Day 4: Introduction to Graph Neural Networks

François Drielsma
(with Kazu and Aashwin)

drielsma@slac.stanford.edu

November 18, 2020

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Day 4 Lecture | Outline

SLAC NATIONAL ACCELERATOR LABORATORY

1. **Graph Neural Networks (GNNs): what are they good for?**
   - Motivation behind the creation of GNNs
   - Introduction to graphs, graph-structured data
   - Presentation of typical tasks on graphs
2. **GNN feature learning: how does information flow in graphs ?**
   - Graph convolutions
   - Message passing
   - Graph pooling operations
3. **GNN examples: what are common GNN architectures ?**
   A. Classifying people in a social network with a GCN + *Tutorial*
   B. Particle clustering with MetaLayers + *Tutorial*
   C. Protein graph classification with with DiffPool + *Tutorial*
4. **Conclusions**

Machine Learning: learning a transformation from data (classification, regression, etc.)

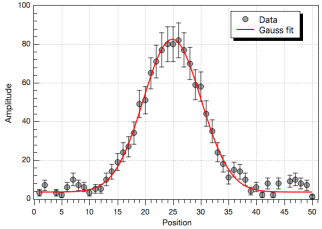
Inductive bias: assumptions about the model in a learning process (very good read)

Machine Learning: learning a transformation from data (classification, regression, etc.)

Inductive bias: assumptions about the model in a learning process (very good read)

Hand-engineered function

- Strong inductive bias
- Extremely limited freedom
- Strong potential for underfitting
- Computationally cheap



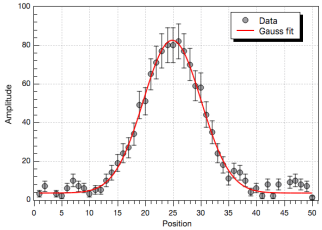
SLAC NATIONAL ACCELERATOR LABORATORY

Machine Learning: learning a transformation from data (classification, regression, etc.)

Inductive bias: assumptions about the model in a learning process (very good read)

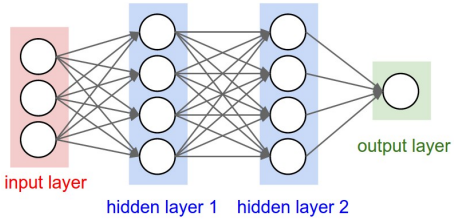
Hand-engineered function

- Strong inductive bias
- Extremely limited freedom
- Strong potential for underfitting
- Computationally cheap



Fully-connected Neural Network

- Virtually no inductive bias
- Arbitrarily infinite freedom
- No by-design overfitting prevention
- Computationally expensive for deep nets

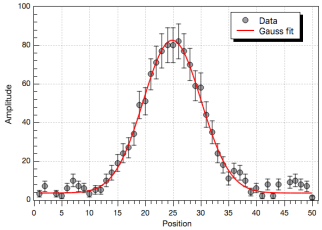


Machine Learning: learning a transformation from data (classification, regression, etc.)

Inductive bias: assumptions about the model in a learning process (very good read)

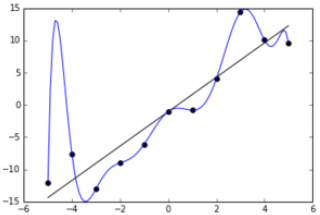
Hand-engineered function

- Strong inductive bias
- Extremely limited freedom
- Strong potential for underfitting
- Computationally cheap



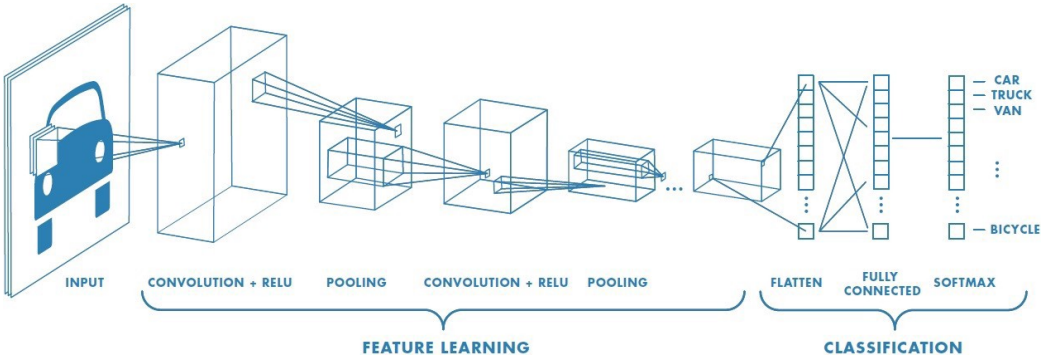
Fully-connected Neural Network

- Virtually no inductive bias
- Arbitrarily infinite freedom
- No by-design overfitting prevention
- Computationally expensive for deep nets



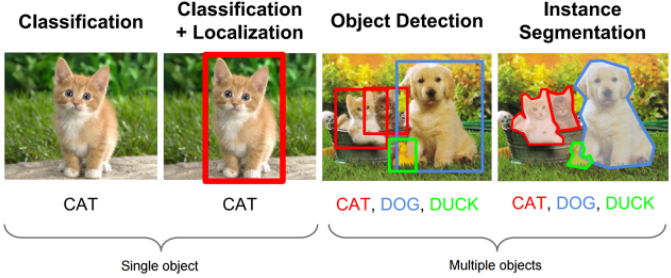
Convolutional Neural Networks: freedom of MLPs but **locality** and **translation invariance**

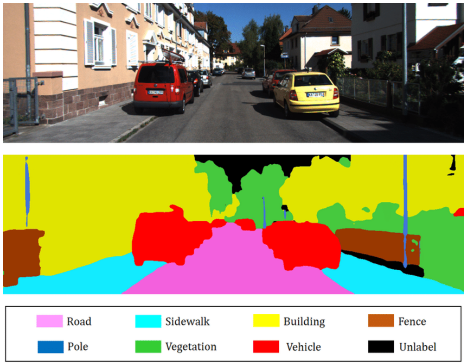
- Inductive biases:
  - Strong correlation between adjacent input pixels (receptive field small compared to full input)
  - Model should not care about object position in the data (filters are shared)



Convolutional Neural Networks: freedom of MLPs but **locality** and **translation invariance**

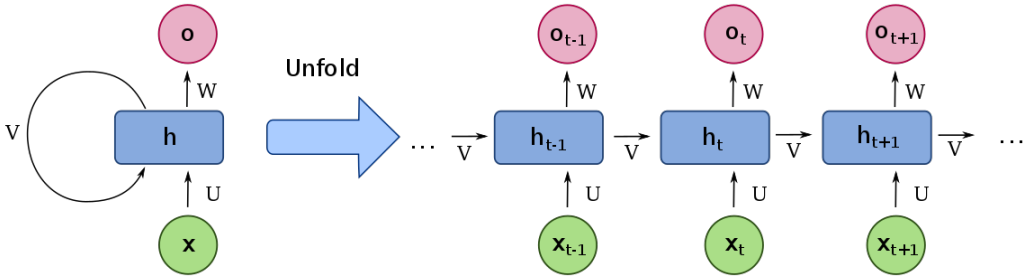
- Inductive biases:
  - Strong correlation between adjacent input pixels (receptive field small compared to full input)
  - Model should not care about object position in the data (filters are shared)
- **State-of-the-art** in Computer Vision, exclusively for **data on a grid**





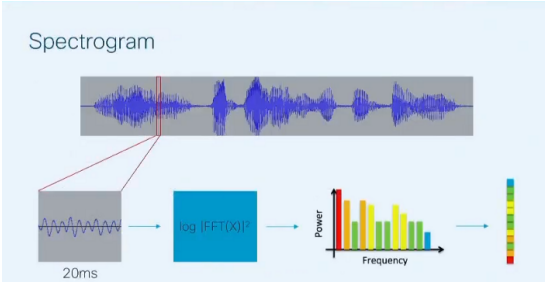
Recurrent Neural Network: freedom of MLPs but with **locality** and **temporal invariance**

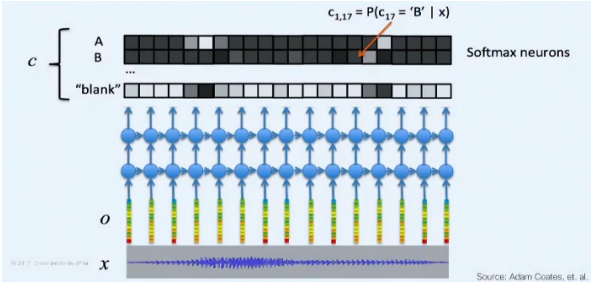
- Inductive biases:
  - Strong correlation between successive inputs (hidden state h_t takes h_{t-1} as input)
  - Model should not care about absolute time but rather sequence



Recurrent Neural Network: freedom of MLPs but with **locality** and **temporal invariance**

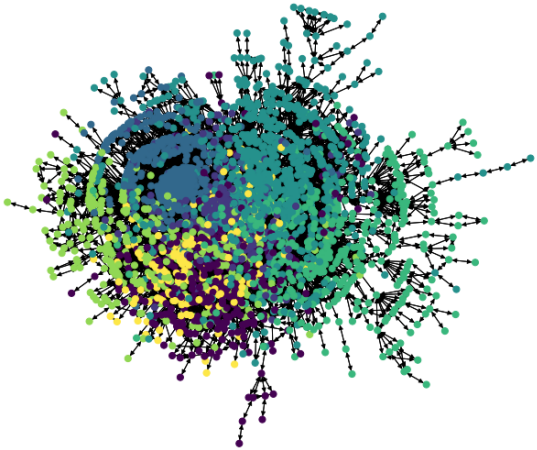
- Inductive biases:
  - Strong correlation between successive inputs (hidden state h_t takes h_{t-1} as input)
  - Model should not care about absolute time but rather sequence
- **State-of-the-art** in **speech recognition**, exclusively for **time-ordered data**





Motivation

What if the data is structured but is **neither spatially nor temporally ordered** ?

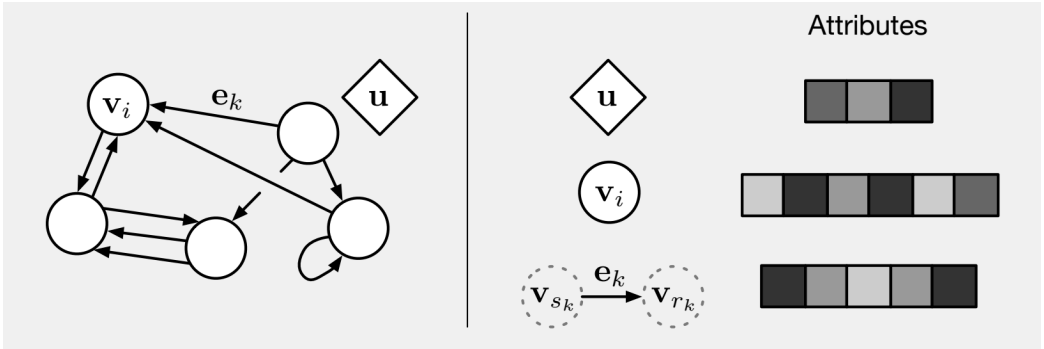


Social network: people and relationships

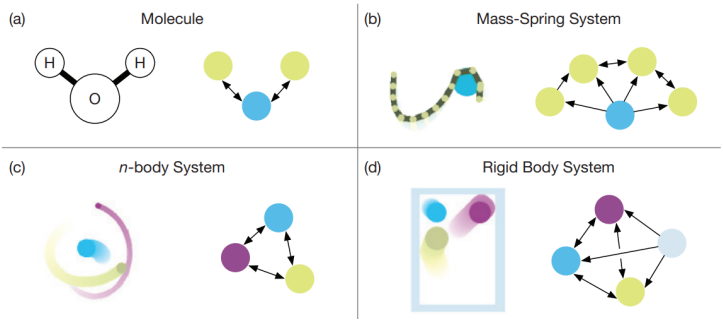
Referencing: papers and citation linkage

Graphs generalize concept of **locality** to **arbitrarily-ordered data**

- A set of **nodes**, $\{\boldsymbol{v_i}\}_{i=1}^{N_v}$ (e.g. person in social network, paper in an arXiv, etc.)
- A set of **edges**, $\{\boldsymbol{e_k}\}_{k=1}^{N_e}$ (e.g. friendship in social network, citation, etc.)
- A global state, $\boldsymbol{u}$, associated with one realization of the data (full network, full arXiv, etc.)

