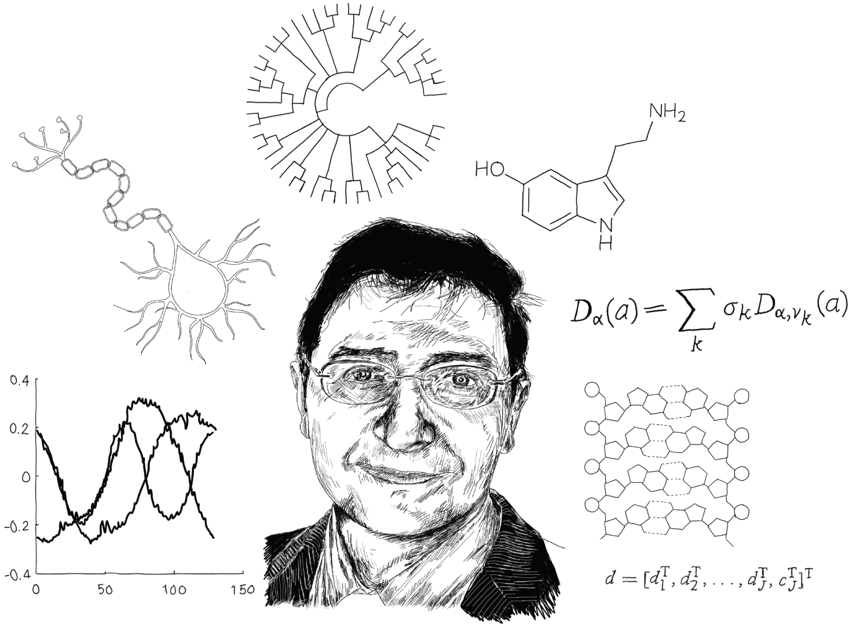
Geometric deep learning and diffusion approach across scales


NH$_2$
HO
N
H
$D_\alpha(a) = \sum_k \sigma_k D_{\alpha,\nu_k}(a)$
0.4
0.2
0
-0.2
-0.4
0
50
100
150
$d = [d_1^T, d_2^T, \ldots, d_J^T, c_J^T]^T$

Prof. Pietro Lio’

PhD Complex Systems and Non Linear dynamics (Engineering, Firenze)
PhD Genetics (Pavia)

Dept of Computer Science and Technology
Cambridge Center for AI in Medicine
University of Cambridge
pl219@cam.ac.uk

UNIVERSITY OF CAMBRIDGE CCAIM Academia Europaea AAIA Asia-Pacific Artificial Intelligence Association ellis European Laboratory for Learning and Intelligent Systems

Plan

- Geometric Deep Learning

- The interesting crosstalk between diffusion models and graph representation learning.

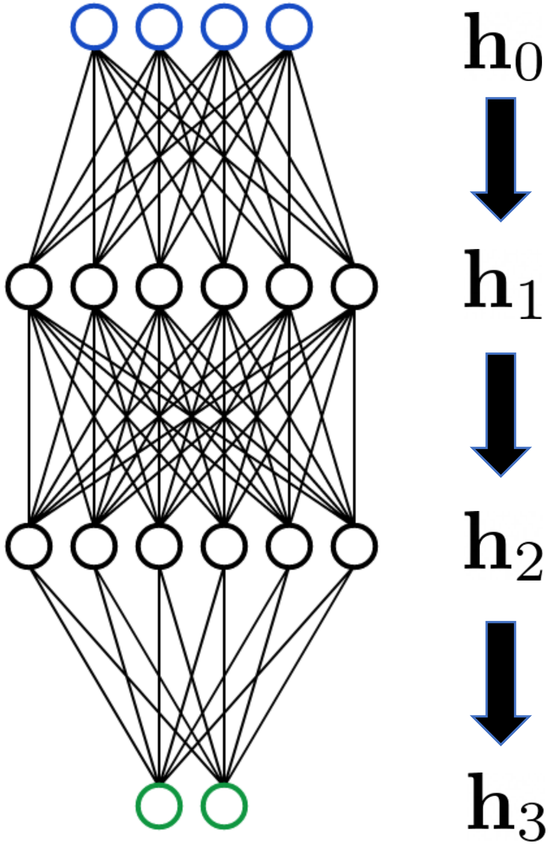
- Methological applications across scales (molecules, cells, tissues, full body, population)

Neural Networks

- Constructed of layers, take vector as input, multiply by weight matrix Wn, add bias vector bn, apply non-linear element-wise activation function

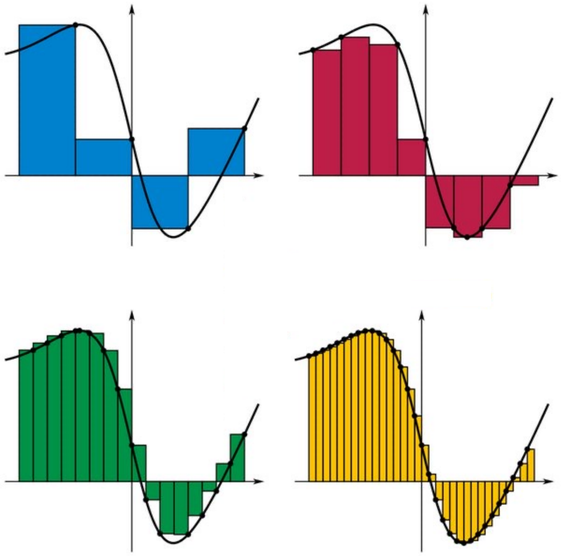
$$\mathbf{h}_{n+1} = \sigma(W_n \mathbf{h}_n + \mathbf{b}_n)$$

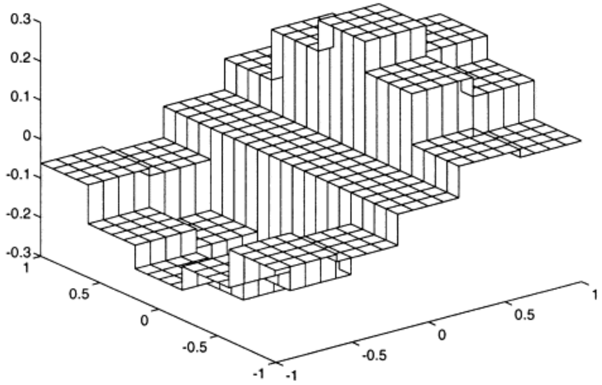
- Stack the layers to make a neural network



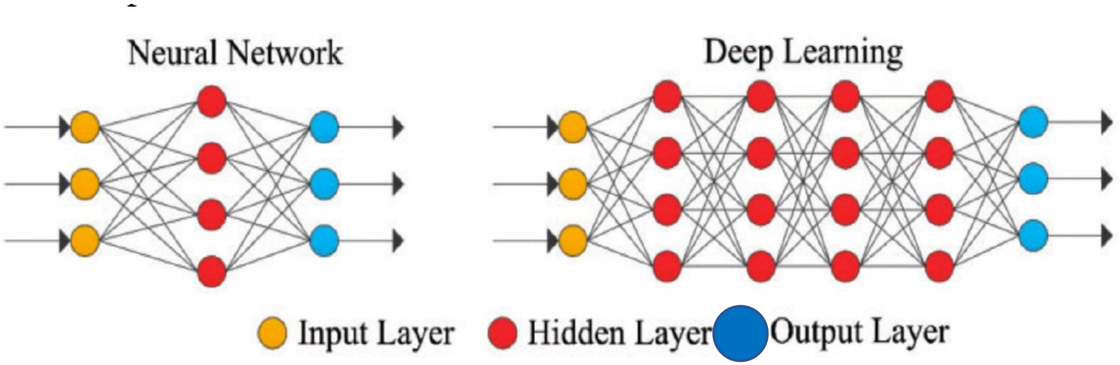
Neural Networks - Benefits

- Universal Function Approximator – can approximate any function to an arbitrary degree
- Not just universal approximators but seem to be better at generalizing in certain situations, than other universal approximators (large amount of data)
- Train with gradient descent and backpropagation (if a model is differentiable it can be trained this way), can construct complicated architectures

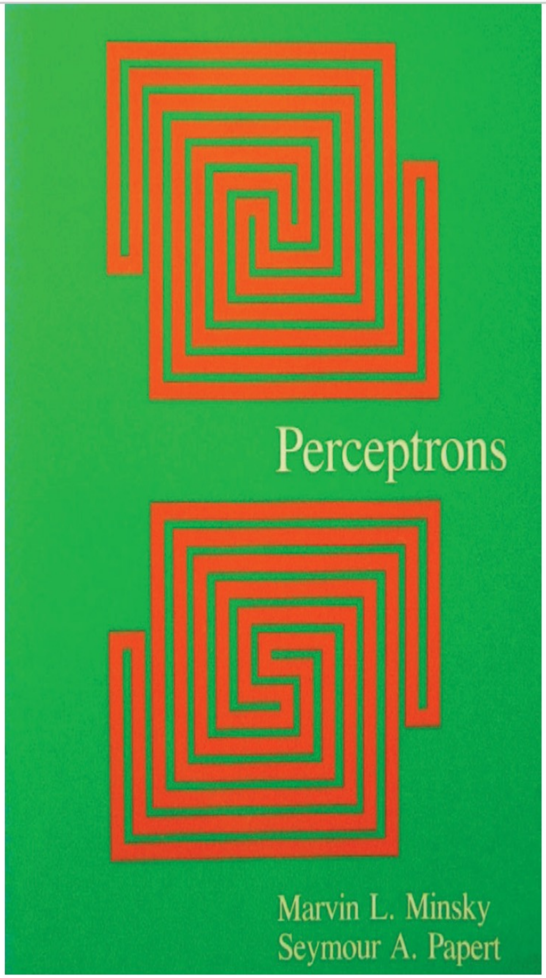




Complexity

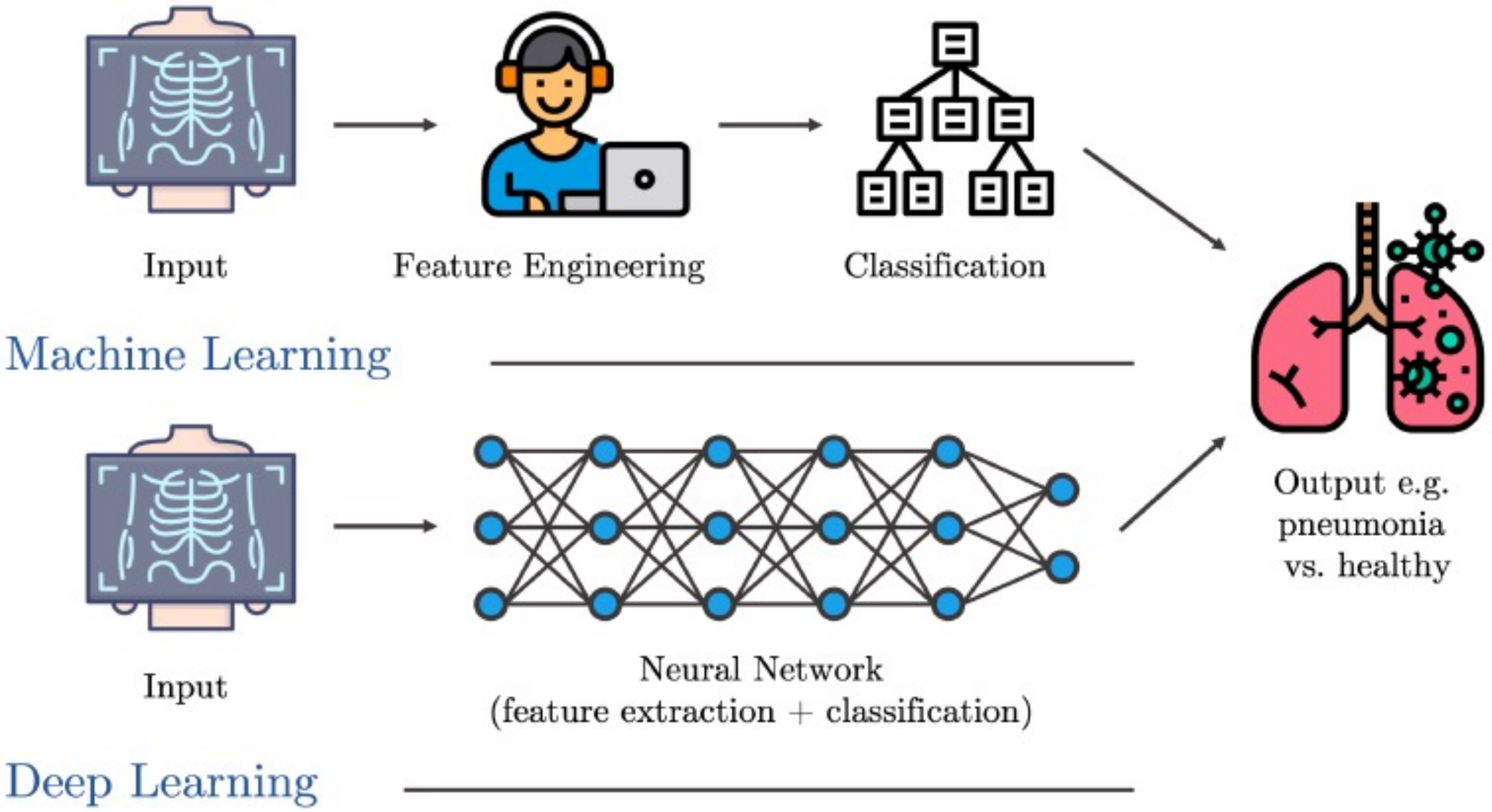


Link to Image Source

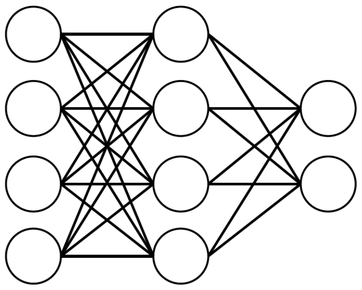


The book cover of an expanded edition of *Perceptrons*. The two red spirals look the same but they aren't. The top one is two disconnected spirals, but the bottom one is a single connected spiral, which you can verify by tracing the insides of the loops with a pencil. Minsky and Papert proved that a perceptron cannot distinguish between these two objects. Can you see the difference without tracing? Why not?

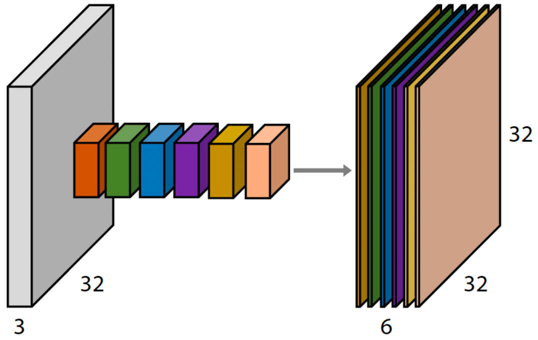
A traditional machine learning/bioinformatics vs deep learning



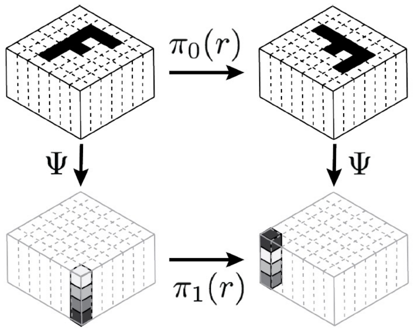
Architectures



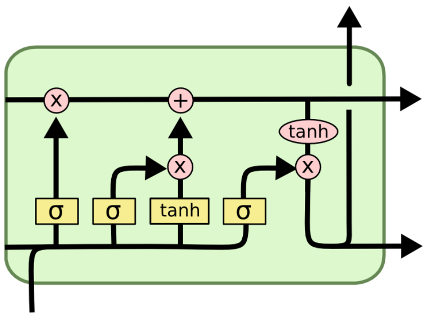
Perceptrons
Function regularity



CNNs
Translation

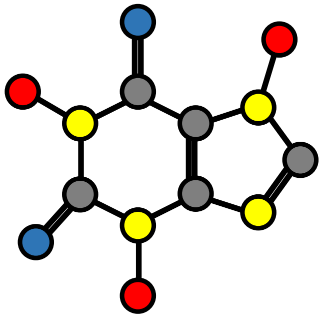


Spherical CNNs
Rotation



LSTMs
Time warping

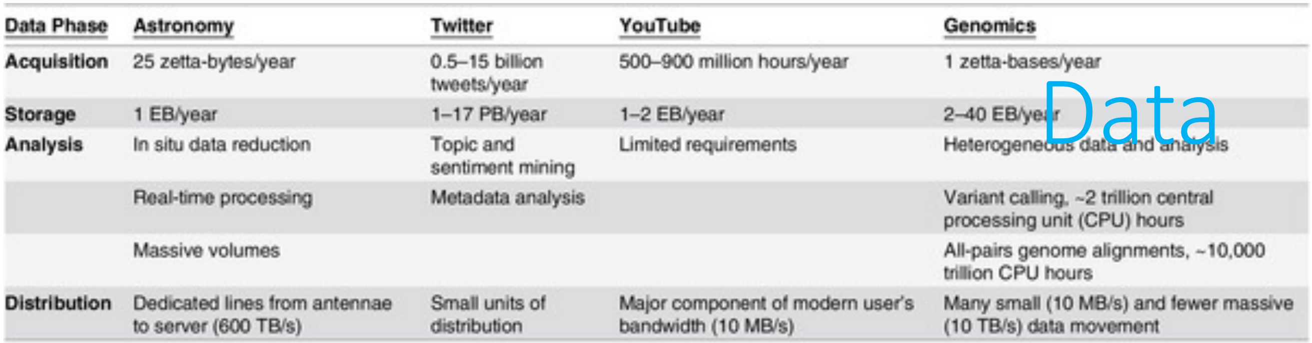
Deep Sets
Permutation



GNNs
Permutation

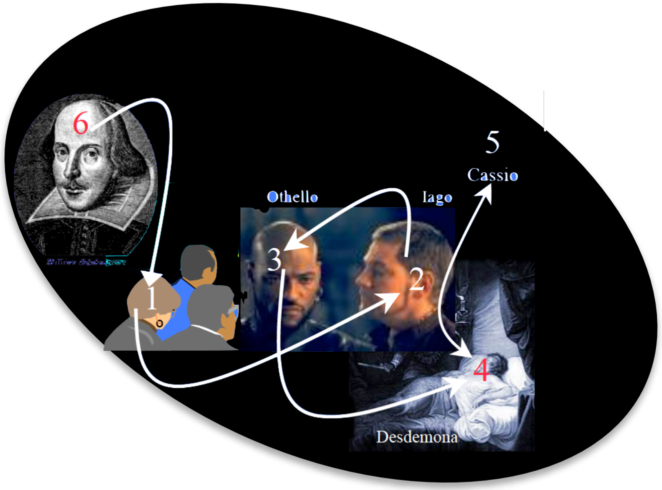
Intrinsic CNNs
Isometry / Gauge choice

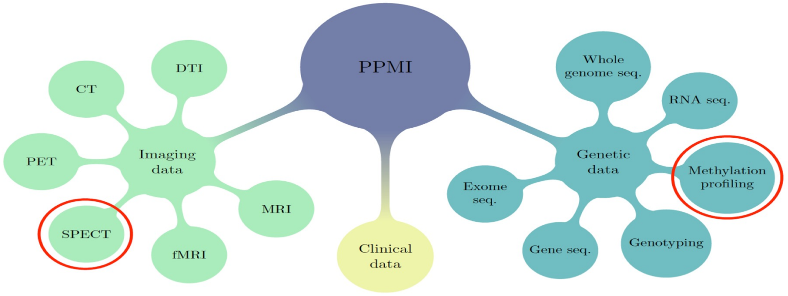
Data

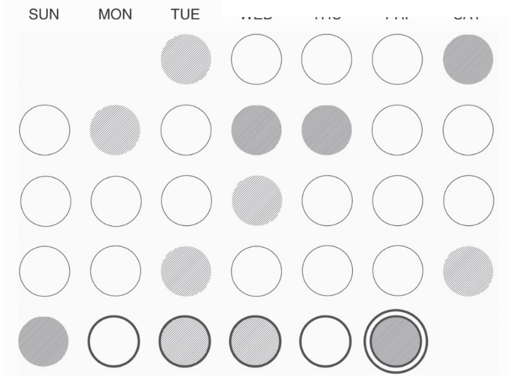


Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

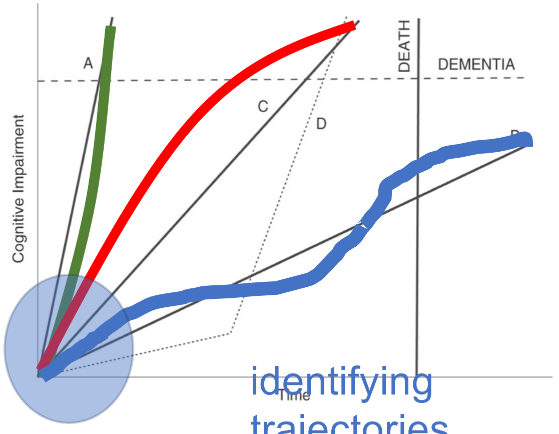
doi:10.1371/journal.pbio.1002195.t001



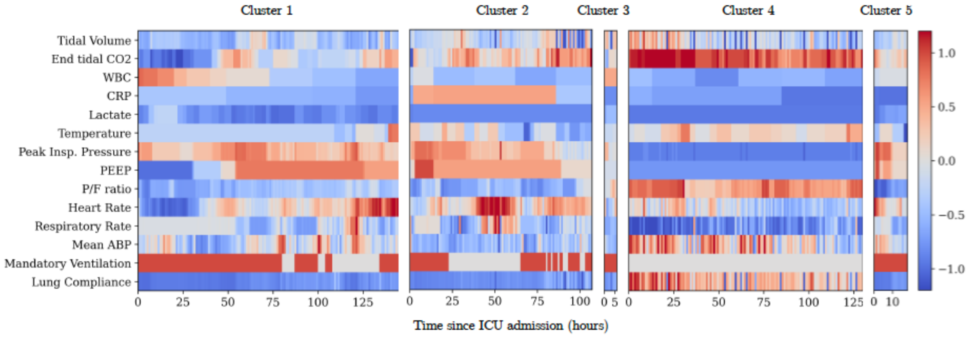




A series of “good days” and “bad days



identifying trajectories

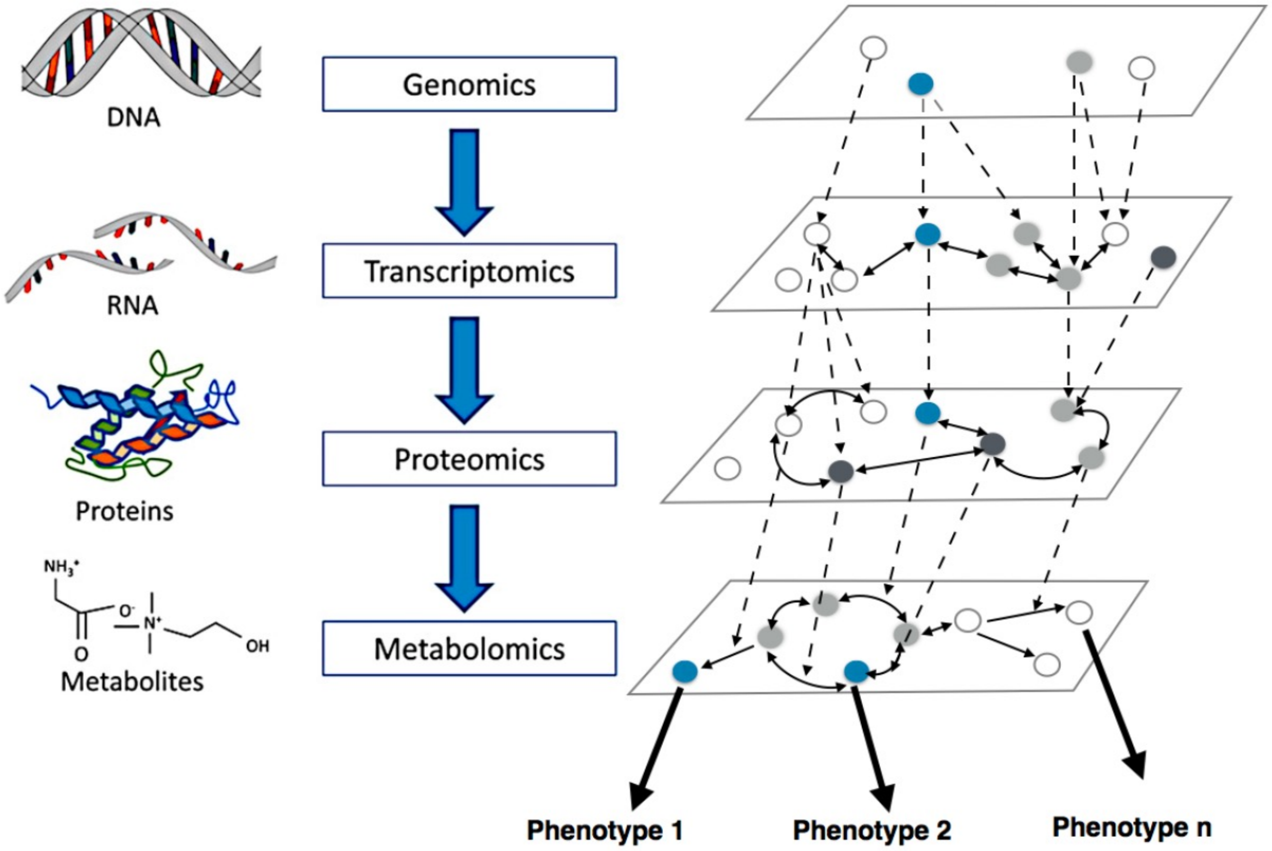


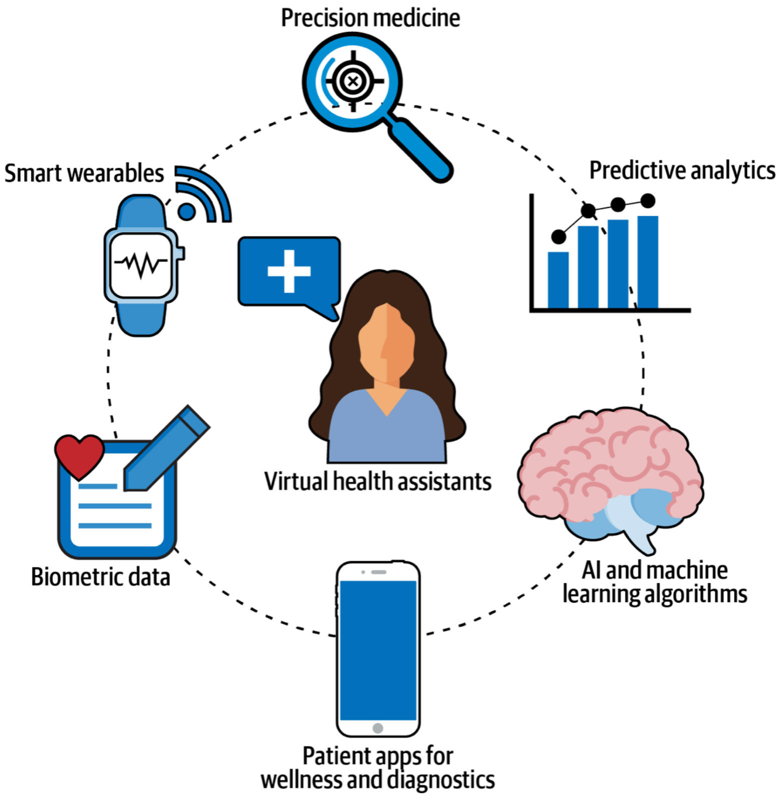
Patient stratification

Benchside

Narrowing the distance

Bedside



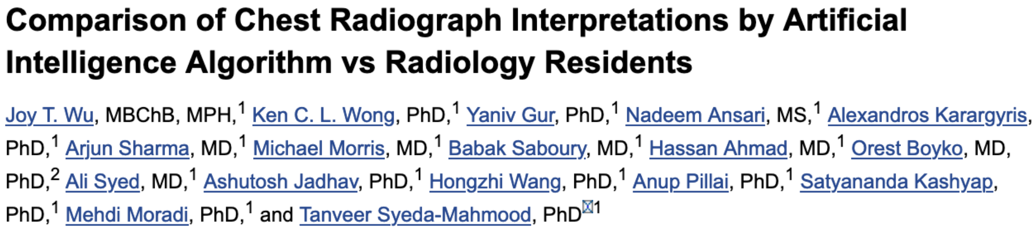

Precision medicine
Smart wearables
Predictive analytics
Virtual health assistants
Biometric data
AI and machine learning algorithms
Patient apps for wellness and diagnostics

Disease state management platform

Making expert knowledge scalable

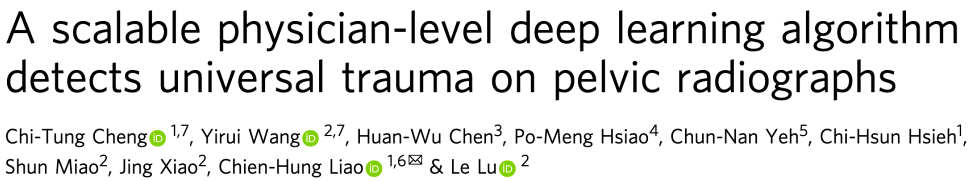
Dermatologist-level classification of skin cancer with deep neural networks

Andre Esteva[1]*, Brett Kuprel[1]*, Roberto A. Novoa[2,3], Justin Ko[2], Susan M. Swetter[2,4], Helen M. Blau[5] & Sebastian Thrun[6]



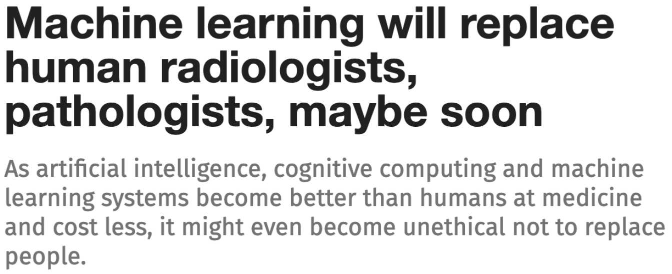
Comparison of Chest Radiograph Interpretations by Artificial Intelligence Algorithm vs Radiology Residents

Joy T. Wu, MBChB, MPH,[1] Ken C. L. Wong, PhD,[1] Yaniv Gur, PhD,[1] Nadeem Ansari, MS,[1] Alexandros Karargyris, PhD,[1] Arjun Sharma, MD,[1] Michael Morris, MD,[1] Babak Saboury, MD,[1] Hassan Ahmad, MD,[1] Orest Boyko, MD, PhD,[2] Ali Syed, MD,[1] Ashutosh Jadhav, PhD,[1] Hongzhi Wang, PhD,[1] Anup Pillai, PhD,[1] Satyananda Kashyap, PhD,[1] Mehdi Moradi, PhD,[1] and Tanveer Syeda-Mahmood, PhD✉[1]



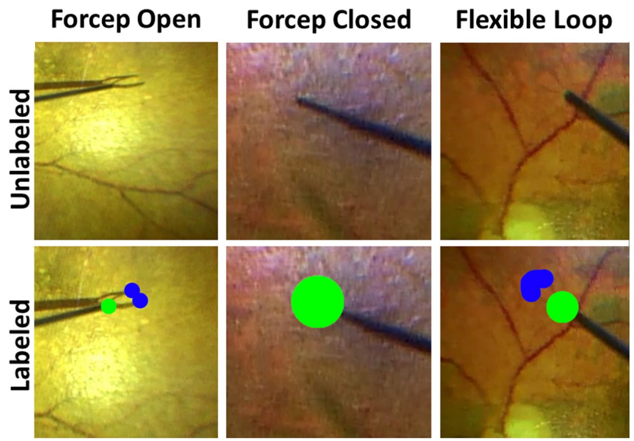
A scalable physician-level deep learning algorithm detects universal trauma on pelvic radiographs

Chi-Tung Cheng[1,7], Yirui Wang[2,7], Huan-Wu Chen[3], Po-Meng Hsiao[4], Chun-Nan Yeh[5], Chi-Hsun Hsieh[1], Shun Miao[2], Jing Xiao[2], Chien-Hung Liao[1,6]✉ & Le Lu[2]

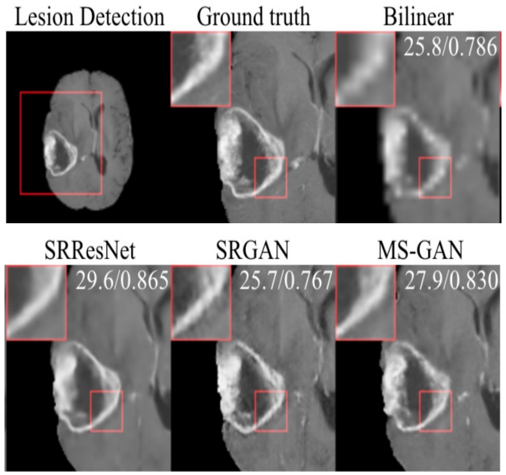


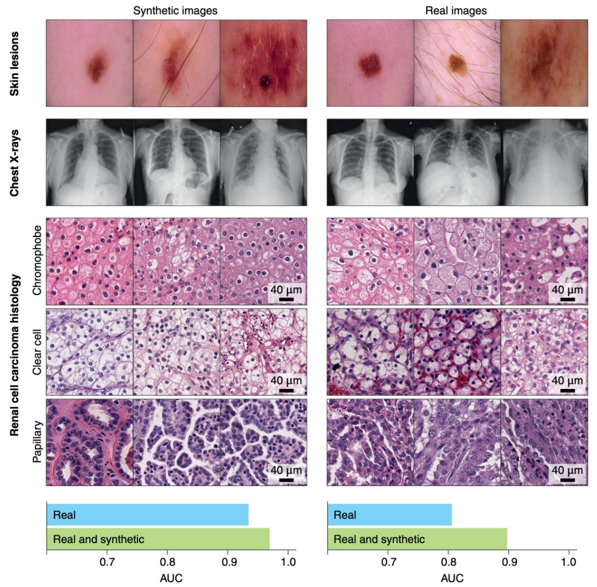
Machine learning will replace human radiologists, pathologists, maybe soon

As artificial intelligence, cognitive computing and machine learning systems become better than humans at medicine and cost less, it might even become unethical not to replace people.



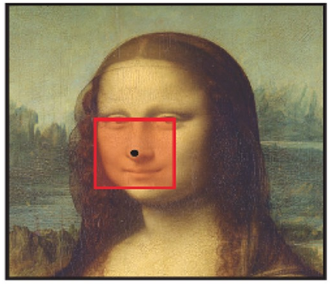
Super resolution



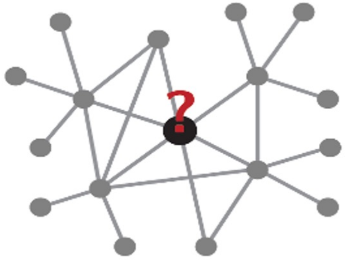


Geometric Deep Leaning

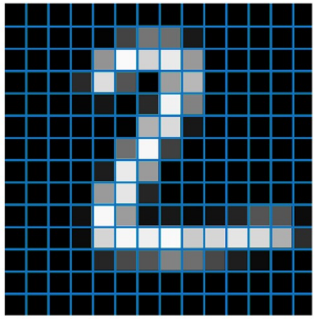
Learning on **irregular** domains



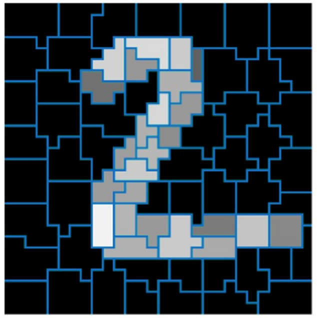
Image



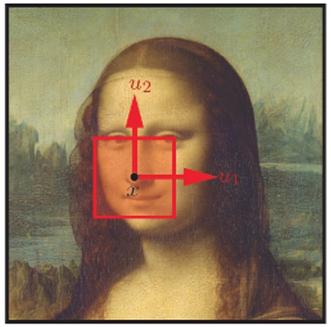
Graph



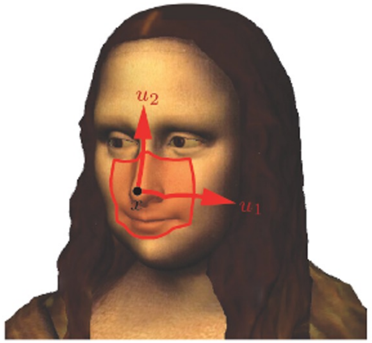
Regular grid



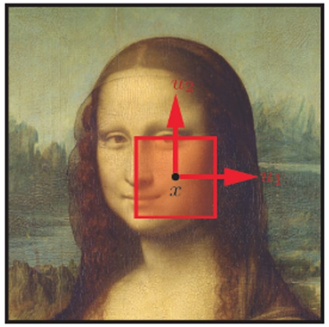
Superpixels



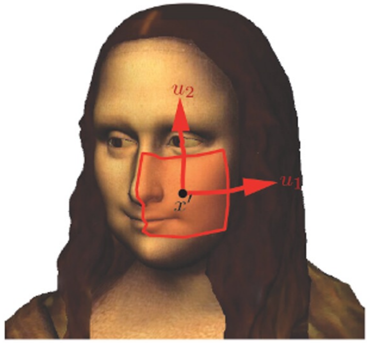
Euclidean



Non-Euclidean



Euclidean



Non-Euclidean

Credits to **Michael Bronstein**

Mathematical formulation

- Graphs: collections of *objects* (**nodes**) + *interactions* (**edges**) between them

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

- Formally, a graph is a tuple of nodes (V) and edges (E).
  - Edges typically operate over *pairs* of nodes, i.e. E ⊆ V x V.
  - Depending on context, nodes may be referred to as *vertices*, and edges as *links* or *relations*.

