
DTU

DTU

A hands on tutorial on

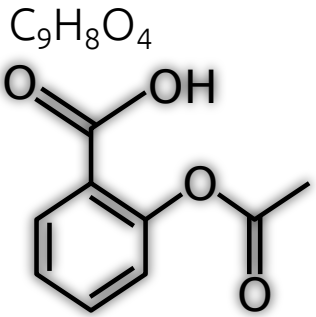
# Equivariant graph neural networks

# Agenda

- Motivation: Modeling molecules
- Graph neural networks
- Invariance and equivariance
- Exercises

# Motivation: Modeling molecules

# A molecule

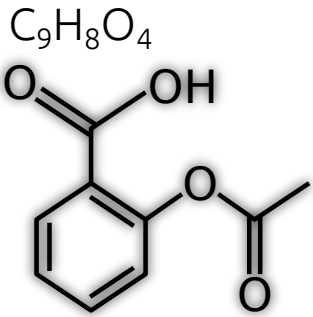



SMILES: O=C(Oc1ccccc1C(=O)O)C

# Issues with modeling molecules

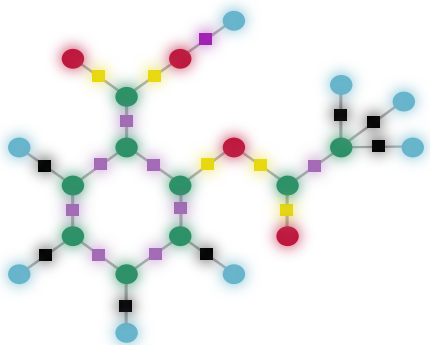
- Molecules have different number of atoms
- Atoms have different types
- Atoms have different number of bonds / connections
- Connections depend on distance and context

