Graph Neural Networks in Biology: Introduction

Alexander Schönhuth
Luna Pianesi

UNIVERSITÄT BIELEFELD
Faculty of Technology

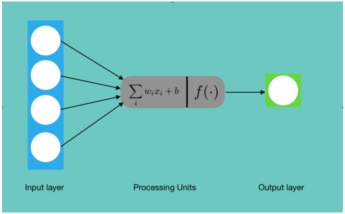
Bielefeld University
April 15, 2025

Graph Neural Networks: Motivation

Neural Networks

NEURONS

LINEAR + ACTIVATION FUNCTION



$$\text{output} = a(w^T \cdot x + b)$$

Note: replace f in Figure by a!

Neuron: linear function followed by activation function

Examples

- Linear regression:

$$a = \text{Id}$$

a is identity function

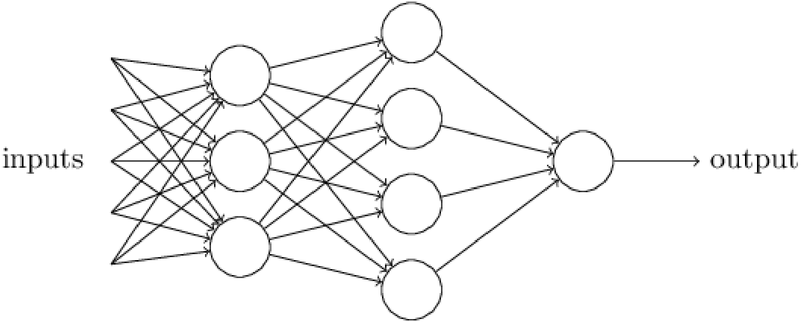
- Perceptron:

$$a(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

a is step function

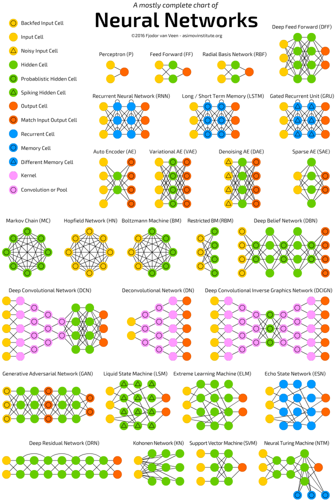
Neural Networks

Concatenating Neurons

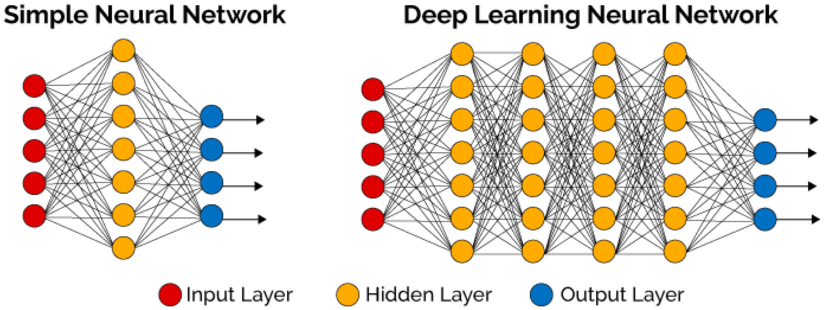


Neural Networks

Architectures (chart from 2016)



Deep Neural Networks



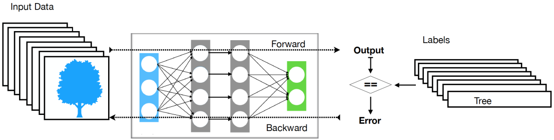
Width = Number of nodes in a hidden layer
Depth = Number of hidden layers
Deep = depth ≥ 8 (for historical reasons)

