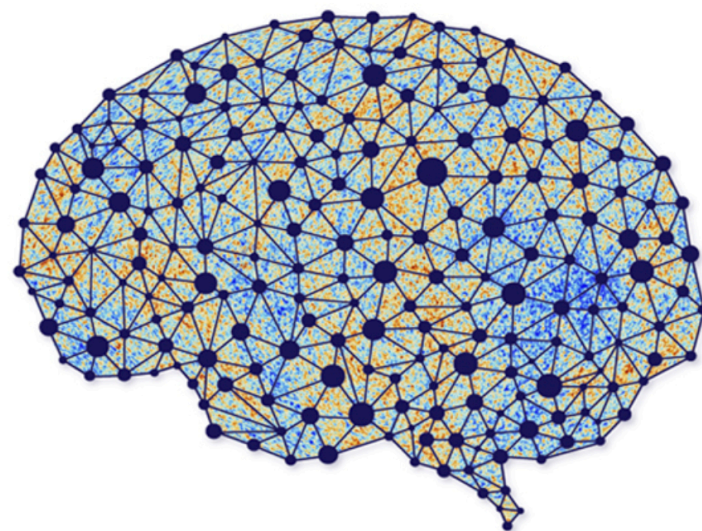


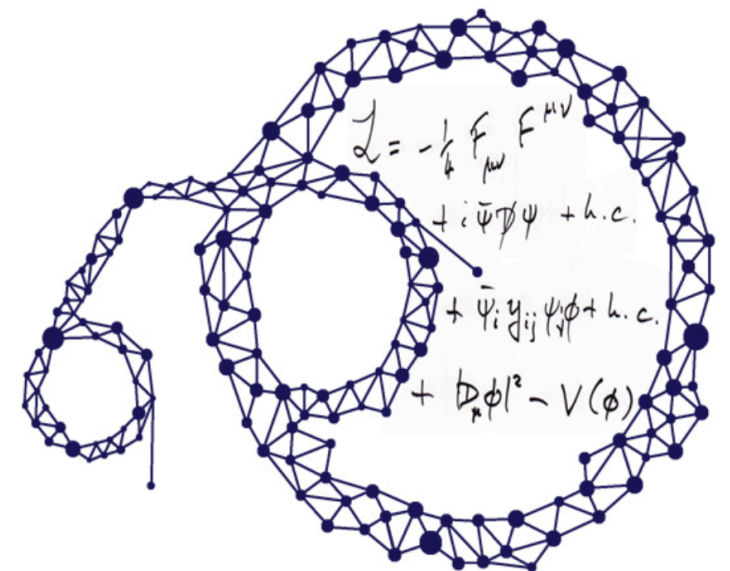
# Physics 361 - Machine Learning in Physics

## Lecture 14 – Graph Neural Networks

March 6<sup>th</sup> 2025



AI  
 $\cap$   
Universe



Moritz Münchmeyer

# Simulation-Based Inference

A quick look into explicit  
inference with  
Differentiable Simulations

# Recall: Implicit vs Explicit inference

- In most situations a simulation does not provide a probability density (likelihood)  $\mathcal{L}(x | \theta)$  of observations given parameters. Such simulations are sometimes called **implicit models**.
- Implicit means that their **likelihood cannot be computed explicitly**, i.e. it is not computationally tractable. We only get samples of the simulation.
- On the other hand, models or simulations that do provide a likelihood are called **explicit models**. Recall for example Gaussian likelihoods.
- A key problem in **explicit inference is to marginalize over the latent variables**, such as the random initial conditions of a simulation.

$$p(x|\theta) = \int dz \, p(x, z|\theta)$$

- For comparison to our study of SBI I now want to discuss a specific example of explicit inference. This approach is also sometimes called “**forward modelling**” or **Bayesian hierarchical modelling**. This is a general approach to many problems that is important to know.

# Explicit inference with probabilistic forward modelling

- Assume that we have a simulator, called the “**forward model**”  $f$ , depending on **parameters**  $\theta$  that describes the evolution of a system starting from a **signal**  $\mathbf{s}$  (e.g. the initial conditions of the forward process):