# A molecule

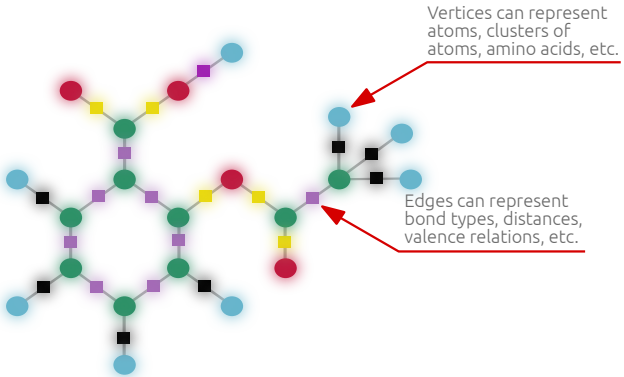



SMILES: O=C(Oc1ccccc1C(=O)O)C

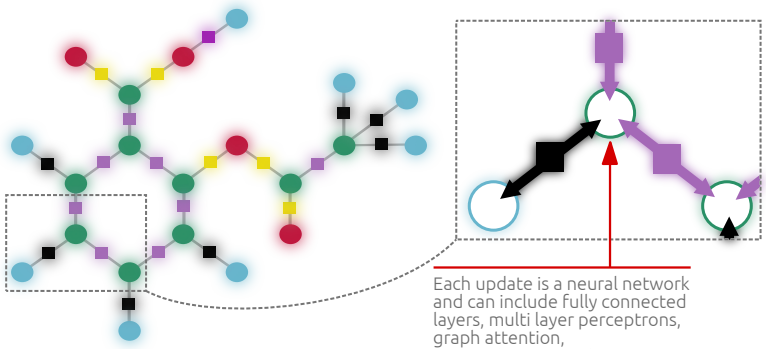
# A molecule as a graph



# Nodes and edges

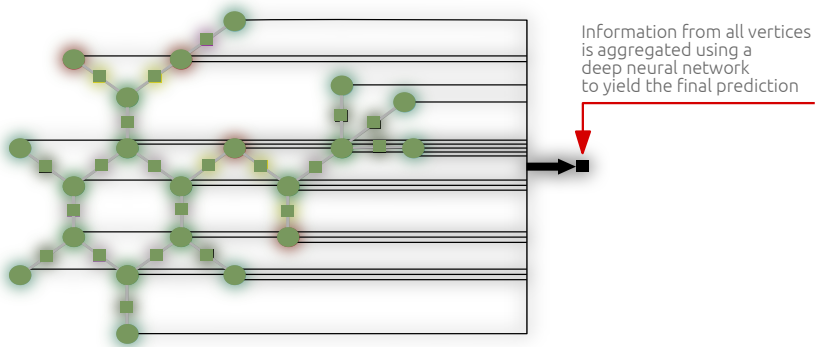


# Message passing graph neural network



Each update is a neural network and can include fully connected layers, multi layer perceptrons, graph attention, symmetic aggregation etc.

# Aggregation for graph level prediction



# Possible modeling goals

Graph classification Is a molecule is toxic or not?

Graph regression What is the solubility of the molecule?

Graph generation Generate novel molecule with desired properties

Node regression What is the net force acting on each atom?

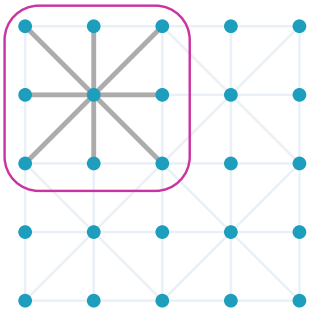
# Graph neural networks

# Message passing neural networks

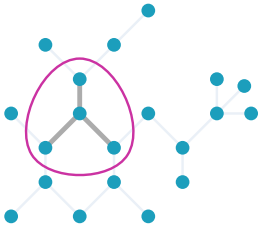
1. Nodes (and edges) have features and states (scalars, vectors, etc.)
2. States initialized with zero, or with constant/learned representation depending on node/edge type
3. Nodes send/receive messages that are neural network functions of node and edge states
4. Each node aggregates incoming messages and updates its state.
5. The message passing is repeated multiple time—with the same or different neural network messages/update
6. All nodes states are aggregated to form a graph level representation

# Types of convolutions

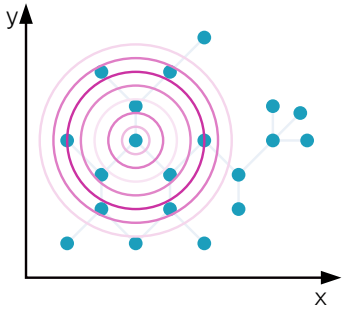
2-d convolution



Graph convolution



Continuous filter convolution



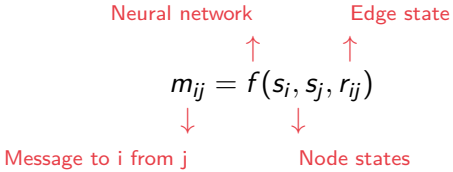

Partially adapted Zonghan Wu et al.

# Node and edge features/states

- Node features/states
  - Constant features
  - Learned representation dependent on node type
  - (Multiple) scalars, (multiple) vectors, etc.
  - Some or all updated in the aggregation step
- Edge features
  - Constant features, e.g. distance between nodes
  - Learned representation dependent on node/edge type
  - Typically not updated, but edge update GNNs exist

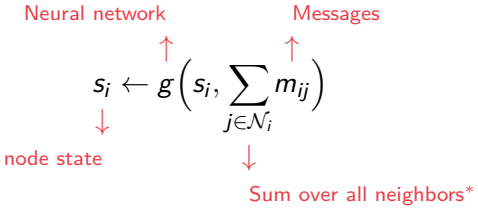
# Messages

Non-linear function of sending node, edge, and receiving node.



# Node level aggregation

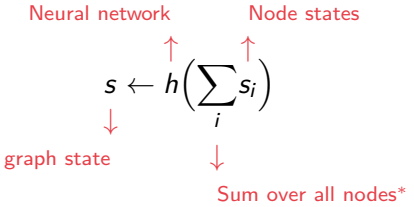
Typically a non-linear transformation of the summed messages and previous node state




*Instead of summing the messages, any permutation invariant operation could be used.