Neural Networks

Formal Definition

- Let $\mathbf{x}^l \in \mathbb{R}^{d(l)}$ be all outputs from neurons in layer l, where $d(l)$ is the *width* of layer l.
- Let $y \in V$ be the output.
- Let $\mathbf{x} =: \mathbf{x}^0$ be the input.
- Then
$$\mathbf{x}^l = \mathbf{a}^l(\mathbf{W}^{(l)}\mathbf{x}^{l-1} + \mathbf{b}^l)$$
where $\mathbf{a}^l(.) = (a_1^l(.), ..., a_{d(l)}^l(.))$, $\mathbf{W}^{(l)} \in \mathbb{R}^{d(l)\times d(l-1)}$, $\mathbf{b}^l \in \mathbb{R}^{d(l)}$
- The function f representing a neural network with L layers (with depth L) can be written
$$y = f(\mathbf{x}^0) = f^{(L)}(f^{(L-1)}(...(f^{(1)}(\mathbf{x}^{(0)}))...))$$
where $\mathbf{x}^l = f^{(l)}(\mathbf{x}^{l-1}) = \mathbf{a}^{\mathbf{l}}(\mathbf{W}^{(\mathbf{l})}\mathbf{x}^{l-1} + \mathbf{b}^{\mathbf{l}})$

Training: Backpropagation



- *E.g.* let X be a set of images, labels 1 and 0: tree or not
- Let
$$f_{(\mathbf{w},\mathbf{b})} : X \to \{0,1\} \quad \text{and} \quad \hat{f} : X \to \{0,1\}$$
be the network function ($f_{\mathbf{w},\mathbf{b}}$) and the true function ($\hat{f}$)
- $L(f_{(\mathbf{w},\mathbf{b})}, \hat{f})$ loss function, differentiable in network parameters $\mathbf{w}, \mathbf{b}$
- *Back Propagation*: Minimize $L(f, \hat{f})$ through gradient descent
  - ☞ Heavily parallelizable!
- Decisive: Ratio number of parameters and training data

Why Neural Networks?

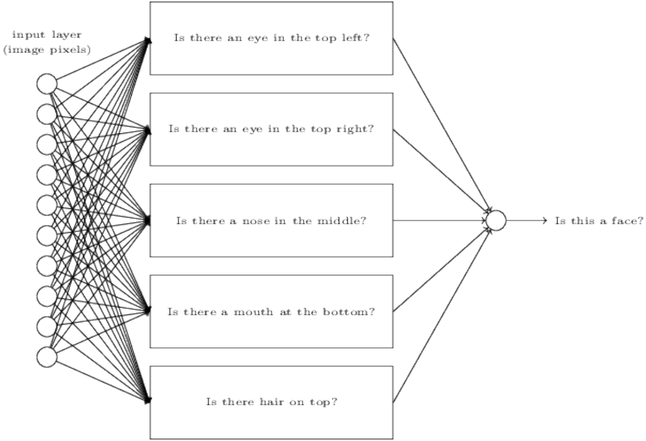
Why Neural Networks?

Given an (unknown) functional relationship $f : \mathbb{R}^d \to V$, why should we learn f by approximating it with a neural network?

Practical, Intuitive Consideration

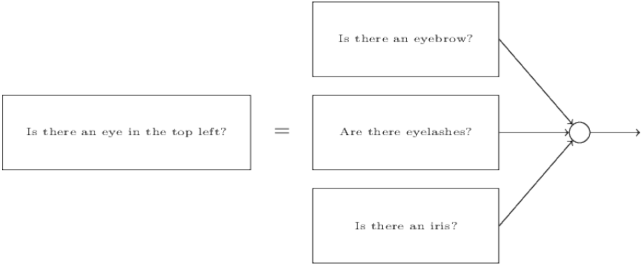
Deep Learning

Intuitive Explanation



- *Face recognition*: decompose classification task into subtasks

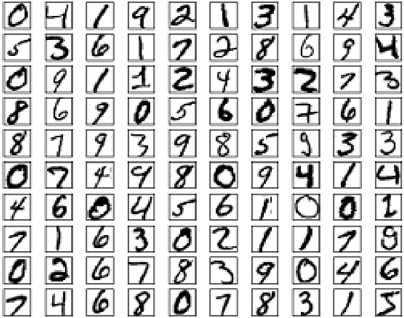
Deep Learning is Intuitive



- *Face recognition*: decompose subtask (eye recognition) into sub-subtasks
- Subtasks are composed into overall task “layer by layer”

