INTRODUCTION TO GRAPH NEURAL NETWORKS

Savannah Thais
Columbia University
03/05/2023
APS March Meeting 2023 Tutorial

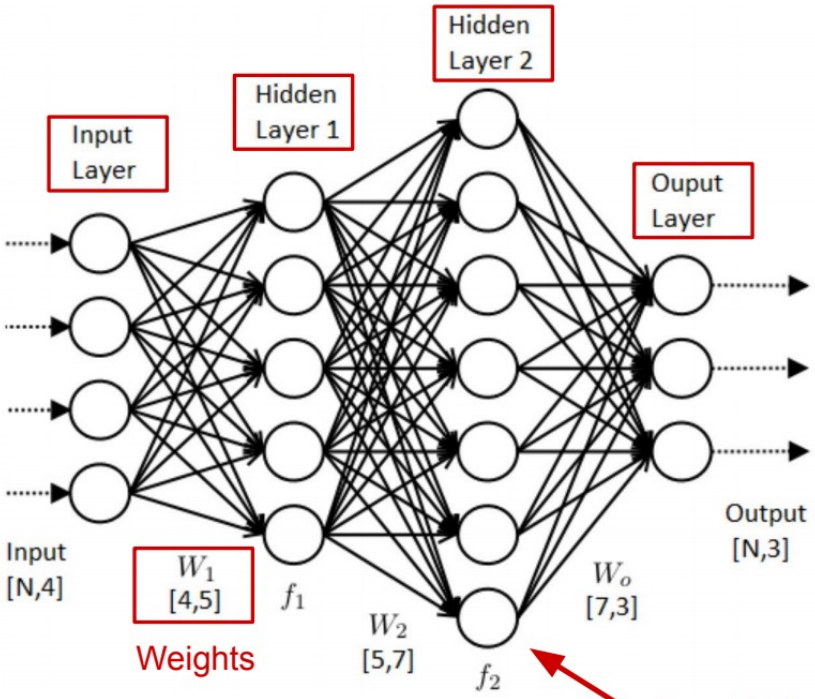
Outline

1. What are graphs and graph neural networks?
2. What are some physics tasks we can use GNNs for?
3. Some fun extensions of GNNs!
4. A hands on introductory exercise in Pytorch

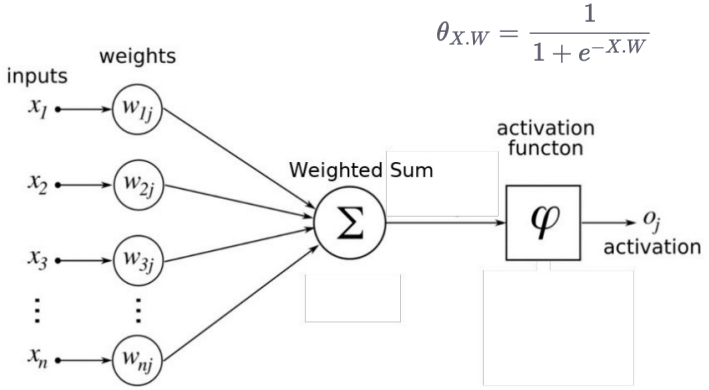
What are graphs and graph neural networks?

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'



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

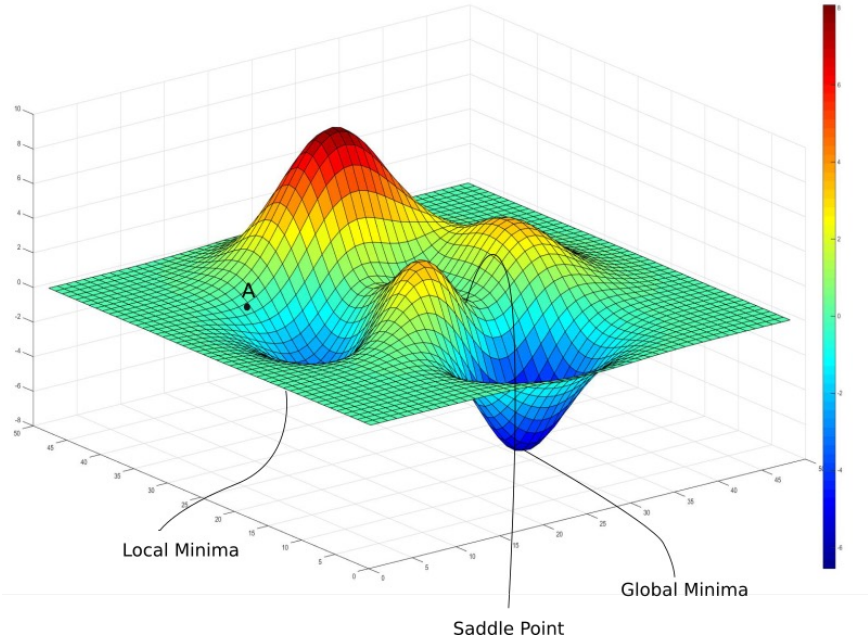


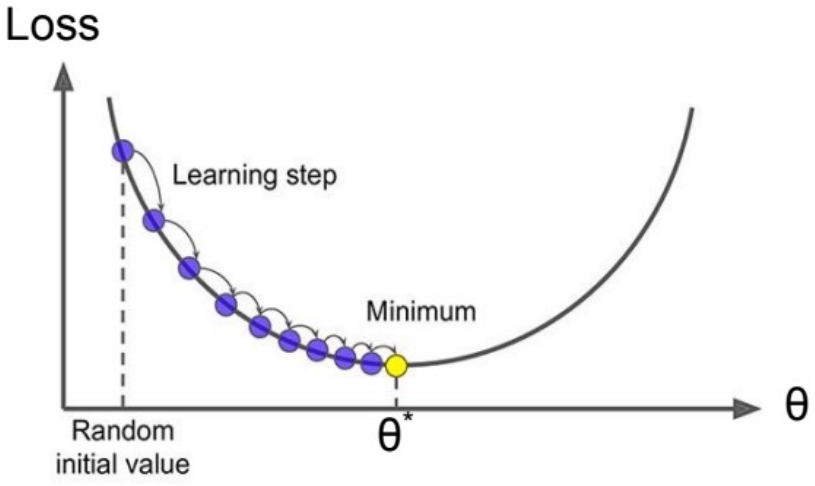
Gradient Descent

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

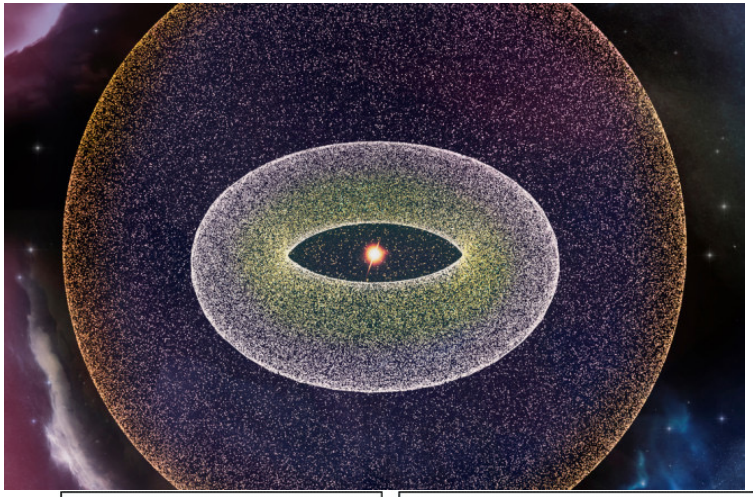
$$W_{k+1} \leftarrow W_k - \alpha \nabla L(W_k)$$