- We can represent edges x, **A** ∈ $\mathbb{R}^{|V| \text{ x } |V|}$, such that:

$$a_{ij} = \begin{cases} 1 & (i, j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

(…and many more, which we will not cover here)

Some interesting graph types

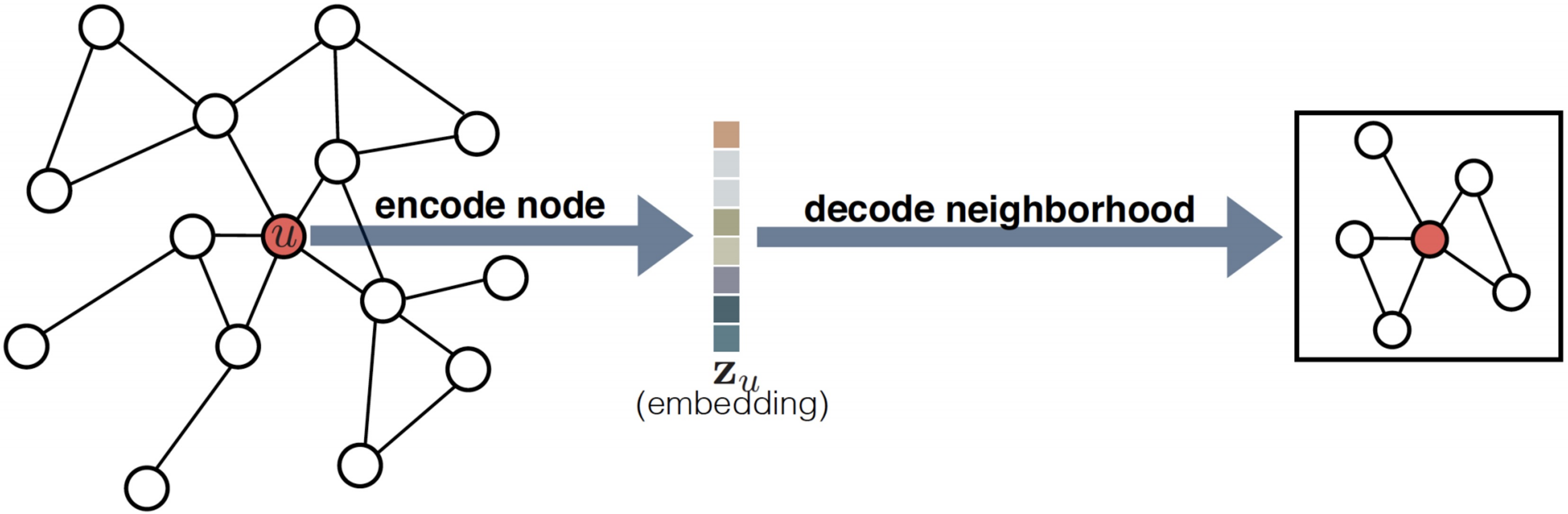
- **Undirected**: (u, v) ∈ E ⇒ (v, u) ∈ E (equivalently, $\mathbf{A}^T = \mathbf{A}$)
  - e.g. in a social network, *friendship* links are (usually?) *bidirectional*

- **Weighted**: provided **edge weight**, w_{ij} for every edge (i, j) ∈ E
  - e.g. in a road network, weights may specify *distances* or *speeds*

- **Multirelational**: various **edge types**; (u, t, v) ∈ E if (u, v) linked by type t
  - e.g. in a knowledge graph, types encode different *relations* (“is-parent”, “is-spouse”, …)

- **Heterogeneous**: various **node types**
  - e.g. in a biomedical interaction graph, nodes may be *drugs*, *proteins* or *diseases*

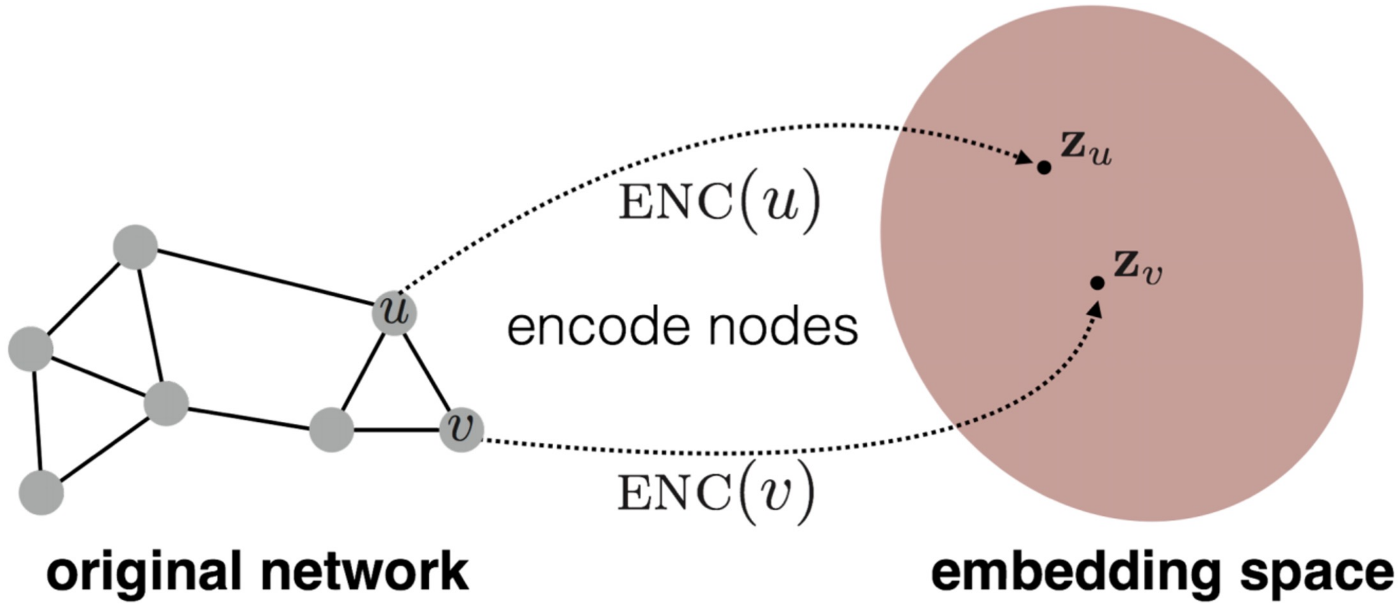
Encoder-decoder Setup



Credits to **Will Hamilton**

Node Embeddings

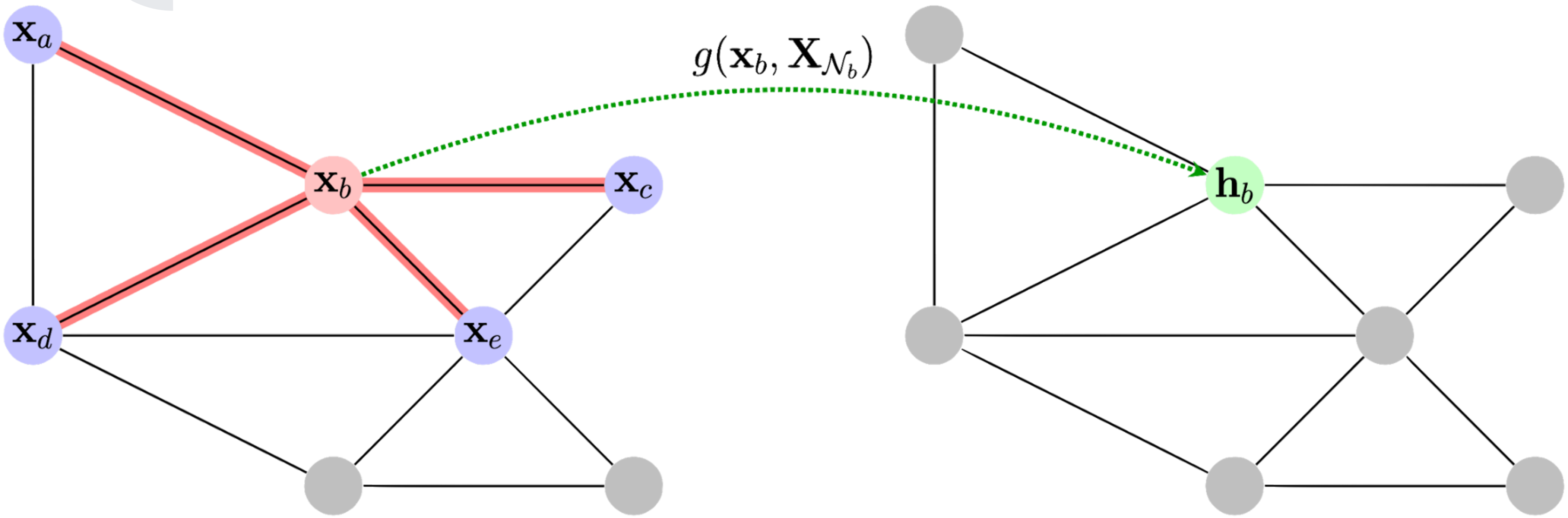
Discover good ways to **embed** nodes into vectors $\mathbf{z}_u$ using an *encoder* function



Deep learning on graphs *before* GNNs! (use the graph structure **implicitly**)

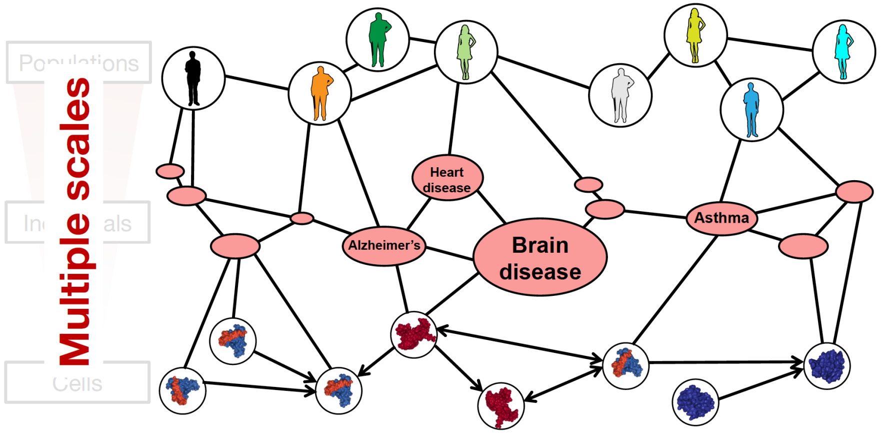
Credits to **Will Hamilton**

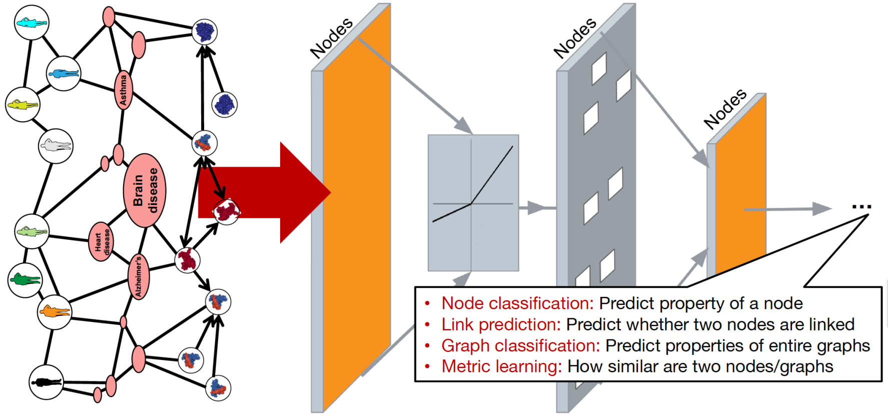
A recipe for **graph** neural networks, visualised



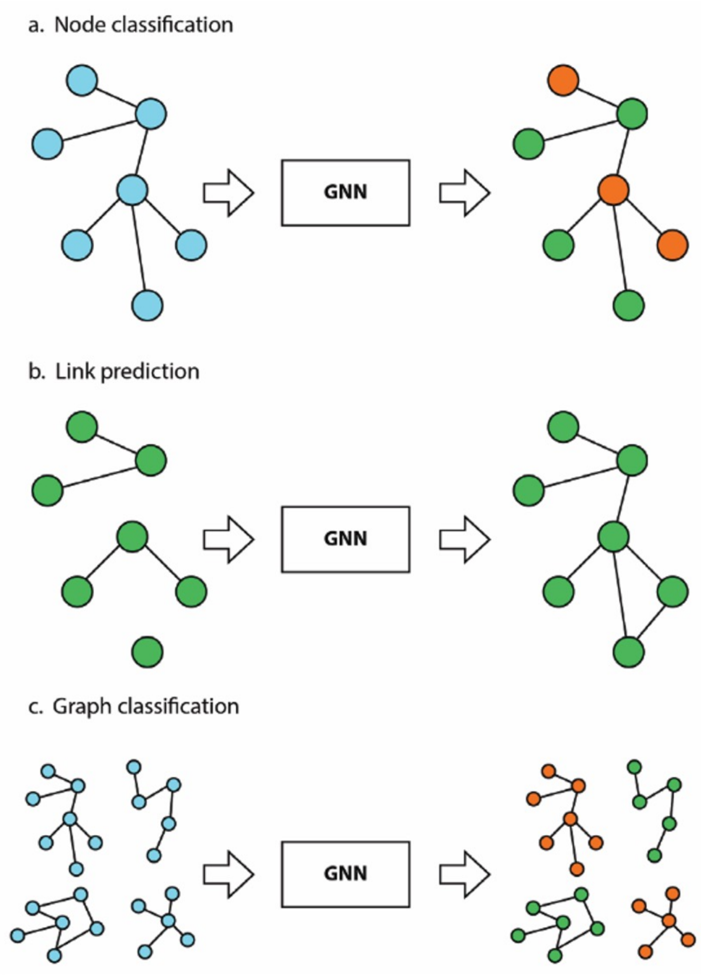
$$\mathbf{X}_{\mathcal{N}_b} = \{\!\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c, \mathbf{x}_d, \mathbf{x}_e\}\!\}$$

Graph Neural Networks





- Node classification: Predict property of a node
- Link prediction: Predict whether two nodes are linked
- Graph classification: Predict properties of entire graphs
- Metric learning: How similar are two nodes/graphs

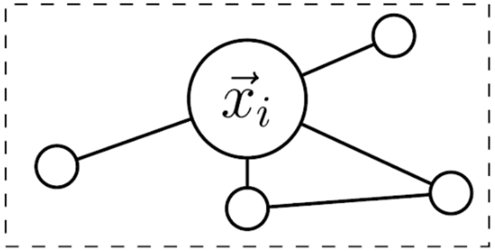


a. Node classification

b. Link prediction

c. Graph classification

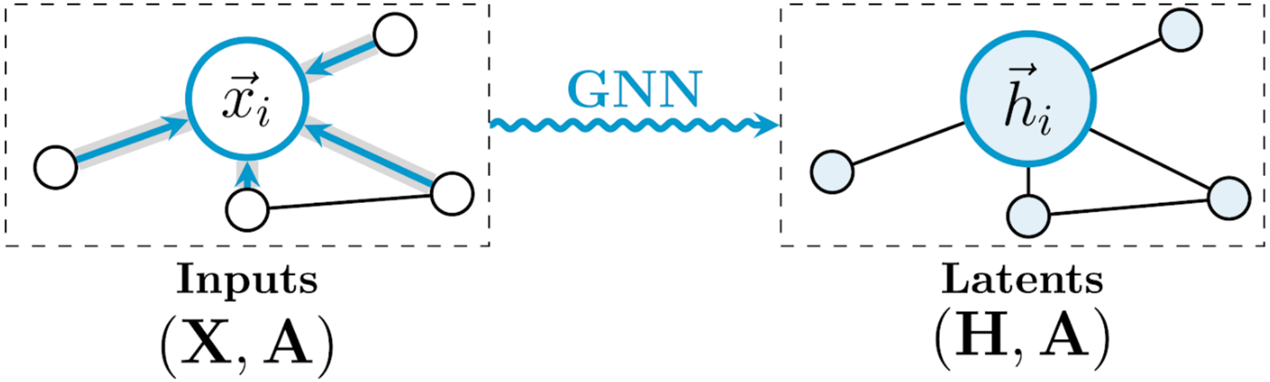
How to use GNNs?



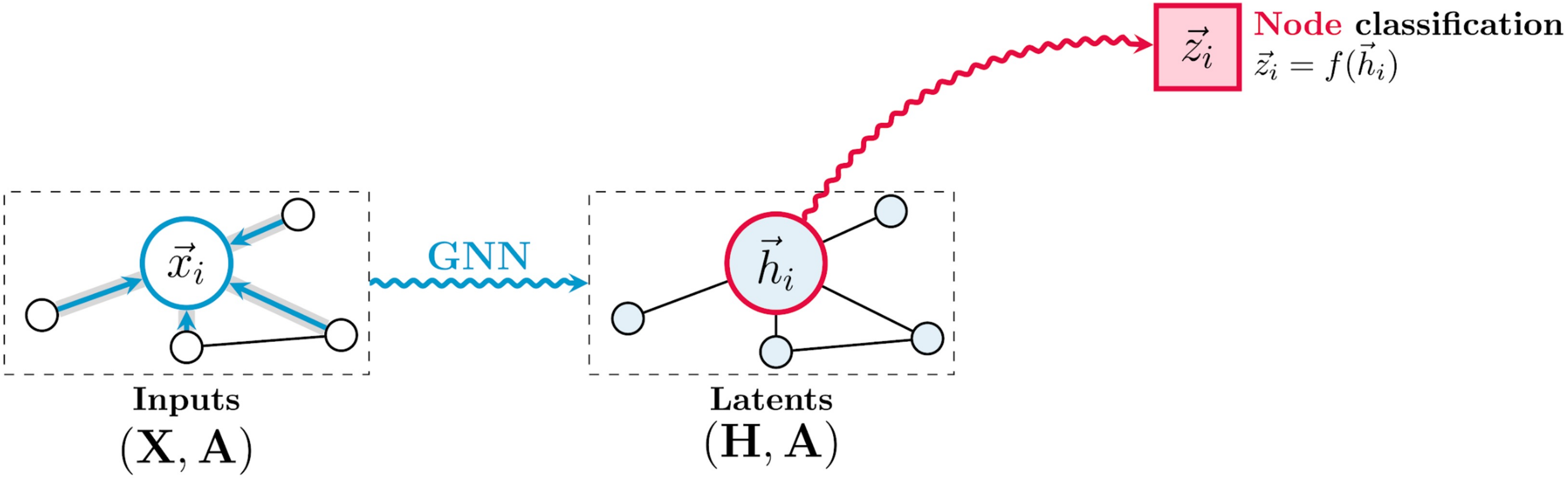
Inputs

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

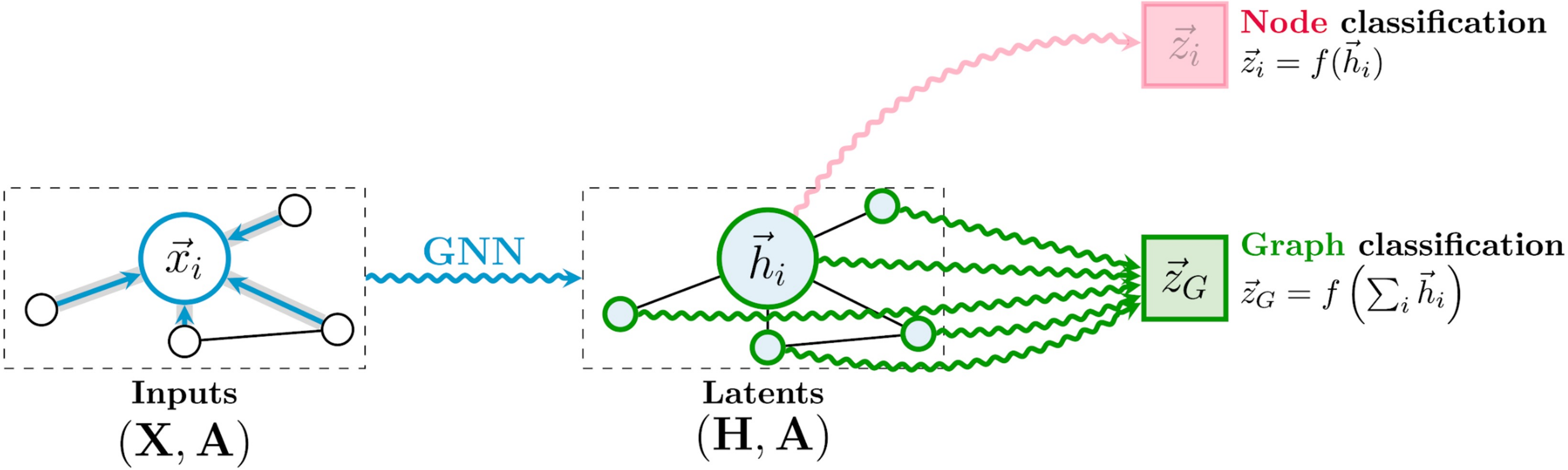
How to use GNNs?



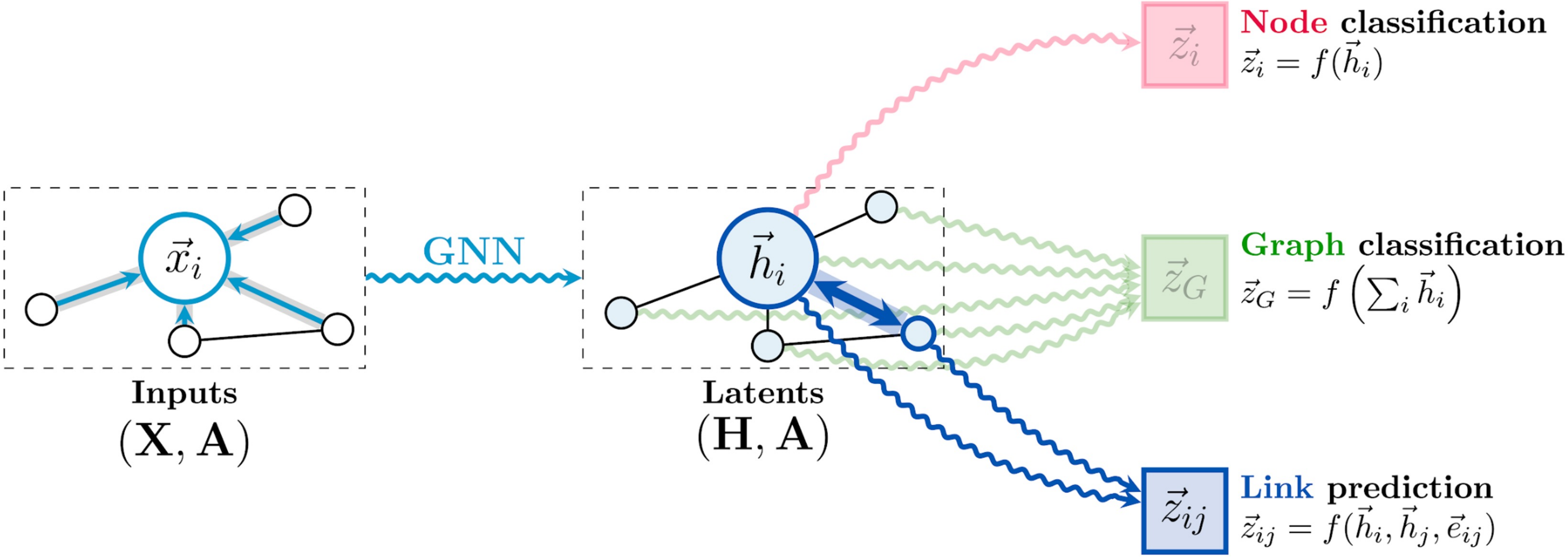
How to use GNNs?


$\vec{x}_i$
GNN
$\vec{h}_i$
$\vec{z}_i$
Node classification
$\vec{z}_i = f(\vec{h}_i)$
Inputs
$(\mathbf{X}, \mathbf{A})$
Latents
$(\mathbf{H}, \mathbf{A})$

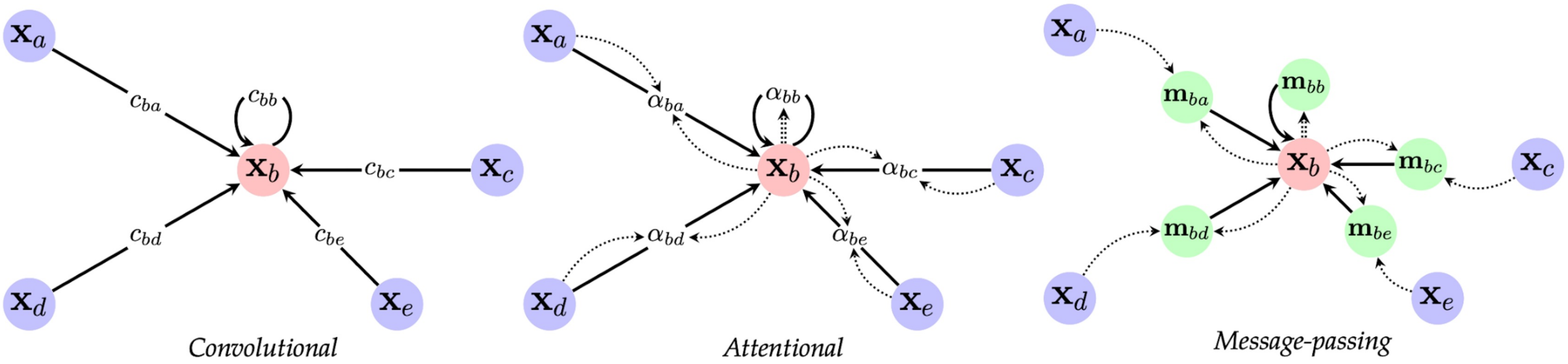
How to use GNNs?


$\vec{x}_i$
GNN
$\vec{h}_i$
$\vec{z}_i$
Node classification
$\vec{z}_i = f(\vec{h}_i)$
$\vec{z}_G$
Graph classification
$\vec{z}_G = f\left(\sum_i \vec{h}_i\right)$
Inputs
$(\mathbf{X}, \mathbf{A})$
Latents
$(\mathbf{H}, \mathbf{A})$

How to use GNNs?


$\vec{x}_i$
GNN
$\vec{h}_i$
Inputs
$(\mathbf{X}, \mathbf{A})$
Latents
$(\mathbf{H}, \mathbf{A})$
$\vec{z}_i$
Node classification
$\vec{z}_i = f(\vec{h}_i)$
$\vec{z}_G$
Graph classification
$\vec{z}_G = f\left(\sum_i \vec{h}_i\right)$
$\vec{z}_{ij}$
Link prediction
$\vec{z}_{ij} = f(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$

The three “flavours” of GNN layers



Convolutional

Attentional

Message-passing

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

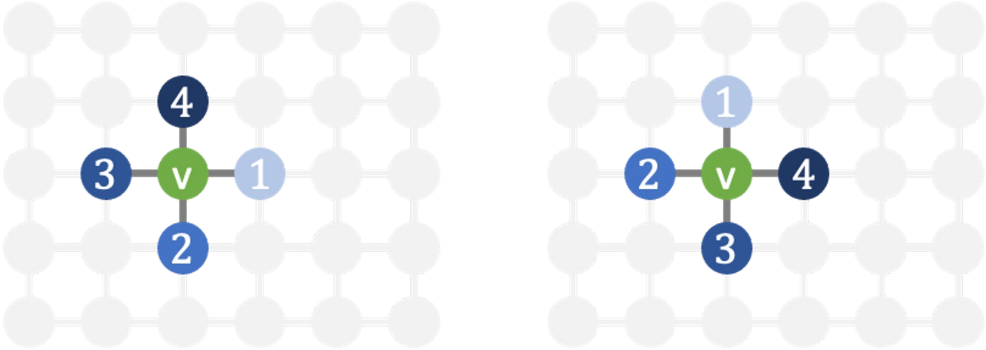
$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j\in\mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$

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

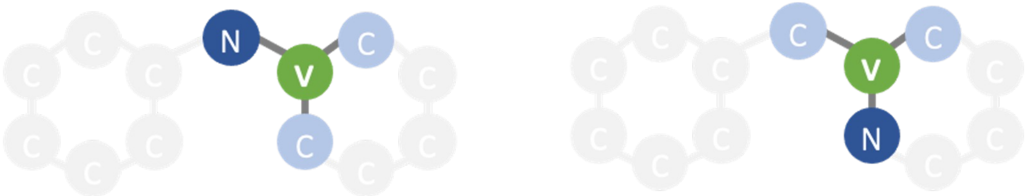
Credits to **Petar velickovic**

Directional GNN

Nodes in GNNs do not know where Neighbours Are coming from: the aggregation is symmetric



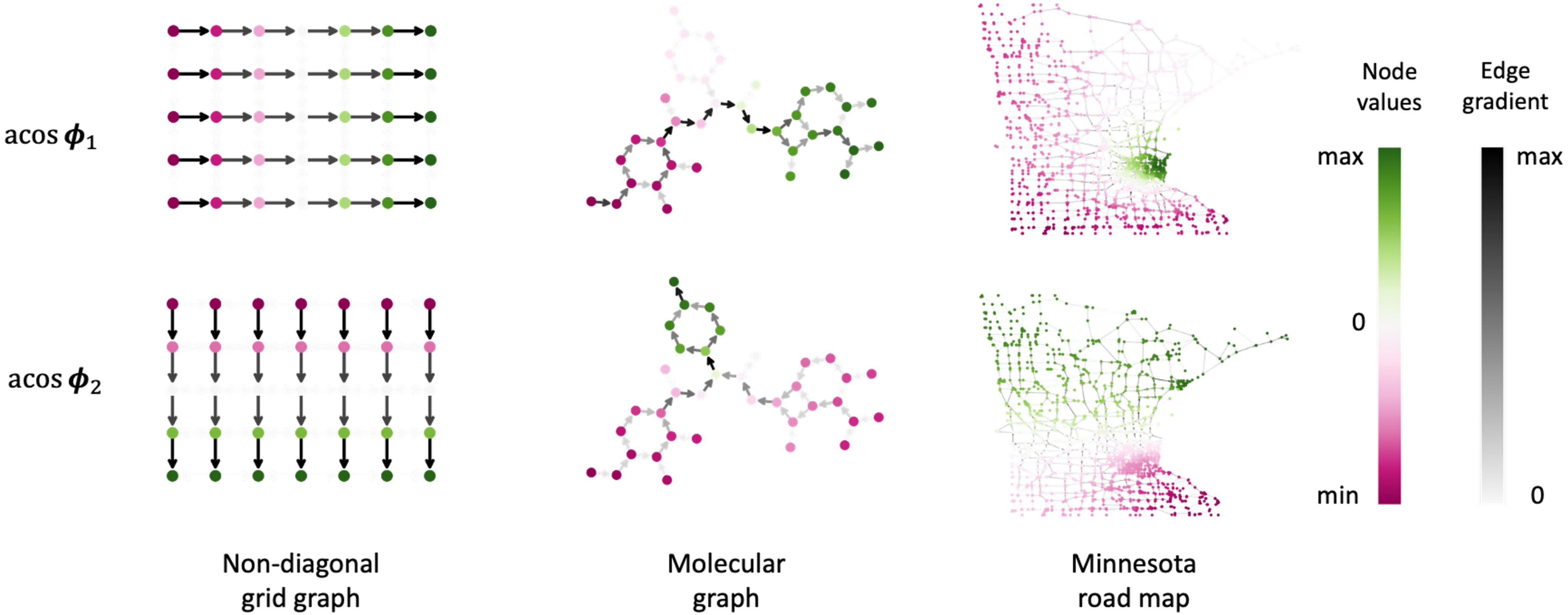
Grid graph



Molecular graph

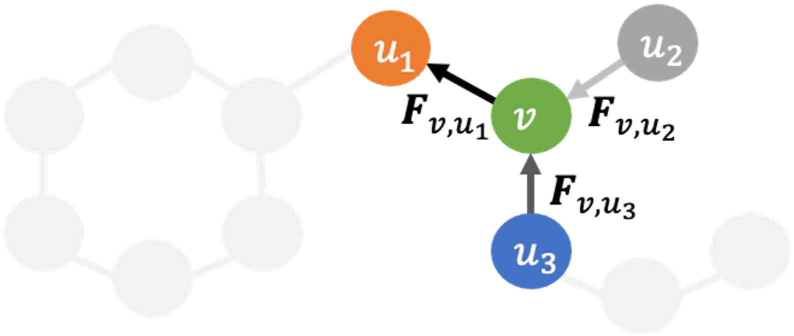
Credits to **Beaini,Passaro,Corso,Hamilton**

Directional GNN:Eigenvectors Can Give Structural Information



Credits to **Beaini,Passaro,Corso,Hamilton,Lio'**

Directional Aggregators



v	Features of the node receiving the message
$u_{1,2,3}$	Features of the neighbouring nodes
$\boldsymbol{F}_{v,u}$	Directional vector field between the node v and u

Directional smoothing aggregation $\boldsymbol{B}_{\boldsymbol{av}}(\boldsymbol{F})\boldsymbol{x}$

$$\frac{|\boldsymbol{F}_{v,u_1}|u_1 + |\boldsymbol{F}_{v,u_2}|u_2 + |\boldsymbol{F}_{v,u_3}|u_3}{|\boldsymbol{F}_{v,u_1}| + |\boldsymbol{F}_{v,u_2}| + |\boldsymbol{F}_{v,u_3}|}$$

$$\frac{\text{Absolute weighted sum}}{\text{Sum of the absolute weights}}$$

Directional derivative aggregation $\boldsymbol{B}_{\boldsymbol{dx}}(\boldsymbol{F})\boldsymbol{x}$

$$\frac{\boldsymbol{F}_{v,u_1}(u_1 - v) + \boldsymbol{F}_{v,u_2}(v - u_2) + \boldsymbol{F}_{v,u_3}(v - u_3)}{|\boldsymbol{F}_{v,u_1}| + |\boldsymbol{F}_{v,u_2}| + |\boldsymbol{F}_{v,u_3}|}$$

$$\frac{\text{Weighted } \textit{forward} \text{ derivative with } u_1 + \text{Weighted } \textit{backward} \text{ derivative with } u_2 + \text{Weighted } \textit{backward} \text{ derivative with } u_3}{\text{Sum of the absolute weights}}$$

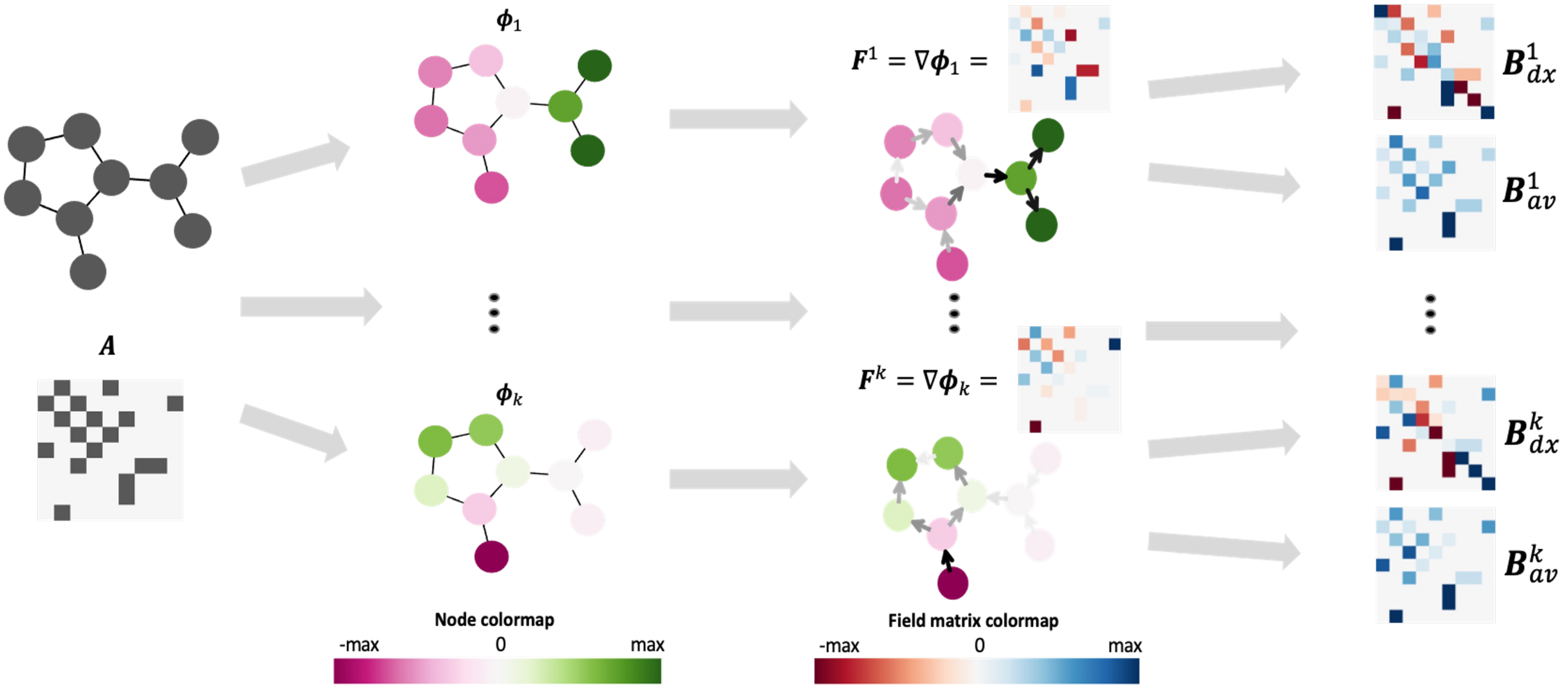
Directional Graph Networks. D. Beaini, S. Passaro*, V. Létourneau, W. Hamilton, G. Corso, P. Liò*

Directional GNN

Input graph

Compute first k non-trivial eigenvectors

Compute the gradient



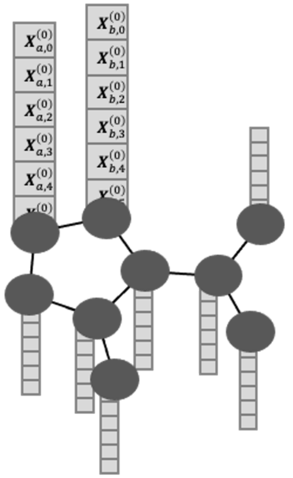
Directional Graph Neural Networks

Input graph

Aggregation function

Update function

Input graph



Aggregation matrices from preprocessing

$$\left\{\begin{matrix} \boldsymbol{B}^1_{dx} \\ \boldsymbol{B}^1_{av} \\ \vdots \\ \boldsymbol{B}^k_{dx} \\ \boldsymbol{B}^k_{av} \end{matrix}\right\}$$

$$\boldsymbol{Y}^{(0)} = \text{concat}\left\{\begin{matrix} \boldsymbol{D}^{-1}\boldsymbol{A}\,\boldsymbol{X}^{(0)} \\ \left|\boldsymbol{B}^1_{dx}\,\boldsymbol{X}^{(0)}\right| \\ \boldsymbol{B}^1_{av}\,\boldsymbol{X}^{(0)} \\ \vdots \\ \left|\boldsymbol{B}^k_{dx}\,\boldsymbol{X}^{(0)}\right| \\ \boldsymbol{B}^k_{av}\,\boldsymbol{X}^{(0)} \end{matrix}\right\}$$

$$\boldsymbol{X}^{(1)} = \text{MLP}\left(\boldsymbol{Y}^{(0)}\right)$$

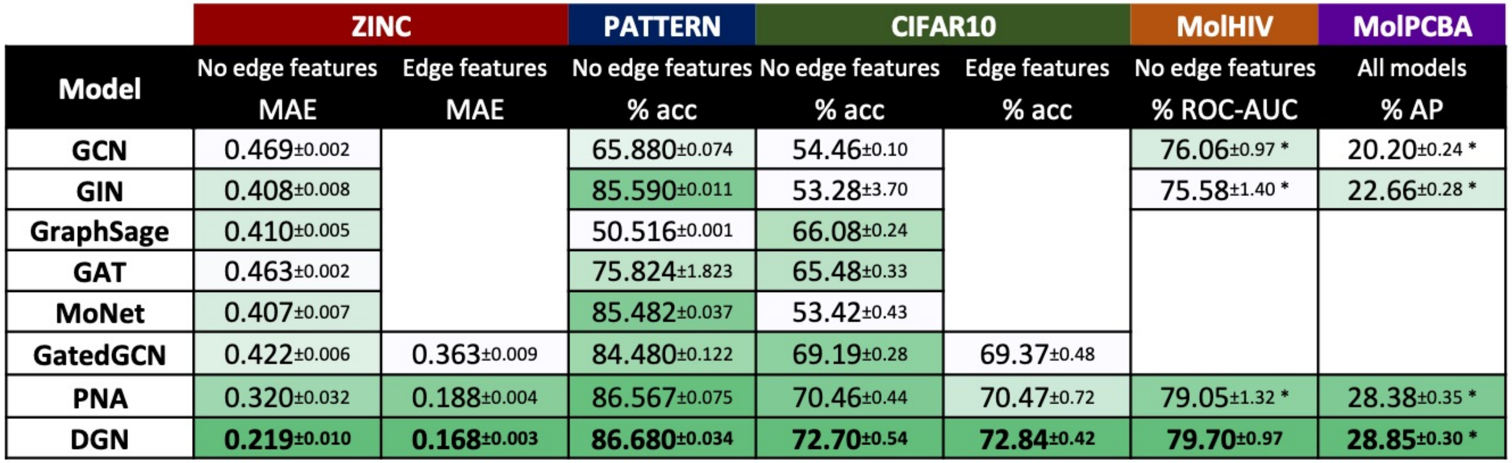
Next GNN layer

$$t \rightarrow t+1$$

$$\boldsymbol{X}^{(t)} \rightarrow \boldsymbol{X}^{(t+1)}$$

Directional Graph Neural

- Directional graph neural networks can alleviate over-smoothing and over-squashing.
- In particular, the Laplacian eigenfunctions reveal directions that can counteract over-smoothing and over-squashing by allowing efficient propagation of information between distant nodes instead of following a diffusion process.



