Graph Neural Networks Demystified

An overview of the essential stuffs in Stanford CS224W (Lectures 1~9)
with oversimplified examples

Ng Yen Kaow

Embeddings

- Relatively small vectors associated with each object where similar objects have similar embeddings
- Using the embeddings of graph elements, various tasks can be performed
  - Cluster nodes in a graph
  - Predict properties of a node
  - Predict if two nodes may be connected
  - Classify entire graphs
- To perform each task, use the embedding with a suitable ML method
  - e.g. clustering can be performed with k-means

Obtaining embeddings

- Embeddings can be formed with or learned from features
  - Node-level features
    - Degree
    - Centrality (eigenvector/ betweenness/ closeness)
    - Clustering coefficient
    - Graphlets
    - Structure-based features
  - Link-level features
    - Distance-based features
    - Local/global neighborhood overlap
  - Graph-level features
    - Graph kernels
- Task-independent embeddings can be learned from unsupervised learning

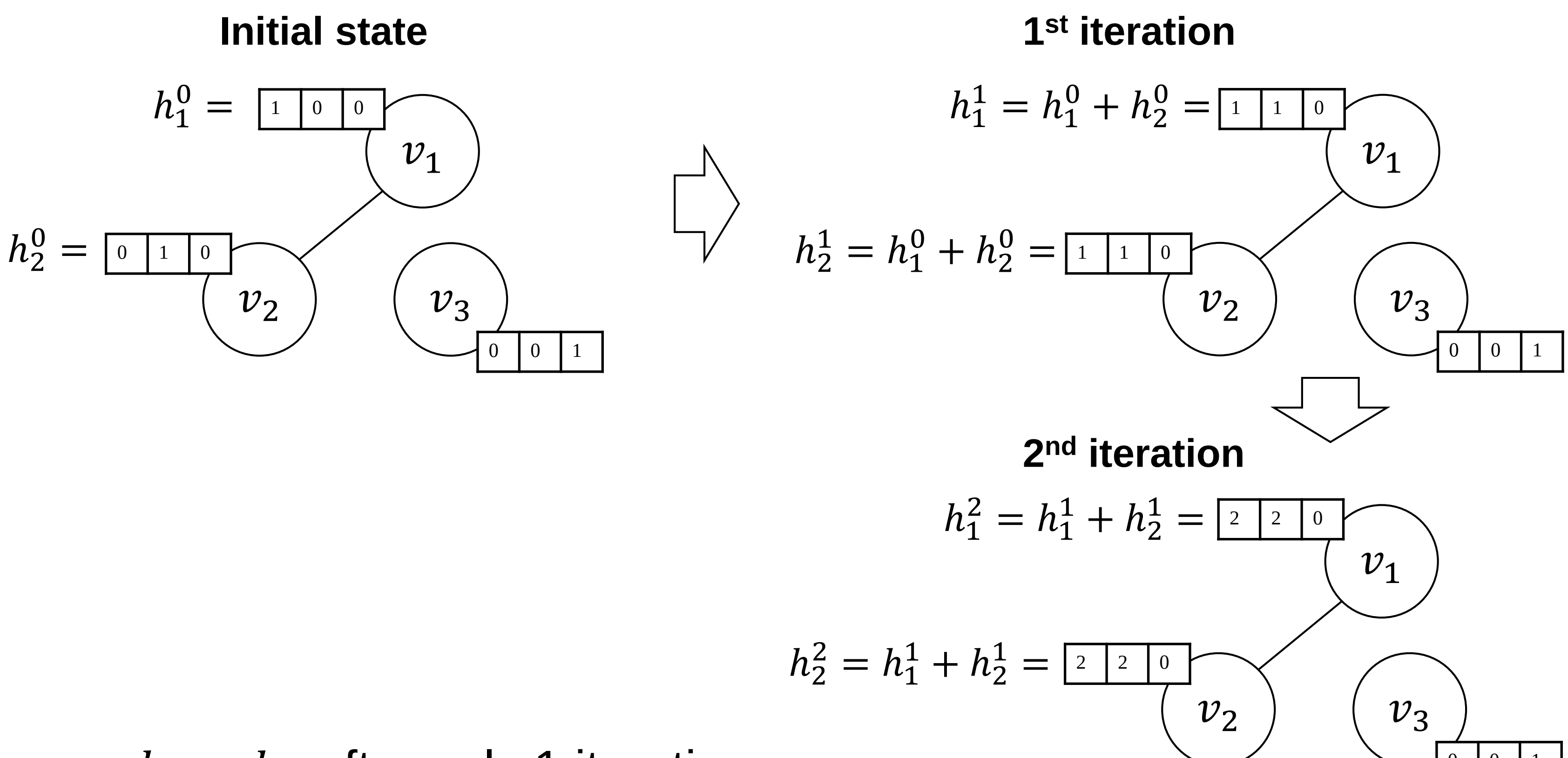
Task-independent embeddings

- Unsupervised extraction by random walks
  - **DeepWalk**
    - Estimate pairwise distance between nodes (hence their co-occurrence probability)
      - Usable for finding product relatedness in recommender
    - Node embeddings
      1. Estimate node distances with random walks
      2. Train a neural network (with node input and embedding output) such that distances between embeddings agree with estimated distances
  - **Anonymous Walk**
    - Embeddings for entire graphs
  - **Simpler method: just add up neighbors**

Embeddings by adding neighbors

- Sum up the features of (self and) neighbor nodes
  - Features of nodes in close proximity will become similar

