Graph Neural Network

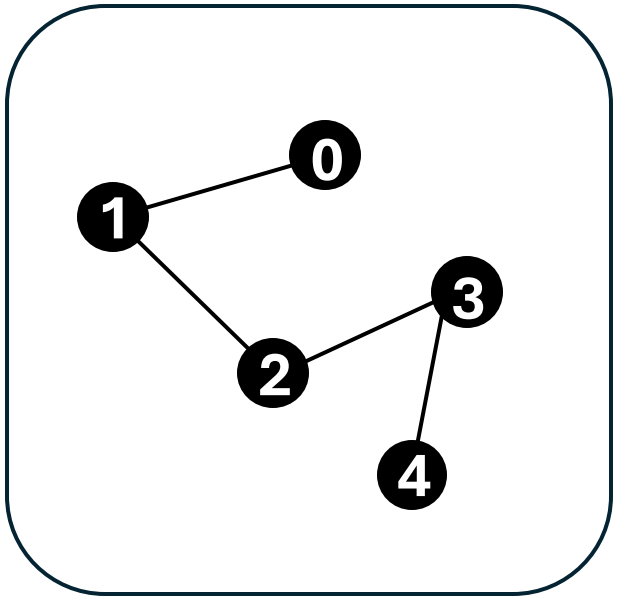
Application to Molecule Solubility Prediction

Graphs

Introduction

What is Graphs?

- **Graph is an abstract data structure in computer science**
- **“A set of objects, and the connections between them, are naturally expressed as a graph”**



$G(V, E)$

How to Represent Graph Data Structure?

Adjacency List

[[0, 1], [1, 2], [2, 3], [3, 4]]

Adjacency Matrix

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

Vertex

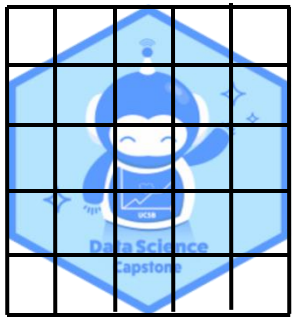
Edges

Graphs

Graph Data

What type of data can be represented as Graphs?

Images



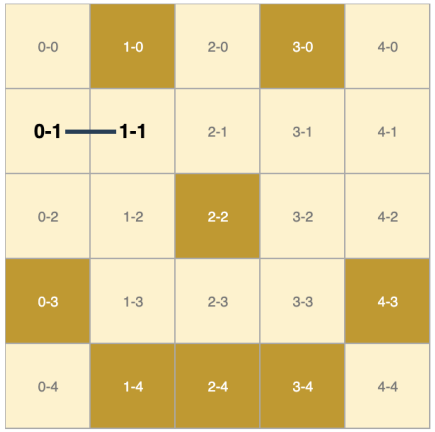
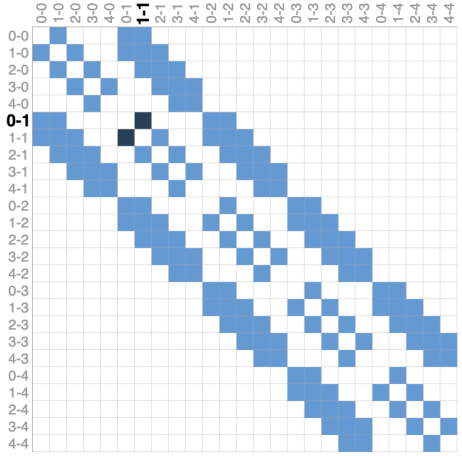
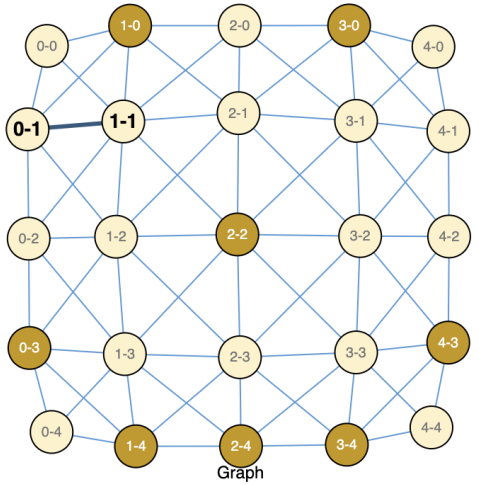


