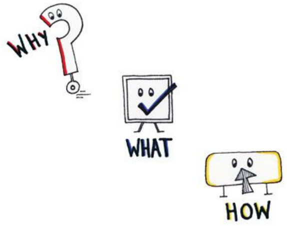
Graph Representation

Analysis of complex interconnected data

McGill
School of Computer Science

Quick Notes

- Third assignment is due on Oct 18th
  - Submit 2 files (report.pdf, code.zip) as a Group (pairs or two or individual) in Mycourses
- **Tue., Oct. 19, 2021:** Project Proposal Presentations
  - Why & What: Introduction and Motivation, Related Work, Problem Definition, Dataset Description
  - Writeup: 2 pages, due Oct 20th [8pt]
  - Presentation: 2 mins (2-3 slides), slides due Oct 18th [2pt]
  - Email the slides to the course email, use **Google Slides**
  - We will merge them all together, and you will go over it in cla
- Any questions?

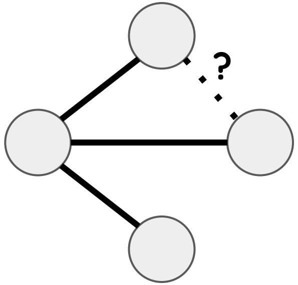


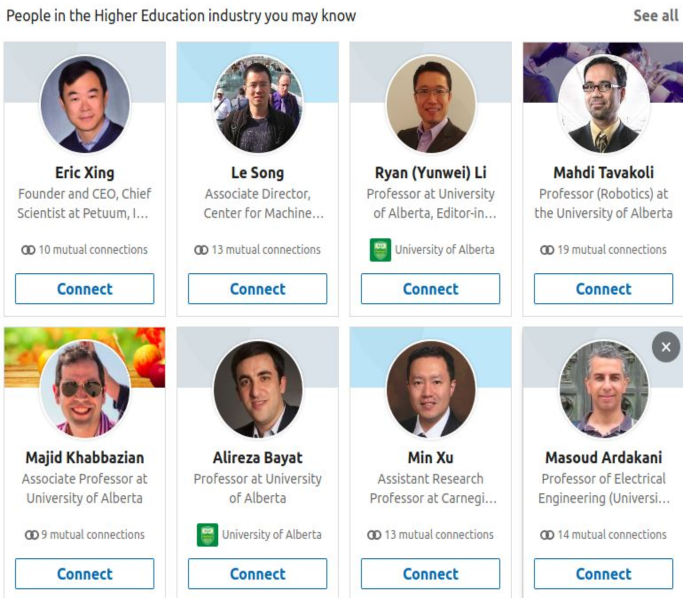
Deadlines

- assignment 1 due on Sep. 20th
- assignment 2 due on Oct. 4th
- assignment 3 due on Oct. 18th
- project proposal slides due on Oct. 18th
- project proposal due on Oct. 20th
- Reviews (first round) due on Oct. 27th
- project proposal slides due on Nov. 3rd
- project progress report due on Nov. 5th
- Reviews (second round) due on Nov. 12th
- project final report slides due on Nov. 29th
- project final report due on Dec. 7th
- Reviews (third round) due on Dec. 14th
- project revised report and rebuttal due on Dec. 20th
- note: dates are tentative, subject to change

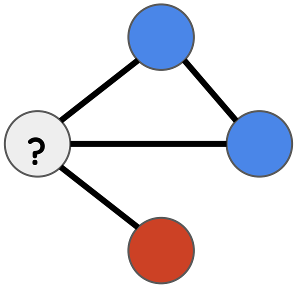
Common prediction tasks

- Link Prediction



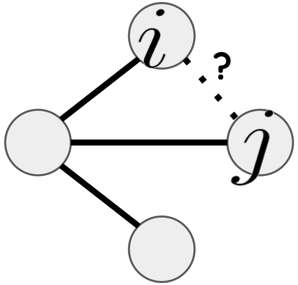


- Node Classification



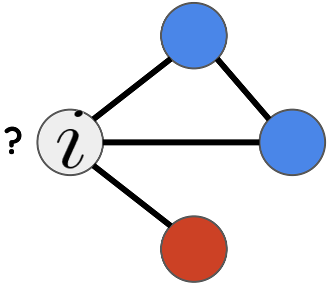
Common prediction tasks

- Link Prediction



$$z(i, j)$$

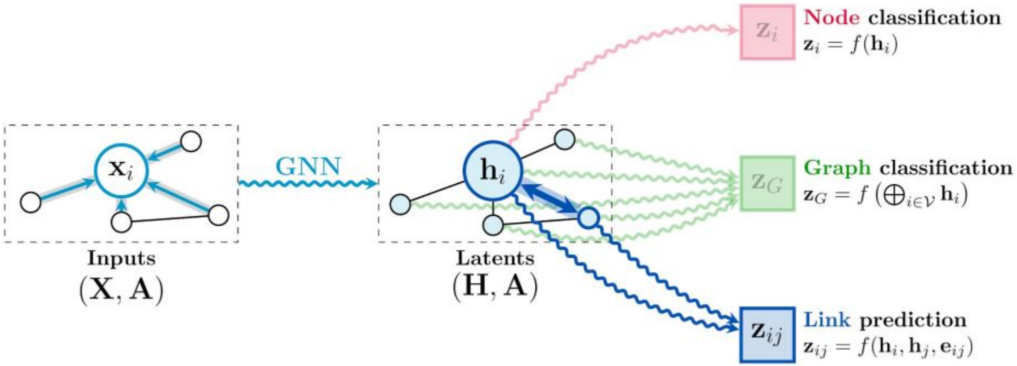
- Node Classification



$$z(i)$$

Graph Representation Learning

- Link Prediction: $z(i, j)$
- Node Classification: $z(i)$
- Graph Classification: $z(G)$



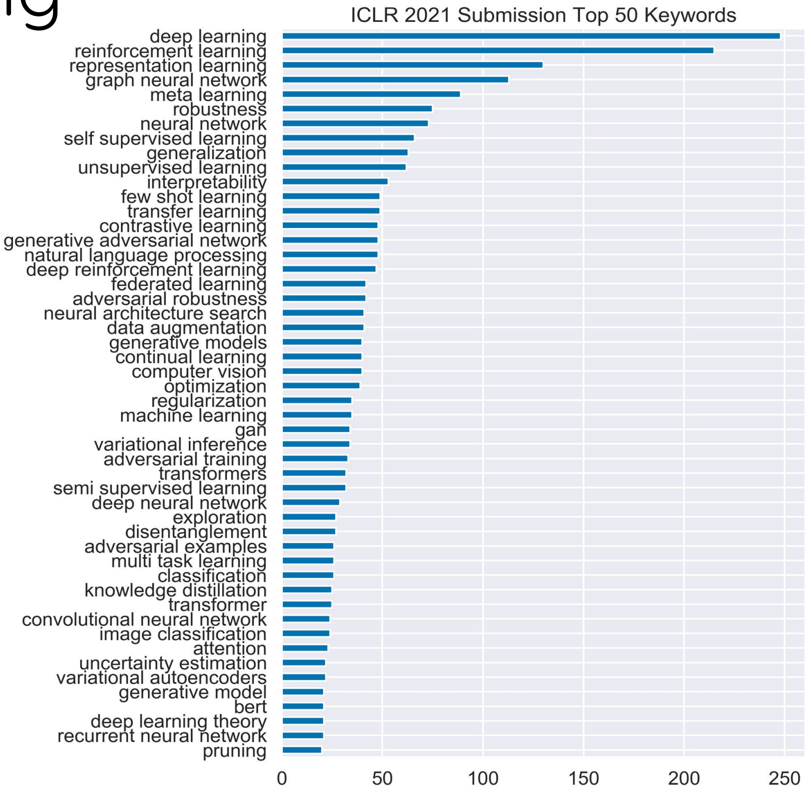
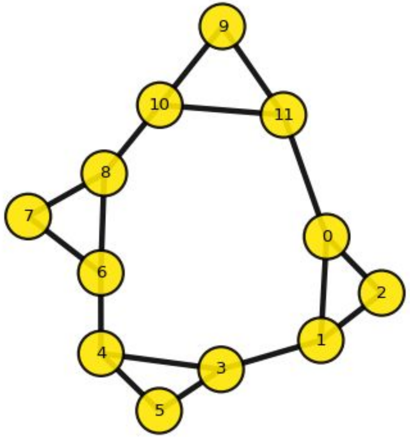


