INTRODUCTION TO GRAPH NEURAL NETWORKS

Savannah Thais
Columbia University
03/16/2025
GDS Short Course

What are graphs?

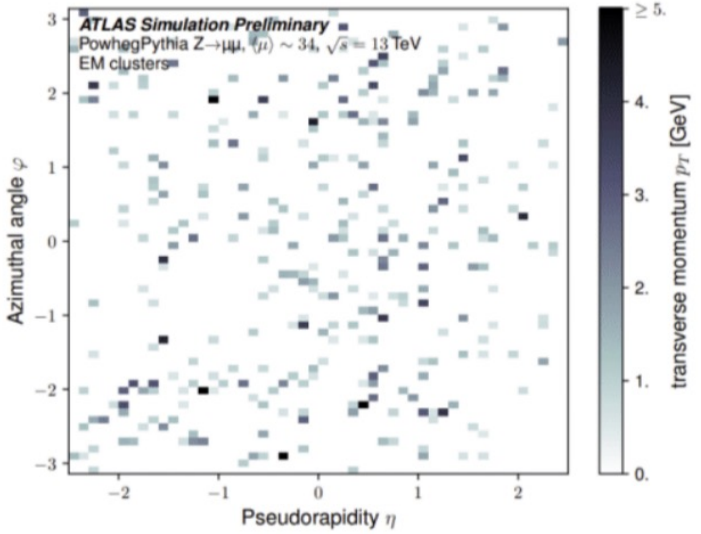
How Should We Represent Data?

If we want to use deep learning, what data format should we use?

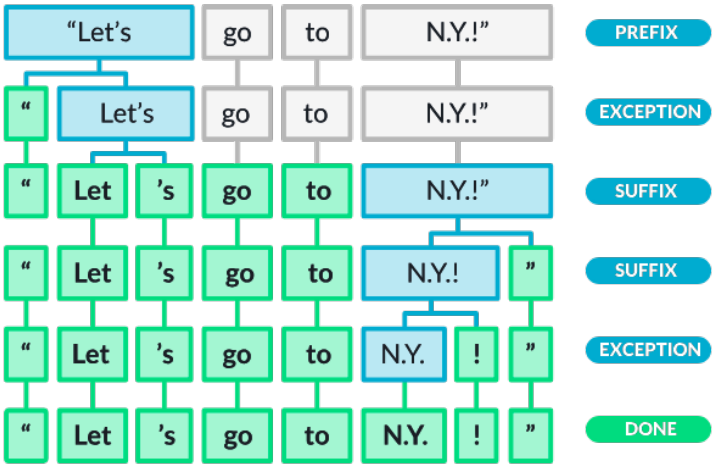
Tabular

	A	B	C	D	E	F
1	Country	Salesperson	Order Date	OrderID	Units	Order Amount
2	USA	Fuller	1/01/2011	10392	13	1,440.00
3	UK	Gloucester	2/01/2011	10397	17	716.72
4	UK	Bromley	2/01/2011	10771	18	344.00
5	USA	Finchley	3/01/2011	10393	16	2,556.95
6	USA	Finchley	3/01/2011	10394	10	442.00
7	UK	Gillingham	3/01/2011	10395	9	2,122.92
8	USA	Finchley	6/01/2011	10396	7	1,903.80
9	USA	Callahan	8/01/2011	10399	17	1,765.60
10	USA	Fuller	8/01/2011	10404	7	1,591.25
11	USA	Fuller	9/01/2011	10398	11	2,505.60
12	USA	Coghill	9/01/2011	10403	18	855.01
13	USA	Finchley	10/01/2011	10401	7	3,868.60
14	USA	Callahan	10/01/2011	10402	11	2,713.50
15	UK	Rayleigh	13/01/2011	10406	15	1,830.78
16	USA	Callahan	14/01/2011	10408	10	1,622.40
17	USA	Farnham	14/01/2011	10409	19	319.20
18	USA	Farnham	15/01/2011	10410	16	802.00

Image



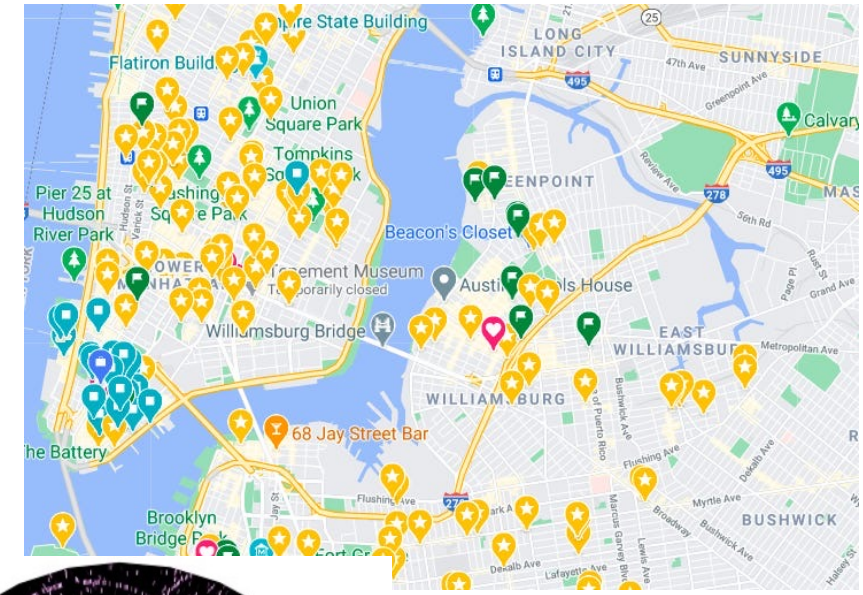
Sequence

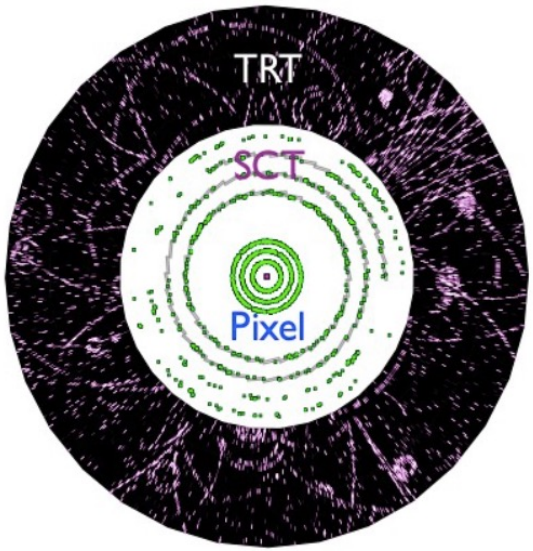


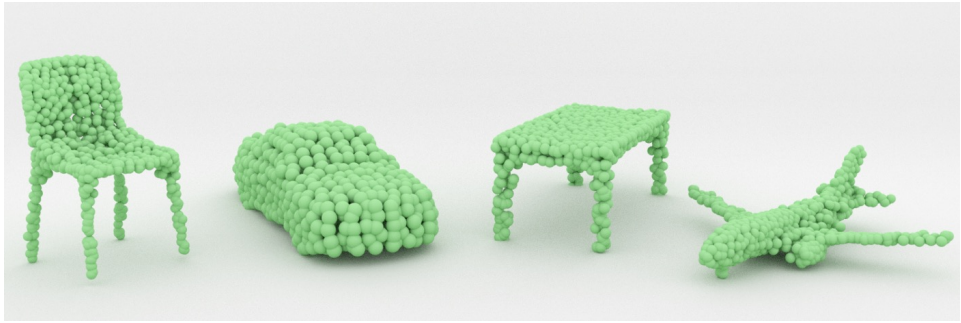
Point Cloud

Point Clouds

Many types of data can be represented as a point cloud

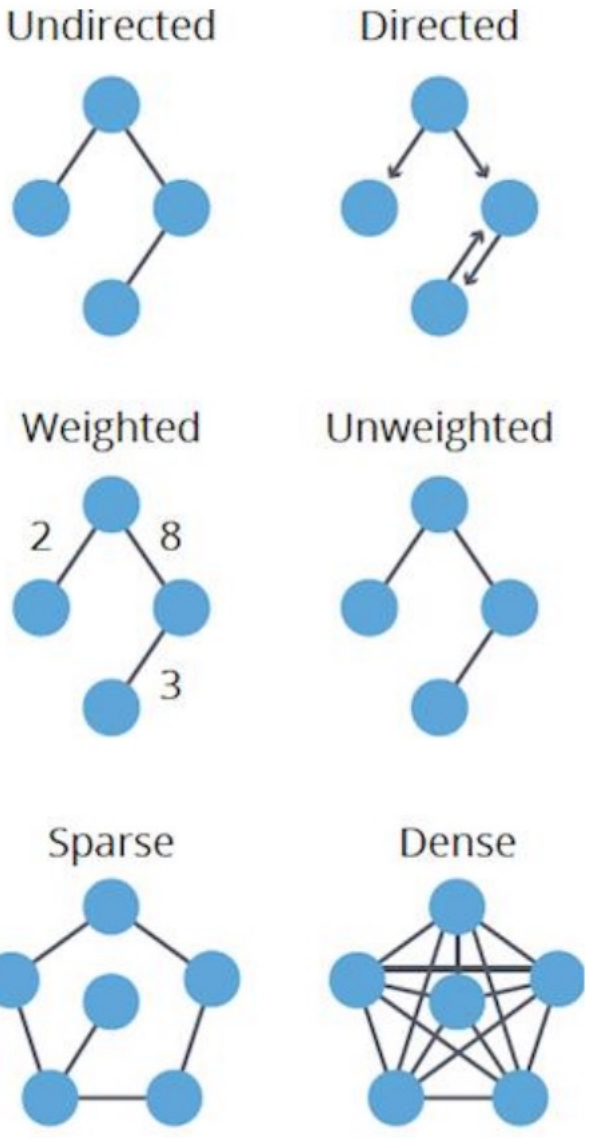






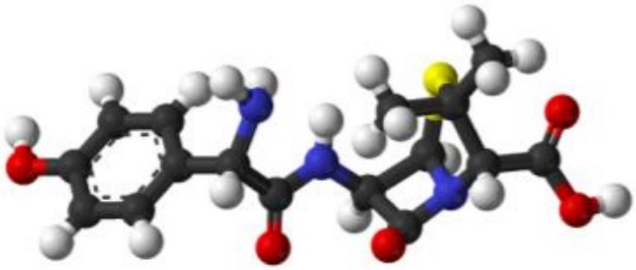
Graphs

- If we add relational information to a point cloud we get a graph
  - Nodes: vertices $u \in V$ with associated information $\mathbf{x}_u \in \mathbb{R}^{d_V}$
    - spatial coordinates, features, etc
  - Edges: connections between nodes $(u, v) \in E$
    - Can be directed or undirected, can have associated information $\mathbf{e}_{u,v} \in \mathbb{R}^{d_E}$
  - Graphs can represent many types of relational/geometric data
- Inherent geometric inductive bias
  - By including edges we encode information about data structure and can localize computation

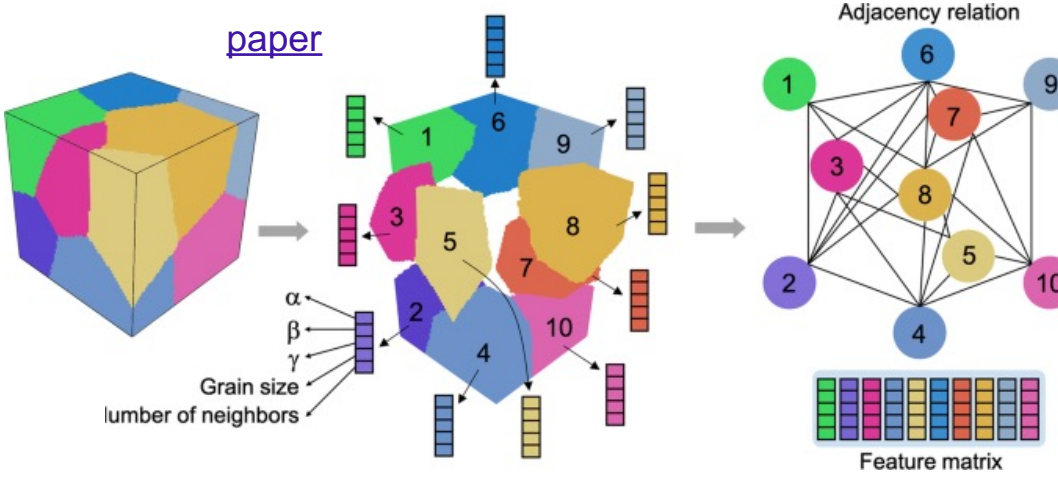


Graphs for Scientific Systems

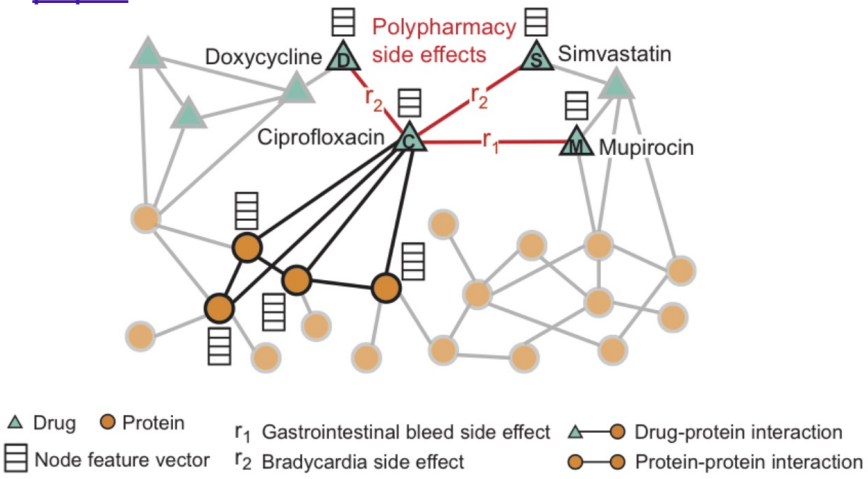
Graphs are an intuitive representation for all kinds of geometric, structured, variable length physical science data



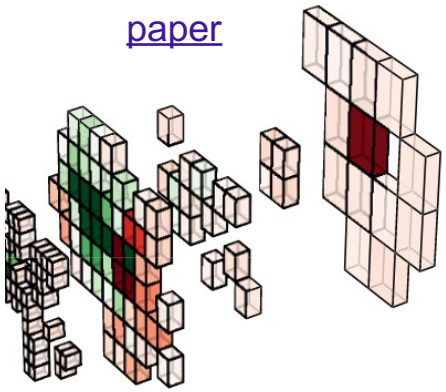
paper

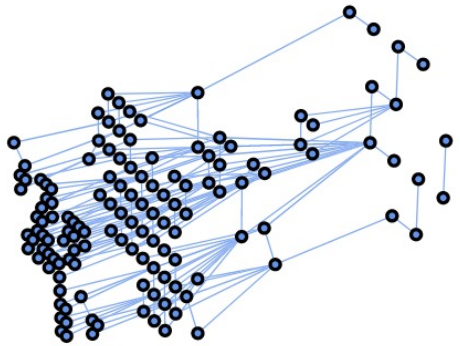


paper



paper





What are graph neural networks?