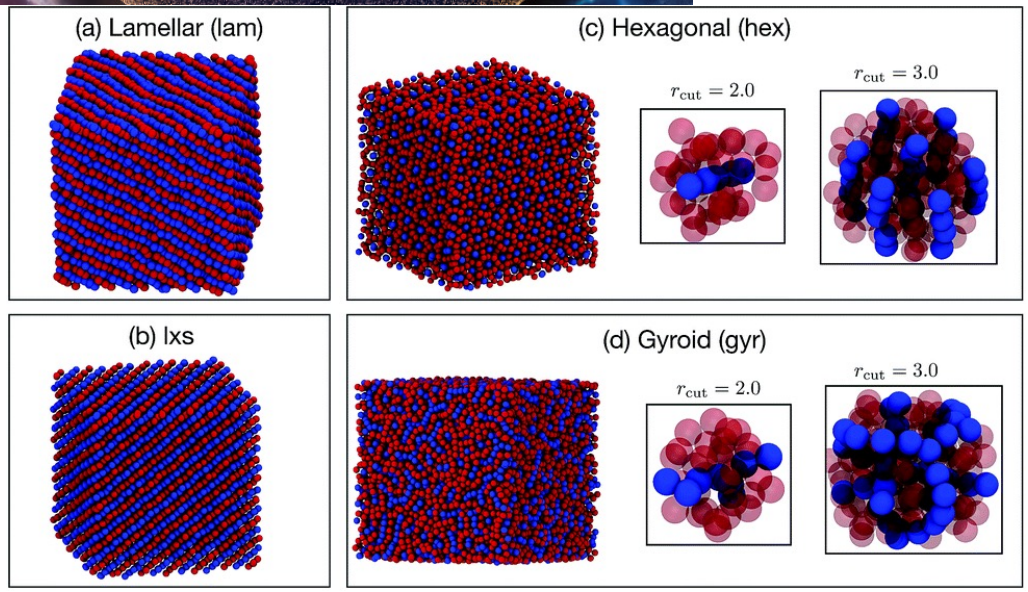


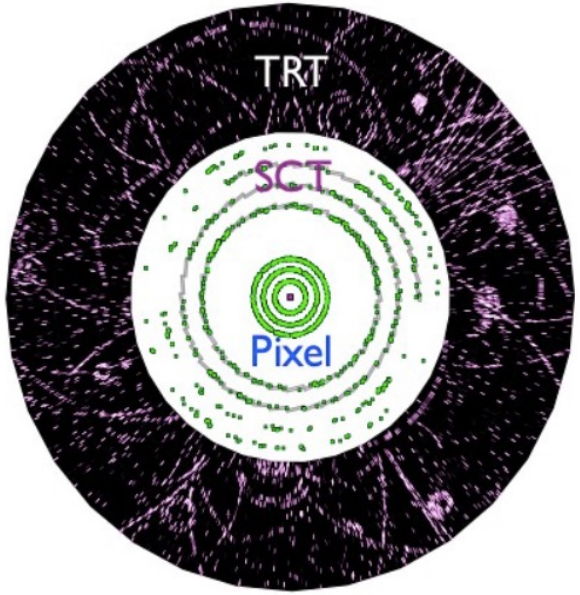


Point Clouds

Many types of data can be represented as a point cloud

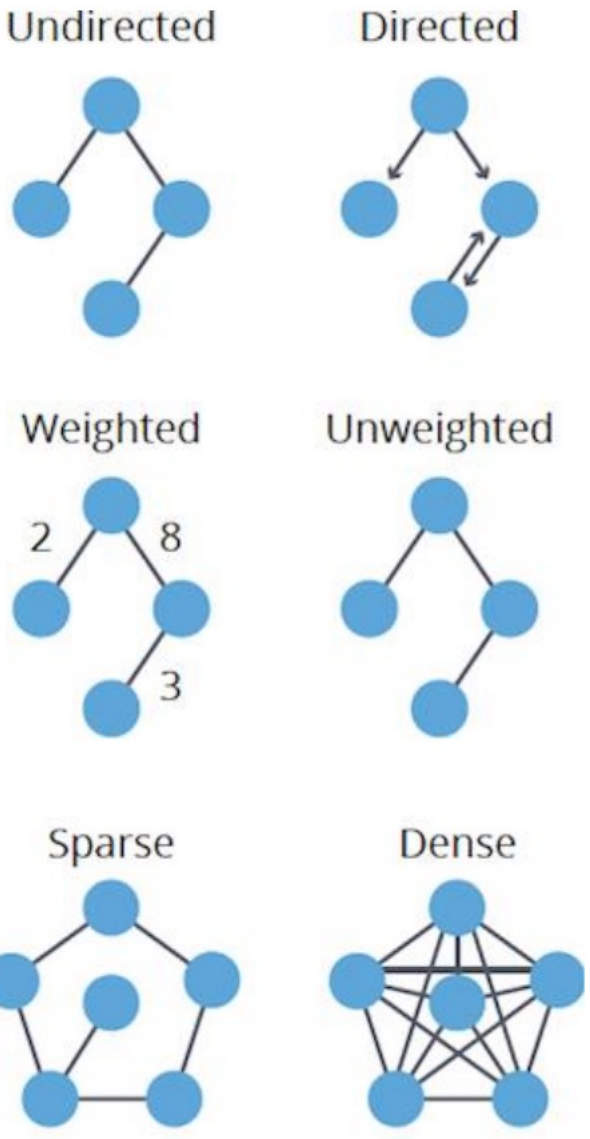






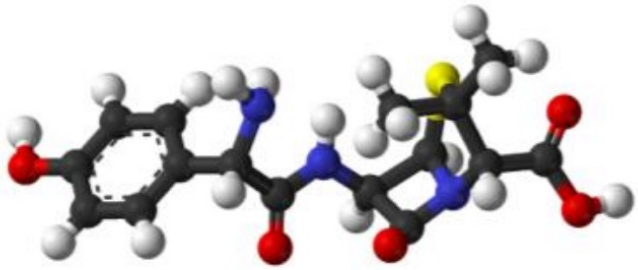
Graphs

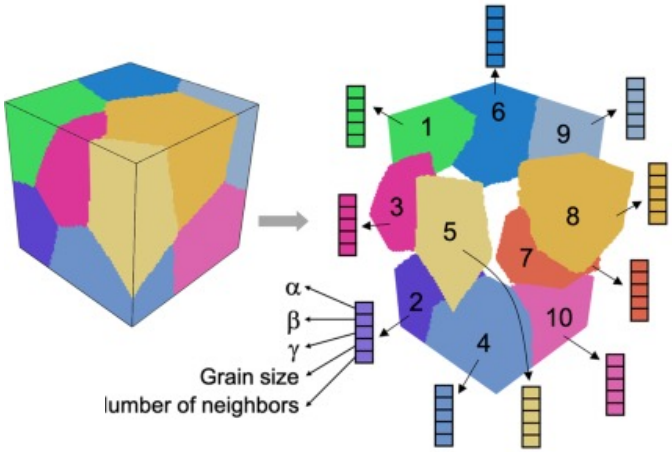
- If we add relational information to a point cloud we get a graph
  - Nodes: vertices $u \in V$ with associated information $\mathbf{x}_u \in \mathbb{R}^{d_V}$
    - spatial coordinates, features, etc
  - Edges: connections between nodes $(u, v) \in E$
    - Can be directed or undirected, can have associated 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

