Deep Learning

Lecture 9: Graph neural networks

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Today

- Graphs
- Graph neural networks
- Special cases
- Applications

Motivation

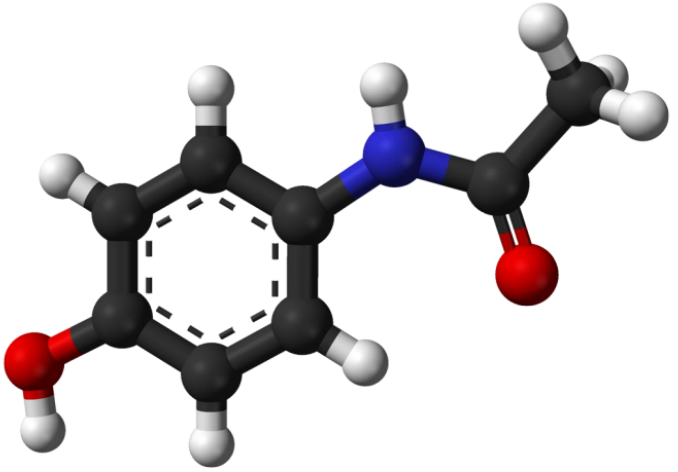
Many real-world problems do not fit into the tabular format of machine learning. Instead, many problems involve data that is naturally represented as *graphs*:

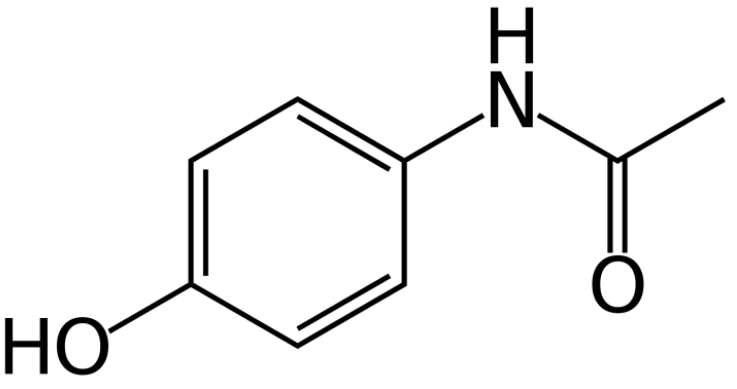
- social networks
- traffic networks
- biomolecular graphs
- scene graphs
- ... and many more!

The case of molecules

Molecules are naturally represented as graphs, where nodes represent atoms and edges represent bonds.

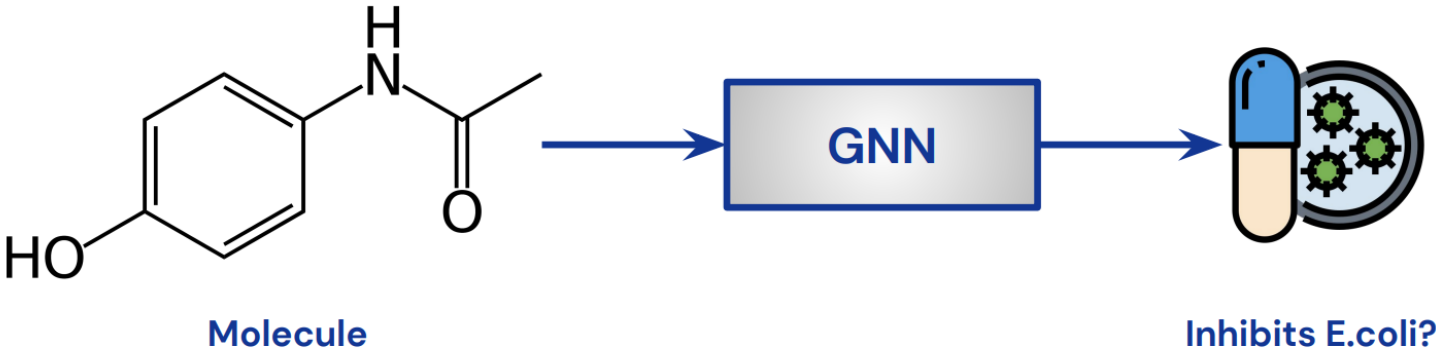
Features can be associated with each node and edge, e.g. the atomic number, the number of bonds, etc.





An interesting problem is to predict whether a molecule is a potent drug or not.

- Binary classification on whether the drug will inhibit bacterial growth (E. coli).
- Train a graph neural network (GNN) on a curated dataset $O(10^4)$ of known drugs.



—

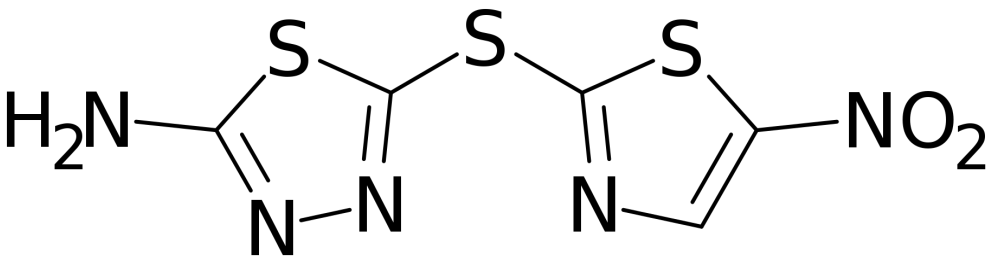
Once a GNN can accurately predict whether a molecule is a potent drug, we can use it on arbitrary new graphs to identify potential drugs:

- Run on large dataset of candidates.
- Select top-100 with the highest predicted potency.
- Manually inspect top-100.

Once a GNN can accurately predict whether a molecule is a potent drug, we can use it on arbitrary new graphs to identify potential drugs:

- Run on large dataset of candidates.
- Select top-100 with the highest predicted potency.
- Manually inspect top-100.

This very approach led to the discovery of *Halicin*, a previously overlooked compound that is a highly potent antibiotic!



NEWS | 20 February 2020

Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

Jo Marchant

Escherichia coli bacteria, coloured green, in a scanning electron micrograph. Credit: Stephanie Schuller/SPL

A pioneering machine-learning approach has identified powerful new types of antibiotic from a pool of more than 100 million molecules – including one that works against a wide range of bacteria, including tuberculosis and strains considered untreatable.

Dual use of artificial-intelligence-powered drug discovery

An international security conference explored how artificial intelligence (AI) technologies for drug discovery could be misused for de novo design of biochemical weapons. A thought experiment evolved into a computational proof.

Fabio Urbina, Filippa Lentzos, Cédric Invernizzi and Sean Ekins

The Swiss Federal Institute for NBC (nuclear, biological and chemical) Protection —Spiez Laboratory— convenes the 'convergence' conference series[1] set up by the Swiss government to identify developments in chemistry, biology and enabling technologies that may have implications for the Chemical and Biological Weapons Conventions. Meeting every two years, the conferences bring together an international group of scientific and disarmament experts to explore the current state of the art in the chemical and biological fields and their trajectories, to think through potential security implications and to consider how these implications can most effectively be managed internationally.

The meeting convenes for three days of discussion on the possibilities of harm, should the intent be there, from cutting-edge chemical and biological technologies. Our drug discovery company received an invitation to contribute a presentation on how AI technologies for drug discovery could potentially be misused.

Risk of misuse

The thought had never previously struck us. We were vaguely aware of security concerns around work with pathogens or

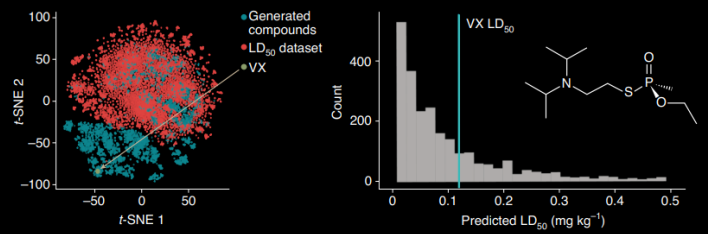


Fig. 1 | A *t*-SNE plot visualization of the LD_{50} dataset and top 2,000 MegaSyn AI-generated and predicted toxic molecules illustrating VX. Many of the molecules generated are predicted to be more toxic in vivo in the animal model than VX (histogram at right shows cut-off for VX LD_{50}). The 2D chemical structure of VX is shown on the right.

published computational machine learning models for toxicity prediction in different areas, and, in developing our presentation to the Spiez meeting, we opted to explore how AI could be used to design toxic molecules. It was a thought exercise we had not considered before that ultimately evolved into a computational proof of concept for making biochemical weapons.

