Tutorial 2: Project 1 / GCN

2020.10.29

Qizhi Li

Outline

- Drug Molecular Toxicity Prediction
- Introduction of GCN

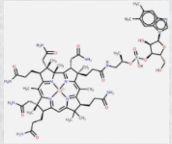
Drug Molecular Toxicity Prediction

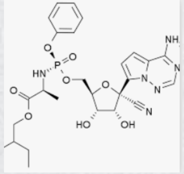
- Task
- SMILES
- AUC
- Submission Requirements

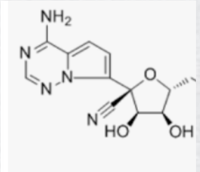
Task

Learn

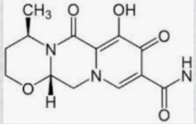
Toxic

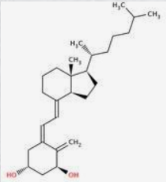


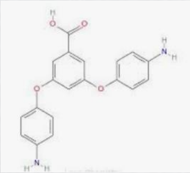




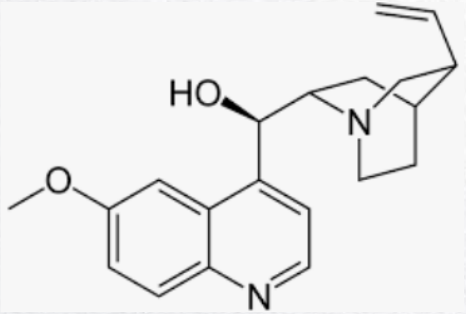
Non toxic







Predict



Toxic

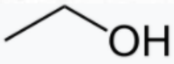
Or

Non toxic

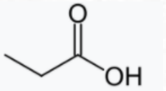
SMILES (Simplified Molecular Input Line Entry System)

Allows a user to represent a chemical structure in a way that can be used by the computer

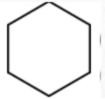
1. **Simple Chains**: Combining atomic symbols and bond symbols. `CCO`



2. **Branches**: the branch between parenthesis. `CCC(=O)O`



3. **Rings**: identify the opening and closing ring atom. **C1**CCCC**C1**



4. **Charged Atoms**: The Charged Atoms between brackets. `[Ti+4]` `[Ti++++]`

Ti^{4+}

Connection signal:

-	Single bond
=	Double bond
#	Triple bond
*	Aromatic bond

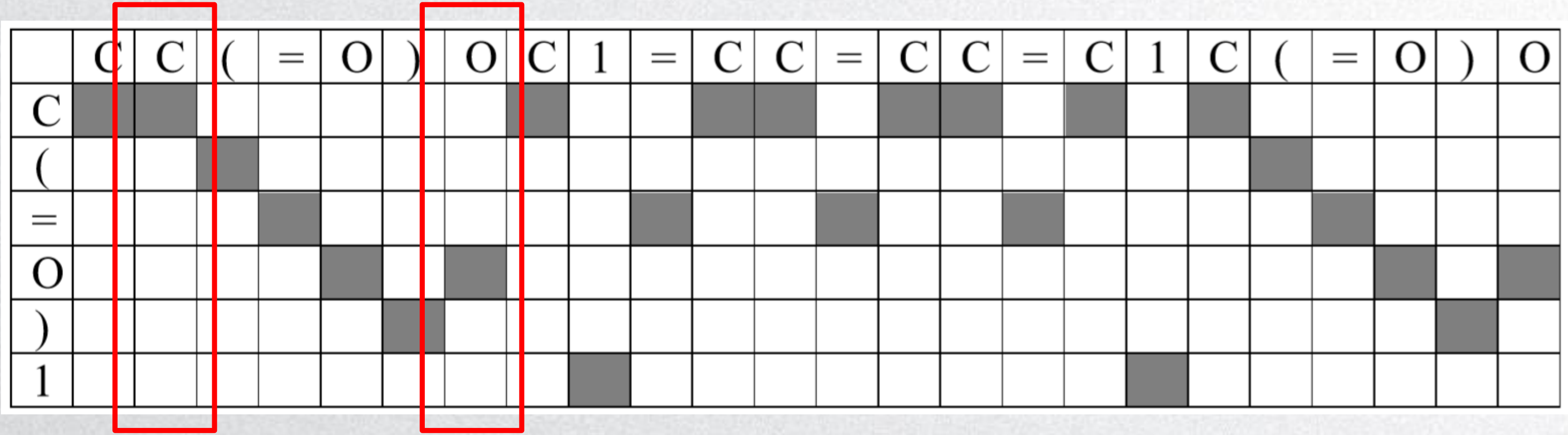
*To get the **Adjacency matrix** of the molecular, you may need to preprocess your data(**SMILES**) with **rdkit** or **pysmiles**.*

Dataset

1. names_labels.txt: Toxic or not

2. names_smiles.txt: ascii string of SMILES of each molecule

3. names_onehots.npy: one hot of each Ascii char



$[1,0,0,0,0,0]^T$ $[0,0,0,1,0,0]^T$

Evaluation

- Remember what we have learned about the confusion matrix

		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

		Prediction	
		1	0
Label	1	True Positive	False Negative
	0	False Positive	True Negative

$$F1 = \frac{2 \times \text{Prec} \times \text{Recall}}{\text{Prec} + \text{Rec}}$$

