SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Kipf & Welling, 2017

Urchade Zaratiana

Semi-supervised Node Classification

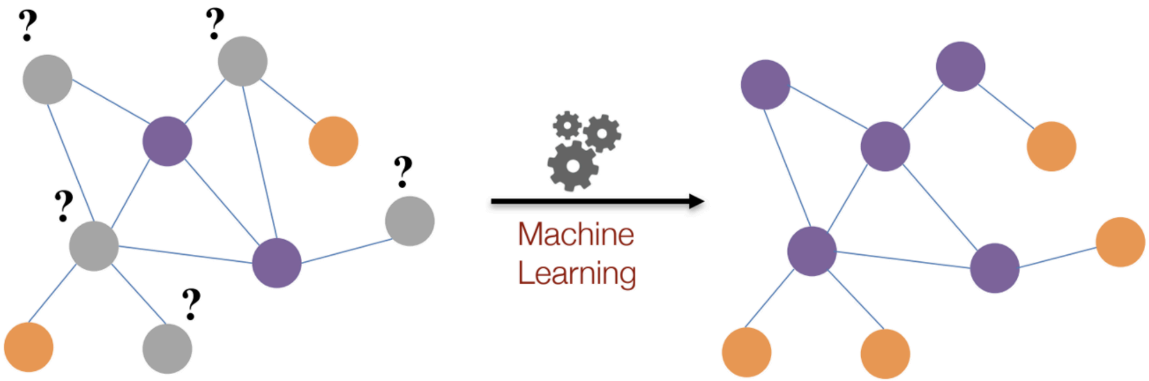


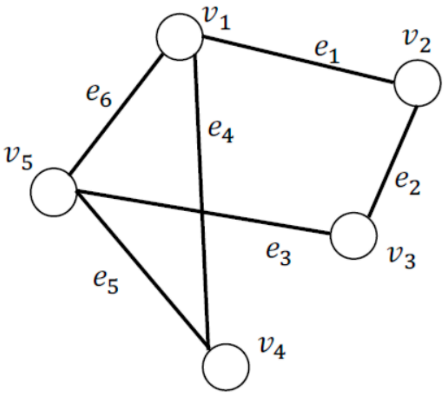
Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Definitions

- A graph $G = (V, E)$ is defined by a set of nodes V and a set of edges E between theses nodes

- A convenient way to represent graphs is through an adjacency matrix A:
  - $(v_i, v_j) \in V^2$
    - $A[v_i, v_j] = 1$ if $(v_i, v_j) \in E$
    - $A[v_i, v_j] = 0$, otherwise



$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Node degree

- In a graph $G = (V, E)$, the **degree** of a node $v_i \in V$ is the *number of nodes that are adjacent* to v_i

$$d(v_i) = \sum_{j=1}^{N} A_{ij}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

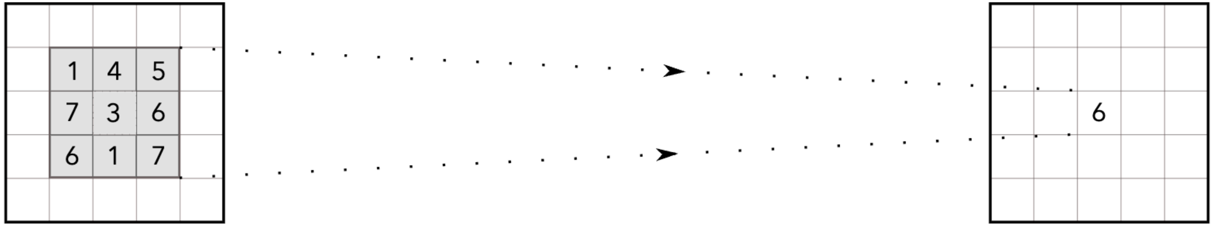
$$d(v_1) = A_{11} + A_{12} + A_{13} + A_{14} + A_{15} = 3$$

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

Degree matrix:

- $D_{ii} = d(v_i)$
- $D_{ij} = 0$ if $i \neq j$ (diagonal matrix)