Image form https://www.youtube.com/watch?v=uF53xsT7mjc , also recommended to watch: https://www.youtube.com/watch?v=8owQBFAHw7E

Graph Representation

What are the ways that we can represent graphs or nodes in a graph?



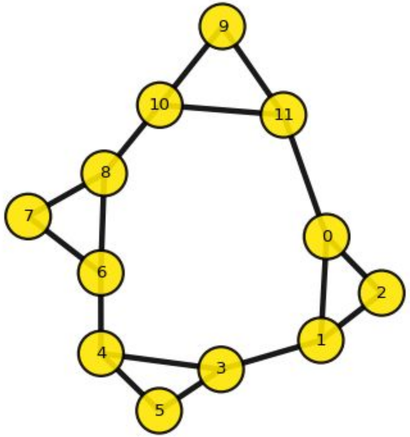
Graph Representation

What are the ways that we can represent graphs or nodes in a graph?

Adjacency matrix: $A \in \{0,1\}^{N \times N}$

$$h_i = [0,0,0,0,0,0,0,0,1,1,0,1]^\top$$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	1	1	0	1
11	1	0	0	0	0	0	0	0	0	1	1	0



How can we compute number of common neighbors of two nodes with this?

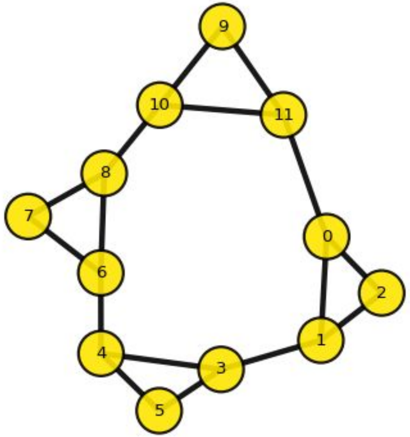
Graph Representation

What are the ways that we can represent graphs or nodes in a graph?

Adjacency matrix: $A \in \{0,1\}^{N \times N}$

$$h_i = [0,0,0,0,0,0,0,0,1,1,0,1]^\top$$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	1	1	0	1
11	1	0	0	0	0	0	0	0	0	1	1	0



How can we compute number of common neighbors of two nodes with this? $h_i^\top h_j$

How else to represent graphs/nodes?

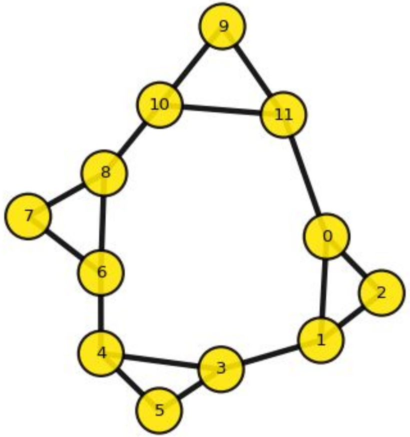
Graph Representation

What are the ways that we can represent graphs or nodes in a graph?

Adjacency matrix: $A \in \{0,1\}^{N \times N}$

$$h_i = [0,0,0,0,0,0,0,0,1,1,0,1]^\top$$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	1	1	0	1
11	1	0	0	0	0	0	0	0	0	1	1	0



How can we compute number of common neighbors of two nodes with this? $h_i^\top h_j$

How else to represent graphs/nodes? Laplacian,

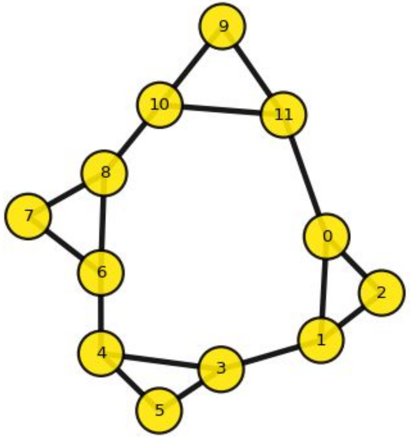
Graph Representation

What are the ways that we can represent graphs or nodes in a graph?

Adjacency matrix: $A \in \{0,1\}^{N \times N}$

$$h_i = [0,0,0,0,0,0,0,0,1,1,0,1]^\top$$

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	1	1	0	1
11	1	0	0	0	0	0	0	0	0	1	1	0



How can we compute number of common neighbors of two nodes with this? $h_i^\top h_j$

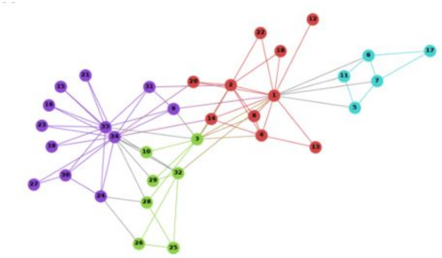
How else to represent graphs/nodes? Laplacian, k-smallest nontrivial eigenvectors of Graph Laplacian a.k.a. Laplacian eigenmaps (LE)

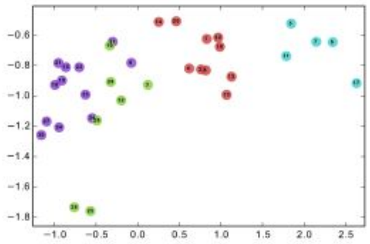
Graph Representation

embed the graph in vector space: $G \to \mathbb{R}^{n \times d}$

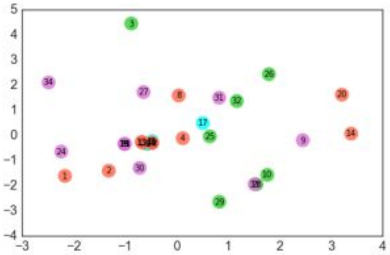
i.e. map each node to a vector: $h_i : i \in G \to \mathbb{R}^d$

- distance in the embedded space ⇒ link prediction
- decision boundaries in the embedded space ⇒ node classification

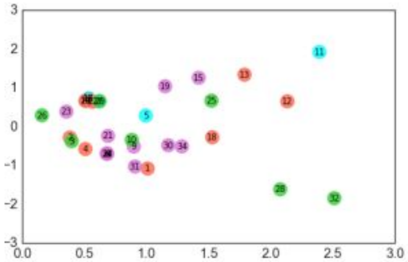




(a) Output: DeepWalk



(e) Output: LE



(f) Output: SVD

See A Tutorial on Network Embeddings, 2018

What is a good representation?