Building GNNS: Permutation Equivariance

- Many real world objects don't have natural ordering → need a permutation symmetric function of inputs: f(PX)=Pf(X)
- For example, Deep Sets:

e.g. DeepSets: take a set X and two approximators (MLPs) ϕ, ψ

$$f(X) = \phi\left(\sum_{x \in X} \psi(x)\right)$$

- ϕ: *any* permutation invariant function of countable sets
- $\sum_{x \in X}$: sum aggregation is permutation-invariant
- $\psi(x)$: latent representation of each element

- This gives a global prediction on a set, in graphs we have more structure and might require node or edge level predictions

- Instead, we can define a locally permutation invariant function

Node Neighborhoods:

- Neighborhood of u: $N(u) = \{v : (u, v) \in E\}$
- Neighborhood node features: $X_{N(u)} = \{\{x_v : v \in N(u)\}\}$
- Neighborhood edge features: $E_{N(u)} = \{\{e_{uv} : (u, v) \in E\}\}$

Permutation invariance: $f(\boldsymbol{PX}, \boldsymbol{PAP}^T) = f(\boldsymbol{X}, \boldsymbol{A})$ → graph-level predictions
Permutation equivariance: $f(\boldsymbol{PX}, \boldsymbol{PAP}^T) = \boldsymbol{P}f(\boldsymbol{X}, \boldsymbol{A})$ → node-level predictions
Permutation equivariant function of graphs:

$$f(X, A) = \begin{pmatrix} - & g(\boldsymbol{x}_1, X_{N(1)}, E_{N(1)}) & - \\ - & g(\boldsymbol{x}_2, X_{N(2)}, E_{N(2)}) & - \\ & \cdots & \\ - & g(\boldsymbol{x}_{|V|}, X_{N(|V|)}, E_{N(|V|)}) & - \end{pmatrix}$$

- $f(X, A)$: permutation equivariant function of graphs
- $g(\boldsymbol{x}_1, X_{N(1)}, E_{N(1)})$: local function operating on each node's neighborhood… needs to be permutation invariant!
- equivariance enforced by applying g to all nodes equally

Slide from Gage DeZoort

Building GNNs: Message Passing

Message Passing (MPNN) Layers:
Framework for many equivariant graph updates

At each layer *k*, compute messages in each node's neighborhood:

$$\boldsymbol{m}_{uv}^{(k)} = \psi^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}, \boldsymbol{e}_{uv}^{(k-1)}\right)$$

Aggregate messages in a permutation-invariant way:

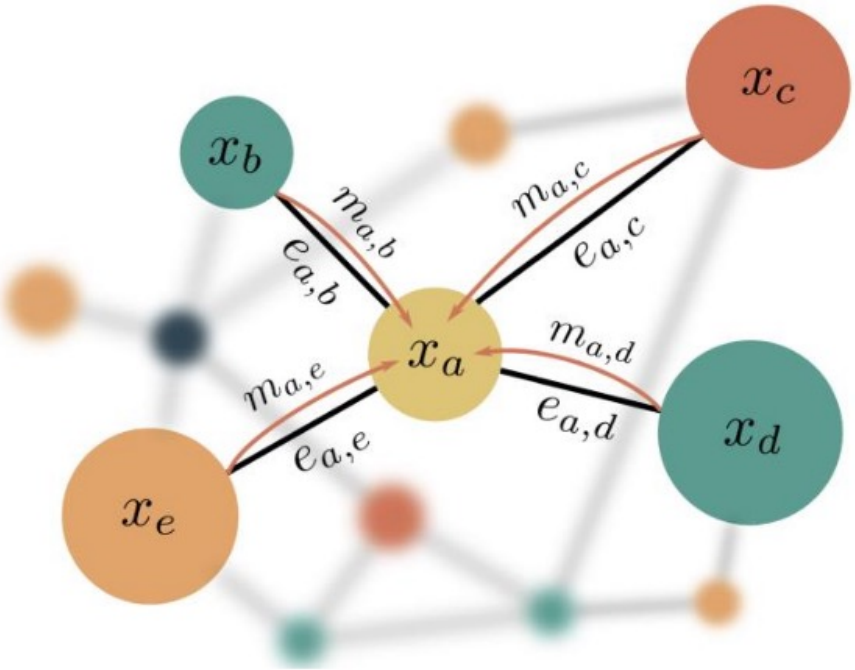
$$\boldsymbol{a}_u^{(k)} = \bigoplus_{v \in N(u)} \boldsymbol{m}_{uv}^{(k)}$$

Messages passed only from u's direct neighbors

Any permutation invariant operation (e.g. sum, mean, max)

Update the node's state based on the messages it received:

$$\boldsymbol{h}_u^{(k)} = \phi^{(k)}(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{a}_u^{(k)})$$



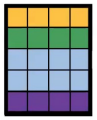
Slide from Gage DeZoort

Graph Neural Networks

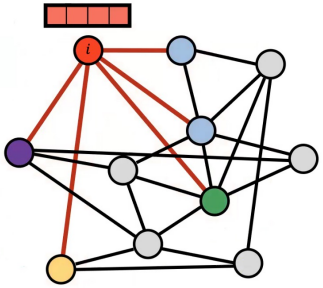
The goal of a (or at least some) GNN(s) is to learn a smart re-embedding of the graph data that preserves the relational structure but makes it easier to solve some downstream task

The most general version:

multiset of neighbour features



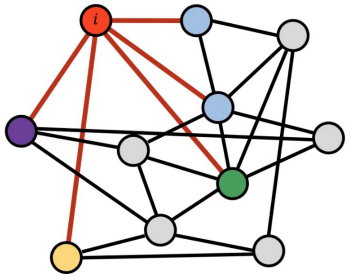
$$\mathbf{X}_{\mathcal{N}_i} = \{\!\{\mathbf{x}_{j \in \mathcal{N}_i}\}\!\}$$



local function

$$\phi\left(\mathbf{x}_i, \mathbf{X}_{\mathcal{N}_i}\right)$$

permutation invariant



$$f(X, A) = \begin{pmatrix} - & g(\boldsymbol{x}_1, X_{N(1)}, E_{N(1)}) & - \\ - & g(\boldsymbol{x}_2, X_{N(2)}, E_{N(2)}) & - \\ & \dots & \\ - & g(\boldsymbol{x}_{|V|}, X_{N(|V|)}, E_{N(|V|)}) & - \end{pmatrix}$$

permutation equivariant function of graphs

local function operating on each node's neighborhood... needs to be permutation invariant!

equivariance enforced by applying g to all nodes equally

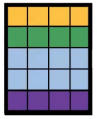
permutation-invariant aggregation operator, e.g. sum

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

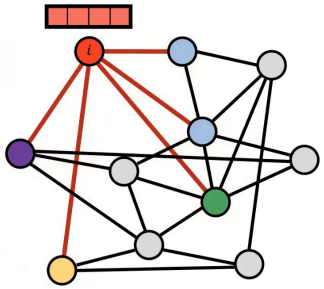
learnable functions

GNNs: Nodes Updates

multiset of neighbour features



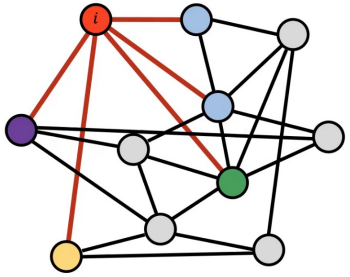
$$\mathbf{X}_{\mathcal{N}_i} = \{\mathbf{x}_{j \in \mathcal{N}_i}\}$$



local function

$$\phi\left(\mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\right)$$

permutation invariant



Several variations on input to learnable functions:

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

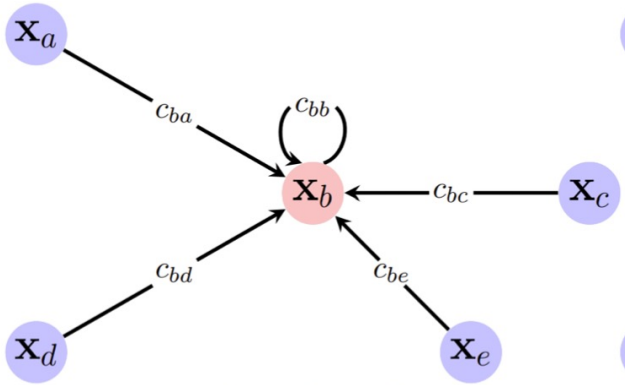
"convolutional"

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

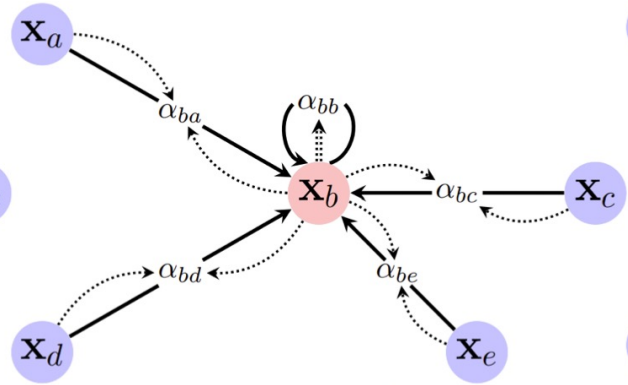
"attentional"

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

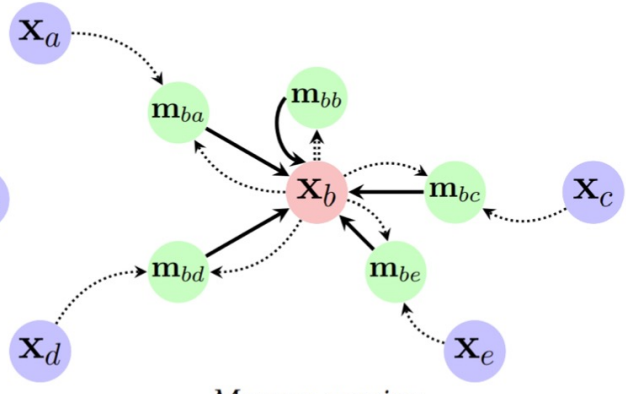
"message passing"



