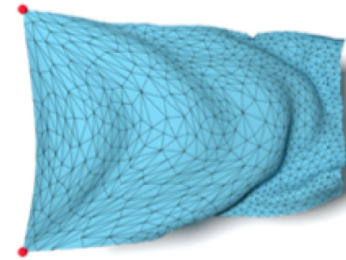
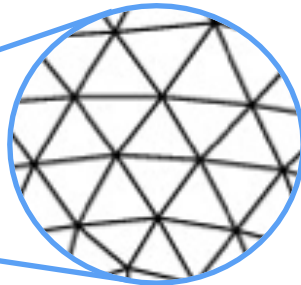
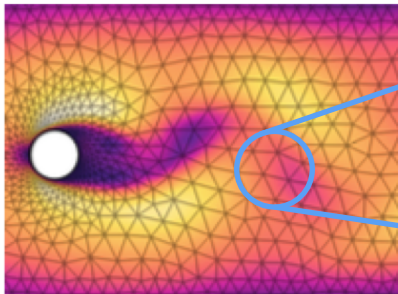


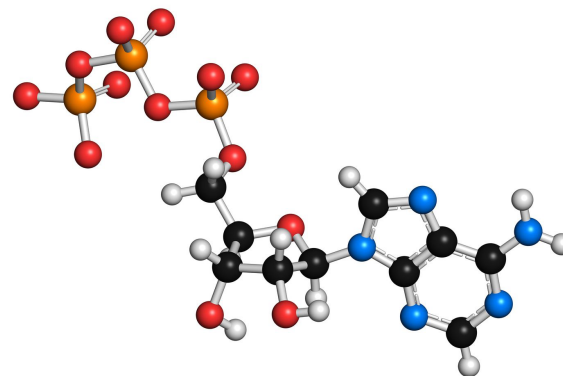
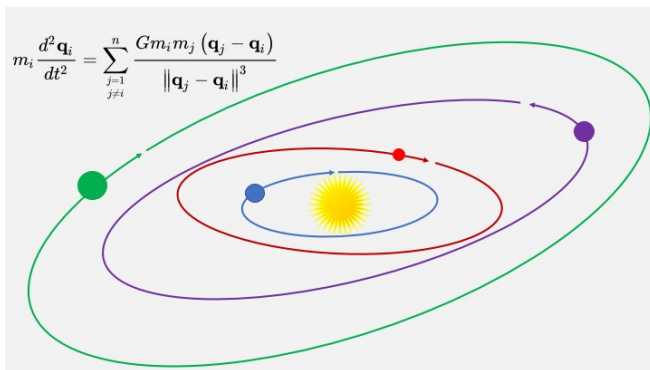
GNNs for Particle- and Mesh-Based Simulation (I)



Why GNNs for physic?

- **Graph Neural Networks (GNNs)** are specifically designed to operate on **graph-structured data**.
- Many physical systems can be naturally represented as graphs:
 - **Nodes** represent **entities** (e.g., particles, atoms, agents).
 - **Edges** represent **relationships or interactions between them** (e.g., forces, distances, connections).

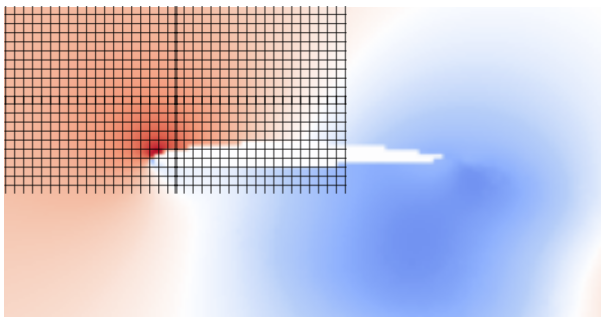
N-body
problem



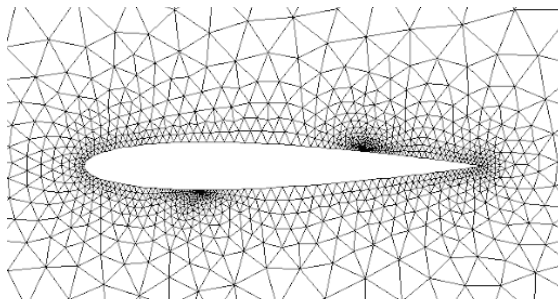
Molecules

Why GNNs for physic?

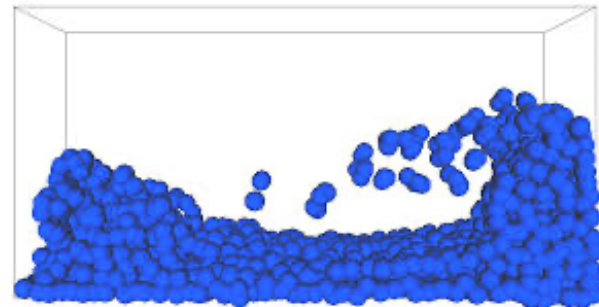
- Other physical systems take place in a **continuous space domain** (e.g., fluid dynamics, elastic and plastic deformation of solid bodies).
- These domains need to be discretised into a finite number of degrees of freedom using:
 - A structure discretisation (typically a cartesian grid) → CNN
 - An unstructured mesh → GNN
 - A Lagrangian discretisation → GNN



Cartesian grid



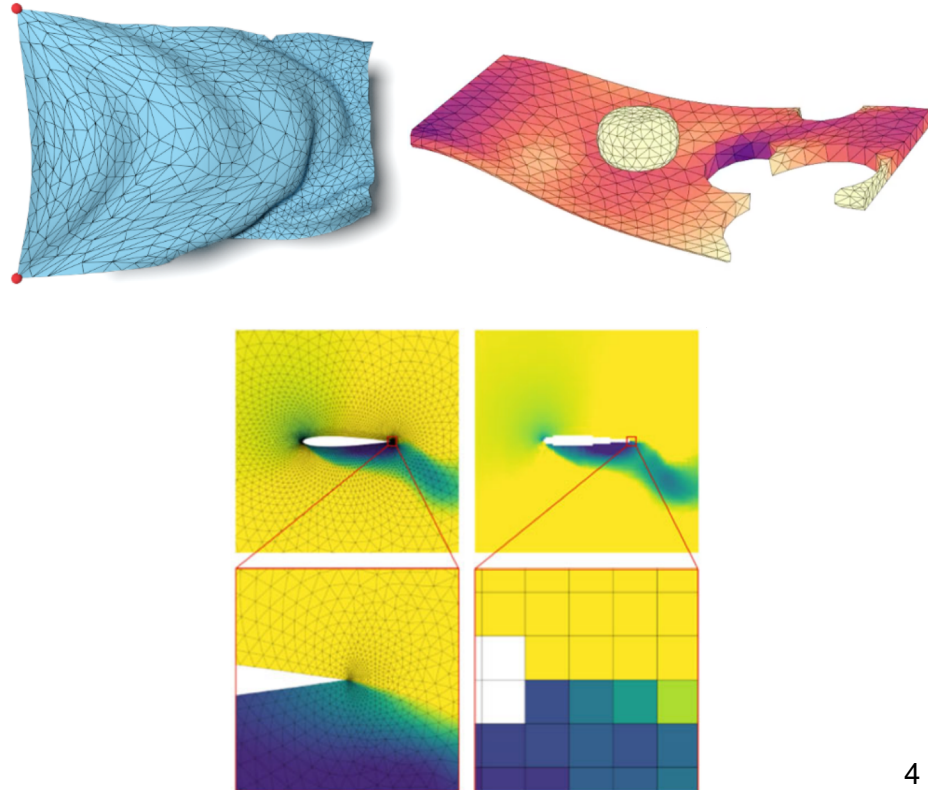
Unstructured mesh



Lagrangian discretisation (SPH)