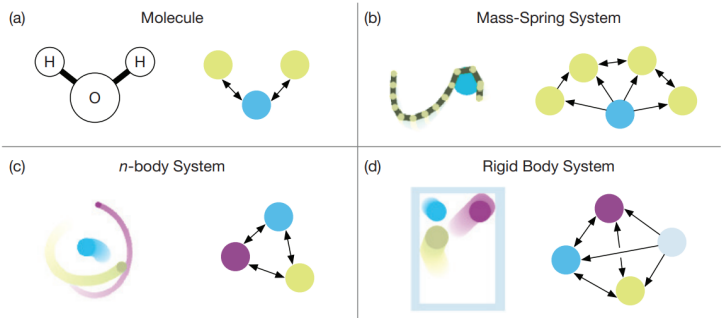


Graphs are often the natural choice to **represent correlations** in a physical data set



(a) **Nodes**: atoms (Z, A, etc.)
Edges: chemical bonds

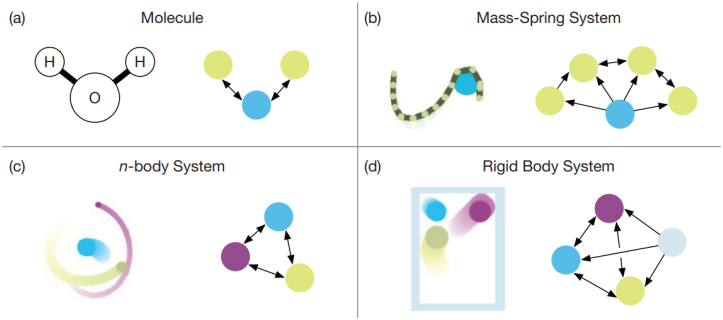
Graphs are often the natural choice to **represent correlations** in a physical data set



(a) **Nodes**: atoms (Z, A, etc.)
Edges: chemical bonds

(b) **Nodes**: discrete masses
Edges: spring force between masses

Graphs are often the natural choice to **represent correlations** in a physical data set

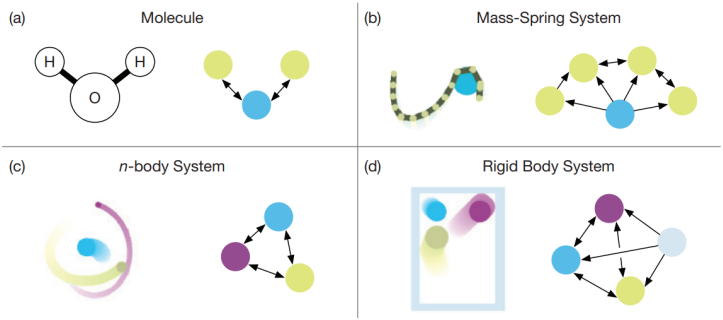


(a) **Nodes**: atoms (Z, A, etc.)
Edges: chemical bonds

(b) **Nodes**: discrete masses
Edges: spring force between masses

(c) **Nodes**: planets, molecules
Edges: attractive/repulsive forces

Graphs are often the natural choice to **represent correlations** in a physical data set



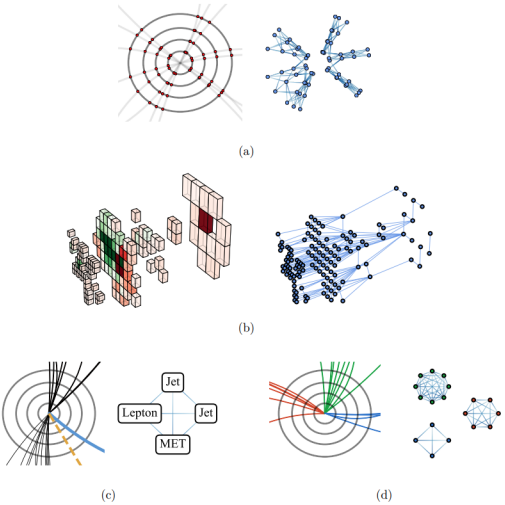
(a) **Nodes**: atoms (Z, A, etc.)
Edges: chemical bonds

(b) **Nodes**: discrete masses
Edges: spring force between masses

(c) **Nodes**: planets, molecules
Edges: attractive/repulsive forces

(d) **Nodes**: molecules, walls
Edges: attractive/repulsive forces

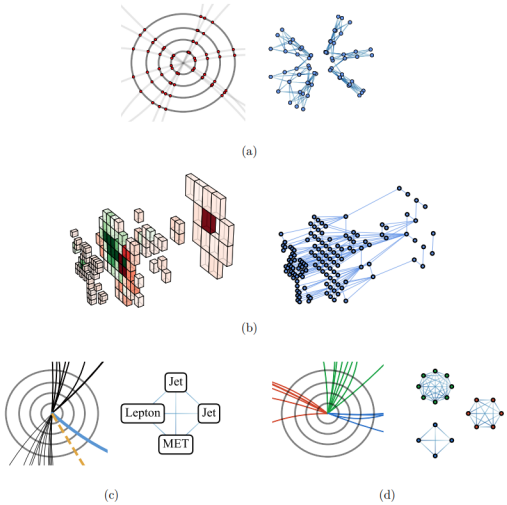
Graphs are often the natural choice to **represent correlations** in a physical data set



(a) **Nodes**: tracker hits

Edges: hit connections (tracks)

Graphs are often the natural choice to **represent correlations** in a physical data set



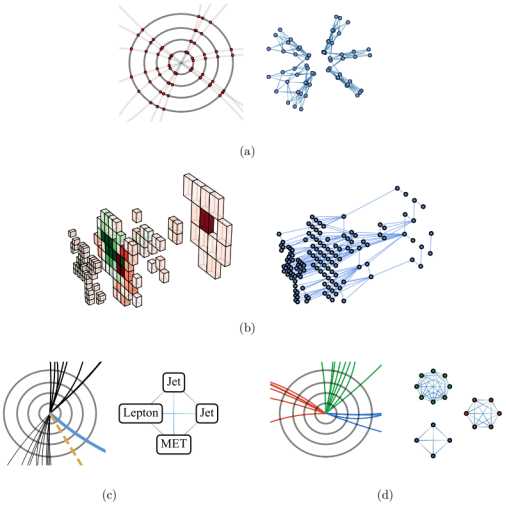
(a) **Nodes**: tracker hits
Edges: hit connections (tracks)

(b) **Nodes**: calorimeter cells
Edges: cell correlations (particles)

Graphs | In HEP

SLAC NATIONAL ACCELERATOR LABORATORY

Graphs are often the natural choice to **represent correlations** in a physical data set



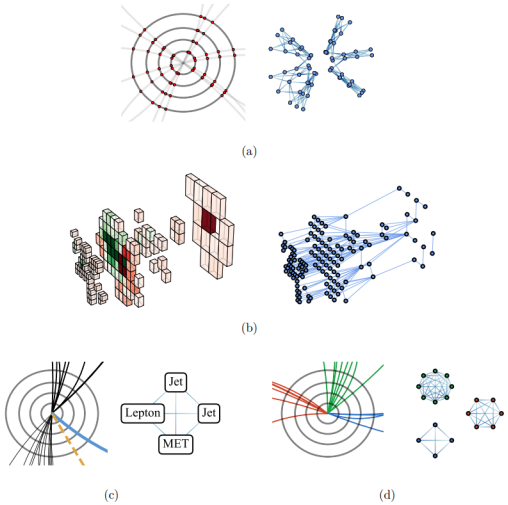
(a) **Nodes**: tracker hits
Edges: hit connections (tracks)

(b) **Nodes**: calorimeter cells
Edges: cell correlations (particles)

(c) **Nodes**: particle objects
Edges: particle correlations (events)

Graphs | In HEP

Graphs are often the natural choice to **represent correlations** in a physical data set



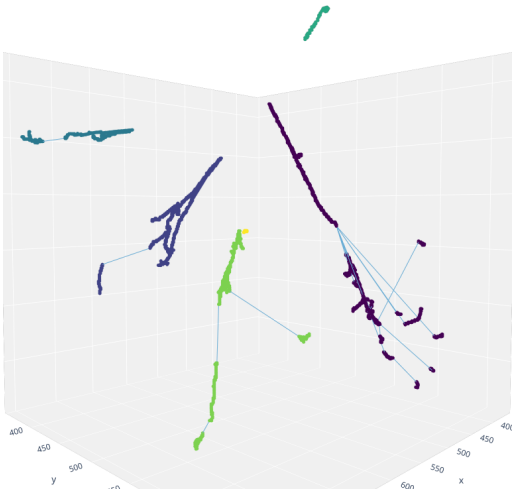
(a) **Nodes**: tracker hits
Edges: hit connections (tracks)

(b) **Nodes**: calorimeter cells
Edges: cell correlations (particles)

(c) **Nodes**: particle objects
Edges: particle correlations (events)

(d) **Nodes**: jet particles
Edges: correlations (jets)

Graphs are often the natural choice to **represent correlations** in a physical data set



(a) **Nodes**: tracker hits
Edges: hit connections (tracks)

(b) **Nodes**: calorimeter cells
Edges: cell correlations (particles)

(c) **Nodes**: particle objects
Edges: particle correlations (events)

(d) **Nodes**: jet particles
Edges: correlations (jets)

(*) **Nodes**: particles
Edges: parentage or superstructure

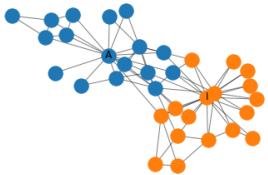
Graph Neural Networks are used to infer information about their three main components:

Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)

Karate Club

- **Nodes**: people in a club
- **Edges**: friendship status
- Club splits, who goes with A and who with I ?

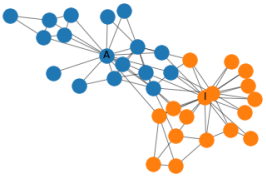


Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)

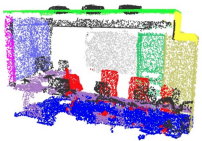
Karate Club

- **Nodes**: people in a club
- **Edges**: friendship status
- Club splits, who goes with A and who with I ?



ShapeNet

- **Nodes**: point in cloud
- **Edges**: proximity
- Classify points into different categories



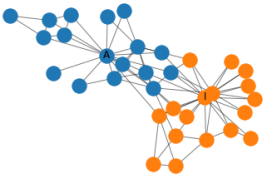
SLAC NATIONAL ACCELERATOR LABORATORY

Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)

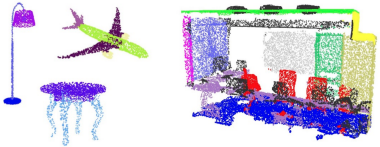
Karate Club

- **Nodes**: people in a club
- **Edges**: friendship status
- Club splits, who goes with A and who with I ?



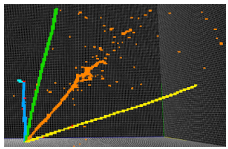
ShapeNet

- **Nodes**: point in cloud
- **Edges**: proximity
- Classify points into different categories



Particle classification

- **Nodes**: particles
- **Edges**: parentage
- Identify particle type (p, e, γ, μ, π)

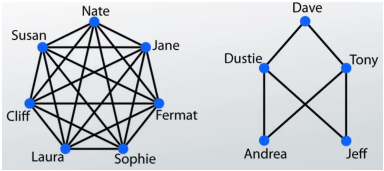


Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering

Link classification

- **Nodes**: people in a group
- **Edges**: potential links
- Are Nate and Laura friends, family, coworkers?

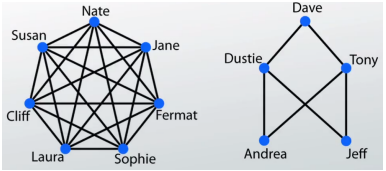


Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering

Link classification

- **Nodes**: people in a group
- **Edges**: potential links
- Are Nate and Laura friends, family, coworkers?



Scene comprehension

- **Nodes**: objects in scene
- **Edges**: all-to-all
- Is object A the same shape as B ? Is it bigger ?

Graphs

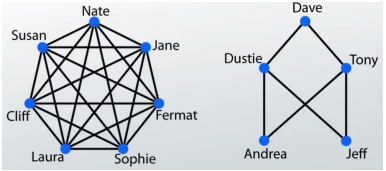
Task categories

Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering

Link classification

- **Nodes**: people in a group
- **Edges**: potential links
- Are Nate and Laura friends, family, coworkers?

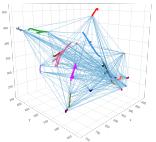


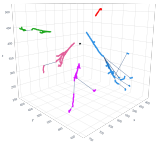
Scene comprehension

- **Nodes**: objects in scene
- **Edges**: all-to-all
- Is object A the same shape as B ? Is it bigger ?

Particle clustering

- **Nodes**: particles
- **Edges**: all-to-all
- Is this edge a true parentage edge ?





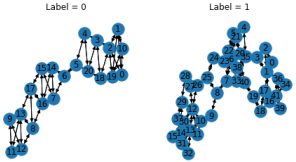
SLAC NATIONAL ACCELERATOR LABORATORY

Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering
3. **Graph**: graph classification

Enzyme classification

- **Nodes**: AA structures
- **Edges**: spatial proximity
- What is the purpose of this enzyme (protein) ?

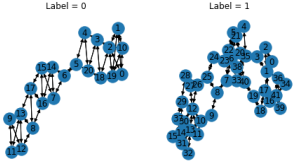


Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering
3. **Graph**: graph classification

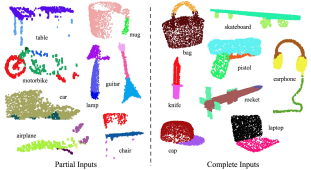
Enzyme classification

- **Nodes**: AA structures
- **Edges**: spatial proximity
- What is the purpose of this enzyme (protein) ?



Point cloud classification

- **Nodes**: point in cloud
- **Edges**: proximity
- What class of object does it belong to ?



Graphs

Task categories

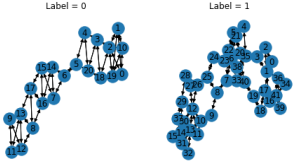
SLAC NATIONAL ACCELERATOR LABORATORY

Graph Neural Networks are used to infer information about their three main components:

1. **Nodes**: node classification, segmentation(, clustering)
2. **Edge**: edge classification, clustering
3. **Graph**: graph classification

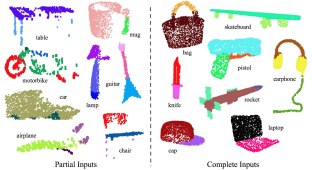
Enzyme classification

- **Nodes**: AA structures
- **Edges**: spatial proximity
- What is the purpose of this enzyme (protein) ?



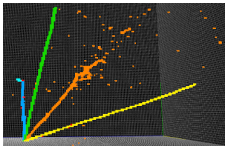
Point cloud classification

- **Nodes**: point in cloud
- **Edges**: proximity
- What class of object does it belong to ?



Interaction classification

- **Nodes**: particles
- **Edges**: parentage
- What class of interaction is this?

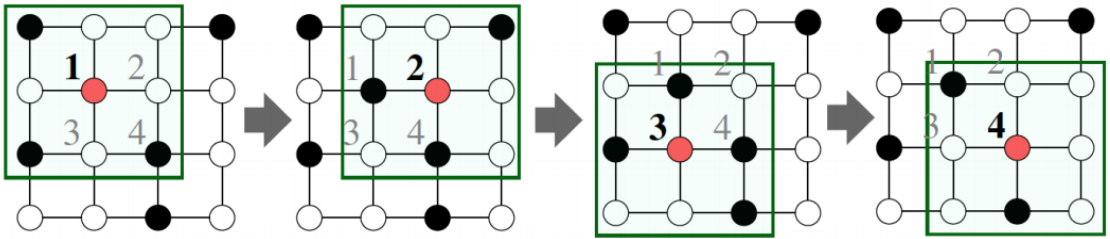


To perform the aforementioned tasks, the graph needs to develop **embeddings**, i.e. map the input features to a space in which the task can be performed.

