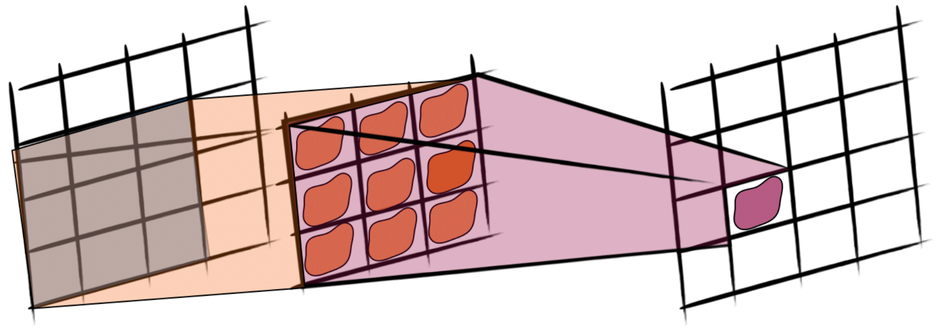
Modern Deep Learning in Physics

Sommersemester 2020/23

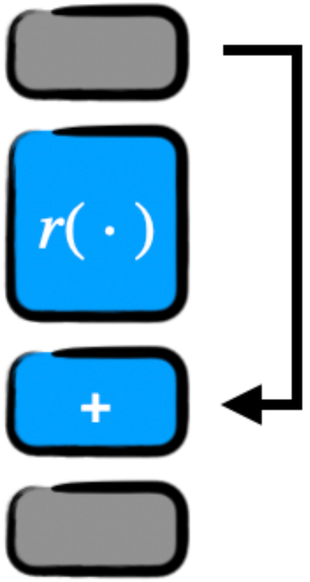
Lecture 8

Recap from Last Week

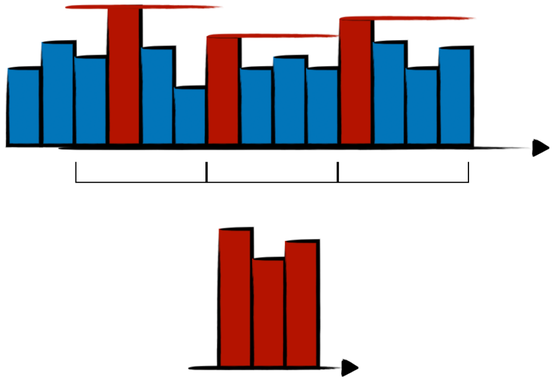
Convolutions



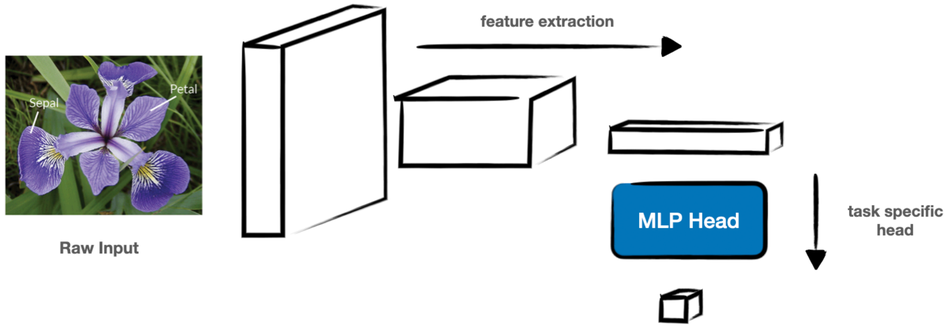
Residual Connections



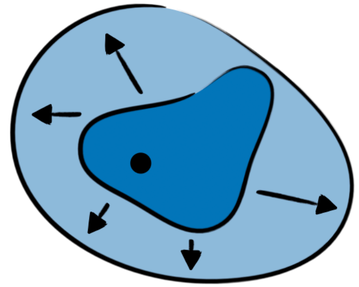
Translational Equi- & Invariance



ConvNets



Nested Function Spaces

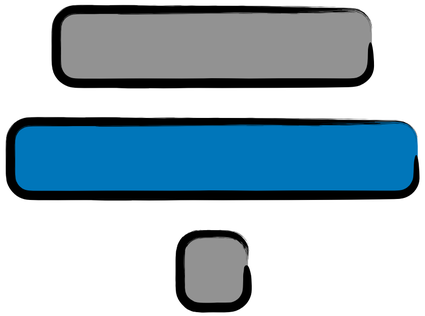


Short Recap: what we have achieved

We started with **universal function approximation**

any $f(x) = \mathbb{R}^n \rightarrow \mathbb{R}$ **can be represented by an MLP**
(given enough data)

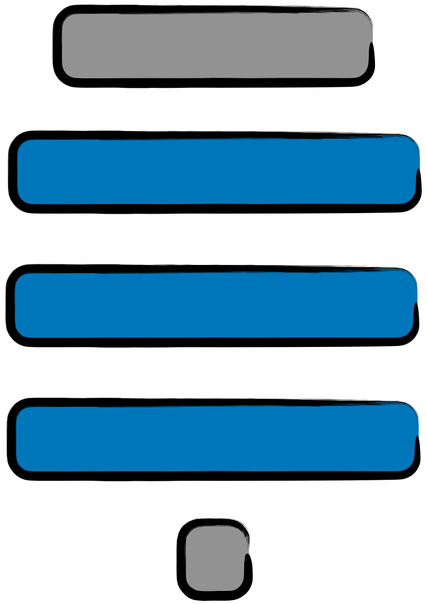
Single Layer MLP



Short Recap: what we have achieved

We discussed benefits of depth:
deep networks more expressive than shallow
with same # parameters

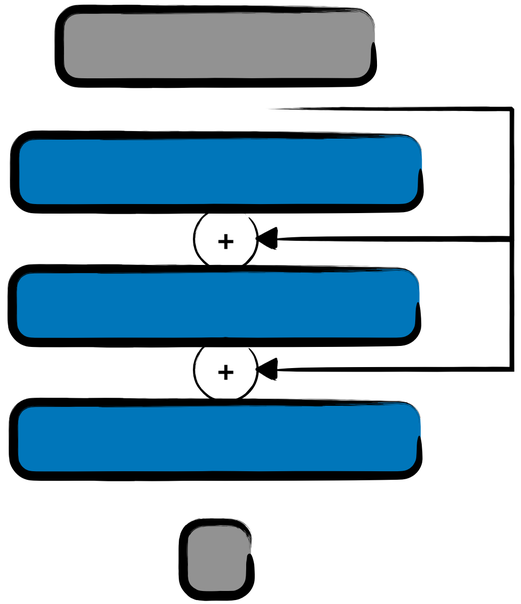
Deep MLP



Short Recap: what we have achieved

residual connections produce nested function classes and

Deep MLP with residual connections

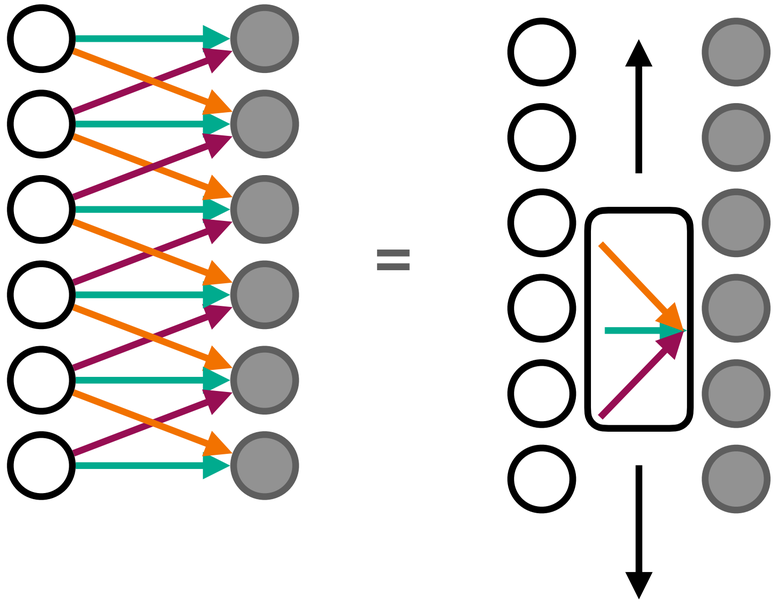


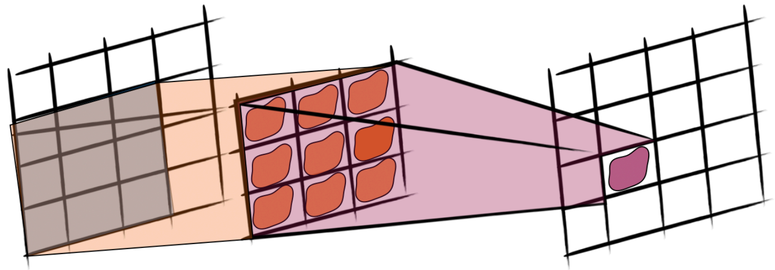
Short Recap: what we have achieved

With CNN, we adapt the structure of the network to the data type $\mathbb{R}^n = \mathbb{R}^{w \times h}$ to exploit inductive bias

CNN: Two Key Ideas
- *Local Connectivity*
- *Weight Sharing*

yields translation equi- and later in-variance



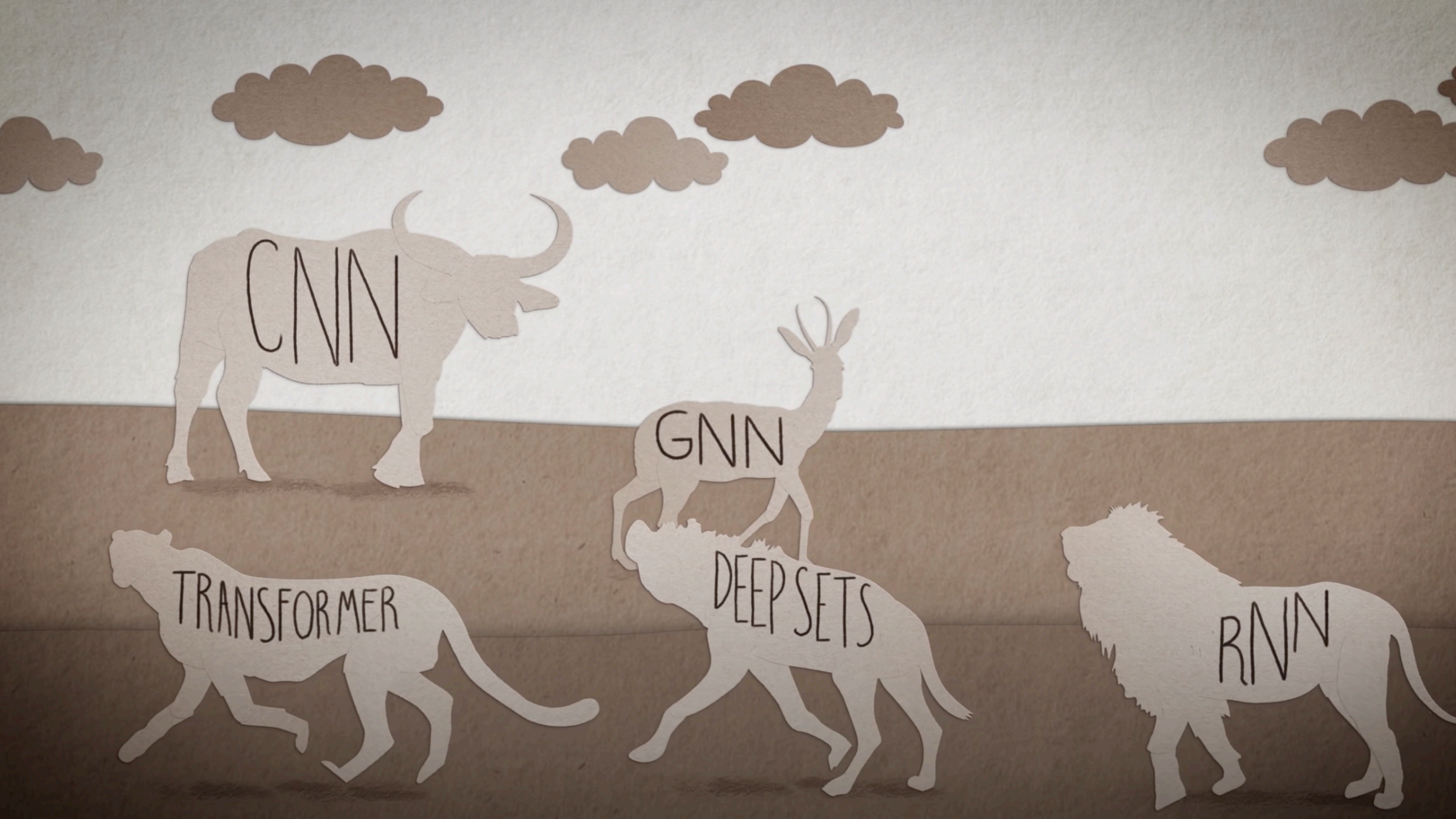


A General Trend

This is a general trend in Deep Learning
specific data types ↔ network architectures

to encode known properties of the in function family

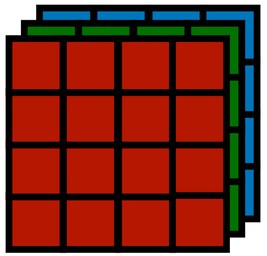
Data Type	***Symmetry***	***Network***
Grids/n-D arrays	*Translation*	*CNN*
Full Graph	*Permutation*	*Transformers*
Any Graphs	*Permutation*	*GNN*
Sets of Objects	*Permutation*	*Deep Set*
Sequences	*Time Warping*	*LSTM*
...	...	...

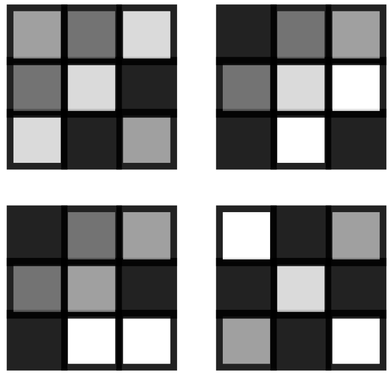

CNN
GNN
TRANSFORMER
DEEP SETS
RNN

Generalizing CNN

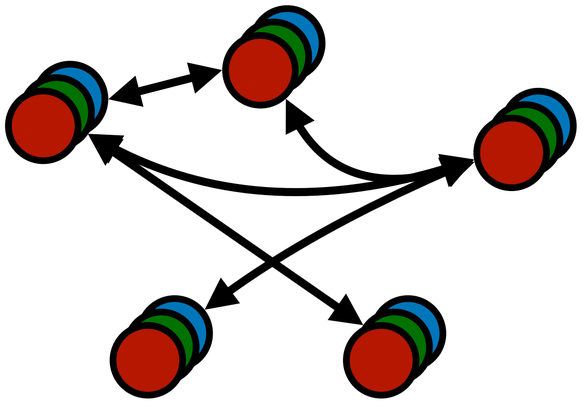
What should we do, if our data is not arranged in the, can we do something similar to CNNs & maintain similar features

Large Spatially Arranged Data





Limited Memory Filters Extracting Useful Signals



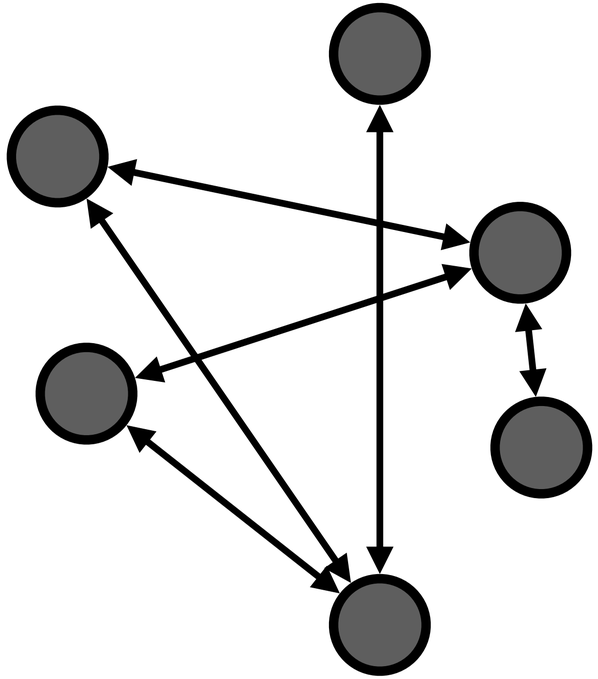
Generalize from pixels connected spatially to sets of generic objects that are connected in some arbitrary way

Graphs

Sets of non-regularly connected objects are best **described as graphs**

Defined as tuple (N,E)

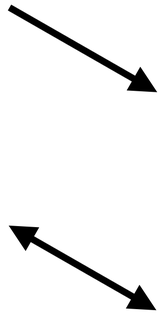
- Nodes (or Vertices)
- Edges

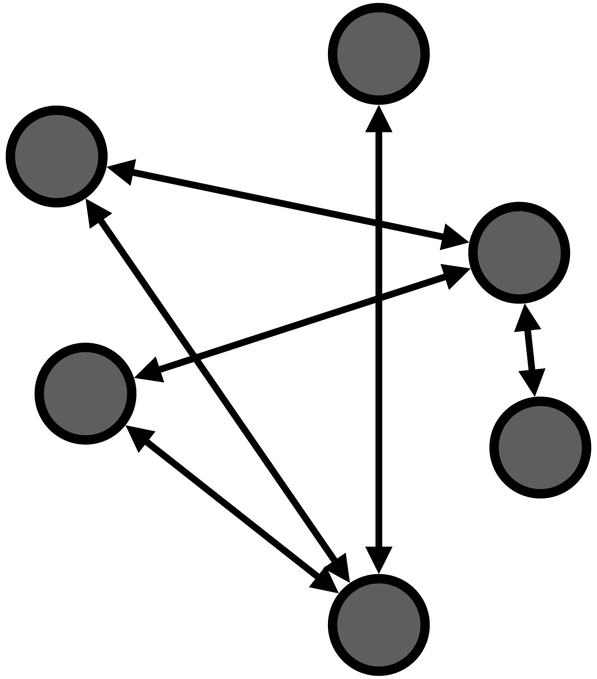


Directed and Undirected Graphs

Edges can be either bidirectional or unidirectional

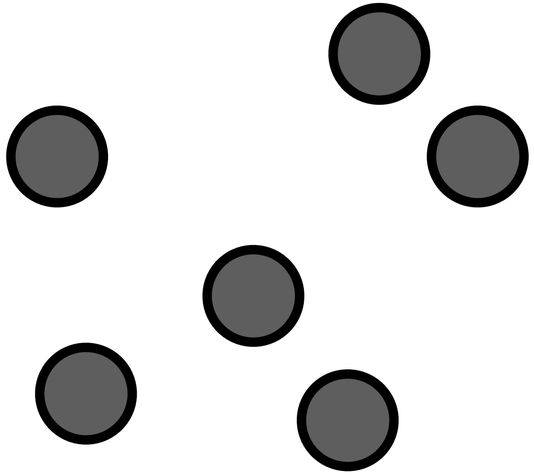
- Directed Edge
- Undirected Edge



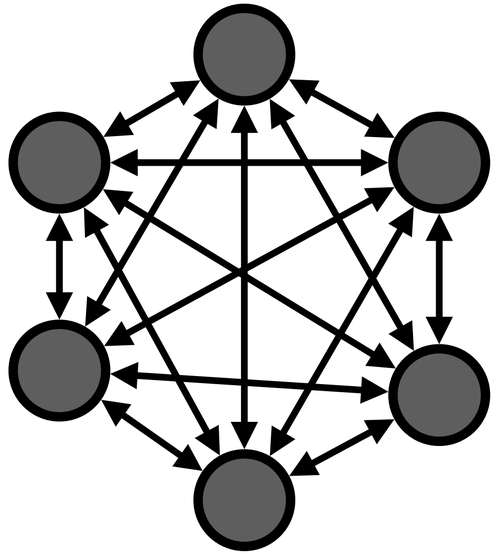


Graphs

Some special cases…



No connectivity (“bag of objects”)