Representation for node i: $h_i : i \in G \rightarrow \mathbb{R}^d$

Preserves the edge structure based on cross-entropy loss:

$$\sum_{(i,j)\in E} \log \sigma(h_i^\top h_j) + \sum_{(i,j)\notin E} \log(1 - \sigma(h_i^\top h_j))$$

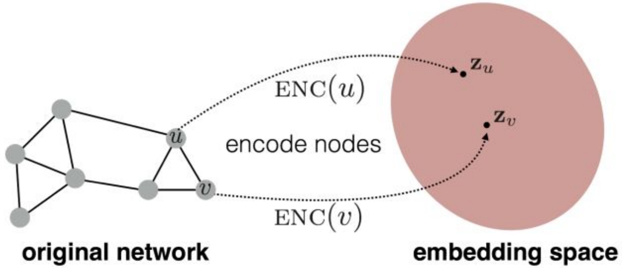
This can be trained unsupervised, and puts connected nodes closeby

Deepwalk, node2vec and LINE redefine this based on nodes that co-occur in a (short) random walk

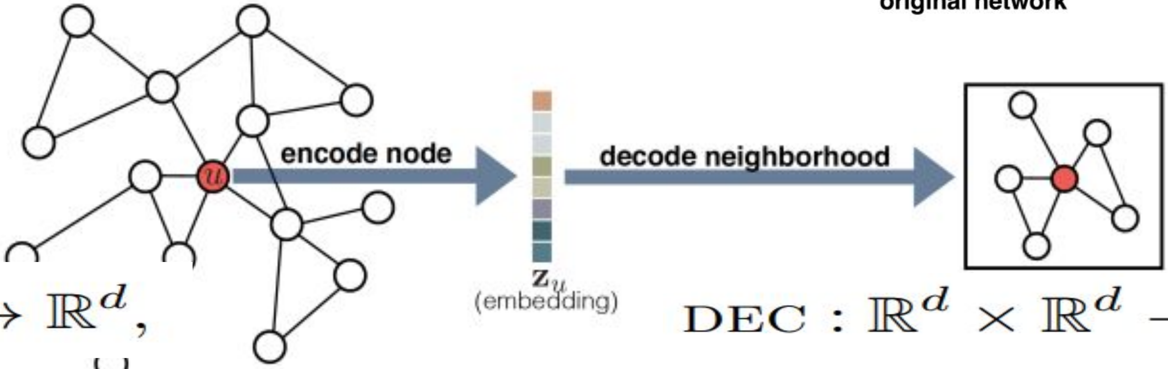
slides based on https://petar-v.com/talks/GNN-Wednesday.pdf

An Encoder-Decoder Perspectiv

Encoder gives low dimensional embedding that summarizes the graph position and structure in local neighbourhood



Decoder reconstructs this neighbourhood given the embedding of the node



$$\text{ENC} : \mathcal{V} \to \mathbb{R}^d,$$

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+.$$

$$\mathcal{L} = \sum_{i,j} l(DEC(h_i, h_j), S(i, j))$$

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

A summary of shallow embedding algorithms

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$

learn embeddings for each node such that the inner product between the learned embedding vectors approximates some deterministic measure of node similarity

gives identical to the solution for spectral clustering, i.e. d smallest eigenvectors of the Laplacian

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

A summary of shallow embedding algorithms

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u, v], ..., \mathbf{A}^k[u, v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$

matrix-factorization

$$\mathcal{L} \approx \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{S}\|_2^2,$$

learn embeddings for each node such that the inner product between the learned embedding vectors approximates some deterministic measure of node similarity

Deterministic measure of similarity ⇒ stochastic measure of neighbourhood overlap

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

A summary of shallow embedding algorithms

Method	Decoder	Similarity measure	Loss function
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u,v]$
Graph Fact.	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
GraRep	$\mathbf{z}_u^\top \mathbf{z}_v$	$\mathbf{A}[u,v], ..., \mathbf{A}^k[u,v]$	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
HOPE	$\mathbf{z}_u^\top \mathbf{z}_v$	general	$\|\text{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u,v]\|_2^2$
DeepWalk	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_\mathcal{G}(v \mid u)$	$-\mathbf{S}[u,v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$
node2vec	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_\mathcal{G}(v \mid u)$ (biased)	$-\mathbf{S}[u,v] \log(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v))$

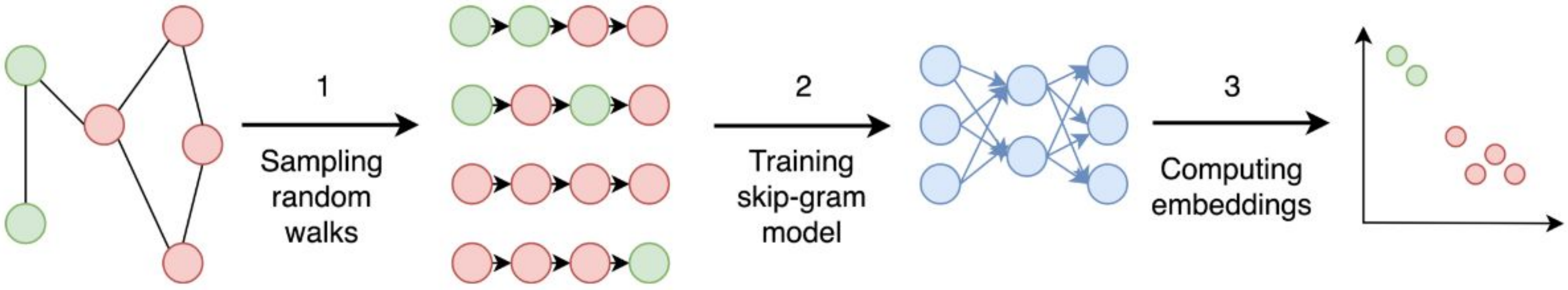
matrix-factorization

$\mathcal{L} \approx \|\mathbf{Z}\mathbf{Z}^\top - \mathbf{S}\|_2^2,$

node embeddings are optimized so that two nodes have similar embeddings
if they tend to co-occur on short random walks over the graph
Similarity is probability of visiting v on a fixed length random walk from u

https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

Deepwalk



Phases of DeepWalk approach

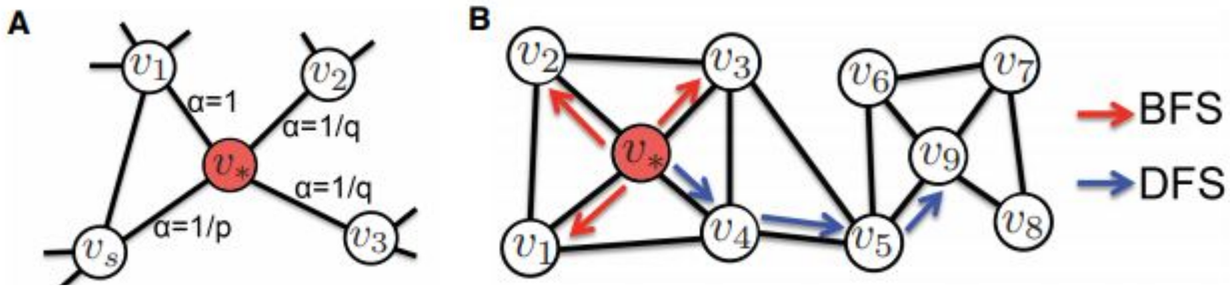
- 32 to 64 random walks from each node of a length of about 40 steps
- Random walks as sentences, maximize probability of predicting neighbour nodes

https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007

https://arxiv.org/pdf/1403.6652.pdf

Node2vec

Similar to Deepwalk but interpolates between random walks that discover larger neighborhood (Q), and those that stay local (P)