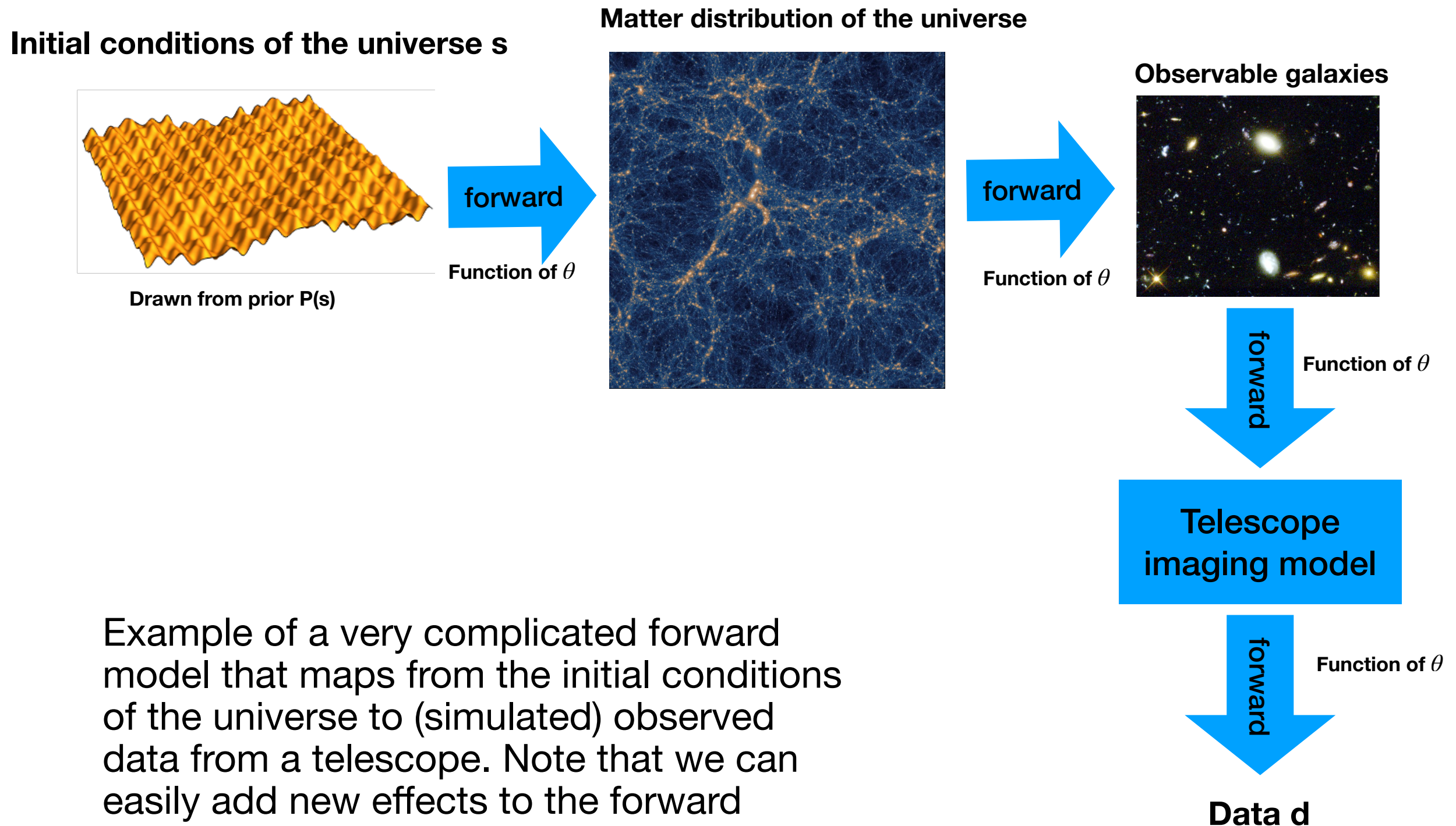
$$f(\mathbf{s}, \Theta)$$

- We want to infer the **parameters**  $\theta$  and the **signal**  $\mathbf{s}$  from **data**  $\mathbf{d}$ , which we take to be a **the forward evolved signal plus some observational noise**.

$$\mathbf{d} = f(\mathbf{s}) + n$$

- This setup is also called an “**inverse problem**”, i.e. we want to reconstruct the signal from the data, assuming that we know the forward model as a function of some parameters. Inverse problems are generally ill-posed (need to regularize).

# Example: Cosmology forward model



Example of a very complicated forward model that maps from the initial conditions of the universe to (simulated) observed data from a telescope. Note that we can easily add new effects to the forward model. Clearly the computational challenge is enormous.

# Example: Cosmology forward model

- Assuming the observational noise is Gaussian we can write an explicit likelihood of the form

$$\log \mathcal{L}(\mathbf{d}|\mathbf{s}, \Theta) = -\frac{1}{2}(\mathbf{f}(\mathbf{s}, \Theta) - \mathbf{d}^{obs})^T \mathbf{N}^{-1}(\mathbf{f}(\mathbf{s}, \Theta) - \mathbf{d}^{obs}) + \text{const.}$$

- To complete the posterior we need to add a prior for the parameters  $\theta$  and  $\mathbf{s}$  which we want to infer.
- Then we need to sample the posterior. Since  $\mathbf{s}$  is usually very high dimensional, normal MCMC will not work.
- However there are sampling methods that work in very high dimensions, in particular **Hamiltonian Monte Carlo (HMC)** and versions of Variational Inference (VI). These **require the forward model to be differentiable**.
- Optimization in very high dimensions requires derivatives to find the minimum. For this reason **differentiable simulations** have become an important topic all over physics.
- More details about this approach in cosmology and references can be found in my cosmology lecture notes.



# Examples: differentiable cosmology simulations

- <https://arxiv.org/abs/2010.11847> FlowPM: Distributed TensorFlow Implementation of the FastPM Cosmological N-body Solver
- <https://arxiv.org/abs/2211.09815> Differentiable Cosmological Simulation with Adjoint Method
- [https://www.youtube.com/watch?v=Epsgh6vr0qs&ab\\_channel=ParticleMeshWithDerivatives](https://www.youtube.com/watch?v=Epsgh6vr0qs&ab_channel=ParticleMeshWithDerivatives)
- <https://arxiv.org/abs/2002.00965> Bayesian de-lensing delight: sampling-based inference of the primordial CMB and gravitational lensing

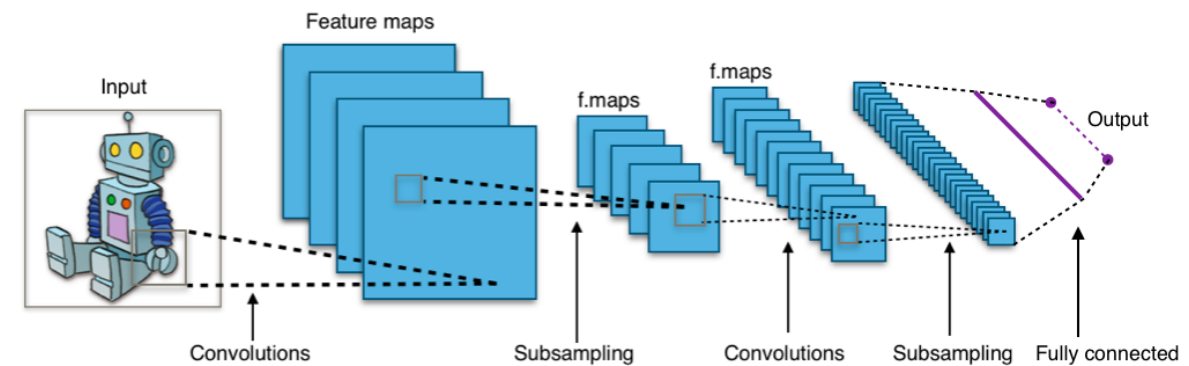
# Graph Neural Networks (GNN)

Graph data



# Data on regular grids

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



**Grids:**

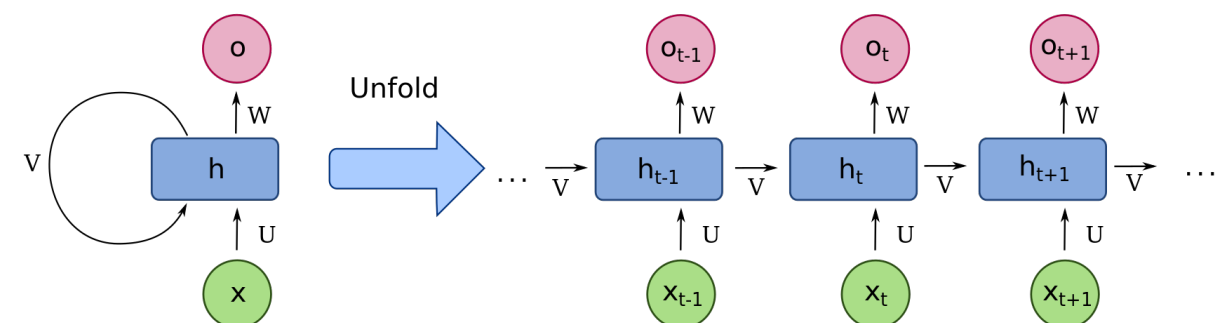
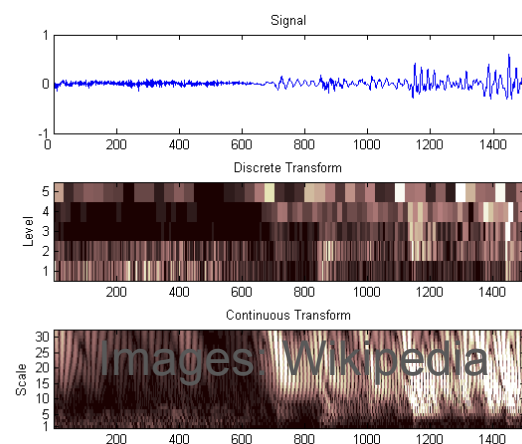
Image: (HxWxC)

