CSCI446/946 Big Data Analytics

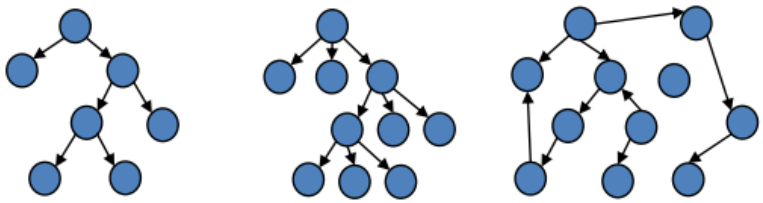
Week 12 Social Media Analytics and Deep Graph Learning

School of Computing and Information Technology
University of Wollongong Australia

Spring 2022

Modelling complex data structures

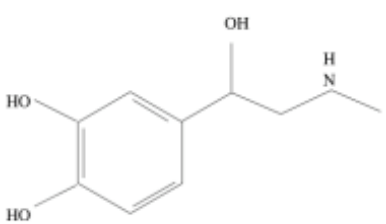
- So far we have introduced methods for modelling large sets of data represented as either
  - Vectors
  - Sequences
- What about more complex data structures?
  - Binary trees
  - N-arry trees
  - Graphs

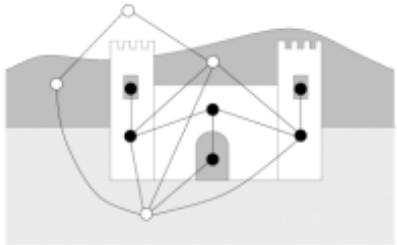


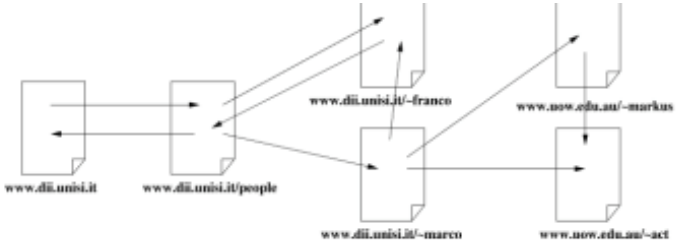
- Big Data is increasingly represented in complex data structures
  - To represent relationships between entities
  - I.e. social networks, Web, …

Modelling complex data structures

- Modelling of Data Graphs
  - Conventional NN methods assume that the inputs are independent.
    - Except recurrent systems which can model time sequence information.
  - Fact is that nothing is ever truly independent.
    - Oftentimes such dependencies can be neglected without significantly influencing the quality of results.
  - Applications where dependencies should not be neglected.
    - Examples: Molecular chemistry, WWW, text analytics, image analytics, social networks,…
    - Data that is represented as a tree, or graph.
    - How about modelling complex dependencies?







Graph Neural Networks

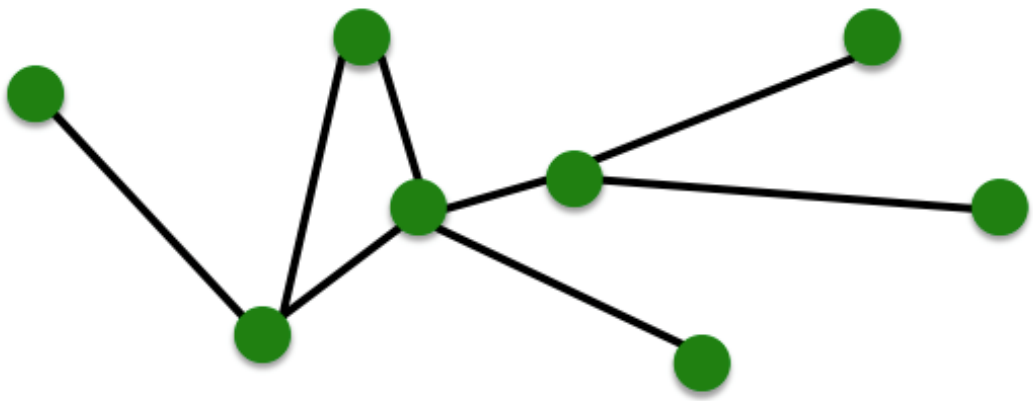
- Graph Neural Learning Systems were introduced since 1993.
  - Labelling Recursive AAM (unsupervised)
  - Graph Self-Organizing Maps (unsupervised)
  - Graph Neural Network (supervised)
  - Convolutional GNN (supervised)
- But these algorithms received broader attention only since 2018.

References:
[1] A. Sperduti, A. Starita, On the access by content capabilities of the LRAAM, ICNN'94.
[2] Hagenbuchner, M, Sperduti, A & Tsoi, AC, A self-organizing map for adaptive processing of structured data, IEEE Transactions on Neural Networks, 2003.
[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," in IEEE Transactions on Neural Networks, 2009.

