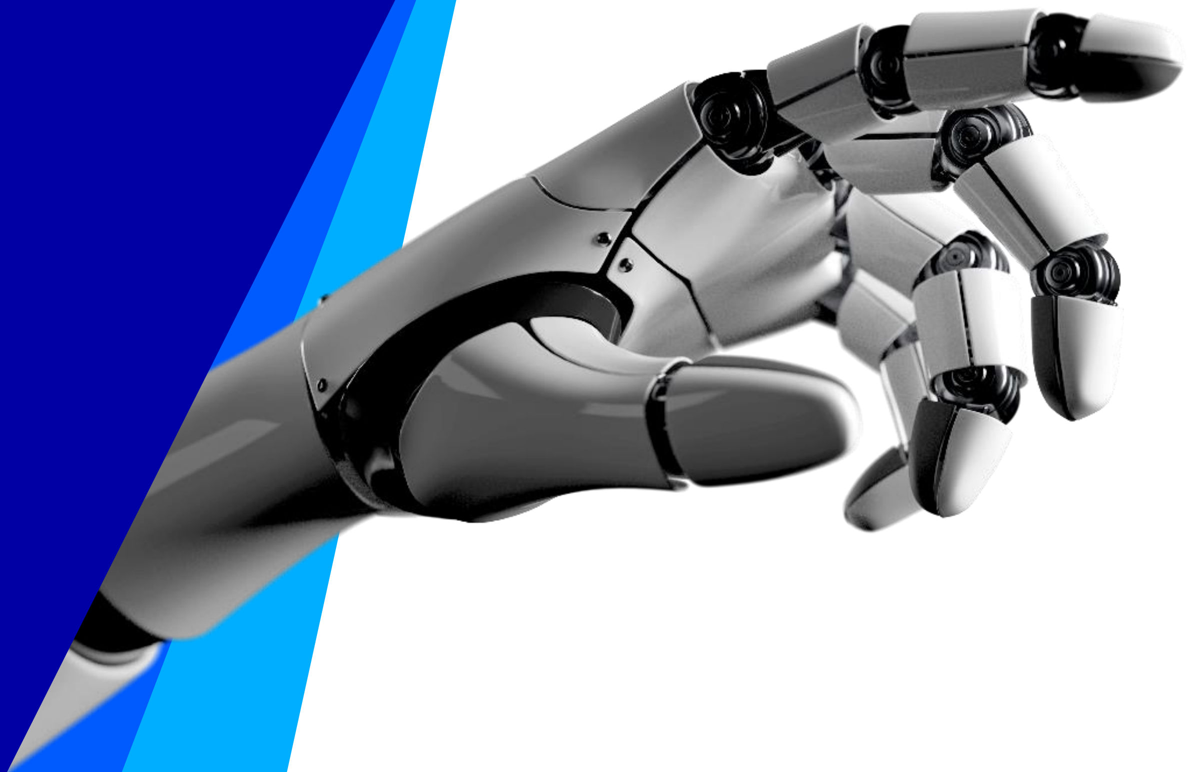


Session 14

PERFORMANCE
EN FINANCEMENT
DE L'INNOVATION

Data augmentation for NLP

- Findings from Chen et al., 2021
  - Token-level augmentations work well for supervised learning
    - Synonym replacement, BERT word replacement, random insertion, random deletion …
  - Sentence-level augmentation usually works the best for semi-supervised learning
    - Backtranslation, label-conditioned generation
  - Augmentation methods can sometimes hurt performance, even in the semi-supervised setting.

Chen, J., Tam, D., Raffel, C., Bansal, M., & Yang, D. (2021). An Empirical Survey of Data Augmentation for Limited Data Learning in NLP. ArXiv, abs/2106.07499.