- **Precision**: the ratio of true class 1 cases in those with prediction 1

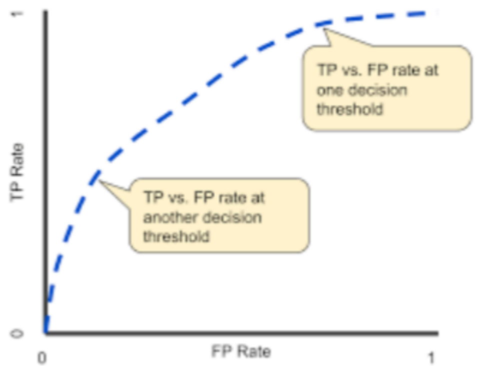
$$\text{Prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**: the ratio of cases with prediction 1 in all true class 1 cases

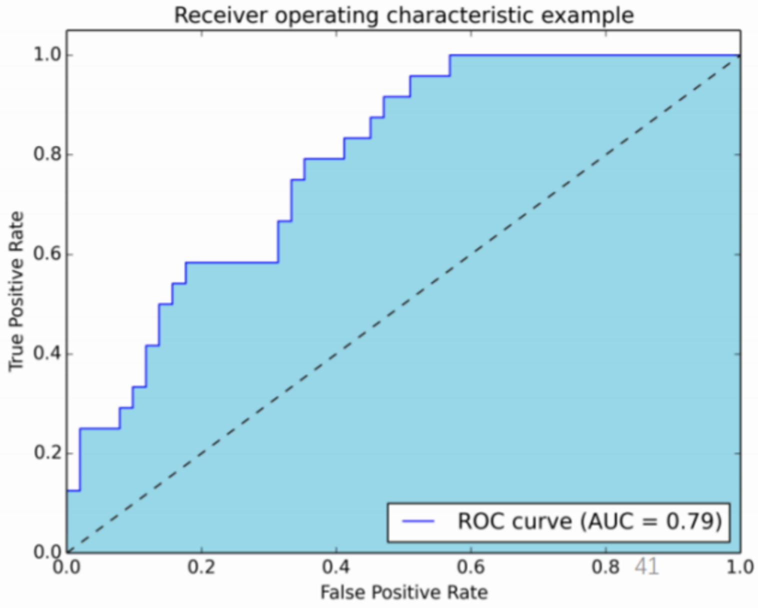
$$\text{Rec} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- These are the basic metrics to measure the classifier

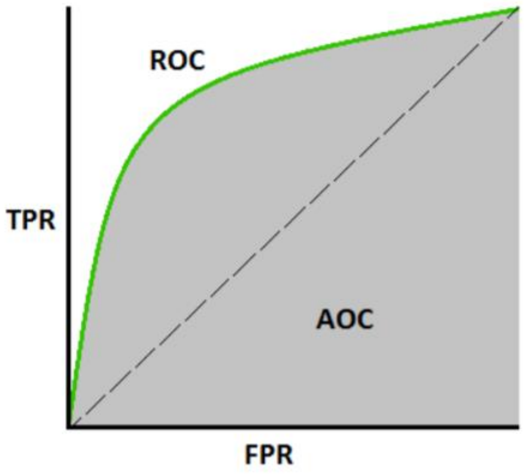
Area Under ROC Curve (AUC)



- A performance measurement for classification problem at various thresholds settings
- Tells how much the model is capable of distinguishing between classes
- The higher, the better
- Receiver Operating Characteristic (ROC) Curve
  - TPR against FPR
  - TPR/Recall/Sensitivity $=\frac{TP}{TP+FN}$
    FPR=1-Specificity= $\frac{FP}{TN+FP}$



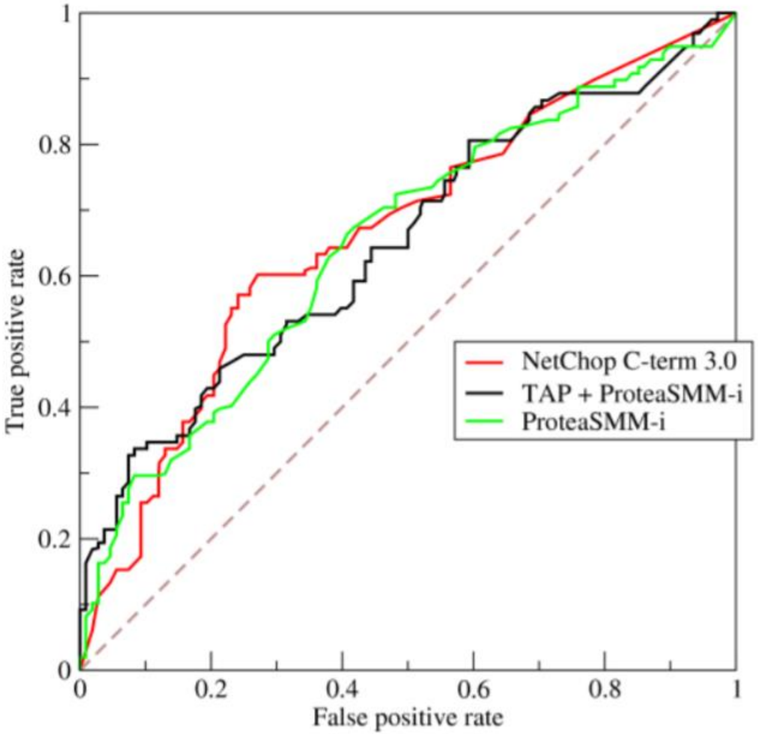
AUC (cont.)



