Graph Neural Networks

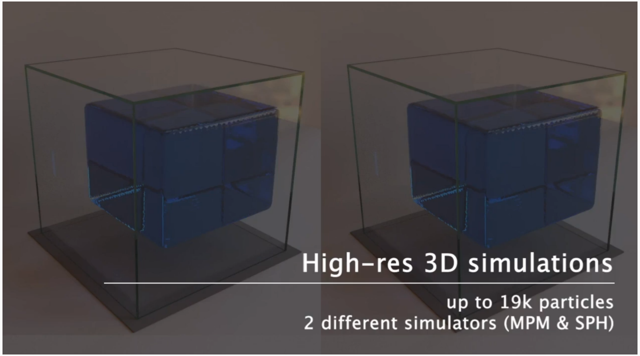
From fundamentals to physics application

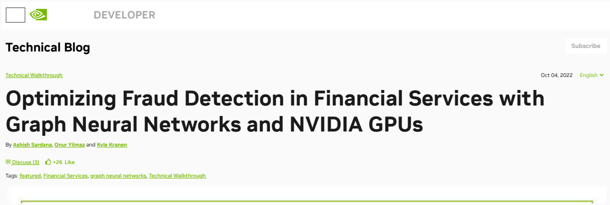
Ilias Tsaklidis itsaklid@uni-bonn.de

07/03/2023

inverted CERN School of Computing

What is all about Graph Neural Networks ?




DEVELOPER

Technical Blog

Subscribe

Technical Walkthrough

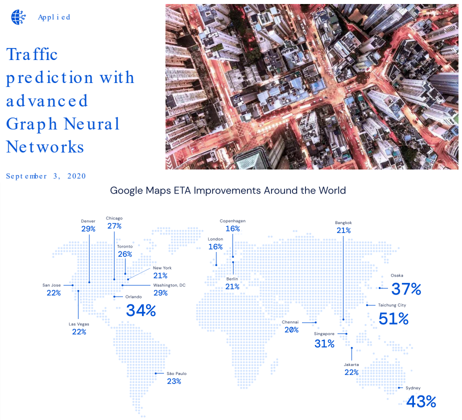
Oct 04, 2022 English

Optimizing Fraud Detection in Financial Services with Graph Neural Networks and NVIDIA GPUs

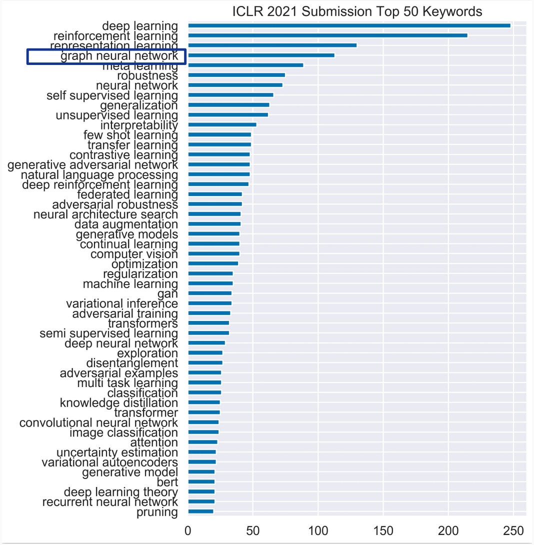
By Ashish Sardana, Onur Yilmaz and Kyle Kranen

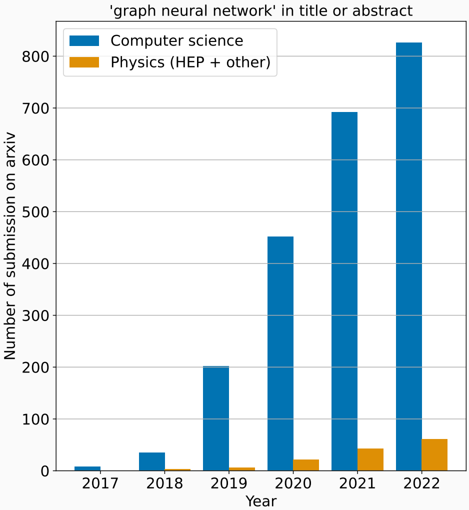
Discuss (3) +26 Like

Tags: featured, Financial Services, graph neural networks, Technical Walkthrough



A hot research topic





What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

Mainly discussing the core ideas

What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

Mainly discussing the core ideas

After this lecture you will hopefully have a clear idea:

What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

Mainly discussing the core ideas

After this lecture you will hopefully have a clear idea:

1. Why GNNs are a powerful tool

What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

Mainly discussing the core ideas

After this lecture you will hopefully have a clear idea:

1. Why GNNs are a powerful tool
2. How to build a graph

What this lecture is about

Aiming at the particle physicist who uses GNNs from an engineering point of view

Mainly discussing the core ideas

After this lecture you will hopefully have a clear idea:

1. Why GNNs are a powerful tool
2. How to build a graph
3. How to choose an appropriate GNN for your problem

Outline

1. Data structures and relational inductive biases

Outline

1. Data structures and relational inductive biases

2. Elements of Graph Theory

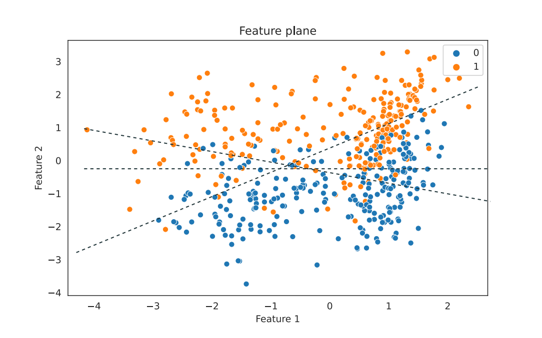
Outline

1. Data structures and relational inductive biases
2. Elements of Graph Theory
3. Graph Neural Mechanisms

Outline

1. Data structures and relational inductive biases
2. Elements of Graph Theory
3. Graph Neural Mechanisms
4. Applications in HEP

A general recipe for supervised machine learning



Machine Learning

$\vec{x}, y \Longrightarrow \hat{f}(\vec{x}; \hat{\theta})$

Initialize parameters $\hat{\theta}$ → Repeat N times: [Model probability → Compute loss → Update parameters] → $\hat{f}(x; \theta)$

e.g. sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

e.g. MSE

$$L = \sum_{i=1}^{D} |\sigma(z)_i - y_i|$$

e.g. gradient descend

$$\hat{\theta}' = \hat{\theta} - \frac{\partial L}{\partial \hat{\theta}}$$

Combinatorial generalization

A key signature of human intelligence is the ability to make "infinite use of finite means" (Chomsky N.)

Combinatorial generalization

A key signature of human intelligence is the ability to make "infinite use of finite means" (Chomsky N.)

Universal approximation theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any continuous function to arbitrary precision with a finite number of nodes.

Combinatorial generalization

A key signature of human intelligence is the ability to make "infinite use of finite means" (Chomsky N.)

Universal approximation theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any continuous function to arbitrary precision with a finite number of nodes.

1. Good: A neural network can solve any problem.

Combinatorial generalization

A key signature of human intelligence is the ability to make "infinite use of finite means" (Chomsky N.)

Universal approximation theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any continuous function to arbitrary precision with a finite number of nodes.

1. Good: A neural network can solve any problem.
2. Bad: Does not specify the number of nodes.

Combinatorial generalization

A key signature of human intelligence is the ability to make "infinite use of finite means" (Chomsky N.)

Universal approximation theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any continuous function to arbitrary precision with a finite number of nodes.

1. Good: A neural network can solve any problem.
2. Bad: Does not specify the number of nodes.

Combinatorial generalization requires enormous computational power

Inductive bias

Inductive bias

a set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Inductive bias

a set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Example 1

Occam's Razor expresses a preference for simplicity

Inductive bias

Inductive bias

a set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Example 1

Occam's Razor expresses a preference for simplicity

Example 2

A Bayesian model expresses inductive bias through the choice and parameterization of the prior distribution

Inductive bias

Inductive bias

a set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered.

Example 1

Occam's Razor expresses a preference for simplicity

Example 2

A Bayesian model expresses inductive bias through the choice and parameterization of the prior distribution

Relational inductive bias may be enforced by the choice of data structure

Relational reasoning

Some profound definitions

Relational reasoning

Some profound definitions

entity
an element with attributes

Relational reasoning

Some profound definitions

entity

an element with attributes

relation

a property between entities

Relational reasoning

Some profound definitions

entity

an element with attributes

relation

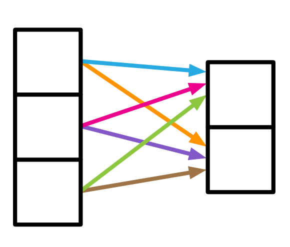
a property between entities

rule

a function that maps entities and relations to other entities and relations. e.g. is entity X heavier than entity Y?