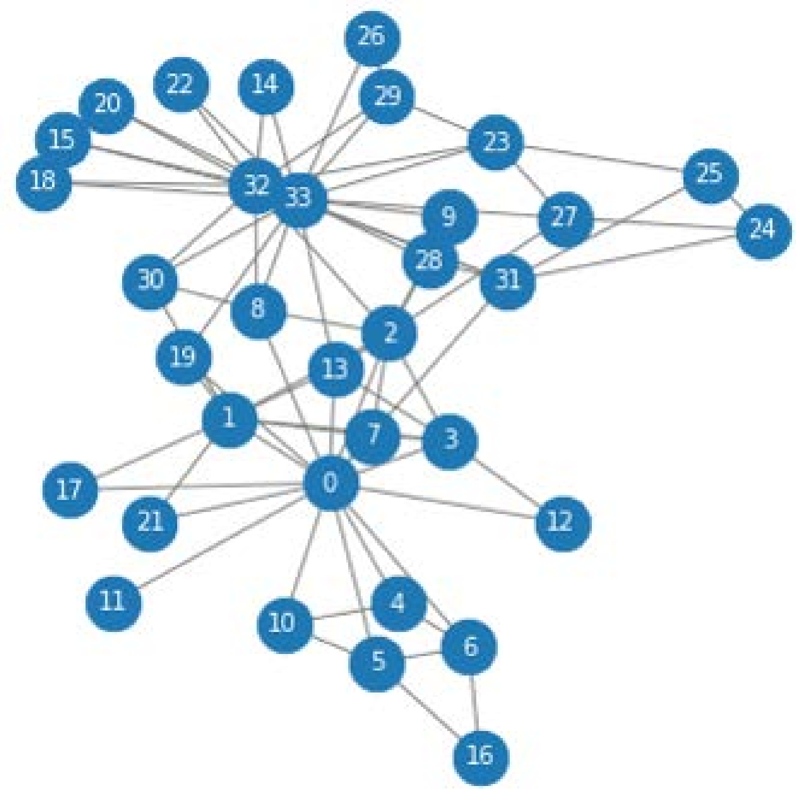
Example: Let h_i^j denote features of node i at iteration j and let $h_1^0 = (1 \quad 0 \quad 0)$, $h_2^0 = (0 \quad 1 \quad 0)$, and $h_3^0 = (0 \quad 0 \quad 1)$



  - $h_1 \equiv h_2$ after only 1 iteration

Embeddings by adding neighbors

- To cluster nodes in a graph, will it work if we
  1. Start with a unique feature for each node, and
  2. Repeatedly add up neighboring features, and
  3. Finally, cluster the resultant features with some method like *k*-means?



