Graphs in Machine Learning

Daniele Calandriello

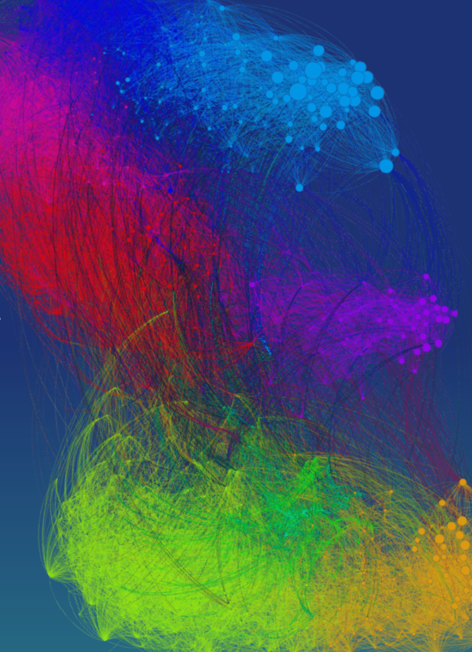
DeepMind Paris, France

Collaborators: Achraf Azize,
Michal Valko

Based on material by: Petar Veličković, Marc Lelarge

https://petar-v.com/communications.html

https://dataflowr.github.io



Message passing GNNs

recapping

A recipe for graph neural networks

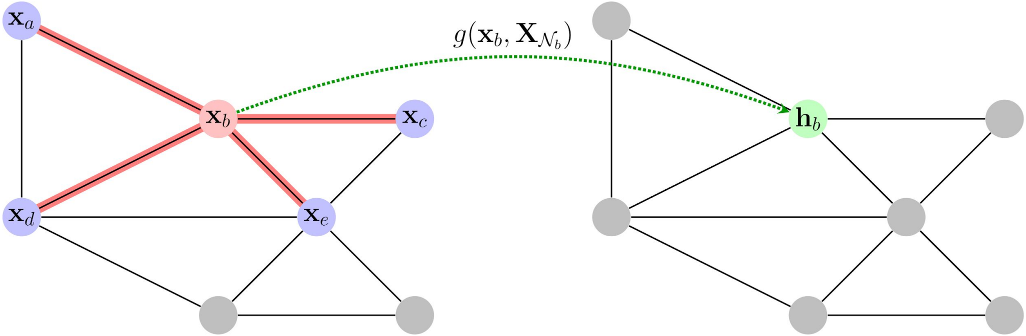
$X \in \mathbb{R}^{n \times d}$ $\quad n = |V|$

We can construct permutation equivariant functions $f(\mathbf{X}, \mathbf{A})$ by appropriately applying an invariant g over all local neighbourhoods:

features with dim d $\quad$ adjacency matrix

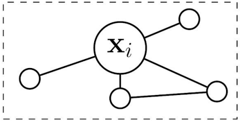
$$f(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} — & g(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) & — \\ — & g(\mathbf{x}_2, \mathbf{X}_{\mathcal{N}_2}) & — \\ & \vdots & \\ — & g(\mathbf{x}_n, \mathbf{X}_{\mathcal{N}_n}) & — \end{bmatrix}$$

A recipe for graph neural networks, visualised



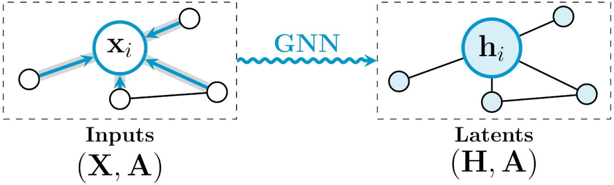
$$\mathbf{X}_{\mathcal{N}_b} = \{\!\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\!\}$$