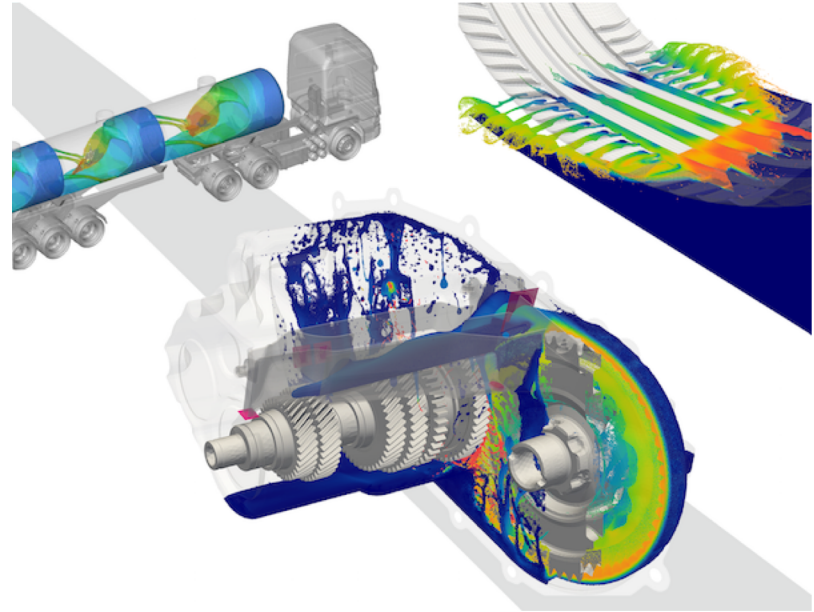
Why GNNs for physic?

- **Unstructured meshes** are preferred over Cartesian grids because
 - They allow us to represent complex geometries accurately.
 - They allow us to **modify the resolution over space** and efficiently capture large spatial gradients.



Why GNNs for physic?

- **Lagrangian** (or particle-based) simulations can also be suitable for multi-physics problems and problems with moving boundaries.



Content

1. Introduction
2. Graph Convolution and Message-Passing
3. Particle-based simulation with GNNs
 - Systems of particles
 - Lagrangian simulations of fluids
 - Inverse problems
 - Momentum conservation
4. Mesh-based simulation with GNNs
 - Fluid, structural and cloth dynamics
 - Graph pooling and unpooling
 - Multi-scale GNN
 - Rotation equivariance

1. Introduction

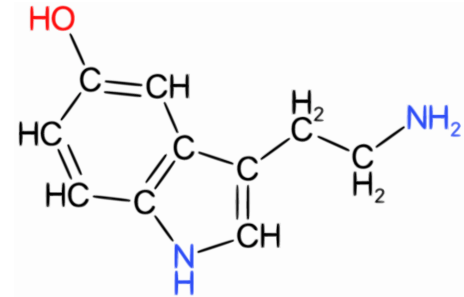
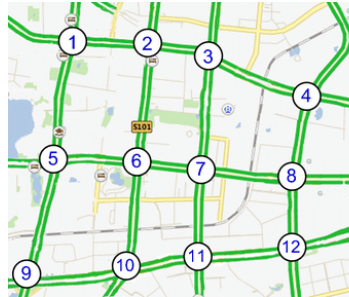
- [illegible]

- In the Euclidean space the data points can be represented as **vectors or tensors** and the distance between them is computed using Euclidean distance, i.e.,

$$d := \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_N - q_N)^2}$$

Introduction – Graphs

- In other applications the data is represented in a **non-Euclidean space** using **graphs**
- Graphs can be used to represent social networks, citation networks, traffic networks, molecules and proteins

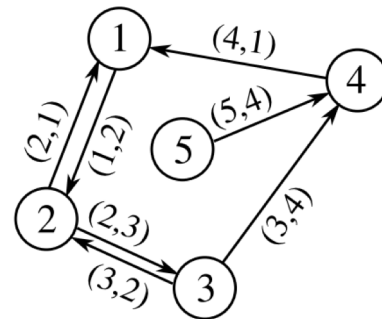


- Graphs are structures composed of nodes (vertices) and edges (connections) between these nodes
- Nodes can represent entities (such as objects, individuals, or concepts), and edges can represent relationships or interactions between these entities

Types of graphs

A graph is a pair of sets, $G := (V, E)$, where $V := \{ i \mid 1 \leq i \leq |V| \}$ is a finite set of nodes and $E := \{ (i, j) \mid i, j \in V \}$ is a finite set of pairs of nodes

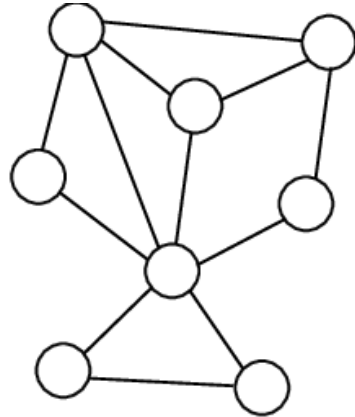
E.g., a graph with nodes $V = \{1, 2, 3, 4, 5\}$
and edges $E = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (5, 4)\}$



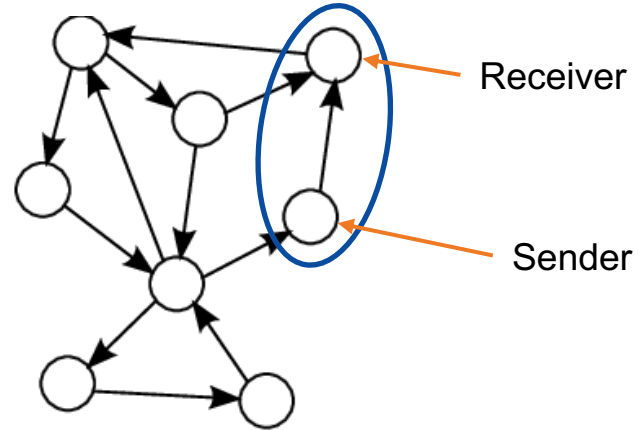
Graphs can fall within different categories:

- Undirected graph or Directed graph
- (Simple) graph or Multigraph
- Homogeneous graph or Heterogeneous graph
- Pseudograph
- Hypergraph

Undirected vs directed graph



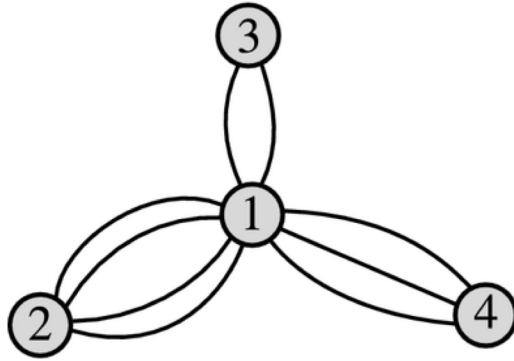
Undirected graph



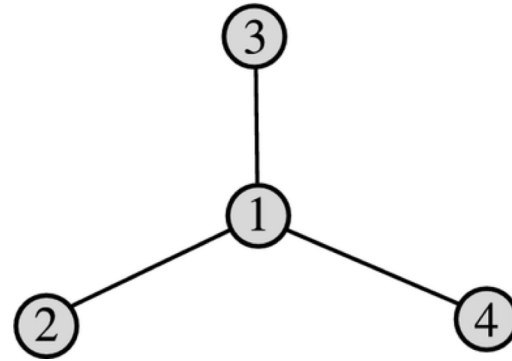
Didirected graph

- Undirected graph: the edges have no direction and represent a two-way relationship. The tuples (i, j) and (j, i) refer to the same edge (unordered tuple).
- Directed graph (or digraph): the edges have a specific direction and represent a one-way relationship. The tuples (i, j) and (j, i) correspond to different edges (ordered tuple).