	ZINC		PATTERN	CIFAR10		MolHIV	MolPCBA
Model	No edge features	Edge features	No edge features	No edge features	Edge features	No edge features	All models
	MAE	MAE	% acc	% acc	% acc	% ROC-AUC	% AP
GCN	0.469±0.002		65.880±0.074	54.46±0.10		76.06±0.97 *	20.20±0.24 *
GIN	0.408±0.008		85.590±0.011	53.28±3.70		75.58±1.40 *	22.66±0.28 *
GraphSage	0.410±0.005		50.516±0.001	66.08±0.24			
GAT	0.463±0.002		75.824±1.823	65.48±0.33			
MoNet	0.407±0.007		85.482±0.037	53.42±0.43			
GatedGCN	0.422±0.006	0.363±0.009	84.480±0.122	69.19±0.28	69.37±0.48		
PNA	0.320±0.032	0.188±0.004	86.567±0.075	70.46±0.44	70.47±0.72	79.05±1.32 *	28.38±0.35 *
DGN	**0.219±0.010**	**0.168±0.003**	**86.680±0.034**	**72.70±0.54**	**72.84±0.42**	**79.70±0.97**	**28.85±0.30 ***

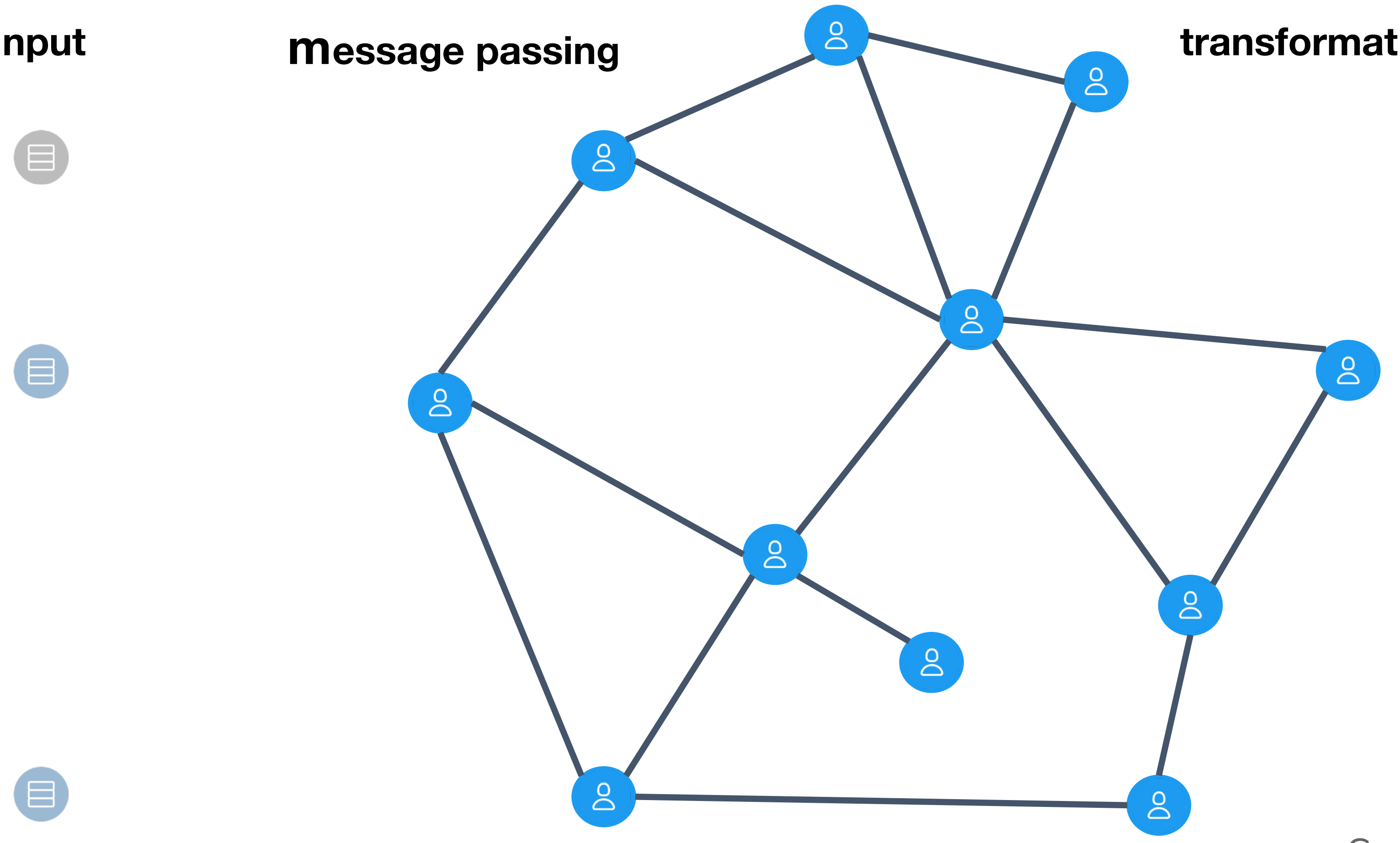
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



Credits to **Francesco di Giovanni**

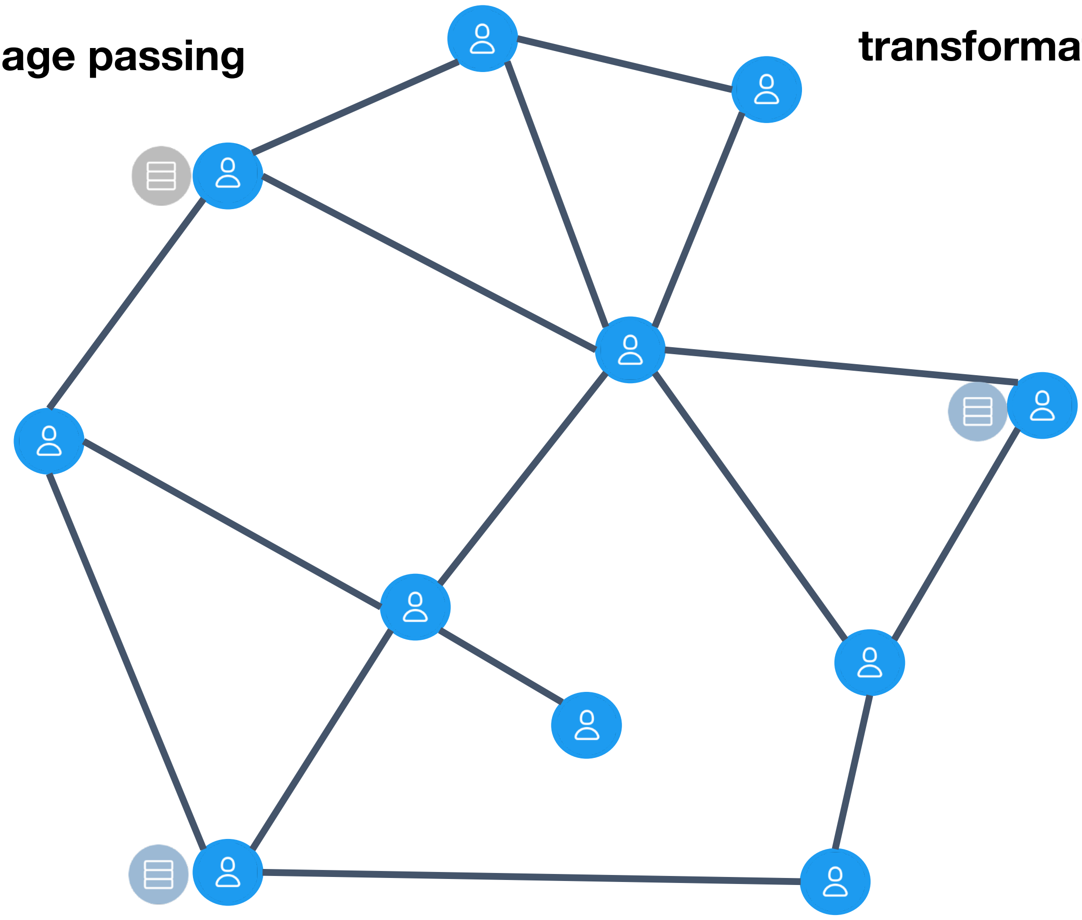
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



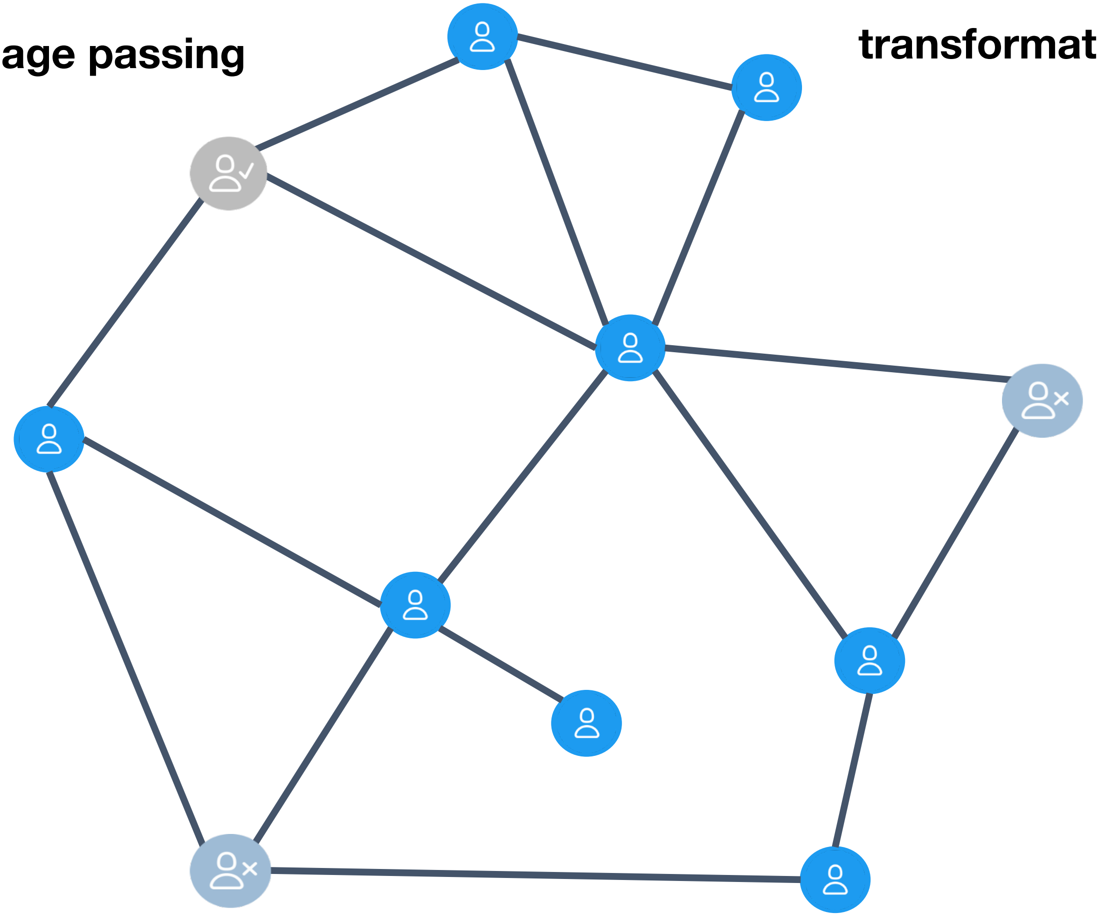
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



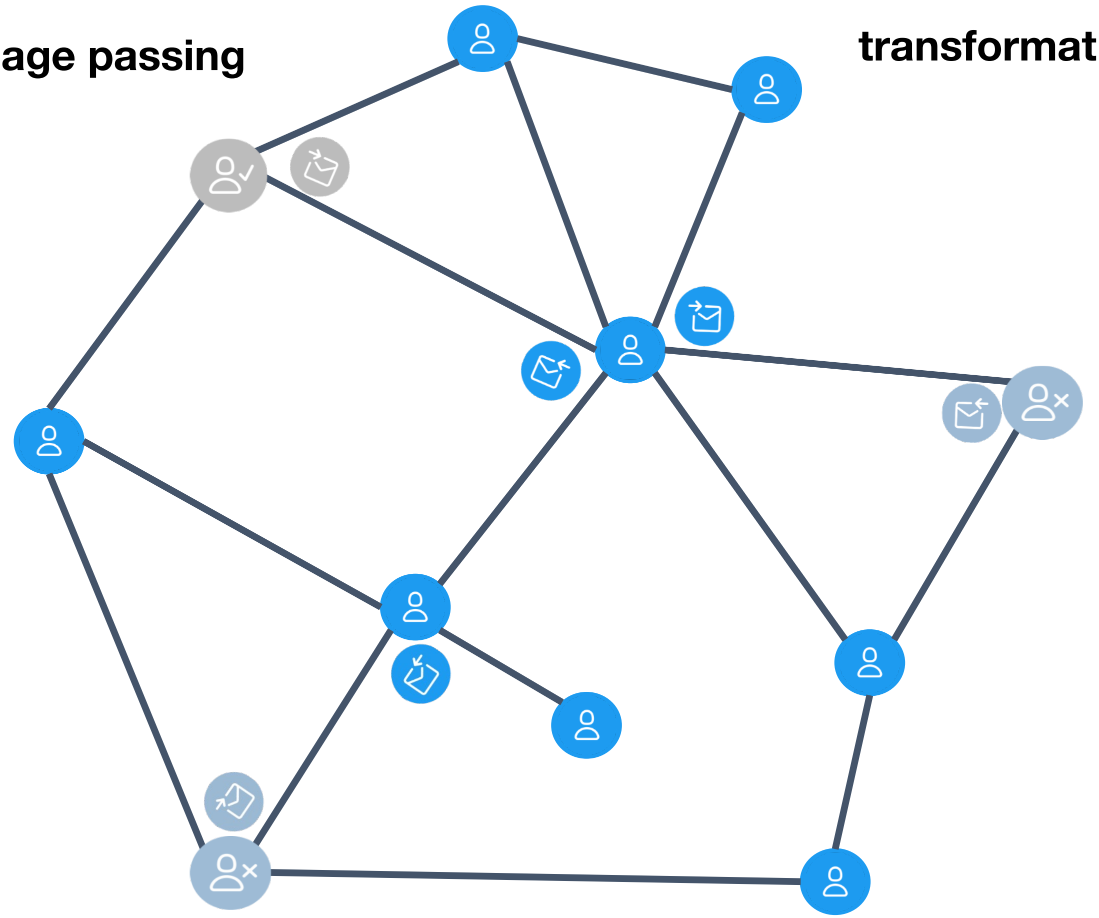
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



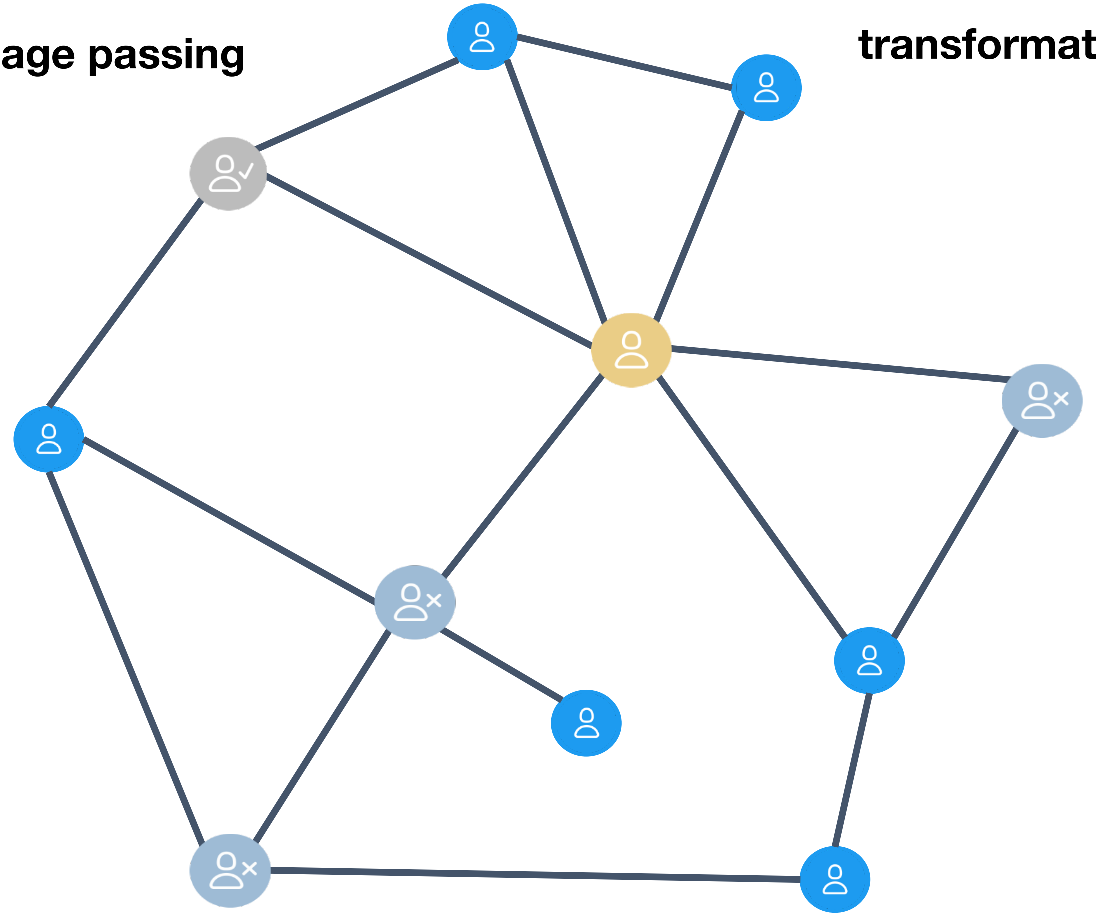
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



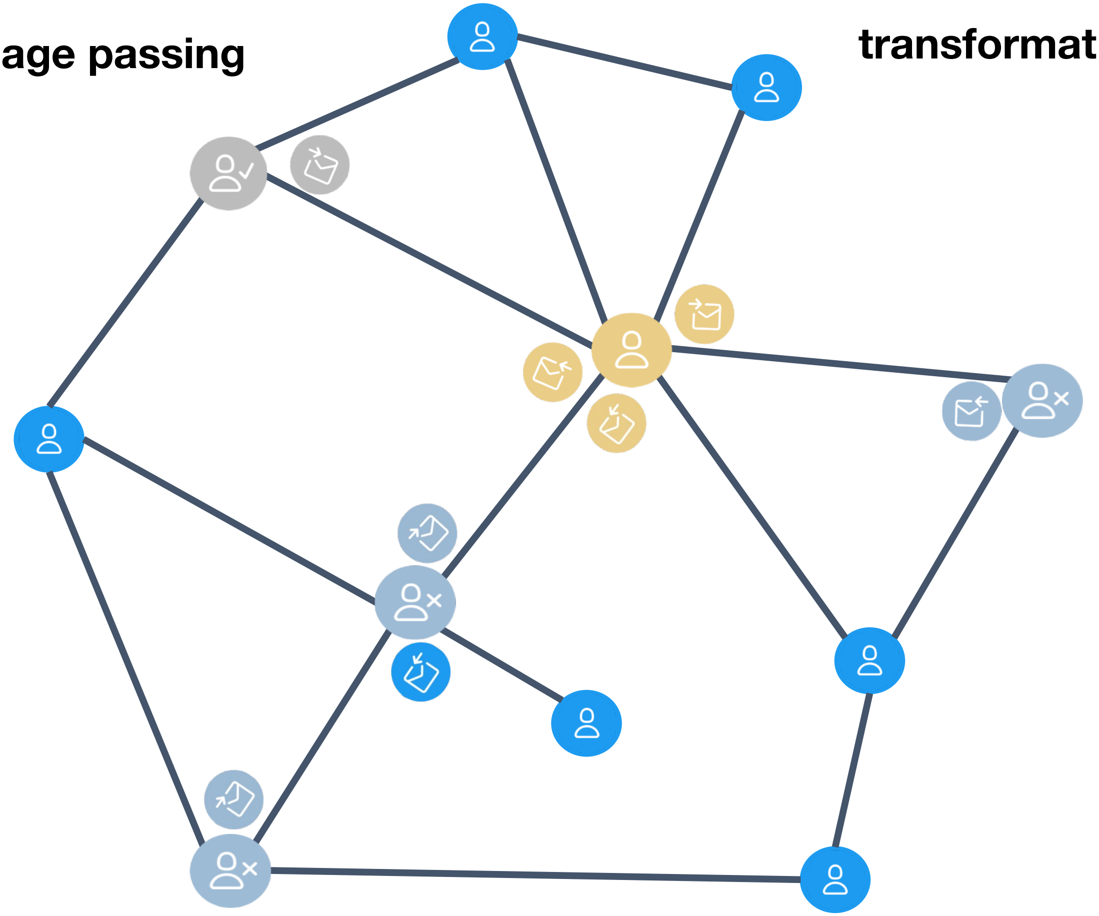
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



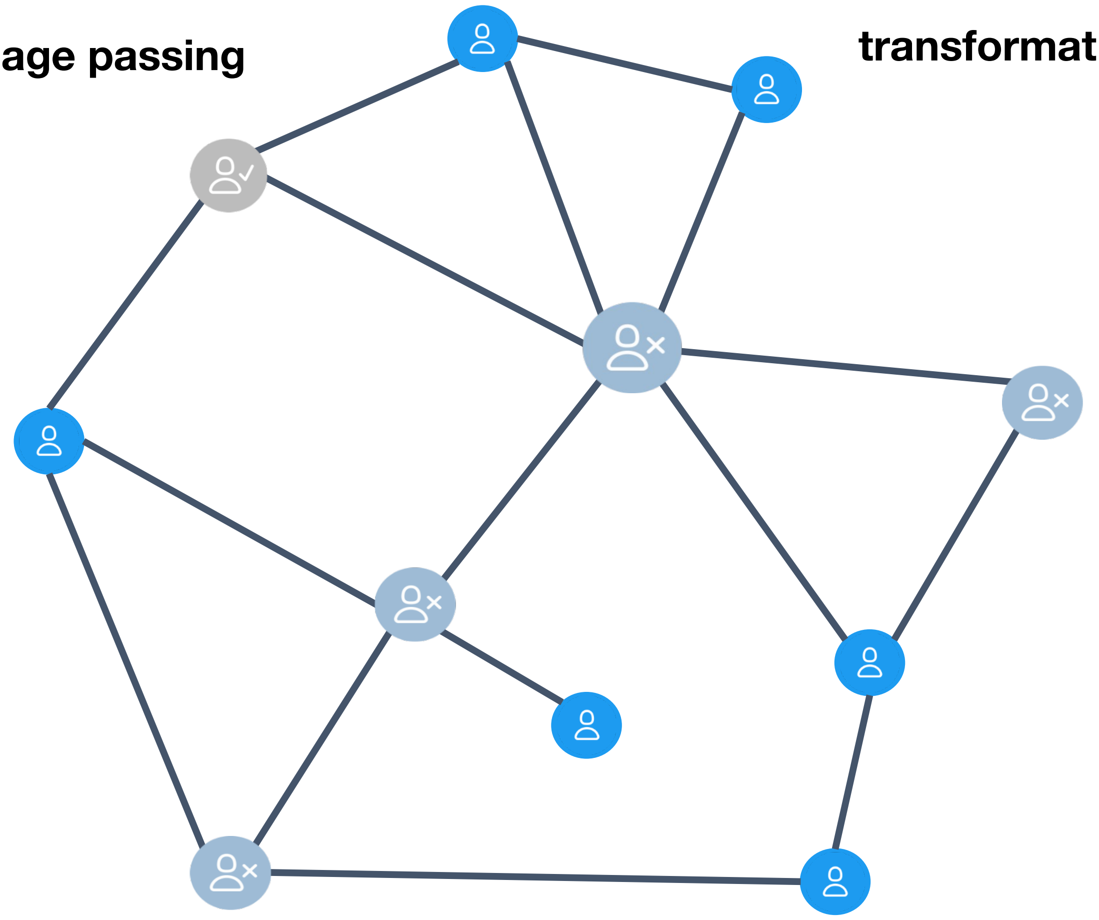
Message passing neural network as diffusion of information on a graph

input

message passing

transformation

output



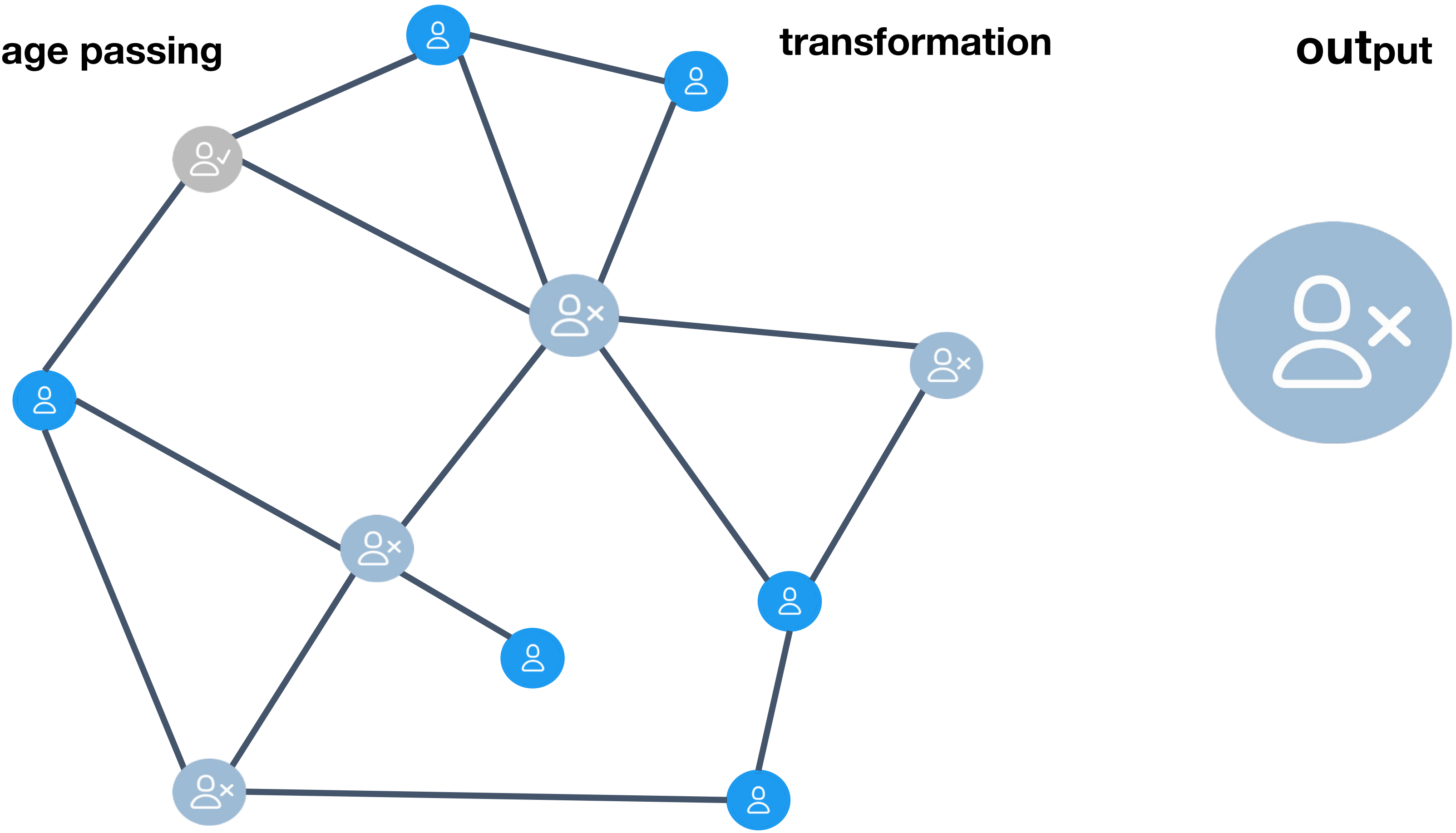
Message passing neural network as diffusion of information on a graph

input

message passing

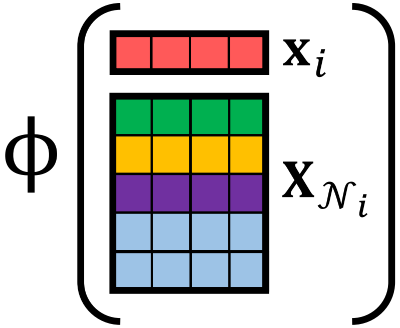
transformation

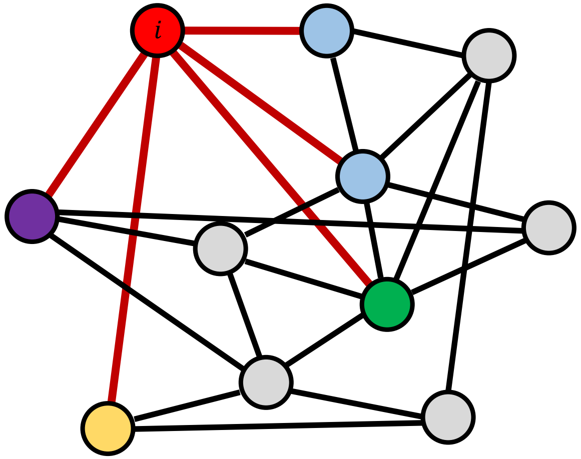
output



Graph Neural Networks: Message Passing "flavour"

local function





learnable message function

$$f(\mathbf{x}_i) = \phi\Big(\mathbf{x}_i, \underset{j \in \mathcal{N}_i}{\square}\, \psi(\mathbf{x}_i, \mathbf{x}_j)\Big)$$

Gilmer et al. 2017; Battaglia et al 2018; Wang et B. 2018

Spatial discretisation

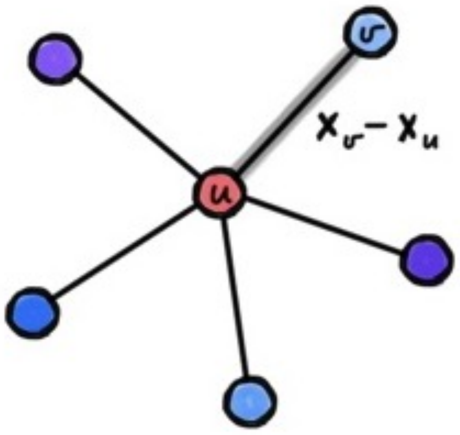
$$\frac{\partial \boldsymbol{x}(u,t)}{\partial t} = \mathrm{div}[\nabla \boldsymbol{x}(u,t)] \qquad (1)$$

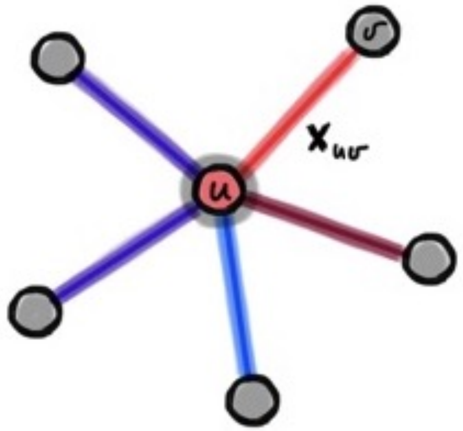
gradient - flow along edges

$$(\nabla \boldsymbol{X})_{uv} = \boldsymbol{x}_u - \boldsymbol{x}_v$$

divergence - aggregation of edges

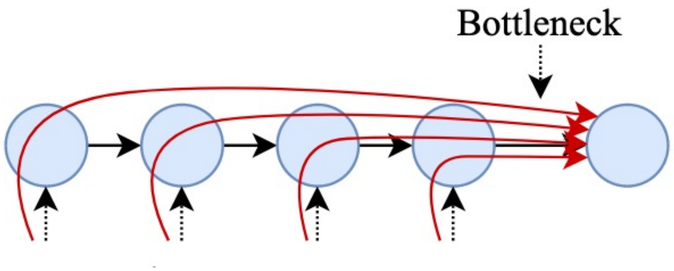
$$\big(\mathrm{div}(\boldsymbol{X})\big)_u = \sum\nolimits_v w_{uv} x_{uv}$$

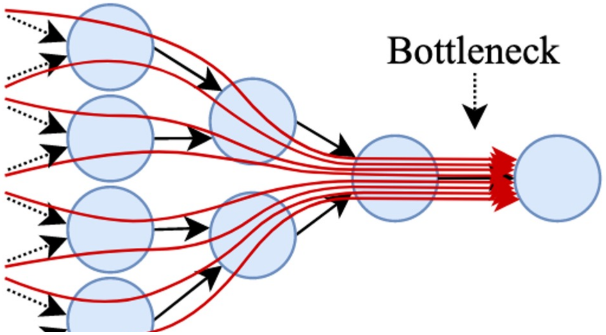




$$\dot{\boldsymbol{X}}(t) = \big(\boldsymbol{A}\big(\boldsymbol{X}(t)\big) - \boldsymbol{I}\big)\boldsymbol{X}(t) \quad (2)$$

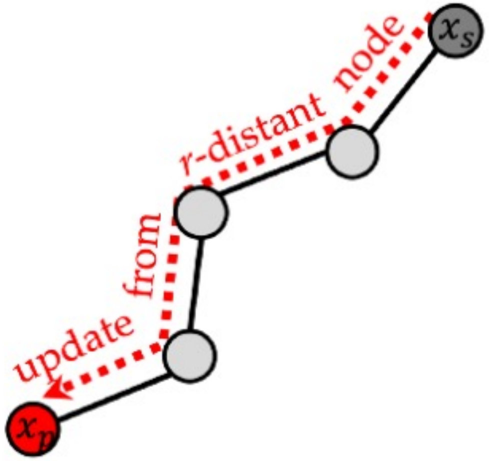
Over-squashing and over-smoothing

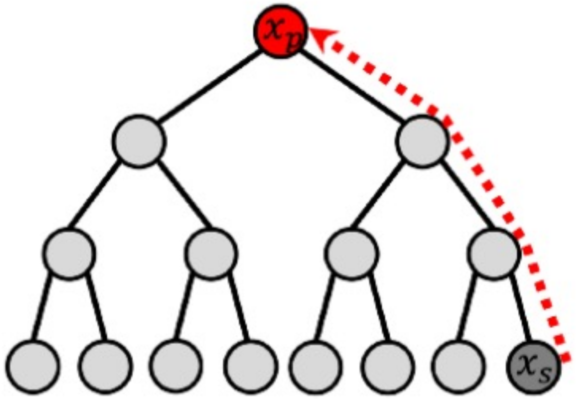




Oversquashing: inability for GNNs to propagate informative signals between distant nodes and is a major bottleneck to training deep GNN models

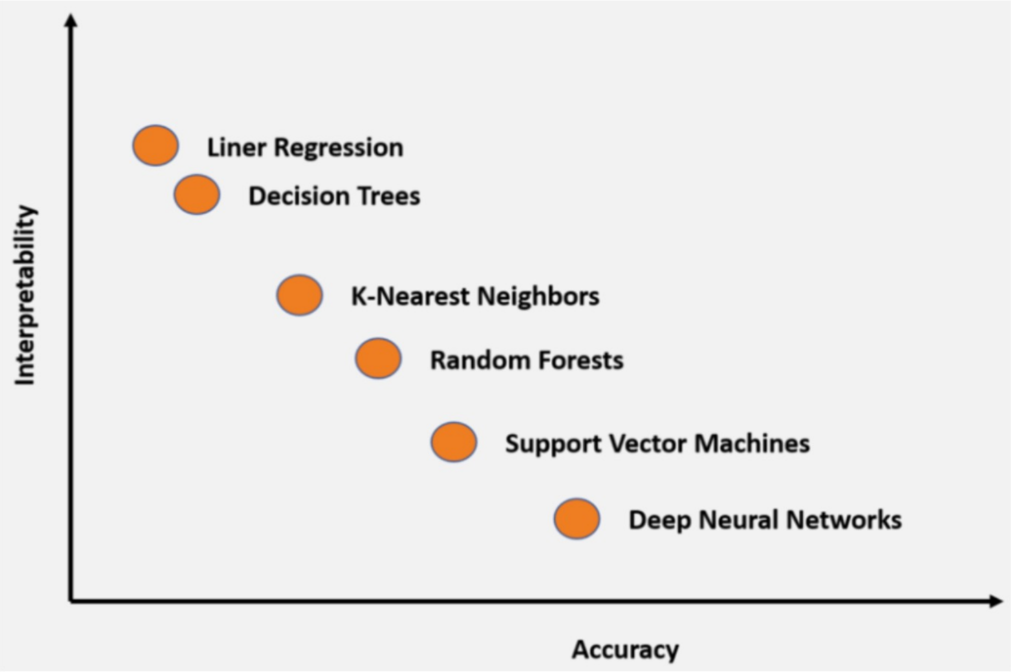
Over-smoothing: node representations become indistinguishable and prediction performance severely degrades when the number of layers increase

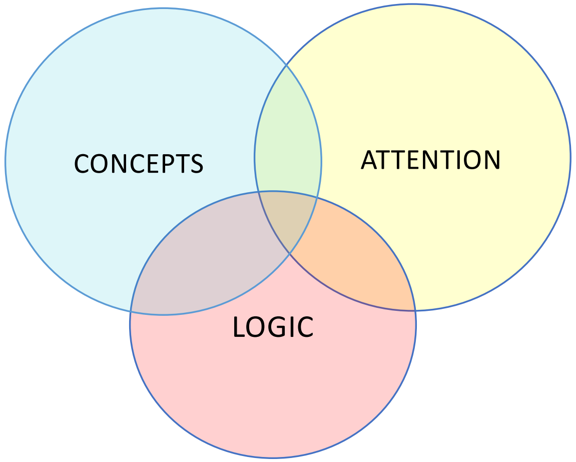




Uri Alon and Eran Yahav (2020) On the Bottleneck of Graph Neural Networks and its Practical Implications. J. Topping, F. Di Giovanni et al., Understanding over-squashing and bottlenecks on graphs via curvature (2021) arXiv:2111.14522

Interpretability



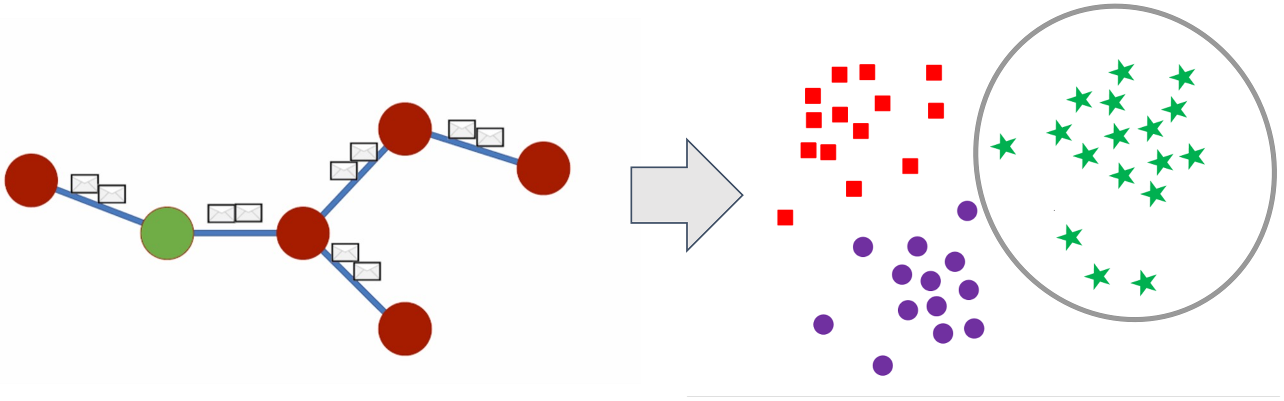


One model is more interpretable than another if it is easier for a human to understand how it makes predictions than the other model.

