Lecture 11: Graph Learning

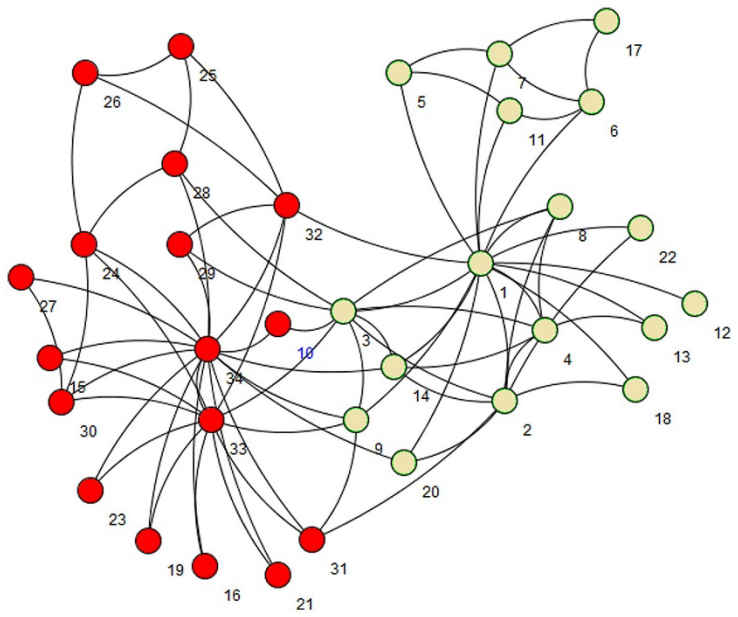
Olexandr Isayev

Department of Chemistry, CMU

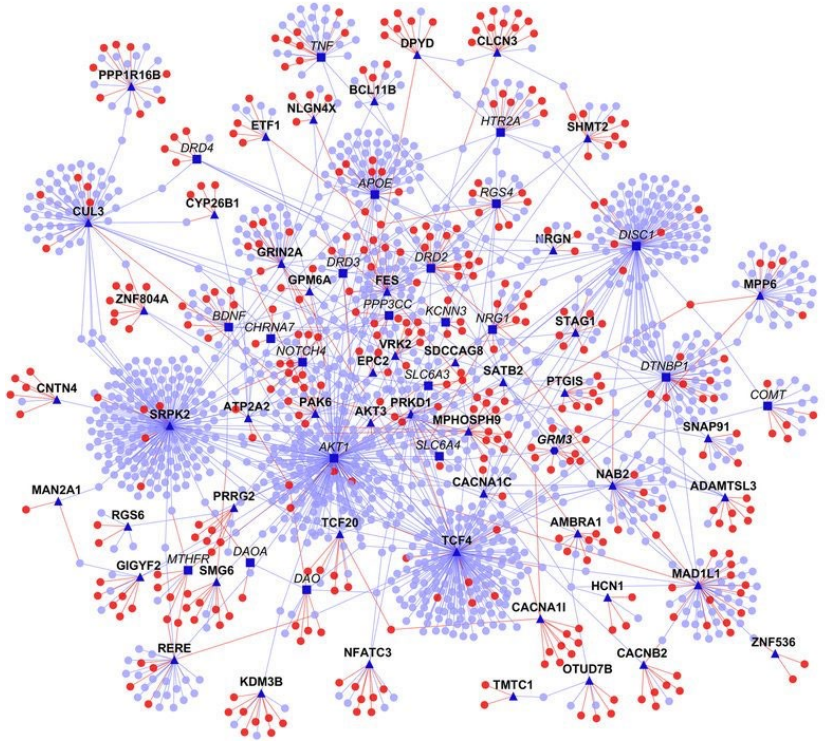
olexandr@cmu.edu

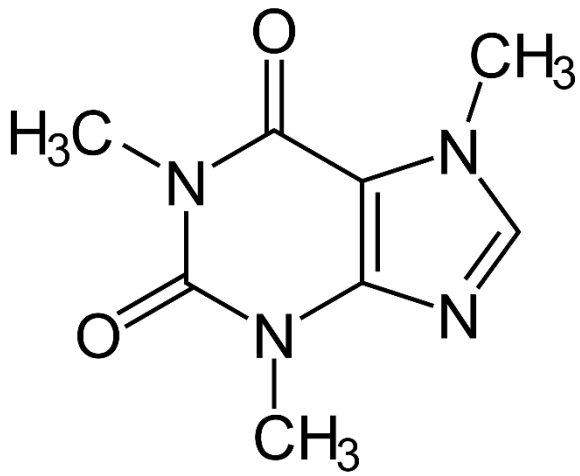
A lot of real-world data does not “live” on grids

Social networks
Citation networks
Communication networks
Multi-agent systems



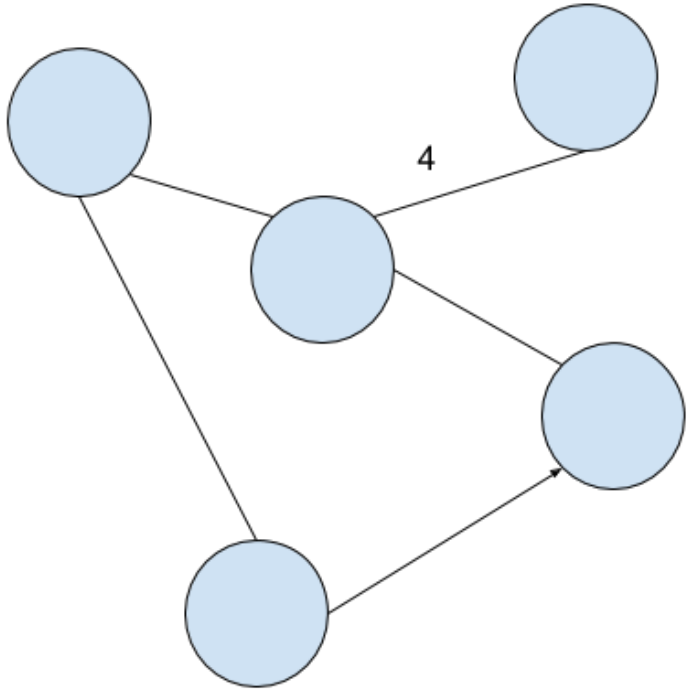
Protein interaction networks
Reaction neworks
Molecules and materials!





What is a Graph?

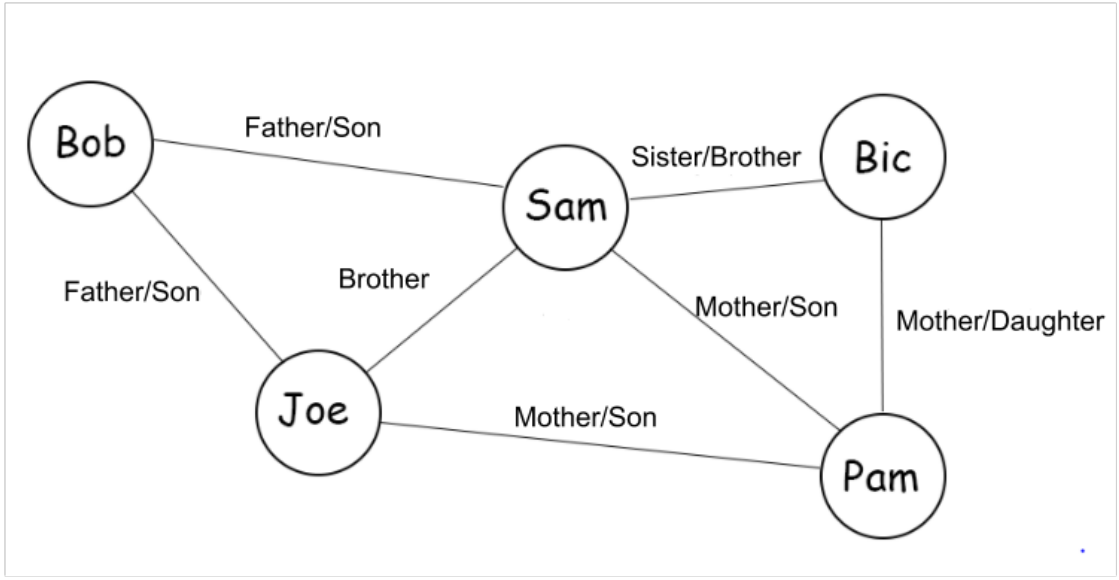
- Graph is a Data Structure that consists of vertices/nodes and edges.
- The Edges can be non-directional, directional, weighted and non-weighted.
- Each Node has features associated with it.

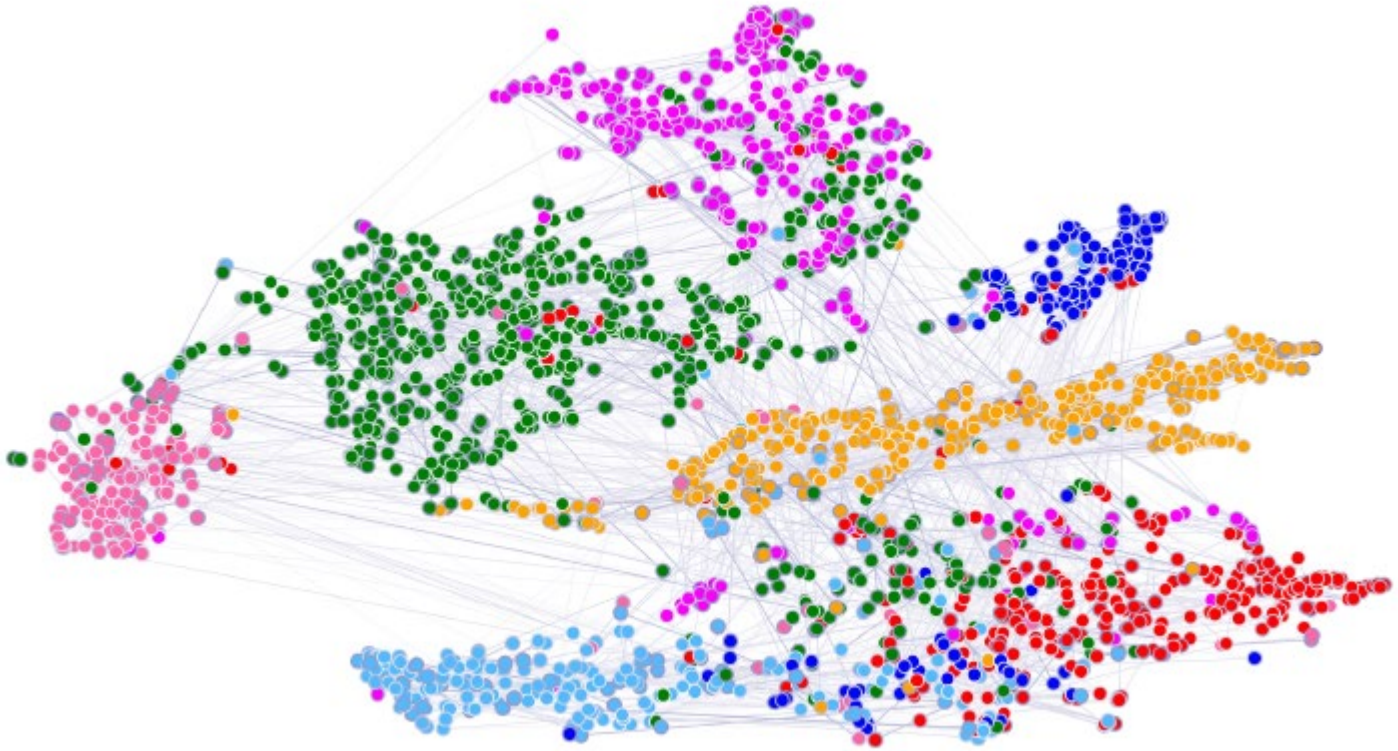


A Graph Network

Source: https://www-cs.stanford.edu/people/jure/pubs/graphrepresentation-ieee17.pdf

A Graph Network – Social Relations

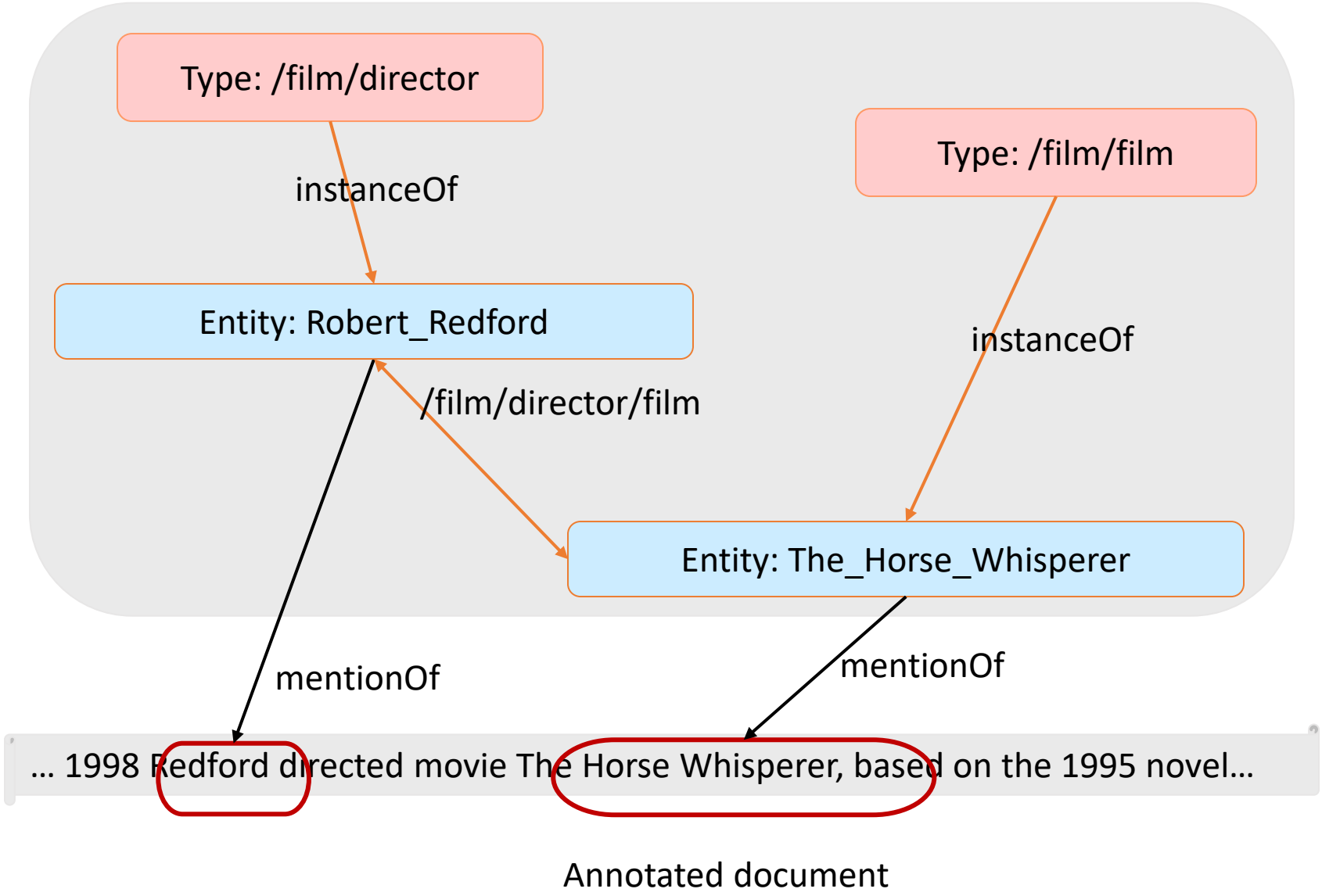




Cora Dataset – Dataset of Scientific Publications

[1] Peter Velickovic, Guillem Cucurull, Arantxa Casanova, and Adriana Romero. Graph Attention Networks. In International Conference on Learning Representations, 2018

Annotated Web with Knowledge Graph



Why Graph Learning?

- Many systems of interactions in the natural world can be easily modeled using Graphs.
  - Molecular Structures, Social Interactions, Biological Protein Networks, Financial Networks etc.
- It is not always possible to fully represent these complex relationships in a compact manner in the conventional tabular form (Euclidean data) which can then be used for learning using the standard algorithms.
- In order to make inferences/decisions from these Networks directly we need Graph learning.

Graph Learning in Medical Applications

- Many Biological Systems can be compactly represented with the help of graphs.
- **Disease Prediction**[a]: Patients are represented as nodes with features associated with medical images. Used to find out the probability of a certain disease appearing in a individual.
- **Link Prediction**[b]**:** Drug-Disease Prediction, Protein-protein interaction, Drug-drug interaction.
- **Neuron Modelling:** Neurons in the Brain can be well represented by Graphs.

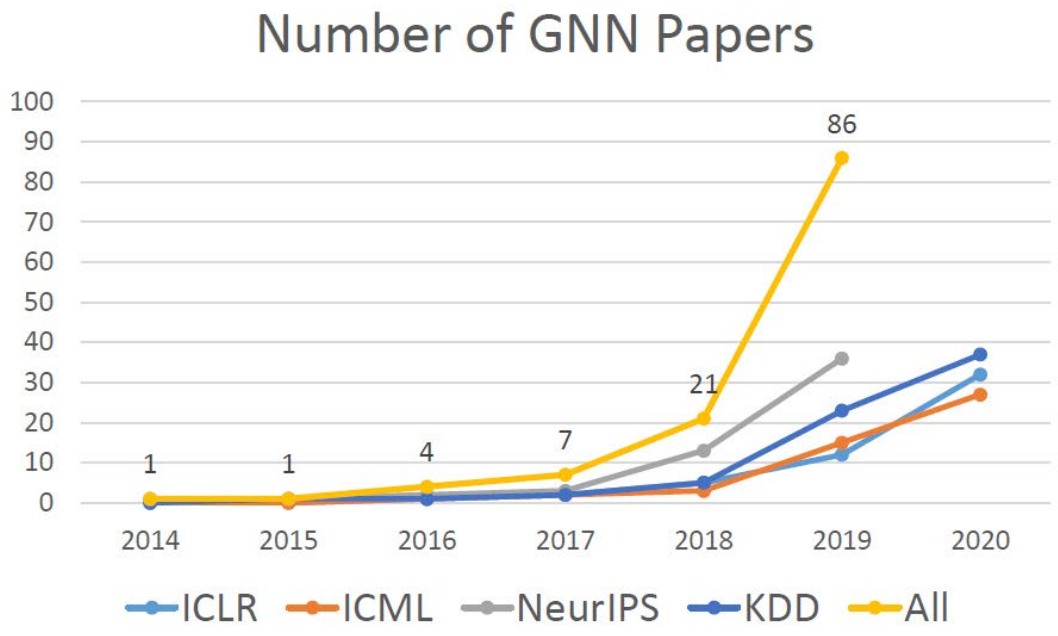
[a] Sarah Parisot, Sofia Ira Ktena, Enzo Ferrante, Matthew Lee, Ricardo Guerrero, Ben Glocker, Daniel Rueckert - Disease Prediction using Graph Convolutional Networks: Application to Autism Spectrum Disorder and Alzheimer's Disease, 2018.
[b] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, Huan Sun - Graph embedding on biomedical networks: methods, applications and evaluations, 2019.