Multigraph vs Simple graph



Multigraph

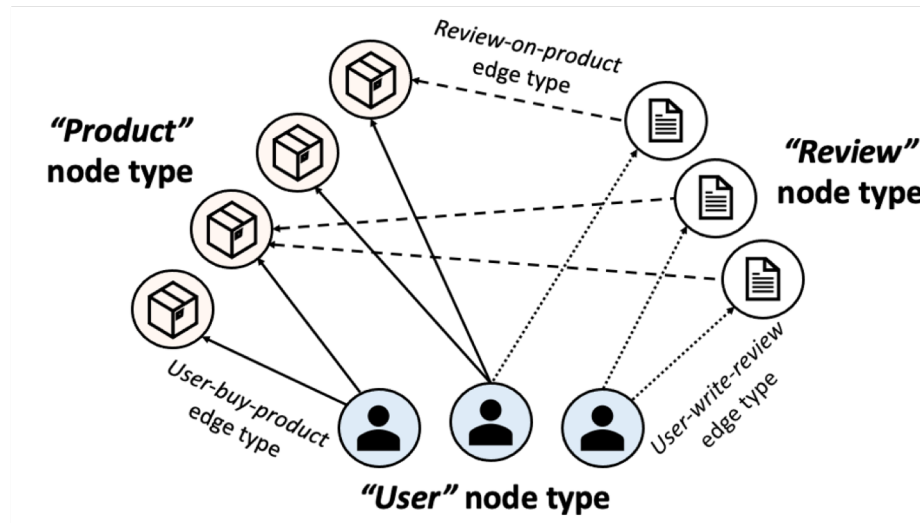


Simple graph

- Multigraph: multiple edges with same end nodes are allowed. In this case, E must be an order set.
- (Simple) graph: more than one edge between any two nodes are not allowed.

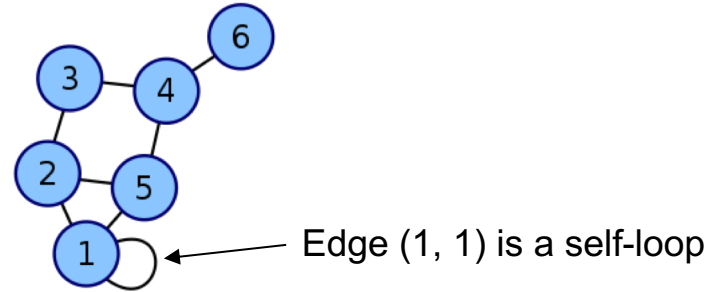
Homogeneous vs Heterogeneous graph

- Homogeneous graph: There is only one type of nodes and one type of edges.
- Heterogeneous graph: There are multiple types of nodes and multiple types of edges.
E.g., citation networks and e-commerce networks.

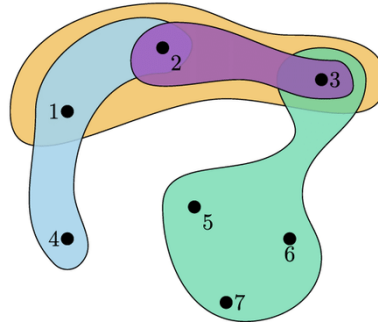


Special types of graph

- Pseudograph: self-loops are allowed



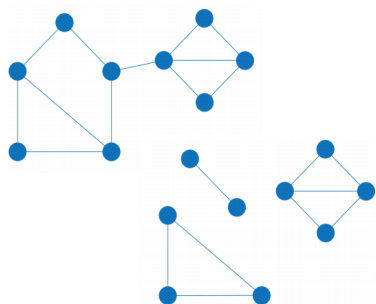
- Hypergraph: generalization of a graph in which an edge can join any number of nodes



- Complete graph (or fully connected graph): a graph in which each node is connected to every other node

Special types of graph

- Disconnected graph: there is no path between any two nodes of the graph.

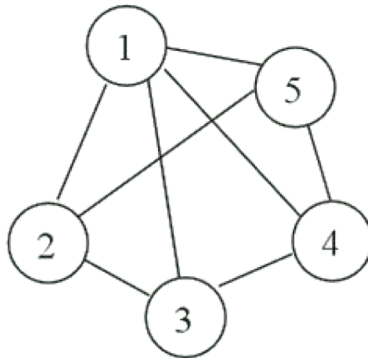


This graph consists of 4 disconnected subgraphs.

In message-passing based GNNs **there is no batch dimension**. Instead, **we use a larger disconnected graph** consisting of several of the training samples. This just requires changing the index assign to each node.

Adjacency (undirected graphs)

- Adjacent node of a node i : any node that is connected to i by an edge
- Neighbourhood of a node i : all the vertices adjacent to i
- Degree of a node i : number of edges that are incident to the node (or neighbourhood size)
- Distance between two nodes: the number of edges (*hops*) in the shortest path connecting the two nodes

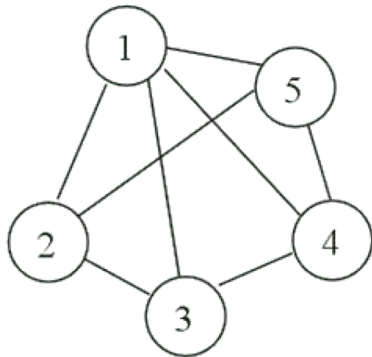


- The neighbourhood of node 5 is $\{1, 2, 4\}$, it has degree 3.
- Node 5 and 3 are two hops away.

Adjacency (undirected graphs)

Adjacency matrix: **symmetric** square matrix that represents the connections between the nodes in a graph. The matrix element in the i -th row and the j -th column indicates whether there is an edge connecting nodes i and j . This can be:

- Unweighted: The elements of the matrix are either 0 (no edge) or 1 (edge).
- Weighted: Each element represents the weight of the edge between the corresponding pair of nodes. If there is no edge, the entry is typically represented as 0 or $\pm\infty$.



Unweighted adjacency matrix

	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	1
3	1	1	0	1	0
4	1	0	1	0	1
5	1	1	0	1	0

Weighted adjacency matrix

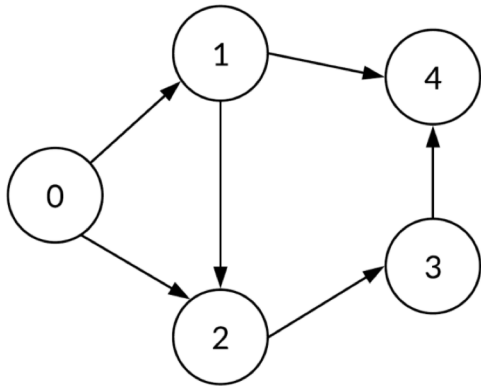
	1	2	3	4	5
1	0	1.3	0.7	5.2	4.2
2	1.3	0	1.5	0	9
3	0.7	1.5	0	1.6	0
4	5.2	0	1.6	0	0.1
5	4.2	9	0	0.1	0

Adjacency (directed graphs)

The adjacency, degree and distance are defined for the incoming and outgoing directions.
For instance,

- Indegree of a node: number of incoming edges at the node.
- Outdegree of a node: number of outgoing edges at the node.