Basic Concepts of a Graph

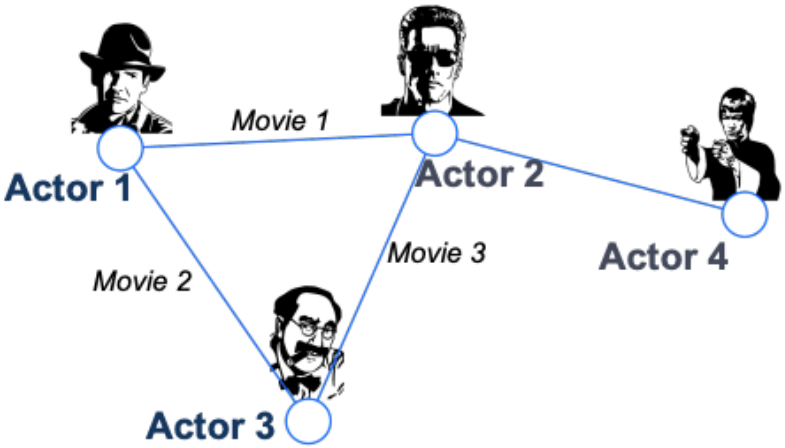
Part1

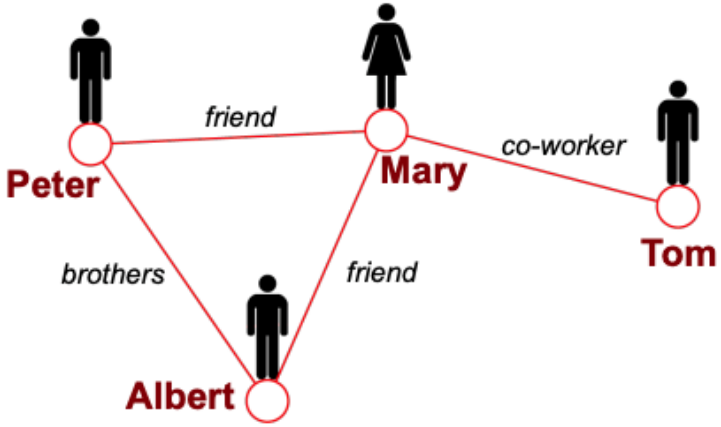
Entities of a Graph

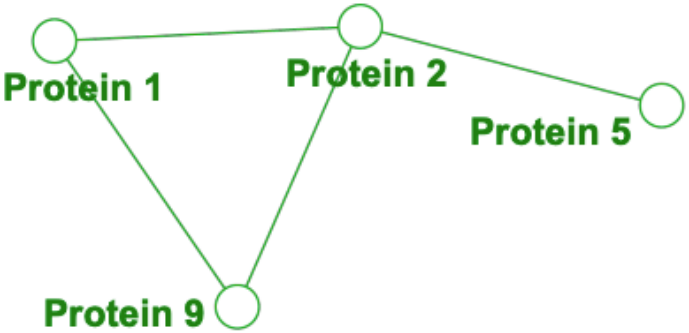


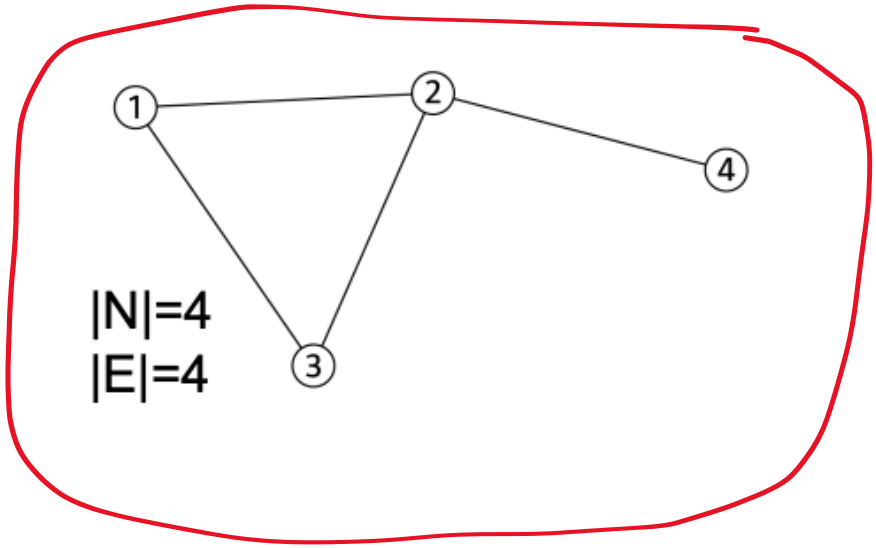
- **Objects:** nodes, vertices N
- **Interactions:** links, edges E
- **System:** network, graph $G(N,E)$

Graphs: A Common Language





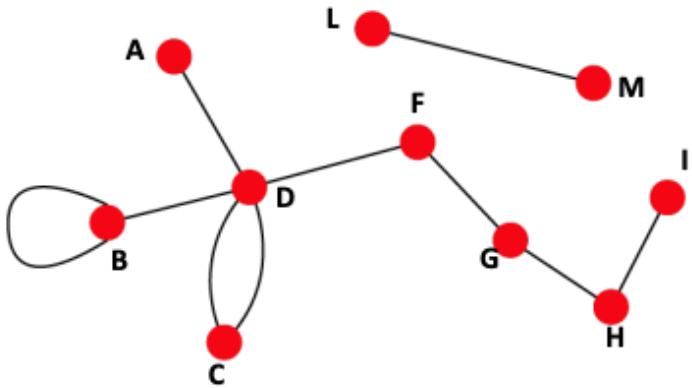




Directed vs. Undirected Graphs

Undirected

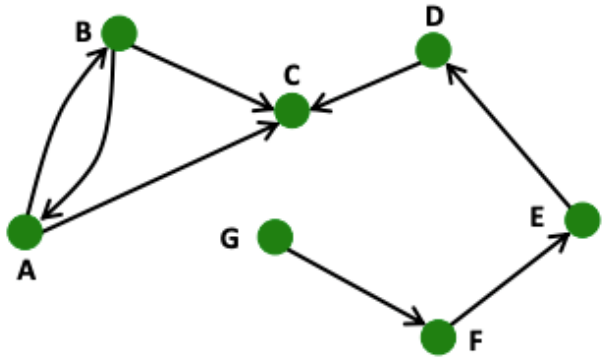
- **Links:** undirected (symmetrical, reciprocal)



- **Examples:**
  - Collaborations
  - Friendship on Facebook

Directed

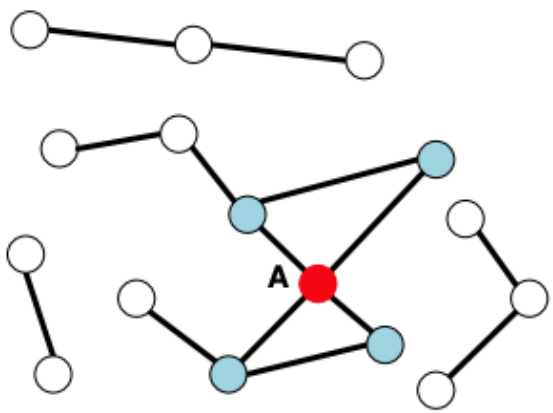
- **Links:** directed (arcs)



- **Examples:**
  - Phone calls
  - Following on Twitter

Node Degrees

Undirected

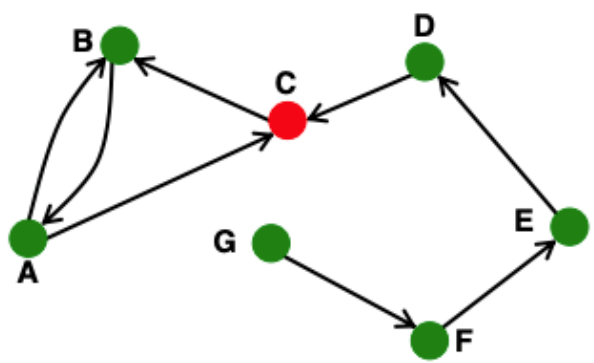


Node degree, k_i: the number of edges adjacent to node ***i***

$$k_A = 4$$

Avg. degree: $\bar{k} = \langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$

Directed



In directed networks we define an **in-degree** and **out-degree.**

The (total) degree of a node is the sum of in- and out-degrees.

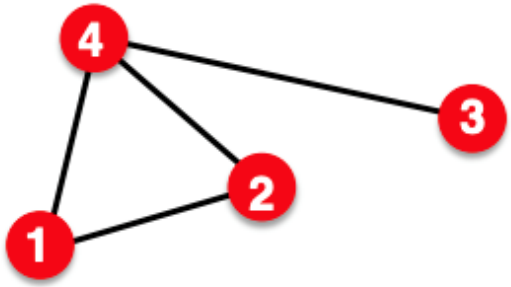
$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

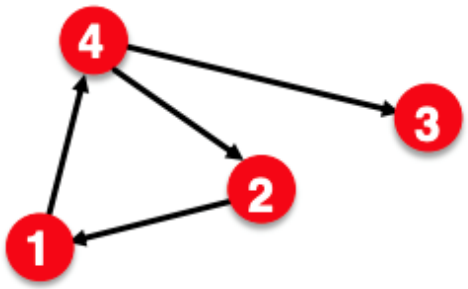
$$\bar{k} = \frac{E}{N} \qquad \overline{k^{in}} = \overline{k^{out}}$$

Source: Node with $k^{in} = 0$

Sink: Node with $k^{out} = 0$

Representing Graphs: Adjacency Matrix





$\boldsymbol{A_{ij} = 1}$ if there is a link from node ***i*** to node ***j***

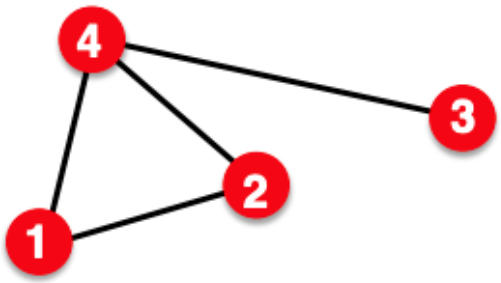
$\boldsymbol{A_{ij} = 0}$ otherwise

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

Note that for a directed graph (right) the matrix is not symmetric.