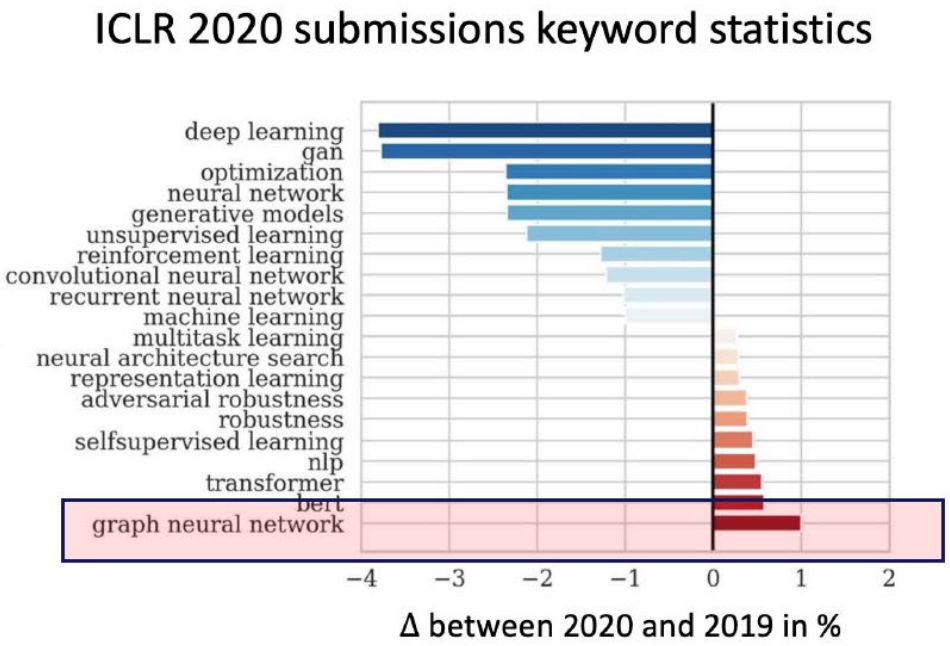
Graph Neural Networks

- Directly Operates on the Graph Structured Data.
- Typically used for the purpose of node classification, edge classification or a classification of the whole graph network as a whole (Supervised Learning).
- Unsupervised learning frameworks also exist such as Graph Autoencoders.
- Typically the Graph Neural Networks consist of two matrices:
  - A Feature Description Matrix(H) that consists of the features of all nodes. (Dimensions - Nodes x Features per node)
  - A Adjacency matrix(A) that encodes the structure of edges.

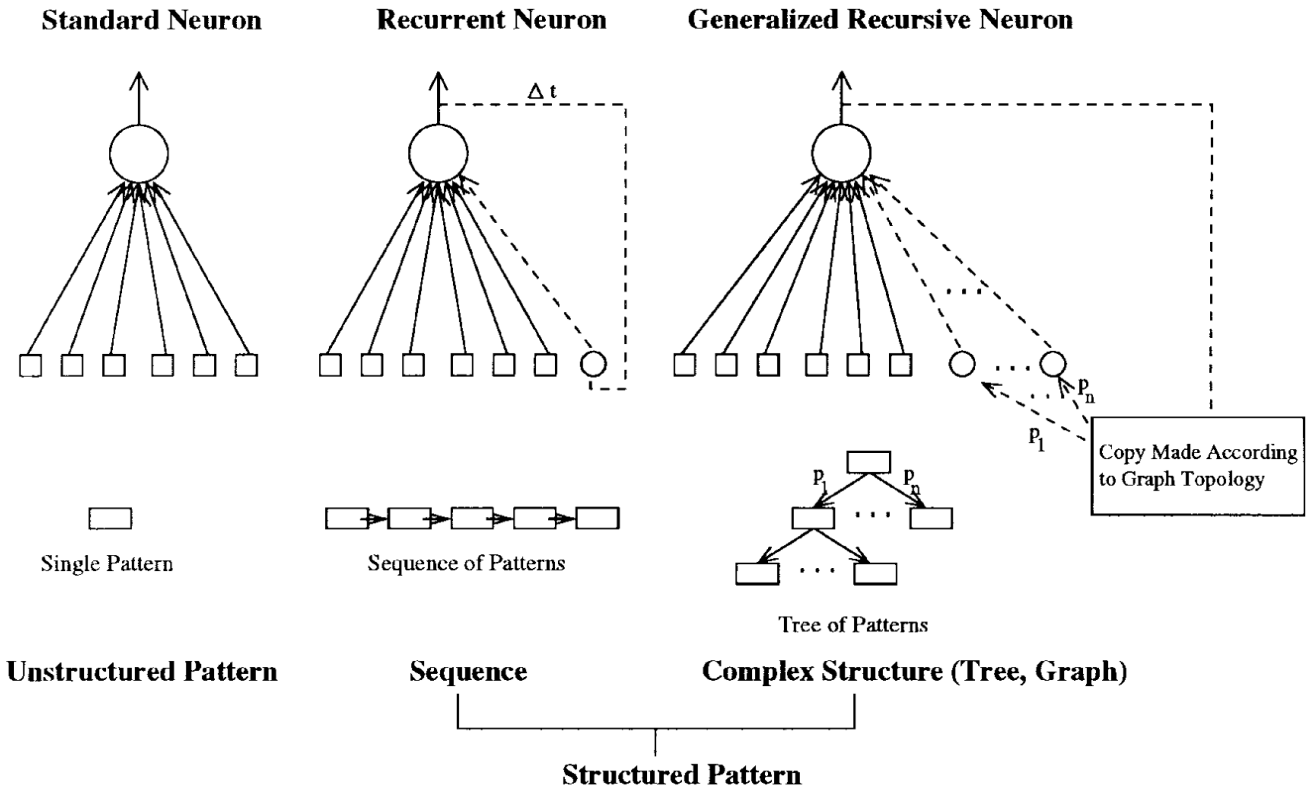
Graph Network Layer: $H^{(l+1)} = f(H^{(l)}, A)$

A Rapidly Growing Area





Understanding GNN as RNN



The RNN on sequences can be generalized to trees and DAGs.

The Question

What Vertices are Most Important?

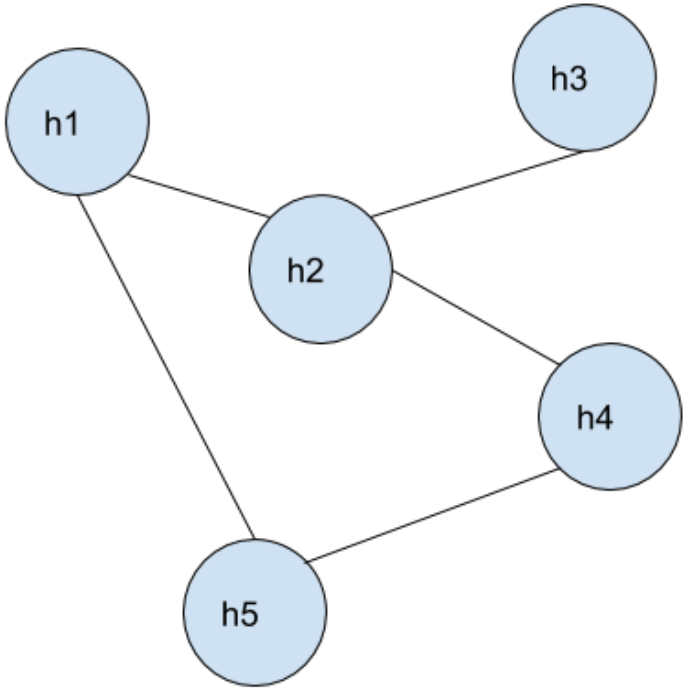
Everyday Understandings

- Important = prominent
- Important = admired
- Important = linchpin
- Important = listened to
- Important = in the know
- Important = gate keeper
- Important = involved

Translations

Ordinary Description	Possible Network Interpretation
prominent	Vertex is “visible” to many other vertices
admired	Vertex is “chosen” by many other vertices
listened to	Vertex is “received” by many other vertices
in the know	Vertex is short distance from many sources of information
linchpin	Vertex irreplaceable part
gate keeper	Vertex stands between one part of graph and another
involved	Vertex connected to many parts of graph

Adjacency Matrix

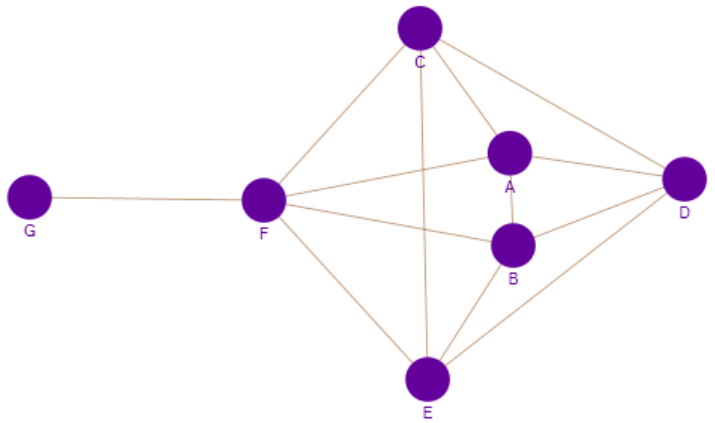


$$\boldsymbol{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Degree Centrality

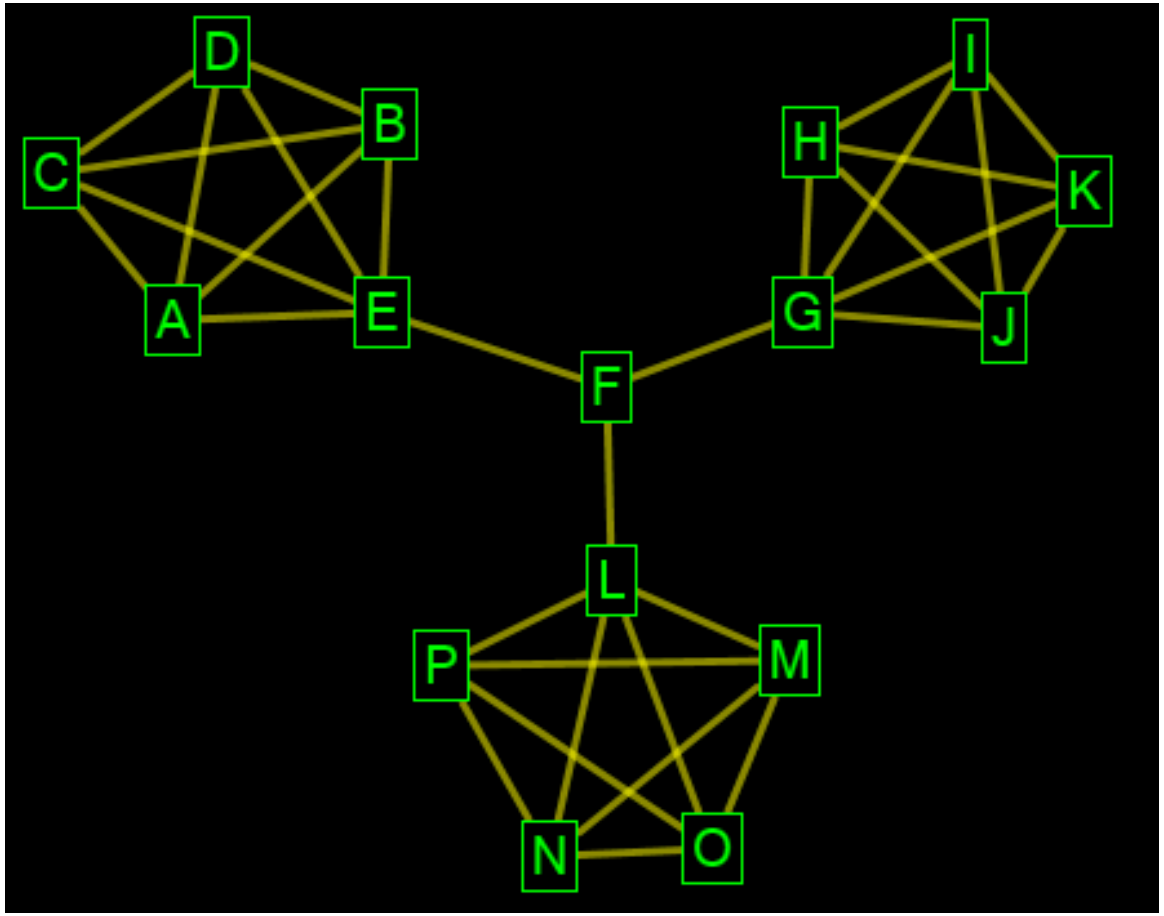
$$C_D(v_i) = k_i$$

	A	B	C	D	E	F	G
A	-	1	1	1	0	1	0
B		-	0	1	1	1	0
C			-	1	1	1	0
D				-	1	0	0
E					-	1	0
F						-	1
G							-



	C_D
A	4
B	4
C	4
D	4
E	4
F	4
G	1

Degree Centrality Can Mislead



Closeness Centrality

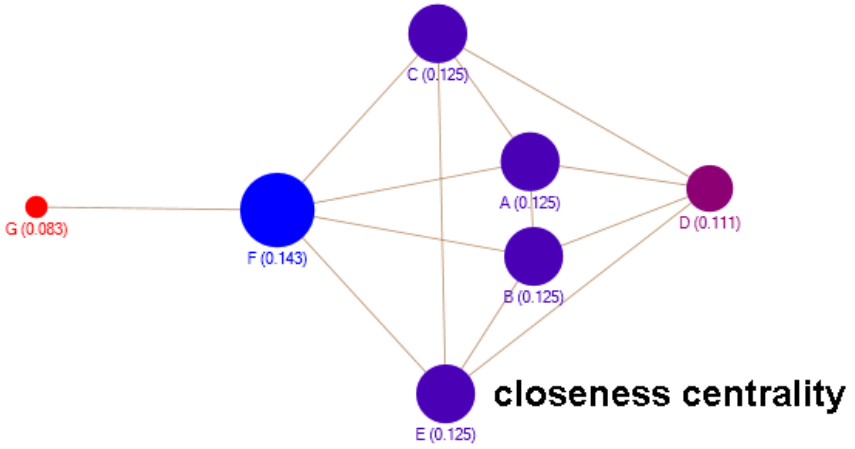
- Closeness = 1/total distance to other vertices

$$C_c(v) = 1 \Big/ \sum_{i=1}^{n} d(v, v_i)$$

$$C_c(A) = \frac{1}{d(AB) + d(AC) + d(AD) + d(AE) + d(AF) + d(AG)}$$

$$C_c(A) = \frac{1}{1 + 1 + 1 + 2 + 1 + 2}$$

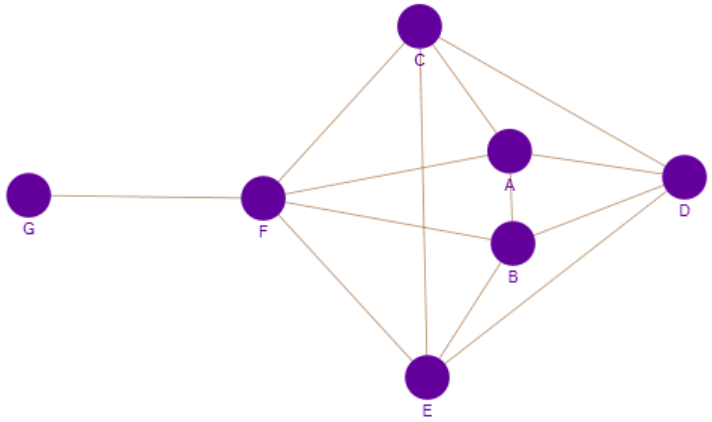
$$C_c(A) = \frac{1}{8} = 0.125$$



Betweenness Centrality

$$C_b(v_i) = \sum \frac{n_{sti}}{g_{st}}$$

- Fraction of shortest paths that include vertex

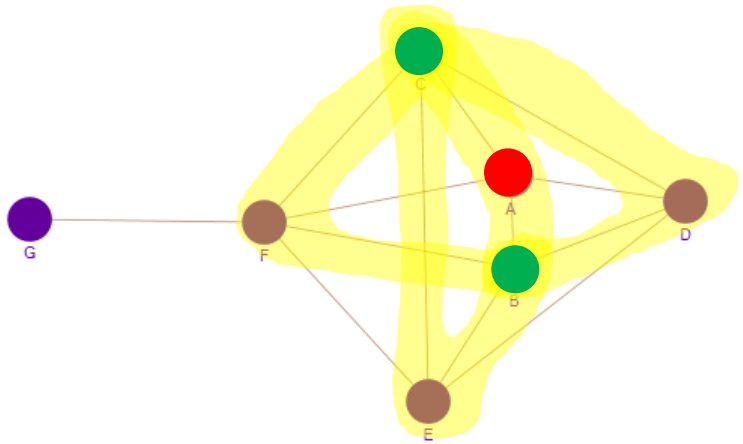


	A	B	C	D	E	F	G
A	-	1,1	1,1	1,1	2,4	1,1	2,1
B		-	2,4	1,1	1,1	1,1	2,1
C			-	1,1	1,1	1,1	2,1
D				-	1,1	2,4	3,4
E					-	1,1	2,1
F						-	1,1
G							-

Betweenness Centrality

- Fraction of shortest paths that include vertex

Example: Calculate betweenness centrality of vertex A



$$C_b(v_i) = \sum \frac{n_{sti}}{g_{st}} = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 0.75$$

Vertex Centrality Comparison

- Usually centrality metrics positively correlated
- When not, something interesting going on

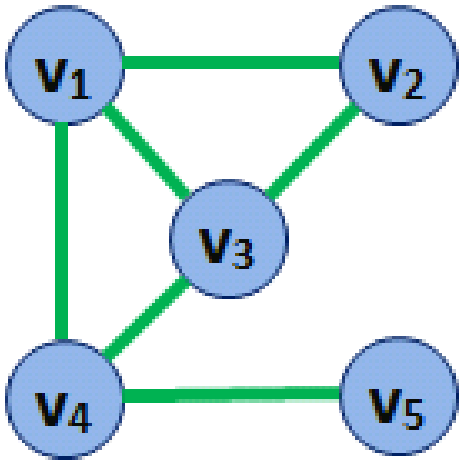
	Low Degree	Low Closeness	Low Betweenness
High Degree		cluster that is far from the rest of the network	connections are redundant - communication bypasses this vertix
High Closeness	tied to important or active alters		Probably multiple paths in the network
High Betweenness	ties are crucial for network flow	Very rare cell. Ties from a small number of objects to many others.	

