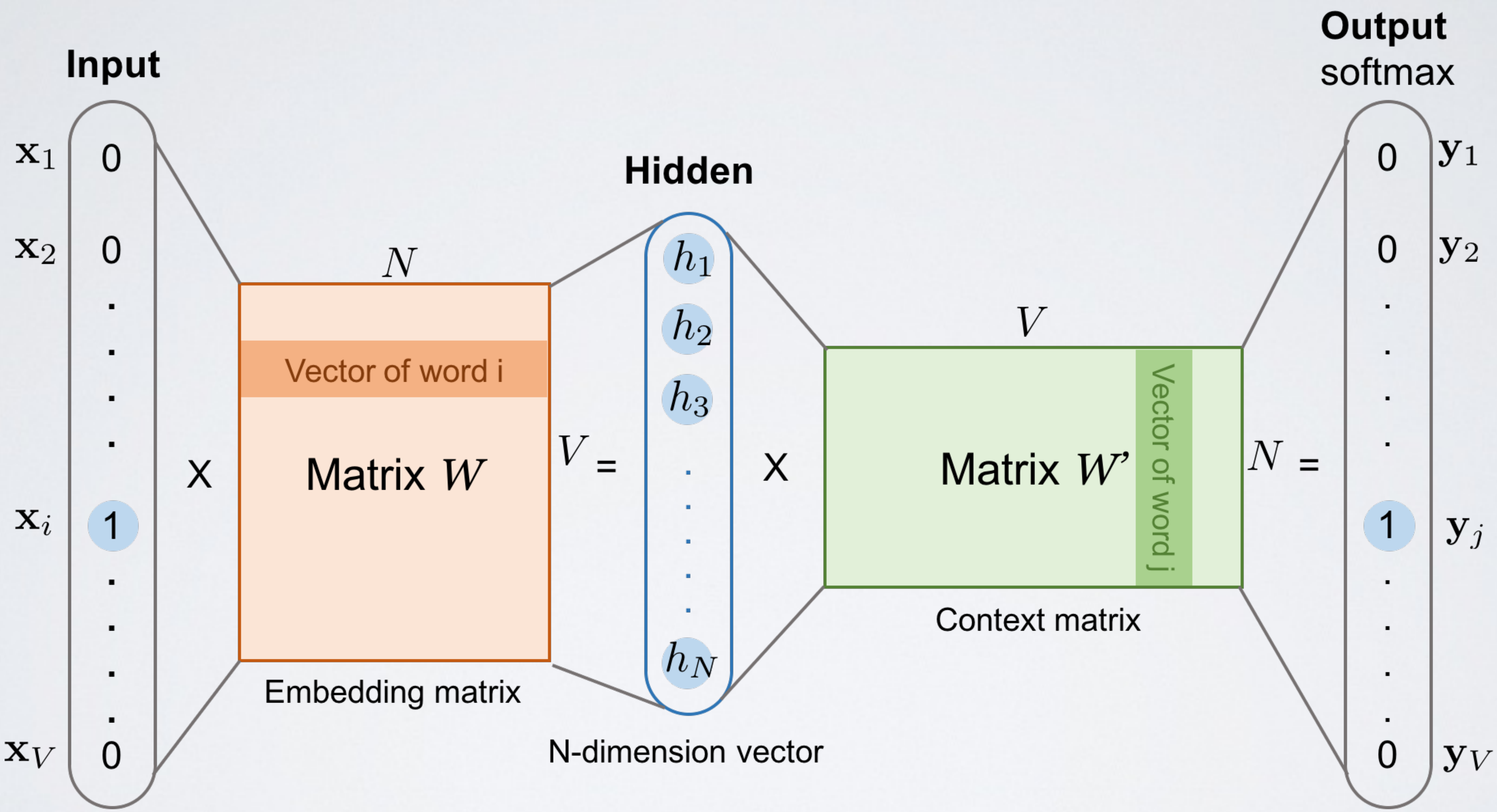
SKIPGRAM





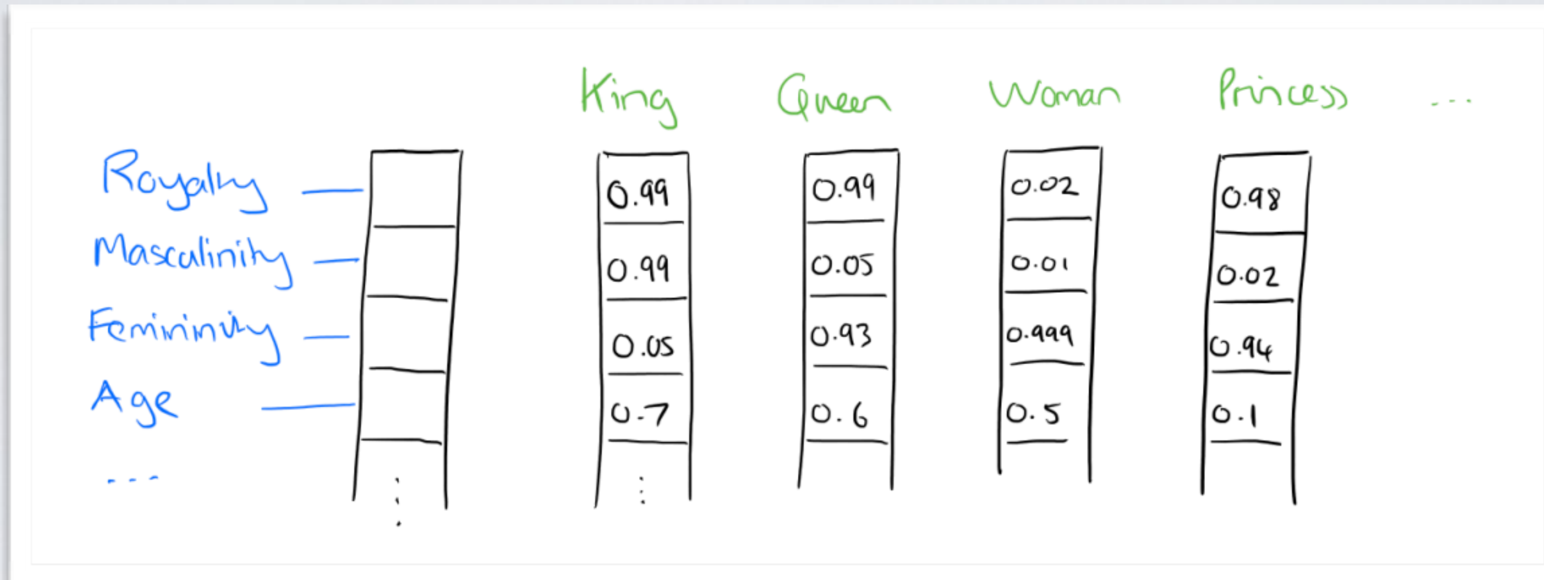
https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

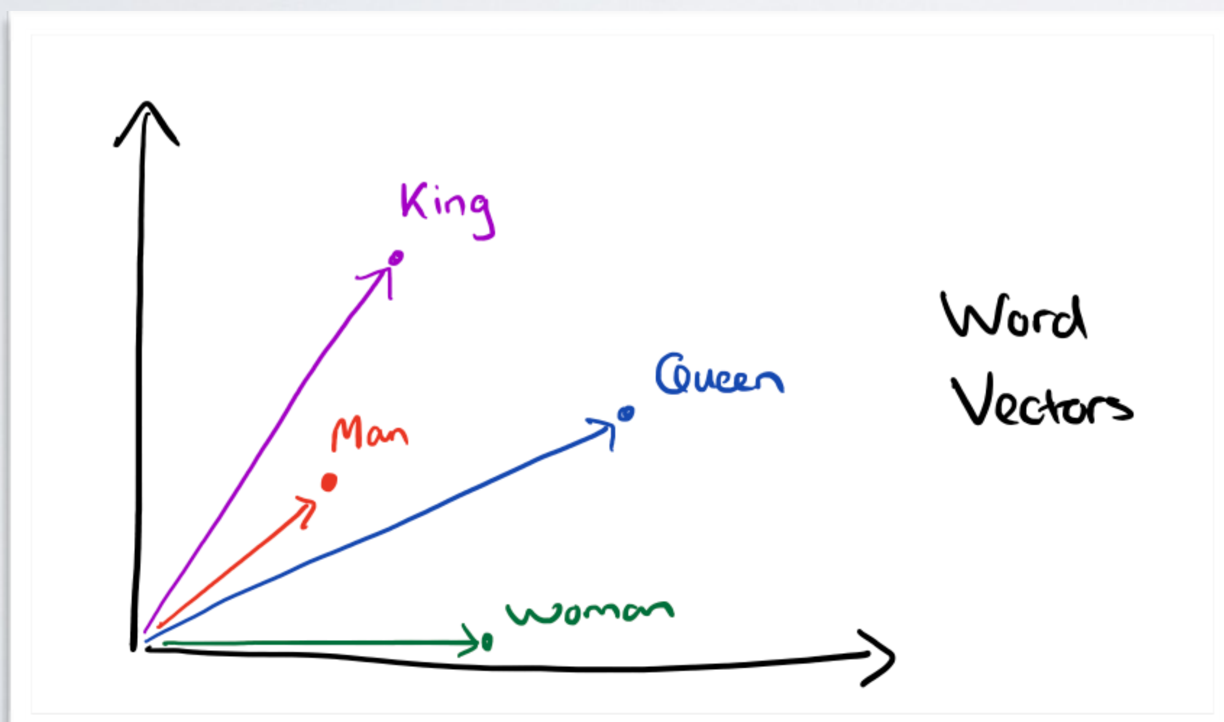
SKIPGRAM

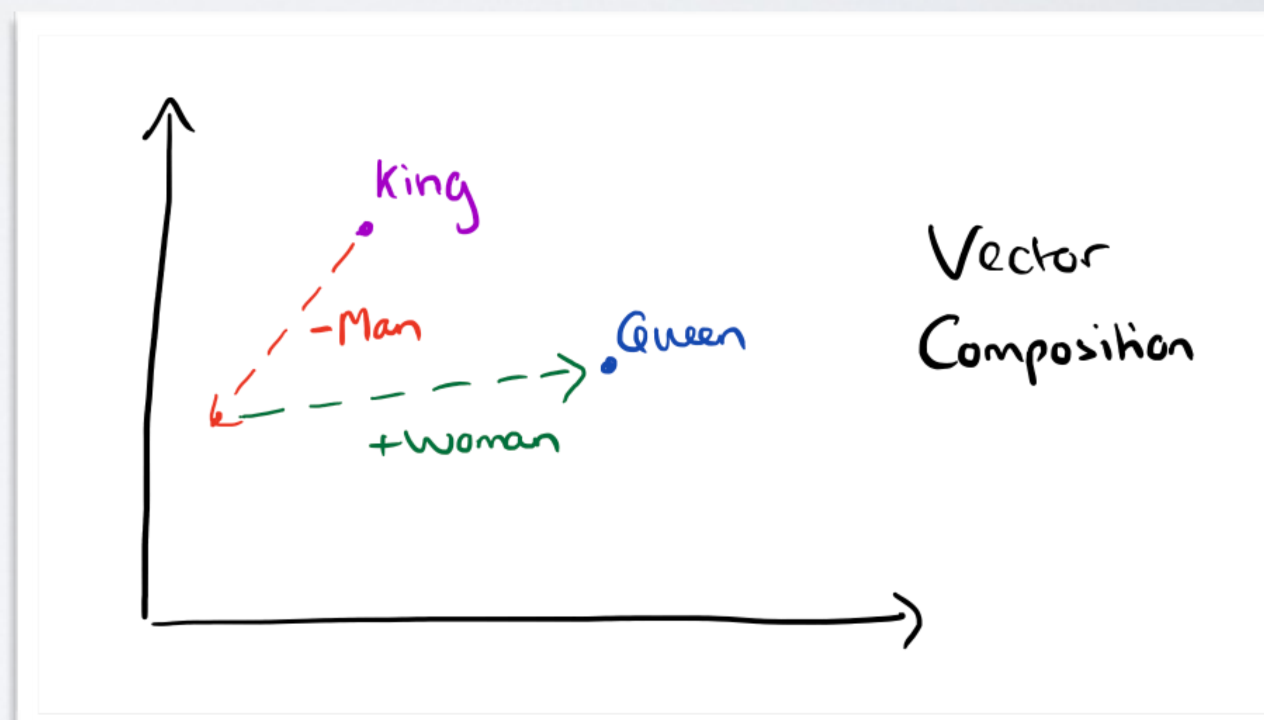


https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b

GENERIC "SKIPGRAM"







[https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/]

GENERIC "SKIPGRAM"

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

[https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/]

GENERIC "SKIPGRAM"

- Algorithm that takes an input:
  - The element to embed
  - A list of "context" elements
- Provide as output:
  - An embedding with interesting properties
    - Works well for machine learning
    - Similar elements are close in the embedding
    - Somewhat preserves the overall structure

DEEPWALK

- Skipgram for graphs:
  - 1)Generate "sentences" using random walks
  - 2)Apply Skipgram

- Parameters: dimensions *d*, RW length *k*

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701-710). ACM.

NODE2VEC

- Use biased random walk to tune the context to capture *what we want*
  - "Breadth first" like RW => local neighborhood (edge probability ?)
  - "Depth-first" like RW => global structure ? (Communities ?)
  - 2 parameters to tune:
    - **p**: bias towards revisiting the previous node
    - **q**: bias towards exploring undiscovered parts of the network

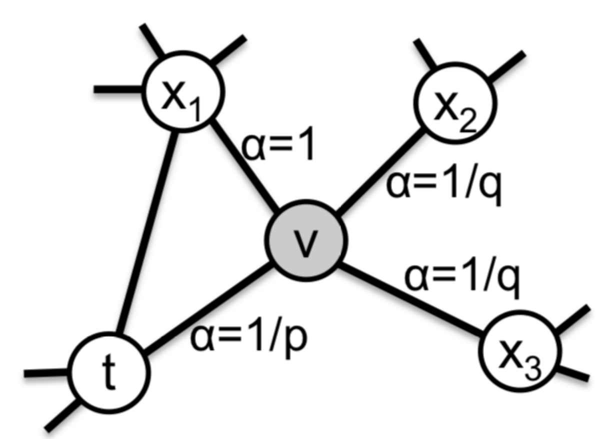


Figure 2: Illustration of the random walk procedure in *node2vec*. The walk just transitioned from t to v and is now evaluating its next step out of node v. Edge labels indicate search biases α.

Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864). ACM.

RANDOM WALK METHODS

- What is the objective function ?
- How to interpret the distance between nodes in the embedding ?