Ciravegna, Gabriele, et al. "Human-Driven FOL Explanations of Deep Learning." IJCAI. 2020.

GCExplainer

- **Graph Concept Explainer** (GCExplainer)
- Key Ideas:
  - Concept-based explanations
  - Global explanations
  - Quantitative evaluation



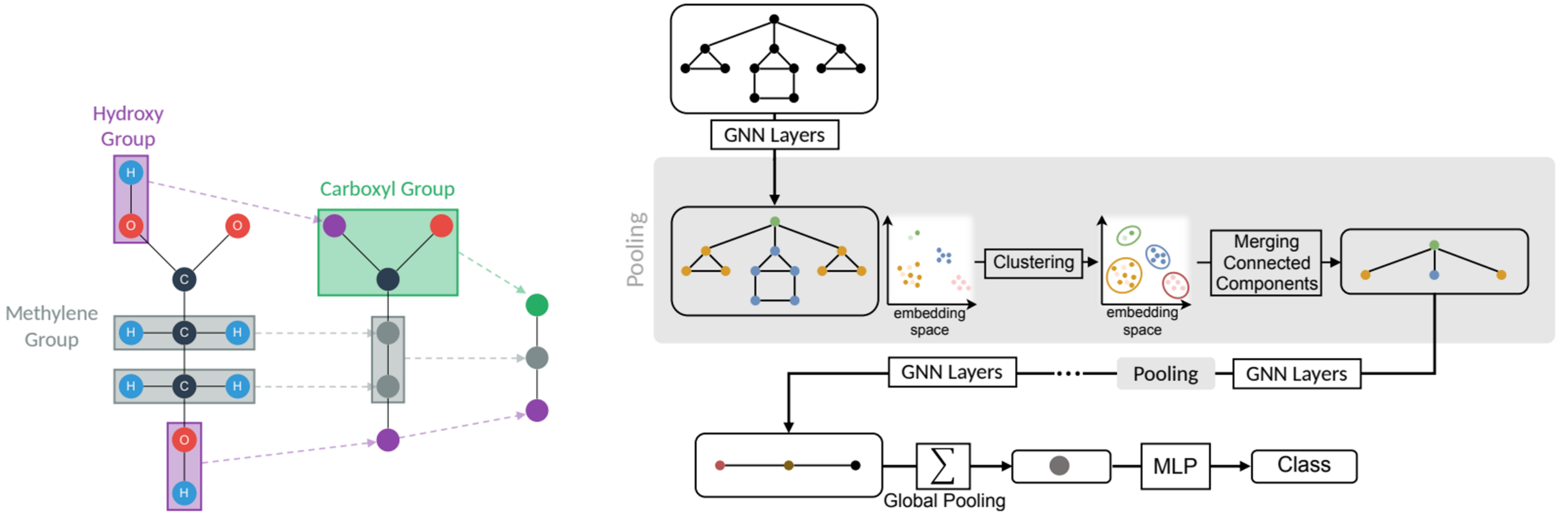
Graph Neural Network reasons on relational data

Graph Concept Explainer clusters the GNN’s embeddings

Magister, Lucie Charlotte, et al (PL). "Gcexplainer: Human-in-the-loop concept-based explanations for graph neural networks." arXiv preprint arXiv:2107.11889 (2021).

Hierarchical Explainable Latent Pooling (HELP)

- Clusters node embeddings and merges connected components clustered together
- Discovered a hierarchy of Concepts



(a) (b)

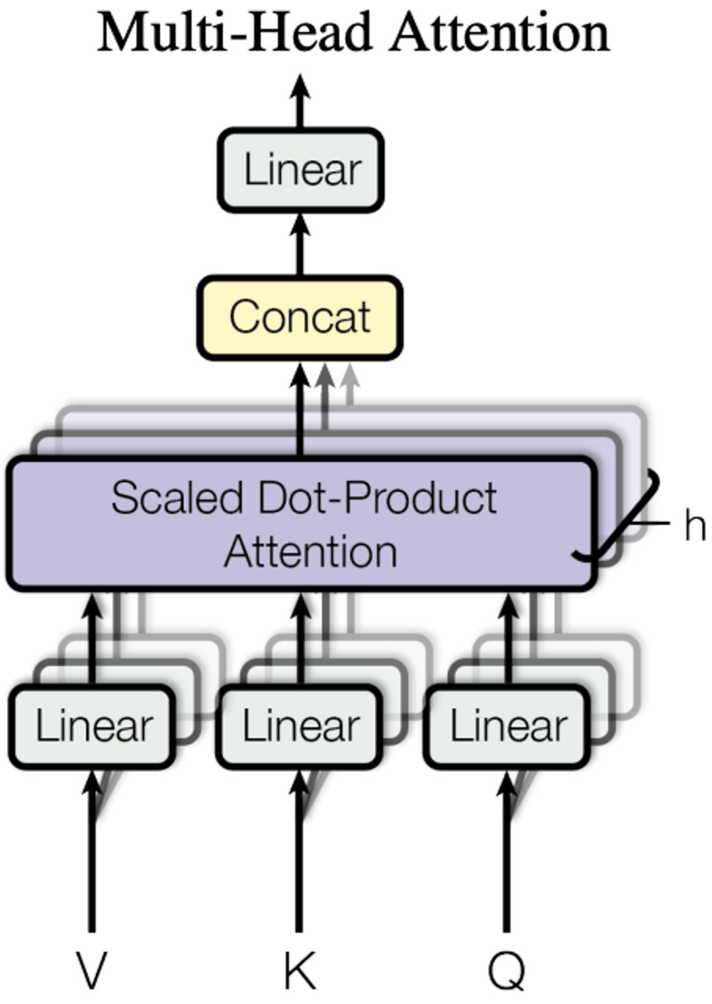
Jürß, Jonas, et al (PL). "Everybody Needs a Little HELP: Explaining Graphs via Hierarchical Concepts." arXiv preprint arXiv:2311.15112 (2023).

A note on Transformers

- Transformers **are** Graph Neural Networks!
  - Fully-connected graph
  - Attentional flavour

- The sequential information is injected through the **positional embeddings**. Dropping them yields a fully-connected GAT model.

- Attention can be seen as inferring a “soft” adjacency matrix.



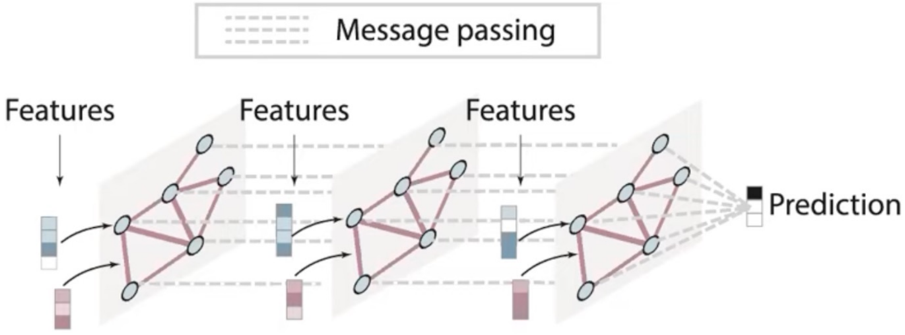
See Joshi (The Gradient; 2020).

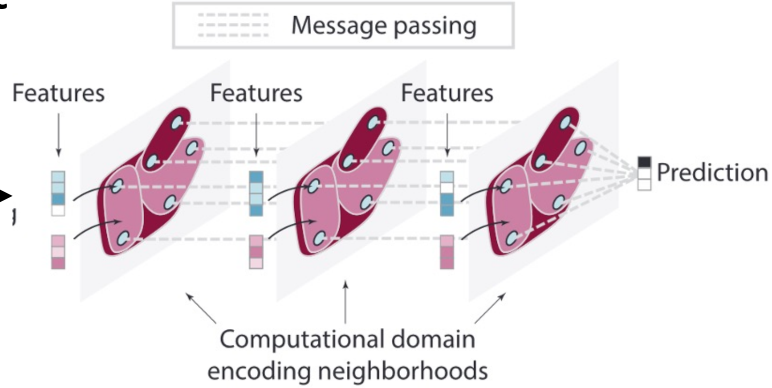
Topological Neural Networks

- Topological Neural Networks (TNNs) are deep learning architectures that extract knowledge from data associated with topologically rich systems such as protein structures, city traffic maps, or citation networks.

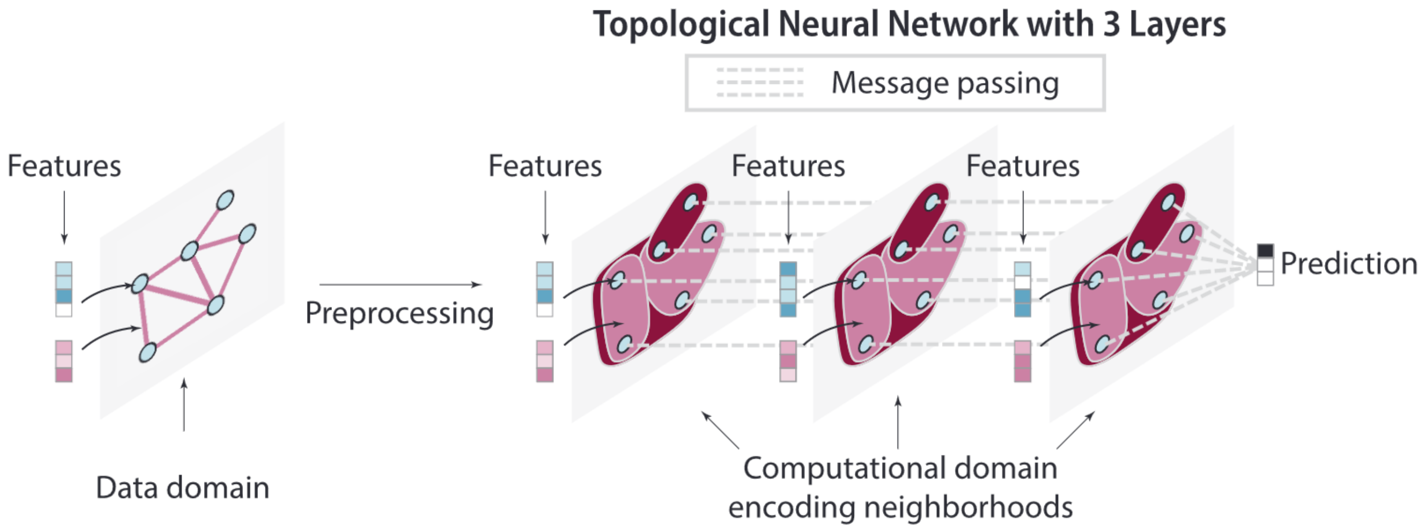
- A TNN, like a GNN, is comprised of stacked layers that transform data into a series of features. Each layer leverages the fundamental

tational domains,





Topological Neural Networks


Topological Neural Network with 3 Layers
Message passing
Features
Features
Features
Features
Preprocessing
Prediction
Data domain
Computational domain encoding neighborhoods

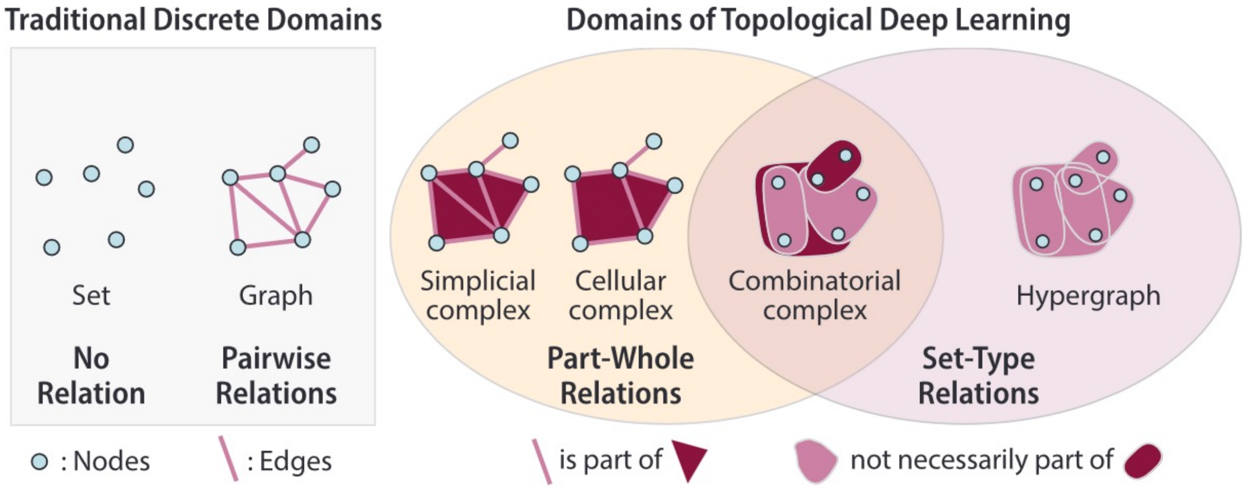
Data associated with a complex system are features defined on a data domain, which is preprocessed into a computational domain that encodes interactions between the system’s components with neighborhoods.

The TNN’s layers use message passing to successively update features and yield an output, e.g. a categorical label in classification or a quantitative value in regression. The output represents new knowledge extracted from the input data.

Domains

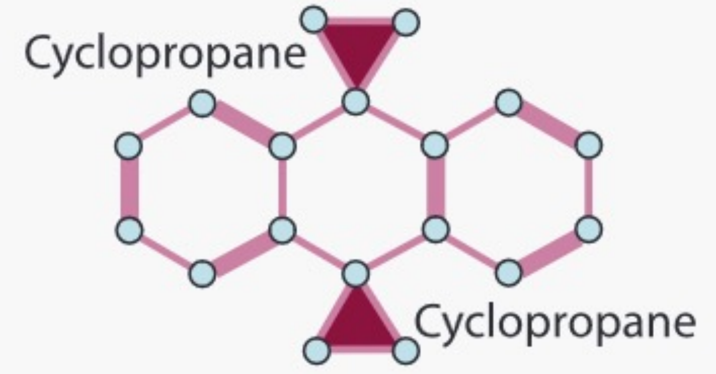
In Topological Deep Learning (TDL), data are features defined on discrete domains. Traditional examples of discrete domains include sets and graphs.

The domains of TDL generalize the pairwise relations of graphs to part-whole and set-types relations that permit the representation of more complex relational structure (see figure below: Nodes in blue, (hyper)edges in pink, and faces in dark red)

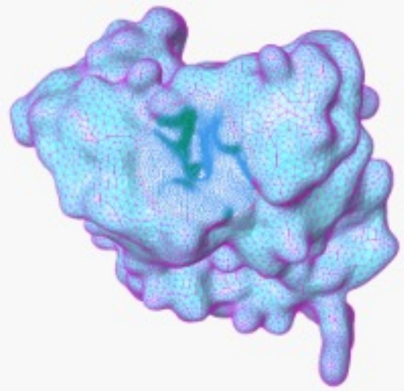


Simplicial complexes

Simplicial complexes (SCs) generalize graphs to incorporate hierarchical part-whole relations through the multi-scale construction of cells. Nodes are rank 0 cells that can be combined to form edges (rank 1 cells).



Edges are, in turn, combined to form faces (rank 2 cells), which are combined to form volumes (rank 3 cells), and so on. As such, an SC’s faces must be triangles, volumes must be tetrahedrons, and so forth.



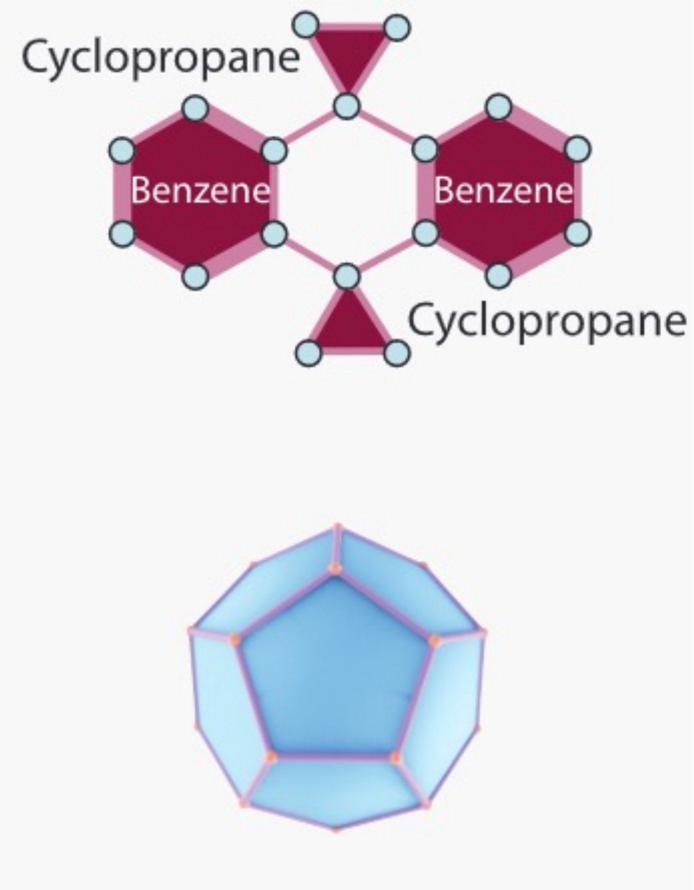
SCs are commonly used to encode discrete representations of 3D geometric surfaces represented with triangular meshes.

Cellular complexes

Cellular complexes (CCs) generalize SCs such that cells are not limited to simplexes: faces can involve more than three nodes, volumes more than four faces, and so on.

This flexibility endows CCs with greater expressivity than SCs.

A practitioner should consider employing this domain when studying a system that features part-whole interactions between more than three nodes, such as a molecule with benzene rings



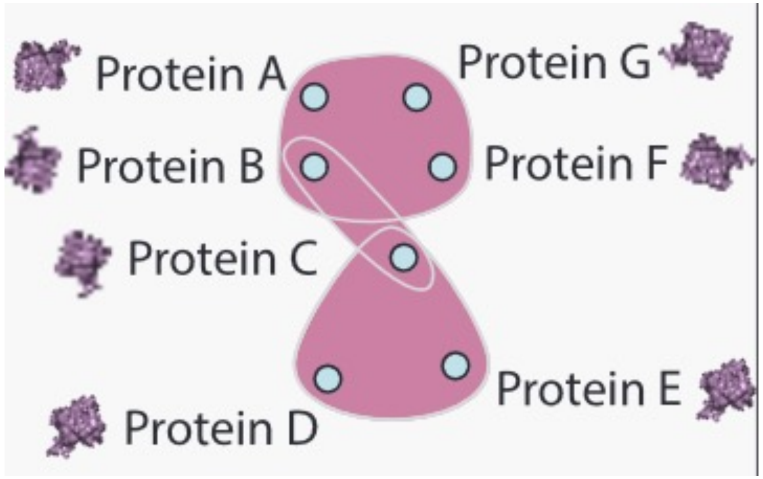
Hypergraphs

Hypergraphs (HGs) extend graphs in that their edges, called hyperedges, can connect more than two nodes. Connections in HGs represent set-type relationships, in which participation in an interaction is not implied by any other relation in the system.

This makes HGs an ideal choice for data with abstract and arbitrarily large interactions of equal importance, such as semantic text and citation networks.

Protein interaction networks also exhibit this property: an interaction between proteins requires a precise set of molecules—no more and no less.

The interaction of Proteins A, B, and C does not imply an interaction between A and B on their own.



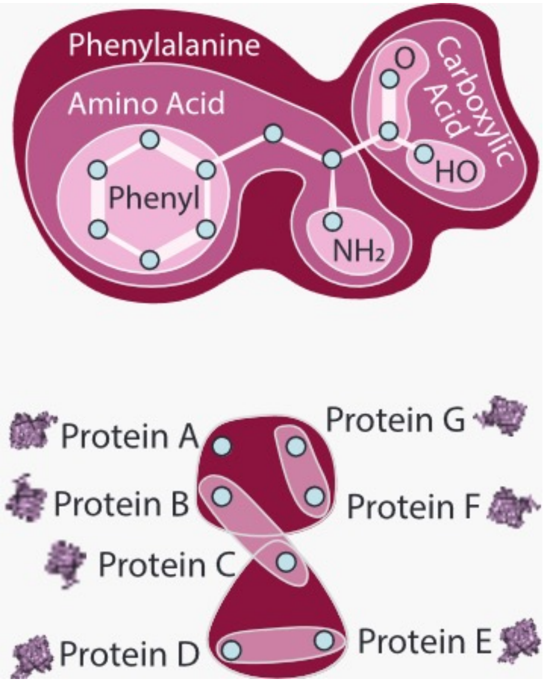
Combinatorial complexes

Combinatorial complexes generalise CCs and HGs to incorporate both part-whole and set-type relationships .

The benefit of this can be observed in the example of molecular representation.

The strict geometric constraints of simplicial and cellular complexes are too rigid for capturing much of hierarchical structure observed in molecules.

By contrast, the flexible but hierarchically ranked hyperedges of a combinatorial complex can capture the full richness of molecular structure.



Combinatorial Complexes

Combinatorial complexes including

(a) sequences and images,

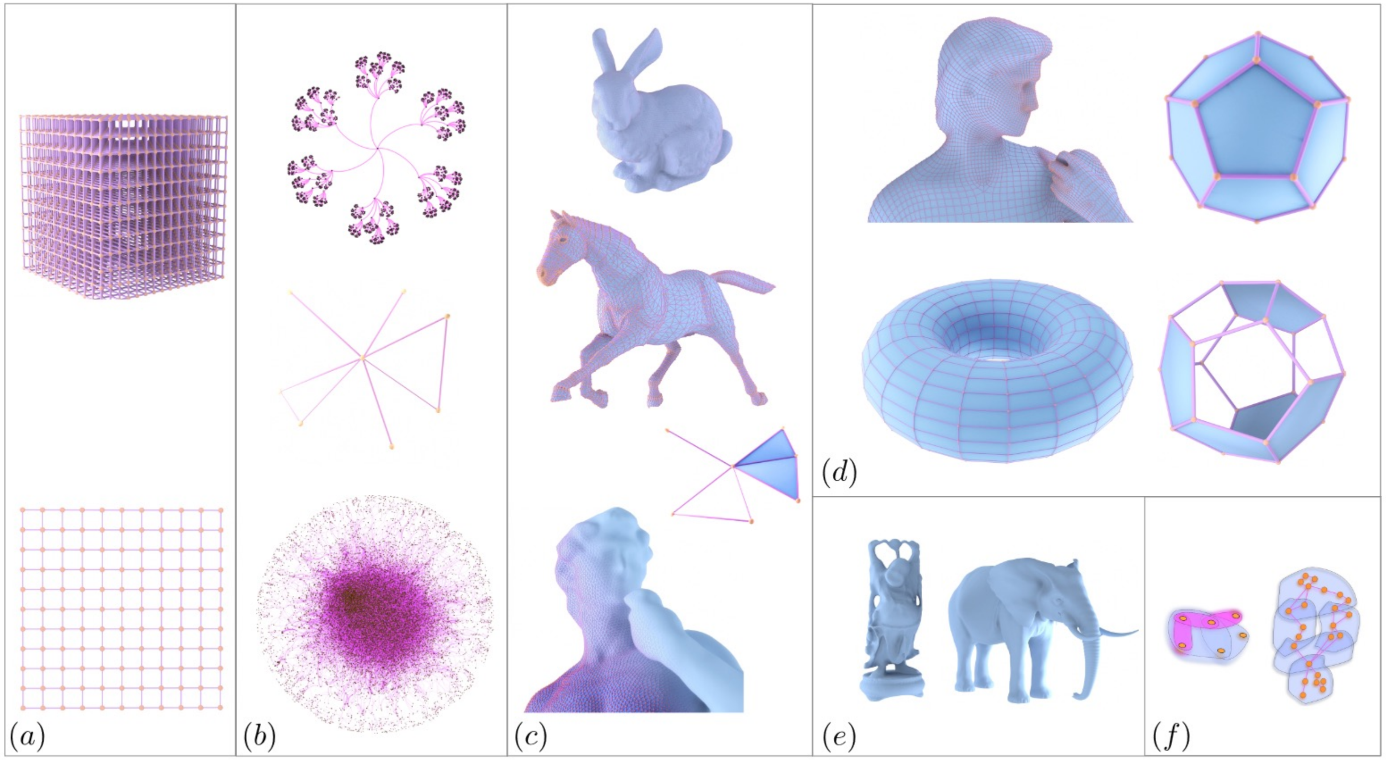
(b) graphs,

(c) 3D shapes and simplicial complexes,

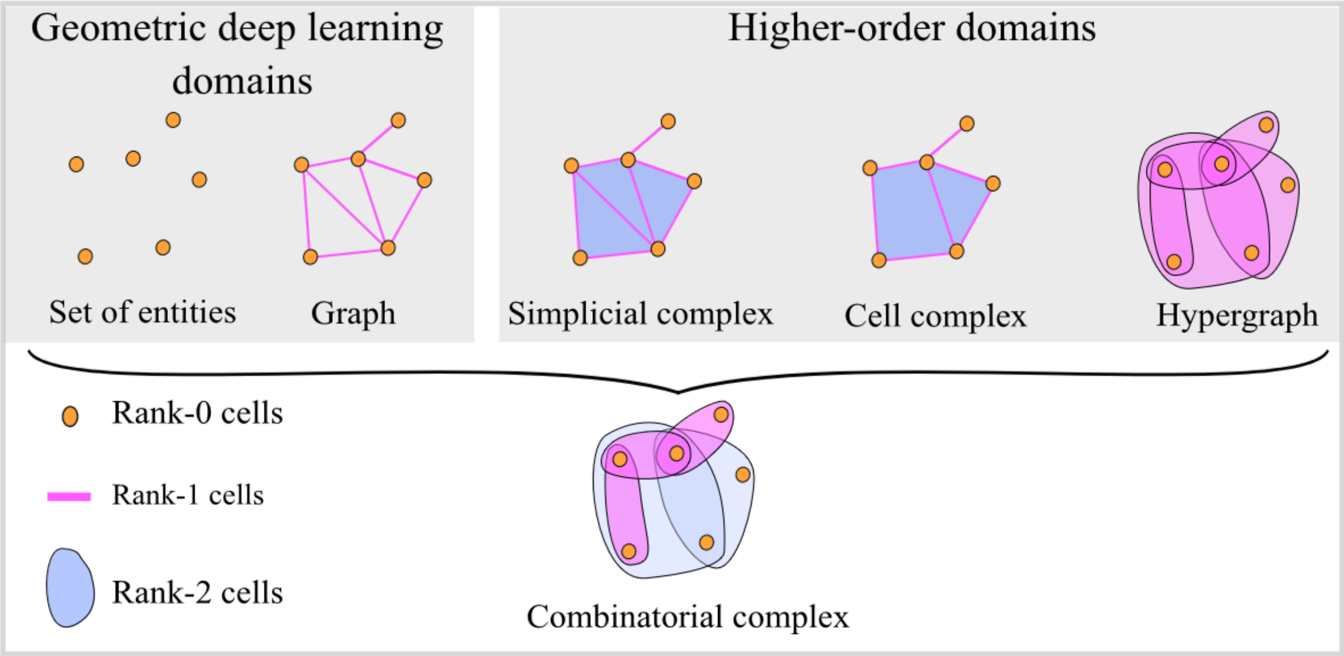
(d) cubical and cellular complexes,

(e) discrete manifolds, and

(f) hypergraphs.



The landscape of the Combinatorial Complexes



Sets have entities with no connections, graphs encode binary relations between vertices, simplicial and cell complexes model hierarchical higher-order relations, and hypergraphs accommodate arbitrary set-type relations with no hierarchy.

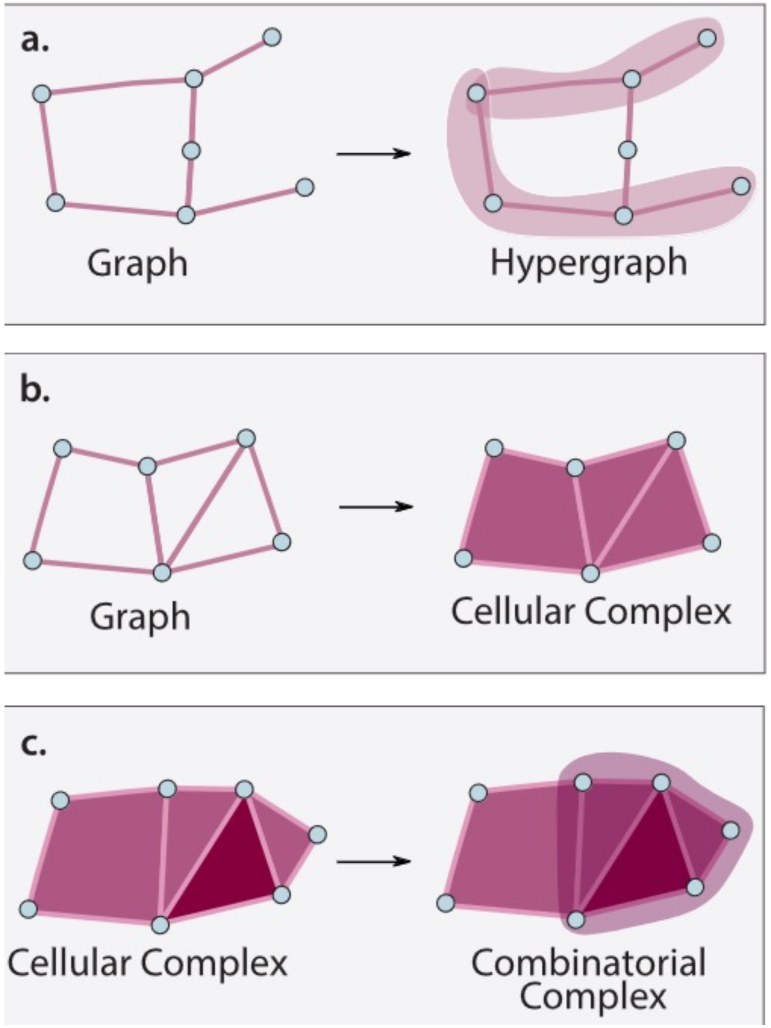
Combinatorial complexes (CCCs) generalise graphs, simplicial and cell complexes, and hypergraphs. CCCs are equipped with set-type relations as well as with a hierarchy of these relation.

Lifting topological domains

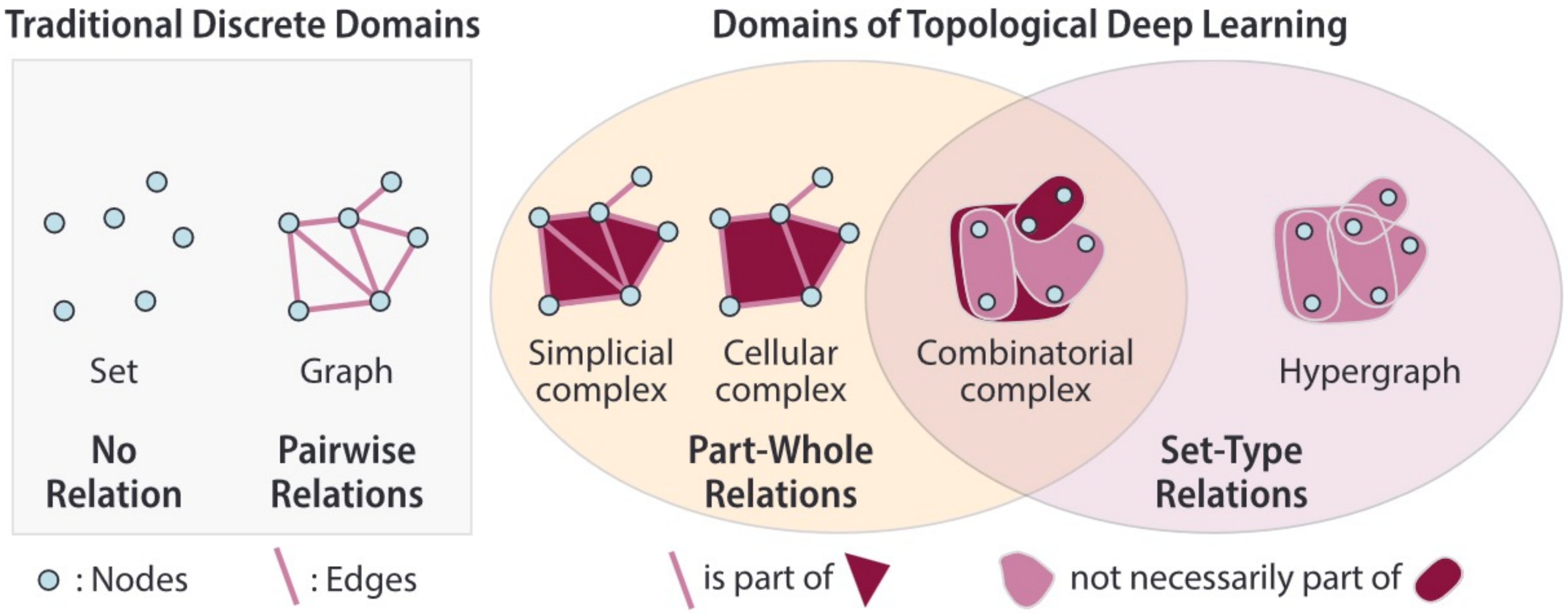
(a) A graph is lifted to a hypergraph by adding hyperedges that connect groups of nodes.

(b) A graph can be lifted to a cellular complex by adding faces of any shape.

(c) Hyperedges can be added to a cellular complex to lift the structure to a combinatorial complex.