Eigenvector

- Adjacency matrix redistributes vertex contents
- Some vector of contents is in equilibrium
- These are the eigenvector centralities

What is an Eigenvector?

- Consider a graph & its 5x5 adjacency matrix, A

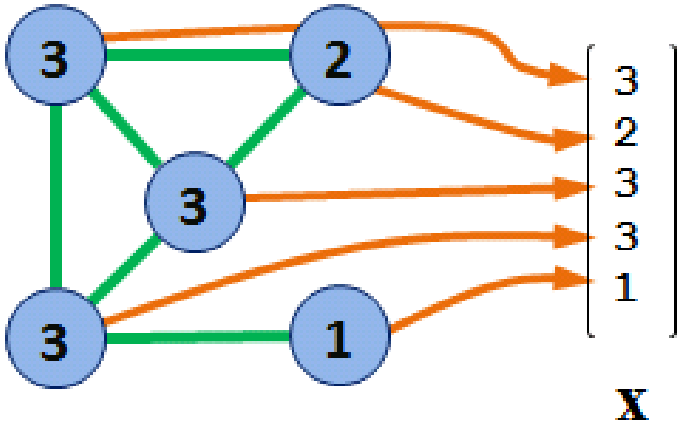


$$\begin{pmatrix} - & 1 & 1 & 1 & 0 \\ & - & 1 & 0 & 0 \\ & & - & 1 & 0 \\ & & & - & 1 \\ & & & & - \end{pmatrix}$$

A

And then consider a vector, x…

- a 5x1 vector of values, one for each vertex in the graph. In this case, we've used the degree centrality of each vertex.



What happens when…

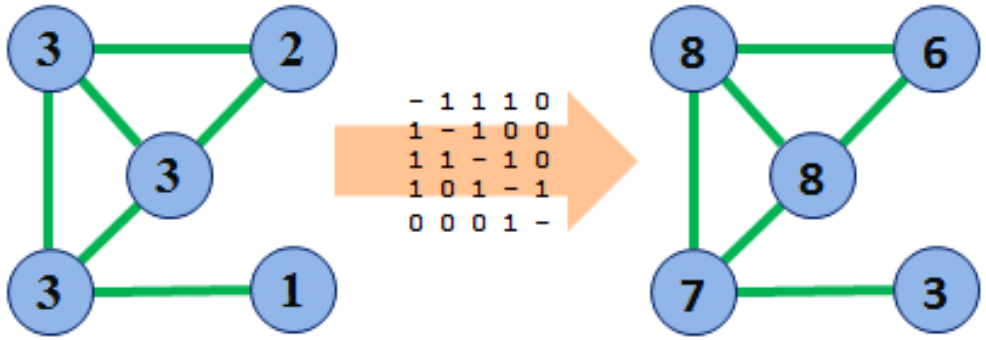
- …we multiply the vector x by the matrix A?

- The result, of course, is another 5x1 vector.

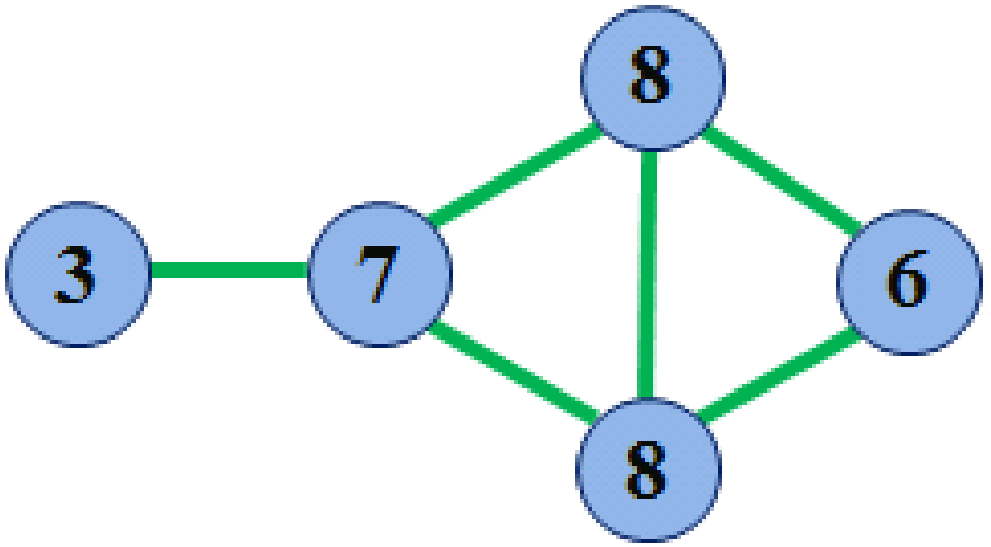
$$\mathbf{A} \times \mathbf{x} = \begin{pmatrix} -1 & 1 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & - \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 3 \\ 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0x3 + 1x2 + 1x3 + 1x3 + 0x1 \\ 1x3 + 0x2 + 1x3 + 0x3 + 0x1 \\ 1x3 + 1x2 + 0x3 + 1x3 + 0x1 \\ 1x3 + 0x2 + 1x3 + 0x3 + 1x1 \\ 0x3 + 0x2 + 0x3 + 1x3 + 0x1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 8 \\ 7 \\ 3 \end{pmatrix}$$

$\mathbf{A}^x\mathbf{x}$ diffuses the vertex values

- Look at first element of resulting vector
- The 1s in the A matrix "pick up" values of each vertex to which the first vertex is connected
- Result value is sum of values of these vertices.



Intuitiveness Visible on Rearrangement



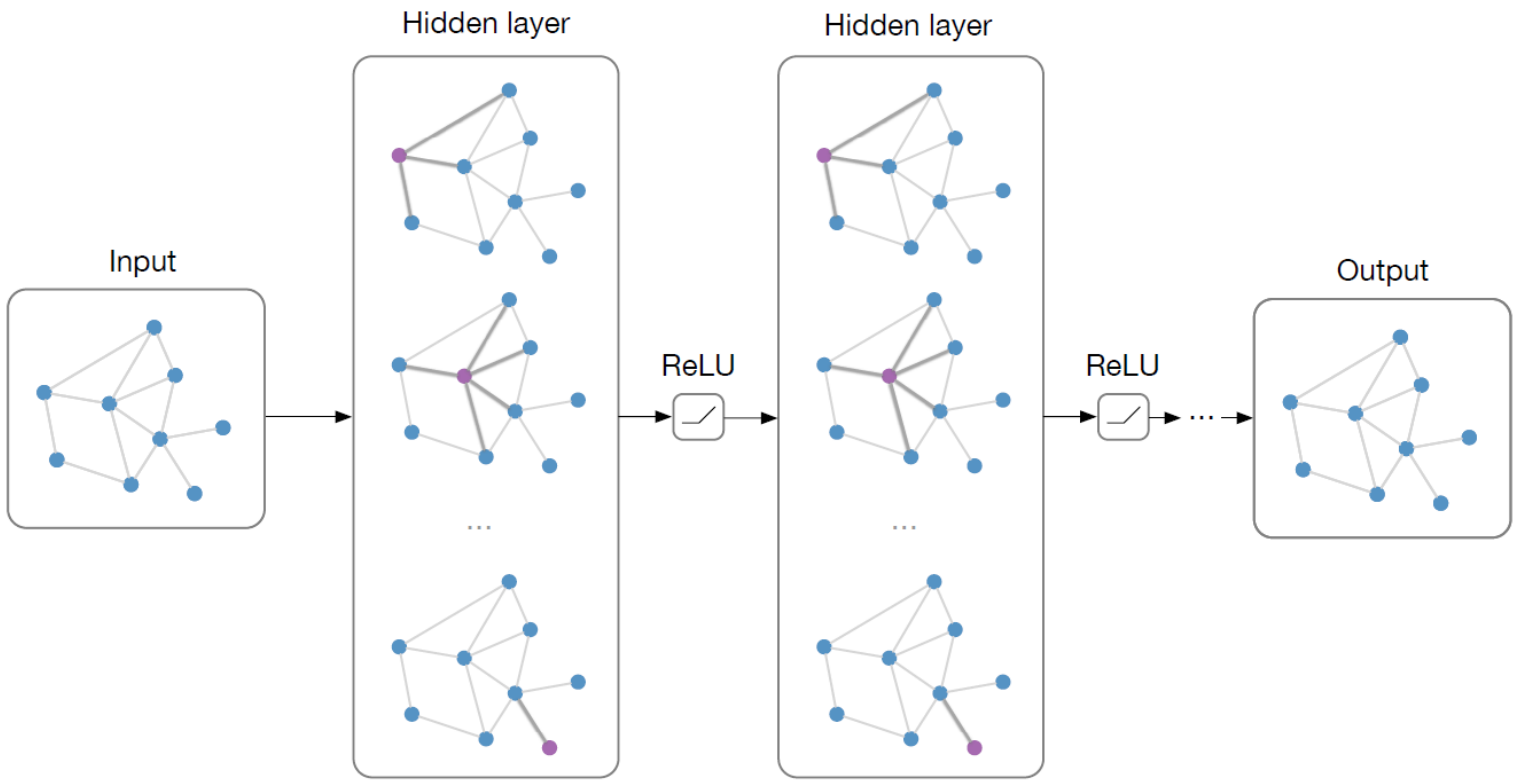
Deep learning on graphs and beyond

Graph neural nets (GNNs)

Graph convolutional neural networks (GCNN)

Graph Neural Networks (GNNs)

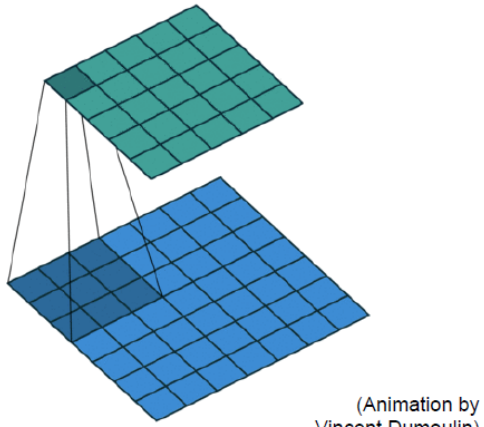
The bigger picture:



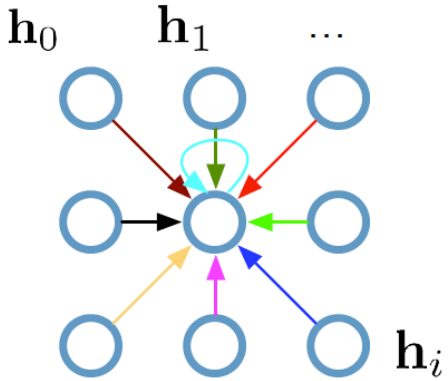
Main idea: Pass messages between pairs of nodes & agglomerate

Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

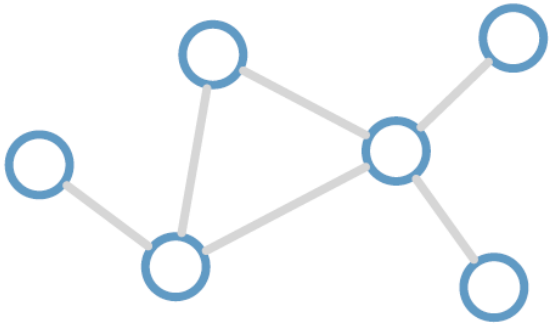
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

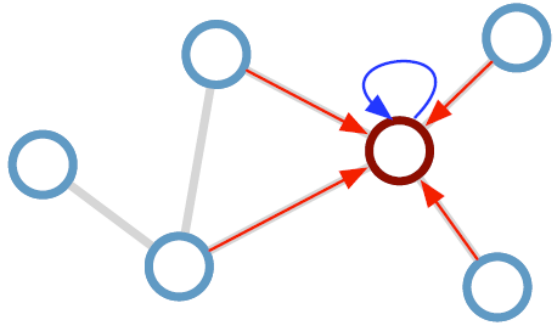
$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

Convolutional neural networks (on graphs)

Consider this undirected graph:



Calculate update for node in red:

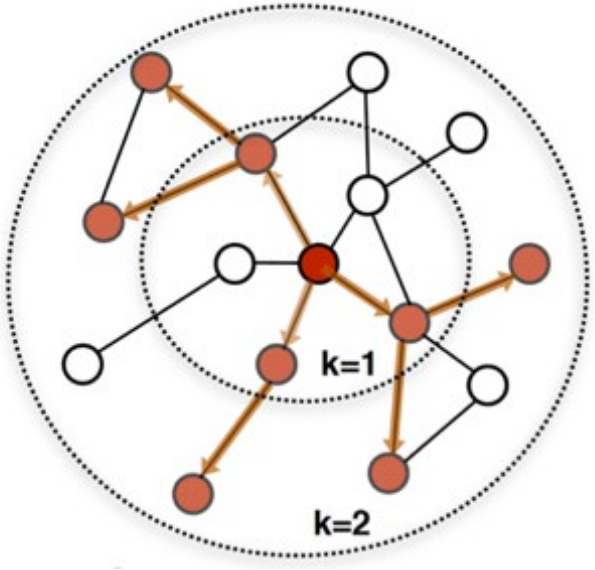


Update rule:

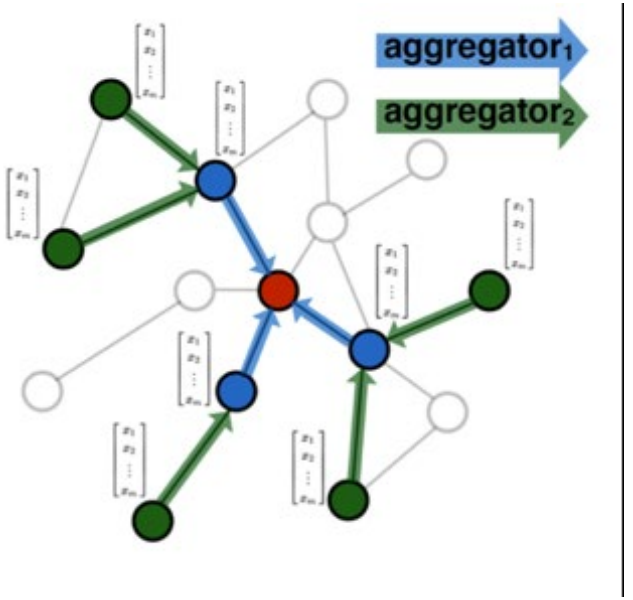
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

Graph defines computation



Determine node computation graph



Propagate and transform information

Graph Convolutional Networks

- Analogous to the Convolutional Neural Network.
- Can Handle Edges with Weights.
- We Transform the Matrix encoding the Graph Structure from the Spatial to the Spectral Domain using Fourier Transform.
- From Spectral Graph Analysis the Graph Laplacian is given by :

$$L = I_n - D^{-1/2} W D^{-1/2}$$

Here **W** is the Weighted Adjacency Matrix. **D** is a Diagonal Degree Matrix – $D_{ii} = \sum_j W_{ij}$,

- The Laplacian Matrix is diagonalizable by the Fourier Basis

$U = [u_0, \ldots, u_{n-1}] \in \mathbb{R}^{n \times n}$ such that

$$L = U \Lambda U^T \text{ where } \Lambda = \text{diag}([\lambda_0, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{n \times n}$$

- **U** consists of the complete set of orthonormal Eigenvectors of **L**.