TPR: true positive rate
FPR: false positive rate

- It's the relationship between TPR and FPR when the threshold is changed from 0 to 1
- In the top right corner, threshold is 0, and every thing is predicted to be positive, so both TPR and FPR is 1
- In the bottom left corner, threshold is 1, and every thing is predicted to be negative, so both TPR and FPR is 0
- The size of the area under this curve (AUC) is an important metric to binary classifier
- Perfect classifier get AUC=1 and random classifier get AUC = 0.5

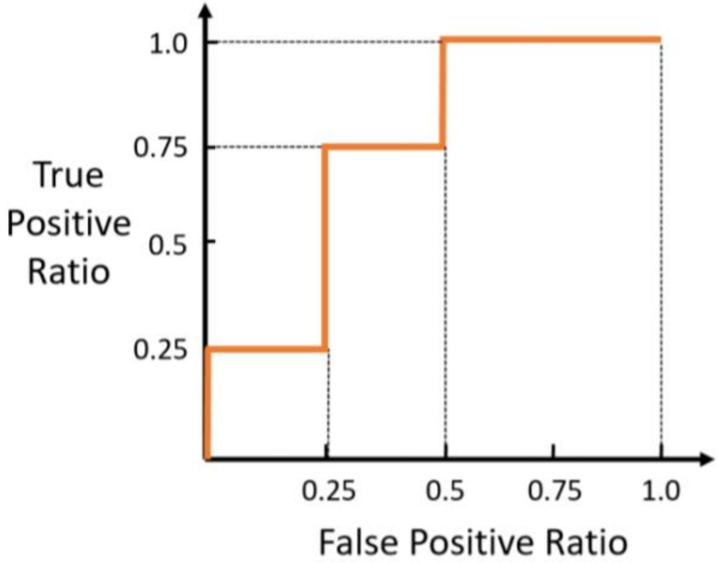
AUC (cont.)



- It considers all possible thresholds.
- Various thresholds result in different true/false positive rates.
- As you decrease the threshold, you get more true positives, but also more false positives.

- From a random classifier you can expect as many true positives as false positives. That's the dashed line on the plot. AUC score for the case is 0.5. A score for a perfect classifier would be 1. Most often you get something in between.

AUC example



AUC = 0.75

Prediction	Label
0.91	1
0.85	0
0.77	1
0.72	1
0.61	0
0.48	1
0.42	0
0.33	0

Assignment Requirements

1. Output: the probability of current drugs are toxic [0, 1] (Softmax and sigmoid might be useful)
2. Model: Any DNN architecture. (CNN, RNN, GCN, etc)

   Anyone who can solve the problem by using graph embedding of the smiles can have the ranking of their **GCN output score** increased by 8
3. You can choose one of the following three versions

   Python 3.6 or Python 3.7

   a. NumPy, Pandas and TensorFlow-gpu 1.15.0.
   b. NumPy, Pandas and TensorFlow-cpu 1.15.0.
   c. NumPy, Pandas and PyTorch 1.1.0.

Submission Requirement

1. To Canvas (Less than 200MB)
    a. train.py
    b. test.py (Less than 60s)
    c. output_ student_id.txt
    d. Weight of your models (Folder)
    e. Report

```
├──518XXXXXXXXX.zip
│    │ ─output_518XXXXXXXXX.txt
│    │ ─test.py
│    │ ─train.py
│    │ ─Report.pdf
│    │ ─other files
│    └──weights (Take TensorFlow as an example)
│          │ ─checkpoint
│          │ ─model.data-00000-of-00001
│          │ ─model.index
│          │ ─model.meta
```

2. To Kaggle

   output_ student_id.txt (follow the format of output_sample.txt)

   The output_ student_id.txt you uploaded to **Canvas**, uploaded to **Kaggle** and the **running output of the submitted model (test.py)** should be the same, or you violate Honor Code

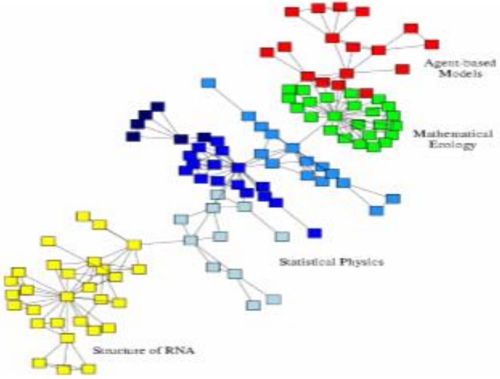
Demo

1. The entire folder demo: https://github.com/Ritchiegit/CS410_2020_fall_project_1
2. Kaggle notebook demo: https://www.kaggle.com/qizhili/project1-cnn-demo