General blueprint for learning on graphs

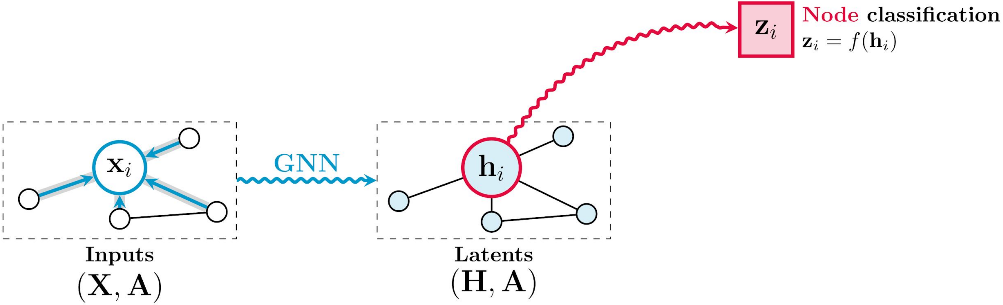


Inputs
$(\mathbf{X}, \mathbf{A})$

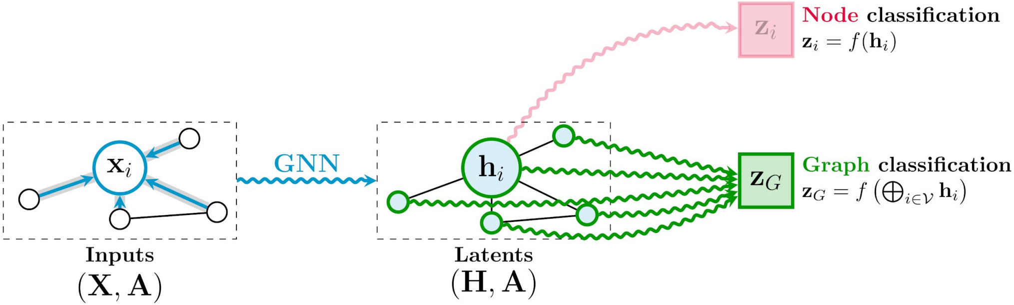
General blueprint for learning on graphs



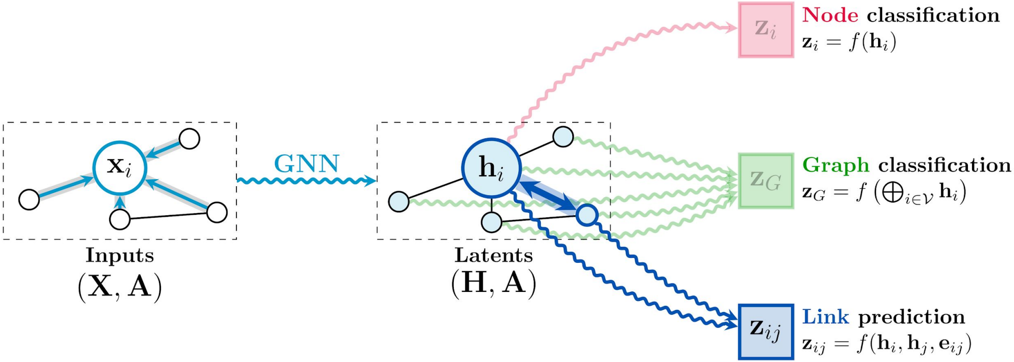
General blueprint for learning on graphs



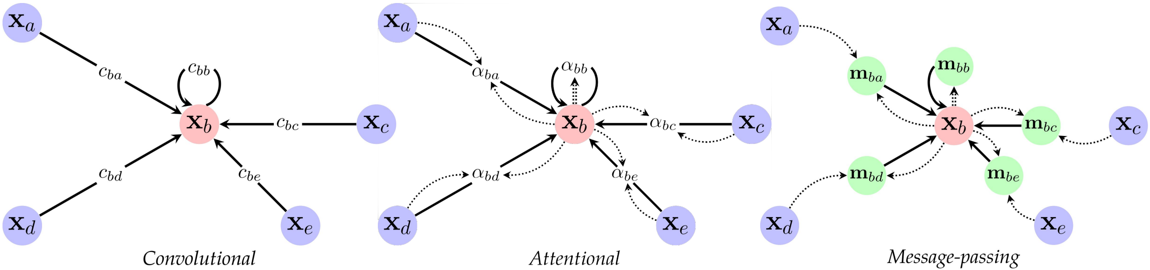
General blueprint for learning on graphs



General blueprint for learning on graphs



The three "flavours" of GNN layers



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

Spectral GNNs

the Laplacian strikes back

Convolutions on a graph

GCNs start from convolution, but then replaces it with permutation $\{$in,equ$\}$ivariance

$\hookrightarrow$ how far can we take vanilla convolutions in graph ML?

Convolutions on a graph

GCNs start from convolution, but then replaces it with permutation $\{$in,equ$\}$ivariance
↳ how far can we take vanilla convolutions in graph ML?

The convolution theorem defines a very attractive identity:

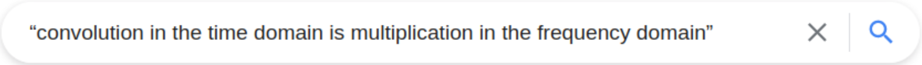
$$(\mathbf{x} \star \mathbf{y})(\xi) = \widehat{\mathbf{x}}(\xi) \cdot \widehat{\mathbf{y}}(\xi) \quad \text{with } \widehat{\mathbf{x}}(\xi) = \int_{-\infty}^{\infty} x(u) e^{-i\xi u} du$$

Convolutions on a graph

GCNs start from convolution, but then replaces it with permutation {in,equ}ivariance
↳ how far can we take vanilla convolutions in graph ML?