Topological Deep learning

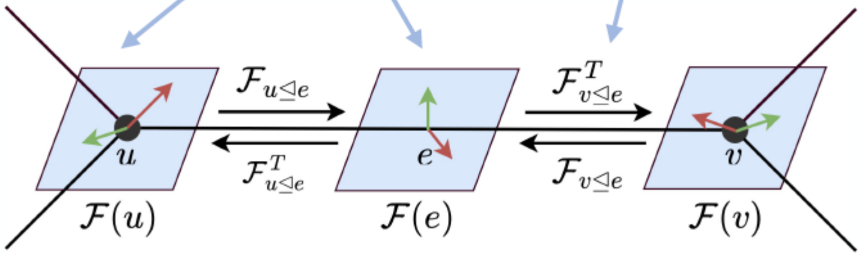


from Papillon et al. 2023

Sheaf Neural Networks

Vector spaces

Linear maps



$\mathcal{F}_{u \trianglelefteq e}$
$\mathcal{F}^T_{v \trianglelefteq e}$
u
e
v
$\mathcal{F}^T_{u \trianglelefteq e}$
$\mathcal{F}_{v \trianglelefteq e}$
$\mathcal{F}(u)$
$\mathcal{F}(e)$
$\mathcal{F}(v)$

Sheaf Laplacian

$$L_{\mathcal{F}} = \begin{bmatrix} \sum_{v_1 \trianglelefteq e} \mathcal{F}^T_{v_1 \trianglelefteq e} \mathcal{F}_{v_1 \trianglelefteq e} & \cdots & -\mathcal{F}^T_{v_1 \trianglelefteq e} \mathcal{F}_{v_n \trianglelefteq e} \\ \vdots & \ddots & \vdots \\ -\mathcal{F}^T_{v_n \trianglelefteq e} \mathcal{F}_{v_n \trianglelefteq e} & \cdots & \sum_{v_1 \trianglelefteq e} \mathcal{F}^T_{v_n \trianglelefteq e} \mathcal{F}_{v_n \trianglelefteq e} \end{bmatrix}$$

Normalised Sheaf Laplacian

$$\Delta_{\mathcal{F}} = D^{-\frac{1}{2}} L_{\mathcal{F}} D^{-\frac{1}{2}}$$

$$\frac{\partial}{\partial t} X(t) = -\Delta_{\mathcal{F}} X(t)$$

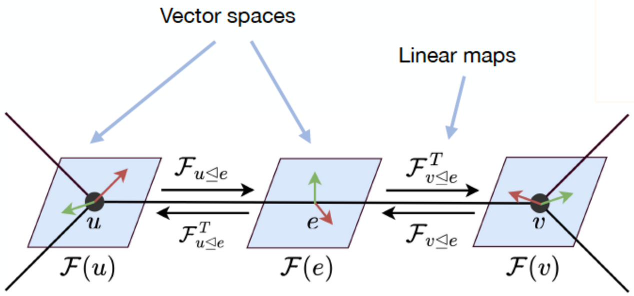
Sheaf Heat Diffusion PDE

Generalises GCNs

$$X_{t+1} = X_t - \sigma(\Delta_{\mathcal{F}(t)}(I \otimes W_1^t) X_t W_2^t)$$

Sheaf Convolutional Network

Sheaf Neural Networks


Vector spaces
Linear maps
$\mathcal{F}_{u\trianglelefteq e}$
$\mathcal{F}^T_{v\trianglelefteq e}$
u
e
v
$\mathcal{F}^T_{u\trianglelefteq e}$
$\mathcal{F}_{v\trianglelefteq e}$
$\mathcal{F}(u)$
$\mathcal{F}(e)$
$\mathcal{F}(v)$

Sheaf Laplacian

$$L_{\mathcal{F}} = \begin{bmatrix} \sum_{v_1\trianglelefteq e} \mathcal{F}^T_{v_1\trianglelefteq e}\mathcal{F}_{v_1\trianglelefteq e} & \cdots & -\mathcal{F}^T_{v_1\trianglelefteq e}\mathcal{F}_{v_n\trianglelefteq e} \\ \vdots & \ddots & \vdots \\ -\mathcal{F}^T_{v_n\trianglelefteq e}\mathcal{F}_{v_n\trianglelefteq e} & \cdots & \sum_{v_1\trianglelefteq e} \mathcal{F}^T_{v_n\trianglelefteq e}\mathcal{F}_{v_n\trianglelefteq e} \end{bmatrix}$$

Normalised Sheaf Laplacian

$$\Delta_{\mathcal{F}} = D^{-\frac{1}{2}} L_{\mathcal{F}} D^{-\frac{1}{2}}$$

$$\frac{\partial}{\partial t} X(t) = -\Delta_{\mathcal{F}} X(t)$$

Sheaf Heat Diffusion PDE

Generalises GCNs

$$X_{t+1} = X_t - \sigma(\Delta_{\mathcal{F}(t)}(I \otimes W_1^t)X_t W_2^t)$$

Sheaf Convolutional Network

Barbero et al, Sheaf Neural Networks with Connection Laplacians,
Bodnar et al., Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns, arxiv

Sheaf Neural Networks

Attention matrix Adjacency matrix

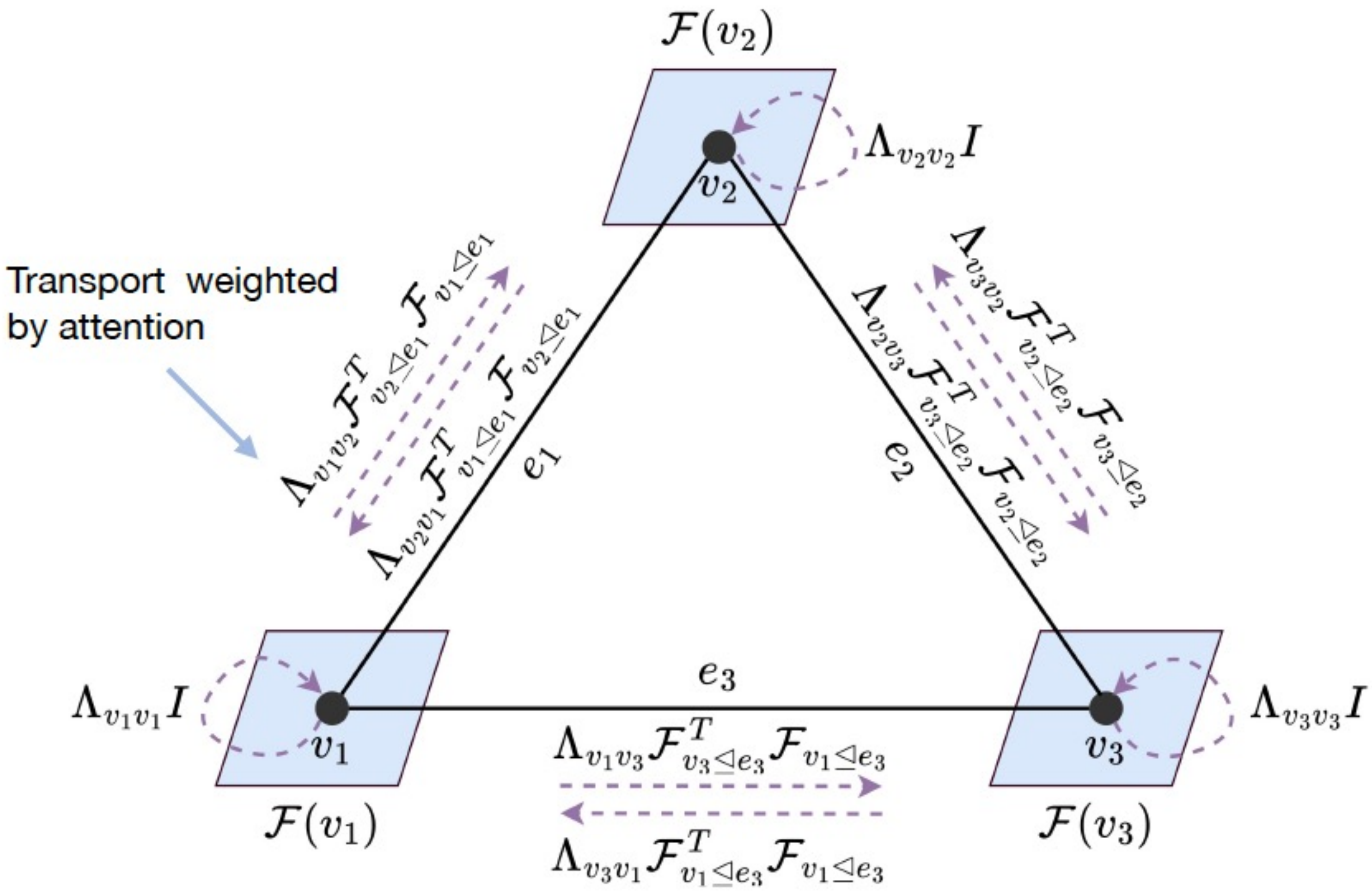
$$\frac{\partial}{\partial t} X(t) = (\Lambda \odot A_{\mathcal{F}} - I) X(t)$$

Sheaf Attention Network

$$X_{t+1} = X_t + \sigma((\Lambda \odot A_{\mathcal{F}} - I)(I \otimes W_1^t) X_t W_2^t)$$

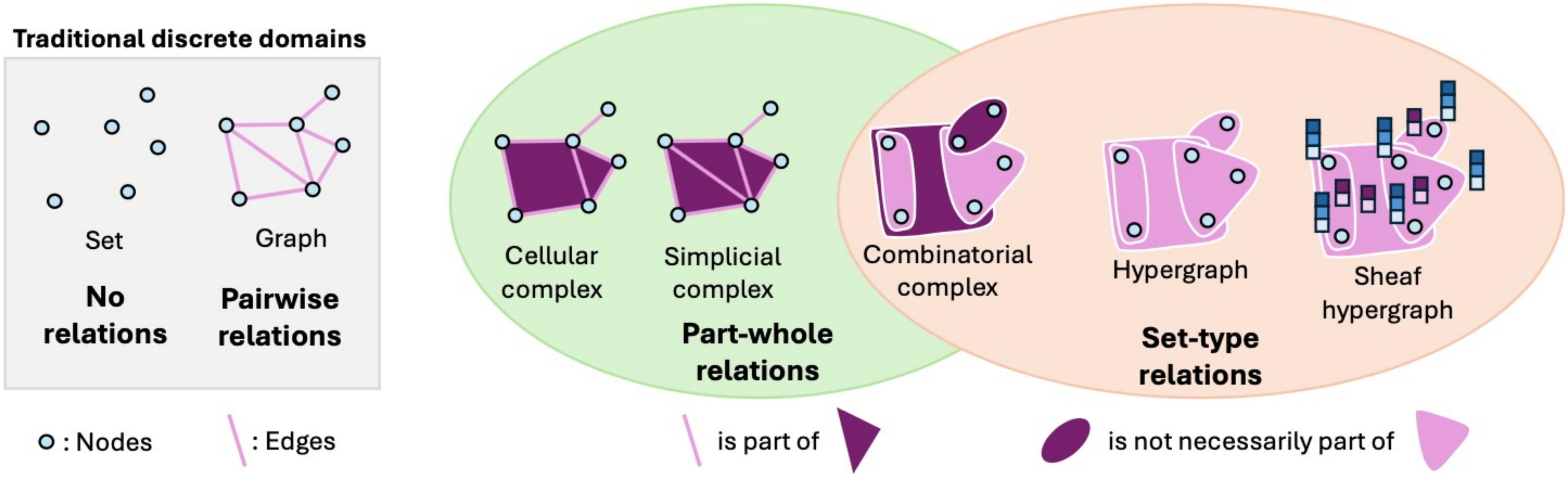
Generalises GATs

Sheaf Neural Networks


Transport weighted by attention

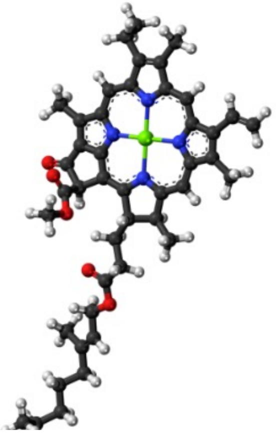
$\mathcal{F}(v_2)$
$\Lambda_{v_2v_2}I$
v_2
$\Lambda_{v_1v_2}\mathcal{F}^T_{v_2\trianglelefteq e_1}\mathcal{F}_{v_1\trianglelefteq e_1}$
$\Lambda_{v_2v_1}\mathcal{F}^T_{v_1\trianglelefteq e_1}\mathcal{F}_{v_2\trianglelefteq e_1}$
e_1
$\Lambda_{v_3v_2}\mathcal{F}^T_{v_2\trianglelefteq e_2}\mathcal{F}_{v_3\trianglelefteq e_2}$
$\Lambda_{v_2v_3}\mathcal{F}^T_{v_3\trianglelefteq e_2}\mathcal{F}_{v_2\trianglelefteq e_2}$
e_2
e_3
$\Lambda_{v_1v_1}I$
v_1
$\Lambda_{v_1v_3}\mathcal{F}^T_{v_3\trianglelefteq e_3}\mathcal{F}_{v_1\trianglelefteq e_3}$
v_3
$\Lambda_{v_3v_3}I$
$\mathcal{F}(v_1)$
$\mathcal{F}(v_3)$
$\Lambda_{v_3v_1}\mathcal{F}^T_{v_1\trianglelefteq e_3}\mathcal{F}_{v_1\trianglelefteq e_3}$

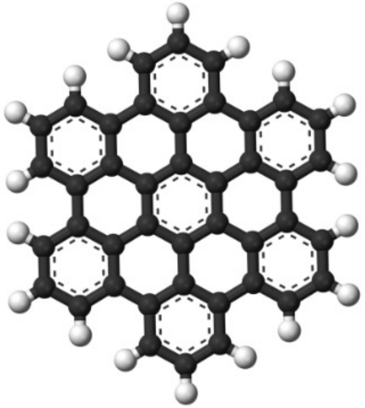
Topological Deep learning

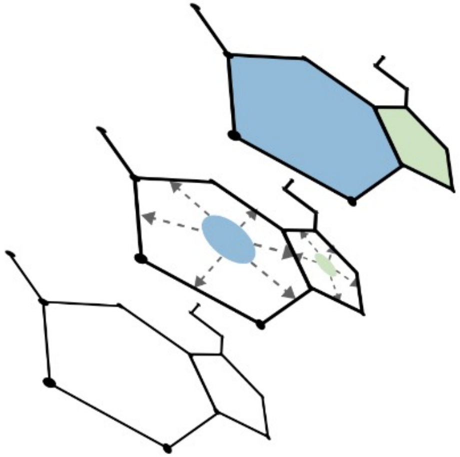


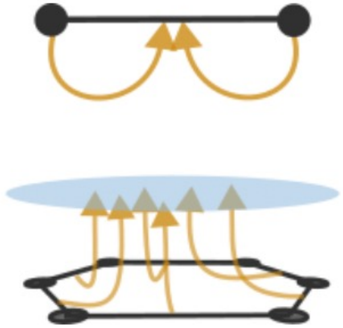
- A taxonomy of topological domains. Adapted from Papillon et al.

Sheaf for Bigger molecules



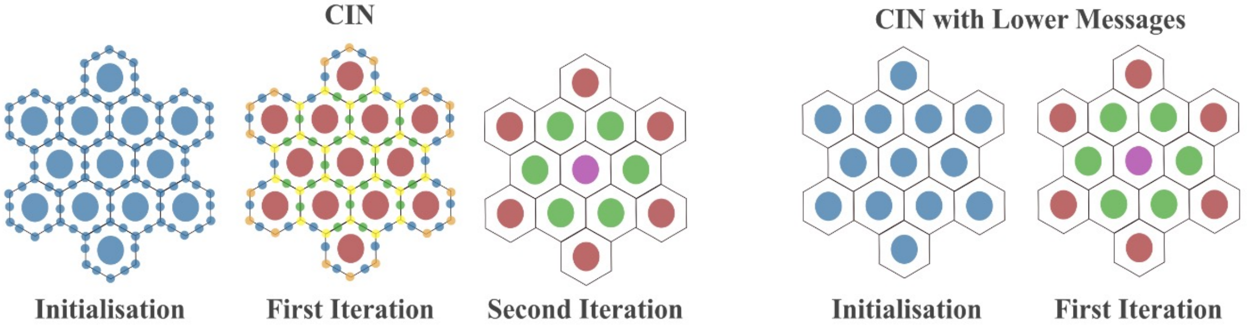






Cellular lifting process. Given an input graph G, we attach closed two-dimensional rings to the boundary of the induced cycles of G. The result is a 2D regular cell complex C.

Boundary messages received by an edge (top) and a ring (bottom)



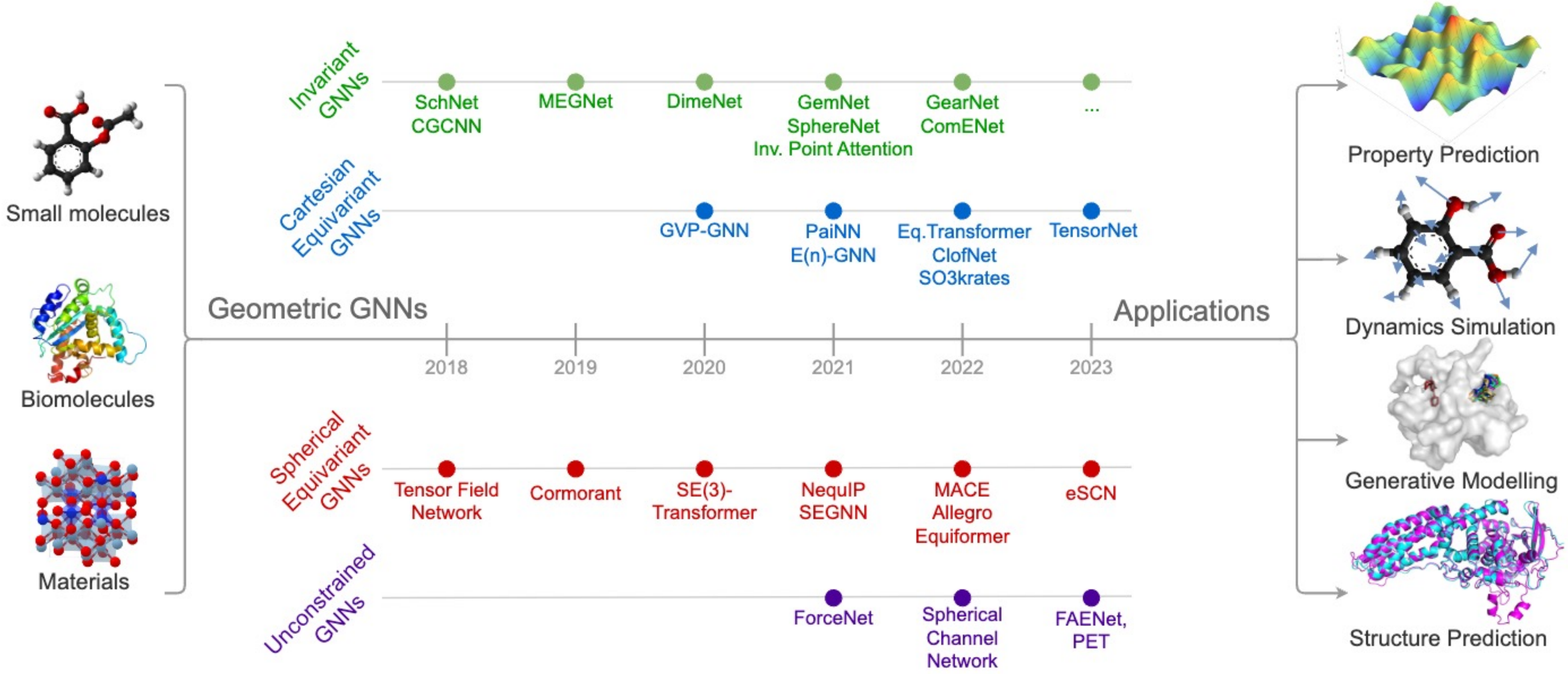
Lorenzo Giusti et al., CIN++: Enhancing Topological Message Passing, arxiv

Geometric Deep Learning Across Scales:

Molecules
Cell
Tissues
Patients
Population

Molecules

Geometric Deep Learning has many applications on molecular data

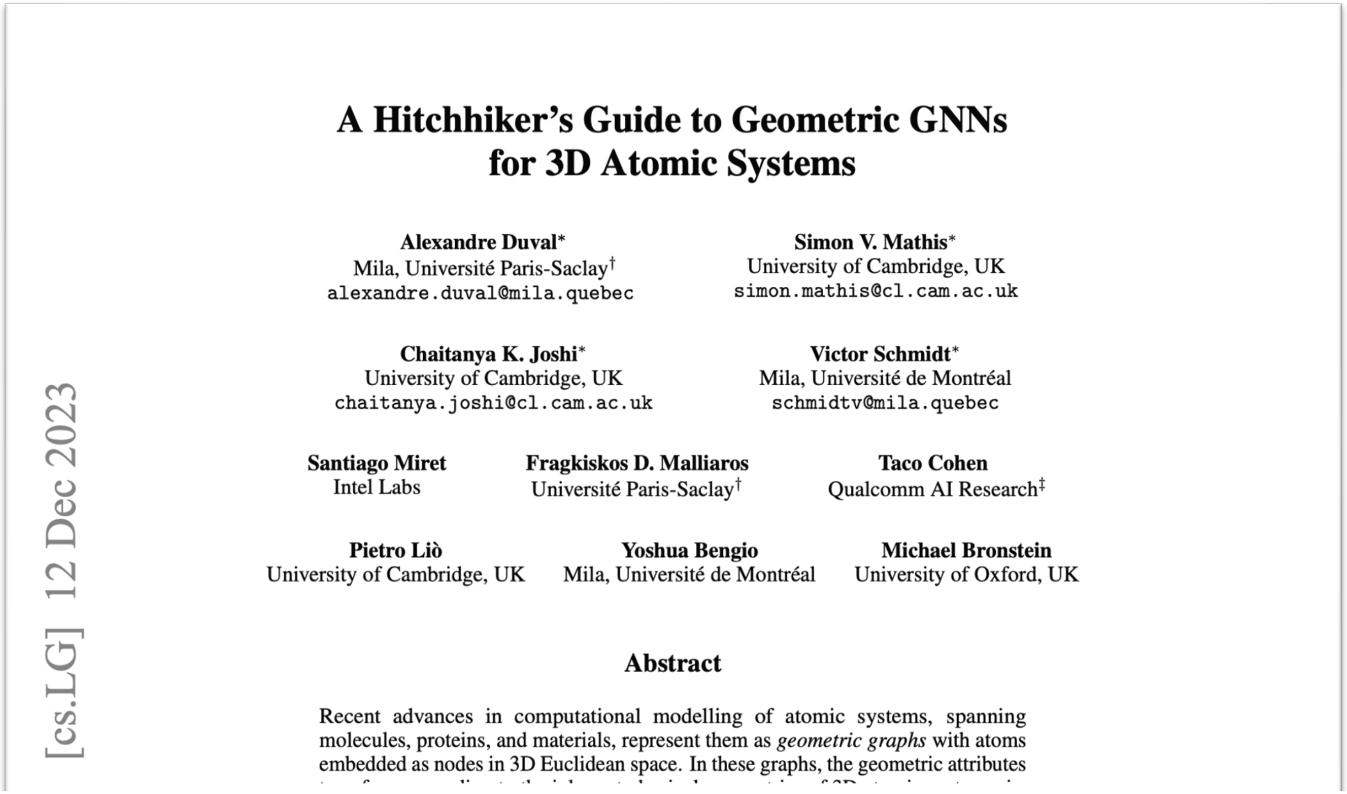


A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems

Illustration courtesy Chaitanya Joshi

If you're interested in this space, check out our tutorial paper.

2312.07511.pdf (arxiv.org)



A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems

Alexandre Duval*
Mila, Université Paris-Saclay[†]
alexandre.duval@mila.quebec

Simon V. Mathis*
University of Cambridge, UK
simon.mathis@cl.cam.ac.uk

Chaitanya K. Joshi*
University of Cambridge, UK
chaitanya.joshi@cl.cam.ac.uk

Victor Schmidt*
Mila, Université de Montréal
schmidtv@mila.quebec

Santiago Miret
Intel Labs

Fragkiskos D. Malliaros
Université Paris-Saclay[†]

Taco Cohen
Qualcomm AI Research[‡]

Pietro Liò
University of Cambridge, UK

Yoshua Bengio
Mila, Université de Montréal

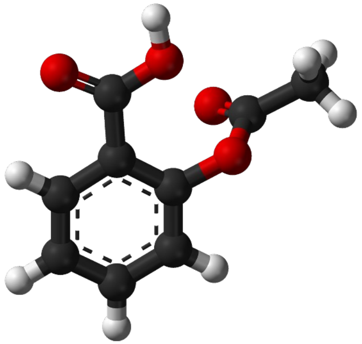
Michael Bronstein
University of Oxford, UK

[cs.LG] 12 Dec 2023

Abstract

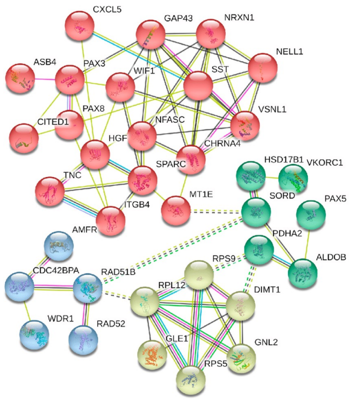
Recent advances in computational modelling of atomic systems, spanning molecules, proteins, and materials, represent them as *geometric graphs* with atoms embedded as nodes in 3D Euclidean space. In these graphs, the geometric attributes

Graphs are everywhere in biology

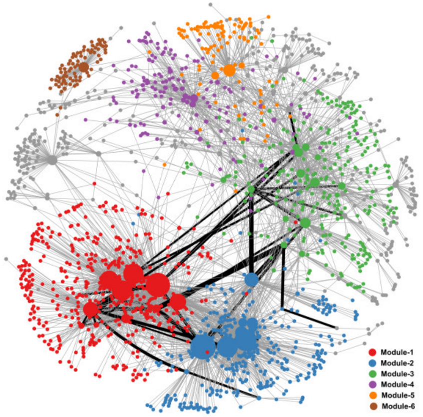


Drug molecules

Protein structure



Protein-protein interaction networks

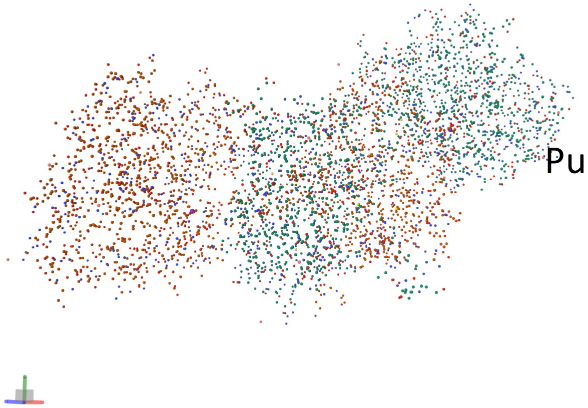


Gene regulatory networks

Unsurprisingly, ***Graph Neural Networks (GNNs)*** *have achieved remarkable results in* ***biological modelling***

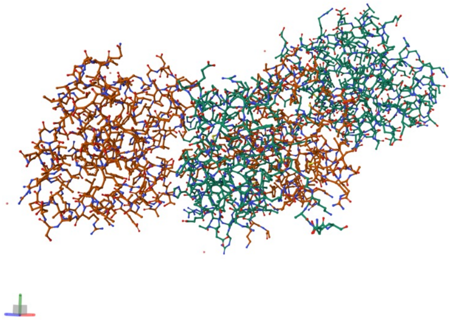
Slide credit: Chaitanya Joshi

There are several ways of representing protein structures

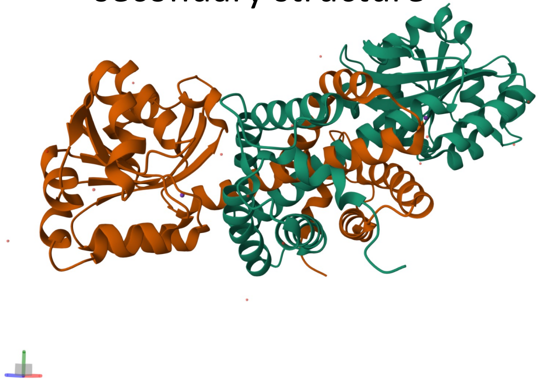


Pure point cloud

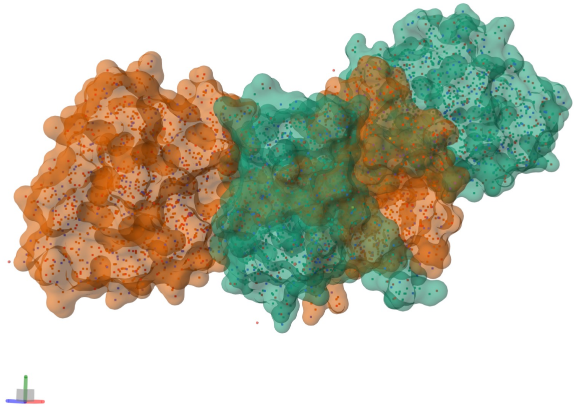
with bonds

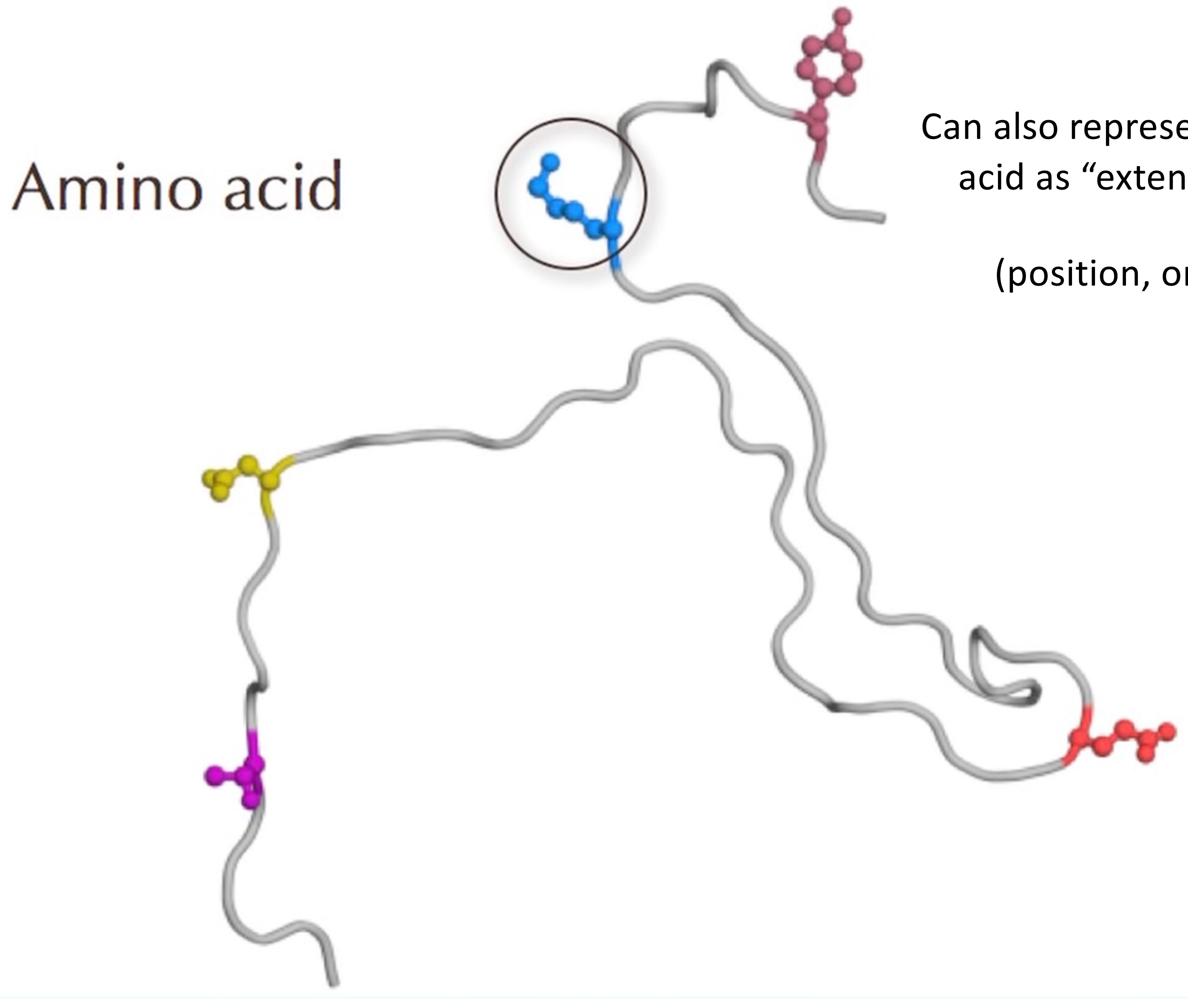


with protein-specific secondary structure



as surface

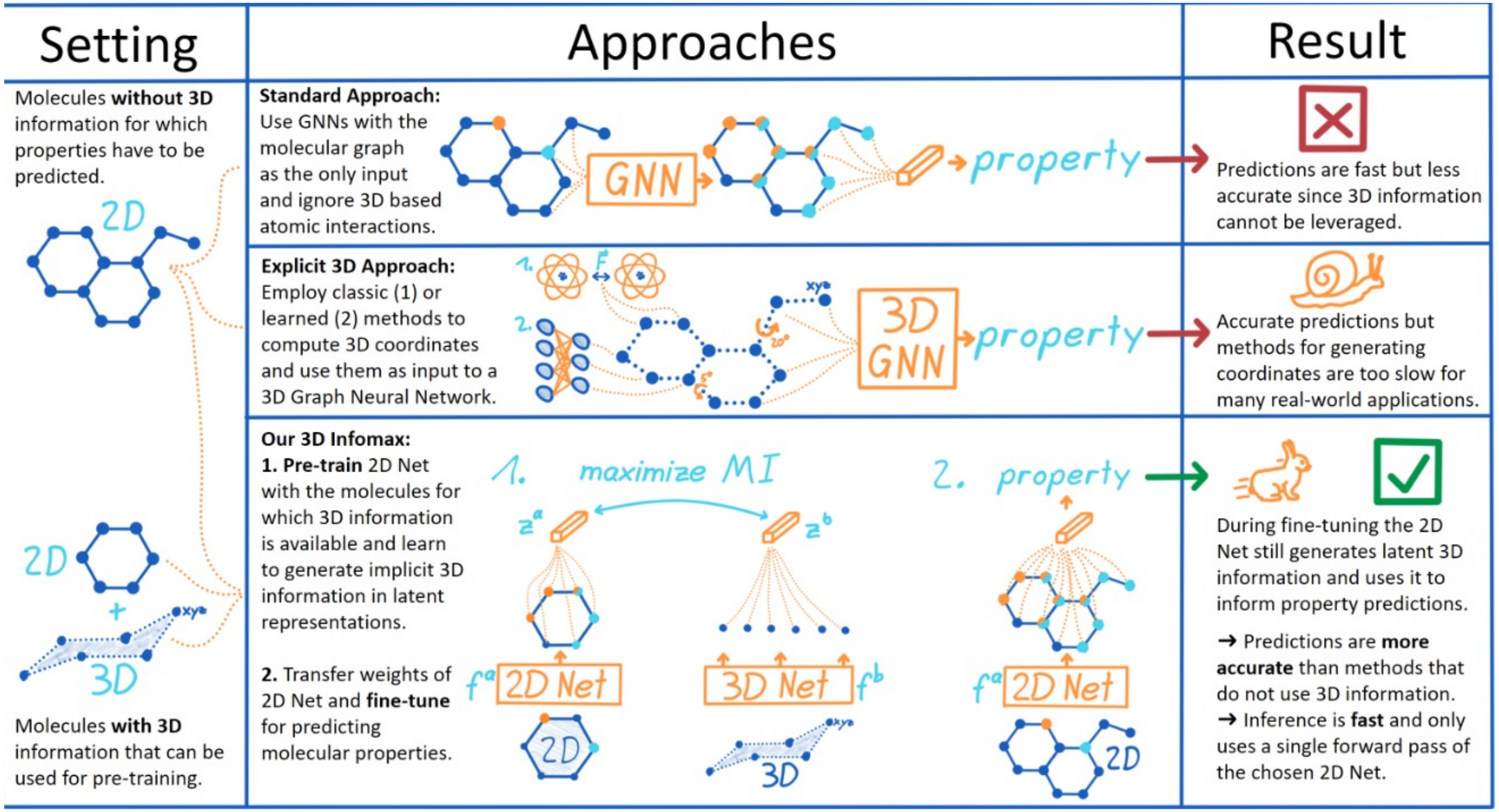




Can also represent each amino acid as “extended object”:

(position, orientation) $\in \mathbb{R}^3 \times \mathrm{SO}(3)$

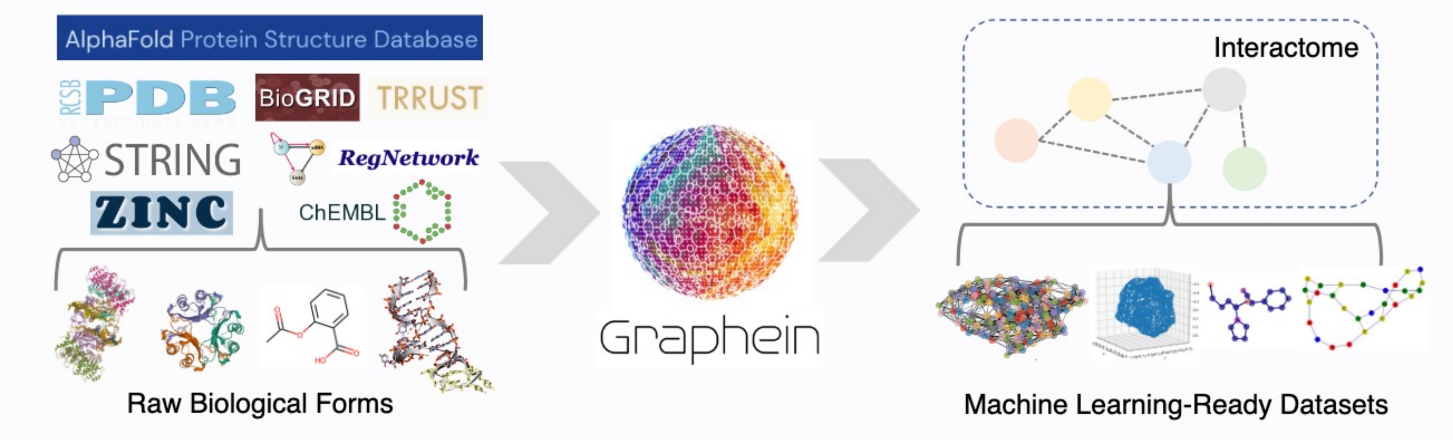
3D Infomax

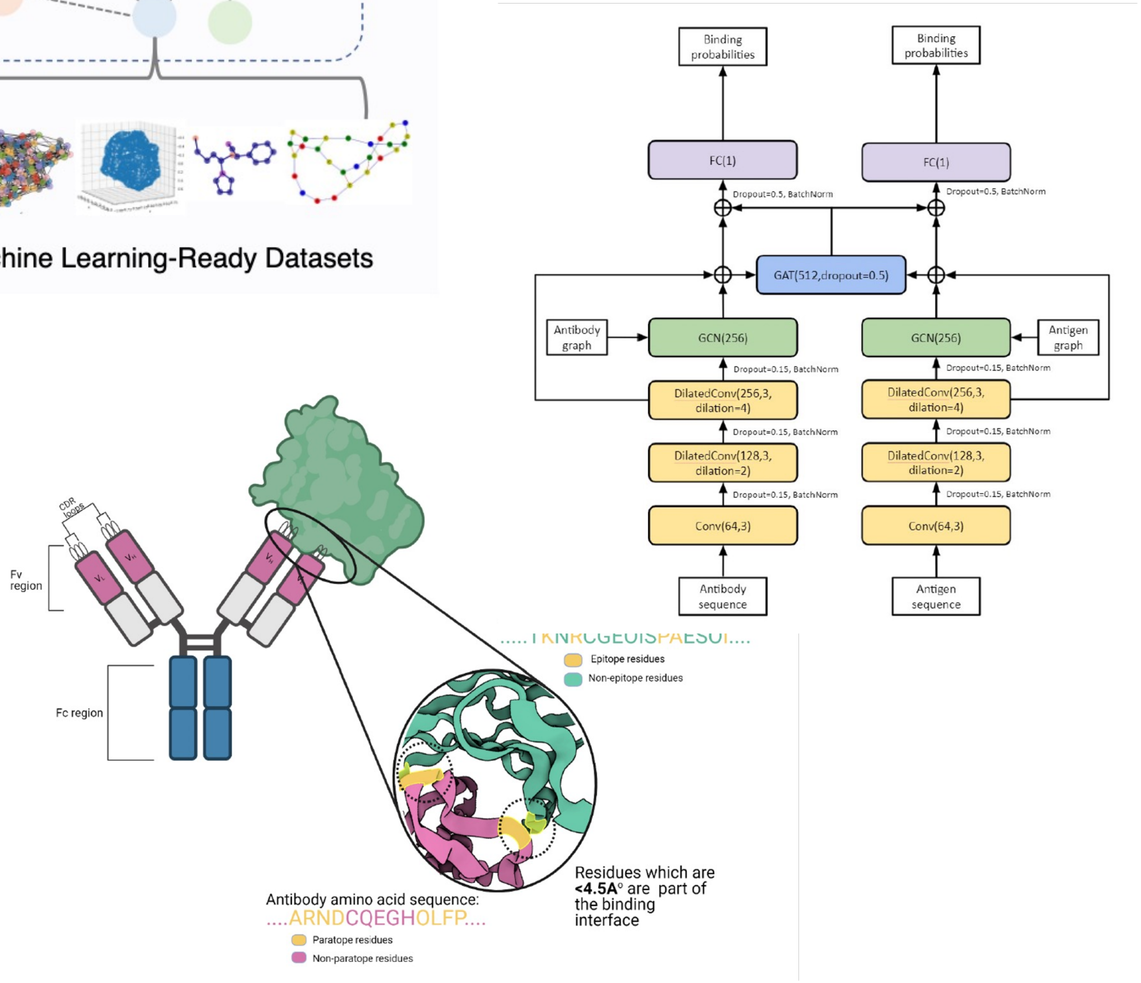


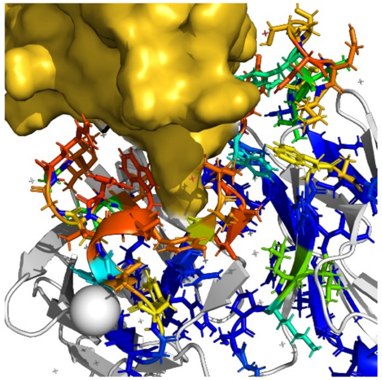
Setting	Approaches	Result
Molecules **without 3D** information for which properties have to be predicted.	**Standard Approach:** Use GNNs with the molecular graph as the only input and ignore 3D based atomic interactions.	Predictions are fast but less accurate since 3D information cannot be leveraged.
	Explicit 3D Approach: Employ classic (1) or learned (2) methods to compute 3D coordinates and use them as input to a 3D Graph Neural Network.	Accurate predictions but methods for generating coordinates are too slow for many real-world applications.
Molecules **with 3D** information that can be used for pre-training.	**Our 3D Infomax:** **1. Pre-train** 2D Net with the molecules for which 3D information is available and learn to generate implicit 3D information in latent representations. **2.** Transfer weights of 2D Net and **fine-tune** for predicting molecular properties.	During fine-tuning the 2D Net still generates latent 3D information and uses it to inform property predictions. → Predictions are **more accurate** than methods that do not use 3D information. → Inference is **fast** and only uses a single forward pass of the chosen 2D Net.