Extending convolution to Graphs

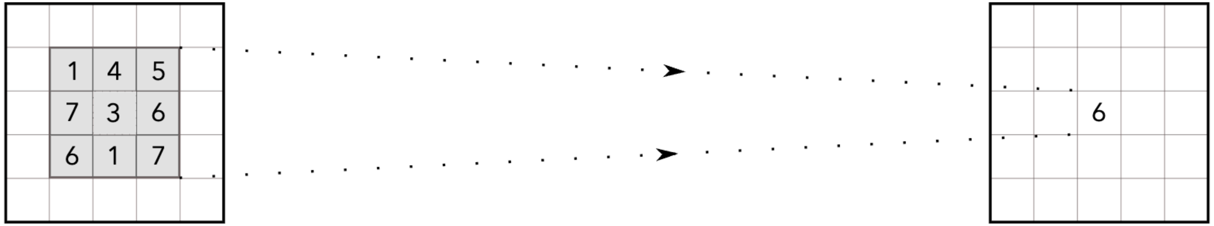


Convolution in CNNs

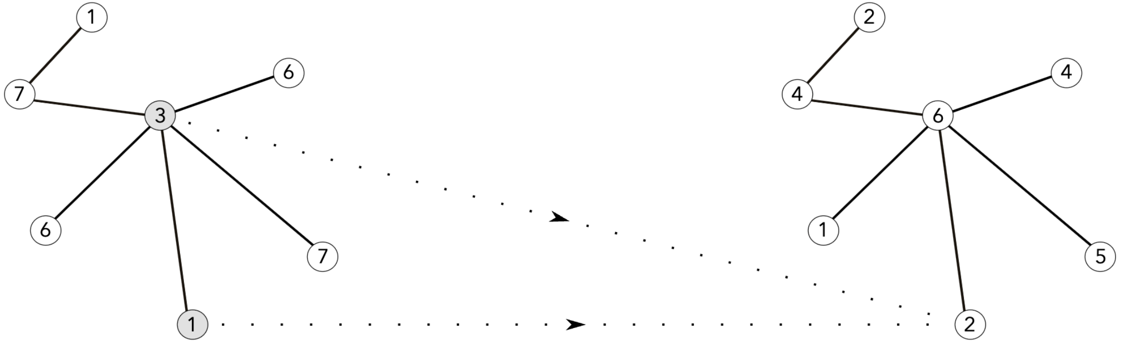
Localized Convolution in GNNs

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

Extending convolution to Graphs



Convolution in CNNs



Localized Convolution in GNNs

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

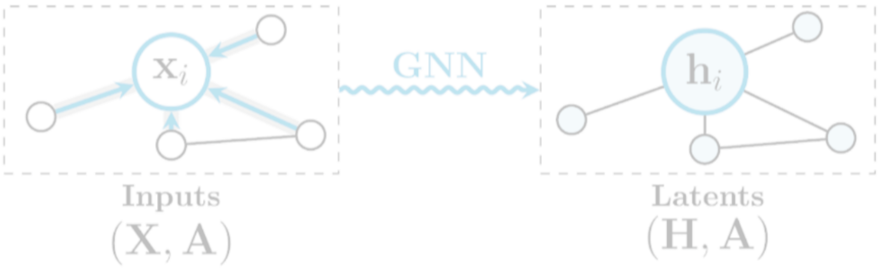
Very simple Graph Neural Network

- Given the input node feature matrix $X \in R^{N \times D_0}$ and adjacency matrix $A \in R^{N \times N}$

- Simple neighborhood aggregation $H^{(l)} = \sigma\left(A H^{(l-1)} W^{(l)}\right) \in R^{N \times D_l}$
  - $X = H^{(0)}$
  - $H^{l} \in R^{N \times D_l}$ the representation of the nodes at l-th layer
  - $A \in R^{N \times N}$ the adjacency matrix
  - $W^{(l)} \in R^{D_{l-1} \times D_l}$ is a weight matrix for the l-th neural network layer
  - $\sigma(\,.\,)$ is a non-linear activation function like the ReLU

- Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself.