Video: (TxHxWxC)

**Time Series (also grids)**

Text: (N)-dim sequence

Speech: (N)-dim sequence



# Data on graphs

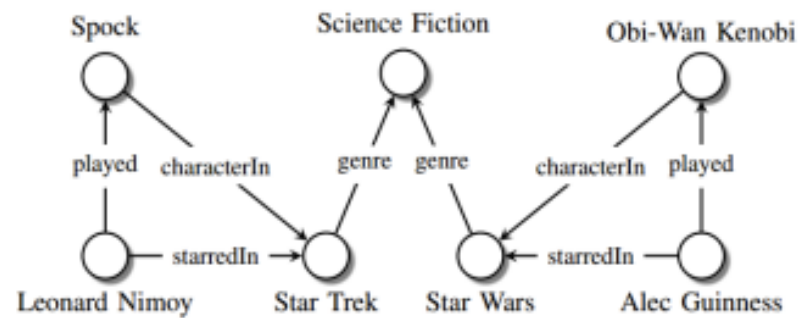


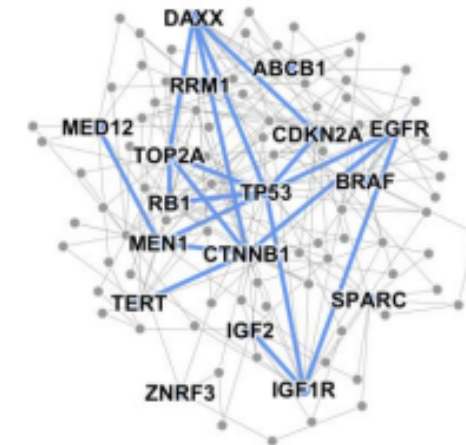
Image credit: [Maximilian Nickel et al](#)

## Knowledge Graphs



Image credit: [SalientNetworks](#)

## Computer Networks



## Disease Pathways

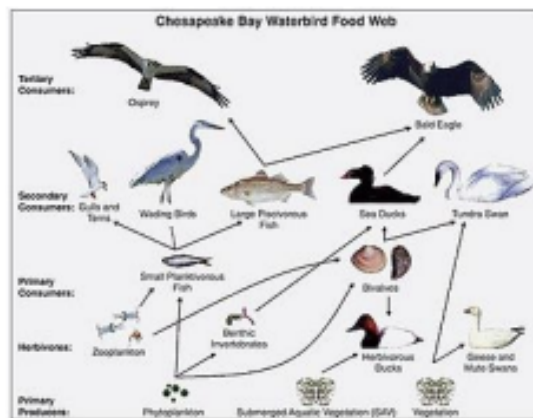


Image credit: [Wikipedia](#)

## Food Webs

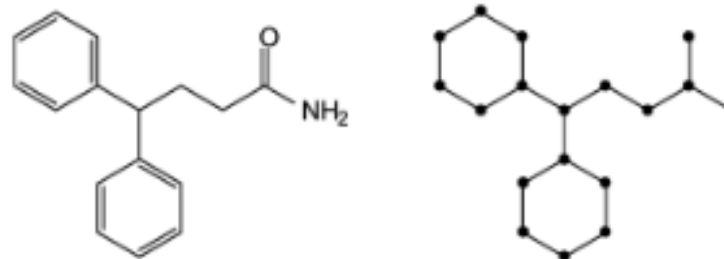


Image credit: [MDPI](#)

## Molecules

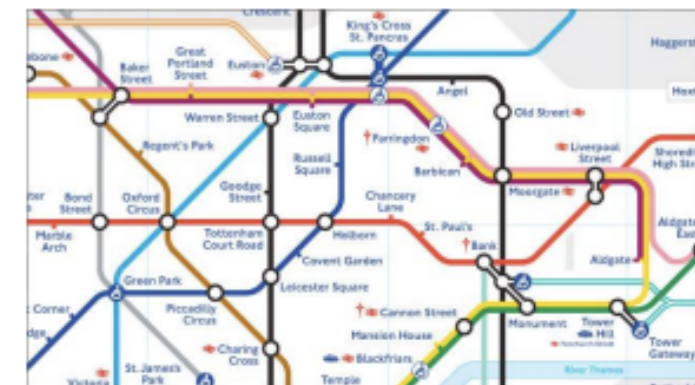


Image credit: [visitlondon.com](#)

## Underground Networks

# Data on graphs



Image credit: [Medium](#)

**Social Networks**

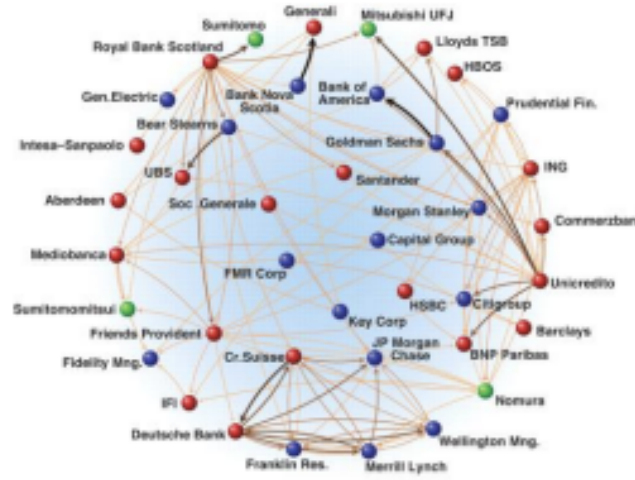


Image credit: [Science](#)

**Economic Networks**



Image credit: [Lumen Learning](#)

**Communication Networks**



**Citation Networks**



Image credit: [Missoula Current News](#)

**Internet**

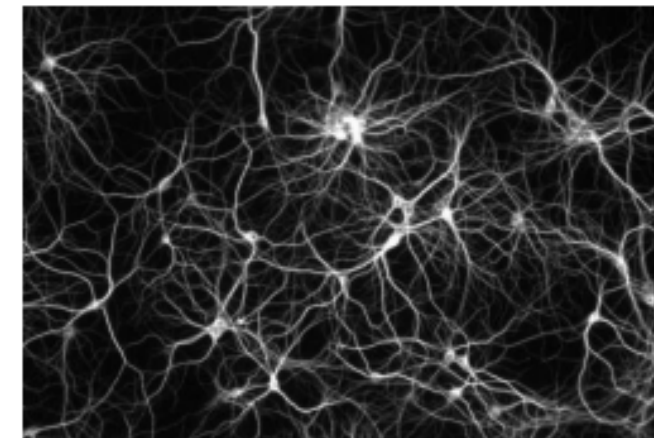


Image credit: [The Conversation](#)