Convolutional



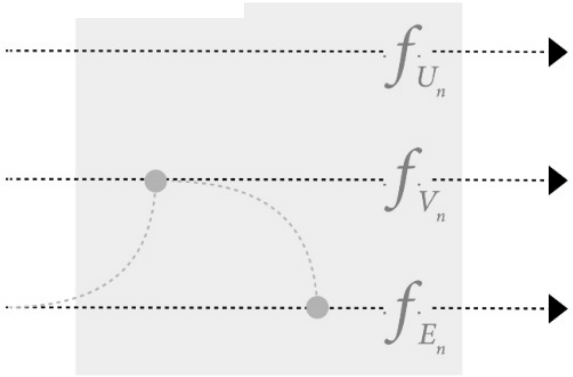
Attentional



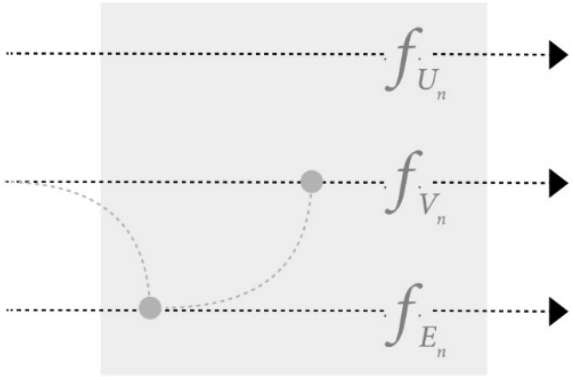
Message-passing

GNNs: Edge Updates

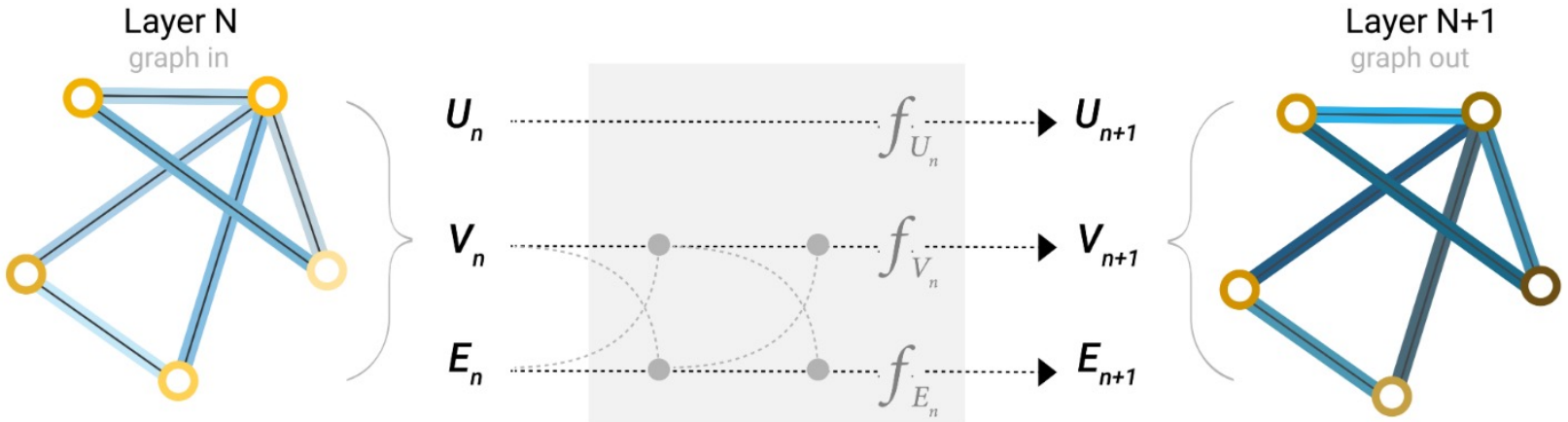
$$e_{t+1} = f_R(o_{1,t}, o_{2,t}, r)$$



Node Then Edge Learning



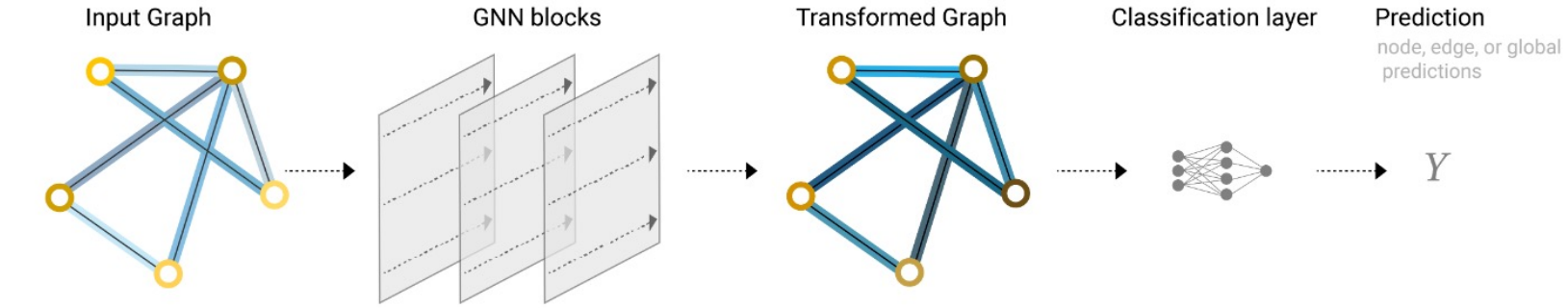
Edge Then Node Learning



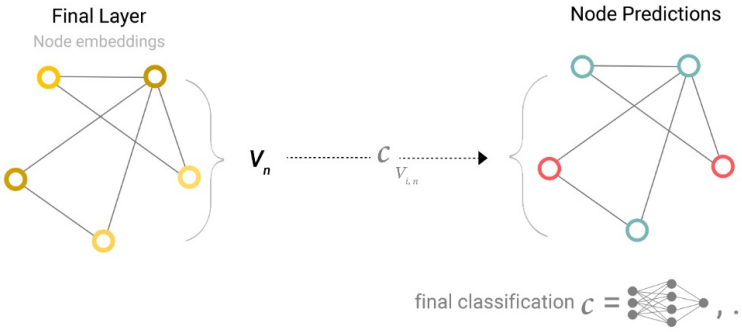
Weave Layer

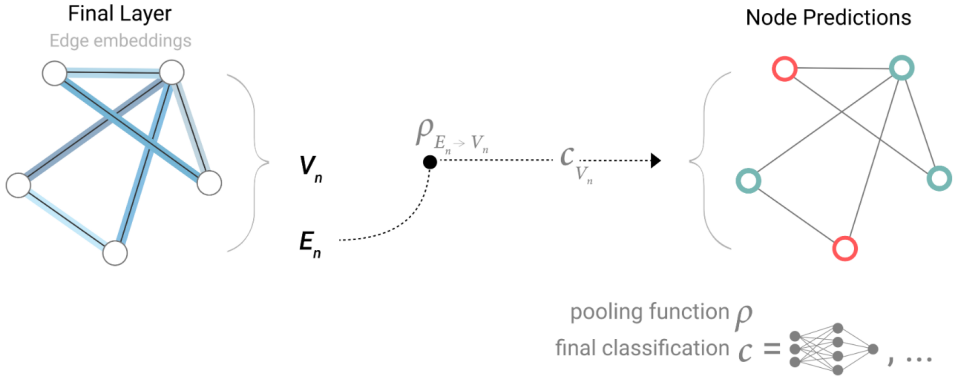
Images from Distill

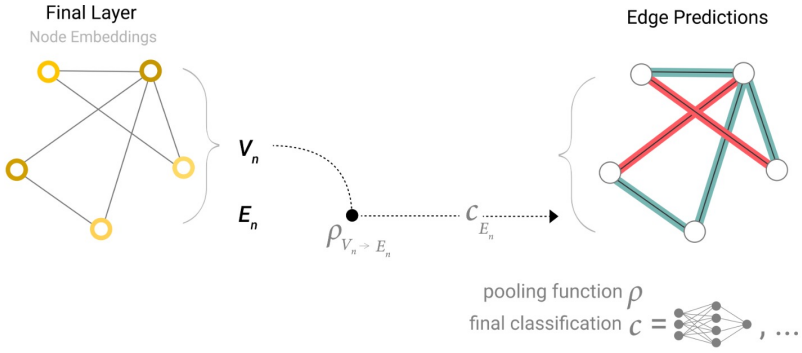
Using GNNs

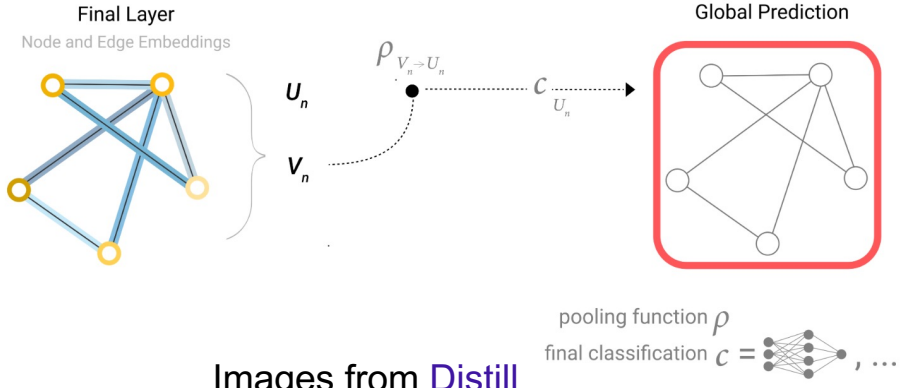


An end-to-end prediction task with a GNN model.





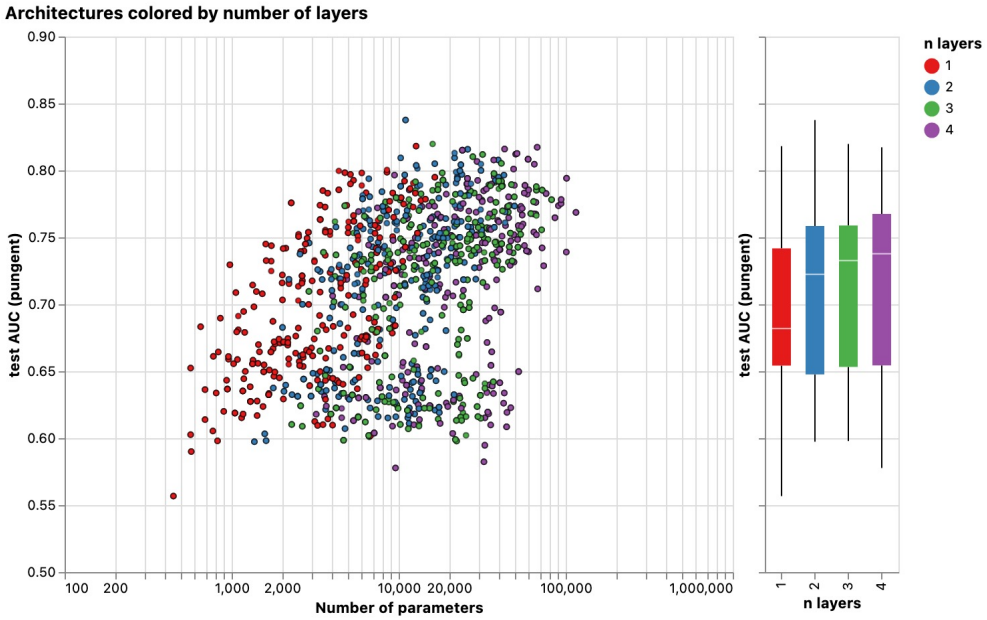


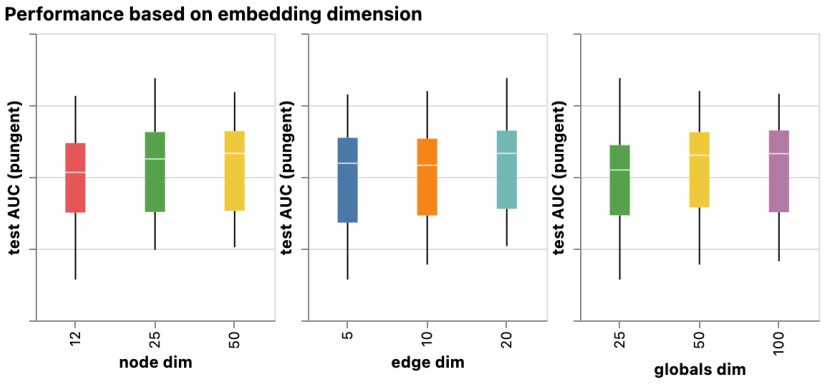


Images from Distill

Key Considerations

- Graph construction has a large impact on performance
  - Edges can represent real connections or simply be a route for information flow
- Relationship between network size and performance is unclear
  - GNNs generally parameter efficient, small networks can perform well
  - But message construction networks are generally small, can be an issue for large graphs



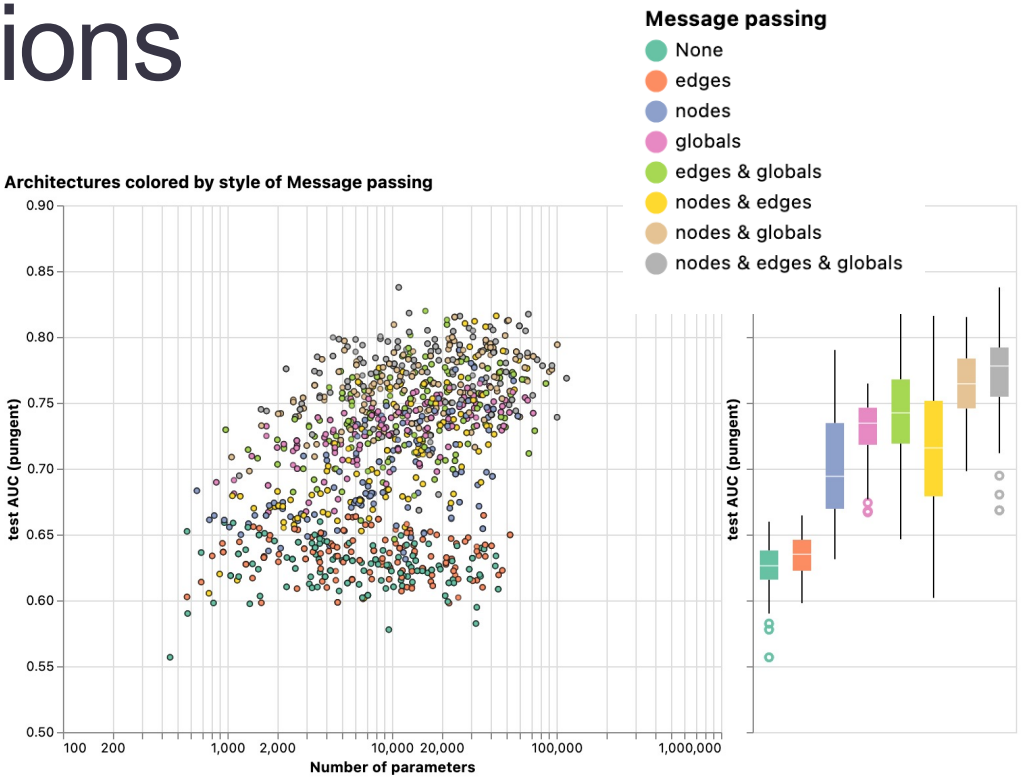


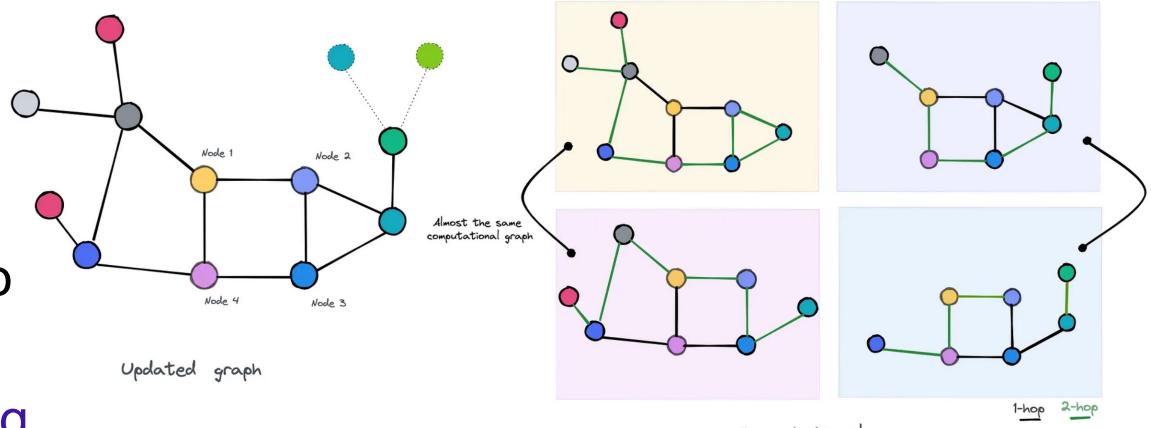
Aggregate performance of models across varying node, edge, and global dimensions.

Images from Distill

Key Considerations

- Relationship between architecture and expressivity is unclear
  - More complex messages tend to yield better performance
  - Different aggregations yield different performance
- GNNs have some limitations
  - Most cannot distinguish graph isomorphisms
  - Vulnerable to adversarial attacks and noise (similar to CNNs)
  - Vulnerable to oversmoothing



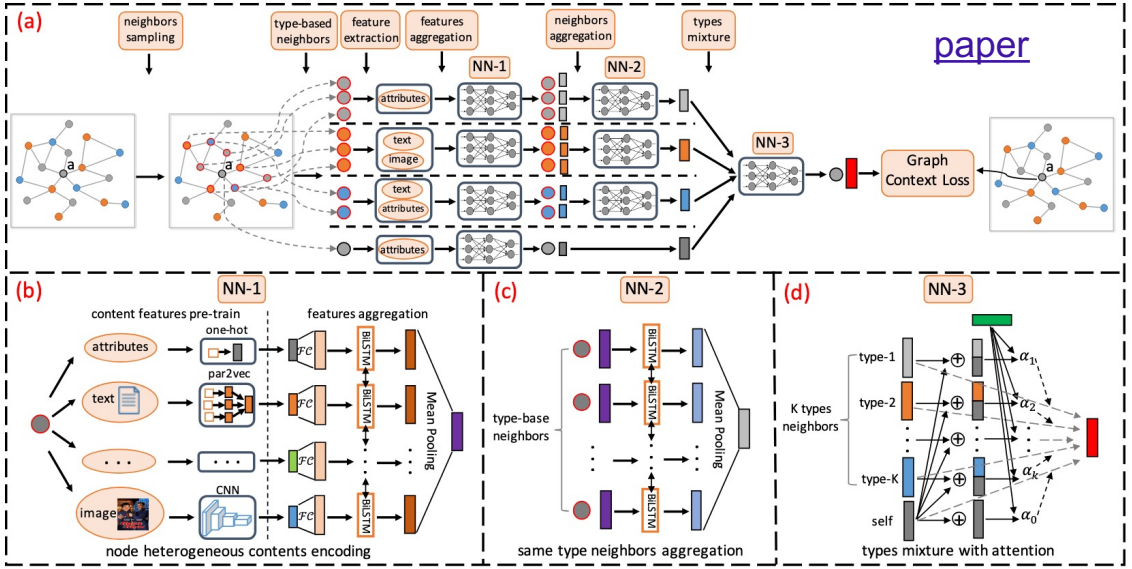


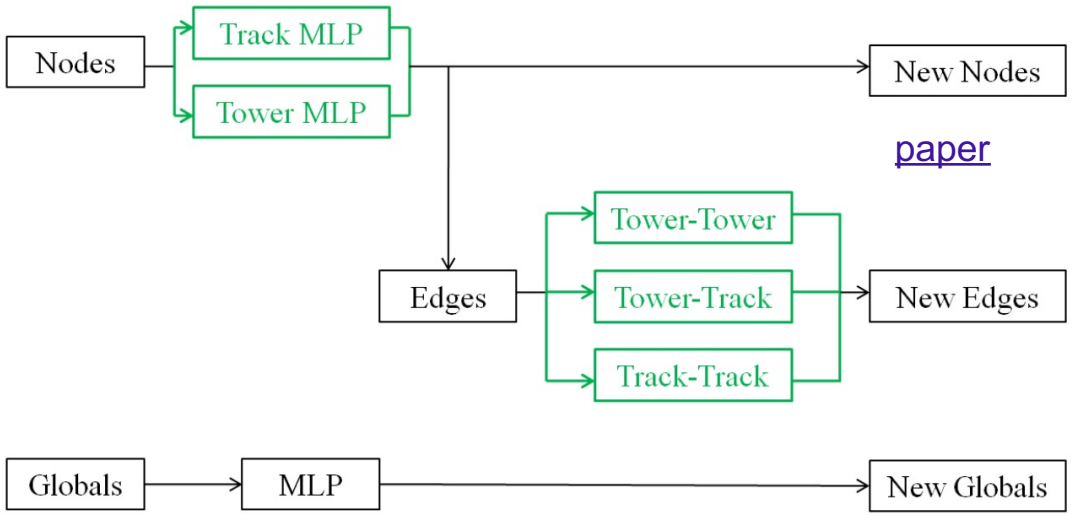
Images from Distill

Active/Advanced Topics in GNNs

Heterogenous Graphs

- Different types of nodes
  - Can have the same features + a type label
  - Or different features + separate message constructions
- Different types of edges (multigraphs)
  - Can use a type label (for edge classification or prediction)
  - Or have different message passing schemes for different edge types