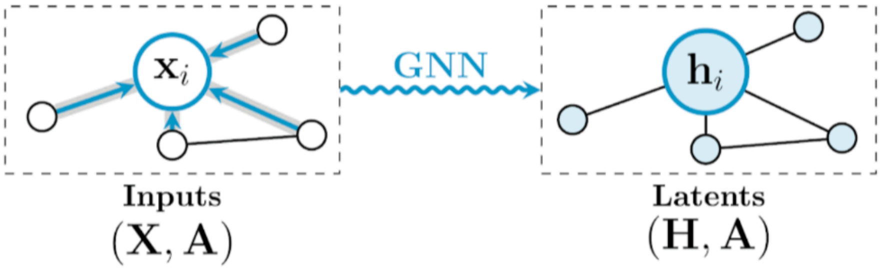
Node-wise update: $h_i^{(l)} = \sigma\left(\sum_{j \in N_i} h_j^{(l-1)} W^{l}\right)$



Very simple Graph Neural Network

- Given the input node feature matrix $X \in R^{N \times D_0}$ and adjacency matrix $A \in R^{N \times N}$
- Simple neighborhood aggregation $H^{(l)} = \sigma\left(A H^{(l-1)} W^{(l)}\right) \in R^{N \times D_l}$
  - $X = H^{(0)}$
  - $H^{l} \in R^{N \times D_l}$ the representation of the nodes at l-th layer
  - $A \in R^{N \times N}$ the adjacency matrix
  - $W^{(l)} \in R^{D_{l-1} \times D_l}$ is a weight matrix for the l-th neural network layer
  - $\sigma(\,.\,)$ is a non-linear activation function like the ReLU
- Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself.

Node-wise update: $h_i^{(l)} = \sigma\left(\sum_{j \in N_i} h_j^{(l-1)} W^{l}\right)$



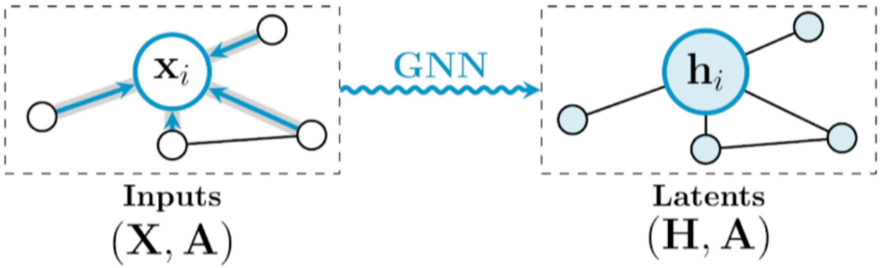
Very simple Graph Neural Network

- Given the input node feature matrix $X \in R^{N \times D_0}$ and adjacency matrix $A \in R^{N \times N}$

- Simple neighborhood aggregation $H^{(l)} = \sigma\left(A H^{(l-1)} W^{(l)}\right) \in R^{N \times D_l}$
  - $X = H^{(0)}$
  - $H^{l} \in R^{N \times D_l}$ the representation of the nodes at l-th layer
  - $A \in R^{N \times N}$ the adjacency matrix
  - $W^{(l)} \in R^{D_{l-1} \times D_l}$ is a weight matrix for the l-th neural network layer
  - $\sigma(\,.\,)$ is a non-linear activation function like the ReLU

- Multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself.