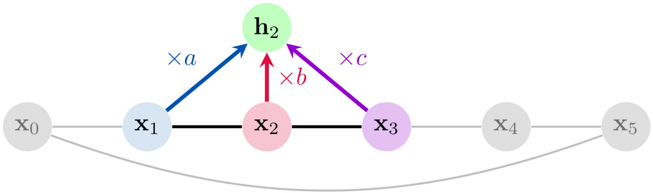
The convolution theorem defines a very attractive identity:

$$(\mathbf{x} \star \mathbf{y})(\xi) = \widehat{\mathbf{x}}(\xi) \cdot \widehat{\mathbf{y}}(\xi) \quad \text{with } \widehat{\mathbf{x}}(\xi) = \int_{-\infty}^{\infty} x(u) e^{-i\xi u} du$$



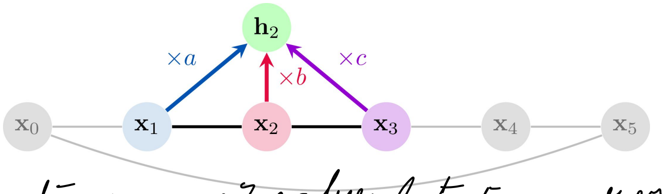
Convolutions on a graph

For special graphs (e.g., direct cycle) we can directly define a convolution over it:



Convolutions on a graph

For special graphs (e.g., direct cycle) we can directly define a convolution over it:



circulant matrix : same 3 values but at every row, move right

This convolution can be represented using a **circulant** matrix $C([b, c, 0, \dots, 0, a])$

$$f(\mathbf{X}) = \begin{bmatrix} b & c & & & a \\ a & b & c & & \\ & \ddots & \ddots & \ddots & \\ & & a & b & c \\ c & & & a & b \end{bmatrix} \begin{bmatrix} — & \mathbf{x}_0 & — \\ — & \mathbf{x}_1 & — \\ & \vdots & \\ — & \mathbf{x}_{n-2} & — \\ — & \mathbf{x}_{n-1} & — \end{bmatrix}$$

Properties of circulants, and their eigenvectors

Circulant matrices commute: $C(\mathbf{a})C(\mathbf{b})\mathbf{X} = C(\mathbf{b})C(\mathbf{a})\mathbf{X}$, for any parameters $\mathbf{a}$, $\mathbf{b}$
$\hookrightarrow$ matrices that commute are jointly **diagonalisable**

Properties of circulants, and their eigenvectors

Circulant matrices commute: $C(\mathbf{a})C(\mathbf{b})\mathbf{X} = C(\mathbf{b})C(\mathbf{a})\mathbf{X}$, for any parameters $\mathbf{a}$, $\mathbf{b}$

↳ matrices that commute are jointly **diagonalisable**

↳ the eigenvectors of circulants are the discrete Fourier basis!

$$\phi = \frac{1}{\sqrt{n}}\left(1, e^{\frac{2\pi il}{n}}, e^{\frac{2\pi i\cdot 2\cdot l}{n}}, \ldots, e^{\frac{2\pi i\cdot(n-1)\cdot l}{n}}\right)$$

Properties of circulants, and their eigenvectors

Circulant matrices commute: $C(\mathbf{a})C(\mathbf{b})\mathbf{X} = C(\mathbf{b})C(\mathbf{a})\mathbf{X}$, for any parameters $\mathbf{a}$, $\mathbf{b}$

↳ matrices that commute are jointly **diagonalisable**

↳ the eigenvectors of circulants are the discrete Fourier basis!

$$\phi = \frac{1}{\sqrt{n}} \left(1, e^{\frac{2\pi il}{n}}, e^{\frac{2\pi i \cdot 2 \cdot l}{n}}, \ldots, e^{\frac{2\pi i \cdot (n-1) \cdot l}{n}}\right)$$

If we stack these Fourier basis vectors ϕ into a matrix $\mathbf{\Phi}$ we recover the discrete Fourier transform (DFT) as multiplication by $\mathbf{\Phi}^*$ (adjoint).

Properties of circulants, and their eigenvectors

We can now eigendecompose any circulant as $C(\boldsymbol{\theta}) = \boldsymbol{\Phi}\boldsymbol{\Theta}\boldsymbol{\Phi}^*$
$\hookrightarrow$ here $\boldsymbol{\Theta}$ is a diagonal matrix of $C(\boldsymbol{\theta})$'s eigenvalues $\widehat{\boldsymbol{\theta}}$.