ENCODER DECODER FRAMEWORK

Minimize a global loss defined as:

$$L = \sum_{(v_i, v_j) \in E} \ell(DEC(z_i, z_j), s_{\mathcal{G}}(v_i, v_j))$$

DEC: Decoder function (e.g., $DEC(z_i, z_j) = z_i^T z_j$)

$s_{\mathcal{G}}$: Ground truth similarity (e.g., $s_{\mathcal{G}(v_i, v_j)} = A_{ij}$)

ℓ: Chosen loss function (e.g., $\ell(a, b) = |a - b|$)

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

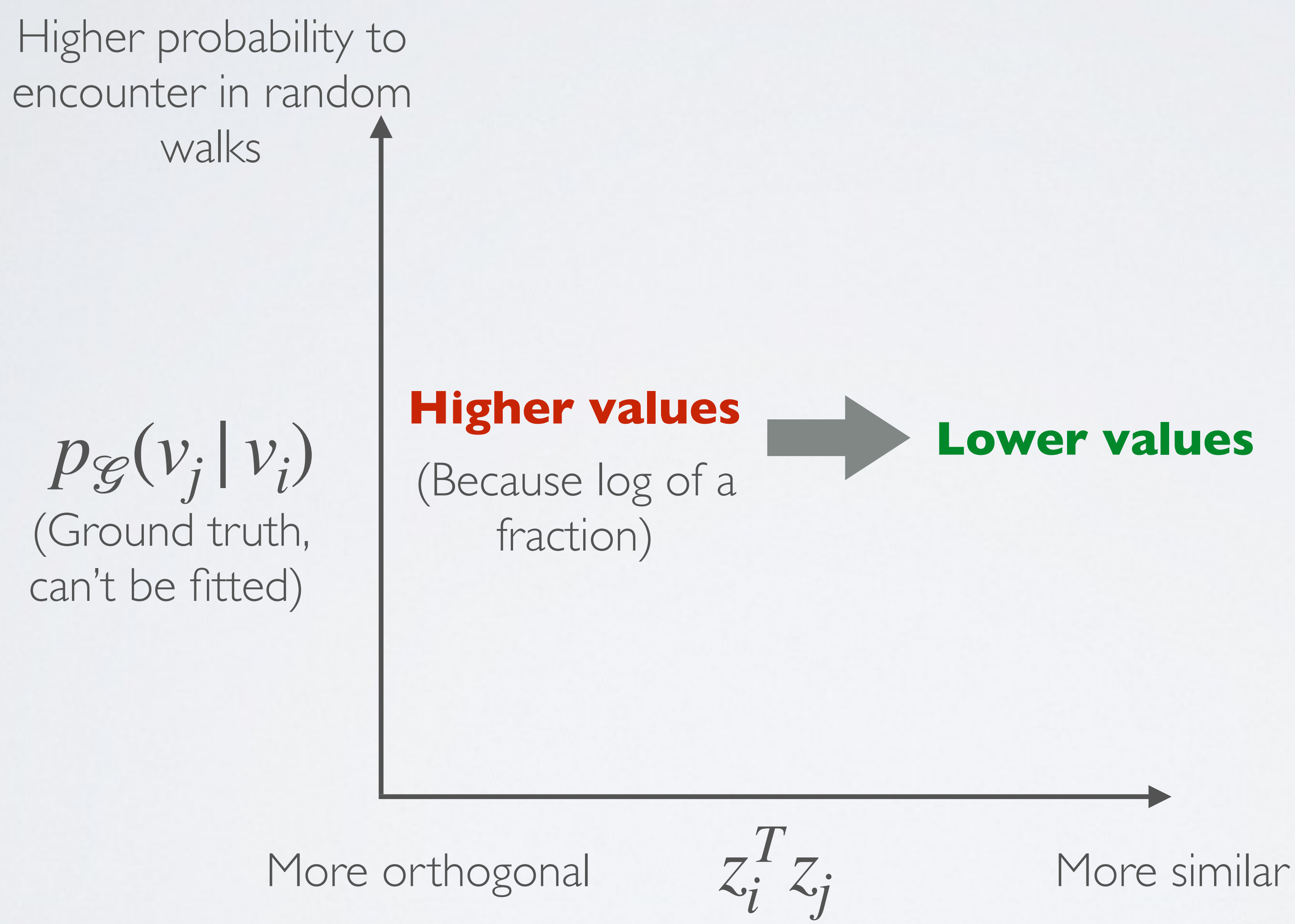
ENCODER DECODER FRAMEWORK

Type	Method	Decoder	Proximity measure	Loss function (ℓ)
Matrix factorization	Laplacian Eigenmaps [4]	$\\|\mathbf{z}_i - \mathbf{z}_j\\|_2^2$	general	$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) \cdot s_{\mathcal{G}}(v_i, v_j)$
	Graph Factorization [1]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}$	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
	GraRep [9]	$\mathbf{z}_i^\top \mathbf{z}_j$	$\mathbf{A}_{i,j}, \mathbf{A}_{i,j}^2, ..., \mathbf{A}_{i,j}^k$	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
	HOPE [44]	$\mathbf{z}_i^\top \mathbf{z}_j$	general	$\\|\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) - s_{\mathcal{G}}(v_i, v_j)\\|_2^2$
Random walk	DeepWalk [46]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$
	node2vec [27]	$\frac{e^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_i^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v_j \| v_i)$ (biased)	$-s_{\mathcal{G}}(v_i, v_j) \log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j))$

$p_{\mathcal{G}}(v_j | v_i)$: probability of visiting v_j on a fixed-length random walk started from v_i

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

ENCODER DECODER FRAMEWORK



Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

SOME REMARKS ON WHAT ARE EMBEDDINGS

ADJACENCY MATRIX

- An adjacency matrix is an embedding… (in high dimension)
- That represents the structural equivalence
  - 2 nodes have similar "embeddings" if they have similar neighborhoods
- Standard dimensionality reduction of this matrix can be meaningful
  - Isomap, T-SNE, etc.

GRAPH LAYOUT

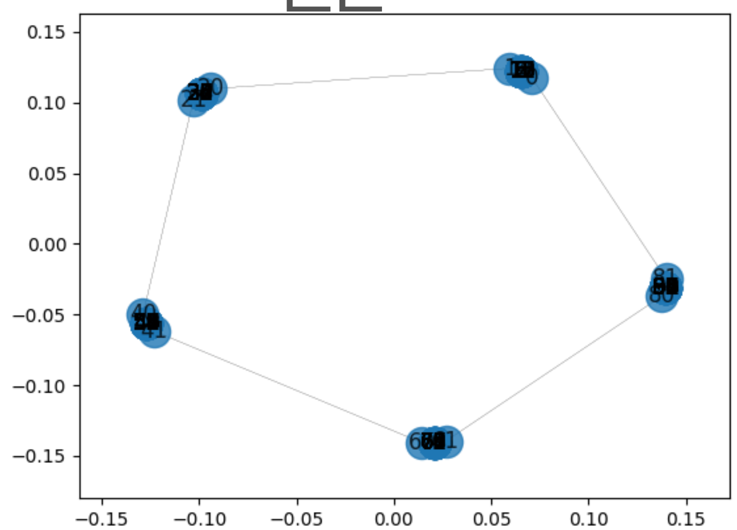
- Graph layouts are also embeddings.
  - Force layout, kamada-kawai ….
- They try to put connected nodes close to each other and non-connected ones "not close"
- Problem: they try to avoid overlaps
- Usually not scalable

VISUALLY ?

