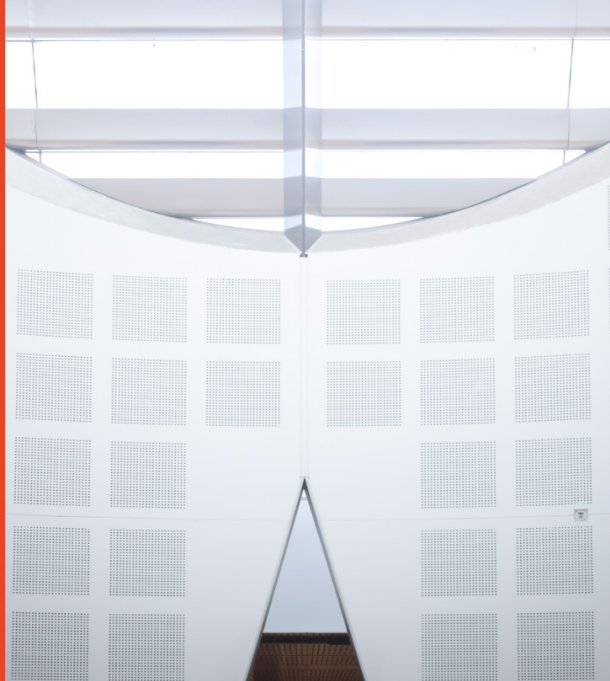
COMP5313/COMP4313 - Large Scale Networks

Week 7: Machine Learning on Graphs (I)

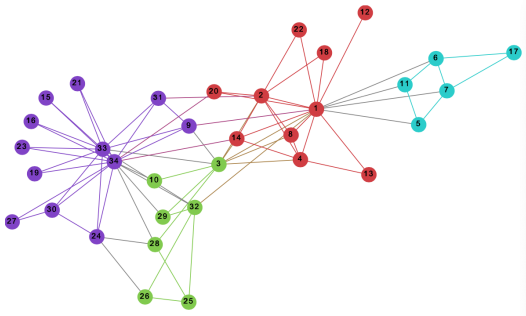
Lijun Chang

April 10, 2025

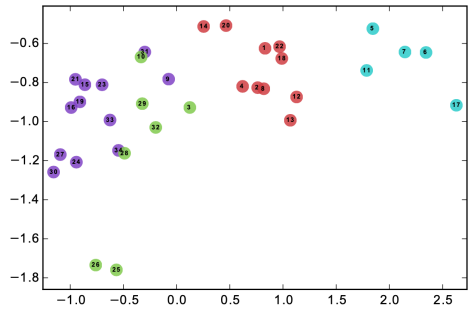
THE UNIVERSITY OF SYDNEY



Graph Representation Learning (Node Embedding)



(a) Zachary's karate club network



(b) Embedding in $\mathbb{R}^2$ (by DeepWalk)

- Node color represents a modularity-based clustering of the graph

Bryan Perozzi, Rami Al-Rfou, Steven Skiena: DeepWalk: online learning of social representations. KDD 2014: 701-710

Outline

Background on Machine/Deep Learning (optional)

Machine Learning on Graphs

Graph Neural Networks (GNNs)

What is Machine Learning?

- Machine learning (ML) is the study of computer algorithms that can improve automatically through data and experience.
- Algorithms and processes that 'learn' in the sense of being able to generalize past data and experiences in order to predict future outcomes.
- A computer program is said to learn form experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

Supervised Learning via Function Approximation

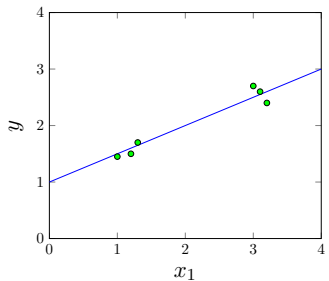
- Experience E is a dataset, containing measures of objects w.r.t a set of features
  - Features are partitioned into input features $\boldsymbol{x} \in \mathbb{R}^p$ and an output feature $y \in \mathbb{R}$
    - $\boldsymbol{x} =$ (type of a property, area, no. of bedrooms), $y =$ price.
    - $\boldsymbol{x} =$ (pixels of an image at each position), $y =$ cat or not.
  - Each example in the dataset is an input-output pair $(\boldsymbol{x}_i, y_i)$
    - the measures of one particular object
  - Training data: $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$

- Task T is mapping inputs $\boldsymbol{x}$ to output y
  - Machine learning is to learn a function $f : \boldsymbol{x} \to y$

- Performance P is how well f fits the training data $\mathcal{D}$, and how well f generalizes to unseen (test) data (i.e., given arbitrary $\boldsymbol{x}$, predict y)
  - To avoid overfitting (to training data), there is an assumption/restriction on the function f (e.g., the degree of a polynomial function)

Linear Regression (for Quantitative Output)