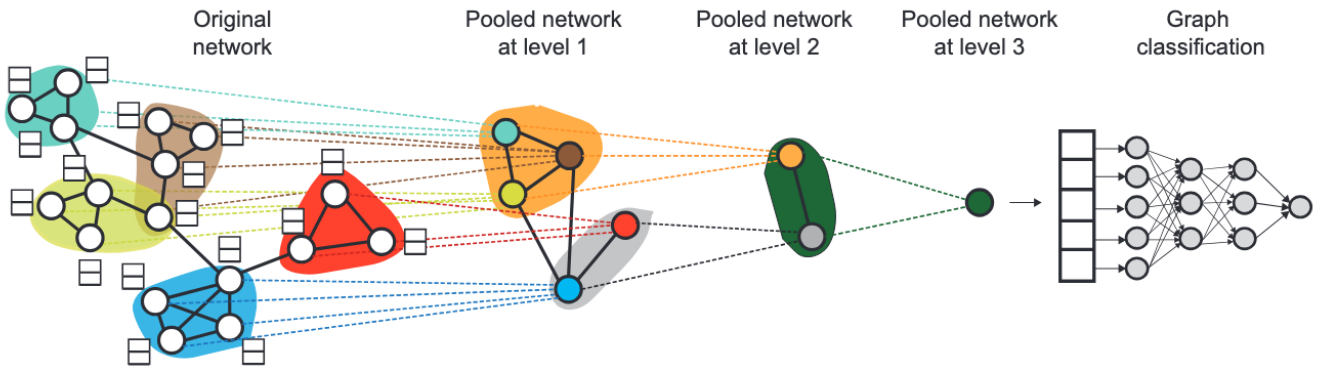




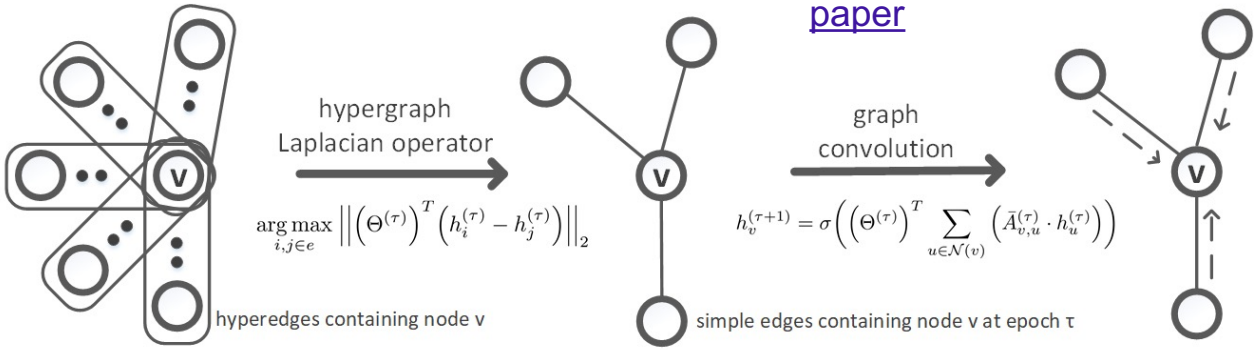
Hyper + Hierarchical Graphs

- Hypergraphs connect multiple nodes per edge
  - One approach is using the graph Laplacian to create a representation of the hypernodes then using a traditional GNN

paper



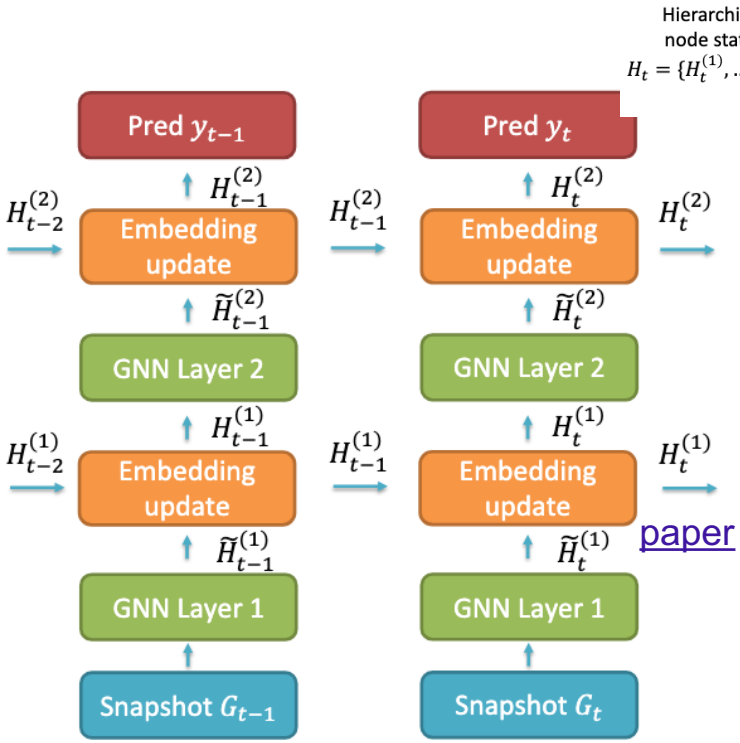
paper



- Hierarchical graphs introduce pooling nodes or nodes that represent full graphs
  - Can use nested GNNs to learn hierarchical representations
  - Requires a differentiable pooling operator

Dynamic Graphs

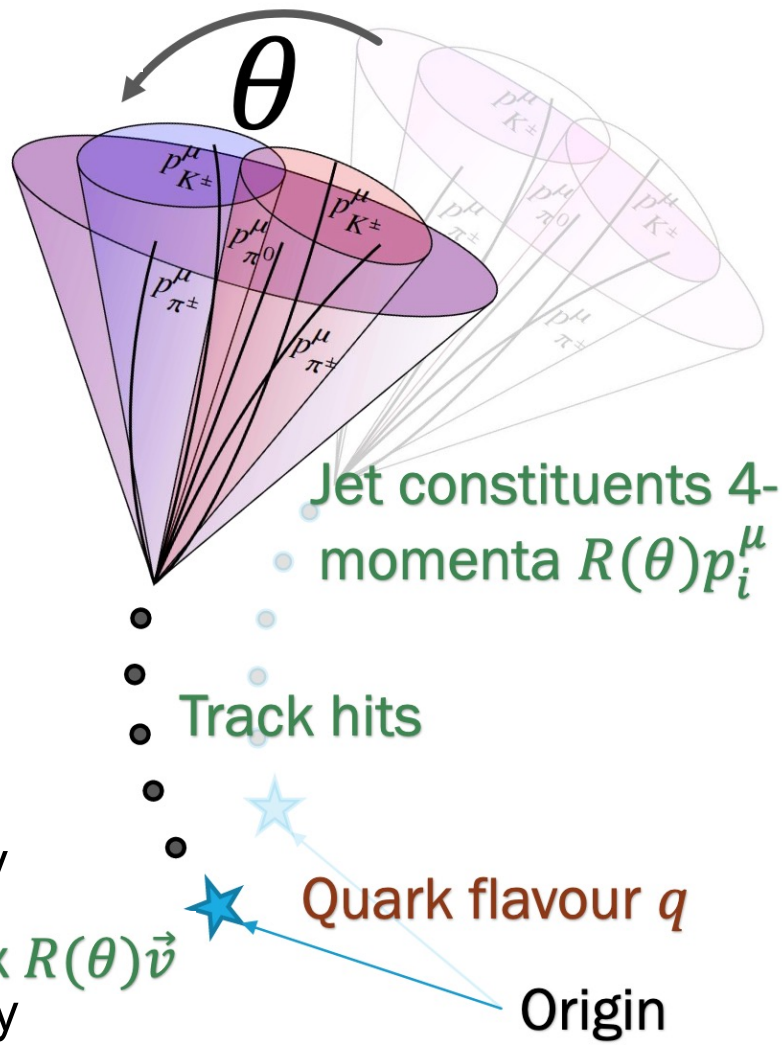
- Many interesting graphs change over time
  - New edges are formed: new social connections, new transactions, etc
  - Nodes are added or removed: new people in network, new discoveries, etc
- Can convert to a static graph by temporal aggregation or unrolling
- Recent approaches with dynamic edges use an embedding update layer
  - Combines messages from previous graph state with messages from current graph state to generate new node embeddings
- Can also use modified transformers or RNNs combined with GNNs
  - Further reading



paper

Including Symmetries

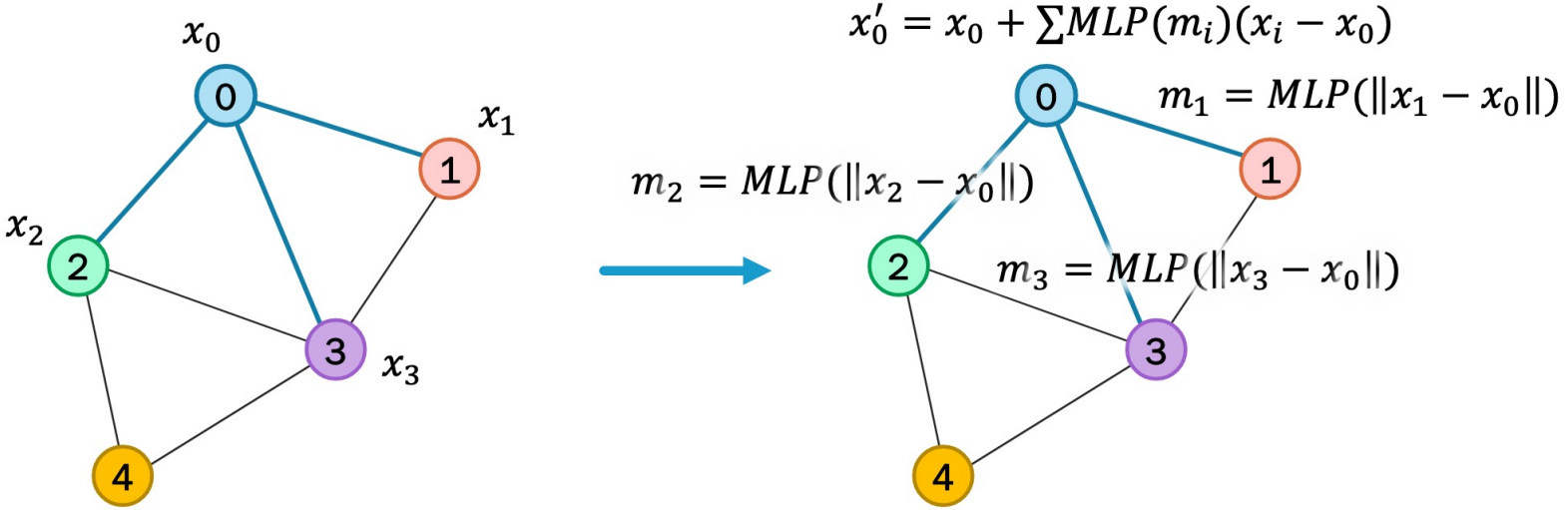
- Consider rotating a jet by angle ϕ, using rotation matrix $R(\theta)$
  - Some predictions (and input features) like the production vertex will rotate with the transformation: "equivariant"
  - Some predictions (and input features) like the jet flavor should not be affected: "invariant"
- Respecting symmetries could have numerous benefits
  - Improved model accuracy and efficiency
  - Reduced training time/resources
  - Better interpretability and generalizability



One Way To Do This

- Consider a point cloud, with behavior that you expect to be invariant under E3 symmetry – 3 dimensional Euclidean (rotational and translational) transformations
- Observe how a transformation R propagates in some arbitrary GNN convolution
- To preserve E3 symmetry, we must choose a specific kind of message passing function:

$$x_0' = x_0 + \sum MLP(m_i)(x_i - x_0)$$



$$\begin{aligned}\|Rx_3 - Rx_0\|^2 &= (Rx_3 - Rx_0)^T(Rx_3 - Rx_0) \\ &= (x_3 - x_0)^T R^T R(x_3 - x_0) \\ &= \|x_3 - x_0\|^2\end{aligned}$$

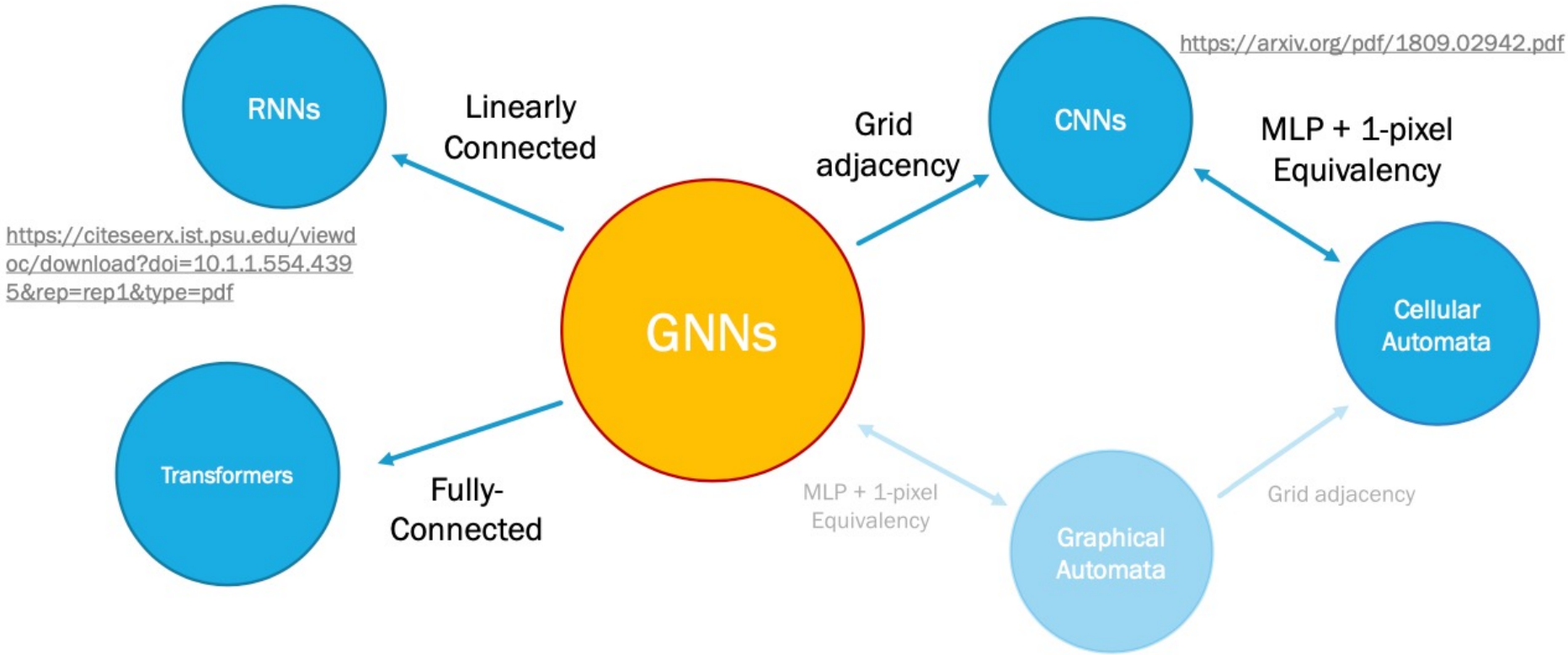
Message passing invariant to rotation and translation

Aggregation equivariant to rotation and translation

$$Rx_0 + \sum MLP(m_i)(Rx_i - Rx_0) = Rx_0'$$

E(N) Equivariant Graph Neural Networks, Sartorras et al

GNNs As General Models



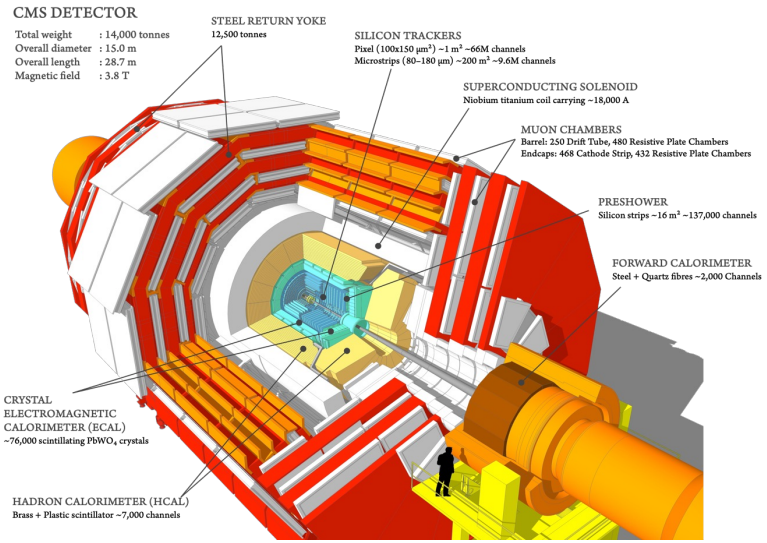
https://github.com/murnanedaniel/GNN-as-Transformer-as-GNN/blob/main/0-Transformer_vs_GNN_Annotated.ipynb
https://arxiv.org/pdf/2012.09699.pdf