Running Example: MNIST Classification

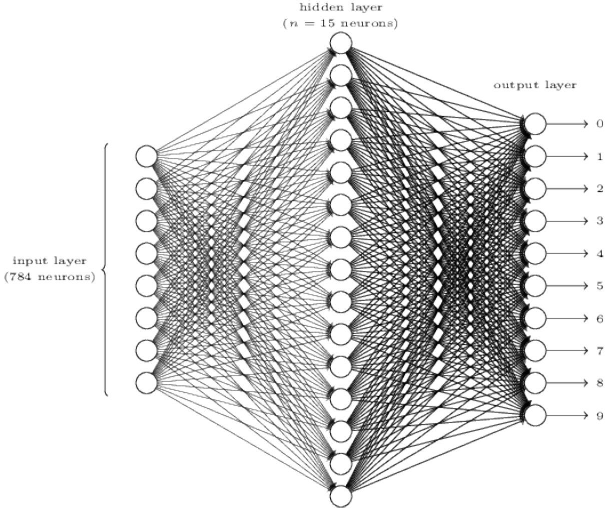
Data, Function



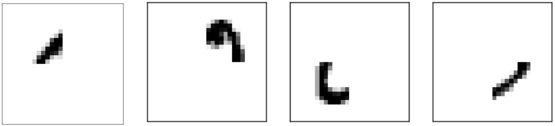
$$f : \mathbb{R}^{28\times 28=784} \longrightarrow \{0, 1, ..., 9\} \tag{1}$$

Running Example

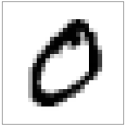
Model Class: NN with 1 hidden layer



Running Example



together makes



Neurons of hidden layer recognize characterizing parts of digit

Theoretical Consideration

The Universal Approximation Theorem

First version formulated by George Cybenko in 1989.

Theorem

A feedforward network with a single hidden layer containing a finite number of neurons can approximate any nonconstant, bounded and continuous function with arbitrary closeness, as long as there are enough hidden nodes.

Why Deep Learning?

Rule of Thumb

One needs approximately

as many training data
as there are parameters

in the class of models

More Layers

Motivation

- We save on neurons/parameters, while increasing number of steps, by increasing depth!

If you are curious about a working example: watch Lecture 02 by Prof. Schönhuth here `https://gds.techfak.uni-bielefeld.de/teaching/2022winter/bioadl`

Why Deep Learning

- We need only $O(n+1)$ (and not $O(2n)$) parameters to model a constellation with $2n$ steps and one symmetry axis
- Hence, we only need $O(n+1)$ training data, and not $O(2n)$ (like SVM or Nearest Neighbour)
- In general $O(n^l)$ (symmetric) steps need only $O(nl)$ training data
- This illustrates why deeper NNs can deal with symmetry invariance in images

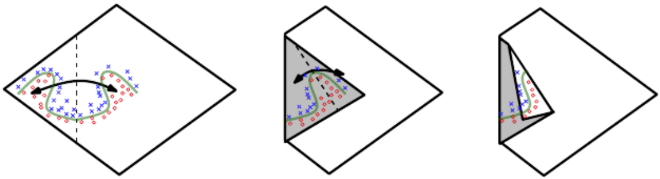
WHY DEEP LEARNING

Theorem (Universal Approximation; Montufar (2014))

Let f be an NN with d inputs, l hidden layers (depth l) of width n each. Then the number of differently labeled regions is

$$O\left(\binom{n}{d}^{d(l-1)} n^d\right) \tag{2}$$

That is, the number of regions that can receive different labels is exponential in the depth (the number of hidden layers) l.



[Montufar 2014]: Every neuron can fold space along an axis

Deep Learning

Assumptions

- Model classes make certain assumptions about properties of the functions they aim to approximate
- Many model classes (such as Nearest Neighbors and SVM's) require *local consistency* and *smoothness*: nearby points are likely to receive the same label
- Deep neural networks make further assumptions such as invariance to shifts, rotations and mirroring

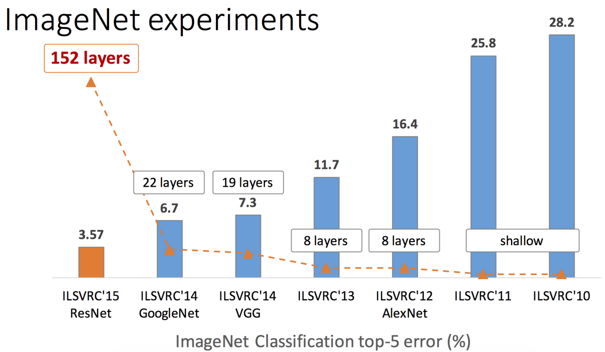
ImageNet and ILSVRC

Dataset and First Results

ImageNet examples: "beading plane", "brown root rot fungus", "scalded milk", "common roundworm"

