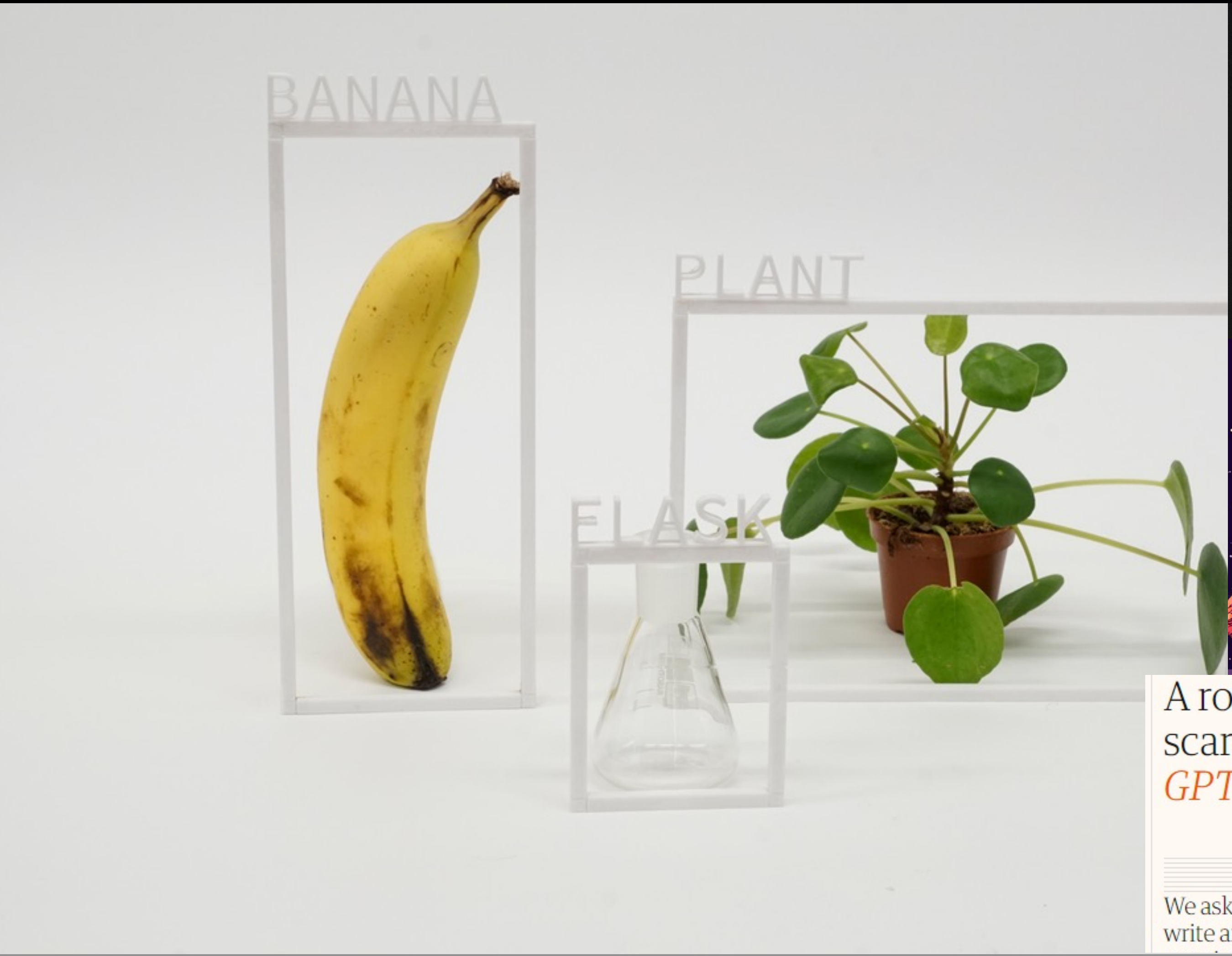
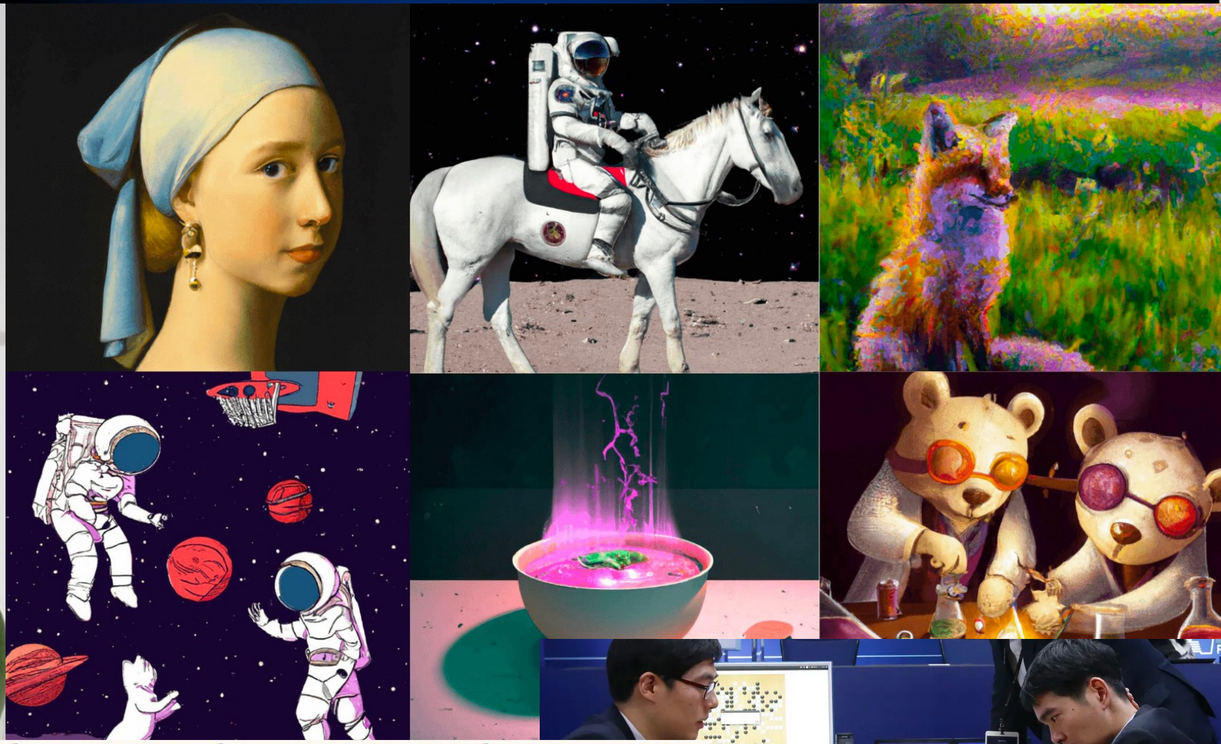
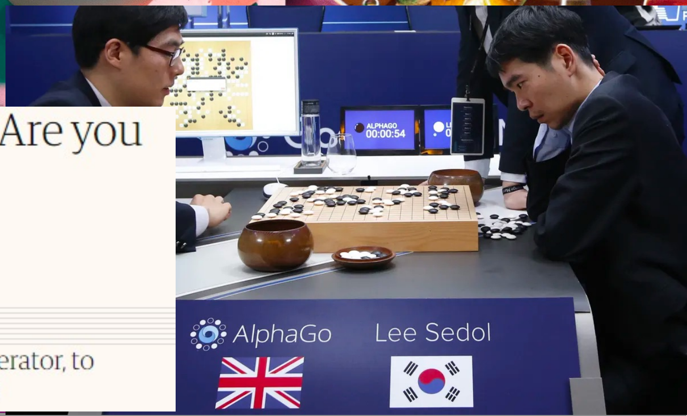




A robot wrote this entire article. Are you scared yet, human?

GPT-3

We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To



Lecture 8: Equivariances and Graphical Neural Networks

Deep Learning 1 @ UvA
Yuki M. Asano

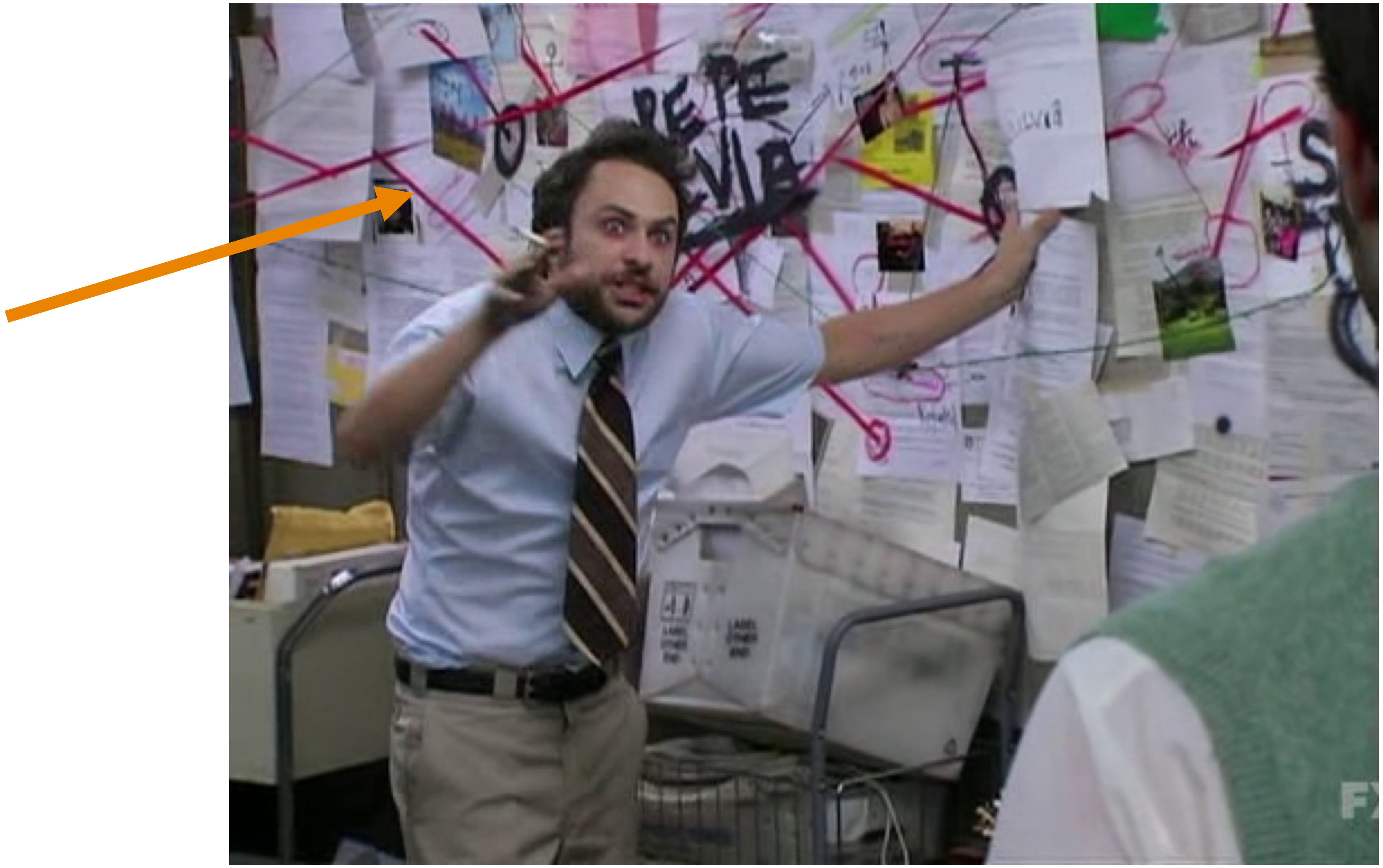
Organisation

- Added all relevant links to canvas at the top (piazza, webpage etc.)
- Reminder that using LISA has advantage of preparing you for your MSc project, where you will be using this
- Practicals today: opportunity to go through some solutions to exercises of assignment 1 – and of course questions about the 2nd assignment
- Thanks for your midway feedback, we've read through it and will provide a compilation at the next lecture.
- We're more than halfway done! You can do this 💪

Lecture overview

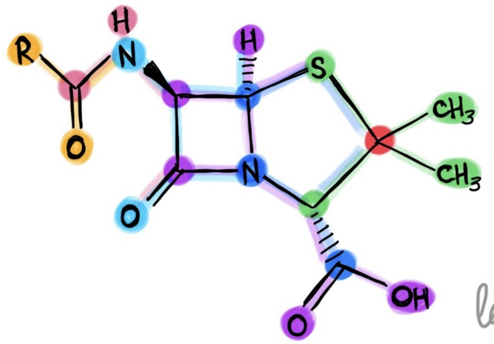
- What makes graphs special?
- Revisiting graphs
- Revisiting convolutions
- Spectral graph convolutions
- Spatial graph convolutions

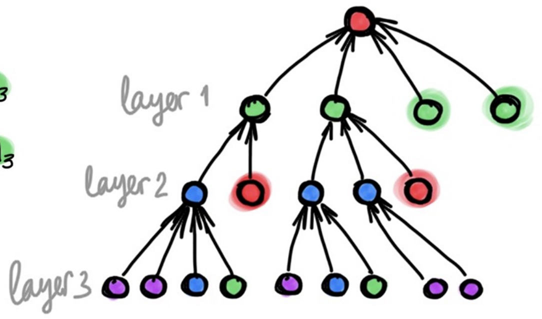
Graphs! They're everywhere

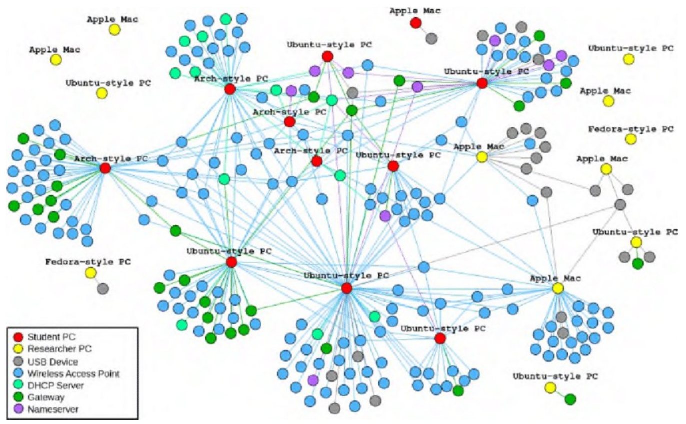


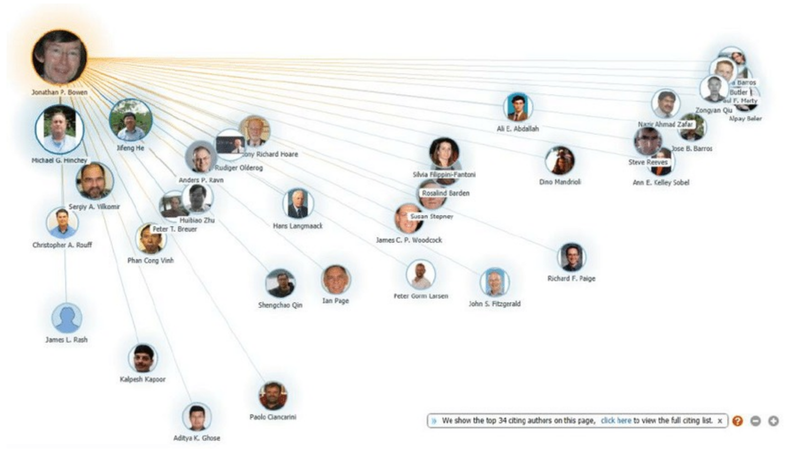
What are graphs?

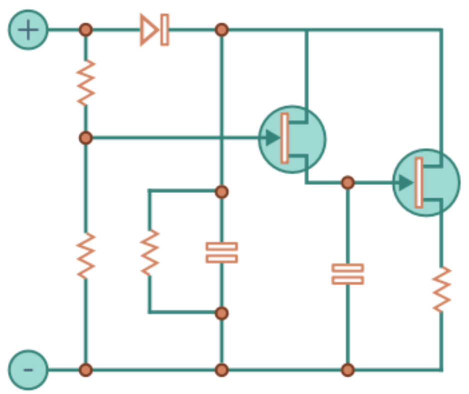
- Data structures of sets of variables (or objects) connected with each other





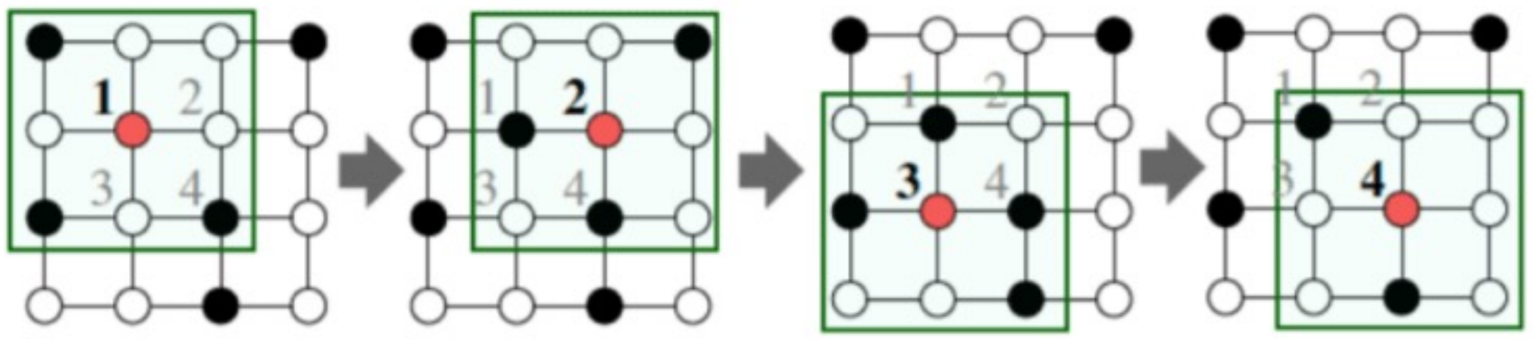



Jonathan P. Bowen
We show the top 34 citing authors on this page, click here to view the full citing list.

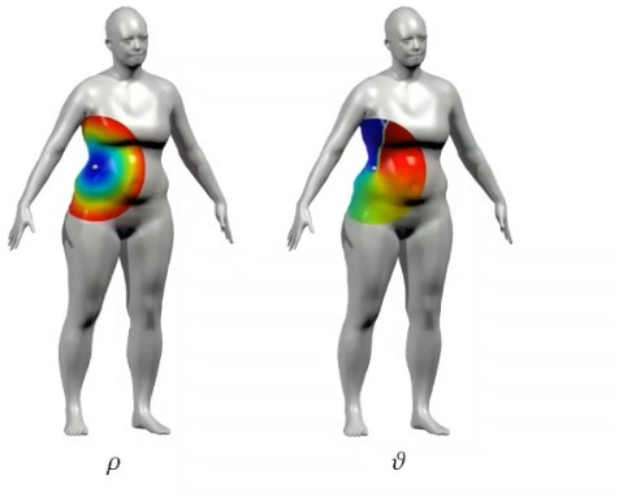


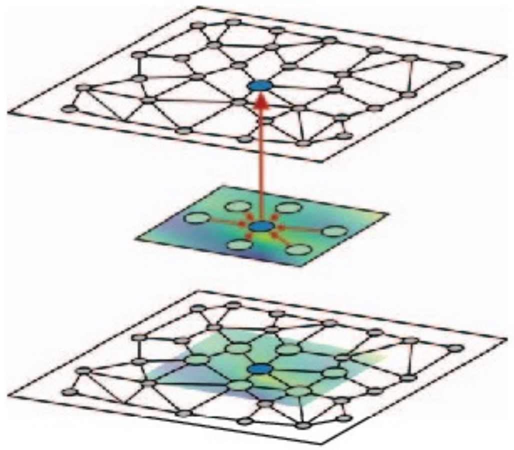
What are graphs?

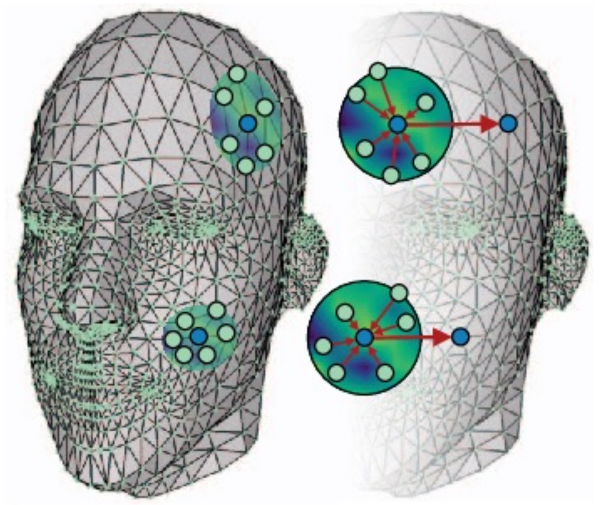
- Images are (or can be posed as) graphs!
- Each node carries a vector of size 3 (R,G,B) values

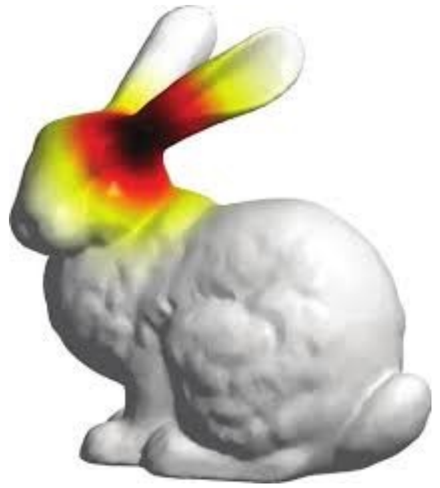


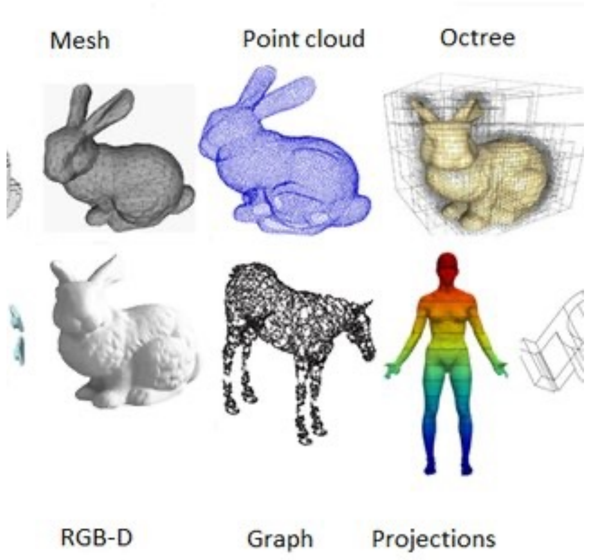
Graphs as geometry.