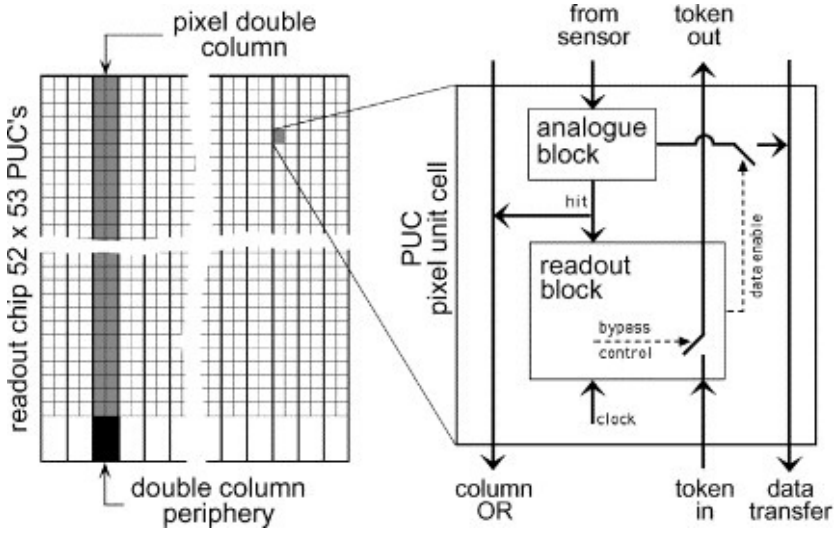
Diagram from Daniel Murnane

Science Applications of GNNs

Particle Physics: LHC Data

- Collision data measured by dedicated subsystems
  - Quantifies interactions with highly granular detectors
  - Readouts must be reconstructed into particle components (tracks, clusters) then full particle candidates and event information
- We can only measure SM particles, so we must (accurately) extrapolate what happened during the initial collision
- Poses many computing challenges
  - Non-fixed size, heterogenous data
  - Varying density/sparsity
  - Very tight computing time and resource constraints

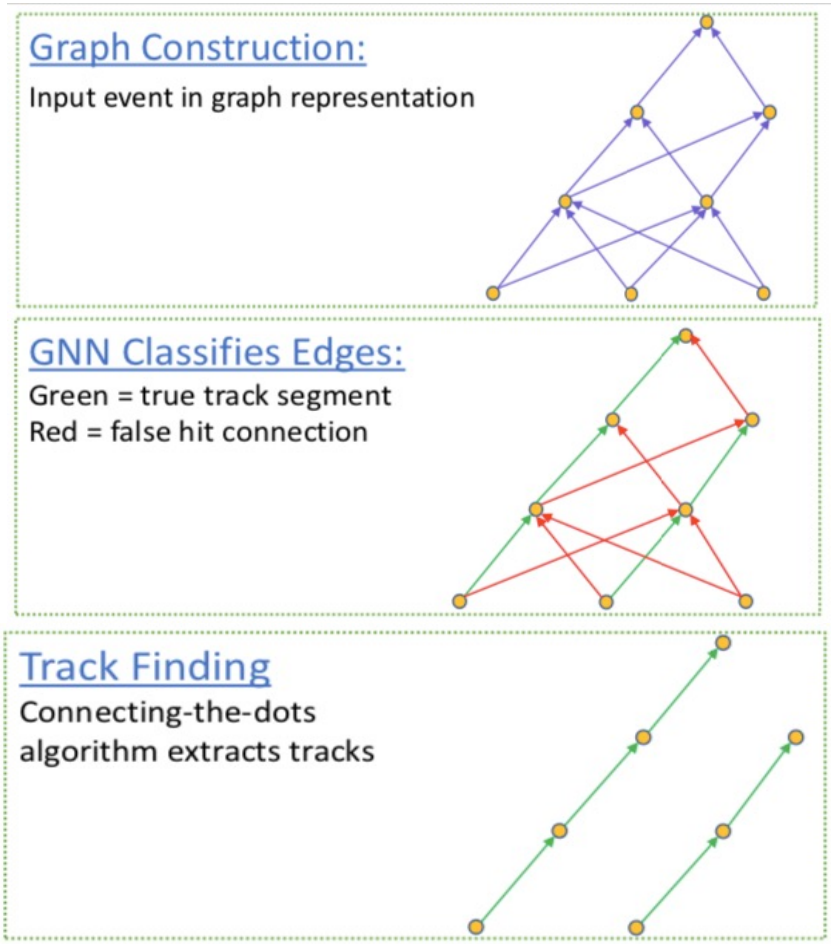


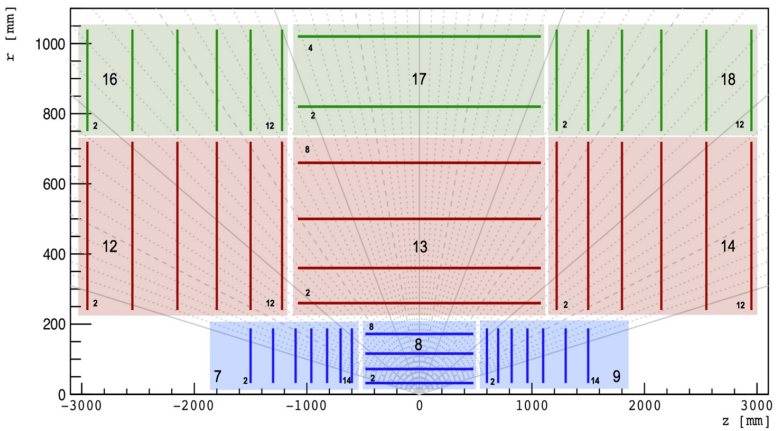


GNNs for Tracking

Basic procedure

1. Form initial graph from spacepoints/hits (pre-processing)
2. Process with GNN to get probabilities of all edges
3. Apply post-processing algorithm to link edges together into tracks and get parameters



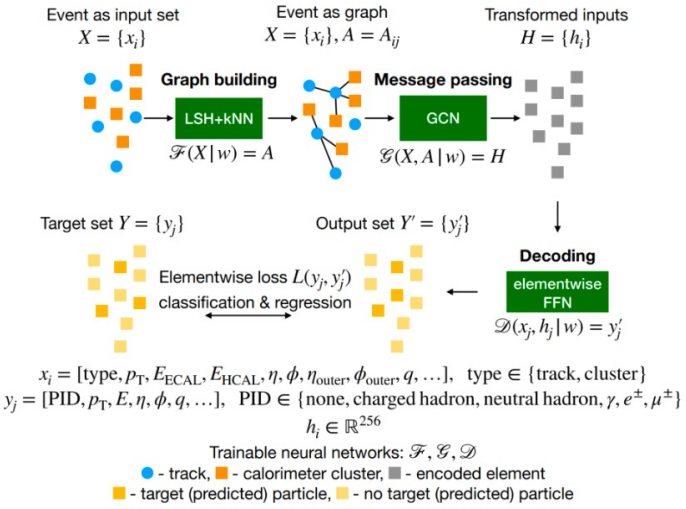


- Many places to improve/innovate
  - Graph construction, architectures, data augmentation…
- Many recent papers (and an open source data set!)
  - Charged particle tracking via edge-classifying interaction networks
  - Performance of a geometric deep learning pipeline for HL-LHC particle tracking
  - Hierarchical Graph Neural Networks for Particle Track Reconstruction
  - Equivariant Graph Neural Networks for Charged Particle Tracking

Particle ID

Particle flow (PF) algorithms combine tracks and calorimeter clusters to form physics objects (electrons, photons, muons, etc.)

PF via Node Classification:
A recent GNN-based PF algorithm considered *heterogeneous nodes* representing tracks, electromagnetic calorimeter (ECAL) clusters, and hadronic calorimeter (HCAL) clusters



Segmentation, classification, and regression → multi-objective learning (one-shot)

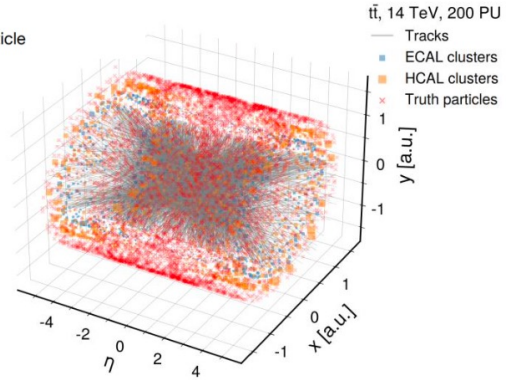
- Targets (per node):

$$y_j = [\text{PID}, p_T, E, \eta, \phi, q]$$
$$\text{PID} \in \{\text{charged hadron}, \text{neutral hadron}, \gamma, e^{\pm}, \mu^{\pm}\}$$

- Nodes:

$$x_i = [\text{type}, p_T, E_{ECAL}, E_{HCAL}, \eta, \phi, \eta_{outer}, \phi_{outer}, q]$$
$$\text{type} \in \{\text{track}, \text{cluster}\}$$

Graph structure computed dynamically, not trained explicitly; ~5000 elements per graph



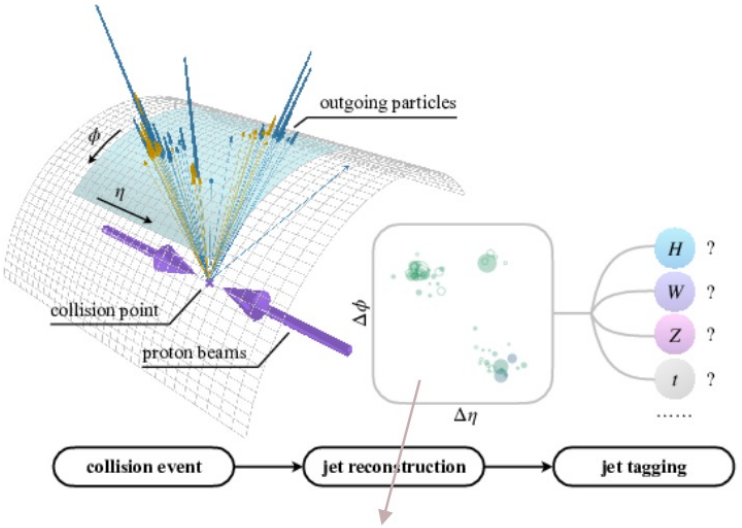
[MLPF: Efficient machine-learned particle-flow reconstruction using graph neural networks](), Pata et al

Jet Tagging

Particle Cloud GNNs
Apply GNNs w/ dynamic graph construction (EdgeConv) to make predictions on point clouds

e.g.

ParticleNet: Jet Tagging via Particle Clouds, Qu and Gouskos

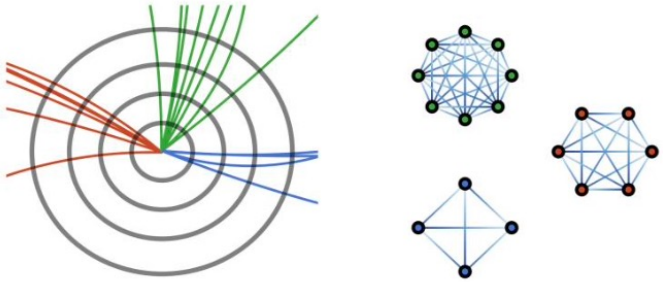


Particles can be viewed as a "particle cloud" of kinematic features at different spatial locations

Particle Graph GNNs
Apply a GNN to a pre-constructed graph with particles as nodes

e.g.

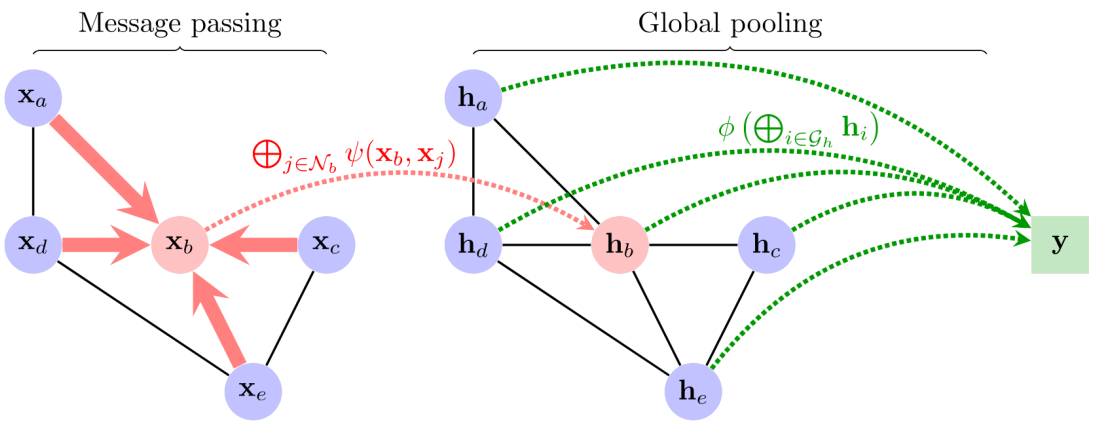
ABCNet: An attention-based method for particle tagging, Mikuni and Canelli
JEDI-net: a jet identification algorithm based on interaction networks, Moreno et al

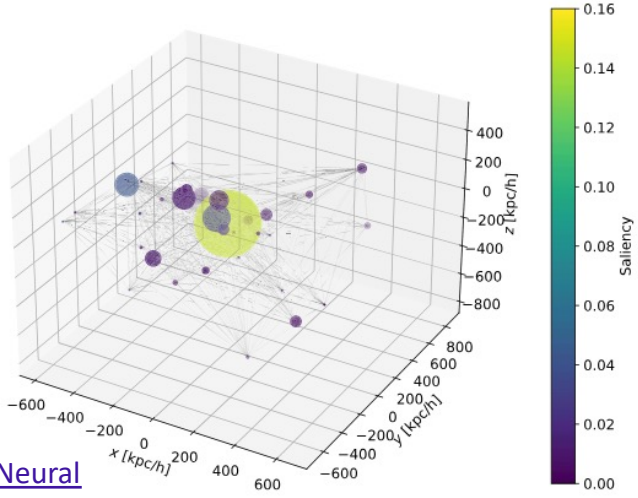


In this case, the particles comprising three jets have been embedded as nodes in fully connected graphs

Astronomy

- Builds graphs of galaxies (nodes) within a halo using radius-based edge construction
  - Motivated by gravity getting weaker as distance increases
- Use message passing GNN then graph pooling to predict halo mass
  - Galaxy positions stored in CoM rest frame → translation invariant, augments training with rotations of halos → rotation invariant
- Compare two different physical simulators to understand how this impacts GNN performance, robustness, and explainability
- Outperforms traditional approach based on stellar masses