Node-wise update: $h_i^{(l)} = \sigma\left(\sum_{j \in N_i} h_j^{(l-1)} W^{l}\right)$



Better version

- Limitation 1: The update exclude the central node
  - Solution $H^{(l)} = \sigma\left(\tilde{A} H^{(l-1)} W^{(l)}\right)$ where $\tilde{A} = A + I$

    Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} h_j^{(l-1)} W^l)$ $\quad N_i \leftarrow N_i \cup \{i\}$

- Limitation 2: summing can bring instabilities
  - Solution $H^{(l)} = \sigma\left(\tilde{D}^{-1} \tilde{A} H^{(l-1)} W^{(l)}\right)$ where $\tilde{D}$ is the degree matrix of $\tilde{A}$

    Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} \frac{1}{|N_i|} h_j^{(l-1)} W^l)$

A
```
[[0. 1. 0. 1. 1.]
 [1. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 1. 1. 0.]]
```
=> $\tilde{A}$
```
[[1. 1. 0. 1. 1.]
 [1. 1. 1. 0. 0.]
 [0. 1. 1. 0. 1.]
 [1. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1.]]
```
=> $\tilde{D}^{-1}\tilde{A}$
```
[[0.25  0.25  0.    0.25  0.25 ]
 [0.333 0.333 0.333 0.    0.   ]
 [0.    0.333 0.333 0.    0.333]
 [0.333 0.    0.    0.333 0.333]
 [0.25  0.    0.25  0.25  0.25 ]]
```

Better version

- Limitation 1: The update exclude the central node
  - Solution $H^{(l)} = \sigma\left(\tilde{A} H^{(l-1)} W^{(l)}\right)$ where $\tilde{A} = A + I$

    Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} h_j^{(l-1)} W^l)$ $\quad N_i \leftarrow N_i \cup \{i\}$

- Limitation 2: summing can bring instabilities
  - Solution $H^{(l)} = \sigma\left(\tilde{D}^{-1} \tilde{A} H^{(l-1)} W^{(l)}\right)$ where $\tilde{D}$ is the degree matrix of $\tilde{A}$

    Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} \frac{1}{|N_i|} h_j^{(l-1)} W^l)$

A
```
[[0. 1. 0. 1. 1.]
 [1. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1.]
 [1. 0. 1. 1. 0.]]
```
=> $\tilde{A}$
```
[[1. 1. 0. 1. 1.]
 [1. 1. 1. 0. 0.]
 [0. 1. 1. 0. 1.]
 [1. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1.]]
```
=> $\tilde{D}^{-1}\tilde{A}$
```
[[0.25  0.25  0.    0.25  0.25 ]
 [0.333 0.333 0.333 0.    0.   ]
 [0.    0.333 0.333 0.    0.333]
 [0.333 0.    0.    0.333 0.333]
 [0.25  0.    0.25  0.25  0.25 ]]
```

Kipf & Welling GCN

- Symmetric normalization

  Graph convolution update rule $\sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l-1)}W^{(l)}\right)$

  Node-wise update: $h_i^{(l)} = \sigma(\sum_{j\in N_i}\frac{1}{\sqrt{|N_i||N_j|}}h_j^{(l-1)}W^l)$

- GCN is the most popular GNN

$\tilde{A}$

```
[[1. 1. 0. 1. 1.]
 [1. 1. 1. 0. 0.]
 [0. 1. 1. 0. 1.]
 [1. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1.]]
```

=> $\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$

```
[[0.25  0.289 0.    0.289 0.25 ]
 [0.289 0.333 0.333 0.    0.   ]
 [0.    0.333 0.333 0.    0.289]
 [0.289 0.    0.    0.333 0.289]
 [0.25  0.    0.289 0.289 0.25 ]]
```

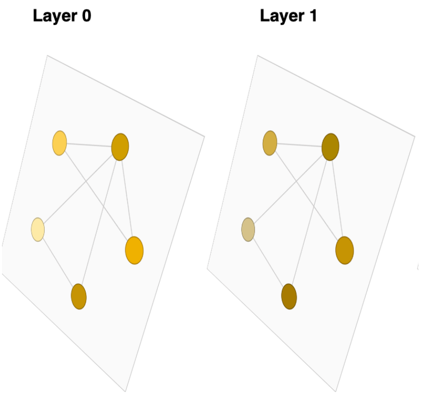
GCN Pytorch code

```python
class GCN(nn.Module):

    def __init__(self, in_feat=1433, hid_feat=64, num_classes=7):

        super().__init__()

        self.W_0 = nn.Linear(in_feat, hid_feat)

        self.W_1 = nn.Linear(hid_feat, num_classes)

    def forward(self, X, A):
        """
        inputs:
            X: Node features of shape [N, in_feat]
            A: Adjacency matrix of shape [N, N]
        returns:
            O: Logits of shape [N, num_classes]
        """

        H = A @ self.W_0(X) # [N, N] [N, hid_feat]

        H = torch.relu(H)

        O = A @ self.W_1(H) # num_nodes, dim

        return O # [N, num_classes]
```