Properties of circulants, and their eigenvectors

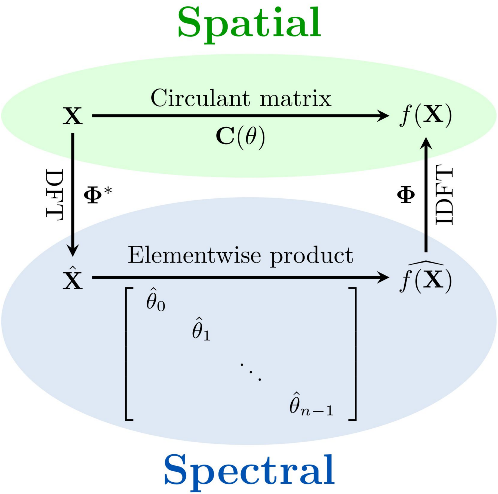
We can now eigendecompose any circulant as $C(\boldsymbol{\theta}) = \boldsymbol{\Phi}\boldsymbol{\Theta}\boldsymbol{\Phi}^*$
↳ here $\boldsymbol{\Theta}$ is a diagonal matrix of $C(\boldsymbol{\theta})$'s eigenvalues $\widehat{\boldsymbol{\theta}}$.

The convolution theorem naturally follows:

$$f(\mathbf{x}) = C(\boldsymbol{\theta})\mathbf{X} = \boldsymbol{\Phi}\boldsymbol{\Theta}\boldsymbol{\Phi}^* = \boldsymbol{\Phi}\begin{bmatrix}\widehat{\boldsymbol{\theta}}_0 & & \\ & \ddots & \\ & & \widehat{\boldsymbol{\theta}}_n\end{bmatrix}\boldsymbol{\Phi}^{*\mathbf{X}} = \boldsymbol{\Phi}(\widehat{\boldsymbol{\theta}} \cdot \widehat{\mathbf{X}})$$

and as long as we know $\boldsymbol{\Phi}$ we can express convolutions as multiplications in $\widehat{\boldsymbol{\theta}}$.

The spectral CNN blueprint



From spectral CNN to spectral GNN

For which convolutions we know $\mathbf{\Phi}$?
↳ cycle $\rightarrow$ DFT

From spectral CNN to spectral GNN

For which convolutions we know $\mathbf{\Phi}$?

↳ cycle $\rightarrow$ DFT

grid $\rightarrow$ n-way DFT

general graph $\rightarrow$????

From spectral CNN to spectral GNN

For which convolutions we know $\boldsymbol{\Phi}$?

↳ cycle $\rightarrow$ DFT

grid $\rightarrow$ n-way DFT

general graph $\rightarrow$ eigenvectors of Laplacian! ↳ captures the structure of the graph

From spectral CNN to spectral GNN

For which convolutions we know $\mathbf{\Phi}$?

↳ cycle $\rightarrow$ DFT

grid $\rightarrow$ n-way DFT

general graph $\rightarrow$ eigenvectors of Laplacian!

This allows us to re-express $\mathbf{L} = \mathbf{\Phi}\mathbf{\Theta}\mathbf{\Phi}^{\mathsf{T}}$, as before

↳ changing the eigenvalues in $\mathbf{\Theta}$ expresses any operation that commutes with $\mathbf{L}$

commonly referred to as the graph Fourier transform (Bruna et al., ICLR'14)

From spectral CNN to spectral GNN

For which convolutions we know $\boldsymbol{\Phi}$?
↳ cycle $\rightarrow$ DFT
grid $\rightarrow$ n-way DFT
general graph $\rightarrow$ eigenvectors of Laplacian!

This allows us to re-express $\mathbf{L} = \boldsymbol{\Phi}\boldsymbol{\Theta}\boldsymbol{\Phi}^\mathsf{T}$, as before
↳ changing the eigenvalues in $\boldsymbol{\Theta}$ expresses any operation that commutes with $\mathbf{L}$
commonly referred to as the graph Fourier transform (Bruna et al., ICLR'14)

To convolve with some feature matrix $\mathbf{X}$ we do as usual (the diagonal can be **learned**):

$$f(\mathbf{x}) = \boldsymbol{\Phi} \begin{bmatrix} \widehat{\boldsymbol{\theta}}_0 & & \\ & \ddots & \\ & & \widehat{\boldsymbol{\theta}}_n \end{bmatrix} \boldsymbol{\Phi}^* \mathbf{X} = \boldsymbol{\Phi}(\widehat{\boldsymbol{\theta}} \cdot \widehat{\mathbf{X}})$$

Spectral GNNs in practice

Directly learning the eigenvalues is typically inappropriate
$\hookrightarrow$ Not localised, doesn't transfer to other graphs, computationally expensive, ...