The adjacency matrix is **not necessarily symmetric**. The matrix element in the i -th row and the j -th column indicates whether there is an edge from node i to node j .



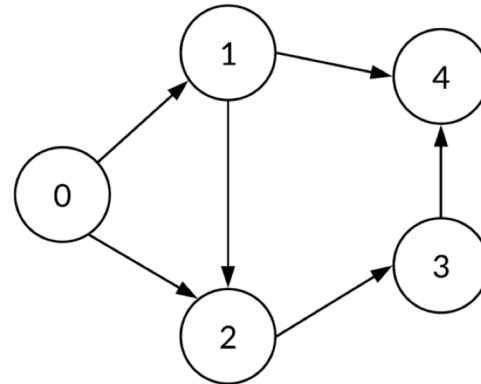
	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

The sum is the
outdegree

Edge list

- The adjacency matrix allows for efficient matrix-based operations (e.g., aggr., degree).
- However, for sparse graphs, the adjacency matrix is inefficient in terms of memory usage.
- For such cases, alternative representations are **adjacency lists** and **edge lists**.
- Edge list: List of all the edges in the graph, where each edge is represented as a tuple or pair of nodes. Commonly used in message-passing algorithms. As a con, the lookup time is large (e.g., finding the neighbours of a node).

Edge list: `[[0, 1], [0, 2], [1, 2], [2, 3], [1, 4], [3, 4]]`



Attributes of a graph

The inputs to GNN models are:

- Edge list/Adjacency matrix
- Node attributes or features. This is generally a vector for each node. The node feature matrix has in the i -th row the feature vector of node i .
- Edge attributes or features (optional attribute). The edge feature matrix has in the i -th row the feature vector of the i -th edge in the edge list.

Attributes of a graph: Particle based simulation

- The adjacent nodes of each node are created following different techniques:
 - Nodes within a radius R
 - k nearest neighbours

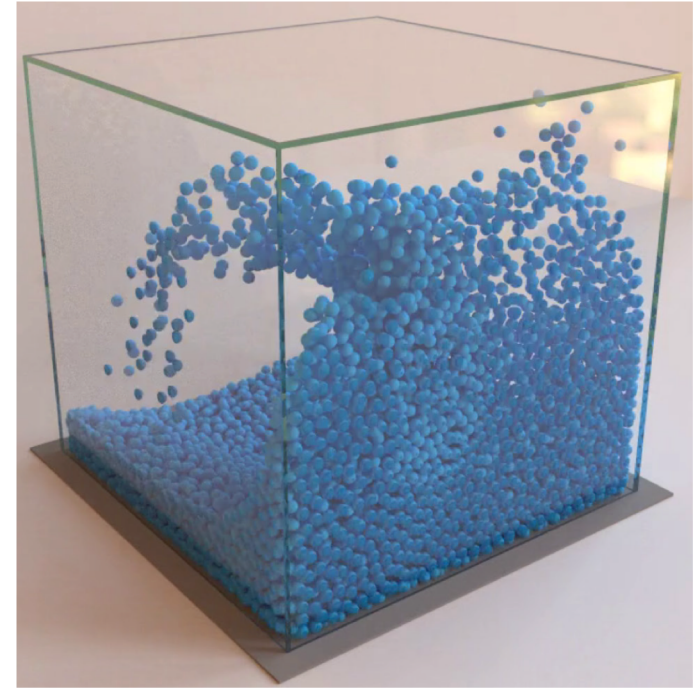
Valid input attributes could be:

- Features of node each node i :

$$\mathbf{v}_i = [\mathbf{x}_i \mid \dot{\mathbf{x}}_i \mid g \mid f_i \mid d_i \mid m_i]$$

- Particle's position
- Particle's velocity
- Gravity
- External forces
- Particle's distance to the wall
- Masks

- Features of each edge (i, j) : $\mathbf{e}_{ij} = [\mathbf{x}_j - \mathbf{x}_i \mid d_{ij}]$



Attributes of a graph: Mesh based simulation

- The edges can be constructed as before or by meshing (e.g., Delaunay triangularisation).

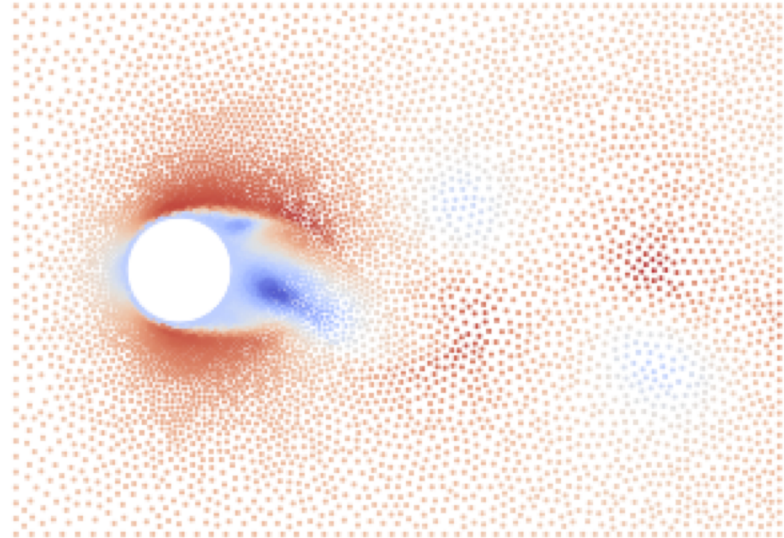
Valid output attributes could be:

- Features of node each node i :

$$\mathbf{v}_i = [\mathbf{u}_i \mid f_i \mid d_i \mid m_i]$$

- Eulerian magnitudes (e.g., velocity and pressure)
- External forces
- Node's distance to the wall
- Masks

- Features of each edge (i, j) : $\mathbf{e}_{ij} = [\mathbf{x}_j - \mathbf{x}_i]$



Horizontal component of the
velocity field

2. Graph Convolution and Message-Passing

Graph convolution

- A graph may have a variable number of nodes and each node may have a different degree. Besides the notion of *up*, *down*, *right*, and *left* may not be defined (e.g., in a citation network) or defined in a continuous way using a coordinates system (e.g., in a physical simulation)
- Graph convolution are a generalisation of convolutions from grid data to graph data.
- They can be of two types
 - Spectral graph convolutions: A learnable kernel is convolved with the node features in the graph spectral domain, defined by the adjacency matrix of the graph.
 - Spatial graph convolutions: Neighboring nodes' features are processed and aggregated to update the central node's features. Commonly referred to as ***Message Passing***.

Spectral graph convolution

- **Only valid for undirected graphs.**
- Spectral-based methods have foundation in graph signal processing and apply a filter in the spectral domain.
- To transform a graph signal , $\mathbf{v} \in \mathbb{R}^{|V|}$, to the spectral domain:
 1. Compute the normalised graph Laplacian, \hat{L} . The graph Laplacian is given by $L = D - A$, where A is the adjacency matrix and D is the degree matrix – a diagonal matrix indicating degree of each node ($D_{ii} = \sum_j A_{i,j}$).

Instead we use the **normalised graph Laplacian**, given by

$$\hat{L} = I - D^{-1/2} A D^{-1/2}$$

Since its eigenvalues are bounded between 0 and 2, it provides better numerical stability than the unnormalised graph Laplacian.