The basic operation for CNNs is the **convolution layer**

- Aggregate a the pixel features within some distance by multiplying it by shared filter
- Formally: $\boldsymbol{x}_{i,j}^{(l+1)} = \sigma(\sum_{k \in \mathcal{N}(i)} \boldsymbol{w}_{j,k} \boldsymbol{x}_{k,j}^{(l)})$

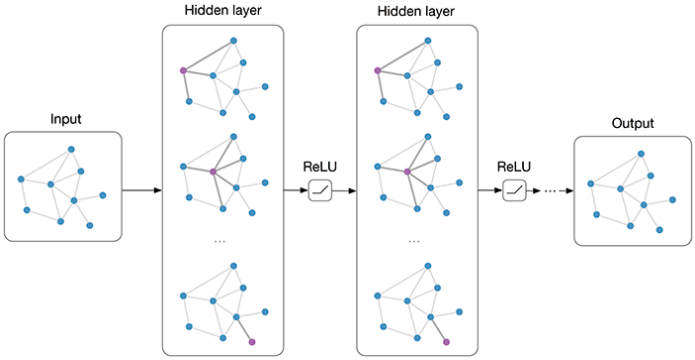
  with $x_{i,j}^{(l)}$ the j^{th} feature of pixel i at layer l and $w_{j,k}$ the k^{th} cell of filter j



Graph Convolutional Networks (GCNs) are an early implementation of GNNs which take:

- An (N, N_v) matrix, V, of **node features**
- An (N, N) **adjacency matrix**, A, only used for neighborhood aggregation

  NB: The adjacency matrix is a dense formulation of edges, in which $A_{ij} = 1$ if the corresponding edge $e_{\{k|i=s_k, j=r_k\}}$ exists in the graph, $A_{ij} = 0$ otherwise



GNNs don't have regular neighborhoods, but why not use the adjacency matrix?

$$h_i^{(l+1)} = \sigma(\sum_{j=1}^{N} A_{i,j} h_j^{(l)} W^{(l)}), \qquad \text{or} \quad H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}), \tag{1}$$

where σ is an activation function, $W^{(l)}$ is an $(N_v^{(l-1)}, N_v^{(l)})$ **weight matrix shared across all nodes** (similar to to the convolution filters of CNNs).

GNNs don't have regular neighborhoods, but why not use the adjacency matrix?

$$h_i^{(l+1)} = \sigma(\sum_{j=1}^{N} A_{i,j} h_j^{(l)} W^{(l)}), \qquad \text{or} \quad H^{(l+1)} = \sigma(A H^{(l)} W^{(l)}), \tag{1}$$

where σ is an activation function, $W^{(l)}$ is an $(N_v^{(l-1)}, N_v^{(l)})$ **weight matrix shared across all nodes** (similar to to the convolution filters of CNNs).

Two caveats:

- A must contain self-loops (otherwise no self-preservation): $\hat{A} = A + I$
- The nodes with more connections (higher degree) have larger features. That is not desirable, so divide by degree matrix $\hat{D}$ of $\hat{A}$.

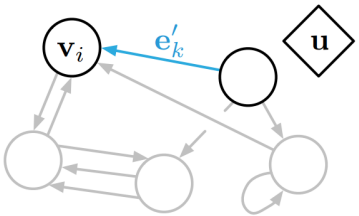
$$h_i^{(l+1)} = \sigma(\frac{1}{\hat{d}_i} \sum_{j=1}^{N} \hat{A}_{i,j} h_j^{(l)} W^{(l)}), \qquad \text{or} \quad H^{(l+1)} = \sigma(\hat{D}^{-1} \hat{A} H^{(l)} W^{(l)}), \tag{2}$$

The GCN neighborhood aggregation method, while elegant in its one-to-one similarity with pixel convolutions in CNN, is inherently limited:

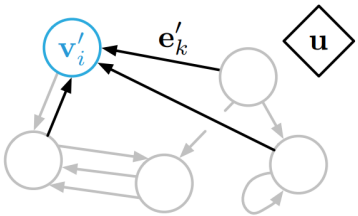
- It does not support edge-specific features
- It does not support explicit graph-specific features
- It does not support other neighborhood aggregation strategies than a simple sum

$\rightarrow$ How can we build features with a graph in all its complexity ?

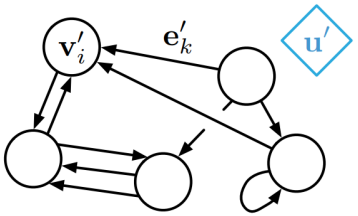
The most general for GNNs is called **message passing**



(a) Edge update



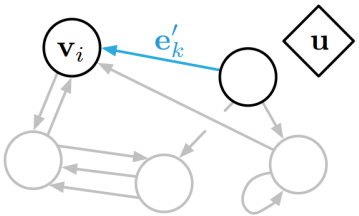
(b) Node update



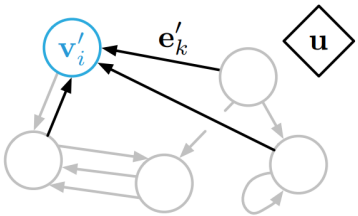
(c) Global update

The most general for GNNs is called **message passing**

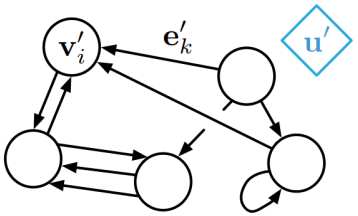
1. **Edge update**: fold adjacent node features and global features into an edge
   $\boldsymbol{e}'_k = \phi_e(\boldsymbol{e}_k, \boldsymbol{v}_{r_k}, \boldsymbol{v}_{s_k}, \boldsymbol{u})$, with ϕ_e a learnable function



(a) Edge update



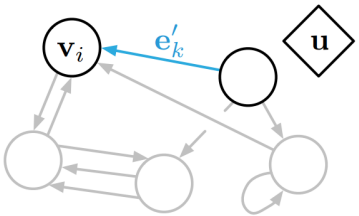
(b) Node update



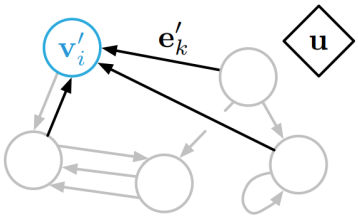
(c) Global update

The most general for GNNs is called **message passing**

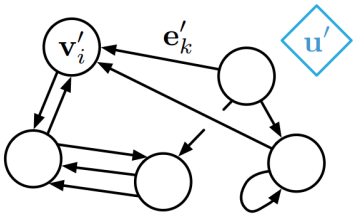
1. **Edge update**: fold adjacent node features and global features into an edge
   $\boldsymbol{e}'_k = \phi_e(\boldsymbol{e}_k, \boldsymbol{v}_{r_k}, \boldsymbol{v}_{s_k}, \boldsymbol{u})$, with ϕ_e a learnable function
2. **Node update**: fold adjacent edge features and global features into a node
   $\boldsymbol{v}'_i = \phi_v(\rho_{e,v}(E'_i), \boldsymbol{x}_i, \boldsymbol{u})$, with ϕ_v a learnable function, $\rho_{e,v}$ an aggregator and $E_i = \{\mathbf{e}_k\}_{k \in \mathcal{N}(i)}$



(a) Edge update



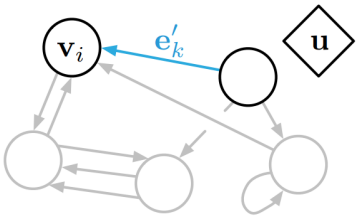
(b) Node update



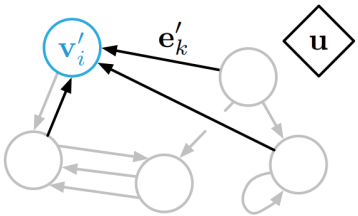
(c) Global update

The most general for GNNs is called **message passing**

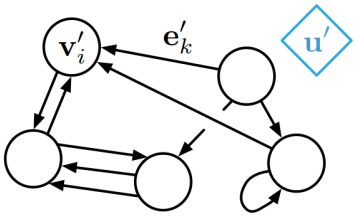
1. **Edge update**: fold adjacent node features and global features into an edge
   $\boldsymbol{e}'_k = \phi_e(\boldsymbol{e}_k, \boldsymbol{v}_{r_k}, \boldsymbol{v}_{s_k}, \boldsymbol{u})$, with ϕ_e a learnable function
2. **Node update**: fold adjacent edge features and global features into a node
   $\boldsymbol{v}'_i = \phi_v(\rho_{e,v}(E'_i), \boldsymbol{x}_i, \boldsymbol{u})$, with ϕ_v a learnable function, $\rho_{e,v}$ an aggregator and $E_i = \{\mathbf{e}_k\}_{k\in\mathcal{N}(i)}$
3. **Global update**: fold all node and edge features to update the global state
   $\boldsymbol{u}' = \phi_u(\rho_{e,u}(E'), \rho_{v,u}(V'), \boldsymbol{u})$, with ϕ_u a learnable function, $E' = \{\boldsymbol{e}_k\}_{k=1}^{N_e}$ and $E' = \{\boldsymbol{v}_i\}_{i=1}^{N_v}$



(a) Edge update

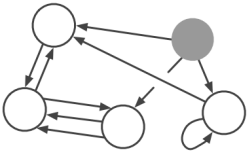


(b) Node update

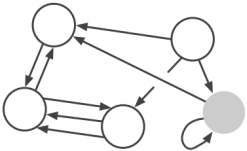


(c) Global update

The basic operation for GNNs is called **neighborhood aggregation** or **message passing**

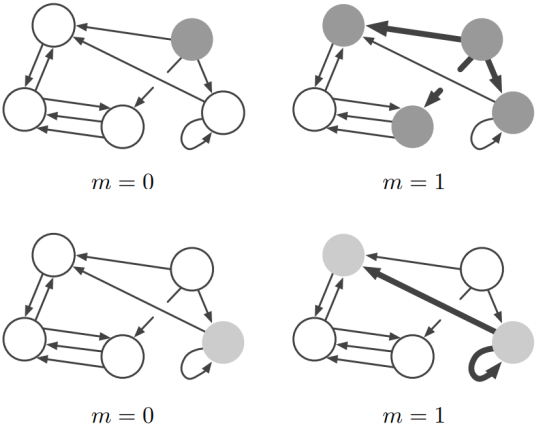


$m = 0$

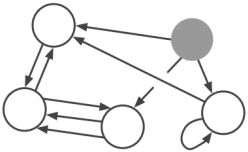


$m = 0$

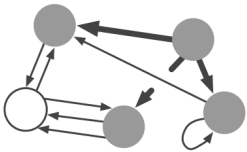
The basic operation for GNNs is called **neighborhood aggregation** or **message passing**



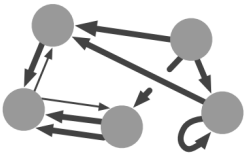
The basic operation for GNNs is called **neighborhood aggregation** or **message passing**



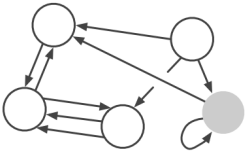
$m = 0$



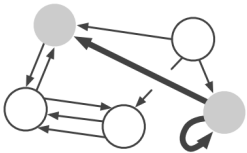
$m = 1$



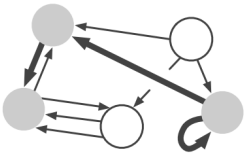
$m = 2$



$m = 0$



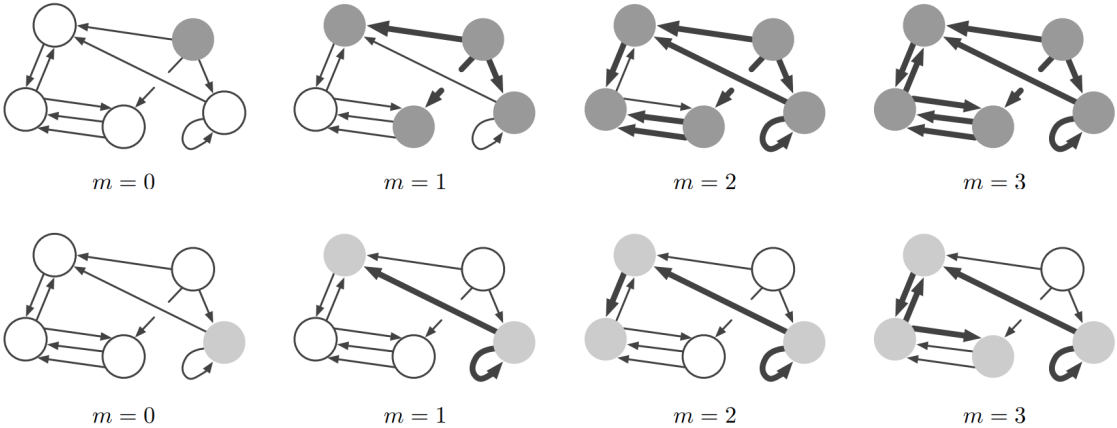
$m = 1$



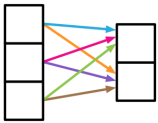
$m = 2$

SLAC NATIONAL ACCELERATOR LABORATORY

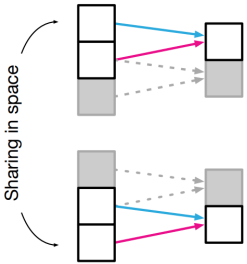
The basic operation for GNNs is called **neighborhood aggregation** or **message passing**



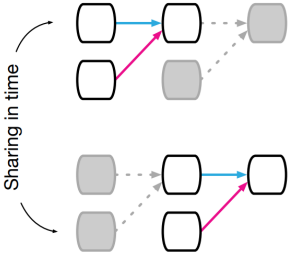
Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations