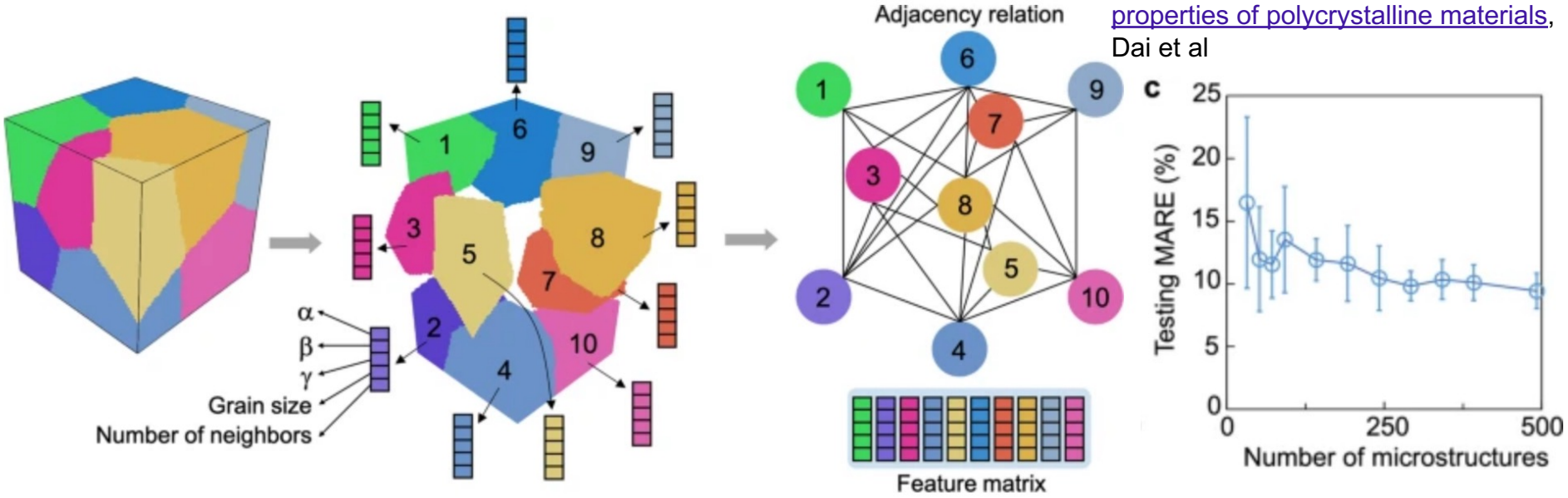


Inferring Halo Masses with Graph Neural Networks, Villanueva-Domingo et al

Materials Science

- Properties of polycrystalline materials depend on atomic lattice structure within grains and microstructures between grains
- Microstructure graph represents grains as nodes, physical contact between grains as edges
  - Node features: Euler angles, grain size, number of neighboring grains
- Use a GCN to predict magnetostriction of different alloys
  - Achieves low (~10% prediction error) across data sets
  - Can be used to prioritize synthesis and experiments

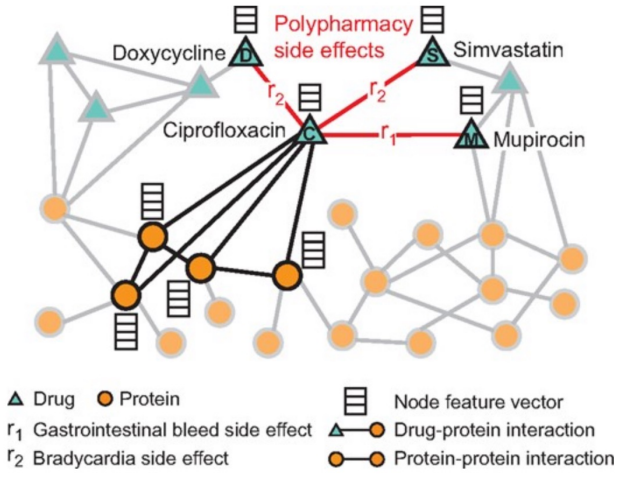
Graph neural networks for an accurate and interpretable prediction of the properties of polycrystalline materials, Dai et al

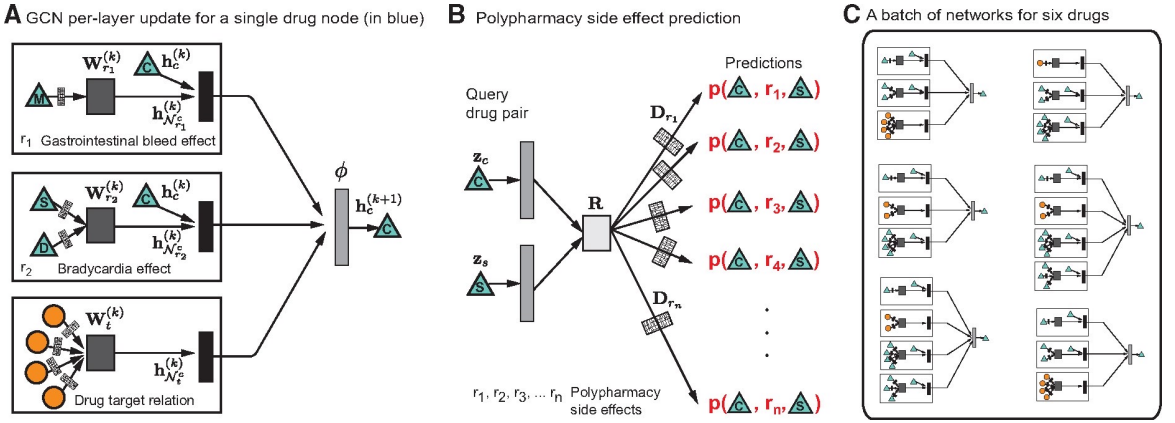


Bioinformatics

- Uses multimodal graph composed of proteins and drugs
  - Edges can be protein-protein interactions, drug-protein interactions (drug targets), or drug-drug interactions (side effects)
- GCN autoencoder to predict side effects of drug pairs
  - Multirelational link prediction on multimodal graph
- Encoder GCN updates node representations, shared decoder uses drug pairs to calculate edge probabilities
  - Different decoders based on node types

$$J_r(i,j) = -\log\, p_r^{ij} - \mathbb{E}_{n\sim P_r(j)}\, \log\,\left(1 - p_r^{in}\right), \quad J = \Sigma_{(v_i,r,v_j)\in\mathcal{R}}\, J_r(i,j).$$

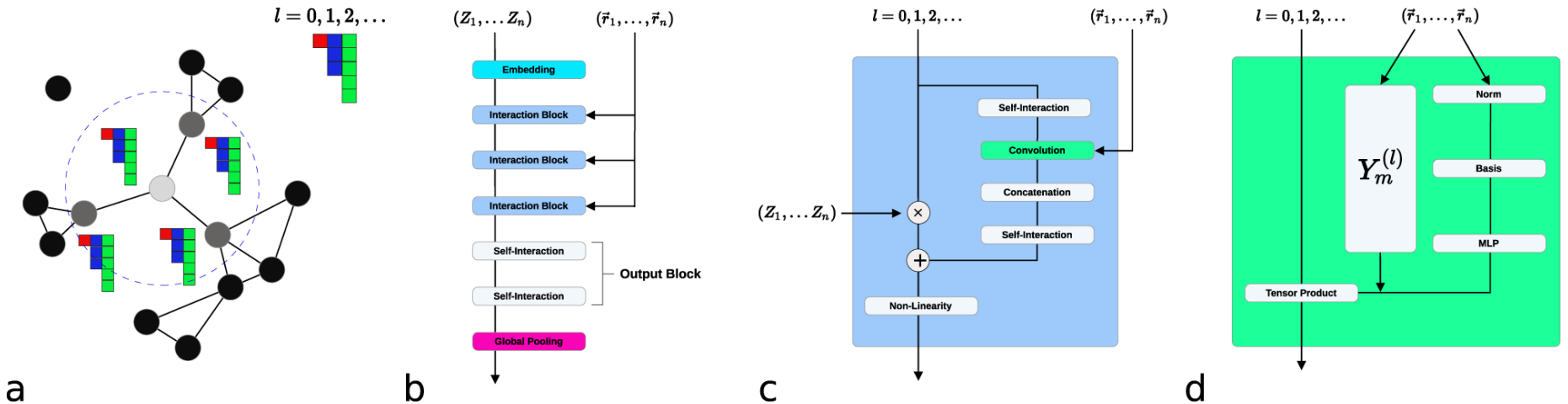




Modeling polypharmacy side effects with graph convolutional networks, Zitnik et al

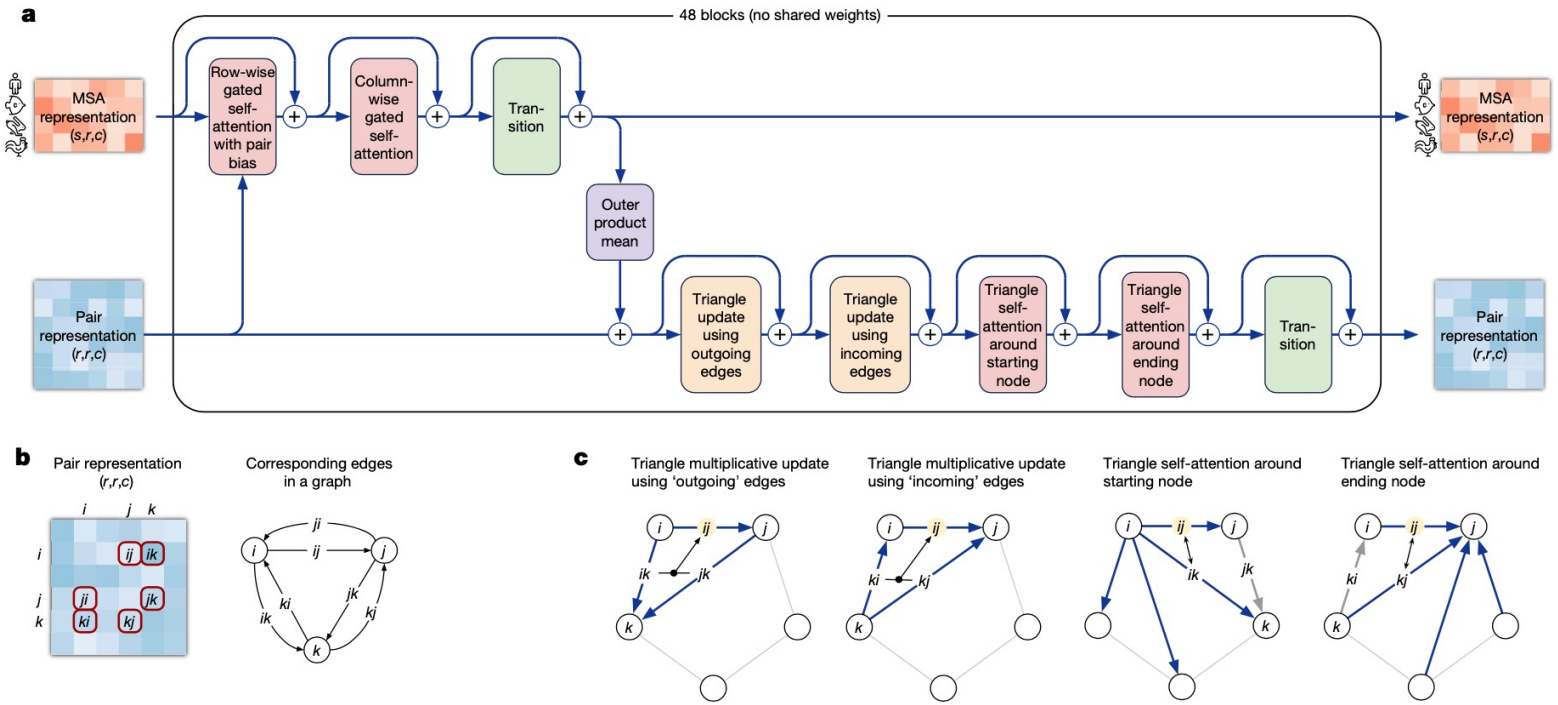
Molecular Physics

- Uses E(3)-equivariant GNN to calculate interatomic potentials
  - Computationally intractable task for complex molecules
  - Can then use negative gradient of the potential to calculate forces on each atom as function of their positions
- Uses radius based graphs with individual atoms as nodes
  - Nodes contain scalar, vector, and tensor features as direct sum of irreps of the symmetry group
- GNN uses convolutions parameterized by spherical harmonics and learnable radial functions
  - Also includes self-interaction update term in each interaction block



Protein Folding

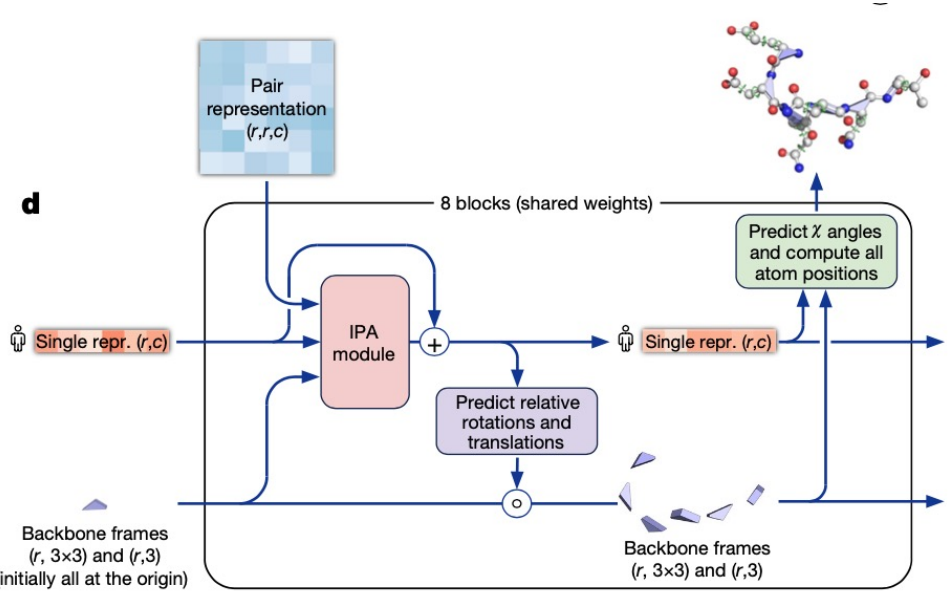
- Predicts final 3D coordinates of all heavy atoms using amino acid sequences and residue proximity graphs as inputs
- First stage (Evoformer), produces new embeddings of input sequences and residue pairs
  - Updates sequence representation using self attention and residues
  - Updates residue graph using sequence updates, self attention, and triangle inequality biasing terms