Spectral GNNs in practice

Directly learning the eigenvalues is typically inappropriate
↳ Not localised, doesn't transfer to other graphs, computationally expensive, ...

Instead, make the eigenvalues a polynomial function of the eigenvalues of $\mathbf{L}$
↳ i.e., $f(\mathbf{X}) = \mathbf{\Phi} p_k(\mathbf{\Theta})\mathbf{\Phi}^\intercal = p_k(\mathbf{L})\mathbf{X}$
↳ Cubic splines (Bruna et al., ICLR'14)
Chebyshev polynomials (Defferrard et al., NeurIPS'16)
Cayley polynomials (Levie et al., Trans. Sig. Proc.'18)

Spectral GNNs in practice

Directly learning the eigenvalues is typically inappropriate
↳ Not localised, doesn't transfer to other graphs, computationally expensive, ...

Instead, make the eigenvalues a polynomial function of the eigenvalues of $\mathbf{L}$
↳ i.e., $f(\mathbf{X}) = \mathbf{\Phi} p_k(\mathbf{\Theta})\mathbf{\Phi}^\intercal = p_k(\mathbf{L})\mathbf{X}$
↳ Cubic splines (Bruna et al., ICLR'14)
Chebyshev polynomials (Defferrard et al., NeurIPS'16)
Cayley polynomials (Levie et al., Trans. Sig. Proc.'18)

This is equivalent to some **conv-GNN!**
↳ Most efficient spectral approaches "spatialise" themselves in similar ways
The "spatial-spectral" divide is often *not really a divide* but a **design tool!**

The Transformer positional encodings and beyond

Transformers signal that the input is a sequence of words by using positional embeddings

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\mathsf{model}}}), \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\mathsf{model}}}),$$

The Transformer positional encodings and beyond

Transformers signal that the input is a sequence of words by using positional embeddings

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}), \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}),$$

Very similar to the DFT eigenvectors!
$\hookrightarrow$ interpretation as basis/eigenvectors of the grid graph assumed by transformers

The Transformer positional encodings and beyond

Transformers signal that the input is a sequence of words by using positional embeddings

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}), \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}),$$

Very similar to the DFT eigenvectors!
↳ interpretation as basis/eigenvectors of the grid graph assumed by transformers

Can use this idea to run Transformers with positional embeddings for general graphs!

The Transformer positional encodings and beyond

Transformers signal that the input is a sequence of words by using positional embeddings

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}), \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}),$$

Very similar to the DFT eigenvectors!
↳ interpretation as basis/eigenvectors of the grid graph assumed by transformers

Can use this idea to run Transformers with positional embeddings for general graphs!
↳ just feed some eigenvectors of the graph Laplacian (columns of $\mathbf{\Phi}$)

The Transformer positional encodings and beyond

Transformers signal that the input is a sequence of words by using positional embeddings

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}}), \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}}),$$

Very similar to the DFT eigenvectors!
↳ interpretation as basis/eigenvectors of the grid graph assumed by transformers

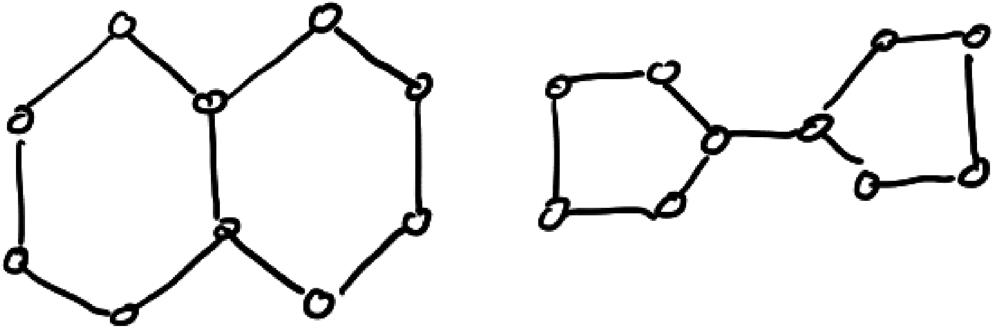
Can use this idea to run Transformers with positional embeddings for general graphs!
↳ just feed some eigenvectors of the graph Laplacian (columns of $\boldsymbol{\Phi}$)
↳ Another flavor of Graph Transformers! (Dwivedi & Bresson, 2021)