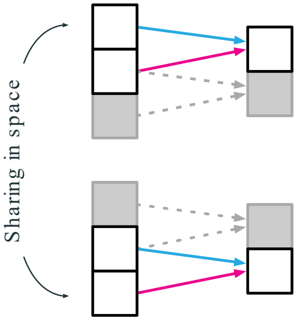
Relational inductive bias in CNNs

Fully connected



Entities: Nodes
Relations: All-to-all
Relational inductive bias: weak
Invariance: -

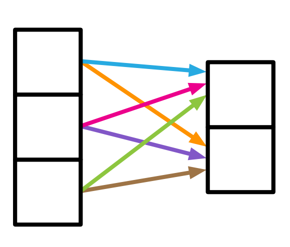
Convolutional



Entities: Grid elements
Relations: Local
Relational inductive bias: Locality
Invariance: Spatial translation

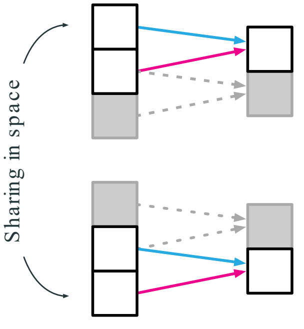
Relational inductive bias in CNNs

Fully connected



Entities: Nodes
Relations: All-to-all
Relational inductive bias: weak
Invariance: -

Convolutional

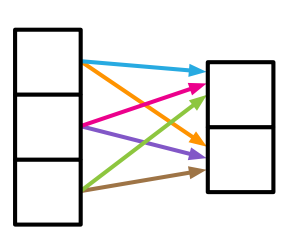


Entities: Grid elements
Relations: Local
Relational inductive bias: Locality
Invariance: Spatial translation

Locality: the arguments to the relational rule are entities in close proximity.

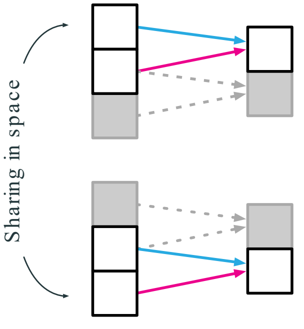
Relational inductive bias in CNNs

Fully connected



Entities: Nodes
Relations: All-to-all
Relational inductive bias: weak
Invariance: -

Convolutional



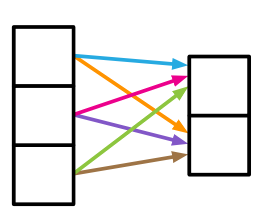
Entities: Grid elements
Relations: Local
Relational inductive bias: Locality
Invariance: Spatial translation

Locality: the arguments to the relational rule are entities in close proximity.

Translation invariance: reusage of the same rule across localities in the input.

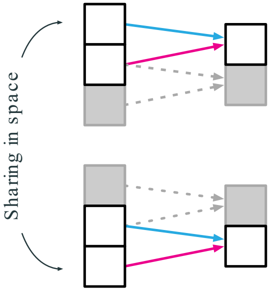
Relational inductive bias in CNNs

Fully connected

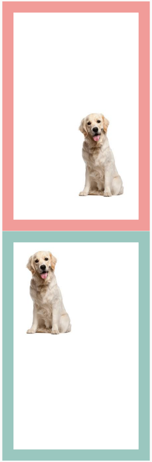


Entities: Nodes
Relations: All-to-all
Relational inductive bias: weak
Invariance: -

Convolutional



Entities: Grid elements
Relations: Local
Relational inductive bias: Locality
Invariance: Spatial translation



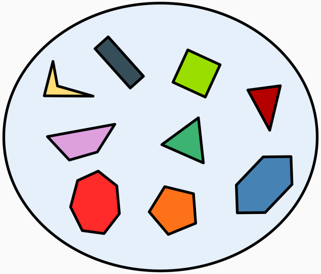
Locality: the arguments to the relational rule are entities in close proximity.

Translation invariance: reusage of the same rule across localities in the input.

Relational inductive bias of unordered entities

Set

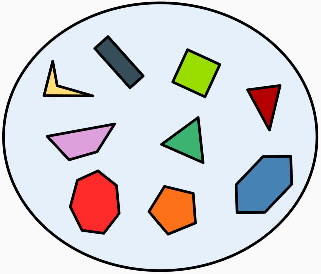
Entities whose order is irrelevant.



Relational inductive bias of unorderded entities

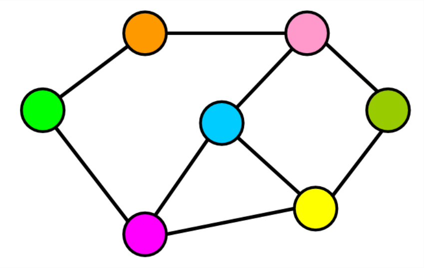
Set

Entities whose order is irrelevant.



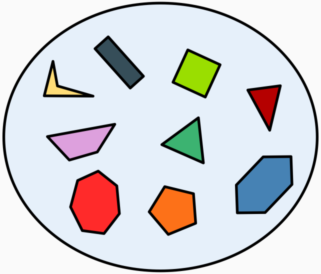
Graph

A set with pair-wise relations



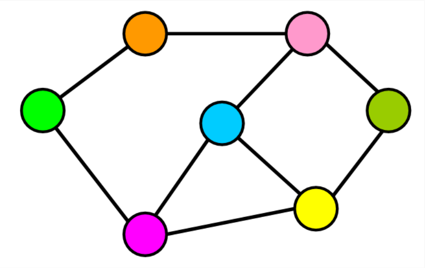
Set

Entities whose order is irrelevant.



Graph

A set with pair-wise relations



A relational inductive bias arises from the absence of canonical order

Exploit it by allowing predictions to depend on symmetric functions

Symmetries of the function

Permutation equivariance

The output of the function is permuted in the same way as the input.

Permutation invariance

The output of the function is the same independantly of the permutation of the input.

Equivariance

$$f(x_i, x_j) = (y_i, y_j)$$

$$f(x_j, x_i) = (y_j, y_i)$$

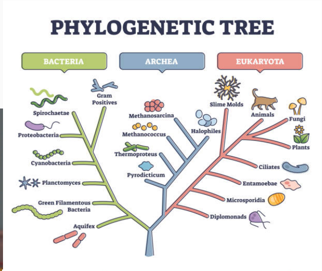
Invariance

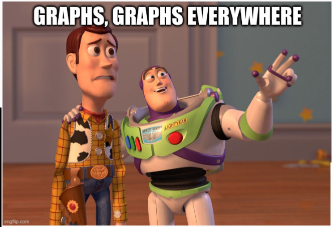
$$f(x_i, x_j) = y_k$$

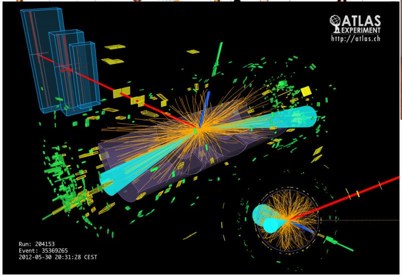
$$f(x_j, x_i) = y_k$$

Social Networks
Complex pairwise connections

Phylogenetic trees
Trees are a particular type of graphs
(directed and acyclic graphs)

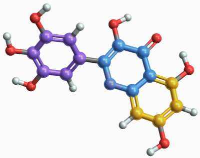


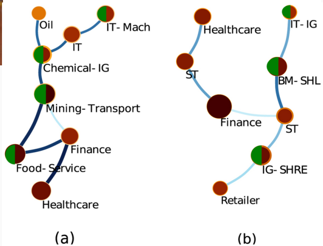




Energy deposits in a detetor
Any complex set of elements can be represented as a graph.
Constructing the graph depends on several factors.
More on this will follow...