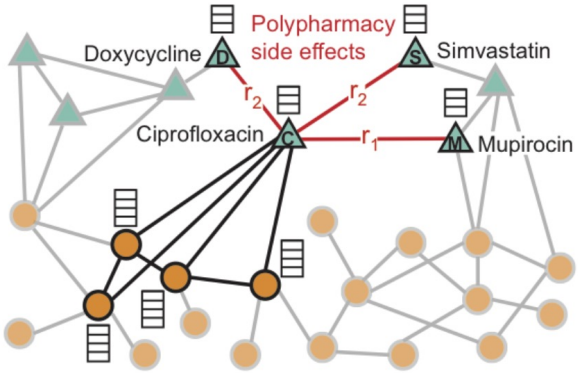


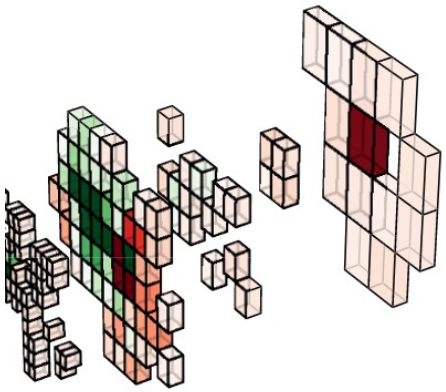
Graphs

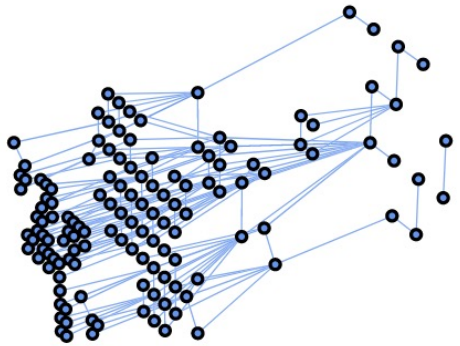
Graphs are an intuitive representation for all kinds of geometric, structured, variable length physics data











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: 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:

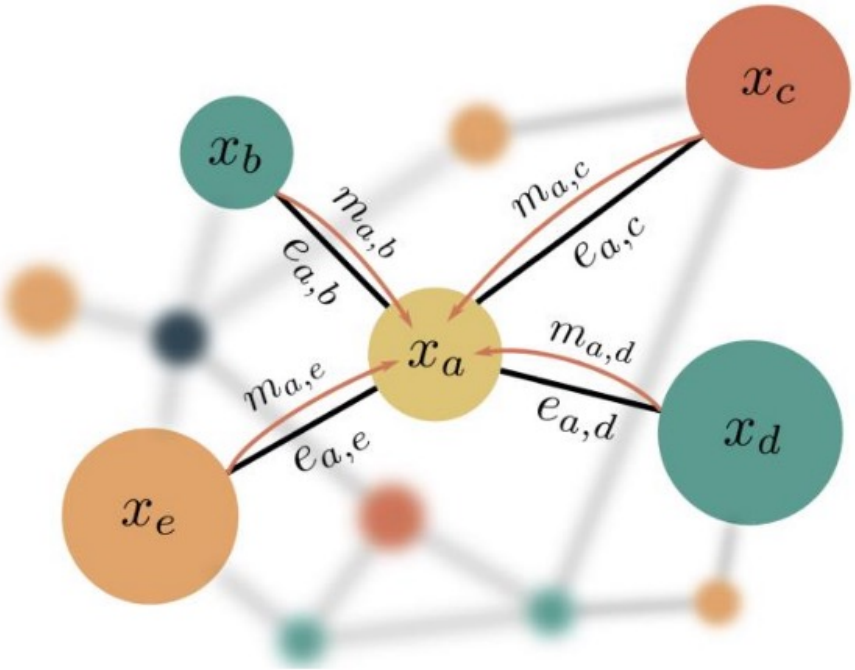
$$\boldsymbol{a}_u^{(k)} = \bigoplus_{v \in N(u)} \boldsymbol{m}_{uv}^{(k)}$$

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:

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



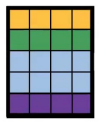
Slide from Gage DeZoort

Graph Neural Networks

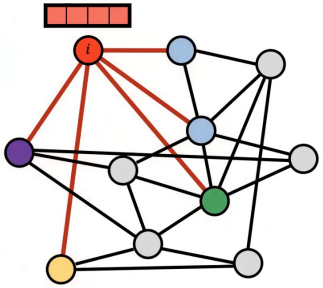
The goal of a (or at least some) GNN(s) is to learn a smart re-embedding of the graph structure that preserves the relational structure but makes it easier to solve some downstream task

The most general version:

multiset of neighbour features



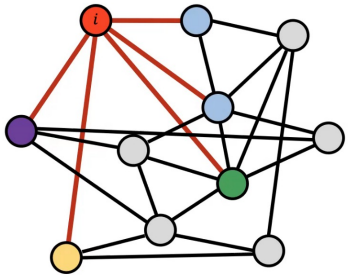
$$\mathbf{X}_{\mathcal{N}_i} = \{\!\{\mathbf{x}_{j \in \mathcal{N}_i}\}\!\}$$



local function

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

permutation invariant



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

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

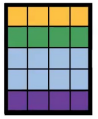
permutation-invariant aggregation operator, e.g. sum

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

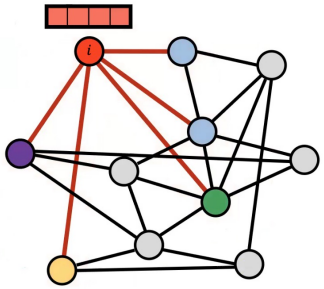
learnable functions

Graph Neural Networks

multiset of neighbour features



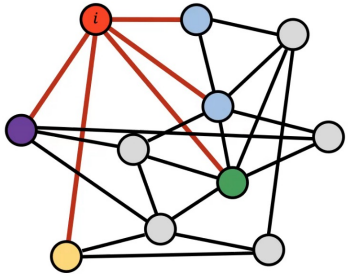
$$\mathbf{X}_{\mathcal{N}_i} = \{\mathbf{x}_{j \in \mathcal{N}_i}\}$$



local function

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

permutation invariant



Several variations on input to learnable functions:

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

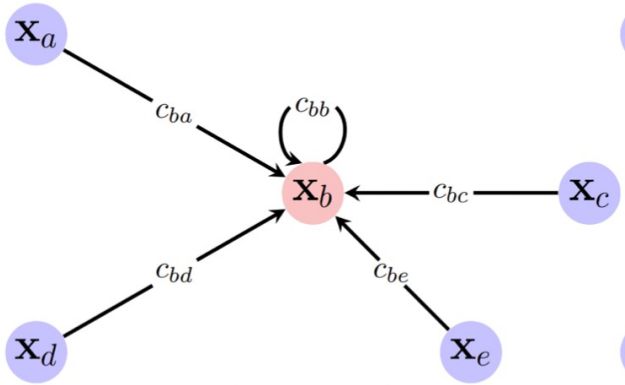
"convolutional"

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

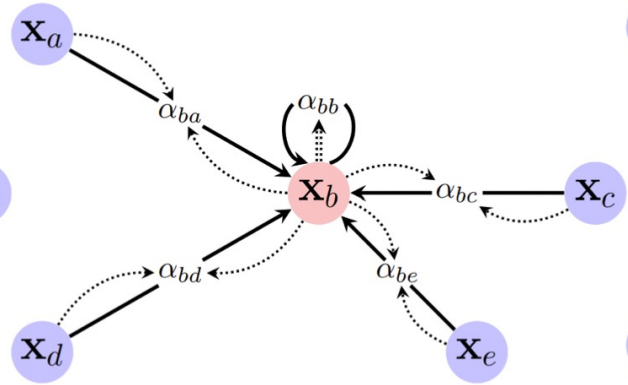
"attentional"

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

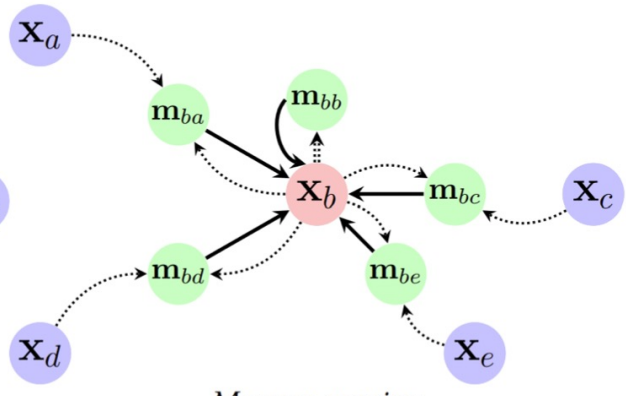
"message passing"



Convolutional



Attentional

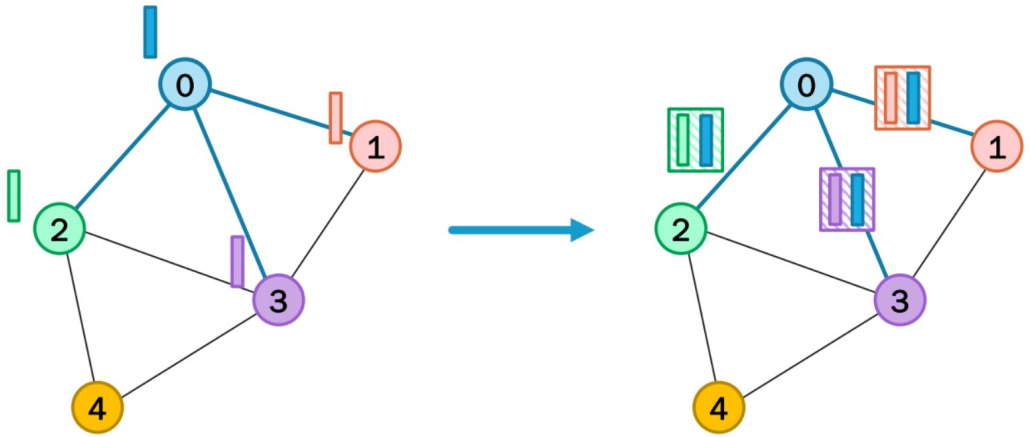


Message-passing

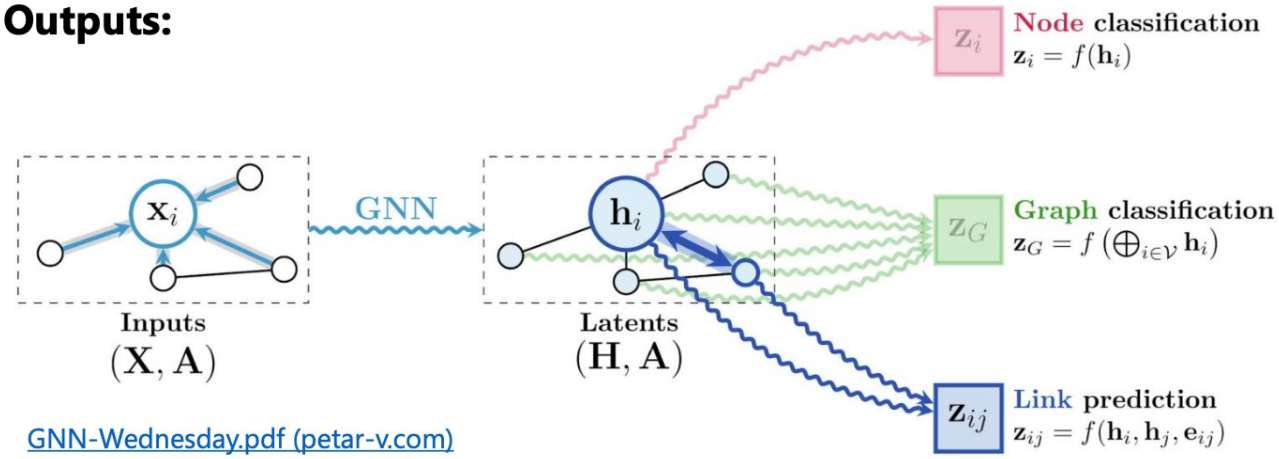
Graph Neural Networks

We can also update the graph edges

- Isotropic message passing can't differentiate importance of neighbors
- *An*isotropic message passing: encode a combination of node and neighbor along each edge



Outputs:

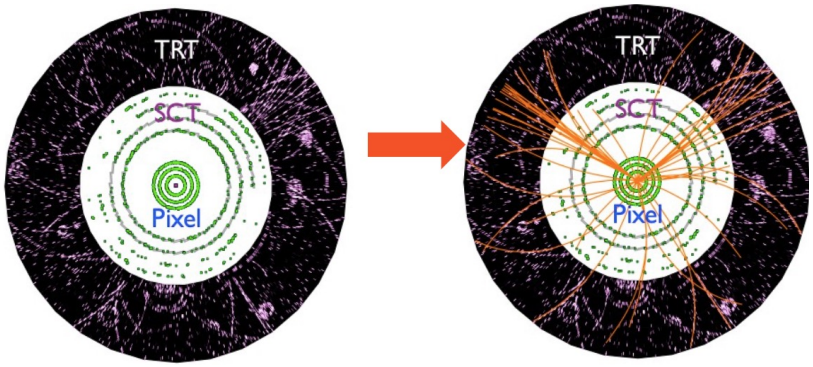


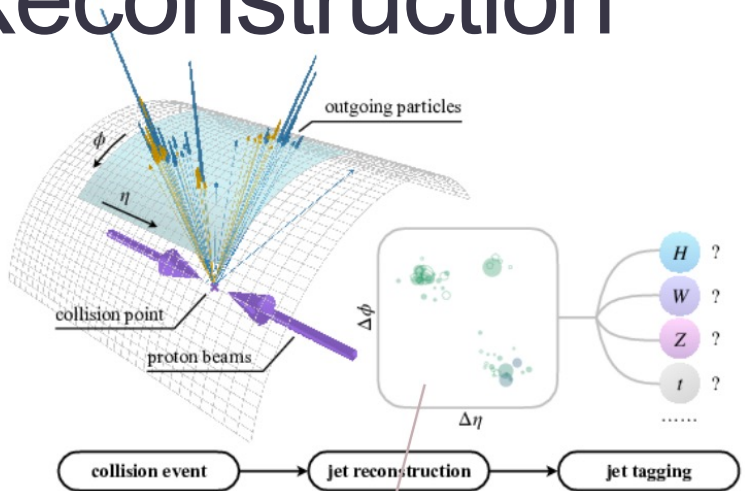
GNN-Wednesday.pdf (petar-v.com)

And we can use the updated graph in many ways

What are some physics tasks for GNNs?

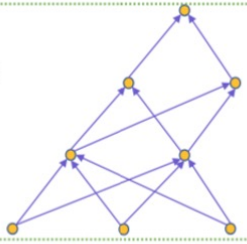
Charged Particle Reconstruction





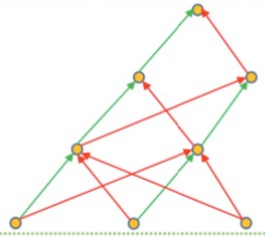
Graph Construction:

Input event in graph representation



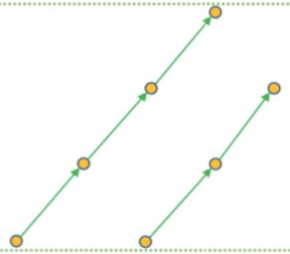
GNN Classifies Edges:

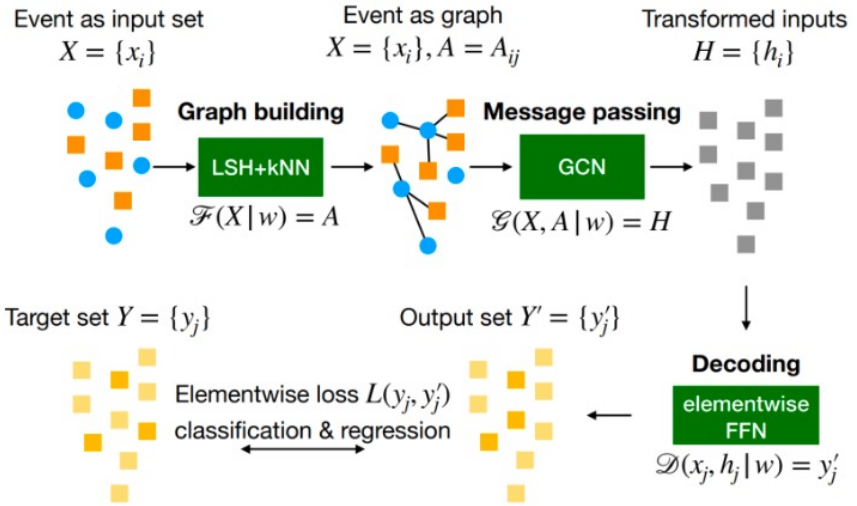
Green = true track segment
Red = false hit connection



Track Finding

Connecting-the-dots algorithm extracts tracks





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

$y_j = [\text{PID}, p_T, E, \eta, \phi, q, \ldots]$, PID $\in$ {none, charged hadron, neutral hadron, $\gamma, e^{\pm}, \mu^{\pm}$}

$h_i \in \mathbb{R}^{256}$

Trainable neural networks: $\mathcal{F}, \mathcal{G}, \mathcal{D}$

● - track, ■ - calorimeter cluster, ■ - encoded element

■ - target (predicted) particle, ■ - no target (predicted) particle

Predicting Astronomical Motion

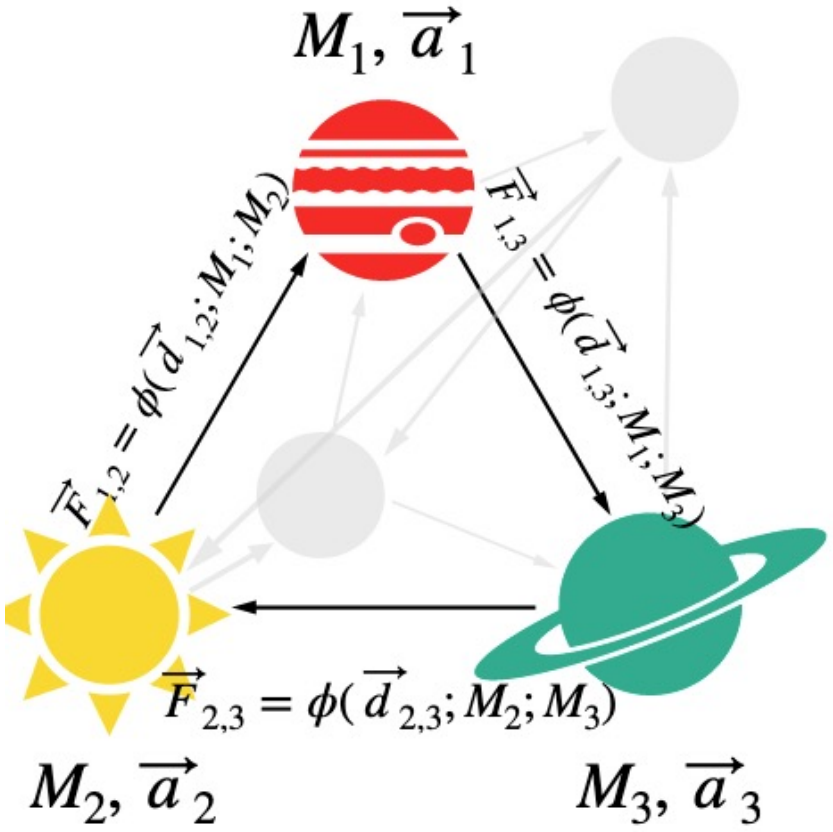
1. Our inputs are the positions of the bodies

2. They are converted into pairwise distances

3. **Our model tries to guess a mass for each body**

4. **It then also guesses a force, that is a function of distance and masses**

5. **Using Newton's laws of motion ($\sum \vec{F} = M\vec{a}$) it converts the forces into accelerations**