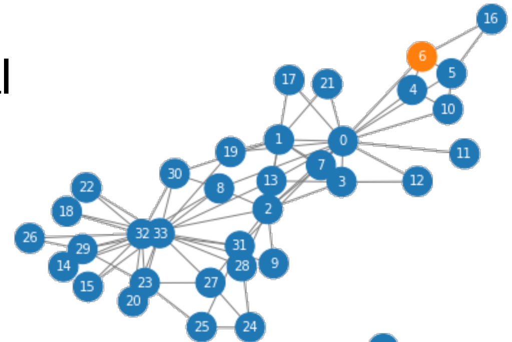
Let's try with karate club network

Based on https://tkipf.github.io/graph-convolutional-networks/

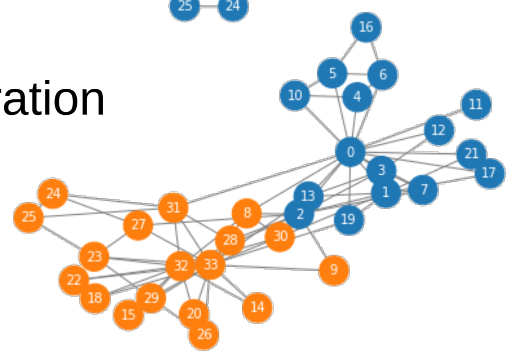
Embeddings by adding neighbors

- Karate club

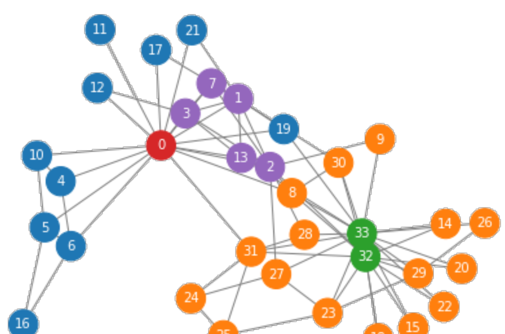
Initial



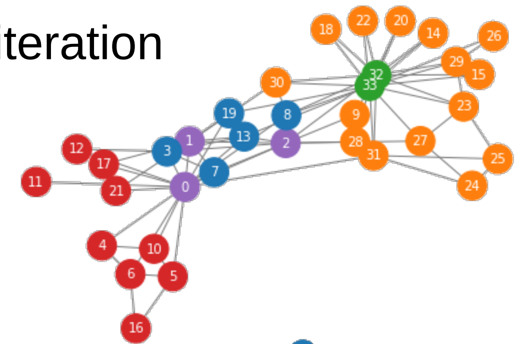
1^{st} iteration



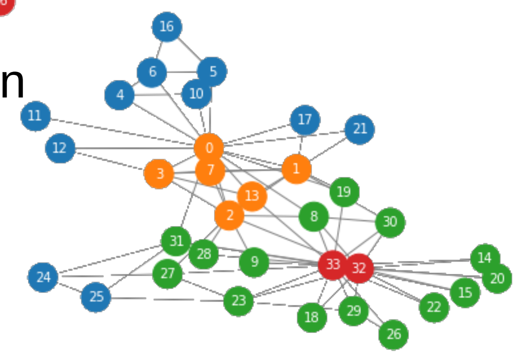
2^{nd} iteration



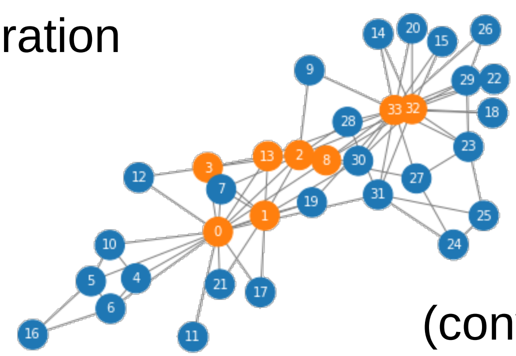
3^{rd} iteration



4^{th} iteration



5^{th} iteration



(converged)

(number of clusters determined with silhouette method)

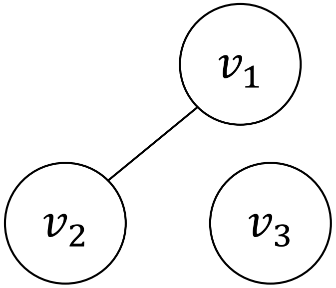
Adding neighbors w/ linear algebra

- Let matrix H be a matrix where each row is a node and each column is a feature
  - H have dim $|V| \times d$
- Let A be an adjacency matrix
  - Let $\hat{A} = A + I$ where I is the identify matrix
- Then, **sum** is simply $\hat{A}H$

$$\begin{pmatrix} a & b & c \\ & \dots & \\ & \dots & \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} ah_1 + bh_2 + ch_3 \\ \dots \\ \dots \end{pmatrix}$$

Note that node order is not important

e.g.



$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} h_1 + h_2 \\ h_1 + h_2 \\ h_3 \end{pmatrix}$$

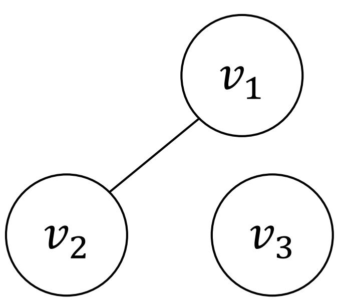
Adding neighbors w/ linear algebra

- Let matrix H be a matrix where each row is a node and each column is a feature
  - H have dim $|V| \times d$
- Let A be an adjacency matrix
  - Let $\hat{A} = A + I$ where I is the identify matrix
- Further **normalize** each row of $\hat{A}$ to sum to 1

$$\begin{pmatrix} 1/3 & 1/3 & 1/3 \\ & \ldots & \\ & \ldots & \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} (h_1 + h_2 + h_3)/3 \\ \ldots \\ \ldots \end{pmatrix}$$

Note that **normalize** does the same thing as **mean**

e.g.



$$\begin{pmatrix} .5 & .5 & 0 \\ .5 & .5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} (h_1 + h_2)/2 \\ (h_1 + h_2)/2 \\ h_3 \end{pmatrix}$$

Adding neighbors w/ linear algebra

- Let matrix H be a matrix where each row is a node and each column is a feature
  - H have dim $|V| \times d$
- Let A be an adjacency matrix
  - Let $\hat{A} = A + I$ where I is the identify matrix
- Further **normalize** each row of $\hat{A}$ to sum to 1
  - To perform this normalization, it suffices that we let $\hat{A} \leftarrow D^{-1}\hat{A}$ where D is the diagonal node degree matrix
    - In PyTorch, use `torch.nn.functional.normalize(A, p=1, dim=1)`
  - Or, use $\hat{A} \leftarrow D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}$, the spectral variant
    - In PyTorch, use

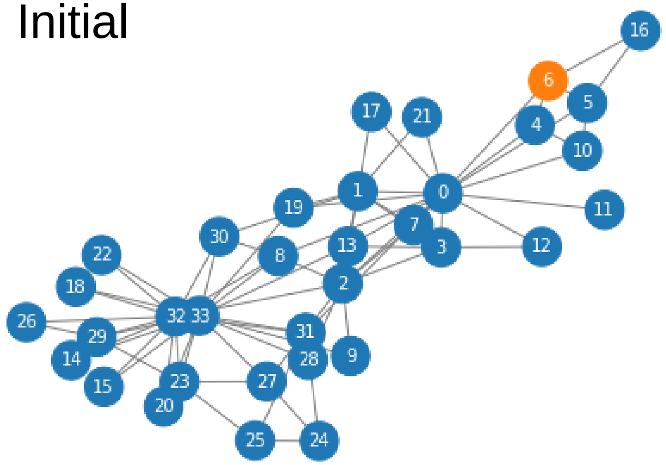
```
D = torch.diag(torch.sum(A, 1)).inverse().sqrt()
D = torch.mm(torch.mm(D, A), D)
```