$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} -\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - \gamma \mathbb{E}_{v_n \sim P_n(\mathcal{V})}[\log(-\sigma(\mathbf{z}_u^\top \mathbf{z}_{v_n}))].$$

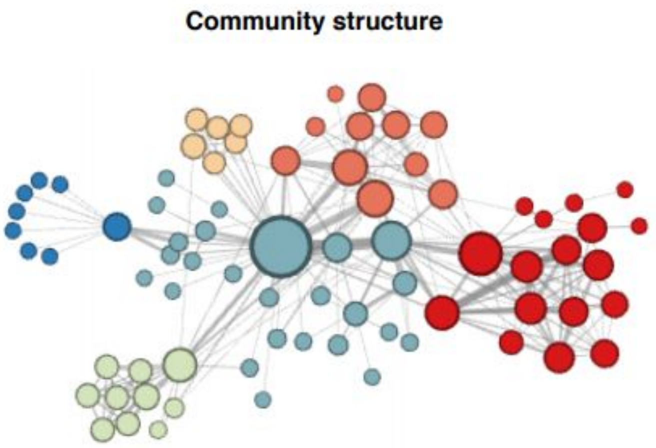
Negative samping

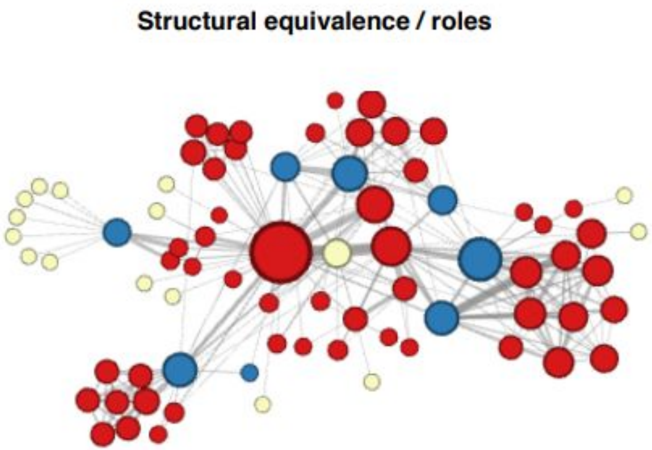
https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_3-Node_Embeddings.pdf

Node2Vec Different ways to embed

Embedding so that nodes

- in the same cluster are placed close together (DFS)
- with similar roles are placed close together (BFS)





https://arxiv.org/pdf/1607.00653.pdf

Limitations of Shallow Embeddings

No parameter sharing ⇒ less scalable

Ignores features or attributes

Inherently transductive ⇒ can not process unseen nodes

Read more:
A Tutorial on Network Embeddings, 2018 &
Representation Learning on Graphs, 2017 &
GLR book's chapter on node embedding, 2020

From Shallow Embeddings to Graph Neural Nets

- No parameter sharing ⇒ less scalable
- Ignores features or attributes
- Inherently transductive ⇒ can not process unseen nodes

optimized a unique embedding vector for each node ⇒ more complex encoder models, graph neural networks which work based on feature propagation

$$f(X, A)$$

- Number of parameters doesn't grow with graph size but feature dimension
- Naturally incorporates node features
- Inherently inductive ⇒ infer embedding for unseen nodes

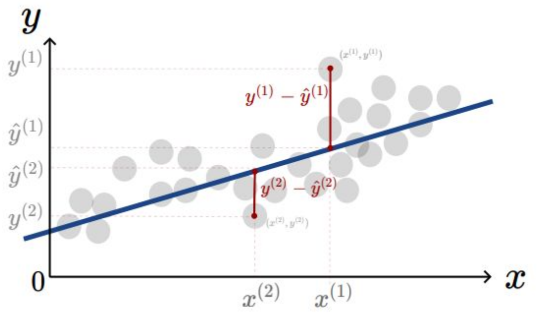
Watch https://www.cs.mcgill.ca/~wlh/grl_book/files/hamilton_grl_talk.mp4

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$

$$x = [1, x_1, \ldots, x_D]^\top$$

$$w = [w_0, w_1, \ldots, w_D]^\top$$



Model: linear combination of features and weights
Learning: find the weights that minimize a cost function
Cost: sum of losses per individual point

$$J(w) = \frac{1}{2} \sum_{n=1}^{N} \left(y^{(n)} - w^\top x^{(n)} \right)^2$$

$$w^* = \arg\min_w J(w)$$

$$X = \begin{bmatrix} x^{(1)\top} \\ x^{(2)\top} \\ \vdots \\ x^{(N)\top} \end{bmatrix} = \begin{bmatrix} x_1^{(1)}, & x_2^{(1)}, & \cdots, & x_D^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)}, & x_2^{(N)}, & \cdots, & x_D^{(N)} \end{bmatrix} \in \mathbb{R}^{N \times D}$$

one instance

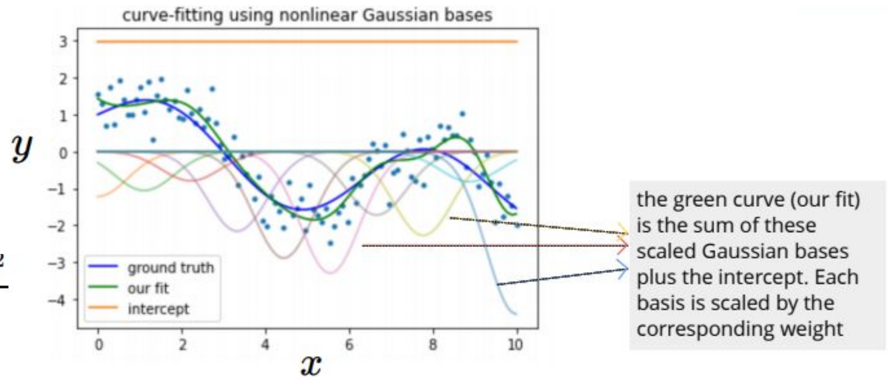
one feature

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive, use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
  - Transform the input with nonlinearities then apply linear model

Example: perfect nonlinear fit with linear model and 10 nonlinear Gaussian bases

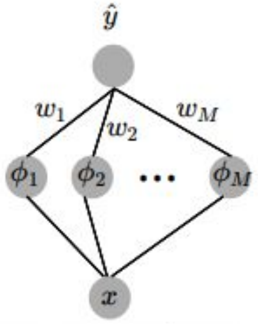
$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



Neural Networks - Short Intro

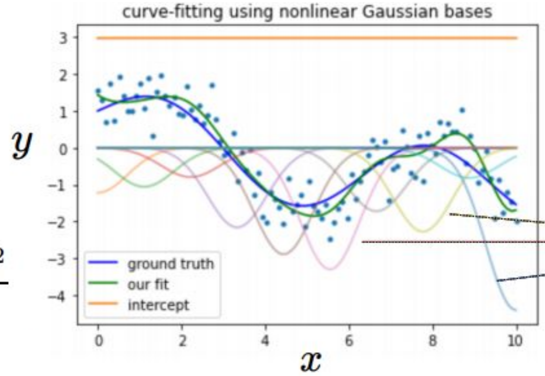
- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$

- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$



Gaussian bases

$$\phi_k(x) = e^{-\frac{(x-\mu_k)^2}{s^2}}$$



the green curve (our fit) is the sum of these scaled Gaussian bases plus the intercept. Each basis is scaled by the corresponding weight

Neural Networks - Short Intro

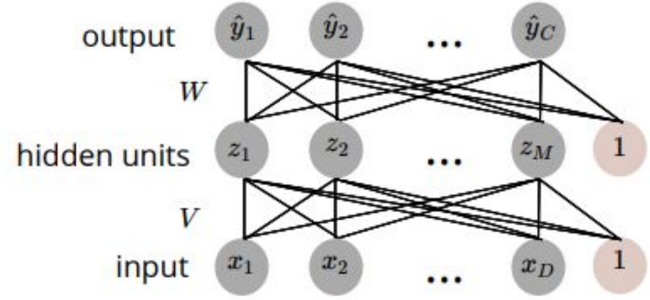
- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Neural networks use **adaptive** nonlinear bases
  - Learning the (weights of) nonlinear bases

$$\hat{y} = g(W\, h(V\, x))$$

non-linearities are applied elementwise

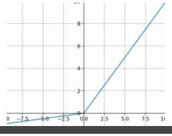
$$\hat{y}_c = g\left(\sum_m W_{c,m} h\left(\sum_d V_{m,d} x_d\right)\right)$$

$x \in \mathbb{R}^{D\times 1}$
$V \in \mathbb{R}^{M\times D}$
$Z = h(Vx) \in \mathbb{R}^{M\times 1}$
$W \in \mathbb{R}^{C\times M}$
$y \in \mathbb{R}^{C\times 1}$



  - The most common non-linearity

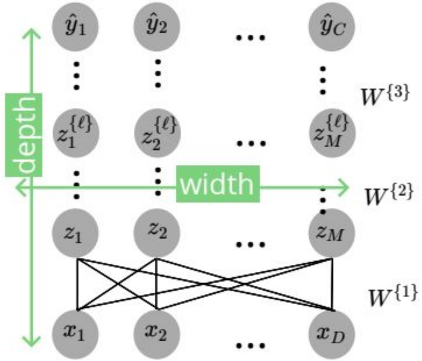
leaky ReLU $h(x) = \max(0, x) + \gamma \min(0, x)$



But what is a neural network?

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases



$z^{\{l\}} = h(W^{\{l\}} z^{\{l-1\}})$

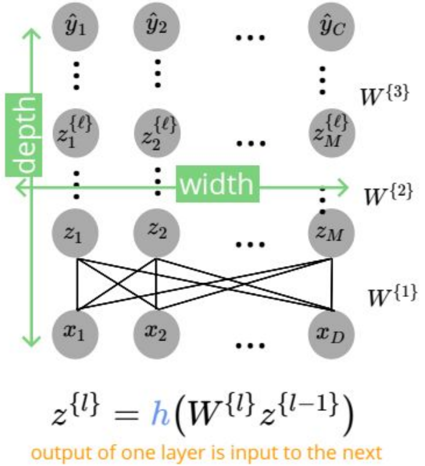
output of one layer is input to the next

[But what is a neural network?]

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases

Can we feed an adjacency matrix to this? E.g. flatten the matrix into a vector of length n^2

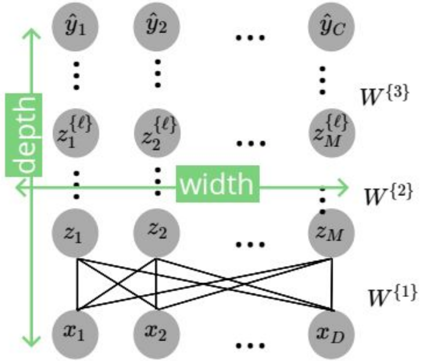


[But what is a neural network?]

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases

Can we feed an adjacency matrix to this? Not the best choice



$z^{\{l\}} = h\left(W^{\{l\}} z^{\{l-1\}}\right)$

output of one layer is input to the next

[But what is a neural network?]

Permutation invariance

function f that takes an adjacency matrix A as input should be:

- Permutation Invariance

$$f(PAP^\top) = f(A)$$

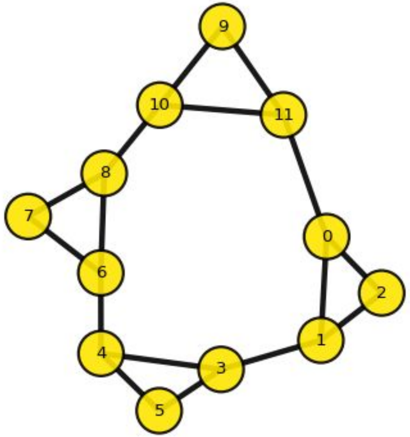
or

- Permutation Equivariance

$$f(PAP^\top) = Pf(A)$$

where P is a permutation matrix that reorders nodes

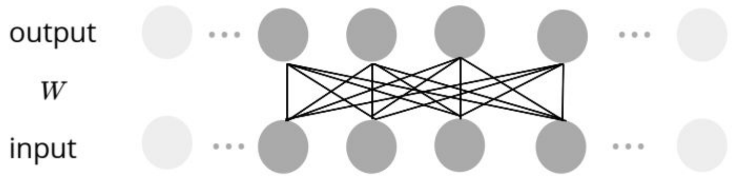
	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	0	0	0	0	0	0	0	0	1
1	1	0	1	1	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	1	1	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	0	1	0	0	0
8	0	0	0	0	0	0	1	1	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	0	0	0	0	0	0	0	0	1	1	0	1
11	1	0	0	0	0	0	0	0	0	1	1	0

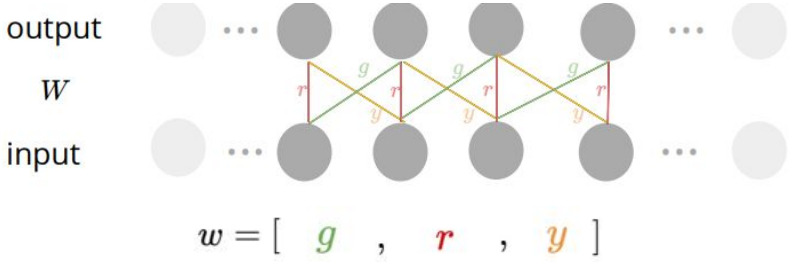


Since changing order of nodes in the adjacency does not change the graph

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases
- Parameter sharing: elements of w of the same color are tied together