Molecules and their dynamics
naturally represented as graphs





Business and financing
Complex inter-dependencies between entities

Take home messages

1. Relational inductive biases can improve a learning algorithm.

Take home messages

1. Relational inductive biases can improve a learning algorithm.

2. The relational inductive bias in graphs is the absence of canonical order of the entities.

Take home messages

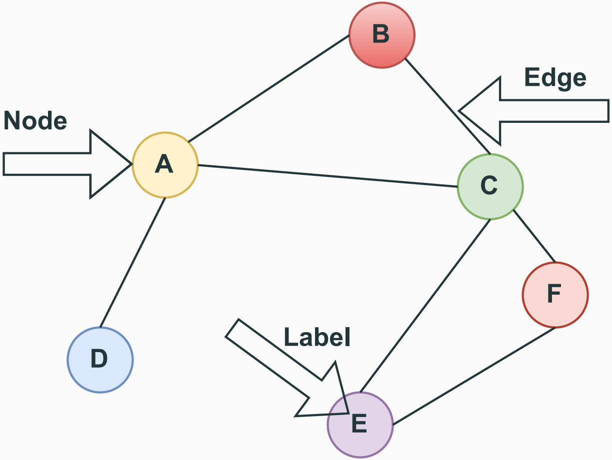
1. Relational inductive biases can improve a learning algorithm.
2. The relational inductive bias in graphs is the absence of canonical order of the entities.
3. This relational inductive bias manifests itself as permutation invariance and permutation equivariance.

What is a graph ?

Graph (Computer Science)

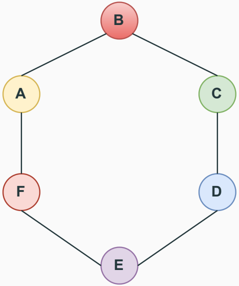
A non-linear data structure consisting of a set of elements and their relations.

G = (u, V, E)

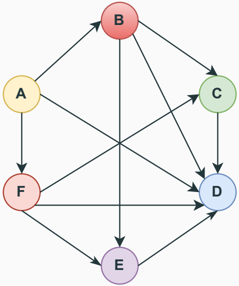


A non exhaustive graph taxonomy

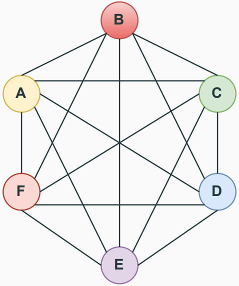
Some typical graph types you may encounter



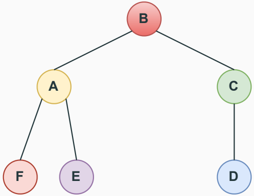
Regular Graph
All nodes have the same degree



Directed Graph
The edged have a direction



Fully Connected Graph
All nodes are interconnected



Acyclic Graph
No cyclic paths in the graph

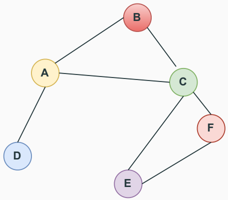
How to represent a graph

Adjacency matrix

A square matrix whose elements indicate whether pairs of nodes are adjacent or not in the graph.

Feature matrix

A matrix with individual measurable properties or characteristics of a phenomenon.



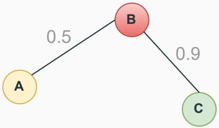
	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	0	0	0
C	1	1	0	0	1	1
D	1	0	0	0	0	0
E	0	0	1	0	0	1
F	0	0	1	0	1	0

Adjacency matrix (N x N)

	F_x	F_y	F_z	F_w
A	f_{Ax}	f_{Ay}	f_{Az}	f_{Aw}
B	f_{Bx}	f_{By}	f_{Bz}	f_{Bw}
C	f_{Cx}	f_{Cy}	f_{Cz}	f_{Cw}
D	f_{Dx}	f_{Dy}	f_{Dz}	f_{Dw}
E	f_{Ex}	f_{Ey}	f_{Ez}	f_{Ew}
F	f_{Fx}	f_{Fy}	f_{Fz}	f_{Fw}

Feature matrix (N x F)

Other graph representations



Adjacency list

A ----> {B}
B ---> {A, C}
C ----> {B}

Weighted matrix

	A	B	C
A	0	0.5	0
B	0.5	0	0.9
C	0	0.9	0

Degree matrix D

	A	B	C
A	1	0	0
B	0	2	0
C	0	0	1

Laplacian matrix L = D - A

	A	B	C
A	1	-1	-1
B	-1	2	-1
C	-1	-1	1

Coordinate List (COO)

A	B
B	A
B	C
C	B

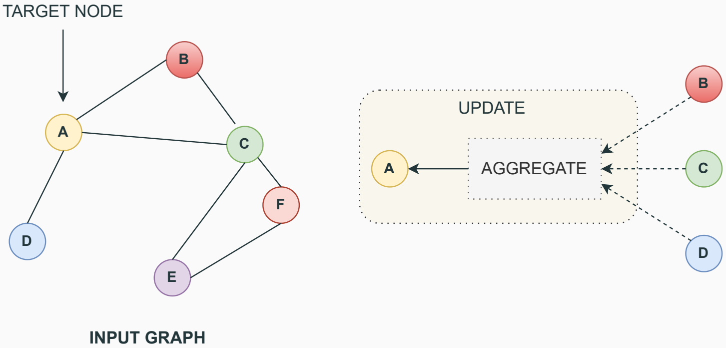
Take home messages

1. Entities = nodes; Relations = edges

Take home messages

1. Entities = nodes; Relations = edges
2. A graph at it simplest form can be defined by an adjacency matrix and a feature matrix.

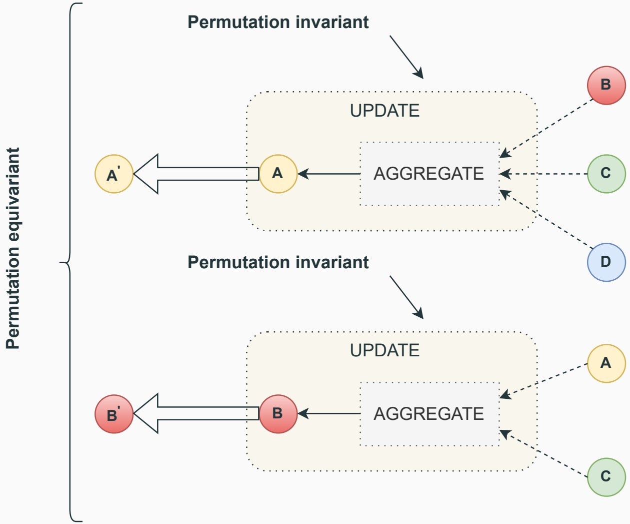
This is the core idea behind every graph neural network architecture!



Let h_u^k be the state of node u in step k

$$\mathbf{h}_{\mathrm{u}}^{\mathrm{k+1}} = \mathrm{UPDATE}^{\mathrm{k}}\left(\mathbf{h}_{\mathrm{u}}^{(\mathrm{k})}, \mathrm{AGGREGATE}^{\mathrm{k}}\left(\{\mathbf{h}_{\mathrm{v}}^{\mathrm{k}}, \forall \mathrm{v} \in \mathrm{N}(\mathrm{u})\}\right)\right) \tag{1}$$

Symmetries

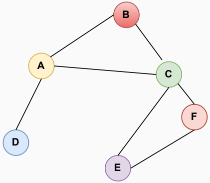


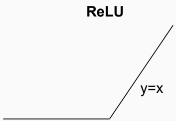
Two operations stacked together, one invariant and one equivariant.

Neural Message passing

The simplest choice is the SUM aggregator.

$$\mathbf{h}^{k+1} = \sigma(\underbrace{\mathbf{W}^{(k)}_{neigh}}_{\text{UPDATE}} \underbrace{\sum_{v \in \mathcal{N}(u)}}_{\text{AGGREGATE}} \mathbf{h}_v^k + \underbrace{\mathbf{b}^{(k)}}_{\text{Omit}})$$





Adjacency matrix

0	1	1	1	0	0
1	0	1	0	0	0
1	1	0	0	1	1
1	0	0	0	0	0
0	0	1	0	0	1
0	0	1	0	1	0

Node Embedding

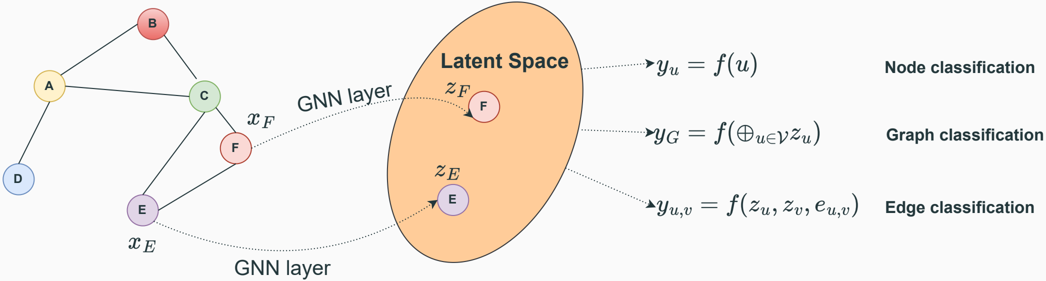
f_{Ax}	f_{Ay}	f_{Az}	f_{Aw}
f_{Bx}	f_{By}	f_{Bz}	f_{Bw}
f_{Cx}	f_{Cy}	f_{Cz}	f_{Cw}
f_{Dx}	f_{Dy}	f_{Dz}	f_{Dw}
f_{Ex}	f_{Ey}	f_{Ez}	f_{Ew}
f_{Fx}	f_{Fy}	f_{Fz}	f_{Fw}