**Networks of Neurons**

# Graphs

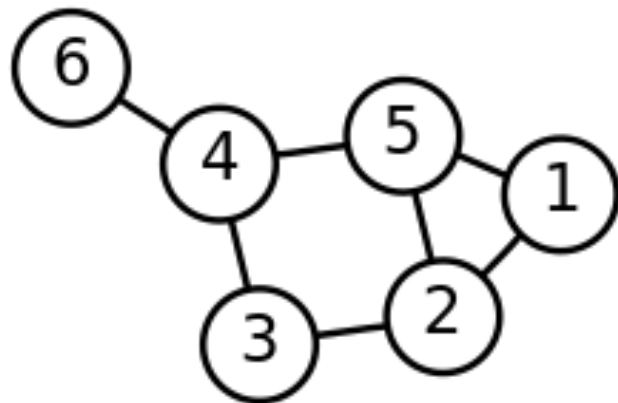
- A graph consists of **nodes** and **edges**.
- It is a very general structure that can represent many different forms of data.
- Examples
  - a regular grid is a specific graph (so you can consider a CNN as a specific subset of GNN).
  - In a molecule, the atoms would be nodes and the bondings (ionic, covalent etc) would be edges.
- A graph can be directed or undirected. For example, in a subway network, it can take a different time to go from A to B than from B to A.



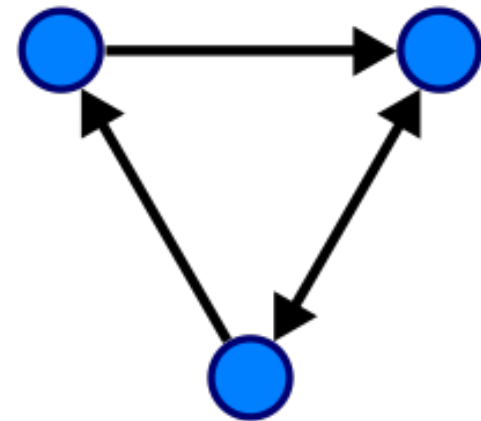
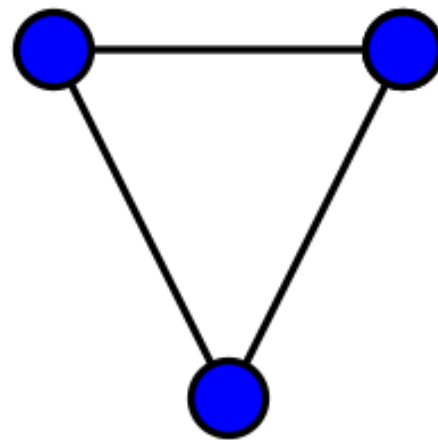
# Definition of a graph

A Graph is a pair  $(V, E)$ , where  $V$  is a set whose elements are called vertices and  $E$  is a set of (un)ordered pairs of vertices  $\{v_1, v_2\}$ , whose elements are called edges.

Occasionally, this definition is being modified to include a general feature of the graph. In that case, graph is defined as a three-tuple  $(V, E, u)$ .



Im. Source:  
Wikipedia

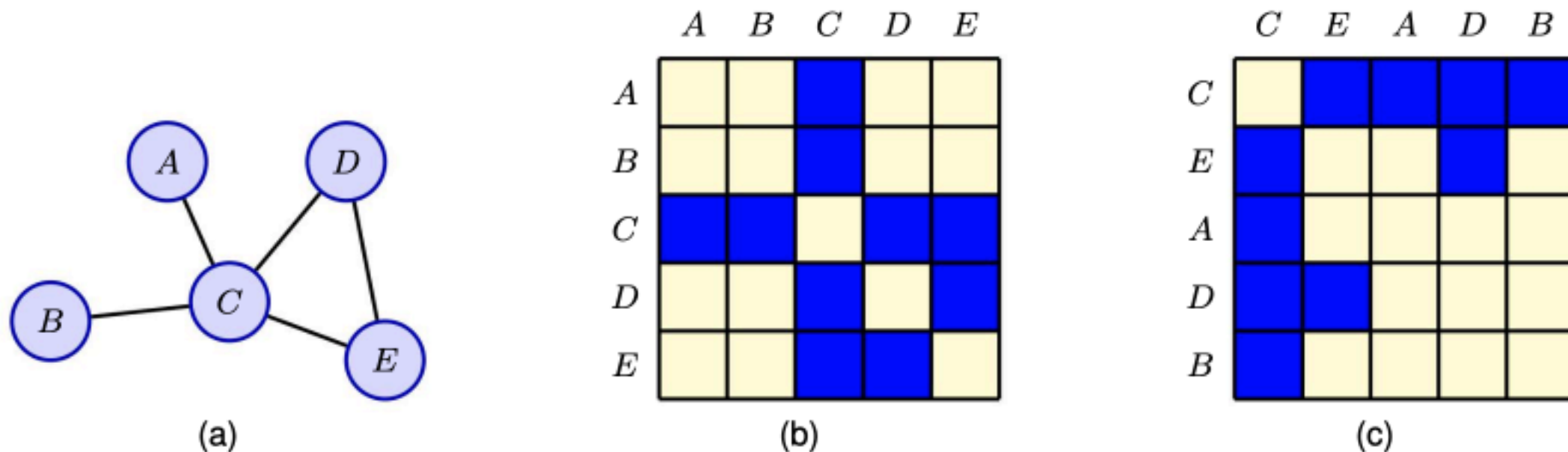


Directed and undirected fully-connected graphs

The edges, nodes, and global features, can all carry information.

# Adjacency matrix

Im. Source: [bishopbook.com](http://bishopbook.com)



**Figure 13.2** An example of an adjacency matrix showing (a) an example of a graph with five nodes, (b) the associated adjacency matrix for a particular choice of node order, and (c) the adjacency matrix corresponding to a different choice for the node order.

- For undirected edges, the adjacency matrix is symmetric.
- The graph neural network should be invariant under re-ordering (permutation) of the nodes. This is an “inductive bias”.

# Learning with graphs

The key objective is to provide a framework to learn (and predict) from graph-represented data.