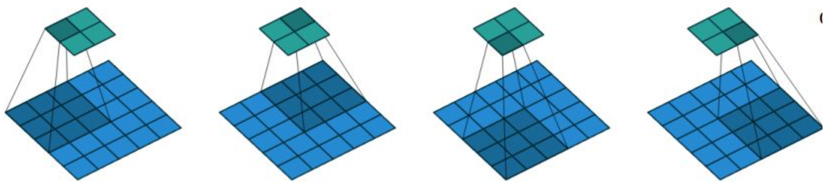




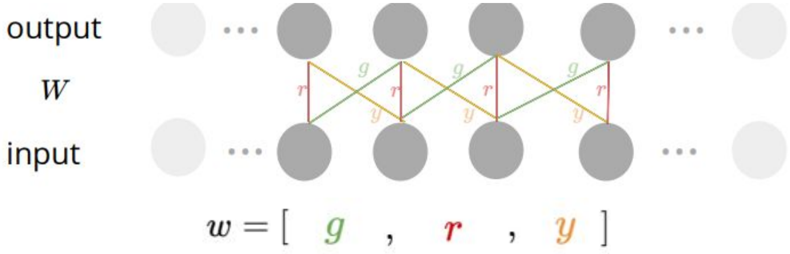
1D convolution layer

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases
- Parameter sharing: elements of w of the same color are tied together



2D Convolution

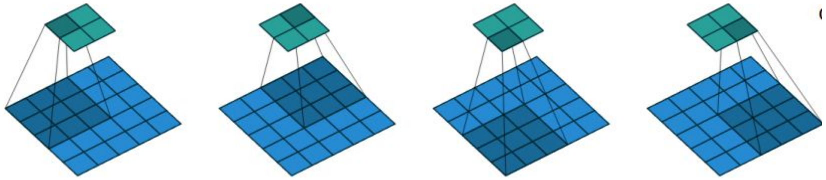


1D convolution layer

https://cs231n.github.io/convolutional-networks/

Neural Networks - Short Intro

- Linear regression: $f(x) = w^\top x = \sum_d w_d x_d$
- More expressive: use nonlinear bases: $f(x) = w^\top \Phi = \sum_d w_d \phi_d(x)$
- Deep networks stack/compose layers of adaptive nonlinear bases
- Parameter sharing: elements of w of the same color are tied together

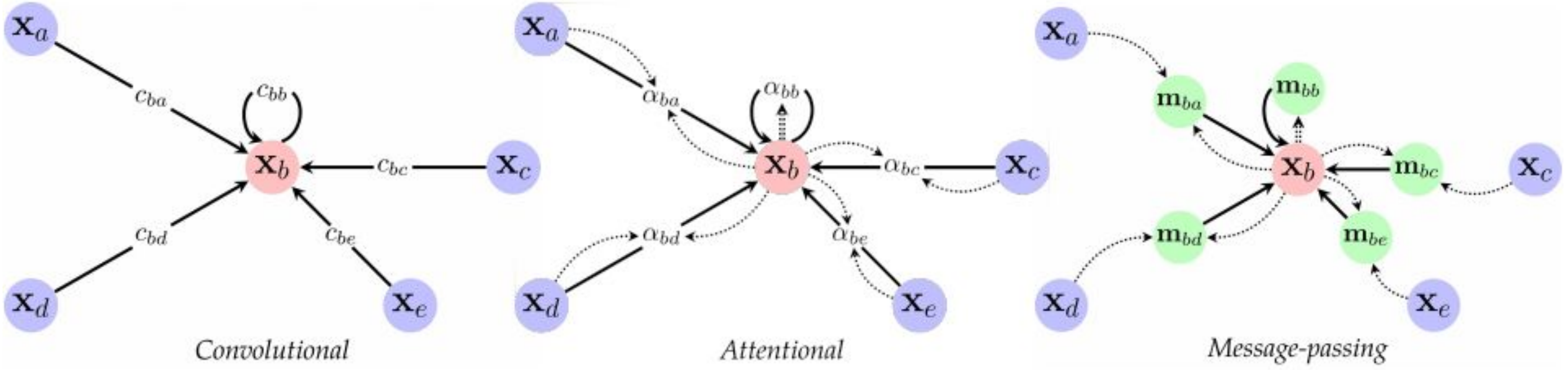


Can we have convolution for graphs?

2D Convolution https://cs231n.github.io/convolutional-networks/

Graph Neural Networks

Use the local neighbourhood similar to convolution on images



$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right) \qquad \mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right) \qquad \mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

From https://petar-v.com/talks/GNN-Wednesday.pdf

Attributed Graphs

$$f(X, A)$$

If we have:

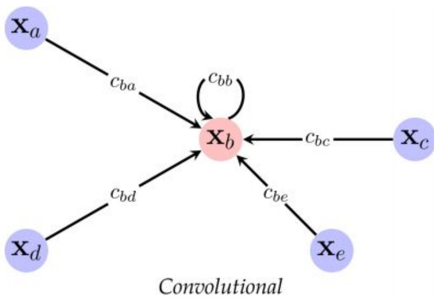
$$A_{ij} = \begin{cases} 1, & i \text{ links to } j \\ 0, & \text{otherwise} \end{cases} \qquad X_{ik} = \begin{cases} 1, & i \text{ has } k \\ 0, & \text{otherwise} \end{cases}$$

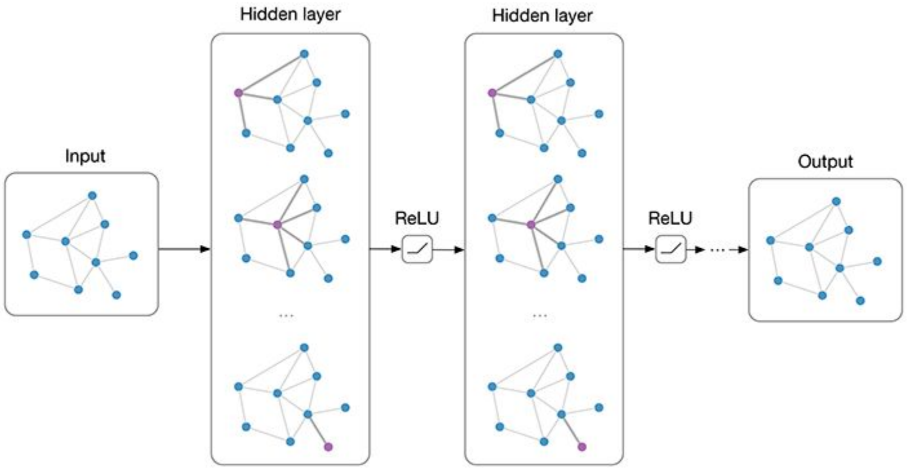
Then simple matrix multiplication of A and X, AX, gives us the number of neighbors of a particular attribute/type for each node, i.e.

- k^{th} column of AX shows the number of type k neighbors for all nodes,
  - e.g., number of 'male' friends each person has.
- i^{th} row of AX shows the number of neighbors node i for all types,
  - e.g., number of friends 'smith' has of each type, say male and female

Convolutional GNN

GCN (Kipf & Welling, ICLR'17)



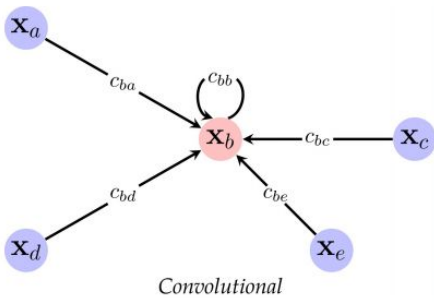


Multi-layer Graph Convolutional Network (GCN) with first-order filters.