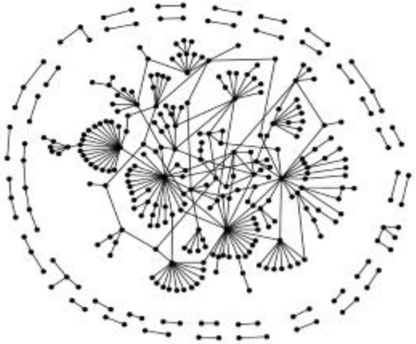
Introduction of GCN

- Compared with Image and text
- Idea
- Formulate the Model
- Example

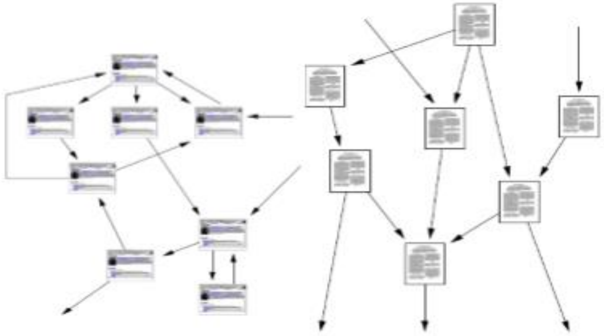
Social networks



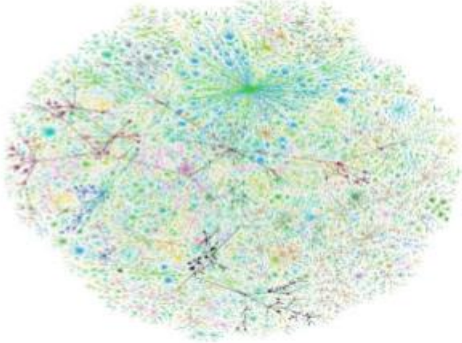
Economic networks



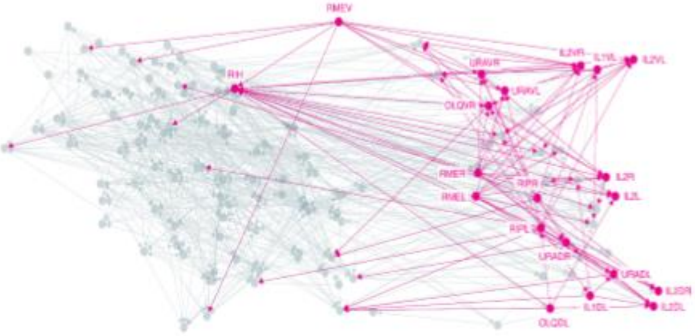
Communication networks



Information networks: Web & citations



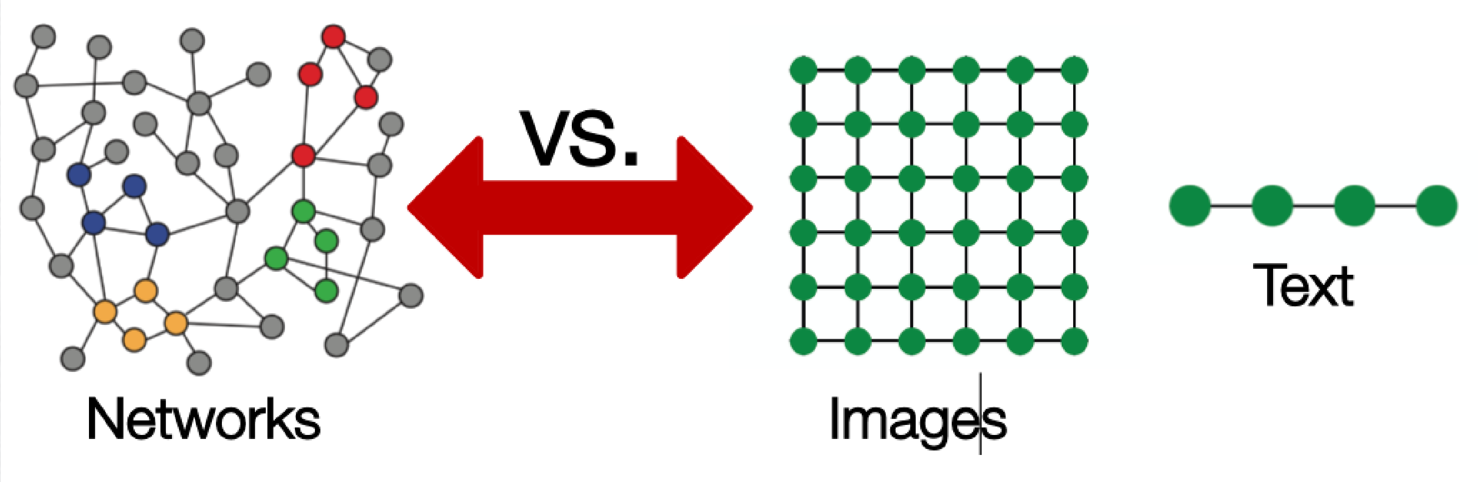
Internet



Networks of neurons

Compare with Image and text

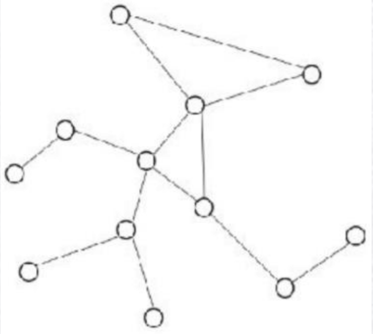
1. Networks are far more complex
    a. No fixed node ordering or reference point
    b. Often dynamic and have multimodal features