Adjacency Matrix: Sparse

Undirected



$$A_{ij} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

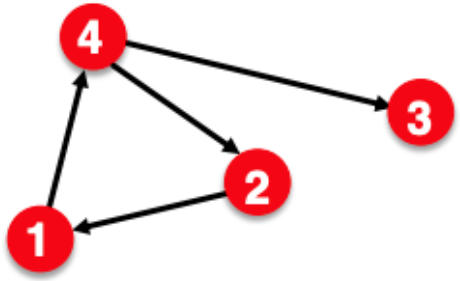
$$A_{ij} = A_{ji}$$
$$A_{ii} = 0$$

$$k_i = \sum_{j=1}^{N} A_{ij}$$

$$k_j = \sum_{i=1}^{N} A_{ij}$$

$$L = \frac{1}{2}\sum_{i-1}^{N} k_i = \frac{1}{2}\sum_{ij}^{N} A_{ij}$$

Directed



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

$$A_{ij} \neq A_{ji}$$
$$A_{ii} = 0$$

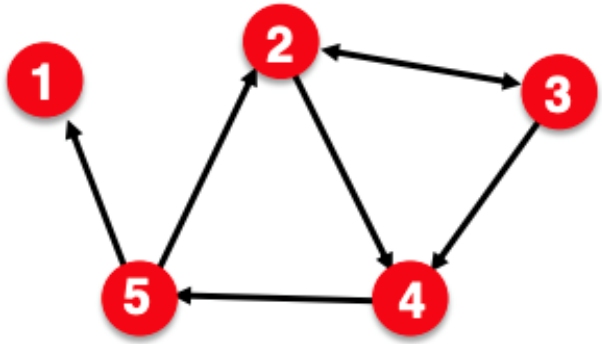
$$k_i^{out} = \sum_{j=1}^{N} A_{ij}$$

$$k_j^{in} = \sum_{i=1}^{N} A_{ij}$$

$$L = \sum_{i=1}^{N} k_i^{in} = \sum_{j=1}^{N} k_j^{out} = \sum_{i,j}^{N} A_{ij}$$

Representing Graphs: Edge list

- **Represent graph as a list of edges:**
  - (2, 3)
  - (2, 4)
  - (3, 2)
  - (3, 4)
  - (4, 5)
  - (5, 2)
  - (5, 1)




```
>>> import networkx as nx
>>> G = nx.Graph() >>> DG = nx.DiGraph()
>>> G.add_edges_from([(1, 2), (1, 3)])
```

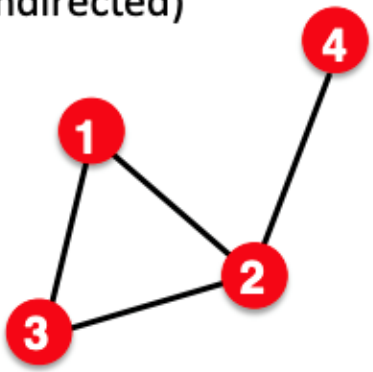
Create a graph via network:
https://networkx.org/documentation/stable/tutorial.html

Node and Edge Attributes

- Possible options:
  - Weight (e.g., frequency of communication)
  - Ranking (best friend, second best friend…)
  - Type (friend, relative, co-worker)
  - Sign: Friend vs. Foe, Trust vs. Distrust
  - Properties depending on the structure of the rest
  - of the graph: Number of common friends

Weighted vs. Unweighted Graphs

- **Unweighted**
  (undirected)



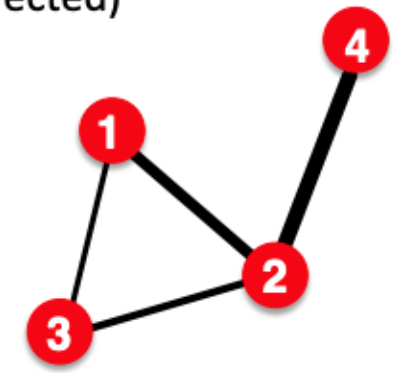
$$A_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1}^{N} A_{ij} \qquad \bar{k} = \frac{2E}{N}$$

Examples: Friendship, Hyperlink

- **Weighted**
  (undirected)



$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

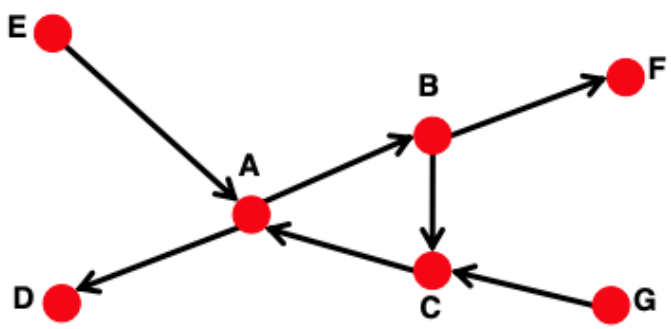
$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

$$E = \frac{1}{2}\sum_{i,j=1}^{N} nonzero(A_{ij}) \qquad \bar{k} = \frac{2E}{N}$$

Examples: Collaboration, Internet, Roads

Connectivity of Directed Graphs

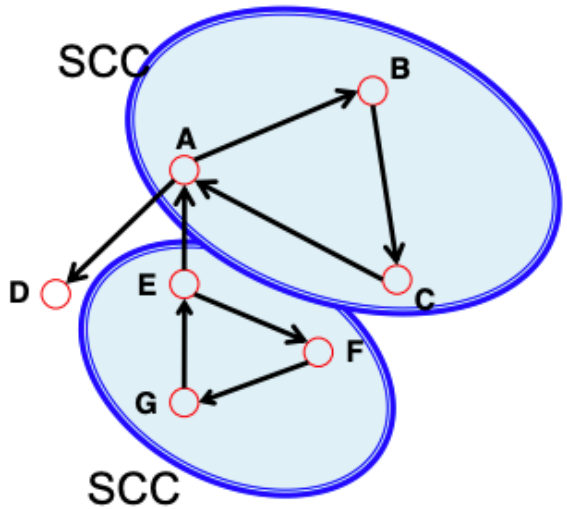
- **Strongly connected directed graph**
  - has a path from each node to every other node and vice versa (e.g., A-B path and B-A path)
- **Weakly connected directed graph**
  - is connected if we disregard the edge directions

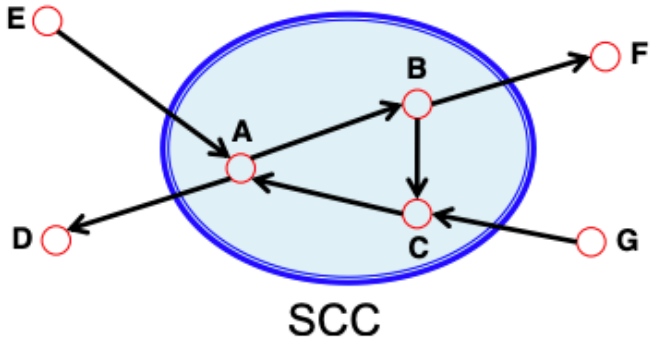


Graph on the left is connected but not strongly connected (e.g., there is no way to get from F to G by following the edge directions).