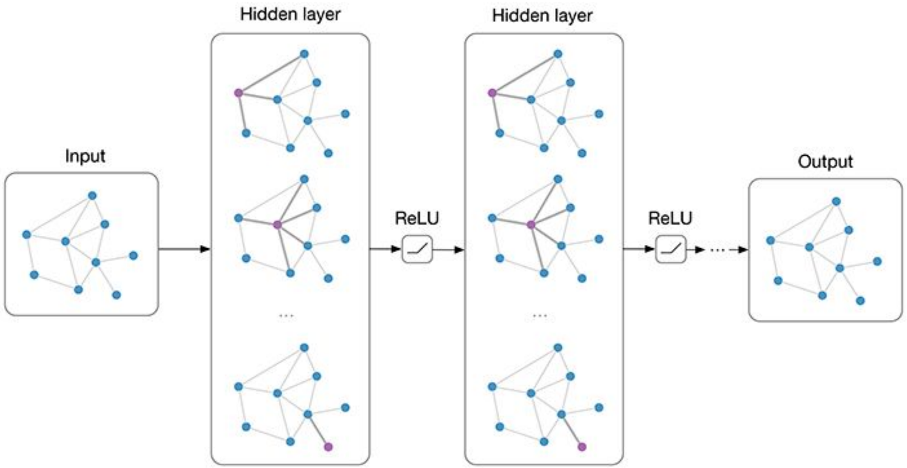
$$H^{l+1} = \phi(AH^lW^l)$$

From
https://petar-v.com/talks/GNN-Wednesday.pdf
https://www.youtube.com/watch?v=uF53xsT7mjc

Convolutional GNN



GCN (Kipf & Welling, ICLR'17)



Multi-layer Graph Convolutional Network (GCN) with first-order filters.

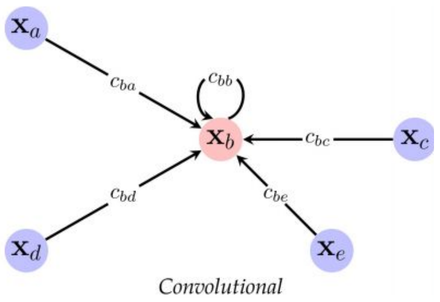
$$H^{l+1} = \phi(AH^lW^l)$$

$$H^{l+1} = \phi(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^lW^l)$$

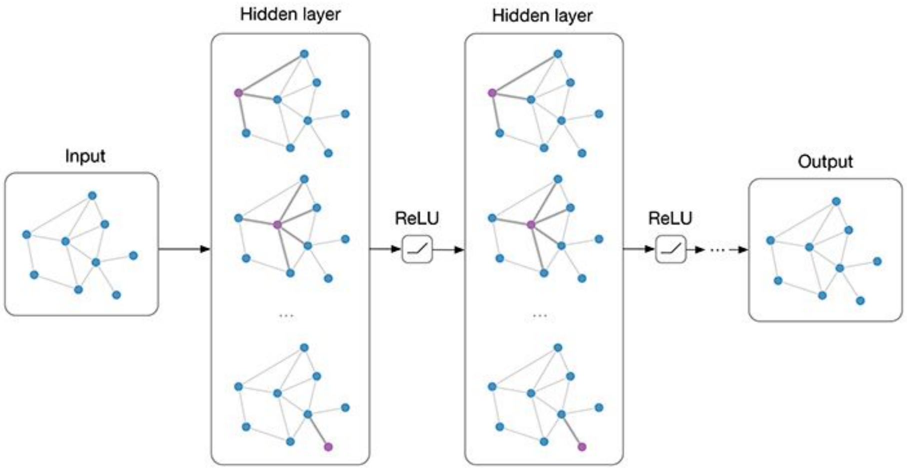
$$\hat{A} = A + I$$

From
https://petar-v.com/talks/GNN-Wednesday.pdf
https://www.youtube.com/watch?v=uF53xsT7mjc

Convolutional GNN



GCN ([Kipf & Welling, ICLR'17])



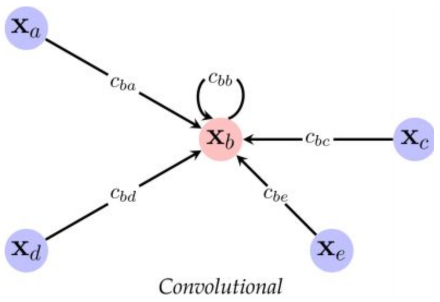
Multi-layer Graph Convolutional Network (GCN) with first-order filters.

$$H^{l+1} = \phi(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^l W^l)$$

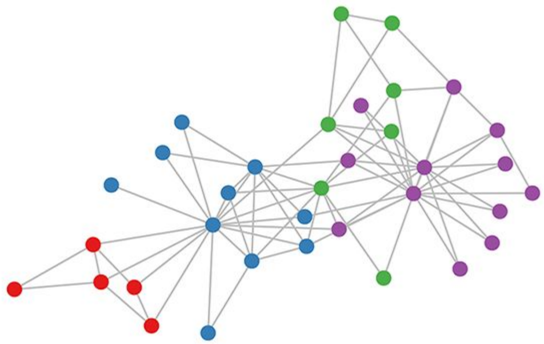
$$h_i^{l+1} = \phi(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} h_j^l W^l)$$

From
https://petar-v.com/talks/GNN-Wednesday.pdf
https://www.youtube.com/watch?v=uF53xsT7mjc

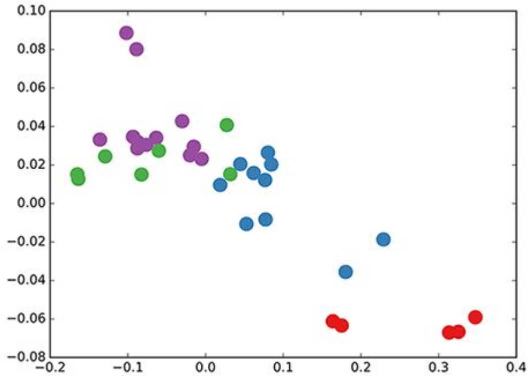
Convolutional GNN



GCN (Kipf & Welling, ICLR'17)



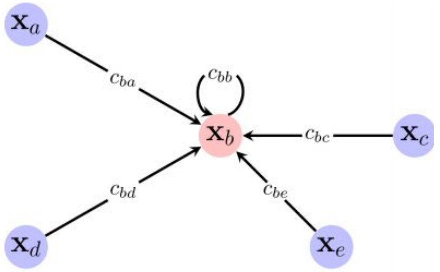
$$X = I$$



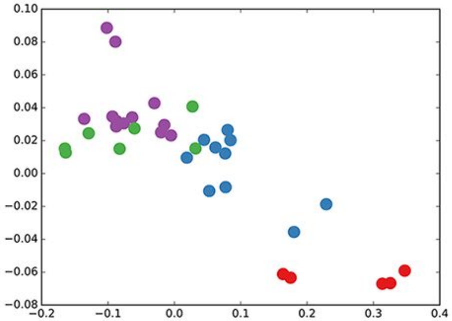
3-layer with random weights (untrained!)