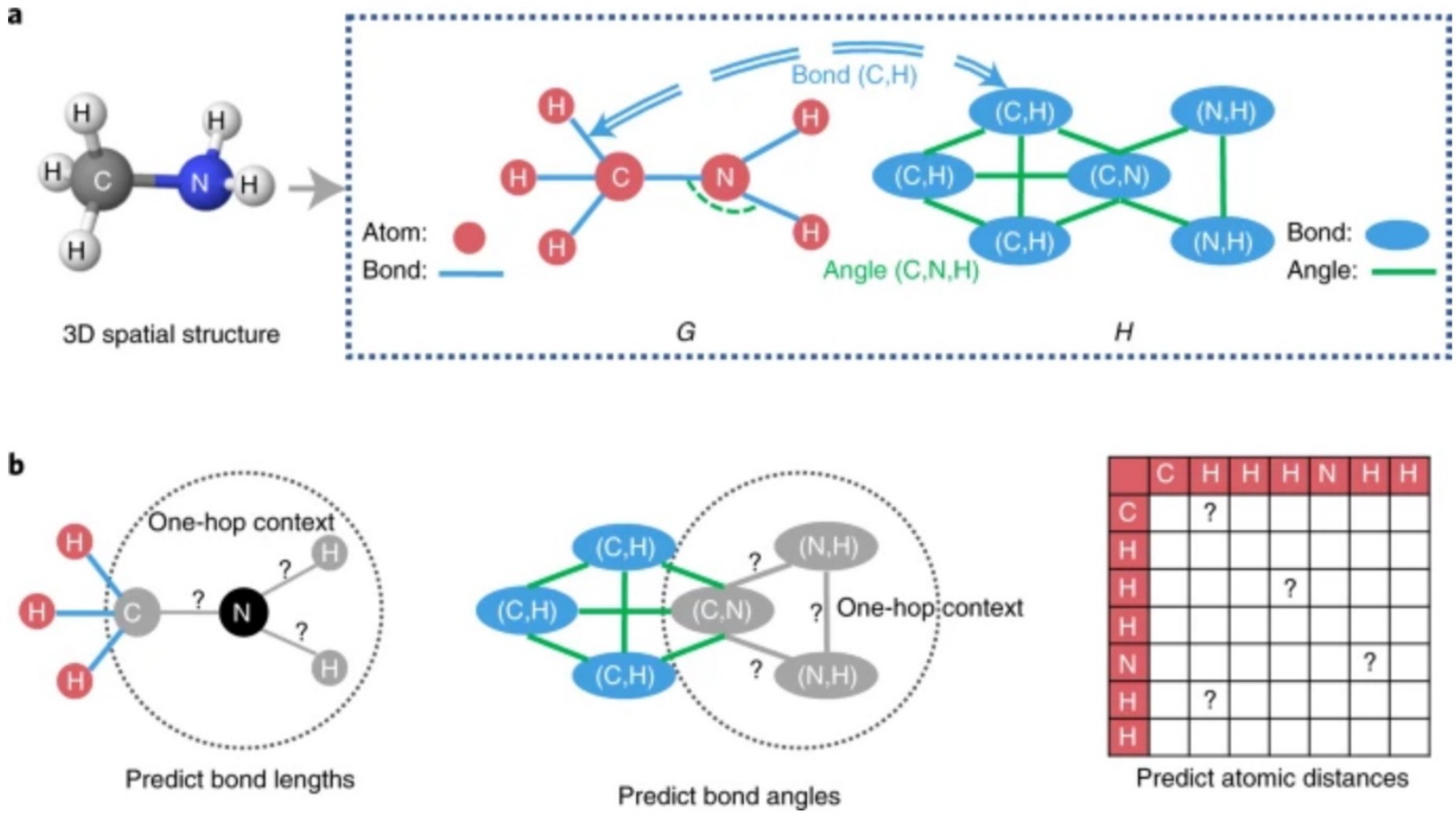
6. **Finally, it compares this predicted acceleration, with the true acceleration from the data**