Generation of new toxic molecules

We had previously designed a commercial

be used to help derive compounds for the treatment of neurological diseases (details of the approach are withheld but were available during the review process). The underlying generative software is built on, and similar to, other open-source software that is readily available[1]. To narrow the universe of molecules, we chose to drive the generative model towards compounds such as the nerve agent VX, one of the most toxic chemical warfare agents developed during the twentieth century — a few salt-sized

Graphs

Basics

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$.

Edges can be represented by an adjacency matrix

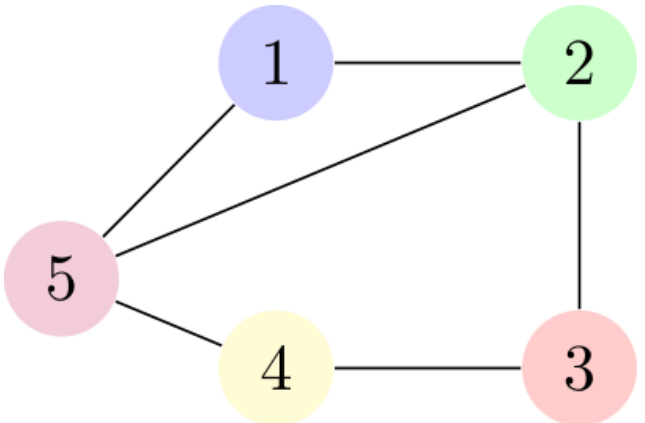
$$\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|},$$

where $\mathbf{A}_{ij} = 1$ if there is an edge from node i to j, and $\mathbf{A}_{ij} = 0$ otherwise.

The features of the nodes are represented by a matrix

$$\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$$

where d is the dimensionality of the node features.



$$\mathbf{X} = \begin{bmatrix} a & b \\ c & d \\ e & f \\ g & h \\ i & j \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

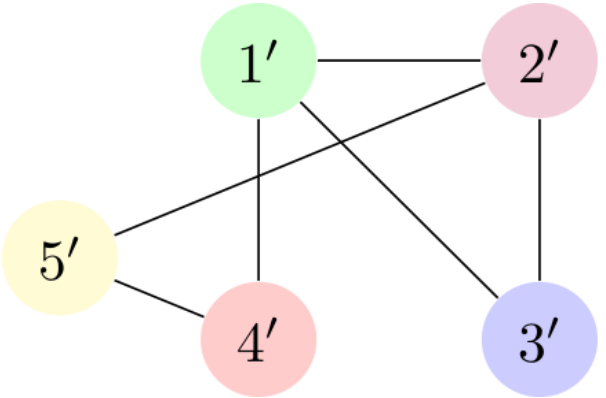
Tasks

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its $(\mathbf{X}, \mathbf{A})$ representation, we want to make

- graph-level predictions $y \in \mathcal{Y}$, using graph-level functions $f(\mathbf{X}, \mathbf{A})$,
- nodel-level predictions $\mathbf{y} \in \mathcal{Y}^{|\mathcal{V}|}$, using node-level functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$.

Permutation matrices

A permutation matrix $\mathbf{P} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is a matrix with $|\mathcal{V}|$ rows and columns, where each row and column contains a single 1 and the remaining entries are 0.



$$\mathbf{PX} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \\ e & f \\ g & h \\ i & j \end{bmatrix} = \begin{bmatrix} c & d \\ i & j \\ a & b \\ e & f \\ g & h \end{bmatrix}$$

$$\mathbf{PAP}^T = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Permutation invariance

The very representation $(\mathbf{X}, \mathbf{A})$ of a graph imposes a *node ordering*, which does not align with the nodes and edges being unordered. Permuting the nodes of the graph should not modify the results!

For graph-level tasks, we want permutation invariance, i.e.

$$f(\mathbf{PX}, \mathbf{PAP}^T) = f(\mathbf{X}, \mathbf{A})$$

for all permutation matrices $\mathbf{P}$.

Permutation equivariance

For node-level tasks, we want permutation equivariance, i.e.

$$\mathbf{F}(\mathbf{PX}, \mathbf{PAP}^T) = \mathbf{PF}(\mathbf{X}, \mathbf{A})$$

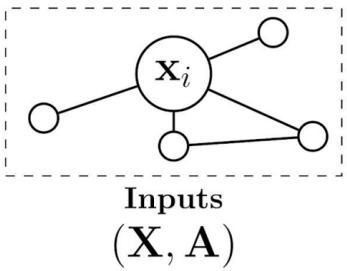
for all permutation matrices $\mathbf{P}$.

Graph neural networks

Blueprint

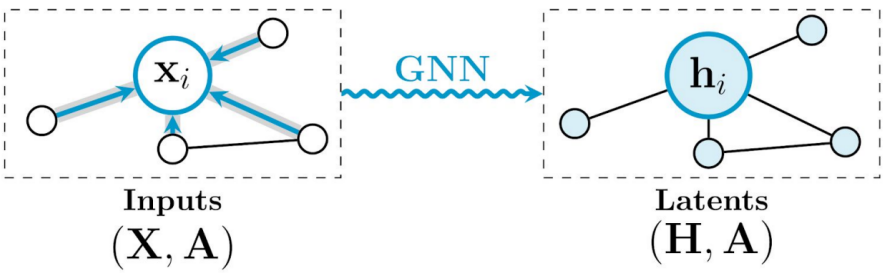
Graph neural networks (GNNs) are neural networks that operate on graphs. They implement graph-level permutation invariance and node-level permutation equivariance.

The general blueprint is to stack permutation equivariant function(s), optionally followed by a permutation invariant function.



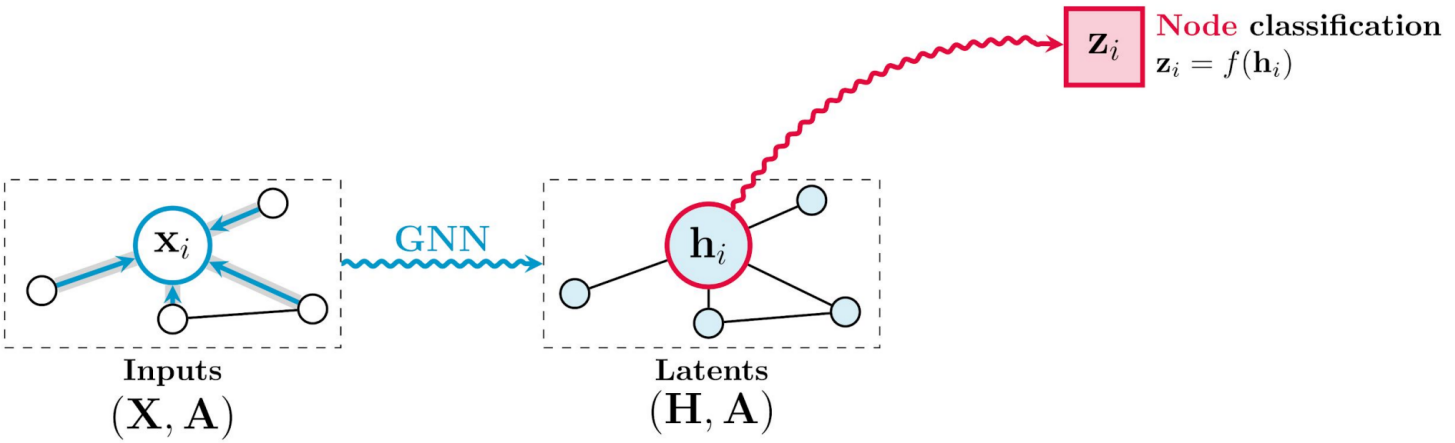
—

Image credits: Petar Veličkovi. , Everything is Connected: Graph Neural Networks, 2023.