GNNs’ expressiveness

and new architectural directions

https://dataflowr.github.io/website/modules/graph3/

A problematic pair of graphs



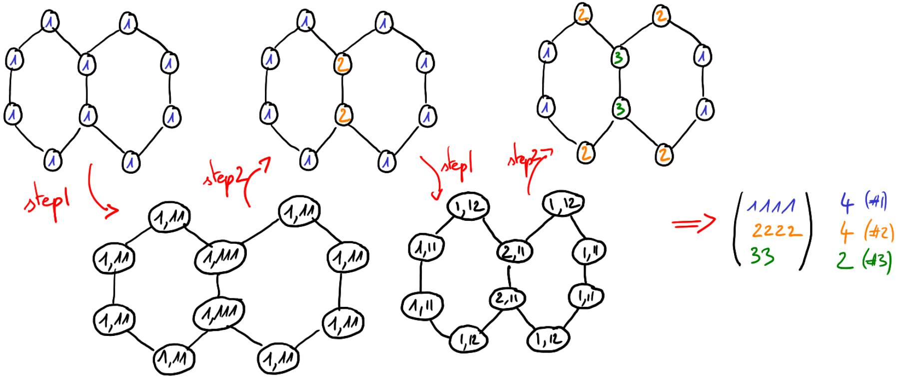
Separating power

Let $\mathcal{F}$ be a set of functions $f : \mathcal{X} \to \mathbb{R}$, then $\mathcal{F}$'s equivalence relation $\rho(\mathcal{F})$ on $\mathcal{X}$ is:

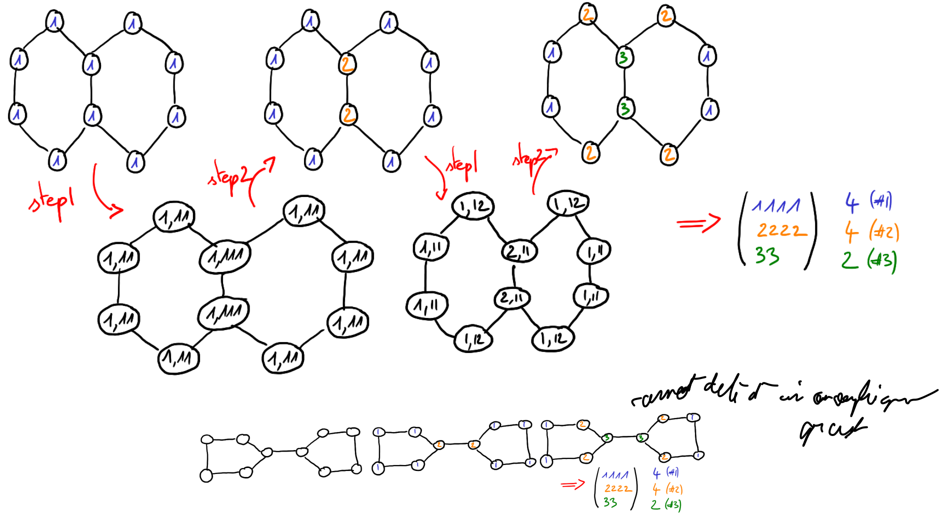
$$(\mathbf{x}, \mathbf{x}') \in \rho(\mathcal{F}) \iff \forall f \in \mathcal{F}, f(\mathbf{x}) = f(\mathbf{x}').$$

Given two sets of functions $\mathcal{F}$ and $\mathcal{H}$, $\mathcal{F}$ is more separating than $\mathcal{H}$ if $\rho(\mathcal{F}) \subset \rho(\mathcal{H})$.

2-Weisfeiler-Lehman test

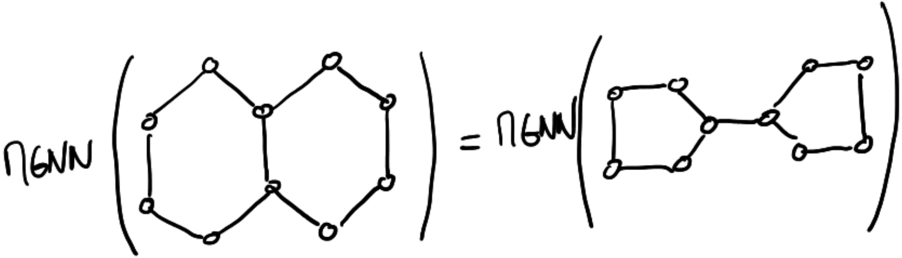


2-Weisfeiler-Lehman test

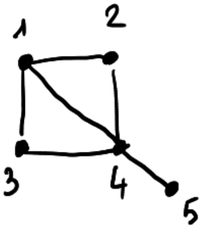


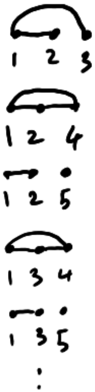
How Powerful Are Graph Neural Networks?, Xu et al., ICLR 2019