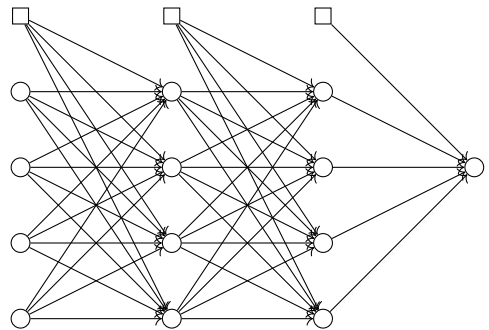
- Problem setting:
  - Features: $\boldsymbol{x} =$ (type of a property, area, no. of bedrooms), $y =$ price.
  - Learn a linear function $f(\boldsymbol{x}) = b + \sum_{i=1}^{p} w_i x_i = b + \boldsymbol{x}^\top \boldsymbol{w}$ to approximate y
    - The learning algorithm is to estimate $b, w_1, \ldots, w_p$.
    - We assume column vectors
  - Performance is measured by the mean square error on the training dataset $\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2$.
    - N: the number of training examples
- Matrix form (by ignoring b):
  $\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2$
  - $\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$ and $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{bmatrix}$
- Assume the features are independent, then
  $\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \mathrm{MSE}(\boldsymbol{w}) = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$.



Multi-Layer Perceptron (MLP)

Now, let's consider one input example $(\boldsymbol{x}, y)$

- Linear regression: $f_1(\boldsymbol{x}) = \boldsymbol{w}_1^\top \boldsymbol{x} + b_1$
- What if we have multiple outputs?
  - $f_i(\boldsymbol{x}) = \boldsymbol{w}_i^\top \boldsymbol{x} + b_i$
  - $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ where $\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots \end{bmatrix}^\top$
- What if we stack multiple layers?
  - $\boldsymbol{f}^{(k)}(\boldsymbol{x}^{(k-1)}) = \boldsymbol{W}^{(k)}\boldsymbol{x}^{(k-1)} + \boldsymbol{b}^{(k)}$
  - But this is still a linear function!
- Remedy: add a nonlinear activation function to each hidden unit.
  - E.g., $f_i(\boldsymbol{x}) = \mathrm{ReLU}(\boldsymbol{w}_i^\top \boldsymbol{x} + b_i)$



Every edge carries a scalar weight

Multi-Layer Perceptron (MLP)

- MLP is also known as *deep feedforward network* or *feedforward neural network* or *fully-connected neural network*.

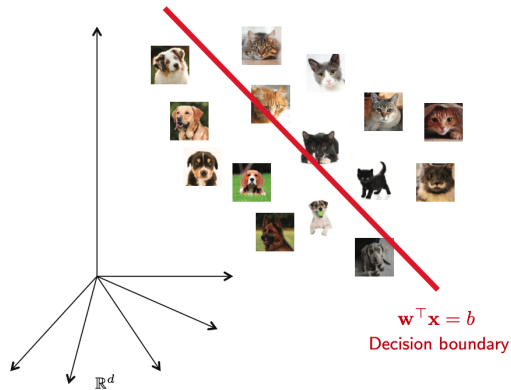
- The functions learned have the form $f(\boldsymbol{x}) = L^{(K)}(\sigma(\ldots(L^{(2)}(\sigma(L^{(1)}(\boldsymbol{x})))))).$
  - A composition of affine functions $L^{(k)}(\boldsymbol{x}^{(k-1)}) = \boldsymbol{W}^{(k)}\boldsymbol{x}^{(k-1)} + \boldsymbol{b}^{(k)}$ with nonlinear activation functions σ (e.g., ReLU)
    - The activation function acts on each component of a vector or matrix.
  - The hidden layers add depth to the network.
    - It is that depth which has allowed the composite function f to be so successful in deep learnings

- The universal approximation theorem says that with one hidden layer, MLP can approximate any well-behaved function $f(\cdot)$ within an arbitrarily small error ε.[1]

[1] https://en.wikipedia.org/wiki/Universal_approximation_theorem

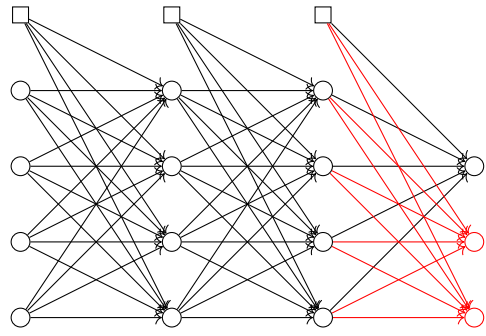
Classification (for Qualitative Output)

- Problem setting:
  - Features: $\boldsymbol{x} =$ (pixels of an image at each position), $y =$ cat or dog.
    - $\boldsymbol{x} \in \mathbb{R}^d$ (e.g., for 1024×1024 images $d \approx 10^6$).
  - Learn a classification function $f : \mathbb{R}^d \to \{1, \ldots, C\}$ in C classes
  - Performance is measured by maximum likelihood
- Traditional methods: logistic regression, linear discriminant analysis, separating hyperplane.