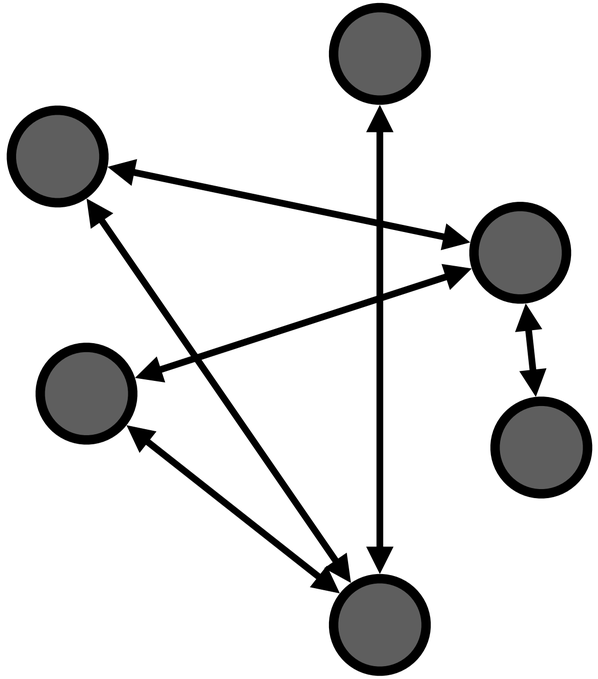


Full Connectivity

Graphs

The **graph structure is the “substrate”** onto which we put out data - it’s not the data itself

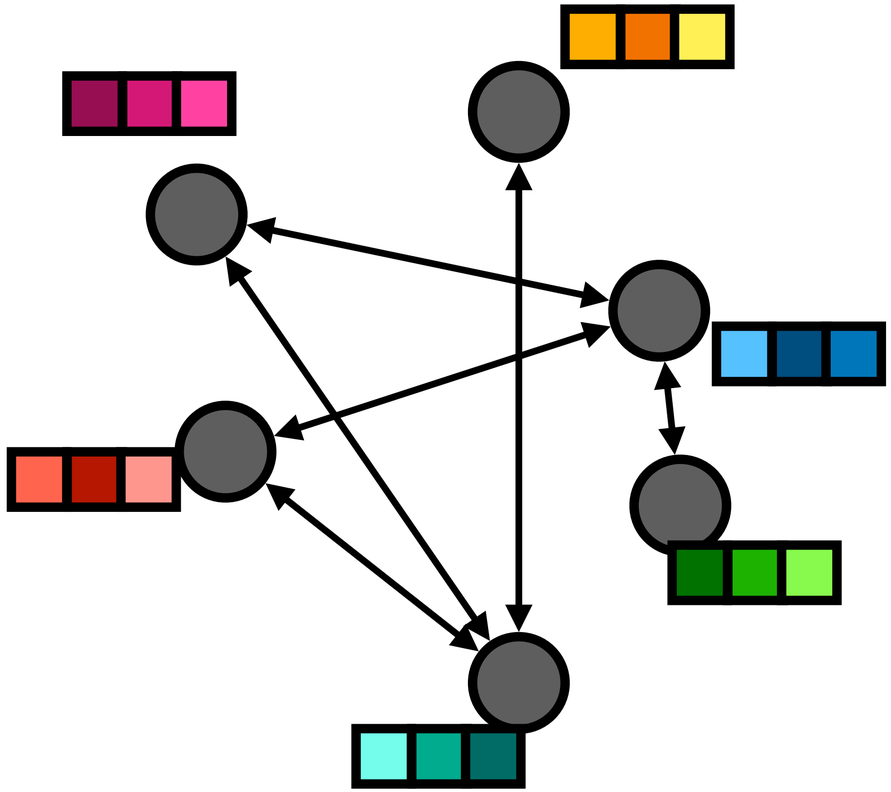
Can attach data to both of the key components of a graph



Graphs

The **graph structure is the “substrate”** onto which we put out data - it’s not the data itself

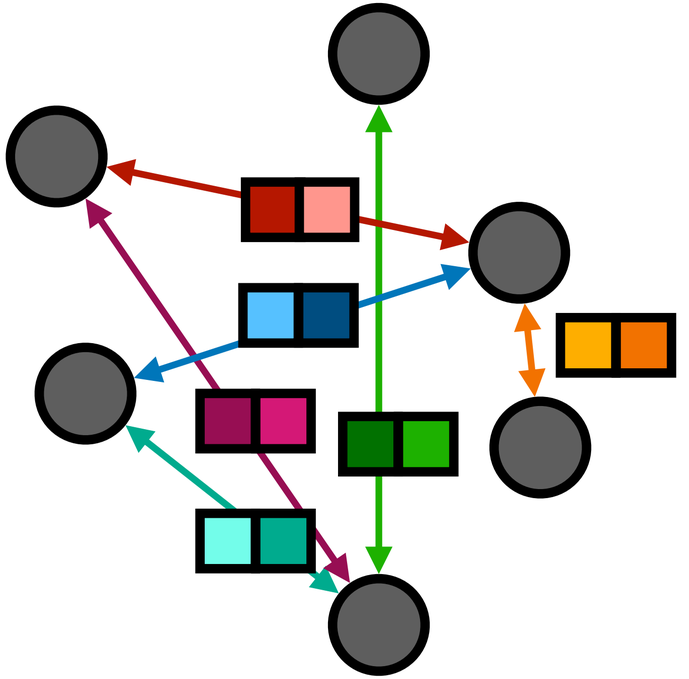
attach features $\mathbb{R}^3$ to nodes



Graphs

The **graph structure is the “substrate”** onto which we put out data - it’s not the data itself

attach features $\mathbb{R}^2$ to edges



Examples

Graph structured data is everywhere!

Node Features

Age Hair Color

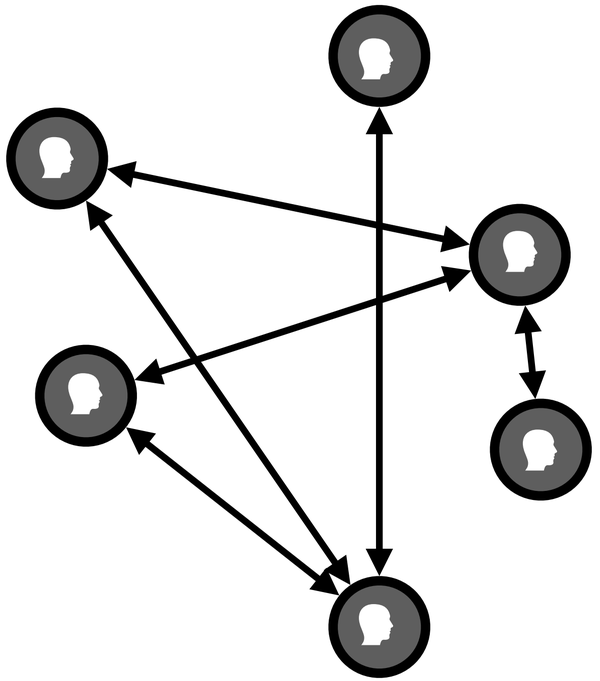
Height Location

Edge Features

Type of Relationship

Strength of Relationship

Length of Relationship



Social Networks

Examples

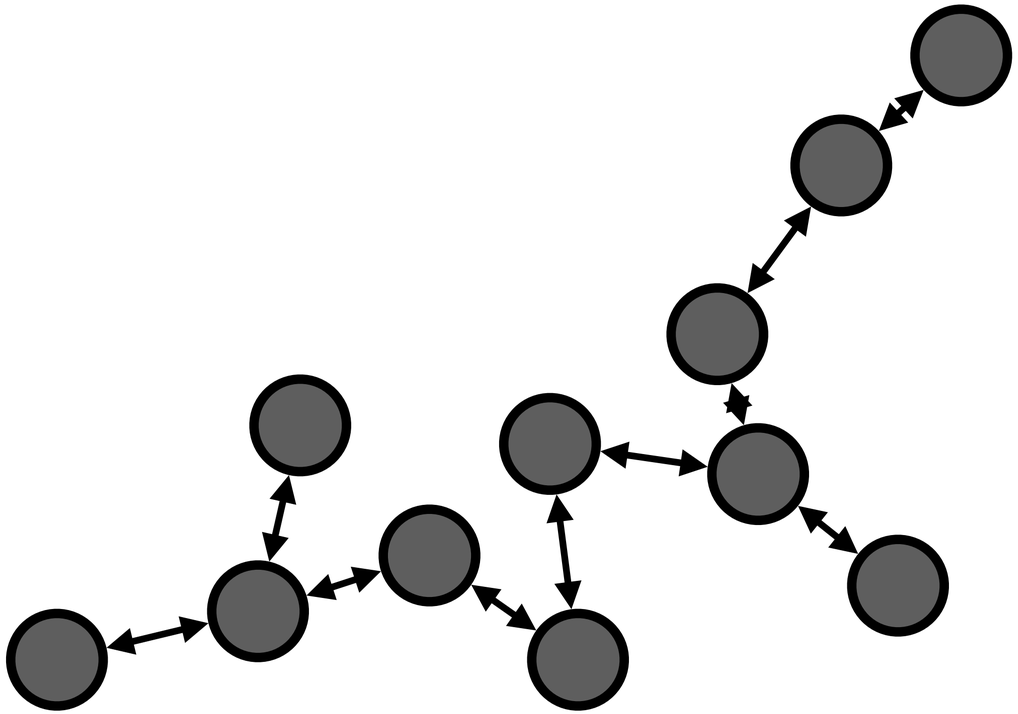
Graph structured data is everywhere!

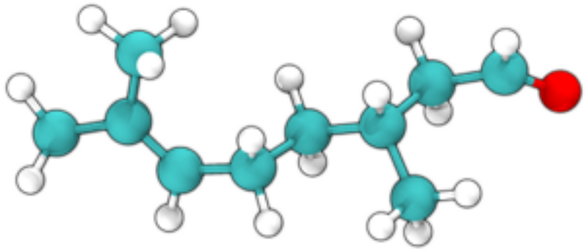
Node Features

Position

Atom Type

Edge Features





[src]

Molecules

Examples

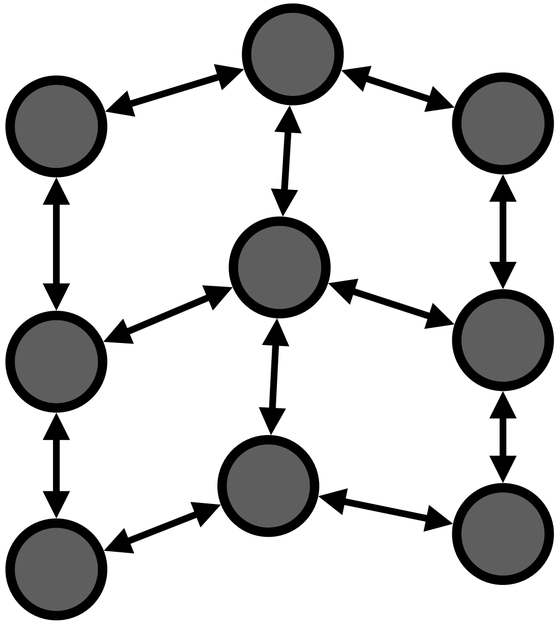
Graph structured data is everywhere!

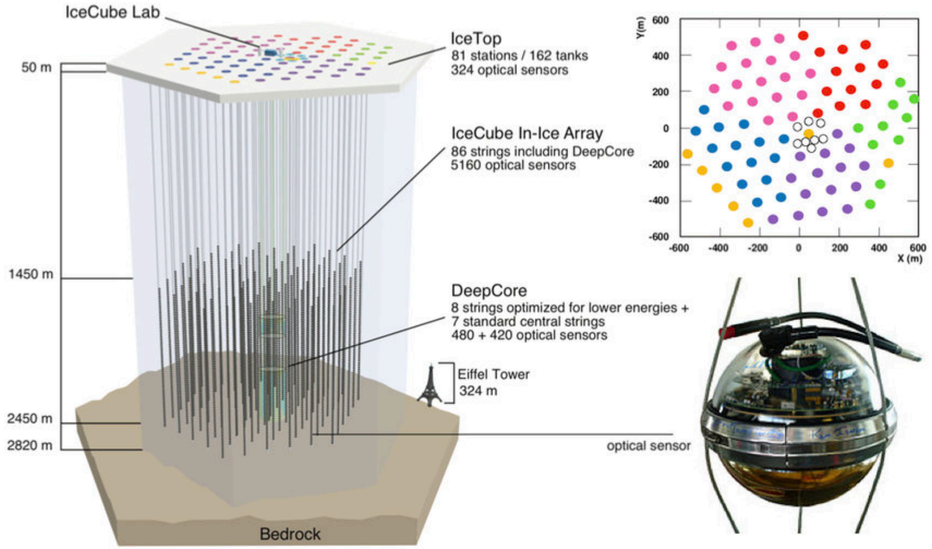
Node Features

Position

Atom Type

Edge Features





Particle Detectors

Examples

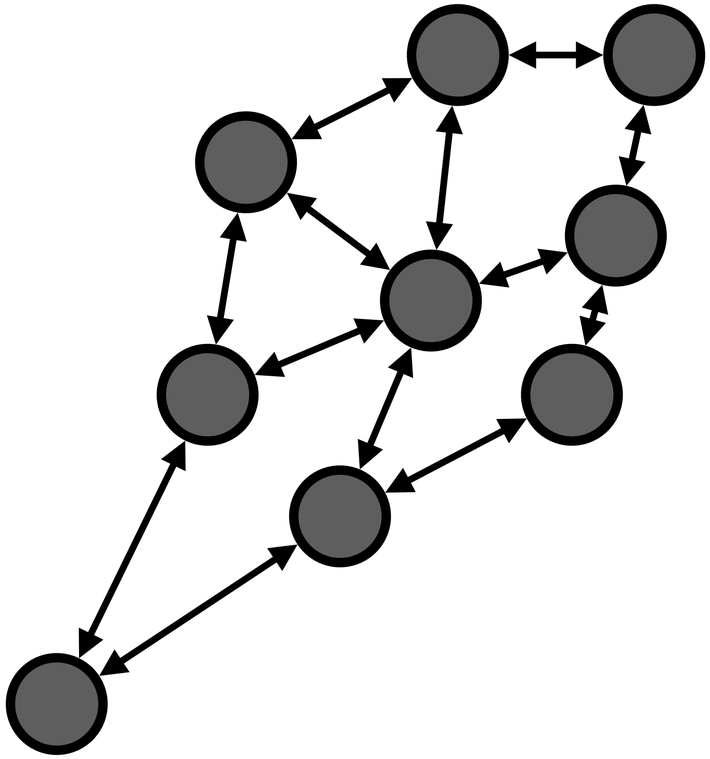
Graph structured data is everywhere!

Node Features

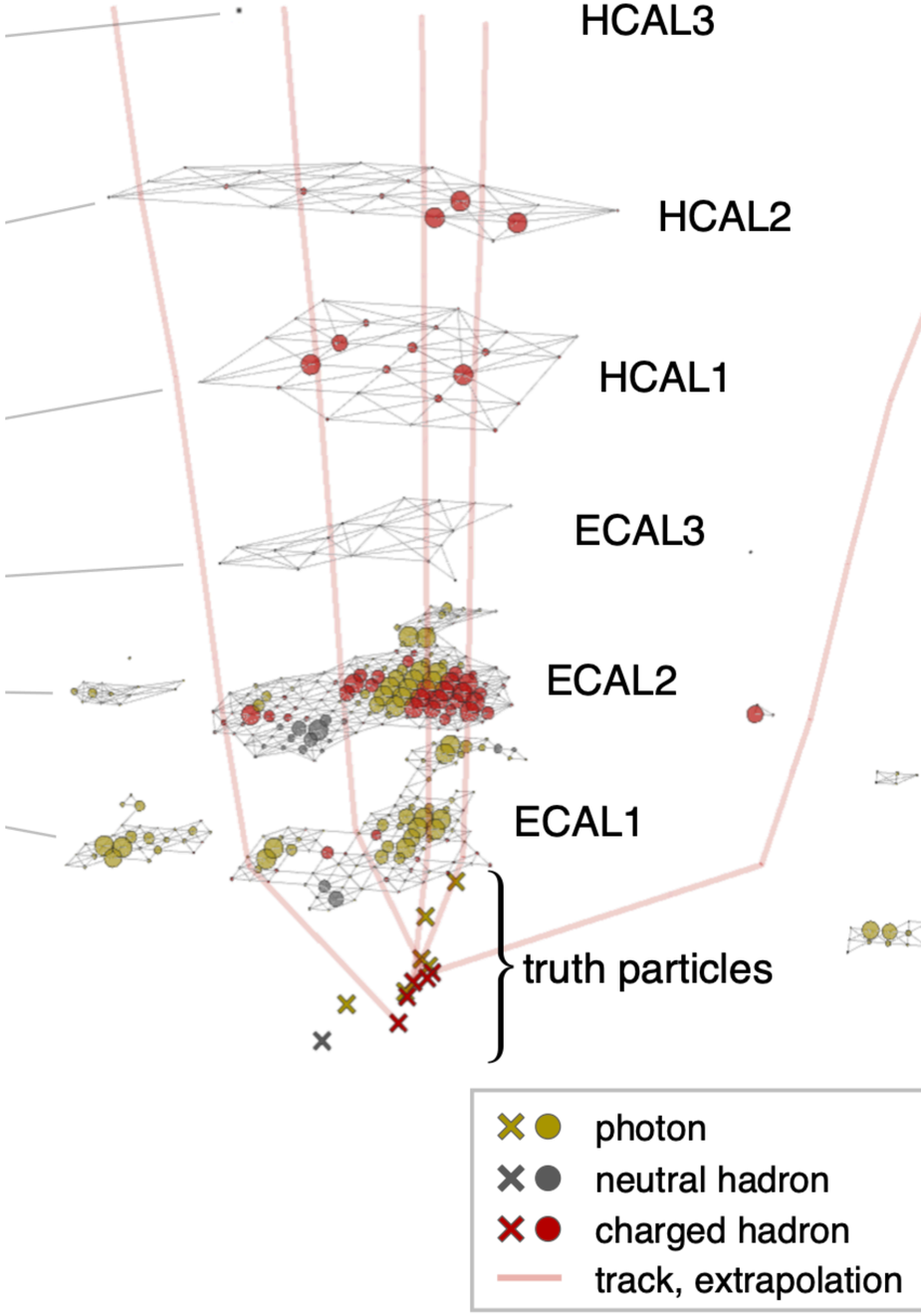
Position

Observed Signal

Edge Features

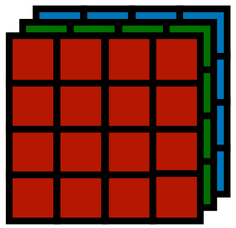


Particle Detectors



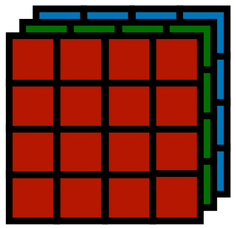
Examples

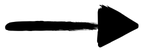
In fact, even images can be thought of as graphs