Normalized $\hat{A}$ is in general **not symmetric**

Adding neighbors w/ linear algebra

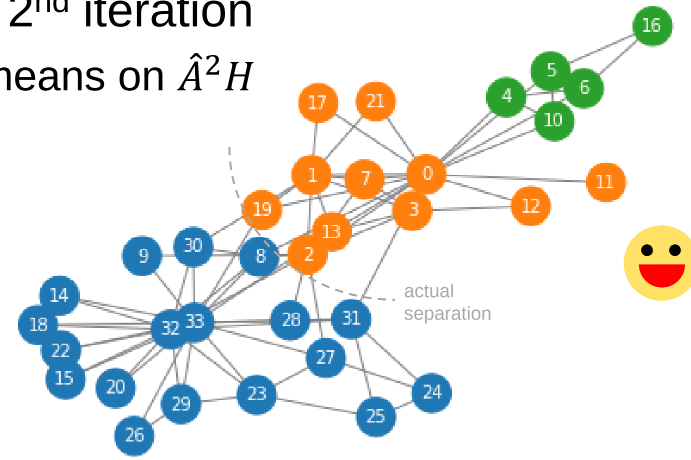
- Redo karate club with normalized $\hat{A} \leftarrow D^{-1}\hat{A}$

Initial



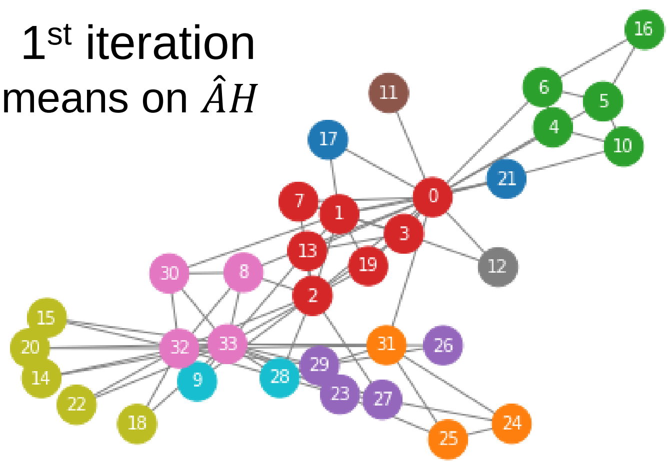
2nd iteration

k-means on $\hat{A}^2 H$



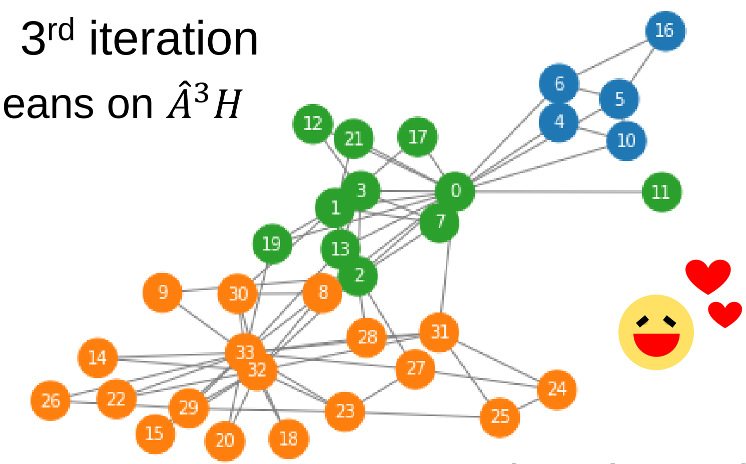
1st iteration

k-means on $\hat{A} H$



3rd iteration

k-means on $\hat{A}^3 H$



(no change)

(number of clusters determined with silhouette method)

Adding neighbors: evaluation

- Why do we need normalization
  - Without normalization, feature values for the nodes of high centrality would quickly add up, making them distinct from the nodes of low centrality
- How many iterations should be used?
  - Each iteration would “bunch up” neighboring features of 1 hop away (receptive field)
  - With that in mind, we should determine the number of iterations by the nature of the graph
  - The earliest (RNN-like) GNNs are iterated until convergence but these ideas were quickly replaced by (CNN-like) GNNs where the number of iterations is fixed as defined by the number of layers

Relationship to PageRank

For simplicity consider eigenvector centrality problem, that is, the undirected version of PageRank

- Problem Statement. Suppose each node v corresponds to a value x_v which is the sum of its neighbors' values

  $$x_v = \frac{1}{\lambda} \sum_{u \in N(v)} x_u$$

  where λ is some given constant and positive factor

  Given adjacency matrix A of a graph G, solve x_v for all $v \in G$
- Problem is equivalent to that of finding vector $\boldsymbol{x}$ such that

  $$\lambda \boldsymbol{x} = A\boldsymbol{x}$$

  - Solutions are all the eigenvectors that maximize $\boldsymbol{x}^\top A\boldsymbol{x}$
- Problem is also equivalent to that of **adding up all neighboring single-valued features, but excluding that of self, until convergence**

Adding neighbors: evaluation

- Benefits of strategy
  - Simplicity
  - Efficiently computed with adjacency matrix
- Disadvantage of strategy
  - Embeddings produced are of size of the number of nodes in the graph
    $\Rightarrow$ Learn a **transformation matrix** $W: R^{|V|} \rightarrow R^{d}$ for some smaller d

Transformation matrix W

- W is typically a linear transformation layer of size $|V| \times d$ where d is the target dimensionality of the embeddings