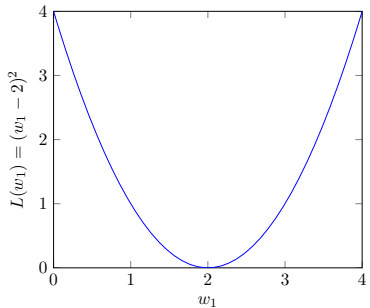
Classification in MLP (for Qualitative Output)

- There is one output unit for each class
  - $\boldsymbol{z} = \boldsymbol{W}^{(K)}\boldsymbol{x}^{(K-1)} + \boldsymbol{b}^{(K)}$
- Apply the softmax function to the output layer
  - $\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$
  - This is regarded as the probability $P(\text{y} = i \mid \boldsymbol{x}, \boldsymbol{\theta})$ of seeing class i for input $\boldsymbol{x}$ and parameters $\boldsymbol{\theta} = \{\boldsymbol{b}^{(1)}, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{b}^{(K)}, \boldsymbol{W}^{(K)}\}$
- Maximize the likelihood $\prod_{i=1}^{N} P(\text{y} = y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$
  - Equivalent to minimize the negative log-likelihood loss function $\ell(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \log P(\text{y} = y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$



How to Find the Best Parameters (i.e., How to Train the Model)?

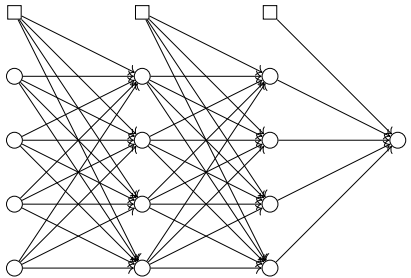
- For linear regression, we aim to minimize $\mathrm{MSE}(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$.
  - Luckily, we got a closed-form solution $\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$.
- In general, we aim to minimize the loss function $\ell(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log P(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta})$
  - Unlikely to have closed-form solution
- Gradient-based optimization
  - $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}) = \left[\frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_1} \quad \frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_2} \quad \cdots\right]^\top$ points the uphill direction
  - $\ell(\boldsymbol{\theta})$ decreases the fastest along the vector direction $-\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$
  - $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \rho\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)})$
    - ρ is the learning rate.



- Stochastic Gradient Descent (SGD) for computational efficiency
  - Instead of using all training data to compute $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}^{(t)})$, use a small random subset (called mini-batch)

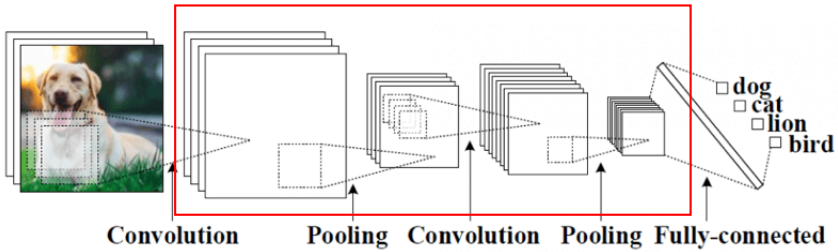
Learning Procedure

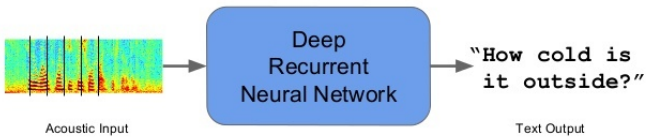
- One-step learning procedure for a mini-batch of the training data:
  - Perform forward propagation through the deep neural network for the subset of training data.
  - Compute the loss at the output layer for the subset of training data.
  - Backpropagate to compute the gradient and to update the weight parameters $\boldsymbol{W}^{(k)}$ in each layer.



Every edge carries a scalar weight

Deep Neural Network Architectures for Specific Applications





Outline

Background on Machine/Deep Learning (optional)

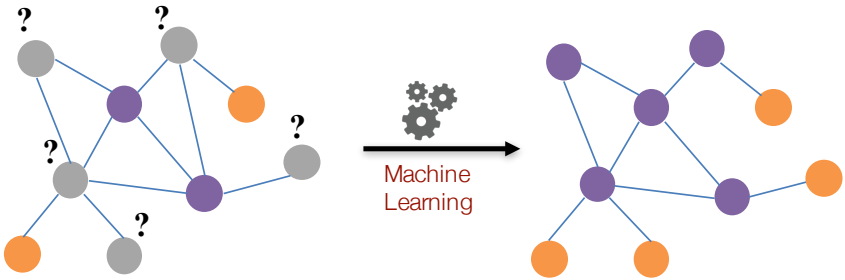
Machine Learning on Graphs

Graph Neural Networks (GNNs)