Image Pixels



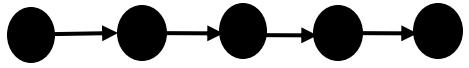
Adjacency Matrix

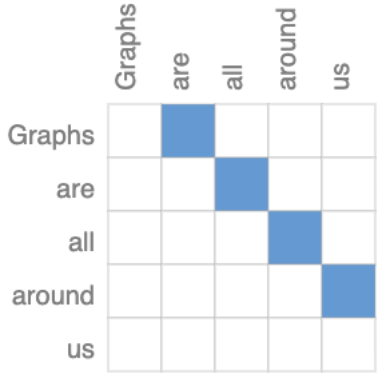


Graph

Texts

Graph are all around us



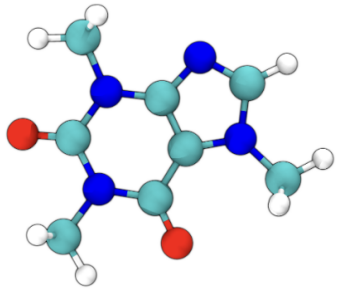


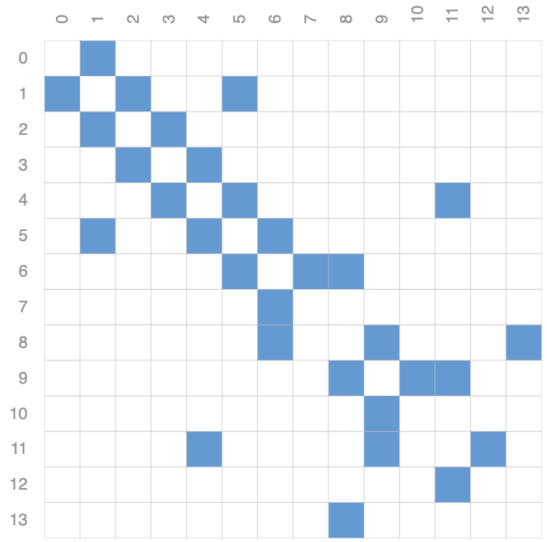
Graphs

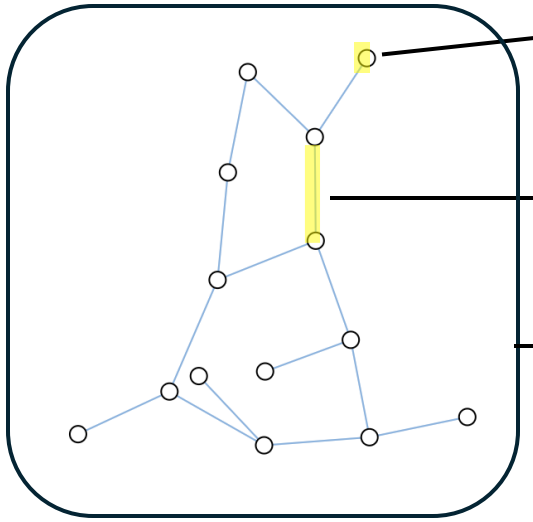
Focus

Our Focus: Graph-level Prediction

Molecular Structure





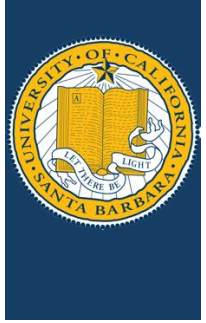


Node Features (Atom):
[atomic_num, formal_charge, ...]

Edge Features (Bond):
[bond_type, stereo, ...]

Graph Features (Property):
[solubility]

https://pytorch-geometric.readthedocs.io/en/2.6.1/_modules/torch_geometric/utils/smiles.html



Overview

ESOL

```python
# use rdkit to generate molecule features while importing dataset
import rdkit
from torch_geometric.datasets import MoleculeNet

# Load the ESOL dataset
data = MoleculeNet(root='.', name="ESOL")
data
```

Packages: PyTorch geometric, RDKit

Dataset: torch_geometric.datasets.**MoleculeNet**

ESOL is a small dataset consisting of water solubility data for 1128 compounds.

ESOL → SMILES `CN1C=NC2=C1C(=O)N(C(=O)N2C)C` → PyTorch geometric RDKit `from_smiles()` → *torch_geometric.data.Data instance*

Data.x [['atomic_num', 'chirality' , 'degree', 'formal_charge', num_hs', 'num_radical_electrons', 'hybridization', 'is_aromatic', 'is_in_ring']]

Data.y [[solubility]]

Data.edge_att [['bond_type', 'stereo', 'is_conjugated']]

Data.edge_index Adjacency list

https://pytorch-geometric.readthedocs.io/en/2.6.1/_modules/torch_geometric/utils/smiles.html

Data

A close look to the Dataset

Data.x

```
# Investigating the features
# Shape: [num_nodes, num_node_features]
data[0].x
```