Connectivity of Directed Graphs

- **Strongly connected components (SCCs)** can be identified, but not every node is part of a nontrivial strongly connected component.



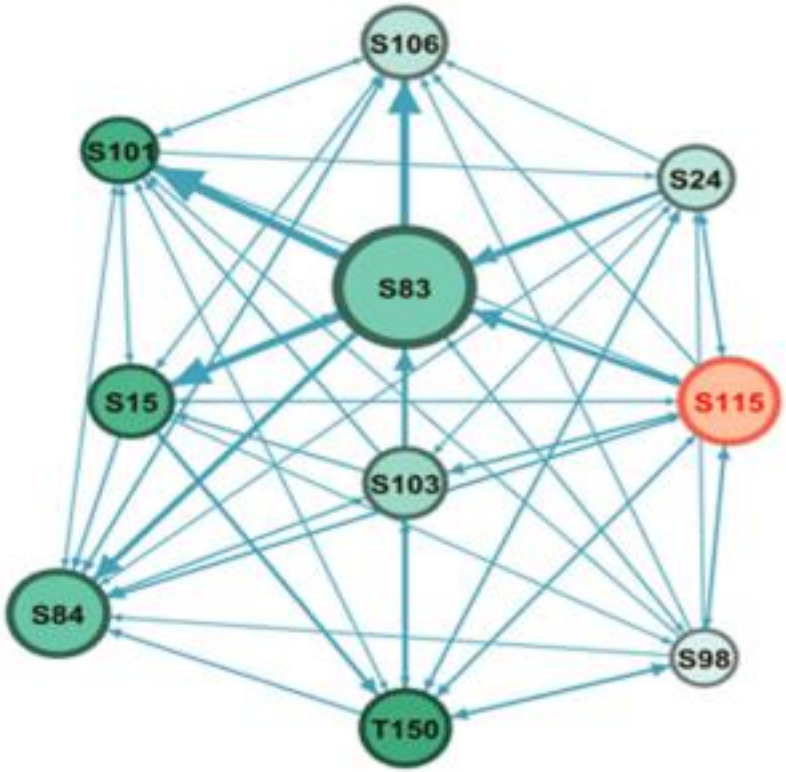


In-component: nodes that can reach the SCC,

Out-component: nodes that can be reached from the SCC.

Activity level

Very active



Parameter	Value
Indegree	27
Outdegree	50
Degree	77
Closeness	0.83
Betweenness	28.8

Compared to group

Indegree	30.18
outdegree	30.18
Degree	60.36
Closeness centrality	0.76
Betweenness	3.45

Basic Measures of a Graph

A. Grover, J. Leskovec. KDD 2016.

Part2

Measures of Centrality

- Nodes that are central to the graph have a significant impact on the properties of the network, such as its *density*, *pairwise shortest path distances*, *connectivity*, and *clustering behavior*.
- Many of these nodes are hub nodes, with high degrees that are a natural result of the dynamical processes of large network generation.
- To an undirected graph, measures of centrality:
  - Degree centrality
  - Closeness centrality
  - Betweenness centrality
  - Rank centrality

Measures of Prestige

- A related notion of centrality is ***prestige***, which is relevant for directed graphs.
  - For example, on Twitter, an actor with a larger number of followers has greater prestige. On the other hand, following a large number of individuals does not bring any prestige.
  - PageRank is a measure of prestige.
- It is possible to generalize centrality measures to directed graphs.
  - Degree prestige
  - Proximity prestige
  - Rank prestige

Degree Centrality

- The degree centrality $C_D(i)$ of a node i of an undirected graph is equal to the degree of the node, divided by the maximum possible degree of the nodes: $|V| - 1$.
- If $Degree(i)$ is the degree of node i, then the degree centrality $C_D(i)$ of a node i is defined as follows:

$$C_D(i) = \frac{Degree(i)}{n - 1}$$

Degree Prestige

- Degree prestige is defined for directed graphs only, and uses the **indegree** of the node, rather than its degree.
  - Only a high indegree contributes to the prestige because the indegree of a node can be viewed as a vote for the popularity of the node.
- The degree prestige $P_D(i)$ of a node i is defined as follows:

$$P_D(i) = \frac{Indegree(i)}{n-1}$$

Gregariousness

- The notion of centrality can also be extended to the node **outdegree**.
- This is defined as the gregariousness of a node.
- The gregariousness $G_D(i)$ of a node i is defined as follows:

$$G_D(i) = \frac{Outdegree(i)}{n-1}$$

Closeness Centrality

- The notion of closeness centrality is meaningfully defined with respect to *undirected* and *connected* graphs.
- The average shortest path distance, starting from node i, is denoted by $AvDis(i)$ and is defined in terms of the pairwise shortest path distances $Dist(i,j)$, between nodes i and j as follows:

$$AvDist(i) = \frac{\sum_{j=1}^{n} Dist(i,j)}{n-1}$$

- The closeness centrality is simply the inverse of the average distance of other nodes to node i.

$$C_C(i) = 1/AvDist(i)$$

Proximity Prestige

- Proximity prestige can be used to measure prestige in directed graphs.
- To compute the proximity prestige of node i, the shortest path distance to node i from all other nodes is computed.
- The first step is to determine the set of nodes $Influence(i)$ that can reach node i with a directed path.
  - For example, the Twitter network, $Influence(i)$ corresponds to all recursively defined followers of node i.
- The value of $AvDist(i)$ can now be computed only with respect to the influence set $Influence(i)$.

$$AvDist(i) = \frac{\sum_{j \in Influence(i)} Dist(j, i)}{|Influence(i)|}$$

- Note that distances are computed from node j to i, and not vice versa.

Proximity Prestige

- Nodes that have less influence should be penalized.
- While its low average distance to its influence set suggests high prestige, its small influence set suggest that it cannot be considered a node with high prestige.
- To account for this, a multiplicated penalty factor is included in the measure that corresponds to the fractional size of the influence set of node i.

$$InfluenceFraction(i) = \frac{|Influence(i)|}{n-1}$$

- Then, the proximity prestige $P_P(i)$ is defined as follows:

$$P_P(i) = \frac{InfluenceFraction(i)}{AvDist(i)}$$

Betweenness Centrality

- While closeness centrality is based on notions of distances, it does not account for the criticality of the node in terms of the number of shortest paths that pass through it.
- To have the greatest control of the flow of information between other actors in a social network, we measure the betweenness centrality.