Typical Machine Learning Tasks on Graphs

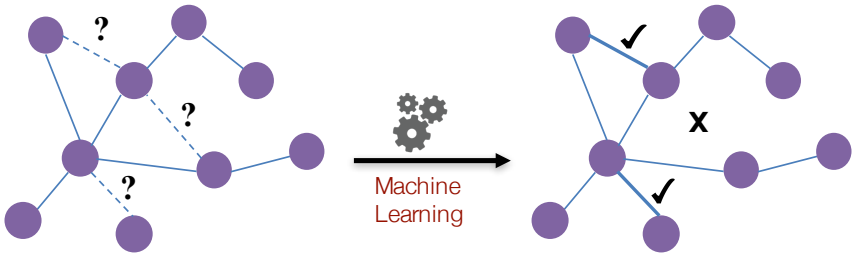
- Node classification
  - Predict a type of a given node (e.g., topic of documents in a citation network)
- Link prediction
  - Predict whether two nodes are (or will be) linked
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks
- Graph classification
  - Predict a type of a given graph

Node Classification



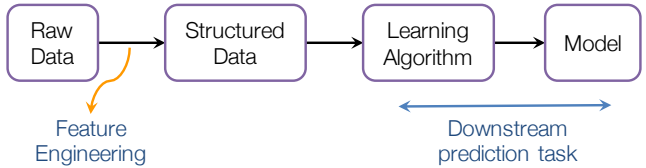
Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

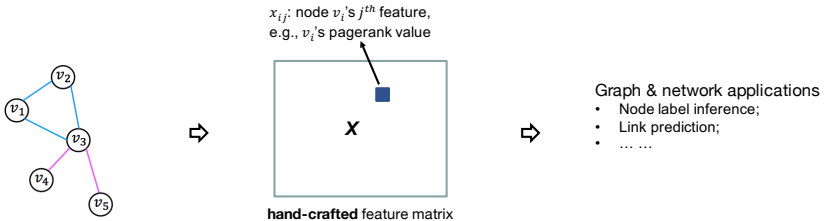
Link Prediction



Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

Traditional ML Life Cycle

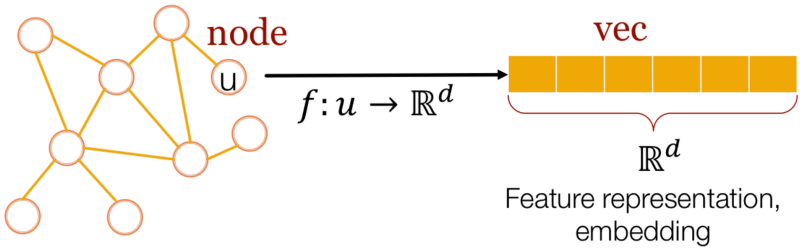




Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

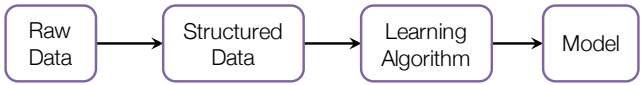
Feature Engineering/Learning

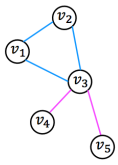
- Task dependent/independent feature engineering/learning for machine learning on graphs

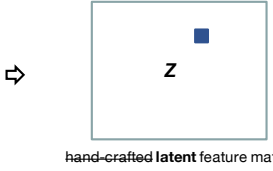


Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

Deep Learning-Driven ML Life Cycle





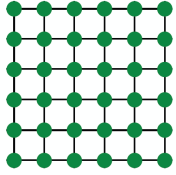


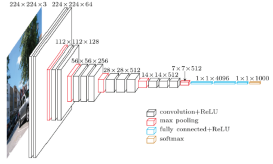
Graph & network applications
- Node label inference;
- Link prediction;
- … …

Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

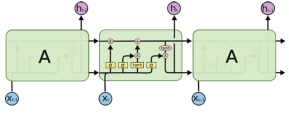
Challenges

- Modern deep learning toolbox is designed for simple sequences or grids
  - Convolutional Neural Networks (CNNs) for fixed-size images/grids ...





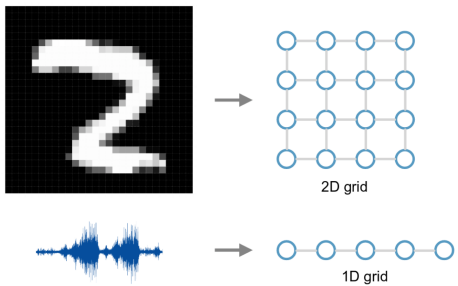
  - Recurrent Neural Networks (RNNs) or word2vec for text/sequences ...



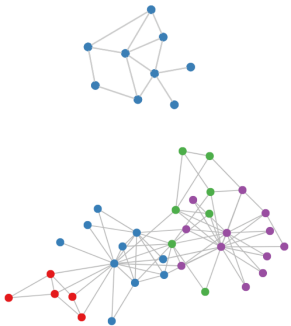
Leskovec, J., Hamilton, W. L., Ying, R., and Sosic, R.: Representation learning on networks. WWW 2018 Tutorial.

Challenges

- Graphs are known for the flexibility of the data structure
  - Arbitrary size, complex topographical structure (i.e., no spatial locality like grids), different number of neighbors, no fixed node ordering or reference point



Vs.