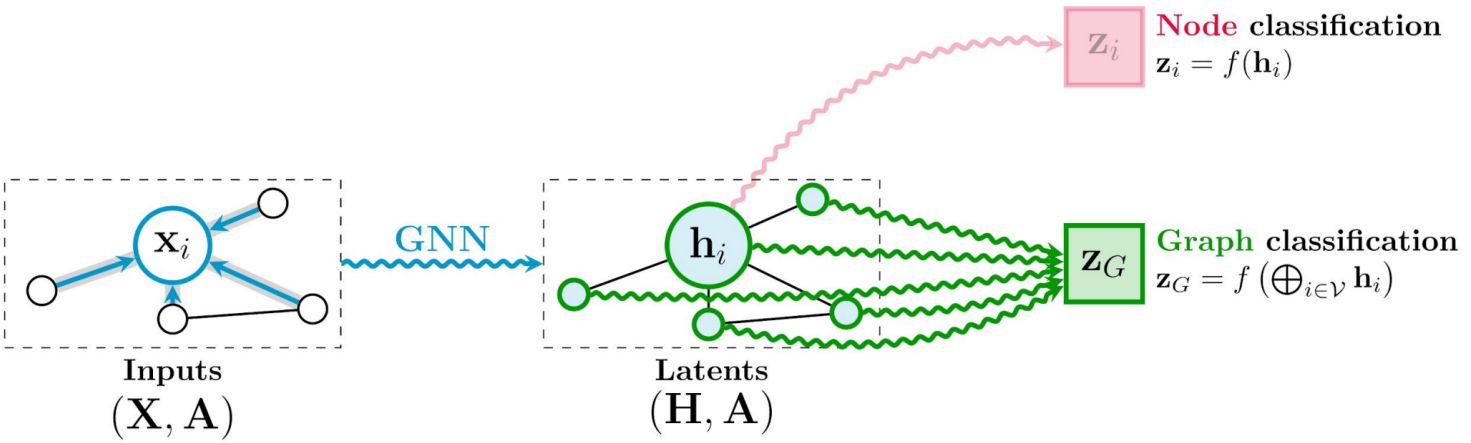
—

Image credits: Petar Veličkovi. , Everything is Connected: Graph Neural Networks, 2023.



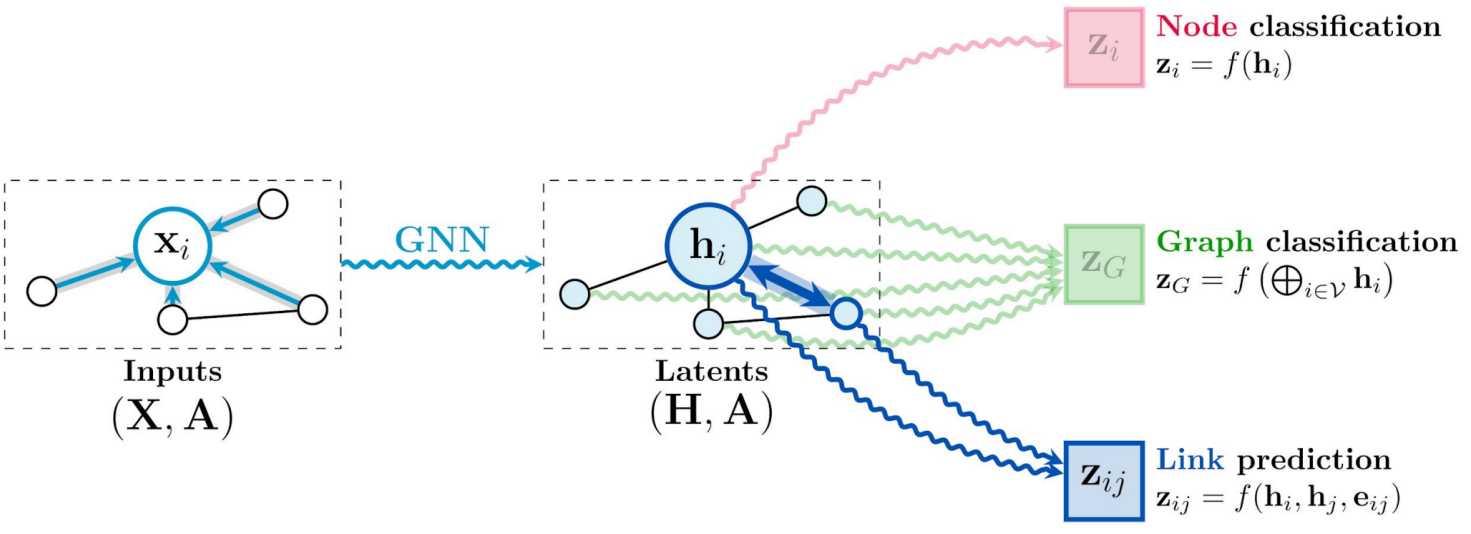
—

Image credits: Petar Veličkovi. , Everything is Connected: Graph Neural Networks, 2023.



—

Image credits: Petar Veličkovi. , Everything is Connected: Graph Neural Networks, 2023.



—

Image credits: Petar Veličkovi. , Everything is Connected: Graph Neural Networks, 2023.

If we denote $\mathbf{h}_i = \phi(\mathbf{x}_i, \mathbf{X})$ for $i \in \mathcal{V}$, then

$$f(\mathbf{X}, \mathbf{A}) = g(\bigoplus_{i \in \mathcal{V}} \mathbf{h}_i)$$

$$\mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{\mathcal{V}} \end{bmatrix}$$

where g is an arbitrary function, ϕ is a *shared* permutation invariant function in $\mathbf{X}$, and $\bigoplus$ is a permutation invariant aggregator (e.g., sum, average or max).

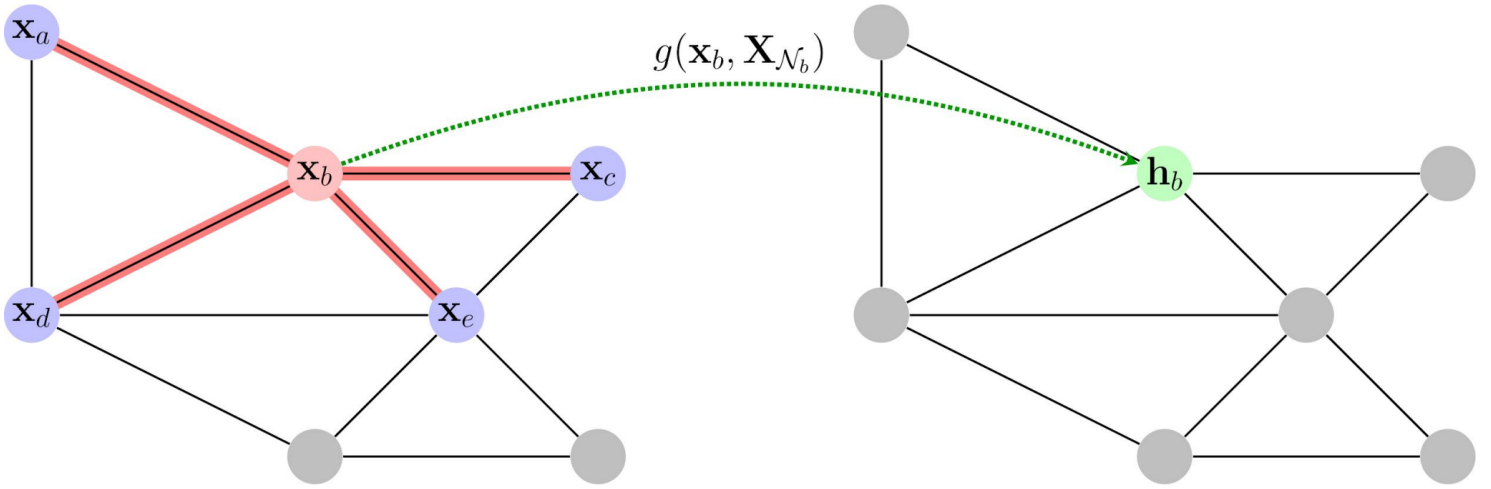
Locality

A strong inductive bias of graph neural networks is based on locality. It assumes that the information about a node is most relevant to its close neighbors rather than distant ones.