- Combined with the adjacency matrix $\hat{A}$, we now have a complete matrix formulation for computing embedding h_v of a node v from (itself and) its neighbors, in the form of

$$h_v \leftarrow \left(\hat{A}\right)_v HW$$

  where

  - $\left(\hat{A}\right)_v$ is the row in $\hat{A}$ for the node v, and
  - H is a matrix containing the features/embeddings of all the nodes (of course, only the rows in H with non-zero entries in $\left(\hat{A}\right)_v$ are needed for computing h_v)

- **Variations in this formula lead to various frameworks**

Variations

- Message-aggregation ($\mathrm{MSG\text{-}AGG}$)
  - First transform features/embeddings (MSG), then aggregate transformed embeddings (AGG)

$$h_v \leftarrow \underbrace{(\hat{A})_v \overbrace{(HW)}^{\text{message}}}_{\text{aggregate}}$$

- Separate computation of self and neighbors
  - Exclude entry for v from $(\hat{A})_v$, and let

$$h_v \leftarrow \mathrm{AGG}\left(\overbrace{(\hat{A})_v HW}^{\text{Aggregate only neighbors}}, \overbrace{h_v W'}^{\text{Self}}\right)$$

Learn a different transformation for self

Also denoted as B

where AGG is, for instance, concatenation

Frameworks

- **Graph Convolutional Network** (GCN)

  $$h_v \leftarrow (\hat{A})_v (HW) \quad \text{(basically just } \mathrm{MSG\text{-}AGG}\text{)}$$

- **GraphSAGE**
  - Exclude entry for v from $(\hat{A})_v$

$$h_v \leftarrow \underbrace{\left(\underbrace{\mathrm{CONCAT}\left(\overbrace{\mathrm{AGG}\left((\hat{A})_v H \right)}^{\text{Aggregate neighbors}}, \overbrace{h_v}^{\text{Self}} \right)}_{\text{Concatenate self \& aggregated neighbors}} \right) W}_{\text{Transform}}$$

AGG can be one of many options including MLP, LSTM, *etc.*

(Why use these? See Graph Isomorphism Network)

$\Rightarrow$ AGG is learnable

GraphSAGE

- $h_v \leftarrow \left(\mathrm{CONCAT}\left(\mathrm{AGG}\left(\left(\hat{A}\right)_v H\right), h_v\right)\right) W$
  - As mentioned AGG is learnable
  - $\hat{A}$ is also learnable
    - Generalize adjacency to **attention weights** α_{uv}

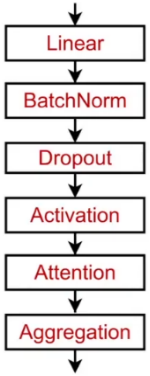
$$\alpha_{vu} = \frac{\exp(e_{vu})}{\sum_{x \in N(v)} \exp(e_{vx})}$$

where e_{vu} is a measure of how related u and v are, usually computed as $\mathrm{LINEAR}\left(\mathrm{CONCAT}(h_v W, h_u W)\right)$

- PyTorch Geometric (PyG) has implementations of these frameworks (GCN, GraphSAGE)

In practical use

- At this point we have not mentioned activation function or other elements of DL
  - For activation function just let $\sigma(h_v) \leftarrow h_v$
  - Mix and match as you like



- Embeddings can be used for many downstream tasks
  - We have earlier used k-means for clustering the final output
  - Better performed by constructing a neural network directly with the GNN layers

In practical use

- Adding graph elements
  - Features
    - Similar to feature engineering
  - Virtual nodes
    - Connecting all the nodes in a sparse but apparent subgraph to a virtual node will allow those nodes to better communicate
  - Virtual edges
    - Create new graph by systematically adding edges
    - Example: Given a bipartite graph, breaking the graph into two of only nodes of the same type is good for some analyses
      - Let A be the adjacency matrix of the bipartite graph G
      - A^2 then gives the number of paths of distance 2 between nodes in G
        $\Rightarrow$ an adjacency matrix between nodes of the same type
        $\Rightarrow$ allows us to separate G into two graphs, each of same node type
      - $A + A^2$ can form an adjacency matrix with heterogeneous edges