Highly accurate protein structure prediction with AlphaFold, Jumper et al

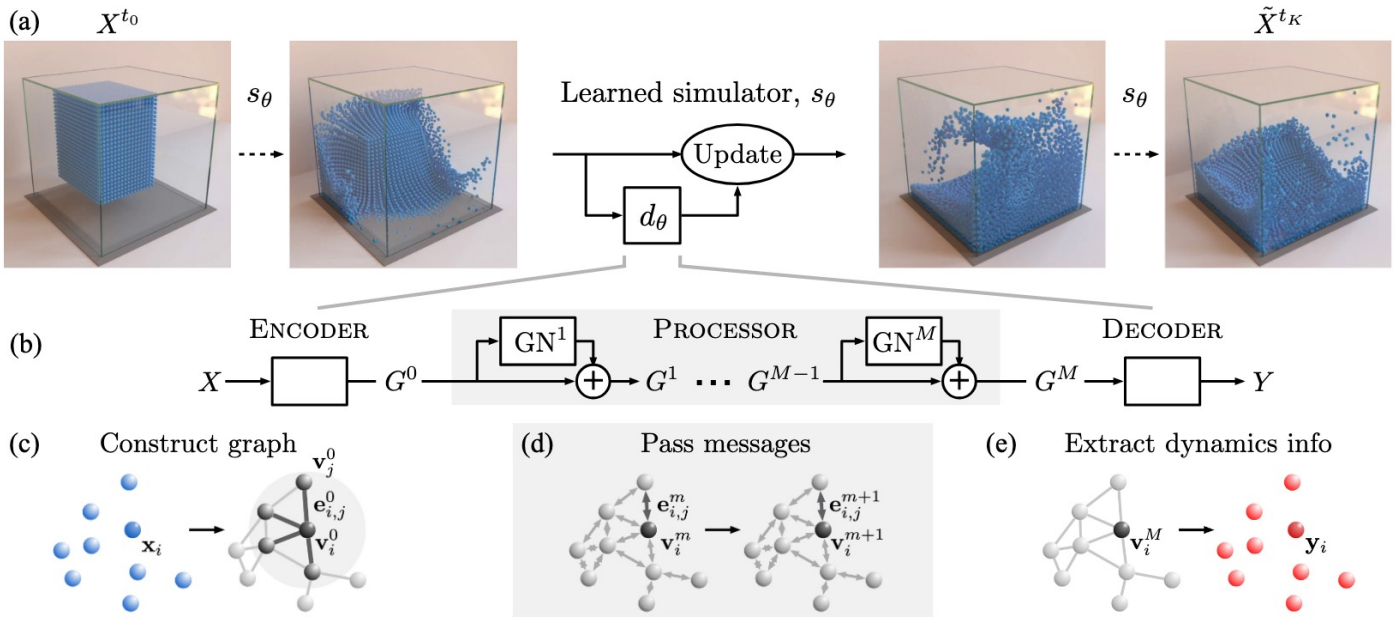
Protein Folding

- Second stage predicts translation and rotation of all residues
  - Includes an equivariant attention module that makes final protein structure invariant to rotation and translation
  - Attention values used to update individual residue positions
- Several types of inductive biases:
  - Evolutionary (homologues)
  - Relational (residue graph structure)
  - Geometric (equivariant attention in position update)
  - Physical (triangle inequality logit)
  - Scientific knowledge (information sharing between residue graphs and sequences)



Physical Simulations

- Learned physics simulator turns particle-based representation of physical system into a graph at each simulation time step
- Message passing GNN re-embeds graph that is decoded to predict next time step iteration of the system
- Draw parallels between pieces of the architecture and physics
  - Messages ~ calculations of physical quantities, learned decoder ~dynamics
- Successfully models multiple systems and generalizes well



Learning to Simulate Complex Physics with Graph Networks, Alvaro-Sanchez et al

Conclusions

- Graphs are a natural data representation for many kinds of relational, geometrically structured, or varied length data

- Graph-based learning methods can leverage geometric or relational information for a range of tasks
  - Node and graph, classification and regression, link prediction
  - Many different GNN approaches and architectures can work well for a problem, important to explore different options and think about benchmarking

- GNNs are not without limitations
  - Susceptible to noise and oversmoothing, not always the best solution for a given learning problem

- Many exciting open areas of research + applications
  - Equivariant models, new modes of graph construction & message passing, etc
  - Many, many fields of science

Reading List

- A Gentle Introduction to Graph Neural Networks, Distill (blog post)
- Machine Learning with Graphs, Weights and Biases (tutorials and blog posts)
- Everything is Connected: Graph Neural Networks, Petar Velickovic (paper)
- Representation Learning on Graphs: Methods and Applications, William L. Hamilton, Rex Ying, Jure Leskovec (paper)
- Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, Michael M. Bronstein, Joan Bruna, Taco Cohen, Petar Veličković (book)
- Learning Mechanics with Machines, Steven Sun (course)
- Physics-based Deep Learning, Phillip Holl et al (book)
- ML and the Physical Sciences (workshop series)
- Physics Inspired Graph Neural Networks, Michael Bronstein (lecture)
- Graph neural networks for materials science and chemistry, Reiser et al (paper)
- Graph Neural Networks in Particle Physics: Implementations, Innovations, and Challenges, Savannah Thais et al (paper)
- A Compact Review of molecular property prediction with GNNs, Wieder et al (paper)

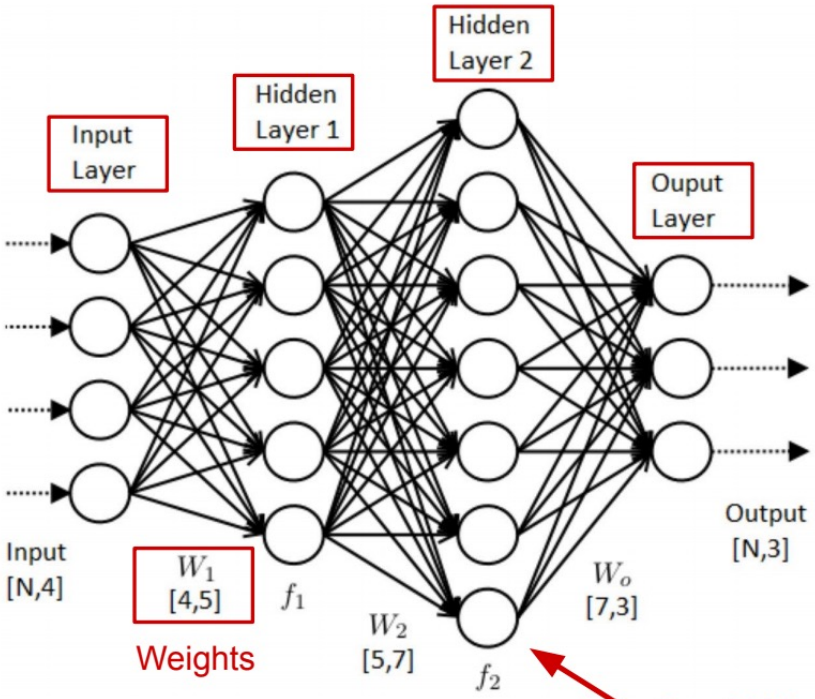
Thank you!

st3565@columbia.edu @basicsciencesav

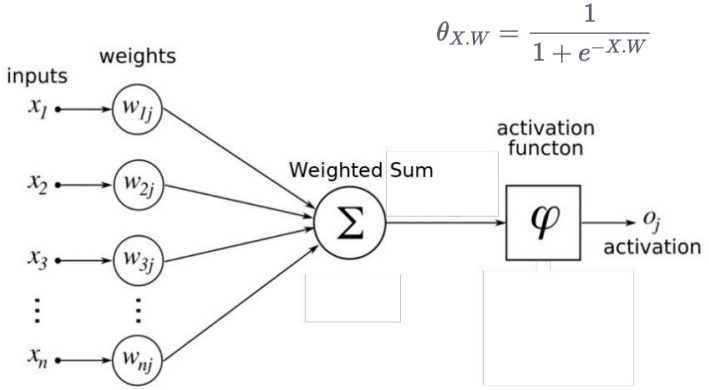
Backup

Neural Networks

- Loosely based on biological neurons
- Linear transformations between layers represented as matrices of weights W and biases b
- Each node calculates activation function of weighted sum of inputs to determine if information is passed on
- Extremely powerful dimensional transformation
  - Can efficiently train very deep networks to 'approximate any function'



$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$

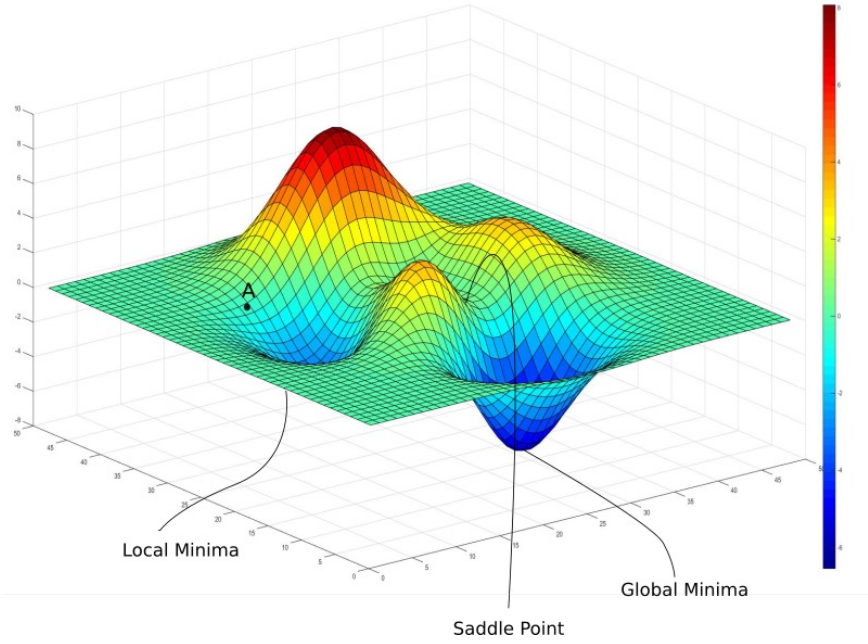


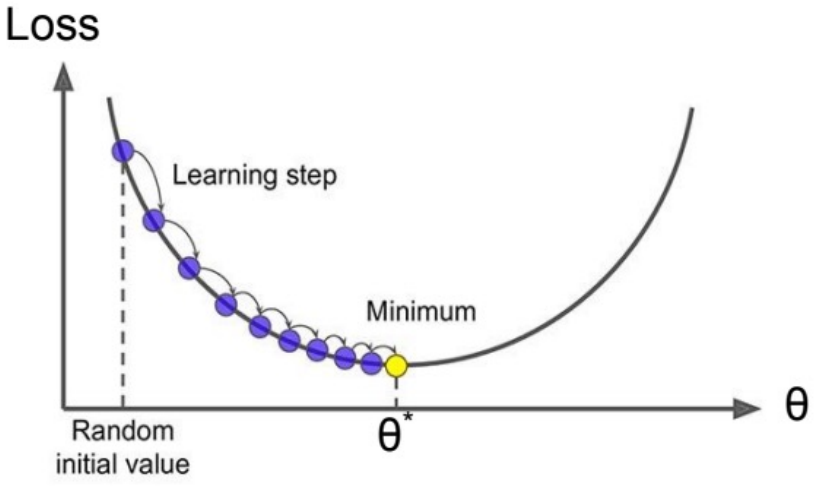
Gradient Descent

Intuition: Want to efficiently find the model parameters θ that minimize the loss function L

- We normally minimize things by evaluating the derivatives
  - This is the basis of gradient descent → your loss function must be differentiable!
  - But this is very difficult for complicated models with many parameters
- Basic procedure: at each training iteration (epoch) update model parameters to move evaluation point in opposite direction of gradient of loss function in direction of steepest ascent

$$W_{k+1} \leftarrow W_k - \alpha \nabla L(W_k)$$





Another Way To Do Equivariance

- Build filters (graph convolutions) that are equivariant under the desired symmetry
- Example: SE(3) equivariance using spherical harmonics to parameterize the learned filters and inputs/outputs
  - And CG coefficients to combine tensors

Learned Parameters

Convolutional kernels...

with no symmetry:

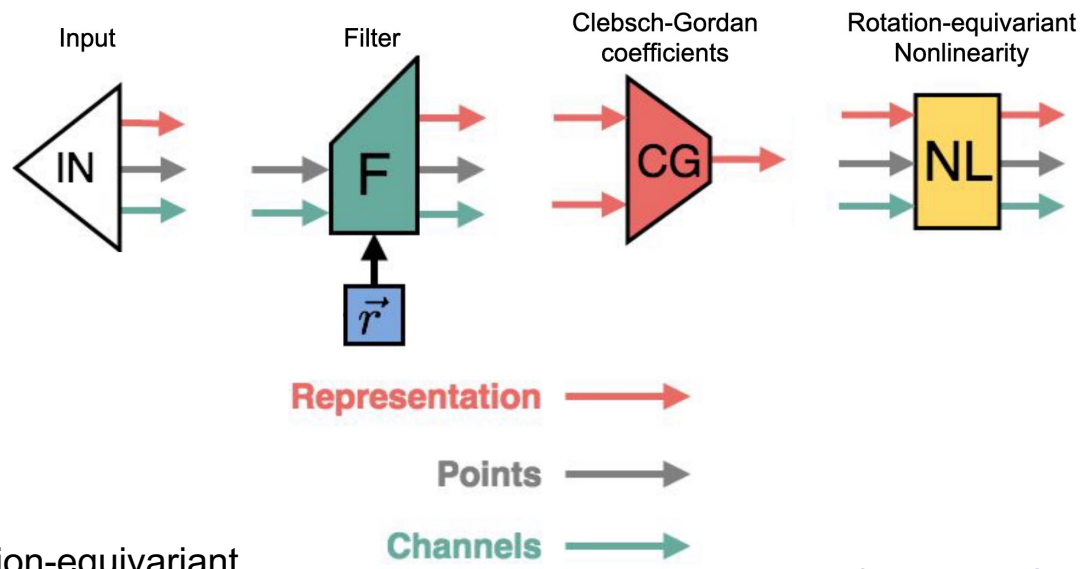
$$W(\vec{r})$$

with 2D circular harmonics:

$$R(r)e^{im\phi}$$

with 3D spherical harmonics:

$$R(r)Y_l^m(\hat{r})$$

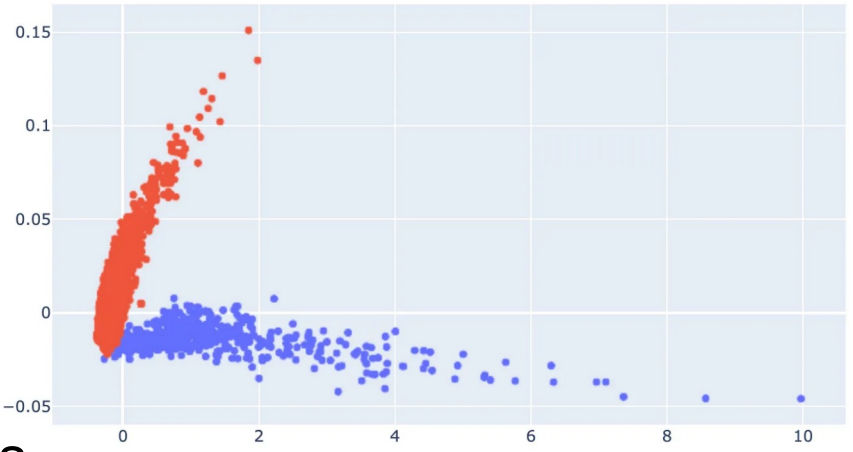


Tensor Field Networks: Rotation- and translation-equivariant neural networks for 3D point clouds, Thomas et al

Images from Tess Smidt

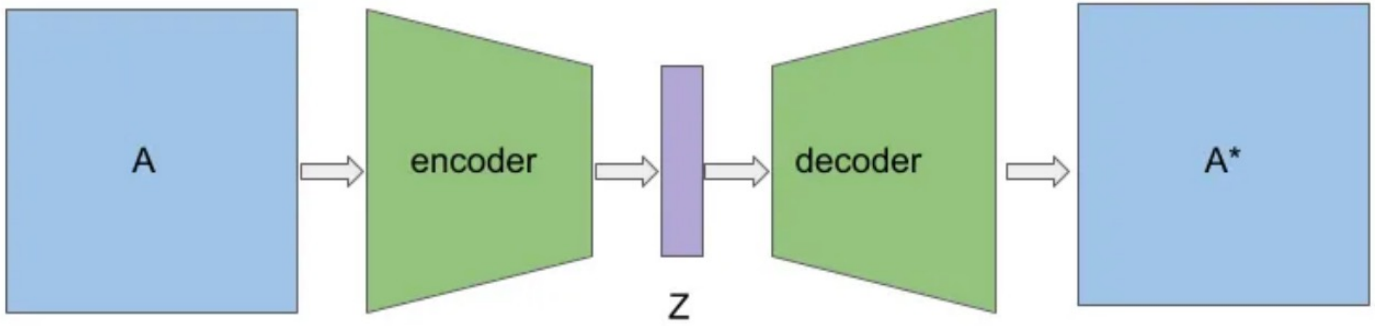
Graph Autoencoders

- Autoencoders aim to transform input data (encoder)to a lower dimensional space (Z) in way where it can be accurately reproduced (decoder)
  - For graphs, encoder is typically a node update GNN, decoder can be the dot product of new node representations
- Can use the encoding space in various ways
  - Learn about the nodes by examining distribution in embedding space
  - Predict new relations between nodes through decoding
  - Generate new graphs by sampling from latent space



PCA projection of the embeddings when GAE model is fully trained. Blue colors are disease nodes and red colors are gene nodes.