Spectral graph convolution

- To transform the matrix of node features, $X \in \mathbb{R}^{|V| \times F}$, to the spectral domain:
 2. Compute the eigenvalues and eigenvectors of \hat{L} . \hat{L} can be factored as:

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

where $U \in \mathbb{R}^{|V| \times |V|}$ is the matrix of (unitary) eigenvectors ordered by decreasing eigenvalues

$$\Lambda \in \mathbb{R}^{|V| \times |V|}$$

is the diagonal matrix of eigenvalues.

3. Apply the **graph Fourier transform**:

$$\mathcal{F}(\mathbf{v}) = U^T \mathbf{v}$$

- To recover the graph signal, the **inverse graph Fourier transform** is applied:

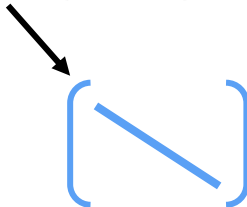
$$\mathcal{F}^{-1}(\tilde{\mathbf{v}}) = U \tilde{\mathbf{v}}$$

Spectral graph convolution

The graph convolution of the input signal $\mathbf{v} \in \mathbb{R}^{|V|}$ with a filter $\mathbf{g} \in \mathbb{R}^{|V|}$ is defined as

$$\begin{aligned}\mathbf{v} * \mathbf{g} &= \mathcal{F}^{-1}(\mathcal{F}(\mathbf{v}) \odot \mathcal{F}(\mathbf{g})) \\ &= U(U^T \mathbf{v} \odot U^T \mathbf{g})\end{aligned}$$

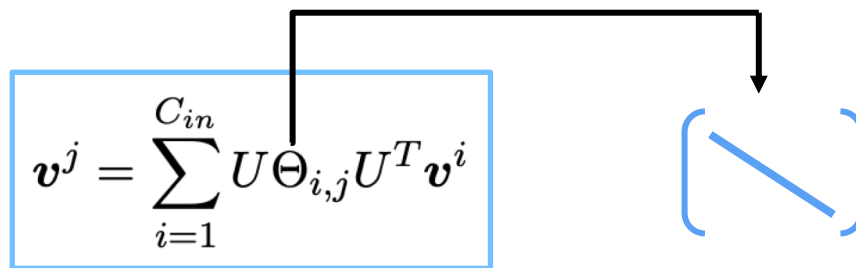
If we denote the filter in the graph Fourier space as $\mathbf{g}_\theta = \text{diag}(U^T \mathbf{g})$, then we can write the spectral graph convolution as

$$\boxed{\mathbf{v} * \mathbf{g} = U \mathbf{g}_\theta U^T \mathbf{v}}$$


All spectral graph convolution follow this definition and differ on the choice of \mathbf{g}_θ .

Spectral CNN (Bruna et al. 2014)

- In Spectral CNNs (presented in *Spectral networks and locally connected networks on graphs*), each convolution has C_{in} input features/signals and C_{out} output features/signals.
- The filter for each pair of input and output channels is a learnable diagonal matrix, denoted as $\Theta_{i,j}$.
- The j -th output feature is



The diagram illustrates the computation of the j -th output feature v^j . On the left, a blue-bordered box contains the equation
$$v^j = \sum_{i=1}^{C_{in}} U \Theta_{i,j} U^T v^i$$
. A black arrow originates from the term $\Theta_{i,j}$ in the equation and points to a blue matrix representation on the right. This matrix is depicted as a square with a diagonal line from the top-left to the bottom-right, indicating it is a diagonal matrix.

- In the Spectral CNN (actually GNN), multiple convolutions (followed by activation functions) are applied sequentially.

Spectral CNN (Bruna et al. 2014)

Since the convolutions in **Spectral CNNs** rely on the **eigenvectors** of the normalised Laplacian matrix, they face two important limitations:

- The learned filters **cannot be applied to a graph with a different structure**.
- The computational complexity of eigen-decomposition is $O(|V|^3)$.

More recent spectral graph convolutions make several simplifications to reduce the computational complexity:

- Chebyshev Spectral CNN (ChebNet) $\rightarrow O(k |E|)$
- Graph Convolution Network (GCN) $\rightarrow O(|E|)$

Chebyshev Spectral CNN (Defferrard et al. 2016)

- In Chebyshev Spectral CNN (ChebNet), presented in *Convolutional neural networks on graphs with fast localized spectral filtering*, **the convolution filters are approximated by the weighted sum of Chebyshev polynomials of the normalised eigenvalue matrix**.
- Specifically,

$$\mathbf{g}_\theta = \sum_{i=0}^k \theta_i T_i(\hat{\Lambda})$$

where θ_i is a learnable scalar, k is the filter order (order of the highest polynomial), T_i is the Chebyshev polynomial of order i and $\hat{\Lambda} = 2\Lambda/\lambda_{\max} - I$ (in $[-1, 1]$) is the normalised eigenvalue matrix.

$$\begin{aligned} \text{Chebyshev polynomials:} \quad & T_0(x) = 1 \\ & T_1(x) = x \\ & T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x). \end{aligned}$$

Chebyshev Spectral CNN (Defferrard et al. 2016)

- The Chebyshev spectral convolution is given by

$$\mathbf{v} * \mathbf{g} = U \underbrace{\left(\sum_{i=0}^k \theta_i T_i(\hat{\Lambda}) \right)}_{\text{Chebyshev filter}} U^T \mathbf{v} = \sum_{i=0}^k \theta_i T_i(2\hat{L}/\lambda_{\max} - I) \mathbf{v}$$

$T_i(2\hat{L}/\lambda_{\max} - I) = U T_i(\hat{\Lambda}) U^T$

- If we assume $\lambda_{\max} = 2$, **the computational complexity is reduced to $O(k |E|)$** , since we do not have to compute any eigenvalue or eigenvector.
- As opposed Spectral CNNs, the **filters are local in space** and can be applied to graphs with a different structure.
- Each node is receiving information from the nodes located **k hops** away.

Graph Convolution Network (Kipf and Welling 2017)

- Same as the Chebyshev spectral convolution with $\lambda_{\max} = 2$, $k = 1$ and $\theta := \theta_0 = -\theta_1$.

$$\mathbf{v} * \mathbf{g} = \theta(2I - \hat{L})\mathbf{v}$$

- In practice, we do

$$\mathbf{v} * \mathbf{g} = \theta(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2})\mathbf{v}$$

with $\tilde{A} := A + I$ and \tilde{D} its degree matrix.

- The Graph Convolution Network (GCN) allows for multi-channel convolution by matrix-matrix multiplication

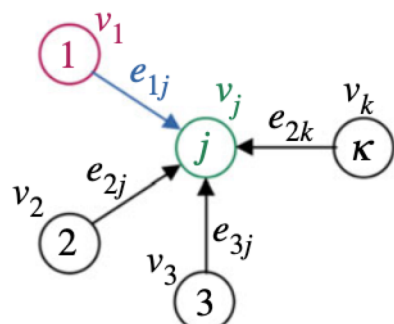
$$\mathbf{V} \leftarrow (\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2})\mathbf{V}\Theta$$

$$\mathbf{V} \in \mathbb{R}^{|V| \times C_{in}}$$

$$\Theta \in \mathbb{R}^{C_{in} \times C_{out}}$$

Graph Convolution Network (Kipf and Welling 2017)

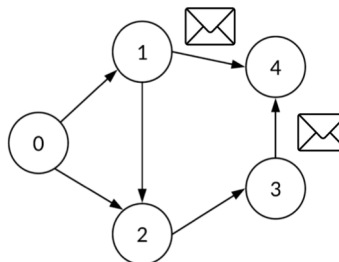
- The GCN spectral graph convolutions can also be written as



$$\mathbf{v}_j \leftarrow \frac{1}{\deg(j)} \Theta^\top \mathbf{v}_j + \sum_{i \in \mathcal{N}(j)} \underbrace{\frac{1}{\sqrt{\deg(i)} \sqrt{\deg(j)}}}_{\text{Message}} \underbrace{\left(\Theta^\top \mathbf{v}_i \right)}_{\text{Aggregation}}$$

Node update

- This can be understood as a form of spatial graph convolution (or message passing).