✓ 0.0s

```
tensor([[8, 0, 2, 5, 1, 0, 4, 0, 0],
        [6, 0, 4, 5, 2, 0, 4, 0, 0],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 1],
        [6, 0, 4, 5, 1, 0, 4, 0, 1],
        [8, 0, 2, 5, 0, 0, 4, 0, 0],
```

Data['smiles']

```
data[0]["smiles"]
```

✓ 0.0s

```
'OCC3OC(OCC2OC(OC(C#N)c1ccccc1)C(O)C(O)C2O)C(O)C(O)C3O '
```

Data.edge_index

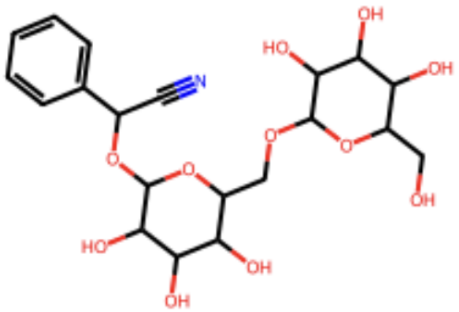
```
# Investigating the edges in sparse COO format
# Shape: [2, num_edges]
# i.e. node 0 is connected to node 1
# i.e. node 1 is connected to node 0
# i.e. node 1 is connected to node 2
# etc.
# NOTE: more efficient representation than adjacency matrix
data[0].edge_index.t()
```

✓ 0.0s

```
tensor([[ 0,  1],
        [ 1,  0],
        [ 1,  2],
        [ 2,  1],
        [ 2,  3],
        [ 2, 30],
```

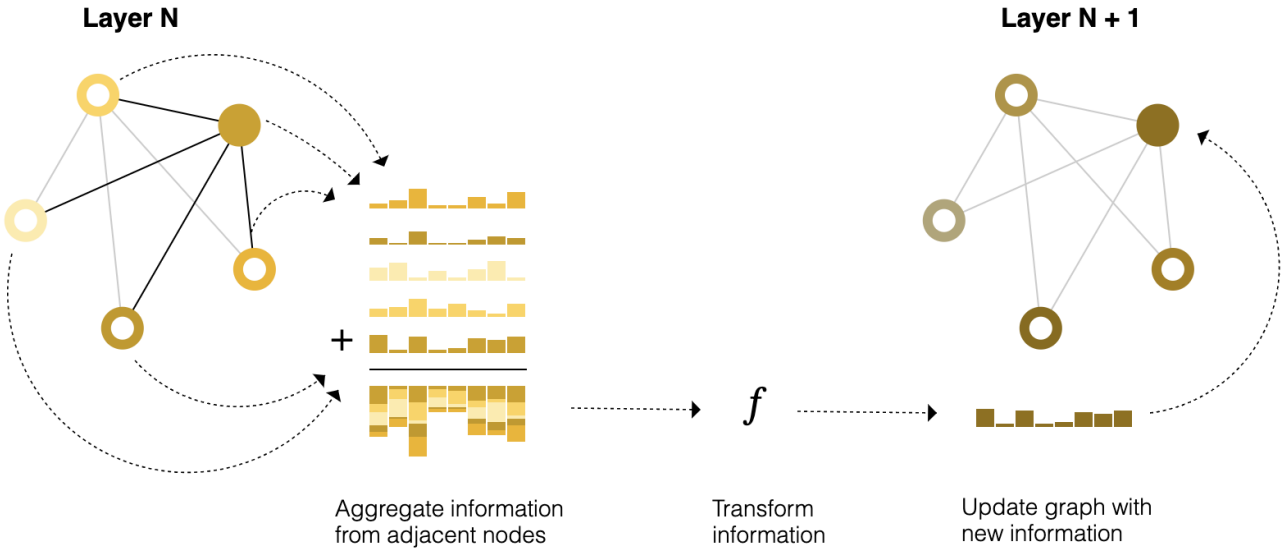
```
from rdkit import Chem
from rdkit.Chem.Draw import IPythonConsole
molecule = Chem.MolFromSmiles(data[0]["smiles"])
molecule
```

✓ 0.0s

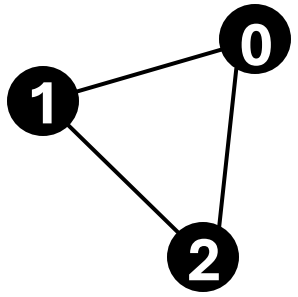


Graph Convolutional Neural Network

Model



$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$$



$$A = \begin{matrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix} \qquad X = \begin{bmatrix} 1 & 2 \\ -3 & 2 \\ 2 & 1 \end{bmatrix}$$

$$\Theta = \begin{matrix} w_{00} & w_{01} & w_{02} & w_{03} & w_{04} & w_{05} \\ w_{10} & w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \end{matrix}$$