From
https://petar-v.com/talks/GNN-Wednesday.pdf
https://www.youtube.com/watch?v=uF53xsT7mjc

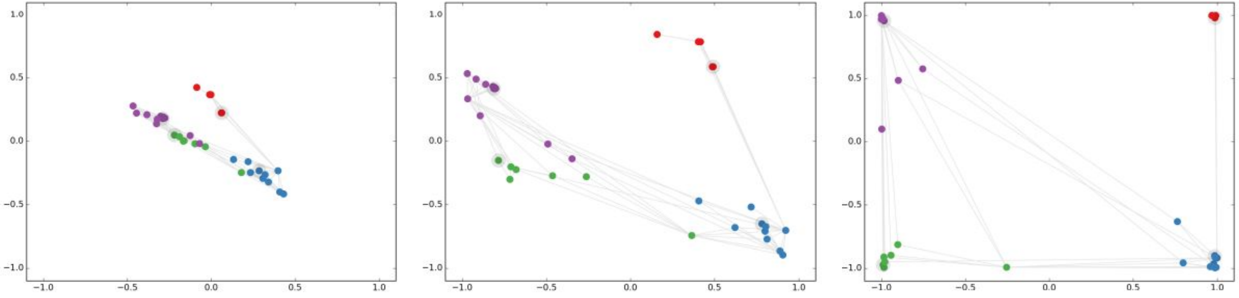
Convolutional GNN



GCN (Kipf & Welling, ICLR'17)



$$X = I$$

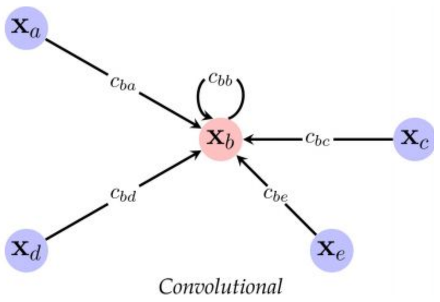


(a) Iteration 25 (b) Iteration 50 (f) Iteration 300

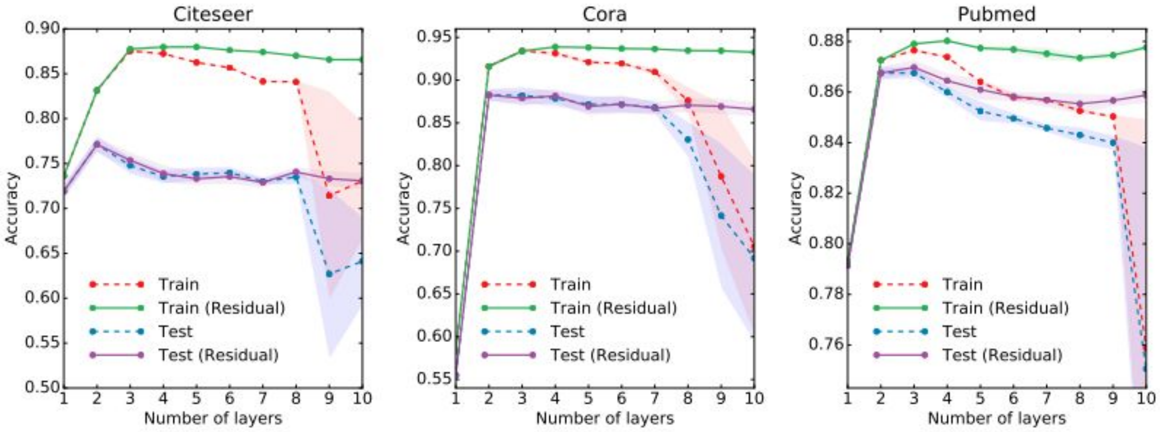
Iteration 0 ⇒ Gets better as we learn the weights

From https://arxiv.org/pdf/1609.02907.pdf

Convolutional GNN



GCN ([Kipf & Welling, ICLR’17](https://arxiv.org/pdf/1609.02907.pdf))



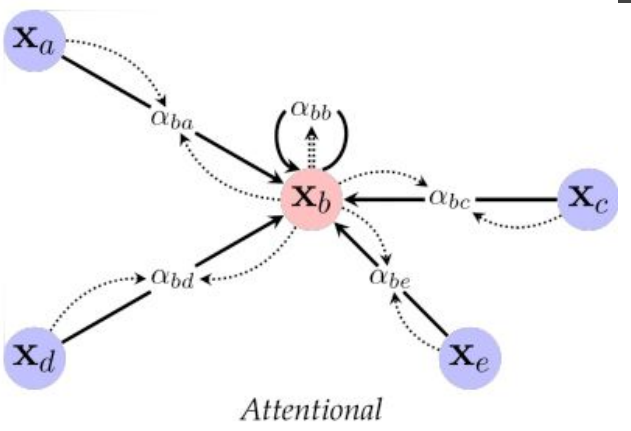
More layers do not help

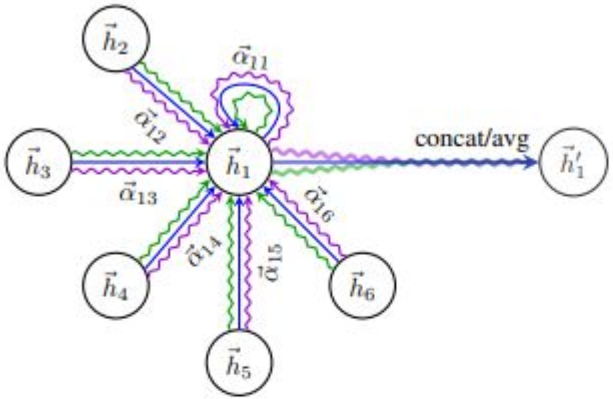
From https://arxiv.org/pdf/1609.02907.pdf

Attentional GNN

GAT (Veličković et al., ICLR'18)

compute scalar value in each edge





Transductive

Method	Cora	Citeseer	Pubmed
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$\mathbf{79.0} \pm 0.3\%$
GAT (ours)	$\mathbf{83.0} \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$\mathbf{79.0} \pm 0.3\%$

From https://petar-v.com/talks/GNN-Wednesday.pdf

Resources: Libraries and Datasets

OGB

Ogb.stanford.edu

TUDataset

Graphlearning.io

PyTorch geometric

github.com/rusty1s/pytorch_geometric

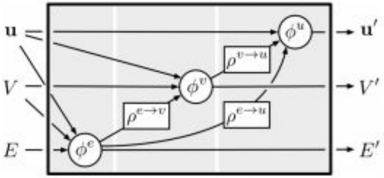
DGL

dgl.ai

https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

Spektral

graphneural.network



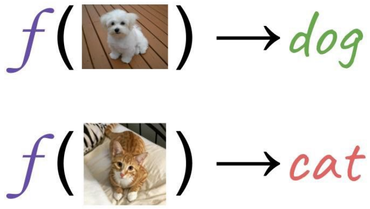
github.com/deepmind/graph_nets

github.com/deepmind/jraph

github.com/graphdeeplearning/benchmarking-gnns

slides based on https://petar-v.com/talks/GNN-Wednesday.pdf

Classification - One slider



- The most common supervised learning setup
- Learns a **function** that maps each input/datapoint to an output/class based on a set of example input-output pairs, a.k.a. labelled data
- This **function** has parameters that are adjusted based on examples in the training set, usually by minimizing a loss defined based on how well the model's output and actual outputs match
- This optimization is commonly based on gradient descent, i.e. adjusting the parameters of model/**function** step by step towards where the loss is decreasing
- Evaluation: since these examples are seen by the model, we test the performance on an hold-out, unseen test set
- Model Selection: The models often have hyperparameters that we do not learn directly but tune them by checking different possible values and measuring the loss on the validation set

train	validation	test