Spatial based convolution

- Similar to CNNs, spatial graph convolutions rely on the aggregation of **messages** sent from *adjacent* nodes.

In a CNN:

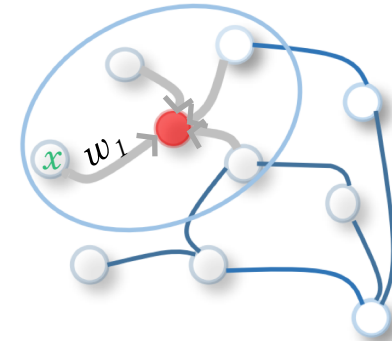
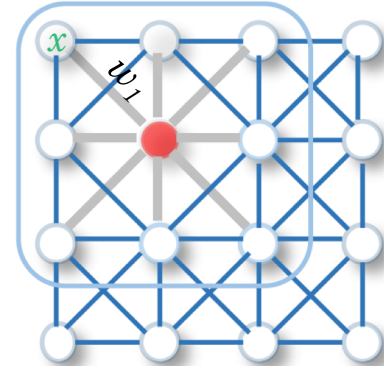
1st Compute the message for each adjacent pixel:

$$m_i = w_i x_i$$

2nd Aggregate the messages:

$$x \leftarrow \sum_i m_i$$

- Adjacent* does not mean *spatially adjacent*, but with an edge directed towards the node.



Spatial based convolution

- The message can be computed in infinite many ways:

- The node feature multiplied by a weight:

- Learnable weight (e.g., **GCN**)
- Weight computed as a function of some learnable parameters and the sender-node and/or receiver-node and/or edge features (e.g., Graph Attention Network and **SplineConv**).

- The message is a function of some learnable parameters and the sender-node and/or receiver-node and/or edge features (e.g., **Interaction Network**) – we will focus on this.

Examples

$$\frac{1}{\sqrt{\deg(i)} \sqrt{\deg(j)}} \left(\Theta^\top \mathbf{v}_i \right)$$

$$h_\theta(\mathbf{e}_{ij}) \mathbf{v}_i$$

↑
weighted B-Spline tensor
product basis

$$\text{MLP}([\mathbf{e}_{ij} | \mathbf{v}_i | \mathbf{v}_j])$$

- The aggregate messages could also be used in many different ways to update the features of the central node.

Message passing (Gilmer et al., 2017)

- To unify a large (and ever increasing) zoo of spatial based convolutions we use a framework known as **Message Passing (MP)** (*Neural message passing for quantum chemistry* by Gilmer et al., 2017)
- It has three steps:

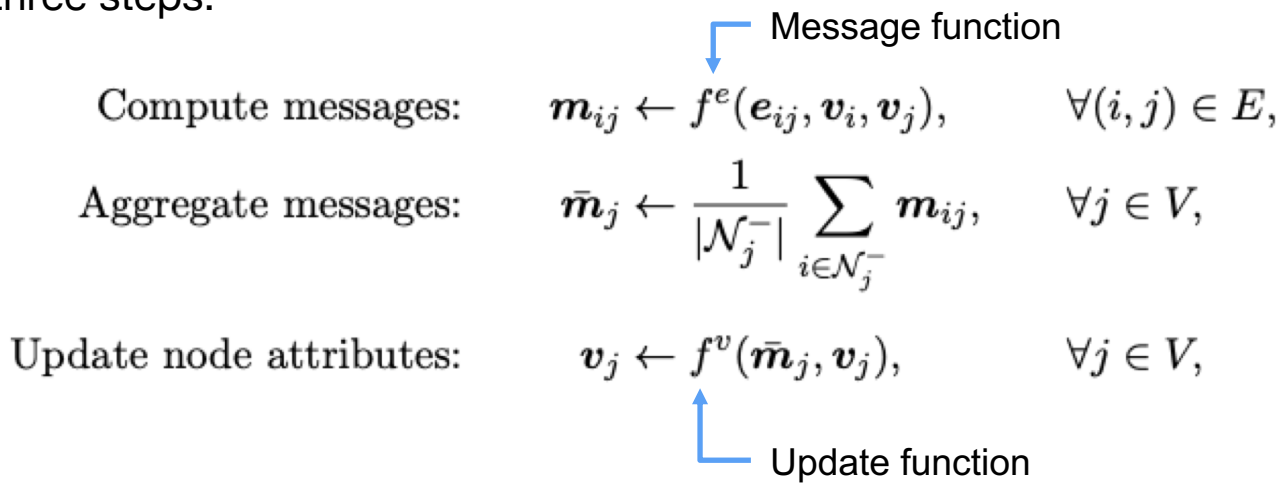


Diagram illustrating the three steps of Message Passing:

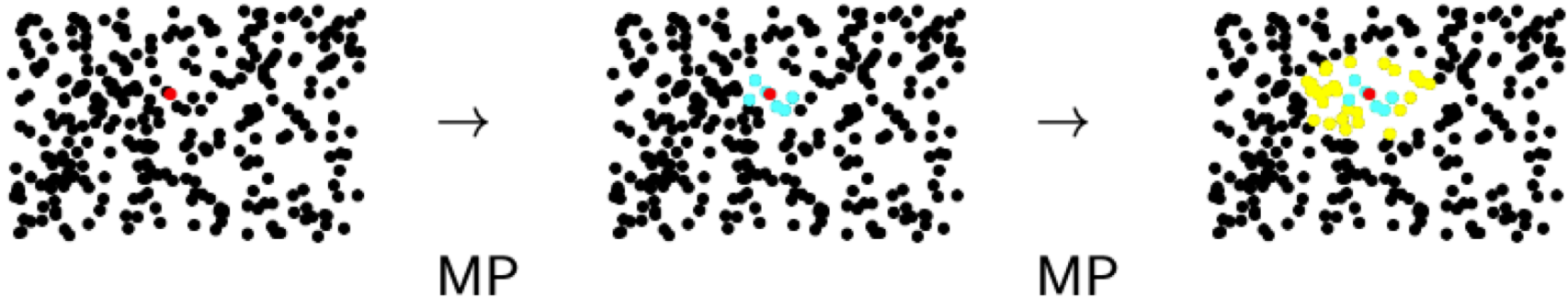
- Compute messages:** $\mathbf{m}_{ij} \leftarrow f^e(\mathbf{e}_{ij}, \mathbf{v}_i, \mathbf{v}_j), \quad \forall (i, j) \in E,$
- Aggregate messages:** $\bar{\mathbf{m}}_j \leftarrow \frac{1}{|\mathcal{N}_j^-|} \sum_{i \in \mathcal{N}_j^-} \mathbf{m}_{ij}, \quad \forall j \in V,$
- Update node attributes:** $\mathbf{v}_j \leftarrow f^v(\bar{\mathbf{m}}_j, \mathbf{v}_j), \quad \forall j \in V,$

Arrows indicate the flow of information: a blue arrow points from the "Message function" label to the "Compute messages" step, and another blue arrow points from the "Update function" label to the "Update node attributes" step.