http://web.stanford.edu/class/cs224w/slides/08-GNN.pdf

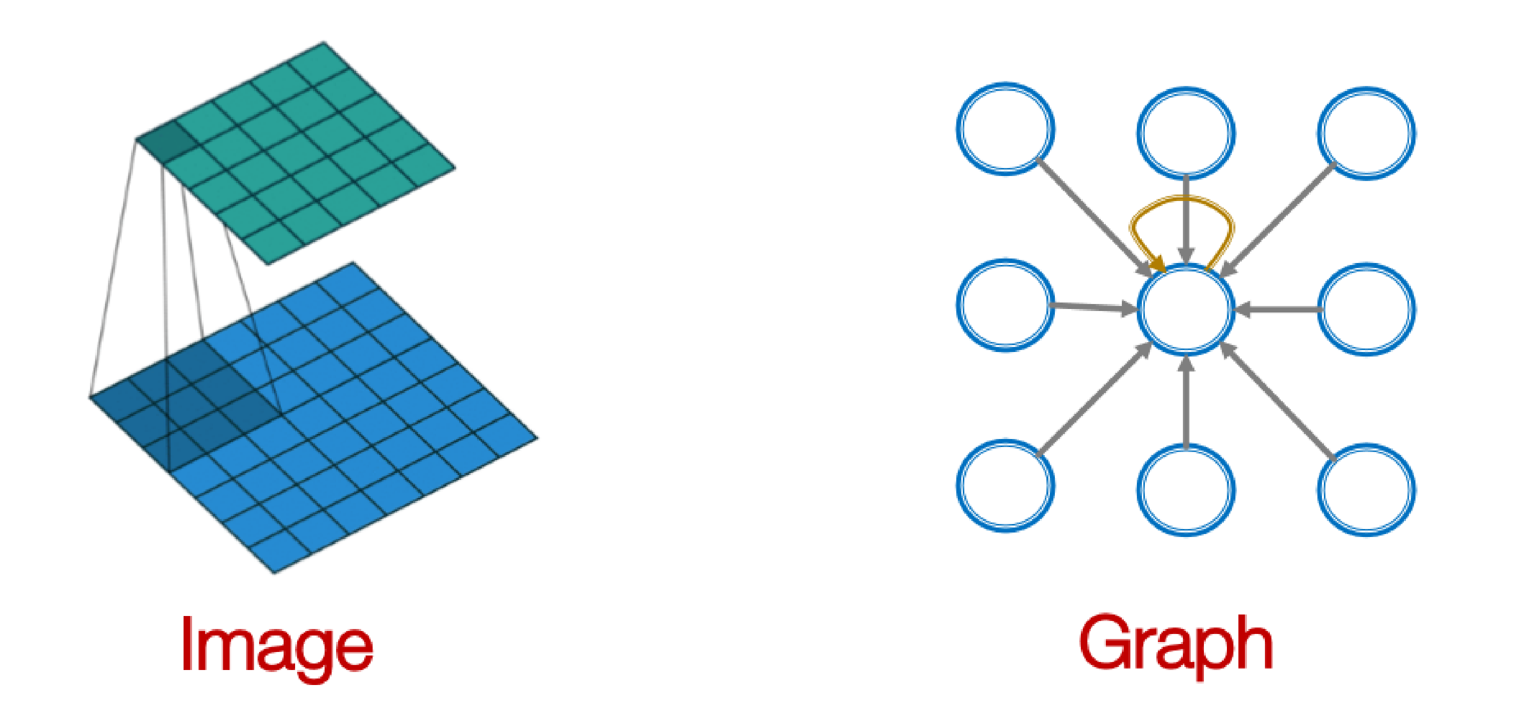
Compare with Image and text

1. Pros of CNNs
    a. Local connections
    b. Shared weights
    c. Use of multiple layers
2. Cons of CNNs
    a. hard to define convolutional and pooling layers for non-Euclidean data