Trainable parameters

$\omega_{1,1}$	$\omega_{1,2}$	$\omega_{1,3}$	$\omega_{1,4}$	$\omega_{1,5}$	$\omega_{1,6}$	$\omega_{1,7}$	$\omega_{1,8}$
$\omega_{2,1}$	$\omega_{2,2}$	$\omega_{2,3}$	$\omega_{2,4}$	$\omega_{2,5}$	$\omega_{2,6}$	$\omega_{2,7}$	$\omega_{2,8}$
$\omega_{3,1}$	$\omega_{3,2}$	$\omega_{3,3}$	$\omega_{3,4}$	$\omega_{3,5}$	$\omega_{3,6}$	$\omega_{3,7}$	$\omega_{3,8}$
$\omega_{4,1}$	$\omega_{4,2}$	$\omega_{4,3}$	$\omega_{4,4}$	$\omega_{4,5}$	$\omega_{4,6}$	$\omega_{4,7}$	$\omega_{4,8}$

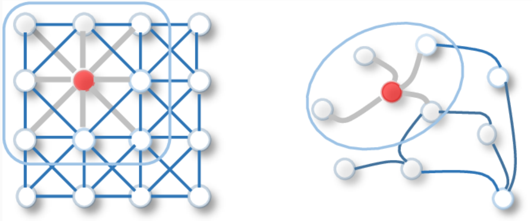
$$\mathbf{H}^{k+1} = \sigma(\mathbf{A}\mathbf{H}^k\mathbf{W}^k)$$

Node embeddings



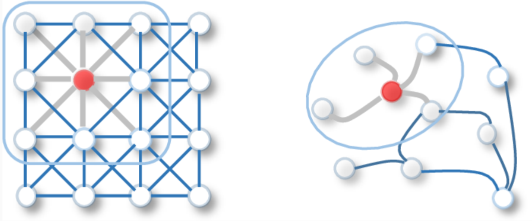
The node embeddings can be further mapped using feed forward layers.

Graph Convolution



Question

Is message passing the equivalent of convolution on graphs ?



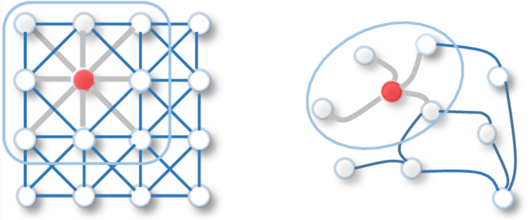
Question

Is message passing the equivalent of convolution on graphs ?

Answer

Not really strictly speaking. Graphs can be strongly heterogeneous.

Graph Convolution



Question

Is message passing the equivalent of convolution on graphs ?

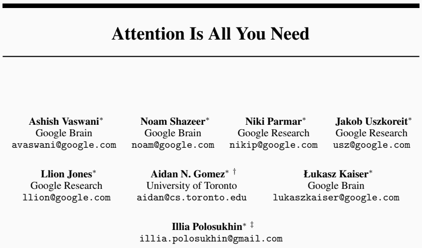
Answer

Not really strictly speaking. Graphs can be strongly heterogeneous.

Kipf and Welling added a normalization term in the aggregation function

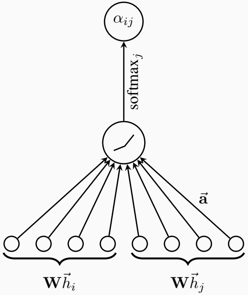
$$\mathbf{h}^{k+1} = \sigma(\mathbf{W}_{neigh}^{(k)} \sum_{v \in \mathcal{N}(u)} \frac{\mathbf{h}_v^k}{\sqrt{|\mathcal{N}_u||\mathcal{N}_v|}})$$

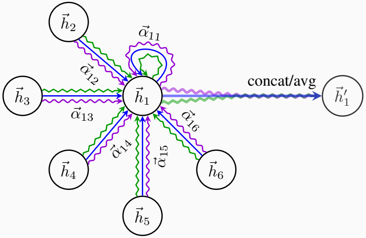
Strong theoretical background based on spectral graph convolution theory



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com





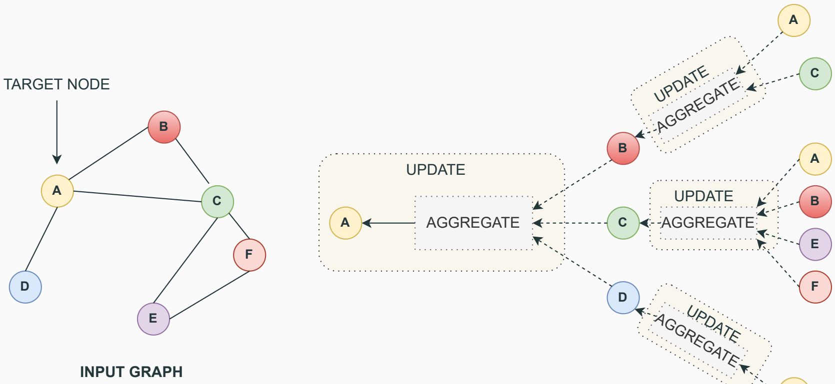
Now the normalization terms are trainable

$$\mathbf{h}^{k+1} = \sigma\Big(\bigoplus_{\forall k}\Big(\mathbf{W}_{neigh}^{(k)} \sum_{v \in \mathcal{N}(u)} a_{u,v,k}\mathbf{h}_v^k\Big)\Big)$$

$$a_{u,v} = \frac{\exp(\mathbf{a}^\top[\mathbf{W}\mathbf{h_u} \oplus \mathbf{W}\mathbf{h_v}])}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{a}^\top[\mathbf{W}\mathbf{h_u} \oplus \mathbf{W}\mathbf{h_v}])}$$

k-hop neighbourhood

If message passing applied k-times a node is aggregating information from its k-hop neighborhood.



Can we use this recipe to aggregate information even from the far distant nodes ?

Oversmoothing

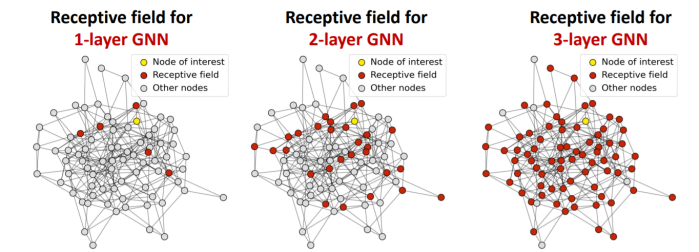
The oversmoothing problem

after several iterations of GNN message passing, the representations for all the nodes in the graph can become very similar to one another.

Oversmoothing

The oversmoothing problem

after several iterations of GNN message passing, the representations for all the nodes in the graph can become very similar to one another.



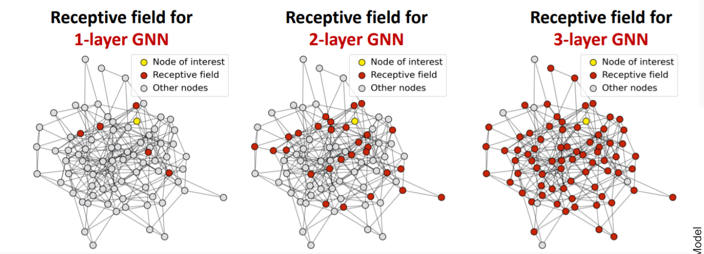
Small number of GNN layers can be used in practice.

Oversmoothing

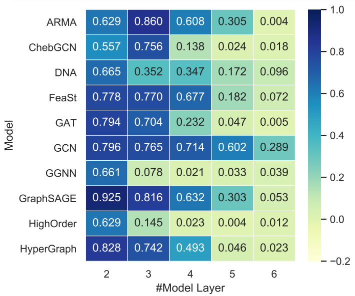
The oversmoothing problem

after several iterations of GNN message passing, the representations for all the nodes in the graph can become very similar to one another.