Outline

Background on Machine/Deep Learning (optional)

Machine Learning on Graphs

Graph Neural Networks (GNNs)

Setup for Node Classification

Assume we have an undirected graph G:

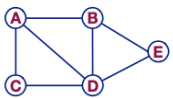
- V is the node set ($|V| = N$)
- $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix (assume binary)
- $\boldsymbol{X} \in \mathbb{R}^{N \times q}$ is a matrix of input node features
  - Categorical attributes, text, image data
    - E.g., profile information in a social network
  - Node degrees, clustering coefficients, etc.
  - Indicator vectors (i.e., one-hot encoding of each node)
- For each labeled node $v \in V_{\text{train}}$, its label is denoted by $\boldsymbol{y}_v \in \mathbb{R}^C$, a one-hot encoding vector of the label
- The task is to predict the labels for nodes in $V \setminus V_{\text{train}}$

A First Naïve Approach

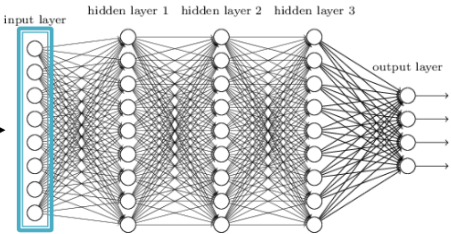
- Ignore the graph structure information (i.e., ignore $\boldsymbol{A}$), and use only $\boldsymbol{X}$ and $\boldsymbol{y}_v$ for training and prediction
- This may perform well on some datasets where $\boldsymbol{X}$ contains a lot of information
- In general, if we incorporate graph structure information, we can do better

A Second Naïve Approach

- Join adjacency matrix and feature matrix, and feed them into a deep neural network



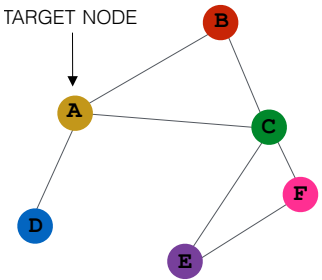
	A	B	C	D	E	Feat	
A	0	1	1	1	0	1	0
B	1	0	0	1	1	0	0
C	1	0	0	1	0	0	1
D	1	1	1	0	1	1	1
E	0	1	0	1	0	1	0



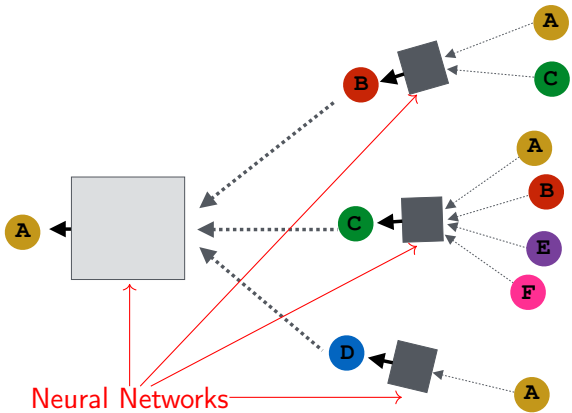
- Issues with this idea:
  - $O(N)$ parameters
  - Not applicable to graphs of different sizes
  - Not invariant to node ordering

Neighborhood Aggregation

- Graph neural network approach: combine graph structure and node features via neighborhood aggregation, which propagates features based on the graph topology.
- Intuition: nodes aggregate information from their neighbors using neural networks

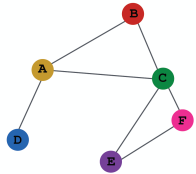


INPUT GRAPH

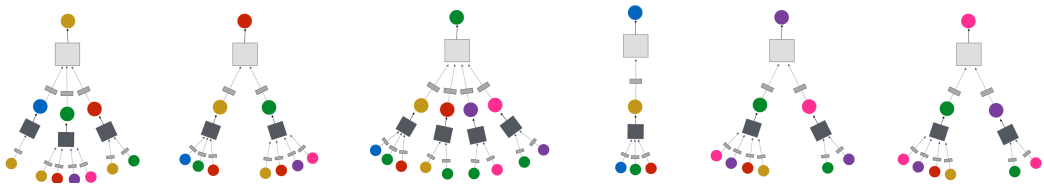


Neighborhood Aggregation

- Intuition: network neighborhood defines a computation graph



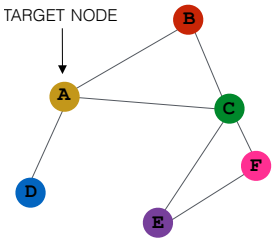
INPUT GRAPH



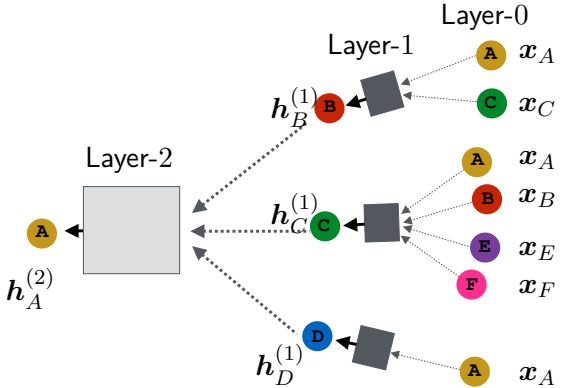
Every node defines a computation graph!