Intro to Graph Neural Networks

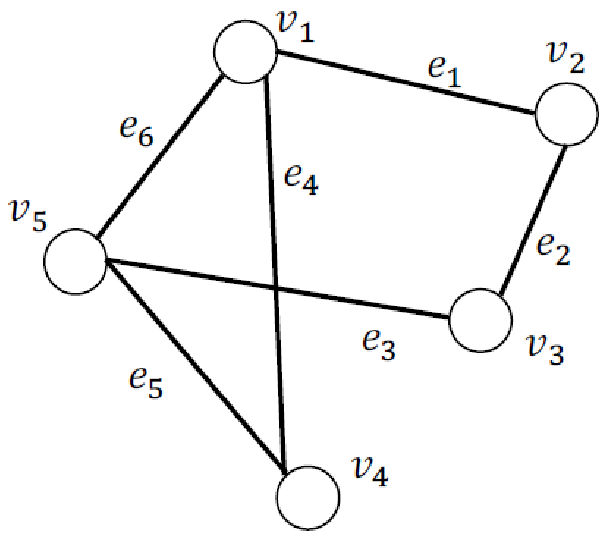
France · Espagne · Royaume-Uni · Portugal · Belgique · Italie · Canada · Brésil · Chili · Colombie

Graphs are everywhere

- Social Networks (Facebook, Twitter, …)
  - GNN applications: Friend recommendation, community detection, bot classification
- Citation networks
  - GNN applications: paper recommendation, topic classification
- Molecules
  - GNN applications: Toxicity prediction, De Novo molecule generation
- Knowledge graphs
  - GNN applications: Link prediction (inferring new facts)
- Netflix, Spotify, Amazon …
  - GNN applications: content recommendation

Definition

- A graph $G = (V, E)$ is defined by a set of nodes V and a set of edges E between these nodes
- A convenient way to represent graphs is through an adjacency matrix A
- $A[v_i, v_j] = 1$ if $(v_i, v_j) \in E$ and $A[v_i, v_j]=0$, 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.

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

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

$$d(v_1) = A_{11} + A_{12} + A_{13} + A_{14} = 0 + 1 + 0 + 1 + 1 = 3$$

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

Th degree matrix of G

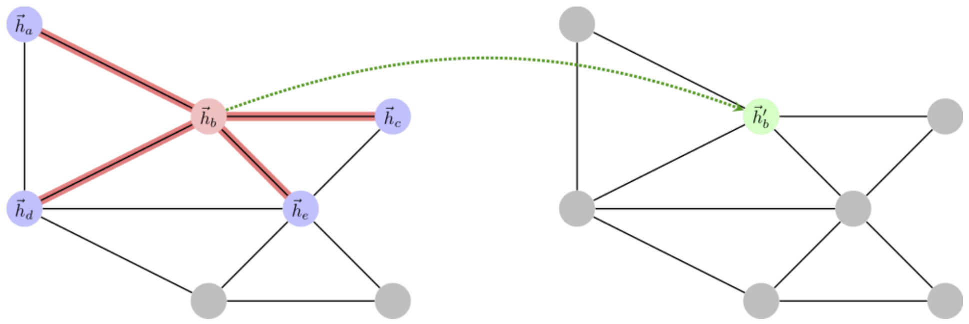
- $D_{ii} = d(v_i)$
- $D_{ij} = 0 \; if \; i \neq j$

Simple GNN

- Give the input node feature matrix
- $X \in R^{N \times D_0}$
- Simple neighborhood aggregation $H^{(l)} = \sigma\left(AH^{(l-1)}W^{(l)}\right)$
  - $X = H^{(0)}$
  - $H^{l} \in R^{N \times D_l}$ 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: $h_i^{(l)} = \sigma(\sum_{j \in N_i} h_j^{(l-1)} W^{l})$



Some limitations

- 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: $h_i^{(l)} = \sigma(\sum_{j \in N_i} h_j^{(l-1)} W^{l})$
- Limitation 2: summing can bring instabilities
  - Solution $H^{(l)} = \sigma\left(\widetilde{D}^{-1}\tilde{A}H^{(l-1)}W^{(l)}\right)$ where $\widetilde{D}$ 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.]]
```
=>
```
[[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 ]]
```
$\widetilde{D}^{-1}\tilde{A}$

Graph convolution network

- Symmetric normalization
  - Graph convolution update rule $\sigma(\widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}H^{(l-1)}W^{(l)})$
  - 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.]]
```