# Graph level aggregation (readout)

Typically a non-linear transformation of the summed node states




*As before, any permutation invariant operation could be used, summing node representations is often the best approach because it reflects the physics.

# Invariance and equivariance

# Handling symmetries

- Physical features change deterministicaly under certain transformations
  We do not need to learn this
  - E.g. energy of a molecule does not change with translation and rotation
  - Forces on atoms translate and rotate with the molecule
- Coordinate systems and ordering are always arbitraty
  The model should not be sensitive to this

# Symmetry-aware modeling

Three general ways to handle symmetries

1. Data augmentation (brute-force)
2. Symmetry-aware descriptors (pre-processing, constraining the data space)
3. Symmetry-aware models (built-in, constraining the function space)
   We focus on this approach

# Sequence translation invariance and equivariance

- Invariant

$$[1, 2, 3, 0, 0] \rightarrow [0, 1, 0, 0, 0]$$
$$[0, 1, 2, 3, 0] \rightarrow [0, 1, 0, 0, 0]$$
$$[0, 0, 1, 2, 3] \rightarrow [0, 1, 0, 0, 0]$$

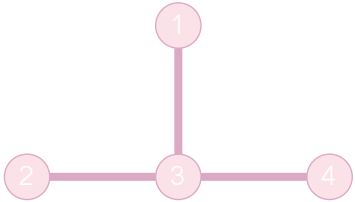
- Equivariant

$$[1, 2, 3, 0, 0] \rightarrow [0, 1, 0, 0, 0]$$
$$[0, 1, 2, 3, 0] \rightarrow [0, 0, 1, 0, 0]$$
$$[0, 0, 1, 2, 3] \rightarrow [0, 0, 0, 1, 0]$$

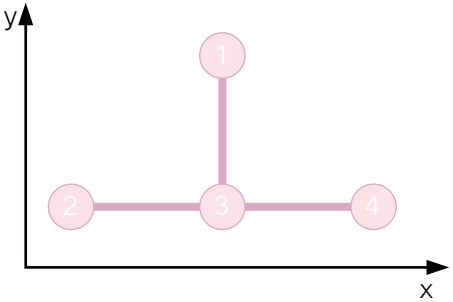
# Setting

- Problem: Graph classification
- Data is a set of graphs
- Each node has a position in a 2-d space
- We want invariance to rotation and translation

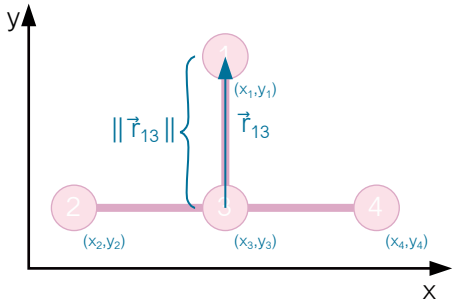
# Setting



# Setting



# Setting



# Rotation and translation of graph embedded in vector space

- Invariant

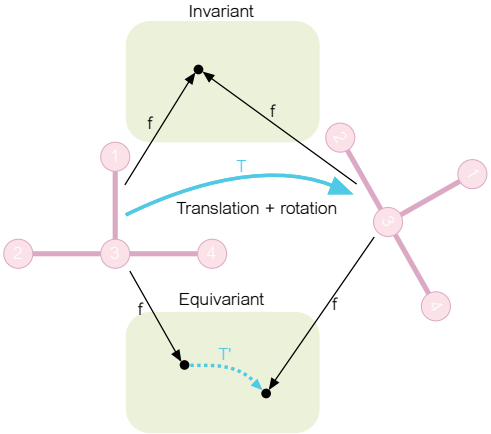
$$\vec{f}\big(T(\vec{x})\big) = \vec{f}(\vec{x})$$

- Equivariant

$$\vec{f}\big(T(\vec{x})\big) = T'\big(\vec{f}(\vec{x})\big)$$

x A node- or graph-level feature
f Some operation performed on the feature
T Rotation+translation operator
T' Corresponding operator in the output space