The main examples are:

- node-level prediction (eg: predict a property of a given node (vertex))
- edge-level prediction (predict the connection bw. two given nodes)
- graph classification (predict a general property of a graph)
- global regression (predict a global property from the graph)
- graph generation

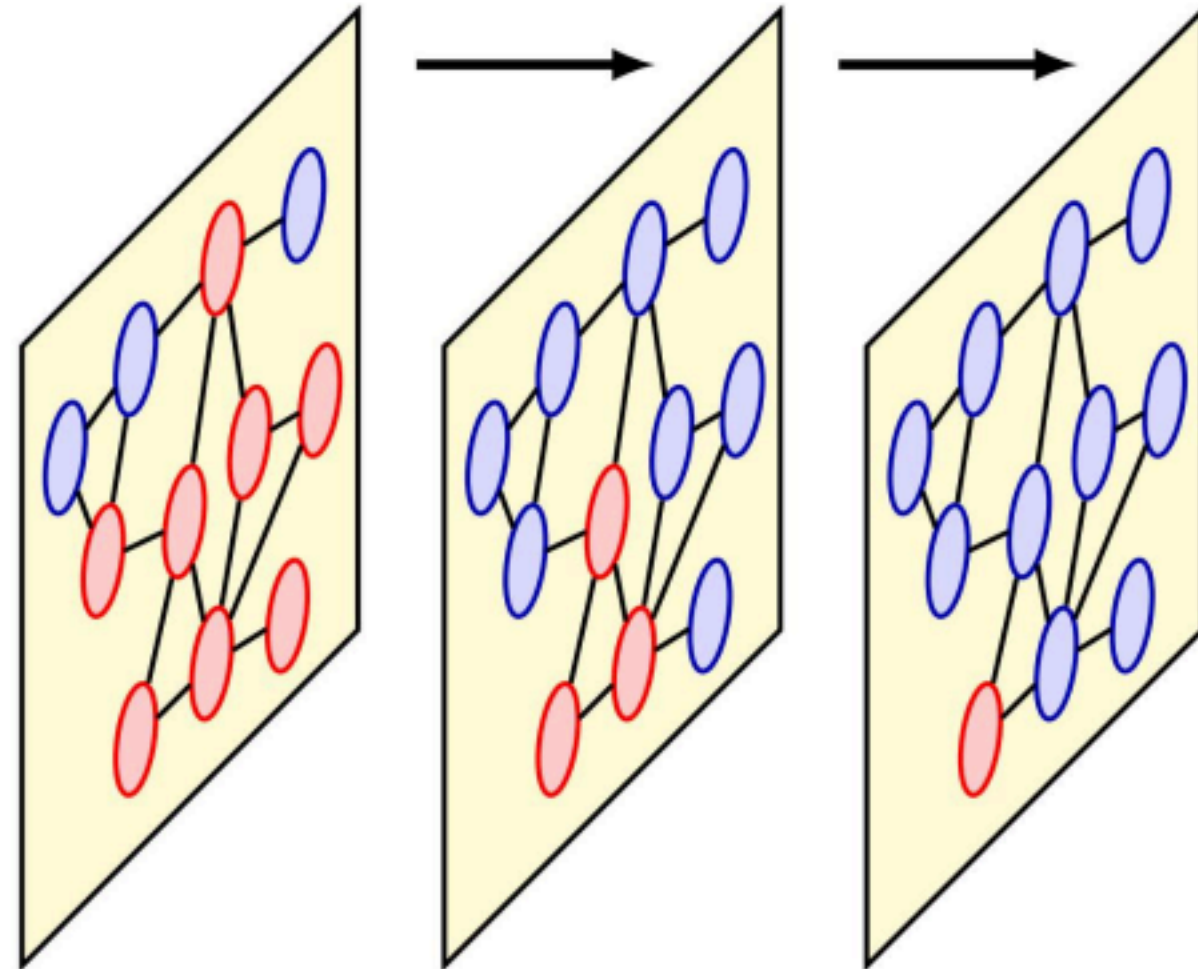


# Graph neural networks

- A GNN **transforms one graph into another graph**, by updating the node, edge and global features. We can stack many such layers to make a deep graph neural network.
- The **edge features, node features and global features** are carrying information in form of a representation (=embedding), which is usually a vector, i.e, a 1-dimensional array of numbers.
- If the goal is to predict a global output (rather than a graph), one usually has **a final aggregation layer** that takes all the information from the output graph and passes it through a fully connected layer.

# Receptive field of GNN

**Figure 13.4** Schematic illustration of information flow through successive layers of a graph neural network. In the third layer a single node is highlighted in red. It receives information from its two neighbours in the previous layer and those in turn receive information from their neighbours in the first layer. As with convolutional neural networks for images, we see that the effective receptive field, corresponding to the number of nodes shown in red, grows with the number of processing layers.



# A simple GNN

We need to construct a method that will be permutation invariant. We do so by aggregating information from neighboring nodes.

## Algorithm 13.1: Simple message-passing neural network

**Input:** Undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$   
Initial node embeddings  $\{\mathbf{h}_n^{(0)} = \mathbf{x}_n\}$   
Aggregate( $\cdot$ ) function  
Update( $\cdot, \cdot$ ) function

**Output:** Final node embeddings  $\{\mathbf{h}_n^{(L)}\}$

// Iterative message-passing

**for**  $l \in \{0, \dots, L - 1\}$  **do**

$\mathbf{z}_n^{(l)} \leftarrow \text{Aggregate} \left( \left\{ \mathbf{h}_m^{(l)} : m \in \mathcal{N}(n) \right\} \right)$

$\mathbf{h}_n^{(l+1)} \leftarrow \text{Update} \left( \mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l)} \right)$

**end for**

**return**  $\{\mathbf{h}_n^{(L)}\}$

$\mathcal{N}$ : nodes that share  
an edge with  
the node  $n$

MLP (fully connected  
 $\mathcal{N}(n)$ )

# Typical aggregation operations

Summation

$$\text{Aggregate} \left( \{ \mathbf{h}_m^{(l)} : m \in \mathcal{N}(n) \} \right) = \sum_{m \in \mathcal{N}(n)} \mathbf{h}_m^{(l)}$$

Average

$$\text{Aggregate} \left( \{ \mathbf{h}_m^{(l)} : m \in \mathcal{N}(n) \} \right) = \frac{1}{|\mathcal{N}(n)|} \sum_{m \in \mathcal{N}(n)} \mathbf{h}_m^{(l)}$$

Often different aggregation operations are combined. For example if we had only average aggregation, counting things becomes difficult.