Adjacency Matrix

$$\hat{A} = A + I = \begin{matrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix} + \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$= \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Normalized Degree Matrix

$$\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$$

$$\widehat{D^{-1/2}} = \begin{matrix} 1/\sqrt{3} & 0 & 0 \\ 0 & 1/\sqrt{3} & 0 \\ 0 & 0 & 1/\sqrt{3} \end{matrix}$$

Projected to Higher Dimensions

$\mathbf{X}\mathbf{\Theta}$ $3 \times 2 \;\; 2 \times 6 = 3 \times 6$

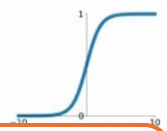
```
# GCN layers
self.initial_conv = GCNConv(data.num_features, embedding_size)
self.conv1 = GCNConv(embedding_size, embedding_size)
self.conv2 = GCNConv(embedding_size, embedding_size)
self.conv3 = GCNConv(embedding_size, embedding_size)

# Output layer
self.out = Linear(embedding_size*2, data.y.shape[1]) # output shape 1 = regression
# twice the size for accomodating the global pooling layer
```

Graph Convolutional Neural Network

Forward Propagation

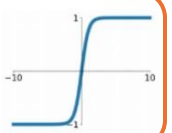
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



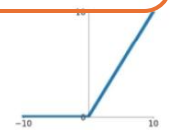
tanh

$\tanh(x)$



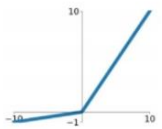
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

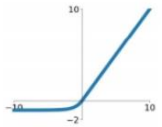


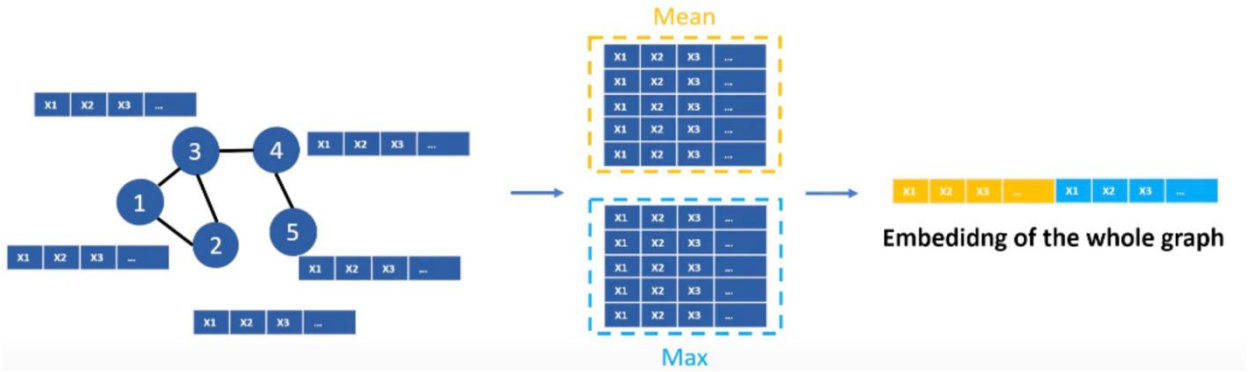
Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

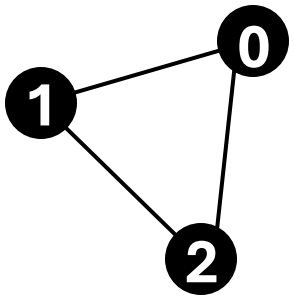
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$





$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta},$$



$\tanh(X'_{3\times 6})$

```python
def forward(self, x, edge_index, batch_index):
    # First Conv layer
    hidden = self.initial_conv(x, edge_index)
    hidden = F.tanh(hidden) # activation function

    # Other Conv layers
    hidden = self.conv1(hidden, edge_index)
    hidden = F.tanh(hidden)
    hidden = self.conv2(hidden, edge_index)
    hidden = F.tanh(hidden)
    hidden = self.conv3(hidden, edge_index)
    hidden = F.tanh(hidden)
```

```python
# Global Pooling Layer (stack different aggregations)
hidden = torch.cat([gmp(hidden, batch_index),
                    gap(hidden, batch_index)], dim=1)
# gmp = global max pooling
# gap = gloabl average pooling
# batch_index is used to select the relavent nodes for the pooling operation

# Apply a final (linear) classifier.
out = self.out(hidden)

return out, hidden
```