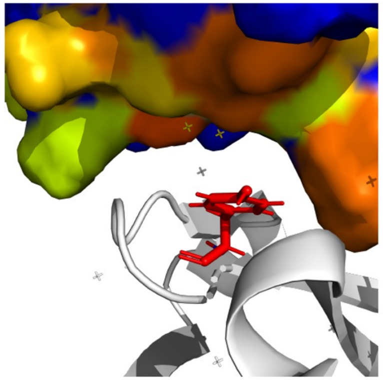
Hannes Stärk, Dominique Beaini, Gabriele Corso, Prudencio Tossou, Christian Dallago, Stephan Günnemann, Pietro Liò 3D Infomax improves GNNs for Molecular Property Prediction https://arxiv.org/abs/2110.04126

Designing proteins





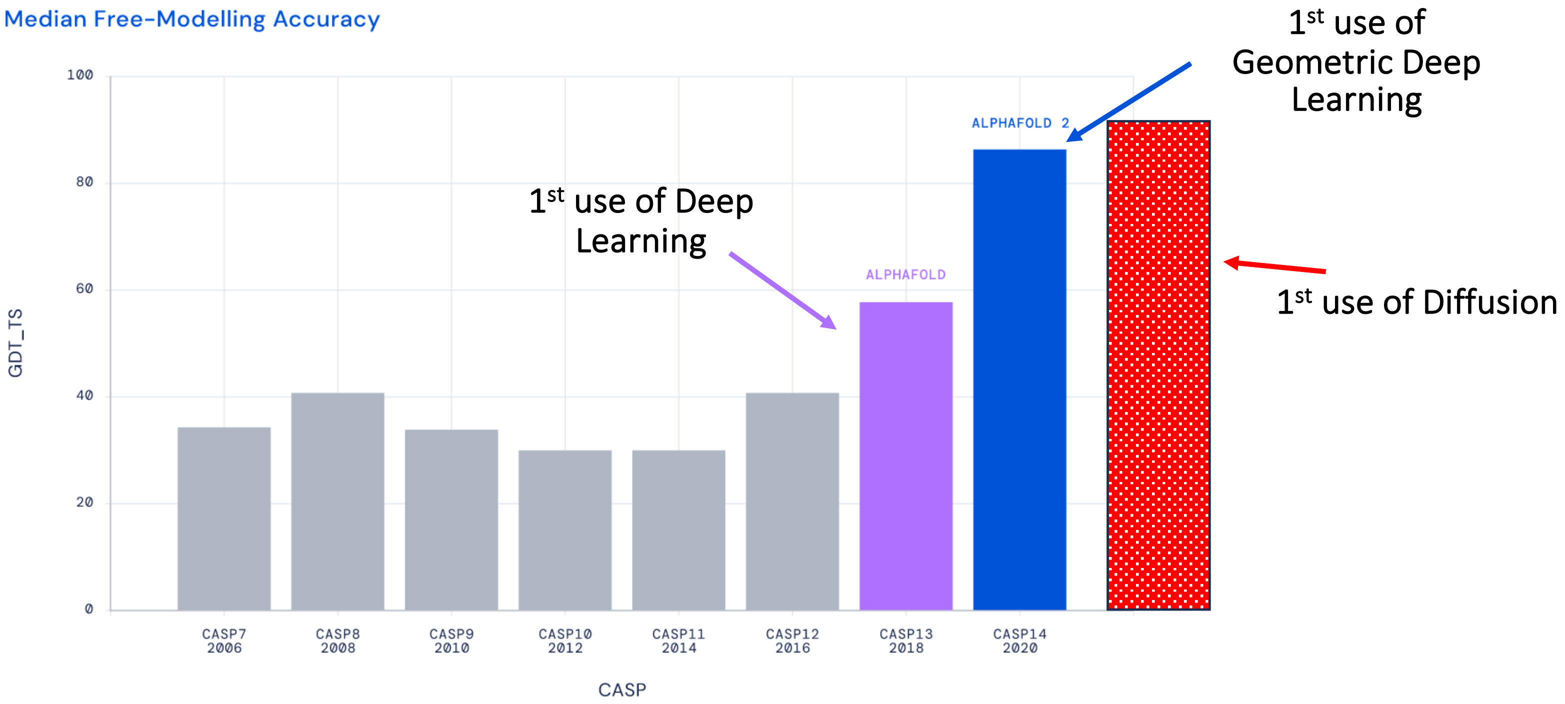




with Arian Jamasb, Andreea Deac, Petar Velickovic, Alice del Vecchio

AlphaFold2-3

Jumper et al. 2021 (DeepMind)+



1st use of Deep Learning

1st use of Geometric Deep Learning

1st use of Diffusion

Structural information has become much more available recently, in part thanks to geometric deep learning.

The importance of symmetries

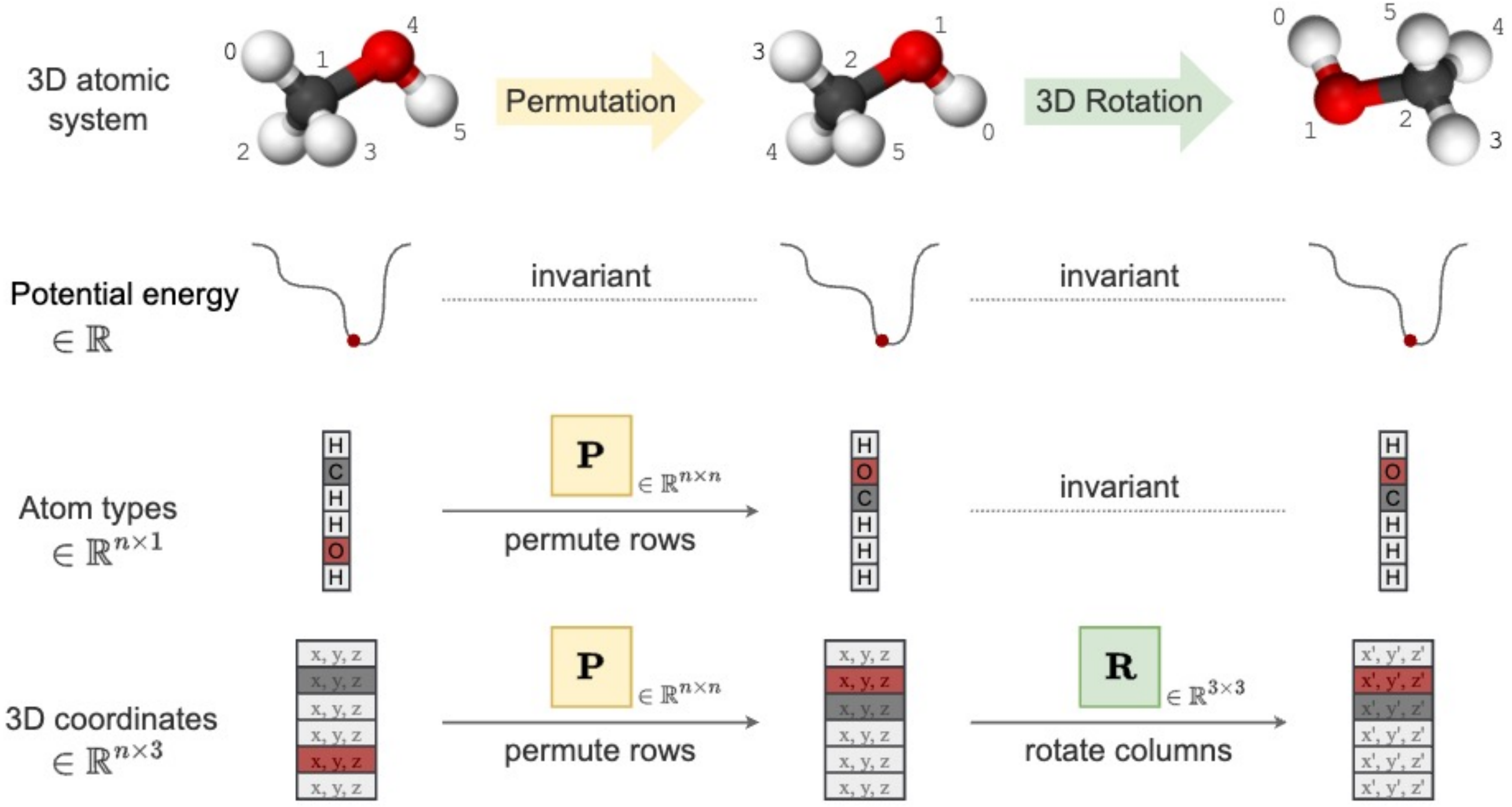
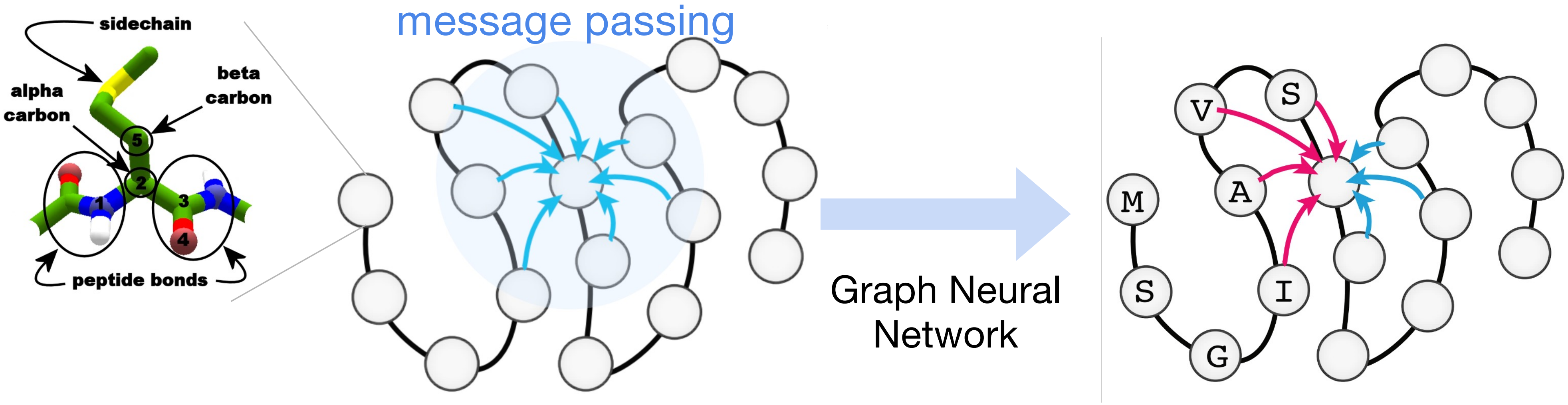


Illustration courtesy Chaitanya Joshi

Protein backbones as graphs

ProteinMPNN – message passing neural network

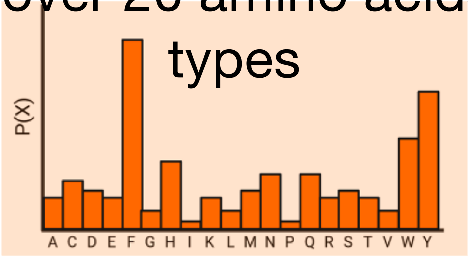


○ Node (amino acid)

Backbone

● Neighbourhood in 3D

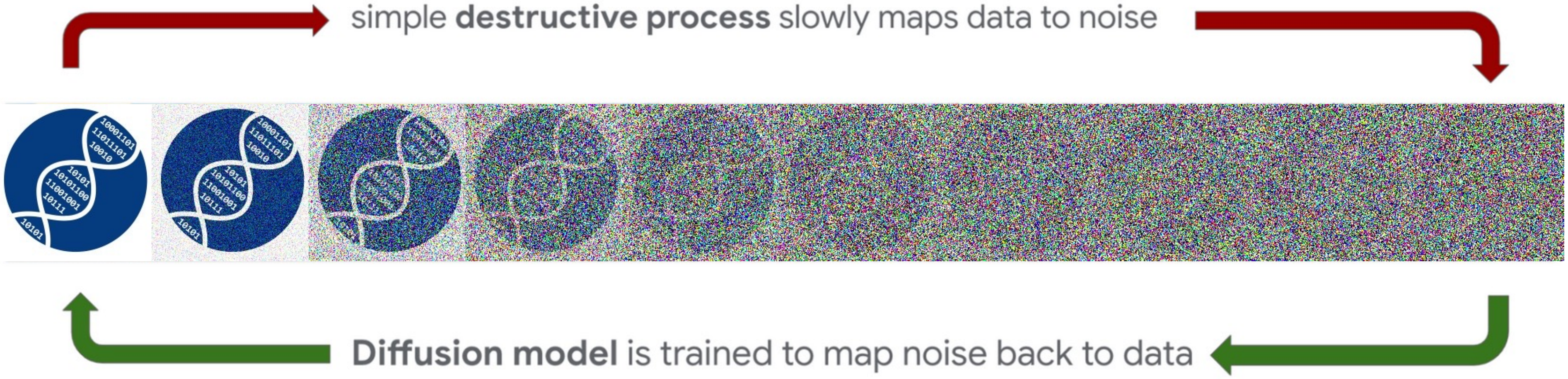
Probability distribution over 20 amino acid types



Ingraham et al. Generative models for graph-based protein design. NeurIPS. 2019.

Diffusion Models (Level 1)

Sohl-Dickenstein et al. 2015 (Stanford)



simple **destructive process** slowly maps data to noise

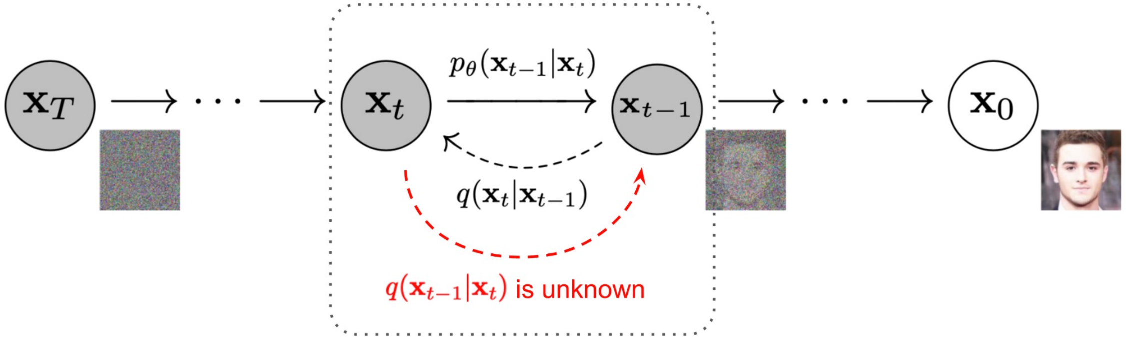
Diffusion model is trained to map noise back to data

Google Research, 2022 & Beyond: Language, Vision and Generative Models (Google Research)

Diffusion Models (Level 2)

Sohl-Dickenstein et al. 2015 (Stanford)

"Diffusion models define a Markovian chain of random diffusion steps, gradually adding noise to sample data until it loses all of its distinguishing features. A neural network is then trained to reverse this process"



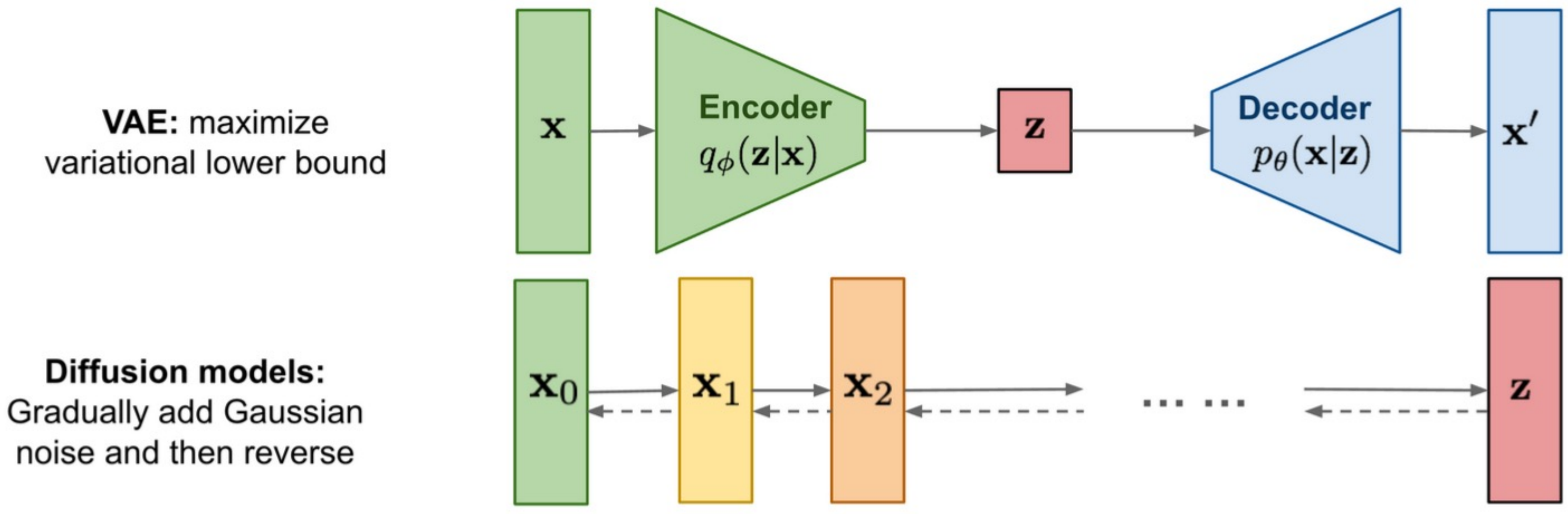
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$
$q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown

Forward Diffusion Process $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$

Reverse (Generative) Diffusion Process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\boxed{\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)}}_{\text{Learned}}, \underbrace{\boxed{\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)}}_{\text{Usually fixed}})$

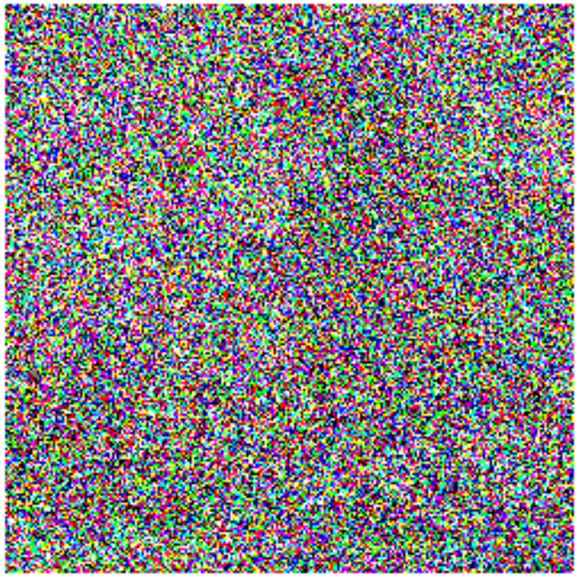
(lilianweng.github.io)

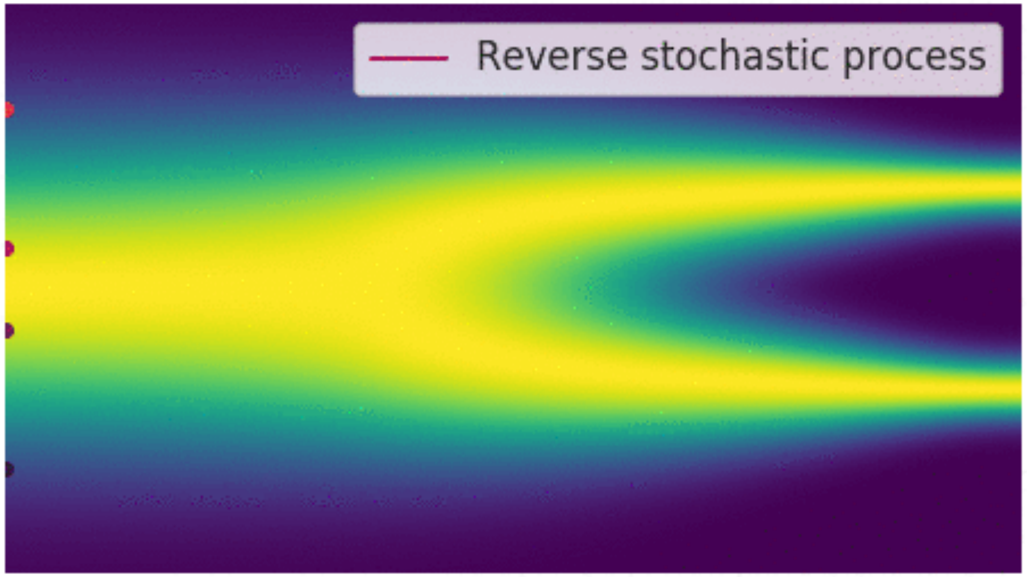
Diffusion models can be seen as a ‘stretched out’ VAE



Credit: lilianweng.github.io

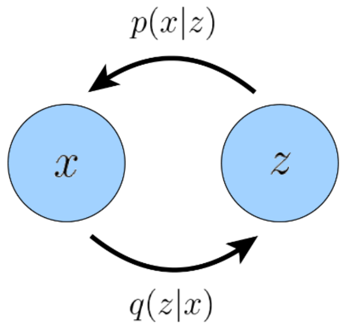
Approximating some target distribution





Credit: lilianweng.github.io

Variational Autoencoders



A Variational Autoencoder graphically represented. Here, encoder q(z|x) defines a distribution over latent variables z for observations x, and p(x|z) decodes latent variables into observations.

$$\mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\right] = \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_{\phi}(\boldsymbol{z}|\boldsymbol{a}}\right] \qquad \text{(Chain Rule of Probability)}$$

$$= \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] + \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{}{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\right] \qquad \text{(Split the Expectation)}$$

$$= \underbrace{\mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}} \qquad \text{(Definition of KL Divergence)}$$

$$q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\phi}(\boldsymbol{x}), \boldsymbol{\sigma}_{\phi}^2(\boldsymbol{x})\mathbf{I})$$
$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I})$$

The encoder of the VAE is commonly chosen to model a multivariate Gaussian with diagonal covariance, and the prior is often selected to be a standard multivariate Gaussian;
the KL divergence term of the ELBO can be computed analytically, and the reconstruction term can be approximated using a Monte Carlo estimate.
Our objective can then be rewritten as:

$$\underset{\boldsymbol{\phi},\boldsymbol{\theta}}{\arg\max}\, \mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] - D_{\mathrm{KL}}(q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z})) \approx \underset{\boldsymbol{\phi},\boldsymbol{\theta}}{\arg\max} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(l)}) - D_{\mathrm{KL}}(q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))$$

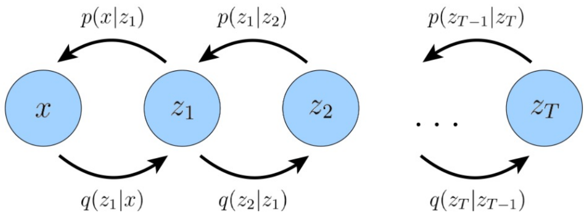
Hierarchical Variational Autoencoders

A Hierarchical Variational Autoencoder (HVAE) is a generalization of a VAE that extends to multiple hierarchies over latent variables.

Under this formulation, latent variables themselves are interpreted as generated from other higher-level, more abstract latents.

Whereas in the general HVAE with T hierarchical levels, each latent is allowed to condition on all previous latents, we focus on a special case which we call a Markovian HVAE (MHVAE).

In a MHVAE, the generative process is a Markov chain; that is, each transition down the hierarchy is Markovian, wher

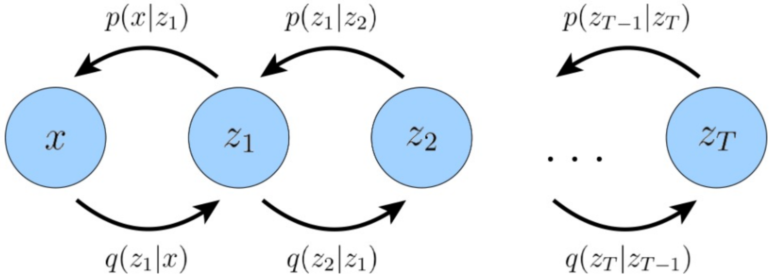


A Markovian Hierarchical Variational Autoencoder with T hierarchical latents. The generative process is modeled as a Markov chain, where each latent zt is generated only from the previous latent zt+1.

Hierarchical Variational Autoencoders

$$p(\boldsymbol{x}, \boldsymbol{z}_{1:T}) = p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$$

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x}) = q_{\boldsymbol{\phi}}(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})$$



A Markovian Hierarchical Variational Autoencoder with T hierarchical latents. The generative process is modeled as a Markov chain, where each latent z_t is generated only from the previous latent z_{t+1}.

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}, \boldsymbol{z}_{1:T})d\boldsymbol{z}_{1:T} \\
&= \log \int \frac{p(\boldsymbol{x}, \boldsymbol{z}_{1:T})q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}d\boldsymbol{z}_{1:T} && \left(\text{Multiply by } 1 = \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right) \\
&= \log \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\frac{p(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right] && (\text{Definition of Expectation}) \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right] && (\text{Apply Jensen's Inequality})
\end{aligned}$$

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{x}, \boldsymbol{z}_{1:T})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\right] = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}_{1:T}|\boldsymbol{x})}\left[\log\frac{p(\boldsymbol{z}_T)p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)}{q_{\boldsymbol{\phi}}(\boldsymbol{z}_1|\boldsymbol{x})\prod_{t=2}^{T} q_{\boldsymbol{\phi}}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1})}\right]$$

Variational Diffusion Models

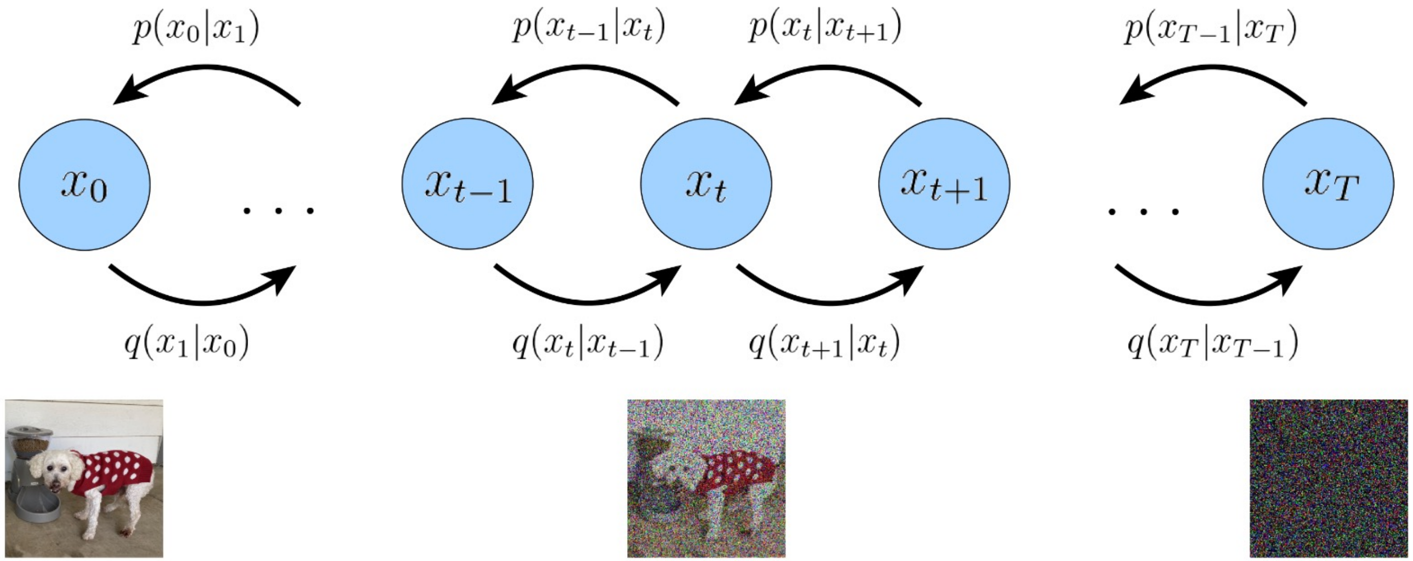
A Variational Diffusion Model (VDM) is simply as a Markovian Hierarchical Variational Autoencoder with three key restrictions:

The latent dimension is exactly equal to the data dimension.

The structure of the latent encoder at each timestep is not learned; it is pre-defined as a linear Gaussian model. In other words, it is a Gaussian distribution centered around the output of the previous timestep.

The Gaussian parameters of the latent encoders vary over time in such a way that the distribution of the latent at final timestep T is a standard Gaussian.

Variational Diffusion Models

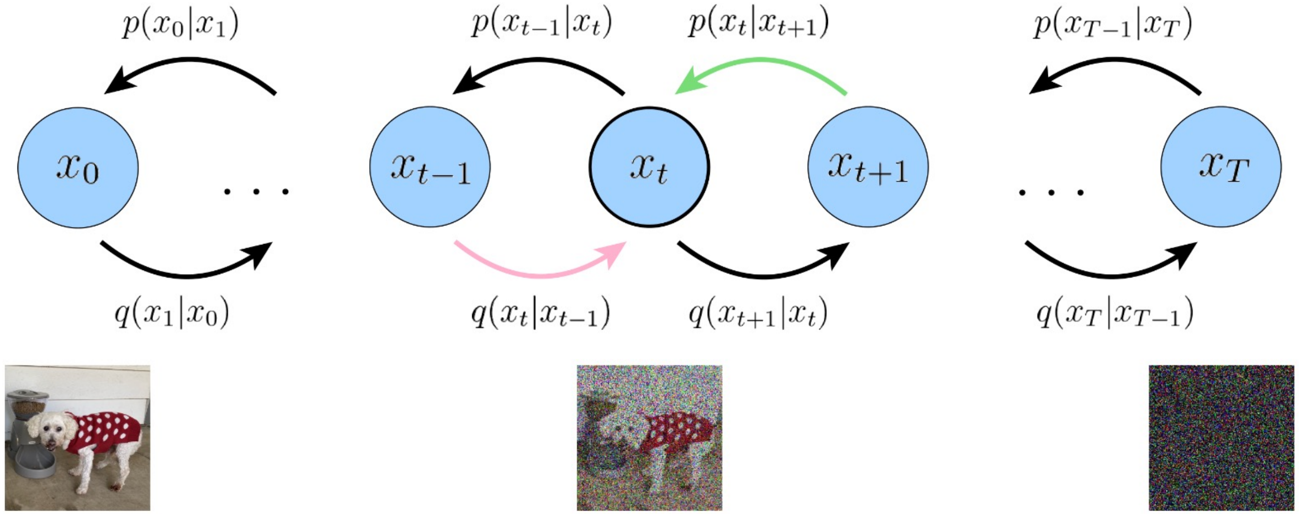


A visual representation of a Variational Diffusion Model; x_0 represents true data observations such as natural images, x_T represents pure Gaussian noise, and x_t is an intermediate noisy version of x_0. Each $q(x_t|x_{t-1})$ is modeled as a Gaussian distribution that uses the output of the previous state as its mean.

Maximising the ELBO for VDM

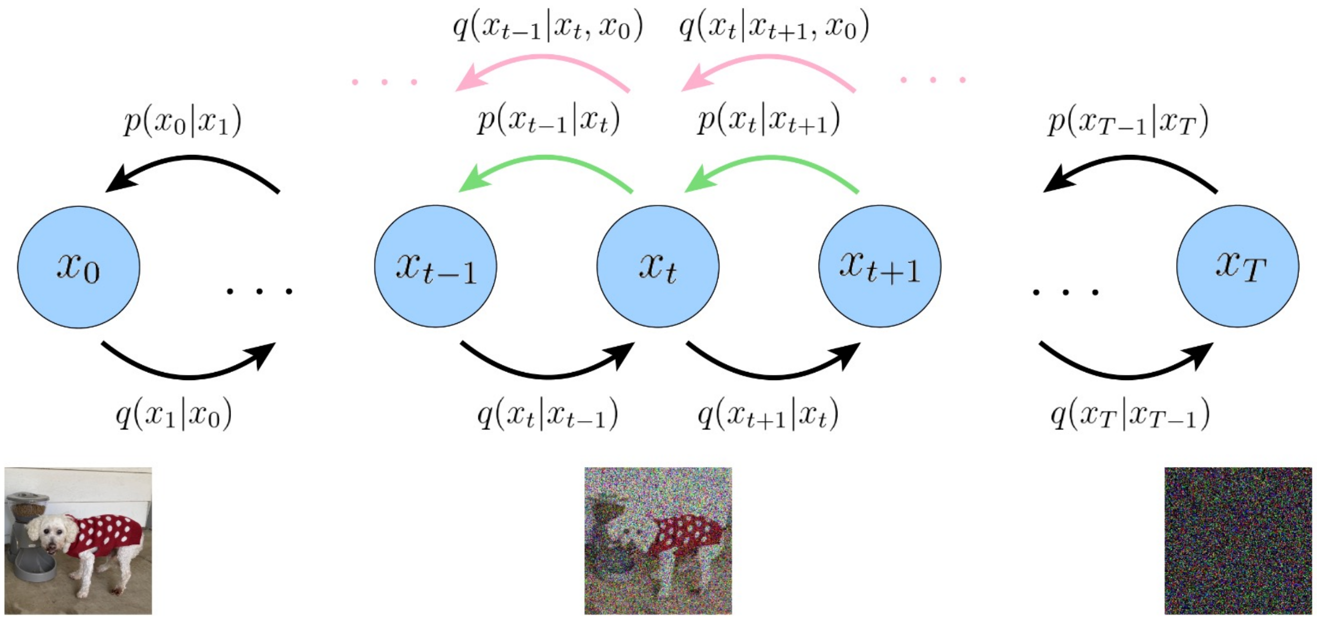
$$
\begin{aligned}
\log p(\boldsymbol{x}) &= \log \int p(\boldsymbol{x}_{0:T}) d\boldsymbol{x}_{1:T} \\
&= \log \int \frac{p(\boldsymbol{x}_{0:T}) q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} d\boldsymbol{x}_{1:T} \\
&= \log \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \right] \\
&\geq \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_{0:T})}{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{\prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \prod_{t=1}^{T-1} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \prod_{t=1}^{T-1} p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \prod_{t=1}^{T-1} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T) p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \prod_{t=1}^{T-1} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\sum_{t=1}^{T-1} \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} \left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right] + \mathbb{E}_{q(\boldsymbol{x}_{T-1}, \boldsymbol{x}_T|\boldsymbol{x}_0)} \left[\log \frac{p(\boldsymbol{x}_T)}{q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1})} \right] + \sum_{t=1}^{T-1} \mathbb{E}_{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_{t+1}|\boldsymbol{x}_0)} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})}{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})} \right] \\
&= \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)} \left[\log p_{\theta}(\boldsymbol{x}_0|\boldsymbol{x}_1) \right]}_{\text{reconstruction term}} - \underbrace{\mathbb{E}_{q(\boldsymbol{x}_{T-1}|\boldsymbol{x}_0)} \left[D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_{T-1}) \parallel p(\boldsymbol{x}_T)) \right]}_{\text{prior matching term}} \\
&\qquad - \sum_{t=1}^{T-1} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_{t-1}, \boldsymbol{x}_{t+1}|\boldsymbol{x}_0)} \left[D_{\mathrm{KL}}(q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) \parallel p_{\theta}(\boldsymbol{x}_t|\boldsymbol{x}_{t+1})) \right]}_{\text{consistency term}}
\end{aligned}
$$

Variational Diffusion Models



a VDM can be optimized by ensuring that for every intermediate x_t, the posterior from the latent above it pθ $(x_t|x_{t+1})$ matches the Gaussian corruption of the latent before it $q(x_t|x_{t-1})$. In this figure, for each intermediate x_t, we minimize the difference between the distributions represented by the pink and green arrows.

Variational Diffusion Models

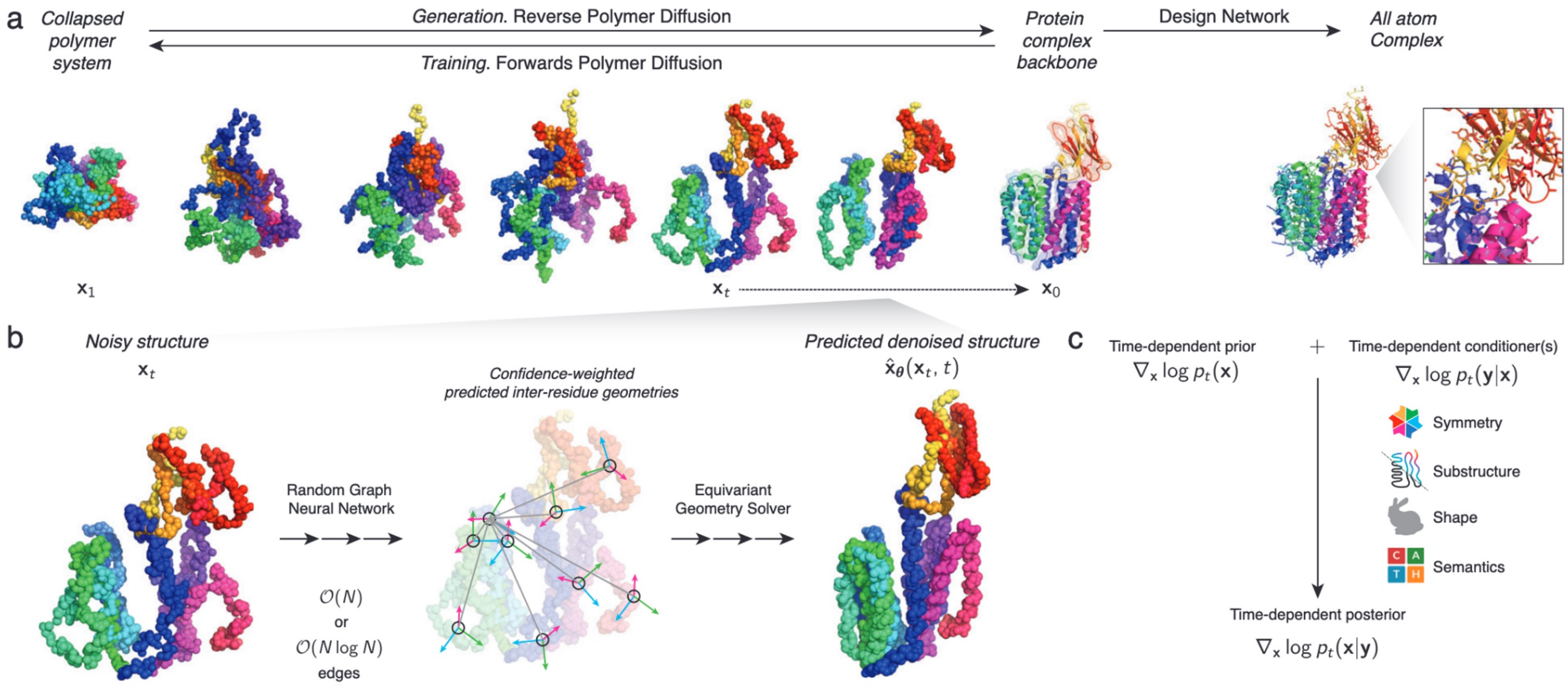


Depicted is an alternate, lower-variance method to optimize a VDM; we compute the form of ground-truth denoising step $q(x_{t-1}|x_t, x_0)$ using Bayes rule, and minimize its KL Divergence with our approximate denoising step pθ $(x_{t-1}|x_t)$. This is once again denoted visually by matching the distributions represented by the green arrows with those of the pink arrows. Artistic liberty is at play here; in the full picture, each pink arrow must also stem from x_0, as it is also a conditioning term.