For a node $i \in \mathcal{V}$, we denote its *neighborhood* as $\mathcal{N}_i = \{j \in \mathcal{V} \mid (i,j) \in \mathcal{E} \vee (j,i) \in \mathcal{E}\} \cup \{i\}$.

Accordingly, ϕ is redefined as a *local* function that is applied to the node i and its neighborhood $\mathcal{N}_i$ to compute $\mathbf{h}_i = \phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})$.

As previously, $\mathbf{F}$ is permutation equivariant if ϕ is permutation invariant in $\mathbf{X}_{\mathcal{N}_i}$.



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

—

Note: g denotes ϕ.

Layers

Permutation equivariant functions $\mathbf{F}$ are often referred to as *GNN layers*.

Similarly to regular layers, a GNN layer computes a new representation

$$\mathbf{H} = \mathbf{F}(\mathbf{X}, \mathbf{A}) = \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{|\mathcal{V}|} \end{bmatrix} = \begin{bmatrix} \phi(\mathbf{x}_1, \mathbf{X}_{\mathcal{N}_1}) \\ \vdots \\ \phi(\mathbf{x}_{|\mathcal{V}|}, \mathbf{X}_{\mathcal{N}_{|\mathcal{V}|}}) \end{bmatrix}$$

from the input representation $(\mathbf{X}, \mathbf{A})$ of the graph.

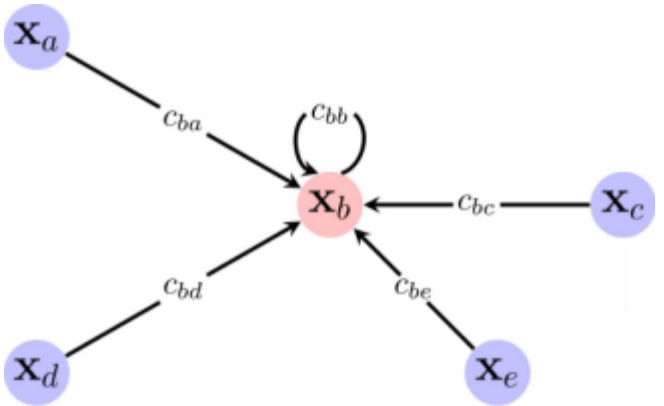
GNN layers are usually classified in three spatial flavors depending on how they implement the propagation operator ϕ:

- Convolutional
- Attentional
- Message-passing

Convolutional

Features of neighboring nodes are aggregated with fixed coefficients c_{ij}:

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



Example:

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \frac{1}{|\mathcal{N}_i|}\mathbf{W}^T\mathbf{x}_j\right)$$

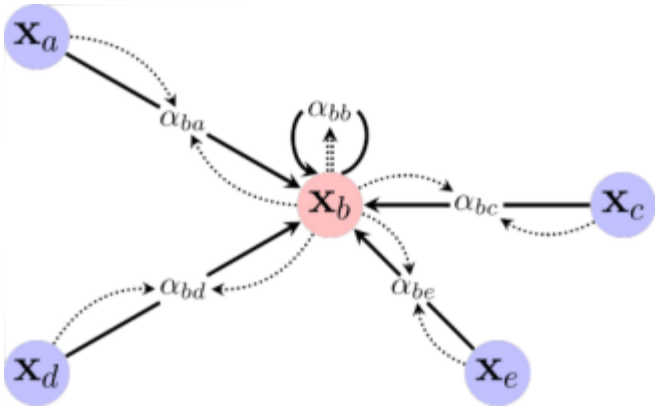
—

Image credits: Bronstein et al., Geometric Deep Learning, 2021.

Attentional

Features of neighboring nodes are aggregated with implicit weights via an attention mechanism:

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



Example:

$$\mathbf{h}_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}^T\mathbf{x}_j\right)$$

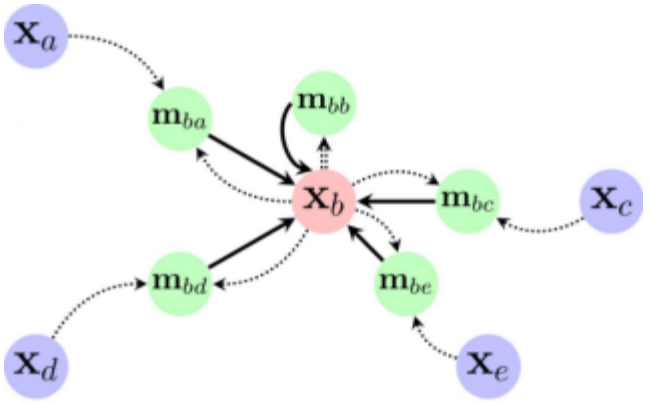
—
Image credits: Bronstein et al., Geometric Deep Learning, 2021.

Message passing

Compute arbitrary vectors (or *messages*) to be sent across the edges of the graph:

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \varphi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

This is the most generic form of GNN layers.



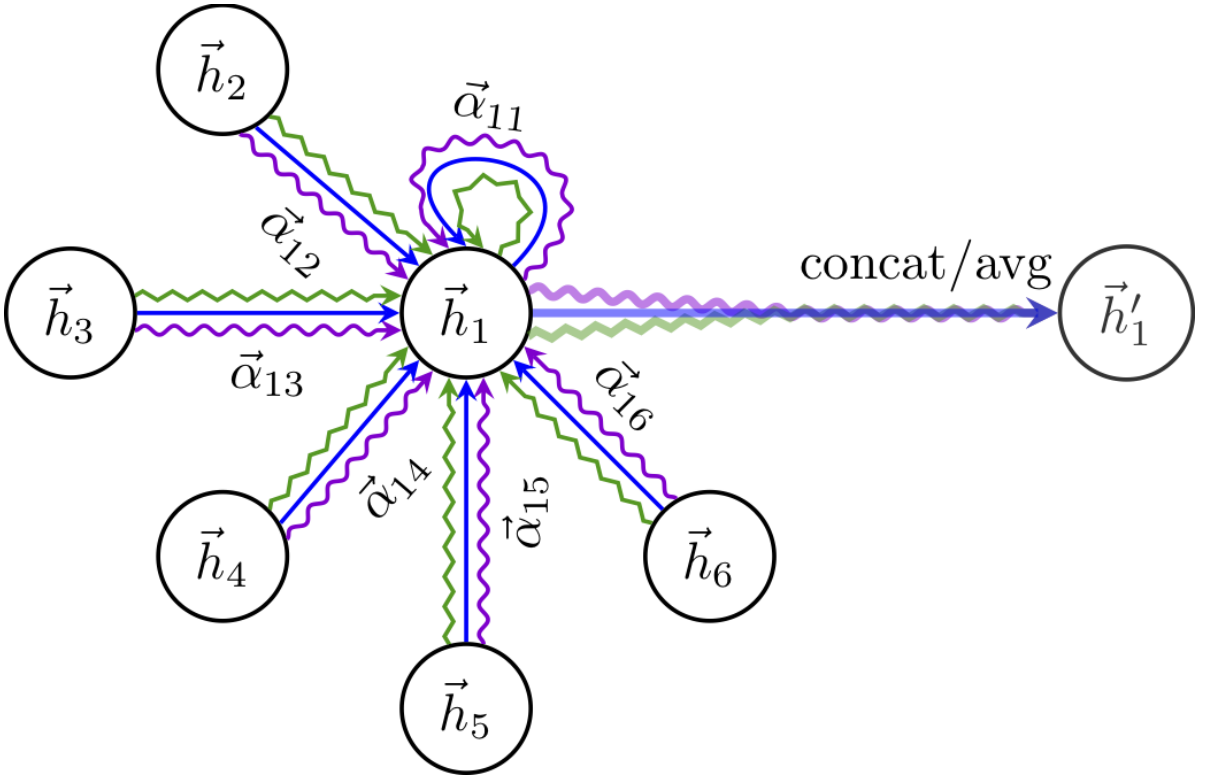
—
Image credits: Bronstein et al., Geometric Deep Learning, 2021.