➢**The graph Fourier Transform of** $x \in \mathbb{R}^n$ is then defined as $\hat{x} = U^T x \in \mathbb{R}^n$

➢**The inverse Operation is given by** $x = U\hat{x}$

Graph Convolution

- The Convolution Operation (*** g**) is defined in the Fourier Domain as follows:

$$x *_{\mathcal{G}} y = U((U^T x) \odot (U^T y))$$

⊙Element-wise Hadamard Product

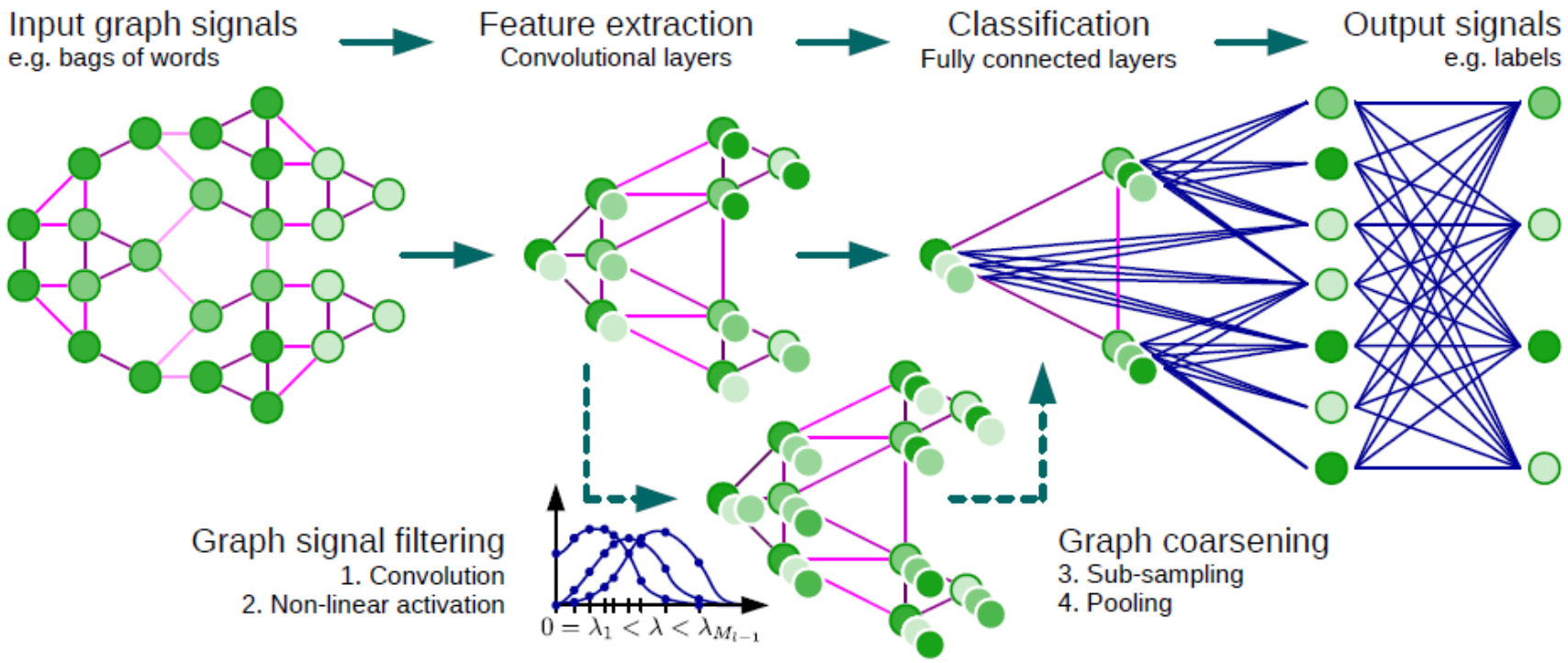
- This gives us the following representation for a node vector being filtered by ($\mathbf{g_\theta}$).

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x.$$

Drawbacks:

- Requires the knowledge of complete graph for performing the convolution operation. This is due to the fact that all the Eigenvalues($\boldsymbol{\lambda}$) of the graph structure has to be calculated for this operation which is very expensive computationally.
- New additions to the Graph requires recalculation of the entire architecture parameters due to change in Eigenvalues($\boldsymbol{\lambda}$) .
- Cannot transfer knowledge from one domain to another. (Transductive Learning)

Graph Convolutional Network - Architecture



[1] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, pages 3844–3852. Curran Associates, Inc., 2016.

Convolutional neural networks (on graphs)

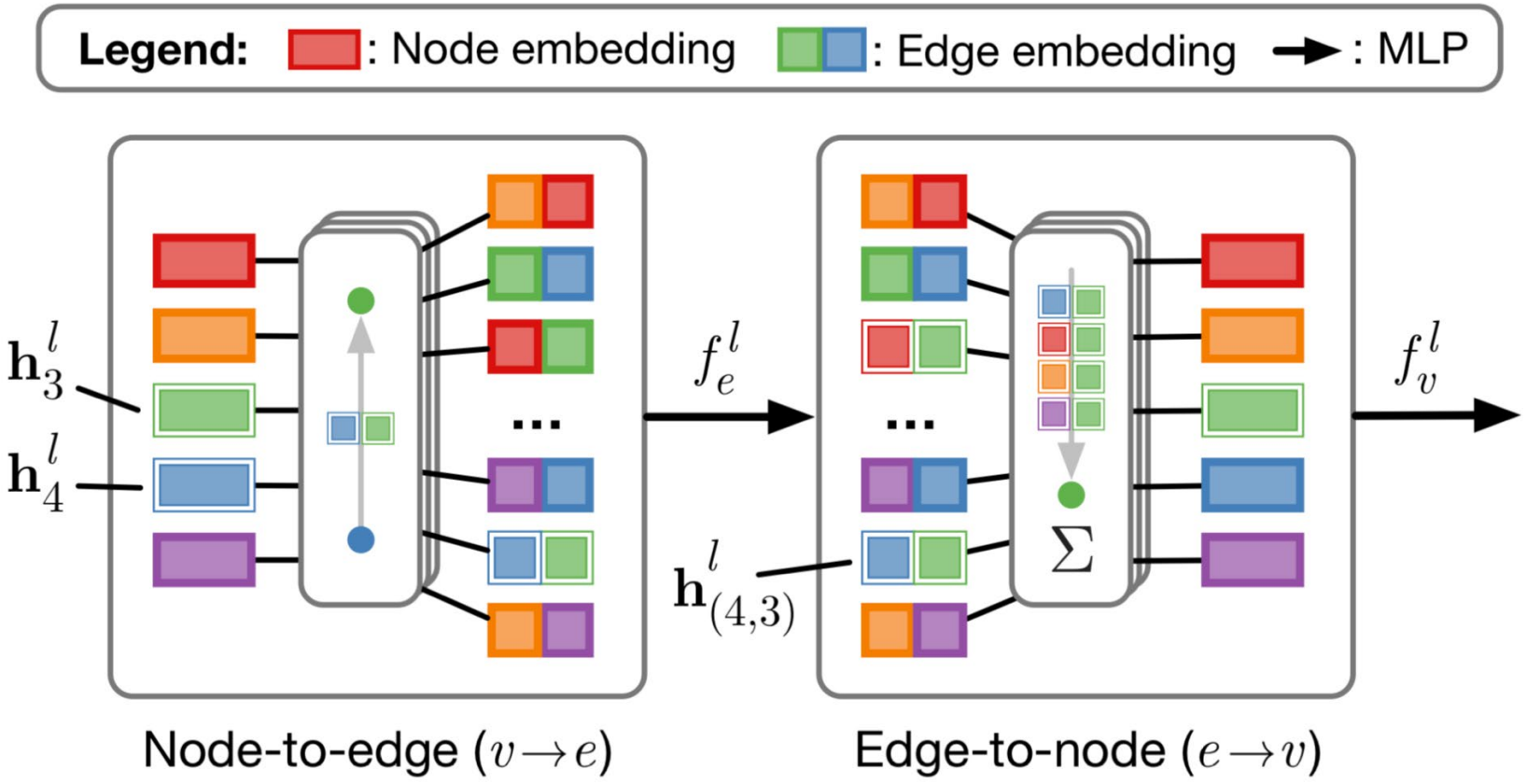
Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)

Limitations:

- Requires gating mechanism
- Only indirect support for edge features

GNNs with edge embeddings



Pros:
- Supports edge features
- More expressive than GCN
- As general as it gets (?)
- Supports sparse matrix ops

Cons:
- Need to store intermediate edge-based activations
- Difficult to implement with subsampling
- In practice limited to small graphs

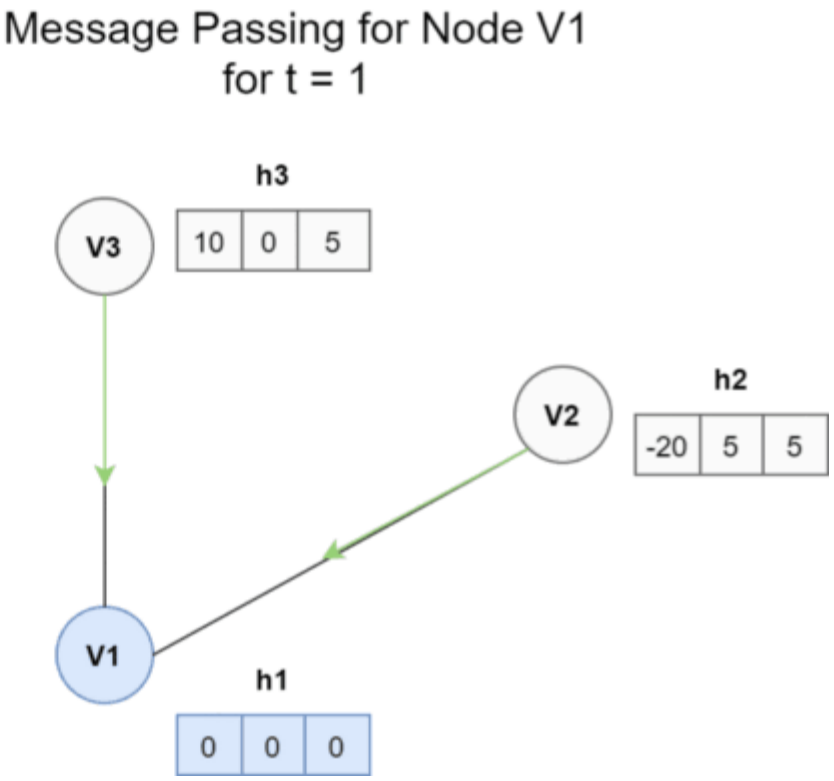
Battaglia et al. (NIPS 2016), Gilmer et al. (ICML 2017), Kipf et al. (ICML 2018)

Message Passing Neural Networks

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

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

ht - hidden state for each node



Message Passing Neural Networks

- Each node in the graph has a hidden state (i.e. feature vector).
- For each node **V**t, we aggregate a function of hidden states and possibly edges of *all* neighboring nodes with the node **V**t itself.
- Then, we update the hidden state of the node **V**t using the the obtained message and the previous hidden state of that node.

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

Then, we update the hidden state of the node **V**t using a simple equation:

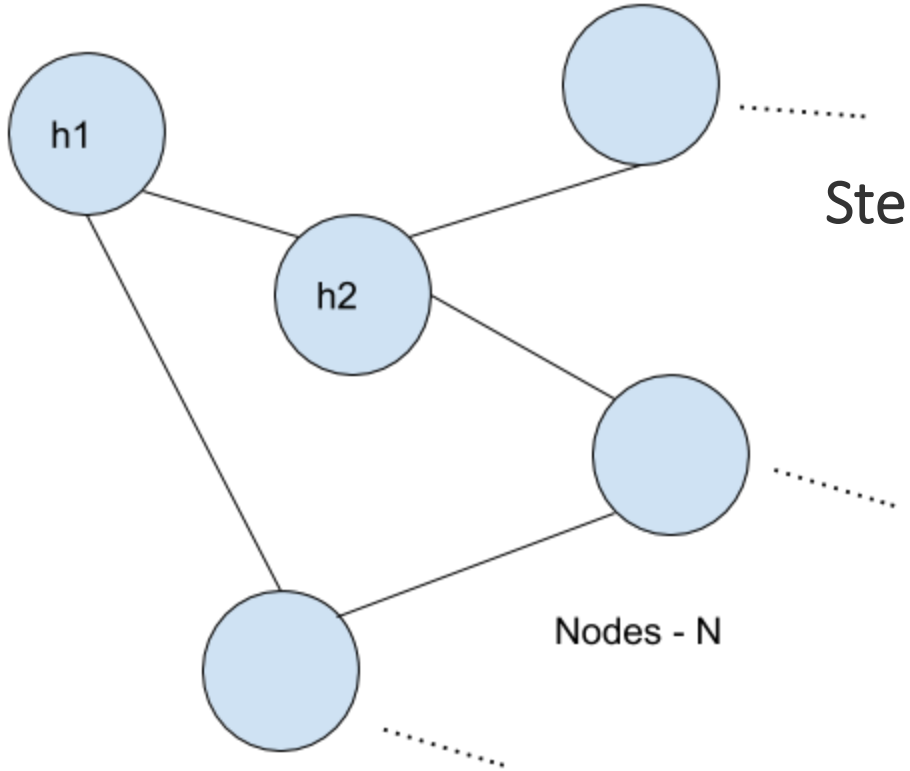
$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

Graph Attention Networks

- Convolutions directly on spatially close neighbors of a graph.
- Attention is a mechanism by which we find out which features of a nearby neighbor node affect the features of the node under consideration and how important they are.

Graph Attention Layer

Basic Building Block of the Graph Attention Network.



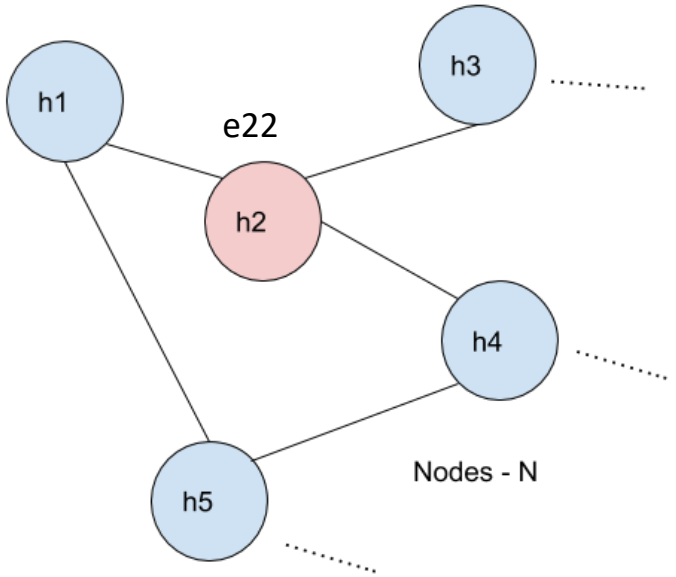
h_i - *feature vector of length F*

Step 1: Linear Transformation

A linear transformation – Weighted Matrix **W** applied to each node.

Dimensions of **W** – (*F',F)*

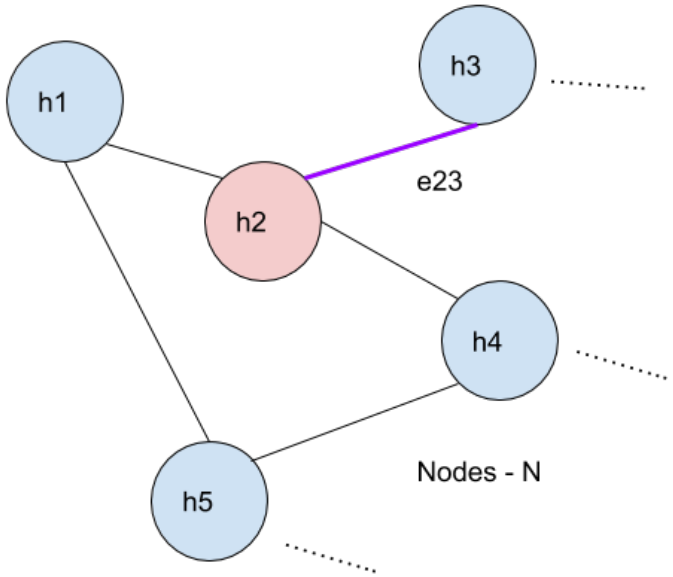
Step 2: Computation of Attention coefficients



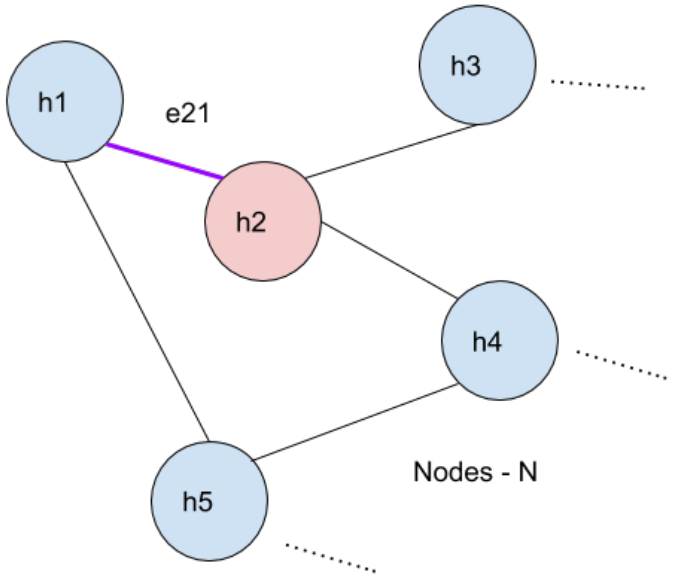
$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$$

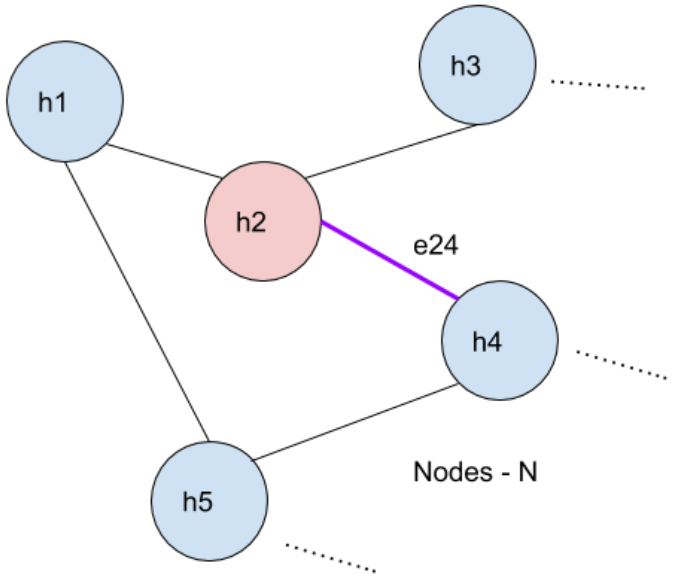
$e_{22} = a(\mathbf{W}h_2 , \mathbf{W}h_2)$