In all examples $x$ and $f(x)$ are in the same 2-d vector space, so we have $T = T'$




Adapted from Elise van der Pol, Daniel E. Worrall

# Vector features

Examples of invariant/equivariant operations

| | |
|---|---|
| $a \cdot \vec{v}$ | Scale |
| $\vec{v}_1 + \vec{v}_2$ | Addition/subtraction |
| $\|\|\vec{v}\|\|$ | Norm |
| $\vec{v}_1 \cdot \vec{v}_2$ | Dot product $= \|\|\vec{v}_1\|\| \cdot \|\|\vec{v}_2\|\| \cdot \cos(\theta)$ |
| $\vec{v}_1 \times \vec{v}_2$ | Cross product (3-d) |
| $\|\|\vec{v}_1 \times \vec{v}_2\|\|$ | Cross product (2-d) $= \|\|\vec{v}_1\|\| \cdot \|\|\vec{v}_2\|\| \cdot \sin(\theta)$ |

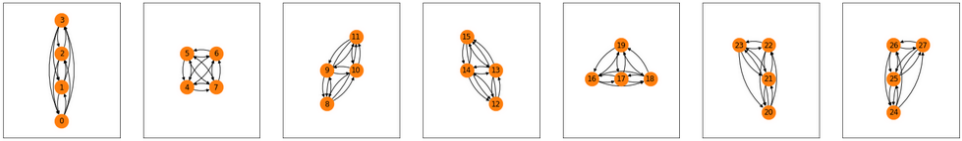
Which operations are invariant and which are equivariant?

# Why vector features?

- It is limited, what can be expressed with scalar features
- Vector features encode local geometry in more detail
- Even if we want an invariant GNN, vector features are useful.

# Exercises

# Tetris dataset



# Invariant GNN

- Messages

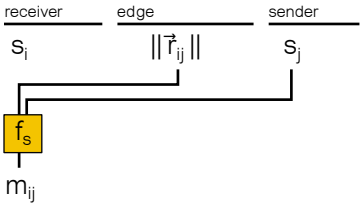
$$m_{ij} = f(s_j, \|\vec{r}_{ij}\|)$$

- Node level aggregation

$$s_i \leftarrow s_i + \sum_{j \in \mathcal{N}_i} m_{ij}$$

- Graph level aggregation

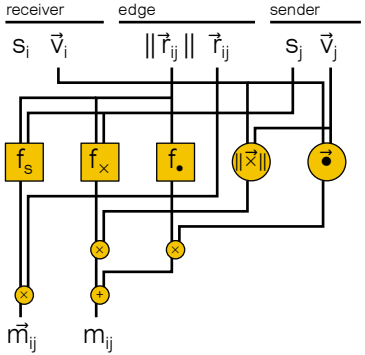
$$s = h\Big(\sum_i s_i\Big)$$



# Equivariant GNN

- Messages

$$
\begin{aligned}
m_{ij} &= f_{\cdot}(s_j, \|\vec{r}_{ij}\|)(\vec{v}_i \cdot \vec{v}_j) + \\
&\quad f_{\times}(s_j, \|\vec{r}_{ij}\|)\|\vec{v}_i \times \vec{v}_j\| \\
\vec{m}_{ij} &= f_s(s_j, \|\vec{r}_{ij}\|)\vec{r}_{ij}
\end{aligned}
$$




- Node level aggregation

$$
s_i \leftarrow s_i + \sum_{j \in \mathcal{N}_i} m_{ij}, \qquad \vec{s}_i \leftarrow \vec{s}_i + \sum_{j \in \mathcal{N}_i} \vec{m}_{ij}
$$

- Graph level aggregation

$$
s = h\Big(\sum_i s_i\Big)
$$

# Exercise

Exercise.ipynb

1. Familiarize yourself with the "tetris" graph dataset
2. Fit invariant graph neural network
3. Fit equivariant graph neural network
4. Extra at the end