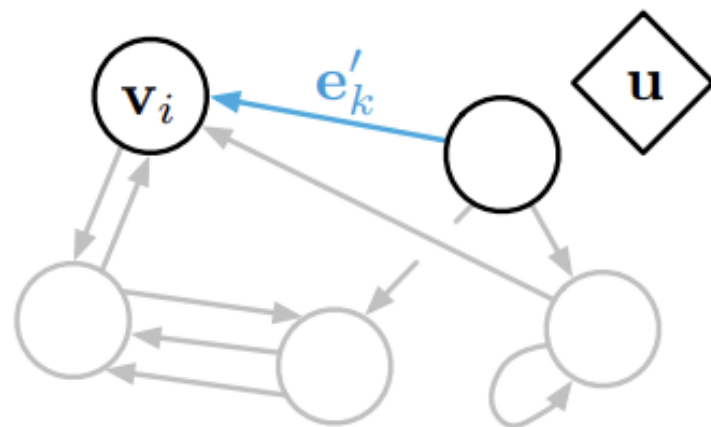
# Message Passing Framework

(Relational inductive biases, deep learning, and graph networks, <https://arxiv.org/abs/1806.01261>)

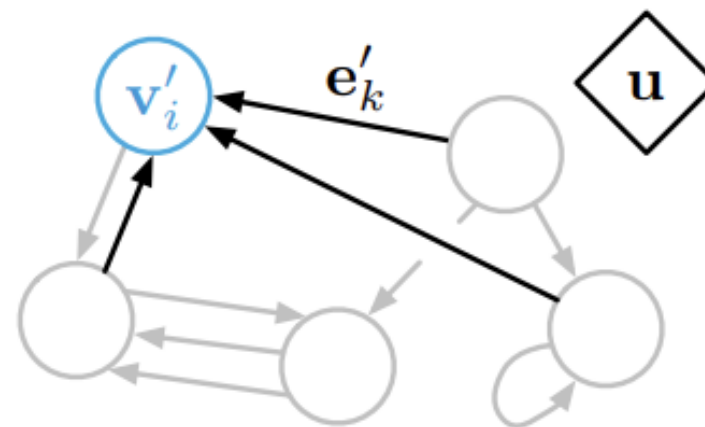
A **Message Passing Graph Neural Nets** is a general class of architectures for learning from graph-represented data.

It is a very general architecture that has become widely used.

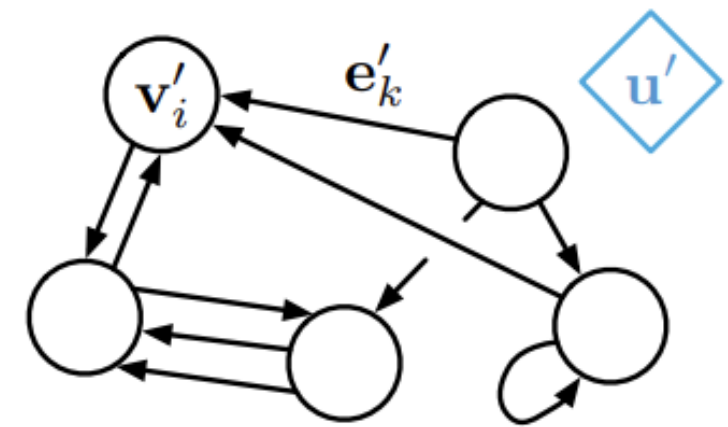
It combines rules for edge, node and global updates in each layer.



(a) Edge update



(b) Node update



(c) Global update

### Algorithm 13.2: Graph neural network with node, edge, and graph embeddings

**Input:** Undirected graph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

Initial node embeddings  $\{\mathbf{h}_n^{(0)}\}$

Initial edge embeddings  $\{\mathbf{e}_{nm}^{(0)}\}$

Initial graph embedding  $\mathbf{g}^{(0)}$

**Output:** Final node embeddings  $\{\mathbf{h}_n^{(L)}\}$

Final edge embeddings  $\{\mathbf{e}_{nm}^{(L)}\}$

Final graph embedding  $\mathbf{g}^{(L)}$

---

```
// Iterative message-passing
for  $l \in \{0, \dots, L-1\}$  do
     $\mathbf{e}_{nm}^{(l+1)} \leftarrow \text{Update}_{\text{edge}}(\mathbf{e}_{nm}^{(l)}, \mathbf{h}_n^{(l)}, \mathbf{h}_m^{(l)}, \mathbf{g}^{(l)})$ 
     $\mathbf{z}_n^{(l+1)} \leftarrow \text{Aggregate}_{\text{node}}(\{\mathbf{e}_{nm}^{(l+1)} : m \in \mathcal{N}(n)\})$ 
     $\mathbf{h}_n^{(l+1)} \leftarrow \text{Update}_{\text{node}}(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}, \mathbf{g}^{(l)})$ 
     $\mathbf{g}^{(l+1)} \leftarrow \text{Update}_{\text{graph}}(\mathbf{g}^{(l)}, \{\mathbf{h}_n^{(l+1)}\}, \{\mathbf{e}_{nm}^{(l+1)}\})$ 
end for
return  $\{\mathbf{h}_n^{(L)}\}, \{\mathbf{e}_{nm}^{(L)}\}, \mathbf{g}^{(L)}$ 
```

$\vec{e}_{nm}$ : edge between node  $n$  and  $m$ .