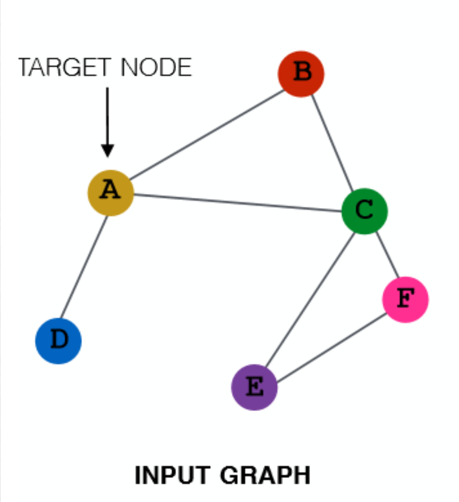
Idea

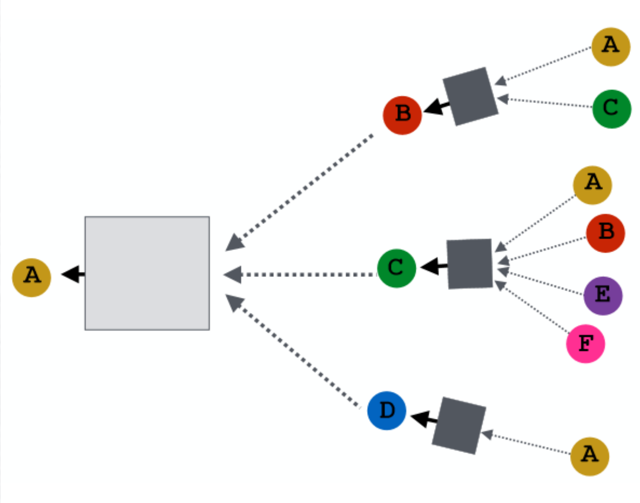
1. Convolution with local neighborhoods



Convolution with local neighborhoods

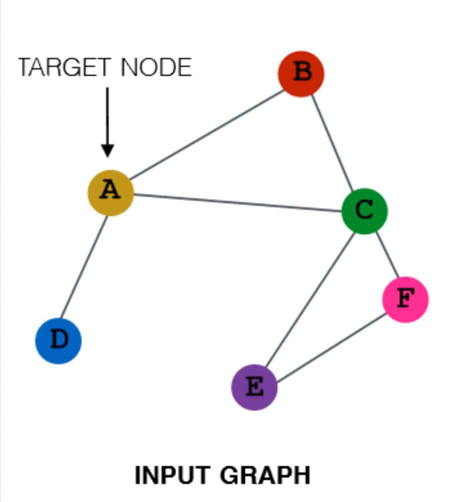
1. Generate node embeddings based on local neighborhoods

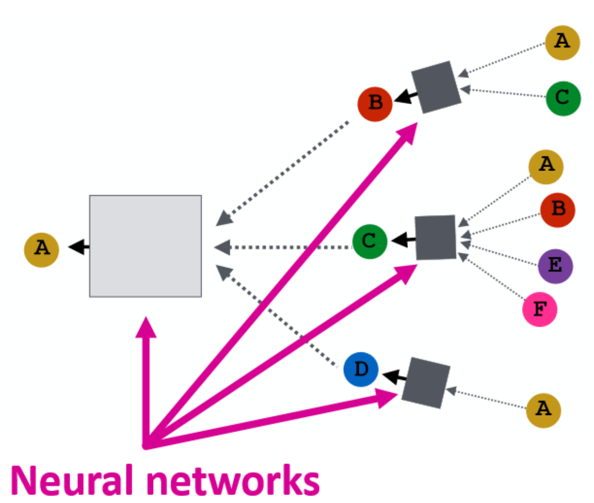




Convolution with local neighborhoods

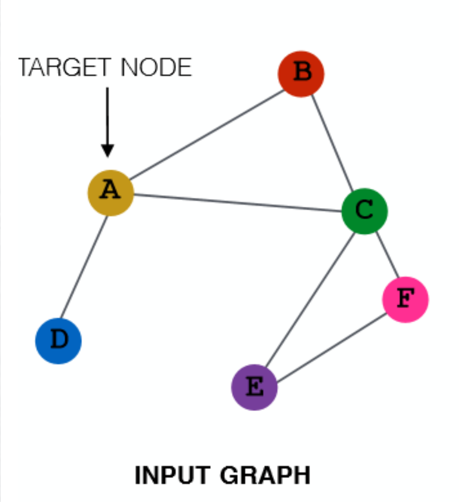
1. Generate node embeddings based on local neighborhoods

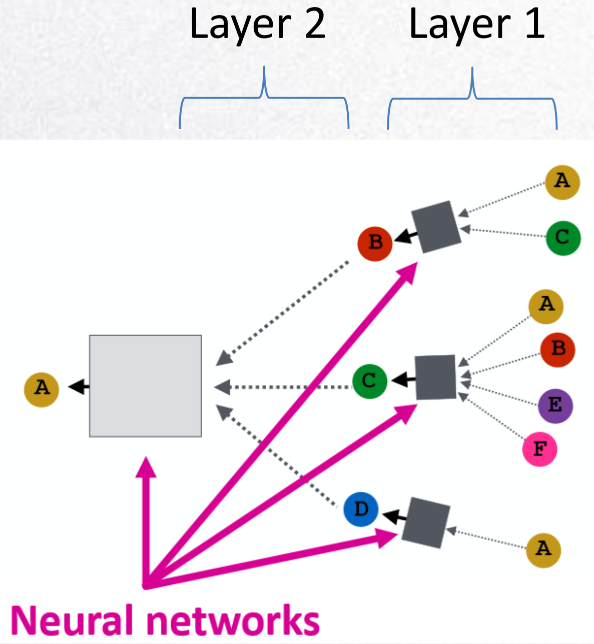




Convolution with local neighborhoods

2. Model can be of arbitrary depth





Formulate the problem

1. V is the vertex set
2. A is Adjacent matrix (with shape $N \times N$)
3. $X \in R^{m \times |V|}$ is a matrix of node features (m is number of feature)
4. Hidden neural network layer

$$H^{(l+1)} = f\left(H^{(l)}, A\right)$$

$$H^{(0)} = X$$

Limitations and solutions

1. Limitations:
   a. Sum up all feature vectors of all neighbor nodes except the node itself
   b. A is not normalized
2. Solution
   a. Enforce self-loop in graph: add identity matrix to A
   b. Normalize A: all rows sum to one
      - i.e. $D^{-1}A$ (D : diagonal node degree matrix)
      - Use symmetrical normalization in practice: $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$
3. Real used propagate rules

$$f(H^{(l)}, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

where $\hat{A} = A + I$ and $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$

Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks[J]. arXiv preprint arXiv:1609.02907, 2016.