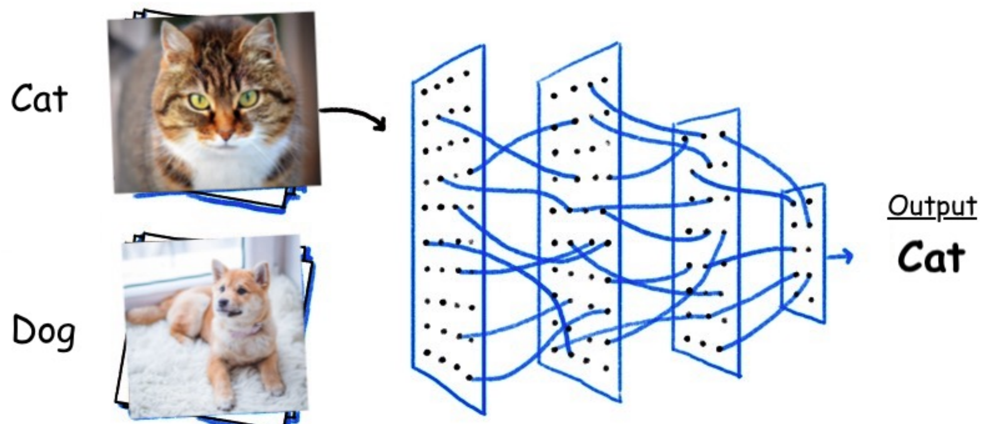


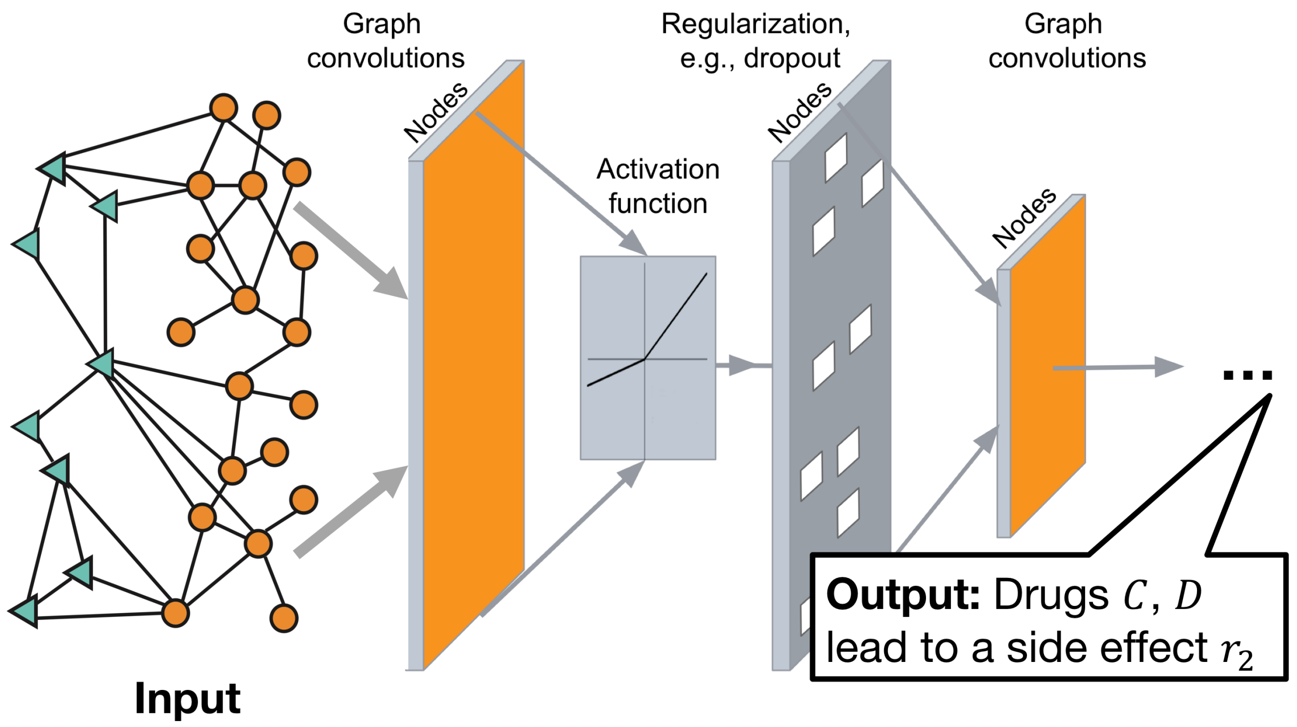




1) Classifying graphs

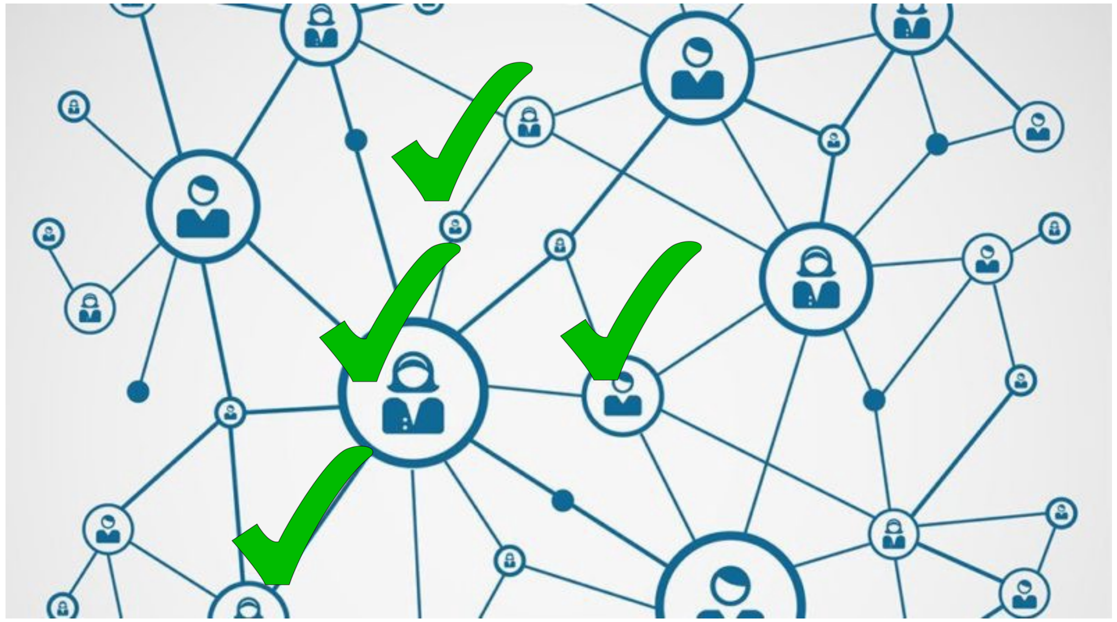
- Making predictions at the level of a graph
- The equivalent of image-level tasks like object recognition




Graph convolutions
Regularization, e.g., dropout
Graph convolutions
Nodes
Nodes
Nodes
Activation function
...
Input
Output: Drugs C, D lead to a side effect r_2

2) Classifying nodes

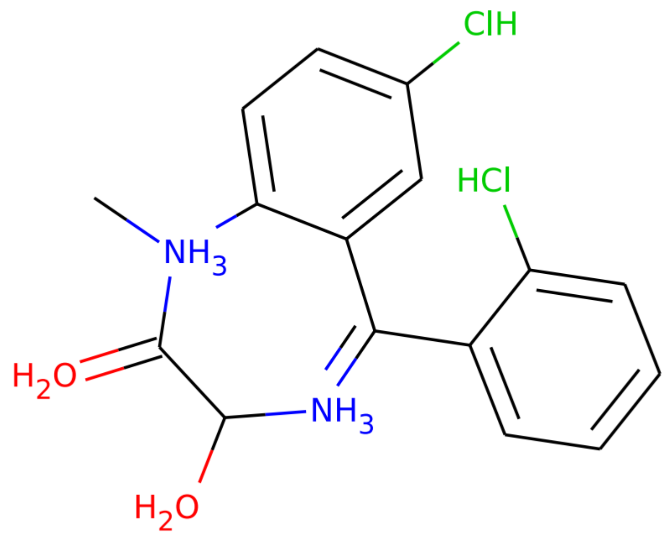
- E.g.: who is likely to have coronavirus?
- The equivalent for images: semantic segmentation



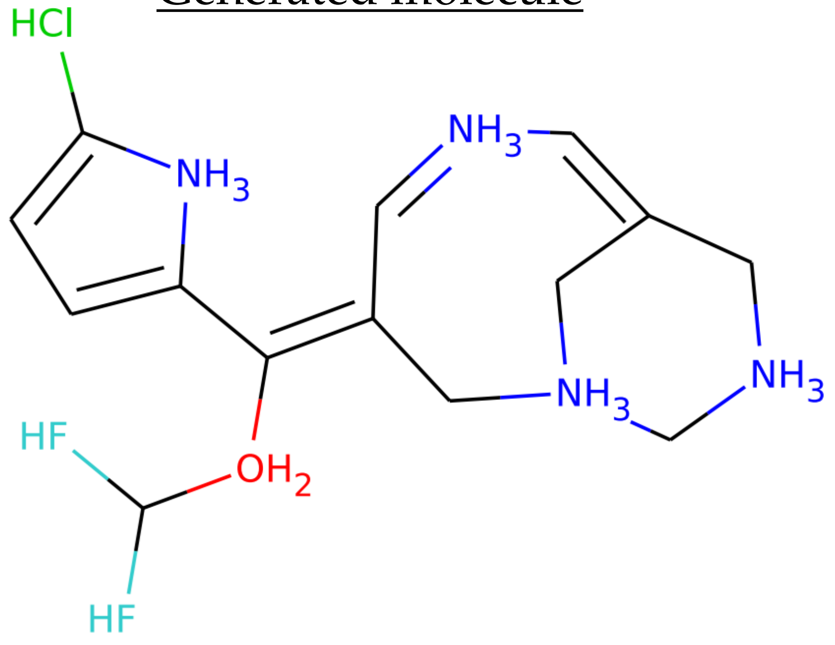
3) Graph generation

- Comparable to image generation

Example molecule



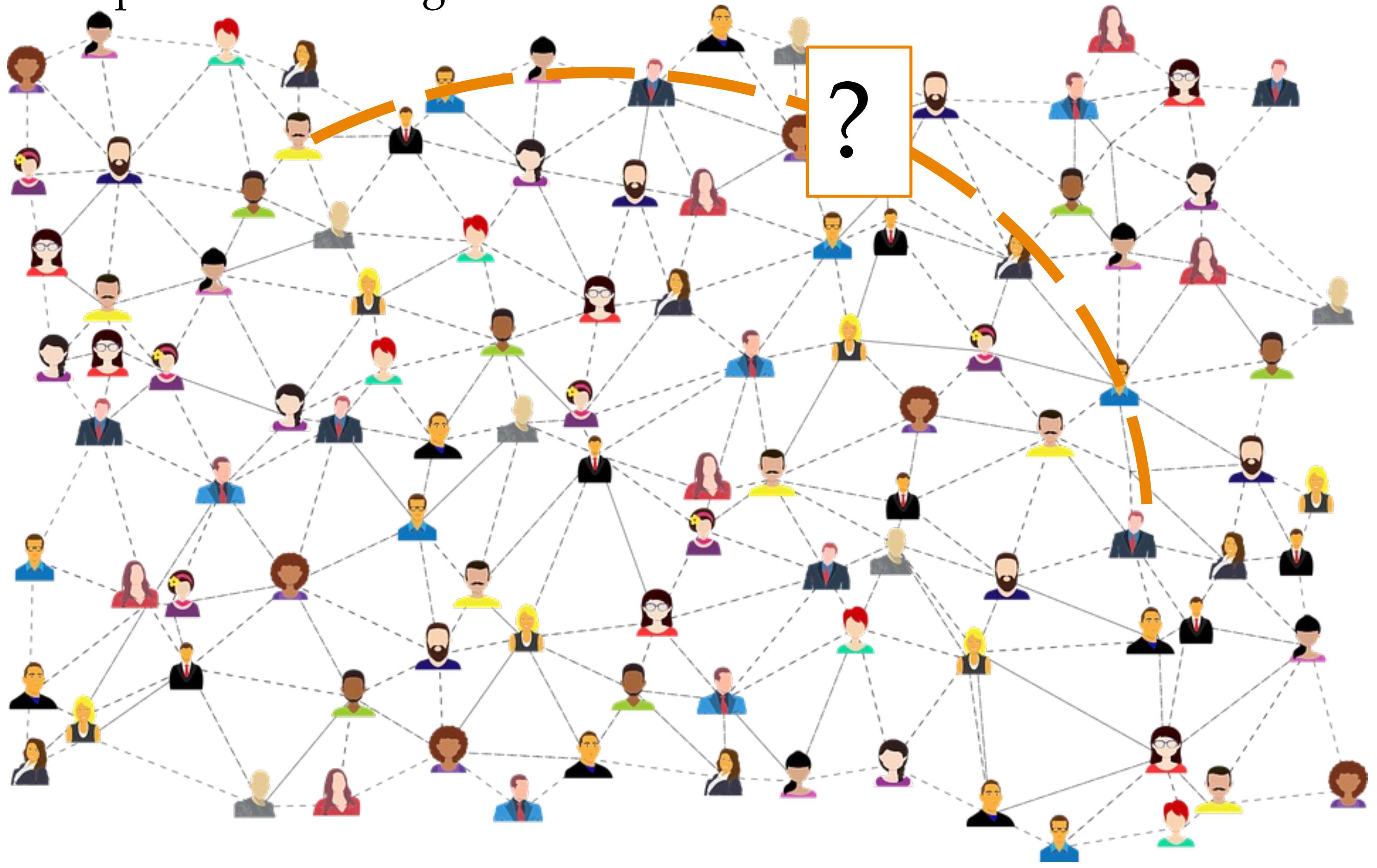
Generated molecule

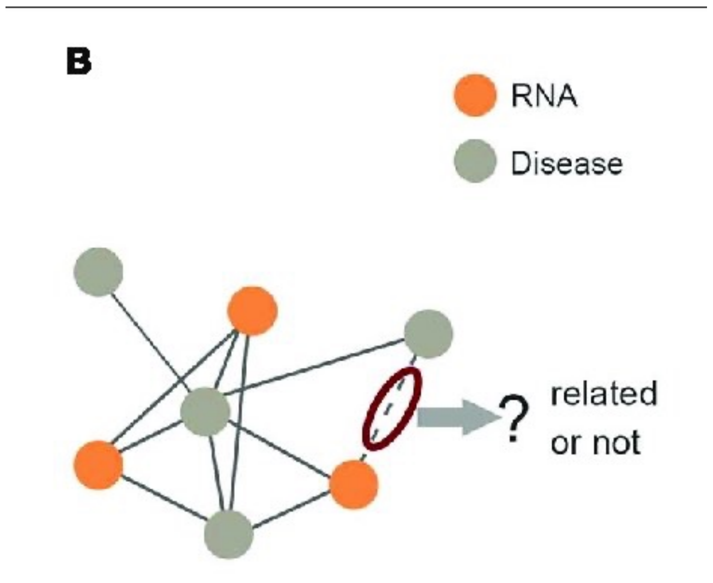


P. Lippe, E. Gavves, Categorical Normalizing Flows via Continuous Transformations ICLR 2021

4) Link/Edge prediction

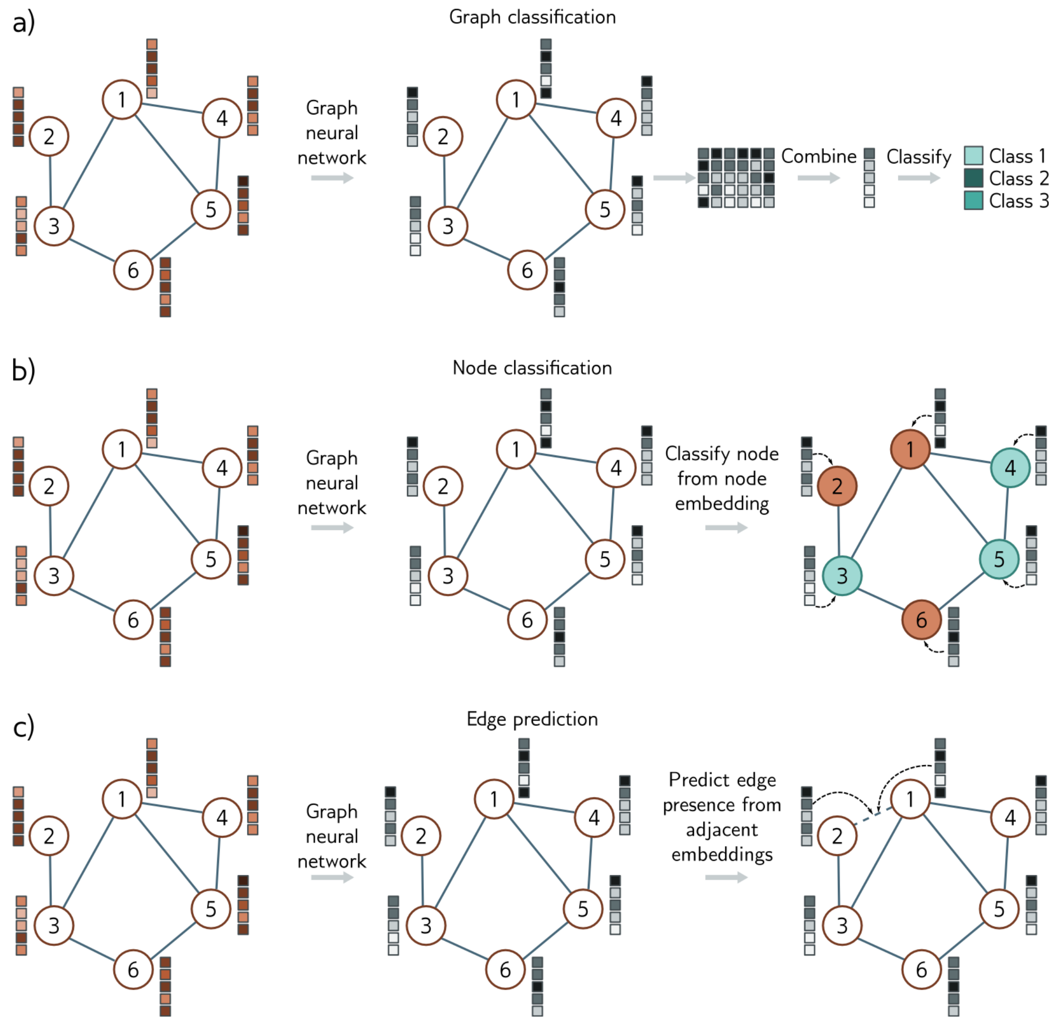
e.g. who is likely connected with whom, what are the relationships between two nodes
no equivalent in images





RNA-Disease association network

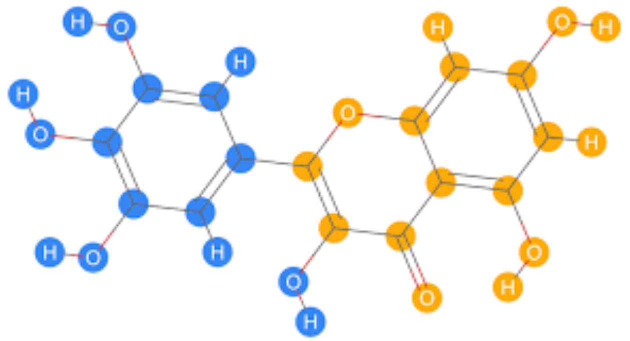
Three tasks visualized: here with nodes that carry features



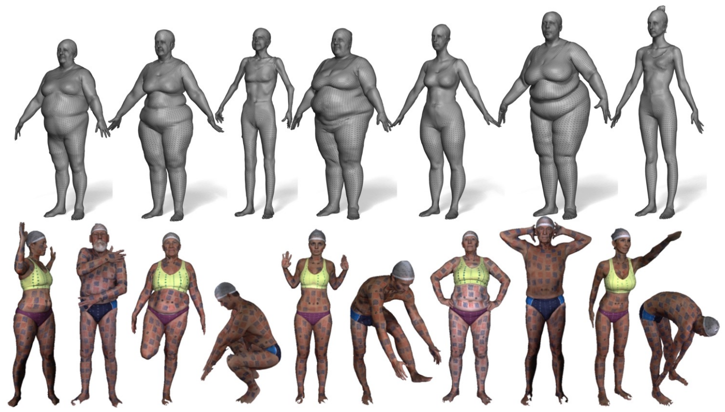
- This is not everything:
  - Edges could carry features
  - Could go beyond pairwise interactions (triangles and other topology)
  - Also community detection (~like image segmentation)

Graphs can be static, varying, or even evolving with time

- Molecule graphs do not change



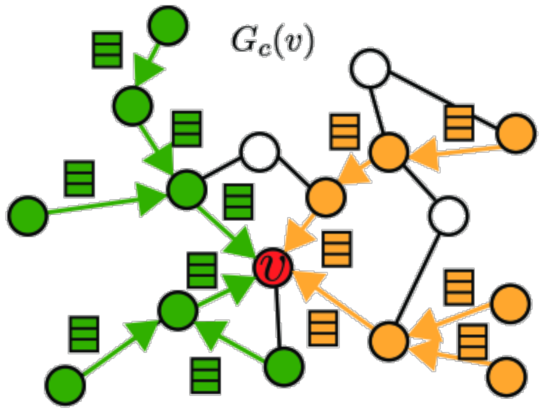
- 3D mesh grids can deform



- Graph relations between objects/people change continuously

Regular structures vs graphs

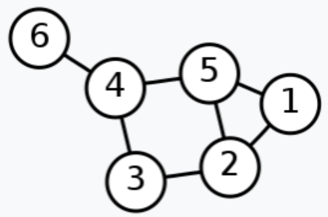
- Regular structures are subsets of graphs
  - E.g., images are grid graphs



- Convolution + pooling
- Local neighborhood: fixed window
- Constant number of neighbors
- With fixed ordering
- Translation equivariance

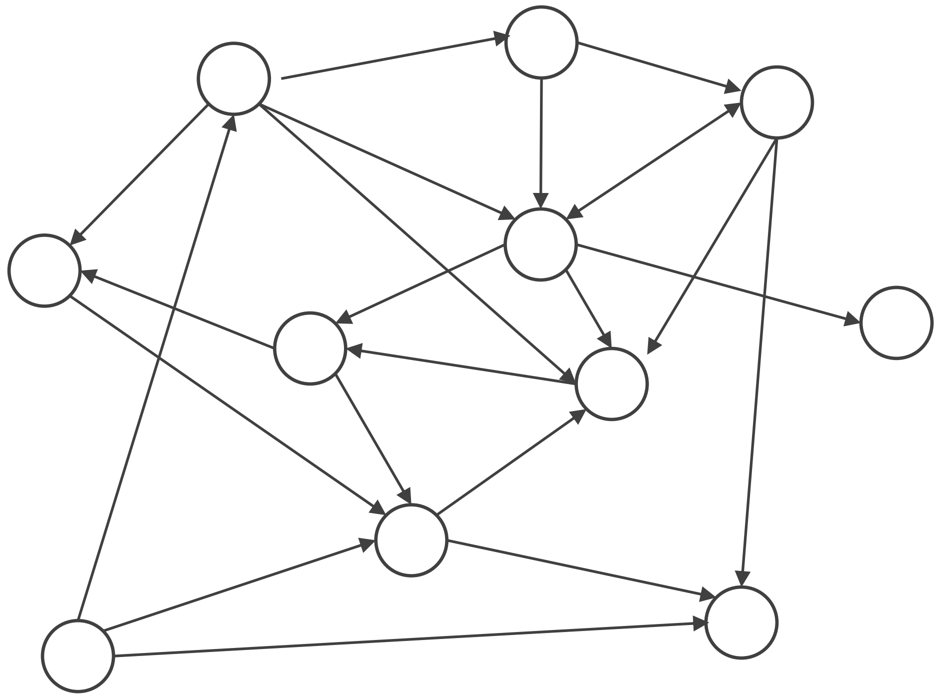
- Message passing + coarsening
- Local neighborhood: 1-hop
- Different number of neighbors
- No ordering of neighbors
- Local permutation equivariance

Revisiting graphs

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$

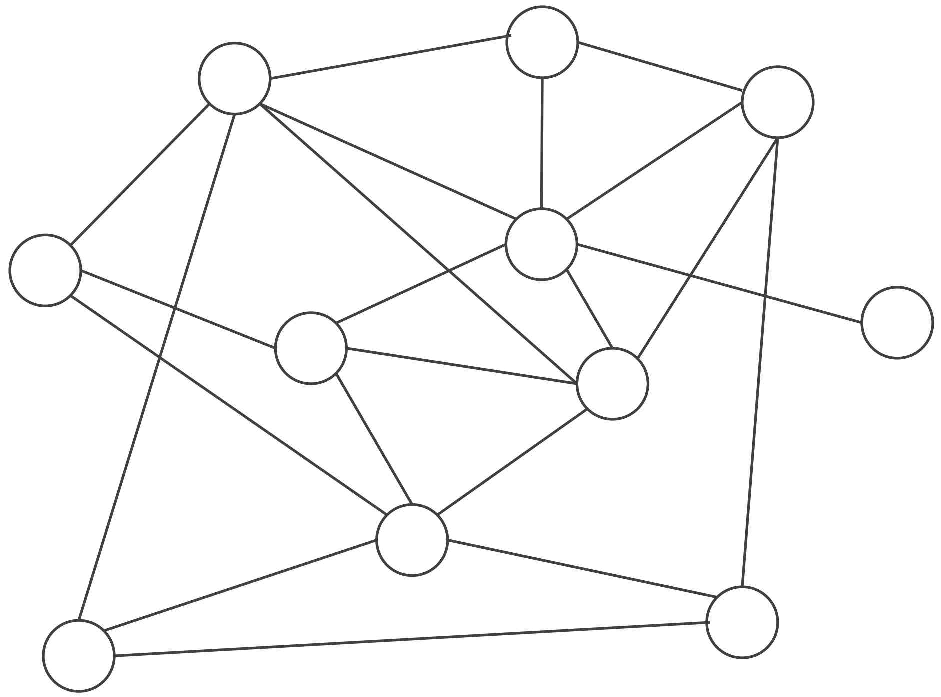
Directed graphs

- Vertices $\mathcal{V} = \{1, \dots, n\}$, also called “nodes”
- Edges $\mathcal{E} = \{(i, j): i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (directed)