Neighborhood Aggregation

- Model can be of arbitrary depth
  - Nodes have embeddings at each layer
  - Layer-0 embedding of node u is its input feature $\boldsymbol{x}_u$
  - Layer-k embedding gets information from nodes that are k hops away

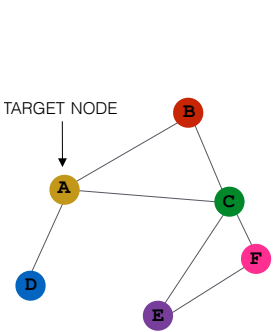


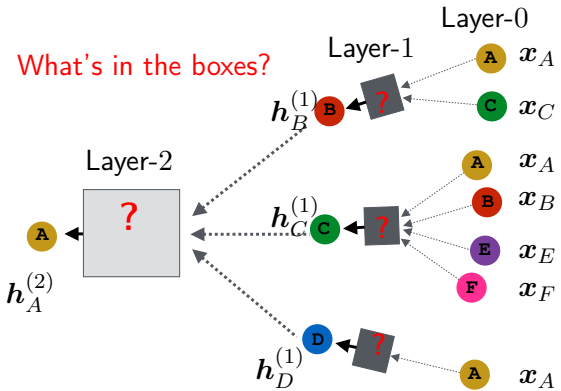
INPUT GRAPH



Neighborhood Aggregation

- Key distinctions of different graph neural networks models are in how they aggregate information across the layers

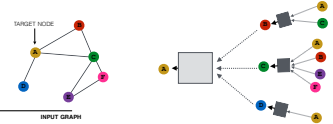




Graph Convolutional Networks (GCN) [2]

Weighted average of neighbor messages and then applying a neural network

- $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u, \forall u \in V$
- For $k = 1, \ldots, K$
  - $\boldsymbol{h}_u^{(k)} = \sigma\left(\boldsymbol{W}^{(k)}\left(\sum_{v \in N(u) \cup \{u\}} \frac{\boldsymbol{h}_v^{(k-1)}}{\sqrt{(d(v)+1)(d(u)+1)}}\right)\right), \quad \forall u \in V$
    - $N(u)$ is the neighbors of u, $d(u) = |N(u)|$ is the degree of u
    - $\boldsymbol{W}^{(k)}$ is the trainable weight parameters
    - $\sigma(\cdot)$ is element-wise activation function (e.g., $\mathrm{ReLu}(x) = \max(0, x)$)
- $\boldsymbol{z}_u = \boldsymbol{h}_u^{(K)}$ is the learned representation of u, $\forall u \in V$.
  - These representation can be fed into any loss function to train the weight parameters. E.g., $\{(\boldsymbol{z}_u, \boldsymbol{y}_u)\}$ can be fed into logistic regression for classification.



[2] Thomas N. Kipf, Max Welling: Semi-Supervised Classification with Graph Convolutional Networks. ICLR (Poster) 2017

Graph Convolutional Networks (GCN)

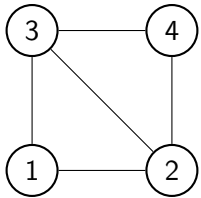
In matrix form

- $\boldsymbol{H}^{(0)} = \boldsymbol{X} \in \mathbb{R}^{N \times q}$
  - $\boldsymbol{H}^{(0)} = \begin{bmatrix} \boldsymbol{h}_1^{(0)} & \cdots & \boldsymbol{h}_N^{(0)} \end{bmatrix}^\top$
- For $k = 1, \ldots, K$
  - $\boldsymbol{H}^{(k)} = \sigma\left(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(k-1)}\boldsymbol{W}^{(k)}\right)$
    - $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$
    - $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$
- $\boldsymbol{Z} = \boldsymbol{H}^{(K)}$

This is called graph convolutional network, because $\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{(k-1)}$ is a (first-order) approximation of localized spectral filter, i.e., graph convolution.

GCN Propagation - A simple example

- Consider the graph below

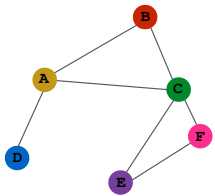


- We have

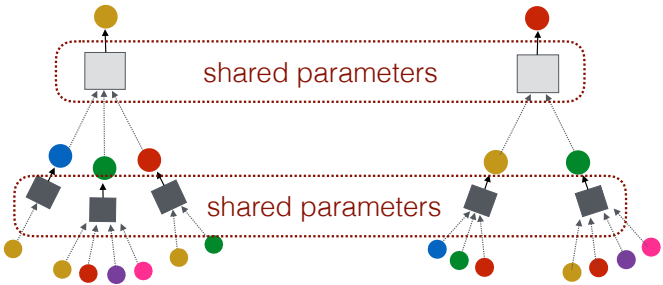
$$\tilde{\boldsymbol{A}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad \tilde{\boldsymbol{D}} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \quad \tilde{\boldsymbol{D}}^{-1} = \begin{bmatrix} 0.33 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 \\ 0 & 0 & 0.25 & 0 \\ 0 & 0 & 0 & 0.33 \end{bmatrix}$$

Inductive Capability

- The same weight parameters are shared for the same layer
- The number of weight parameters is sublinear in N
- This makes training on large graphs possible.