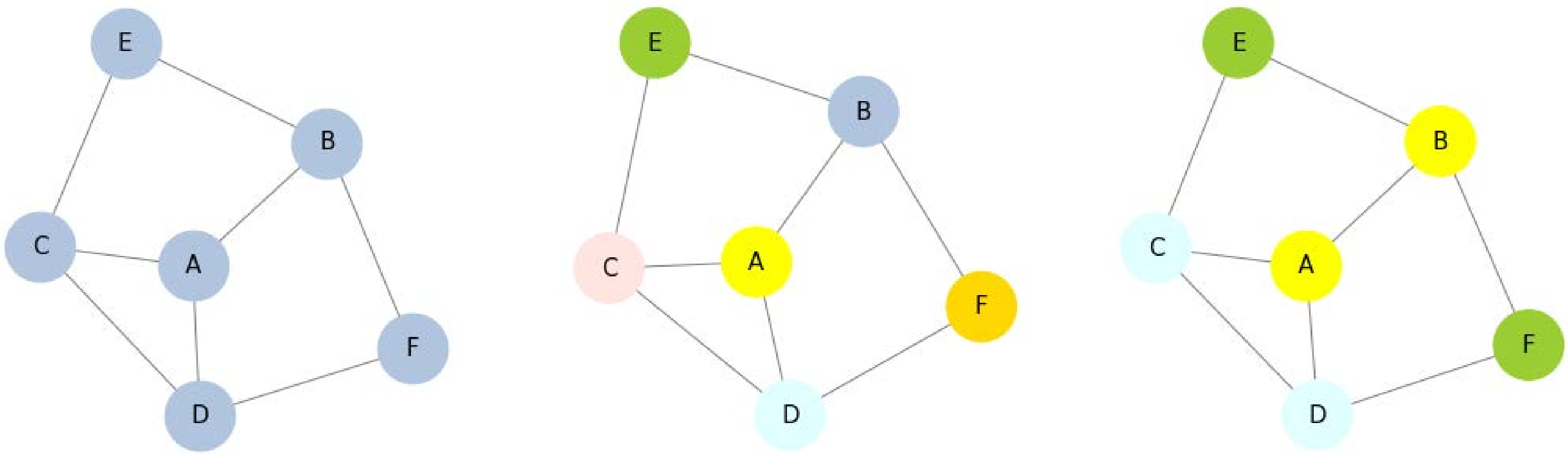
Training GNNs

- Using node embeddings as input to a prediction function
  - Embedding of 1 node can be used directly
  - Embedding of 2 nodes can be
    - Concatenated to form an edge embedding
    - Projected on each other to get their similarity
  - Embeddings of nodes of the entire graph can be
    - Summed, averaged, searched for max/min, *etc.*
    - Clustered, then the clusters summed, average, *etc.*, in a hierarchical fashion

- Using edge embeddings
  - Relational GCN

Distinguishing node embeddings

- Should C and D have the same embedding in the following graphs? Given that features are given by the colors and mutually exclusive (orthogonal)

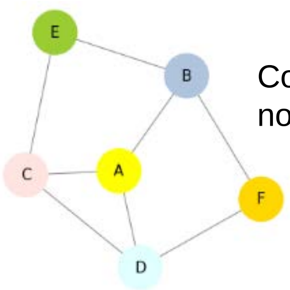


Yes? No? Should?

- How about A and B?
- **Idea**: Two nodes should have the same **embedding** if they have the same **feature** and **neighborhood structure**, and vice versa

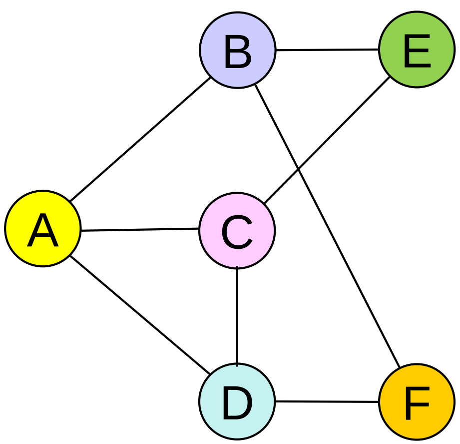
Distinguishing node embeddings

- Given a GCN of 2 layers, the embedding of A is computed as follows

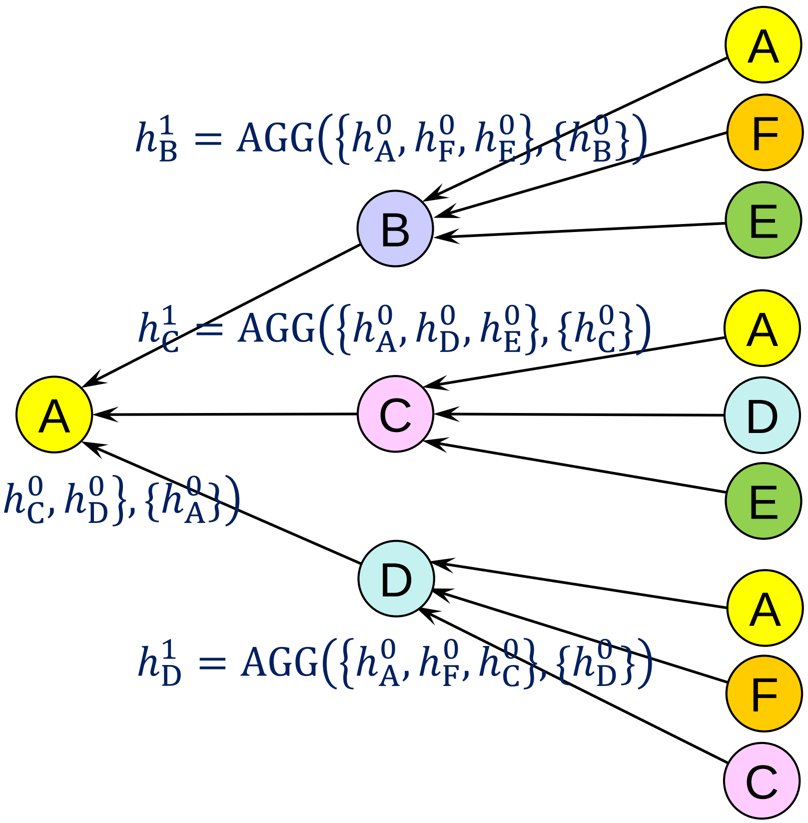


Consider the case where nodes have distinct features

Rearranged w.r.t. distance from A



Computation graph of A's embedding



$$h_{\mathrm{B}}^{1} = \mathrm{AGG}\left(\{h_{\mathrm{A}}^{0}, h_{\mathrm{F}}^{0}, h_{\mathrm{E}}^{0}\}, \{h_{\mathrm{B}}^{0}\}\right)$$