- *ImageNet dataset*: 16 million full color images; 20 000 categories
- *Starting point*: Le, Ranzato, Monga, Devin, Chen, Corrado, Dean & Ng: "Building high-level features using large scale unsupervised learning", 2012, `https://ai.google/research/pubs/pub38115` achieved 15.3 % test accuracy
- *ILSVRC*: Image-Net Large-Scale Visual Recognition Challenge
  - *2012*: 1000 categories; Training 1.2 million images; Validation 50 000 images; Test 150 000 images

Going Deeper

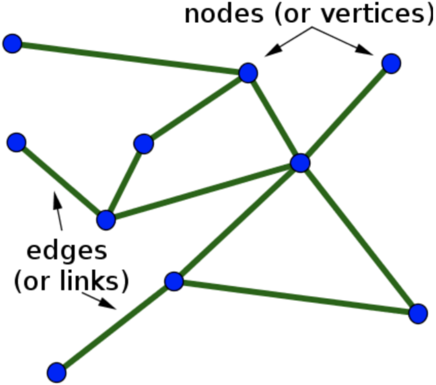


`https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf`; Note: correct error rate for AlexNet is 15.4%

Graph Neural Networks: Introduction

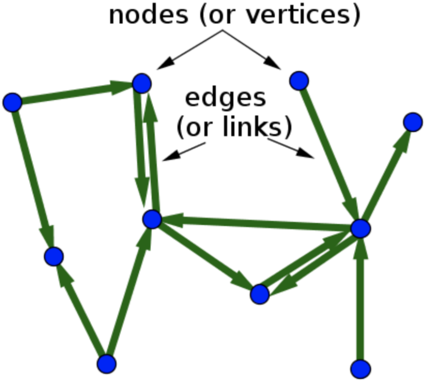
Graphs

Graphs: Introduction



From https://mathinsight.org/network_introduction

Directed Graph



From `https://mathinsight.org/network_introduction`

GRAPHS, ADJACENCY MATRIX: DEFINITION

DEFINITION [GRAPH]:

A graph $G = (V, E)$ has vertices V and edges $E \subset V \times V$. If G is *directed*, the order $(i, j) := (v_i, v_j) \in E$ matters (and edges are often referred to as *arcs*). If G is undirected, (i, j) can be considered unordered, so $(i, j) = (j, i)$.

DEFINITION [ADJACENCY MATRIX]:

Let $G = (V, E)$ be a graph with vertices V and (directed) edges E. The *adjacency matrix* $A = (a_{ij})_{1 \leq i,j \leq |V|}$ is defined by

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

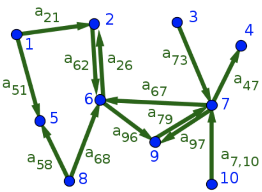
Remark: If G is undirected, $a_{ij} = 1$ implies $a_{ji} = 1$. Hence A is symmetric.

Adjacency Matrix: Example

Definition [Adjacency Matrix]:

Let $G = (V, E)$ be a graph with vertices V and (directed) edges E. The *adjacency matrix* $A = (a_{ij})_{1 \leq i,j \leq |V|}$ is defined by

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4}$$



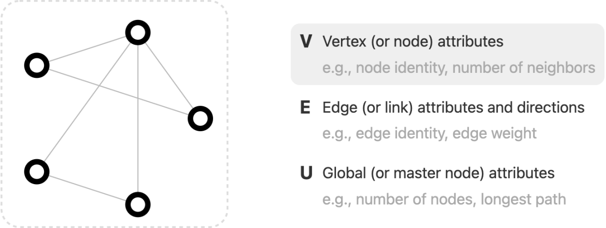
$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