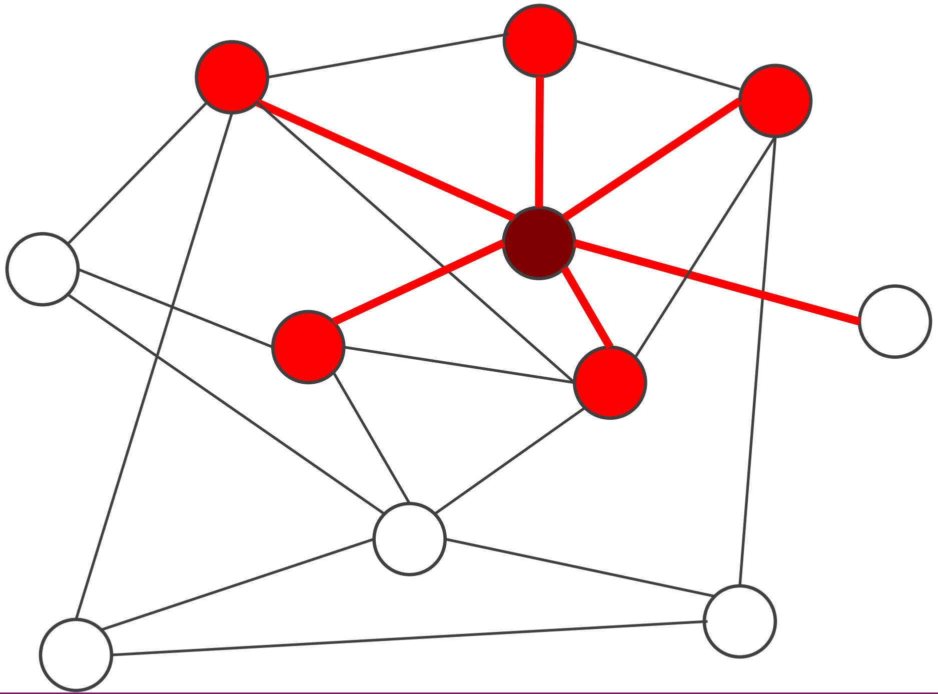
Undirected graphs

- Vertices $\mathcal{V} = \{1, \ldots, n\}$
- Edges $\mathcal{E} = \{(i, j): i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (directed)
- Edges $\mathcal{E} = \{\{i, j\}: i, j \in \mathcal{V}\} \subseteq \mathcal{V} \times \mathcal{V}$ (undirected)



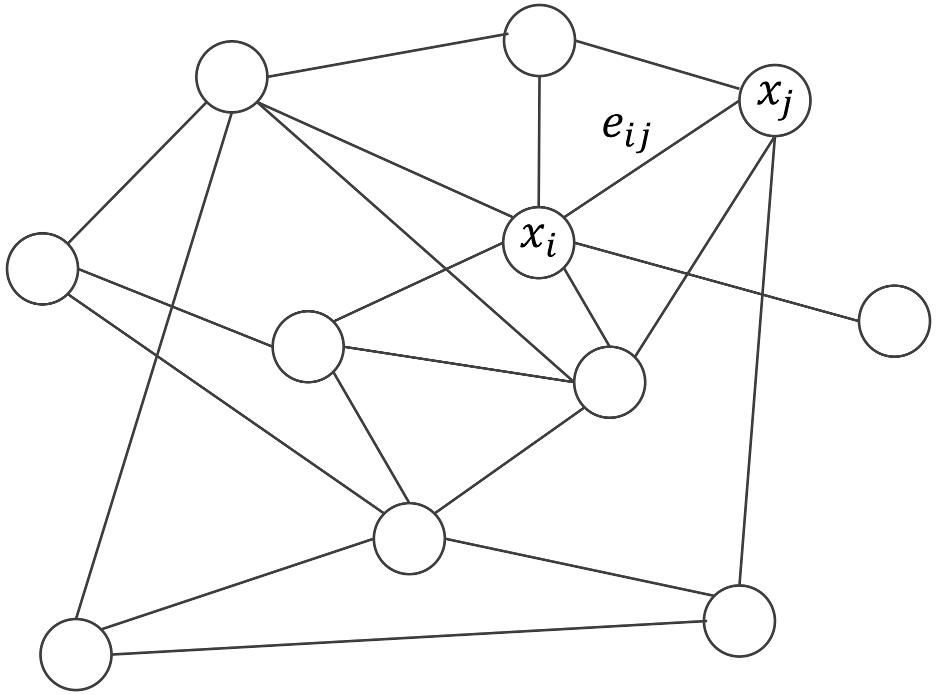
Graph neighborhood

- The neighborhood of a node consists of all nodes directly connected to it
$$\mathcal{N}(i) = \{j: (i,j) \in \mathcal{E}\}$$
- The **degree** of a node is the number of neighbors: $d_i = |\mathcal{N}(i)|$
  - The diagonal matrix D contains all degrees per node



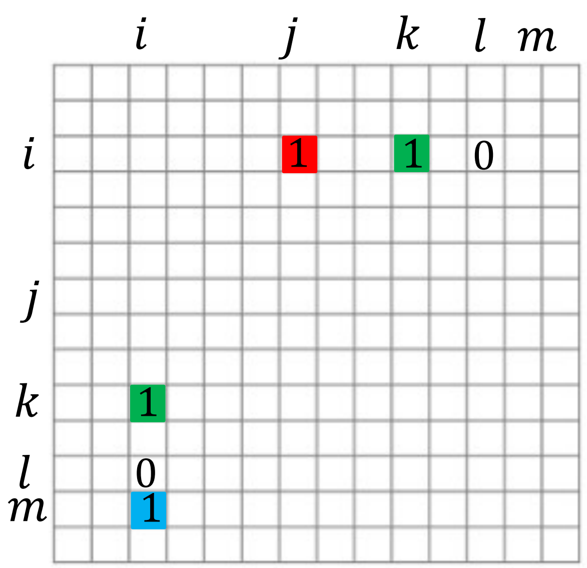
Attributes

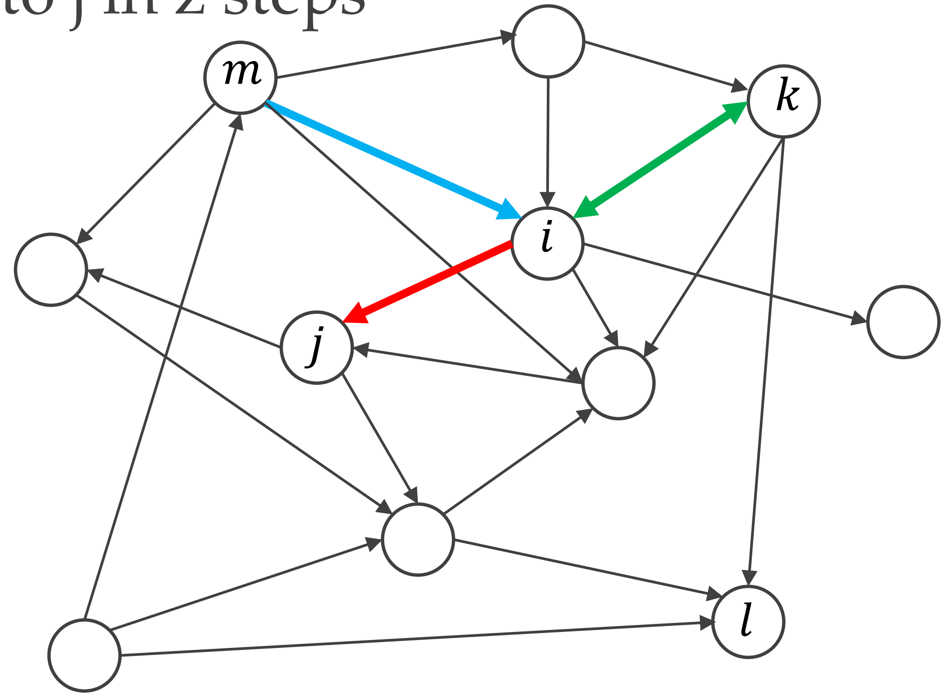
- Node features $\boldsymbol{x}: \mathcal{V} \rightarrow \mathbb{R}^d, X = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$
- Edge features $\boldsymbol{e}_{ij}: \mathcal{E} \rightarrow \mathbb{R}^{d'}$
  - If $d' \in \mathbb{N}$, e.g. d′=1, we simply have a weighted graph



Adjacency matrix

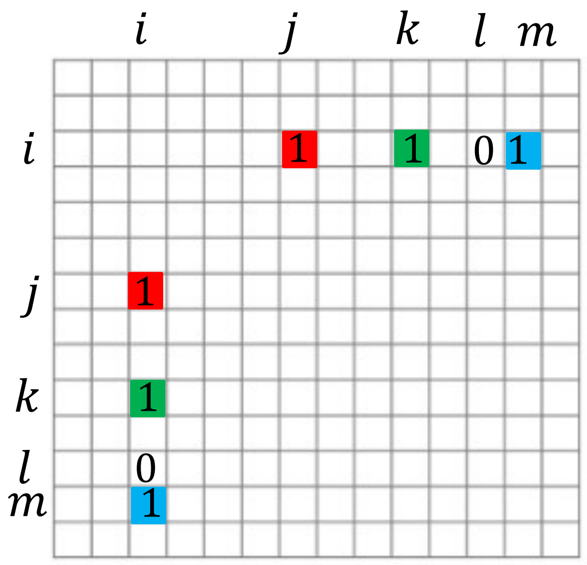
- An $n \times n$ matrix A, for n nodes
- $A_{ij} = \begin{cases} 1 \text{ if } (\mathrm{i}, \mathrm{j}) \in \mathcal{E} \\ 0 \text{ if } (\mathrm{i}, \mathrm{j}) \notin \mathcal{E} \end{cases}$
- $(A^z)_{ij}$: number of paths that go from i to j in z steps

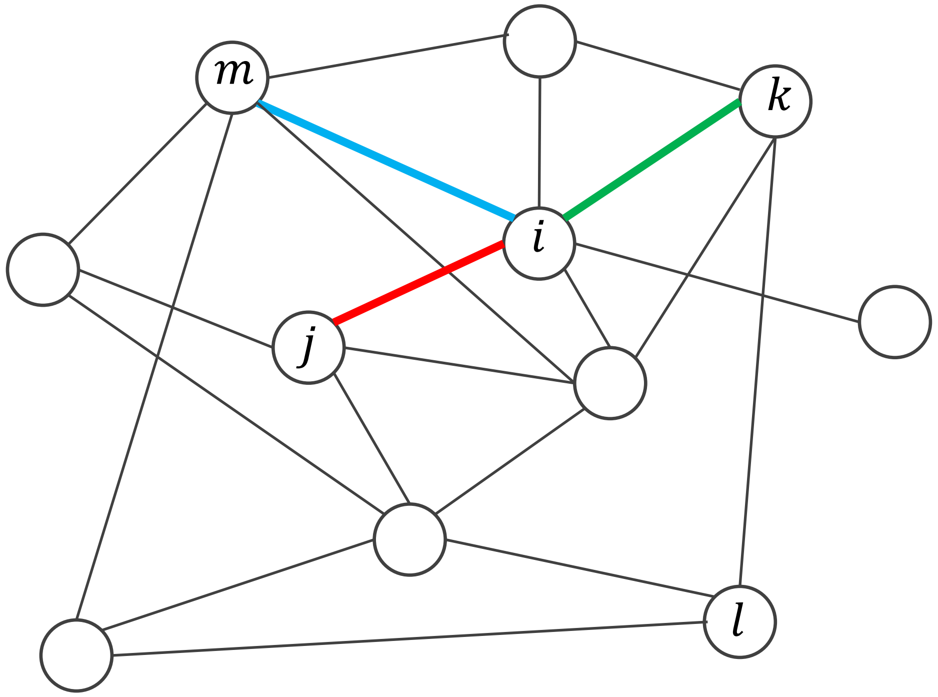




Adjacency matrix for undirected graphs

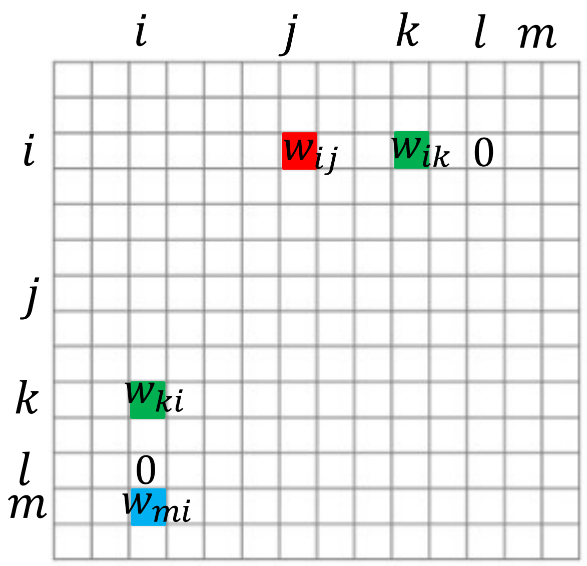
- The adjacency matrix is symmetric for undirected graphs
- Graph indexing is arbitrary: X′ = XP should yield same results

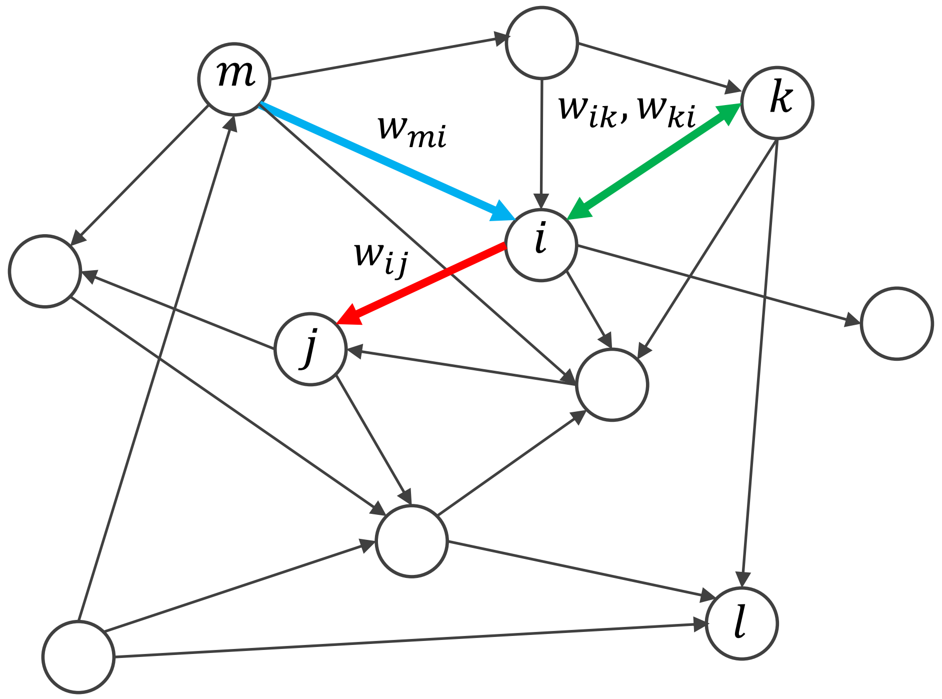




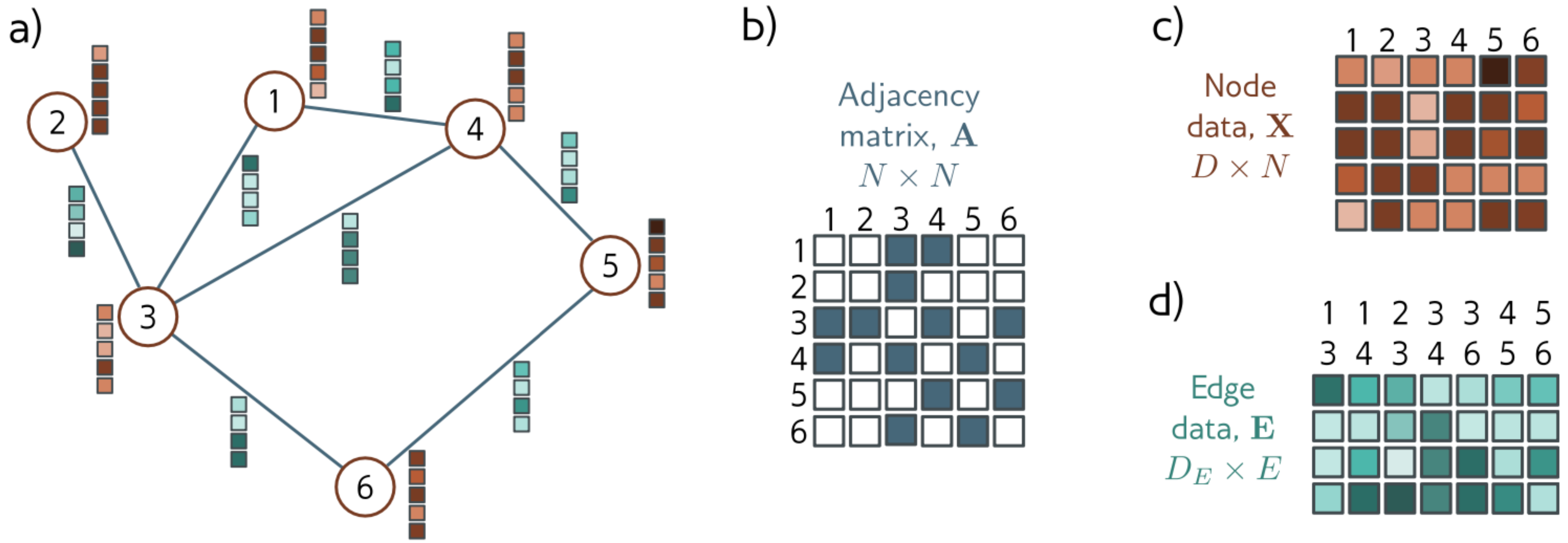
Weighted adjacency matrix

- When the edges have weights, so does the adjacency matrix





Graph representation for us



Quiz:
What does the value of $\exp(A)_{ii}$ tell us about node i?

i) Not much, as all values of exp(A) converge to the same constant number

ii) How connected node i is

iii) How unconnected node i is

iv) How many nodes are ~1.41 hops away from node i

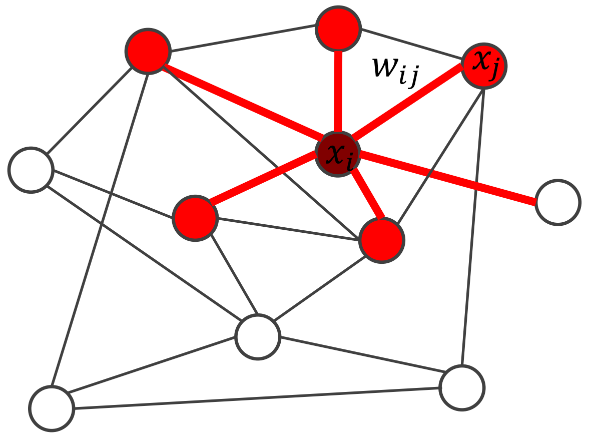
Graph Laplacian

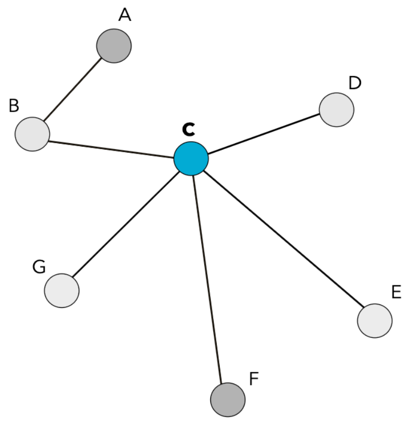
- A matrix representation of a graph: $\Delta = D - A$, with variations:
- Normalize to cancel out skewing by the degree matrix

$$\Delta = D^{-1}(D - A) = I - D^{-1}A$$

- Or for better symmetry

$$\Delta = D^{-1/2}(D - A)D^{-1/2} = I - D^{-1/2}AD^{-1/2}$$





Input Graph G

$$\begin{array}{c|ccccccc}
 & A & B & \mathbf{C} & D & E & F & G \\
\hline
A & 1 & -1 & & & & & \\
B & -1 & 2 & -1 & & & & \\
\mathbf{C} & & -1 & 5 & -1 & -1 & -1 & -1 \\
D & & & -1 & 1 & & & \\
E & & & -1 & & 1 & & \\
F & & & -1 & & & 1 & \\
G & & & -1 & & & & 1
\end{array}$$

Laplacian L of G

Graph Laplacian: meaning

- Is an operator:
- Apply this to features per node X:

  F = LX

- Has analogue meaning to applying Laplacian in images:
  - Approximation to second derivative:
  - Ie. is high when values are changing rapidly
  - More accurately: where the change of value-change is high.
- E.g. LX = 0 if X is constant:

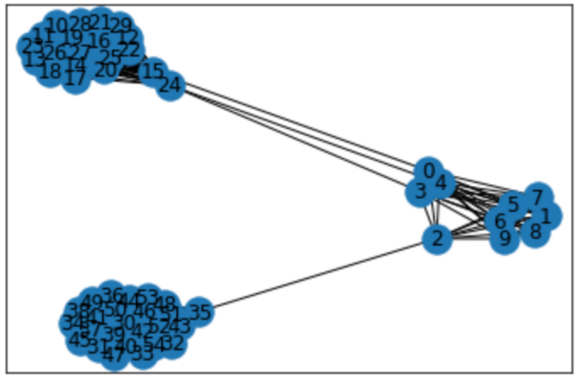
$$\begin{array}{c|ccccccc} & A & B & \mathbf{C} & D & E & F & G \\ \hline A & 1 & -1 & & & & & \\ B & -1 & 2 & -1 & & & & \\ \mathbf{C} & & -1 & 5 & -1 & -1 & -1 & -1 \\ D & & & -1 & 1 & & & \\ E & & & -1 & & 1 & & \\ F & & & -1 & & & 1 & \\ G & & & -1 & & & & 1 \end{array}$$

Applications of the Graph Laplacian

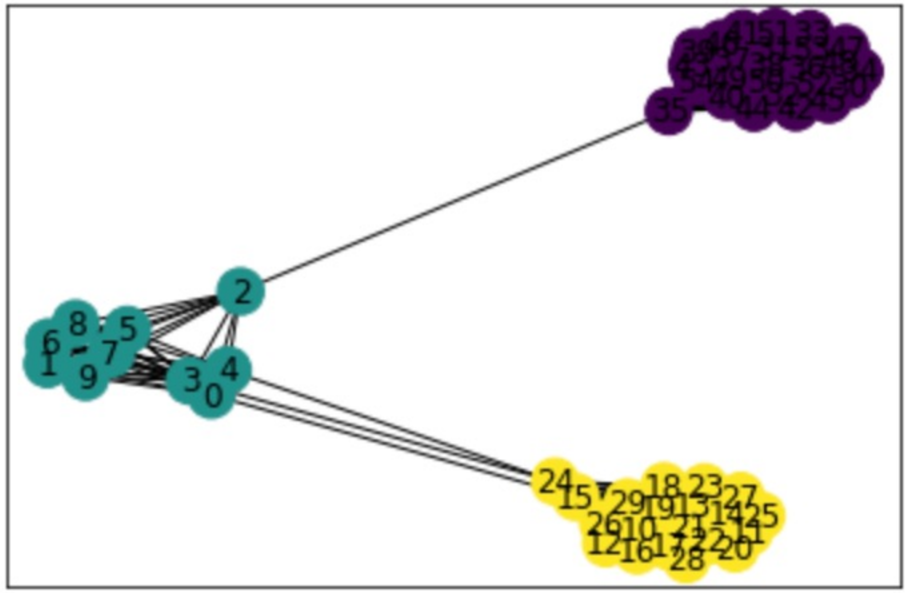
Example: community detection

1. Construct your Laplacian and calculate its Eigenvalues and Eigenvector
2. Using the Eigenvectors, that correspond to the smallest non-zero eigenvalues, as the coordinates of the nodes.
3. Run K-Means clustering on those node coordinates

As an example I made a network of three major clusters with sparse connections between different clusters.



Network of Clusters of different sizes



The results of KMeans Clustering

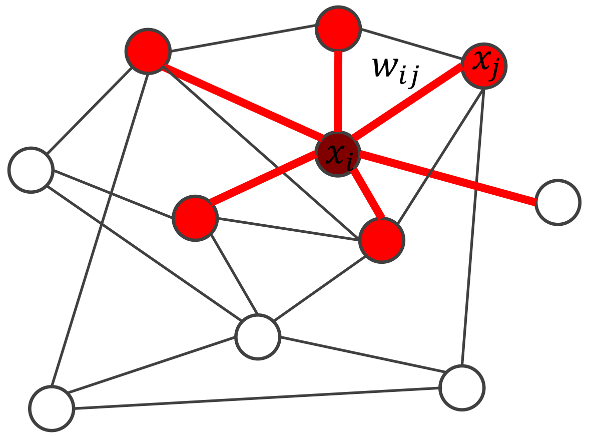
We will use it extensively for constructing convolutions on graphs later.