Mean Average Distance (MAD)



$$D_{i,j} = 1 - \frac{H_{i,:} \cdot H_{j,:}}{|H_{i,:}| \cdot |H_{j,:}|}$$



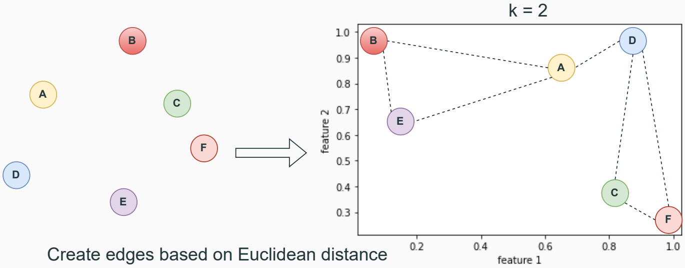
Small number of GNN layers can be used in practice.

Constructing the graph

So far we've naively assumed that the structure of the graph was given. *What do we do if we're only given a feature matrix ?*

So far we've naively assumed that the structure of the graph was given. *What do we do if we're only given a feature matrix ?*

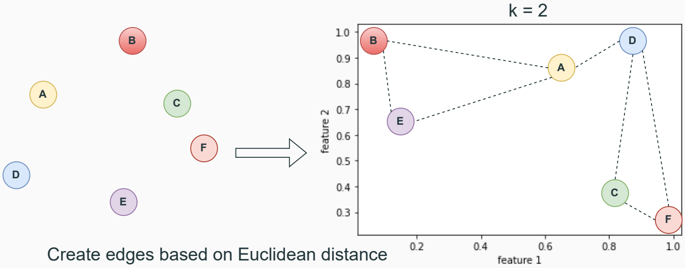
1. k-nearest neighbor graph by approximation



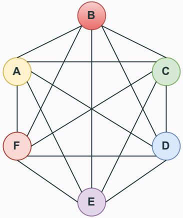
Create edges based on Euclidean distance

So far we've naively assumed that the structure of the graph was given. *What do we do if we're only given a feature matrix ?*

1. k-nearest neighbor graph by approximation
2. fully connected graph



Create edges based on Euclidean distance



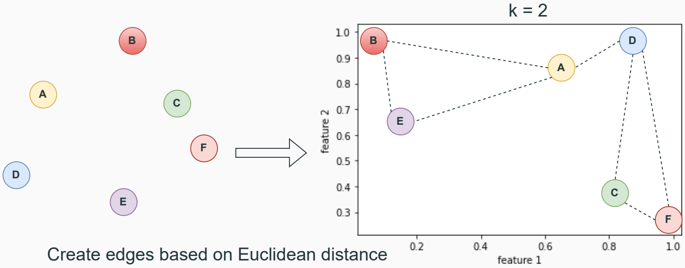
Fully Connected Graph
All nodes are interconnected

Caveat: $\mathcal{O}(n^2)$ scaling, computationally impractical for $n > 100$

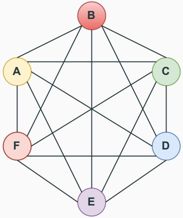
Constructing the graph

So far we've naively assumed that the structure of the graph was given. *What do we do if we're only given a feature matrix ?*

1. k-nearest neighbor graph by approximation
2. fully connected graph



Create edges based on Euclidean distance



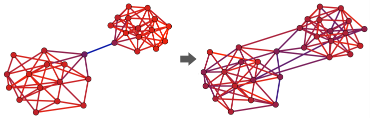
Fully Connected Graph
All nodes are interconnected

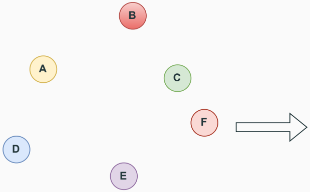
Caveat: $\mathcal{O}(n^2)$ scaling, computationally impractical for $n > 100$

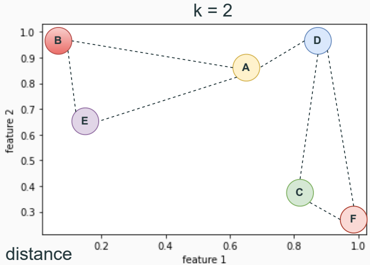
Constructing the graph

So far we've naively assumed that the structure of the graph was given. *What do we do if we're only given a feature matrix ?*

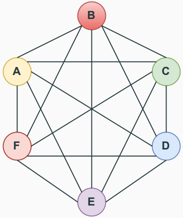
1. k-nearest neighbor graph by approximation
2. fully connected graph
3. dynamic graph







Create edges based on Euclidean distance

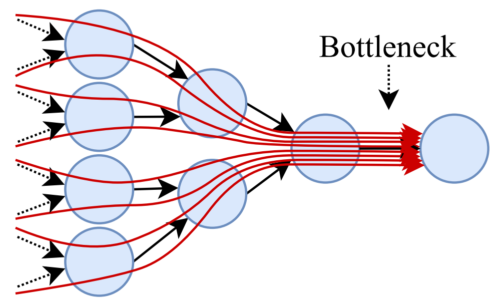


Fully Connected Graph
All nodes are interconnected

Caveat: $\mathcal{O}(n^2)$ scaling, computationally impractical for $n > 100$

Oversquashing

As the number of GNN layers increases, the number of nodes in each node’s receptive field grows exponentially. This information is then compressed into fixed-length node vectors

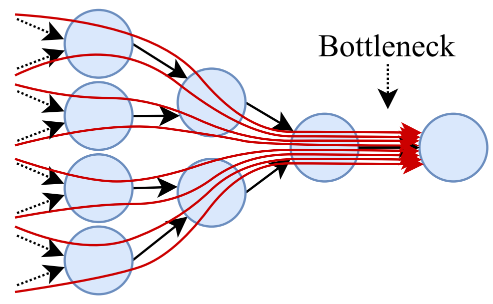


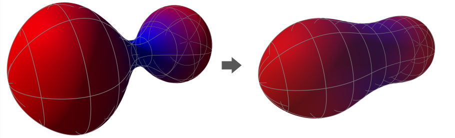
Oversquashing

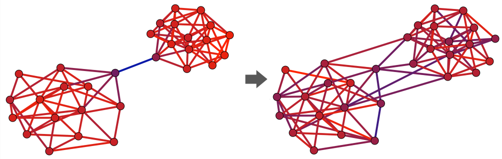
As the number of GNN layers increases, the number of nodes in each node's receptive field grows exponentially. This information is then compressed into fixed-length node vectors

Graph rewiring

it attempts to produce a new graph with a different edge structure that reduces the bottleneck







Take home messages

1. Message passing is the fundamental operation of a GNN.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).
5. Graph attention allows to learn which edges matter.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).
5. Graph attention allows to learn which edges matter.
6. If an adjacency matrix is not given:

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).
5. Graph attention allows to learn which edges matter.
6. If an adjacency matrix is not given:
    - Construct a fully connected graph if the number of nodes is small.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).
5. Graph attention allows to learn which edges matter.
6. If an adjacency matrix is not given:
    - Construct a fully connected graph if the number of nodes is small.
    - Construct a k-nearest graph using Euclidean distances or infer it from different representations.

Take home messages

1. Message passing is the fundamental operation of a GNN.
2. Message passing respects permutation invariance and permutation equivariance.
3. Three main tasks: Node, edge and graph classification.
4. Graph convolution can be used when the edges reflect node similarity (graph homophily).
5. Graph attention allows to learn which edges matter.
6. If an adjacency matrix is not given:
    - Construct a fully connected graph if the number of nodes is small.
    - Construct a k-nearest graph using Euclidean distances or infer it from different representations.
7. Oversmoothing and oversquashing are the most prominent problems with GNNs.

PyTorch geometric

DEEP GRAPH LIBRARY

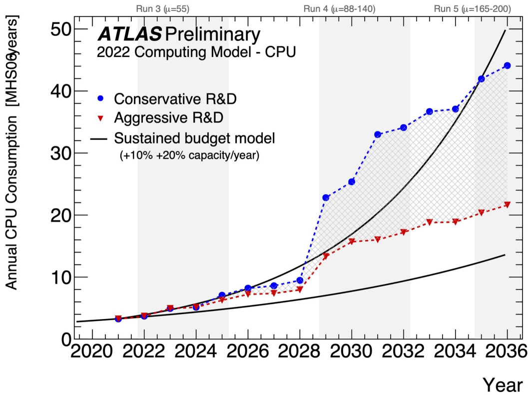
deepmind/
graph_nets

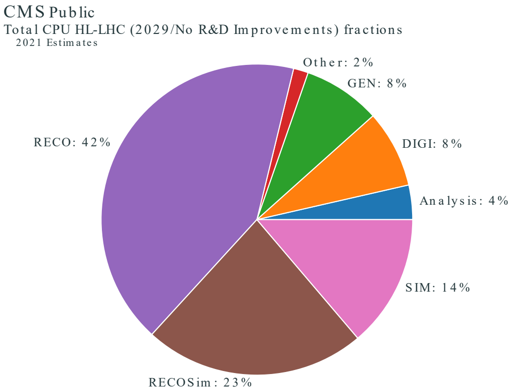
Build Graph Nets in Tensorflow

11 Contributors · 4 Issues · 5k Stars · 777 Forks

Spektral

Why GNNs in HEP?