From `https://mathinsight.org/network_introduction`

Graphs: Storing Information

Graphs: Storing Information I

Graphs can store information in various ways

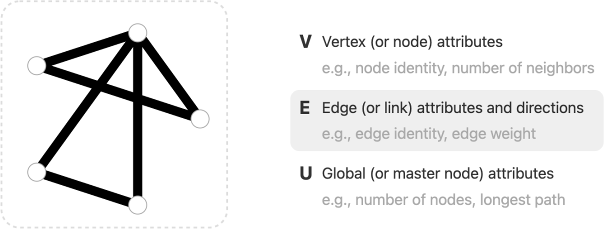


Vertex attributes

From `https://distill.pub/2021/gnn-intro/`

Graphs: Storing Information II

Graphs can store information in various ways

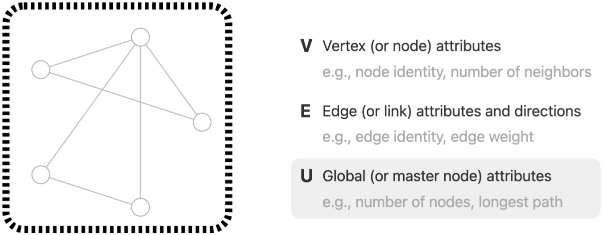


Edge attributes

From `https://distill.pub/2021/gnn-intro/`

GRAPHS: STORINGINFORMATION III

Graphs can store information in various ways

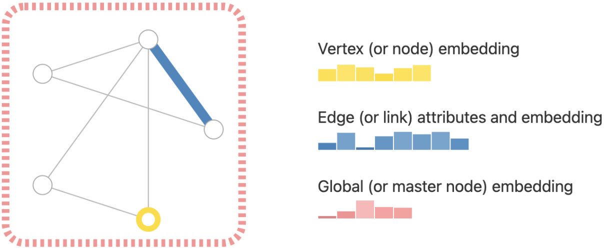


Global attributes

From `https://distill.pub/2021/gnn-intro/`

Graphs: StoringInformation IV

Graphs can store information in various ways

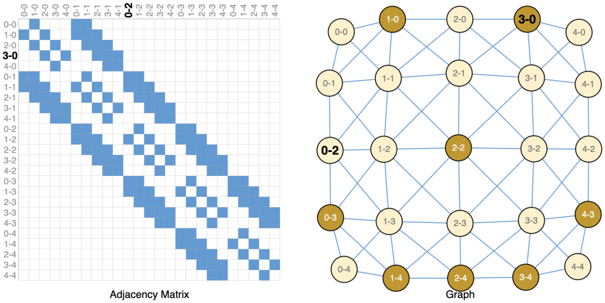


Embeddings: vector-valued information

From `https://distill.pub/2021/gnn-intro/`

Graphs: Examples

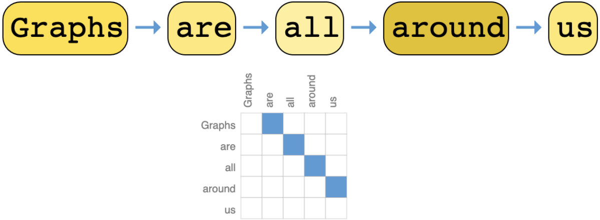
Graphs: Images



Graph and adjacency matrix of an image

From `https://distill.pub/2021/gnn-intro/`

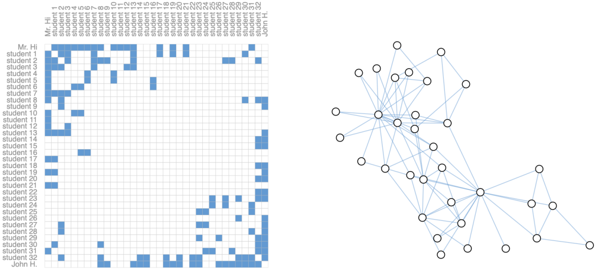
Graphs: Texts



Graph and adjacency matrix of a piece of text

From `https://distill.pub/2021/gnn-intro/`

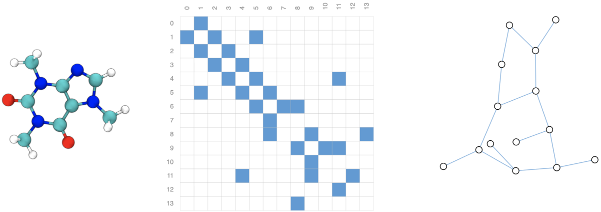
Graphs: Social Networks



Graph and adjacency matrix displaying interactions in karate club

From `https://distill.pub/2021/gnn-intro/`

Graphs: Molecules

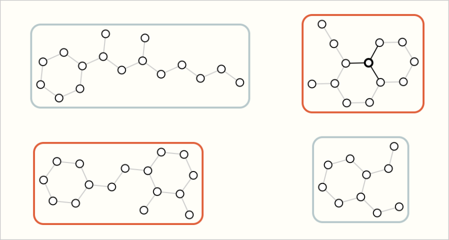


Graph and adjacency matrix of a molecule

From `https://distill.pub/2021/gnn-intro/`

Graphs: Learning Tasks

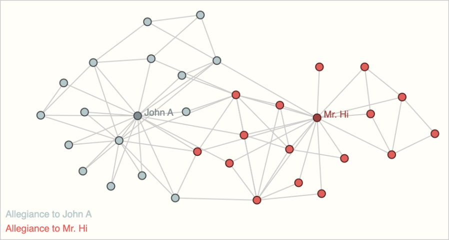
Graph Level Tasks



Structures in molecule graphs. Two rings (red) or not (black).
From `https://distill.pub/2021/gnn-intro/`

- Labels reflect statements about the entire graph.
- If unknown, determine using machine learning.

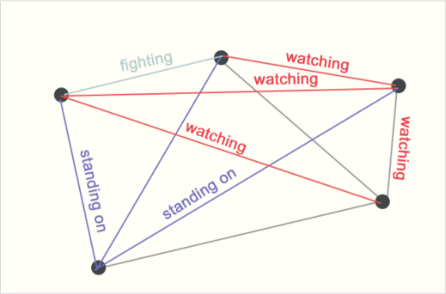
Node Level Tasks



Karate club: Allegiance to either Mr. Hi (red) or John A. (gray)