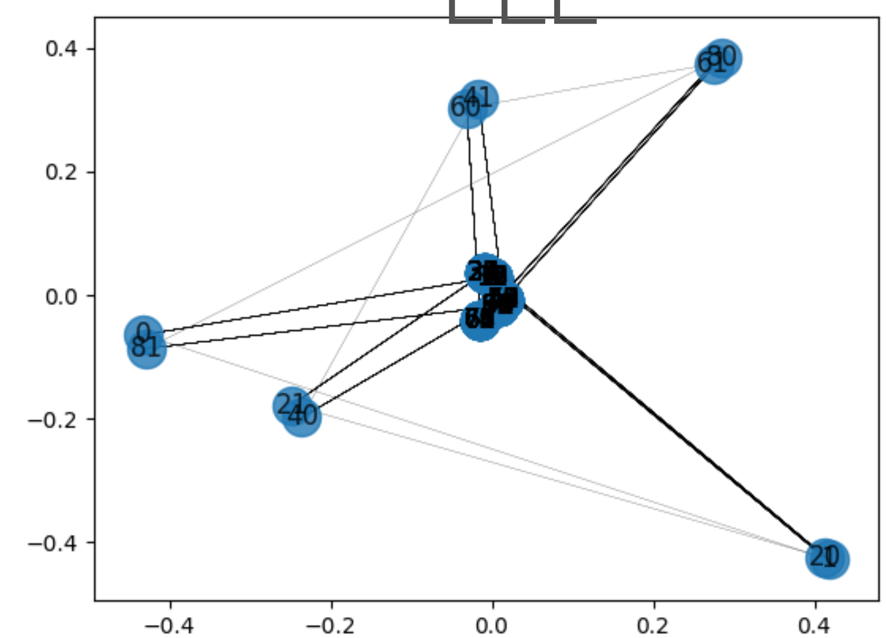
CLIQUE RING

5 cliques or size 20 with 1 edge between them

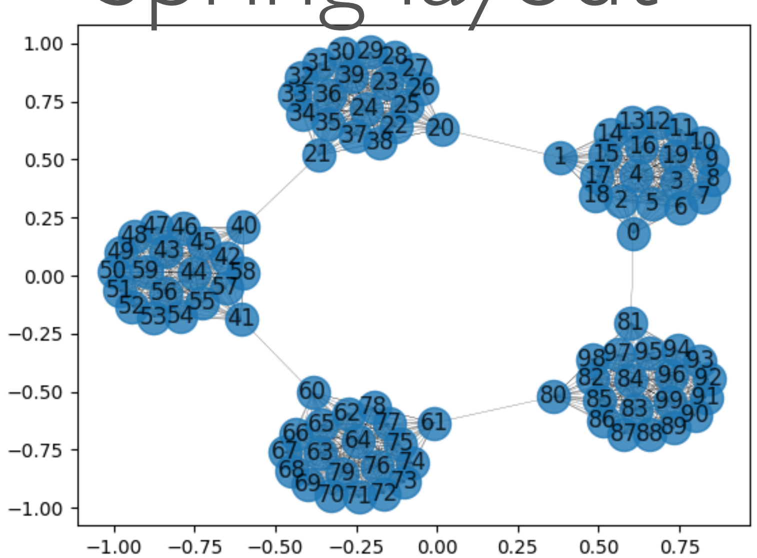
LE



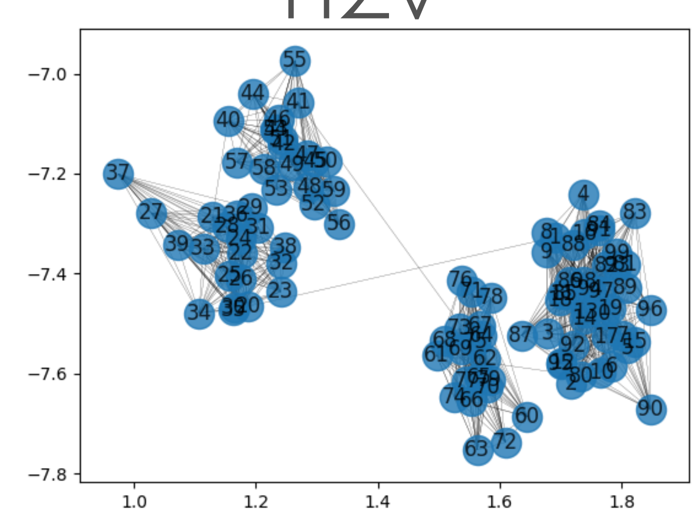
LLE



Spring layout



n2v



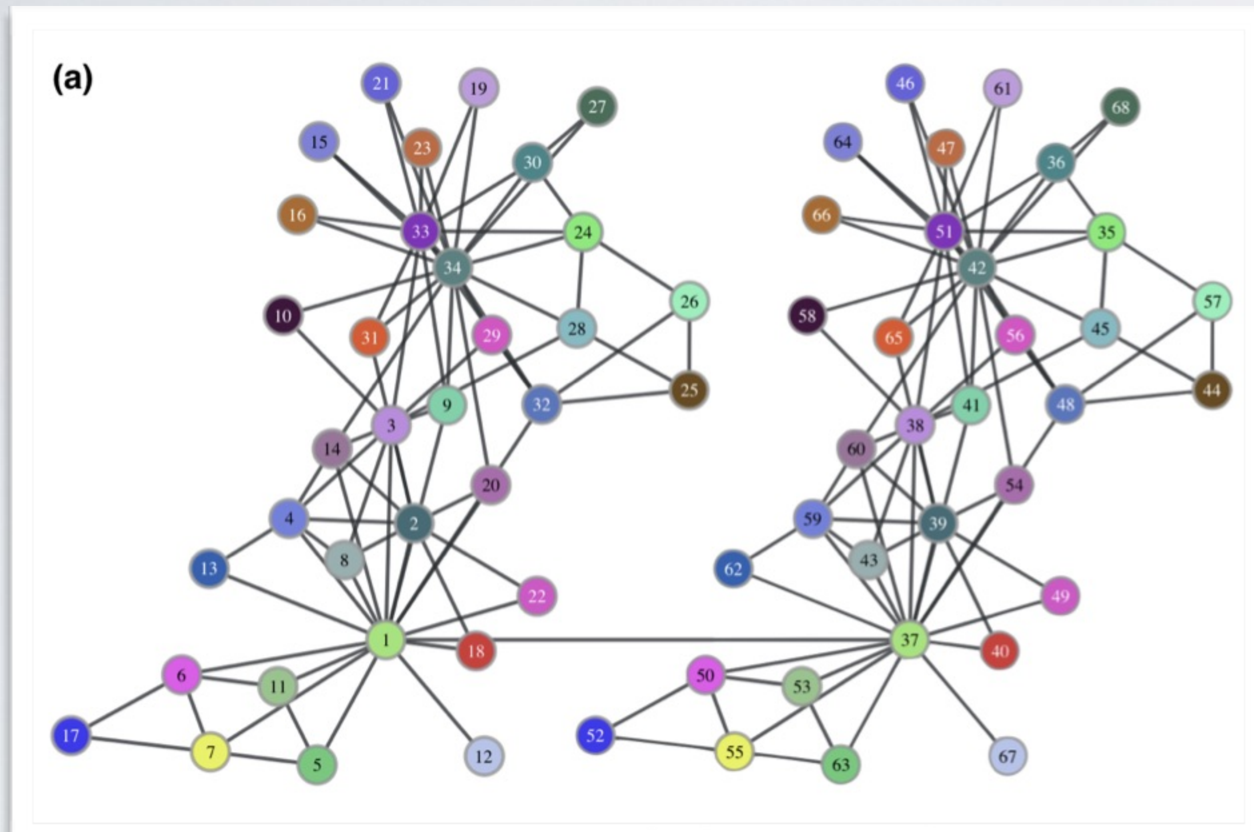
EMBEDDING ROLES

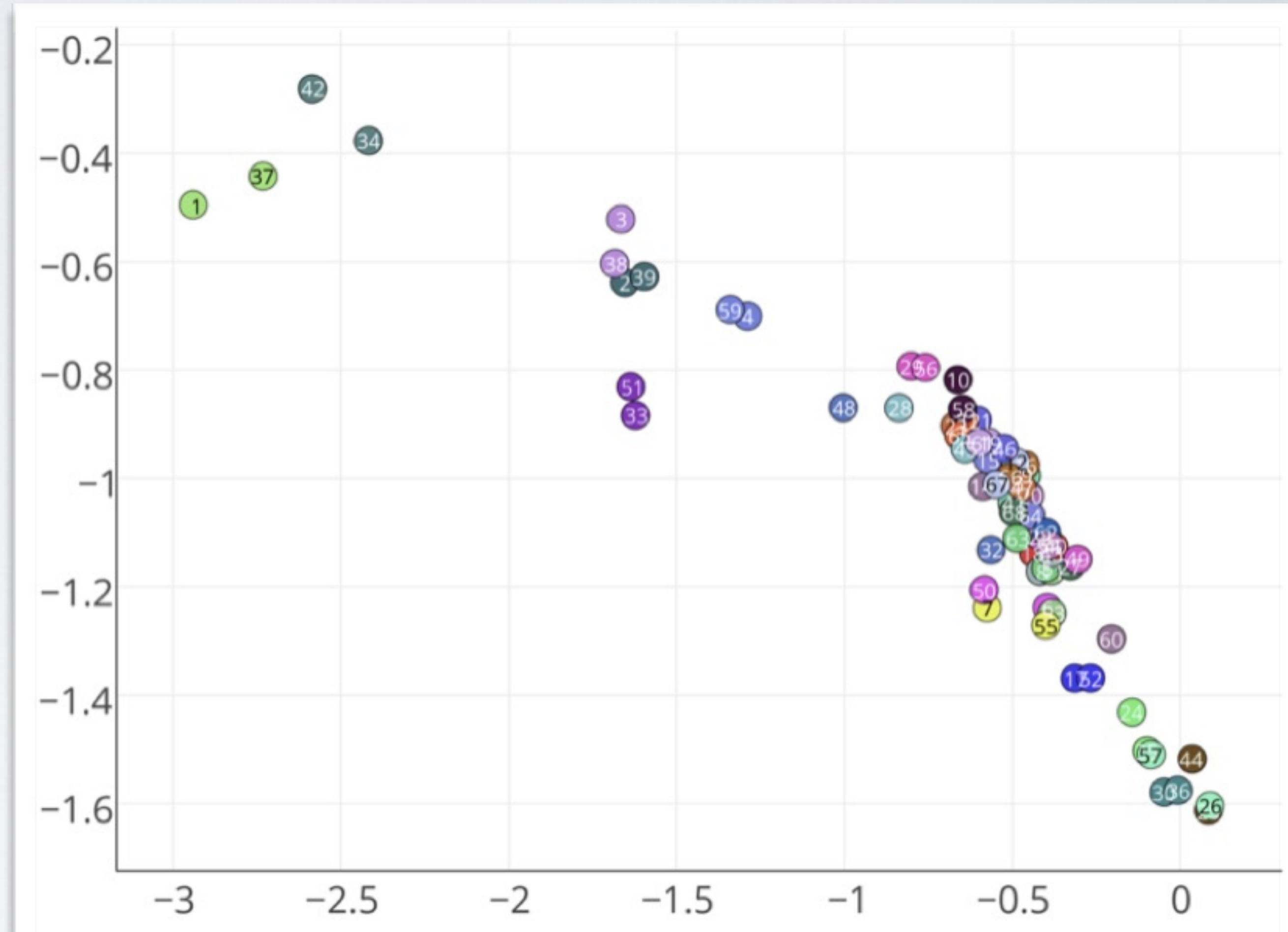
STRUCT2VEC

- In node2vec/Deepwalk, the context collected by RW contain the **labels** of encountered nodes
- Instead, we could memorize the **properties** of the nodes: attributes if available, or computed attributes (degrees, CC, …)
- =>Nodes with a same context will be nodes in a same "position" in the graph
- =>Capture the role of nodes instead of proximity

Ribeiro, L. F., Saverese, P. H., & Figueiredo, D. R. (2017, August). struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 385-394). ACM.

STRUCT2VEC : DOUBLE ZKC





(c) *struc2vec*

Ribeiro, L. F., Saverese, P. H., & Figueiredo, D. R. (2017, August). struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 385-394). ACM.

MEANING OF DISTANCE IN EMBEDDINGS

DISTANCE IN EMBEDDINGS

- In embeddings, each node has an associated vector
- We can compute the distance between vectors
  - Euclidean distance (L2 norm)
  - Manhattan distance (L1 norm)
  - Cosine distance (angle)
  - Dot product (angle and magnitude, =cosine distance for normalized vectors)
- Objective function tells us what the distance should mean
  - Does algorithm succeed in embedding what they want?
  - Does embedding one property preserves somewhat others?

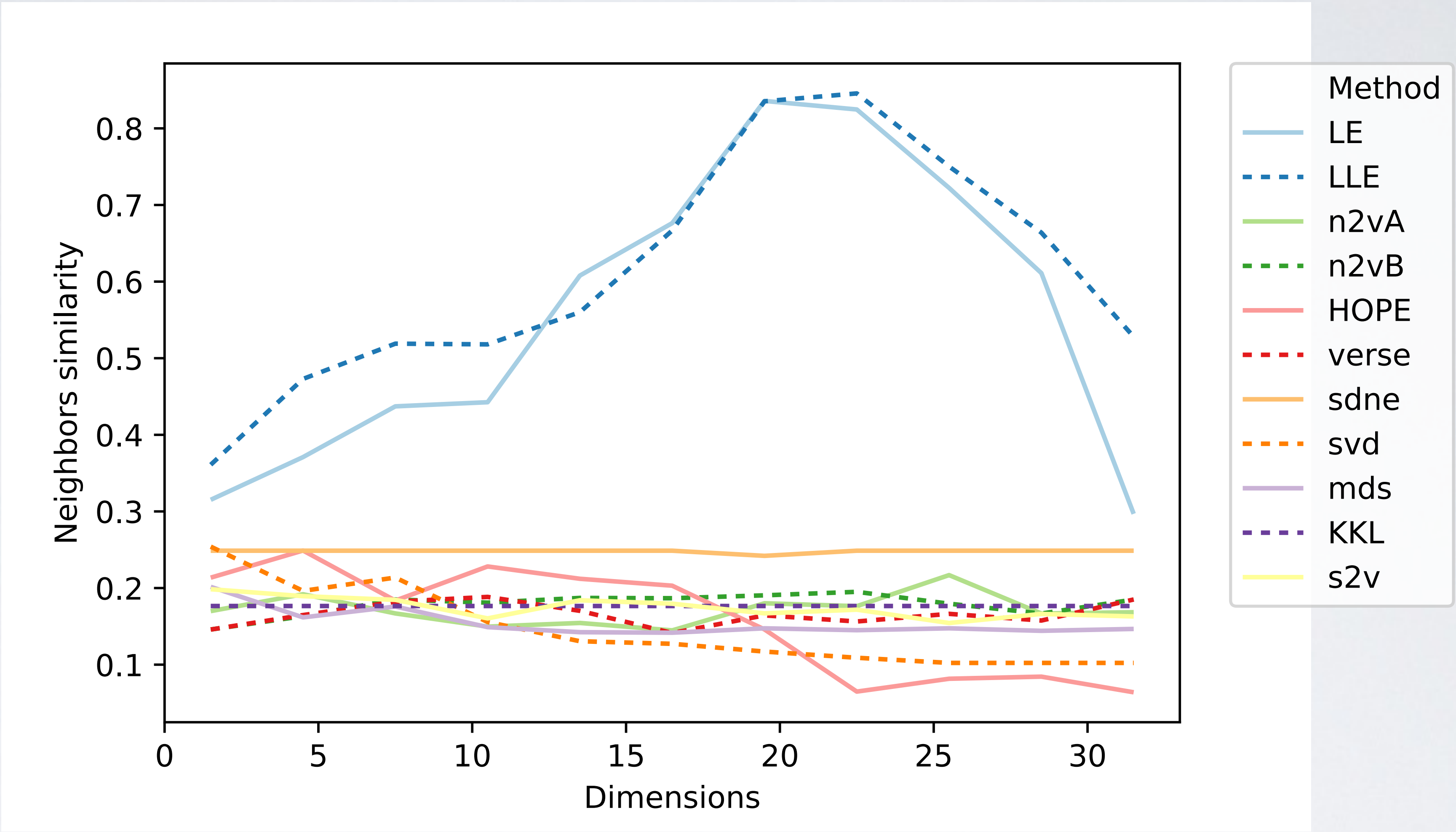
DISTANCE IN EMBEDDINGS

- Several possibilities:
  - Distance preserves the **probability of having an edge**
    - We can reconstruct the network from distances
  - Distance preserves the **similarity of neighborhood**
    - Called Structural equivalence
  - Distance preserves the **role in the network**
    - Hard to define
  - Distance preserves the **community structure**
    - Or another type of mesoscopic organization?

DISTANCE IN EMBEDDINGS

- Distance <=> having an edge?
- For each node:
  - 1)Find the neighbors in the graph. Number of N is k
  - 2)Find the k closest nodes in the embedding
  - 3)Compute the fraction of nodes in common in 1) and 2)
- Compute the average over all nodes

DISTANCE IN EMBEDDINGS



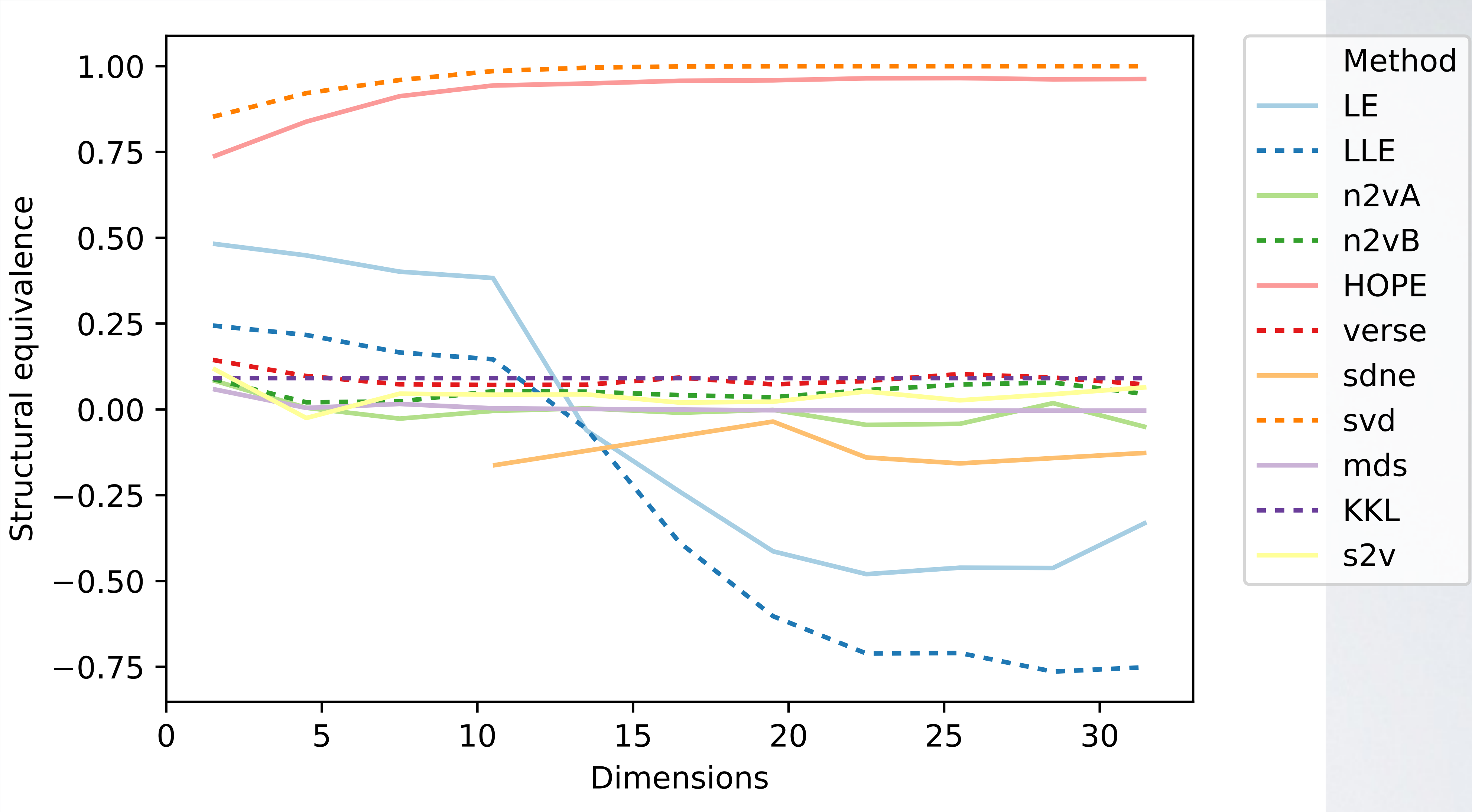
(d) ZKC

Only LE,LLE capture this property

STRUCTURAL EQUIVALENCE

- For each pair of nodes:
  - 1)Compute distance between rows of the adjacency matrix
    - Distance between neighborhoods
  - 2)Compute distance in the embedding
  - 3)Compute Correlation (Spearman) between both ordered sets of values
- =>How strongly both distances are correlated

STRUCTURAL EQUIVALENCE



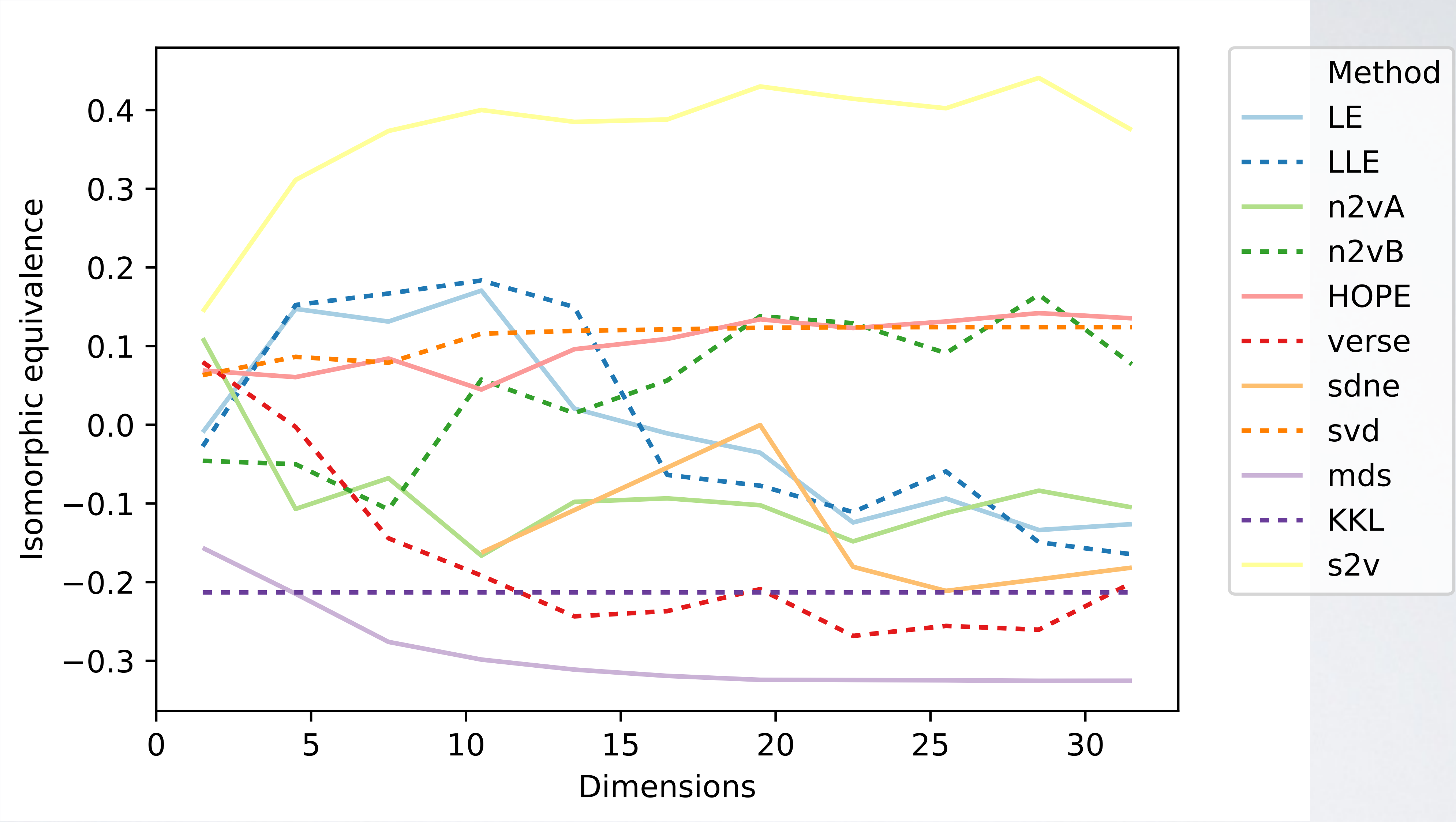
(d) ZKC

svd: dimensionality reduction via SVD
HOPE with Common neighbors as similarity

ROLES: ISOMORPHIC EQUIVALENCE

- For each pair of nodes:
  - 1)Retrieve their unlabeled ego-network
    - Compute the Edit-distance between those networks (# atomic changes to go from one to the other (node/edge addition/removal)
  - 2)Compute distance in the embedding
  - 3)Compute Correlation (Spearman) between both ordered sets of values
- =>How strongly both distances are correlated