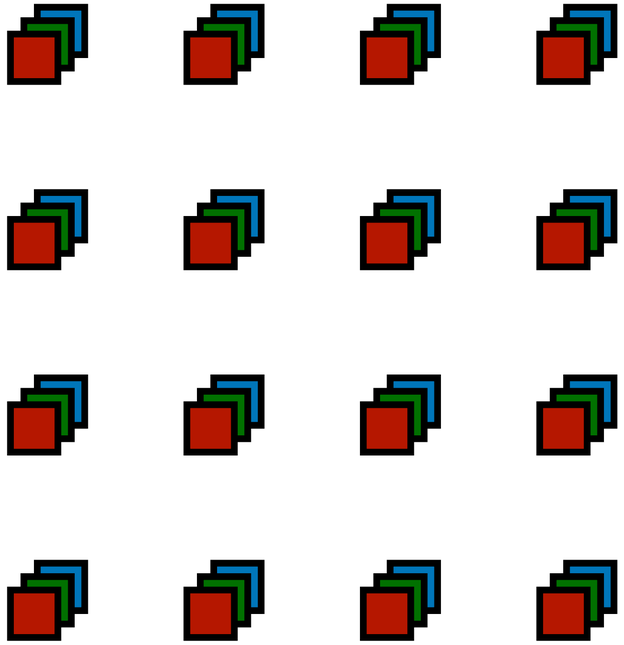


Examples

In fact, even images can be thought of as graphs





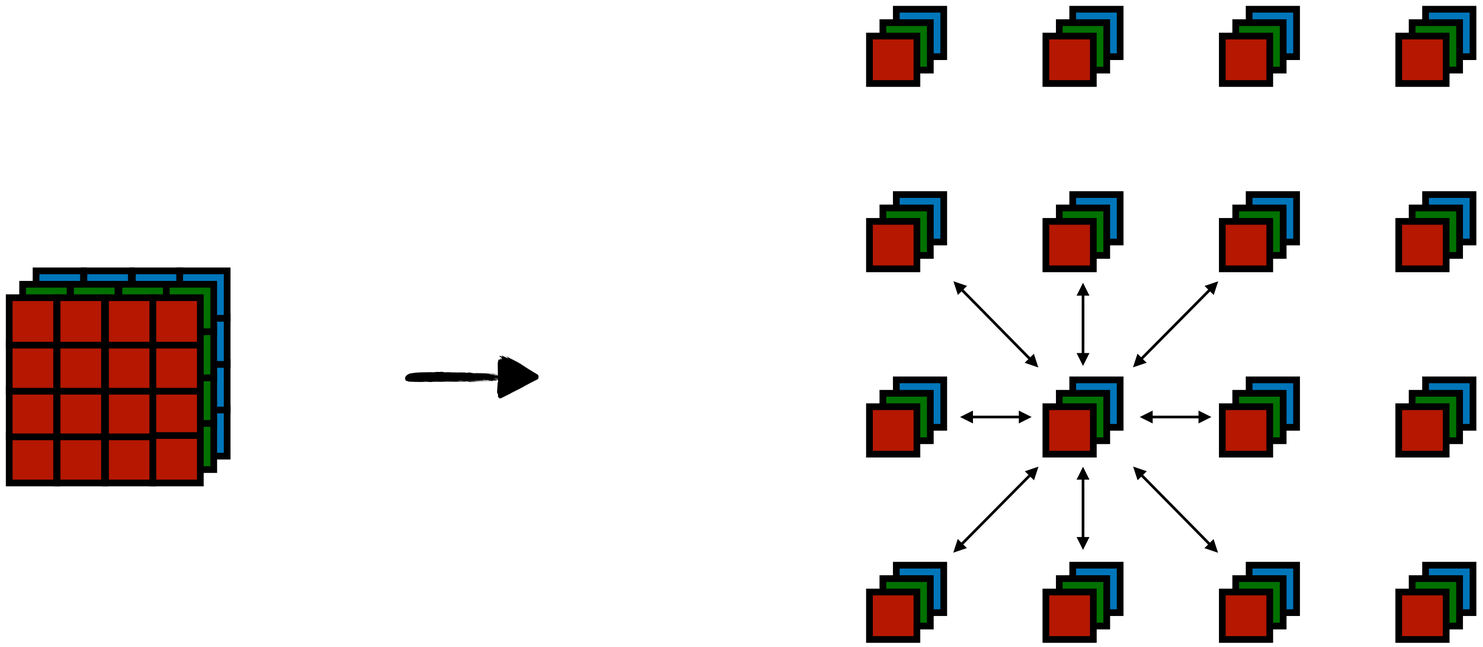


Each pixel is its own object

Node Features: RGB intensities

Examples

In fact, even images can be thought of as graphs

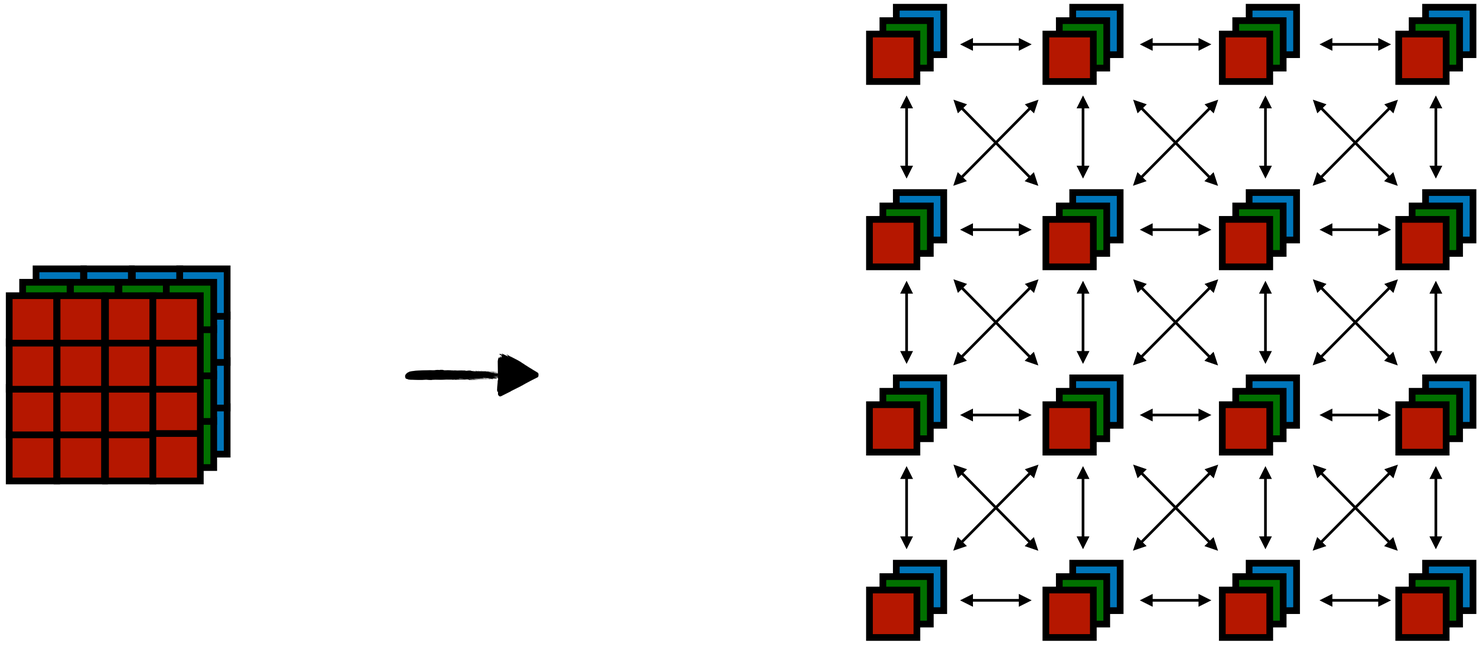


Images: Grids with very regular local connectivity

Edge Features: orientation (to the left / to the right, …)

Examples

In fact, even images can be thought of as graphs



Learning Problems on Graphs

There are a variety of possible learning problems on graphs

Node-Level **Edge-Level** **Graph-Level**

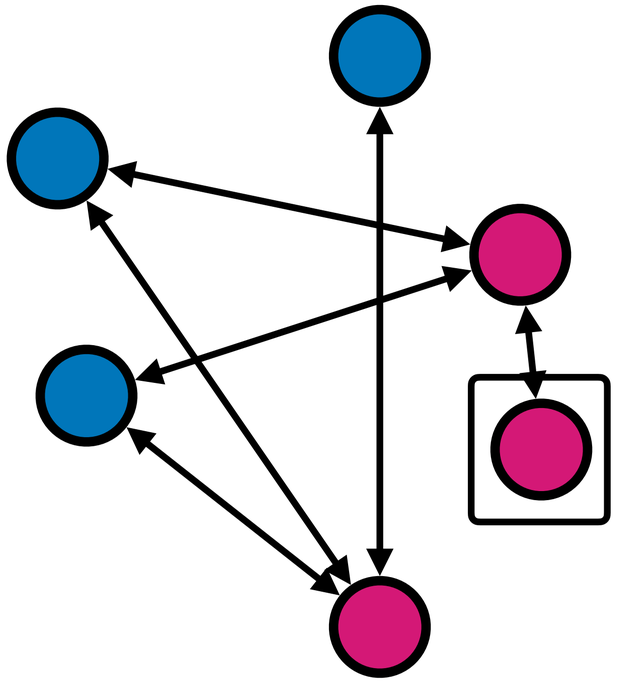
Learning Problems on Graphs

Node-level prediction:

Predict some latent property
of a given node, given the graph.

Example:

What's a person's favorite sports
club given a social network
and observed node features



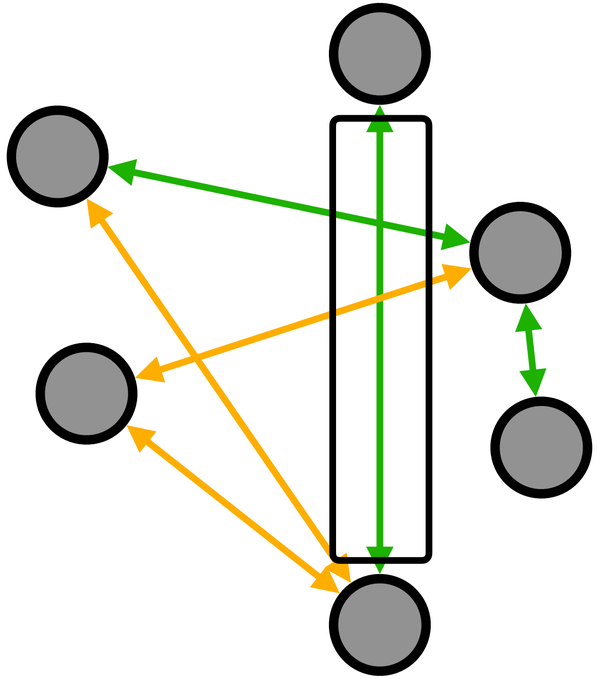
Learning Problems on Graphs

Edge-level prediction:

Predict some latent property
of a given edge, given the graph.

Example:

Given a social network,
which edge corrsponds to a
first-degree familiy relationship?



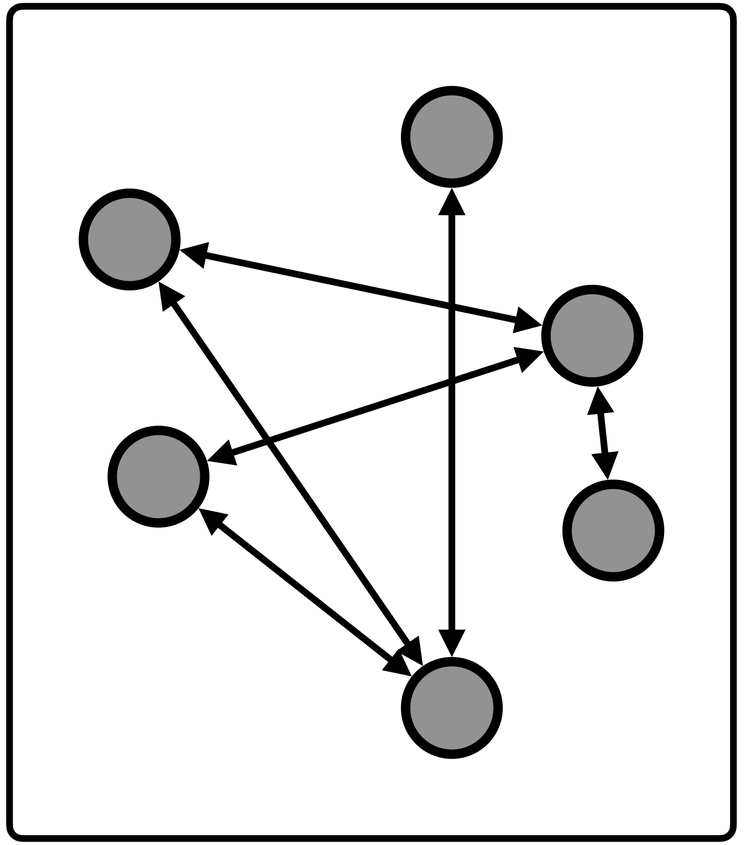
Learning Problems on Graphs

Graph-level prediction:

Predict some latent property
of a given edge, given the graph.

Example:

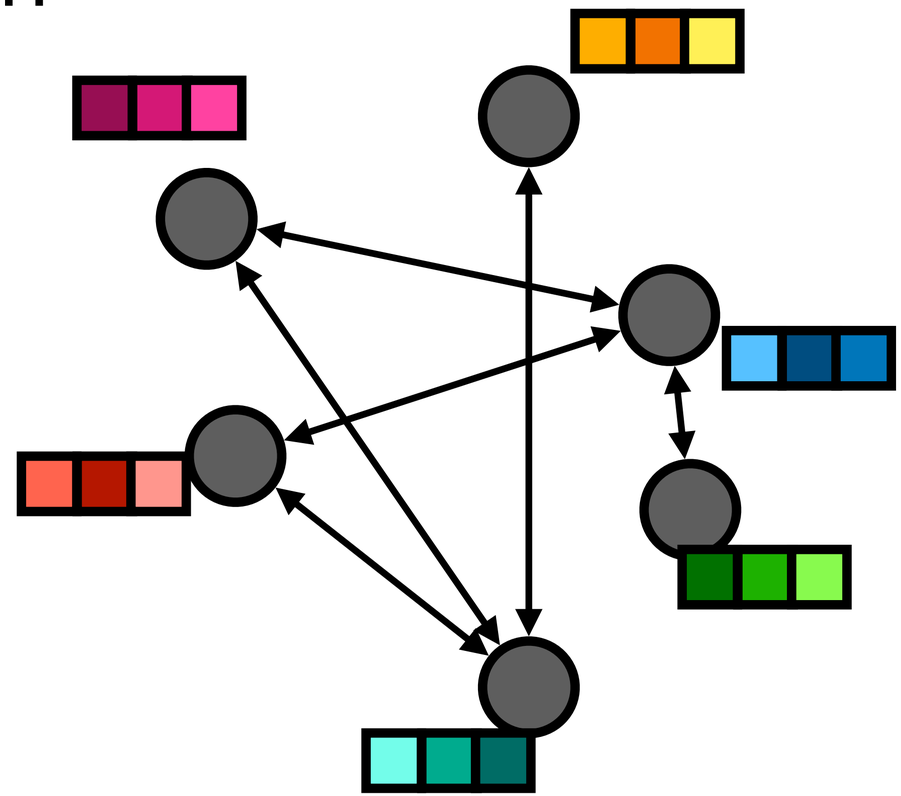
Given a social network,
which edge corrsponds to a
first-degree familiy relationship?



Data Representation of Graphs

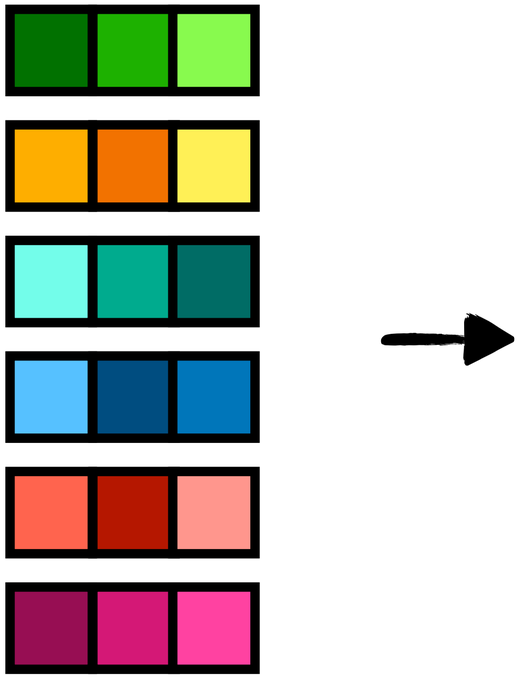
We can compactly represent the graph with matrices, but for this we need to impose an node order