Parallel composition

Each flavor of GNN layers can be composed in parallel and then combined (e.g., by concatenation or average) to form the final representation $\mathbf{H}$ as

$$\begin{aligned}\mathbf{H} &= \text{concat}(\mathbf{H}_1, ..., \mathbf{H}_K) \\ \mathbf{H}_k &= \mathbf{F}_k(\mathbf{X}, \mathbf{A})\end{aligned}.$$

This is similar to having multiple kernels in a convolutional layer or multiple attention heads in an attention layer.



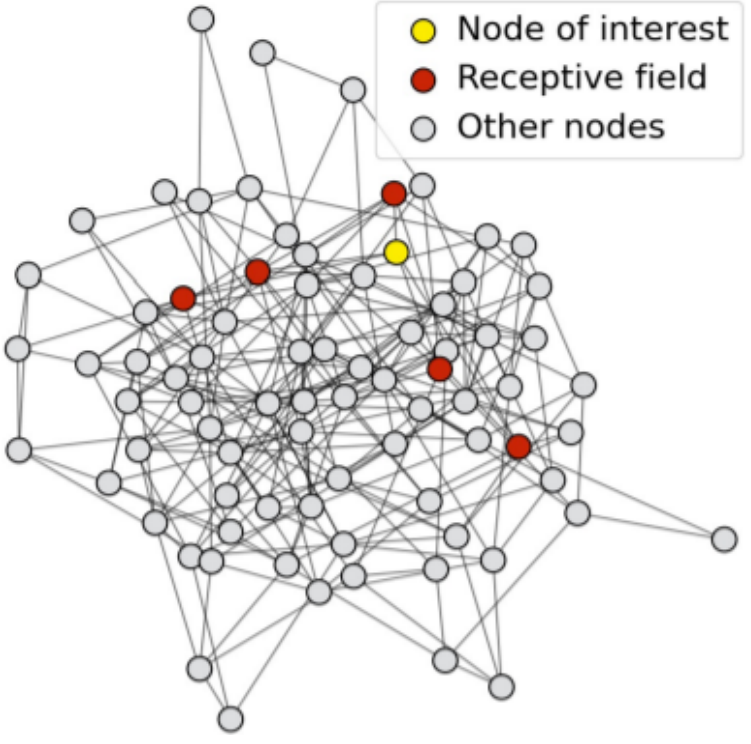
—

Image credits: Petar Veličkovi. , Graph Attention Networks, 2017.

Sequential composition

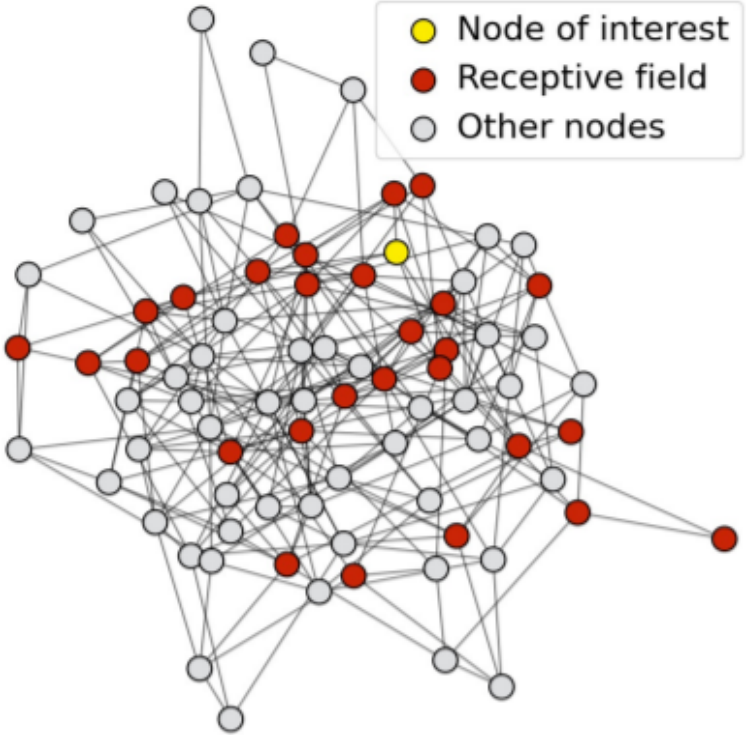
Layers can be stacked in series to form deep graph neural networks:

$$
\begin{aligned}
\mathbf{H}_0 &= \mathbf{X} \\
\mathbf{H}_1 &= \mathbf{F}_1(\mathbf{H}_0, \mathbf{A}) \\
&\dots \\
\mathbf{H}_L &= \mathbf{F}_L(\mathbf{H}_{L-1}, \mathbf{A})
\end{aligned}
$$



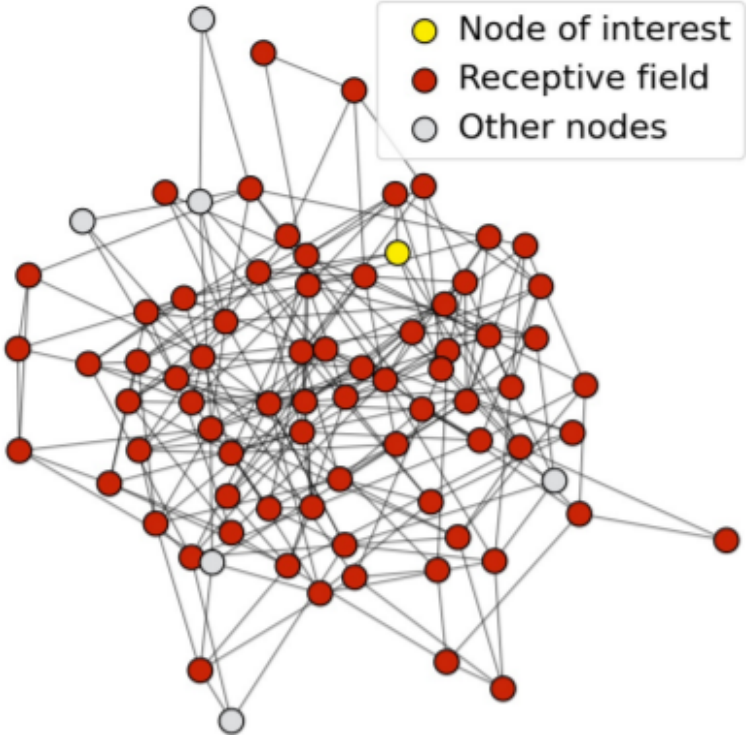
1-layer GNN

Stacking layers in series increases the effective receptive field of each node.



2-layer GNN

Stacking layers in series increases the effective receptive field of each node.



3-layer GNN

Stacking layers in series increases the effective receptive field of each node.