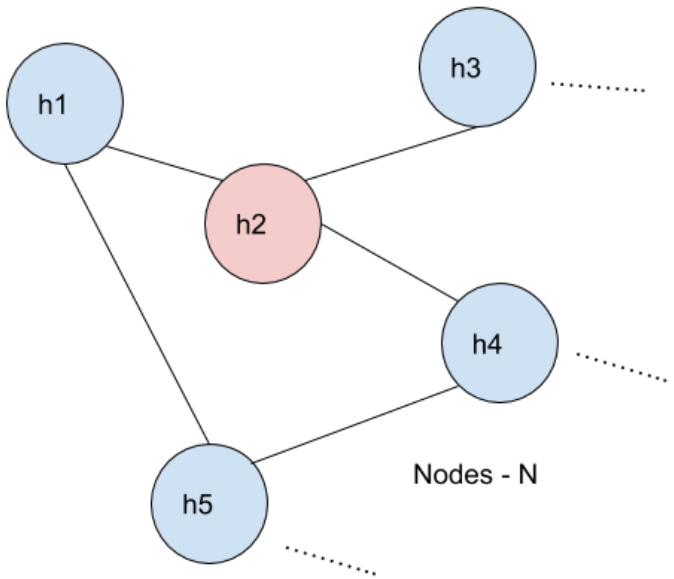
$$e_{23} = a(\mathbf{W}h_2 \, , \mathbf{W}h_3)$$



$$e_{21} = a(\mathbf{W}h_2 \, , \mathbf{W}h_1)$$

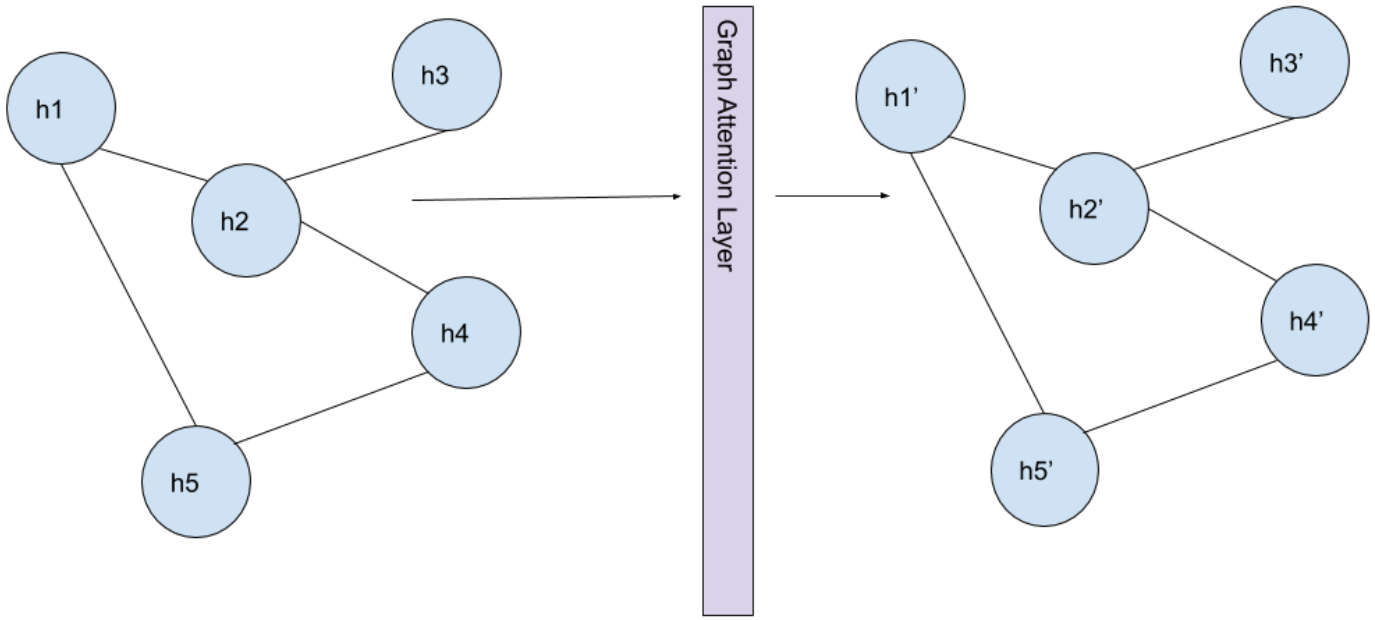


$e_{24} = a(\mathbf{W}h_2 , \mathbf{W}h_4)$



$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

Step 3: Computation of the Final Output Features



$$\vec{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} \vec{h}_j\right).$$

σ is a non-linear Transformation

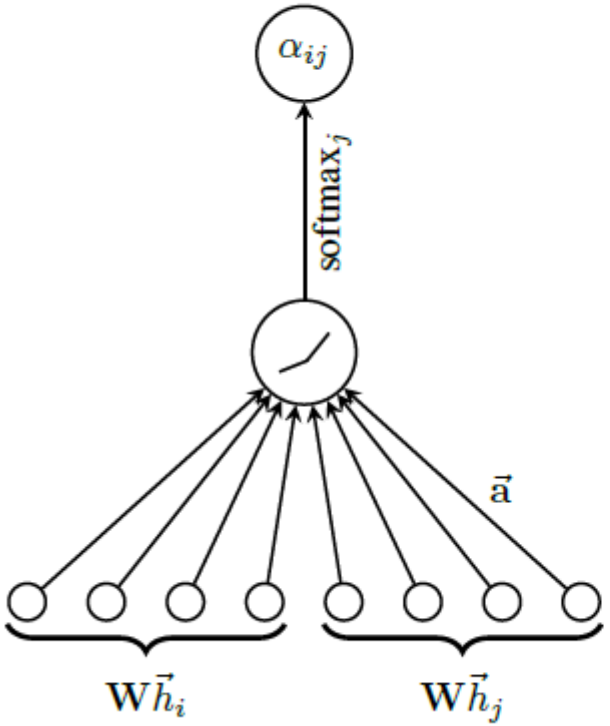
Step 4: Computation of multiple attention mechanisms

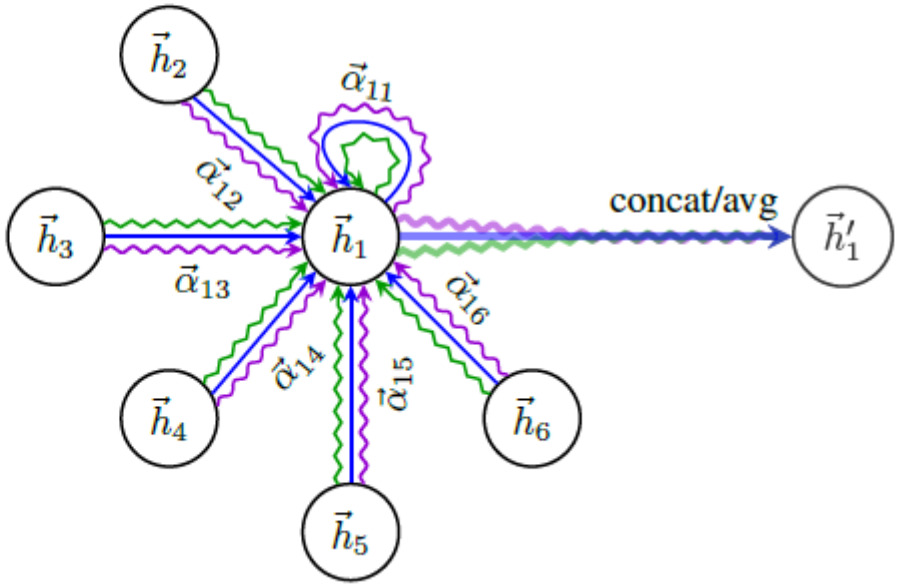
- **Multi-head attention** is employed to stabilize the learning process of self-attention.

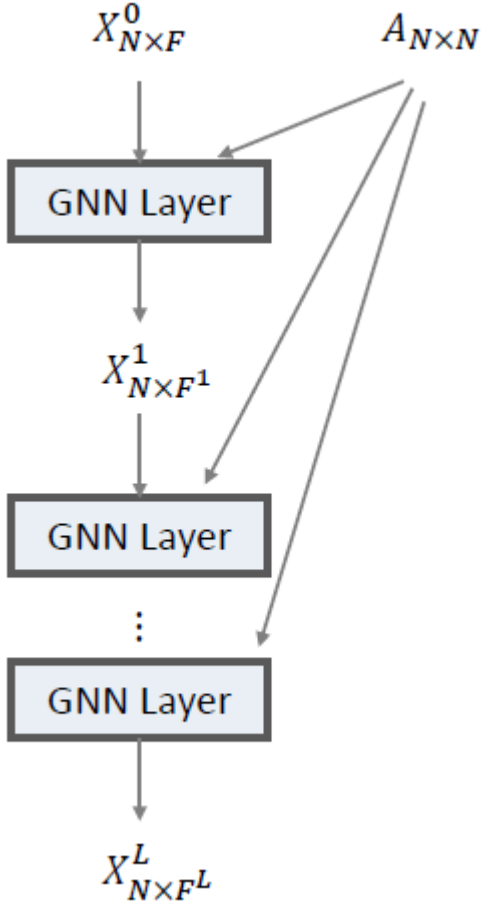
$$\vec{h}_i' = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$$

- **K** independent attention maps.

Final Overview



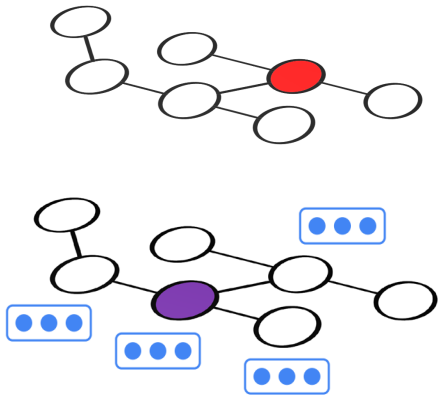




[2] Peter Velickovic, Guillem Cucurull, Arantxa Casanova, and Adriana Romero. Graph Attention Networks. In International Conference on Learning Representations, 2018

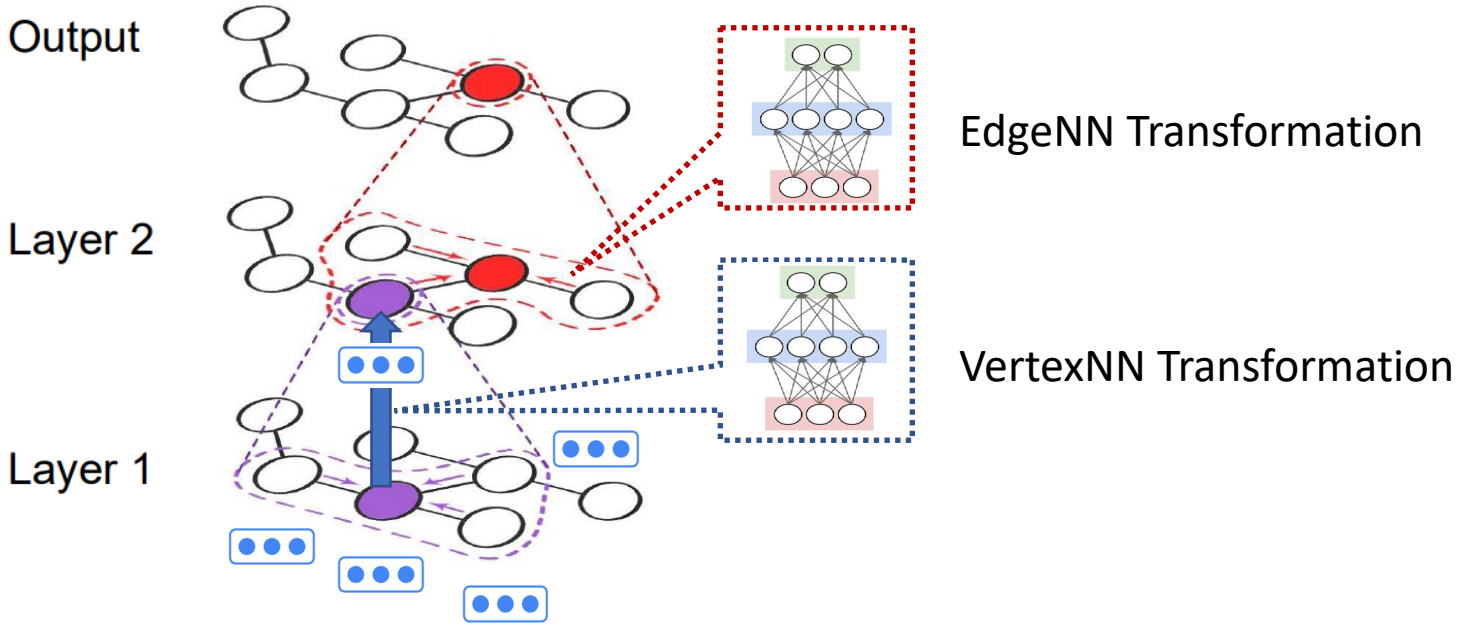
Graph Neural Networks (GNN)

- Information propagation via *Graph*
- Information transformation via *Neural Networks*

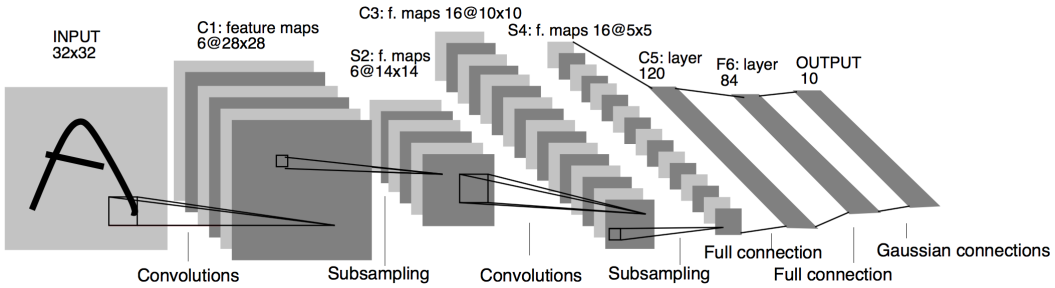


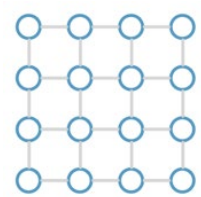
Graph Neural Networks (GNN)

- Information propagation via *Graph*
- Information transformation via *Neural Networks*