ISOMORPHIC EQUIVALENCE



(d) ZKC

Struc2vec only method to embed this property

COMMUNITY STRUCTURE

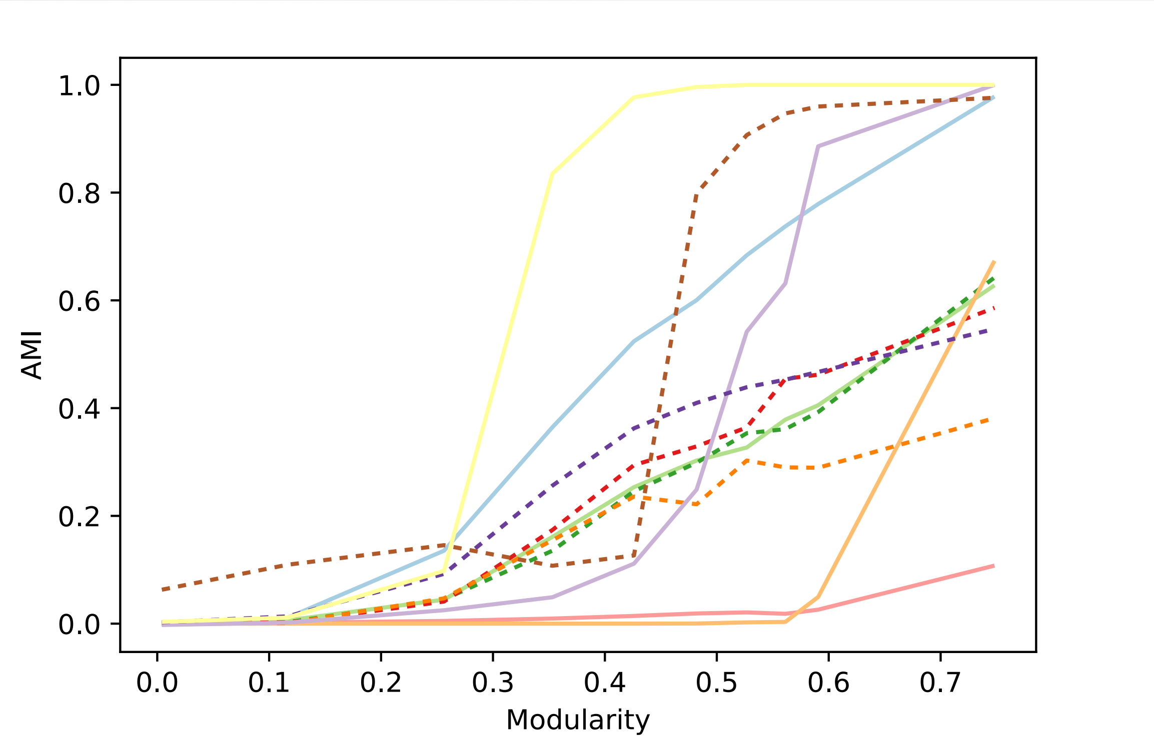
- Idea: if distance preserves community structure:
  - Nodes belonging to the same community should be close in the embedding
- We can use clustering algorithms (k-means…) to discover the communities

COMMUNITY STRUCTURE

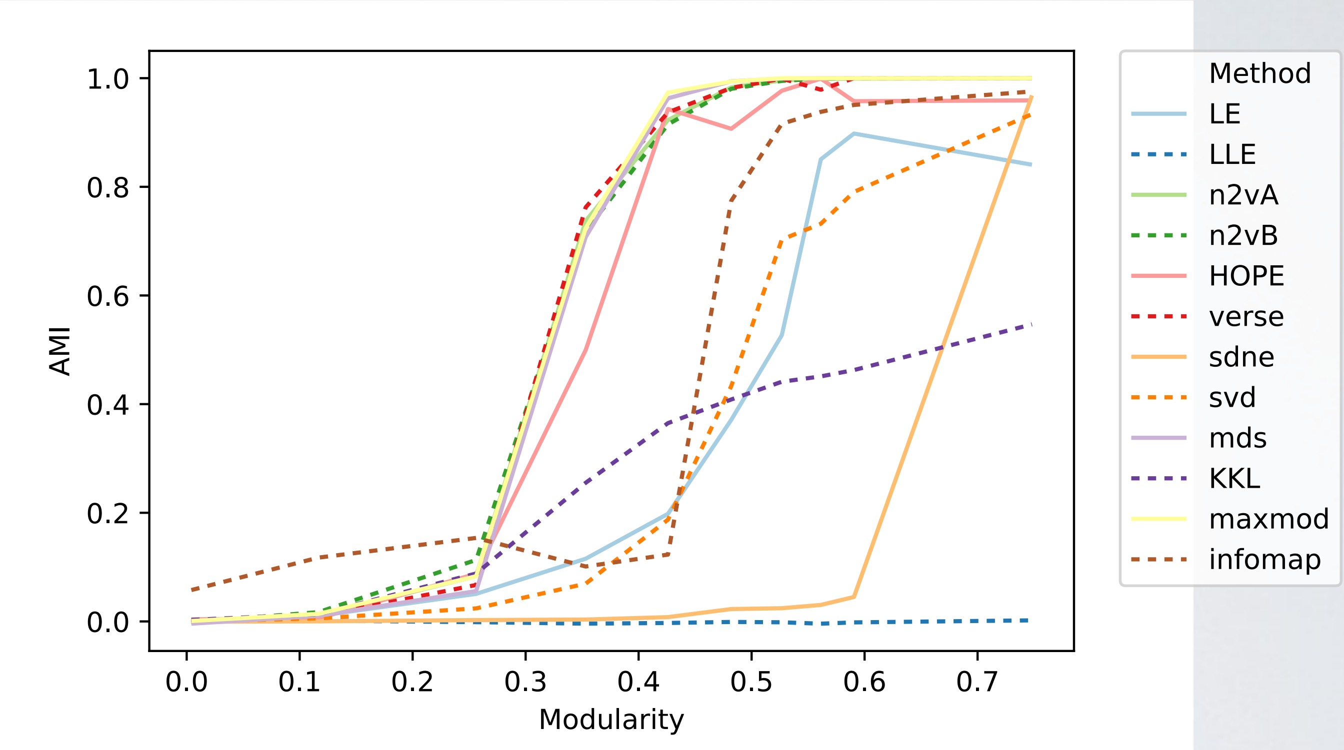
- 1)Create a network with a community structure
- 2)Use k-means clustering on embedding to detect the community structure
- 3)Compare expected to k-means using the aNMI

COMMUNITY STRUCTURE

Planted partitions. 8 communities



(a) Embedding in 2 dimensions



(b) Embedding in 128 dimensions

Node2vec, VERSE, HOPE => Good results in “high” dimensions

COMMUNITY STRUCTURE

- Note: If:
  - we know the number of clusters to find
  - And we can use a large number of dimensions
- =>Embeddings can be better than traditional algorithms

NODE CLASSIFICATION WITH EMBEDDINGS

NODE CLASSIFICATION

- To each node is associated a vector in the embedding
  - This vector corresponds to topological features of the node, used instead of, for instance, centralities
  - Both types of features can be combined
- As usual, a classifier can be trained using those features

NODE CLASSIFICATION

Algorithm	**Dataset**		
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p,q)	0.25, 0.25	4, 1	4, 0.5
Gain of *node2vec* [%]	**22.3**	**1.3**	**21.8**

Some controversies (very recent results)

Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864). ACM.

LINK PREDICTION WITH EMBEDDINGS

Sinha, A., Cazabet, R., & Vaudaine, R. (2018, December). Systematic Biases in Link Prediction: comparing heuristic and graph embedding based methods. In *International Conference on Complex Networks and their Applications* (pp. 81-93). Springer, Cham.

UNSUPERVISED LINK PREDICTION

- Unsupervised link prediction **from embeddings**
- =>Compute the distance between nodes in the embedding
- =>Use it as a similarity score

SUPERVISED LINK PREDICTION

- Supervised link prediction **from embeddings**
- =>embeddings provide features for nodes (nb features: dimensions)
  - Combine nodes features to obtain edge features
- =>Train a classifier to predict edges based on features from the embedding

SUPERVISED LINK PREDICTION

Operator	Result
Average	$(\mathbf{a} + \mathbf{b})/2$
Concat	$[\mathbf{a}_1, \ldots, \mathbf{a}_d, \mathbf{b}_1, \ldots, \mathbf{b}_d]$
Hadamard	$[\mathbf{a}_1 * \mathbf{b}_1, \ldots, \mathbf{a}_d * \mathbf{b}_d]$
Weighted L1	$[\lvert\mathbf{a}_1 - \mathbf{b}_1\rvert, \ldots, \lvert\mathbf{a}_d - \mathbf{b}_d\rvert]$
Weighted L2	$[(\mathbf{a}_1 - \mathbf{b}_1)^2, \ldots, (\mathbf{a}_d - \mathbf{b}_d)^2]$

Combining nodes vectors into edge vectors

SUPERVISED LINK PREDICTION

- How well does it works ?
- According to recent articles
  - Node2vec (2016)
  - VERSE (2018)
- =>These methods are better than the state of the art

Op	**Algorithm**	**Dataset**		
		Facebook	PPI	arXiv
	Common Neighbors	0.8100	0.7142	0.8153
	Jaccard's Coefficient	0.8880	0.7018	0.8067
	Adamic-Adar	0.8289	0.7126	0.8315
	Pref. Attachment	0.7137	0.6670	0.6996
(a)	Spectral Clustering	0.5960	0.6588	0.5812
	DeepWalk	0.7238	0.6923	0.7066
	LINE	0.7029	0.6330	0.6516
	node2vec	0.7266	0.7543	0.7221
(b)	Spectral Clustering	0.6192	0.4920	0.5740
	DeepWalk	**0.9680**	0.7441	0.9340
	LINE	0.9490	0.7249	0.8902
	node2vec	**0.9680**	**0.7719**	**0.9366**
(c)	Spectral Clustering	0.7200	0.6356	0.7099
	DeepWalk	0.9574	0.6026	0.8282
	LINE	0.9483	0.7024	0.8809
	node2vec	0.9602	0.6292	0.8468
(d)	Spectral Clustering	0.7107	0.6026	0.6765
	DeepWalk	0.9584	0.6118	0.8305
	LINE	0.9460	0.7106	0.8862
	node2vec	0.9606	0.6236	0.8477

(a) Average, (b) Hadamard, (c) Weighted-L1, and (d) Weighted-L2

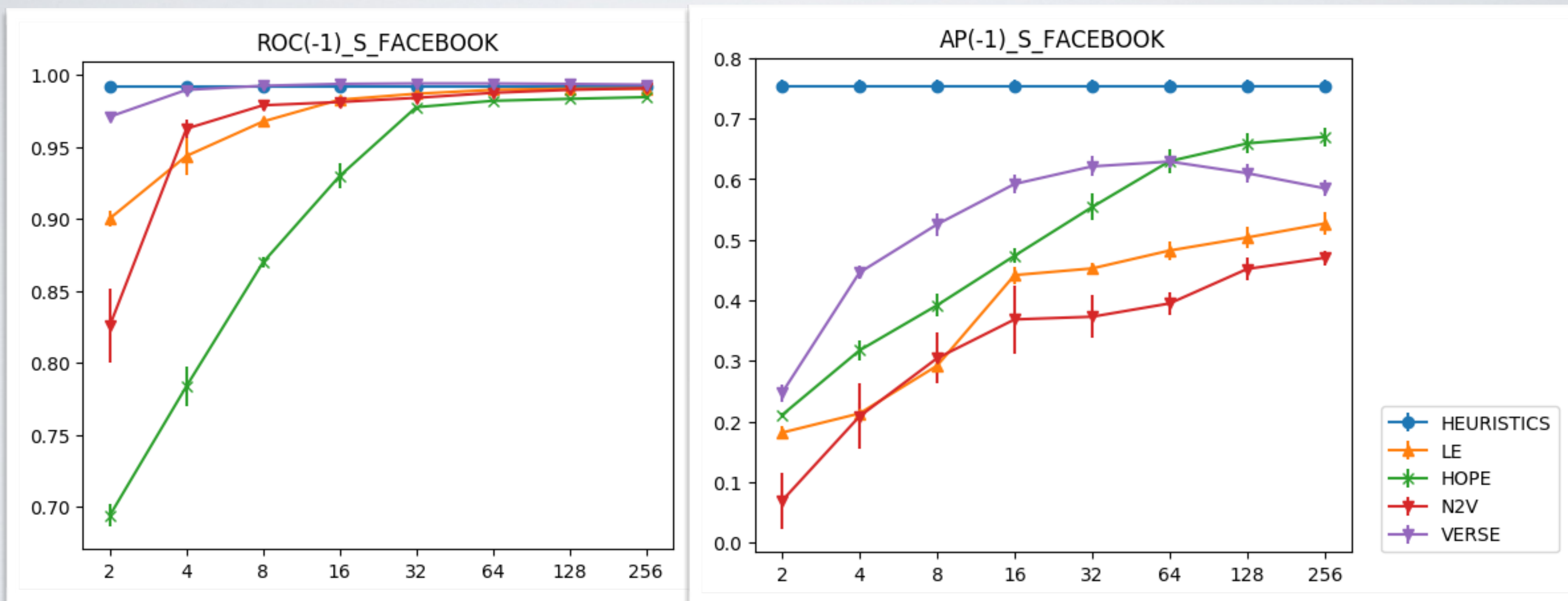
(AUC)

LINK PREDICTION

- Our tests: not really
- Embeddings are better only if we use some particular tests settings
    - Accuracy score on balanced test sets (WRONG)
    - Supervised LP for embeddings compared with unsupervised heuristics

Sinha, A., Cazabet, R., & Vaudaine, R. (2018, December). Systematic Biases in Link Prediction: comparing heuristic and graph embedding based methods. In *International Conference on Complex Networks and their Applications* (pp. 81-93). Springer, Cham.

LINK PREDICTION



LINK PREDICTION

- Possible explanations:
  - *Cherry picking* in original articles
  - Implementation biases (some methods hard to reproduce)
  - Hyper-parameter tuning (hard to do, might lead to overfit if incorrectly done)
- Despite controversies, very interesting research question

Sinha, A., Cazabet, R., & Vaudaine, R. (2018, December). Systematic Biases in Link Prediction: comparing heuristic and graph embedding based methods. In *International Conference on Complex Networks and their Applications* (pp. 81-93). Springer, Cham.

GRAPH CONVOLUTIONAL NETWORKS

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Zhang, Z., Cui, P., & Zhu, W. (2018). Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*.

Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

(DEEP) NEURAL NETWORKS

A deep neural networks can be seen as the chaining of multiple simple machine learning models (e.g., logistic classifier).
The output of a model is the input of the other, all weights optimized simultaneously (backpropagation)

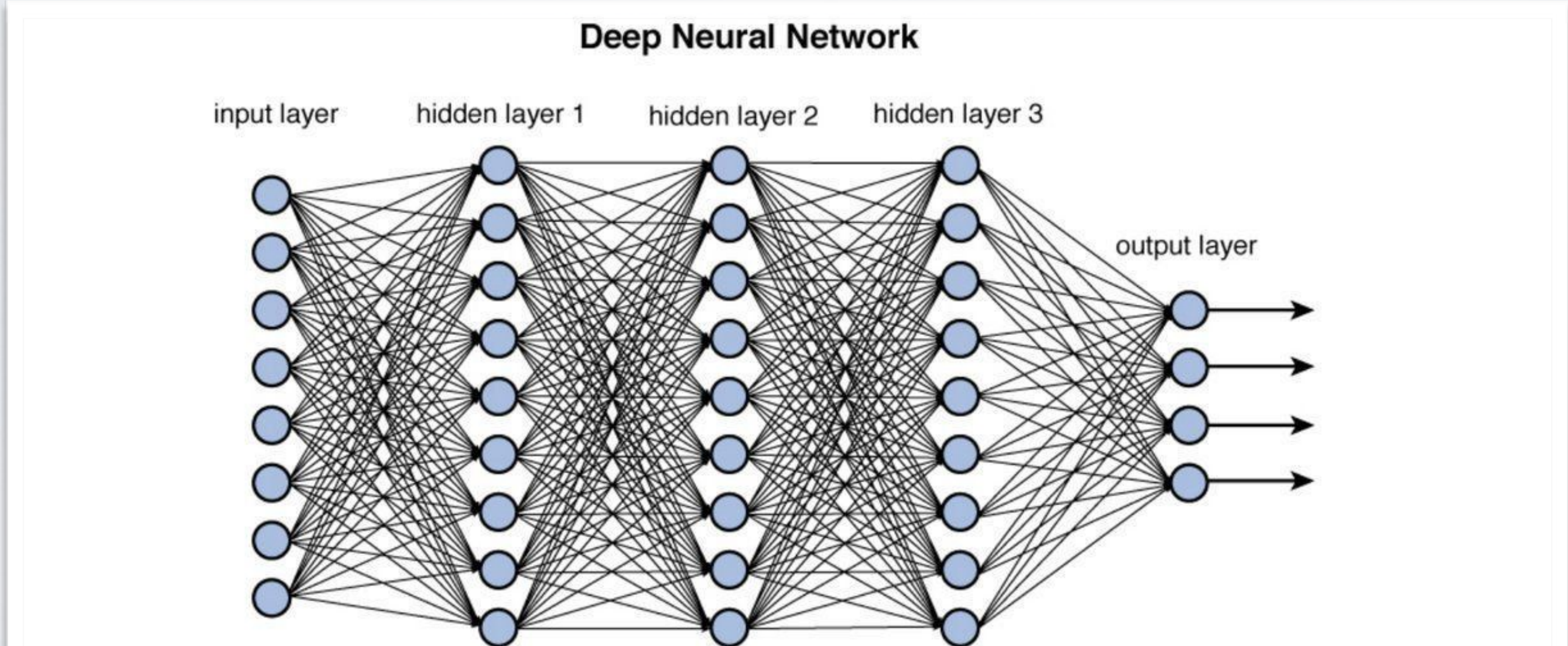


Figure 12.2 Deep network architecture with multiple layers.

https://medium.com/tebs-lab/introduction-to-deep-learning-a46e92cb0022

https://en.wikipedia.org/wiki/Backpropagation

CONVOLUTIONAL NEURAL NETWORK

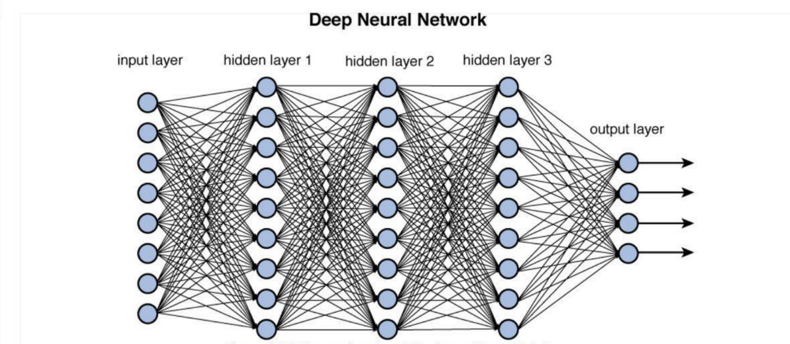
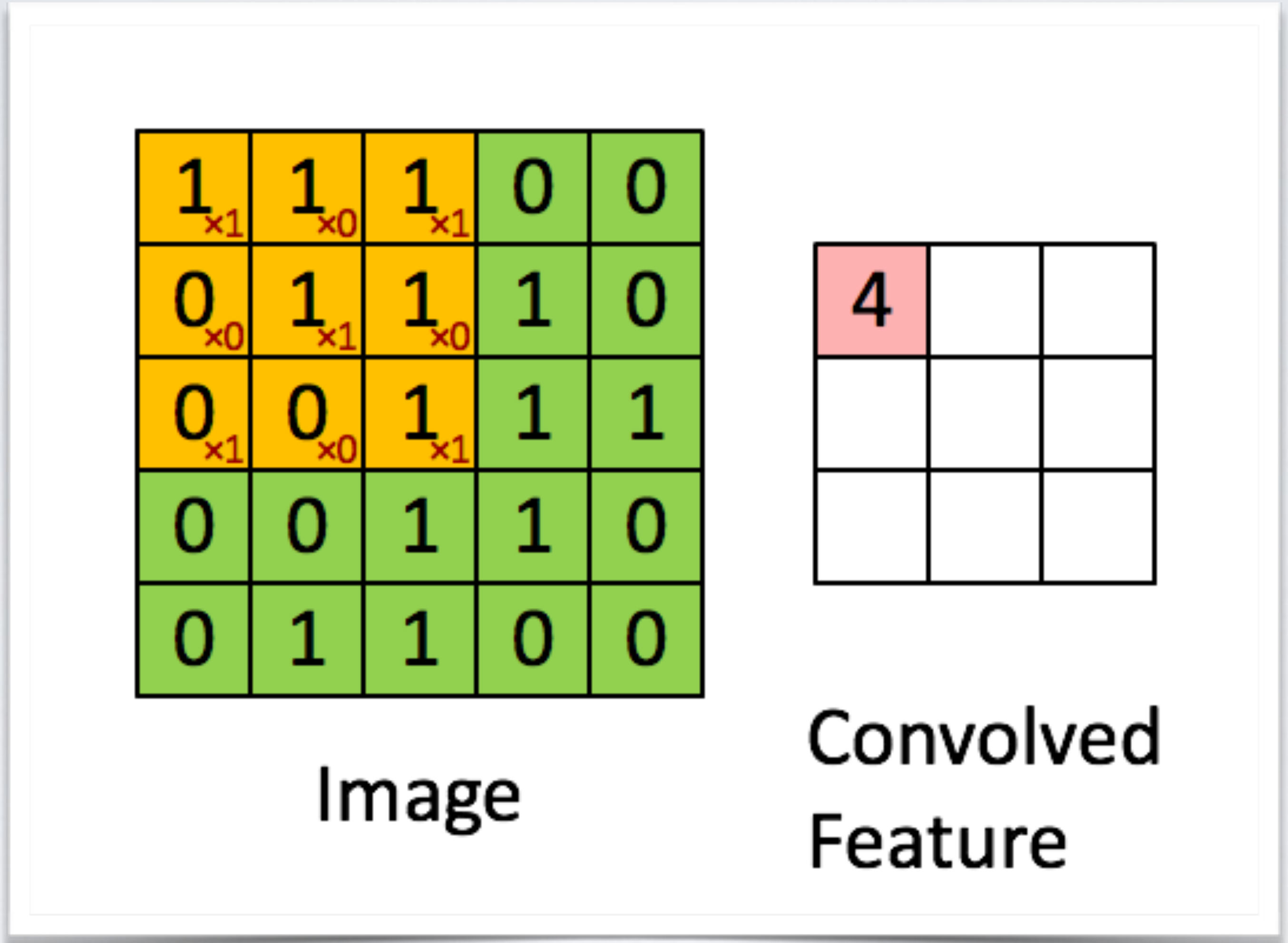


Figure 12.2 Deep network architecture with multiple layers.

- All outputs of a layer connected to all inputs of the next is called **fully connected** layer
  - Learned weights will "cut" some edges (zero weights)
- In input data is structured, one can already use this structure
- **Convolutions** were introduced to work with pictures
  - Adjacency in pixels is meaningful

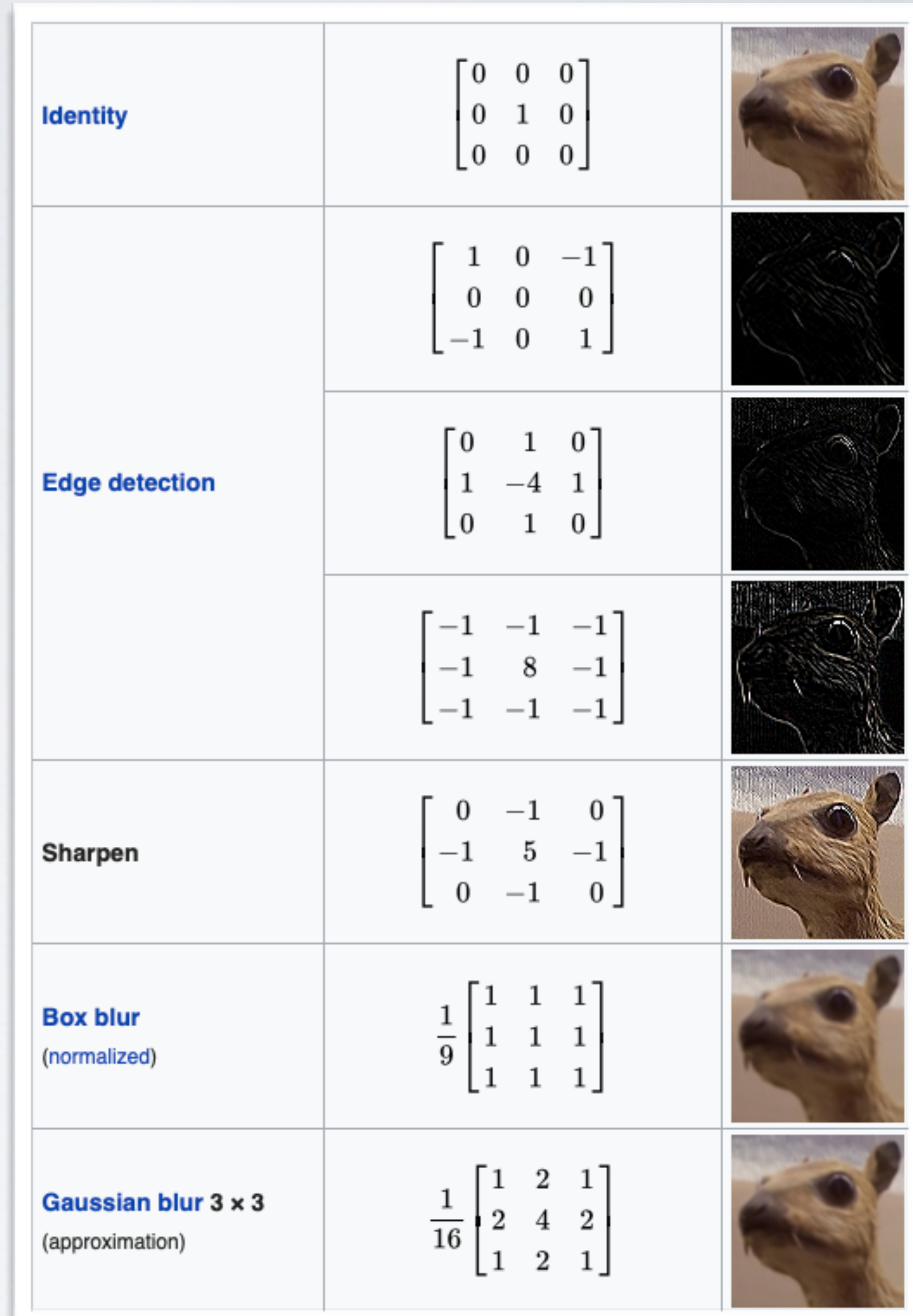
CONVOLUTION



- Extract "features" of "higher level"
  - Pixels => lines, curves, dots => circles, long lines, curvy shapes => eye, hand, leaves => Animal, Car, sky …

CONVOLUTION

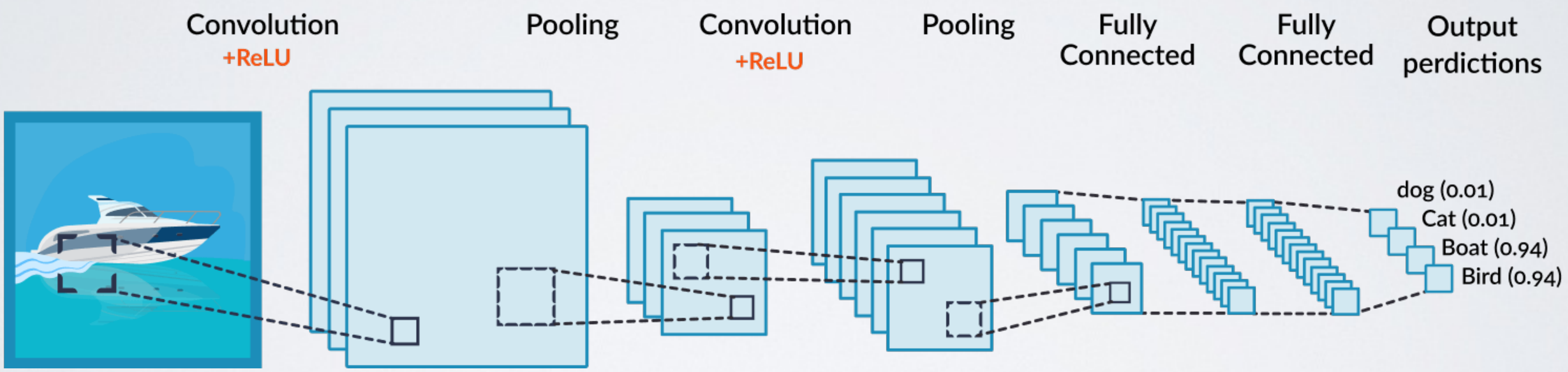
- A convolution is defined by the weights of its kernel
- Which kernel(s) should we use?
- Weights of the kernel can be learnt, too



Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

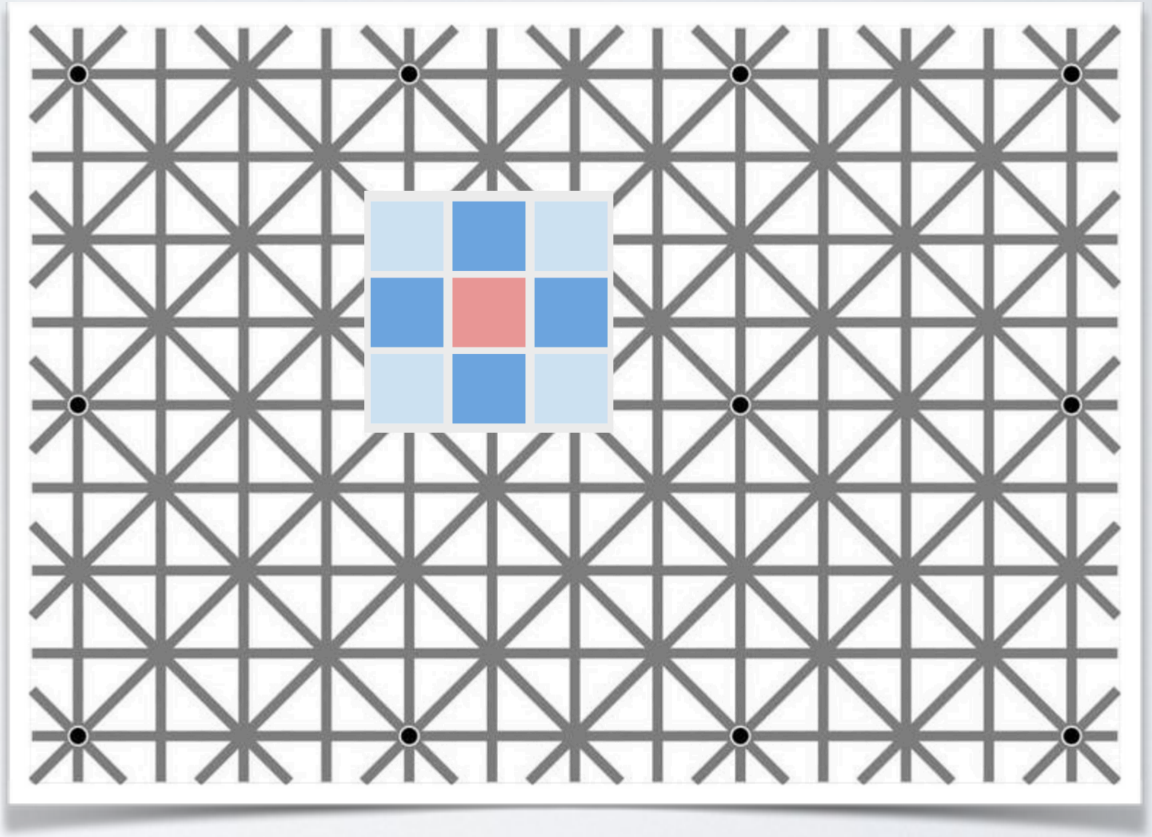
https://en.wikipedia.org/wiki/Kernel_(image_processing)

CONVOLUTIONAL NEURAL NETWORK



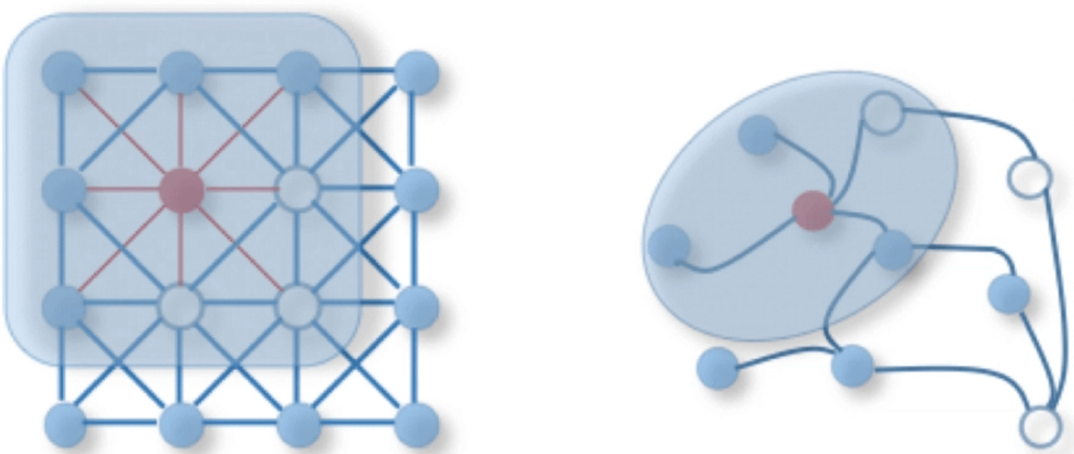
CONVOLUTIONAL NEURAL NETWORK

- Convolution on a picture can be seen as a special case of a graph operation:
  - Combine weights of neighboors
  - With an image represented as a regular grid
- Define convolutions on networks



https://www.inference.vc/how-powerful-are-graph-convolutions-review-of-kipf-welling-2016-2/

GRAPH CONVOLUTION

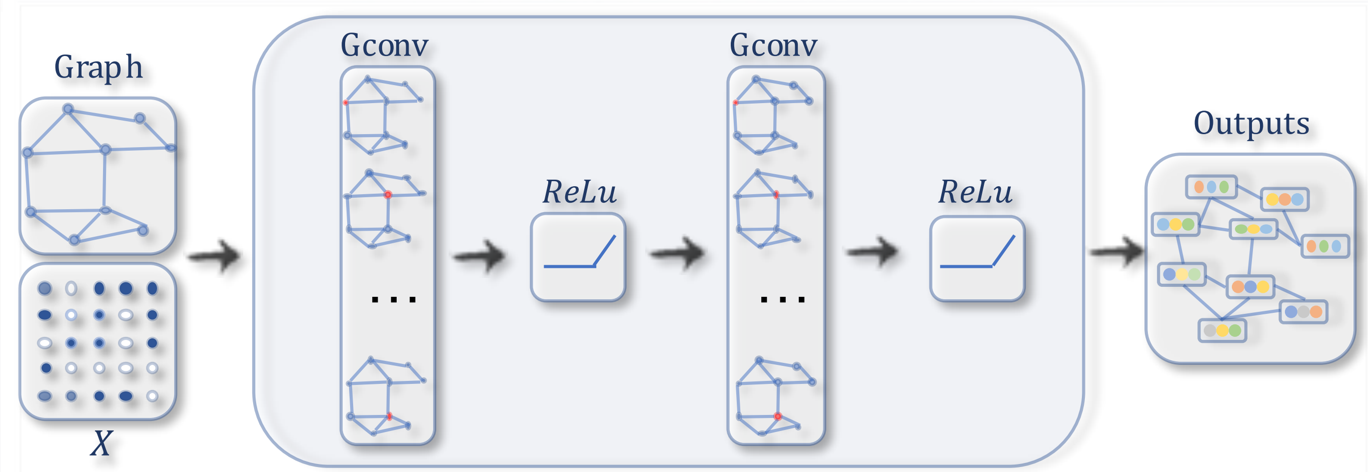


(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Fig. 1: 2D Convolution vs. Graph Convolution.

Stacking convolution layers



Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

GRAPH CONVOLUTION

$$H^{(l+1)} = f(H^{(l)}, A)$$

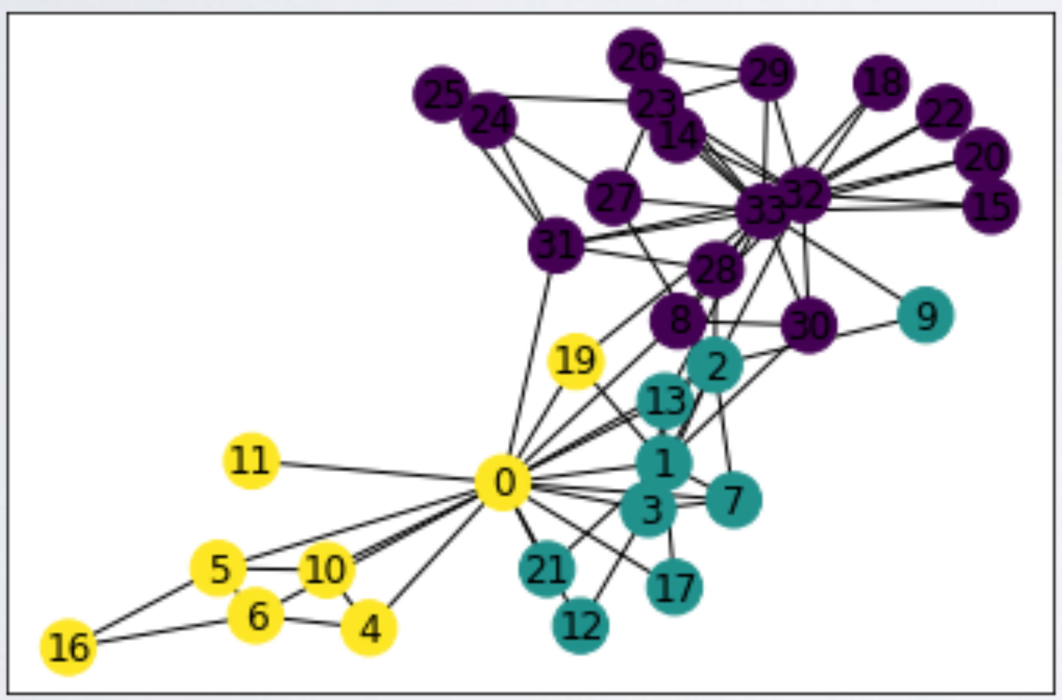
$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

H: node features
A: adjacency matrix ($\hat{A} = A + I$)
l: layer index
D: Degree matrix (degrees on the diagonal)
W: learnable weights
σ: activation fonction (often ReLU)

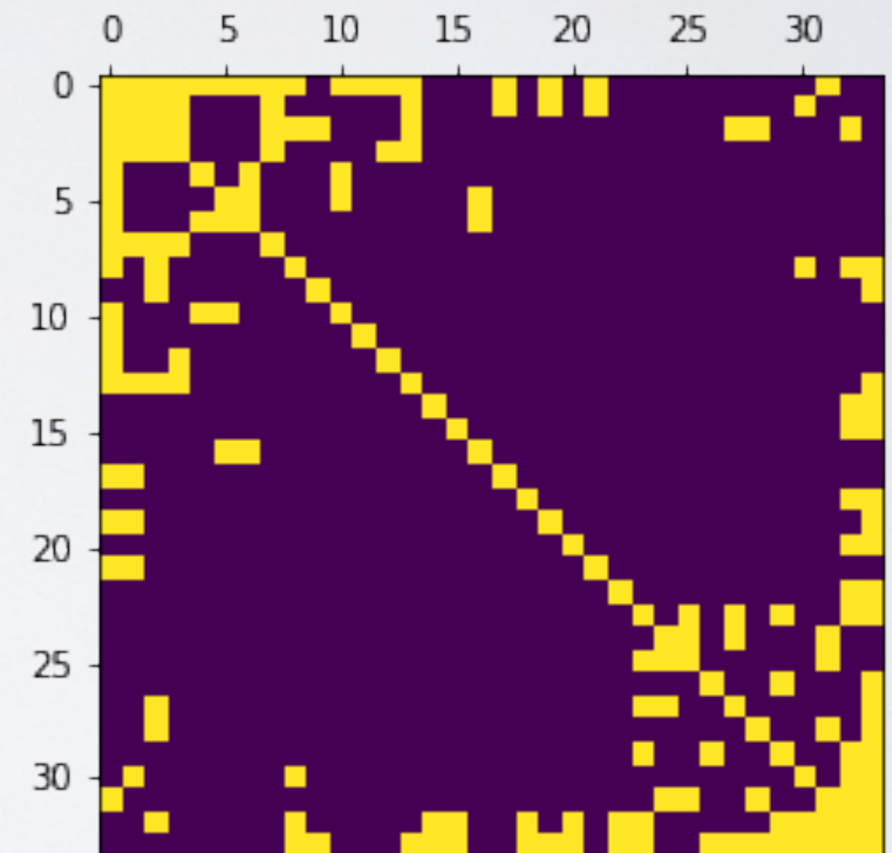
Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

GRAPH CONVOLUTION

- Going through an example of the typical GCN

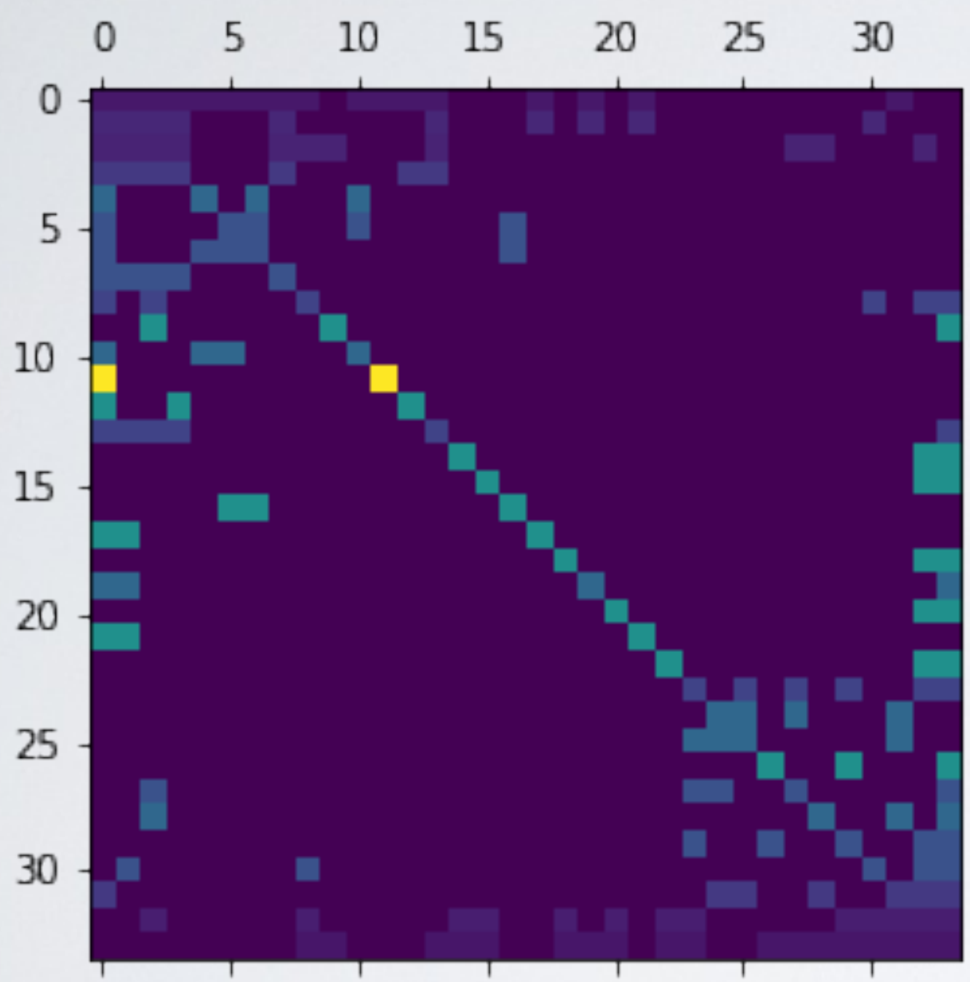


Zackary Karate club
(with communities for reference)