(a) Fully connected

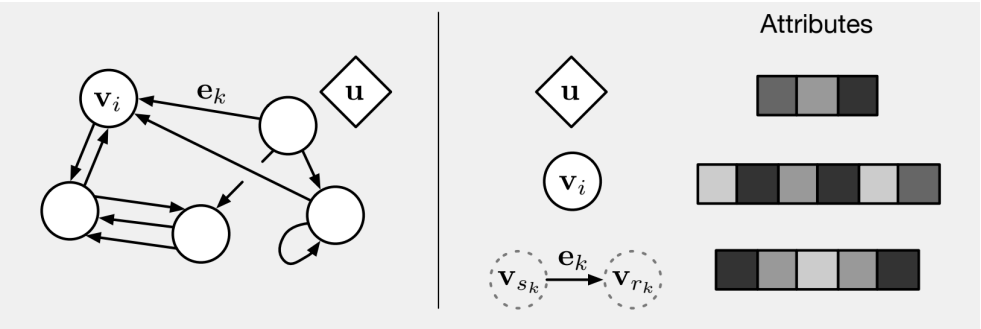


(b) Convolutional



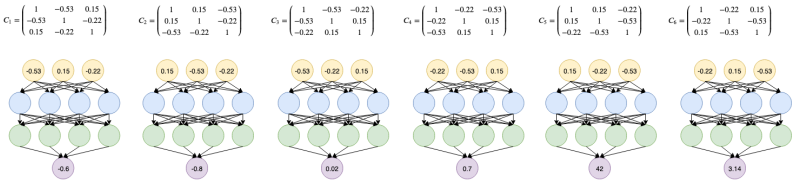
(c) Recurrent

Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations



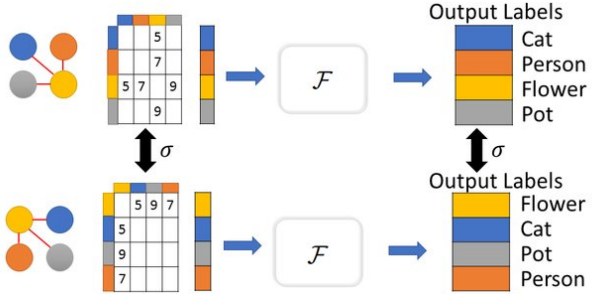
NATIONAL ACCELERATOR LABORATORY

MLP



$$\exists \pi \in \mathfrak{S}_n, f(x_1, x_2, \ldots, x_M) \neq f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)})$$

GNN

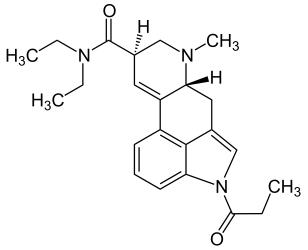


From nodes, sometimes the graph edges come naturally...

- Bonds on a molecule, springs in physical system, etc.

... and sometimes they do not:

- Point clouds, particle clustering, etc.



SLAC NATIONAL ACCELERATOR LABORATORY

From nodes, sometimes the graph edges come naturally...

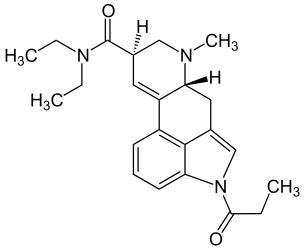
- Bonds on a molecule, springs in physical system, etc.

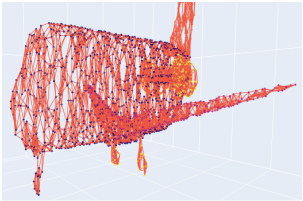
... and sometimes they do not:

- Point clouds, particle clustering, etc.

Strategy depends on individual cases:

- **Distance-based** graphs are common for Euclidean data (max edge length equivalent to CNN receptive field)





From nodes, sometimes the graph edges come naturally...

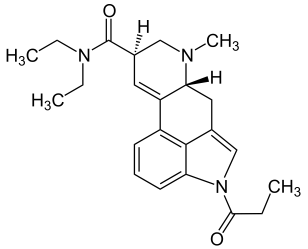
- Bonds on a molecule, springs in physical system, etc.

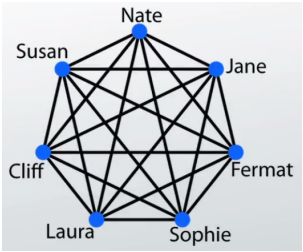
... and sometimes they do not:

- Point clouds, particle clustering, etc.

Strategy depends on individual cases:

- **Distance-based** graphs are common for Euclidean data (max edge length equivalent to CNN receptive field)
- **Complete** graphs offer maximum information flow when no prior knowledge of correlations (e.g. edge prediction)





SLAC NATIONAL ACCELERATOR LABORATORY

From nodes, sometimes the graph edges come naturally...

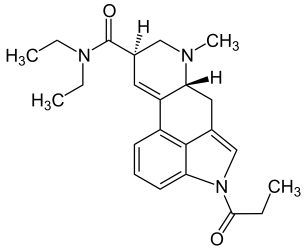
- Bonds on a molecule, springs in physical system, etc.

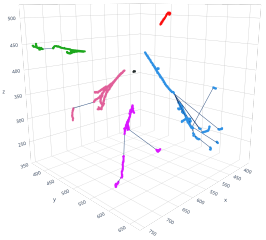
... and sometimes they do not:

- Point clouds, particle clustering, etc.

Strategy depends on individual cases:

- **Distance-based** graphs are common for Euclidean data (max edge length equivalent to CNN receptive field)
- **Complete** graphs offer maximum information flow when no prior knowledge of correlations (e.g. edge prediction)
- **Directed** graphs are preferred over **undirected** graph if edge direction is meaningful (e.g. particle flow)





SLAC NATIONAL ACCELERATOR LABORATORY

From nodes, sometimes the graph edges come naturally...

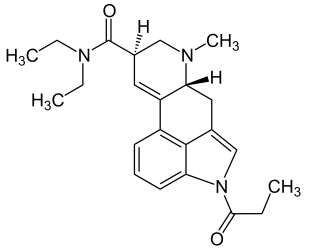
- Bonds on a molecule, springs in physical system, etc.

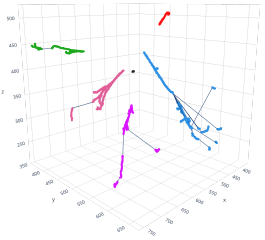
... and sometimes they do not:

- Point clouds, particle clustering, etc.

Strategy depends on individual cases:

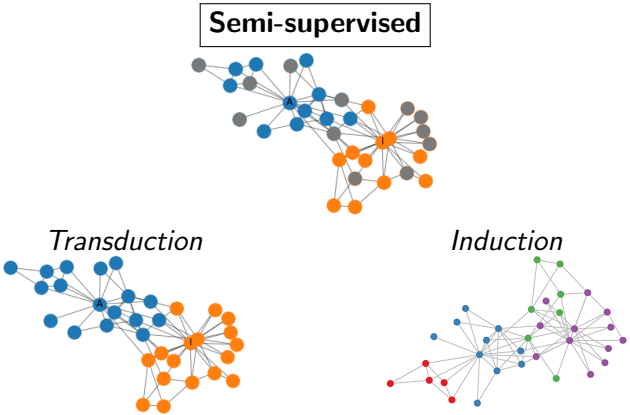
- **Distance-based** graphs are common for Euclidean data (max edge length equivalent to CNN receptive field)
- **Complete** graphs offer maximum information flow when no prior knowledge of correlations (e.g. edge prediction)
- **Directed** graphs are preferred over **undirected** graph if edge direction is meaningful (e.g. particle flow)
- Edges can also **dynamically** evolve during training





GNNs have the peculiarity of supporting either **transductive** of **inductive inference**

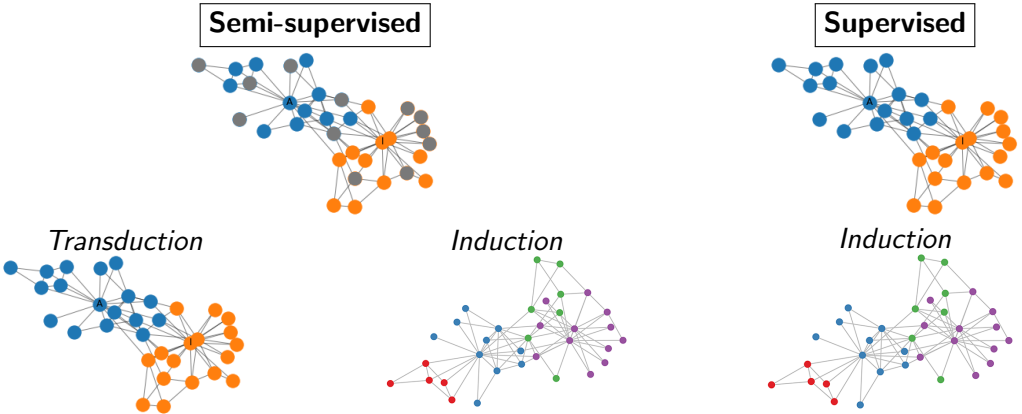
- **Semi-supervised** learning: some of the nodes in the input graph(s) are unlabeled



SLAC NATIONAL ACCELERATOR LABORATORY

GNNs have the peculiarity of supporting either **transductive** of **inductive inference**

- **Semi-supervised** learning: some of the nodes in the input graph(s) are unlabeled
- **Supervised** learning: all of the nodes in the input graphs(s) are labeled



After developing node (+edge) features, if one wants to infer graph-wide properties, the features must be pooled. The simplest method is **single-step aggregation**:

- Recall from earlier: $\boldsymbol{u} = \phi_u(\rho_{e,u}(E'), \rho_{v,u}(V'))$

  with ϕ_u a learnable function, $\rho_{e,u}$, $\rho_{v,u}$ edge and node feature aggregators, respectively

After developing node (+edge) features, if one wants to infer graph-wide properties, the features must be pooled. The simplest method is **single-step aggregation**:

- Recall from earlier: $\boldsymbol{u} = \phi_u(\rho_{e,u}(E'), \rho_{v,u}(V'))$

  with ϕ_u a learnable function, $\rho_{e,u}$, $\rho_{v,u}$ edge and node feature aggregators, respectively
- The aggregators are typically one of sum, mean or max

After developing node (+edge) features, if one wants to infer graph-wide properties, the features must be pooled. The simplest method is **single-step aggregation**:

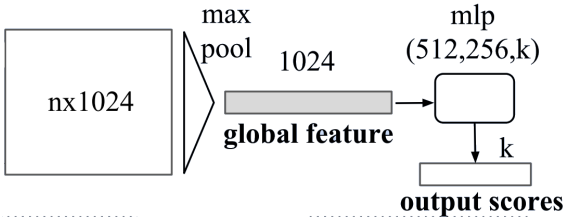
- Recall from earlier: $\boldsymbol{u} = \phi_u(\rho_{e,u}(E'), \rho_{v,u}(V'))$

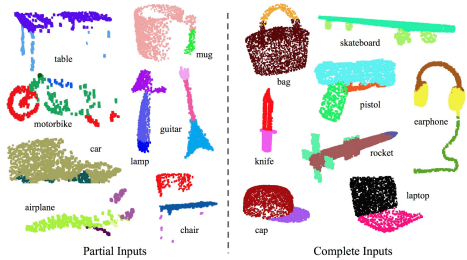
  with ϕ_u a learnable function, $\rho_{e,u}$, $\rho_{v,u}$ edge and node feature aggregators, respectively
- The aggregators are typically one of sum, mean or max
- Function ϕ_u is typically an MLP

After developing node (+edge) features, if one wants to infer graph-wide properties, the features must be pooled. The simplest method is **single-step aggregation**:

- Recall from earlier: $\boldsymbol{u} = \phi_u(\rho_{e,u}(E'), \rho_{v,u}(V'))$
  with ϕ_u a learnable function, $\rho_{e,u}$, $\rho_{v,u}$ edge and node feature aggregators, respectively
- The aggregators are typically one of sum, mean or max
- Function ϕ_u is typically an MLP

Example: **PointNet** used to classify 3D shapes (arXiv:1612.00593)





Single-step not the SOTA because it only convolves local neighborhood of nodes. As in CNN, one wants to progressively extract more / features. **Hierarchical pooling** iteratively uses:

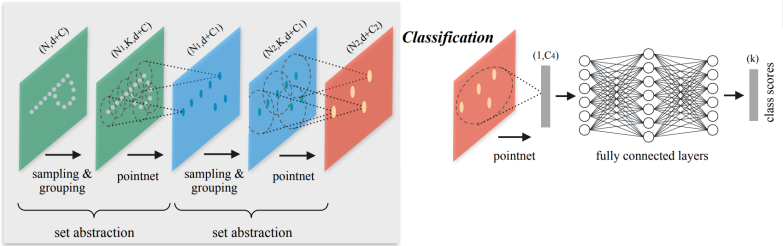
- **Sampling**: reduce the number of nodes between layers
- **Grouping**: aggregate the features of the nodes that are gone

Single-step not the SOTA because it only convolves local neighborhood of nodes. As in CNN, one wants to progressively extract more / features. **Hierarchical pooling** iteratively uses:

- **Sampling**: reduce the number of nodes between layers
- **Grouping**: aggregate the features of the nodes that are gone

Example 1: PointNet++ for point cloud classification

- Iterative farthest point sampling
- Max-pool grouping over neighborhood of new nodes

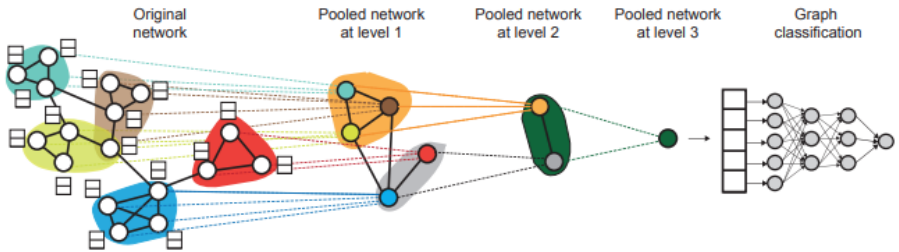


Single-step not the SOTA because it only convolves local neighborhood of nodes. As in CNN, one wants to progressively extract more / features. **Hierarchical pooling** iteratively uses:

- **Sampling**: reduce the number of nodes between layers
- **Grouping**: aggregate the features of the nodes that are gone

Example 2: DiffPool for arbitrary graph classification

- At each step, let the network learn a $(N_v^{(l)}, N_v^{(l+1)})$ **assignment matrix** $S^{(l)}$ (w/ GNN):
  $H^{(l+1)} = {S^{(l)}}^T H^{(l)}, \qquad A^{(l+1)} = {S^{(l)}}^T A^{(l)} S^{(l)}$



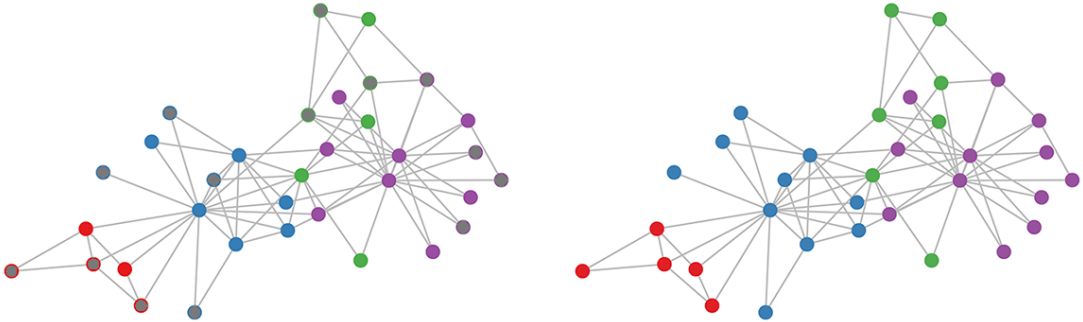
Example A: Social network | Task

SLAC NATIONAL ACCELERATOR LABORATORY

Dataset: **Zachary's Karate club**, famous early (70s) example of a social network problem

- Consists of $N = 34$ nodes (individuals), with no node features (no information)
- Has 78 edges, corresponding to pairs of people who talk to each other outside of the club
- The group splits (in this example four ways), who goes with in what group ?

Goal: Some nodes are known, some are unknown, classify unknown nodes (semi-supervised)



Embedding visualization | t-SNE

SLAC NATIONAL ACCELERATOR LABORATORY

An aside: how to represent point in a feature space of dimension > 3 ?

The t-distributed Stochastic Neighbor Embedding is a transformation which:

1. Computes the similarity between every pair of N points in d dimensions:
   $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$, with $p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$, and σ_i adpatative to local density

Embedding visualization | t-SNE

SLAC NATIONAL ACCELERATOR LABORATORY

An aside: how to represent point in a feature space of dimension > 3 ?

The t-distributed Stochastic Neighbor Embedding is a transformation which:

1. Computes the similarity between every pair of N points in d dimensions:
   $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$, with $p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$, and σ_i adpatative to local density
2. Defines a new set of point, $\{\boldsymbol{y}_i\}_{i=1}^N$, in a new, smaller dimensional d'-space

An aside: how to represent point in a feature space of dimension > 3 ?

The t-distributed Stochastic Neighbor Embedding is a transformation which:

1. Computes the similarity between every pair of N points in d dimensions:
   $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2N}$, with $p_{j|i} = \frac{\exp(-\|\mathbf{x}_i-\mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i-\mathbf{x}_k\|^2/2\sigma_i^2)}$, and σ_i adpatative to local density
2. Defines a new set of point, $\{\boldsymbol{y}_i\}_{i=1}^N$, in a new, smaller dimensional d'-space
3. Defines the t-distributed (strong tails) similarity of points in the new space as
   $q_{ij} = \frac{(1+\|\mathbf{y}_i-\mathbf{y}_j\|^2)^{-1}}{\sum_k\sum_{l\neq k}(1+\|\mathbf{y}_k-\mathbf{y}_l\|^2)^{-1}}$

An aside: how to represent point in a feature space of dimension > 3 ?

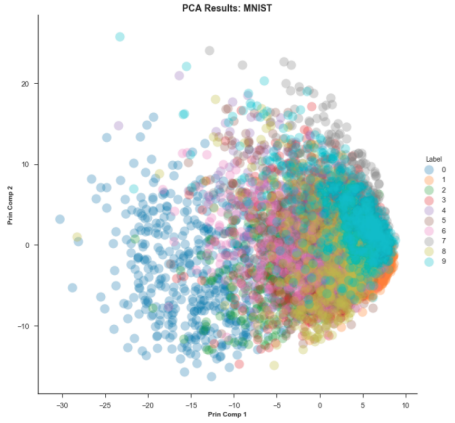
The t-distributed Stochastic Neighbor Embedding is a transformation which:

1. Computes the similarity between every pair of N points in d dimensions:
   $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$, with $p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2)}$, and σ_i adpatative to local density
2. Defines a new set of point, $\{\boldsymbol{y}_i\}_{i=1}^N$, in a new, smaller dimensional d'-space
3. Defines the t-distributed (strong tails) similarity of points in the new space as
   $q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}$
4. Minimize the KL-divergence (see Kazu day 1) between the two set of similarities using gradient decent (optimization)
   $\mathrm{KL}(P \parallel Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$

Embedding visualization | t-SNE

Fundamentally different approach (albeit much more complex) to PCA:

- PCA maximizes variance in the lower-dimensional space

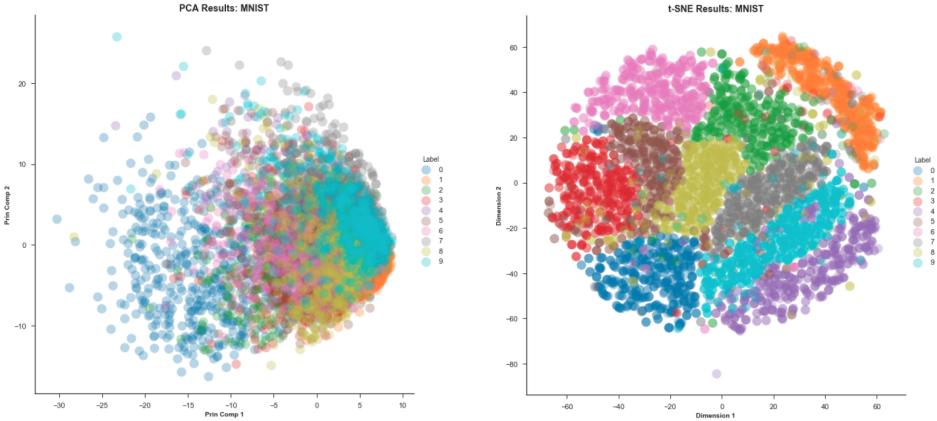


Embedding visualization | t-SNE

NATIONAL ACCELERATOR LABORATORY

Fundamentally different approach (albeit much more complex) to PCA:

- PCA maximizes variance in the lower-dimensional space
- t-SNE preserves data point similarity (close in original space $\rightarrow$ close in lower dimension)

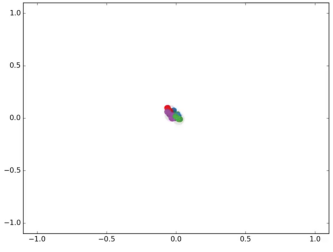


These are the steps of the feedforward network:

- Pass the nodes through two GCN layers
  1. Increase node features from 1 (blank) to 16
  2. Maintain 16 features in the convolution
- Pass the nodes features through a fully connected linear classification layer $(16 \to 4)$
- Apply the class-wise mean cross-entropy loss on node scores

$$L = -\frac{1}{N_c}\frac{1}{N}\sum_i \sum_c s_{i,c} \times \ln(s_{i,c}), \qquad s_i = \text{softmax}(x_i)$$

- Apply storchastic gradient decent 300 times



Watch the embeddings before the classification layer evolve: video

These are the steps of the feedforward network:

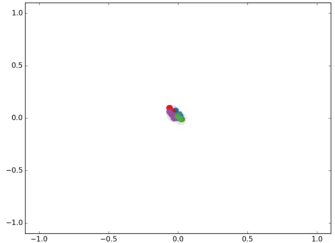
- Pass the nodes through two GCN layers
  1. Increase node features from 1 (blank) to 16
  2. Maintain 16 features in the convolution
- Pass the nodes features through a fully connected linear classification layer $(16 \to 4)$
- Apply the class-wise mean cross-entropy loss on node scores

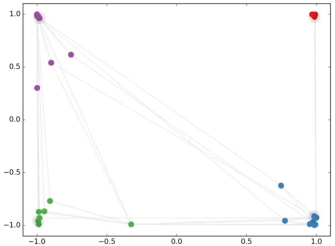
$$L = -\frac{1}{N_c}\frac{1}{N}\sum_i \sum_c s_{i,c} \times \ln(s_{i,c}), \qquad s_i = \text{softmax}(x_i)$$

- Apply storchastic gradient decent 300 times

Watch the embeddings before the classification layer evolve: video

The GNN learns to separate classes in the feature hyperspace





These are the steps of the feedforward network:

- Pass the nodes through two GCN layers
  1. Increase node features from 1 (blank) to 16
  2. Maintain 16 features in the convolution
- Pass the nodes features through a fully connected linear classification layer $(16 \to 4)$
- Apply the class-wise mean cross-entropy loss on node scores

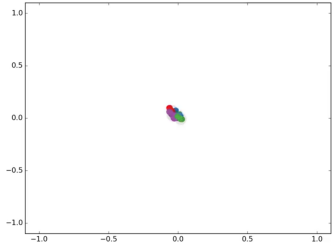
$$L = -\frac{1}{N_c}\frac{1}{N}\sum_i \sum_c s_{i,c} \times \ln(s_{i,c}), \qquad s_i = \text{softmax}(x_i)$$

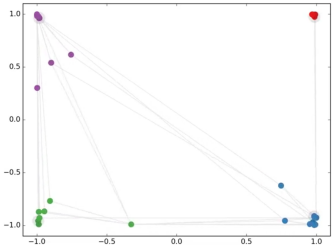
- Apply storchastic gradient decent 300 times

Watch the embeddings before the classification layer evolve: video

The GNN learns to separate classes in the feature hyperspace

Will use this technique for another task in the tutorial!

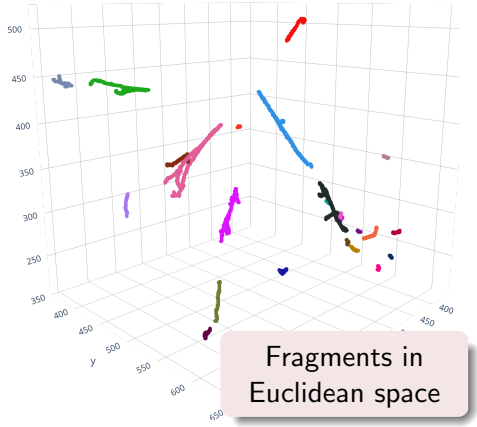




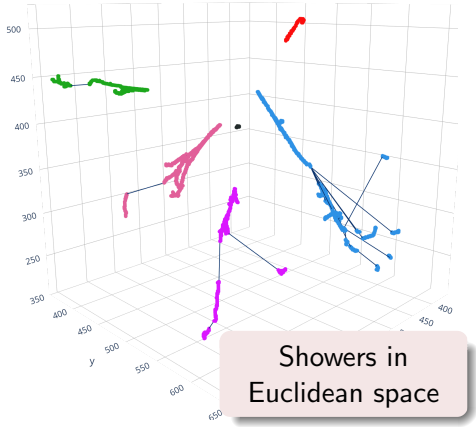
Example B: Clustering | Task

Here the goal is to cluster multiple particle traces into superstructures

- Nodes are particles, edges are connections.



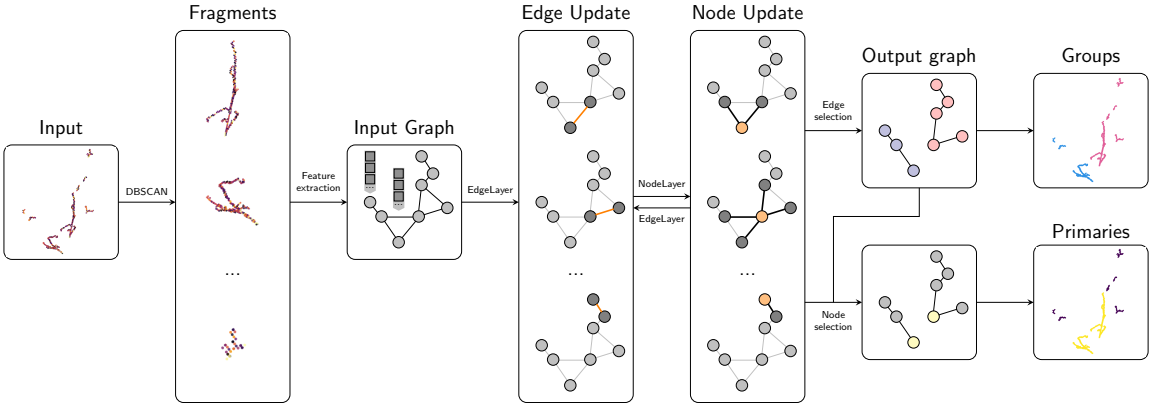
Fragments in Euclidean space



Showers in Euclidean space

Here the goal is to cluster multiple particle traces into superstructures

- Nodes are particles, edges are connections.