$\vec{e}_{nm}$  is an embedding = a vector of information

$\vec{h}_n$ : node  $n$  embedding

$\vec{g}$ : global embedding

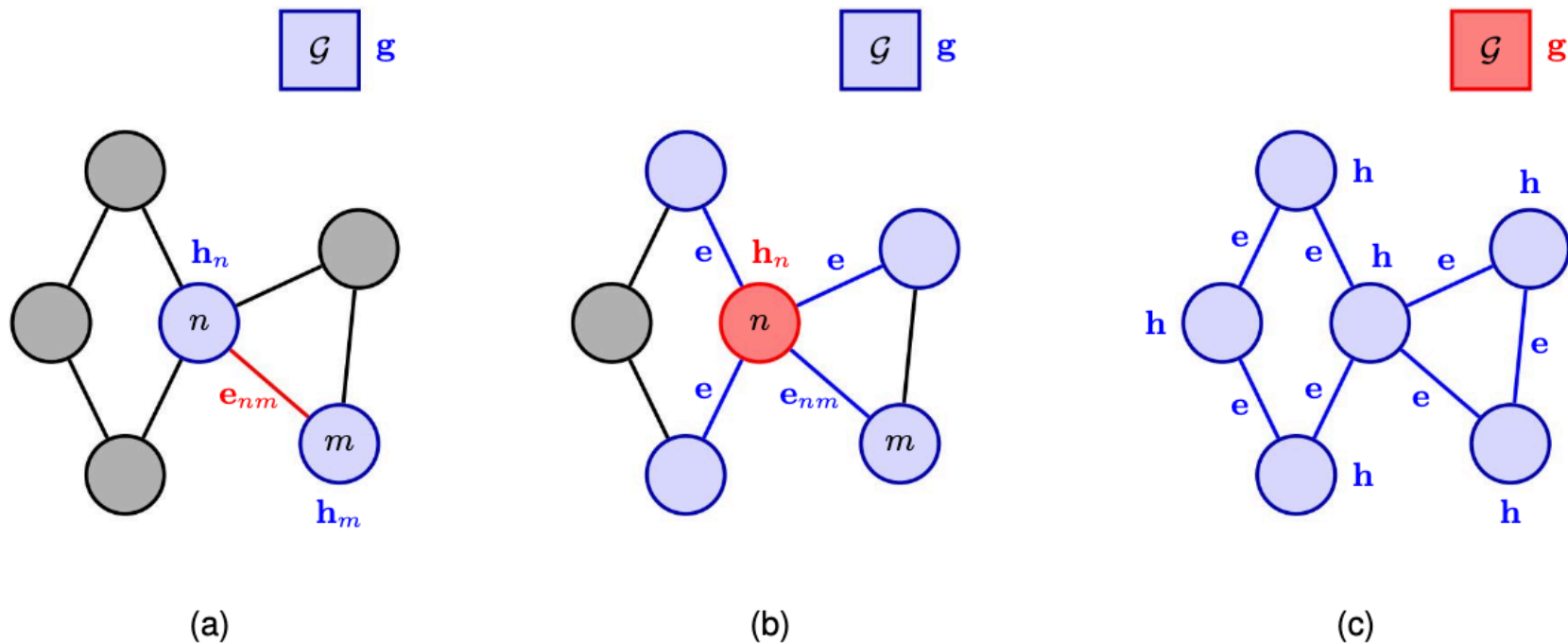


$$\mathbf{e}_{nm}^{(l+1)} = \text{Update}_{\text{edge}}(\mathbf{e}_{nm}^{(l)}, \mathbf{h}_n^{(l)}, \mathbf{h}_m^{(l)}, \mathbf{g}^{(l)}) \quad (13.32)$$

$$\mathbf{z}_n^{(l+1)} = \text{Aggregate}_{\text{node}}(\{\mathbf{e}_{nm}^{(l+1)} : m \in \mathcal{N}(n)\}) \quad (13.33)$$

$$\mathbf{h}_n^{(l+1)} = \text{Update}_{\text{node}}(\mathbf{h}_n^{(l)}, \mathbf{z}_n^{(l+1)}, \mathbf{g}^{(l)}) \quad (13.34)$$

$$\mathbf{g}^{(l+1)} = \text{Update}_{\text{graph}}(\mathbf{g}^{(l)}, \{\mathbf{h}_n^{(l+1)} : n \in \mathcal{V}\}, \{\mathbf{e}_{nm}^{(l+1)} : (n, m) \in \mathcal{E}\}). \quad (13.35)$$



**Figure 13.5** Illustration of the general graph message-passing updates defined by (13.32) to (13.35), showing (a) edge updates, (b) node updates, and (c) global graph updates. In each case the variable being updated is shown in red and the variables that contribute to that update are those shown in red and blue.

The update functions are usually learned by MLPs.



# Using GNN

- The message passing GNN that we have discussed is very general and widely applicable.
- However GNN are not always easy to train. It can take a lot of experimentation to get good results.
- There are also special forms of GNN, which are subsets of what we discussed, but may work better for specific problems.
- For example there are Graph Convolutional Neural Networks which generalize the notion of Convolution to Graphs using spectral graph theory.
- GNN can also be made invariant under specific symmetries. In fact we will explore rotation invariance in the next problem set.

# Code frameworks

tfgnn (TensorFlow)

jraph (Jax)

**PyG** (PyTorch Geometric)



## tensorflow/gnn

TensorFlow GNN is a library to build Graph Neural Networks on the TensorFlow platform.



31

Contributors

17

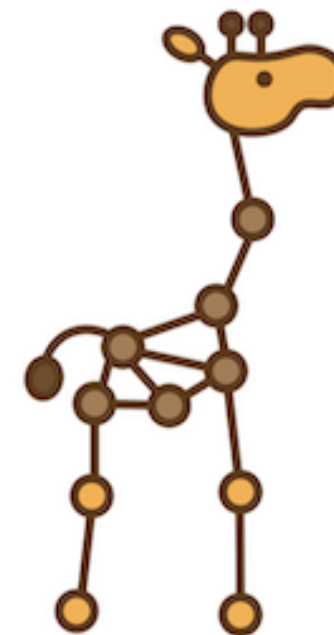
Used by

1k

Stars


163

Forks



# We will use PyG

<https://pytorch-geometric.readthedocs.io/en/latest/>



2.7.0

**INSTALL PYG**

Installation

**GET STARTED**

Introduction by Example

Colab Notebooks and Video Tutorials

**TUTORIALS**

Design of Graph Neural Networks

Working with Graph Datasets

Use-Cases & Applications

Distributed Training

**ADVANCED CONCEPTS**

Advanced Mini-Batching

Memory-Efficient Aggregations

Hierarchical Neighborhood Sampling

Compiled Graph Neural Networks

TorchScript Support



Scaling Up GNNs via Remote Backends

Managing Experiments with GraphGym


CPU Affinity for PyG Workloads

🏠 / PyG Documentation

## PyG Documentation

 **PyG** (*PyTorch Geometric*) is a library built upon  **PyTorch** to easily write and train Graph Neural Networks (GNNs) for a wide range of applications related to structured data.

It consists of various methods for deep learning on graphs and other irregular structures, also known as **geometric deep learning**, from a variety of published papers. In addition, it consists of easy-to-use mini-batch loaders for operating on many small and single giant graphs, **multi GPU-support**, **torch.compile** support, **DataPipe** support, a large number of common benchmark datasets (based on simple interfaces to create your own), and helpful transforms, both for learning on arbitrary graphs as well as on 3D meshes or point clouds.

 Join our Slack community!

### Install PyG

- [Installation](#)

### Get Started

- [Introduction by Example](#)
- [Colab Notebooks and Video Tutorials](#)

### Tutorials

- [Design of Graph Neural Networks](#)
- [Working with Graph Datasets](#)
- [Use-Cases & Applications](#)
- [Distributed Training](#)

### Advanced Concepts

# Examples of GNN in physics

- IceCube particle shower reconstruction
- LHC event reconstruction
- Cosmological parameter estimation from galaxies
- Particle-based simulations