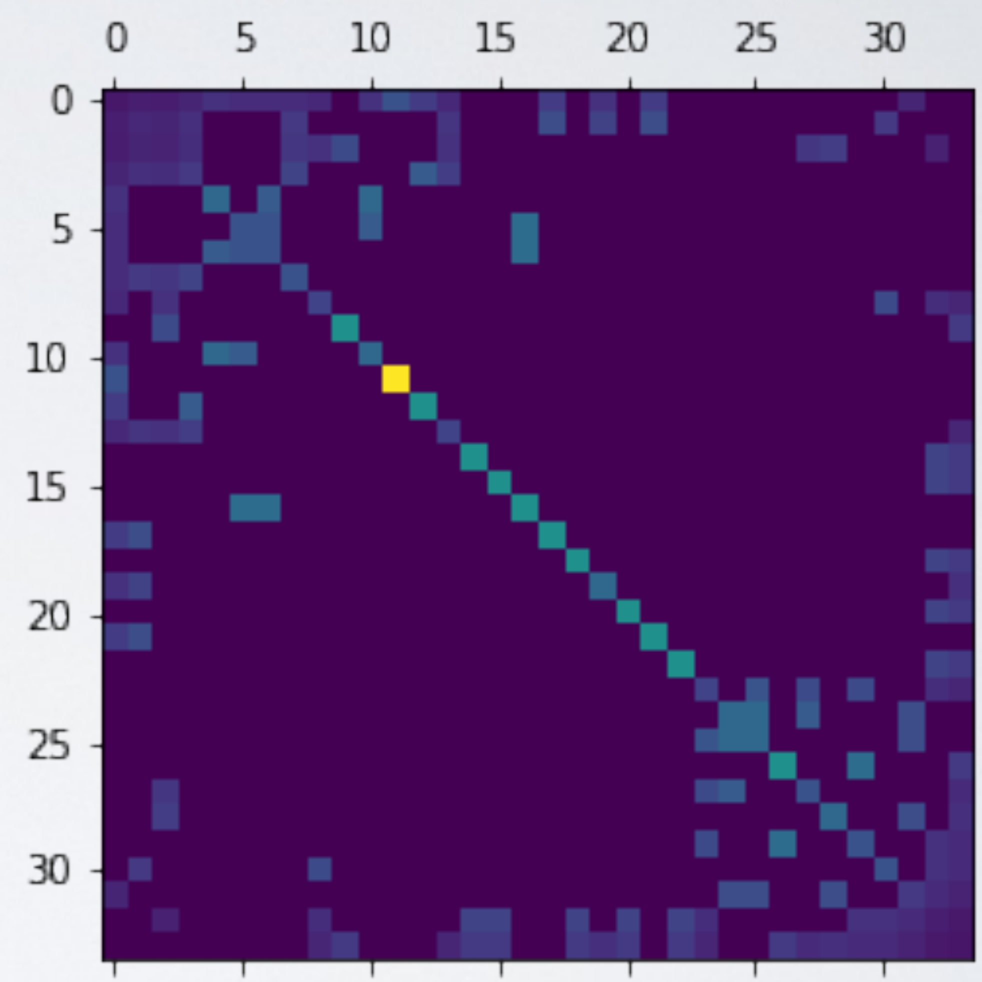
$\hat{A}$

GRAPH CONVOLUTION



$D^{-1}\hat{A}$

Simple average



$D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}$

Weighted average

Normalisation of the adjacency matrix

GRAPH CONVOLUTION

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

$$D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}H$$

Features of the nodes become the (weighted) average of the features of the neighbors

---

W has shape $(X \times Y)$, with X the number of features in input and Y the **desired** number of features in output

GRAPH CONVOLUTION

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Size of the weight matrices by layer

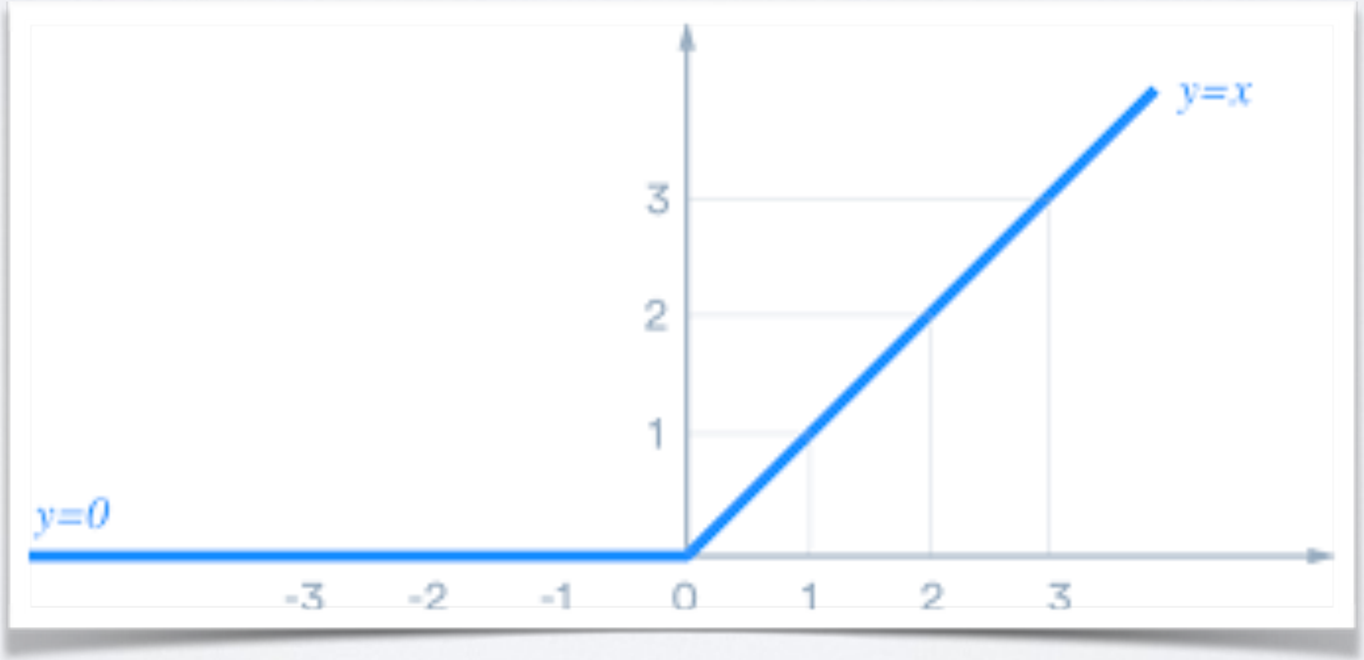
$$W_0 : d_0 \times d_1$$
$$W_1 : d_1 \times d_2$$
$$\dots$$
$$W_n : d_n \times d_{n+1}$$

d_0 is the number of features per node in the original network data, d_{n+1} is the number of desired features (usually followed by a normal classifier, e.g., logistic)

GRAPH CONVOLUTION

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

σ is called an activation function.
It is used to introduce non-linearity.
As of 2019, the most common choice is to use the **ReLU**,
(Rectified Linear Unit)
=>Simple to differentiate and to compute



https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

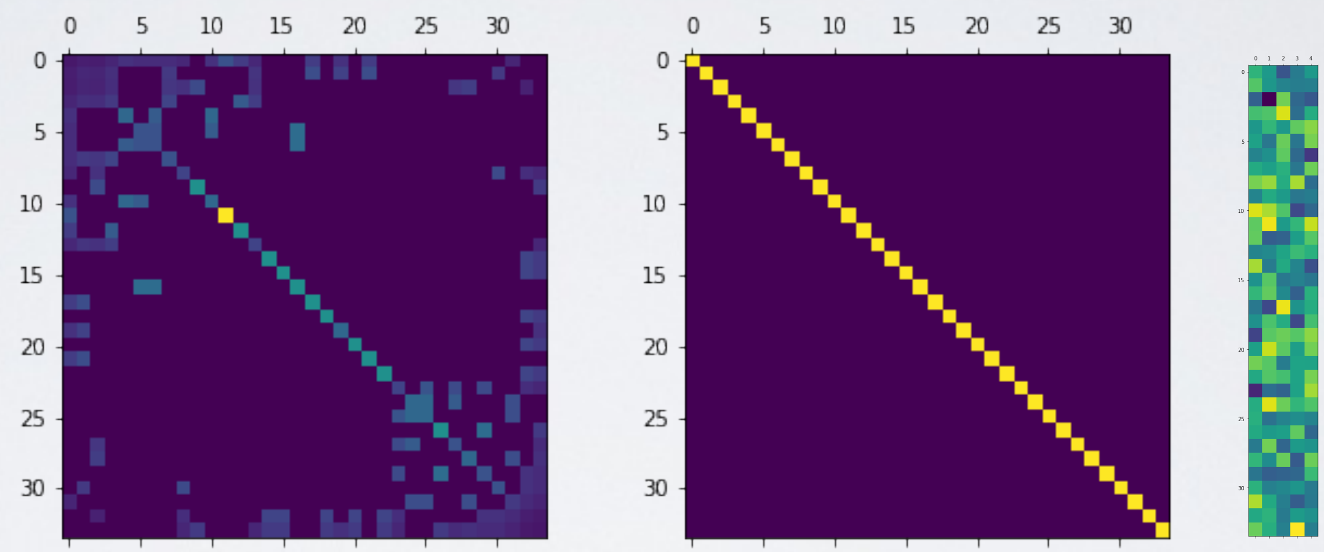
FORWARD STEP

- We can first look at what happens **without weight learning**, i.e., doing only the forward step.
- We set the original features to the identity matrix, $H_0 = I$. Each node's features is a *one hot vector* of itself (1 at its position, 0 otherwise)
- Weights are random (normal distribution centered on 0)
- Two layers, with W sizes $n \times 5, 5 \times 2$

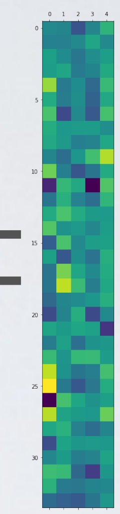
FORWARD STEP

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

L1 = n to 5 features

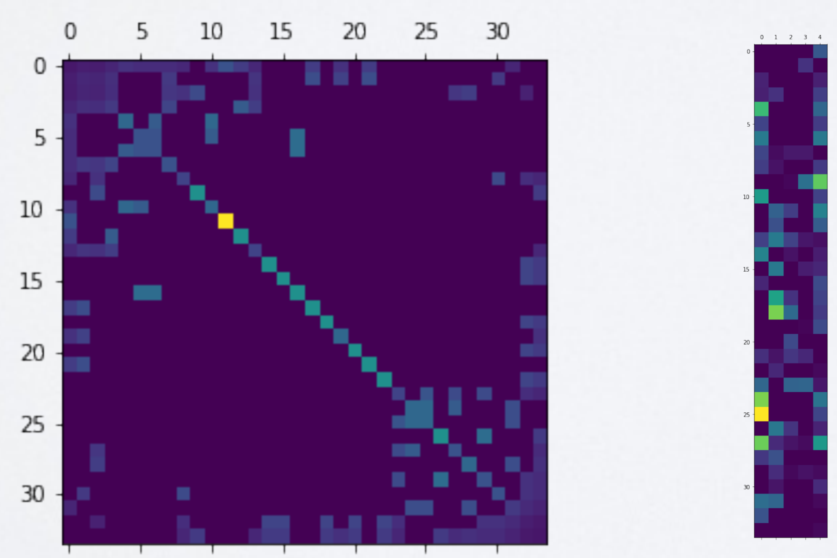


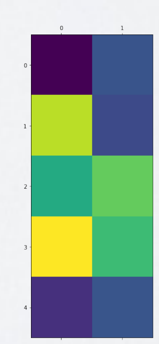
=



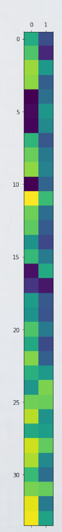
σ=>

L1 = 5 to 2 features



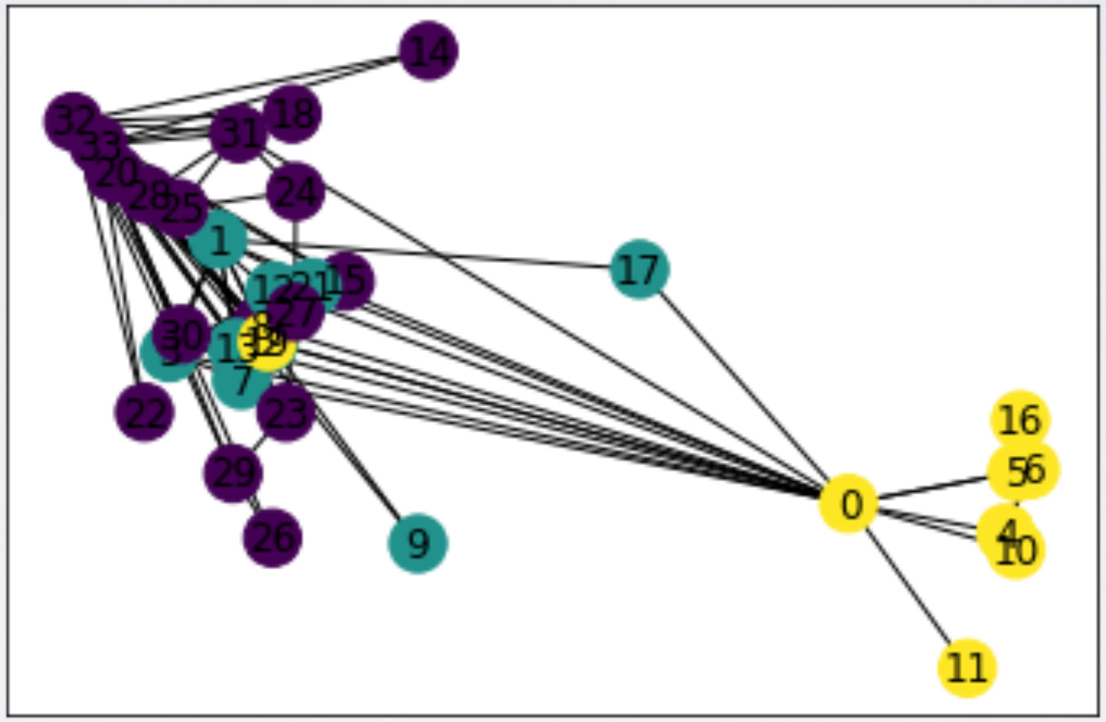


=



FORWARD STEP

Dimension 2

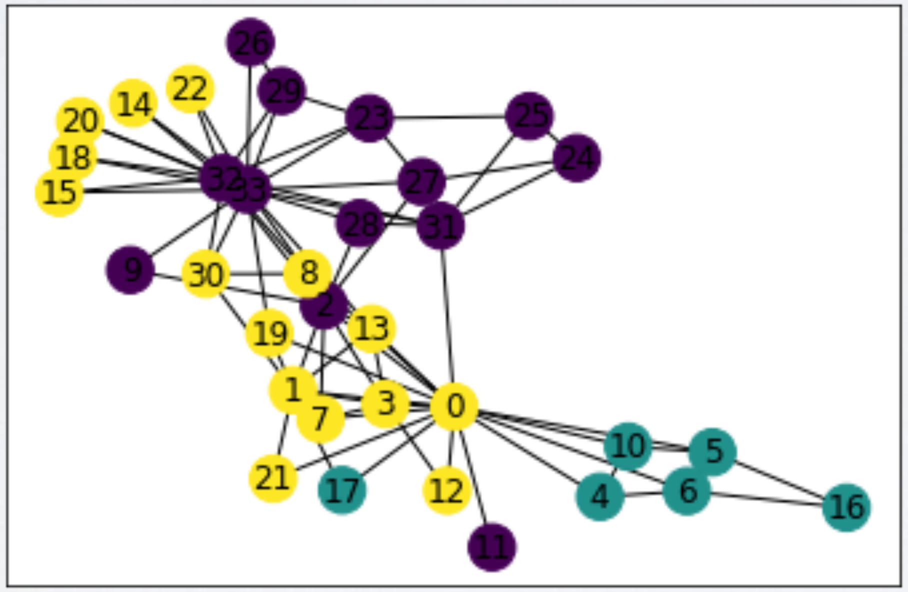


Dimension 1

Even with random weights, some structure is preserved in the "embedding"

FORWARD STEP

K-means on the 2D "embedding"
(paramater k=3 clusters)



(Node positions based on spring layout)

BACKWARD STEP

- To learn the weights, we use a mechanism called **back-propagation**
- Short summary
  - A **loss** function is defined to compare the "predicted values" with ground truth labels (at this point, we need some labels…)
    - Typically, log-likelihood
  - The **derivative** of the cost function relative to weights is computed
  - Weights are updated using **grading descent** (i.e., weights are modified in the direction that will minimize the loss)

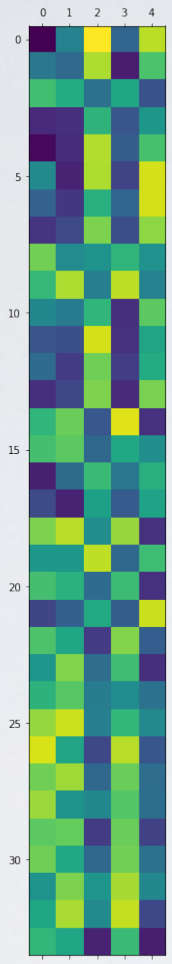
https://en.wikipedia.org/wiki/Backpropagation