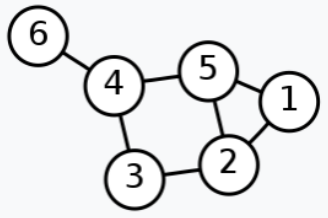
Applied Laplacian written out:

- Also called "local difference operator", note: L is also often written as Δ

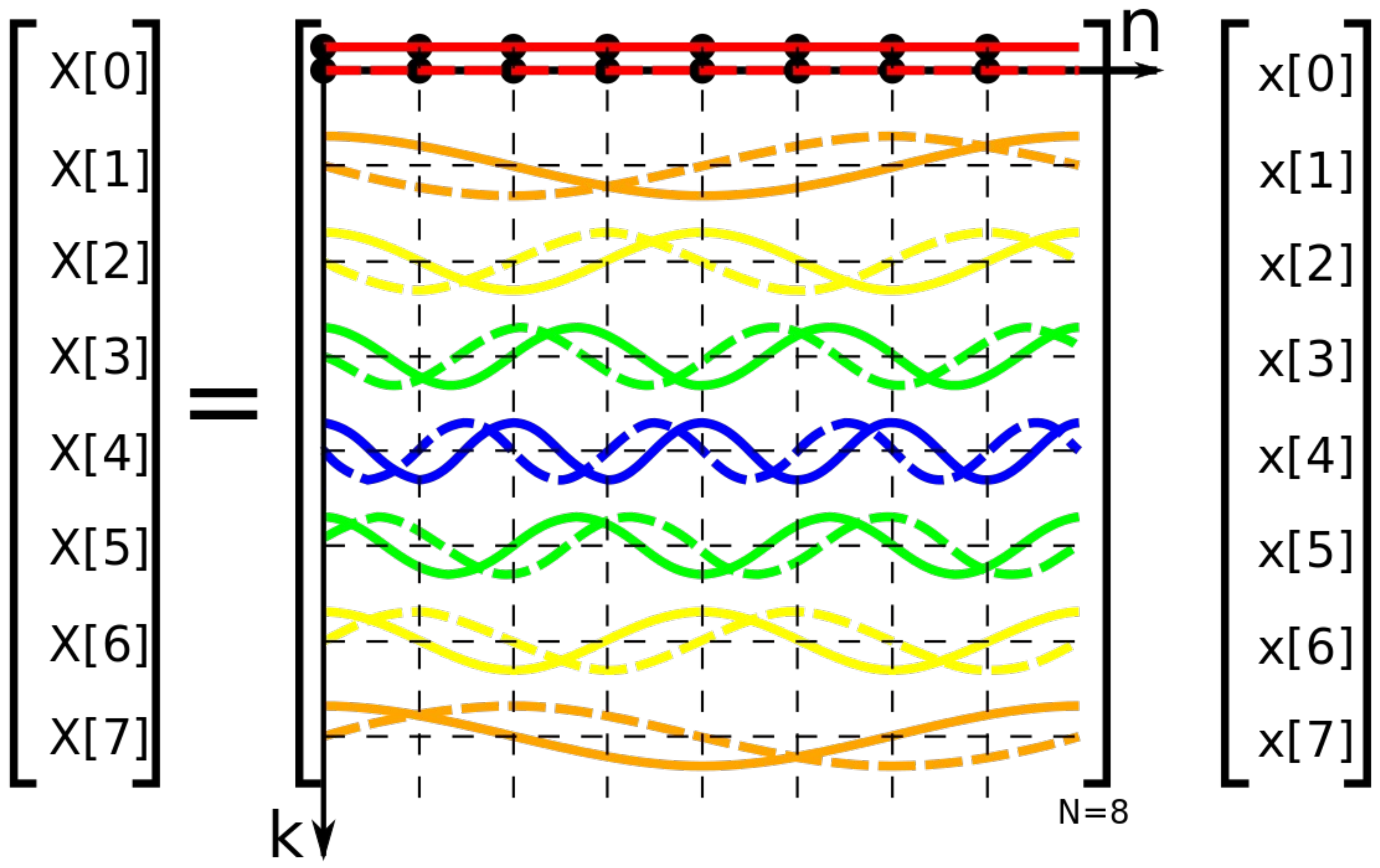
$$(\mathrm{L}\boldsymbol{x})_i = \frac{1}{d_i} \sum_{j \in \mathcal{N}(i)} w_{ij}\,(\boldsymbol{x}_i - \boldsymbol{x}_j)$$

- Notice already: a bit similar to a convolution: local neighbors, summing, (however no parameters here)



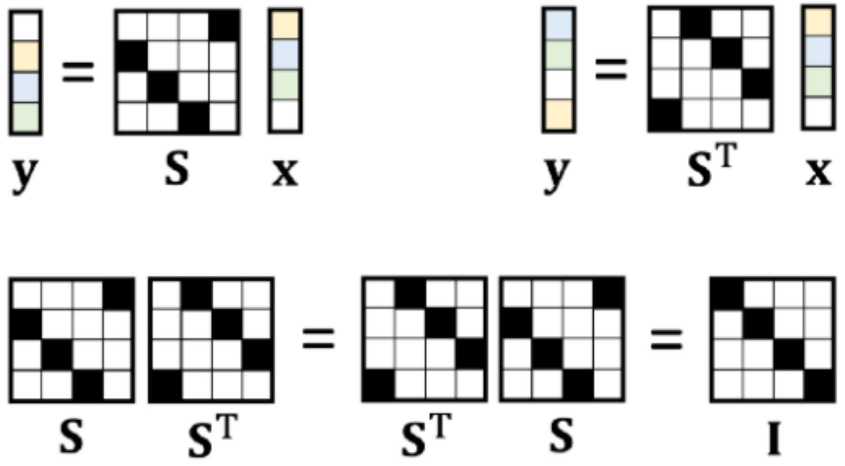
Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&3&0&0\\ 0&0&0&0&3&0\\ 0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\ 1&0&1&0&1&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&1\\ 1&1&0&1&0&0\\ 0&0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 2&-1&0&0&-1&0\\ -1&3&-1&0&-1&0\\ 0&-1&2&-1&0&0\\ 0&0&-1&3&-1&-1\\ -1&-1&0&-1&3&0\\ 0&0&0&-1&0&1 \end{pmatrix}$

Revisiting convolutions with a perspective of Equivariance



The shift operator, a special circulant matrix

- The output is simply shifted inputs
- The shift operator is circulant/Toeplitz
- For $w = [0, 1, \dots, 0]$: the right-shift operator
  - Similar to convolution with lots of zeros.
  - Result: 'shift one' to the right
  - Transpose for left-shift
- The shift operator is an orthogonal matrix (it's a type of permutation)



Now we want to know:

- Which operations C are equivariant to the shift operator*?
- Remember:

f with regards to g:

- Invariance: $f\big(g(x)\big) = f(x)$
- Equivariance: $f\big(g(x)\big) = g(f(x))$

- Ie C(S(x)) should be S(C(x))
- In our case: CSx should be SCx (ie mat muls)
- How can we arrive at a form for C?

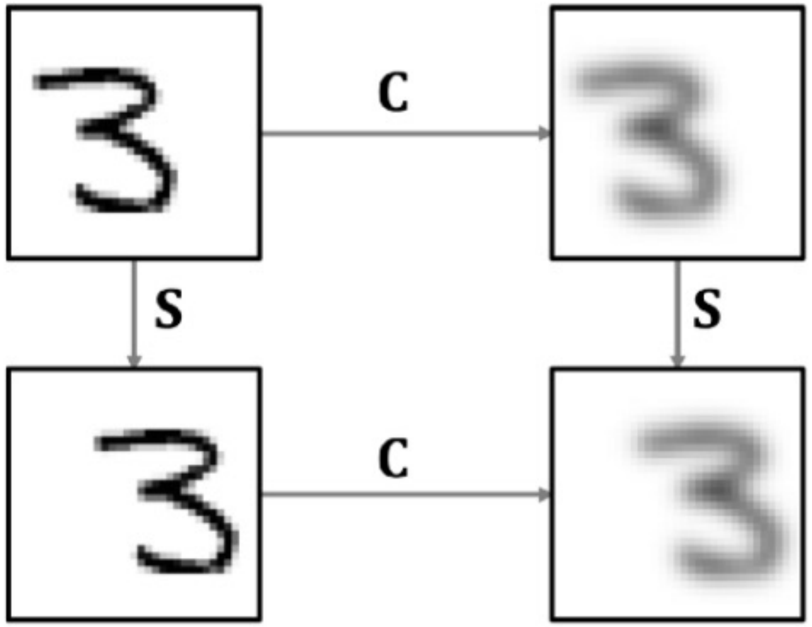


Illustration of shift equivariance as the interchangeability of shift and blur operations.

* The words operator/function/matrix are equivalent here

As it turns out: circulant matrices commute

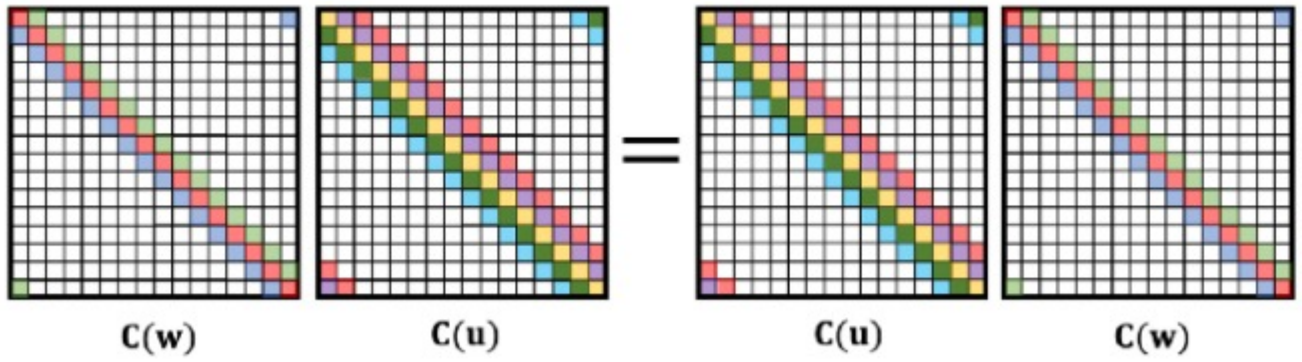
- In general (A,B are matrices):

$$A \cdot B \neq B \cdot A$$

- For circulant matrices:

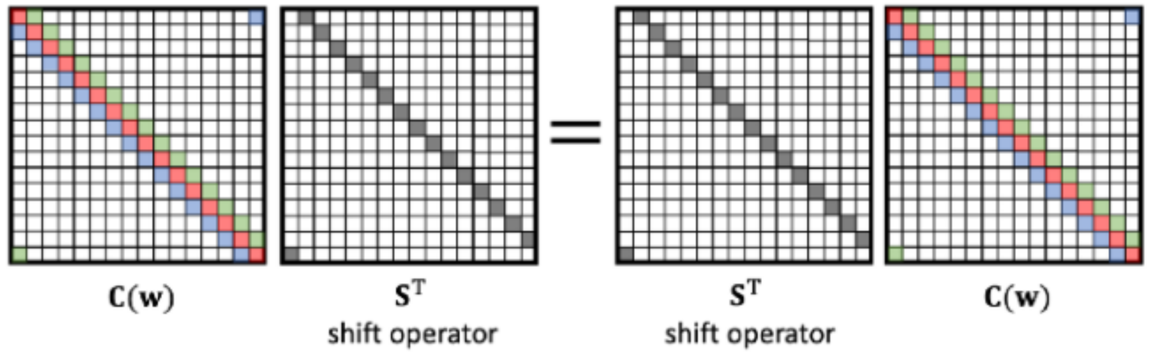
$$C(w) \cdot C(u) = C(u) \cdot C(w)$$

- Convolutions (which are circulant matrices) commute with the shift operator ⇒ 🥳!



What this means: Translation equivariance ⇒ circulant matrices/convolutions

- Circulant matrices enable translation equivariance
  - Change the location of the input
  - The results will be the same (but shifted)



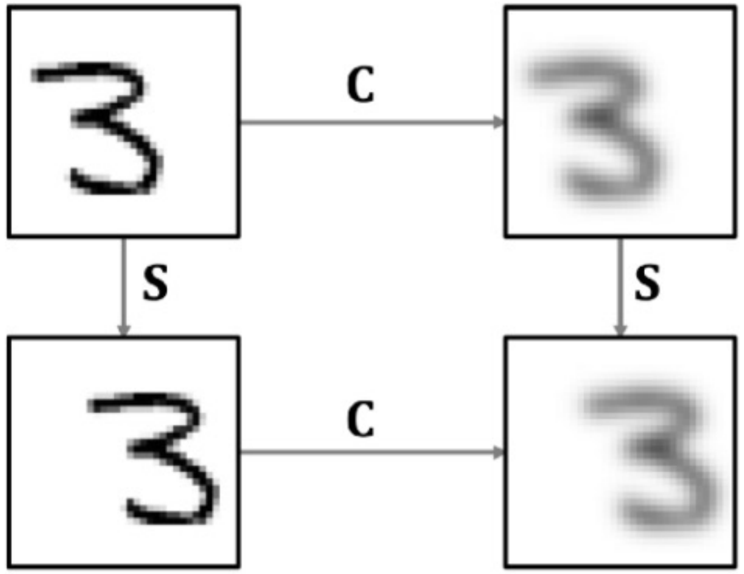


Illustration of shift equivariance as the interchangeability of shift and blur operations.

- In fact convolutions result from equivariance, linearity and locality
- Remember however: CNNs not (quite) equivariant.

Where we are

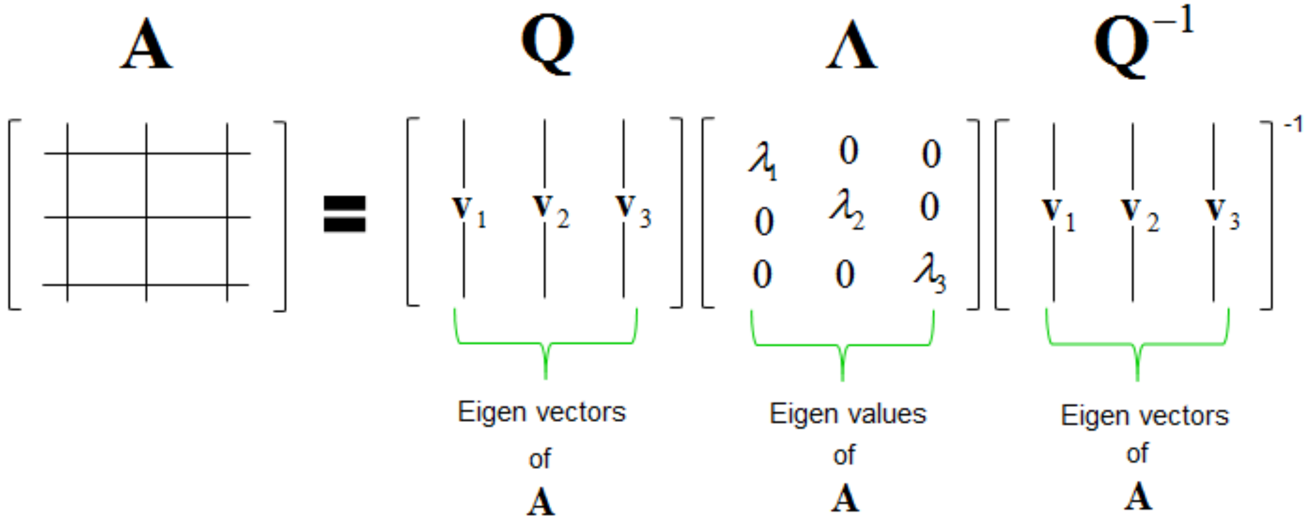
- Want: translation equivariance
  - Introduce shift operator
  - Use the fact that circulant matrices commute
  - To arrive at the result that convolutions fulfill these requirements
- Next:
  - Go one step deeper by using circulant matrices' eigenvectors

Maths: All circulant matrices have the same eigenvectors!

https://github.com/mitmath/1806/blob/master/lectures/Circulant-Matrices.ipynb

- The eigenvalues of the circulant matrices are different
- But the eigenvectors always the same!
  - The eigenvectors of the "translation transformation/operator"
- Let's verify this…

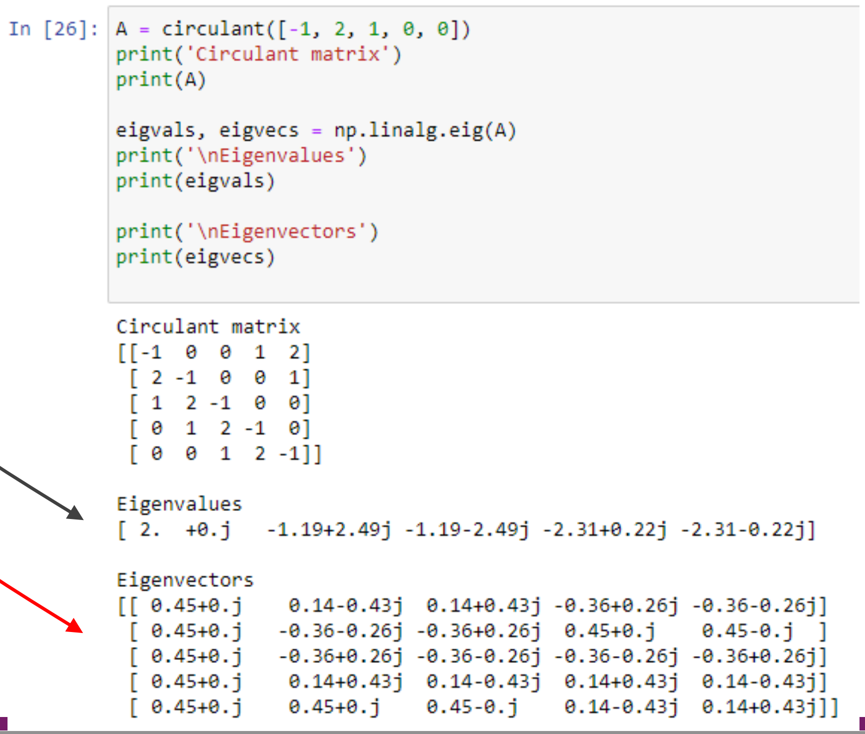
Reminder:



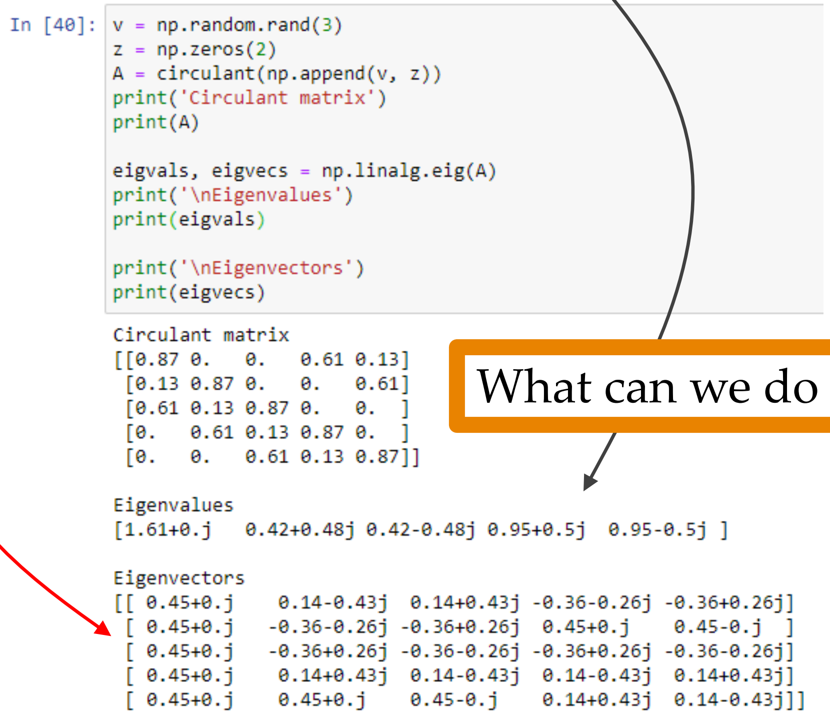
All circulant matrices have the same eigenvectors!

https://github.com/mitmath/1806/blob/master/lectures/Circulant-Matrices.ipynb

- The eigenvalues of the circulant matrices are different
- But the eigenvectors always the same!
  - The eigenvectors of the "translation transformation/operator"



```
In [26]: A = circulant([-1, 2, 1, 0, 0])
         print('Circulant matrix')
         print(A)

         eigvals, eigvecs = np.linalg.eig(A)
         print('\nEigenvalues')
         print(eigvals)

         print('\nEigenvectors')
         print(eigvecs)

Circulant matrix
[[-1  0  0  1  2]
 [ 2 -1  0  0  1]
 [ 1  2 -1  0  0]
 [ 0  1  2 -1  0]
 [ 0  0  1  2 -1]]

Eigenvalues
[ 2.  +0.j   -1.19+2.49j -1.19-2.49j -2.31+0.22j -2.31-0.22j]

Eigenvectors
[[ 0.45+0.j    0.14-0.43j  0.14+0.43j -0.36+0.26j -0.36-0.26j]
 [ 0.45+0.j   -0.36-0.26j -0.36+0.26j  0.45+0.j    0.45-0.j  ]
 [ 0.45+0.j   -0.36+0.26j -0.36-0.26j -0.36-0.26j -0.36+0.26j]
 [ 0.45+0.j    0.14+0.43j  0.14-0.43j  0.14+0.43j  0.14-0.43j]
 [ 0.45+0.j    0.45+0.j    0.45-0.j    0.14-0.43j  0.14+0.43j]]
```



```
In [40]: v = np.random.rand(3)
         z = np.zeros(2)
         A = circulant(np.append(v, z))
         print('Circulant matrix')
         print(A)

         eigvals, eigvecs = np.linalg.eig(A)
         print('\nEigenvalues')
         print(eigvals)

         print('\nEigenvectors')
         print(eigvecs)

Circulant matrix
[[0.87 0.   0.   0.61 0.13]
 [0.13 0.87 0.   0.   0.61]
 [0.61 0.13 0.87 0.   0.  ]
 [0.   0.61 0.13 0.87 0.  ]
 [0.   0.   0.61 0.13 0.87]]

Eigenvalues
[1.61+0.j   0.42+0.48j 0.42-0.48j 0.95+0.5j  0.95-0.5j ]

Eigenvectors
[[ 0.45+0.j    0.14-0.43j  0.14+0.43j -0.36-0.26j -0.36+0.26j]
 [ 0.45+0.j   -0.36-0.26j -0.36+0.26j  0.45+0.j    0.45-0.j  ]
 [ 0.45+0.j   -0.36+0.26j -0.36-0.26j -0.36+0.26j -0.36-0.26j]
 [ 0.45+0.j    0.14+0.43j  0.14-0.43j  0.14-0.43j  0.14+0.43j]
 [ 0.45+0.j    0.45+0.j    0.45-0.j    0.14+0.43j  0.14-0.43j]]
```

What can we do with this?

Circulant eigenvectors ⇔ Shift eigenvectors

- All circulant matrices have the same eigenvectors (up to ordering)
  The shift operator is a circulant matrix
- So: The circulant eigenvectors are the eigenvectors of shift
- Any convolution with any filter $\boldsymbol{w}$ involves the same eigenvectors!
- We can use this for computing convolutions differently & getting a novel perspective on them.