- The **aggregation function** can be any permutation invariant function, typically **sum** or **mean**.

Message passing (Gilmer et al., 2017)

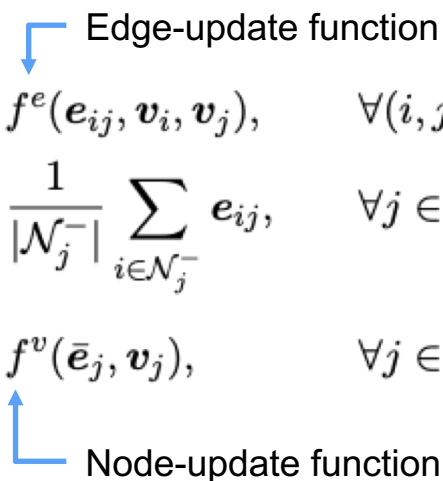
- To process and propagate the node features across the graphs multiple MP layers are applied sequentially:



Message passing (Battaglia et al., 2017)

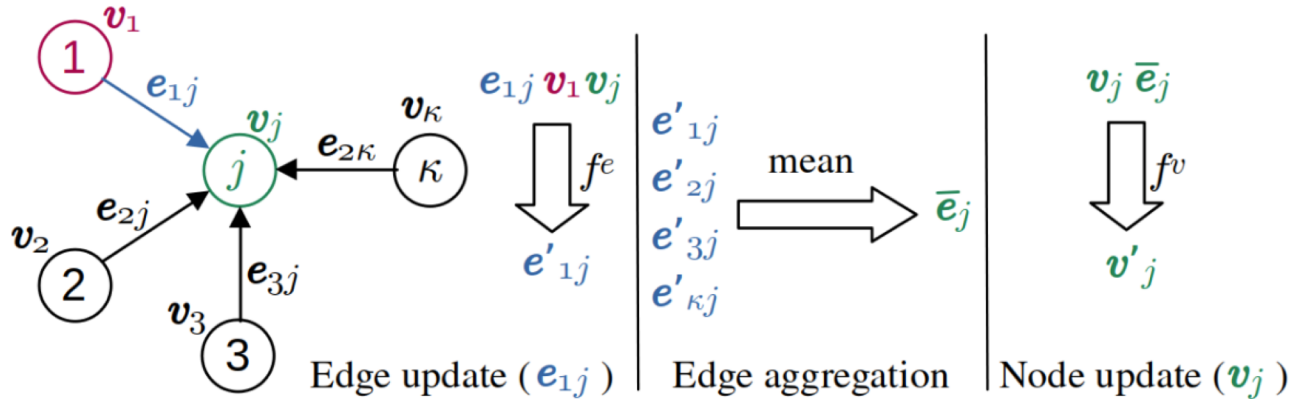
- A more recent version of Message Passing uses the **messages to update the edge attributes** (*Relational inductive biases, deep learning, and graph networks* by Battaglia et al., 2018):

$$\begin{array}{ll}
 \text{Update edge attributes:} & e_{ij} \leftarrow f^e(e_{ij}, v_i, v_j), \quad \forall (i, j) \in E, \\
 \text{Aggregate edge attributes:} & \bar{e}_j \leftarrow \frac{1}{|\mathcal{N}_j^-|} \sum_{i \in \mathcal{N}_j^-} e_{ij}, \quad \forall j \in V, \\
 \text{Update node attributes:} & v_j \leftarrow f^v(\bar{e}_j, v_j), \quad \forall j \in V,
 \end{array}$$



- This helps to learn patterns from the edge attributes.
- It is more expensive to train due to the longest backpropagation chain, but can improve the accuracy.

Message passing (Battaglia et al., 2017)



- Common update functions nowadays are MLPs with a (weighted) skip connection:
 - Edge-update function: $W e_{ij} + \text{MLP}\left(\text{LayerNorm}([e_{ij} | v_i | v_j])\right)$
 - Node-update function: $W v_j + \text{MLP}\left(\text{LayerNorm}([\bar{e}_j | v_j])\right)$

Graph Convolutions – Conclusion

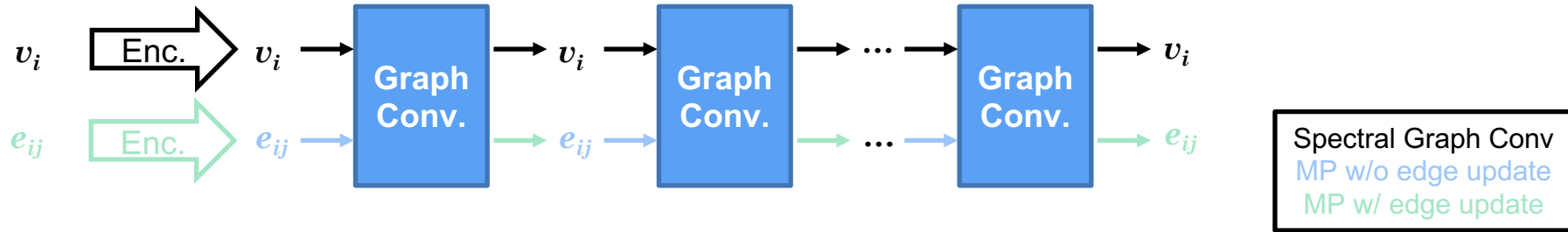
Spatial Graph Conv. (MP)	Spectral Graph Conv.
Directed graphs	Undirected graphs
Node and edge features	Node features
Local information processing	Global or local information processing
$O(E)$	$O(E) - O(V ^3)$

- In DL for physics **we prefer Message Passing** over Spectral Graph Convolutions due to its **efficiency**, ability to work with **different graphs** and **translation invariance** (thanks to assigning the relative position between nodes as input edge features).
- GNN library for pytorch: **PyTorch Geometric**
 - Docs: <https://pytorch-geometric.readthedocs.io>
 - Examples: https://github.com/pyg-team/pytorch_geometric/tree/master/examples



Graph Neural Networks

- Schematic of a typical Graph Neural Network (GNN):



- Depending on the nature of the predictions:
 - Node-level predictions: A decoder (MLP/linear layer) is applied to every v_i .
 - Edge-level predictions: A decoder (MLP/linear layer) is applied to every e_{ij} .
 - Graph-level predictions: (i) A decoder is applied to the flattened node (or edge) features (not permutation invariant), or (ii) a decoder is applied to the aggregated node (or edge) features, or (iii) Graph Pooling is applied.