From `https://distill.pub/2021/gnn-intro/`

- Labels reflect statements about individual nodes.
- Some may be known. Others not: determine using ML.

Edge Level Tasks



Fight scene in image: elements (two fighters, arbiter, audience, mat).
Labels: relationships.
From `https://distill.pub/2021/gnn-intro/`

- Labels reflect statements about edges, so indicate relationships.
- Some relationships known. If not known: determine using ML.

Graphs: Machine Learning Challenges

Neural Networks and Graphs

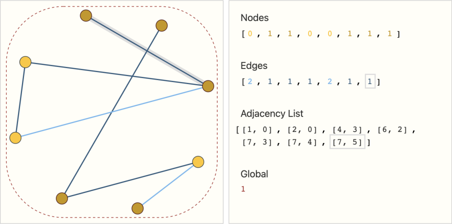
- Techniques for certain graphs available:
  - *Images = Grids:* Convolutional neural networks
  - *Text = Sequences:* Recurrent neural networks, attention networks
- Techniques for arbitrary graphs desirable:
  - *Social networks:* vary (heavily) by application
  - *Molecules:* plenty of different structures
  - *Other applications:* manifold interaction networks
- *Motivation:* Extend existing techniques to general graphs
- *Issue:* Get rid of regularity as a necessary condition

General Graphs: Input

- Neural networks usually expect well-arranged input:
  - Rectangular, grid-like input
  - Sequence type input
  - Arrangement in terms of graph-type evaluation obvious
- Graphs may harbor four types of information:
  - Node information
  - Edge information
  - Global information
  - Connectivity

  How to exploit them by appropriately arranging input?

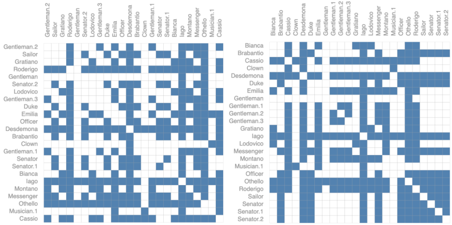
Challenge: Representing Input



Suitable way of storing graph information. Colors: different information.
From `https://distill.pub/2021/gnn-intro/`

- Nodes: node information
- Edges: edge information
- Global: global information
- Adjacency List: connectivity information

Challenge: Permutation Invariance



.

From `https://distill.pub/2021/gnn-intro/`

- Graphs are permutation invariant
- *Goal:* Exploit data in permutation invariant way

Thanks for your attention!