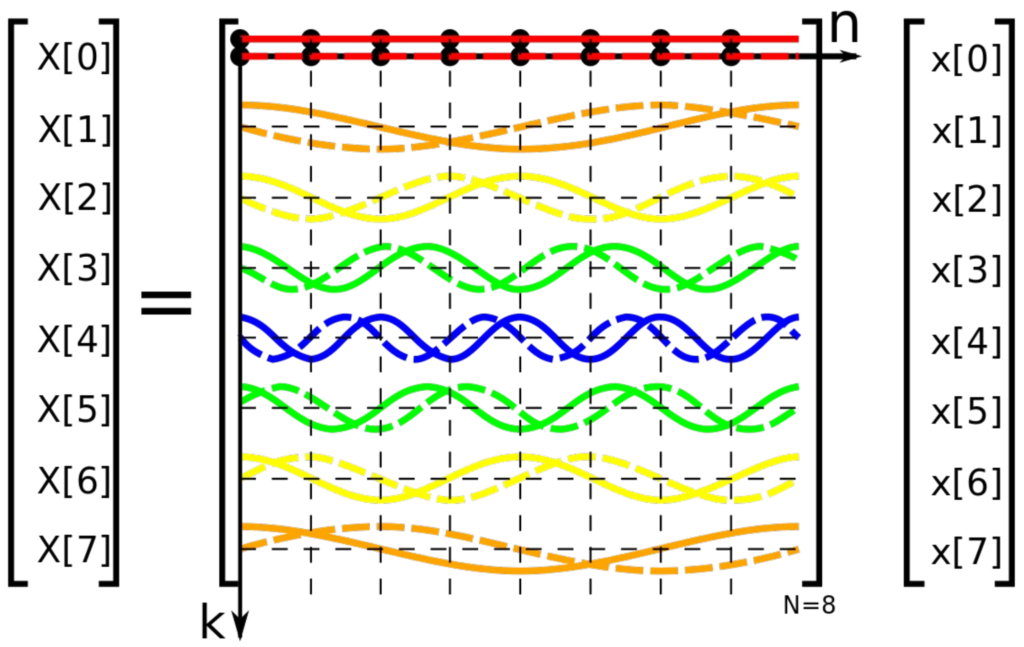
But first: What are the eigenvectors of the shift operator

- The k-th eigenvector of $n \times n$ circulant

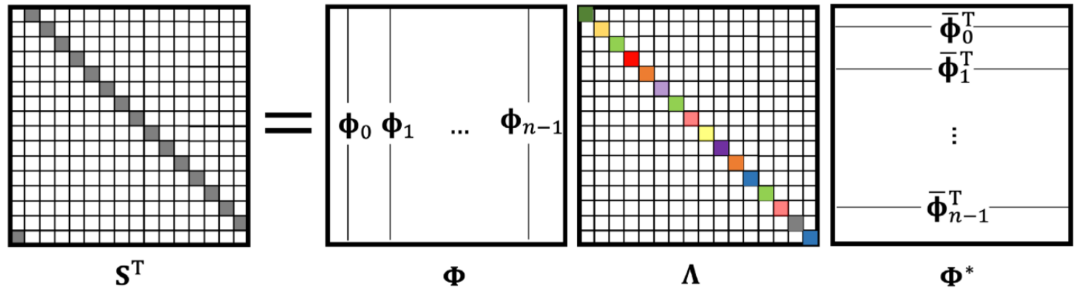
$$e^{(k)} = \begin{bmatrix} \omega_n^{0\cdot k} \\ \omega_n^{1\cdot k} \\ \omega_n^{2\cdot k} \\ \vdots \\ \omega_n^{(n-1)\cdot k} \end{bmatrix}, \text{where } \omega_\text{n} = \exp(\frac{2\pi \cdot i}{n})$$

- Collecting all eigenvectors*

$$\Phi = [e^{(0)} \quad e^{(1)} \quad e^{(2)} \quad \dots \quad e^{(n-1)}]$$



So the shift operator can be written as:



*This is the Discrete Fourier Transform

Computing a convolution *in the frequency domain*

- Thus write a convolution as:

$$\boldsymbol{x} * \boldsymbol{w} = C(\boldsymbol{w}) \cdot \boldsymbol{x}$$

Eigenvectordecomposition $\rightarrow$

$$= \boxed{\left(\boldsymbol{\Phi} \cdot \boldsymbol{\Lambda}(\mathbf{w}) \cdot \boldsymbol{\Phi}^*\right)} \cdot \boldsymbol{x}$$

$$= \underbrace{\boldsymbol{\Phi} \cdot \Big(\underbrace{\boldsymbol{\Lambda}(\mathbf{w}) \cdot (\underbrace{\boldsymbol{\Phi}^* \cdot \boldsymbol{x}}_{\text{Discrete Fourier Transform (with DFT matrix)}})}_{\text{Row-wise multiply with eigenvalues of weight matrix}}\Big)}_{\text{Inverse Discrete Fourier Transform (with Inverse DFT matrix)}}$$

Convolution Theorem

- The Fourier of a convolution is equal to product of individual Fouriers

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \odot \mathcal{F}\{g\} \Rightarrow f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \odot \mathcal{F}\{g\}\}$$

- Convolution in "space" domain is equivalent to matrix multiplication in "frequency/spectral" domain
  - Frequency defined by Fourier bases $\exp(-\frac{i2\pi}{\mathrm{N}} \cdot kn)$
- Discrete case $\mathcal{F}\{f\}$ becomes a matrix multiplication with shift DFT matrix

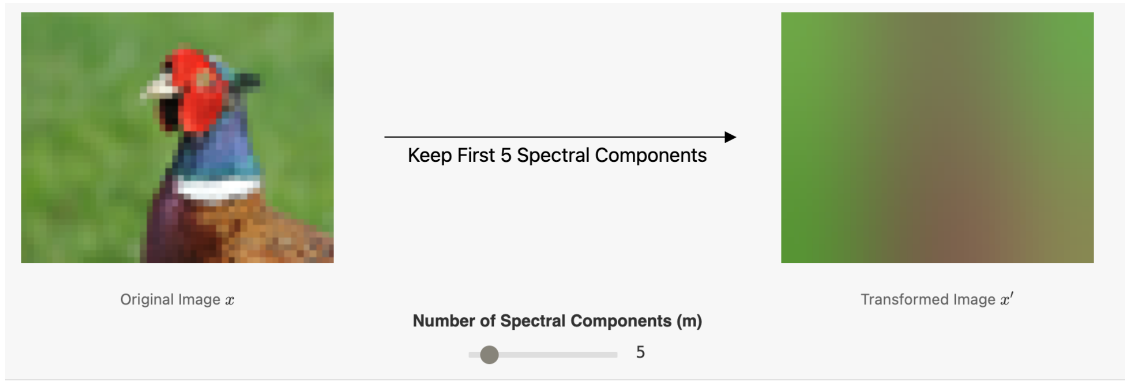
$$\boldsymbol{w} * \boldsymbol{x} = \underbrace{\boldsymbol{\Phi}^{-1}\left(\underbrace{\boldsymbol{\Lambda}(\mathbf{w})}_{\mathcal{F}\{f\}} \cdot \underbrace{(\boldsymbol{\Phi} \cdot \boldsymbol{x})}_{\mathcal{F}\{g\}}\right)}_{\mathcal{F}^{-1}}$$

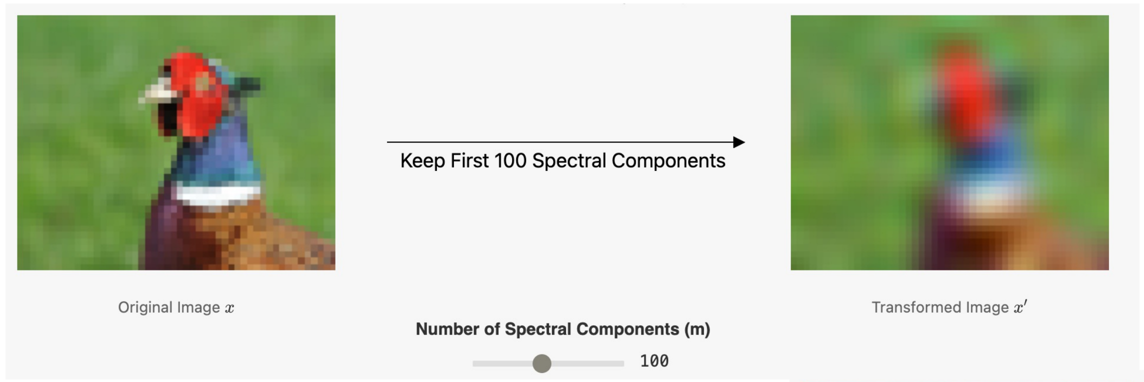
Convolution theorem

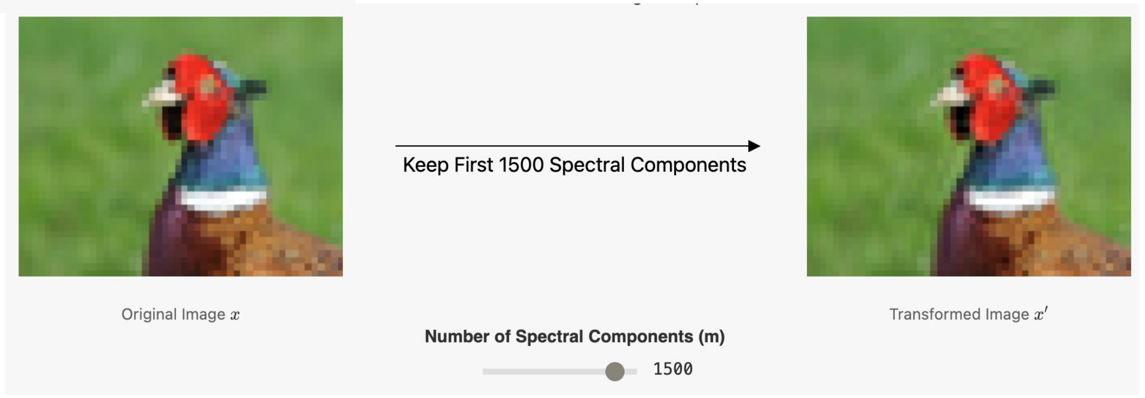
$$\begin{tikzcd}
\mathbf{y} \arrow[d, "\mathbf{\Phi}^{*}\ \text{FFT}"] & & \mathbf{x} \star \mathbf{y} \\
\hat{\mathbf{y}} \arrow[rr, "\begin{bmatrix} \hat{x}_1 & & \\ & \ddots & \\ & & \hat{x}_n \end{bmatrix}"'] & & \widehat{\mathbf{x} \star \mathbf{y}} \arrow[u, "\text{IFFT}\ \mathbf{\Phi}"']
\end{tikzcd}$$

Element-wise product

Frequency representation:

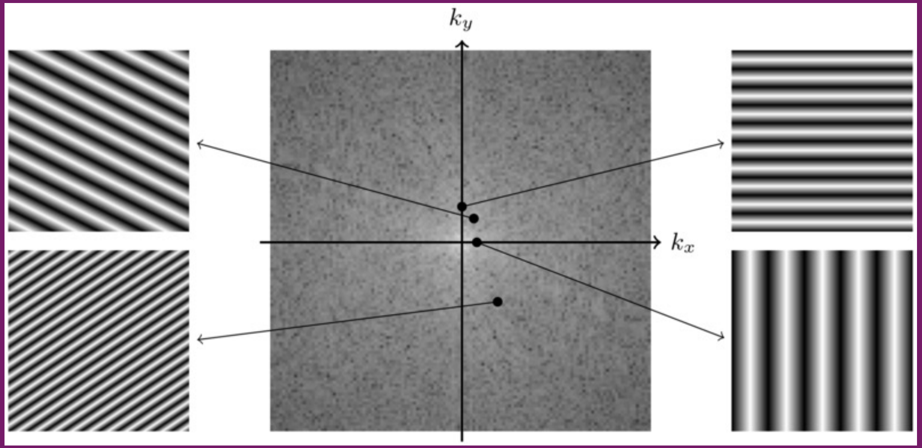

Keep First 5 Spectral Components
Original Image x
Transformed Image x'
Number of Spectral Components (m)
5


Keep First 100 Spectral Components
Original Image x
Transformed Image x'
Number of Spectral Components (m)
100

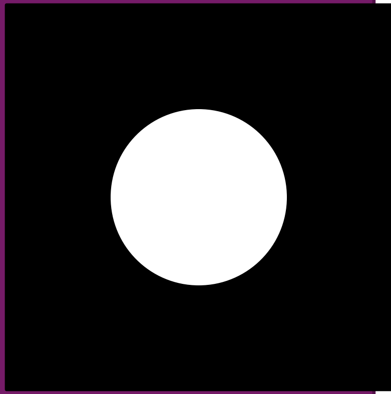

Keep First 1500 Spectral Components
Original Image x
Transformed Image x'
Number of Spectral Components (m)
1500

- Treat image as graph: either 3,5 or 8 neighbors.
- Now transform to frequency domain
- By only keeping some components in the frequency domain and moving back to image space, we get a feel of what it's covering (frequencies)

Quiz: Remember the Fourier transform for images:

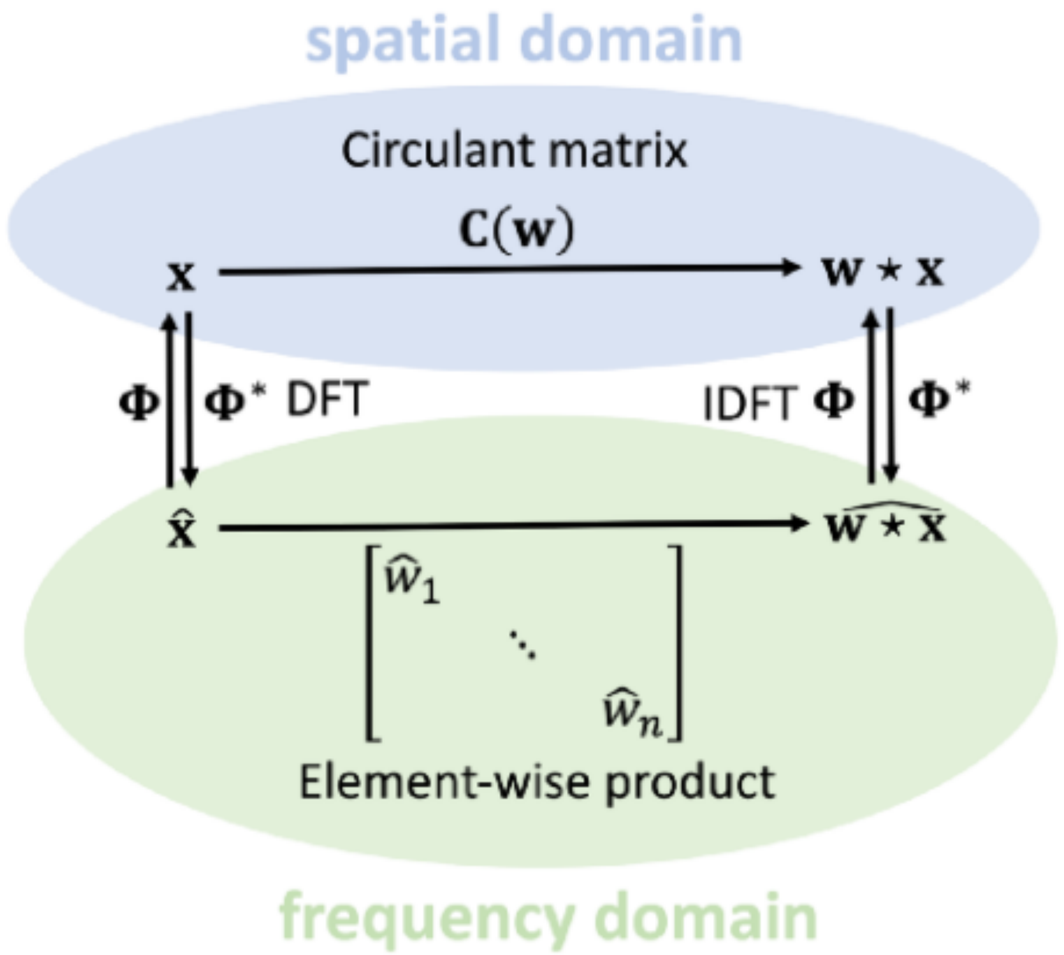


What happens if multiply this spectrum with a filter like this ->
and transform it back to the spatial domain?



1) The image will be more bright
2) The image will be less sharp
3) The outer parts of the image will be darker

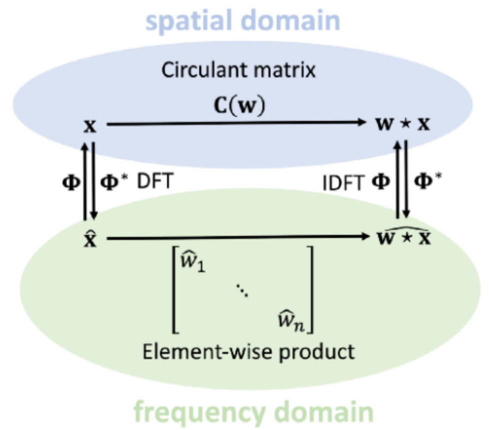
Convolution theorem: $\boldsymbol{x} * \boldsymbol{w} = \boldsymbol{\Phi} \cdot \left(\boldsymbol{\Lambda}(\mathbf{w}) \cdot (\boldsymbol{\Phi}^* \cdot \boldsymbol{x})\right)$



https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

Implications

- If we can compute (inverse) Fouriers and their inverse fast, then we save time (whether conv or DFT is used is determined in CUDA automatically)
- Fast Fourier Transform (FFT): A faster version of DFT to get Φ
  - $O(n \log n)$ vs $O(n^2)$
  - Replace sliding window convolutions with very fast matrix multiplications
- Convolution as result of equivariance



https://towardsdatascience.com/deriving-convolution-from-first-principles-4ff124888028

If translation equivariance leads to CNNs, what else is there?

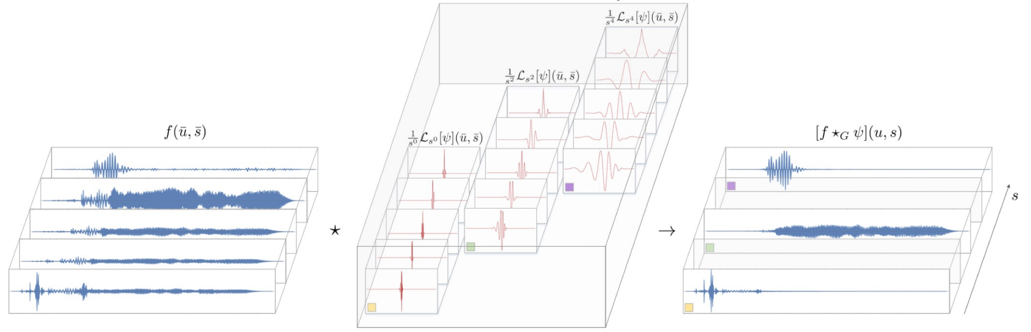
- What we take from this:
  - convolutions can be computed in spectral domain in a simple fashion
  - thanks to a useful eigenspace that diagonalizes *all* convolutions
- Can we generalize to other equivariances beyond translation?

A large field: Group Equivariant Deep Learning

Group convolutional neural networks[1] (G-CNNs) can improve over classical CNNs by:

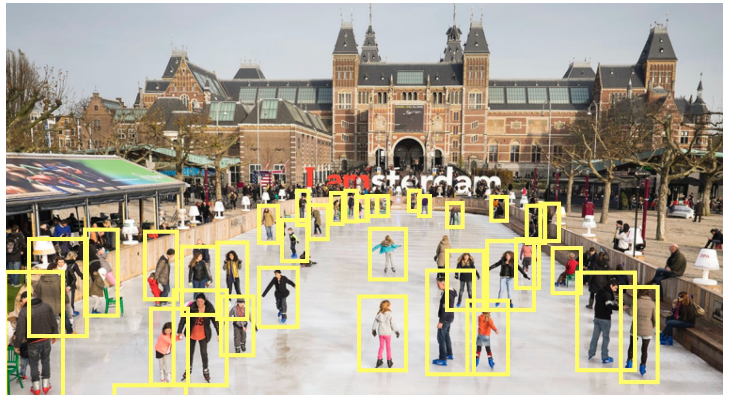
- Allowing weight sharing beyond just translations
- Increasing data efficiency

Symmetries in audio[5]
(translation, scale/pitch)

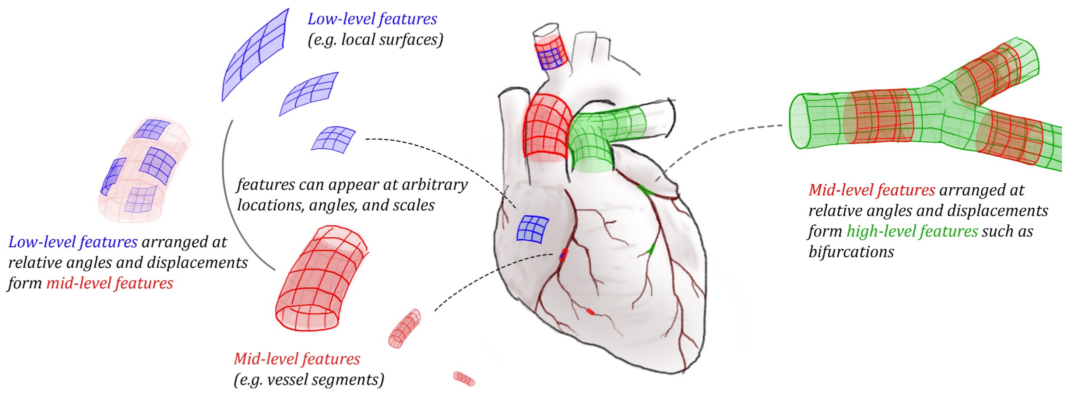


$f(\bar{u}, \bar{s})$
$\frac{1}{s^0}\mathcal{L}_{s^0}[\psi](\bar{u}, \bar{s})$
$\frac{1}{s^2}\mathcal{L}_{s^2}[\psi](\bar{u}, \bar{s})$
$\frac{1}{s^4}\mathcal{L}_{s^4}[\psi](\bar{u}, \bar{s})$
$\star$
$\rightarrow$
$[f \star_G \psi](u, s)$
s

Symmetries in computer vision[3,4]
(translation, scale, rotation, perspective)

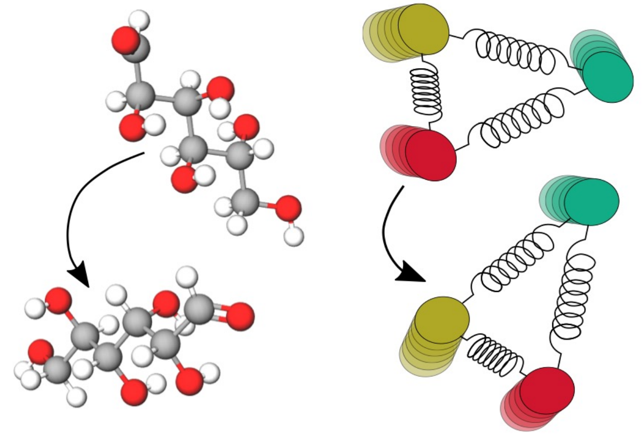


Symmetries in medical image analysis[2,3]
(translation, rotation, scale)



Low-level features
(e.g. local surfaces)
features can appear at arbitrary locations, angles, and scales
Low-level features arranged at relative angles and displacements form mid-level features
Mid-level features
(e.g. vessel segments)
Mid-level features arranged at relative angles and displacements form high-level features such as bifurcations

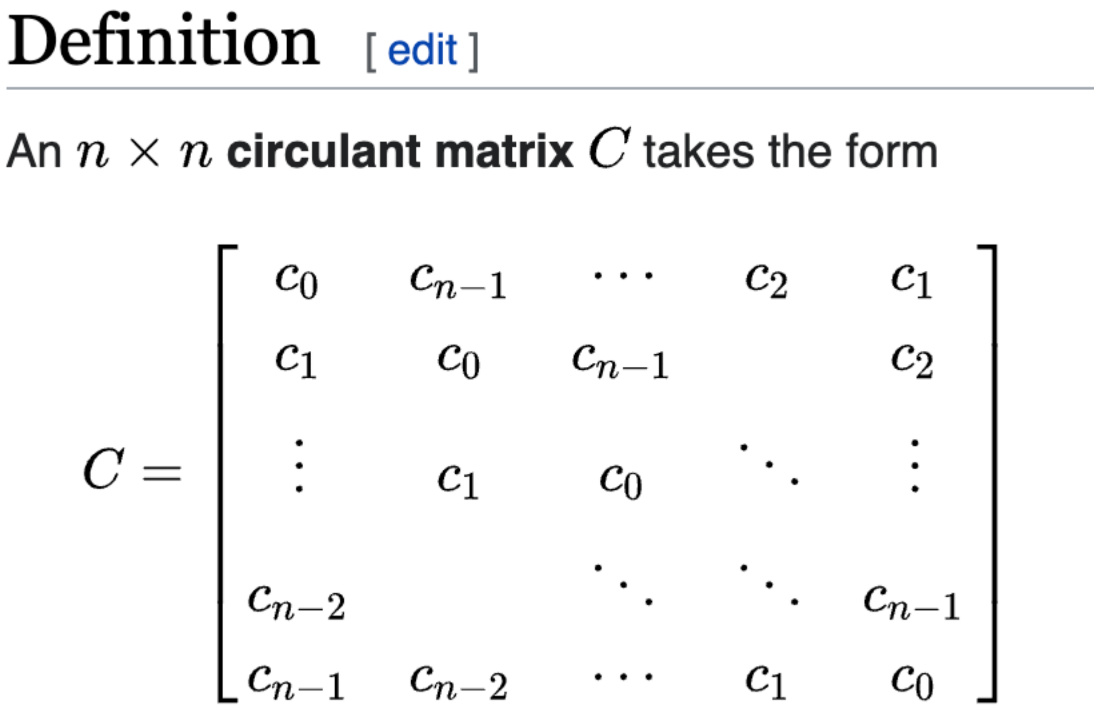
Molecular and Physical systems[6]
(translation, rotation, reflection)



[1] Cohen and Welling "Group equivariant convolutional networks" ICML 2016. [2] Bekkers and Lafarge et al. "Roto-translation covariant convolutional networks for medical image analysis." MICCAI 2018. [3] Bekkers "B-spline CNNs on Lie groups" ICLR 2020 [4] Sosnovik, Szmaja, and Smeulders "Scale-equivariant steerable networks." ICLR 2020 [5] Romero, Bekkers, Tomczak, Hoogeboom "Wavelet Networks: Scale Equivariant Learning From Raw waveforms." arXiv:2006.05259 (2020). [6] Finzi, Marc, et al. "Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data." ICML 2020.