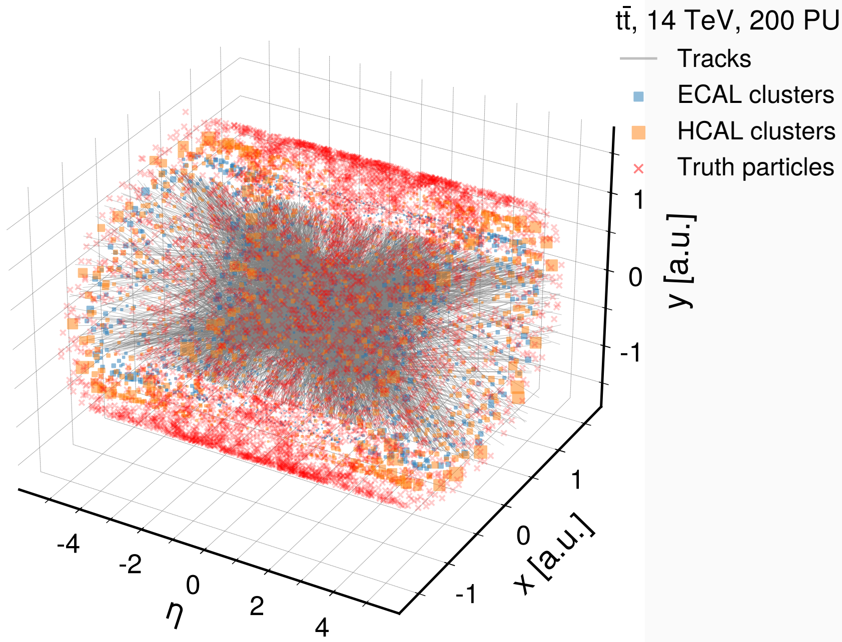




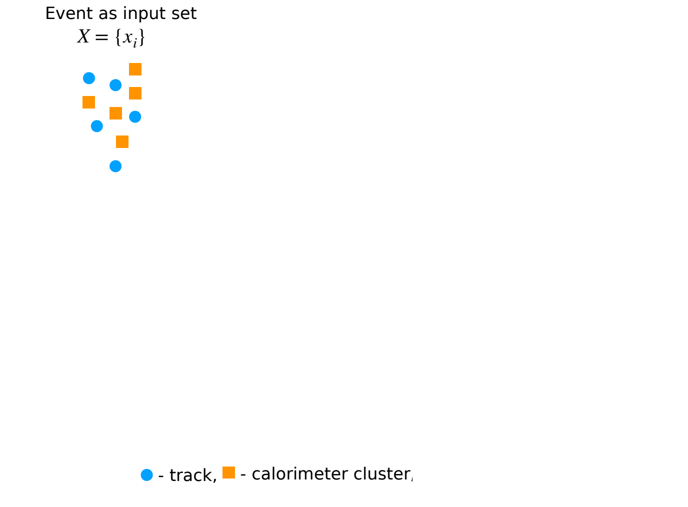
Three main objectives:

1. Improve algorithm performance
2. Accelerate algorithm inference
3. Accelerate data generation/simulation

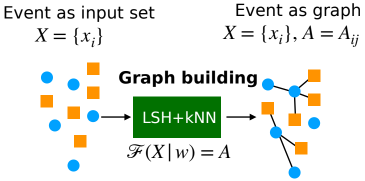
Particle flow



MLPF: a node classification task



MLPF: a node classification task

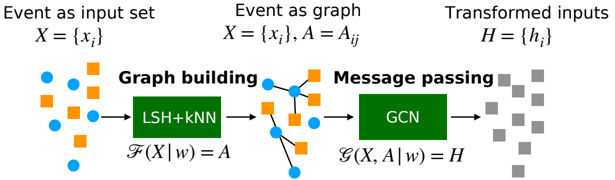


$x_i = [\text{type}, p_\mathrm{T}, E_\mathrm{ECAL}, E_\mathrm{HCAL}, \eta, \phi, \eta_\mathrm{outer}, \phi_\mathrm{outer}, q, \ldots], \ \text{type} \in \{\text{track}, \text{cluster}\}$

Trainable neural networks: $\mathscr{F}$,

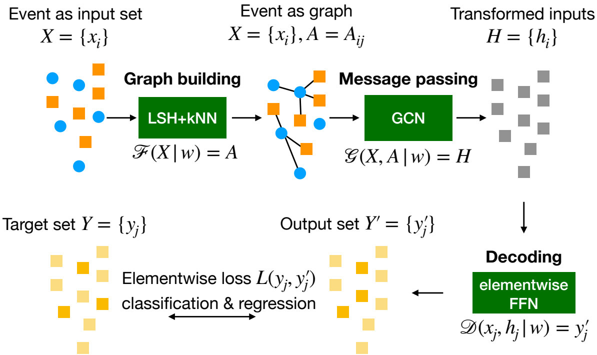
- track, - calorimeter cluster,

MLPF: a node classification task



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

Trainable neural networks: $\mathscr{F}, \mathscr{G}$,
● - track, ■ - calorimeter cluster, ■ - encoded element



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

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

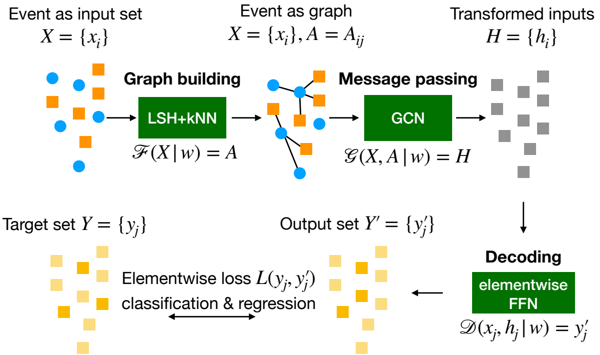
$$h_i \in \mathbb{R}^{256}$$

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

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

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

MLPF: a node classification task



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

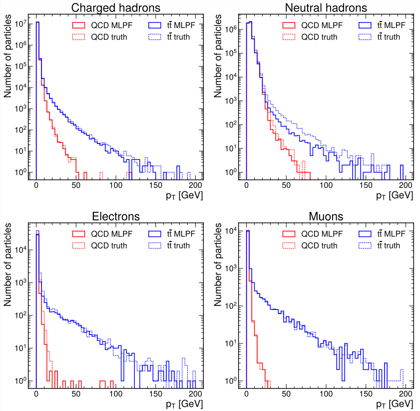
$$y_j = [\text{PID}, p_\mathrm{T}, E, \eta, \phi, q, \ldots], \quad \text{PID} \in \{\text{none}, \text{charged hadron}, \text{neutral hadron}, \gamma, e^\pm, \mu^\pm\}$$

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

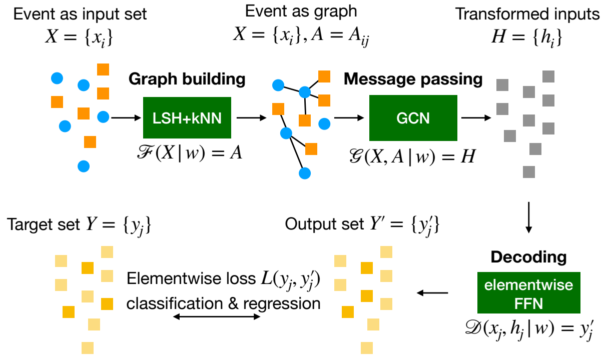
Trainable neural networks: $\mathcal{F}, \mathcal{G}, \mathcal{D}$