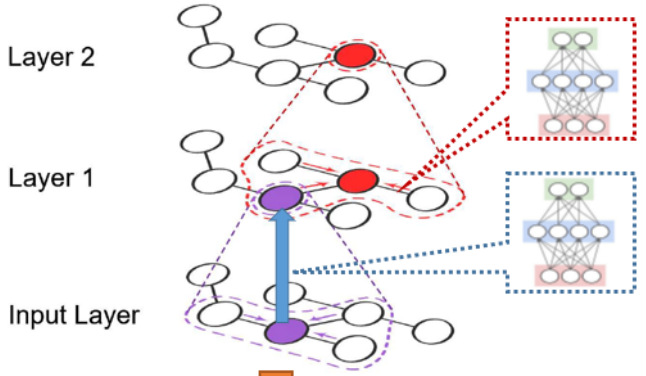
Challenges in Processing GNNs on GPU





2D grid

small and regular

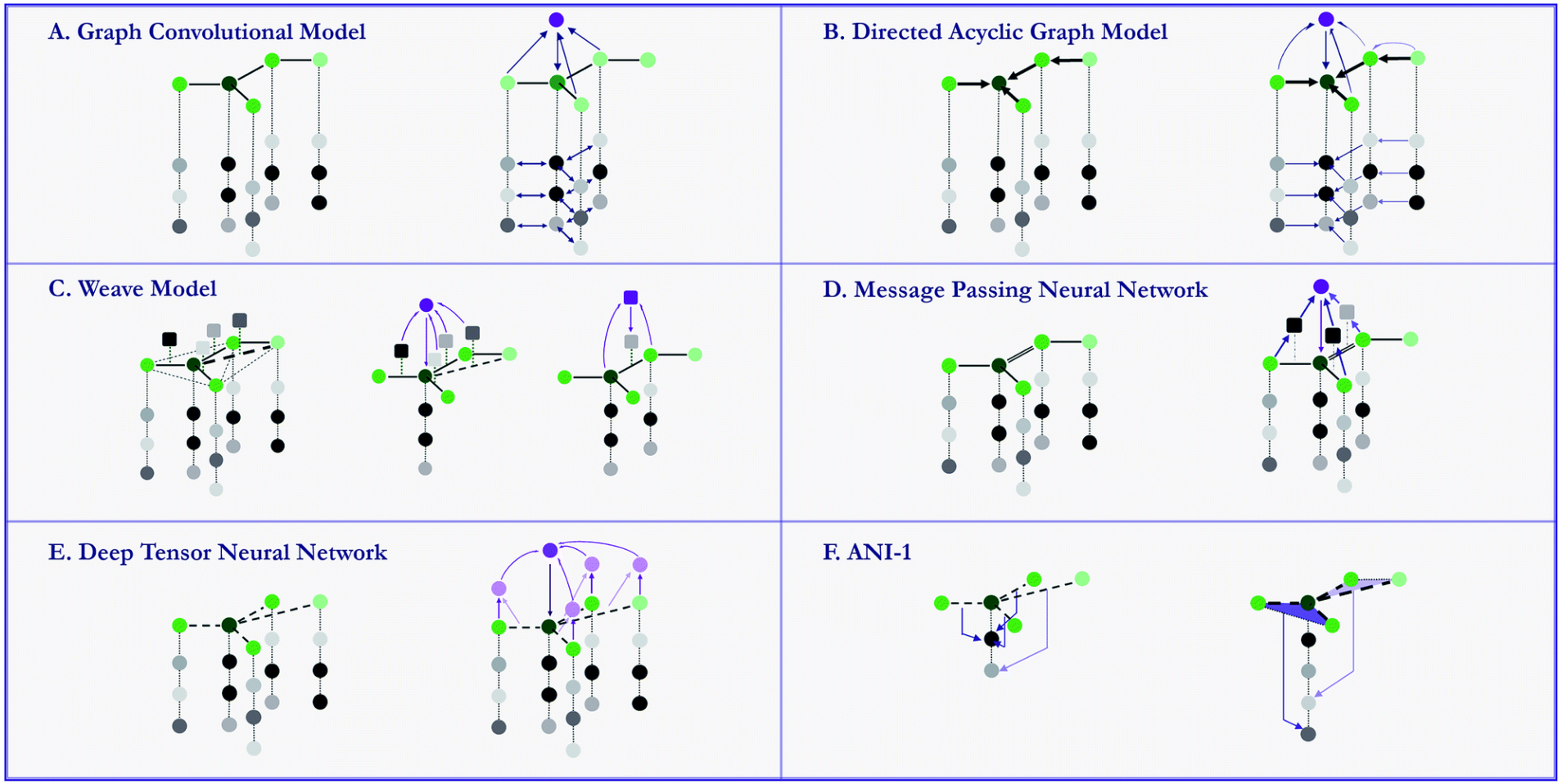


Graphs

large, sparse, and irregular

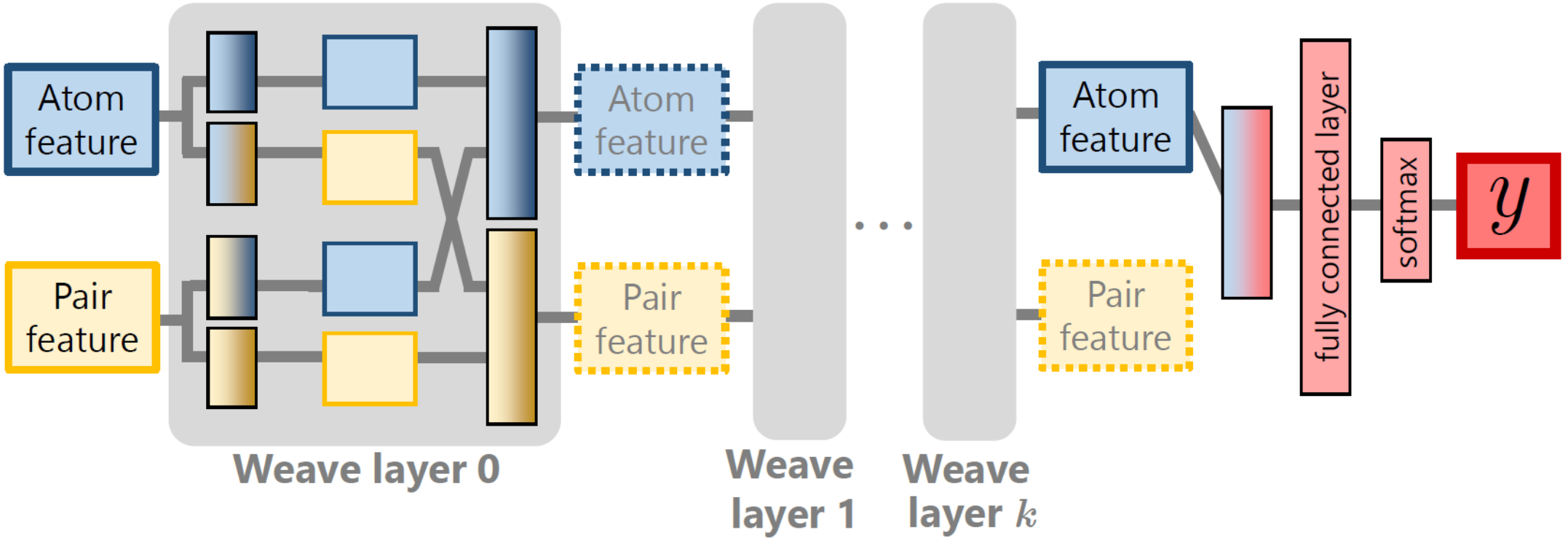
Scalability Problem
Efficiency Problem

Graph based models for molecules



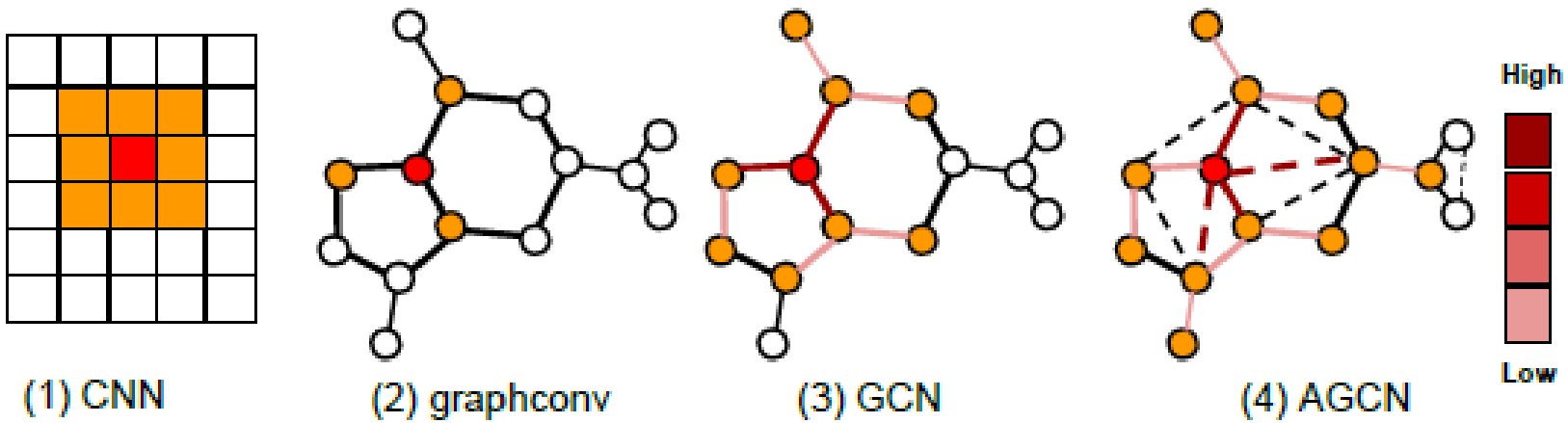
Weave module

Weave module considers not only atoms but also **atom pairs**

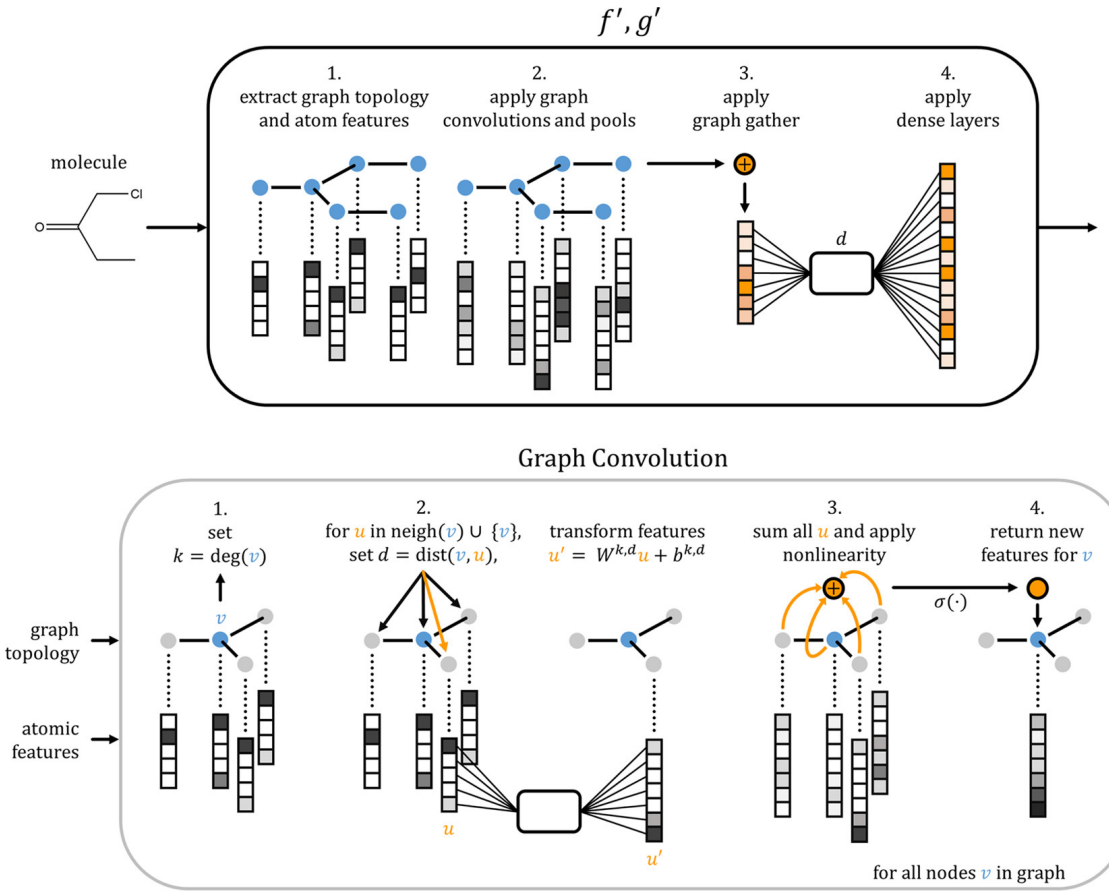


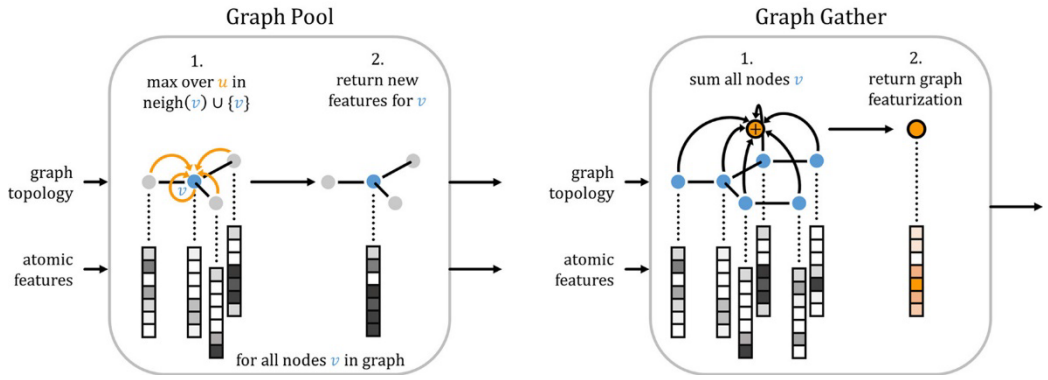
KearnesS, *et al. J Comput-Aided Mol Des*, 30(8): 593-608, 2016

GCNN for molecules



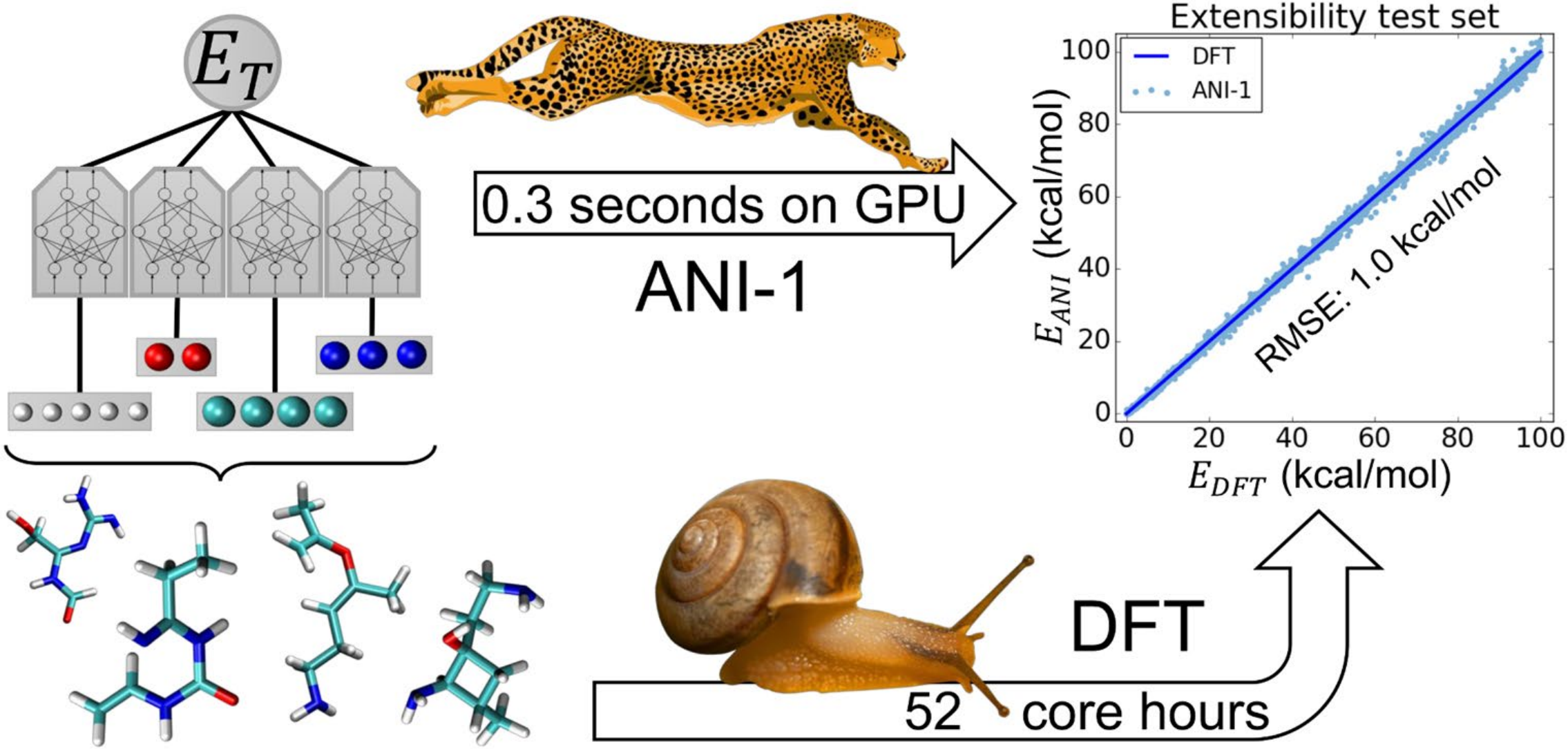
Graph Convolutional Network (GCN)