=>

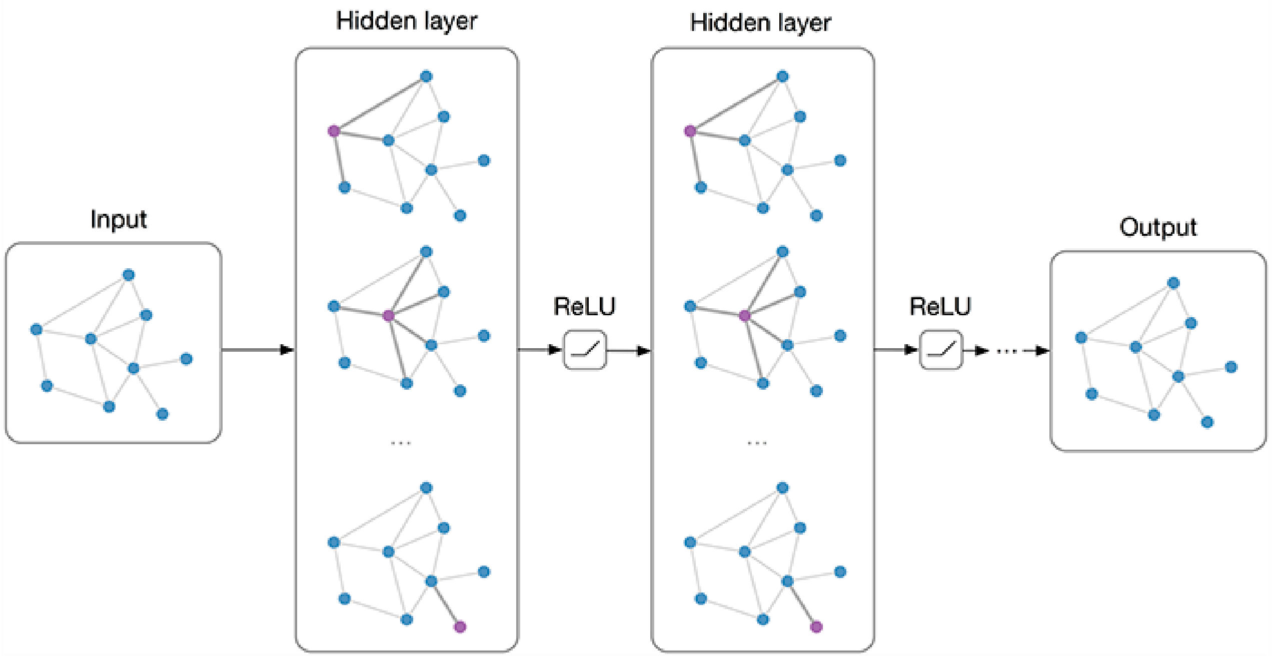
```
[[0.25  0.289 0.    0.289 0.25 ]
 [0.289 0.333 0.333 0.    0.   ]
 [0.    0.333 0.333 0.    0.289]
 [0.289 0.    0.    0.333 0.289]
 [0.25  0.    0.289 0.289 0.25 ]]
```

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

Kipf, Thomas N., and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." ICLR (Poster), 2016.

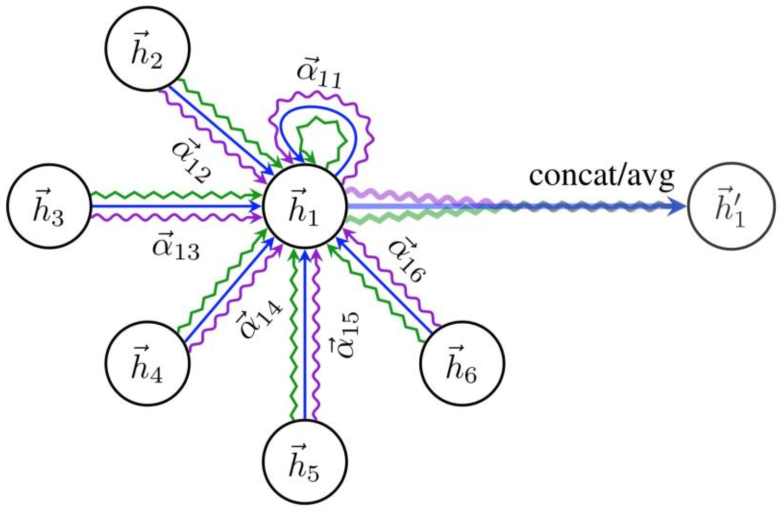
Pytorch session 1

- Semi-supervised learning with Graph Neural Network
  - Sum pooling
  - Mean pooling
  - GCN