Circulant matrices

- Thanks Wikipedia!



Definition [edit]

An $n \times n$ **circulant matrix** C takes the form

$$C = \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}$$

How research gets done part 6

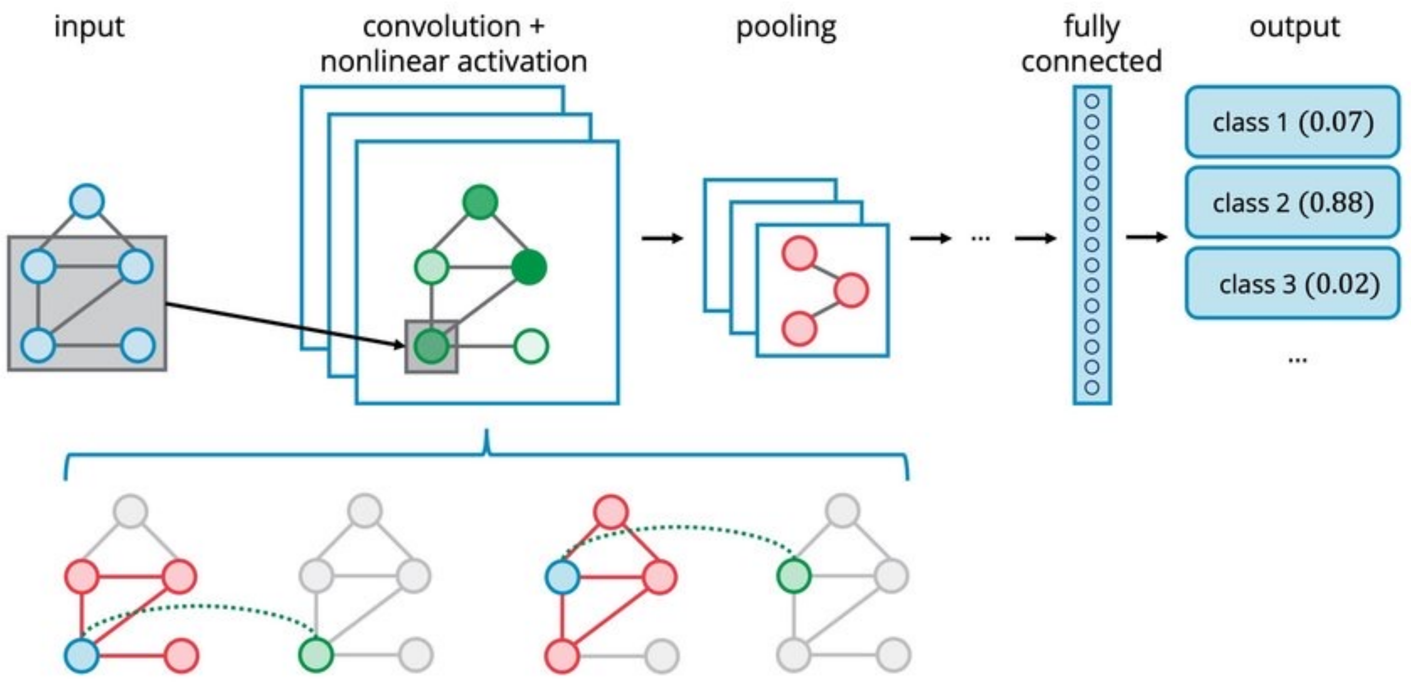
"I was lucky…"

Previous parts:
[fundamental understanding/read papers, how-to-read-papers, implement & tinker with code, realise and seek *funny* moments, MVP/principles/benchmarks/baselines]

Today:

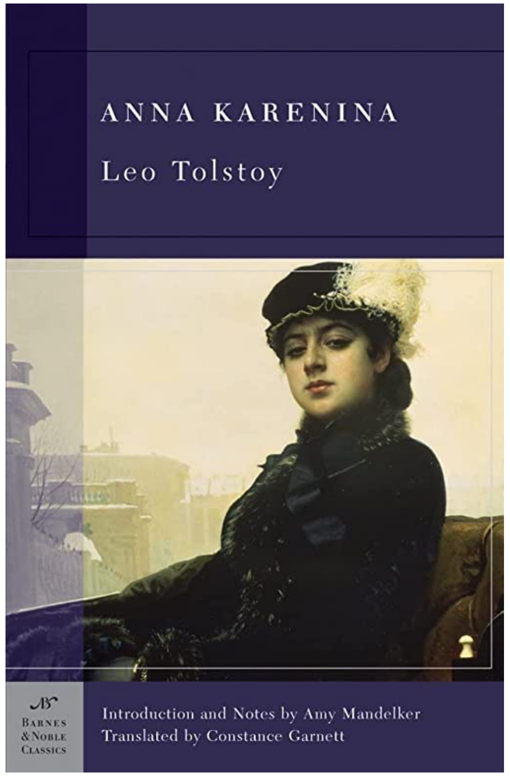
- When to give up and try something else?
  - Impossible to answer: different for everyone.
  - Possible factors to consider:
    - impact *vs.* work-required tradeoff
    - Amount of fun-while-working-on-it
    - Existence of small progresses
    - Opportunity costs (*what could you be doing instead)*
      - These can be big in a field like deep learning
- The more familiar you become in a topic, the more ideas you will get
- Having ideas is almost never the bottleneck after some point
- Evaluating the quality of ideas… requires intuition, which is developed with time

Spectral graph convolutions



From convolutions to spectral graph convolutions

- What is the equivalent of Fourier basis for graphs?
- *Images are always "connected" alike; every graph is unconnected in its own way**
- Idea: use the graph Laplacian or Adjacency as basis

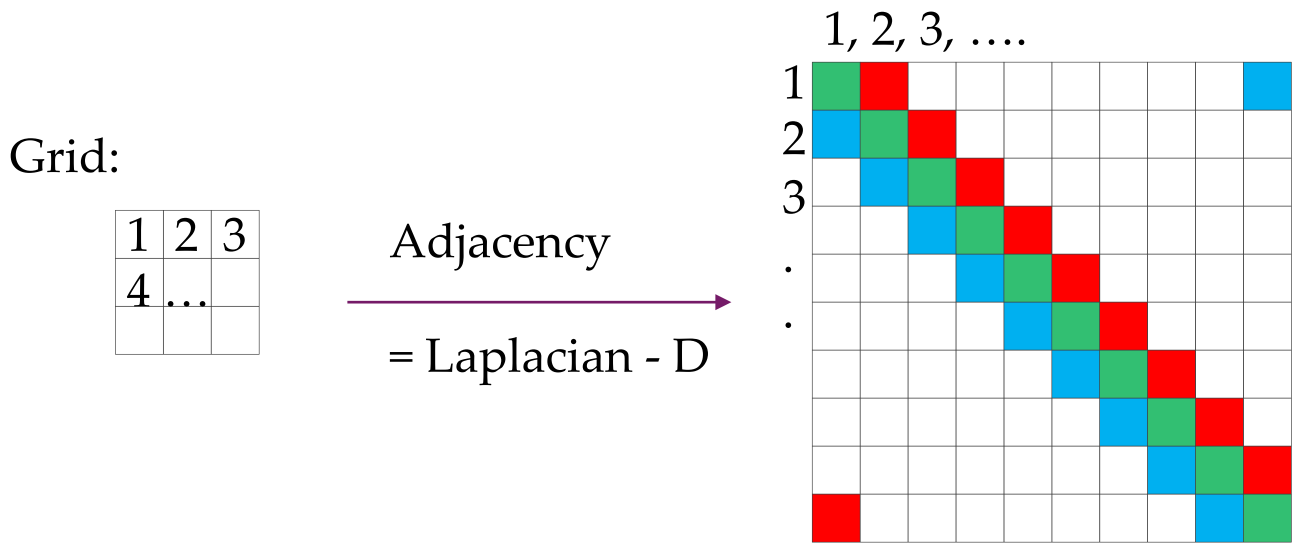

ANNA KARENINA
Leo Tolstoy
Barnes & Noble Classics
Introduction and Notes by Amy Mandelker
Translated by Constance Garnett

After the quote:
"Happy families are all alike; every unhappy family is unhappy in its own way." of Tolstoy

Approach: Use Eigenvectors of Graph Laplacian to replace Fourier

- Eigenvectors of Graph Laplacian as analogy to Fourier Transform.
  - Why?
  - Equivalent on grids:

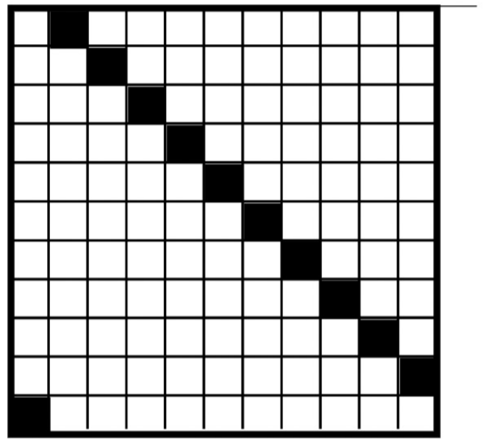
red = right neighbor, blue = left neighbor, green = 0

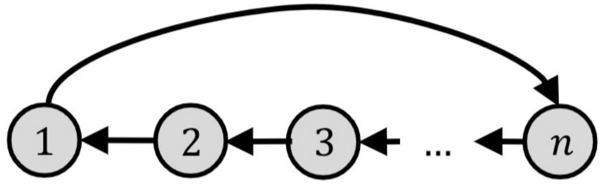


- On a grid this Laplacian also commutes with the shift operator, so equivalent

Actually:

1D grid = ring graph



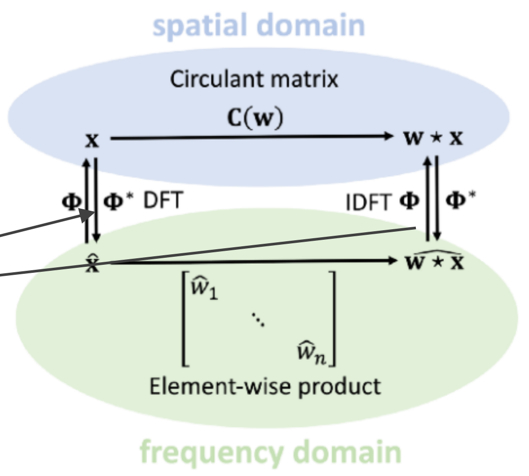


adjacency matrix A of ring graph = Shift operator S^{T}

Further details

- Eigenvectors of Graph Laplacian as analogy to Fourier
  - Equivalent on grids
- For undirected graphs ⇒ symmetric adjacency matrix and graph Laplacian
- For directed graphs ⇒ generalized eigenvectors/Jordan decomposition
  - More elaborate

Replace $\mathbf{\Phi}$ with eigenvectors of graph Laplacian
Otherwise, same thing



Eigendecomposition of Laplacian:

$$L = V \Lambda V^T$$

Applying this basis V is called *graph Fourier Transform:*

$$x' = V^T x$$

This brings the signal x to the *spectral* domain

In analogy to convolutions in frequency domain: We now define spectral graph convolutions

- Similar to regular convolutions
- Compute (Graph) Fourier Transform

$$\hat{\boldsymbol{x}} = \boldsymbol{\Phi}^* \mathbf{x}$$

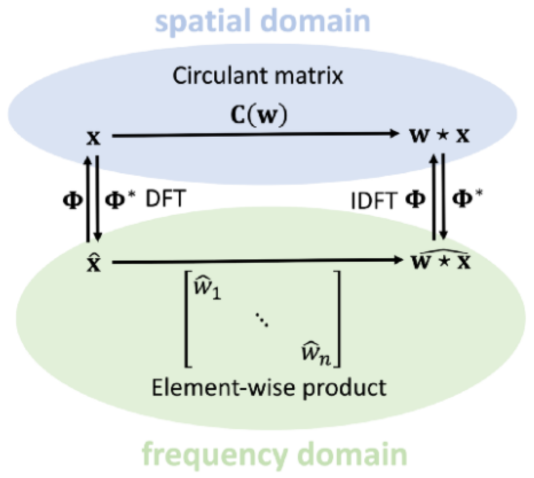
Where $\boldsymbol{\Phi}^*$ are the eigenvectors (conjugate transpose) of graph Laplacian

- Apply learnable filter in Fourier space

$$\hat{\boldsymbol{x}} \odot \hat{\boldsymbol{w}} = \begin{bmatrix} \hat{w}_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \hat{w}_n \end{bmatrix} \hat{\boldsymbol{x}}$$

- Compute Inverse (Graph) Fourier Transform

$$\boldsymbol{x} * \boldsymbol{y} = \boldsymbol{\Phi} \cdot (\hat{\boldsymbol{x}} \odot \hat{\boldsymbol{w}})$$



Where we are, part 2

- Want: translation equivariance
  - Introduce shift operator
  - Use the fact that circulant matrices commute
  - To arrive at the result that convolutions fullfill these requirements
- Actually convolutions can be computed in spectral domain
  - Here, they are materialized as just various diagonal matrices, that re-mix different frequencies given from the eigenspace basis
- We can move into the spectral easily on graphs by using the Laplacian
  - One we do this, defining a spectral convolution on graphs is trivial
  - More precisely: it's the same as for images, except for different bases/transforms

Why the graph Laplacian*?

Apply L to X: For node *v:*

$$
\begin{aligned}
(Lx)_v &= L_v x \\
&= \sum_{u \in G} L_{vu} x_u \\
&= \sum_{u \in G} (D_{vu} - A_{vu}) x_u \\
&= D_v \, x_v - \sum_{u \in \mathcal{N}(v)} x_u
\end{aligned}
$$

<u>It’s Local</u>:

- Aggregating over immediate <u>neighbour</u> features x_u.
- Combining with the node’s <u>own</u> feature x_v.

<u>It’s permutation equivariant</u>:

$$
\begin{aligned}
x &\rightarrow Px \\
L &\rightarrow PLP^T \\
L^i &\rightarrow PL^iP^T
\end{aligned}
$$

Say, with $f(x, L) = Lx$:

$$f(Px, PLP^T) = PLP^T Px = PLx = Pf(x, L)$$

* (it could also be adjacency or other variants of Laplacian)

https://distill.pub/2021/understanding-gnns/

Spectral graph convolution

$$\mathbf{x} \star \mathbf{y} = \mathbf{\Phi} \begin{bmatrix} \hat{y}_1 & & \\ & \ddots & \\ & & \hat{y}_n \end{bmatrix} \mathbf{\Phi}^{\mathrm{T}} \mathbf{x}$$

In order to compute convolution $\mathbf{x} \star \mathbf{y}$

- **Graph FT:** $\hat{\mathbf{x}} = \mathbf{\Phi}^{\mathrm{T}} \mathbf{x}$ $O(n^2)+O(n^3)$
- **Apply filter:** $\hat{\mathbf{x}} \circ \hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 & & \\ & \ddots & \\ & & \hat{y}_n \end{bmatrix} \hat{\mathbf{x}}$ $O(n)$
- **Inverse graph FT:** $\mathbf{x} \star \mathbf{y} = \mathbf{V}(\hat{\mathbf{x}} \circ \hat{\mathbf{y}})$ $O(n^2)$

Some drawbacks of this variant

- Computational complexity of at least $O(n^2)$ vs O(nlog(n)) for spatial convs
- Parameter complexity of $O(n)$ vs O(1)
- Isotropic filters (same filter no matter if small neighborhood or large one)
- Filters that depend on choice of basis and do not generalize across graphs

Easy to increase the field of view with powers of the Laplacian

"Convolutions" as linear combinations of powers of the Laplacian:

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \ldots + w_d L^d = \sum_{i=0}^{d} w_i L^i.$$

Here, w is the a vector of learned weights.

No need to have d=n, can simply stop at say d=3

No need to compute graph FT.

https://distill.pub/2021/understanding-gnns/

Putting it together: stacking graph convolutions

Start with the original features.

$$h^{(0)} = x$$

Color Codes:

🟧 Computed node embeddings.

🟦 Learnable parameters.

Then iterate, for $k = 1, 2, \ldots$ upto K:

$$p^{(k)} = p_{w^{(k)}}(L)$$

Compute the matrix $p^{(k)}$ as the polynomial defined by the filter weights $w^{(k)}$ evaluated at L.

$$g^{(k)} = p^{(k)} \times h^{(k-1)}$$

Multiply $p^{(k)}$ with $h^{(k-1)}$: a standard matrix-vector multiply operation.

$$h^{(k)} = \sigma\left(g^{(k)}\right)$$

Apply a non-linearity σ to $g^{(k)}$ to get $h^{(k)}$.

Quiz: What properties does this polynomial variant have?

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \ldots + w_d L^d = \sum_{i=0}^{d} w_i L^i.$$

$$p^{(k)} = p_{w^{(k)}}(L)$$

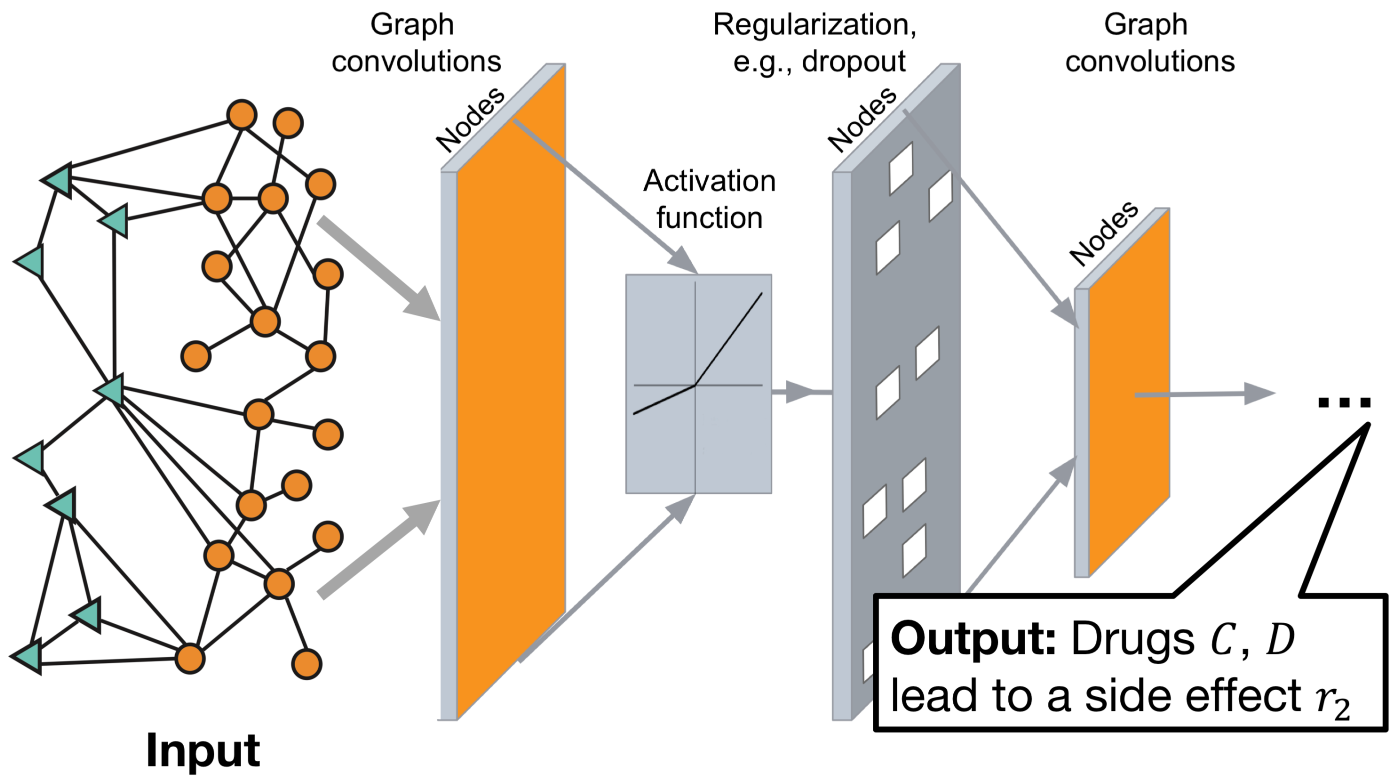
$$g^{(k)} = p^{(k)} \times h^{(k-1)}$$

1) We can simply precompute the powers of L
2) We do not need to *actually* go into the spectral space
3) We do not need to learn a parameter for *every* frequency, but instead only for however many terms of Laplacian powers
4) If d is not n, it does not incorporate global information at early stages
5) All of the above

Some drawbacks of this variant now

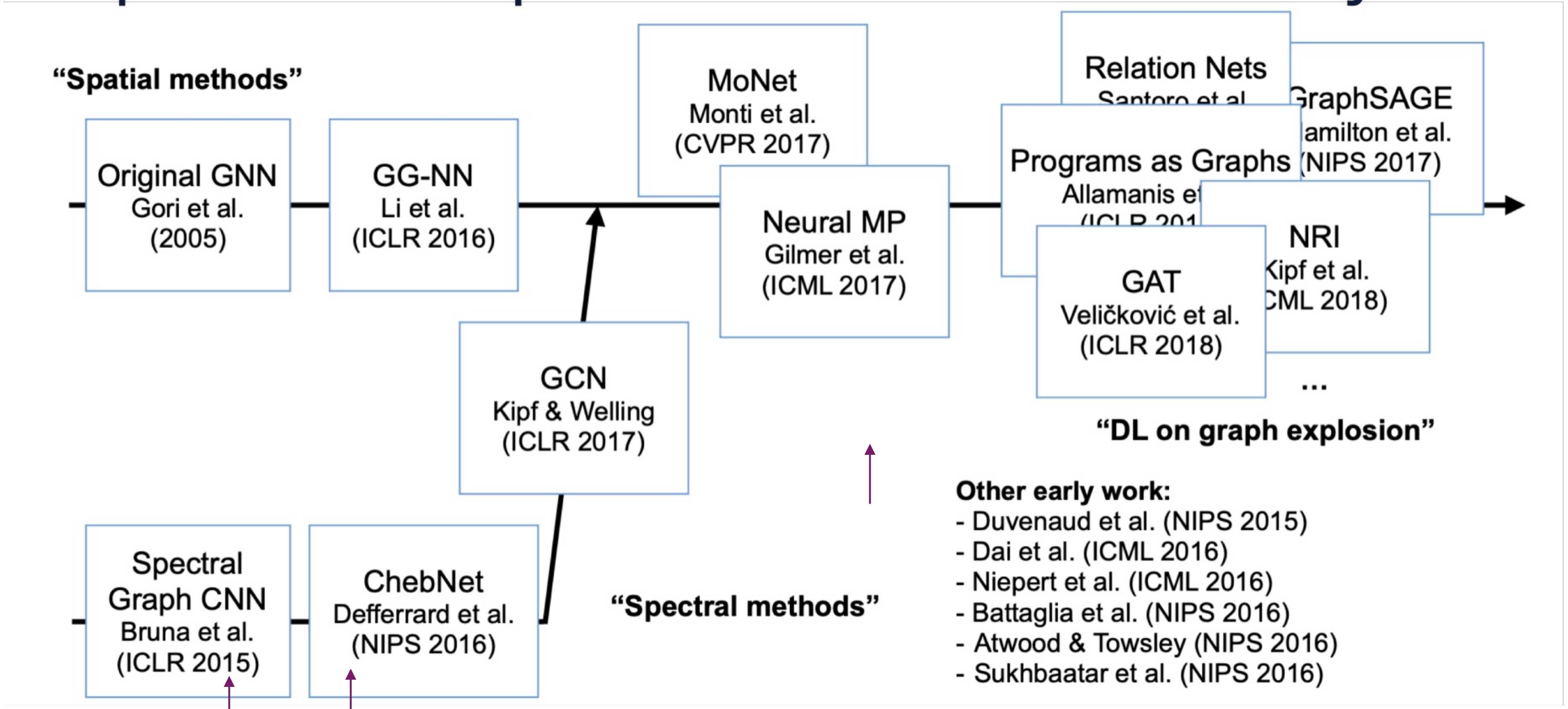
- Computational complexity of at least $O(n^2)$ vs O(nlog(n)) for spatial convs
  - Avoided for polynomial variant
- Parameter complexity of $O(n)$ for original spectral
  - Complexity of O(1) for polynomial variant vs O(1)
- Isotropic filters (same filter no matter if small neighborhood or large one)
- Filters that depend on choice of basis and do not generalize across graphs