Minimize

$$\left| \vec{a}(\text{pred}) - \vec{a}(\text{true}) \right|^2$$

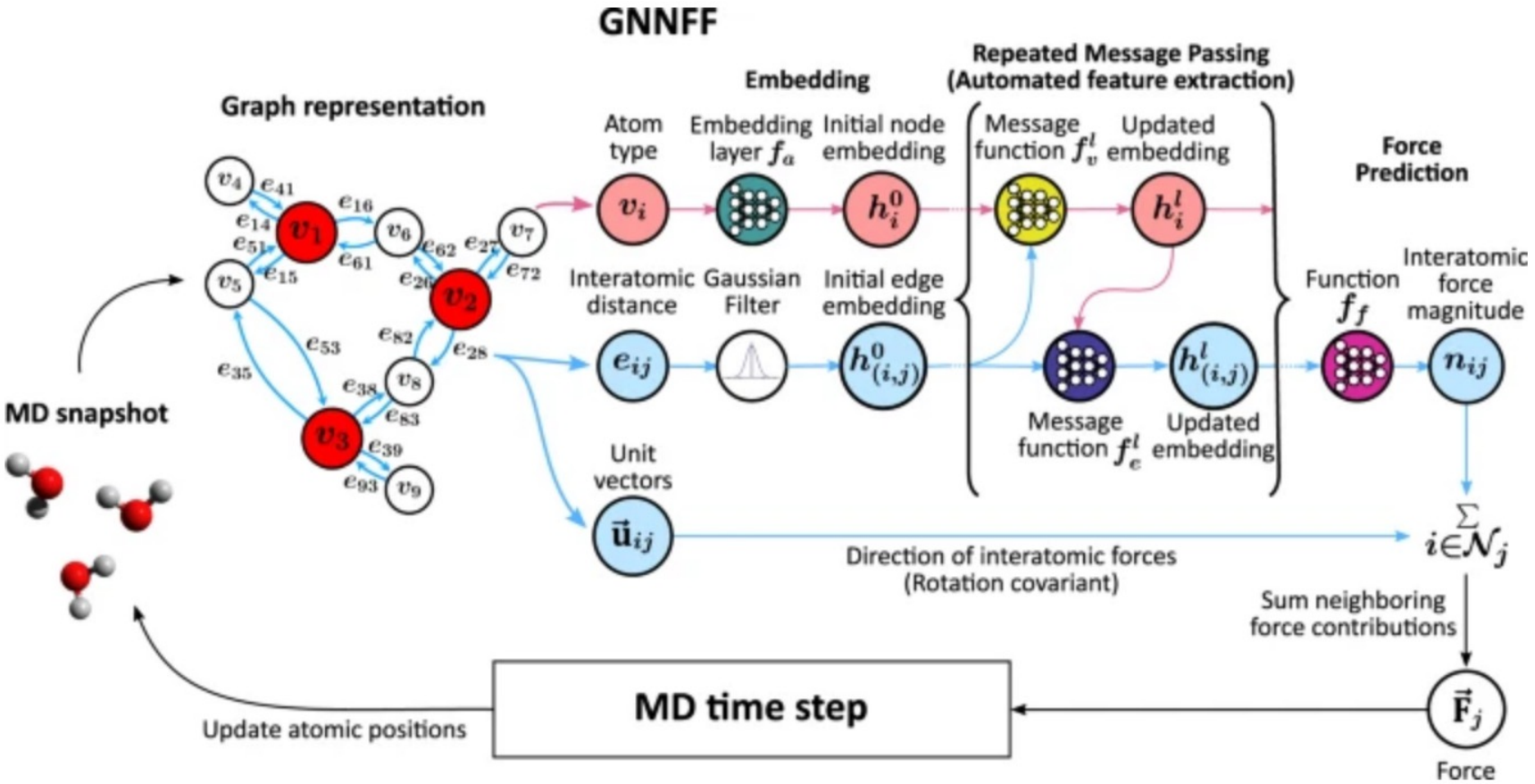
paper

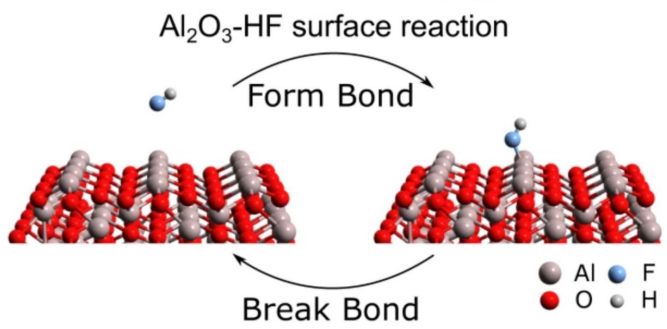
Molecular Property Prediction



paper

Materials Simulations



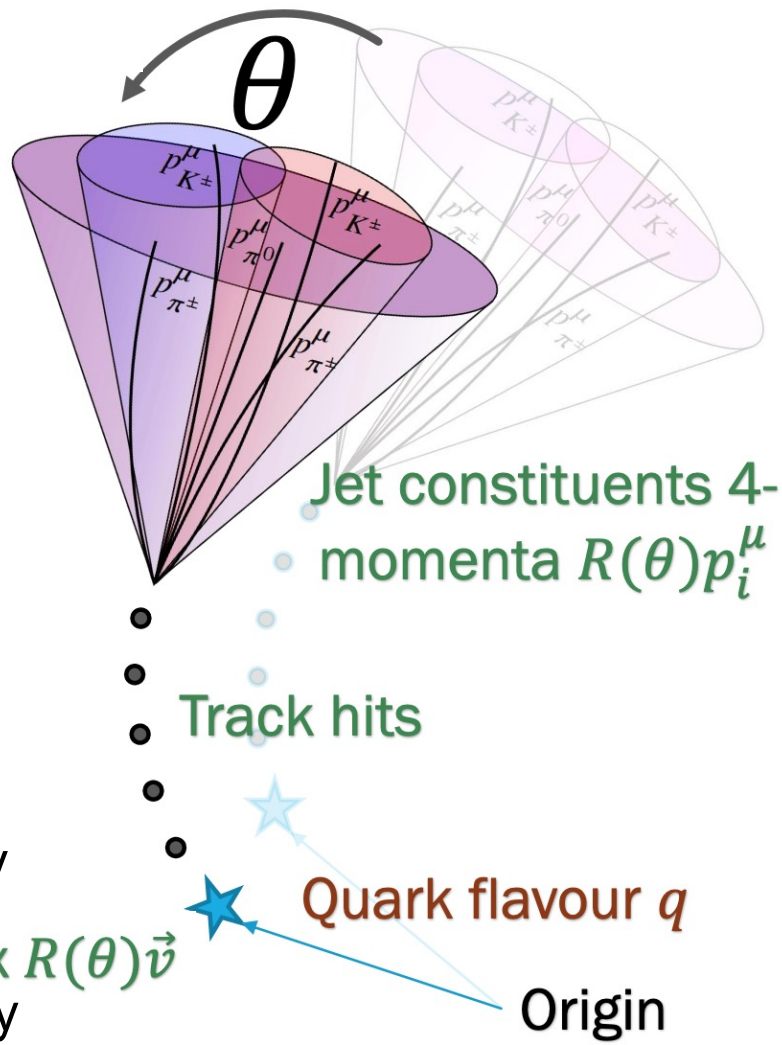


[paper]

Fun Extensions!

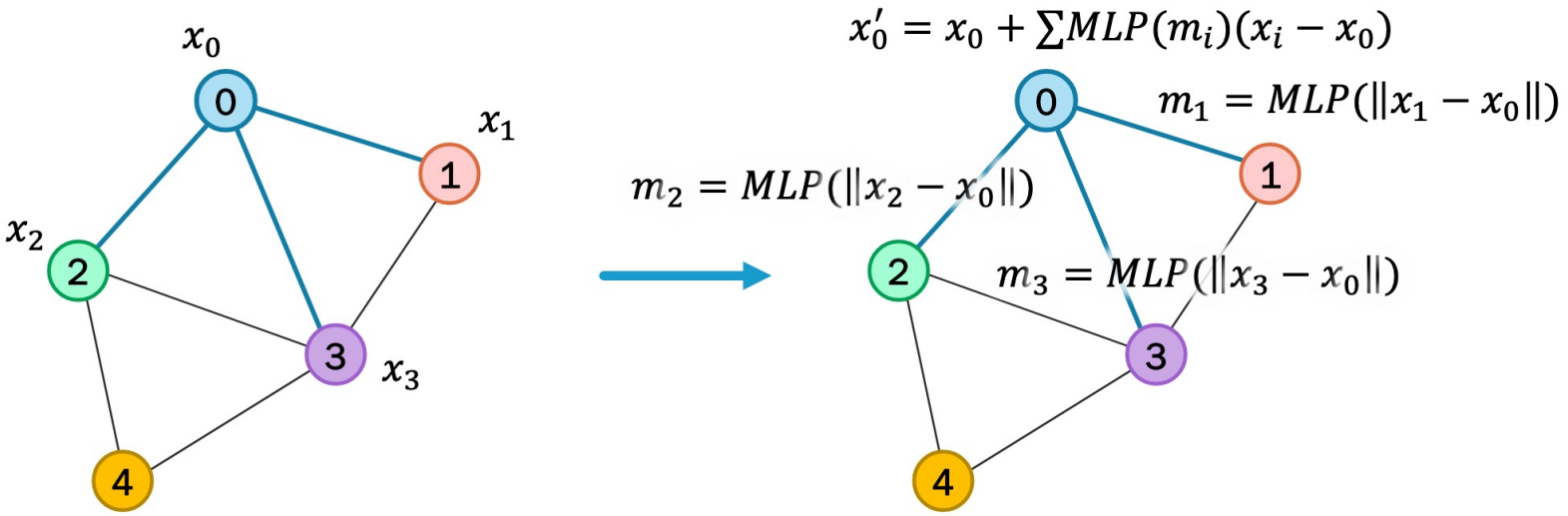
Including Symmetries

- 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



How Do We 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:



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

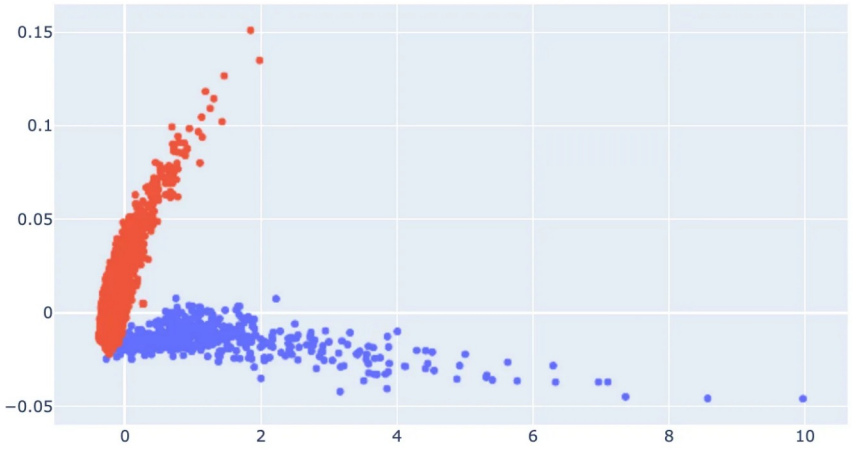
Message passing invariant to rotation and translation

Aggregation equivariant to rotation and translation

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

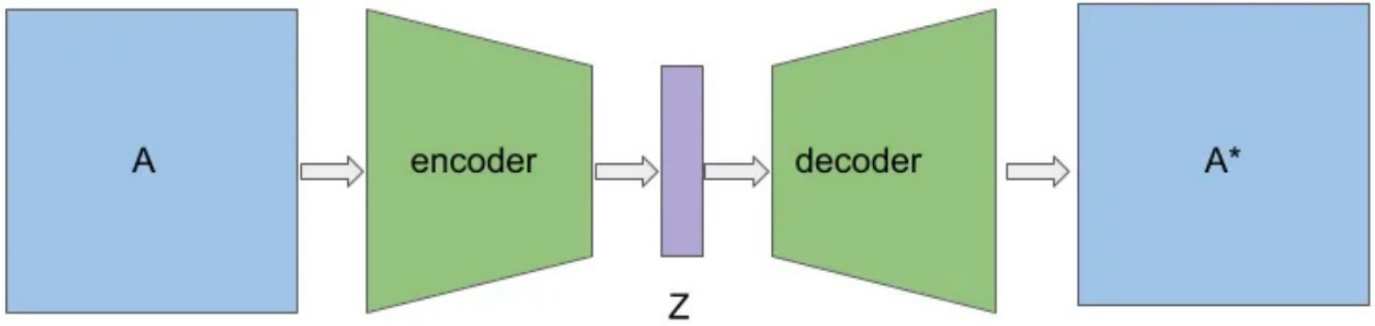
Graph Autoencoders

- 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



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

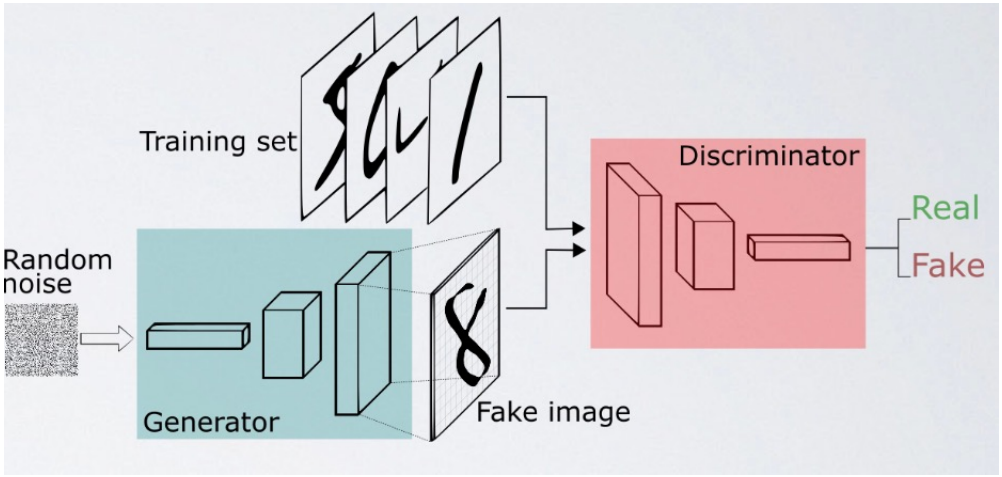
METHOD	AUC	AP
DEEPWALK	78.5± 2.25	77.4± 2.51
NODE2VEC	86.0± 1.98	84.6± 2.36
C-VGAE	**90.8**± 1.72	**91.3**± 1.69

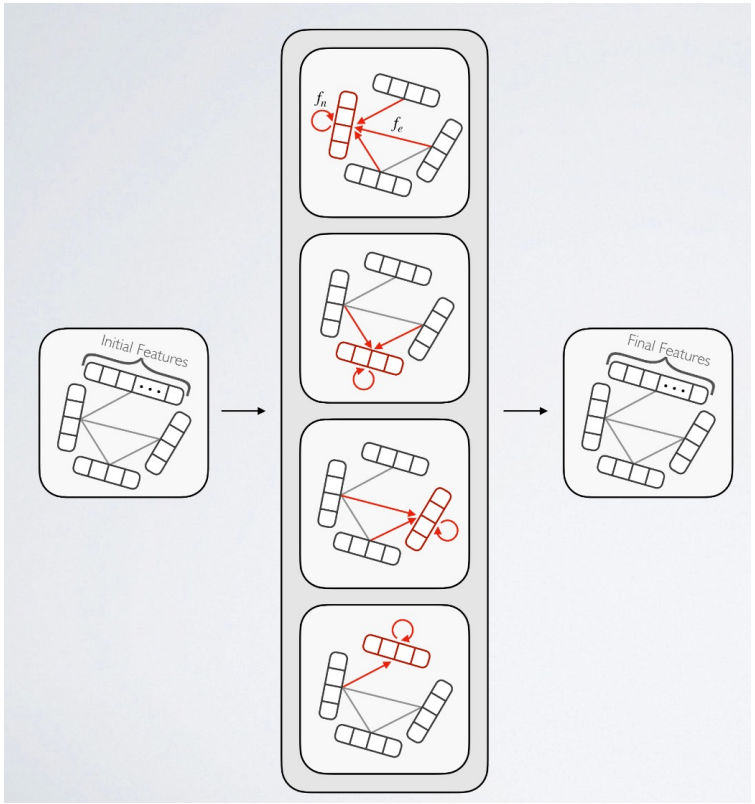


paper

Graph Generators

- 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!





recent talk

Conclusions

- Graphs are a natural representation of physics data
- Graph-based learning methods can leverage geometric information for many types of physics tasks
  - Many different GNN approaches and architectures can work, important to explore different options and think about benchmarking
  - Many techniques/insights from GDL, ML, etc can help improve work
- Active area of research with many exciting developments
  - Symmetry equivariance, generative models, and more

Thank you!

Happy to answer any questions!

✉ st3565@columbia.edu @basicsciencesav