3.1 EXAMPLE

In the following, we consider a two-layer GCN for semi-supervised node classification on a graph with a symmetric adjacency matrix A (binary or weighted). We first calculate $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in a pre-processing step. Our forward model then takes the simple form:

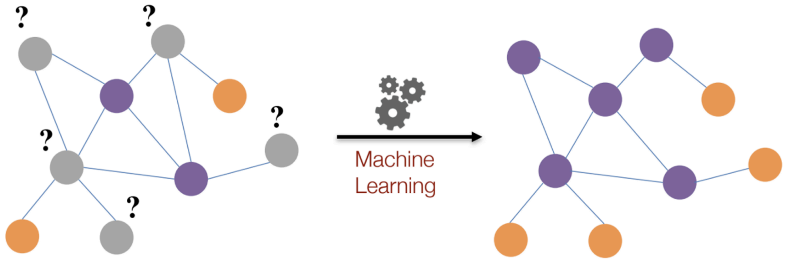
$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) . \tag{9}$$




```python
# A is adjacency matrix (A+I)

# degree matrix
D = A.sum(-1)

# row normalization
A_mean_pooling = torch.diag(1 / D) @ A

# symmetric normalization (Kipf & Welling, 2017)
norm_D = 1 / torch.sqrt(D)
A_GCN = torch.diag(norm_D) @ A @ torch.diag(norm_D) # (kipf & Welling)
```

GCN Pytorch code

```python
def train_model(gcn, data, n_epochs, opt):
    """
    gcn: gcn model
    n_epochs: number of updates
    opt: torch optimizer
    """

    # adjacency matrix [N, N]
    A = data.adj

    # input features [N, in_size]
    X = data.feat

    # labels [N,]
    y = data.y

    # mask for missing labels [N,]
    train_mask = data.train_mask

    for ep in range(n_epochs):

        logits = gcn(X, A) # [N, num_classes]

        # compute loss on available labels
        masked_y = y.masked_fill(train_mask == False, value=-1)
        loss = F.cross_entropy(logits, masked_y, ignore_index=-1)

        # update model
        loss.backward()
        opt.step()
```



Here, $W^{(0)} \in \mathbb{R}^{C\times H}$ is an input-to-hidden weight matrix for a hidden layer with H feature maps. $W^{(1)} \in \mathbb{R}^{H\times F}$ is a hidden-to-output weight matrix. The softmax activation function, defined as $\mathrm{softmax}(x_i) = \frac{1}{\mathcal{Z}}\exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$, is applied row-wise. For semi-supervised multi-class classification, we then evaluate the cross-entropy error over all labeled examples:

$$\mathcal{L} = -\sum_{l\in\mathcal{Y}_L}\sum_{f=1}^{F} Y_{lf}\ln Z_{lf}\,, \tag{10}$$

where $\mathcal{Y}_L$ is the set of node indices that have labels.

The neural network weights $W^{(0)}$ and $W^{(1)}$ are trained using gradient descent. In this work, we perform batch gradient descent using the full dataset for every training iteration, which is a viable option as long as datasets fit in memory. Using a sparse representation for A, memory requirement is $\mathcal{O}(|\mathcal{E}|)$, i.e. linear in the number of edges. Stochasticity in the training process is introduced via dropout (Srivastava et al., 2014). We leave memory-efficient extensions with mini-batch stochastic gradient descent for future work.