Spatial graph convolutions



Output: Drugs C, D lead to a side effect r_2

Spatial vs Spectral - A brief history


"Spatial methods"
Original GNN Gori et al. (2005)
GG-NN Li et al. (ICLR 2016)
MoNet Monti et al. (CVPR 2017)
Neural MP Gilmer et al. (ICML 2017)
Relation Nets Santoro et al.
GraphSAGE Hamilton et al. (NIPS 2017)
Programs as Graphs Allamanis et al. (ICLR 201
NRI Kipf et al. (ICML 2018)
GAT Veličković et al. (ICLR 2018)
...
"DL on graph explosion"
GCN Kipf & Welling (ICLR 2017)
Spectral Graph CNN Bruna et al. (ICLR 2015)
ChebNet Defferrard et al. (NIPS 2016)
"Spectral methods"

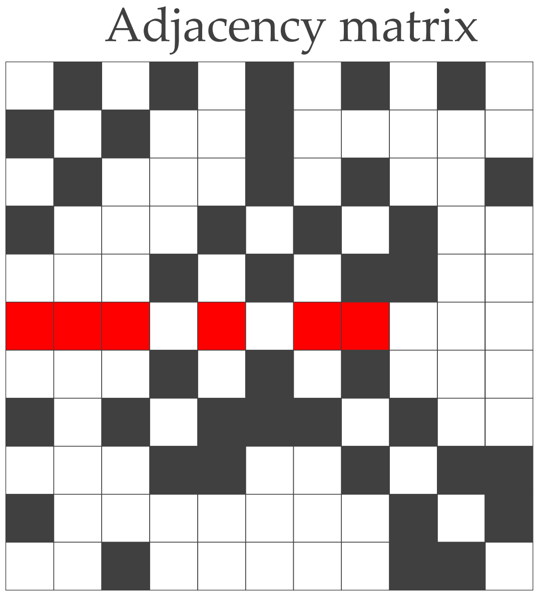
Other early work:
- Duvenaud et al. (NIPS 2015)
- Dai et al. (ICML 2016)
- Niepert et al. (ICML 2016)
- Battaglia et al. (NIPS 2016)
- Atwood & Towsley (NIPS 2016)
- Sukhbaatar et al. (NIPS 2016)

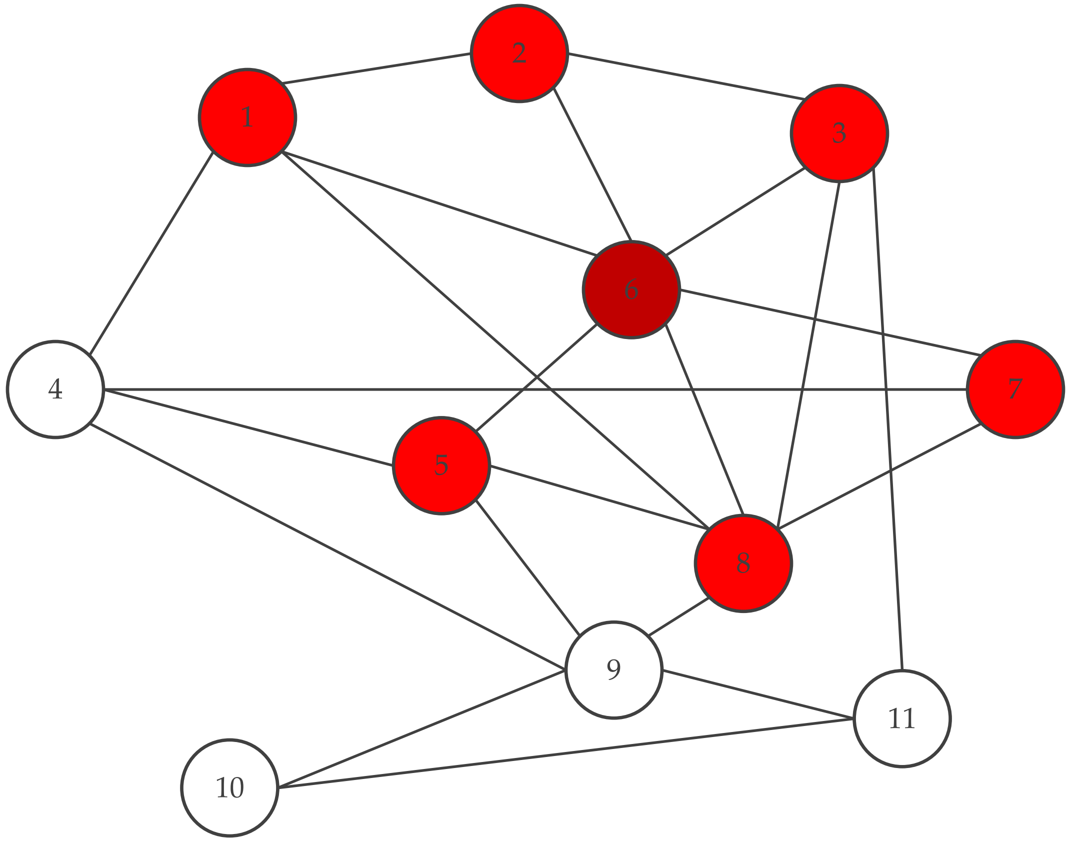
Image borrowed from Kipf

Essentially what we just did

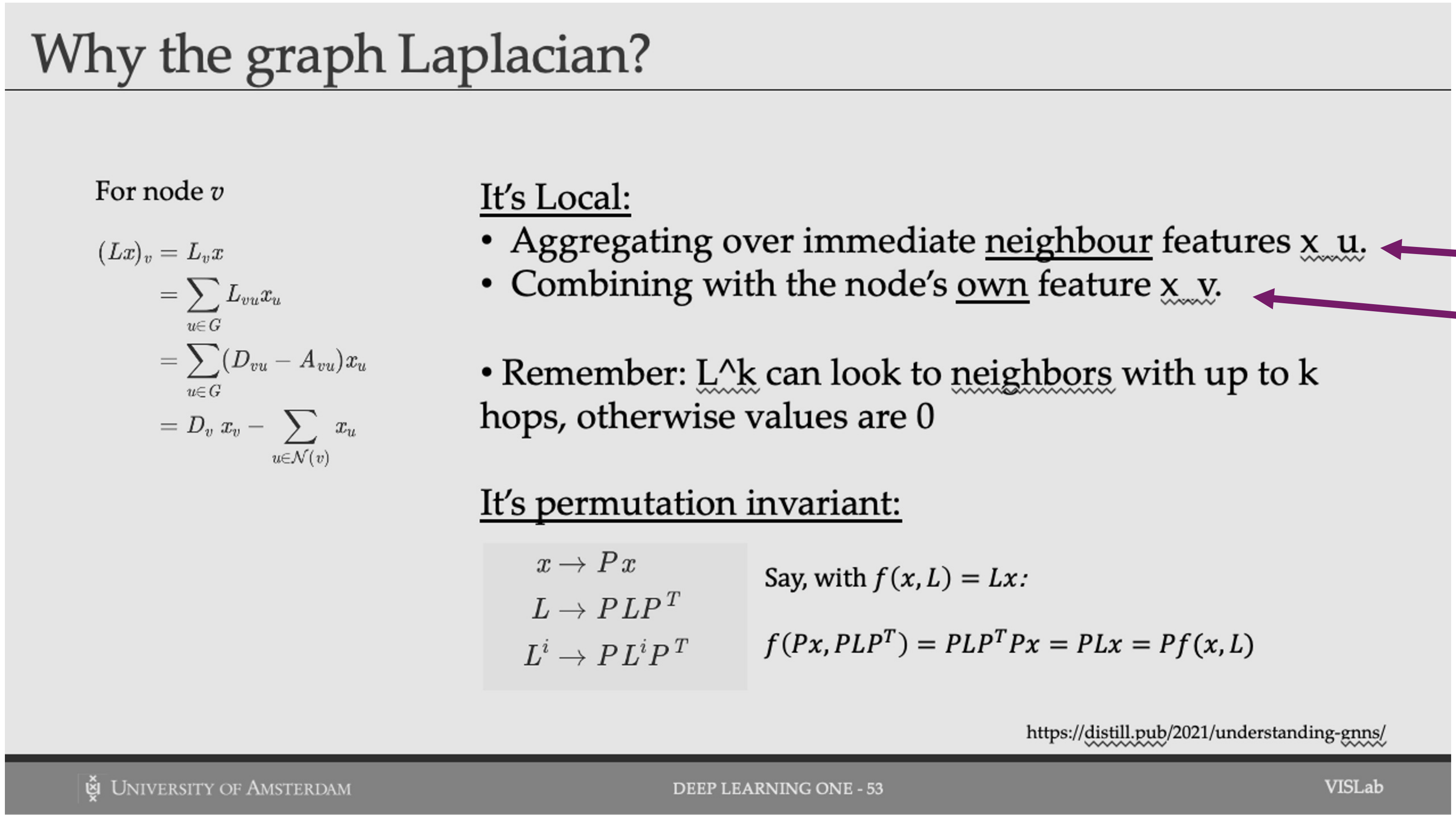
Graph convolutions

- Convolutions as ("local") matrix multiplications





What can we use from the spectral approach?



Why the graph Laplacian?

For node v

$$\begin{aligned}(Lx)_v &= L_v x \\ &= \sum_{u \in G} L_{vu} x_u \\ &= \sum_{u \in G} (D_{vu} - A_{vu}) x_u \\ &= D_v\, x_v - \sum_{u \in \mathcal{N}(v)} x_u\end{aligned}$$

It's Local:

- Aggregating over immediate neighbour features x_u.
- Combining with the node's own feature x_v.

• Remember: L^k can look to neighbors with up to k hops, otherwise values are 0

It's permutation invariant:

$$x \to Px$$
$$L \to PLP^T$$
$$L^i \to PL^iP^T$$

Say, with $f(x, L) = Lx$:

$$f(Px, PLP^T) = PLP^T Px = PLx = Pf(x, L)$$

https://distill.pub/2021/understanding-gnns/

UNIVERSITY OF AMSTERDAM DEEP LEARNING ONE - 53 VISLab

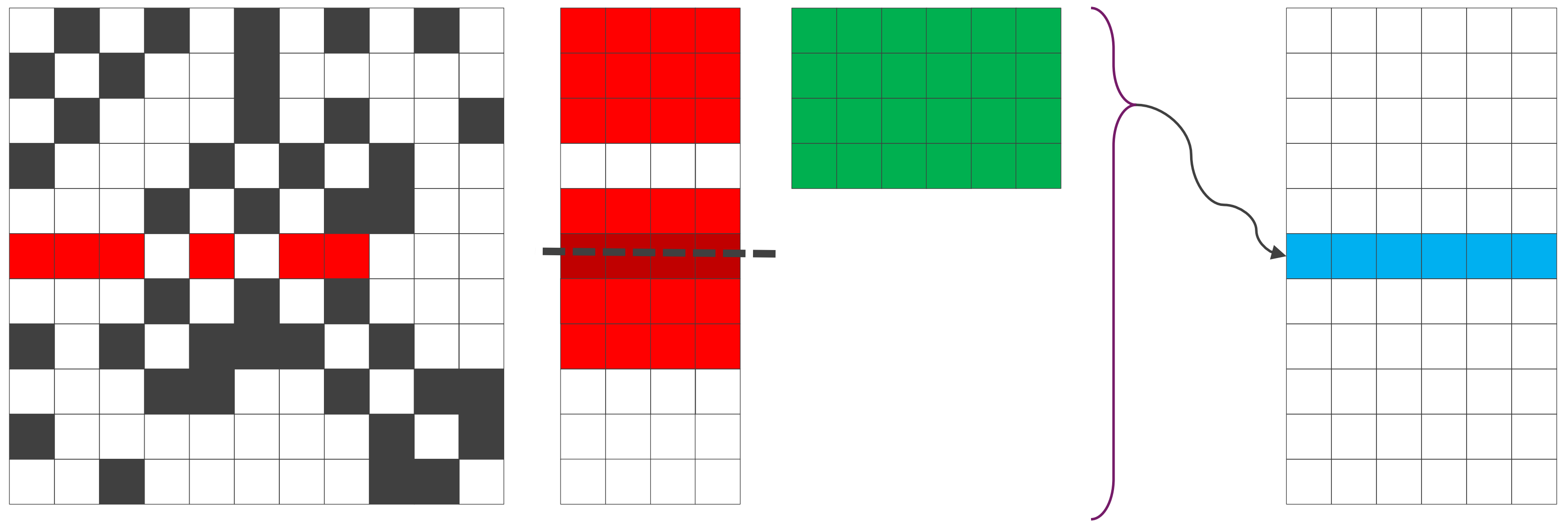
Combine
& aggregate

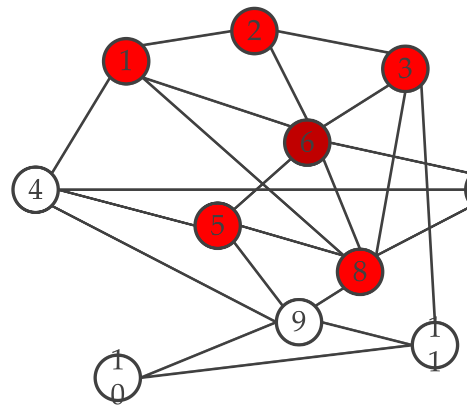
as a generic principle

Graph Convolutional Networks (GCN)

- Main idea: keep this as simple as possible, get larger neighborhood by stacking. Choose polynomial with just order of 1
- Each node has a feature vector (row-wise)
  - Left-multiplying normalized Laplacian, we combine features in neighborhood
  - Right-multiplying with a weight matrix, we "cor

$$y = \mathrm{ReLU}(L_{norm} X W) \qquad L_{norm} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$



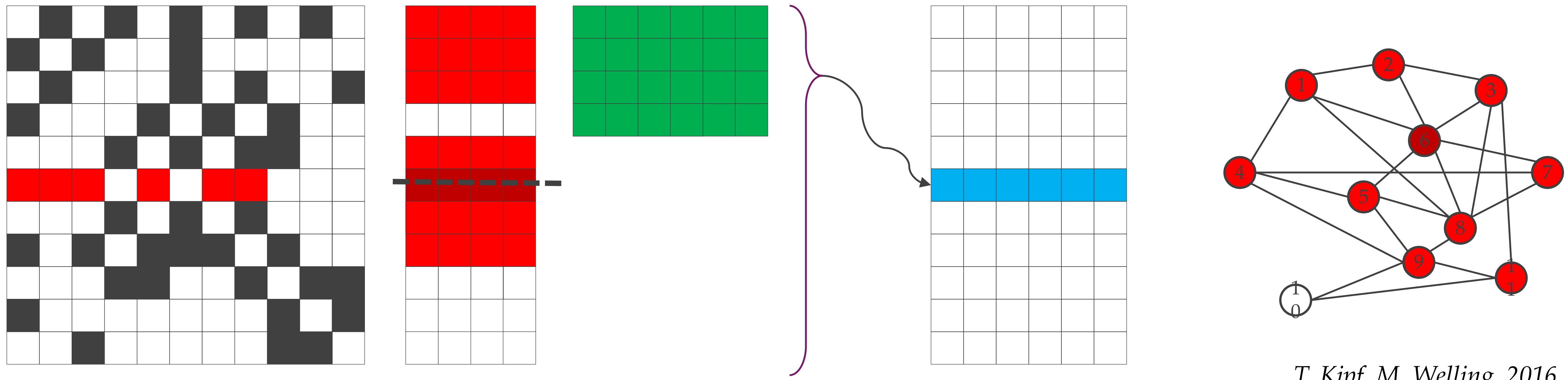


T. Kipf, M. Welling, 2016

Graph Convolutional Networks (GCN)

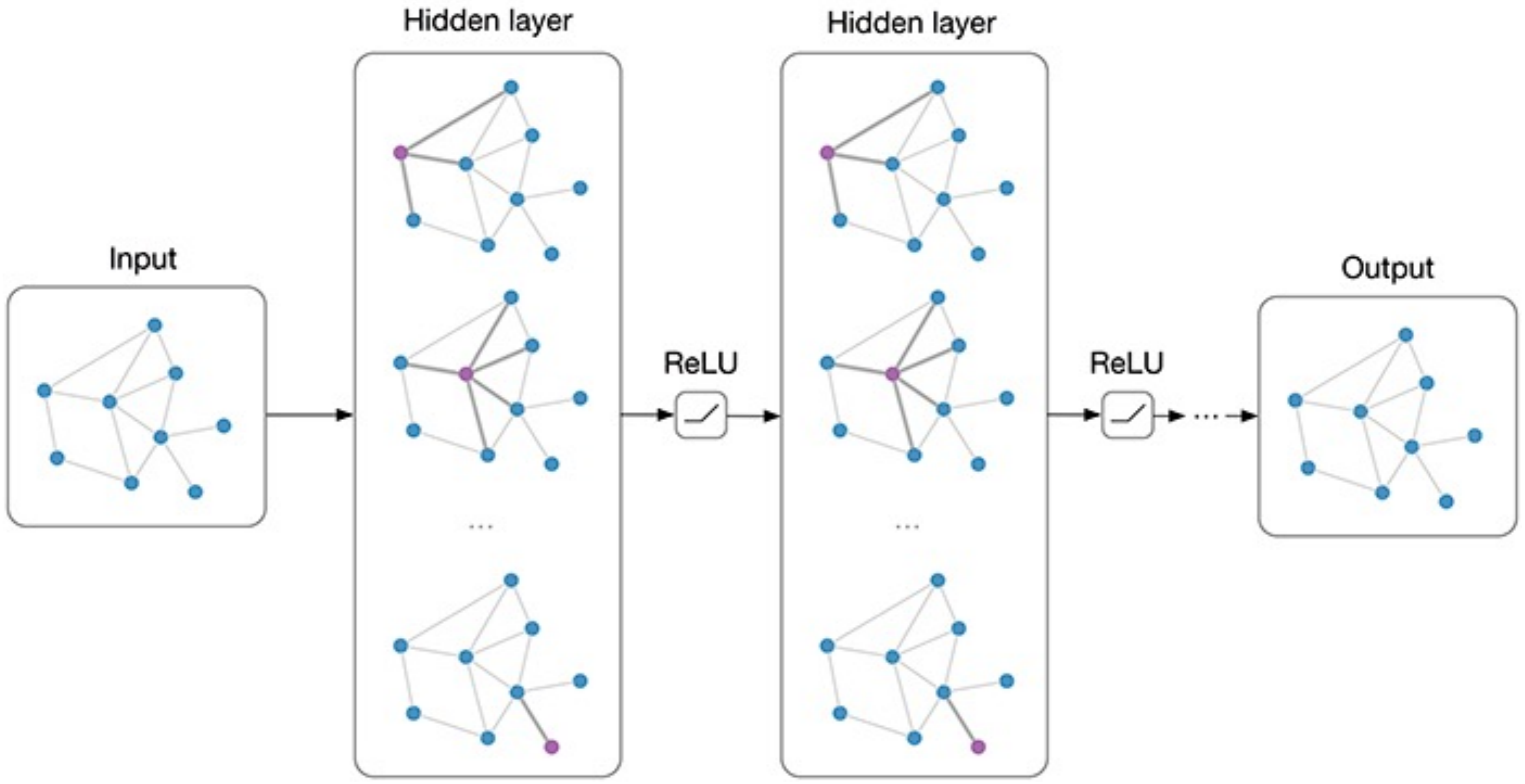
- Each node has a feature vector (row-wise)
  - Left-multiplying normalized Laplacian, we combine features in neighborhood
  - Right-multiplying with a weight matrix, we "convolve"/aggregate
- We can also stack multiple convolution layers: k of those will give k neighborhood

$$y = \text{softmax}(L\ \text{ReLU}(LXW_1)W_2)$$



T. Kipf, M. Welling, 2016

Putting it together:



Has weights $\boldsymbol{W}_1$

Has weights $\boldsymbol{W}_2$

Other kind of aggregation: Graph Attention Networks (GAT)

- Similar but including attention as *aggregation*: $y_j = h\left(\sum_{j \in \mathcal{N}(i)} a_{ij} \boldsymbol{x_j}\right)$

- Using self-attention:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ij})},$$

- where e_{ij} are the self-attention weights (like query == key)

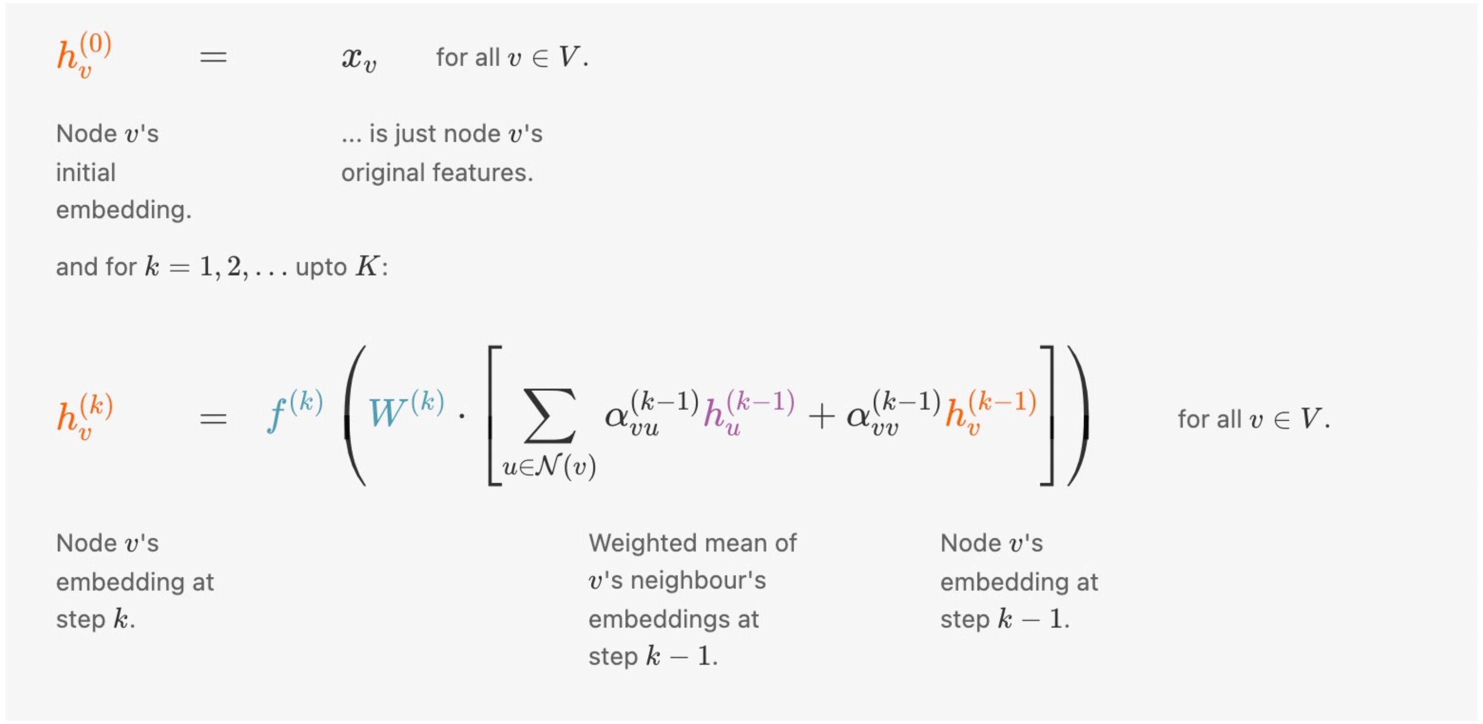
$$e_{ij} = \text{LeakyRELU}\left(\left[\boldsymbol{x}_i \boldsymbol{W}, \boldsymbol{x}_j \boldsymbol{W}\right] \cdot u\right)$$

- u is a weight vector

F. Monti et al., 2017; P. Velickovic et al. 2018

Self-attention for graph convolutions

Written out, GAT looks similar to our the approach with Laplacian but only uses 1-hop neighbors at every step:



$$h_v^{(0)} = x_v \quad \text{for all } v \in V.$$

Node v's initial embedding.

... is just node v's original features.

and for $k = 1, 2, \ldots$ upto K:

$$h_v^{(k)} = f^{(k)}\left(W^{(k)} \cdot \left[\sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k-1)} h_u^{(k-1)} + \alpha_{vv}^{(k-1)} h_v^{(k-1)}\right]\right) \quad \text{for all } v \in V.$$

Node v's embedding at step k.