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

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



MLPF: a node classification task



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

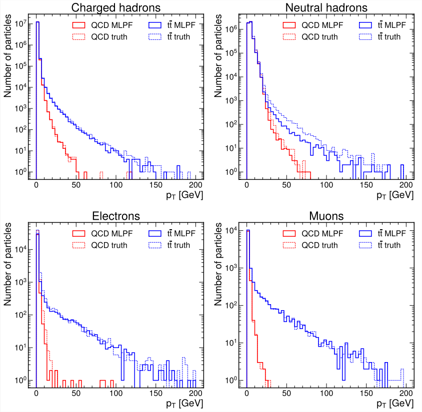
$$y_j = [\text{PID}, p_\text{T}, E, \eta, \phi, q, \ldots], \quad \text{PID} \in \{\text{none}, \text{charged hadron}, \text{neutral hadron}, \gamma, e^\pm, \mu^\pm\}$$

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

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

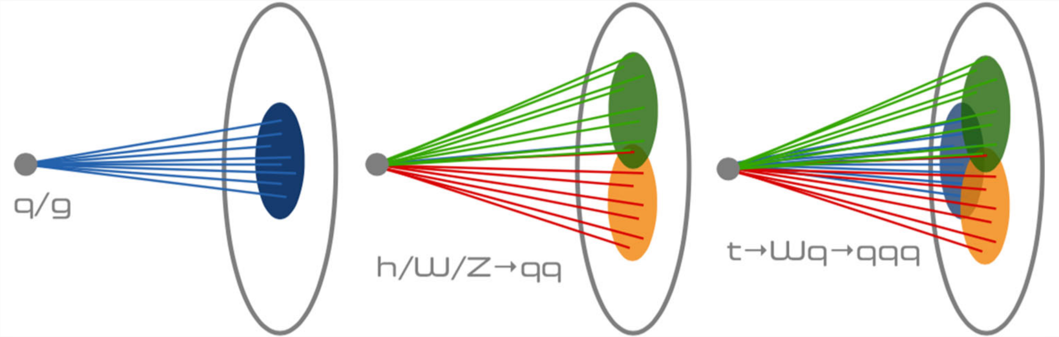
● - track, ■ - calorimeter cluster, ■ - encoded element

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

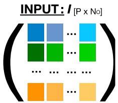


Metric	Charged hadrons Rule-based PF	Charged hadrons MLPF	Neutral hadrons Rule-based PF	Neutral hadrons MLPF
Efficiency	0.953	0.953	0.883	**0.908**
Fake rate	0.000	0.000	0.071	**0.068**
p_T (E) resolution	0.213	**0.137**	0.350	**0.323**
η resolution	**0.240**	0.245	**0.050**	0.058
N resolution	0.004	0.004	0.014	**0.013**

Jet tagging

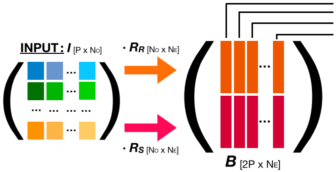


Jedi-Net: a graph classification task



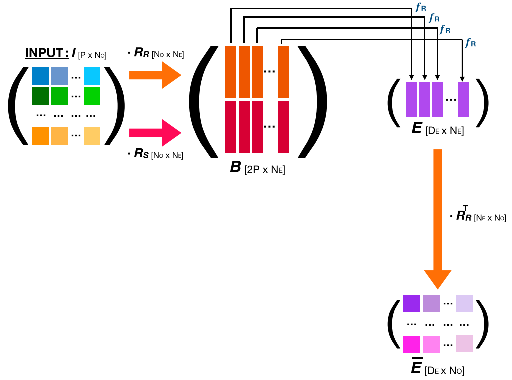
No: # of constituents
P: # of features

Jedi-Net: a graph classification task



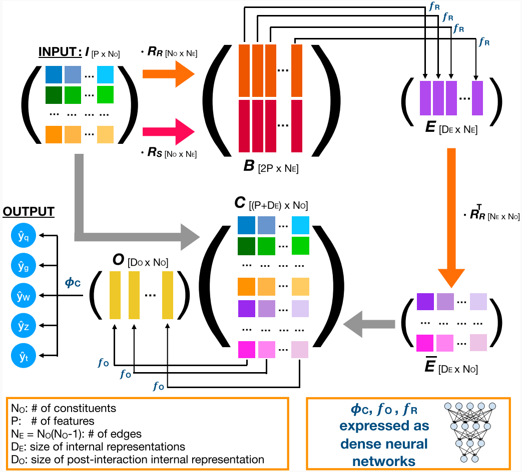
N_O: # of constituents
P: # of features
$N_E = N_O(N_O-1)$: # of edges

Jedi-Net: a graph classification task

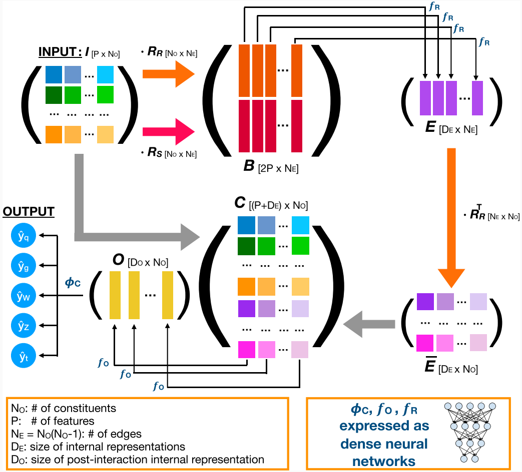


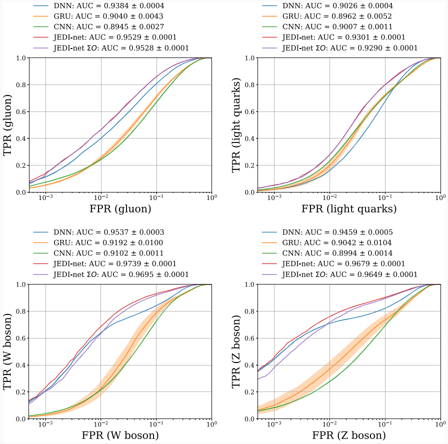
N_O: # of constituents
P: # of features
$N_E = N_O(N_O-1)$: # of edges
D_E: size of internal representations

Jedi-Net: a graph classification task

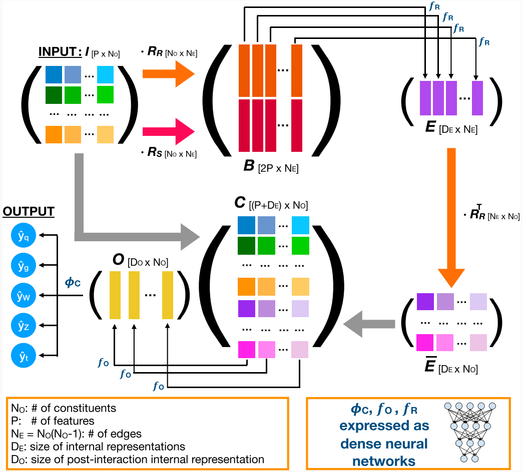


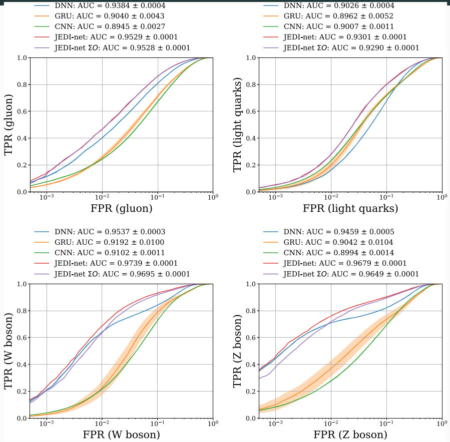
Jedi-Net: a graph classification task





Jedi-Net: a graph classification task





Model	Number of parameters	Number of FLOP	Inference time/batch (ms)
DNN	14725	27 k	1.0 ± 0.2
CNN	205525	400 k	57.1 ± 0.5
GRU	15575	46 k	23.2 ± 0.6
JEDI-net	33625	116 M	121.2 ± 0.4
JEDI-net with $\sum O$	8767	458 M	402 ± 1

Conclusions

I hope we've established:

- Graphs are:

Conclusions

I hope we've established:

- Graphs are:
    1. cool

Conclusions

I hope we've established:

- Graphs are:
    1. cool
    2. everywhere

Conclusions

I hope we've established:

- Graphs are:
  1. cool
  2. everywhere
  3. permutation invariant

Conclusions

I hope we've established:

- Graphs are:
    1. cool
    2. everywhere
    3. permutation invariant

- Basic representations of graphs

Conclusions

I hope we've established:

- Graphs are:
  1. cool
  2. everywhere
  3. permutation invariant
- Basic representations of graphs
- Many different graph architectures, but they are all conceptually doing message passing.