Betweenness Centrality

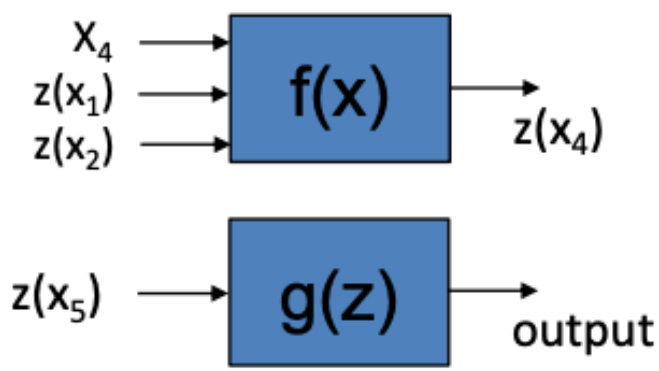
- Let q_{jk} denotes the number of shortest paths between nodes j and k.
- Let $q_{jk}(i)$ be the number of these pairs that pass through node i.
- The fraction of pairs $q_{jk}(i)$ that pass through node i is given by $f_{jk}(i) = q_{jk}(i)/q_{jk}$.
  - $f_{jk}(i)$ is a fraction that indicates the level of control that node i has over node j and k in terms of regulating the flow of information between them.
- The betweenness centrality $C_B(i)$ is the average value of this fraction over all $\binom{n}{2}$ pairs of nodes.

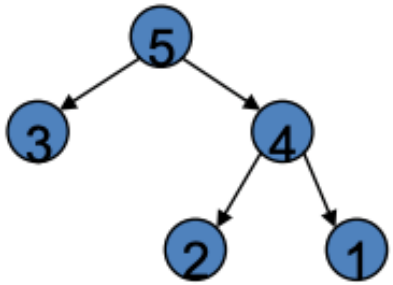
$$C_B(i) = \frac{\sum_{j<k} f_{jk}(i)}{\binom{n}{2}}$$

- The betweenness centrality also lies between 0 and 1, with higher values indicating better betweenness. Unlike closeness centrality, betweenness centrality can be defined for disconnected graphs as well.

GNNs: Basic concept

- Recurrent Neural Networks (RNNs) assume that each input has at most one dependency on another input.
  - Encoded by using one memory state.
- The nodes in a binary tree have at most two dependencies.
  - We can extend Hopfield's concept
  - Use two memory states!

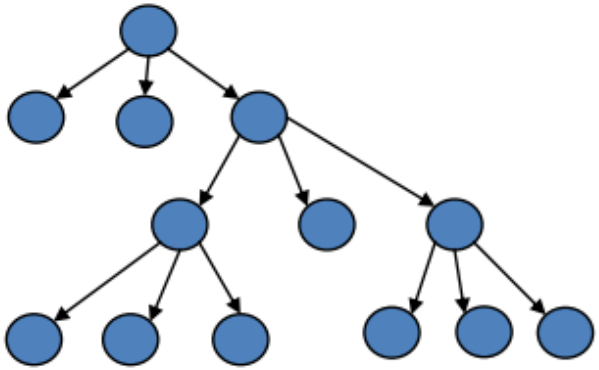


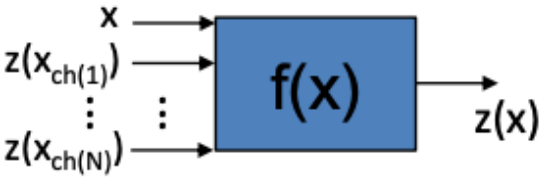


with z(0)=0

GNNs: Basic concept

- The concept can be extended further:
  - N-arry trees
  - Process any acyclic data structure whose (maximum) indegree is known and fixed.
- Back-propagation through Structure (BPTS)
  - Number of memory states = indegree

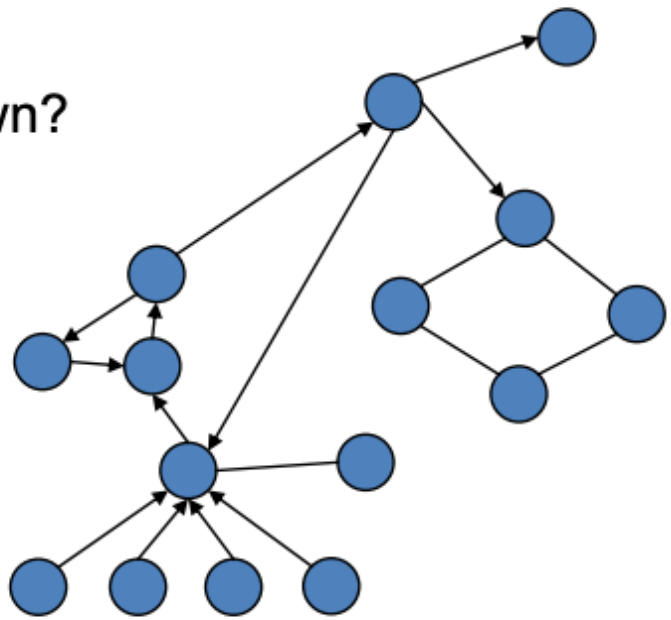




Küchler, Andreas (1996). "Learning Task-Dependent Distributed Representations by Backpropagation Through Structure". Proceedings of International Conference on Neural Networks.

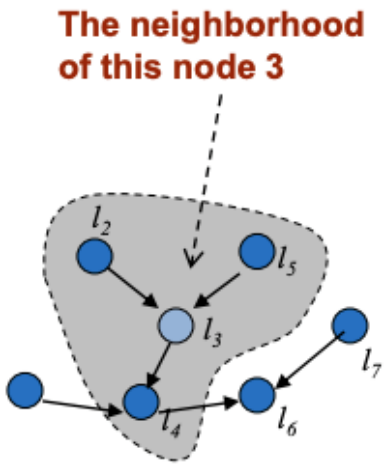
GNNs: The main idea

- But what to do if:
  - the indegree is not known?
  - there are cycles?
  - undirected links?



GNNs: The main idea

- The decision on a node n depends on its neighborhood
  - Labels of neighbor nodes
  - State of neighbor nodes
  - The edges of node n



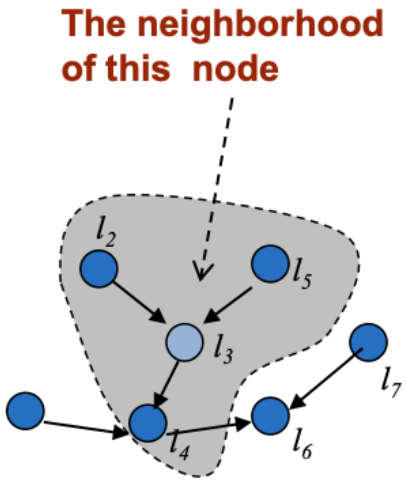
GNN: The main idea

GNN

- GNNs model each node and its neighborhood

As before

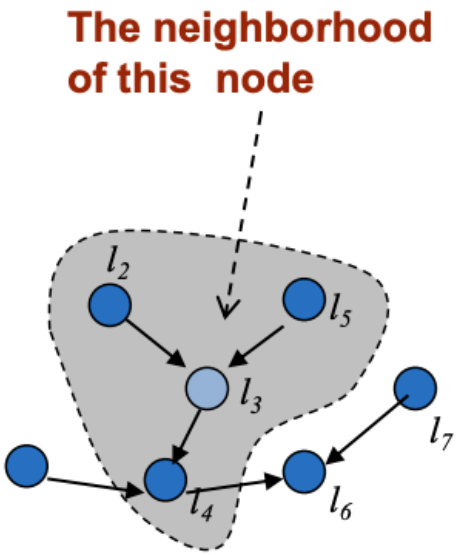
- The information useful to take a decision about a node must be stored somewhere
  - Let us use an internal state z for each node
- A decision on a node depends recursively on neighbor nodes!
  - Use a recursive mechanism
- How to carry out computations?
  - Let us use standard MLPs for this.