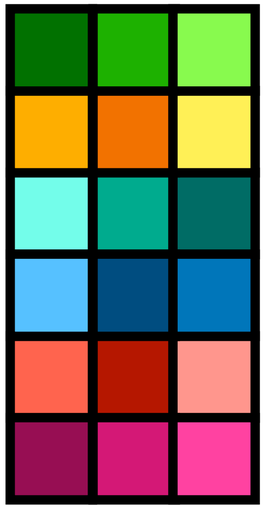
Important this order is arbitrary



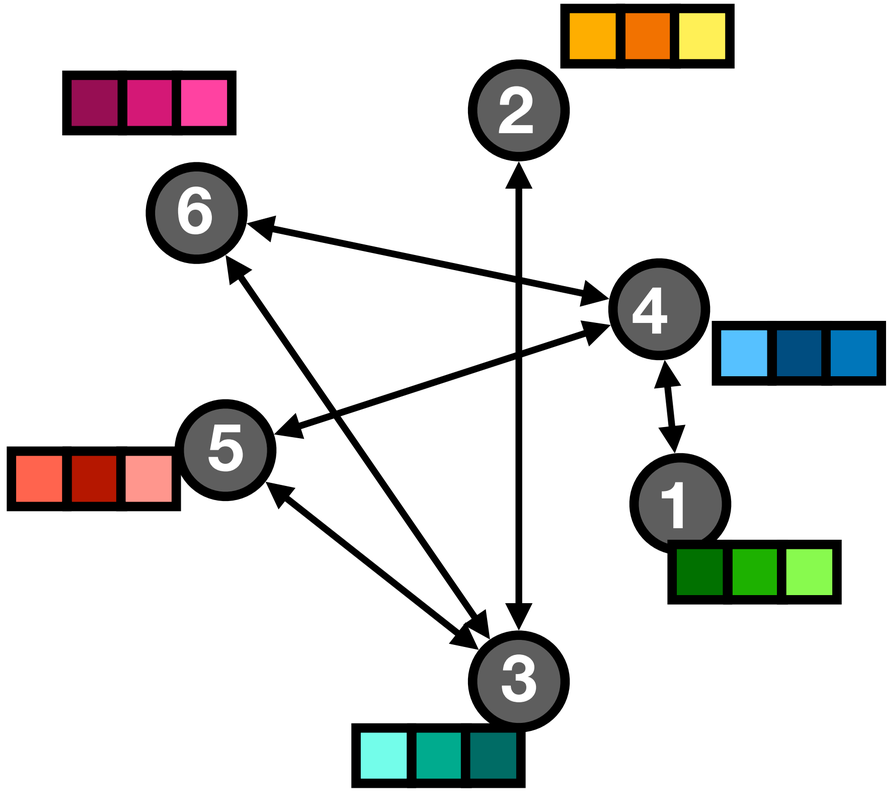
Data Representation of Graphs

With an ordering we can represent the data as a matrix



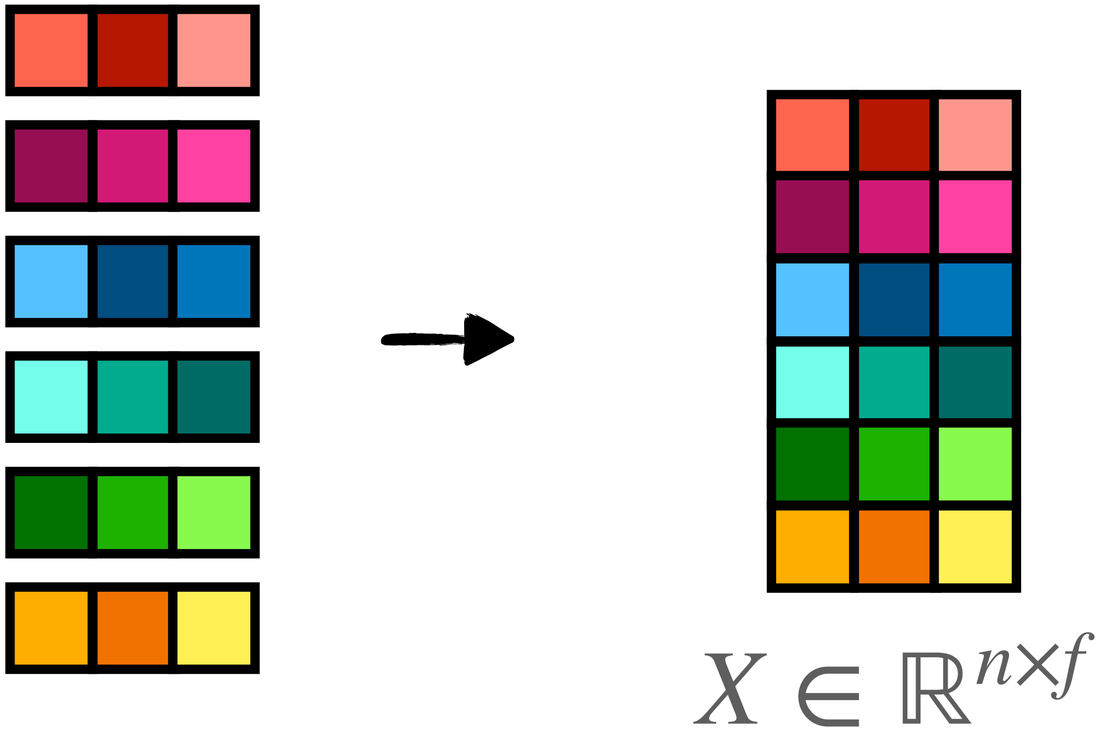


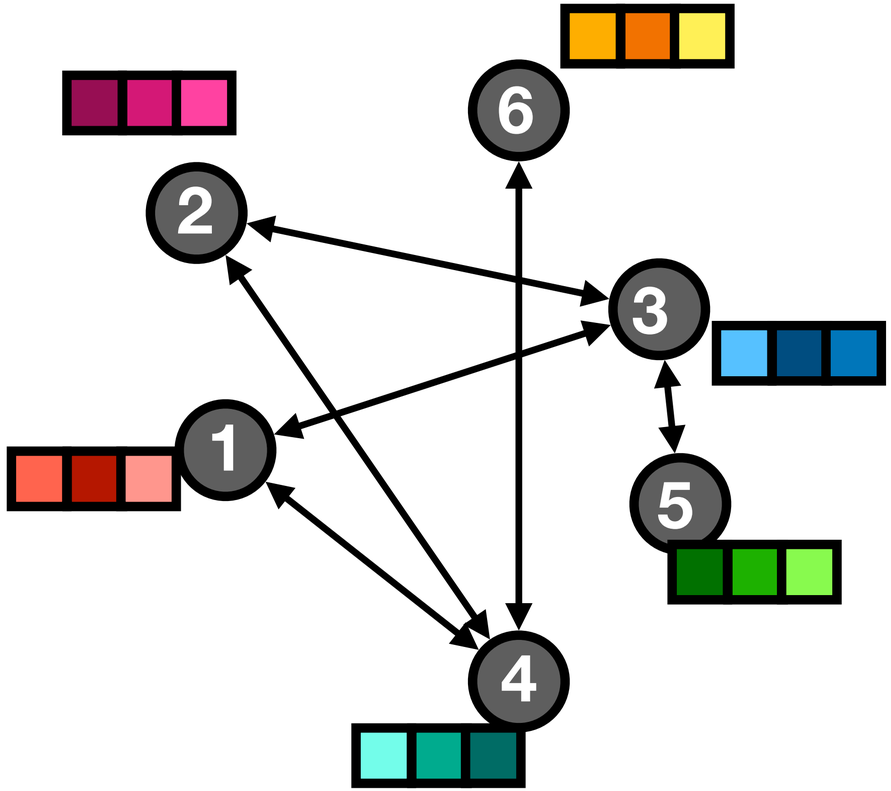
$$X \in \mathbb{R}^{n \times f}$$



Data Representation of Graphs

For a different ordering…

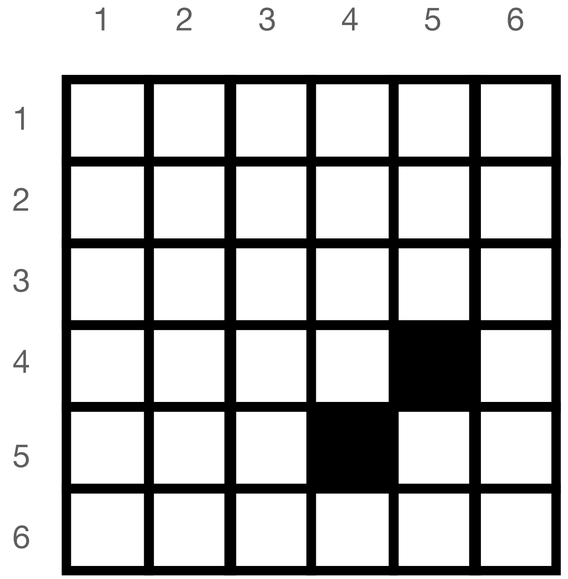


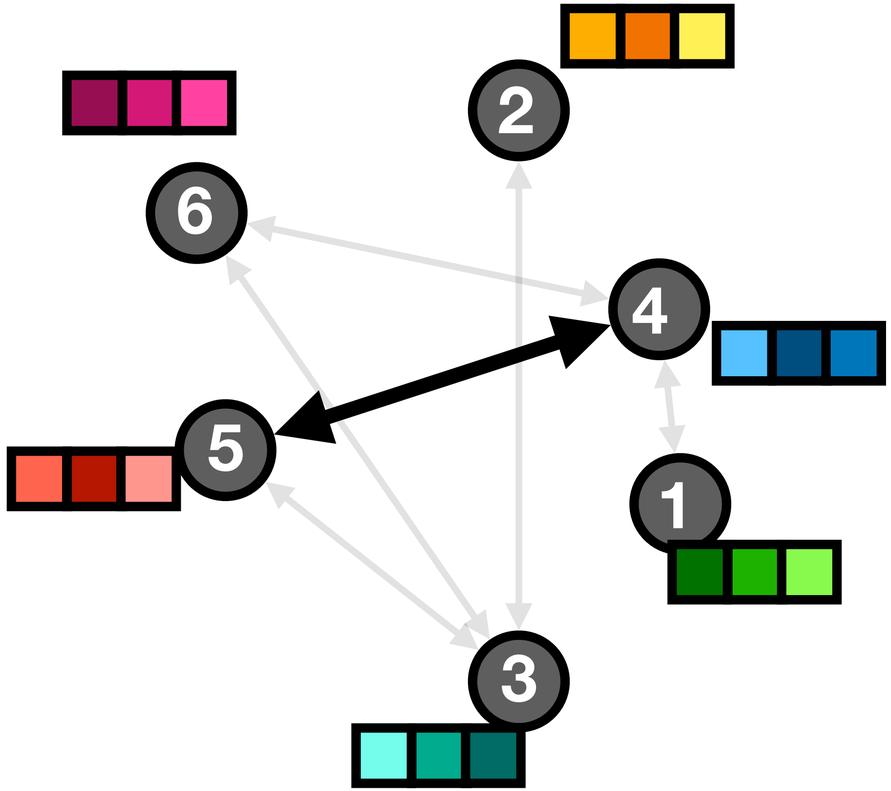


Data Representation of Graphs

The edges can be represented as an

Adjacency Matrix $A \in \mathbb{R}^{n \times n}$

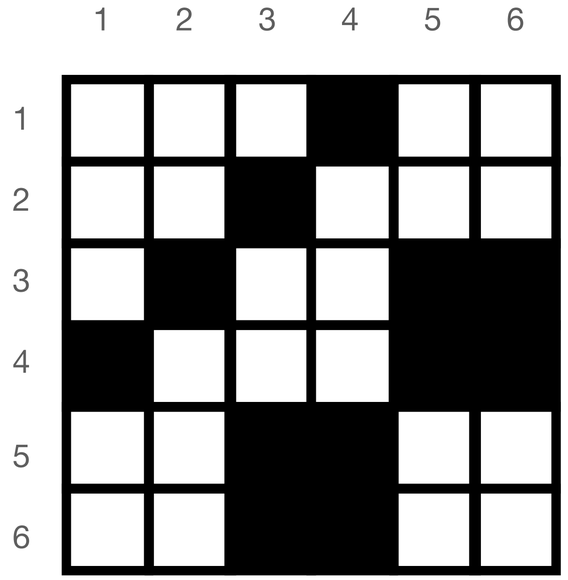


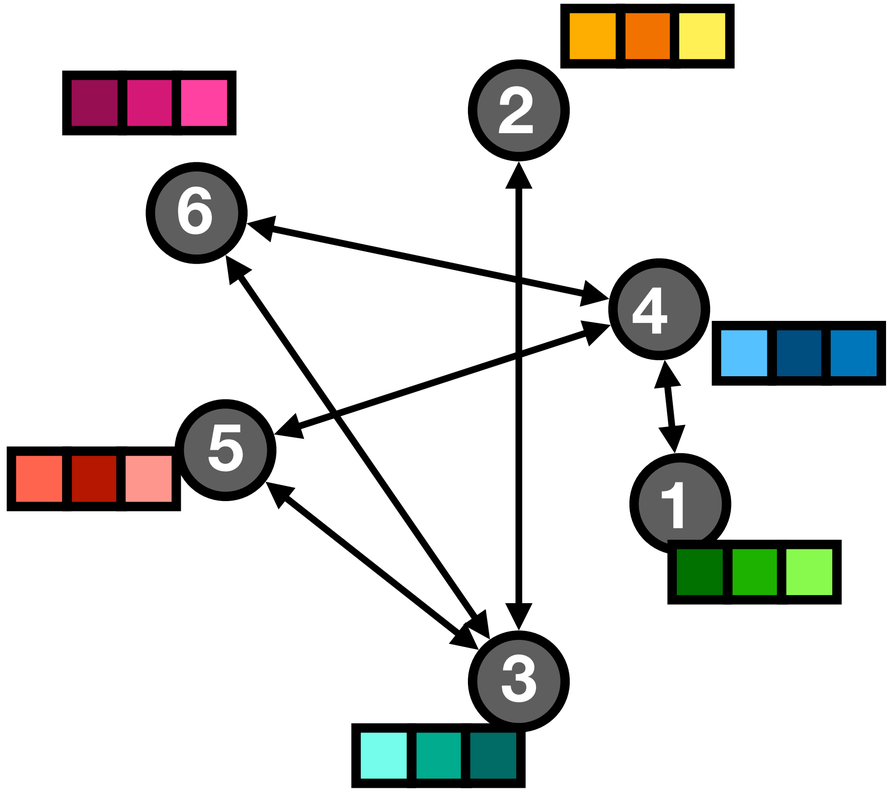


Data Representation of Graphs

The edges can be represented as an

Adjacency Matrix $A \in \mathbb{R}^{n \times n}$

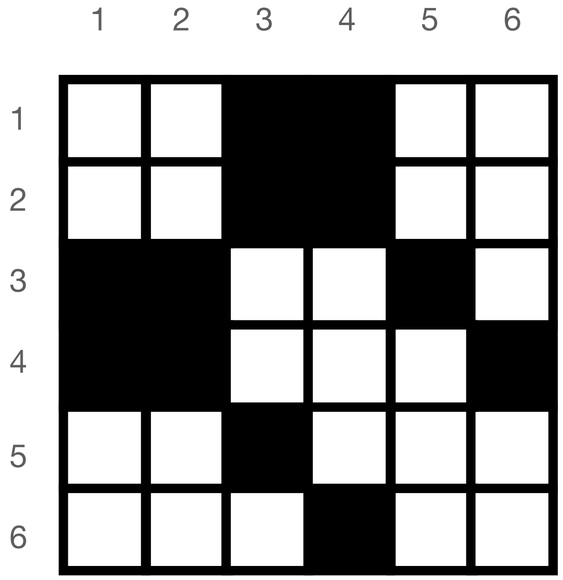


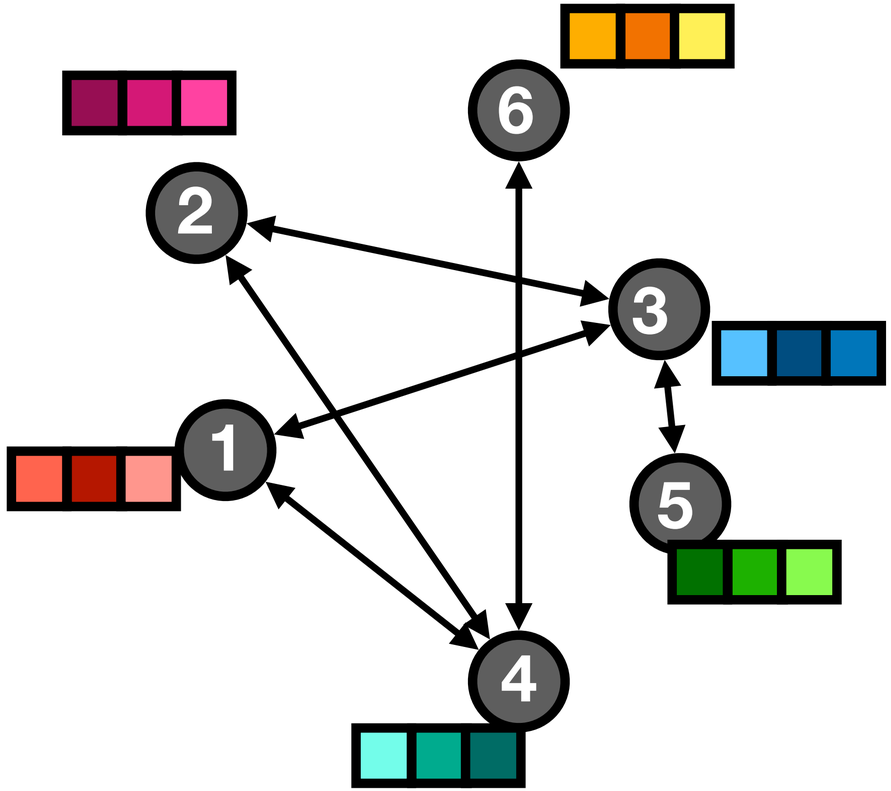


Data Representation of Graphs

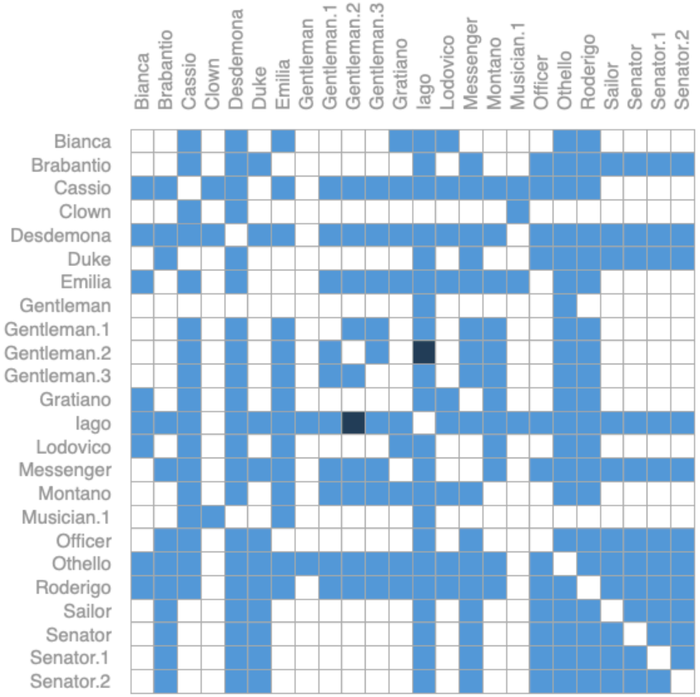
Ordered differently…

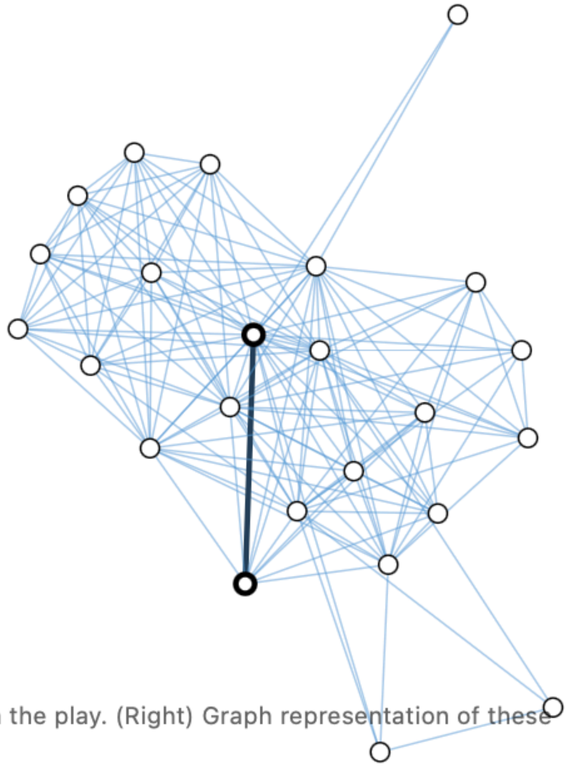
Adjacency Matrix $A \in \mathbb{R}^{n \times n}$





Examples





(Left) Image of a scene from the play “Othello”. (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

Examples

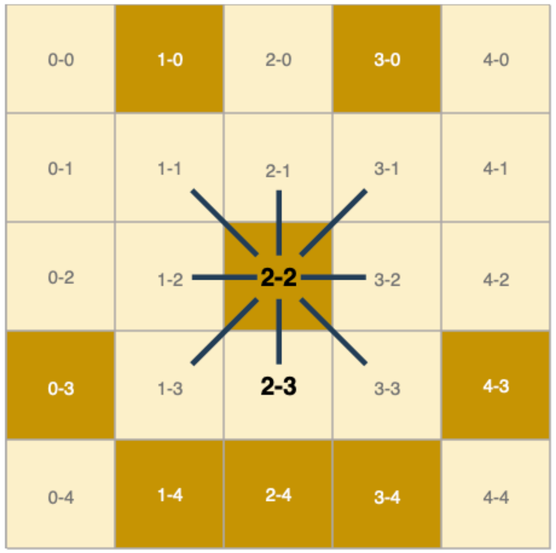
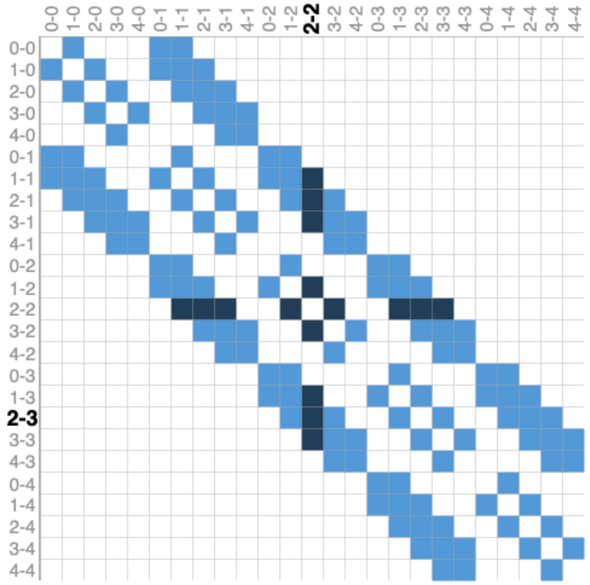
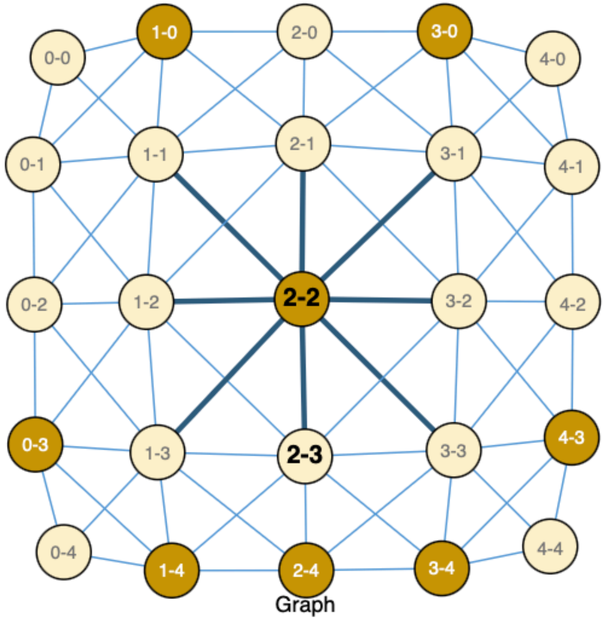


Image Pixels



Adjacency Matrix

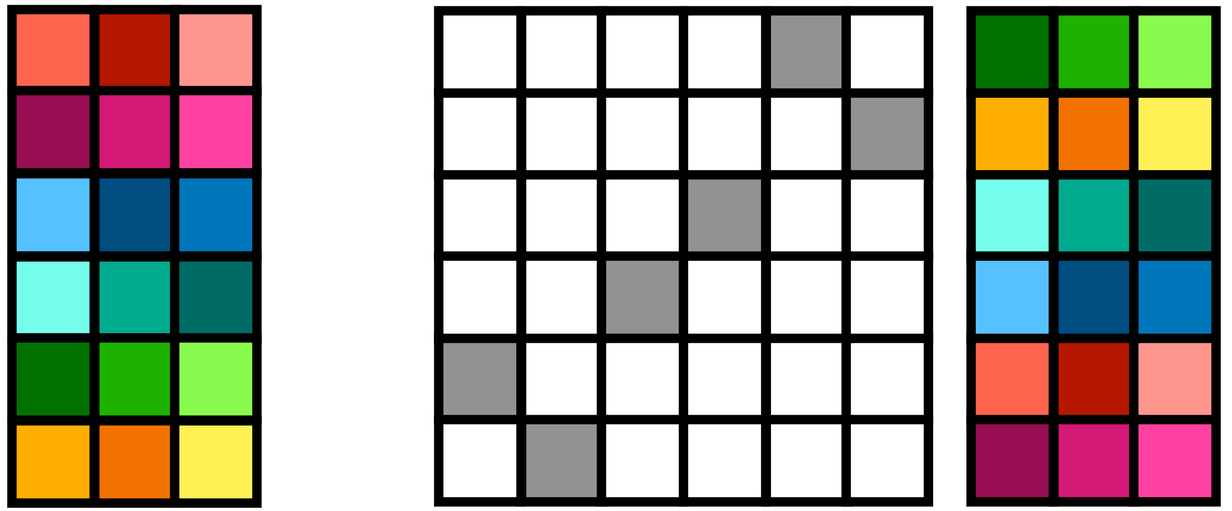


Graph

Click on an image pixel to toggle its value, and see how the graph representation changes.