Weighted mean of v's neighbour's embeddings at step $k-1$.

Node v's embedding at step $k-1$.

Visualisation of computation of attention weight and attention-based computation:

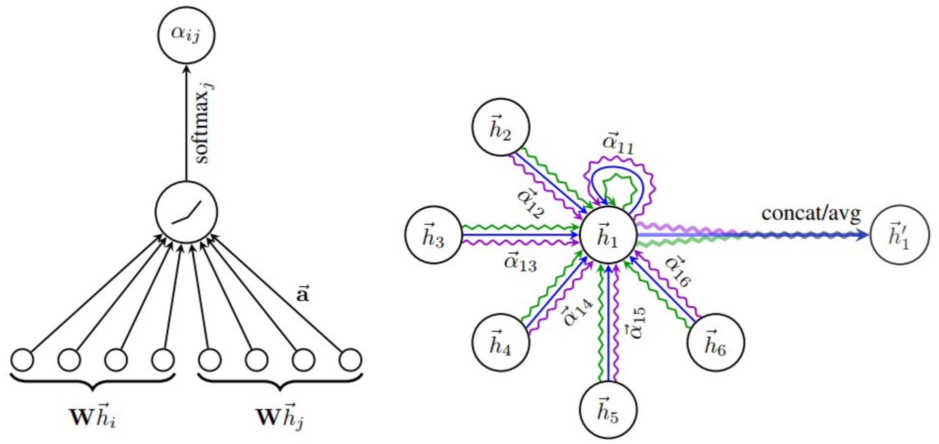
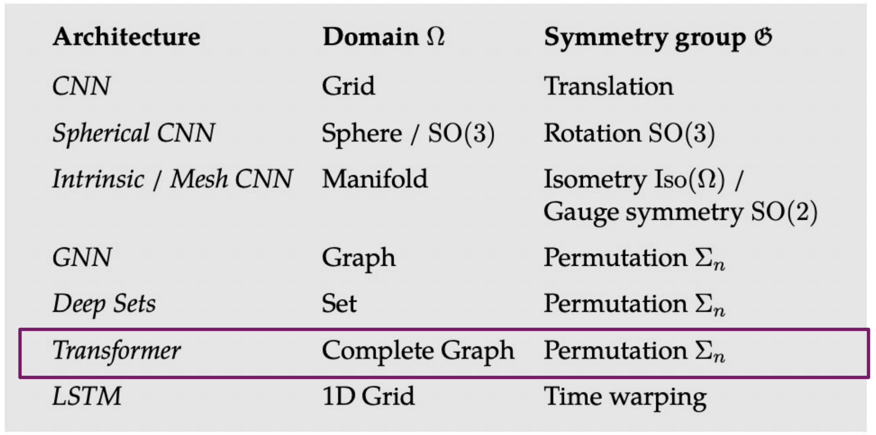


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

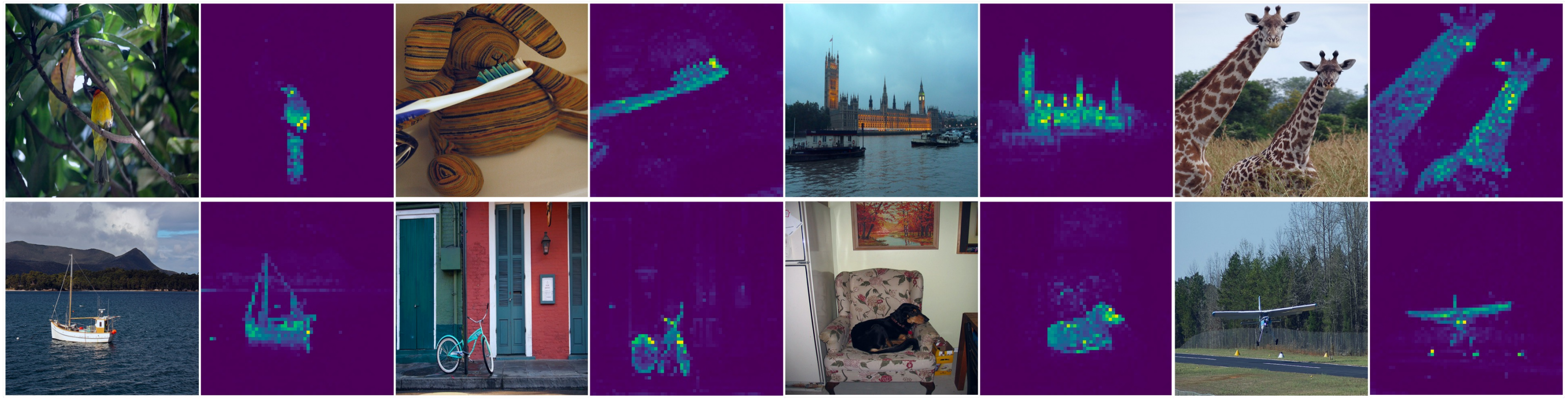
P. Velickovic et al. 2018

Connection to transformers

- When operating on a complete graph, ie with A=1 (a matrix with only ones)
- With the attentional GCN we recover the Transformer
- Can think of computed attention weights as a "soft" adjacency matrix



Architecture	**Domain Ω**	**Symmetry group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / SO(3)	Rotation SO(3)
Intrinsic / Mesh CNN	Manifold	Isometry Iso(Ω) / Gauge symmetry SO(2)
GNN	Graph	Permutation Σ_n
Deep Sets	Set	Permutation Σ_n
Transformer	Complete Graph	Permutation Σ_n
LSTM	1D Grid	Time warping



e.g. attention weights of the [CLS] token to the spatial patches at the last linear layer of a ViT

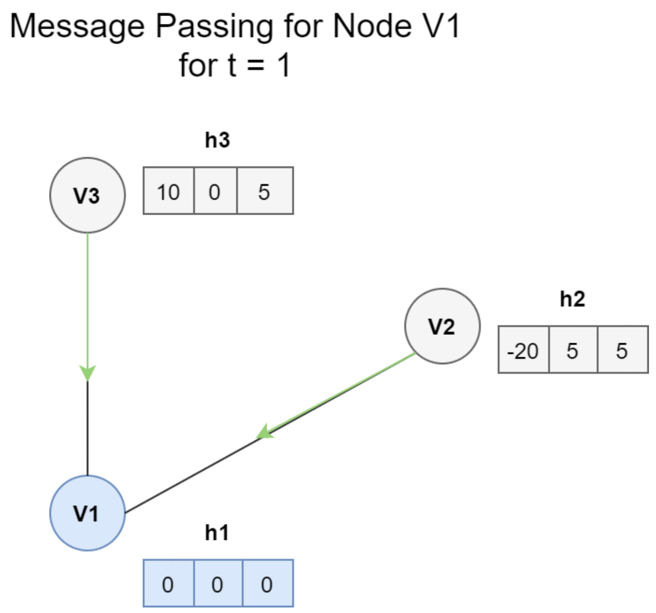
Message Passing Neural Network (MPNN)

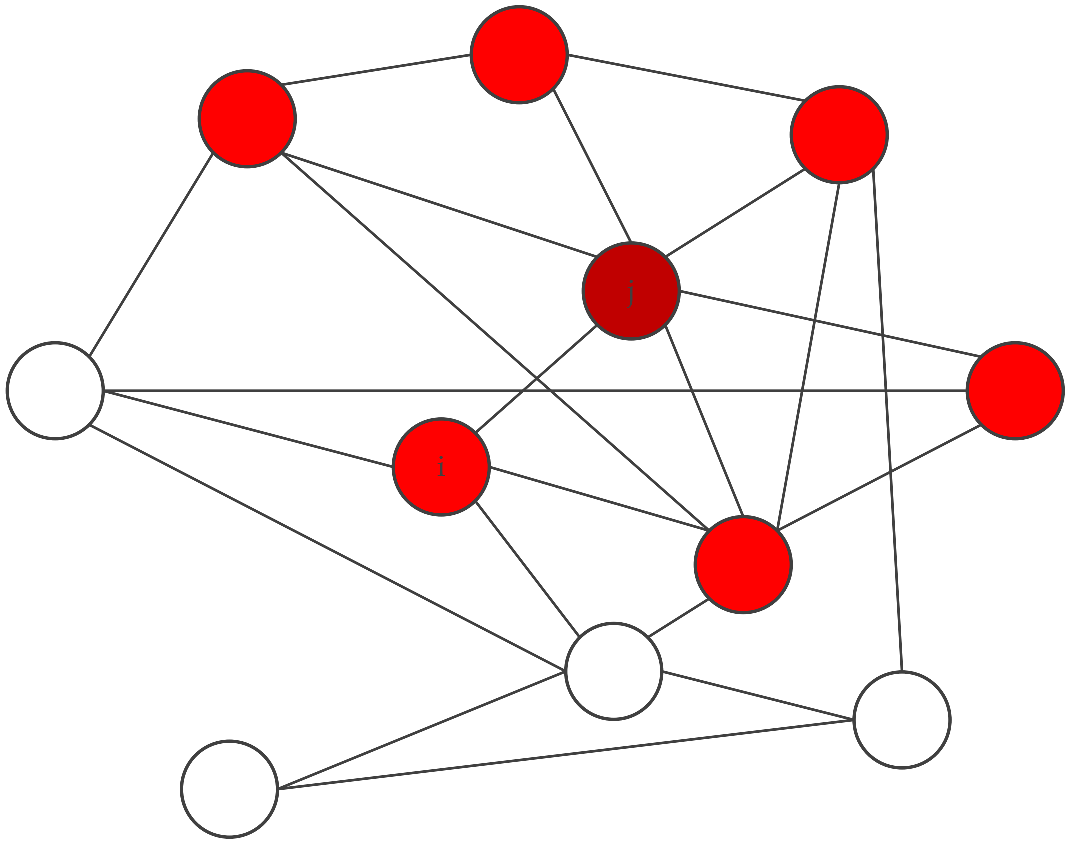
- General aggregation function
  - $y_j = g(\sum_{j \in \mathcal{N}(i)} h(x_i, x_j, e_{ij}, W))$

$$m_v^{t+1} = \sum_{w \in N(v)} h_w^t$$

$$h_v^{t+1} = average(h_v, m_v^{t+1})$$

ht - hidden state for each node

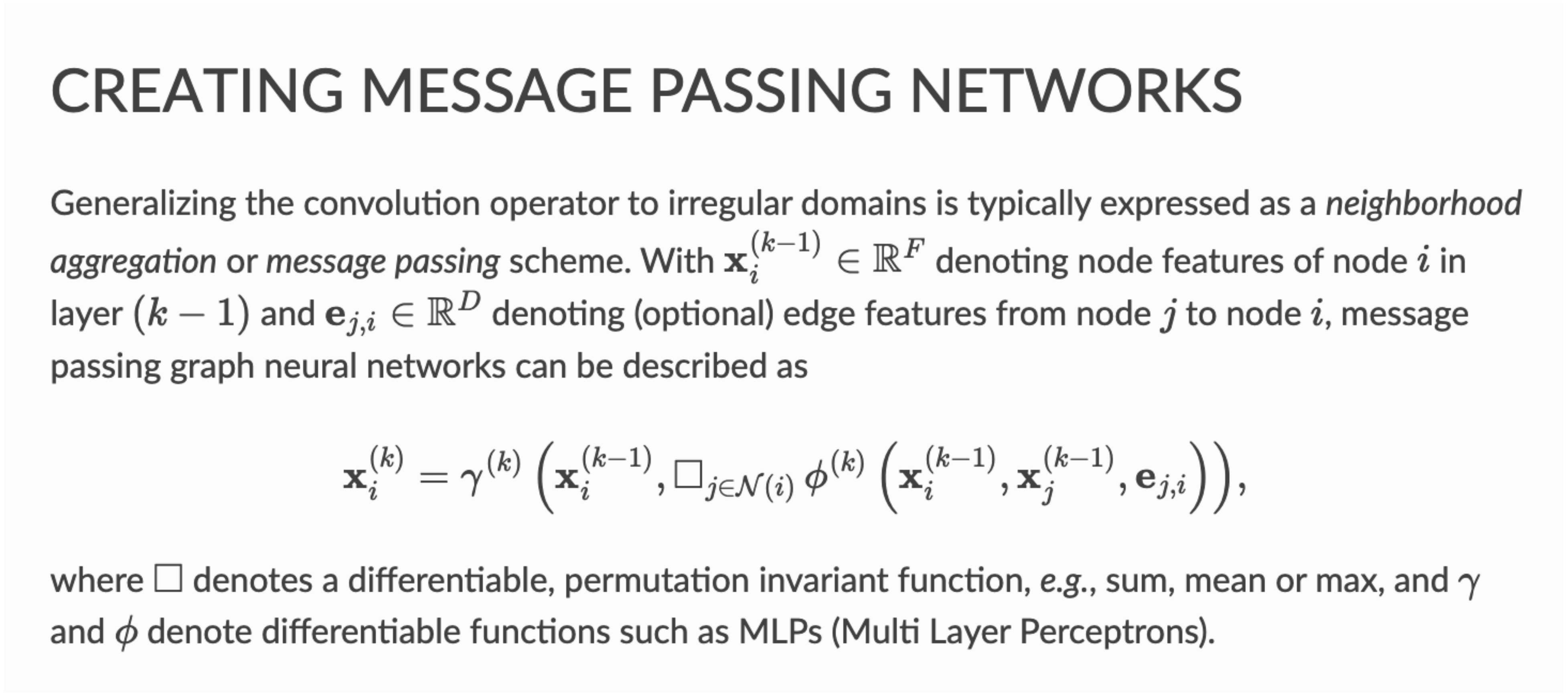




Gilmer et al. 2017

PyTorch Geometric baseclass

- convolution ⊆ attention ⊆ message-passing.



CREATING MESSAGE PASSING NETWORKS

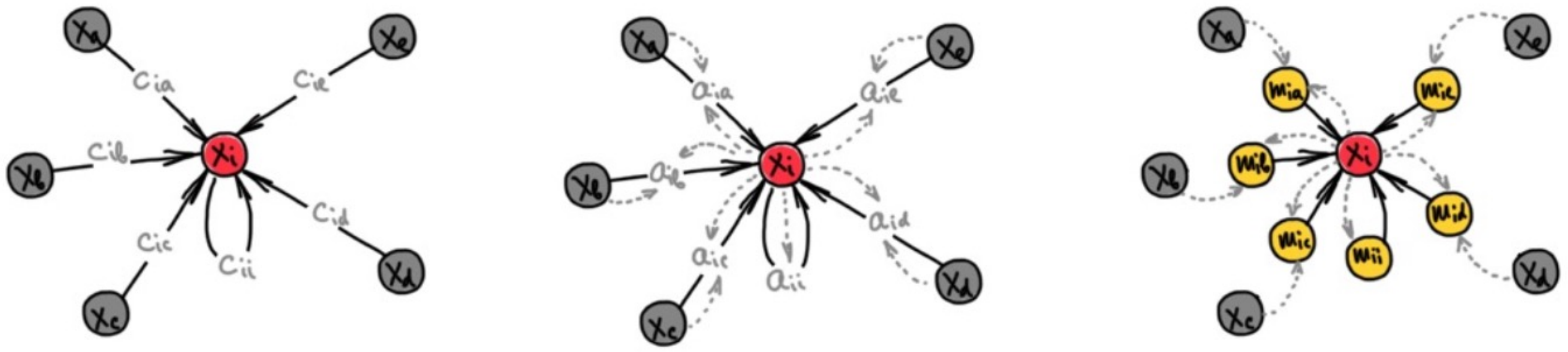
Generalizing the convolution operator to irregular domains is typically expressed as a *neighborhood aggregation* or *message passing* scheme. With $\mathbf{x}_i^{(k-1)} \in \mathbb{R}^F$ denoting node features of node i in layer $(k-1)$ and $\mathbf{e}_{j,i} \in \mathbb{R}^D$ denoting (optional) edge features from node j to node i, message passing graph neural networks can be described as

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \, \phi^{(k)} \left(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{j,i} \right) \right),$$

where $\square$ denotes a differentiable, permutation invariant function, *e.g.*, sum, mean or max, and γ and ϕ denote differentiable functions such as MLPs (Multi Layer Perceptrons).

https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html

Overview



Three "flavours" of GNNs, left-to-right: convolutional, attentional, and general nonlinear message passing flavours. All are forms of message passing. Figure adapted from P. Veličković.

Finally, a note about coarsening graphs

- Group nodes together, saves compute
- Learnable pooling
- Adjacency matrices get updated
- Add, average or max pool the node features

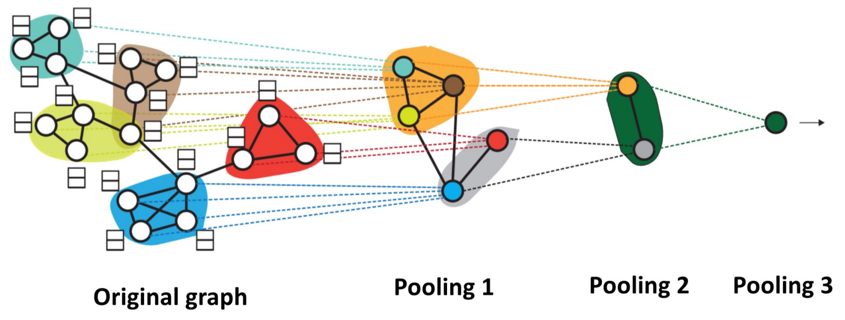
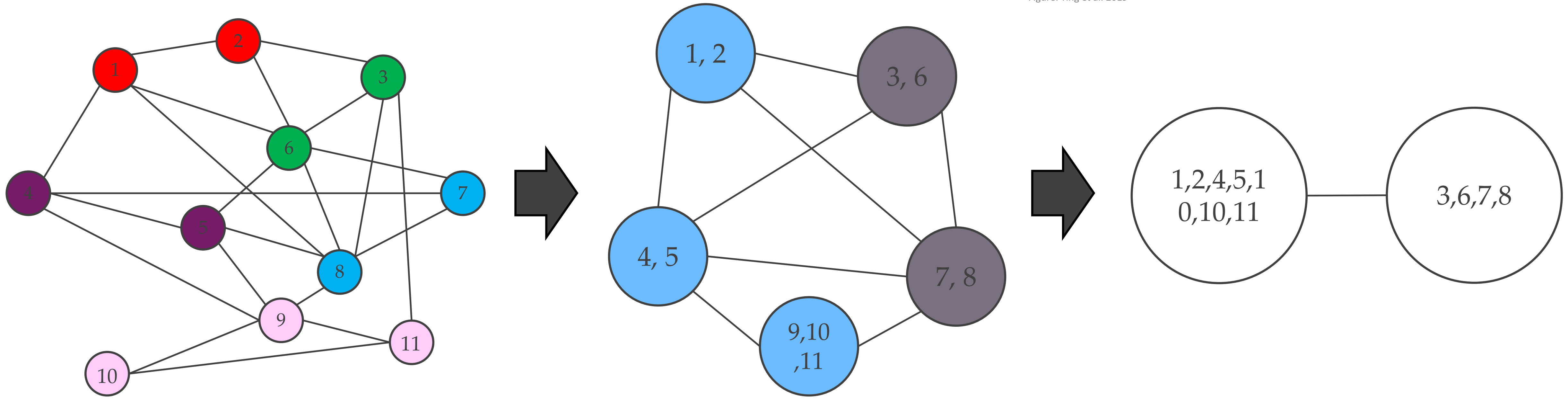


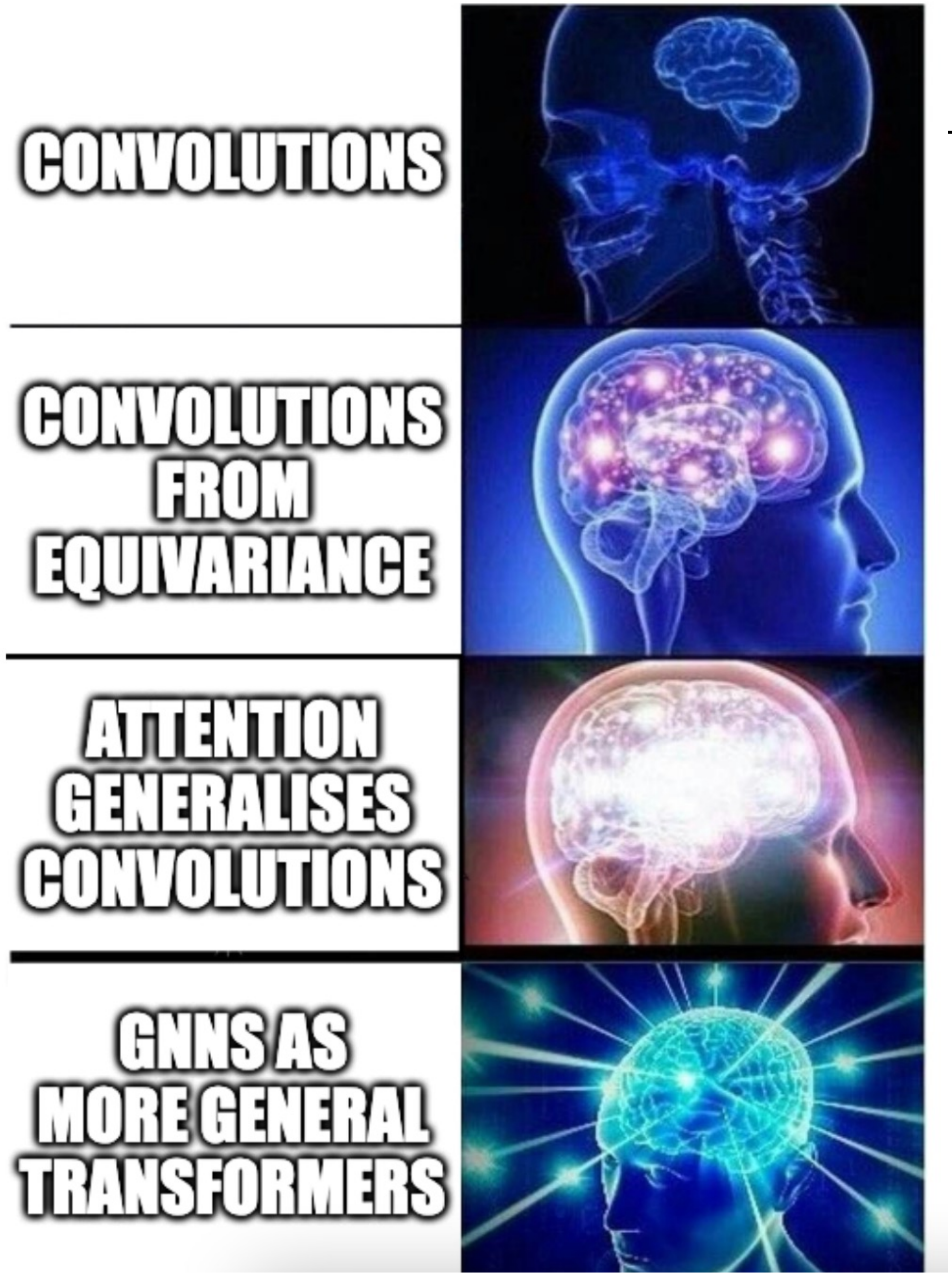
Figure: Ying et al. 2019



Where we are, part 3

- Want: translation equivariance
  - Introduce shift operator
  - Use the fact that circulant matrices commute
  - To arrive at the result that convolutions fullfill these requirements
- Actually convolutions can be computed in spectral domain
  - Here, they are just various diagonal matrices, that re-mix different frequencies given from the eigenspace basis
- We can move into the spectral easily on graphs by using the Laplacian
  - One we do this, defining a spectral convolution on graphs is trivial
  - More precisely: it's the same as for images, except for different bases/transforms
- Instead of spectral convs that use the full-spectrum, we can also just use the polynomial variant with several advantages
- We can also simply do local (spatial) operations at every step, by staying in the neighborhood
  - This yields spatial graph CNNs which come in various flavors (GCN, GAT, MPN)
  - The GAT on a complete graph recovers the transformer architecture, attention can be seen as soft edges
  - MPN encompasses previous approaches to be more generic

The last few lectures


CONVOLUTIONS
CONVOLUTIONS FROM EQUIVARIANCE
ATTENTION GENERALISES CONVOLUTIONS
GNNS AS MORE GENERAL TRANSFORMERS

Summary

- What makes graphs special?
- Revisiting graphs
- Revisiting convolutions
- Spectral graph convolutions and
- Spatial graph convolutions

Extra reading material:

- Graphs, Convolutions, and Neural Networks. Gama et al.
- Understanding Convolutions on Graphs. Daigavane et al.

Break

- We're more than halfway done!
- Have some snacks!