INPUT GRAPH

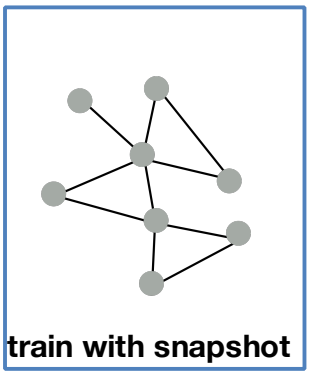


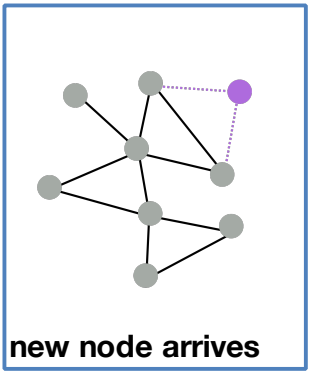
Compute graph for node A

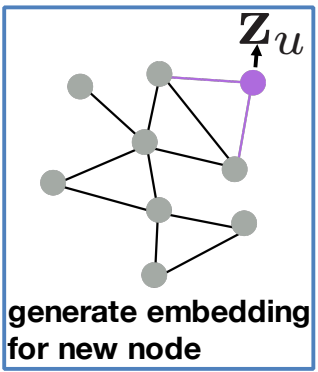
Compute graph for node B

Inductive Capability

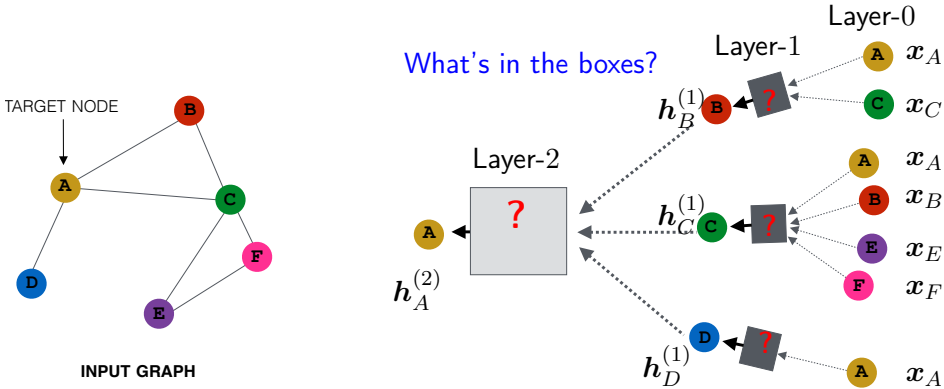
- The same weight parameters are shared for the same layer
- We can generalize to unseen nodes.







Other Models of Graph Neural Networks (GNNs)



- In general, each box conducts two operations
  - $\boldsymbol{a}_u^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_v^{(k-1)} \mid v \in N(u)\right\}\right)$
  - $\boldsymbol{h}_u^{(k)} = \text{COMBINE}^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{a}_u^{(k)}\right)$

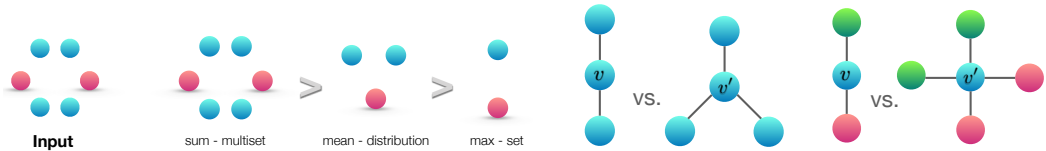
GraphSAGE [3]

- Three variants of $\boldsymbol{a}_u^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_v^{(k-1)} \mid v \in N(u)\right\}\right)$
  - Mean: take a weighted average of neighbors
    - $\boldsymbol{a}_u^{(k)} = \sum_{v \in N(u)} \frac{\boldsymbol{h}_v^{(k-1)}}{|N(u)|}$
  - Pool: Transform neighbor vectors and apply symmetric vector function
    - $\boldsymbol{a}_u^{(k)} = \gamma\left(\left\{\text{ReLU}(\boldsymbol{Q}^{(k)}\boldsymbol{h}_v^{(k-1)}) \mid v \in N(u)\right\}\right)$
    - $\gamma(\cdot)$ is element-wise mean/max, $\boldsymbol{Q}^{(k)}$ is weight parameter
  - LSTM: Apply LSTM to reshuffled neighbors
    - $\boldsymbol{a}_u^{(k)} = \text{LSTM}\left(\left[\boldsymbol{h}_v^{(k-1)} \mid v \in \pi(N(u))\right]\right)$
- $\boldsymbol{h}_u^{(k)} = \text{COMBINE}^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{a}_u^{(k)}\right)$: concatenation followed by a linear mapping
  - $\boldsymbol{h}_u^{(k)} = \text{ReLU}\left(\boldsymbol{W}^{(k)}\begin{bmatrix}\boldsymbol{h}_u^{(k-1)} \\ \boldsymbol{a}_u^{(k)}\end{bmatrix}\right)$