Permutation Matrices

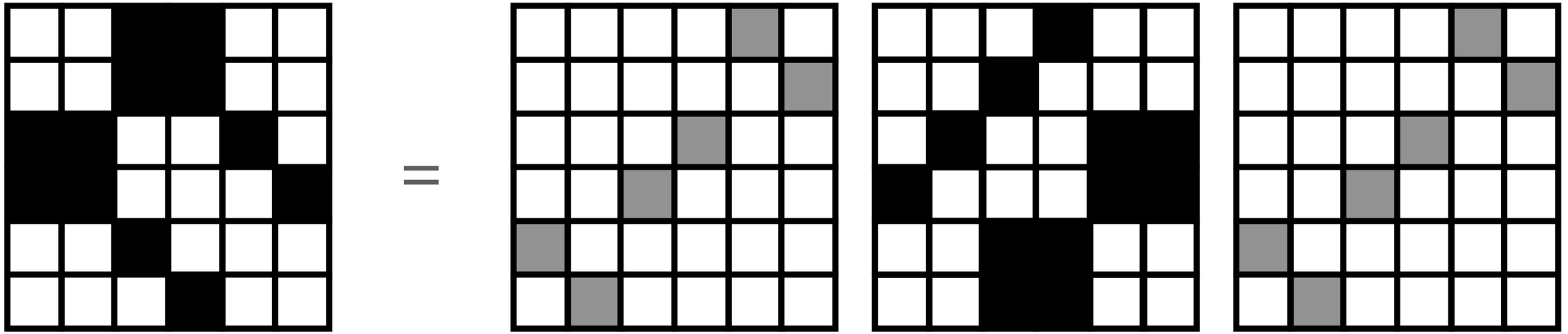
We can represent permutations using a **permutation matrix**



$$X' = PX$$

Permutation Matrices

We can represent permutations using a **permutation matrix**

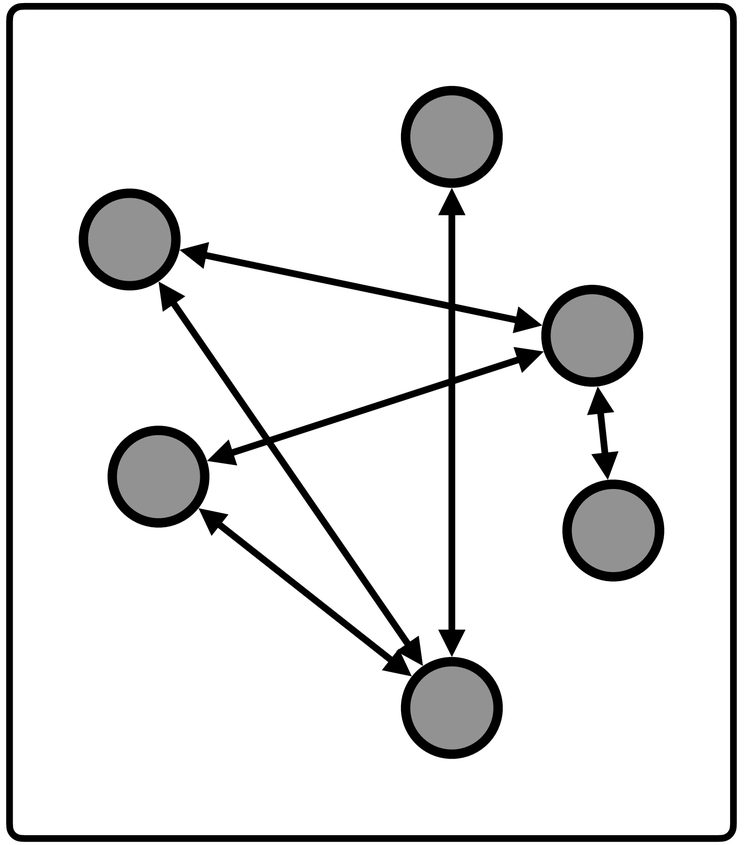


$$A' = PAP^T$$

Function Symmetries

Node-level functions create new node data from old node data

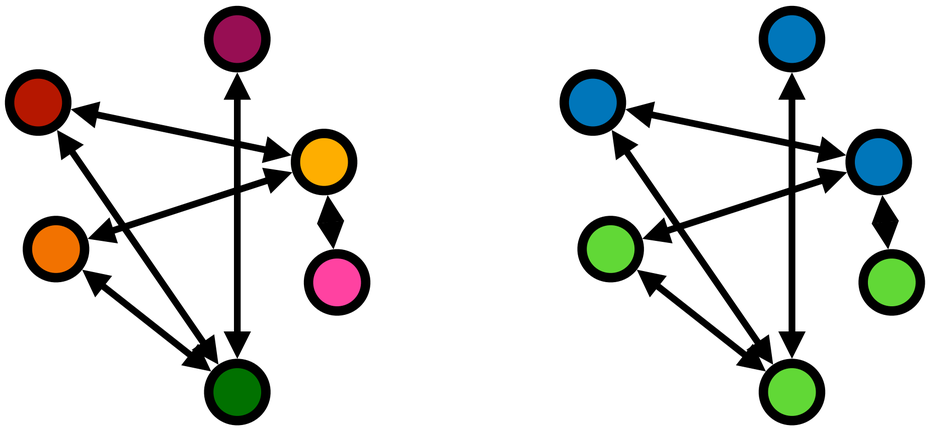
$$Y = f(X)$$

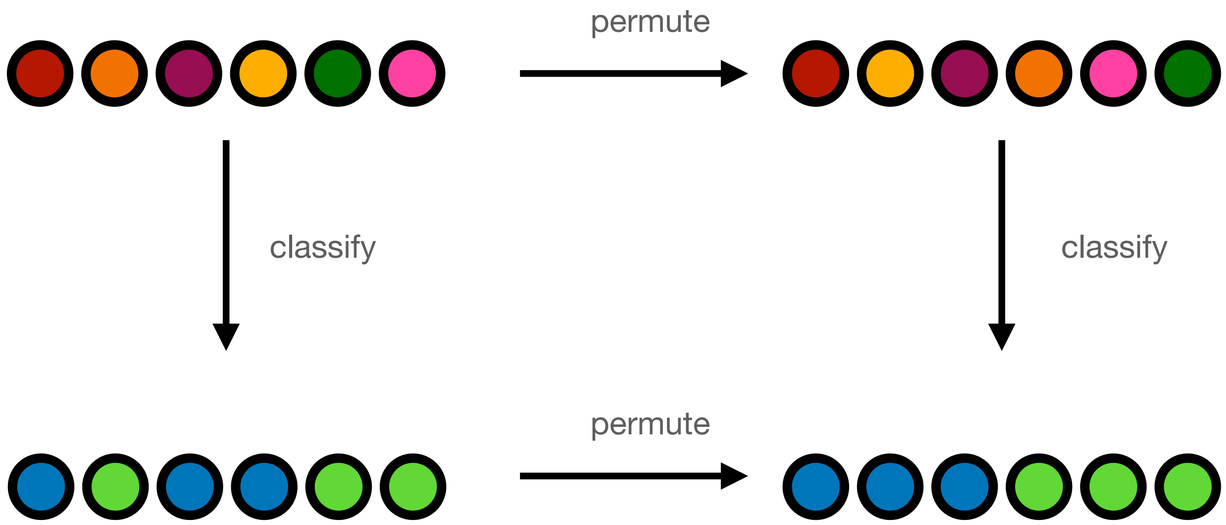


Function Symmetries

Node-level functions should be permutation equivariant

$$Pf(X) = f(PX)$$

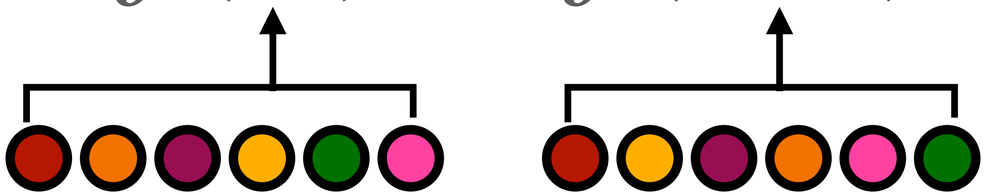




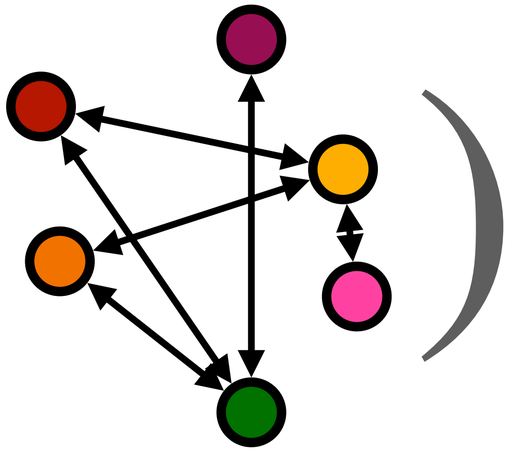
Function Symmetries

Graph-level functions should be permutation-invariant

$$f(X) = f(PX)$$



order should not matter

$$f($$  $$)$$

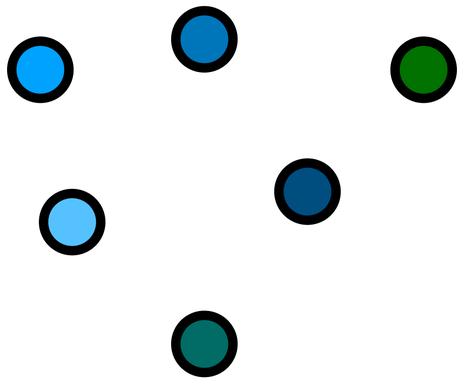
purely a function of the graph

Graphs w/o Edges

Function on Sets

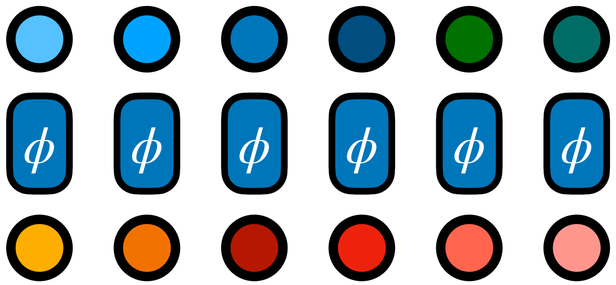
We can get a glimpse how to get permutation invariant functions by looking at **graphs without connectivity**:

Sets



Function on Sets

Without any connectivity, there is no neighborhood, the only way to transform data is element-wise

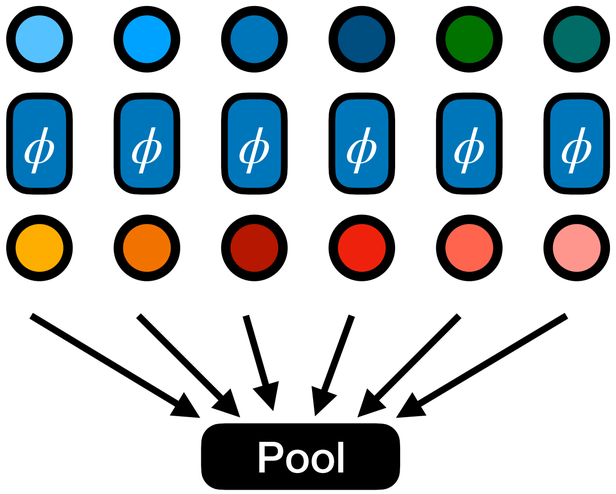


$$x_i \rightarrow y_i = \phi(x_i)$$

No connections = trivially permutation equivariant

Permutation-Invariance

To get combine the outputs we can add a **permutation-invariant pooling function**

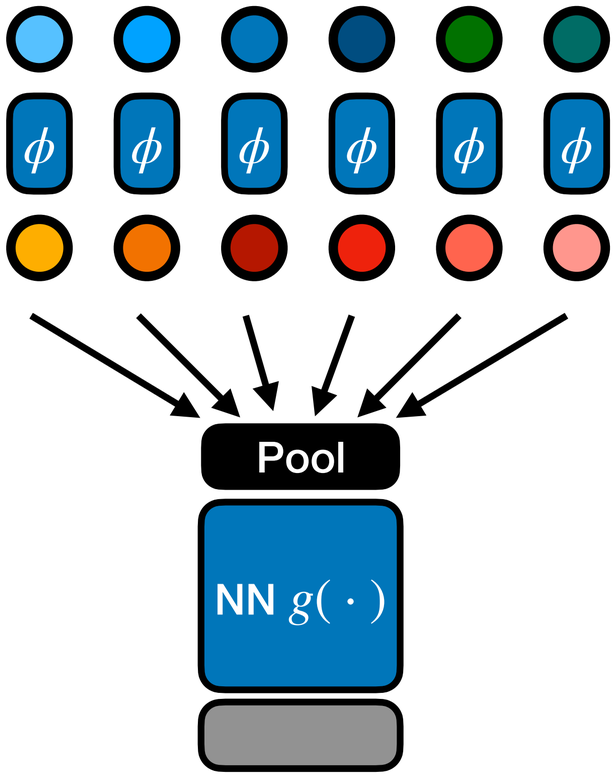


Example:
The sum is permutation-invariant

$$f(x_1, \ldots, x_n) = \sum_i y_i = \sum_i f(x_i)$$

Permutation-Invariance

The pooled output can then be processed by any given "head" function to produce the final output



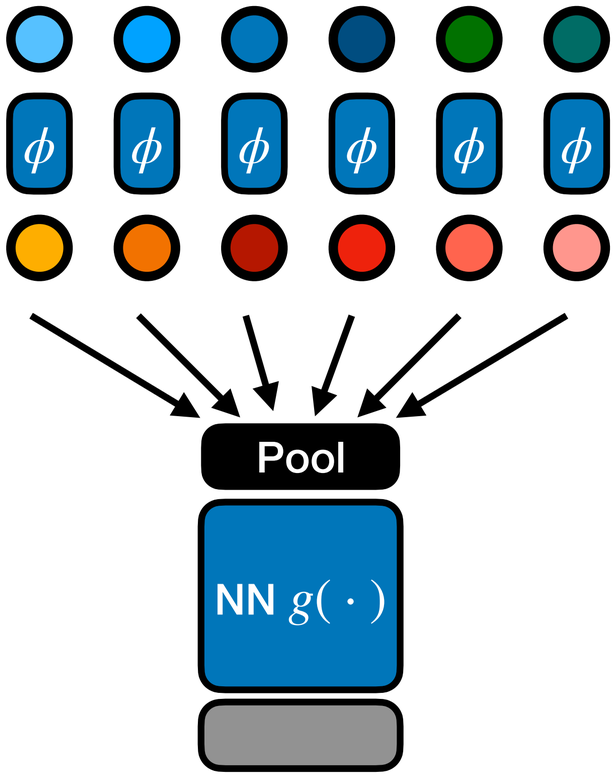
$$f(X) = g(\phi(x_1, \ldots, x_n))$$

the resulting function is expressive via $\phi(\cdot), g(\cdot)$ but permutation invariant

$$f(X) = f(PX)$$

Permutation-Invariance

The pooled output can then be processed by any given “head” function to produce the final output



$$f(X) = g(\phi(x_1, \ldots, x_n))$$

These networks are called:

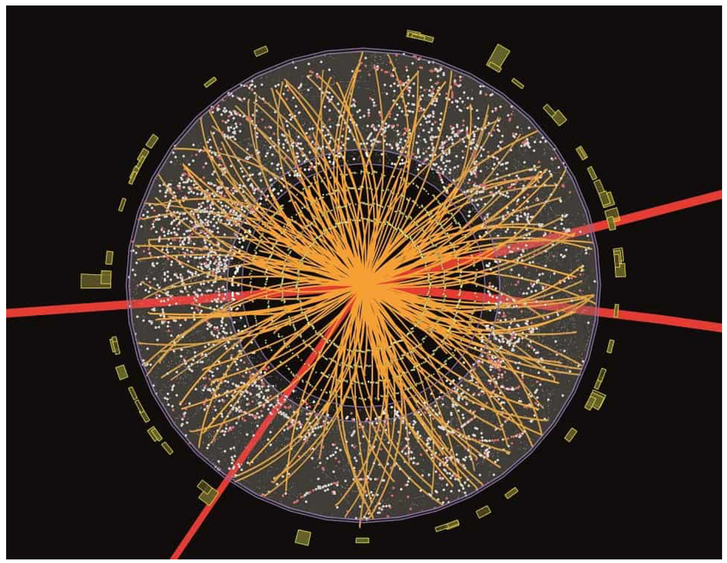
Deep Sets

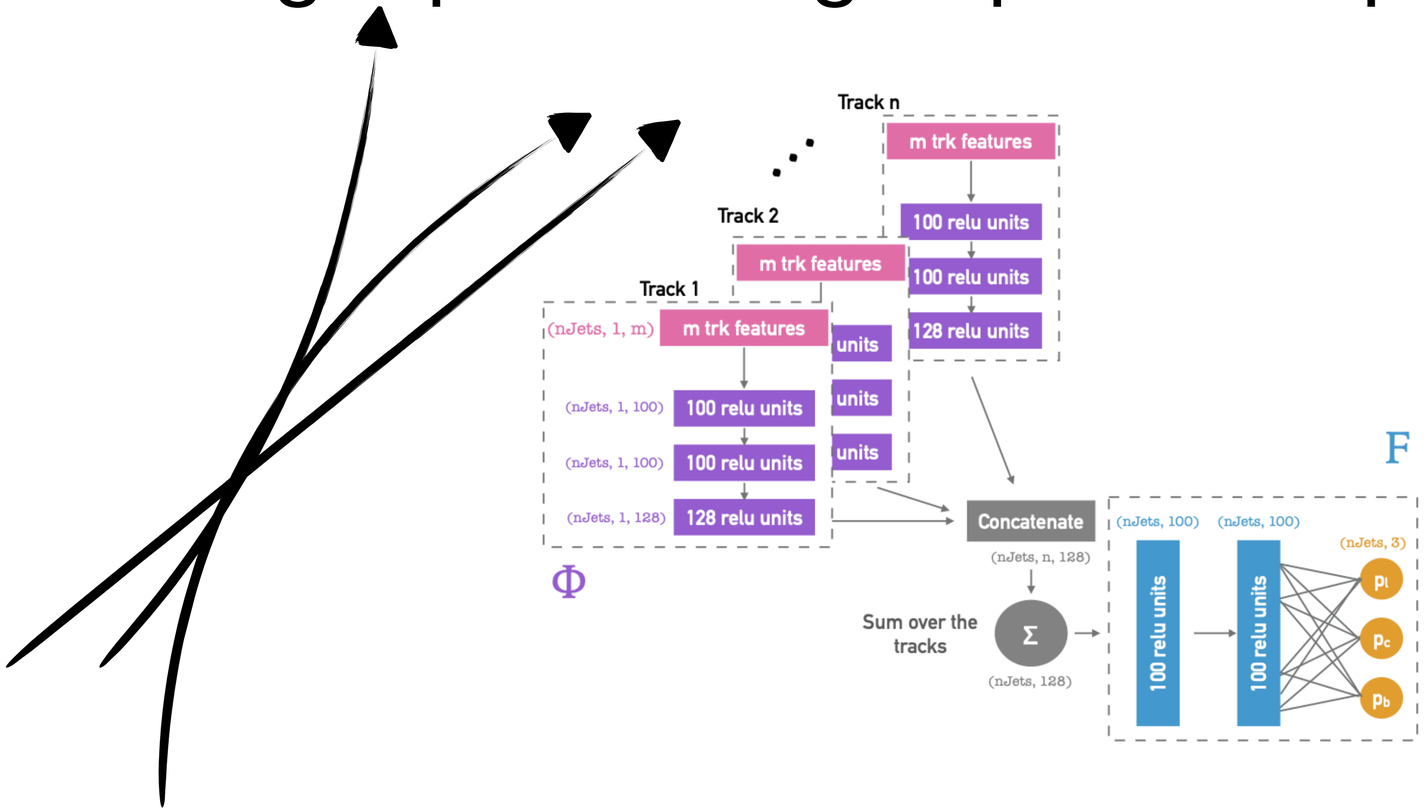
they encode a set of object into some representation

Example

Particle Physics often deals w/ set of objects w/ no order

Here: per-tracks embedding + processing of pooled rep.





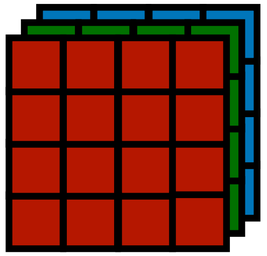
Graphs w/ Edges

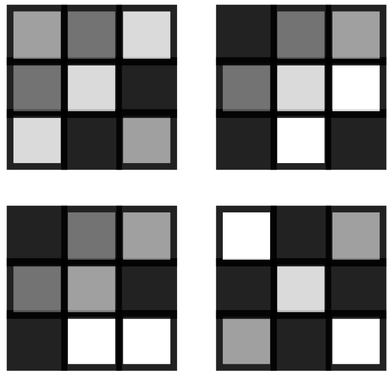
Graph Convolutions

In CNNs we had the idea local processing

- underlying concept for scalability of CNN to arbitrary sizes

How can we extend this to graphs?