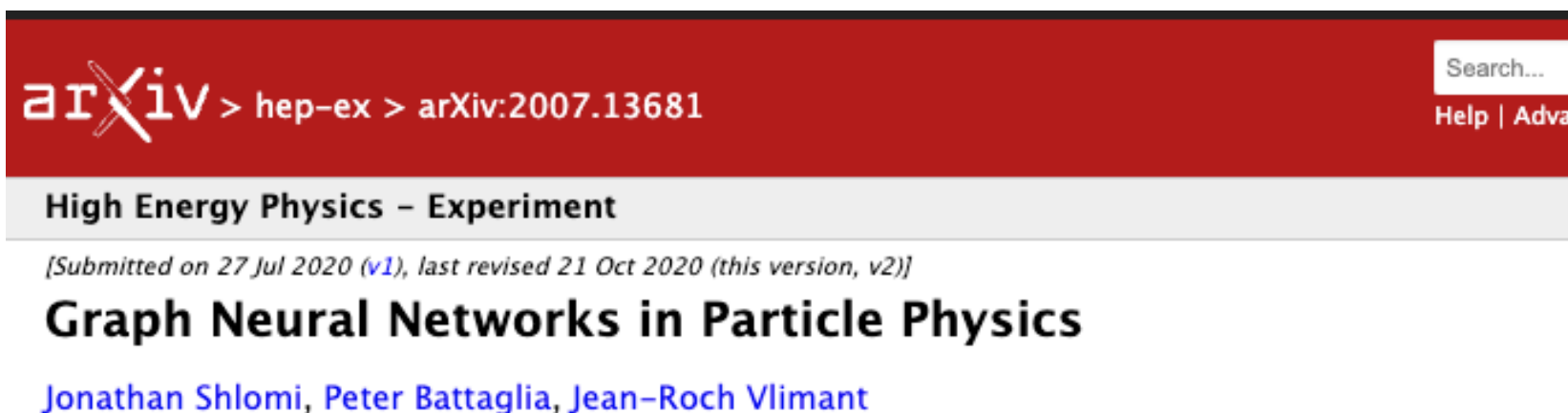
arXiv > hep-ex > arXiv:2209.03042

High Energy Physics – Experiment

*[Submitted on 7 Sep 2022 (v1), last revised 11 Oct 2022 (this version, v3)]*

**Graph Neural Networks for Low-Energy Event Classification & Reconstruction in IceCube**

<https://arxiv.org/abs/2209.03042>



arXiv > hep-ex > arXiv:2007.13681

High Energy Physics – Experiment

*[Submitted on 27 Jul 2020 (v1), last revised 21 Oct 2020 (this version, v2)]*

**Graph Neural Networks in Particle Physics**

[Jonathan Shlomi](#), [Peter Battaglia](#), [Jean-Roch Vlimant](#)

<https://arxiv.org/abs/2007.13681>

arXiv > cs > arXiv:2002.09405

Search...  
Help | Adv

Computer Science > Machine Learning

[Submitted on 21 Feb 2020 (v1), last revised 14 Sep 2020 (this version, v2)]

## Learning to Simulate Complex Physics with Graph Networks

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, Peter W. Battaglia

<https://arxiv.org/abs/2002.09405>

arXiv > astro-ph > arXiv:2411.02496

Search...  
Help | Advan

Astrophysics > Cosmology and Nongalactic Astrophysics

[Submitted on 4 Nov 2024]

## Reconstruction of Continuous Cosmological Fields from Discrete Tracers with Graph Neural Networks

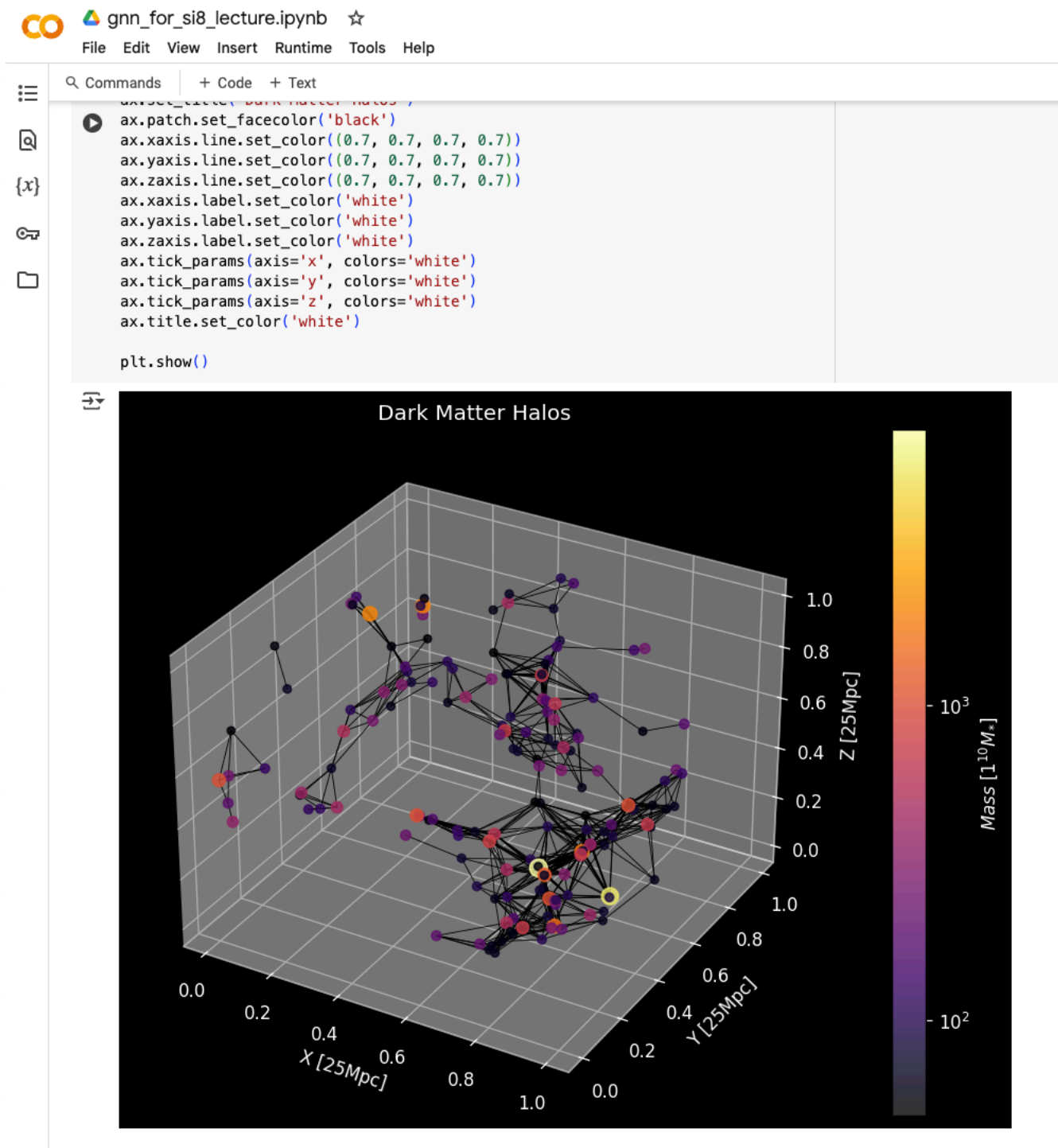
Yurii Kvasiuk, Jordan Krywonos, Matthew C. Johnson, Moritz Münchmeyer

<https://arxiv.org/abs/2411.02496>

As an example from my group

# Our example: GNN for galaxies

The rest of this lecture will be on Colab.



# Course logistics

- **Reading for this lecture:**
  - This lecture was based in part on the book by Bishop, linked on the website.