[3] William L. Hamilton, Zhitao Ying, Jure Leskovec: Inductive Representation Learning on Large Graphs. NIPS 2017: 1024-1034

Graph Isomorphism Network (GIN) [4]

- $\boldsymbol{a}_u^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{\boldsymbol{h}_v^{(k-1)} \mid v \in N(u)\right\}\right) = \sum_{v \in N(u)} \boldsymbol{h}_v^{(k-1)}$



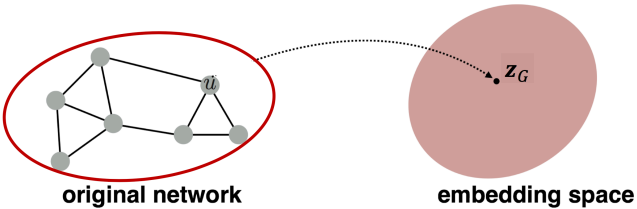
(a) Sum is most expressive (b) Mean/Max fail (c) Mean/Max fail

- $\boldsymbol{h}_u^{(k)} = \text{COMBINE}^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{a}_u^{(k)}\right) = \text{MLP}^{(k)}\left((1+\varepsilon^{(k)})\boldsymbol{h}_u^{(k-1)} + \boldsymbol{a}_u^{(k)}\right)$
  - $\text{MLP}^{(k)}$ is a 2-layer perceptron in the paper
  - $\varepsilon^{(k)} = 0$ performs well in practice

[4] Keyulu Xu, Weihua Hu, Jure Leskovec, Stefanie Jegelka: How Powerful are Graph Neural Networks? ICLR 2019

Graph Embeddings

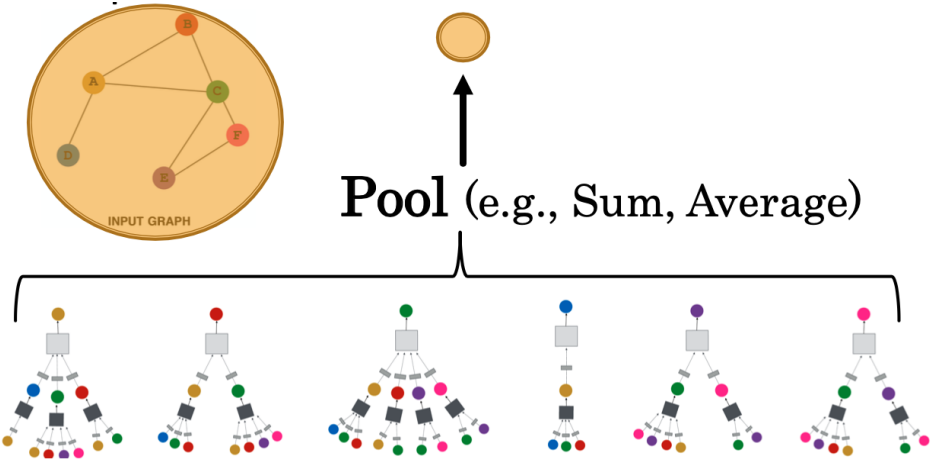
- So far we discussed generating embeddings for individual nodes
- How about mapping entire graphs to embeddings?



- Example applications: graph classification
  - Classifying toxic vs. non-toxic molecules
  - Identifying anomalous graphs

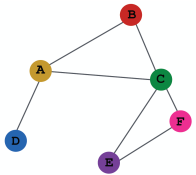
Graph Embeddings

- Obtain graph embedding by pooling node embeddings.

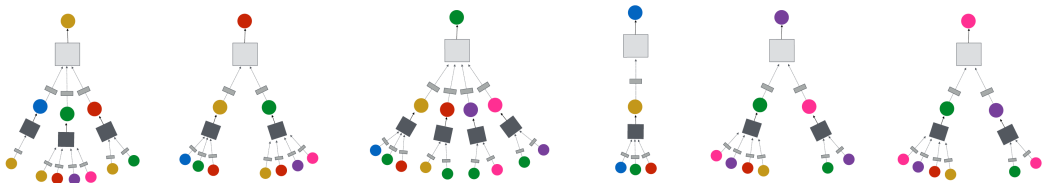


Limitation of GNNs

- Two nodes with the same computation graph will have the same embedding



INPUT GRAPH



Every node defines a computation graph!

Conclusion

- Deep models are able to learn/approximate complex functions $f(\boldsymbol{x})$ with few parameters
- Non-Euclidean nature of graphs makes the problem challenging
- Machine learning on graphs allows:
  - node classification & link prediction & graph classification ...
- Graph neural networks
  - provide end-to-end machine learning
  - are inductive

Reference

- Graph Representation Learning by William L. Hamilton:
  https://www.cs.mcgill.ca/~wlh/grl_book/