Graph Attention Network

- The aggregation in GCN is solely based on the graph structure while (symmetric normalization)
- GAT uses self-attention to determine the weight of neighbors.
  - $a_{ij} = f_{att}(h_i, h_j)$ Attention score
  - $\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k \in N_i} \exp(a_{ik})}$ Normalized attention
  - f_{att} is a neural network (MLP, Multi-head Attention, Transformers)
  - Node-wise update: $h_i^{(l)} = \sigma(\sum_{j \in N_i} \alpha_{ij} h_j^{(l-1)} W^l)$
    - note that $\alpha_{ij} = 0 \; if \; (i, j) \notin E$



Veličković, Petar, et al. "Graph Attention Networks." International Conference on Learning Representations, 2018.

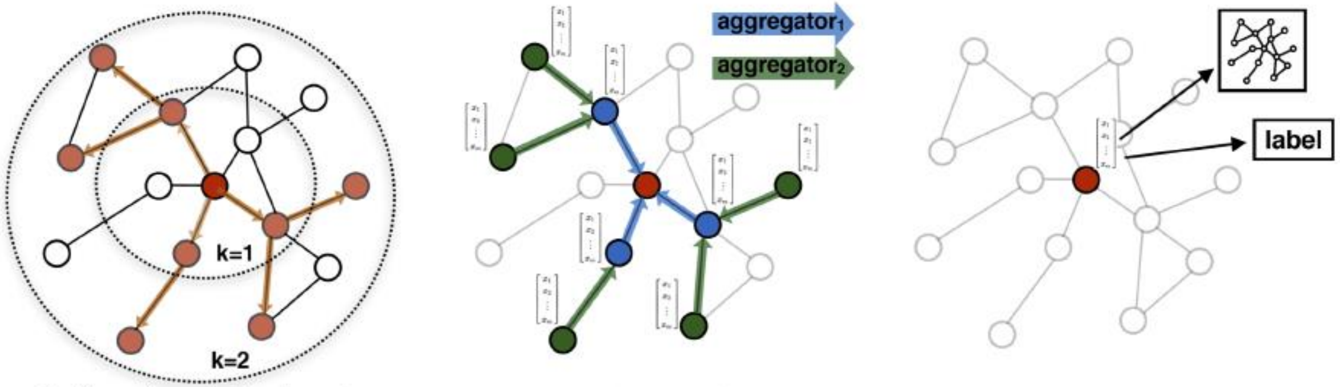
Pytorch session 2

- Semi-supervised learning with Graph Attention Network

GraphSAGE

- Stochastic generalization of Graph Convolution Network
- Useful for massive graphs (100000+ nodes)
- Works well on unseen nodes (inductive tasks)

- For each node in the graph
  - Sample neighborhood
  - Aggregate information
  - Predict label



1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Hamilton, William L., et al. "Inductive Representation Learning on Large Graphs." Advances in Neural Information Processing Systems, vol. 30, 2017

GraphSAGE

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

```
h_v^0 ← x_v, ∀v ∈ V ;   | Initial node features
for k = 1...K do
    for v ∈ V do
        h_N(v)^k ← AGGREGATE_k({h_u^(k-1), ∀u ∈ N(v)});   | Aggregate neighbors· sum, average, max pooling or LSTM
        h_v^k ← σ(W^k · CONCAT(h_v^(k-1), h_N(v)^k))   | Update node representation
    end
    h_v^k ← h_v^k / ||h_v^k||_2, ∀v ∈ V
end
z_v ← h_v^K, ∀v ∈ V   | Final node representation
```

$$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V};$$
$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$$
$$\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$$
$$\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$$
$$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$$

Next session

- Node embedding techniques. Node2Vec, DeepWalk
- Self-supervised representation learning on graphs.
  - Data augmentation for graphs
  - Deep Graph Infomax (DGI)
  - Boostrapped GRL
  - GNN pre-training
  - ...