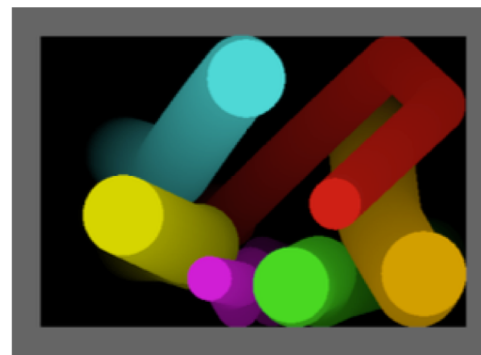
3. Particle-based simulation with GNNs

Particle-based simulation with GNNs

Topics that we will cover:

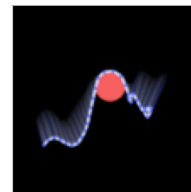
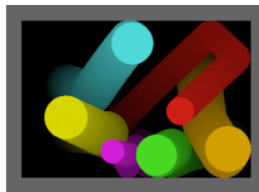
1. Interaction Networks for Learning about Objects, Relations and Physics
2. Learning to Simulate Complex Physics with Graph Networks
3. Lagrangian Fluid Simulation with Continuous Convolutions
4. Inverse Design for Fluid-Structure Interactions using Graph Network Simulators
5. Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics

Interaction Networks for Learning about Objects, Relations and Physics



Interaction Networks for Learning about Physics

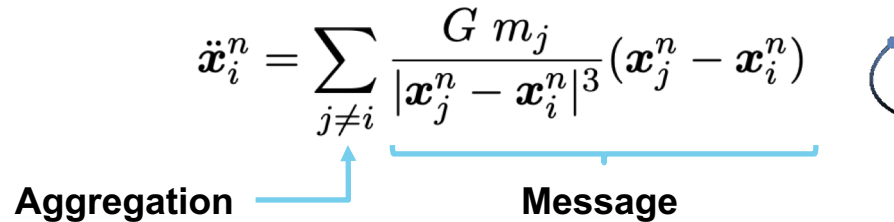
- First work to represent systems of particles as graphs:
 - The nodes represent the particles
 - The edges represent their relationship (e.g., springs, gravitational attraction, distance to collision)



- The message function models the interaction between particles.
- The node-update function models particle dynamics.
- Experimented with: N-body problem, rigid-body collision and discretised string.
- Proved extrapolation to larger systems and thousands of future states.
- Scale to up to 12 particles.

Interaction Networks for Learning about Physics

- Numerical models for particle systems account for **the interactions between particles** and **how these impact the state of each particle**.
- For example, in the the N-body problem:
 - Every particle j attracts each particle i :

$$\ddot{\mathbf{x}}_i^n = \sum_{j \neq i} \underbrace{\frac{G m_j}{|\mathbf{x}_j^n - \mathbf{x}_i^n|^3} (\mathbf{x}_j^n - \mathbf{x}_i^n)}_{\text{Message}} \quad \left(\text{Aggregation} \right)$$




- The position of each particle is updated based on their current state and this interaction (**node update**):

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \dot{\mathbf{x}}_i^n + \Delta t^2 \ddot{\mathbf{x}}_i^n / 2$$


- These dynamics could be modelled by a Message Passing layer.

Interaction Networks for Learning about Physics

Experiments

- N-body problem:
 - $|V| = N$
 - $|E| = |V| \times (|V| - 1)$
 - Input node attributes: velocity and mass
 - Input edge attributes: relative position
 - Training with $|V| = 6$, testing with $|V| = 3, 6$ and 9

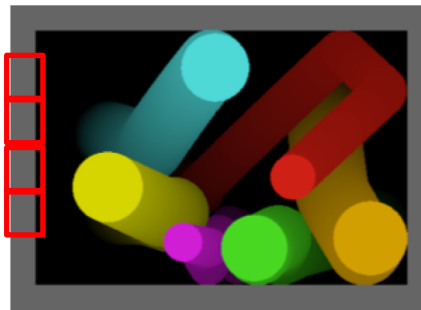


* The inputs are translation invariant

Interaction Networks for Learning about Physics

Experiments

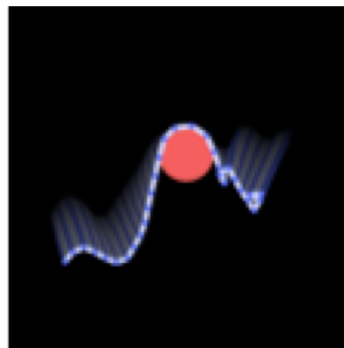
- Bouncing balls:
 - $|V|$ = Number of balls + Number of rectangles discretising the walls
 - $|E| = |V| \times (|V| - 1)$
 - Input node attributes: velocity, inverse of the mass, radius and node type
 - Input edge attributes: relative position and coefficient of restitution
 - Training with 6 balls and testing with 3, 6 and 9 balls



Interaction Networks for Learning about Physics

Experiments

- Discretised string:
 - $|V|$ = Number of nodes on the string and one central ball
 - $|E| = 2 \times (|V| - 2) + 2 (|V| - 1)$ (spring edges + spring-obstacle edges)
 - Input node attributes: velocity, inverse of the mass, gravity and node type
 - Input edge attributes: relative position and relation type



Interaction Networks for Learning about Physics

Experiments

- The output is the velocity at the next time-point, $\dot{\mathbf{x}}_i^{n+1}$
- Trained to minimise the MSE of $\dot{\mathbf{x}}_i^{n+1}$
- This velocity is used to update the position: $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \Delta t \dot{\mathbf{x}}_i^{n+1}$
- Rollouts are produced by updating the input attributes and evaluating iteratively the model.
- The simulations were rolled out for thousands of time-steps during training
- **Message function:** MLP with four hidden layer (150 neurons each) and 50 neurons in the output layer.
- **Node-update function:** MLP with one hidden layer (with 100 neurons) and 2 neurons in the output layer (x and y component of the velocity).

Any baseline to compare with?

GNNs vs MLPs

- MP-based GNNs does not constrain the size of the system to a fix number of nodes.
- MP is permutation invariant. MLPs are not: the feature vectors can be stacked in many different orders. For instance, these three input vectors represent the same system, but nothing guarantees that the velocity predicted for each node is the same in every case.



- MP is translation invariant if we use the relative position between nodes as edge input.
- Note: in MP, the result from the aggregation must be permutation invariant. Thus, the aggregated message cannot be directly computed, for instance, like this:

$$\bar{m}_0 \leftarrow \text{MLP}([v_0|v_1|e_{10}|v_2|e_{20}]) \quad \text{or} \quad \bar{m}_0 \leftarrow \text{MLP}([v_0|v_2|e_{20}|v_1|e_{10}])$$

Interaction Networks for Learning about Physics

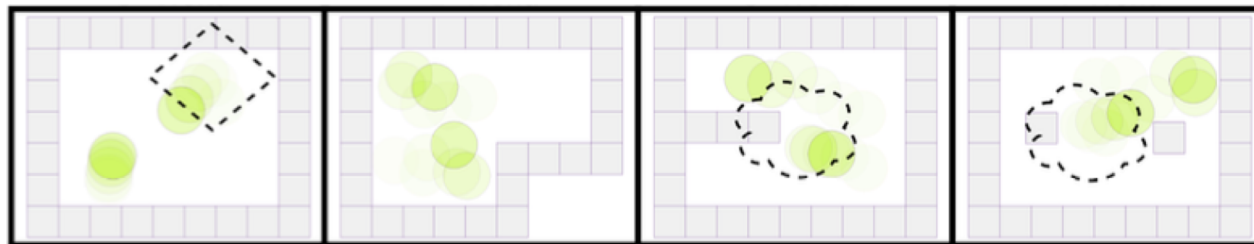
Experiments

- Baseline: MLP which took as input a flattened vector of all of the input data.
- The MLP works only with a fixed number of particles and edges

Domain	Constant velocity	Baseline	Dynamics-only IN	IN
n-body	82	79	76	0.25
Balls	0.074	0.072	0.074	0.0020
String	0.018	0.016	0.017	0.0011

Interaction Networks for Learning about Physics

- Similar work: *A Compositional Object-Based Approach to Learning Physical Dynamics*, Chang et al., 2016.
- They arrived to similar conclusions independently.



Note on Time Predictions

- When evaluating a model autoregressively to produce rollouts the error accumulates and the simulation can diverge in a few time-steps (network evaluations).
- In this work, they mitigated this by adding Gaussian noise to the input positions and velocities (similar to label smoothing LLM).