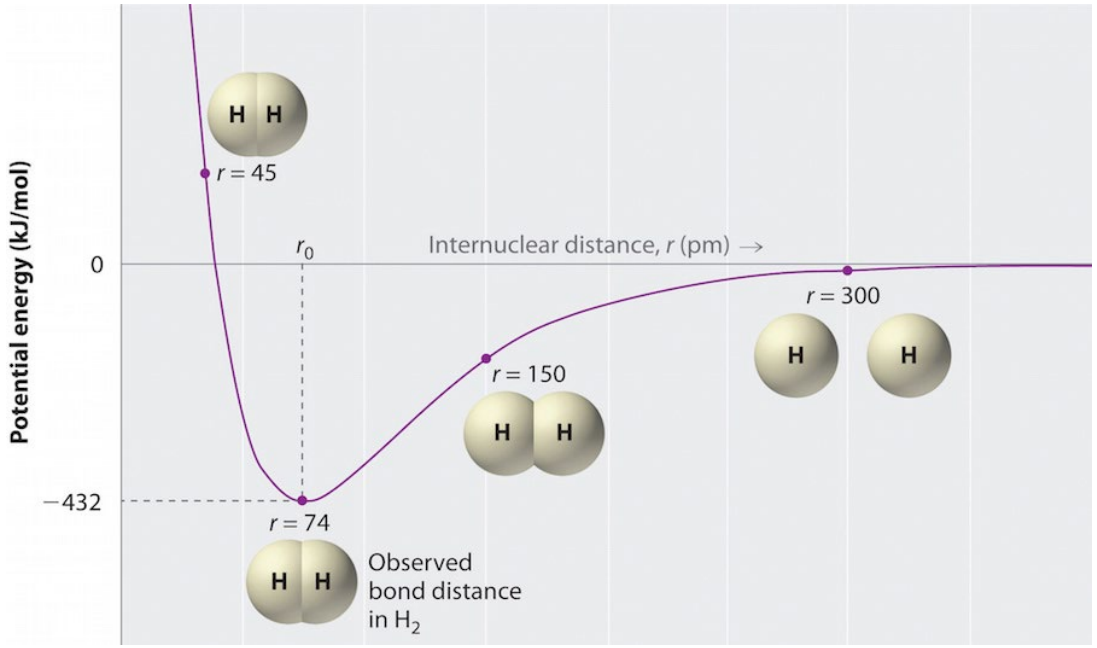
ACS Cent. Sci. 2017, 3, 4, 283-293

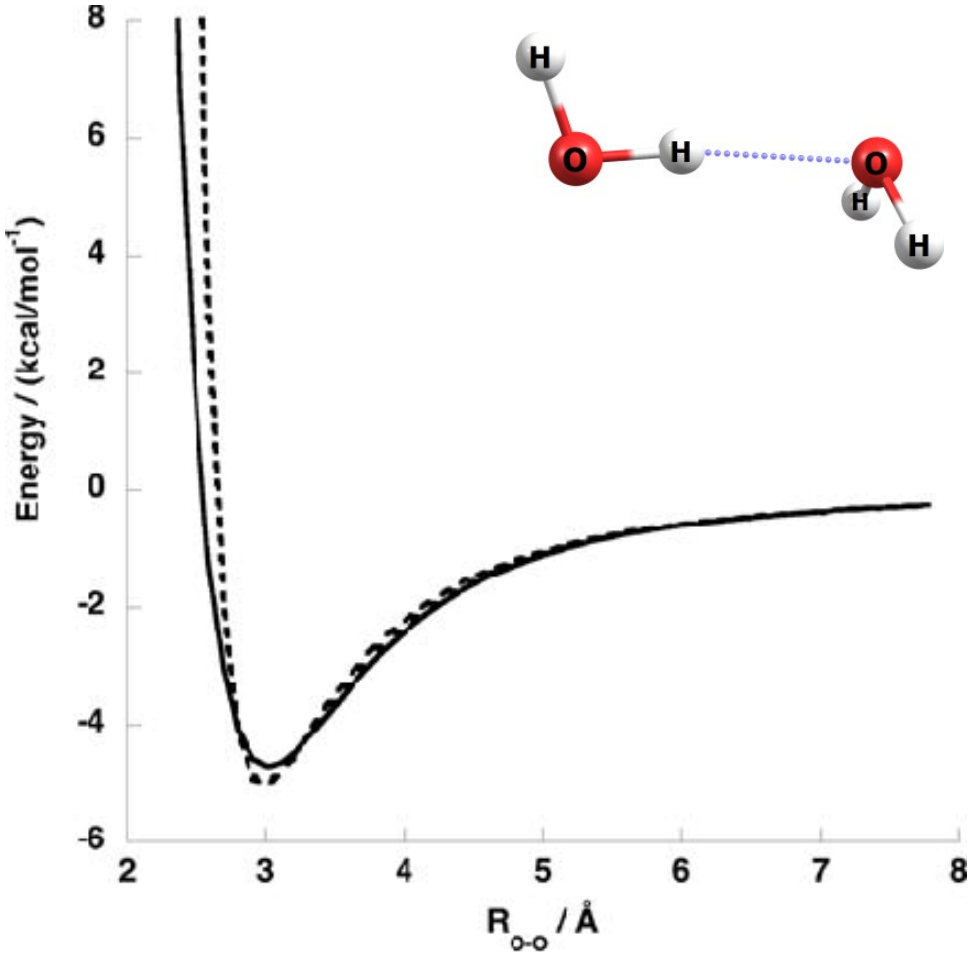
ANI Deep Neural Network



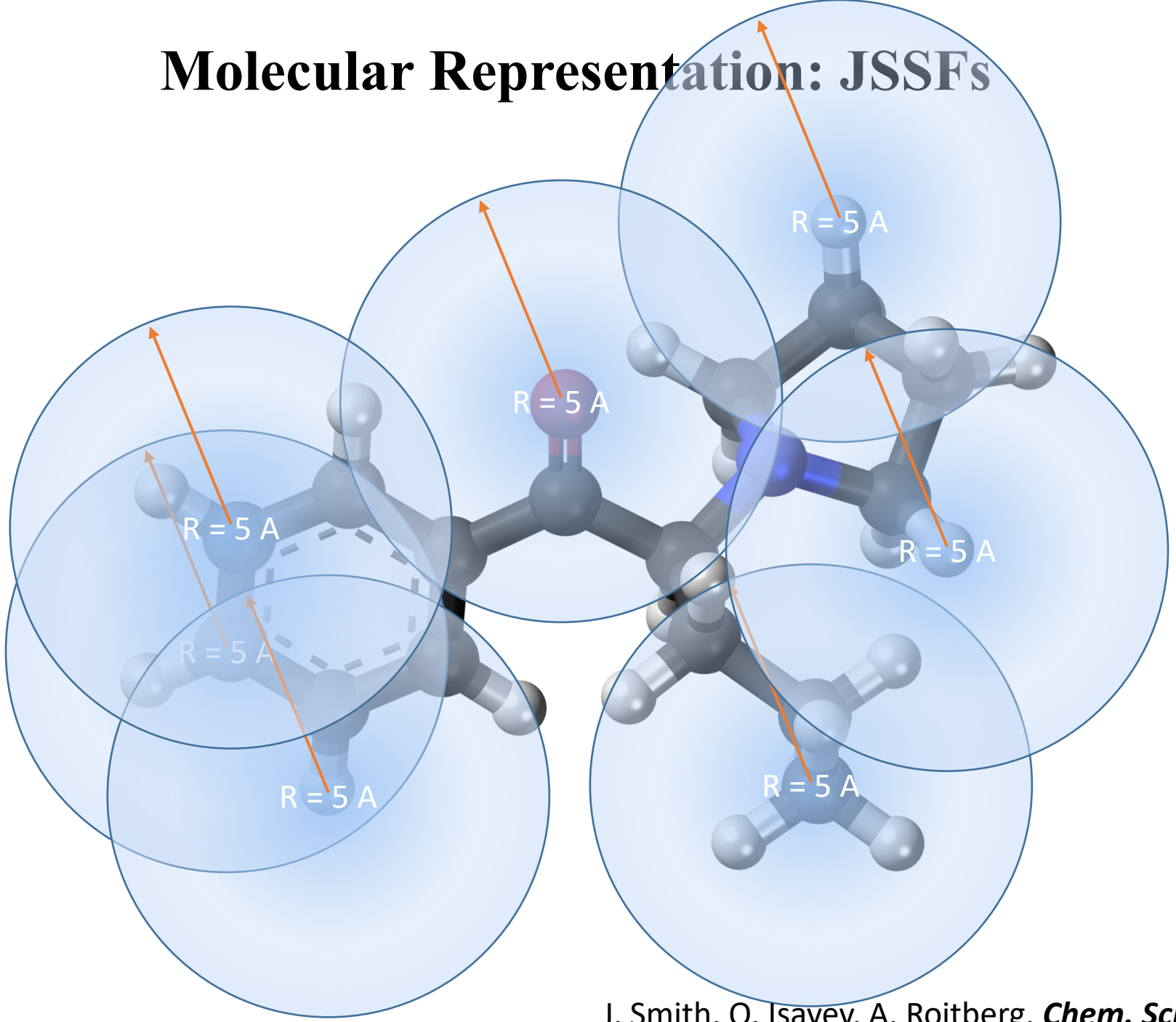
Chem. Sci., 2017, **8**, 3192-3203

Nearsightedness of Chemical Interactions*





Molecular Representation: JSSFs



J. Smith, O. Isayev, A. Roitberg. ***Chem. Sci.***, 2017, **8**, 3192-3203

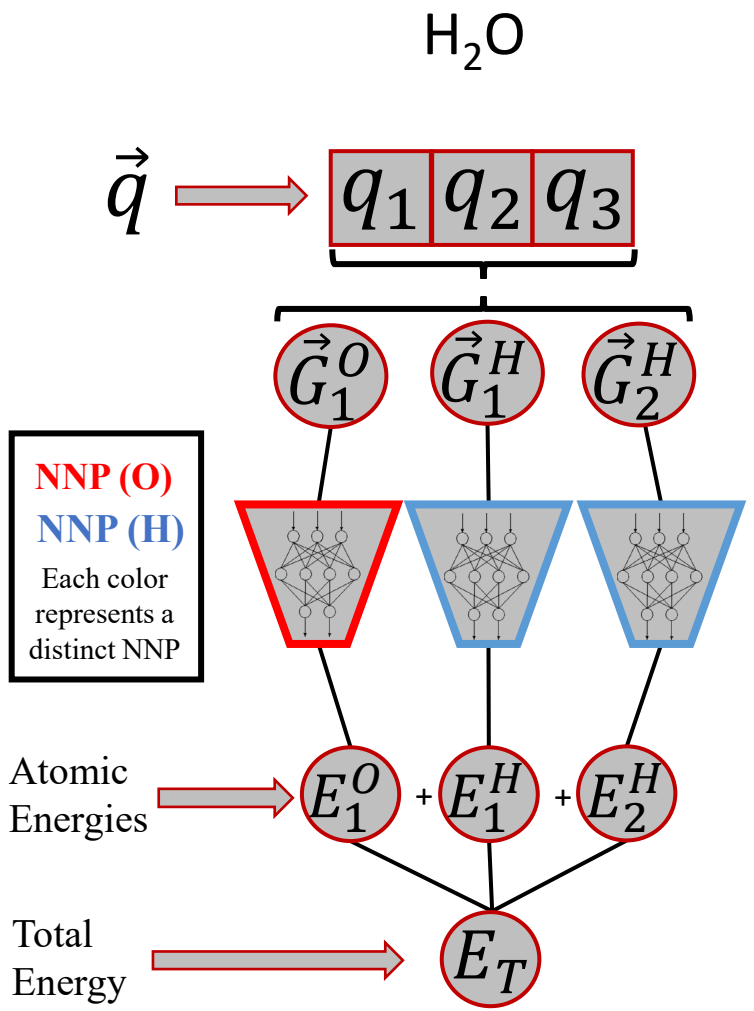
How does ANI-1 work?

Molecular representation (MR)

- Transformation from coordinates to a deep learning friendly input vector
- Accomplished through heavy modifications of Behler and Parrinello symmetry functions[1] or atomic environment vector (AEV or $\vec{G}_i^X$)
- $\vec{G}_i^X$ provides atoms local chemical environment to a cutoff radius
- Mods provide recognizable features in MR
- Mods provide better atomic number differentiation

Neural network potential

- Utilizes AEVs by computing one for each atom
- Total energy takes on a sum of atomic contributions
- Allows training to datasets with many molecules of different size (diverse)
- One NNP per atomic number



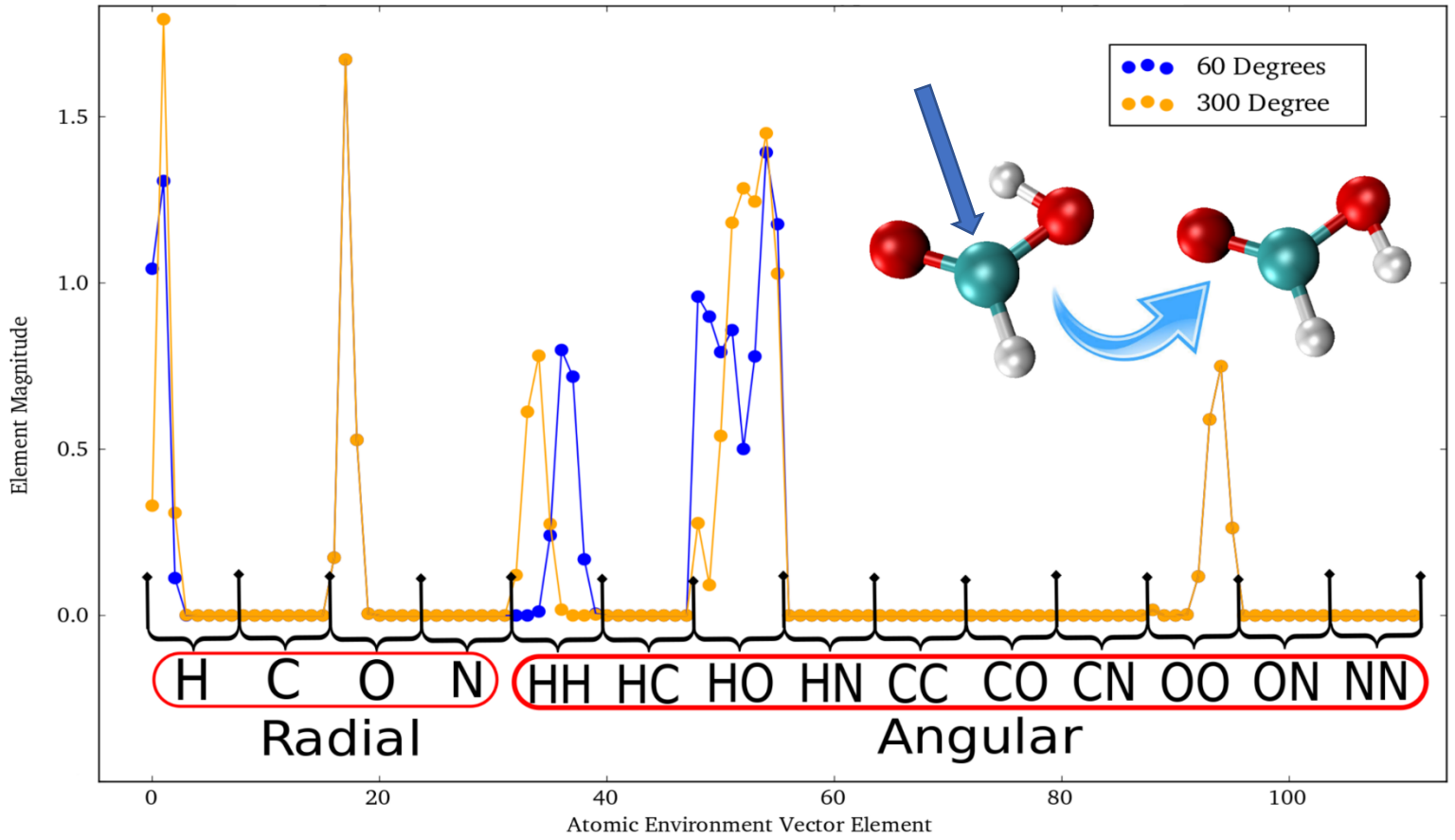
$$E_i = \mathcal{F}^{(Z_i)}(\boldsymbol{G_i})$$

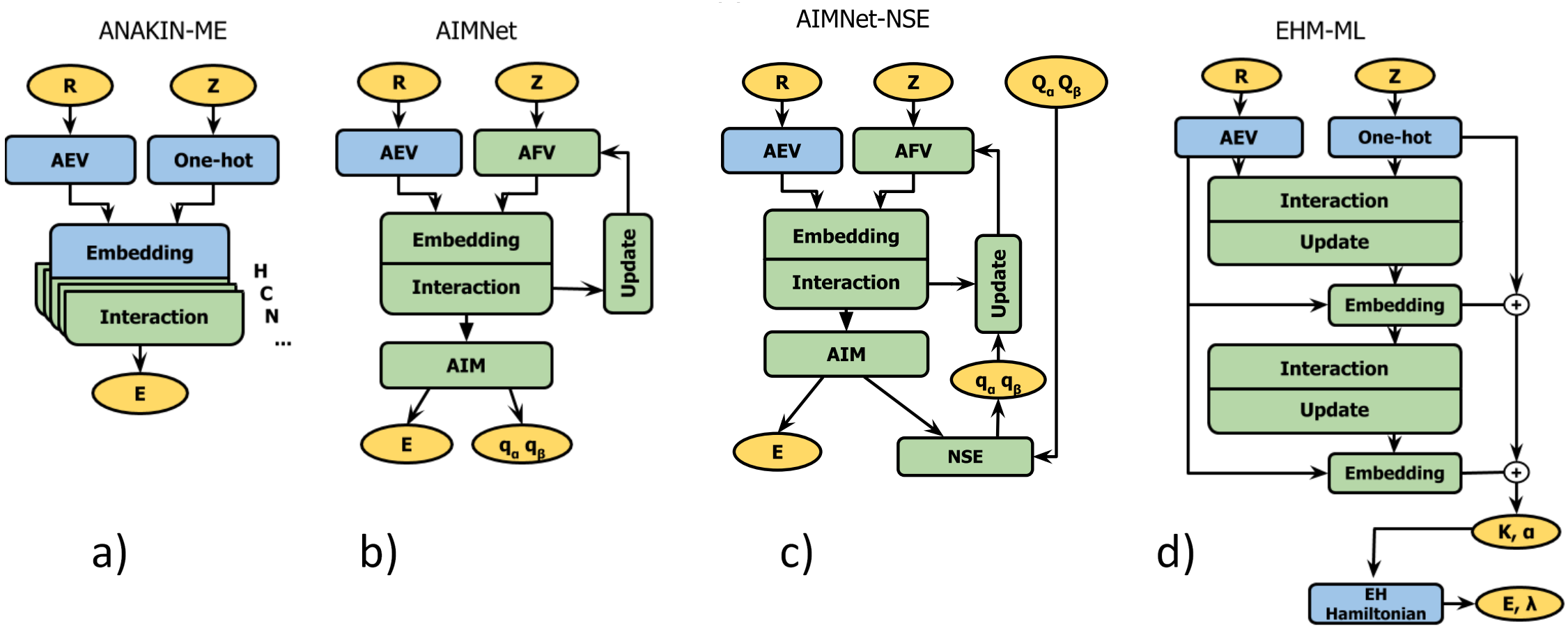
1) J. Behler and M. Parrinello, *Phys. Rev. Lett.*, 2007, **98**, 146401.

J. Smith, O.I., A. Roitberg. ***Chem. Sci.***, 2017, **8**, 3192-3203

Atomic environment vector example

Carbon atom (8 radial and 8 angular symmetry functions)





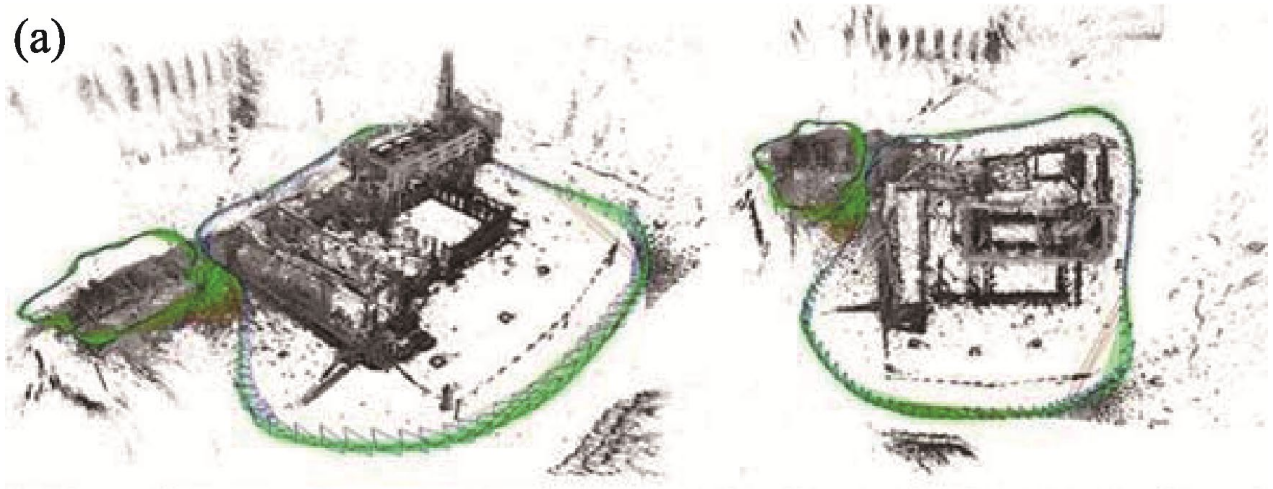
JS Smith, O.Isayev, A. Roitberg; *Journal of Chemical Physics*, (**2018**), 148 (24), 241733