Conclusions

I hope we've established:

- Graphs are:
    1. cool
    2. everywhere
    3. permutation invariant

- Basic representations of graphs

- Many different graph architectures, but they are all conceptually doing message passing.

- Constructing a graph and predicting node/edges/graph labels is possible in HEP.

Further reads

- Graph convolution theoretical motivations 1, 2, 3
- k-nearest graph inference 1, 2, 3
- Generative models and unsupervised learning 1, 2, 3
- How powerful are GNNs ? Graph isomorphism and the WL algorithm 1, 2

BACK-UP

Back to basics

Convolution
a mathematical operation on two functions (f and g) that produces a third function which expresses how the shape of one is modified by the other.

Convolution
a mathematical operation on two functions (f and g) that produces a third function which expresses how the shape of one is modified by the other.

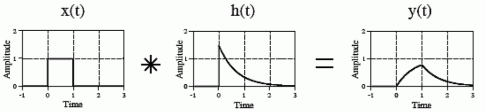


FIGURE 13-5
Example of continuous convolution. This figure illustrates a square pulse entering an RC low-pass filter (Fig. 13-4). The square pulse is convolved with the system's impulse response to produce the output.

Convolution

a mathematical operation on two functions (f and g) that produces a third function which expresses how the shape of one is modified by the other.

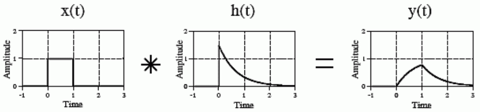


FIGURE 13-5
Example of continuous convolution. This figure illustrates a square pulse entering an RC low-pass filter (Fig. 13-4). The square pulse is convolved with the system's impulse response to produce the output.

Convolution Theorem

under suitable conditions the Fourier transform of a convolution of two functions (or signals) is the pointwise product of their Fourier transforms.

Back to basics

Convolution

a mathematical operation on two functions (f and g) that produces a third function which expresses how the shape of one is modified by the other.

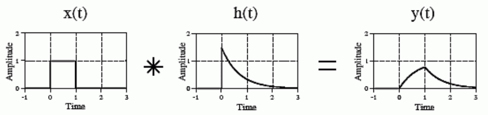
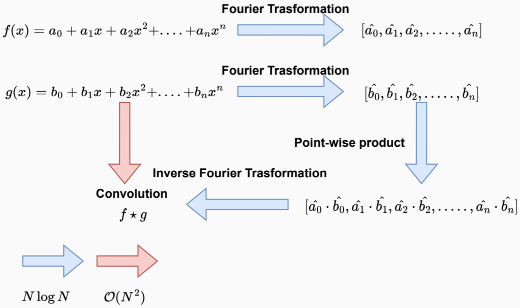


FIGURE 13-5
Example of continuous convolution. This figure illustrates a square pulse entering an RC low-pass filter (Fig. 13-4). The square pulse is convolved with the system's impulse response to produce the output.

Convolution Theorem

under suitable conditions the Fourier transform of a convolution of two functions (or signals) is the pointwise product of their Fourier transforms.



Graph Fourier Tranformation

The graph Fourier transformation is defined as:

$$\mathscr{F}(x) = \mathbf{U^T x},\ \mathscr{F}^{-1}(\hat{x}) = \mathbf{U\hat{x}}$$

where U is the eigenvector matrix of the graph Laplacian.

Graph Fourier Tranformation

The graph Fourier transformation is defined as:

$$\mathscr{F}(x) = \mathbf{U^T x},\ \mathscr{F}^{-1}(\hat{x}) = \mathbf{U\hat{x}}$$

where U is the eigenvector matrix of the graph Laplacian.

The Laplacian matrix can be factored as

$$\mathbf{L} = \mathbf{U}\Lambda U^T$$

where Λ are the eigenvalues of L.

Graph Fourier Tranformation

The graph Fourier transformation is defined as:

$$\mathcal{F}(x) = \mathbf{U^T x},\ \mathcal{F}^{-1}(\hat{x}) = \mathbf{U\hat{x}}$$

where U is the eigenvector matrix of the graph Laplacian.

The Laplacian matrix can be factored as

$$\mathbf{L} = \mathbf{U}\Lambda U^T$$

where Λ are the eigenvalues of L.

Graph convolution

$$g_\theta \star x = U g_\theta U^T x$$

where g_θ is a function of Λ.

Graph Fourier Tranformation

The graph Fourier transformation is defined as:

$$\mathcal{F}(x) = \mathbf{U^T x},\ \mathcal{F}^{-1}(\hat{x}) = \mathbf{U\hat{x}}$$

where U is the eigenvector matrix of the graph Laplacian.

The Laplacian matrix can be factored as

$$\mathbf{L} = \mathbf{U}\Lambda U^T$$

where Λ are the eigenvalues of L.

Graph convolution

$$g_\theta \star x = U g_\theta U^T x$$

where g_θ is a function of Λ.

Problems:

1. Computing the eigencomposition of L can be expensive for large graphs
2. A slight change in the eigenvector affects the whole graph

Graph Convolution Approximation

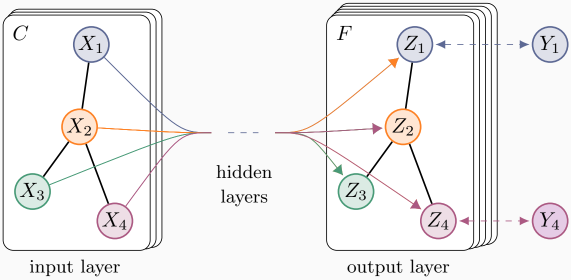
Kipf and Welling approximated $g_\theta(\Lambda)$ as an expansion of Chebyshev coefficients of the adjacency matrix up to **2nd order**.

$$g_\theta \star x = \theta_0{}'x + \theta_1{}'(L - I_N)x = \theta_0{}'x - \theta_1{}'D^{-\frac{1}{2}}AD^{-\frac{1}{2}}x$$

After some empirical tricks:

$$Z = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$$

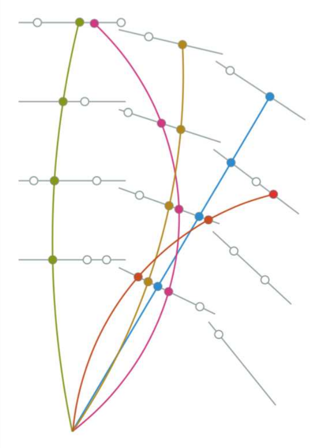
with $\Theta \in \mathcal{R}^{C\times F}$ and $Z \in \mathcal{R}^{N\times F}$. Now the filtering operation has complexity $\mathcal{O}(|\mathcal{E}|FC)$.



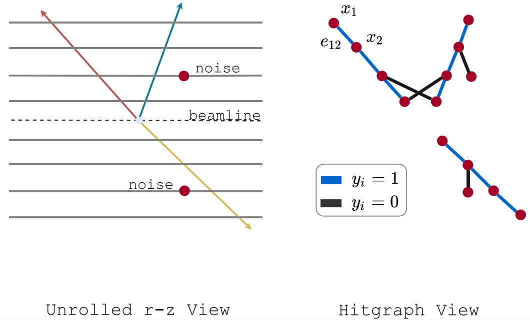
We added a normalization term in the aggregation function

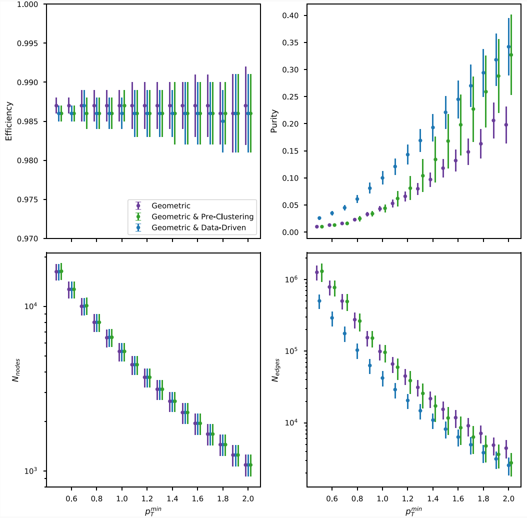
$$\mathbf{h}^{k+1} = \sigma(\mathbf{W}^{(k)}_{neigh} \sum_{v\in\mathcal{N}(u)} \frac{\mathbf{h}^k_v}{\sqrt{|\mathcal{N}_u||\mathcal{N}_v|}})$$

Tracking



Track fitting as edge classification





Track fitting as edge classification