Formulate to be trainable

Initial 0-th layer embeddings are equal to node features

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

Previous layer embedding of *v*

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \ldots, K\}$$

$$\mathbf{z}_v = \mathbf{h}_v^K$$

Average of neighbor's previous layer embeddings

Embedding after K layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

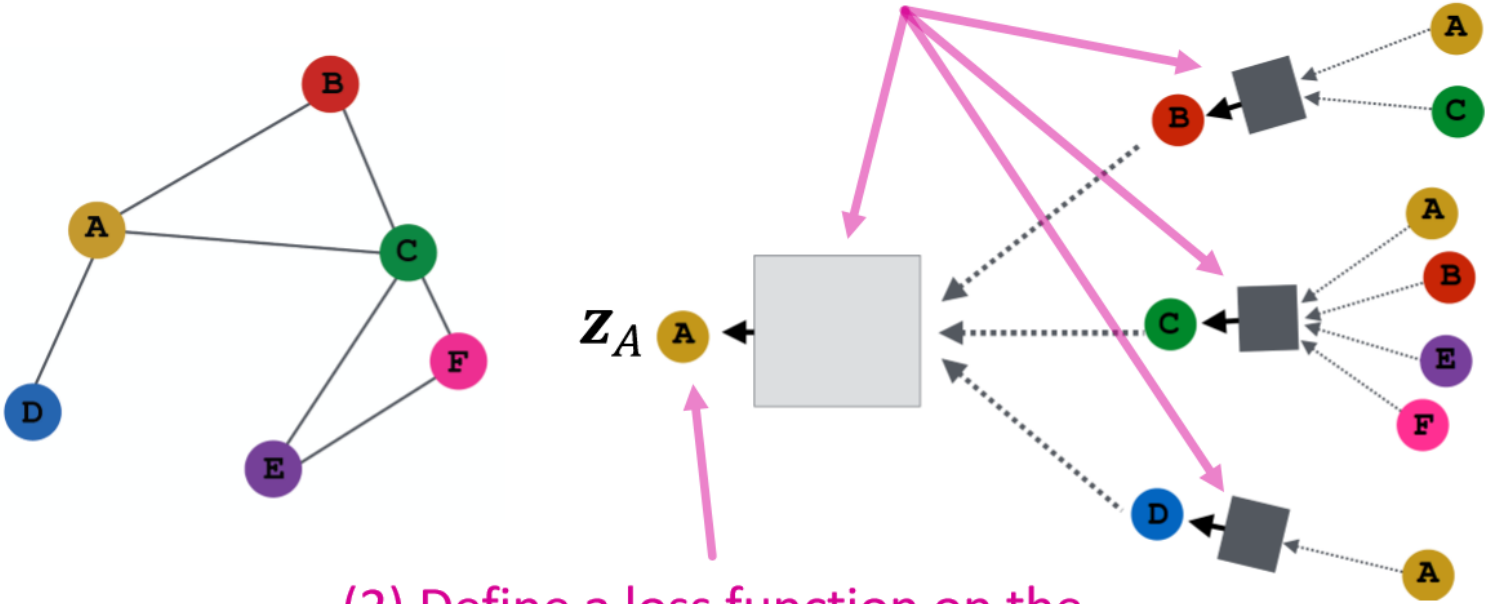
$$\tilde{A} = D^{-1}A + I$$

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)} + \mathbf{W}_0^{(l)}\right)$$

W_1, W_0 is trainable

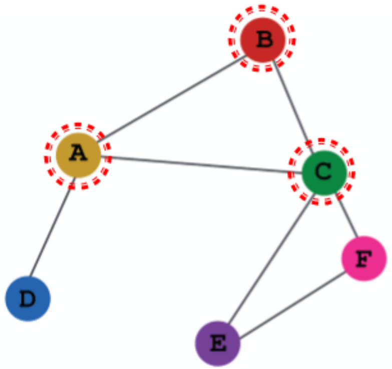
Processing (Define structure)

(1) Define a neighborhood aggregation function



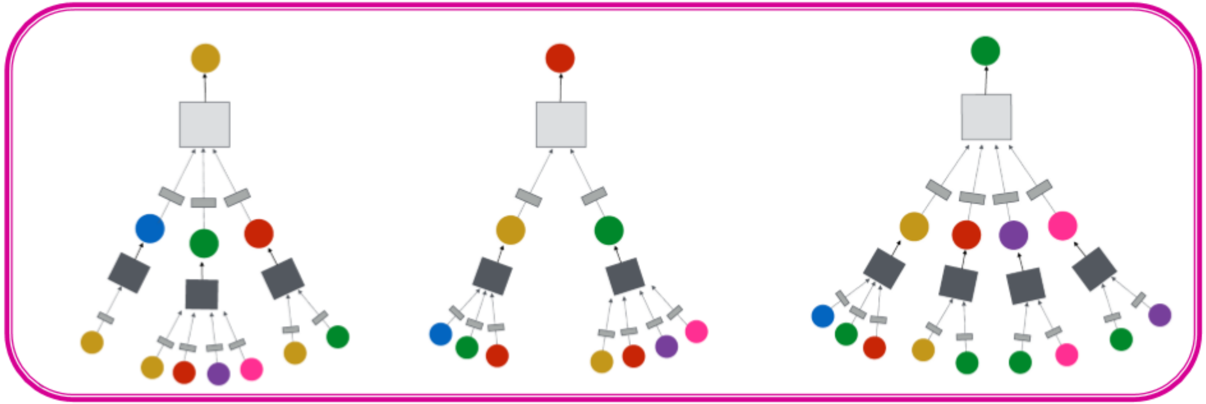
(2) Define a loss function on the embeddings