$$h_{\mathrm{C}}^{1} = \mathrm{AGG}\left(\{h_{\mathrm{A}}^{0}, h_{\mathrm{D}}^{0}, h_{\mathrm{E}}^{0}\}, \{h_{\mathrm{C}}^{0}\}\right)$$

$$h_{\mathrm{A}}^{1} = \mathrm{AGG}\left(\{h_{\mathrm{B}}^{0}, h_{\mathrm{C}}^{0}, h_{\mathrm{D}}^{0}\}, \{h_{\mathrm{A}}^{0}\}\right)$$

$$h_{\mathrm{D}}^{1} = \mathrm{AGG}\left(\{h_{\mathrm{A}}^{0}, h_{\mathrm{F}}^{0}, h_{\mathrm{C}}^{0}\}, \{h_{\mathrm{D}}^{0}\}\right)$$

Distinguishing node embeddings

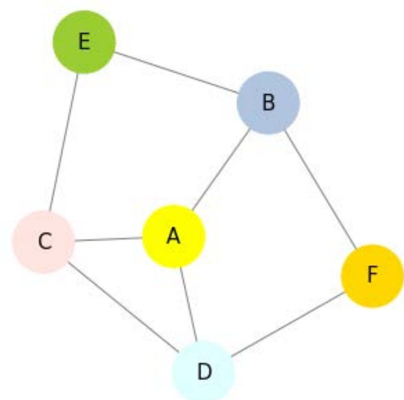
- Given a GCN of 2 layers, the embedding of A is computed as follows
  - Let $h_{\mathrm{A}}^0 = \boxed{1\,|\,0\,|\,0\,|\,0\,|\,0\,|\,0}$, $h_{\mathrm{B}}^0 = \boxed{0\,|\,1\,|\,0\,|\,0\,|\,0\,|\,0}$, $h_{\mathrm{C}}^0 = \boxed{0\,|\,0\,|\,1\,|\,0\,|\,0\,|\,0}$, $h_{\mathrm{D}}^0 = \boxed{0\,|\,0\,|\,0\,|\,1\,|\,0\,|\,0}$, $h_{\mathrm{D}}^0 = \boxed{0\,|\,0\,|\,0\,|\,0\,|\,1\,|\,0}$, $h_{\mathrm{D}}^0 = \boxed{0\,|\,0\,|\,0\,|\,0\,|\,0\,|\,1}$, and let AGG be **addition**. Then
    - $h_{\mathrm{B}}^1 = \mathrm{AGG}\left(\{h_{\mathrm{A}}^0, h_{\mathrm{F}}^0, h_{\mathrm{E}}^0\}, \{h_{\mathrm{B}}^0\}\right) = 1|1|0|0|1|1$
    - $h_{\mathrm{C}}^1 = \mathrm{AGG}\left(\{h_{\mathrm{A}}^0, h_{\mathrm{D}}^0, h_{\mathrm{E}}^0\}, \{h_{\mathrm{C}}^0\}\right) = 1|0|1|1|1|0$
    - $h_{\mathrm{D}}^1 = \mathrm{AGG}\left(\{h_{\mathrm{A}}^0, h_{\mathrm{F}}^0, h_{\mathrm{C}}^0\}, \{h_{\mathrm{D}}^0\}\right) = 1|0|1|1|0|1$
    - $h_{\mathrm{A}}^1 = \mathrm{AGG}\left(\{h_{\mathrm{B}}^0, h_{\mathrm{C}}^0, h_{\mathrm{D}}^0\}, \{h_{\mathrm{A}}^0\}\right) = 1|1|1|1|0|0$

Compute $AGG(X)$ as AH, where A is the adjacency matrix (with self loop), and H is a matrix containing all the embeddings in X

  - Finally the embedding of A is
    - $h_{\mathrm{A}}^2 = \mathrm{AGG}\left(\{h_{\mathrm{B}}^1, h_{\mathrm{C}}^1, h_{\mathrm{D}}^1\}, \{h_{\mathrm{A}}^1\}\right) = 4|2|3|3|2|2$
  - Similarly, $h_{\mathrm{B}}^2 = 2|4|2|2|2|2$
    $h_{\mathrm{C}}^2 = 3|2|4|3|2|1$
    $h_{\mathrm{D}}^2 = 3|2|3|4|1|2$
    $h_{E}^2 = 2|2|2|1|3|1$
    $h_{F}^2 = 2|2|1|2|1|3$

Distinguishing node embeddings

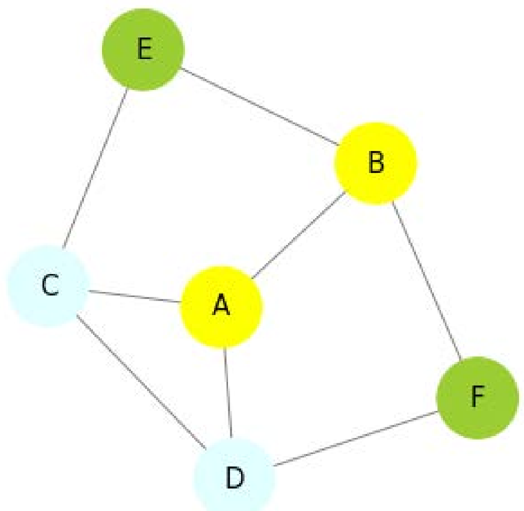
- Given a GCN of 2 layers, the embedding of A is computed as follows
  - Let h_{A}^{0}=$\boxed{1|0|0|0|0|0}$, h_{B}^{0}=$\boxed{0|1|0|0|0|0}$, h_{C}^{0}=$\boxed{0|0|1|0|0|0}$, h_{D}^{0}=$\boxed{0|0|0|1|0|0}$, h_{D}^{0}=$\boxed{0|0|0|0|1|0}$, h_{D}^{0}=$\boxed{0|0|0|0|0|1}$, and let AGG be **addition**. Then
  - $h_A^2 = 4|2|3|3|2|2$
    $h_B^2 = 2|4|2|2|2|2$
    $h_C^2 = 3|2|4|3|2|1$
    $h_D^2 = 3|2|3|4|1|2$
    $h_E^2 = 2|2|2|1|3|1$
    $h_F^2 = 2|2|1|2|1|3$