Message Passing GNNs are as powerful as 2-Weisfeiler-Lehman test



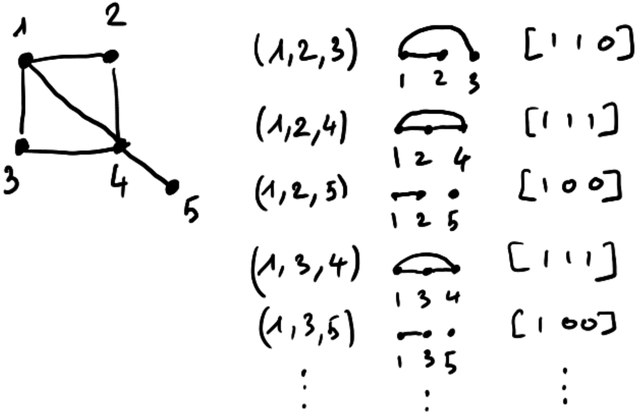
On the Universality of Invariant Networks, Maron et al., ICML 2019





(1,2,3)	[1 1 0]
(1,2,4)	[1 1 1]
(1,2,5)	[1 0 0]
(1,3,4)	[1 1 1]
(1,3,5)	[1 0 0]
⋮	⋮

On the Universality of Invariant Networks, Maron et al., ICML 2019



Minimal required order is $k \geq n^2$ to be able to approximate any invariant function.

Provably Powerful Graph Networks, Maron et al., NeurIPS 2019

Folklore GNN (FGNN)

$$\mathbf{h}'_{i\to j} = g\left(\mathbf{h}_{i\to j}, \sum_{k\in\mathcal{V}} \psi(\mathbf{h}_{i\to k}) \odot \psi(\mathbf{h}_{k\to j})\right)$$

Provably Powerful Graph Networks, Maron et al., NeurIPS 2019

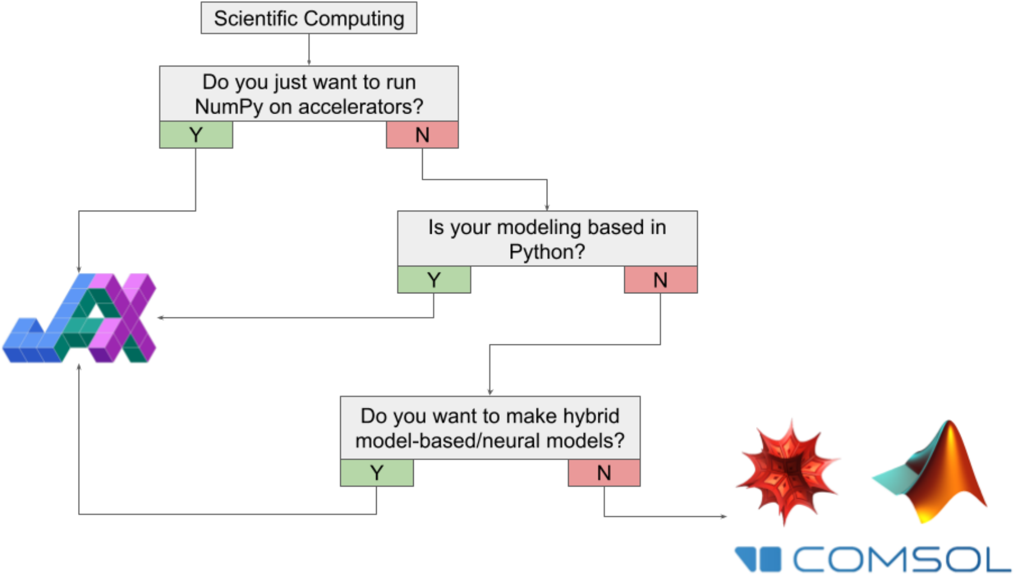
Folklore GNN (FGNN)

$$\mathbf{h}'_{i \to j} = g\left(\mathbf{h}_{i \to j}, \sum_{k \in \mathcal{V}} \psi(\mathbf{h}_{i \to k}) \odot \psi(\mathbf{h}_{k \to j})\right)$$