GNNs: Computing the states

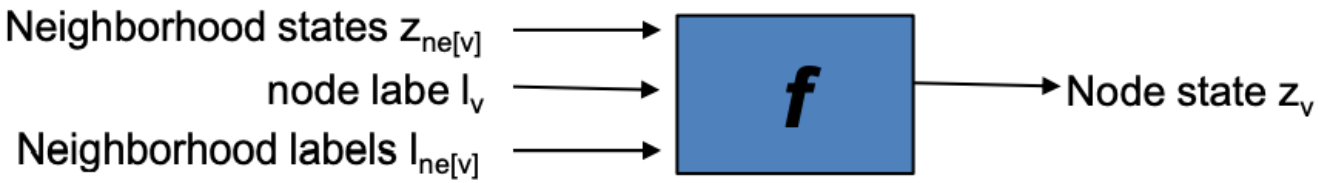
Let us implement this

- ...use an internal state z_v, for each node v
- ... use neighborhood information
- ...use standard MLPs



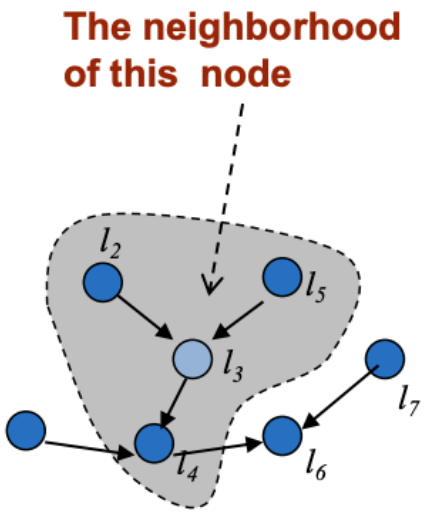
For each node v

- compute the state by a neural network f



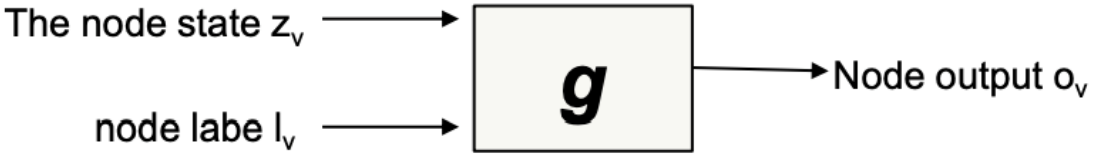
GNN: Computing the outputs

- The internal state z_v stores all the neighbor information about v



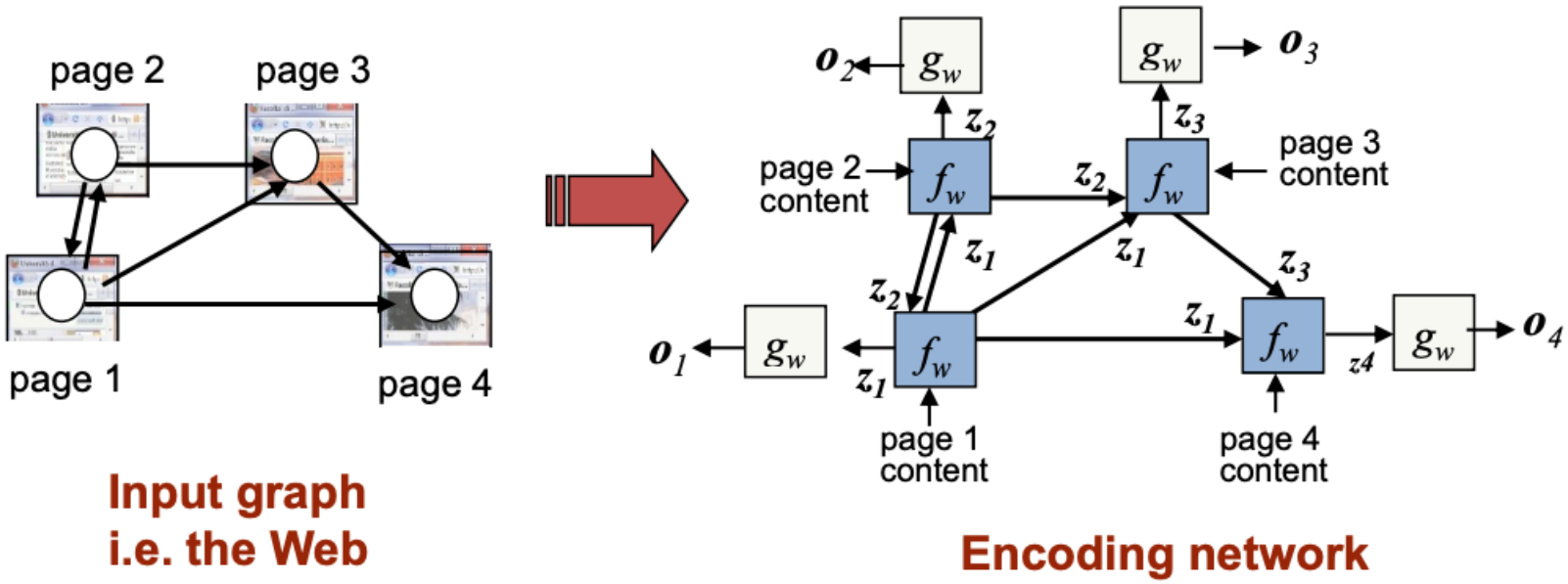
To compute an output for a node v

- Use a second MLP g



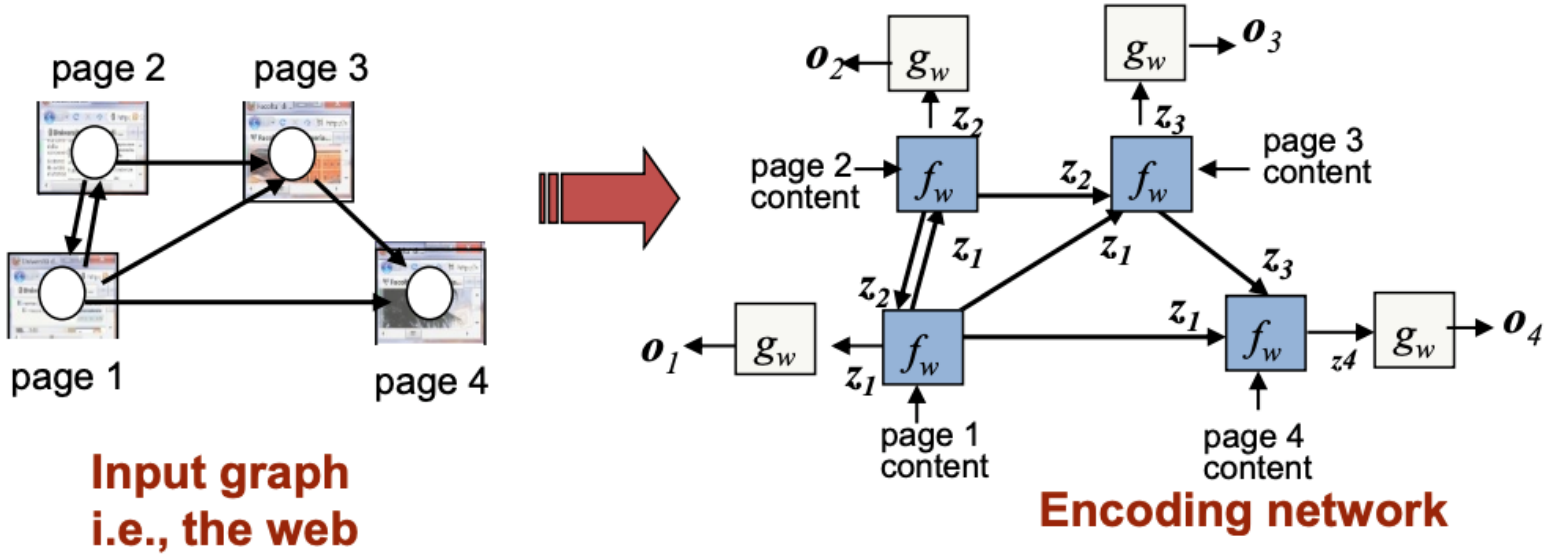
GNN: The encoding network

- Note that f and g are applied to each node in an input graph.
  - So, we have two MLPs for each node
- The collective of these networks is called encoding network.



The encoding network

- The encoding network is a big (recurrent) network.
  - Of the size of a graph.
- All the modules share the same parameters, the network has thus only few parameters.
  - There is only one instance of f and once instance of g.
  - The same f and g are applied to each node.



Training the GNN

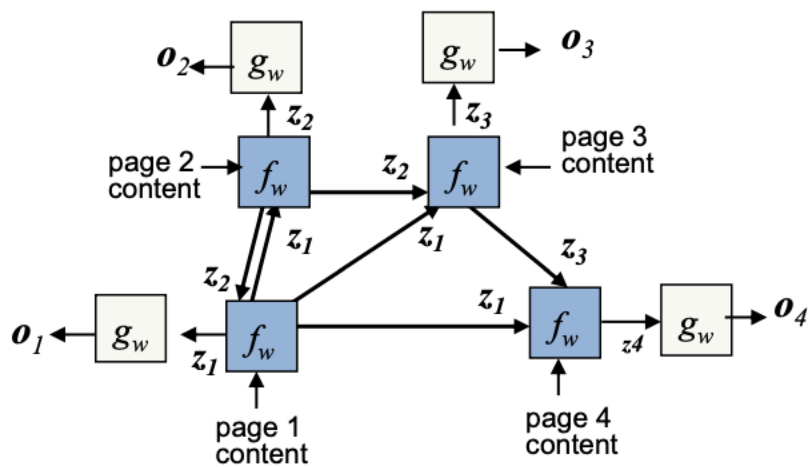
1. **Repeat**
2. Take a graph from the training set and construct the encoding network.
3. *Compute the network outputs* (forward phase).
4. Compute the error.
5. *Compute the gradient w.r.t. the error* (backward phase).
6. Update the weights.
7. **Until** error converged or error reaches a given threshold

Forward phase: computing GNN outputs

The enconding network is cyclic … so how are the outputs computed?

1. Set initial states z(t=0)=0
2. **Repeat**
3. activate units *f* to compute new states z(t+1) for each of the nodes in the graph
4. **Until** z(t) do not change any more
5. Activate g to compute output for each node.

This computes the stable state.



Forward phase: computing GNN outputs

Given that the enconding network is cyclic …

- Does the forward phase always converge?
- Does the forward phase always converge for any initial network condition?

Yes!

- GNNs adopt a mechanism which force the encoding network to be a contracting system.

References:

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "The Graph Neural Network Model," in IEEE Transactions on Neural Networks, 2008.

[2] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, "Computational capabilities of Graph Neural Networks," in IEEE Transactions on Neural Networks, 2008.

Backward phase: Computing GNN gradient

Based on unfolding of the encoding network

- Use the standard method for computing gradient in recurrent networks based on an error function.
- The encoding network is unfolded through time.
  - The result is a feedforward network equivalent to the encoding network.
- The gradient is computed on the unfolding using a backpropagation algorithm.

Properties of GNNs

- Pros:
  - The GNN is an universal approximator.
  - Very little or no pre-processing of data required.
  - Can be used to project graphs to vector space
    - Via the states.
  - Most generic type of NN in existence.
    - Processes vectors, trees, graphs.
- Cons:
  - Currently only available in Matlab and as a Tensorflow implementation:

    http://www.dii.unisi.it/~franco/Research/GNN.php
  - Long term dependency problem.
  - Black box model.

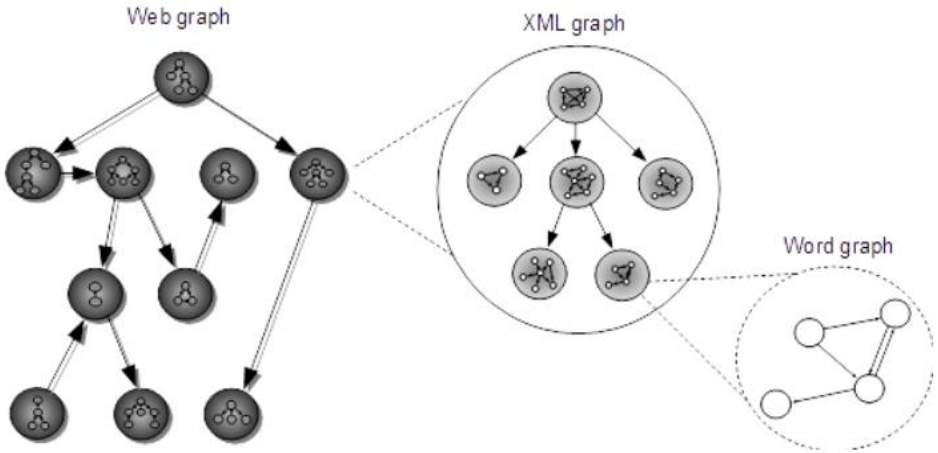
GNN applications

GNNs have been applied on

- Bioinformatics
- Language understanding
- Traffic forescasting
- Sentence extraction
- Web page ranking
- Object detection in images
- Web spam recognition
- Web document classification
- …

GNN variations

- Ensemble GNNs
  - Multiple GNNs (i.e. stacked GNNs)
- Unsupervised GNNs
  - SOM for structured data, probabilistic mapping graph SOM
  - For mapping, matching, projection, or clustering of graphs.
- GNN for Graph of Graphs
  - Process graphs whose nodes are labelled by other graphs.
  - GNN2
- Convolutional GNNs
  - For image, video analytics
- …



References:

[1] M.Hagenbuchner, A.Sperduti, AC.Tsoi, A self-organizing map for adaptive processing of structured data, IEEE Trans. on Neural Networks, 2003.

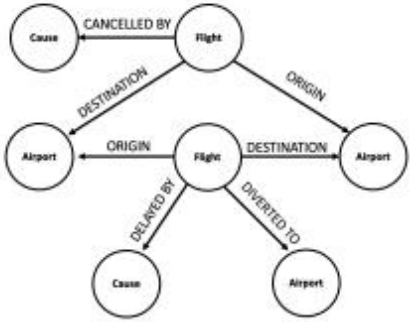
[2] S.J. Zhang, M. Hagenbuchner, F. Scarselli, A.C. Tsoi, Supervised encoding of graph-of-graphs for classification and regression problems, Springer, pp 449-461, 2009.