Variational Diffusion Models

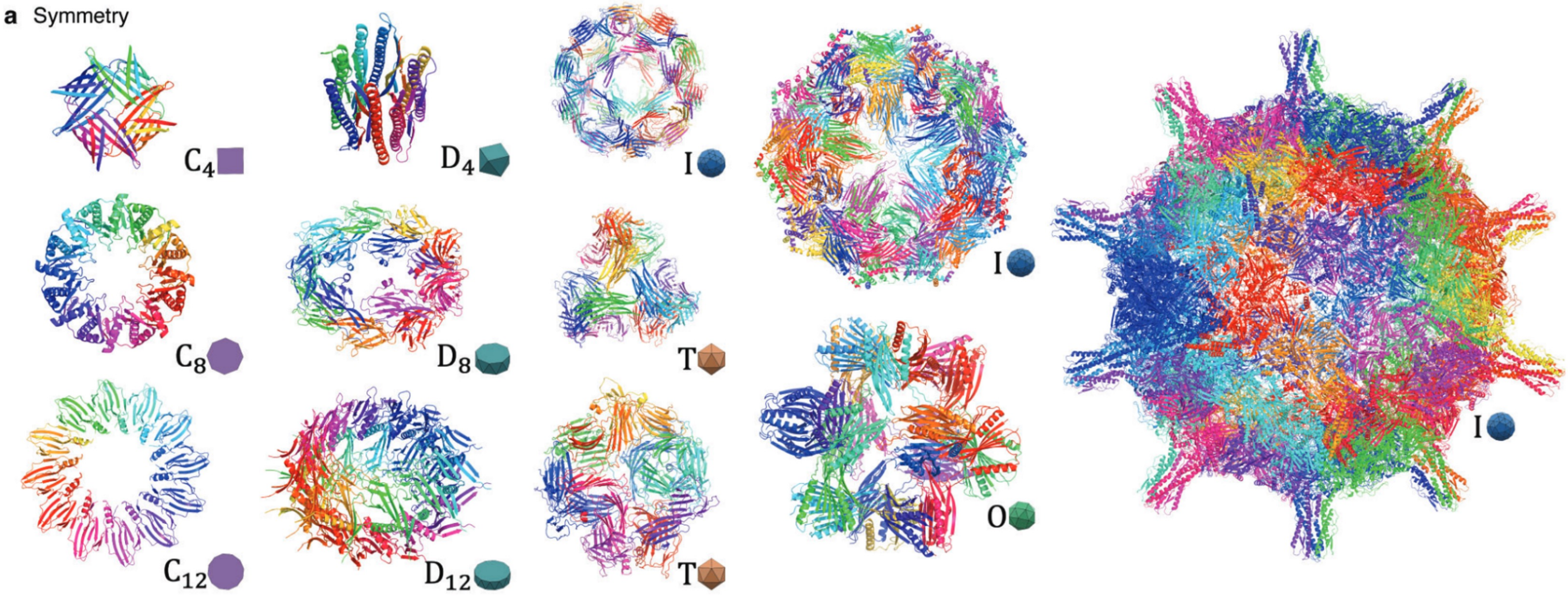
- Variational Diffusion Models as a special case of a Markovian Hierarchical Variational Autoencoder, where three key assumptions enable tractable computation and scalable optimization of the ELBO.
- A VDM boils down to learning a neural network to predict one of three potential objectives:
  - the original source image from any arbitrary noisification of it,
  - the original source noise from any arbitrarily noisified image, or
  - the score function of a noisified image at any arbitrary noise level.

Chroma and RFDiffusion: Diffusion Models for Protein Design


a
Collapsed polymer system
Generation. Reverse Polymer Diffusion
Training. Forwards Polymer Diffusion
Protein complex backbone
Design Network
All atom Complex
$\mathbf{x}_1$
$\mathbf{x}_t$
$\mathbf{x}_0$
b
Noisy structure
$\mathbf{x}_t$
Random Graph Neural Network
$\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ edges
Confidence-weighted predicted inter-residue geometries
Equivariant Geometry Solver
Predicted denoised structure
$\hat{\mathbf{x}}_\theta(\mathbf{x}_t, t)$
c
Time-dependent prior
$\nabla_\mathbf{x} \log p_t(\mathbf{x})$
+
Time-dependent conditioner(s)
$\nabla_\mathbf{x} \log p_t(\mathbf{y}|\mathbf{x})$
Symmetry
Substructure
Shape
C A T H
Semantics
Time-dependent posterior
$\nabla_\mathbf{x} \log p_t(\mathbf{x}|\mathbf{y})$

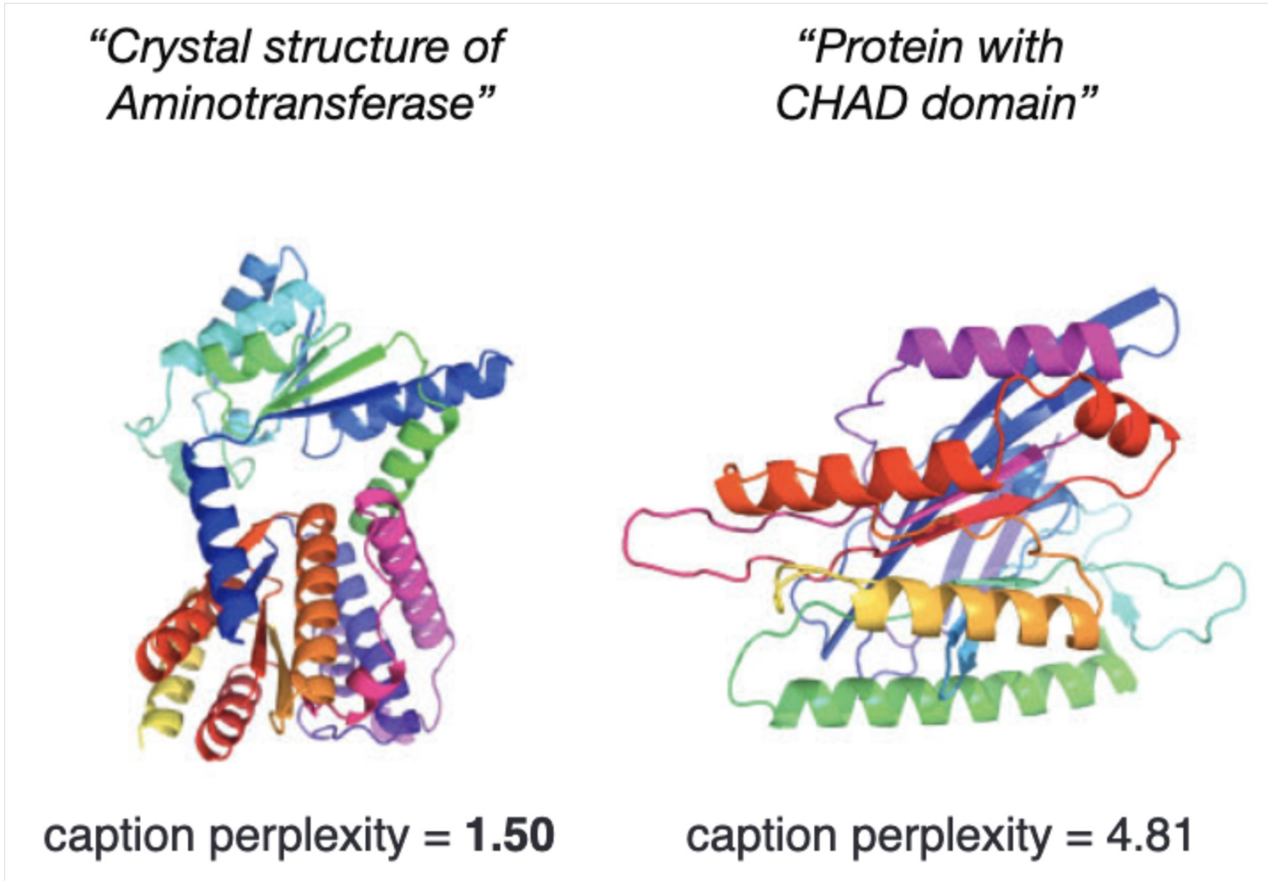
https://www.biorxiv.org/content/10.1101/2022.12.01.518682v1?rss=1

Chroma: Design based on symmetry



https://www.biorxiv.org/content/10.1101/2022.12.01.518682v1?rss=1

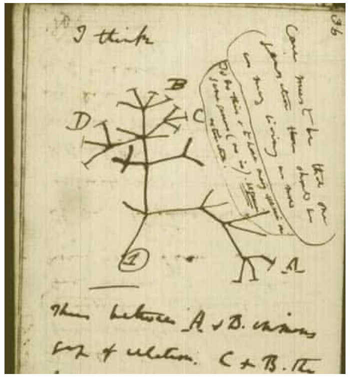
Chroma: Condition based text (i.e. DALLE-x for proteins)


"Crystal structure of Aminotransferase"
"Protein with CHAD domain"
caption perplexity = **1.50**
caption perplexity = 4.81

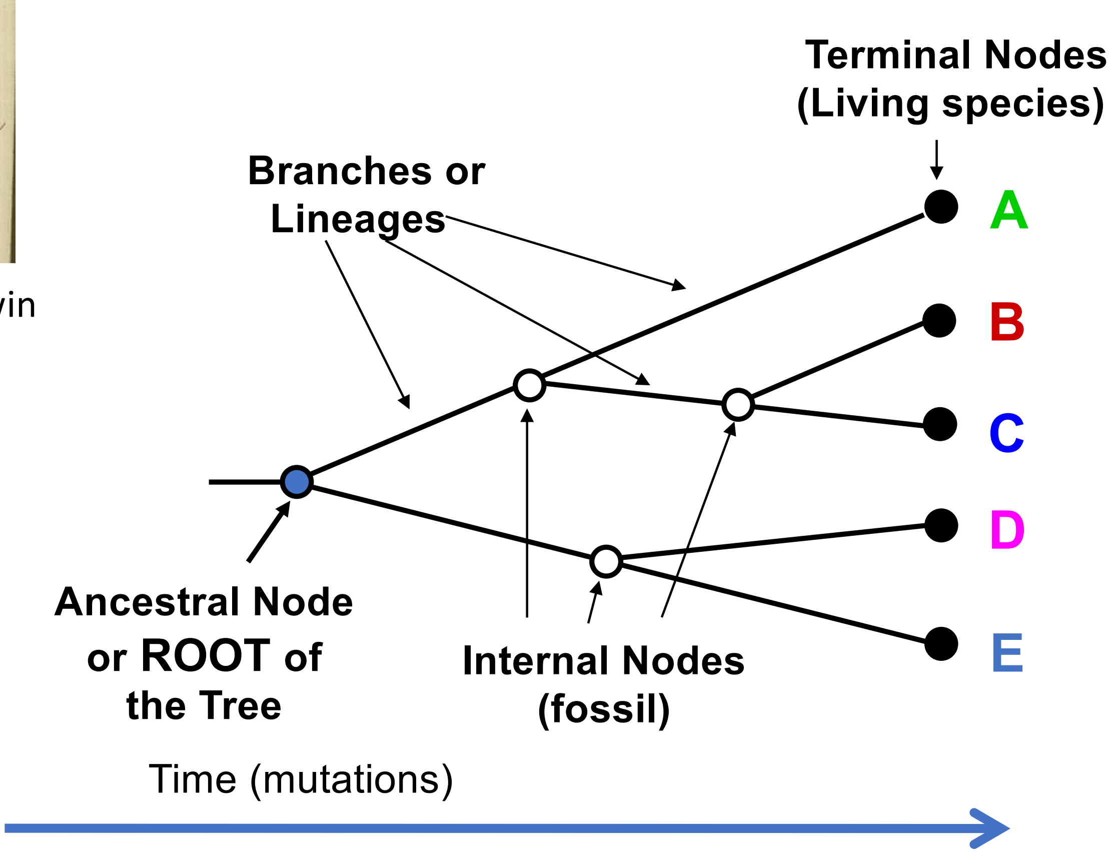
https://www.biorxiv.org/content/10.1101/2022.12.01.518682v1?rss=1

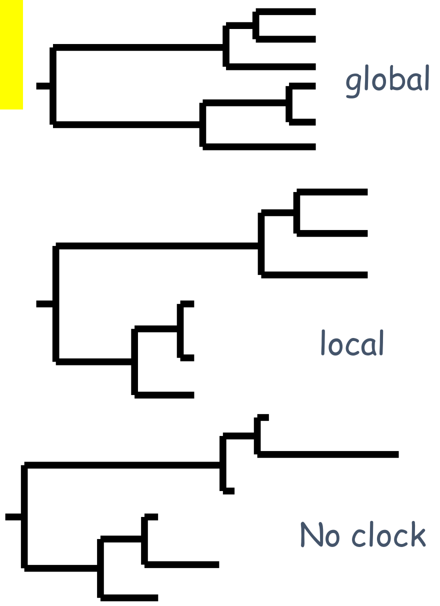
Protein Evolution

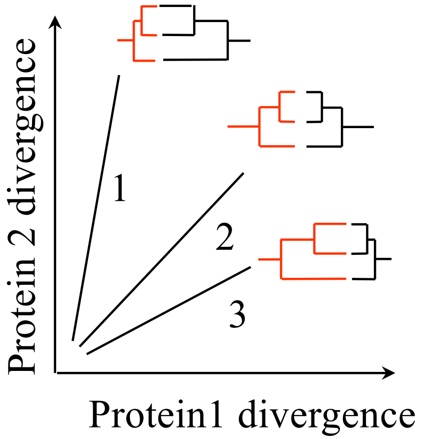
Nothing in Biology Makes Sense Except in the Light of Evolution (Dobzhansky, 1964)



species tree by Darwin

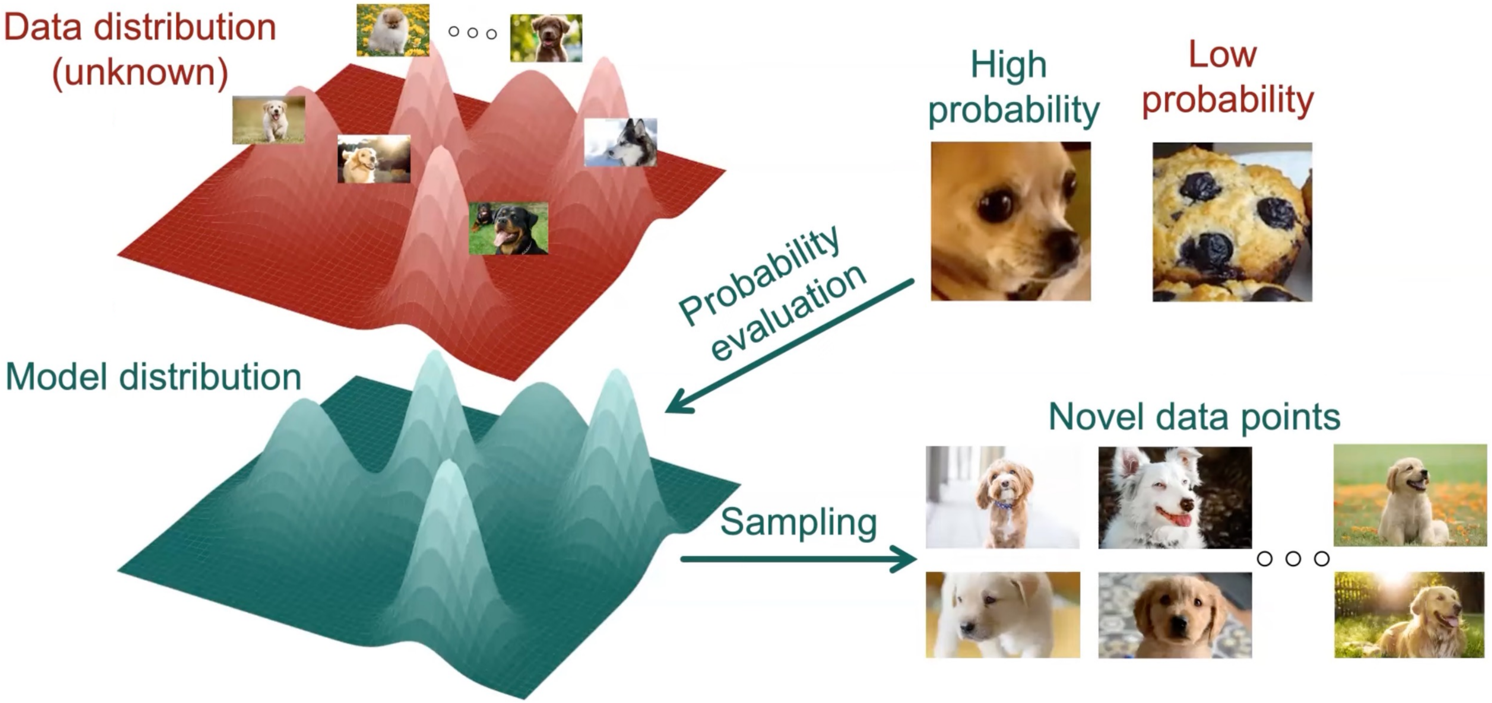




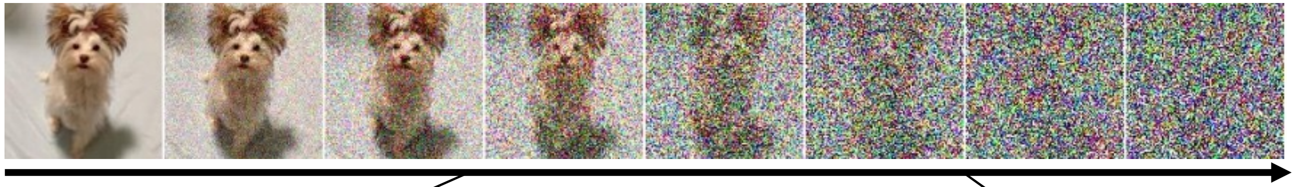


Pietro Lio' and Nick Goldman Models of Molecular Evolution and Phylogeny

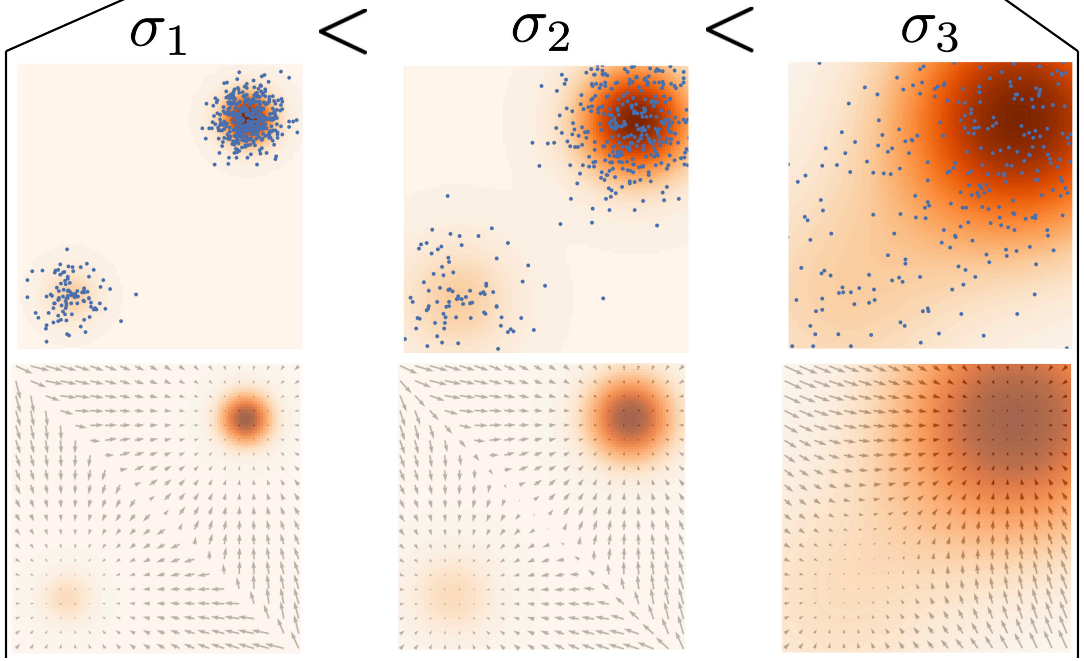
Generative Modelling

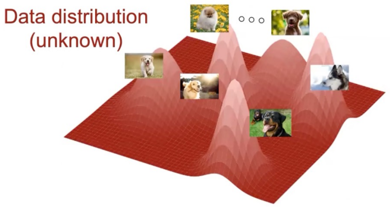


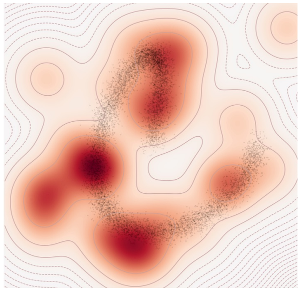
(Credit: Yang Song)



add more and more noise (noise scale σ)

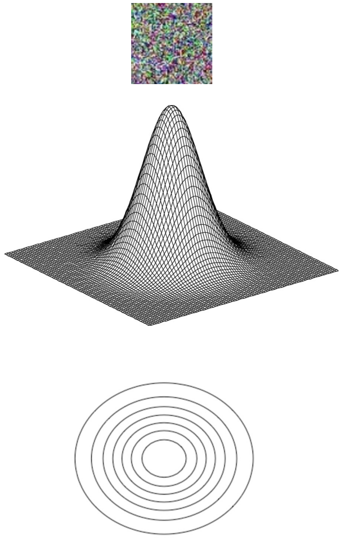






Data distribution
(fully denoised)

Hard to sample from.

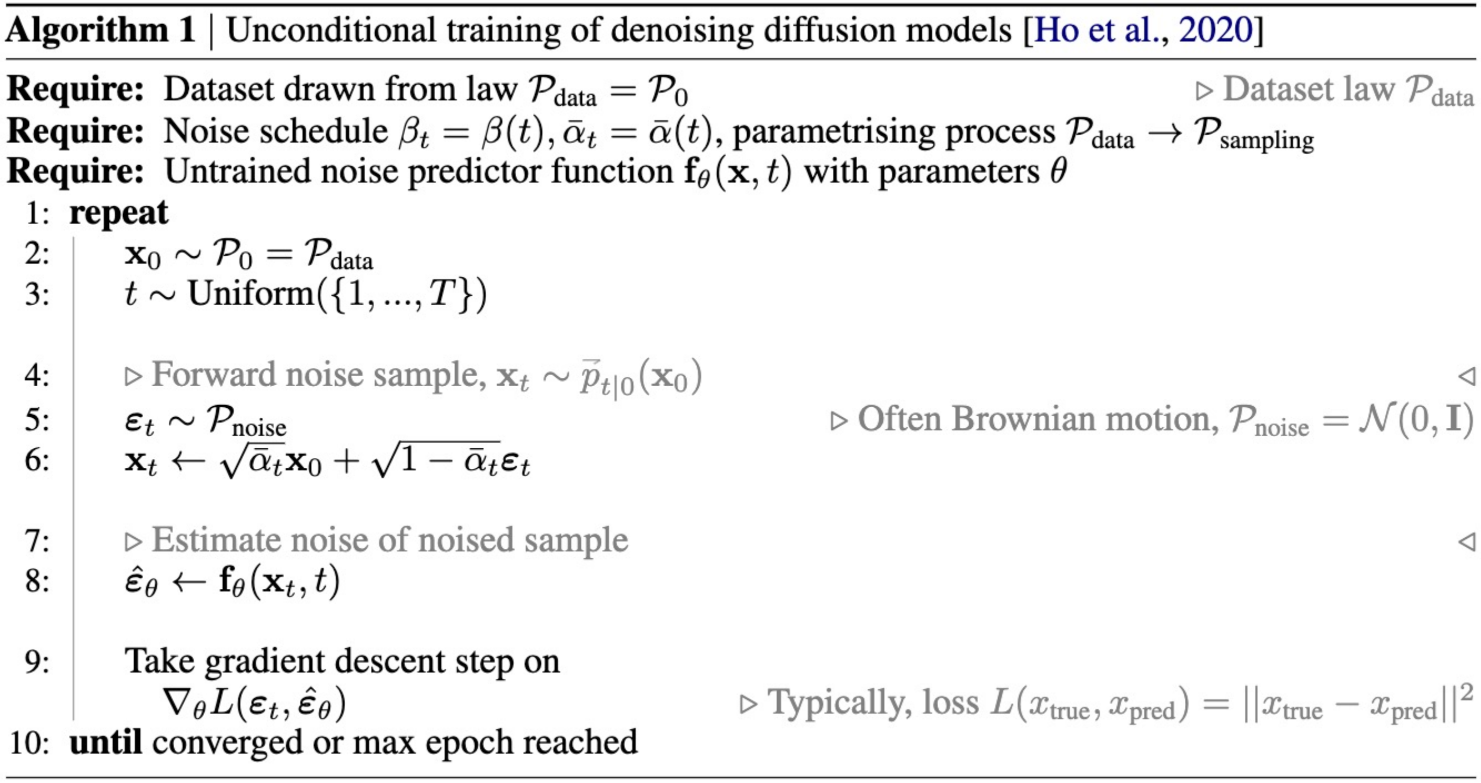


Sampling distribution
(fully noised)

Often a unit Gaussian.

Easy to sample from

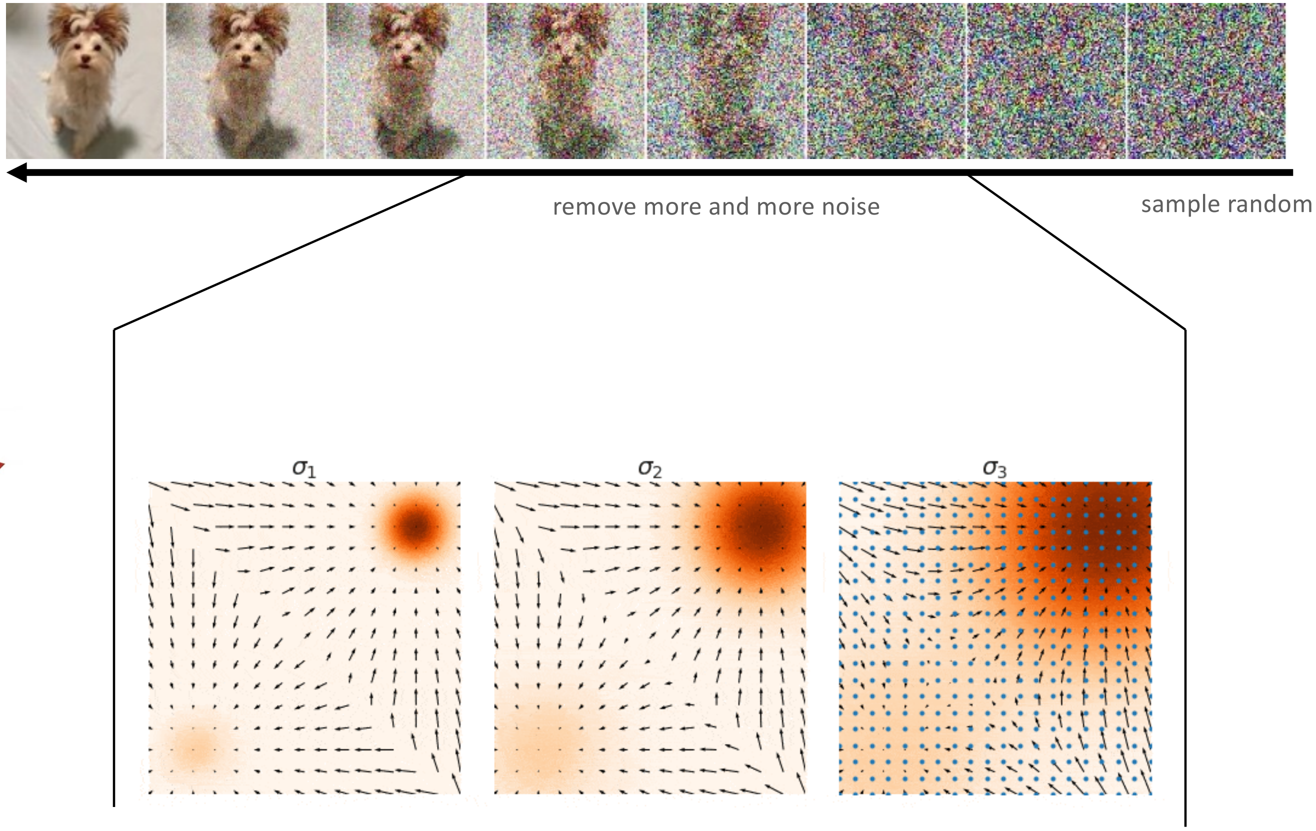
Source: Generative Modeling by Estimating Gradients of the Data Distribution | Yang Song (yang-song.net)



Algorithm 1 | Unconditional training of denoising diffusion models [Ho et al., 2020]

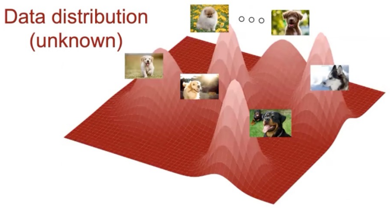
```
Require: Dataset drawn from law $\mathcal{P}_{\text{data}} = \mathcal{P}_0$    ▷ Dataset law $\mathcal{P}_{\text{data}}$
Require: Noise schedule $\beta_t = \beta(t), \bar{\alpha}_t = \bar{\alpha}(t)$, parametrising process $\mathcal{P}_{\text{data}} \to \mathcal{P}_{\text{sampling}}$
Require: Untrained noise predictor function $\mathbf{f}_\theta(\mathbf{x}, t)$ with parameters $\theta$
repeat
    $\mathbf{x}_0 \sim \mathcal{P}_0 = \mathcal{P}_{\text{data}}$
    $t \sim \text{Uniform}(\{1, ..., T\})$

    ▷ Forward noise sample, $\mathbf{x}_t \sim \vec{p}_{t|0}(\mathbf{x}_0)$    ◁
    $\boldsymbol{\varepsilon}_t \sim \mathcal{P}_{\text{noise}}$    ▷ Often Brownian motion, $\mathcal{P}_{\text{noise}} = \mathcal{N}(0, \mathbf{I})$
    $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}_t$

    ▷ Estimate noise of noised sample    ◁
    $\hat{\boldsymbol{\varepsilon}}_\theta \leftarrow \mathbf{f}_\theta(\mathbf{x}_t, t)$

    Take gradient descent step on
        $\nabla_\theta L(\boldsymbol{\varepsilon}_t, \hat{\boldsymbol{\varepsilon}}_\theta)$    ▷ Typically, loss $L(x_{\text{true}}, x_{\text{pred}}) = ||x_{\text{true}} - x_{\text{pred}}||^2$
until converged or max epoch reached
```

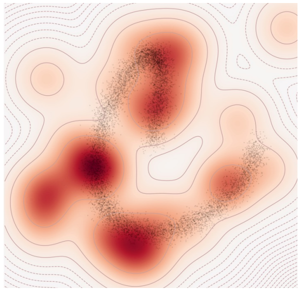
Source: [2312.09236] A framework for conditional diffusion modelling with applications in motif scaffolding for protein design (arxiv.org)



remove more and more noise

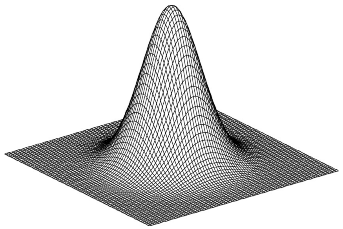
sample random starting point

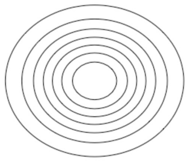




Data distribution
(fully denoised)

Hard to sample from.



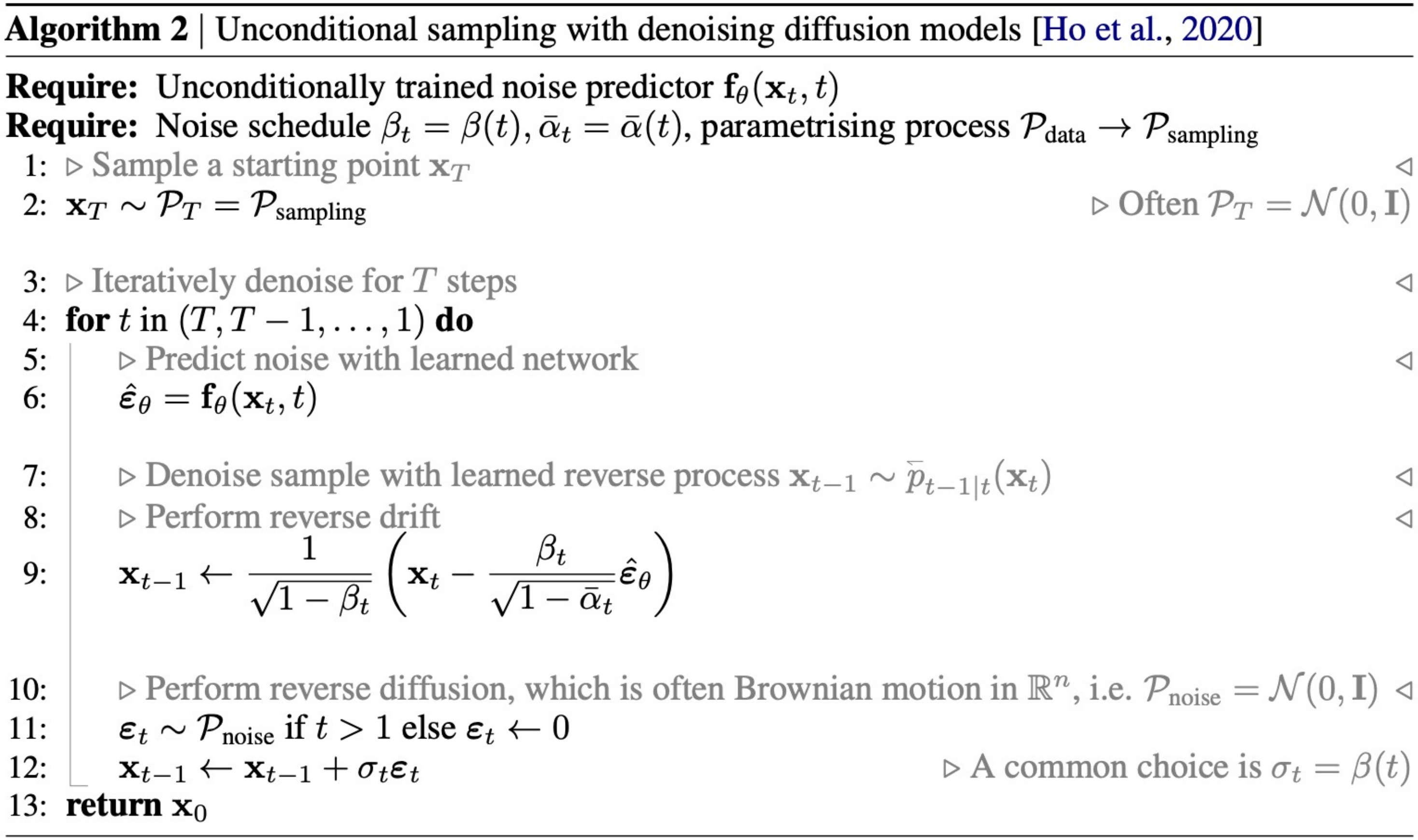


Sampling distribution
(fully noised)

Often a unit Gaussian.