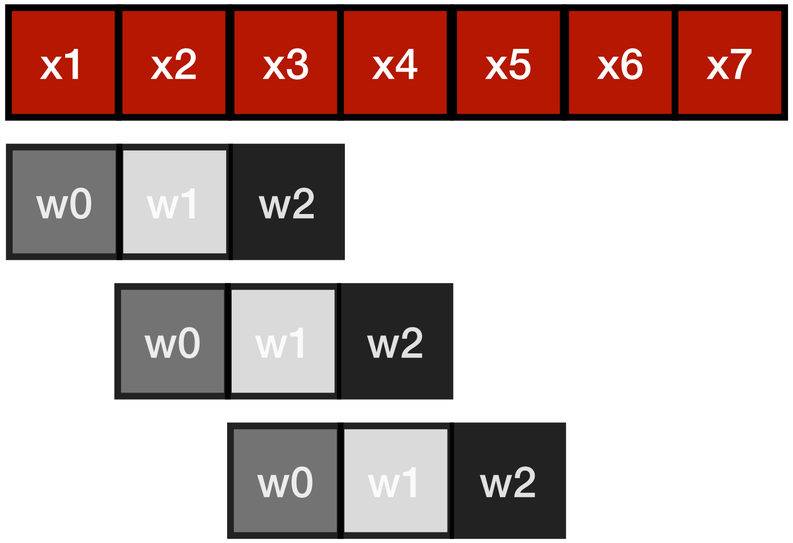




Graph Convolutions

We can go back to 1-D convolutions

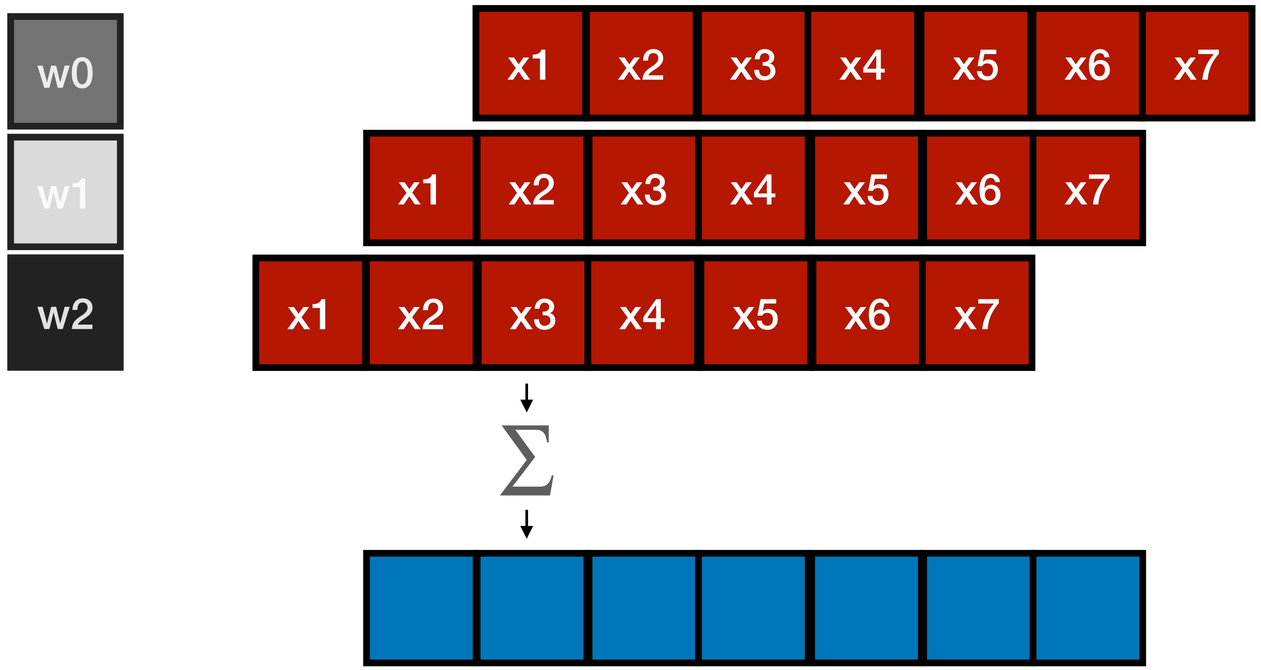
- normally we think of it at sliding the filter across



Graph Convolutions

We can go back to 1-D convolutions

- but we can also look at shifting the data



$$y_i = \sum x_{i+o} w_o$$

Graph Convolutions

We can go back to 1-D convolutions

- can rewrite x_{i+o} as the result of applying a shift operation
- convolution becomes formally like a polynomial

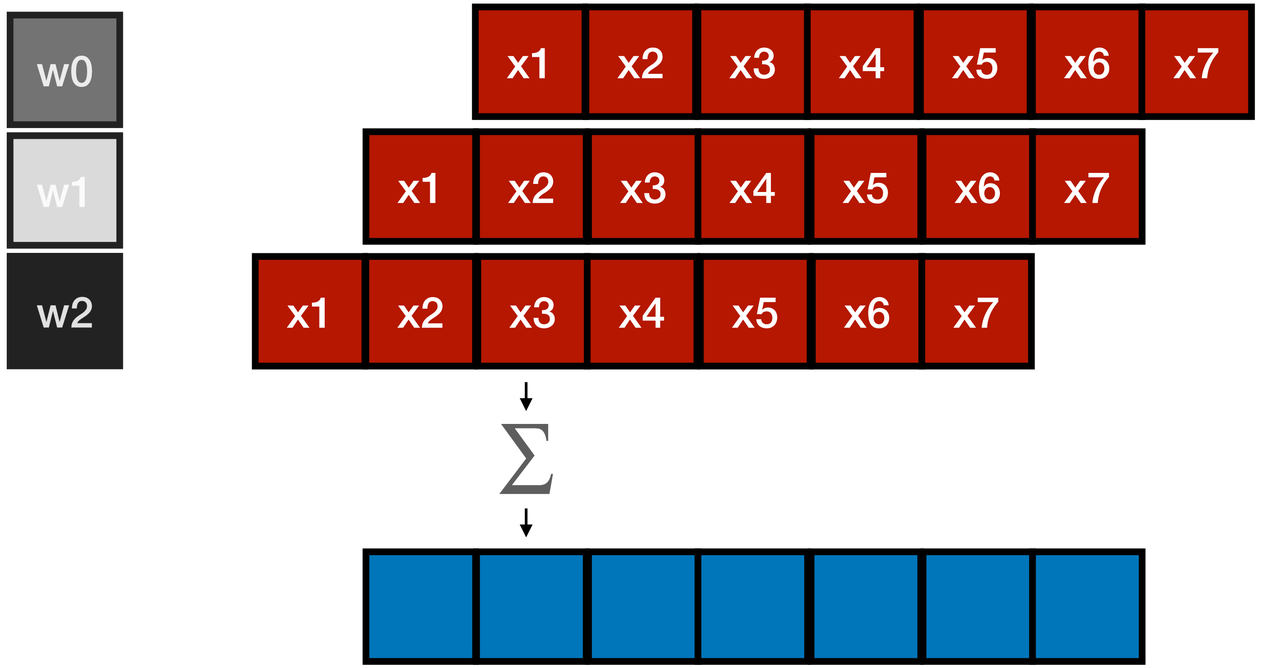
$$x_{i+1} = S(x_i)$$
$$x_{i+2} = S(S(x_i)) = S^2(x_i)$$
$$\dots$$

$$y_i = \sum x_{i+k} w_k$$

$$y_i = \sum_k S^k(x) w_k$$

higher orders in polynomial = contributions from further away

Graph Convolutions

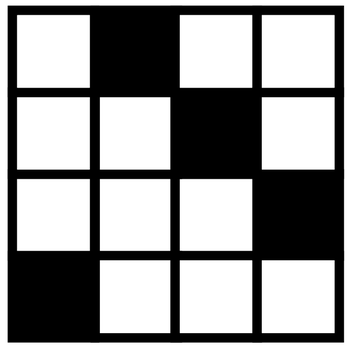


$$y_i = \sum_k S^k(x) w_k$$

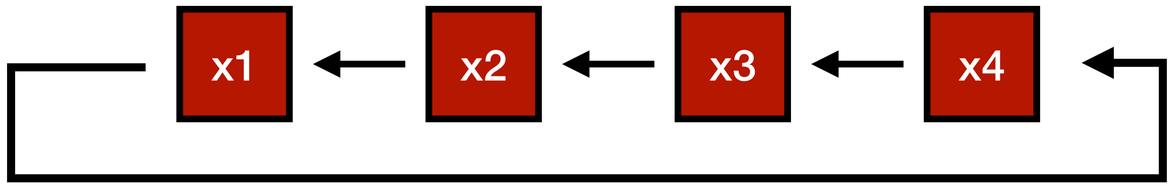
Graph Convolutions

To connect it back to the graph, picture, we can view a single line as a directed graph

- adjacency matrix = diagonal offset by 1

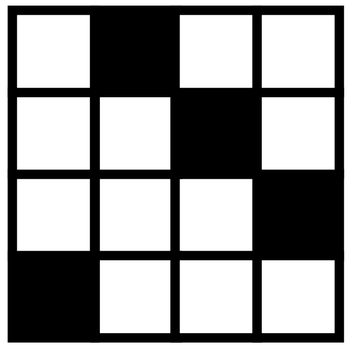
$A =$ 

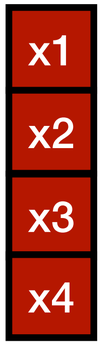
$X =$ 

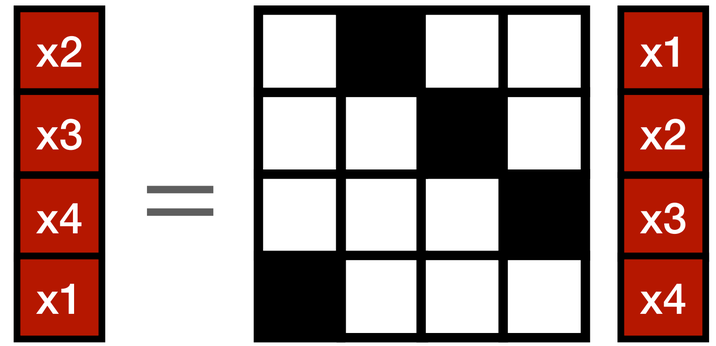


Graph Convolutions

The shift operation $S(x)$ is simply a matrix multiplication by the adjacency matrix of the graph

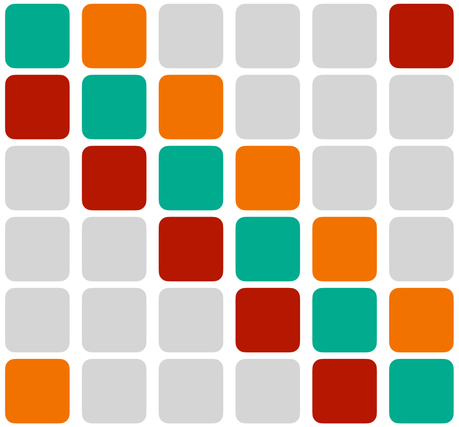
$A =$ 

$X =$ 

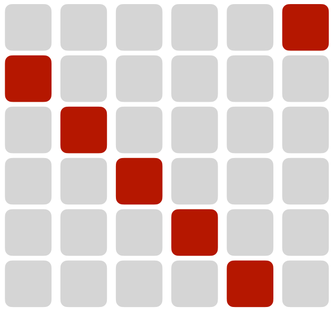


$$S(x) = AX$$

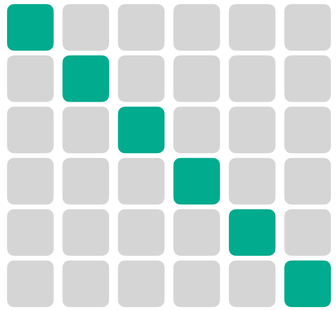
Recovering Convolution



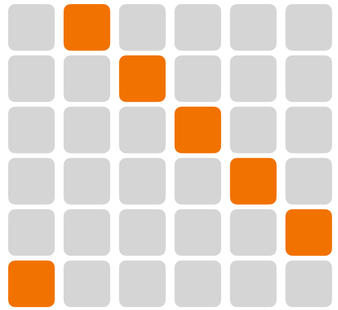
=



+

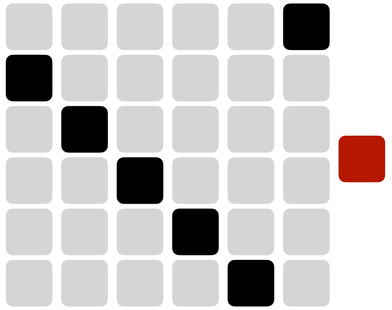


+

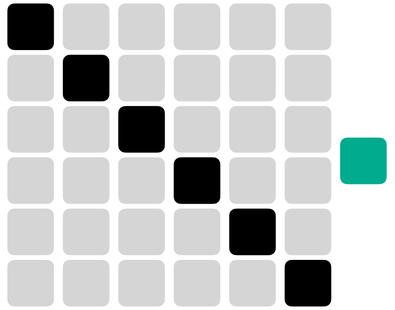


$$y_i = \sum_k S^k(x) w_k$$

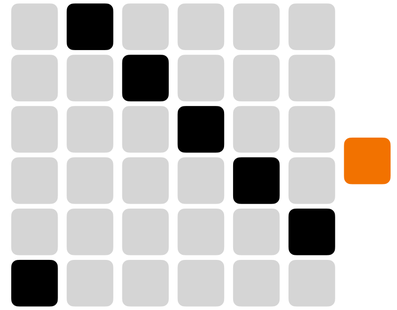
=



+



+



convolution

$S^{k=-1}$ w_{-1} $S^{k=0}$ w_0 $S^{k=1}$ w_1

Graph Convolutions

With this, we can generalize the concept of a convolution to graphs, using a generalized shift operation on graphs

Graph Filter

$$y_i = \sum_k S^k(x) w_k$$

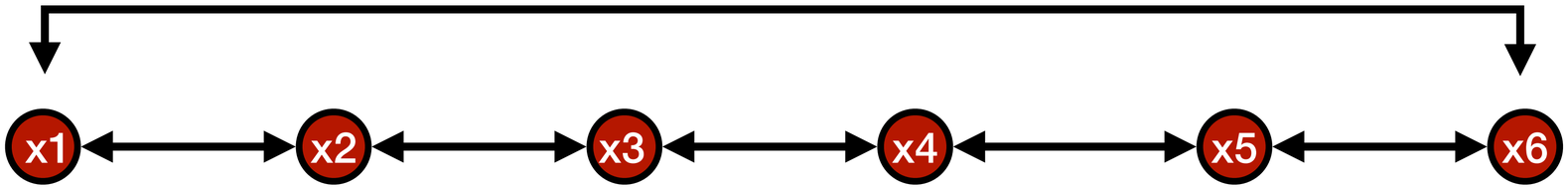
convolution

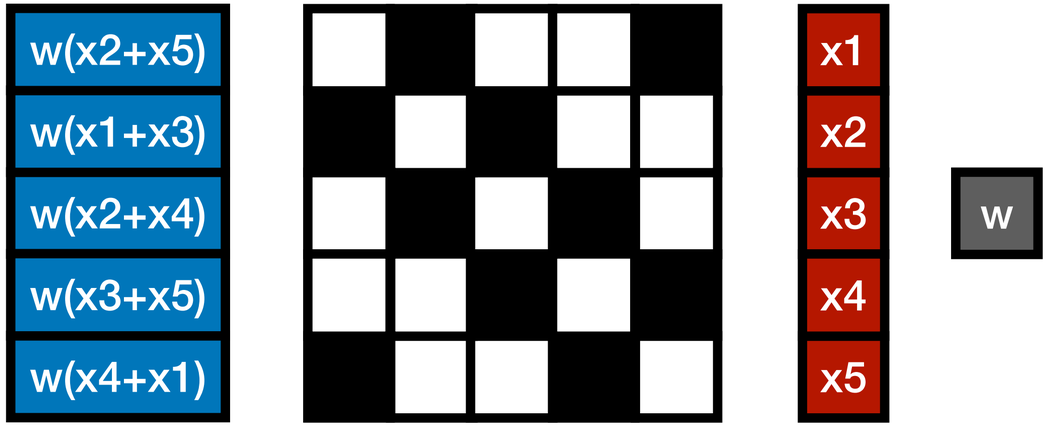
$$Y = \sum_k S^k X W_k$$

graph convolution

Graph Convolutions

Let’s look at what this means for undirected linear graphs

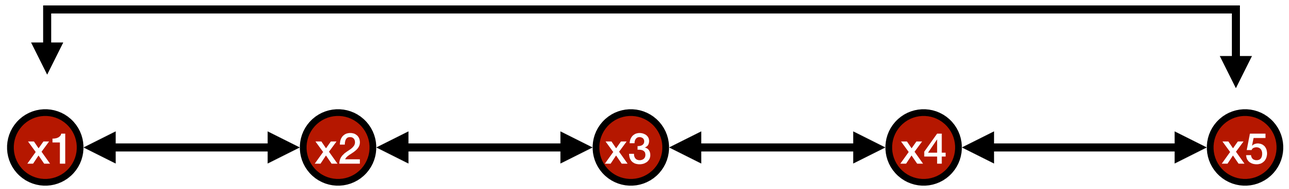




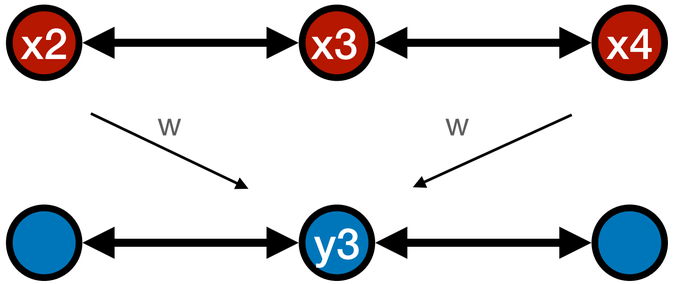
$$SXW_1$$

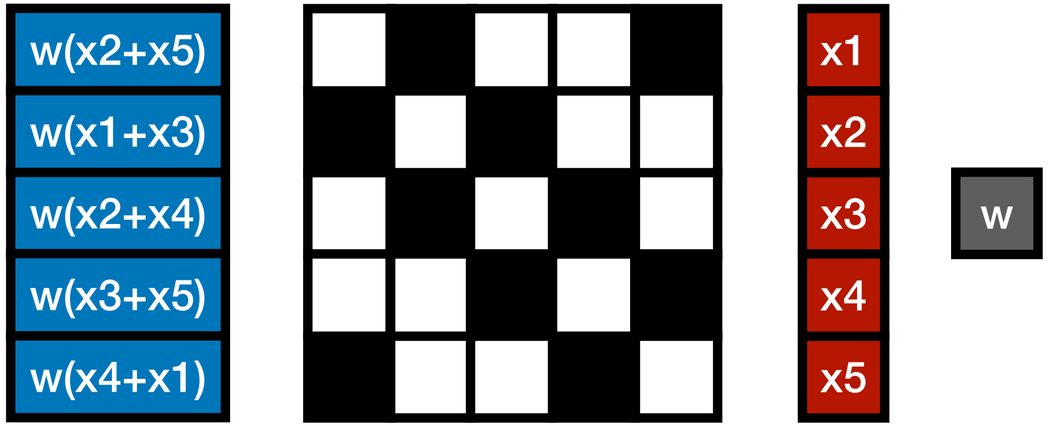
Graph Convolutions

Let's look at what this means for undirected linear graphs



Each node, gets a weighted signal from its local neighborhood, but signal is symmetric (all share the same weight)



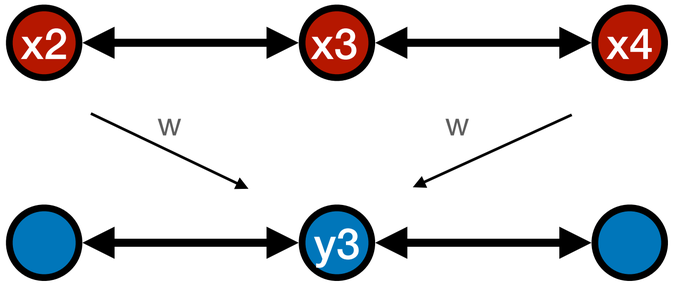


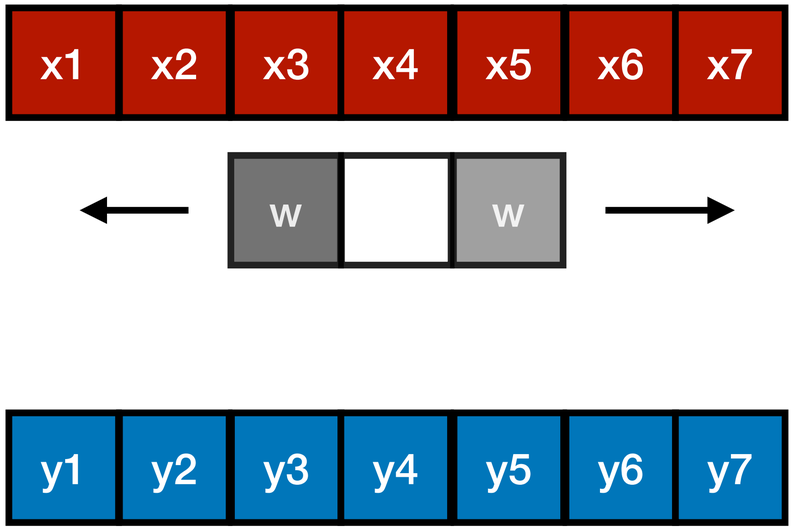
$$SXW_1$$

Graph Convolutions

A “size-1” kernel on a symmetric graph is like a symmetric convolution filter extending to the nearest neighbors