Processing(Training)

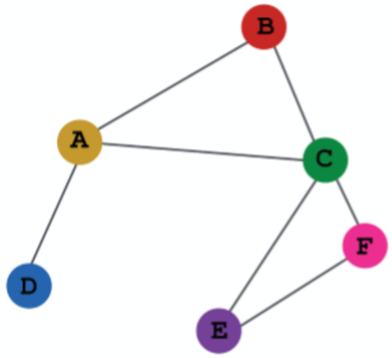


INPUT GRAPH

(3) Train on a set of nodes, i.e., a batch of compute graphs



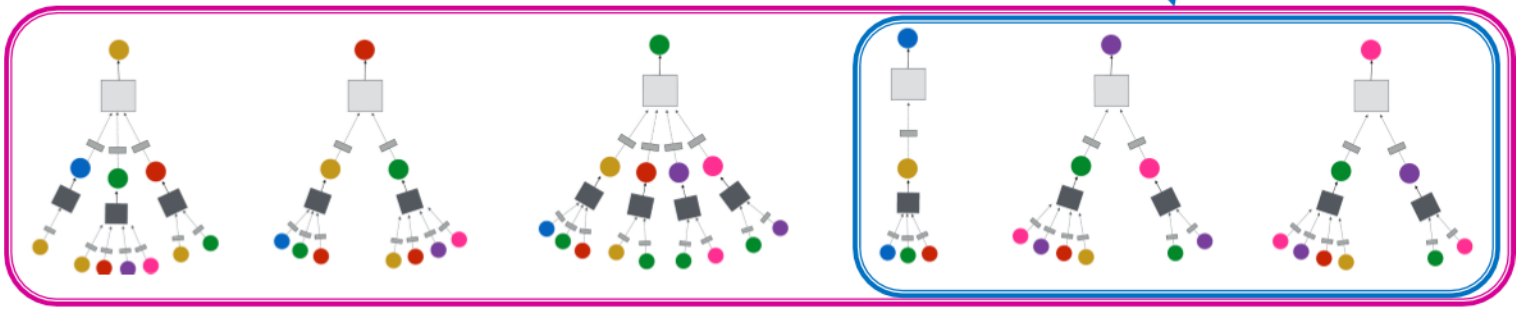
Processing(Predict)



INPUT GRAPH

(4) Generate embeddings for nodes as needed

Even for nodes we never trained on!



Formulate to be trainable

Initial 0-th layer embeddings are equal to node features

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

Previous layer embedding of *v*

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, \dots, K\}$$

$$\mathbf{z}_v = \mathbf{h}_v^K$$

Average of neighbor's previous layer embeddings

Embedding after K layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

$$\tilde{A} = D^{-1}A + I$$

$$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)} + \mathbf{W}_0^{(l)}\right)$$

W_1, W_0 is trainable

Example of GCN

1. Two Convolution layer

```
class GCN(nn.Module):
    """a simple two layer GCN"""
    def __init__(self, nfeat, nhid, nclass):
        super(GCN, self).__init__()
        self.gc1 = GraphConvolution(nfeat, nhid)
        self.gc2 = GraphConvolution(nhid, nclass)

    def forward(self, input, adj):
        h1 = F.relu(self.gc1(input, adj))
        logits = self.gc2(h1, adj)
        return logits
```

https://github.com/tkipf/pygcn

Example of GCN

2. $$\mathbf{H}^{(l+1)} = \sigma\left(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)} + \mathbf{W}_0^{(l)}\right)$$

```python
class GraphConvolution(nn.Module):
    """GCN layer"""
    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.weight = nn.Parameter(torch.Tensor(in_features, out_features))
        if bias:
            self.bias = nn.Parameter(torch.Tensor(out_features))
```

```python
    def forward(self, input, adj):
        support = torch.mm(input, self.weight)  # H * weight
        output = torch.spmm(adj, support)  # adj * H * weight  A * H * W
        if self.bias is not None:
            return output + self.bias
        else:
            return output
```

Example of GCN

1. Build symmetric adjacency matrix and & normalize it & add self loop

```
adj = adj + adj.T.multiply(adj.T > adj) - adj.multiply(adj.T > adj)
adj = normalize(adj + sp.eye(adj.shape[0]))
```

```
def normalize(mx):
    """Row-normalize sparse matrix"""
    rowsum = np.array(mx.sum(1))
    r_inv = np.power(rowsum, -1).flatten()
    r_inv[np.isinf(r_inv)] = 0.
    r_mat_inv = sp.diags(r_inv)
    mx = r_mat_inv.dot(mx)
    return mx
```

$\tilde{A} = D^{-1}A + I$

Resource Of GCN

Summary

1. https://github.com/sungyongs/graph-based-nn

Tutorial

1. http://web.stanford.edu/class/cs224w/slides/08-GNN.pdf
2. http://geometricdeeplearning.com/
3. https://nips.cc/Conferences/2017/Schedule?showEvent=8735

Basic Project

1. https://github.com/tkipf/pygcn
2. https://github.com/tkipf/gcn

Project 1 and GCN

- Task
- SMILES
- AUC
- Requirements
- Compare with Image and Text
- Idea
- Formulate the Model
- Example

THANKS!