Results

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	**Type**	**Nodes**	**Edges**	**Classes**	**Features**	**Label rate**
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	**Citeseer**	**Cora**	**Pubmed**	**NELL**
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)

Colab live demo ? (Semi-supervised Node classification)

https://drive.google.com/file/d/1_SIWhvza31HhjRYY7cu_-_0KlVaiOSw2/view?usp=share_link

Beyond GCNs: Graph attention Network

- The aggregation in GCN is solely based on the graph structure (symmetric normalization)
- GAT uses attention mechanism to dynamically determine the weight of the adjacency matrix.
  - $a_{ij} = f_{att}\left(h_i, h_j\right)$ Attention score
  - $\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k \in N_i} \exp(a_{ik})}$ Normalized attention

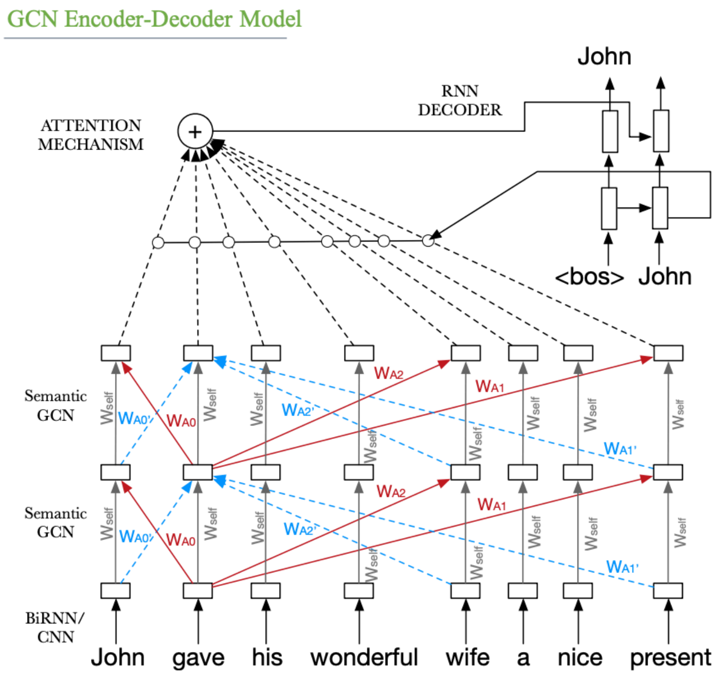
    note that $\alpha_{ij} = 0 \; if \; (i, j) \notin E$

```
[[1. 1. 0. 1. 1.]
 [1. 1. 1. 0. 0.]
 [0. 1. 1. 0. 1.]
 [1. 0. 0. 1. 1.]
 [1. 0. 1. 1. 1.]]
```

  - f_{att} is a neural network (Dot-product, MLP, Multi-head Attention, Transformers, …)

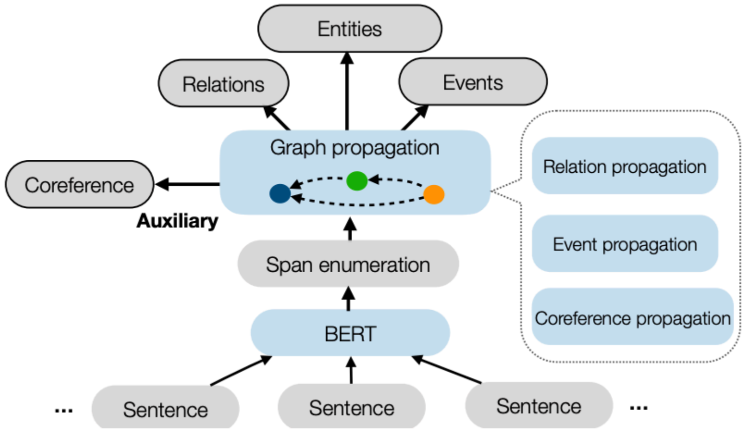
    Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} \alpha_{ij} h_j^{(l-1)} W^l)$

Some NLP application of GCNs



Semantic/synatctic GCNs for Machine Translation
(Marcheggiani, Bastings, Titov, 2018)

GNN for information joint extraction



DyGIE++, Wadden et at., 2019

Thank you !!