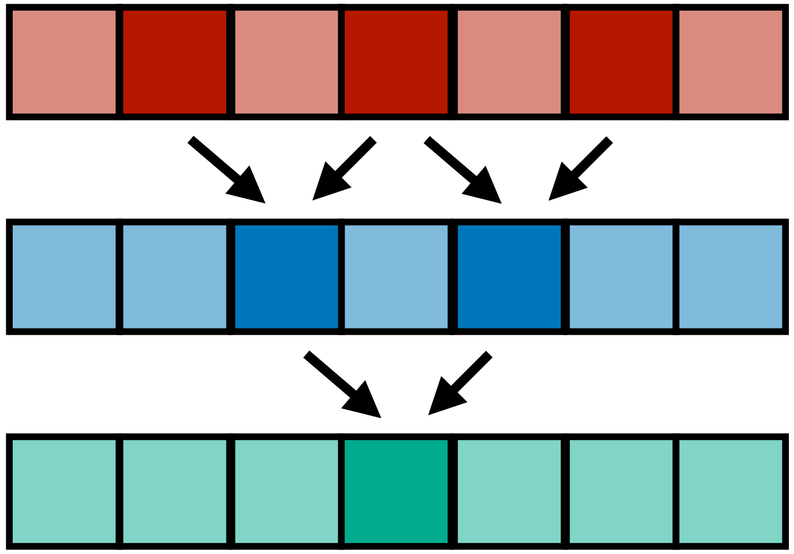
Each node, gets a weighted signal from its local neighborhood, but signal is symmetric (all share the same weight)



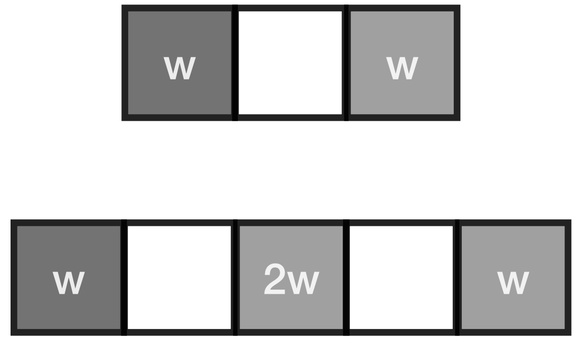


Graph Convolutions

As in the directed convolutions, higher k extend the reach of the neighborhood. Produces “radially symmetric” filters

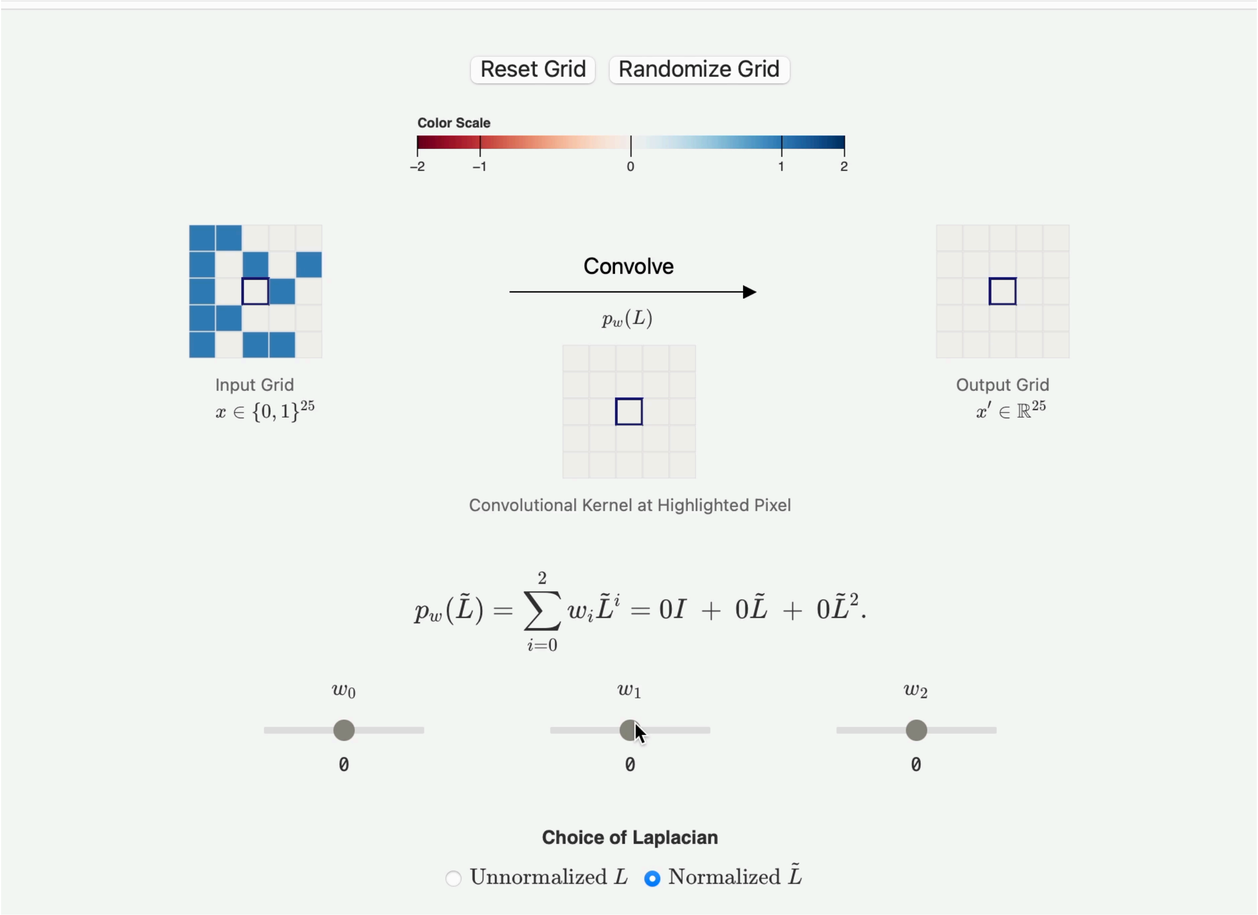


…



…

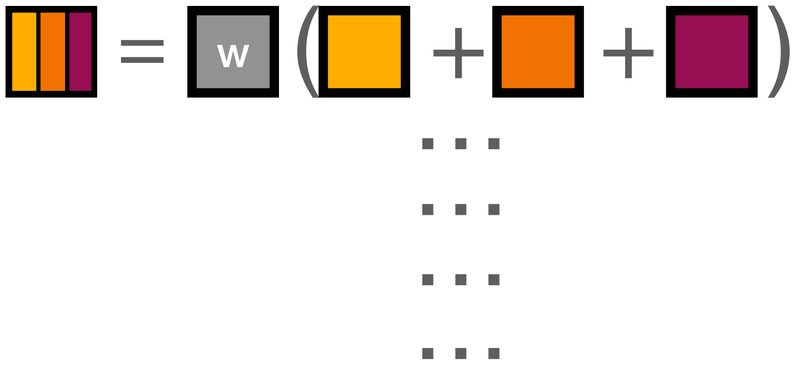
Try it yourself!

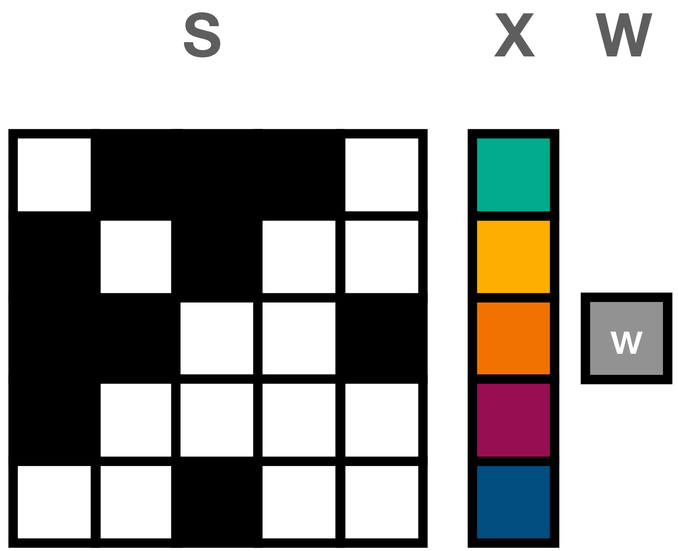


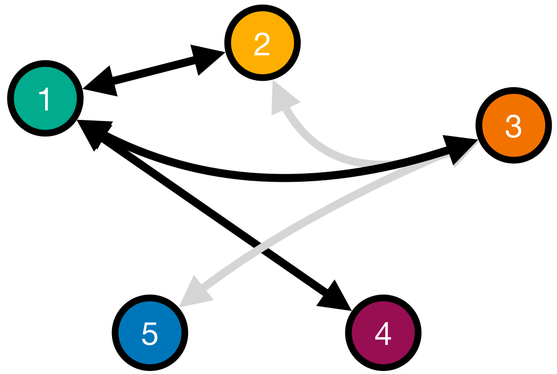
https://distill.pub/2021/understanding-gnns/

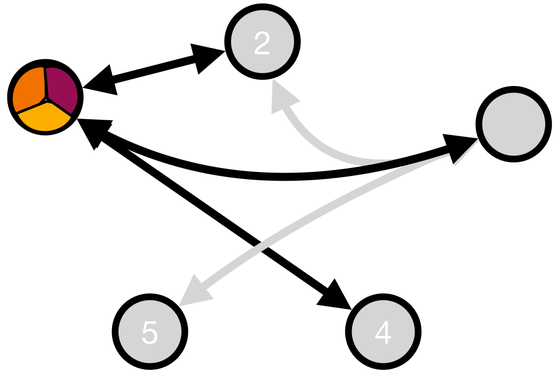
Non-linear Graphs

Our approach extends now easily to arbitrary graphs



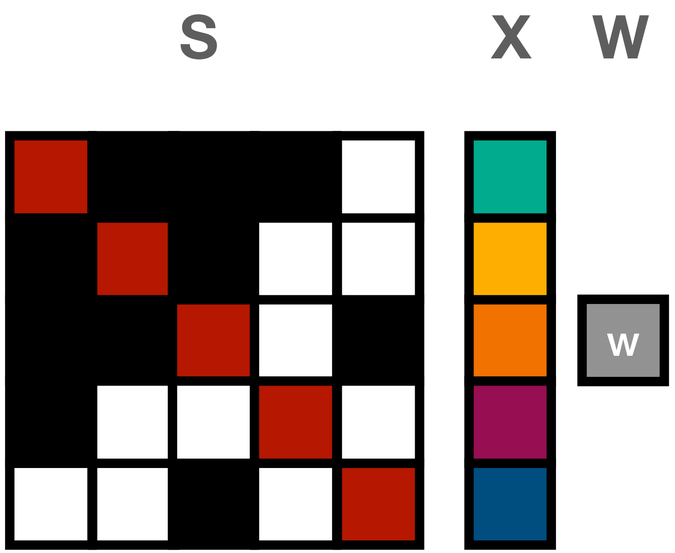


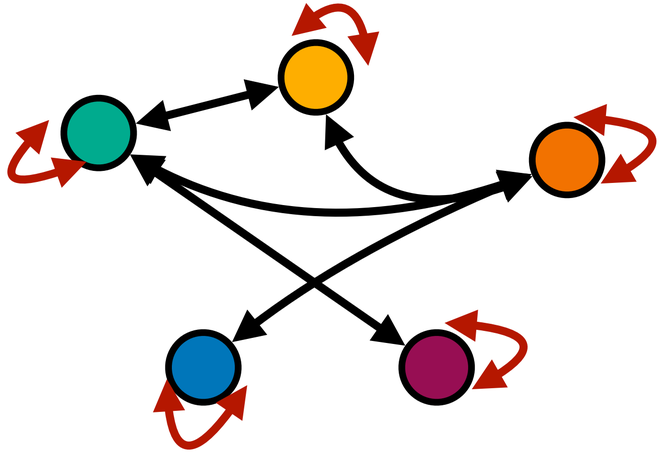




Non-linear Graphs

In order to not forget any previous state, it’s useful to add self-connections as well



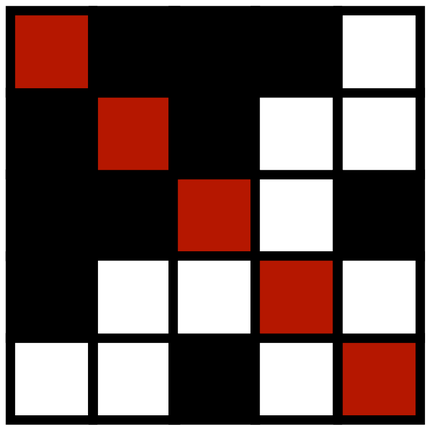


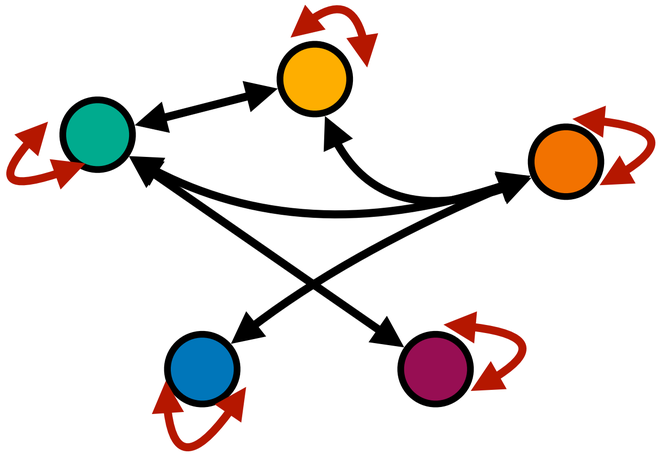
Choosing S

In practice a good choice for S is a combination of the adjacency matrix (neighbohood) and identity (self-links)

- scaled by inverse of number of edges (degree) for stability

$$S = D^{-1/2}(I + A)D^{-1/2}$$





Permutation Equivariance

Under permutation our adjancency and data transform like $A \rightarrow PAP^T$ and $X \rightarrow PX$, i.e. $S \rightarrow PSP^T$

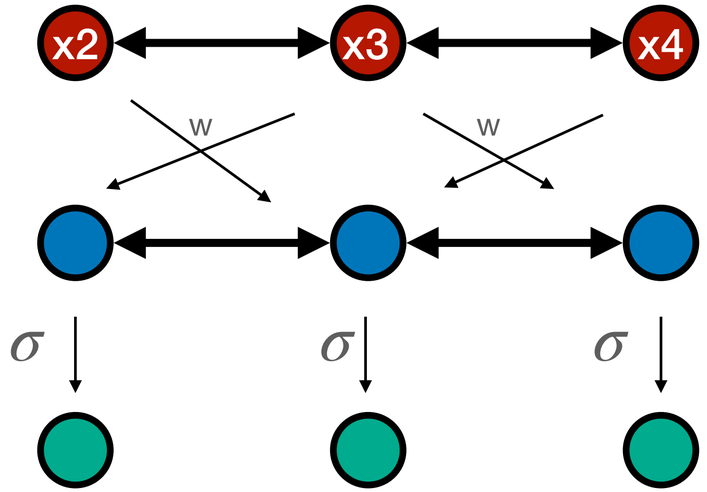
So we can see that:

$$f(X', S') = PSP^T PXW = PSXW = Pf(X, S)$$

- i.e out graph convolution is **permutation equivariant**!

Adding a non-linearity

To complete our “Graph Convolution” operation, we add a element-wise non-linearity :

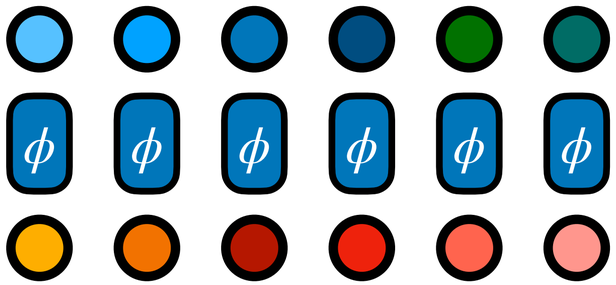


$$f(X) = \sigma(SXW)$$

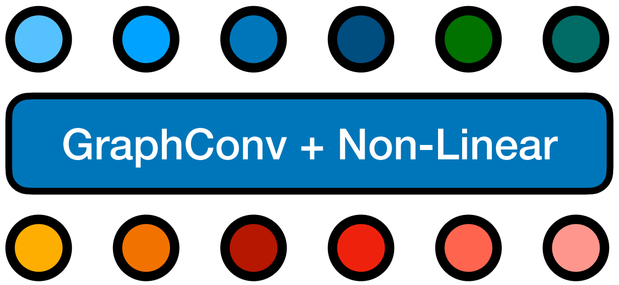
Similar to how we added non-linearities in CNN

Generalizing the Deep Set & CNN

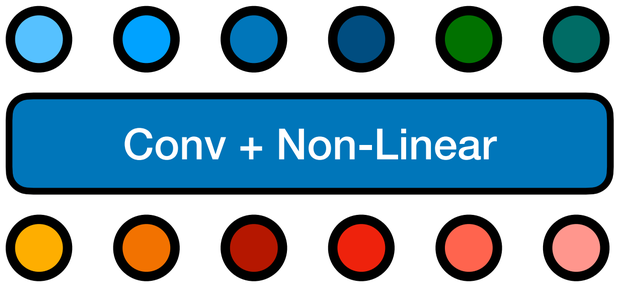
We can now build finally graph neural networks, by combining permutation-equivariant non-linear mappings



Deep Set Layer
(permutation-invariant)



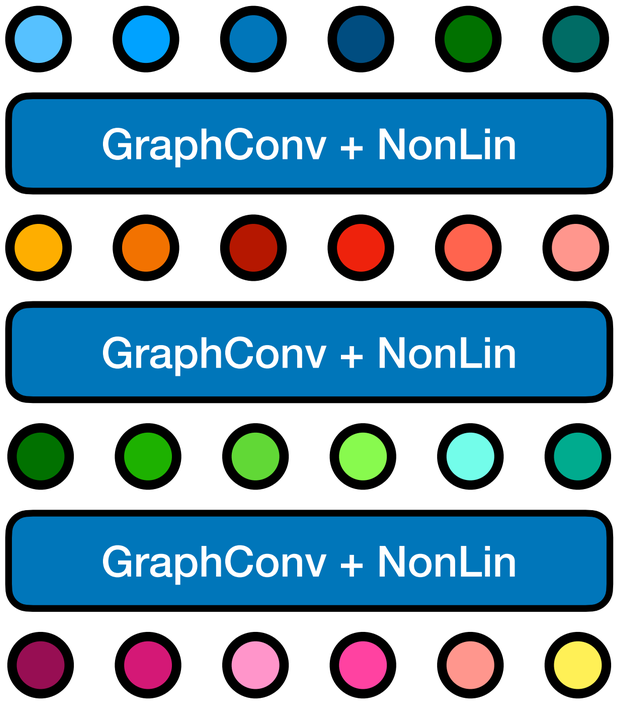
Graph Conv Layer
(permutation-invariant)