Global pooling

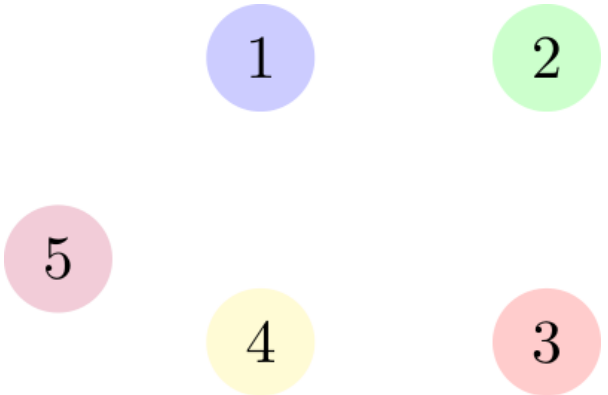
To make graph-level predictions, the node representations $\mathbf{H}$ are aggregated into a single vector

$$\bar{\mathbf{h}} = \bigoplus_{i \in \mathcal{V}} \mathbf{h}_i$$

using a permutation invariant pooling operator (e.g., max, mean, sum, etc.)

$\bar{\mathbf{h}}$ is then usually passed through a regular MLP g to make the final prediction.

Special cases

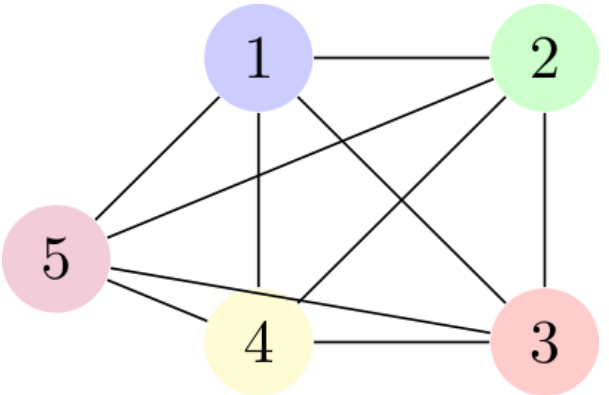


Empty graph

When the set of edges is empty, the graph reduces to a set of isolated nodes. In this case, $\mathcal{N}_i = \{i\}$ for all i and the propagation operator ϕ degenerates to a function φ applied in isolation to each node,

$$g(\bigoplus_{i \in \mathcal{V}} \phi(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i})) = g(\bigoplus_{i \in \mathcal{V}} \varphi(\mathbf{x}_i)).$$

Such a structure is often referred to as a *Deep Set* and can be considered as a special case of a GNN.



Complete graph

When nodes are expected to have a relational structure but the edges are unknown, it is common to assume that all nodes are connected to each other. In this case, $\mathcal{N}_i = \mathcal{V}$ for all i and no coefficient c_{ij} of interaction can be assumed.

For convolutional GNN layers,

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{V}} \varphi(\mathbf{x}_j)\right),$$

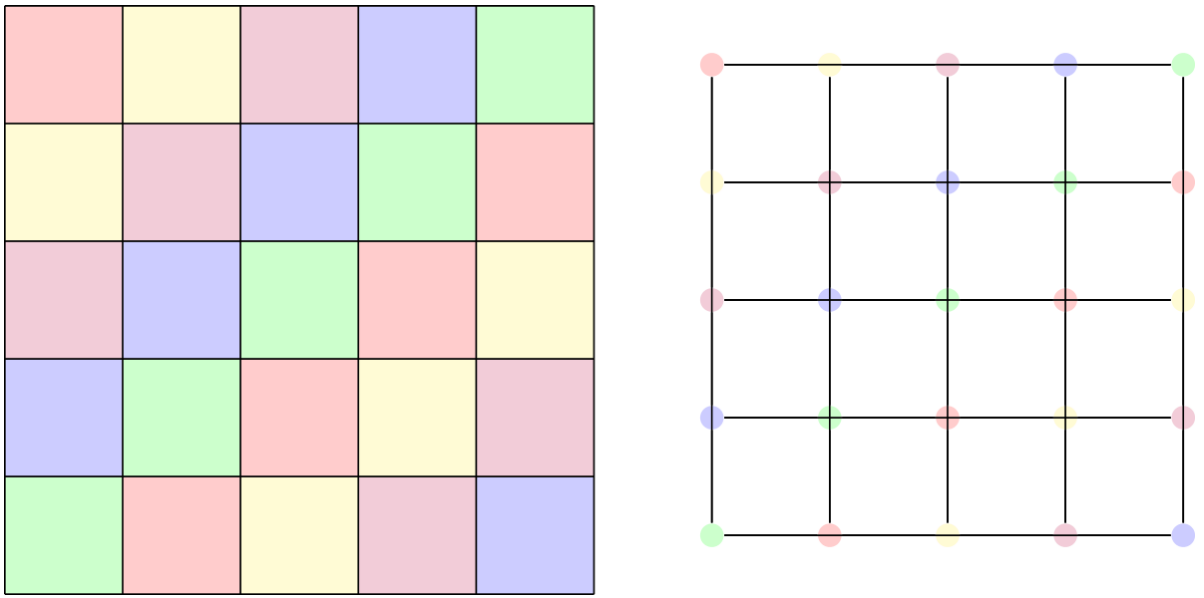
in which $\bigoplus_{j \in \mathcal{V}} \varphi(\mathbf{x}_j)$ is identical for all nodes i. The model is thus equivalently expressive to ignoring that input altogether.

For attentional GNN layers however,

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{V}} a(\mathbf{x}_i, \mathbf{x}_j)\varphi(\mathbf{x}_j)\right),$$

which yields the **self-attention** layer.

In other words, the transformer architecture is a special case of a GNN.



Images as graphs

Images can be represented as graphs, where each pixel is a node and the edges are defined by the spatial adjacency of the pixels.

That is, convolutional neural networks can be seen as a special case of GNNs.

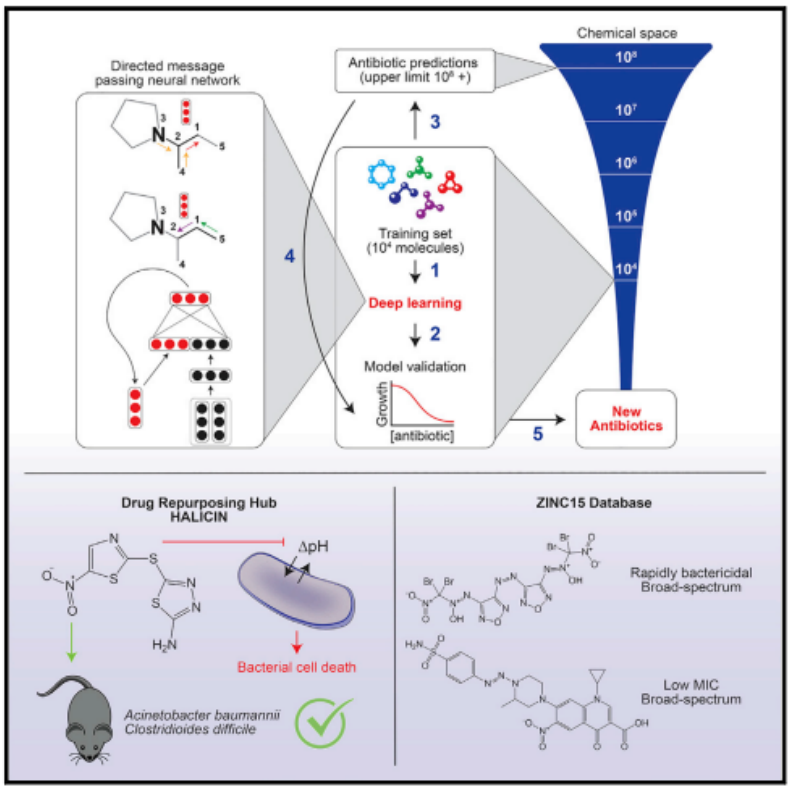
Applications

Graph neural networks for antiobiotic discovery

Cell

A Deep Learning Approach to Antibiotic Discovery

Graphical Abstract



Authors

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, ..., Tommi S. Jaakkola, Regina Barzilay, James J. Collins

Correspondence

regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

In Brief

A trained deep neural network predicts antibiotic activity in molecules that are structurally different from known antibiotics, among which Halicin exhibits efficacy against broad-spectrum bacterial infections in mice.