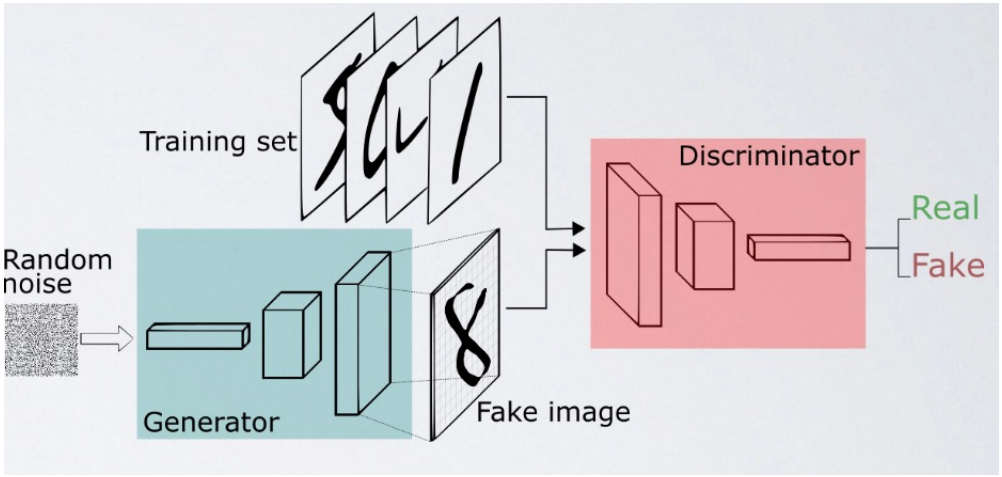
METHOD	AUC	AP
DEEPWALK	78.5± 2.25	77.4± 2.51
NODE2VEC	86.0± 1.98	84.6± 2.36
C-VGAE	**90.8**± 1.72	**91.3**± 1.69

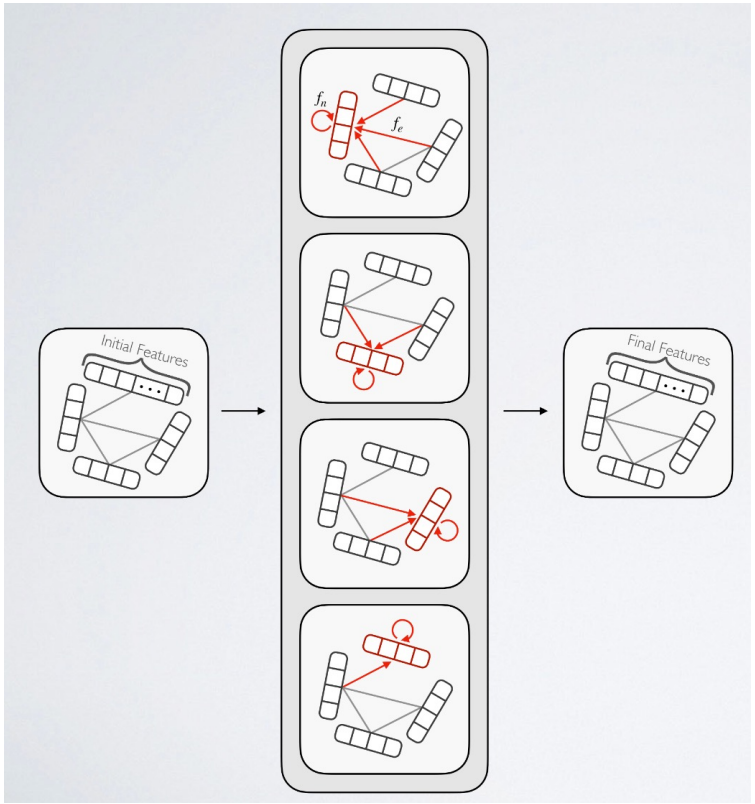


paper

Graph GANs

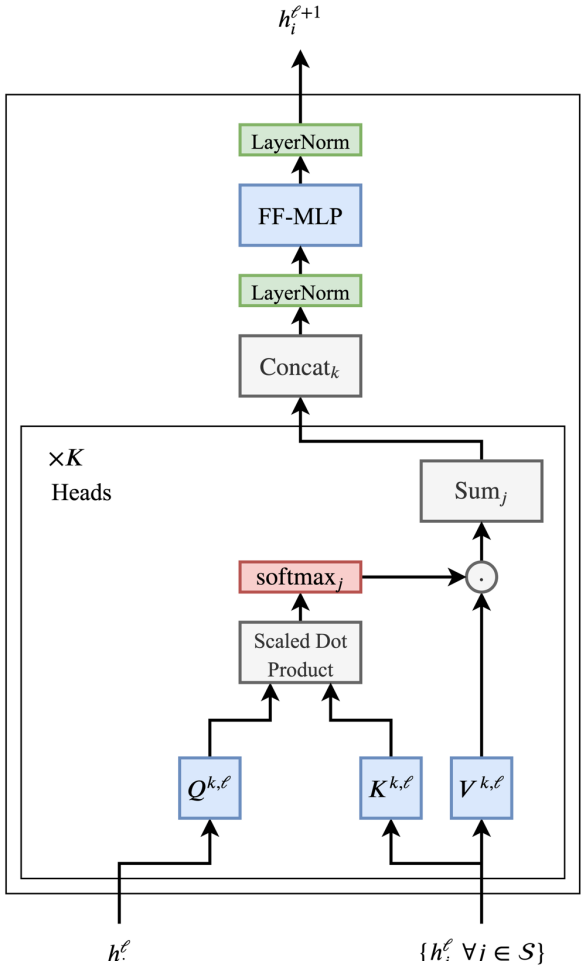
- Graph based Generative Adversarial Network
  - Generator uses message passing on noise graph to recreate input features
  - Discriminator uses message passing to classify graph as real or generated
- Networks are trained with a shared loss function
  - Learns to balance tasks and create high-fidelity synthetic data!





paper

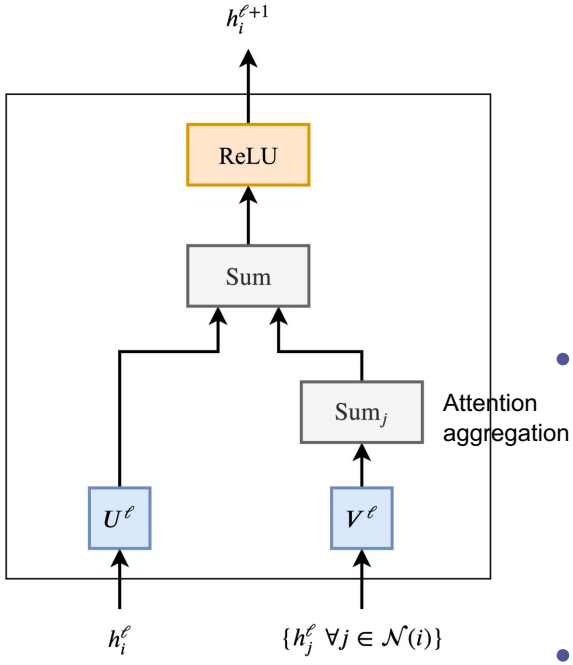
Transformers as GNNs



$$h_i^{\ell+1} = \text{Attention}\left(Q^\ell h_i^\ell, K^\ell h_j^\ell, V^\ell h_j^\ell\right)$$

$$h_i^{\ell+1} = \sum_{j \in \mathcal{S}} w_{ij}\left(V^\ell h_j^\ell\right)$$

$$\text{where } w_{ij} = \text{softmax}_j\left(Q^\ell h_i^\ell \cdot K^\ell h_j^\ell\right)$$

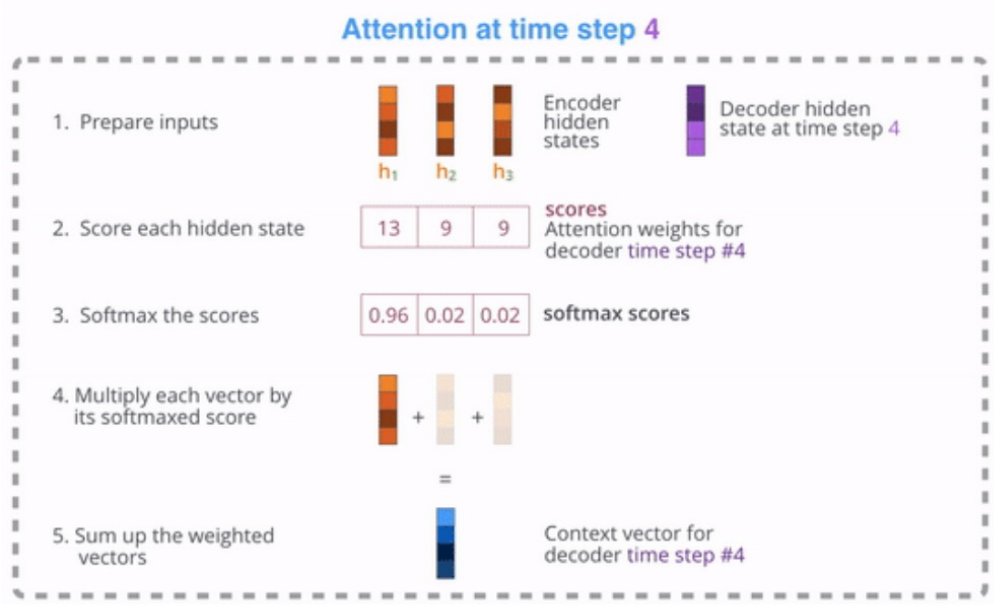


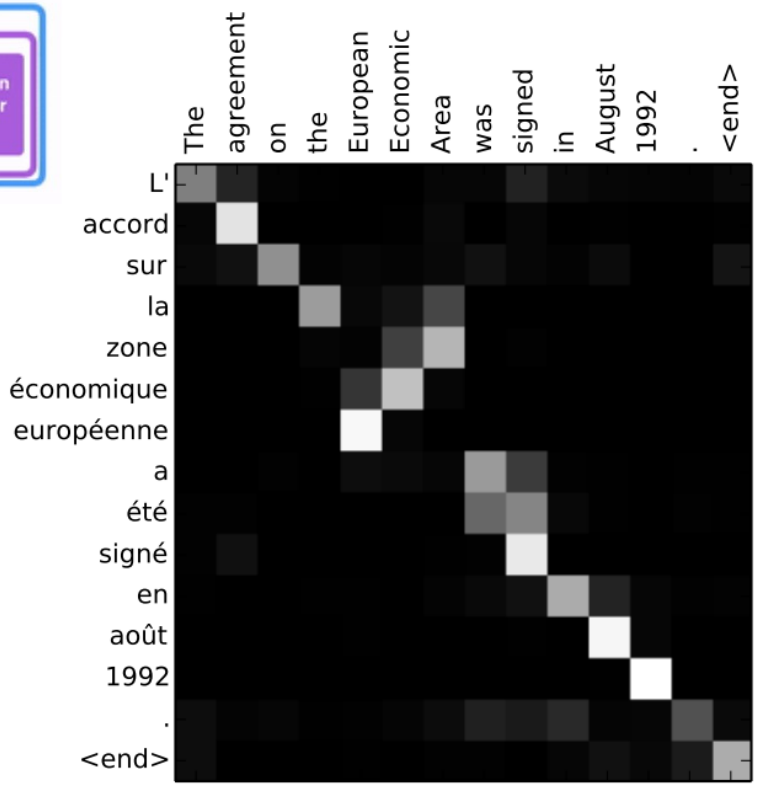
- Transformers developed for sequence to sequence learning
  - Relies on the idea of attention: a way for learning what part of a sequence is most important to the prediction
- Using learned aggregation (attention) and a fully connected graph, GNN becomes a transformer
- Problem specific approaches/training tricks usually explain the difference between GNN and transformer performance

Images from The Gradient

Transformers: Attention

- Transformers developed for sequence to sequence learning
  - Consist of an encoder and decoder with attention
- Attention allows models to learn what part of input to ‘focus’ on
- Example: translation RNNs with attention

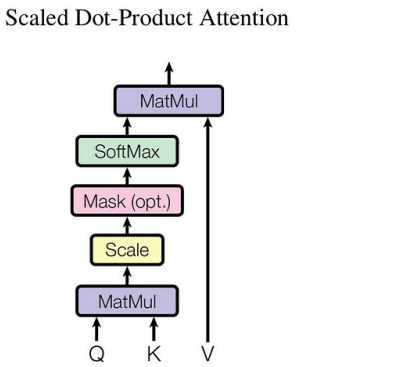


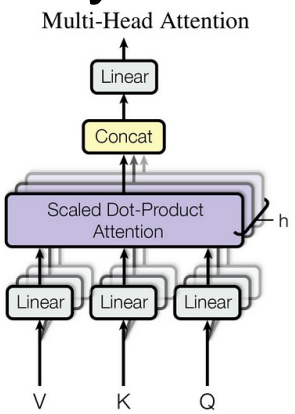


Images from Jay Alammar

Transformers: Is All You Need

- Transformers remove recurrent structure and add multiheaded-attention
  - Add a positional encoding to account for ordering
- Output embedding is trained to predict sequence shifted by one word
  - Q=vector representation of one word, K=vector rep of all words, V=(different) vector rep of all words
- Inference is run iteratively





$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

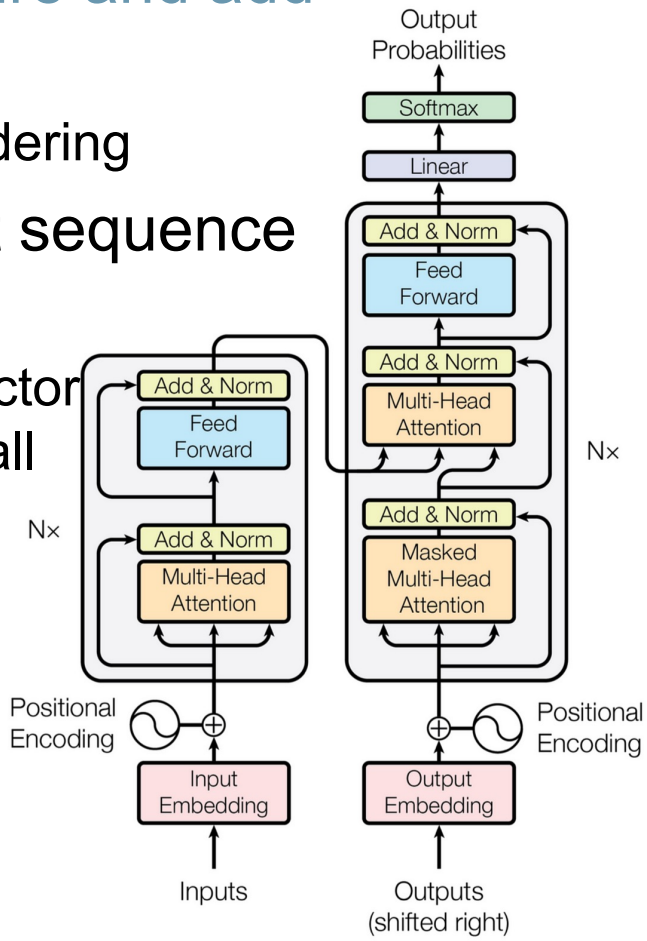


Figure 1: The Transformer - model architecture.

Attention Is All You Need, Vaswani et al