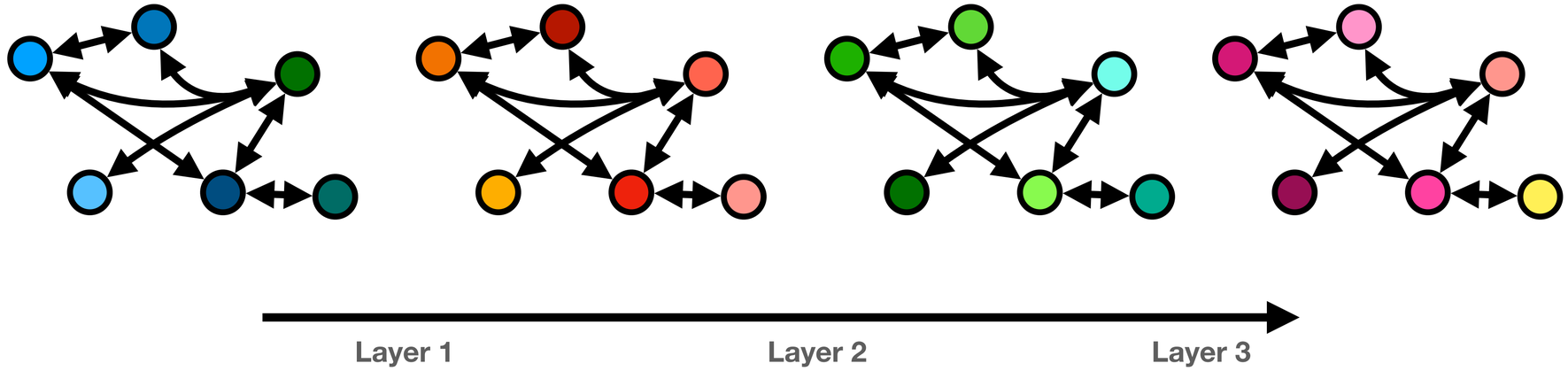


ConvNet Layer
(translation-invariant)

Building Graph Neural Nets

With Graph Convolutions & Non-linearities we can now build deep networks based on these operations

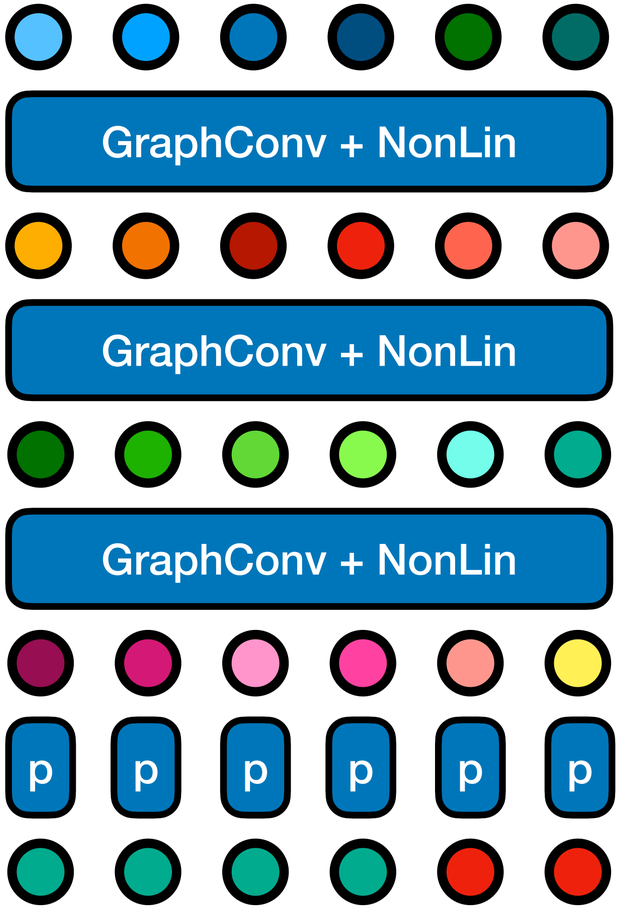


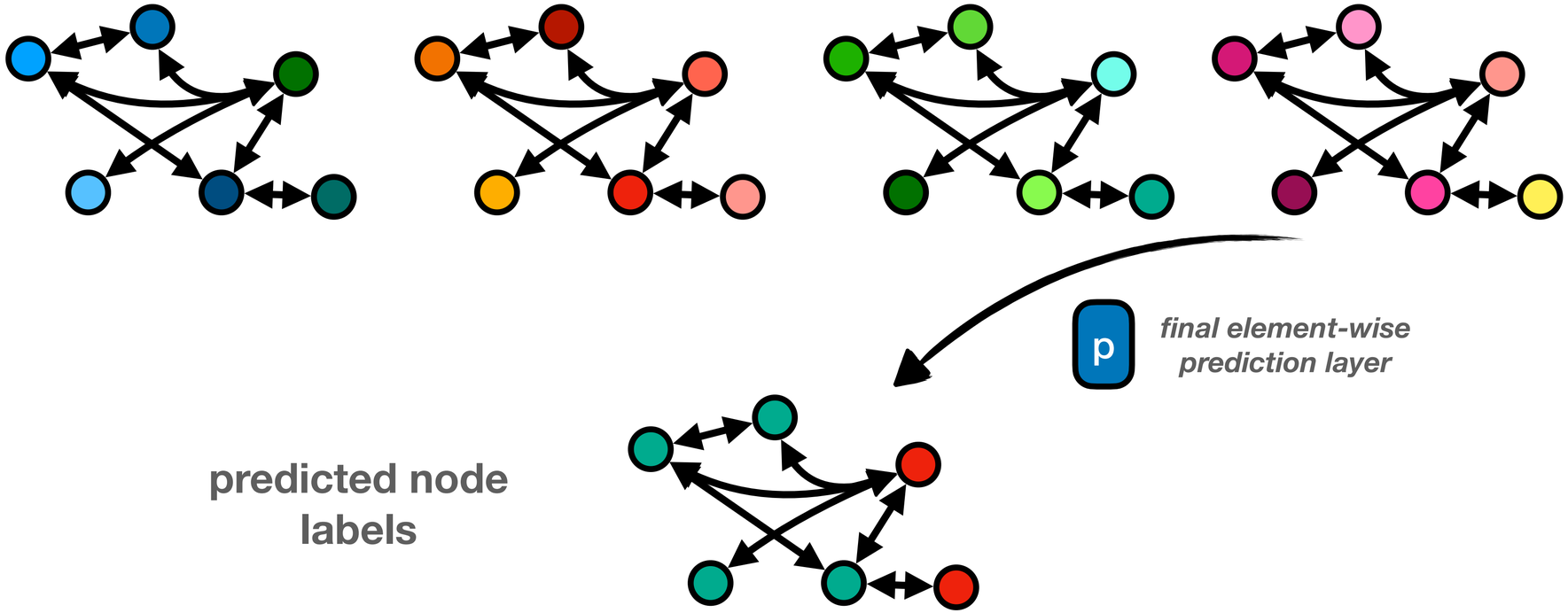


Note: the graph structure doesn't change only the node values through mutliple rounds of accumulating signals from the neighborhood

Node Prediction

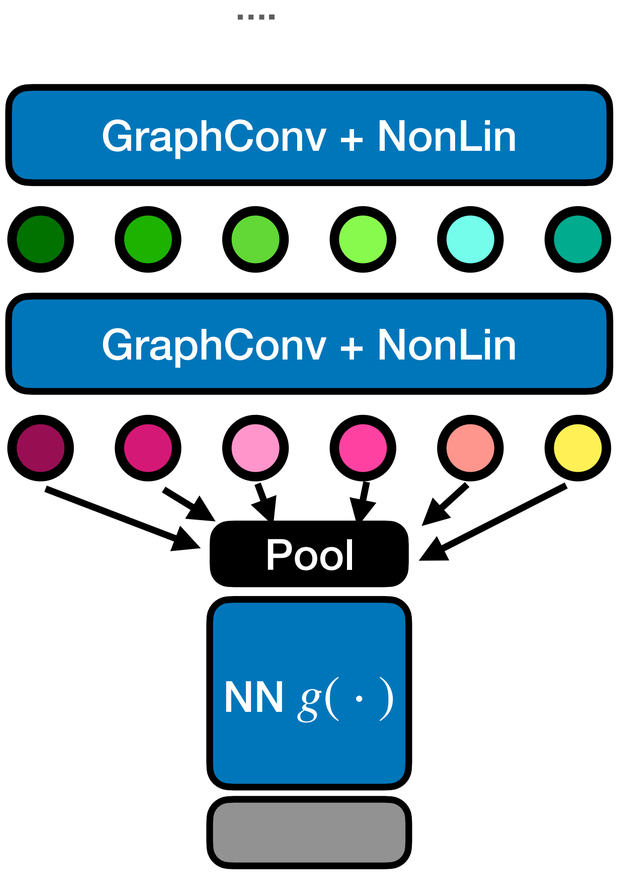
Once a suitable representation of the data (incl. neighborhood) information is collected, node prediction is elementwise

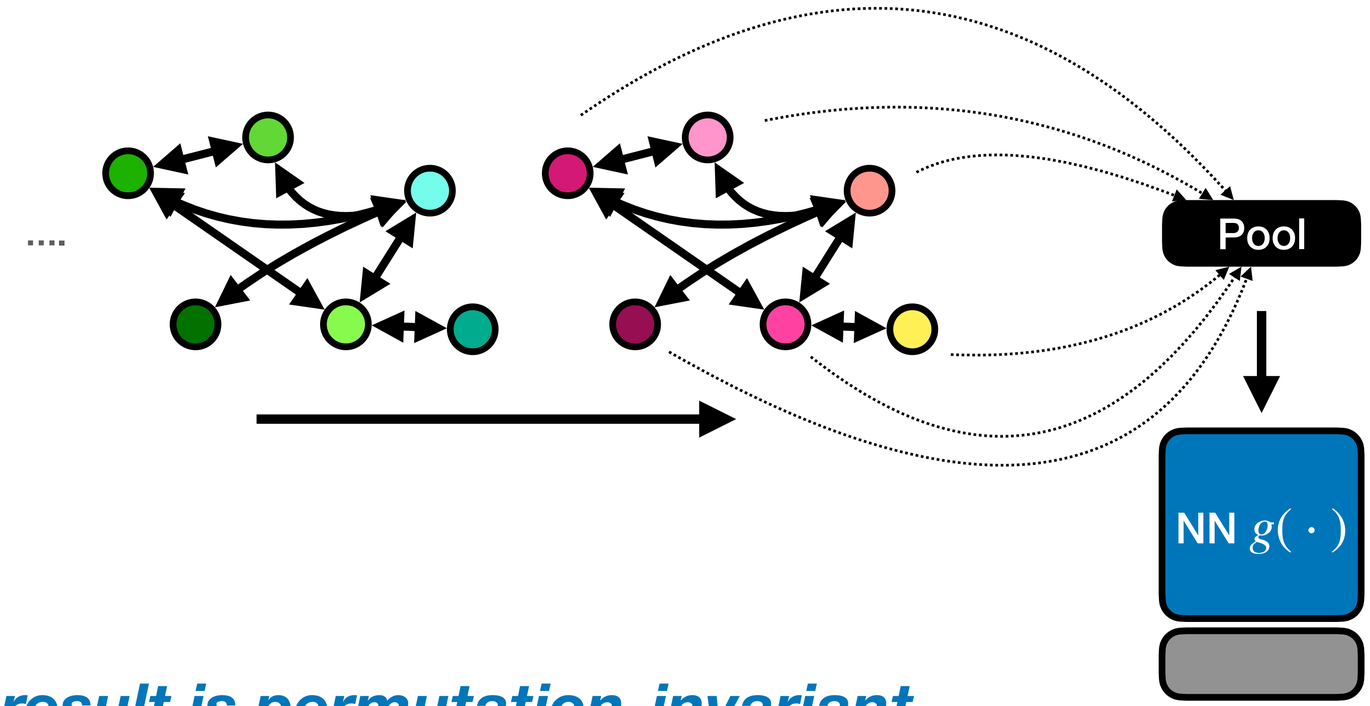




Graph Prediction

Once a suitable representation of the data (incl. neighborhood) information is collected, node prediction is elementwise

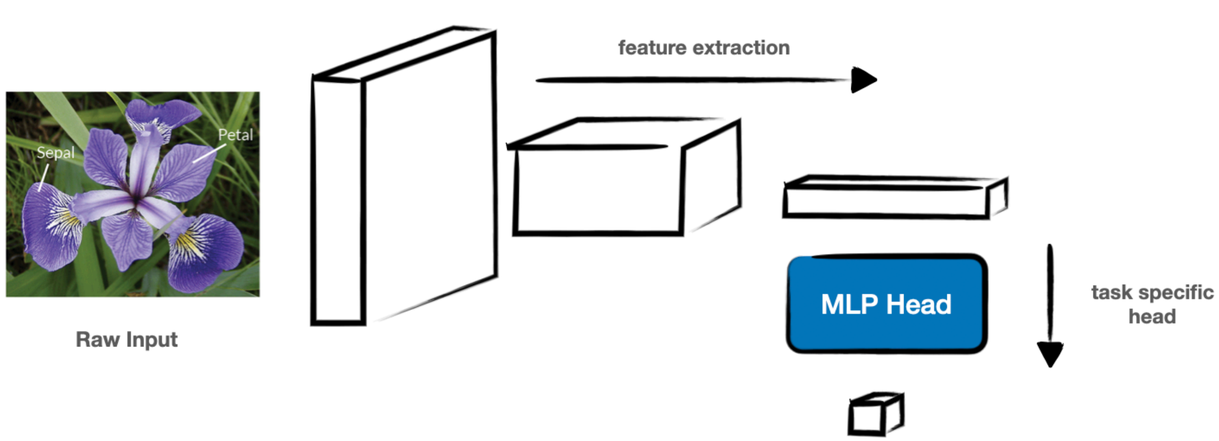


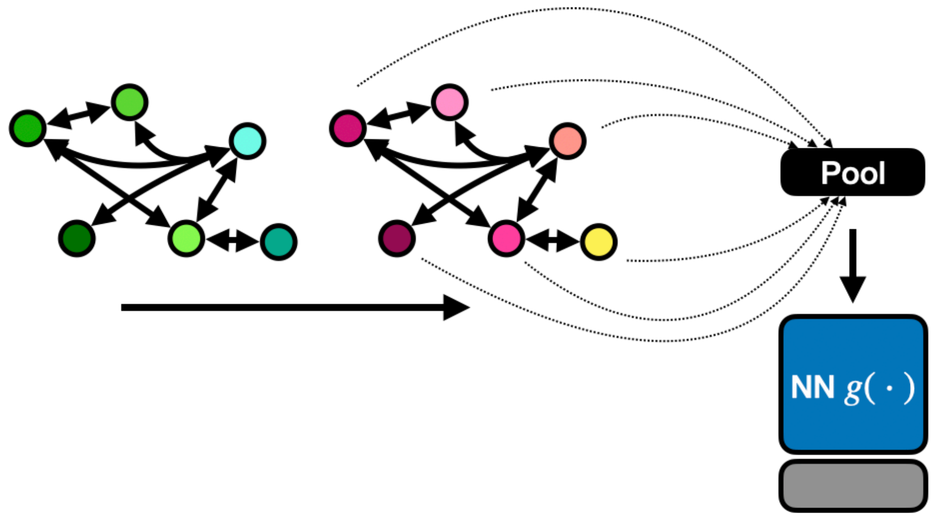


final result is permutation-invariant

CNN v GNN

Same pattern of deep learning: learned data processing (i.e. feature extraction) followed by final high-level task



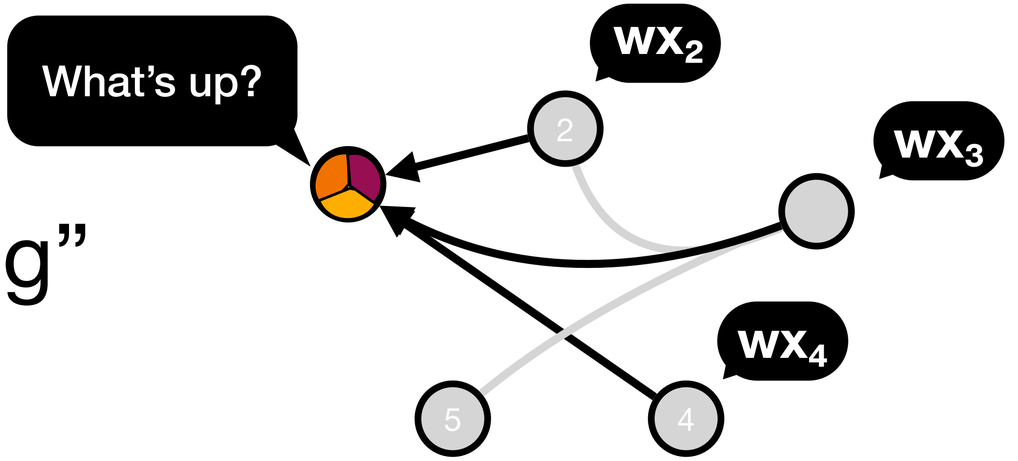


Beyond GCN

We can look at Graph Neural Nets in a more general way

- nodes sending signals (“messages”) to each other
- in graph convolution, messages are fixed by choice of S

GCN are a instance of a more general paradigm of “message passing” graph neural networks



Beyond GCN

General Setup involves three components

message creation

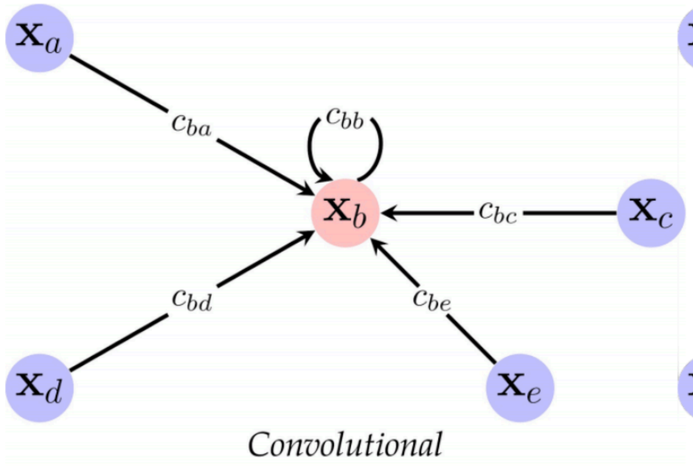
permutation invariant pooling of messages

Non-Linearity

$$\sigma(x_i, \text{Pool}_{j \in N}(m(x_i, x_j)))$$

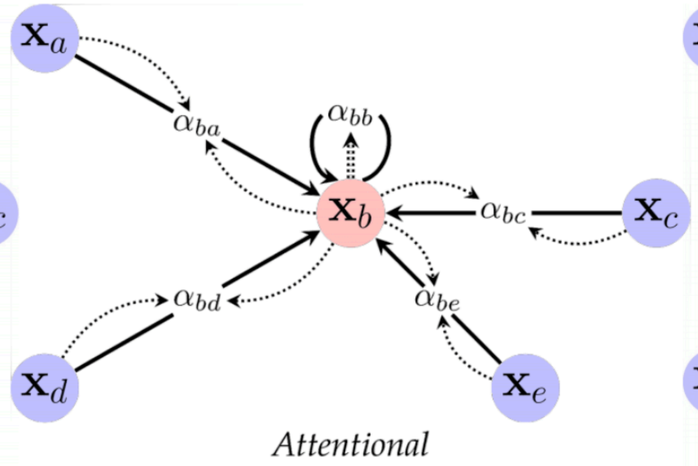
Beyond GCN

In the literature you'll find entire families of GNN types, depending on choice of message, pooling, non-linearity



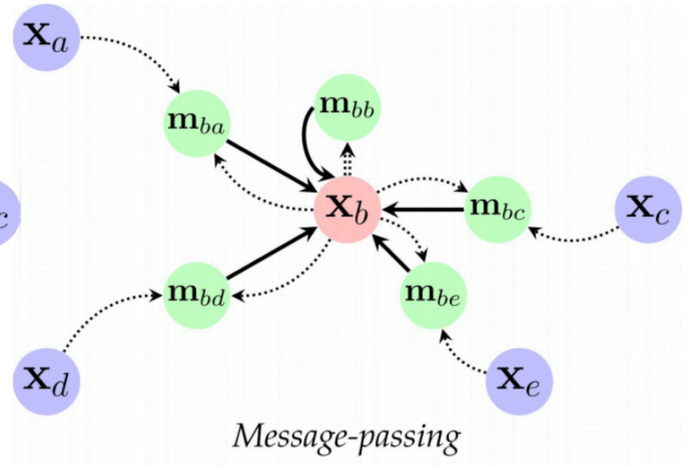
Convolutional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(\mathbf{x}_j)\right)$$



Attentional

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{x}_i, \mathbf{x}_j)\psi(\mathbf{x}_j)\right)$$



Message-passing

$$\mathbf{h}_i = \phi\left(\mathbf{x}_i, \bigoplus_{j \in \mathcal{N}_i} \psi(\mathbf{x}_i, \mathbf{x}_j)\right)$$

bit.ly/2X75EHY

Summary

Graph Neural Networks are a very broad generalization of ideas we had in CNN to “structured data”

Processes signals “on top of a graph” which is assumed to be given externally by exchanging signals across edges

Corrolary: the creation of the graph is of utmost importance as it encodes our domain knowledge (“who talks to whom”)

Some GNN do mutate the graph, but often it’s fixed.

End.