—

Credits: Stokes et al, A Deep Learning Approach to Antibiotic Discovery, 2020.

Graph neural network for object detection in point clouds

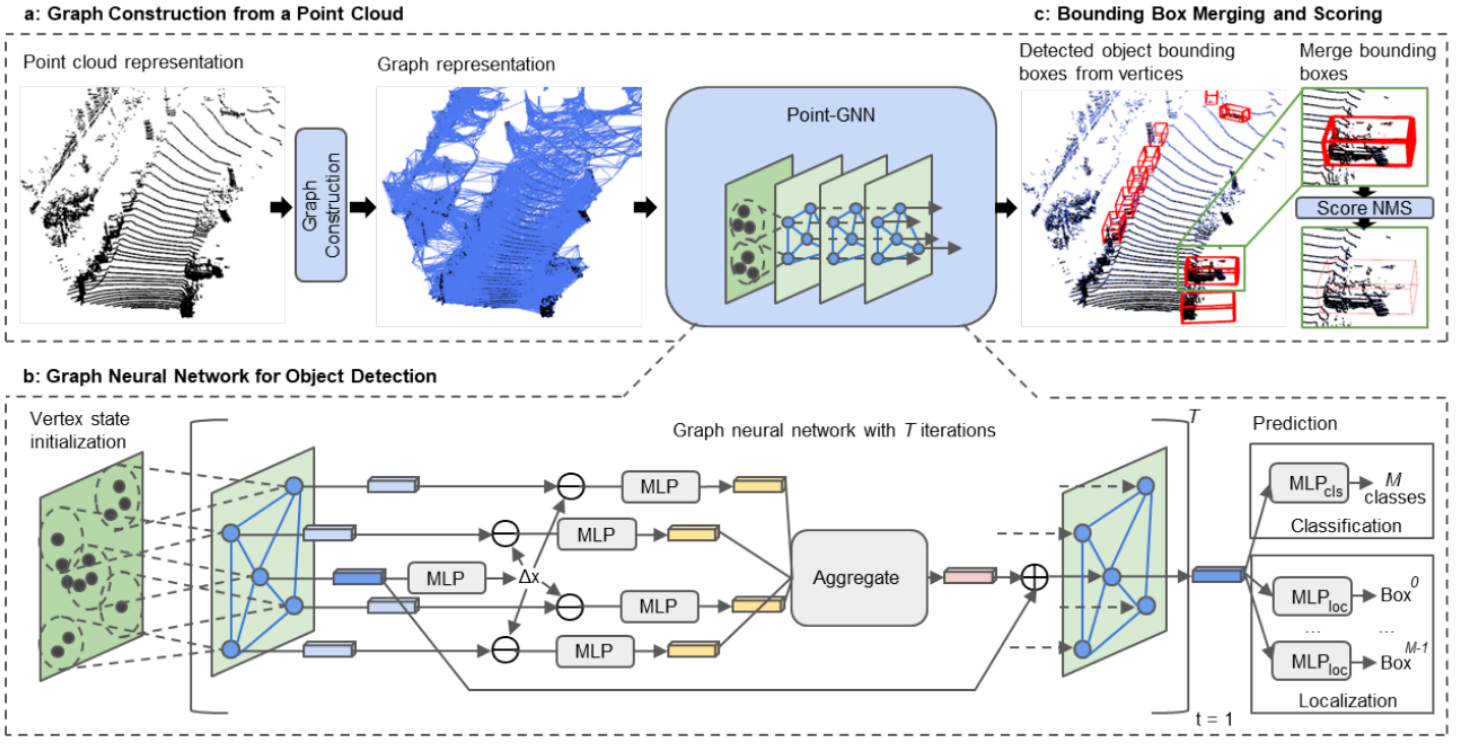
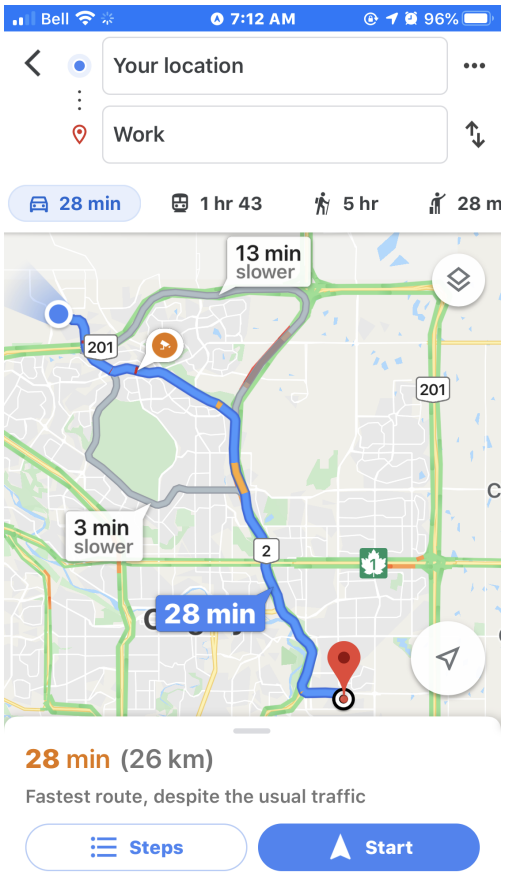


Figure 2. The architecture of the proposed approach. It has three main components: (a) graph construction from a point cloud, (b) a graph neural network for object detection, and (c) bounding box merging and scoring.

—

Credits: Shi and Rajkumar, Point-GNN, 2020.

Travel time prediction in Google Maps



—

Credits: Derrow-Pinion et al, 2021.