    By induction they will be distinct for all subsequent iterations
  - For a graph with **distinct node features**, the embeddings will be distinct under **addition** regardless of **neighborhood structure** or **iterations**
    - With the exception of “twin nodes” that are connected only to each other (in which case they will become equal after the first iteration)

Distinguishing node embeddings

- Given a GCN of 2 layers, the embedding of A is computed as follows
  - Let h_{A}^0=h_{B}^0=$\boxed{1 \mid 0 \mid 0}$, h_{C}^0=h_{D}^0=$\boxed{0 \mid 1 \mid 0}$, h_{E}^0=h_{F}^0=$\boxed{0 \mid 0 \mid 1}$, and let AGG be **addition**. Then, for the following graph



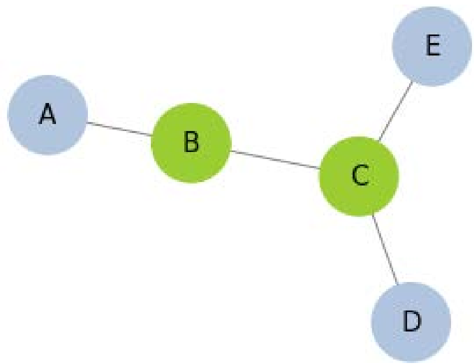
$$h_{\mathrm{A}}^1 = 2|2|0 \qquad h_{\mathrm{A}}^2 = 6|6|4$$
$$h_{\mathrm{B}}^1 = 2|0|2 \qquad h_{\mathrm{B}}^2 = 6|4|4$$
$$h_{\mathrm{A}}^2 \neq h_{\mathrm{B}}^2$$
$$h_{\mathrm{C}}^1 = 1|2|1 \qquad h_{\mathrm{C}}^2 = 5|7|3$$
$$h_{\mathrm{D}}^1 = 1|2|1 \qquad h_{\mathrm{D}}^2 = 5|7|3$$
$$h_{\mathrm{C}}^2 = h_{\mathrm{D}}^2$$
$$h_{\mathrm{E}}^1 = 1|1|1 \qquad h_{\mathrm{E}}^2 = 4|3|4$$
$$h_{\mathrm{F}}^1 = 1|1|1 \qquad h_{\mathrm{F}}^2 = 4|3|4$$
$$h_{\mathrm{E}}^2 = h_{\mathrm{F}}^2$$

  - Two nodes with the **same feature** will always have the same embedding under **addition** if and only if they have the same **neighborhood structure**
    - What about other AGG functions, e.g. **mean**?

Distinguishing node embeddings

- Let h_{A}^0=h_{D}^0=h_{E}^0= $\boxed{1|0}$, h_{B}^0=h_{C}^0= $\boxed{0|1}$, and let AGG be **mean**. Then, for the following graph

Compute $AGG(X)$ as $\hat{A}H$



$h_{\mathrm{A}}^1 = 0.5 \,|\, 0.5$
$h_{\mathrm{B}}^1 = 0.33 \,|\, 0.67$
$h_{\mathrm{C}}^1 = 0.5 \,|\, 0.5$
$h_{\mathrm{D}}^1 = 0.5 \,|\, 0.5$
$h_{\mathrm{E}}^1 = 0.5 \,|\, 0.5$

- As expected, $h_{\mathrm{A}}^1 = h_{\mathrm{D}}^1 = h_{\mathrm{E}}^1$ due to the same feature and neighborhood structure (within 1 hop)
- However, $h_{\mathrm{A}}^1 = h_{\mathrm{C}}^1$ in spite of their differences in both features and neighborhood structure

  ⇒ **mean** cannot get distinct embeddings for distinct nodes

  - Even though this is true only for the first iteration in this example, similar examples can be obtained for any number of layers

Distinguishing node embeddings

- While our earlier examples did not consider the transformation W or the activation function σ, the arguments are just as valid with them considered
- A function that can distinguish the nodes of distinct feature and neighborhood structure is one that is **injective**
  - **mean** and **max** are not injective
  - On the other hand, **sum** has problems as mentioned
- Theorem (Xu *et al.* 2019). Any injective AGG function can be expressed as $\Phi(\sum_{x \in S} f(x))$ for some non-linear Φ and linear f
- Since MLP is able to approximate any function, we can learn Φ and f with non-linear MLP_{Φ} and linear MLP_f

$$\mathrm{AGG} = \mathrm{MLP}_{\Phi}\left(\sum_{x \in S} \mathrm{MLP}_f(x)\right)$$

$\Rightarrow$ **Graph Isomorphism Network (GIN)**