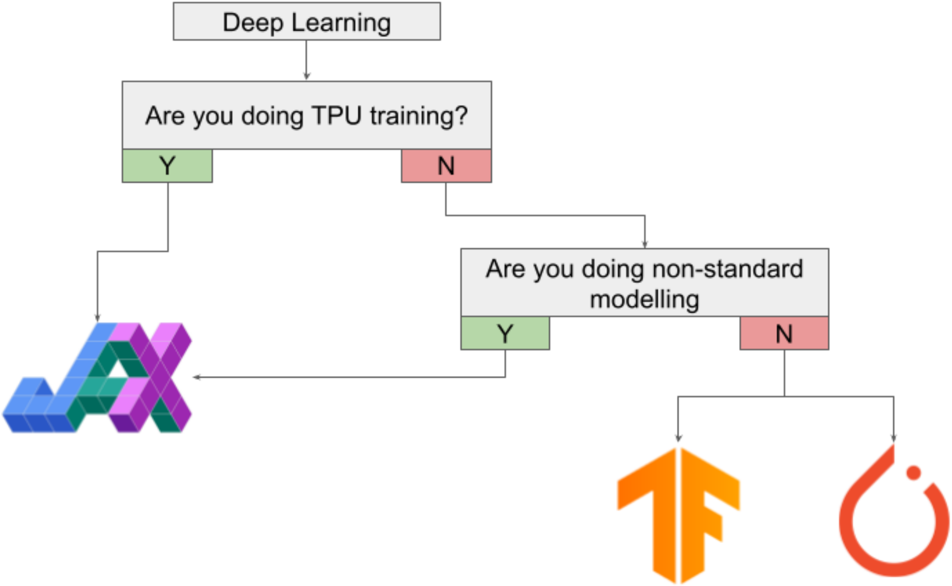
$$\rho(\mathsf{FGNN}) \not\subset \rho(2 - \mathsf{WL}) = \rho(\mathsf{MGNN})$$

JAX intro

just autograd and XLA



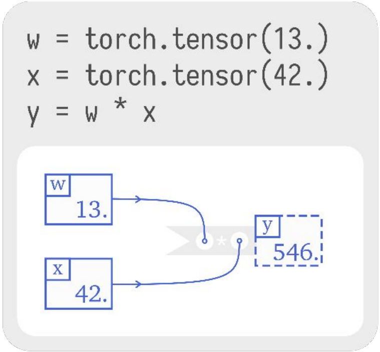
https://www.assemblyai.com/blog/why-you-should-or-shouldnt-be-using-jax-in-2023/



https://www.assemblyai.com/blog/why-you-should-or-shouldnt-be-using-jax-in-2023/

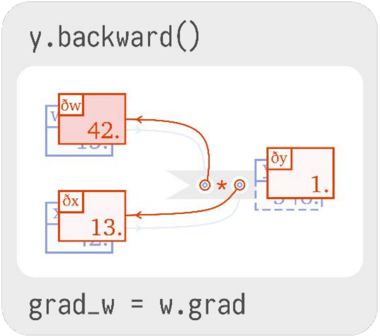
Tape-based autograd, e.g. PyTorch:

Executing code produces graph/tape



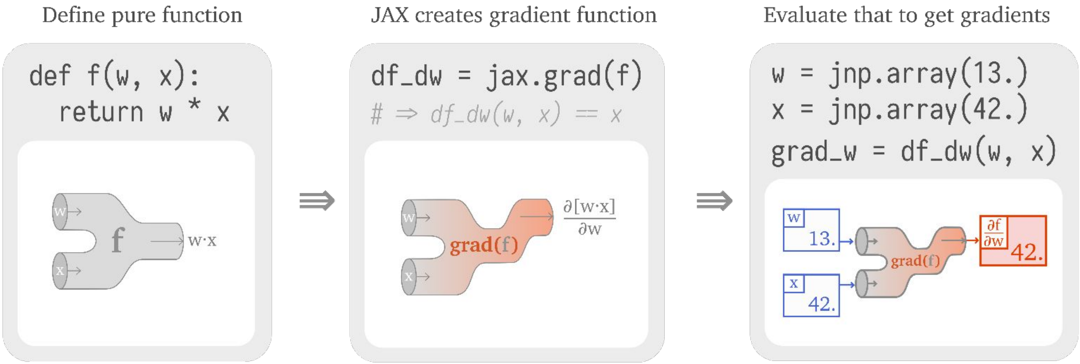
$\Rrightarrow$

Backprop/reverse-mode autodiff
by following the graph/tape



https://sjmielke.com/jax-purify.htm

Pure transformation-based autograd: JAX



https://sjmielke.com/jax-purify.htm

Two introductory colabs

https://github.com/deepmind/dm-haiku/blob/main/docs/notebooks/basics.ipynb

https://github.com/deepmind/dm-haiku/blob/main/docs/notebooks/parameter_sharing.ipynb

Daniele Calandriello
dcalandriello@google.com
ENS Paris-Saclay, MVA 2022/2023
https://sites.google.com/view/daniele-calandriello/