Learning to simulate physics with graph networks

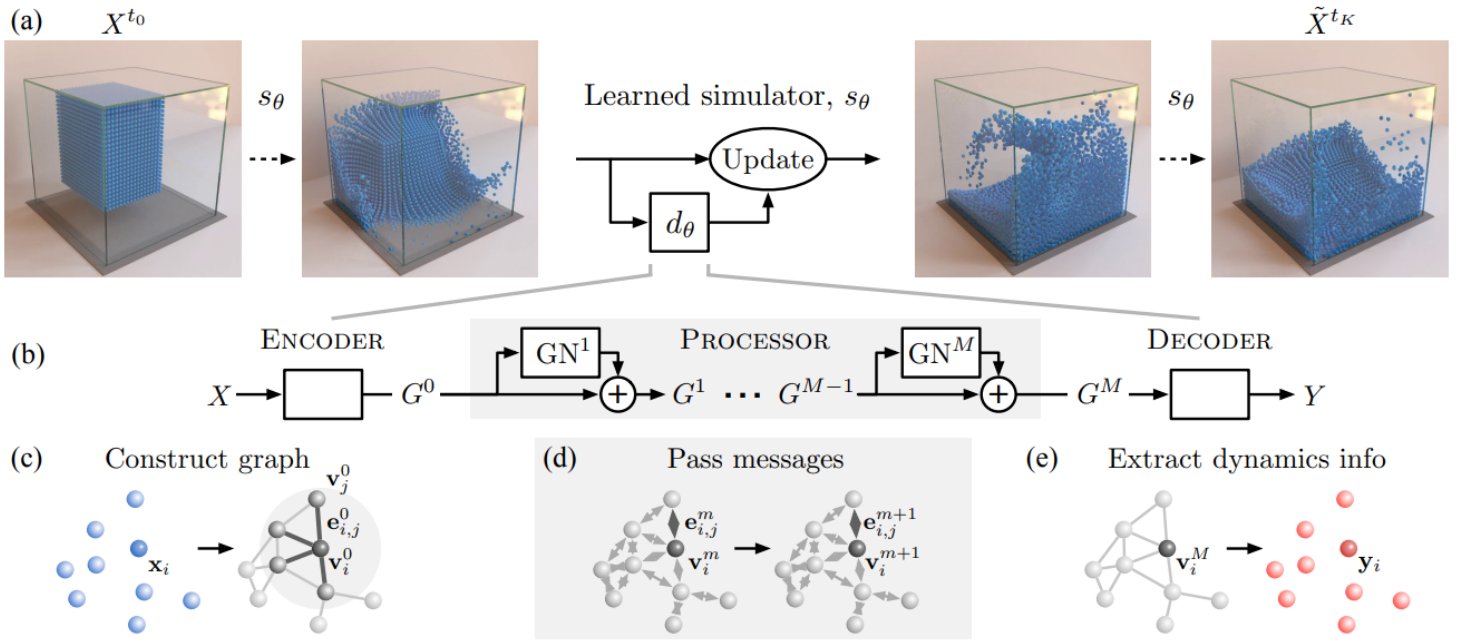
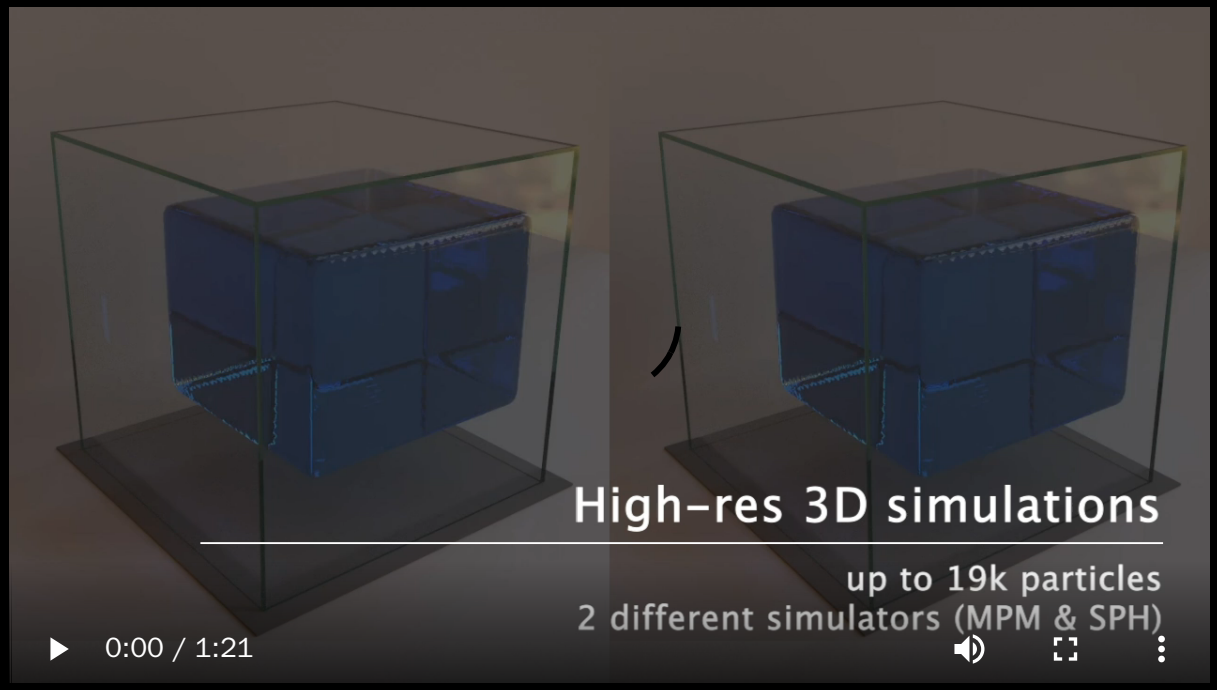


Figure 2. **(a)** Our GNS predicts future states represented as particles using its learned dynamics model, d_θ, and a fixed update procedure. **(b)** The d_θ uses an "encode-process-decode" scheme, which computes dynamics information, Y, from input state, X. **(c)** The ENCODER constructs latent graph, G^0, from the input state, X. **(d)** The PROCESSOR performs M rounds of learned message-passing over the latent graphs, $G^0, \ldots, G^M$. **(e)** The DECODER extracts dynamics information, Y, from the final latent graph, G^M.

—

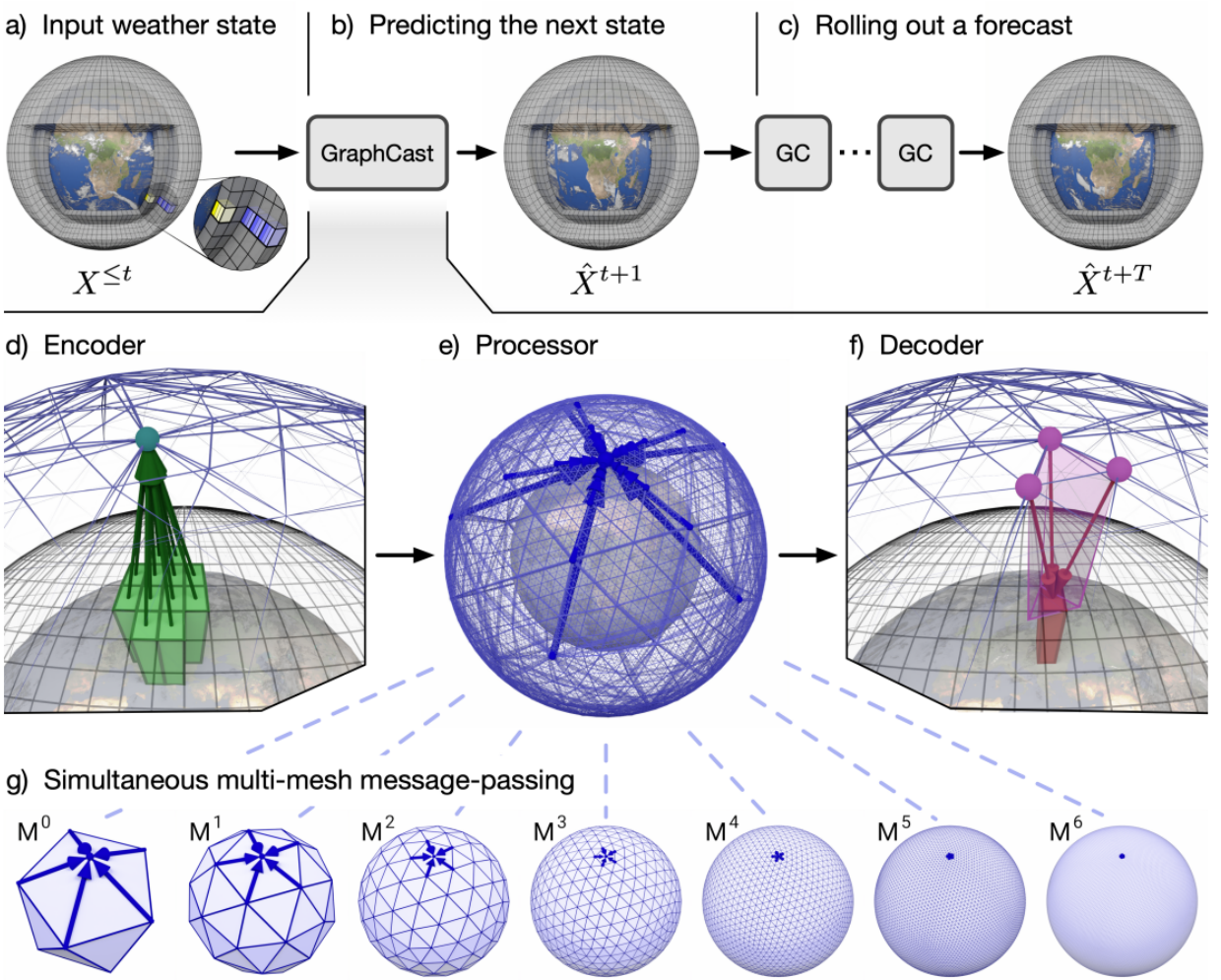
Credits: Sanchez-Gonzalez et al, 2020.



—

Credits: Sanchez-Gonzalez et al, 2020.

Medium-range global weather forecasting



—

Credits: Lam et al, 2022.

The end.