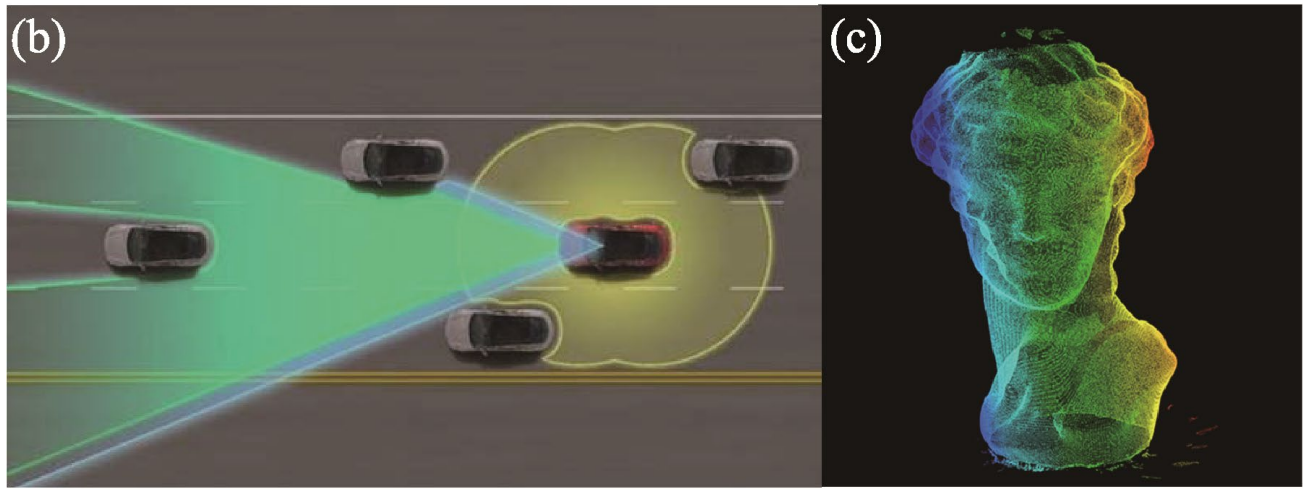
R. Zubatyuk, J.S. Smith, J. Leszczynski, O. Isayev, Accurate and Transferable Multitask Prediction of Chemical Properties with an Atoms-in-Molecule Neural Network. *Science Advances*, **2019**, 5, eaav6490

Zubatyuk, Roman; Smith, Justin; Nebgen, Benjamin T.; Tretiak, Sergei; Isayev, Olexandr (2020): Teaching a Neural Network to Attach and Detach Electrons from Molecules. ChemRxiv. Preprint. https://doi.org/10.26434/chemrxiv.12725276.v1

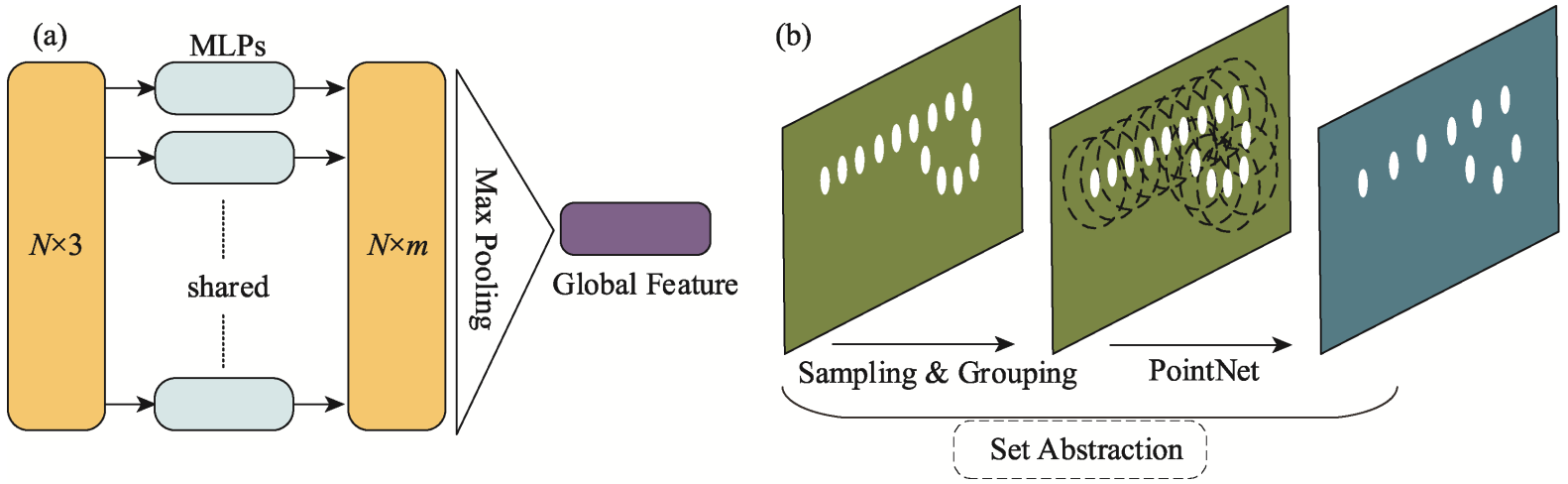
Tetiana Zubatiuk and Olexandr Isayev. Accounts of Chemical Research **2021** 54 (7), 1575-1585
DOI: 10.1021/acs.accounts.0c00868

Deep learning methods for point clouds

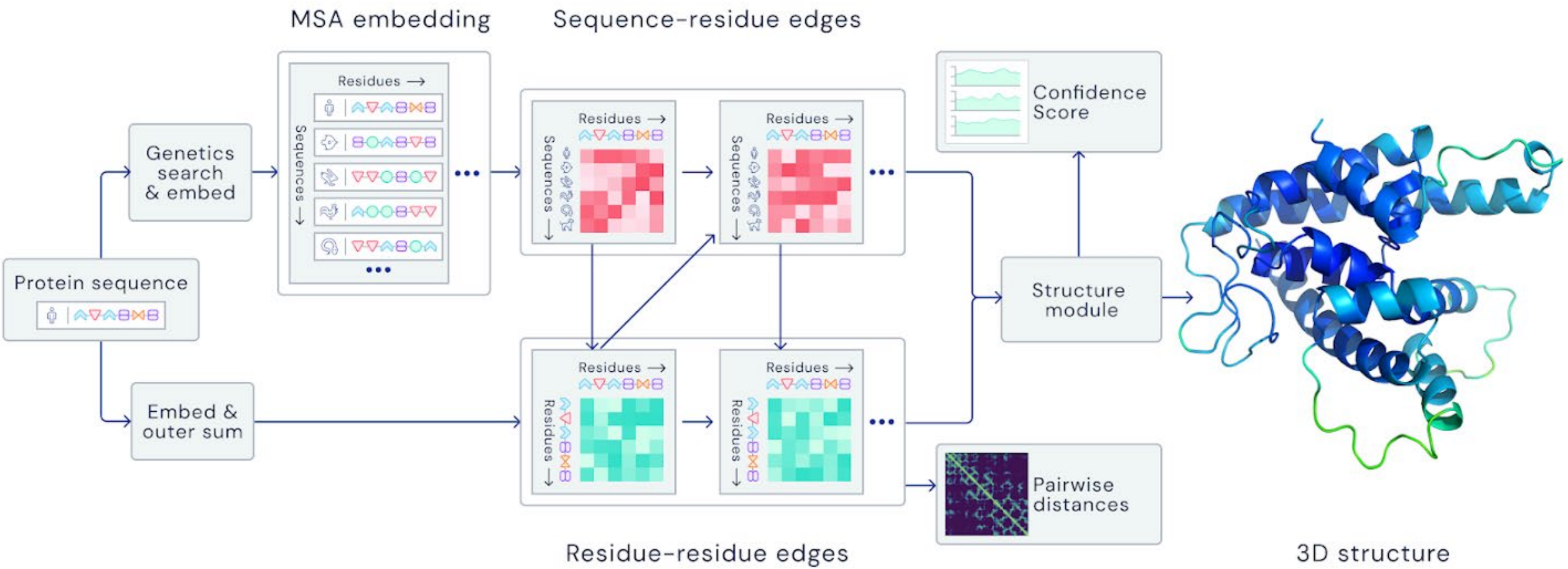




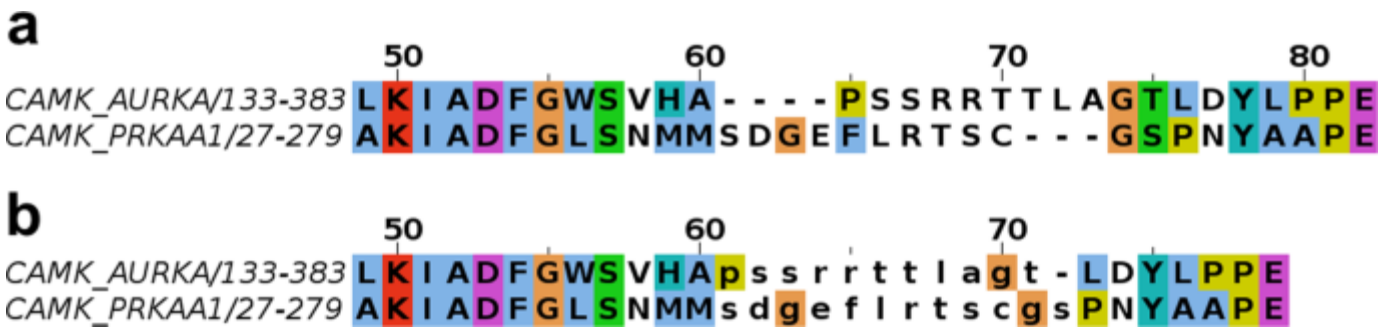
Typical network for point cloud processing

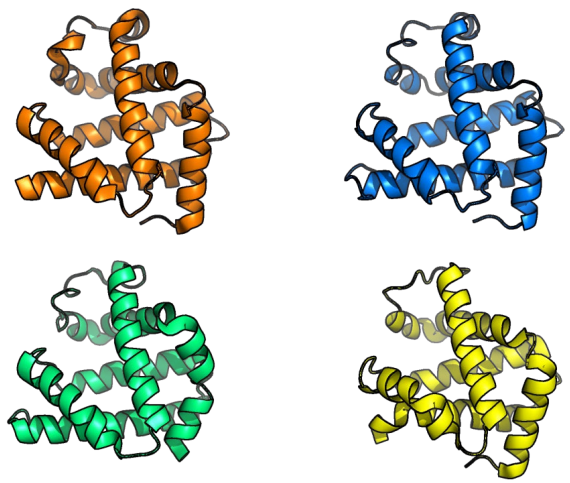


AlphaFold 2

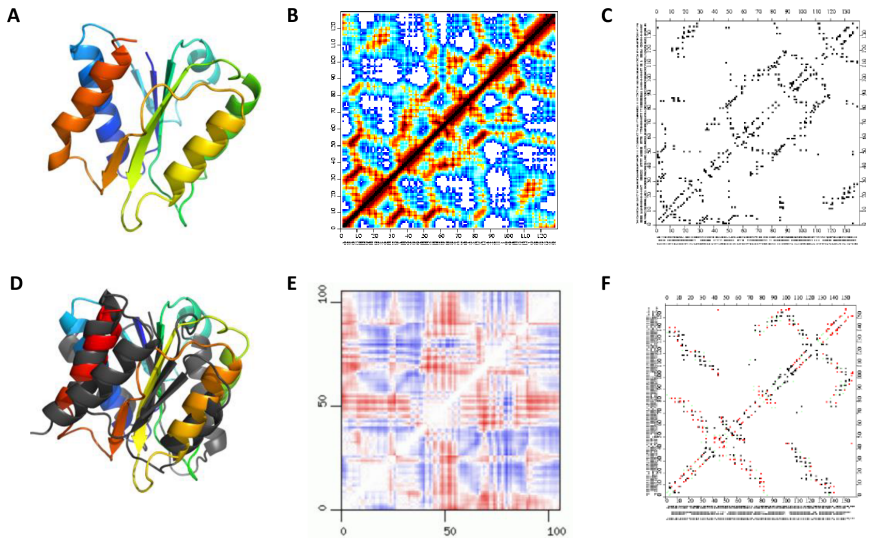

MSA embedding
Sequence-residue edges
Residues
Sequences
Genetics search & embed
Protein sequence
Embed & outer sum
Residue-residue edges
Confidence Score
Structure module
Pairwise distances
3D structure

Sequence Alignment

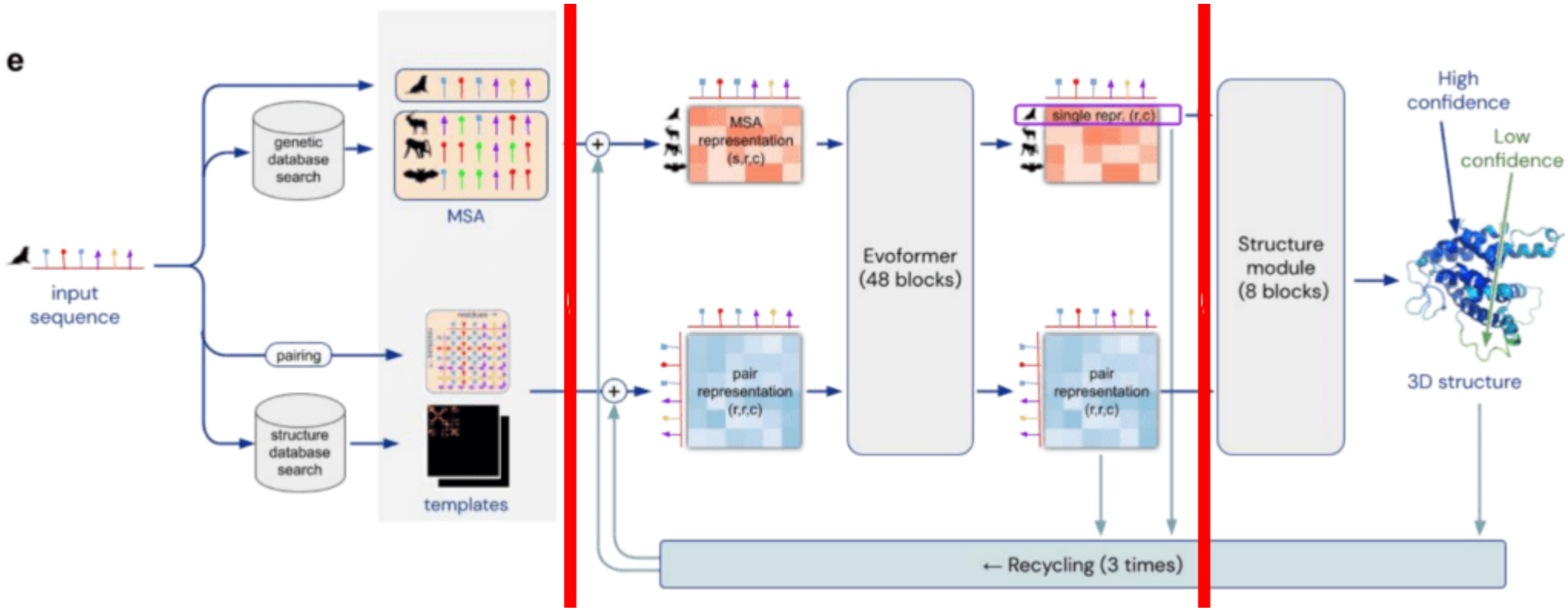




Protein Homologs and Orthologs

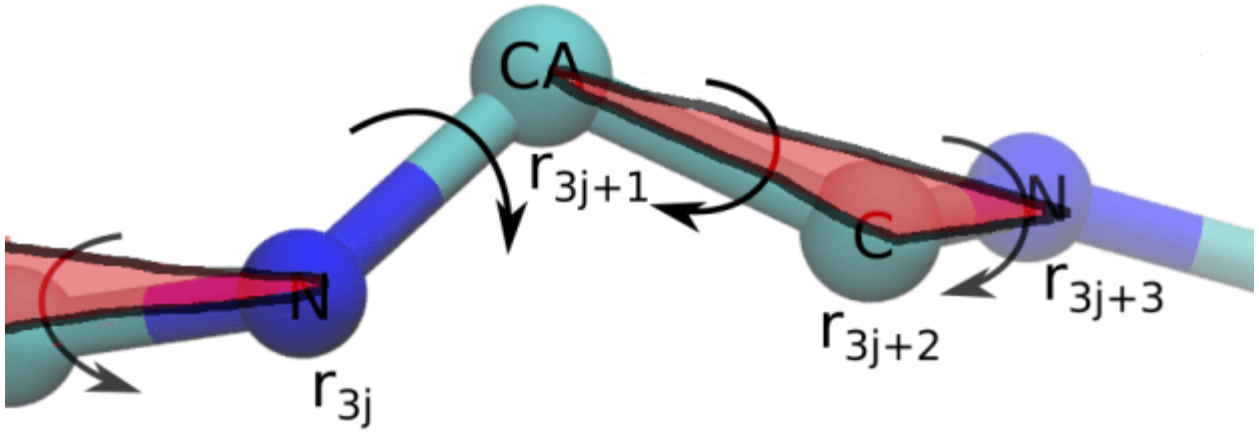


Contact Map


e
input sequence
genetic database search
MSA
pairing
structure database search
templates
MSA representation (s,r,c)
pair representation (r,r,c)
Evoformer (48 blocks)
single repr. (r,c)
pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (3 times)

The Structure Module

The structure module considers the protein as a “residue gas”. Every amino acid is modelled as a triangle, representing the three atoms of the backbone. These triangles float around in space, and are moved by the network to form the structure.



These transformations are parametrised as “affine matrices”, which are a mathematical way to represent translations and rotations in a single 4×4 matrix:

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Assignment

- Please do 1-slide update about your class projects.
- Deadline Sunday, April 16
- https://docs.google.com/presentation/d/14L--g_lbhq1MNrT_yGzHTHdC-rBQXN8ABkpjp9OGztU/edit?usp=sharing

Homework #4: Graph based NNs

In this homework, you are required to implement a graph-based neural network machine learning model to predict the target property (Regression).

There are two datasets in the data folder: train.pt, test.pt. You will train a GNN on the training dataset, then predict on the test dataset.

This HW needs to be implemented with Pytorch Geometric (PyG): https://pytorch-geometric.readthedocs.io/en/latest/

Homework #4: Graph based NNs

You are allowed to build any type of CNN regression model using PyG (PyTorch Geometric)!

You are allowed to use any type of data processing (data augmentation).

Please explore training from scratch or finetuning/pertaining existing model.

Find the best performing model

Use your model to score test.pt