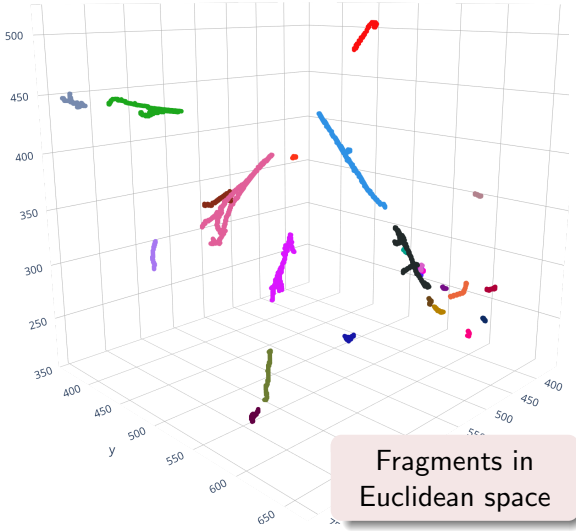
Paper: arXiv:2007.01335

Shower clustering

Network input

Input:

- Fragmented EM showers



Fragments in Euclidean space

Shower clustering

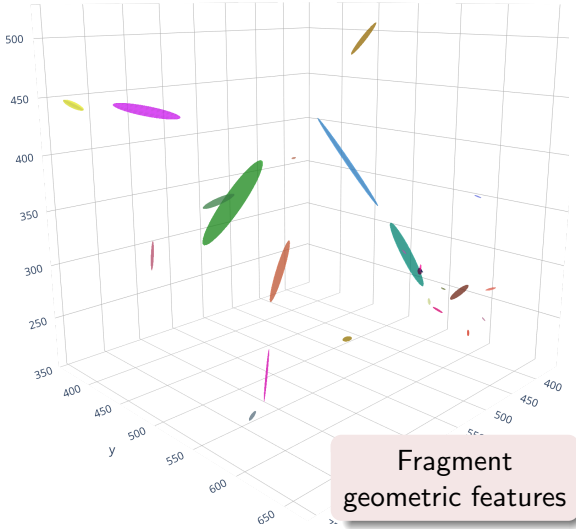
Network input

Input:

- Fragmented EM showers

Node features:

- Centroid
- Covariance matrix, PCA
- Start point, direction (PPN)



Fragment geometric features

Shower clustering

Network input

Input:

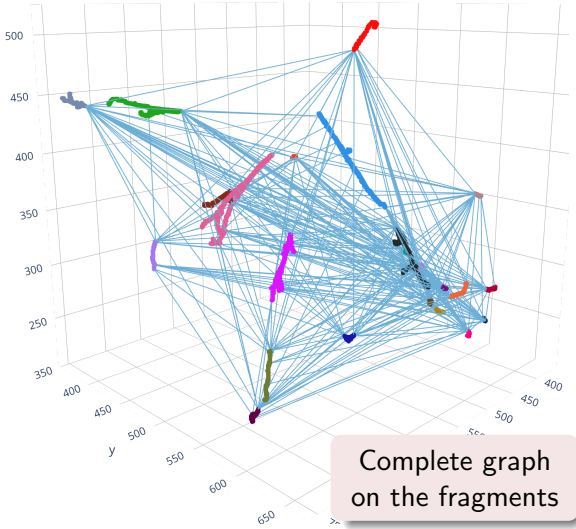
- Fragmented EM showers

Node features:

- Centroid
- Covariance matrix, PCA
- Start point, direction (PPN)

Input graph:

- Connect every node with every other node (**complete graph**)



Complete graph on the fragments

Shower clustering

Network input

Input:

- Fragmented EM showers

Node features:

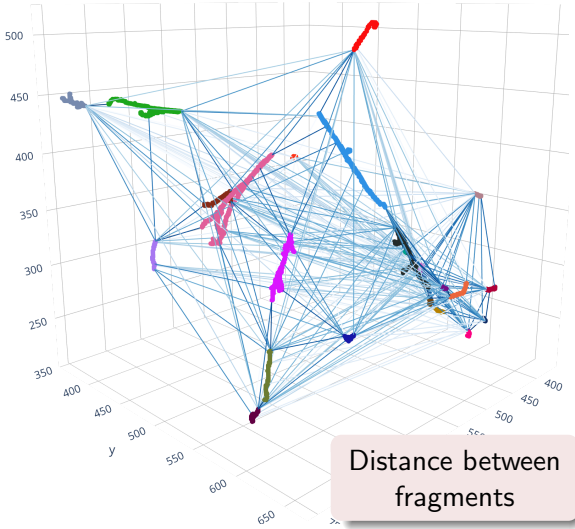
- Centroid
- Covariance matrix, PCA
- Start point, direction (PPN)

Input graph:

- Connect every node with every other node (**complete graph**)

Edge features:

- Displacement vector (+variations)
- Closest points of approach



Distance between fragments

Shower clustering

Edge classification

At each message passing step (MetaLayer: arXiv:1806.01261):

- **Edge update**

  $\boldsymbol{e}'_k = \text{MLP}(\boldsymbol{v}_{s_k}, \boldsymbol{v}_{r_k}, \boldsymbol{e}_k)$

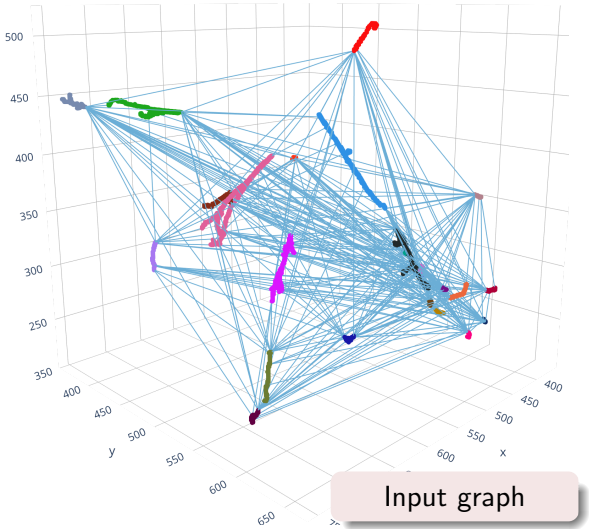
- **Node update**

  $\boldsymbol{m}_{ji} = \text{MLP}(\boldsymbol{v}_j, \boldsymbol{e}'_{\{k|s_k=j,r_k=i\}})$

  $\boldsymbol{v}'_i = \text{MLP}(\boldsymbol{v}_i, \square_{i\in\mathcal{N}(i)}\boldsymbol{m}_{ji})$

After $n = 3$ node+edge updates:

- Edge binary classification



Input graph

Shower clustering

Edge classification

At each message passing step (MetaLayer: arXiv:1806.01261):

- **Edge update**

  $\boldsymbol{e}'_k = \text{MLP}(\boldsymbol{v}_{s_k}, \boldsymbol{v}_{r_k}, \boldsymbol{e}_k)$

- **Node update**

  $\boldsymbol{m}_{ji} = \text{MLP}(\boldsymbol{v}_j, \boldsymbol{e}'_{\{k|s_k=j,r_k=i\}})$

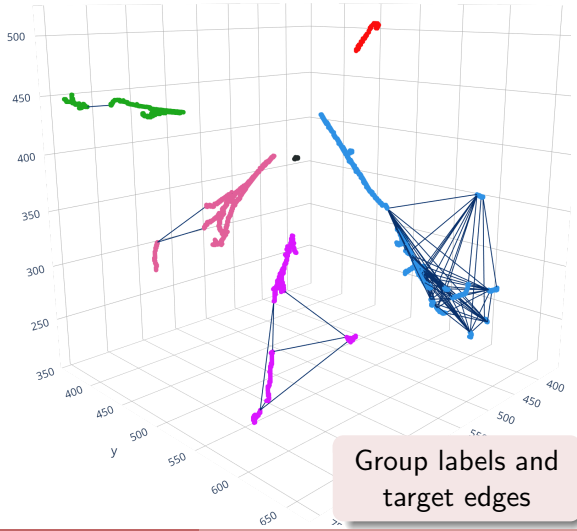
  $\boldsymbol{v}'_i = \text{MLP}(\boldsymbol{v}_i, \square_{i\in\mathcal{N}(i)}\boldsymbol{m}_{ji})$

After $n = 3$ node+edge updates:

- Edge binary classification

Target:

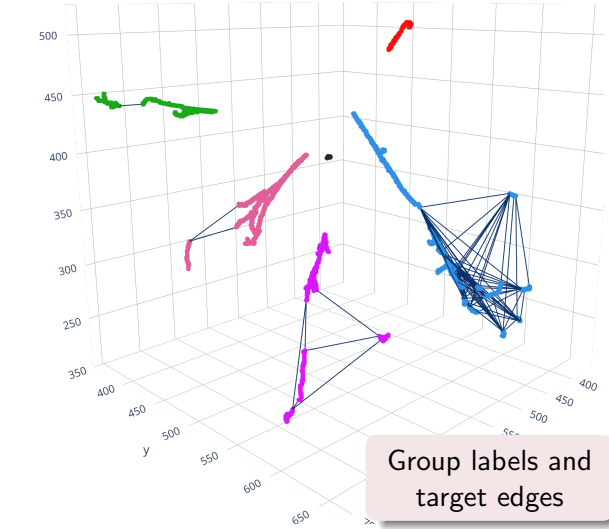
- Predict adjacency matrix $A_{ij} = \delta_{g_i,g_j}$ with $\boldsymbol{g}$ the true partition of the set
- Apply cross-entropy loss



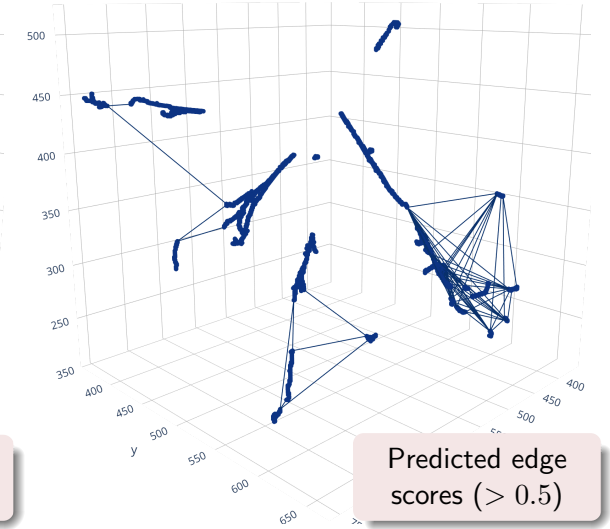
Group labels and target edges

Shower clustering

Example



Group labels and target edges

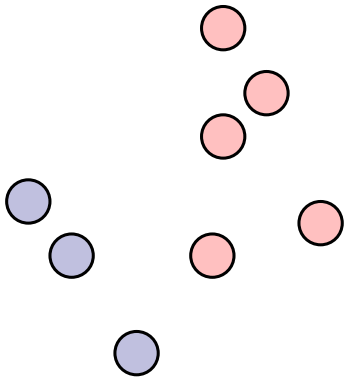


Predicted edge scores (> 0.5)

The network predicts a score matrix S, estimate of the true adjacency matrix A

- How to recover a set partition $\hat{g}$?

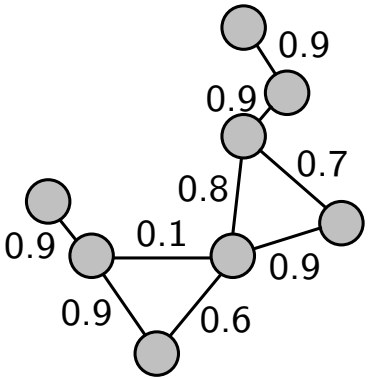
True partition



The network predicts a score matrix S, estimate of the true adjacency matrix A

- How to recover a set partition $\hat{g}$?

Edge scores



NATIONAL ACCELERATOR LABORATORY

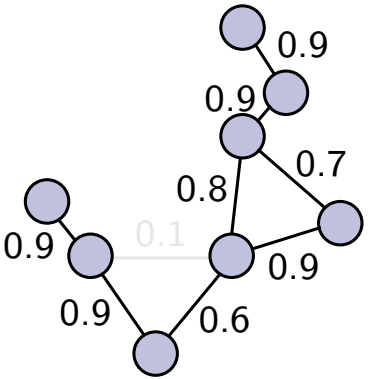
The network predicts a score matrix $\boldsymbol{S}$, estimate of the true adjacency matrix $\boldsymbol{A}$

- How to recover a set partition $\hat{\boldsymbol{g}}$?

We want the partition $\hat{\boldsymbol{g}}$ that minimizes the CE loss, given $\hat{A}_{ij} = \delta_{\hat{g}_i, \hat{g}_j}$, e.g.

$$L(S|g) = -\frac{1}{N_e} \sum_{(i,j)\in E} \delta_{g_i,g_j} \ln(s_{ij}) + (1 - \delta_{g_i,g_j}) \ln(1 - s_{ij})$$

Thresholded graph



$$L \simeq 3.92$$

The network predicts a score matrix $\boldsymbol{S}$, estimate of the true adjacency matrix $\boldsymbol{A}$

- How to recover a set partition $\hat{\boldsymbol{g}}$?

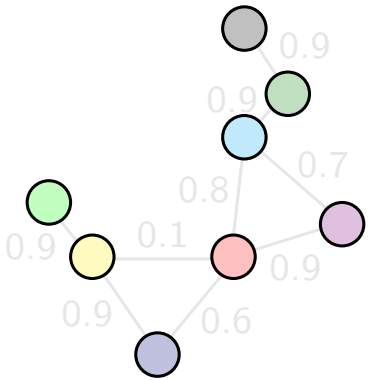
We want the partition $\hat{\boldsymbol{g}}$ that minimizes the CE loss, given $\hat{A}_{ij} = \delta_{\hat{g}_i, \hat{g}_j}$, e.g.

$$L(S|g) = -\frac{1}{N_e} \sum_{(i,j)\in E} \delta_{g_i,g_j} \ln(s_{ij}) + (1 - \delta_{g_i,g_j}) \ln(1 - s_{ij})$$

G is the set of all possible partitions

- Bell number, huge ($B_{20} \simeq 5 \times 10^{13}$), cannot brute force, how to optimize?
- Start with an empty graph

Empty graph



$L \simeq 15.35$

SLAC NATIONAL ACCELERATOR LABORATORY

The network predicts a score matrix $\boldsymbol{S}$, estimate of the true adjacency matrix $\boldsymbol{A}$

- How to recover a set partition $\hat{\boldsymbol{g}}$?