FITTING THE GCN

- We define the same GCN as before
- We define a "semi-supervised" process:
  - Labels are known only for a few nodes (the 2 instructors)
  - The loss is computed only for them
- We run e steps ("epoch") of back-propagation, until convergence

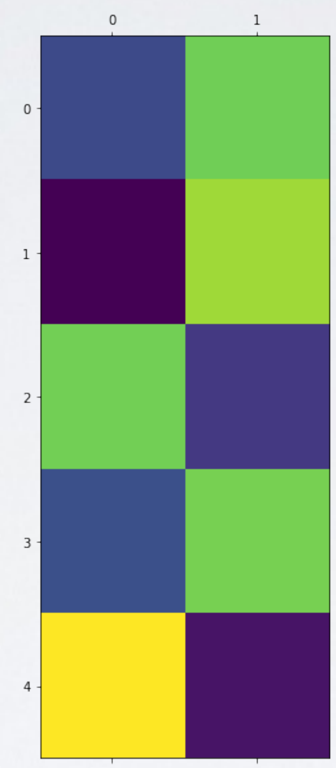
FITTING THE GCN

W_1



Step1:
Each node takes the average features of its neighbors.
W_1 can be seen as "computed" features
(this is because we used I as original features)

W_2



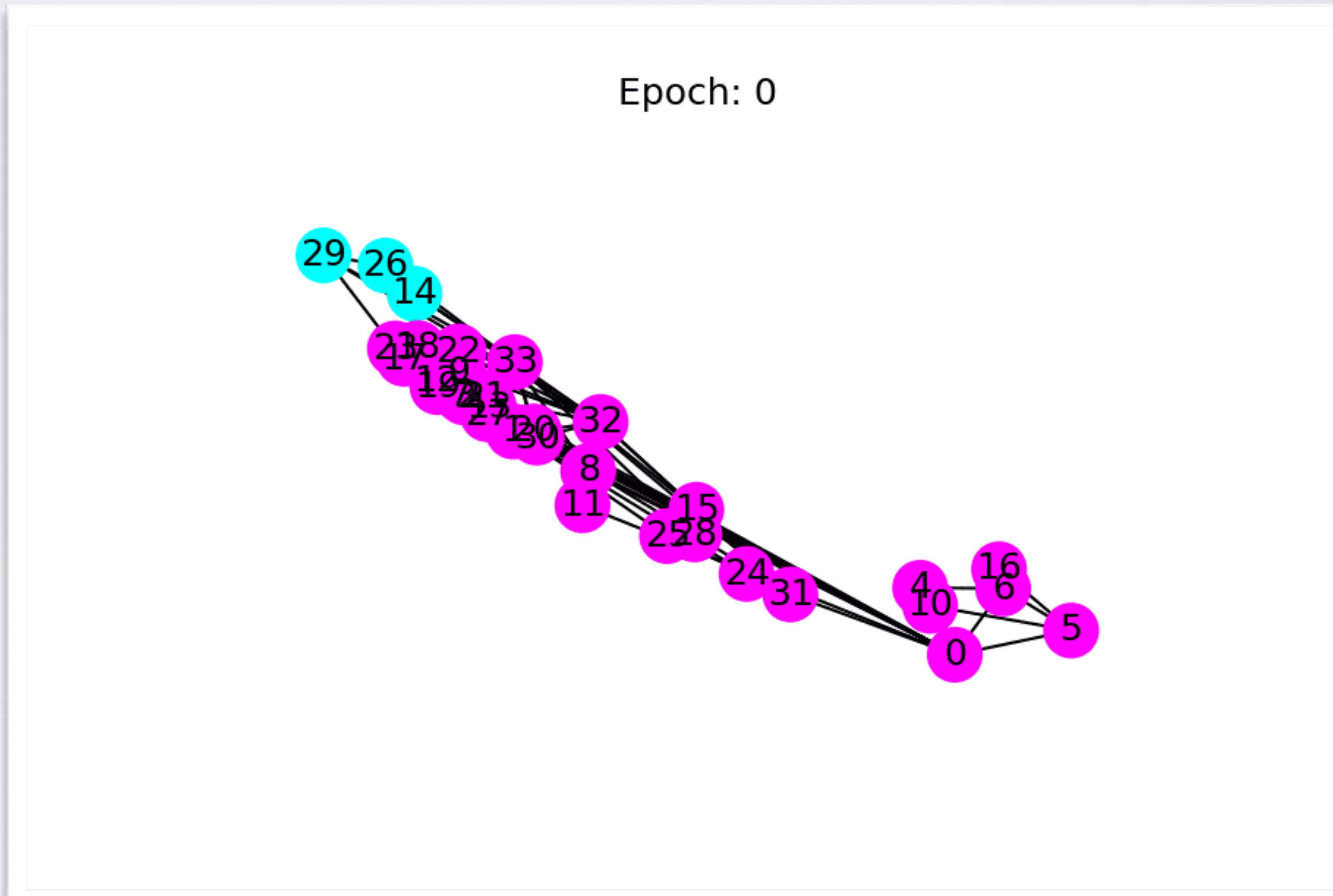
Step2:
After averaging over results of step1 (AH),
each node combines its aggregated features according to this matrix

H

Result:
This is the computed feature vector.
As expected, values for nodes 0 and 33 are opposed

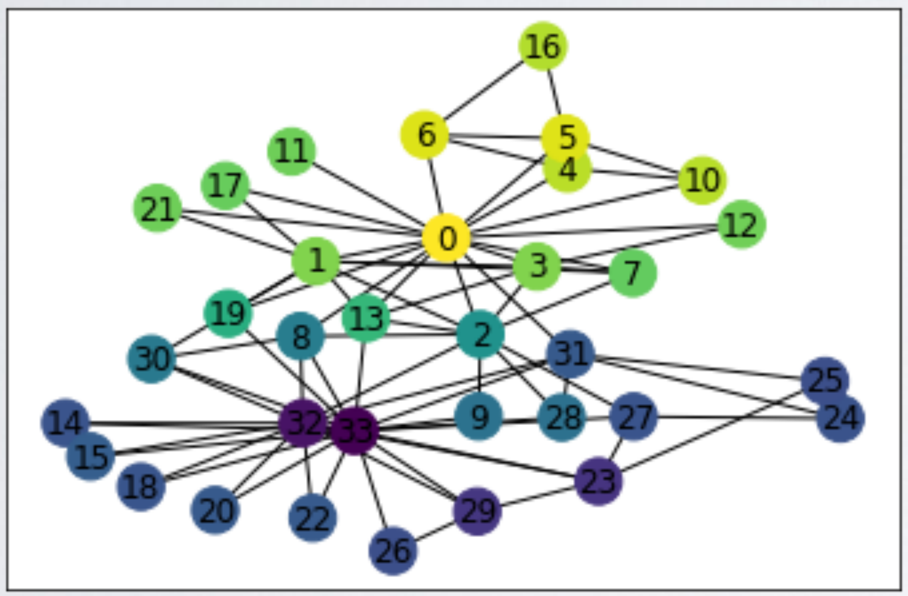
FITTING THE GCN

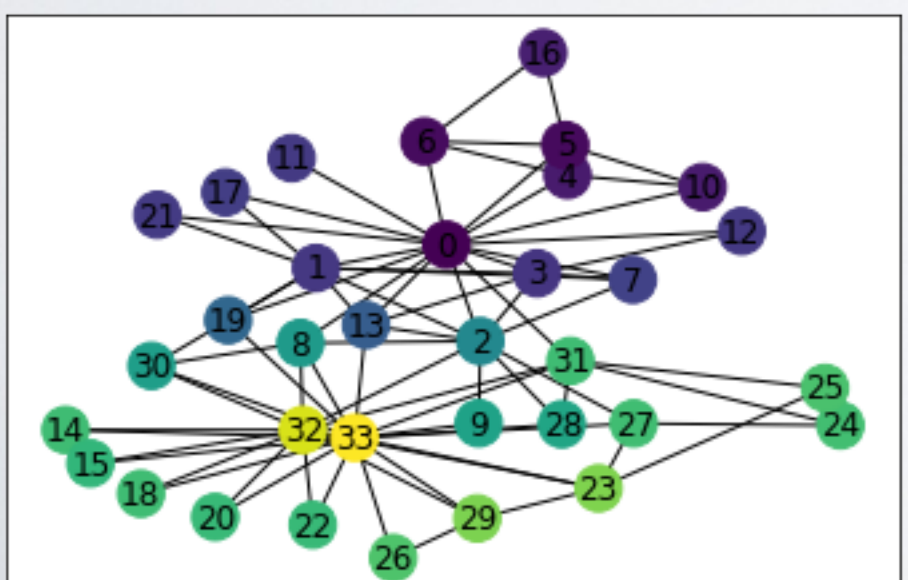
```
Epoch 0 | Loss: 0.6987
Epoch 1 | Loss: 0.6804
Epoch 2 | Loss: 0.6634
Epoch 3 | Loss: 0.6476
Epoch 4 | Loss: 0.6326
Epoch 5 | Loss: 0.6174
Epoch 6 | Loss: 0.6017
Epoch 7 | Loss: 0.5852
Epoch 8 | Loss: 0.5684
Epoch 9 | Loss: 0.5513
Epoch 10 | Loss: 0.5338
Epoch 11 | Loss: 0.5158
Epoch 12 | Loss: 0.4976
Epoch 13 | Loss: 0.4792
Epoch 14 | Loss: 0.4605
Epoch 15 | Loss: 0.4416
Epoch 16 | Loss: 0.4225
Epoch 17 | Loss: 0.4033
Epoch 18 | Loss: 0.3842
Epoch 19 | Loss: 0.3652
Epoch 20 | Loss: 0.3464
Epoch 21 | Loss: 0.3279
Epoch 22 | Loss: 0.3096
Epoch 23 | Loss: 0.2916
Epoch 24 | Loss: 0.2741
Epoch 25 | Loss: 0.2571
Epoch 26 | Loss: 0.2407
Epoch 27 | Loss: 0.2248
Epoch 28 | Loss: 0.2095
Epoch 29 | Loss: 0.1946
Epoch 30 | Loss: 0.1803
Epoch 31 | Loss: 0.1668
Epoch 32 | Loss: 0.1541
Epoch 33 | Loss: 0.1422
Epoch 34 | Loss: 0.1312
Epoch 35 | Loss: 0.1209
Epoch 36 | Loss: 0.1113
Epoch 37 | Loss: 0.1024
Epoch 38 | Loss: 0.0940
Epoch 39 | Loss: 0.0863
Epoch 40 | Loss: 0.0793
Epoch 41 | Loss: 0.0727
Epoch 42 | Loss: 0.0667
Epoch 43 | Loss: 0.0611
Epoch 44 | Loss: 0.0560
Epoch 45 | Loss: 0.0513
Epoch 46 | Loss: 0.0470
Epoch 47 | Loss: 0.0432
Epoch 48 | Loss: 0.0396
Epoch 49 | Loss: 0.0363
Epoch 50 | Loss: 0.0333
```



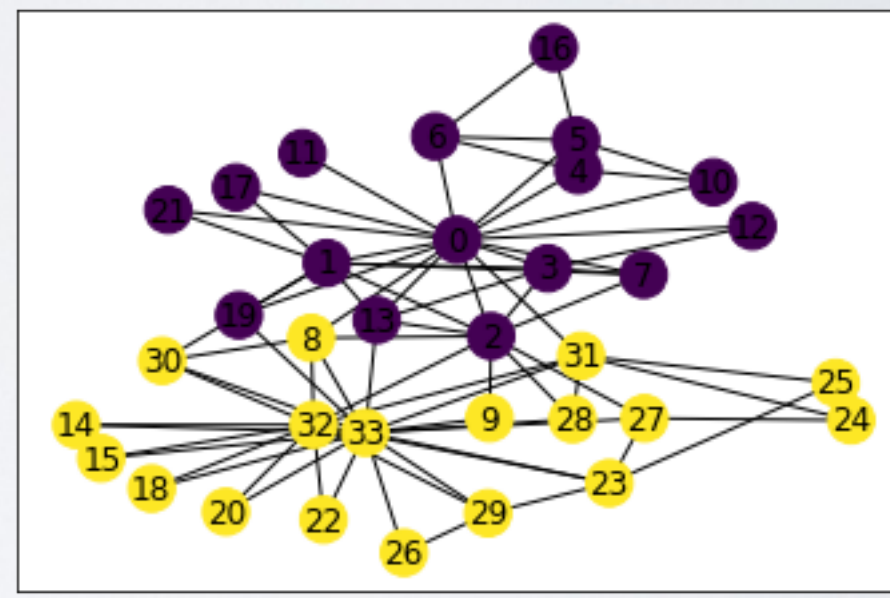
RESULTS

Features values





Highest feature as label



We retrieve the expected "communities"

GCN LITERATURE

- Results are claimed to be above the state of the art
  - Controversies, which is normal for such recent methods

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	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, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

TO CONCLUDE

Many variations proposed already

Very active since 2017

Spawned renewed interest in networks in the ML literature

Hard to predict the future of these techniques.

Approach	Category	Inputs	Pooling	Readout	Time Complexity
GNN* (2009) [15]	RecGNN	A, X, X^e	-	a dummy super node	-
GraphESN (2010) [16]	RecGNN	A, X	-	mean	-
GGNN (2015) [17]	RecGNN	A, X	-	attention sum	-
SSE (2018) [18]	RecGNN	A, X	-	-	-
Spectral CNN (2014) [19]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling	max	$O(n^3)$
Henaff et al. (2015) [20]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling	[illegible]	$O(n^3)$
ChebNet (2016) [21]	Spectral-based ConvGNN	A, X	efficient pooling	sum	$O(m)$
GCN (2017) [22]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
CayleyNet (2017) [23]	Spectral-based ConvGNN	A, X	mean/graclus pooling	-	$O(m)$
AGCN (2018) [40]	Spectral-based ConvGNN	A, X	max pooling	sum	$O(n^2)$
DualGCN (2018) [41]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
NN4G (2009) [24]	Spatial-based ConvGNN	A, X	-	sum/mean	$O(m)$
DCNN (2016) [25]	Spatial-based ConvGNN	A, X	-	mean	$O(n^2)$
PATCHY-SAN (2016) [26]	Spatial-based ConvGNN	A, X, X^e	-	concat	-
MPNN (2017) [27]	Spatial-based ConvGNN	A, X, X^e	-	attention sum/ set2set	$O(m)$
GraphSage (2017) [42]	Spatial-based ConvGNN	A, X	-	-	-
GAT (2017) [43]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
MoNet (2017) [44]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
PGC-DGCNN (2018) [46]	Spatial-based ConvGNN	A, X	sort pooling	attention sum	$O(n^3)$
CGMM (2018) [47]	Spatial-based ConvGNN	A, X	-	concat	-
LGCN (2018) [45]	Spatial-based ConvGNN	A, X	-	-	-
GAAN (2018) [48]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
FastGCN (2018) [49]	Spatial-based ConvGNN	A, X	-	-	-
StoGCN (2018) [50]	Spatial-based ConvGNN	A, X	-	-	-
Huang et al. (2018) [51]	Spatial-based ConvGNN	A, X	-	-	-
DGCNN (2018) [52]	Spatial-based ConvGNN	A, X	sort pooling	-	$O(m)$
DiffPool (2018) [54]	Spatial-based ConvGNN	A, X	differential pooling	mean	$O(n^2)$
GeniePath (2019) [55]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
DGI (2019) [56]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
GIN (2019) [57]	Spatial-based ConvGNN	A, X	-	concat+sum	$O(m)$
ClusterGCN (2019) [58]	Spatial-based ConvGNN	A, X	-	-	-

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.