Easy to sample from

Source: [Generative Modeling by Estimating Gradients of the Data Distribution | Yang Song (yang-song.net)](https://yang-song.net)



Algorithm 2 | Unconditional sampling with denoising diffusion models [Ho et al., 2020]

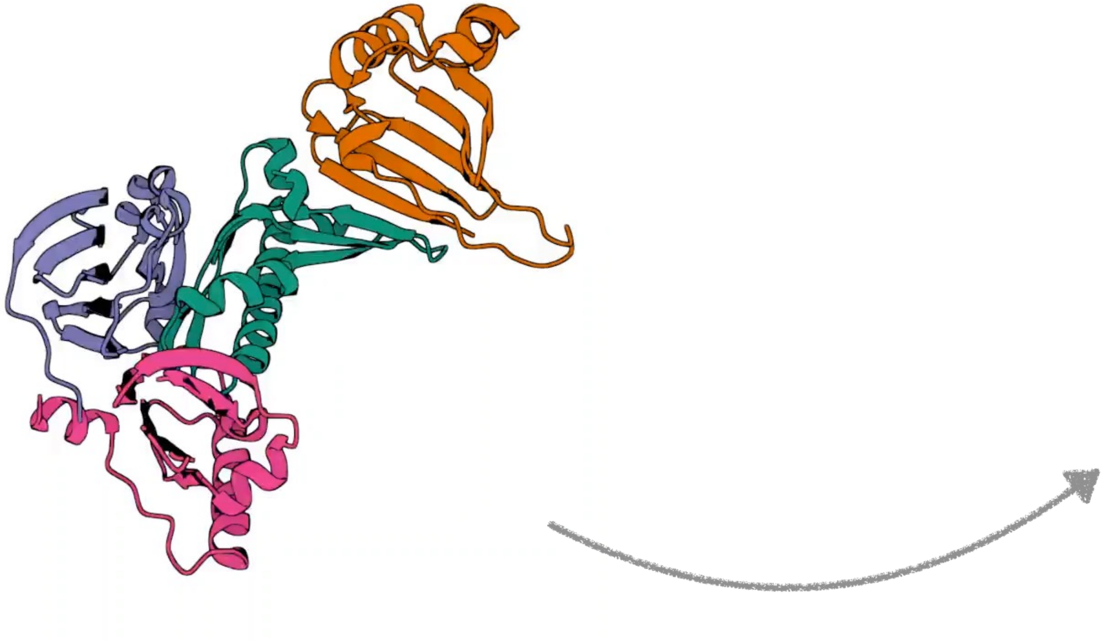
```
Require: Unconditionally trained noise predictor $\mathbf{f}_\theta(\mathbf{x}_t, t)$
Require: Noise schedule $\beta_t = \beta(t), \bar{\alpha}_t = \bar{\alpha}(t)$, parametrising process $\mathcal{P}_{\text{data}} \to \mathcal{P}_{\text{sampling}}$
▷ Sample a starting point $\mathbf{x}_T$ ◁
$\mathbf{x}_T \sim \mathcal{P}_T = \mathcal{P}_{\text{sampling}}$    ▷ Often $\mathcal{P}_T = \mathcal{N}(0, \mathbf{I})$

▷ Iteratively denoise for $T$ steps ◁
for $t$ in $(T, T-1, \dots, 1)$ do
    ▷ Predict noise with learned network ◁
    $\hat{\varepsilon}_\theta = \mathbf{f}_\theta(\mathbf{x}_t, t)$

    ▷ Denoise sample with learned reverse process $\mathbf{x}_{t-1} \sim \overleftarrow{p}_{t-1|t}(\mathbf{x}_t)$ ◁
    ▷ Perform reverse drift ◁
    $\mathbf{x}_{t-1} \leftarrow \frac{1}{\sqrt{1-\beta_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \hat{\varepsilon}_\theta \right)$

    ▷ Perform reverse diffusion, which is often Brownian motion in $\mathbb{R}^n$, i.e. $\mathcal{P}_{\text{noise}} = \mathcal{N}(0, \mathbf{I})$ ◁
    $\varepsilon_t \sim \mathcal{P}_{\text{noise}}$ if $t > 1$ else $\varepsilon_t \leftarrow 0$
    $\mathbf{x}_{t-1} \leftarrow \mathbf{x}_{t-1} + \sigma_t \varepsilon_t$    ▷ A common choice is $\sigma_t = \beta(t)$
return $\mathbf{x}_0$
```

Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023

Source: [2312.09236] A framework for conditional diffusion modelling with applications in motif scaffolding for protein design (arxiv.org)

Diffusion models for protein design

Enable “controllable” design of proteins for many properties

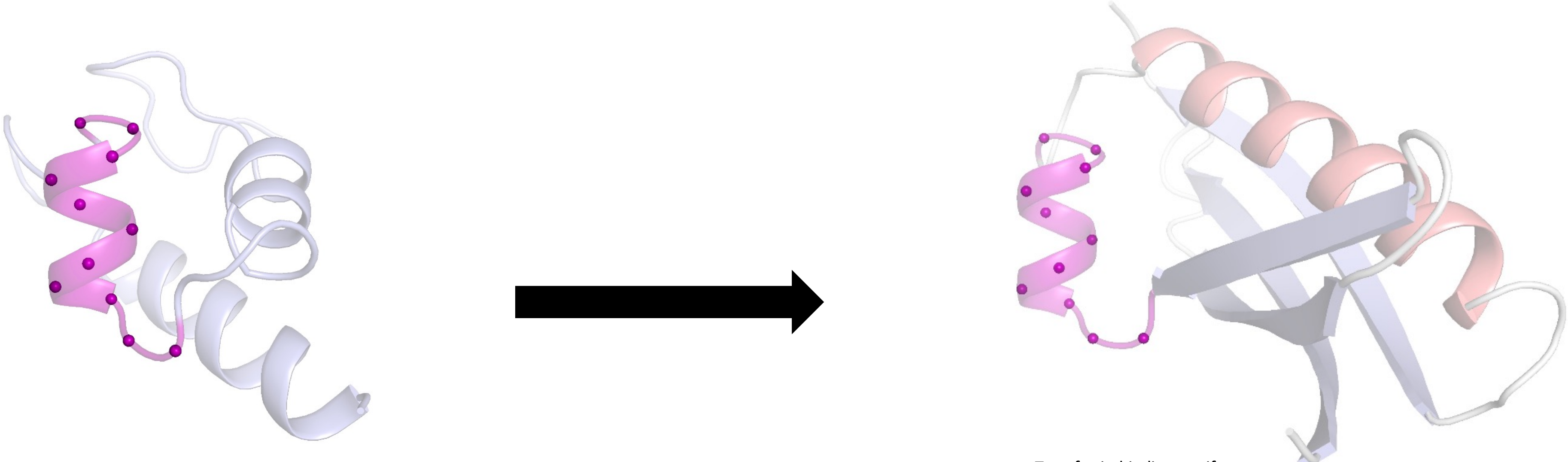


- Symmetry
- Substructure
- Shape
- Semantics
- Dynamical properties

Source: Watson et al. 2023, Ingraham et al. 2022, generatebiomedicines.com

Tackling Motif Scaffolding

Scaffold a certain motif into diverse backbones



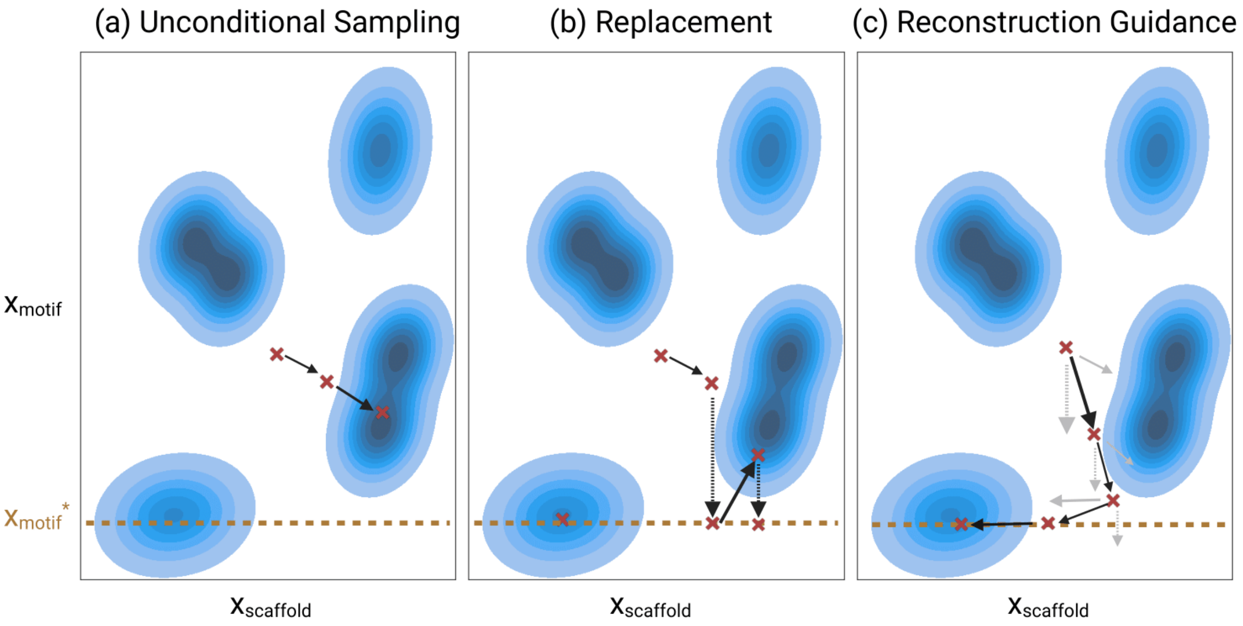
Transferrin-binding motif
6E6R

Transferrin-binding motif
+ new scaffold

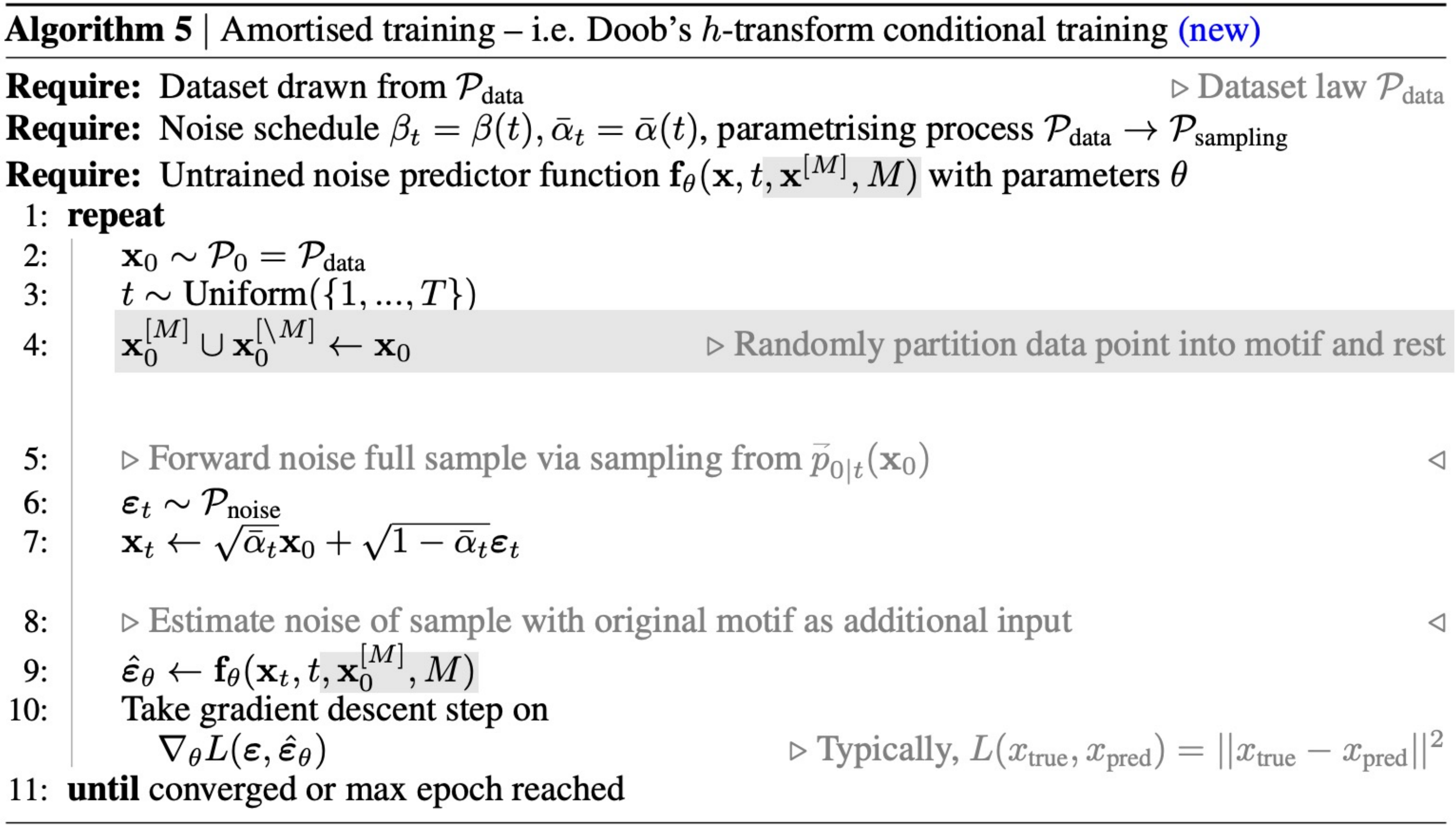
Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023

Many ways to condition diffusion models

Tug-of-war between unconditional and conditional update



Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023



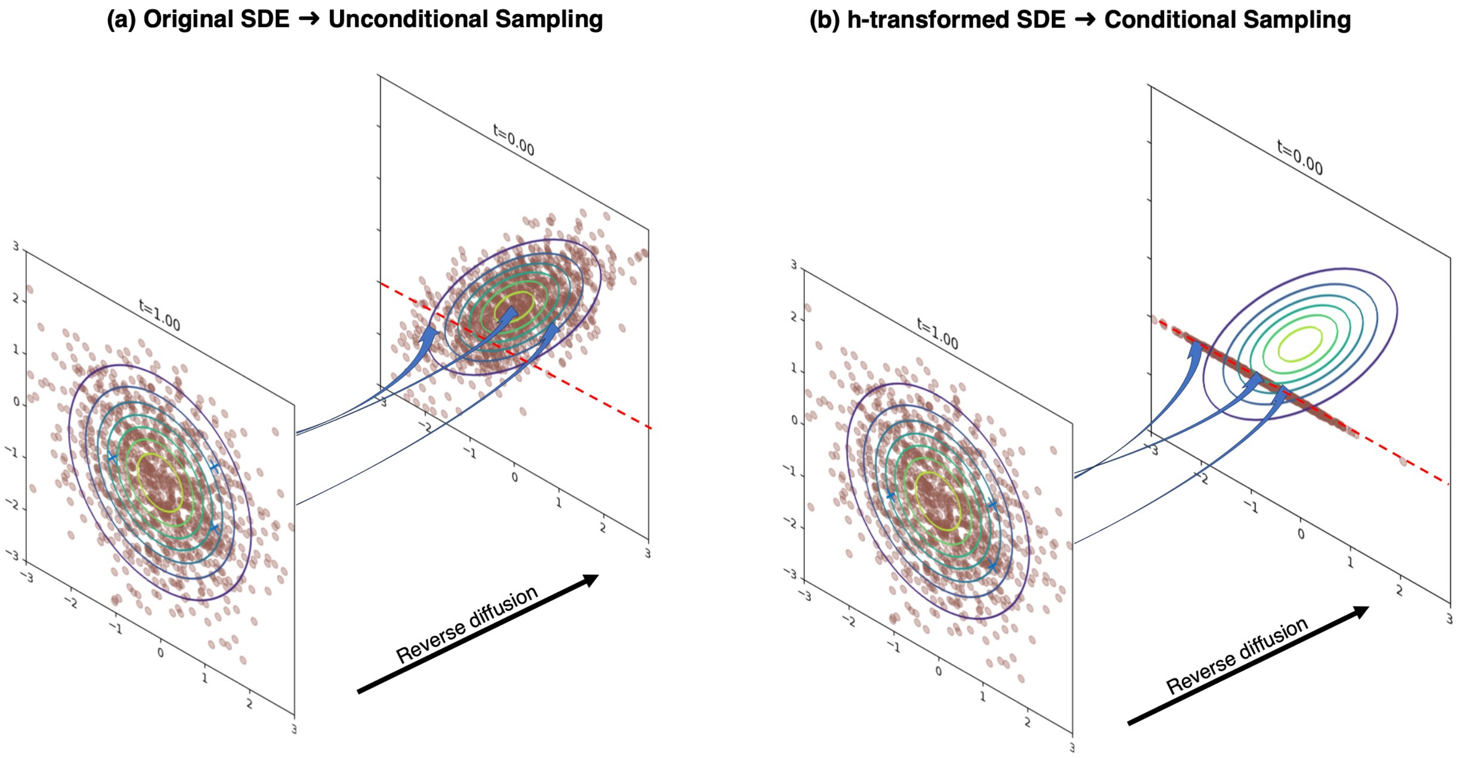
```
Algorithm 5 | Amortised training – i.e. Doob's $h$-transform conditional training (new)
Require: Dataset drawn from $\mathcal{P}_{\text{data}}$    ▷ Dataset law $\mathcal{P}_{\text{data}}$
Require: Noise schedule $\beta_t = \beta(t), \bar{\alpha}_t = \bar{\alpha}(t)$, parametrising process $\mathcal{P}_{\text{data}} \rightarrow \mathcal{P}_{\text{sampling}}$
Require: Untrained noise predictor function $\mathbf{f}_\theta(\mathbf{x}, t, \mathbf{x}^{[M]}, M)$ with parameters $\theta$
repeat
    $\mathbf{x}_0 \sim \mathcal{P}_0 = \mathcal{P}_{\text{data}}$
    $t \sim \text{Uniform}(\{1, ..., T\})$
    $\mathbf{x}_0^{[M]} \cup \mathbf{x}_0^{[\setminus M]} \leftarrow \mathbf{x}_0$    ▷ Randomly partition data point into motif and rest

    ▷ Forward noise full sample via sampling from $\vec{p}_{0|t}(\mathbf{x}_0)$    ◁
    $\boldsymbol{\varepsilon}_t \sim \mathcal{P}_{\text{noise}}$
    $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\varepsilon}_t$

    ▷ Estimate noise of sample with original motif as additional input    ◁
    $\hat{\boldsymbol{\varepsilon}}_\theta \leftarrow \mathbf{f}_\theta(\mathbf{x}_t, t, \mathbf{x}_0^{[M]}, M)$
    Take gradient descent step on
        $\nabla_\theta L(\boldsymbol{\varepsilon}, \hat{\boldsymbol{\varepsilon}}_\theta)$    ▷ Typically, $L(x_{\text{true}}, x_{\text{pred}}) = ||x_{\text{true}} - x_{\text{pred}}||^2$
until converged or max epoch reached
```

Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023

Source: [2312.09236] A framework for conditional diffusion modelling with applications in motif scaffolding for protein design (arxiv.org)

h-Transforming SDE = Conditional Sampling

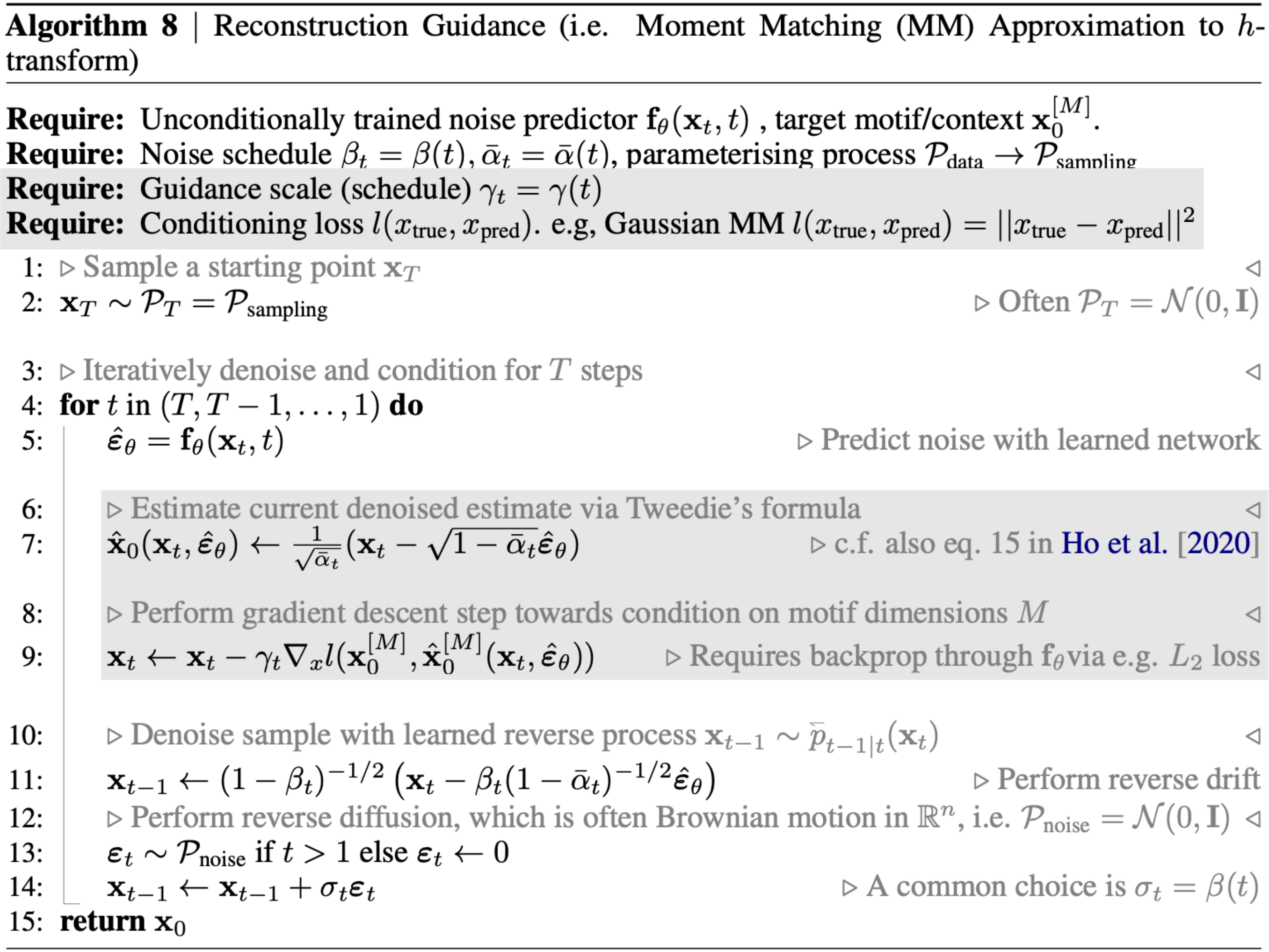
General framework to enforce various conditioning constraints



(a) Original SDE → Unconditional Sampling

(b) h-transformed SDE → Conditional Sampling

Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023



Algorithm 8 | Reconstruction Guidance (i.e. Moment Matching (MM) Approximation to h-transform)

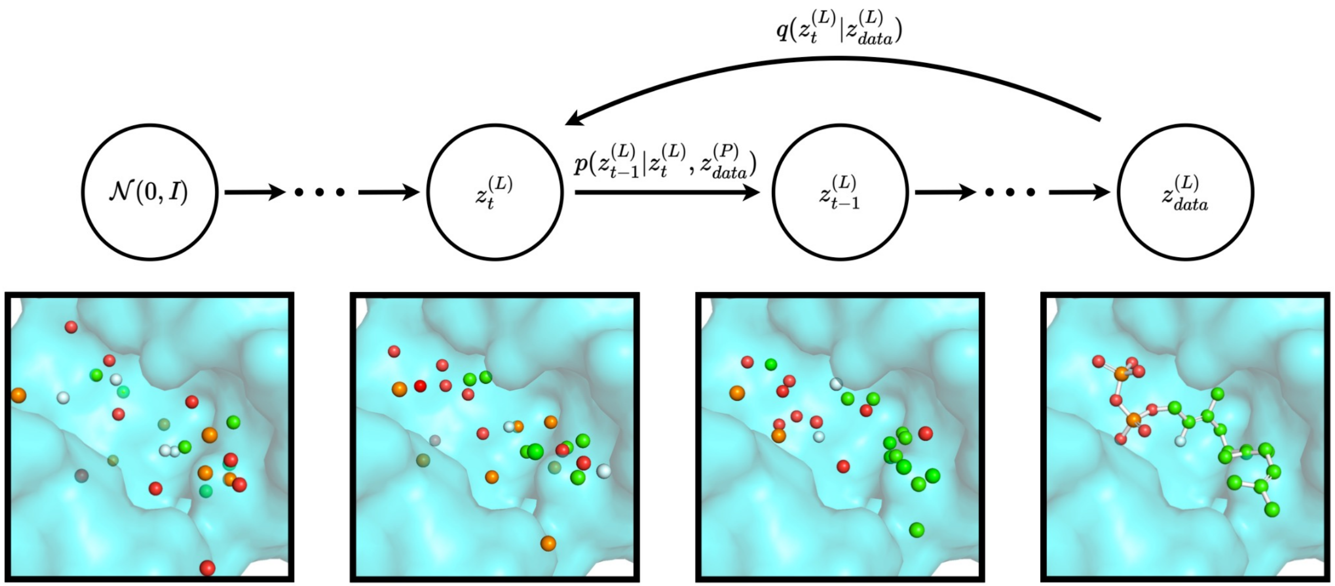
```
Require: Unconditionally trained noise predictor $\mathbf{f}_\theta(\mathbf{x}_t, t)$ , target motif/context $\mathbf{x}_0^{[M]}$.
Require: Noise schedule $\beta_t = \beta(t), \bar{\alpha}_t = \bar{\alpha}(t)$, parameterising process $\mathcal{P}_{\text{data}} \to \mathcal{P}_{\text{sampling}}$
Require: Guidance scale (schedule) $\gamma_t = \gamma(t)$
Require: Conditioning loss $l(x_{\text{true}}, x_{\text{pred}})$. e.g, Gaussian MM $l(x_{\text{true}}, x_{\text{pred}}) = ||x_{\text{true}} - x_{\text{pred}}||^2$
▷ Sample a starting point $\mathbf{x}_T$ ◁
$\mathbf{x}_T \sim \mathcal{P}_T = \mathcal{P}_{\text{sampling}}$    ▷ Often $\mathcal{P}_T = \mathcal{N}(0, \mathbf{I})$

▷ Iteratively denoise and condition for $T$ steps ◁
for $t$ in $(T, T-1, \ldots, 1)$ do
    $\hat{\varepsilon}_\theta = \mathbf{f}_\theta(\mathbf{x}_t, t)$    ▷ Predict noise with learned network

    ▷ Estimate current denoised estimate via Tweedie's formula ◁
    $\hat{\mathbf{x}}_0(\mathbf{x}_t, \hat{\varepsilon}_\theta) \leftarrow \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\hat{\varepsilon}_\theta)$    ▷ c.f. also eq. 15 in Ho et al. [2020]

    ▷ Perform gradient descent step towards condition on motif dimensions $M$ ◁
    $\mathbf{x}_t \leftarrow \mathbf{x}_t - \gamma_t \nabla_x l(\mathbf{x}_0^{[M]}, \hat{\mathbf{x}}_0^{[M]}(\mathbf{x}_t, \hat{\varepsilon}_\theta))$    ▷ Requires backprop through $\mathbf{f}_\theta$via e.g. $L_2$ loss

    ▷ Denoise sample with learned reverse process $\mathbf{x}_{t-1} \sim \overline{p}_{t-1|t}(\mathbf{x}_t)$ ◁
    $\mathbf{x}_{t-1} \leftarrow (1 - \beta_t)^{-1/2} \left(\mathbf{x}_t - \beta_t(1 - \bar{\alpha}_t)^{-1/2}\hat{\varepsilon}_\theta\right)$    ▷ Perform reverse drift
    ▷ Perform reverse diffusion, which is often Brownian motion in $\mathbb{R}^n$, i.e. $\mathcal{P}_{\text{noise}} = \mathcal{N}(0, \mathbf{I})$ ◁
    $\varepsilon_t \sim \mathcal{P}_{\text{noise}}$ if $t > 1$ else $\varepsilon_t \leftarrow 0$
    $\mathbf{x}_{t-1} \leftarrow \mathbf{x}_{t-1} + \sigma_t \varepsilon_t$    ▷ A common choice is $\sigma_t = \beta(t)$
return $\mathbf{x}_0$
```

Didi*, Vargas*, Mathis*, Dutordoir* et al. NeurIPS AI4D3 2023

Source: [2312.09236] A framework for conditional diffusion modelling with applications in motif scaffolding for protein design (arxiv.org)

DiffSBDD: Diffusion for Structure-based Drug Design with Equivariant Diffusion Models

- Both proteins and ligands are represented as all-atom graphs (with coordinates x and feature {i.e. atoms types} vectors h).
- Our model is trained to predict the transitional probability distribution $p_\theta\left(z_{t-1}^{(L)} \middle| z_t^{(L)}, z_{data}^{(P)}\right)$ which is conditioned both on the previous latent state of the ligand $z_t^{(L)}$ and the fixed presentation of the pocket $z_{data}^{(P)}$.
- In practice, samples are constructed using a denoising network $\hat{\epsilon}_\theta$



$q(z_t^{(L)}|z_{\text{data}}^{(L)})$
$\mathcal{N}(0, I)$
$z_t^{(L)}$
$p(z_{t-1}^{(L)}|z_t^{(L)}, z_{\text{data}}^{(P)})$
$z_{t-1}^{(L)}$
$z_{\text{data}}^{(L)}$

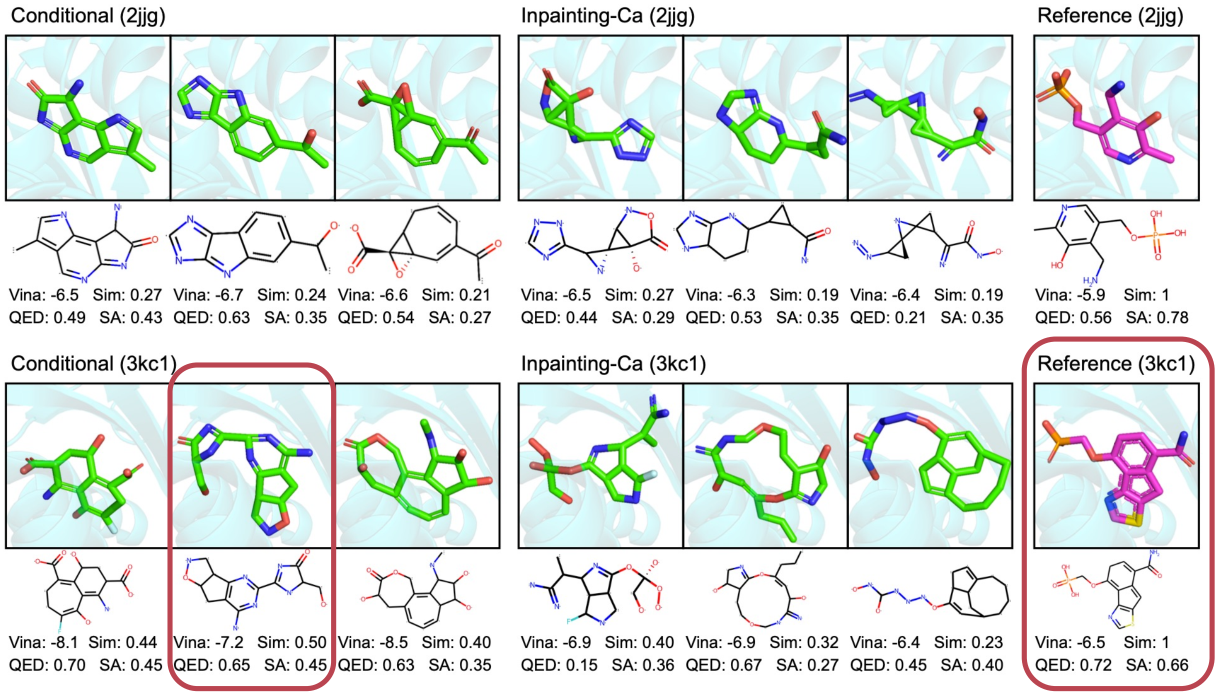
$$\boldsymbol{z}_{\text{data}} = [\boldsymbol{x}, \boldsymbol{h}]$$

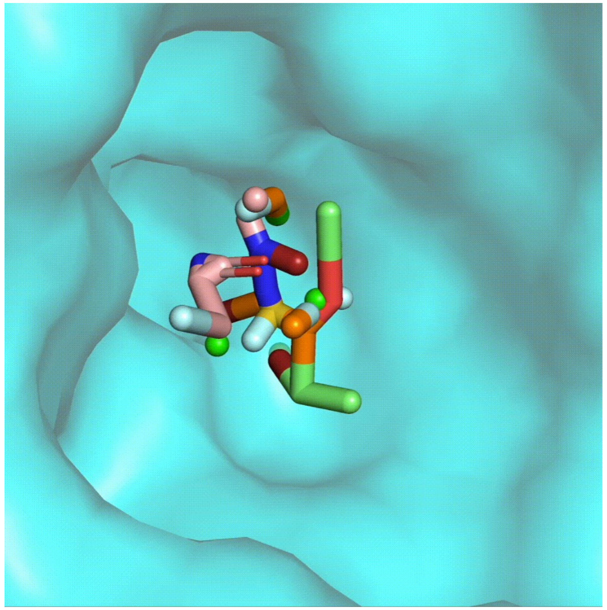
$$\hat{\boldsymbol{\epsilon}}_\theta = \phi_\theta(\boldsymbol{z}_t^{(L)}, \boldsymbol{z}_{\text{data}}^{(P)}, t)$$

SE(3)-equivariant Graph Neural Network

Based on: Schneuing, Arne, et al. "Structure-based drug design with equivariant diffusion models." *NeurIPS MLSB* 2022.

DiffSBDD: Results


Conditional (2jjg)
Inpainting-Ca (2jjg)
Reference (2jjg)
Vina: -6.5 Sim: 0.27 QED: 0.49 SA: 0.43
Vina: -6.7 Sim: 0.24 QED: 0.63 SA: 0.35
Vina: -6.6 Sim: 0.21 QED: 0.54 SA: 0.27
Vina: -6.5 Sim: 0.27 QED: 0.44 SA: 0.29
Vina: -6.3 Sim: 0.19 QED: 0.53 SA: 0.35
Vina: -6.4 Sim: 0.19 QED: 0.21 SA: 0.35
Vina: -5.9 Sim: 1 QED: 0.56 SA: 0.78
Conditional (3kc1)
Inpainting-Ca (3kc1)
Reference (3kc1)
Vina: -8.1 Sim: 0.44 QED: 0.70 SA: 0.45
Vina: -7.2 Sim: 0.50 QED: 0.65 SA: 0.45
Vina: -8.5 Sim: 0.40 QED: 0.63 SA: 0.35
Vina: -6.9 Sim: 0.40 QED: 0.15 SA: 0.36
Vina: -6.9 Sim: 0.32 QED: 0.67 SA: 0.27
Vina: -6.4 Sim: 0.23 QED: 0.45 SA: 0.40
Vina: -6.5 Sim: 1 QED: 0.72 SA: 0.66

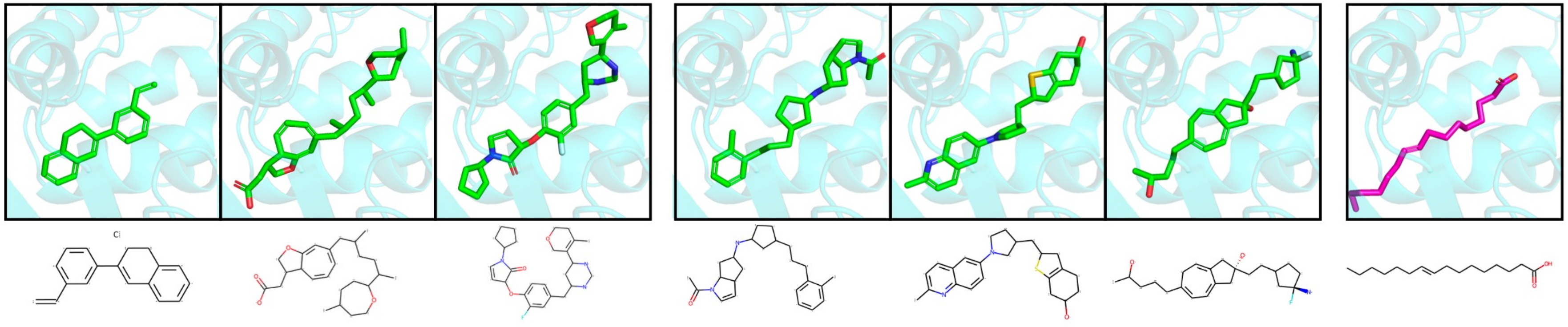


DiffSBDD: Results

Conditional-Ca (6c0b)

Inpainting-Ca (6c0b)

Reference (6c0b)



Vina: -12.8 Sim: 0.05 Vina: -11.9 Sim: 0.12 Vina: -11.5 Sim: 0.06
QED: 0.74 SA: 0.45 QED: 0.66 SA: 0.25 QED: 0.68 SA: 0.25

Vina: -12.4 Sim: 0.07 Vina: -12.3 Sim: 0.07 Vina: -12.2 Sim: 0.12
QED: 0.76 SA: 0.24 QED: 0.85 SA: 0.25 QED: 0.63 SA: 0.34

Vina: -8.40 Sim: 1
QED: 0.36 SA: 0.89

DiffSBDD: Results

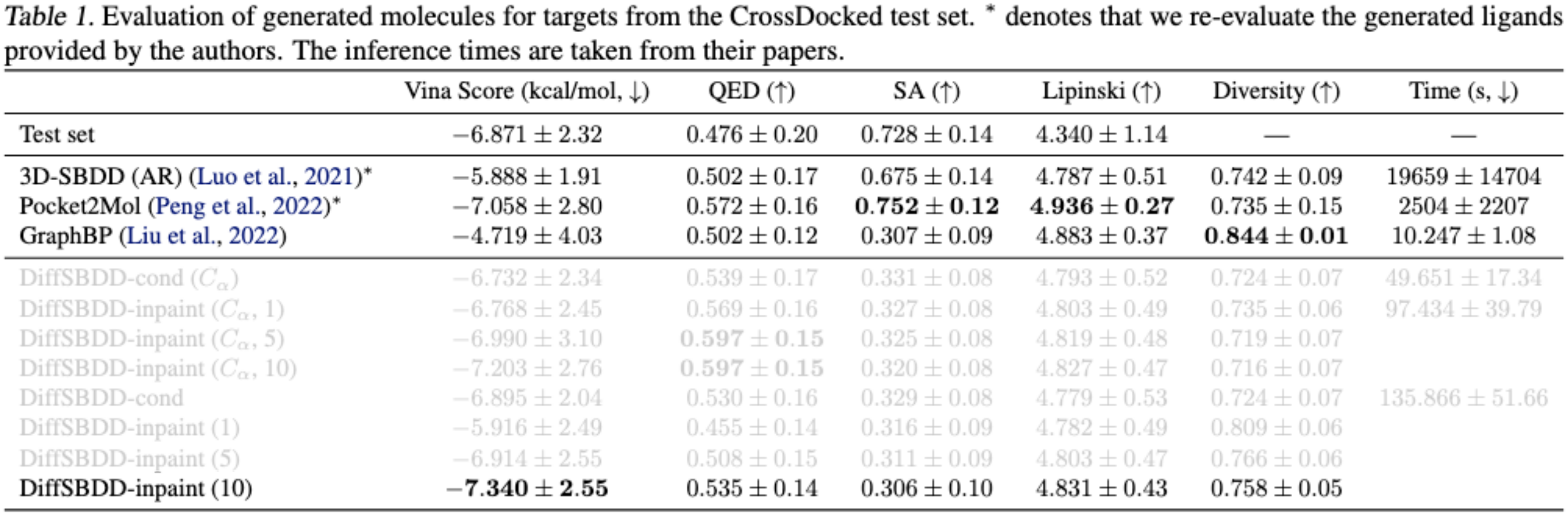


Table 1. Evaluation of generated molecules for targets from the CrossDocked test set. * denotes that we re-evaluate the generated ligands provided by the authors. The inference times are taken from their papers.

	Vina Score (kcal/mol, ↓)	QED (↑)	SA (↑)	Lipinski (↑)	Diversity (↑)	Time (s, ↓)
Test set	-6.871 ± 2.32	0.476 ± 0.20	0.728 ± 0.14	4.340 ± 1.14	—	—
3D-SBDD (AR) (Luo et al., 2021)*	-5.888 ± 1.91	0.502 ± 0.17	0.675 ± 0.14	4.787 ± 0.51	0.742 ± 0.09	19659 ± 14704
Pocket2Mol (Peng et al., 2022)*	-7.058 ± 2.80	0.572 ± 0.16	$\mathbf{0.752 \pm 0.12}$	$\mathbf{4.936 \pm 0.27}$	0.735 ± 0.15	2504 ± 2207
GraphBP (Liu et al., 2022)	-4.719 ± 4.03	0.502 ± 0.12	0.307 ± 0.09	4.883 ± 0.37	$\mathbf{0.844 \pm 0.01}$	10.247 ± 1.08
DiffSBDD-cond (C_α)	-6.732 ± 2.34	0.539 ± 0.17	0.331 ± 0.08	4.793 ± 0.52	0.724 ± 0.07	49.651 ± 17.34
DiffSBDD-inpaint (C_α, 1)	-6.768 ± 2.45	0.569 ± 0.16	0.327 ± 0.08	4.803 ± 0.49	0.735 ± 0.06	97.434 ± 39.79
DiffSBDD-inpaint (C_α, 5)	-6.990 ± 3.10	$\mathbf{0.597 \pm 0.15}$	0.325 ± 0.08	4.819 ± 0.48	0.719 ± 0.07	
DiffSBDD-inpaint (C_α, 10)	-7.203 ± 2.76	$\mathbf{0.597 \pm 0.15}$	0.320 ± 0.08	4.827 ± 0.47	0.716 ± 0.07	
DiffSBDD-cond	-6.895 ± 2.04	0.530 ± 0.16	0.329 ± 0.08	4.779 ± 0.53	0.724 ± 0.07	135.866 ± 51.66
DiffSBDD-inpaint (1)	-5.916 ± 2.49	0.455 ± 0.14	0.316 ± 0.09	4.782 ± 0.49	0.809 ± 0.06	
DiffSBDD-inpaint (5)	-6.914 ± 2.55	0.508 ± 0.15	0.311 ± 0.09	4.803 ± 0.47	0.766 ± 0.06	
DiffSBDD-inpaint (10)	$\mathbf{-7.340 \pm 2.55}$	0.535 ± 0.14	0.306 ± 0.10	4.831 ± 0.43	0.758 ± 0.05	