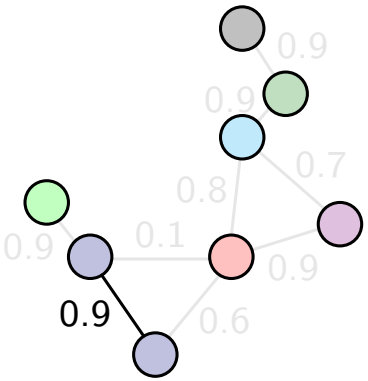
We want the partition $\hat{\boldsymbol{g}}$ that minimizes the CE loss, given $\hat{A}_{ij} = \delta_{\hat{g}_i, \hat{g}_j}$, e.g.

$$L(S|g) = -\frac{1}{N_e} \sum_{(i,j)\in E} \delta_{g_i,g_j} \ln(s_{ij}) + (1 - \delta_{g_i,g_j}) \ln(1 - s_{ij})$$

G is the set of all possible partitions

- Bell number, huge ($B_{20} \simeq 5 \times 10^{13}$), cannot brute force, how to optimize?
- Start with an empty graph
- Iteratively attempt to add most likely edges until edge score < 0.5

First edge



$L \simeq 13.15$

SLAC NATIONAL ACCELERATOR LABORATORY

The network predicts a score matrix $\boldsymbol{S}$, estimate of the true adjacency matrix $\boldsymbol{A}$

- How to recover a set partition $\hat{\boldsymbol{g}}$?

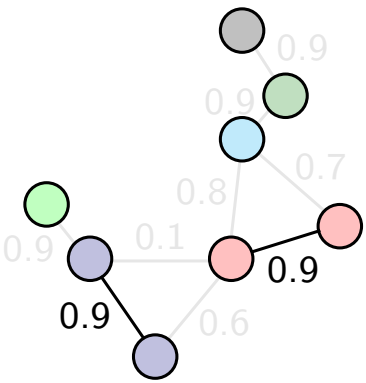
We want the partition $\hat{\boldsymbol{g}}$ that minimizes the CE loss, given $\hat{A}_{ij} = \delta_{\hat{g}_i, \hat{g}_j}$, e.g.

$$L(S|g) = -\frac{1}{N_e} \sum_{(i,j)\in E} \delta_{g_i,g_j} \ln(s_{ij}) + (1 - \delta_{g_i,g_j}) \ln(1 - s_{ij})$$

G is the set of all possible partitions

- Bell number, huge ($B_{20} \simeq 5 \times 10^{13}$), cannot brute force, how to optimize?
- Start with an empty graph
- Iteratively attempt to add most likely edges until edge score < 0.5

Second edge



$L \simeq 10.95$

The network predicts a score matrix $\boldsymbol{S}$, estimate of the true adjacency matrix $\boldsymbol{A}$

- How to recover a set partition $\hat{\boldsymbol{g}}$?

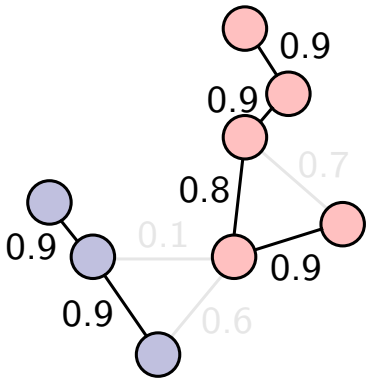
We want the partition $\hat{\boldsymbol{g}}$ that minimizes the CE loss, given $\hat{A}_{ij} = \delta_{\hat{g}_i, \hat{g}_j}$, e.g.

$$L(S|g) = -\frac{1}{N_e} \sum_{(i,j)\in E} \delta_{g_i,g_j} \ln(s_{ij}) + (1 - \delta_{g_i,g_j}) \ln(1 - s_{ij})$$

G is the set of all possible partitions

- Bell number, huge ($B_{20} \simeq 5 \times 10^{13}$), cannot brute force, how to optimize?
- Start with an empty graph
- Iteratively attempt to add most likely edges until edge score < 0.5

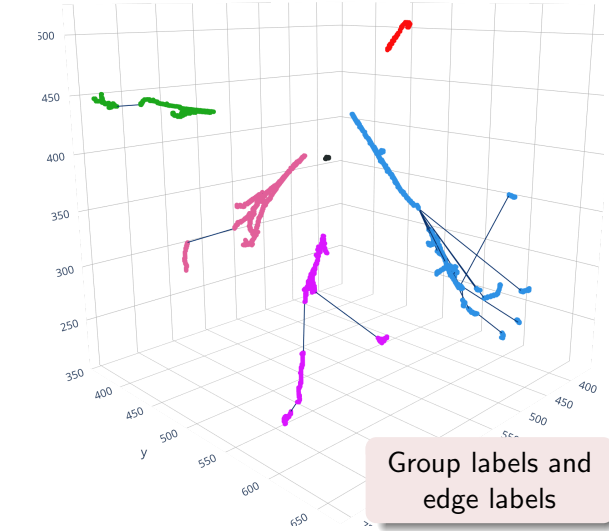
Optimized partition



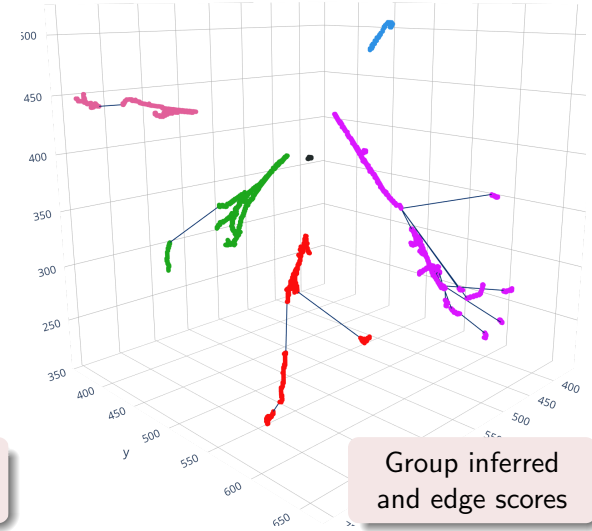
$$L \simeq 2.13$$

Shower clustering

Performance



Group labels and edge labels



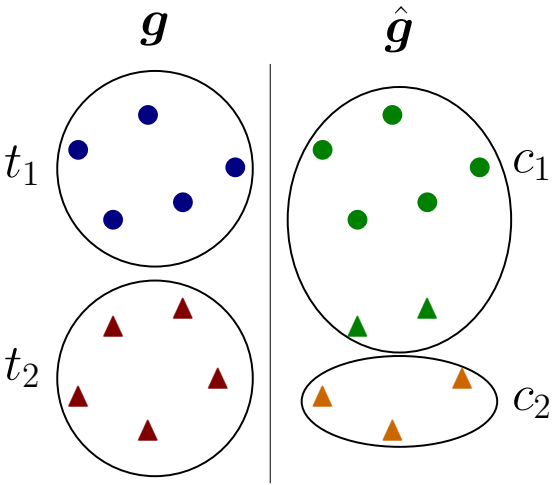
Group inferred and edge scores

Purity $= \frac{1}{N}\sum_{i=1}^{n_p} \max_j |c_i \cap t_j|$

- c_i predicted cluster
- t_j true cluster with highest count in c_i

Efficiency $= \frac{1}{N}\sum_{i=0}^{n_t} \max_j |c_j \cap t_i|$

- c_j pred. cluster with highest count in t_i
- t_i true cluster



SLAC NATIONAL ACCELERATOR LABORATORY

Purity $= \frac{1}{N}\sum_{i=1}^{n_p} \max_j |c_i \cap t_j|$

- c_i predicted cluster
- t_j true cluster with highest count in c_i

Efficiency $= \frac{1}{N}\sum_{i=0}^{n_t} \max_j |c_j \cap t_i|$

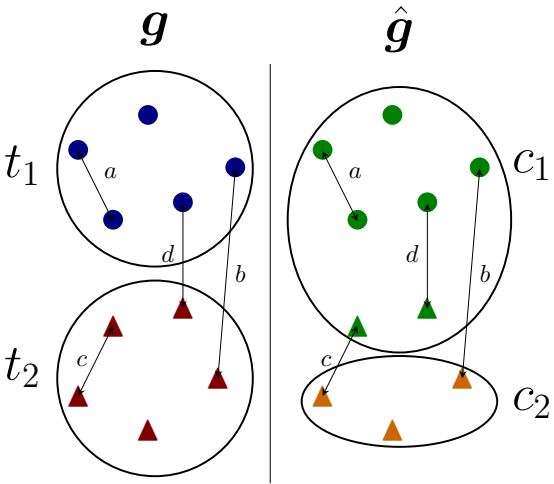
- c_j pred. cluster with highest count in t_i
- t_i true cluster

Adjusted Rand Index (ARI)

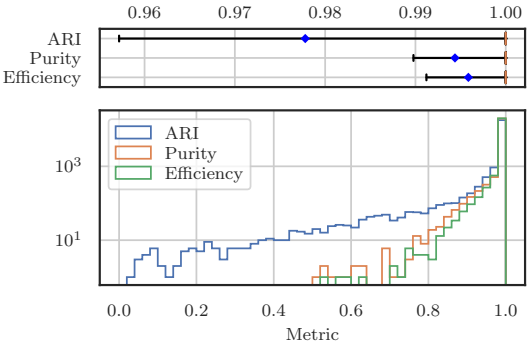
- Measure of overlap of prediction and truth, adjusted for random chance

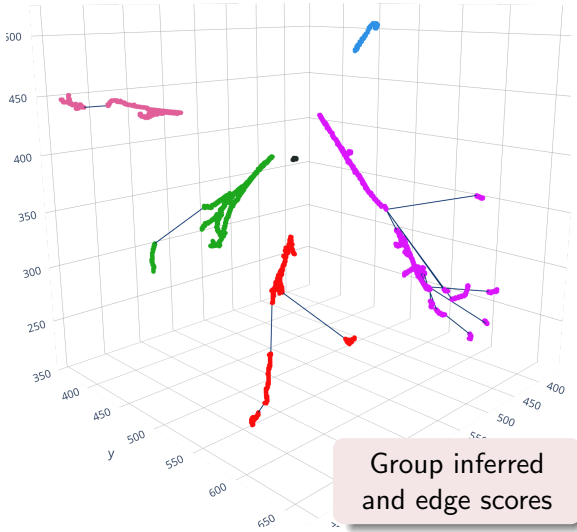
$\text{RI} = \frac{a+b}{a+b+c+d}$

$\text{ARI} = \frac{RI - E(RI)}{1 - E(RI)}$



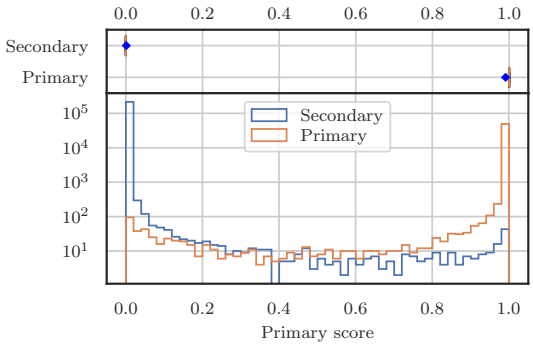
Shower clustering accuracy:

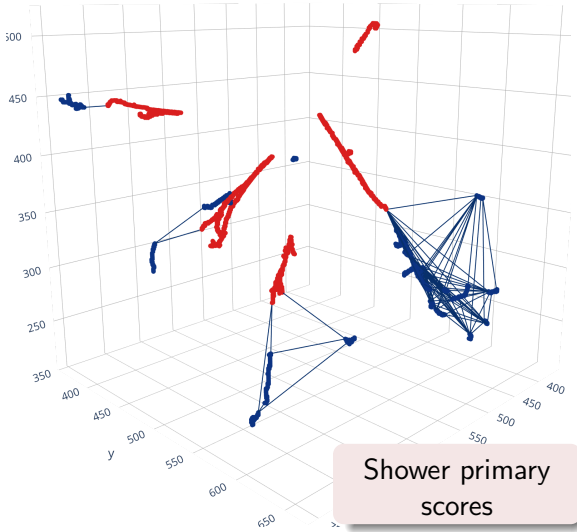




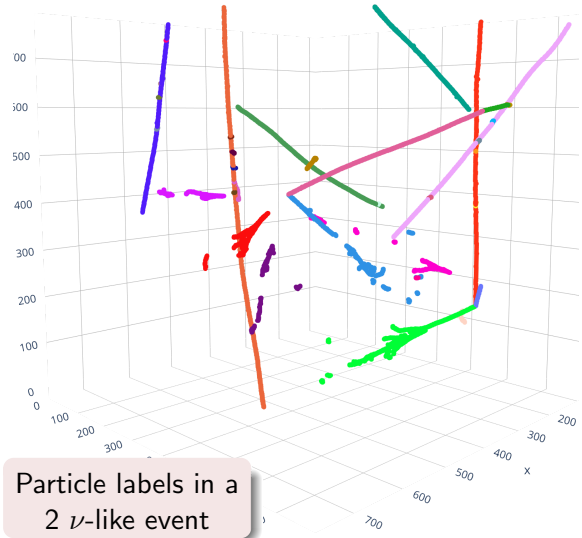
Group inferred and edge scores

Shower primary accuracy (**99.8 %**):





Shower primary scores



Particle labels in a 2 ν-like event

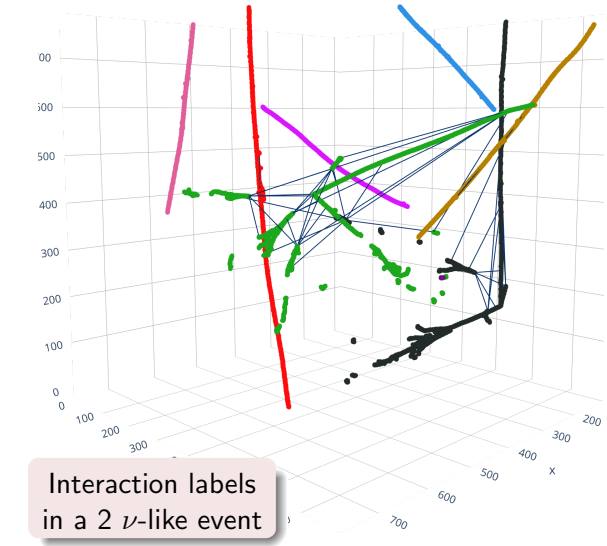
The shower clustering task can be extended to **interaction clustering**:

- Interaction group = particles that originate from the same vertex
- Fragment ID $\rightarrow$ Particle ID
- Particle ID $\rightarrow$ Interaction ID

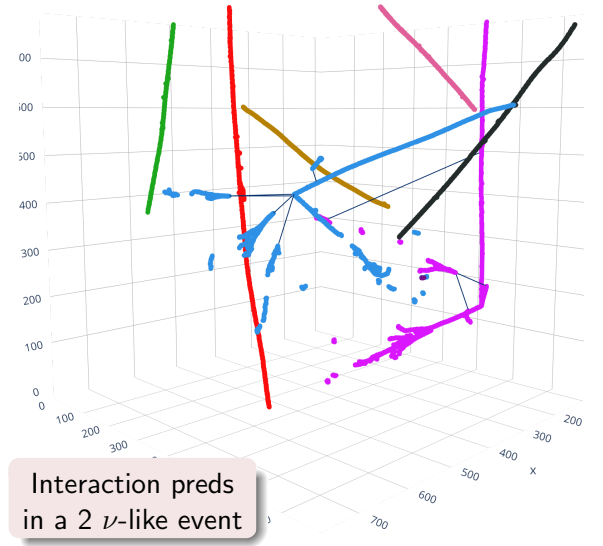
Useful to:

- Separate signal from background
- Resolve pileup

Interaction Clustering | Performance

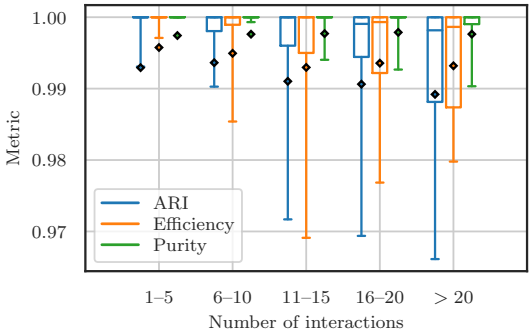


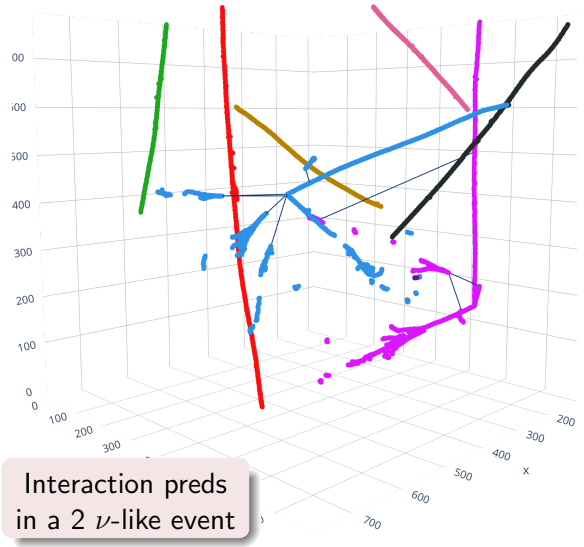
Interaction labels in a 2 ν-like event



Interaction preds in a 2 ν-like event

Interaction clustering performance:

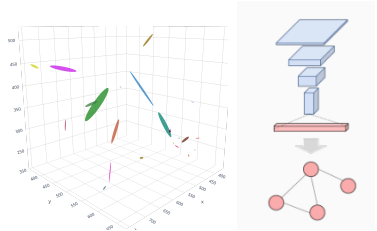


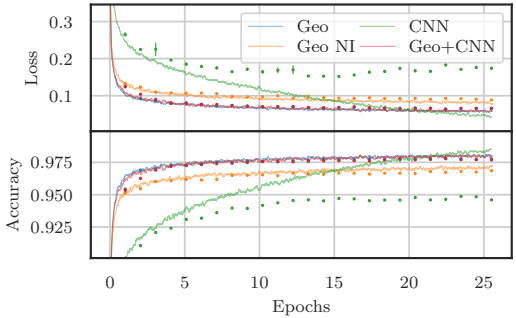


Interaction preds in a 2 ν-like event

A lot of work in optimizing the different components of the GNN chain:

- Feature extractor
  - Geometric (w/ or w/o PPN)
  - CNN



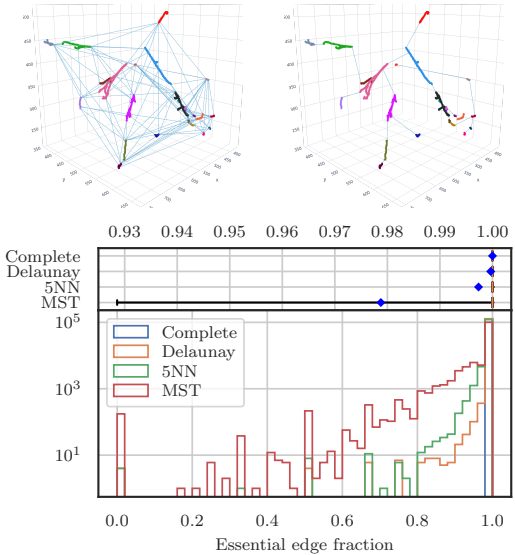


GNN Clustering

Optimization studies

A lot of work in optimizing the different components of the GNN chain:

- Feature extractor
  - Geometric (w/ or w/o PPN)
  - CNN
- Receptive field (graph)
  - Complete, Delaunay, MST, kNN

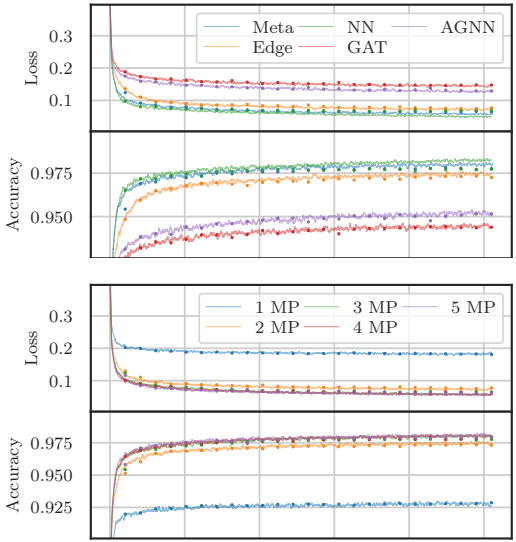


GNN Clustering

Optimization studies

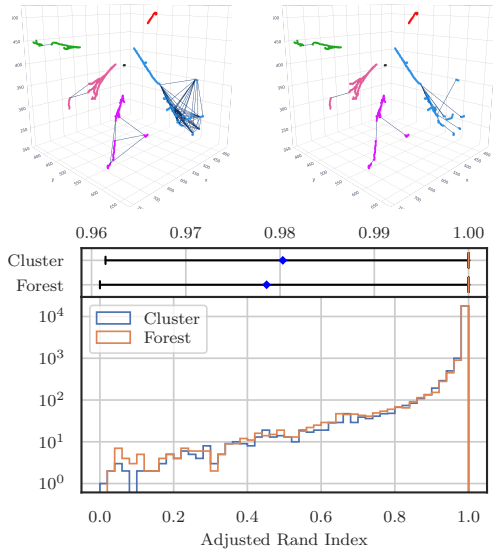
A lot of work in optimizing the different components of the GNN chain:

- Feature extractor
  - Geometric (w/ or w/o PPN)
  - CNN
- Receptive field (graph)
  - Complete, Delaunay, MST, kNN
- Architecture
  - Node updater
  - Number of message passings



A lot of work in optimizing the different components of the GNN chain:

- Feature extractor
  - Geometric (w/ or w/o PPN)
  - CNN
- Receptive field (graph)
  - Complete, Delaunay, MST, kNN
- Architecture
  - Node updater
  - Number of message passings
- Target adjacency matrix
  - Cluster graph
  - Forest

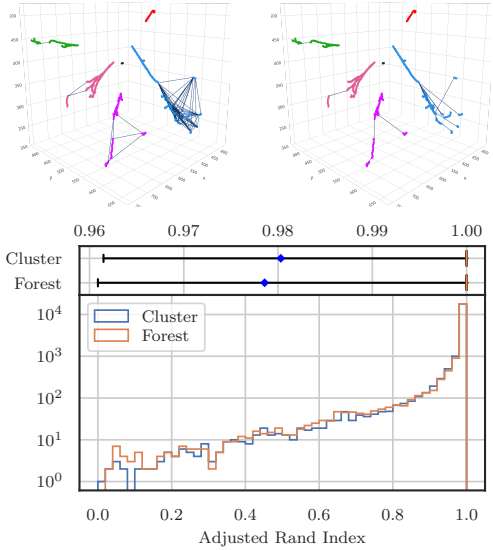


GNN Clustering

Optimization studies

A lot of work in optimizing the different components of the GNN chain:

- Feature extractor
  - Geometric (w/ or w/o PPN)
  - CNN
- Receptive field (graph)
  - Complete, Delaunay, MST, kNN
- Architecture
  - Node updater
  - Number of message passings
- Target adjacency matrix
  - Cluster graph
  - Forest



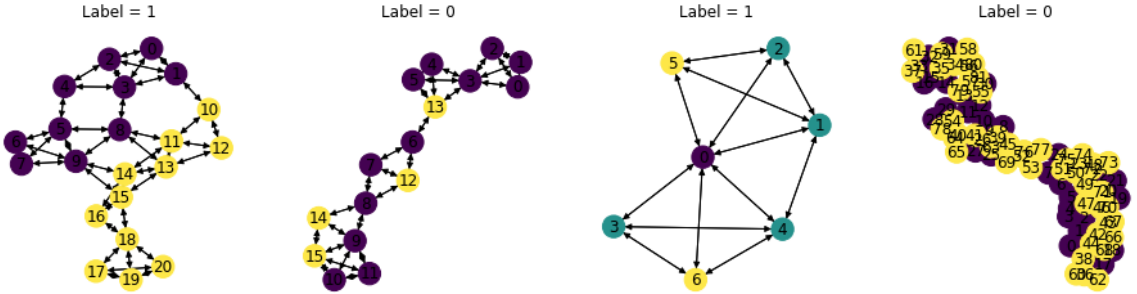
See paper: arXiv:2007.01335

Example C: Proteins | Task

Dataset: PROTEINS is a dataset of graph-encoded proteins

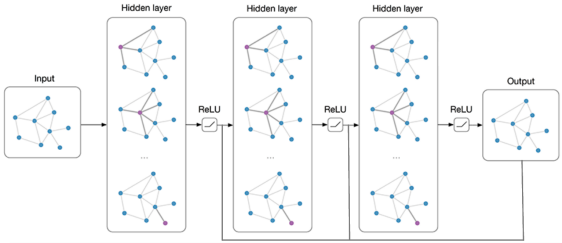
- There are 1113 graphs, one per protein
- The nodes are secondary structure elements (essentially turns, twists, folds, etc.)
- Edges connect adjacent structures (either in the chain, or in space)

Goal: Predict whether the protein is a enzyme, or not



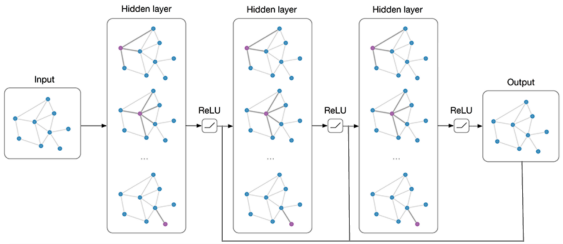
For this example, there are two main ingredients:

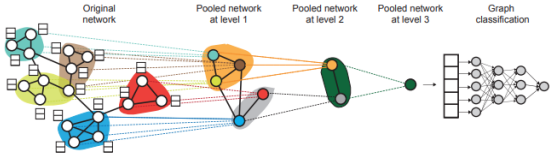
- A 3-layer message passing function: **SAGEConv**
  - $x_i^{(l+1)} = W_1^{(l)} x_i^{(l)} + W_2^{(l)} \cdot \text{mean}_{j \in \mathcal{N}(i)} x_j^{(l)}$
- A linear layer combining the 3 node embeddings into a set of N_v' features
  - $x_i' = W'(x_i^{(1)}, x_i^{(2)}, x_i^{(3)})$



For this example, there are two main ingredients:

- A 3-layer message passing function: **SAGEConv**
  - $x_i^{(l+1)} = W_1^{(l)} x_i^{(l)} + W_2^{(l)} \cdot \text{mean}_{j \in \mathcal{N}(i)} x_j^{(l)}$
- A linear layer combining the 3 node embeddings into a set of N_v' features
  - $x_i' = W'(x_i^{(1)}, x_i^{(2)}, x_i^{(3)})$
- Two-layer graph pooling operation: **Diffpool**
  - First, reduce the number of nodes to N/4, then to N/16
- Aggregate the remaining node features to a 1D array
- Reduce to two features (one score for enzyme=NO, one for YES)





Given the 2-channel output of the network:

- The binary cross entropy loss is computed for each score pair:
  $-\frac{1}{N}(s_0 \ln(s_0) + s_1 \ln(s_1)), \qquad (s_0, s_1) = \mathsf{softmax}(x'_0, x'_1)$
- Update the model weights
- Repeat this for 10 epochs
  $\rightarrow$ Let's move to the notebook!

Conclusions

1. What are Graph Neural Networks (GNNs) good for ?
   - **Generalize** the concept of receptive field in CNNs and succession in RNNs to any **structured data**
   - Infer about objects in an ensemble, but also about **their correlations and superstructures**
2. How does information flow in a graph ?
   - GNNs are Neural Networks with a shared neighborhood aggregation rule: **message passing**
   - The adjacency matrix determines **information flow**
   - The message passing rule is extremely **flexible**... as long as it is **learnable**!

Thank you for your attention!