[3] M. Hagenbuchner, S. Zhang, A.C. Tsoi, A. Sperduti, Projection of undirected and non-positional graphs using self organizing maps, 2009.

Notes of caution

- Research in neural networks is a fast developing field.
- Some of the latest developments are yet to receive broader attention.
  - Leading to delays in deployment.
- Examples:
  - Modelling of graphs…
  - Explanatory systems…
  - Autonomous learning systems…
- Some solutions exists but may not yet be available as toolboxes for R, Python, MapReduce,…

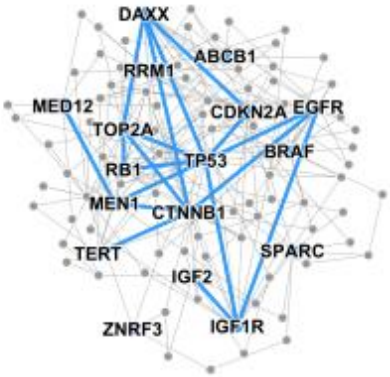
Many Types of Data are Graphs



Event Graphs

Image credit: SalientNetworks

Computer Networks



Disease Pathways

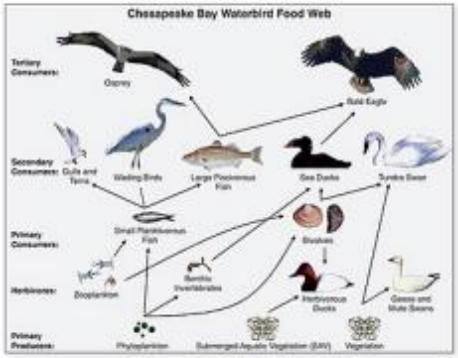


Image credit: Wikipedia

Food Webs

Image credit: Pinterest

Particle Networks

Image credit: visitlondon.com

Underground Networks

Many Types of Data are Graphs

Image credit: Medium

Social Networks

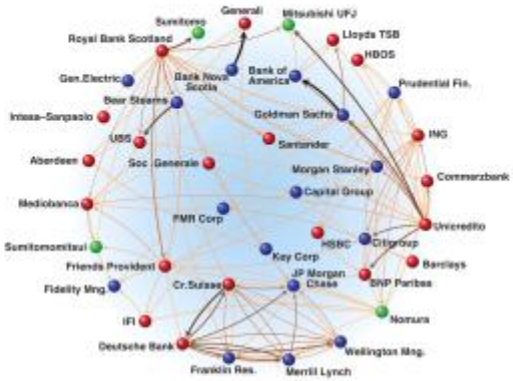


Image credit: Science

Economic Networks

Image credit: Lumen Learning

Communication Networks

Citation Networks

Image credit: Missoula Current News

Internet

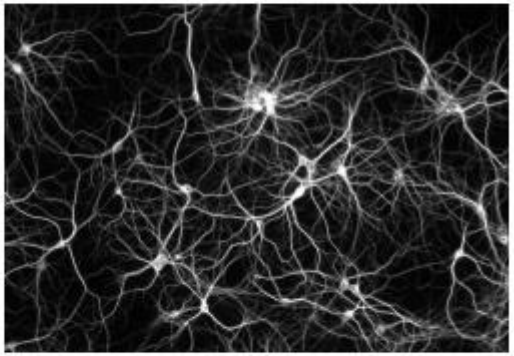


Image credit: The Conversation

Networks of Neurons

Many Types of Data are Graphs

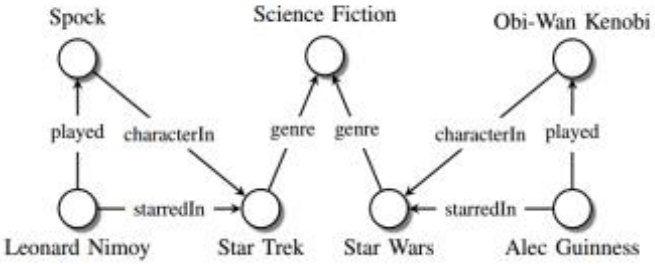


Image credit: Maximilian Nickel et al

Knowledge Graphs

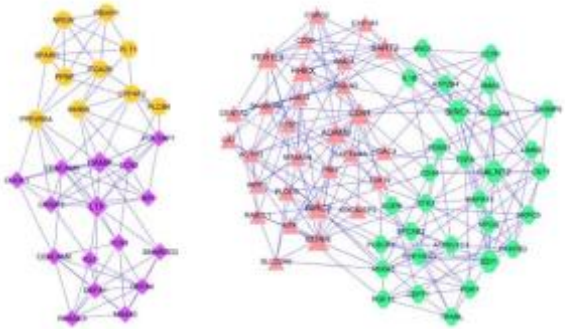


Image credit: ese.wustl.edu

Regulatory Networks

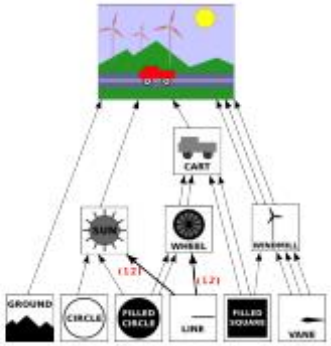


Image credit: math.hws.edu

Scene Graphs

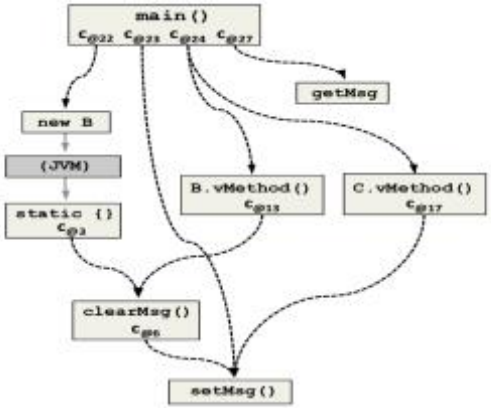


Image credit: ResearchGate

Code Graphs

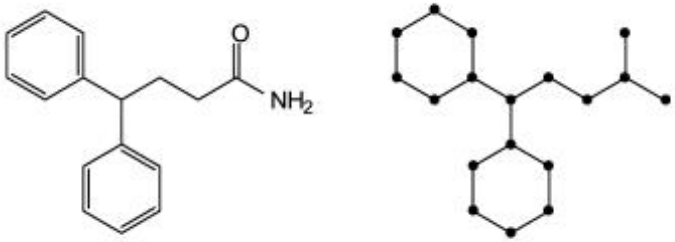


Image credit: MDPI

Molecules

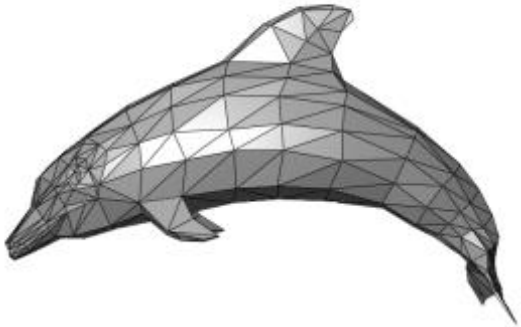


Image credit: Wikipedia

3D Shapes

Graphs and Relational Data

- Main Question:

How do we take advantage of relational structure for better prediction?

Graphs: Machine Learning

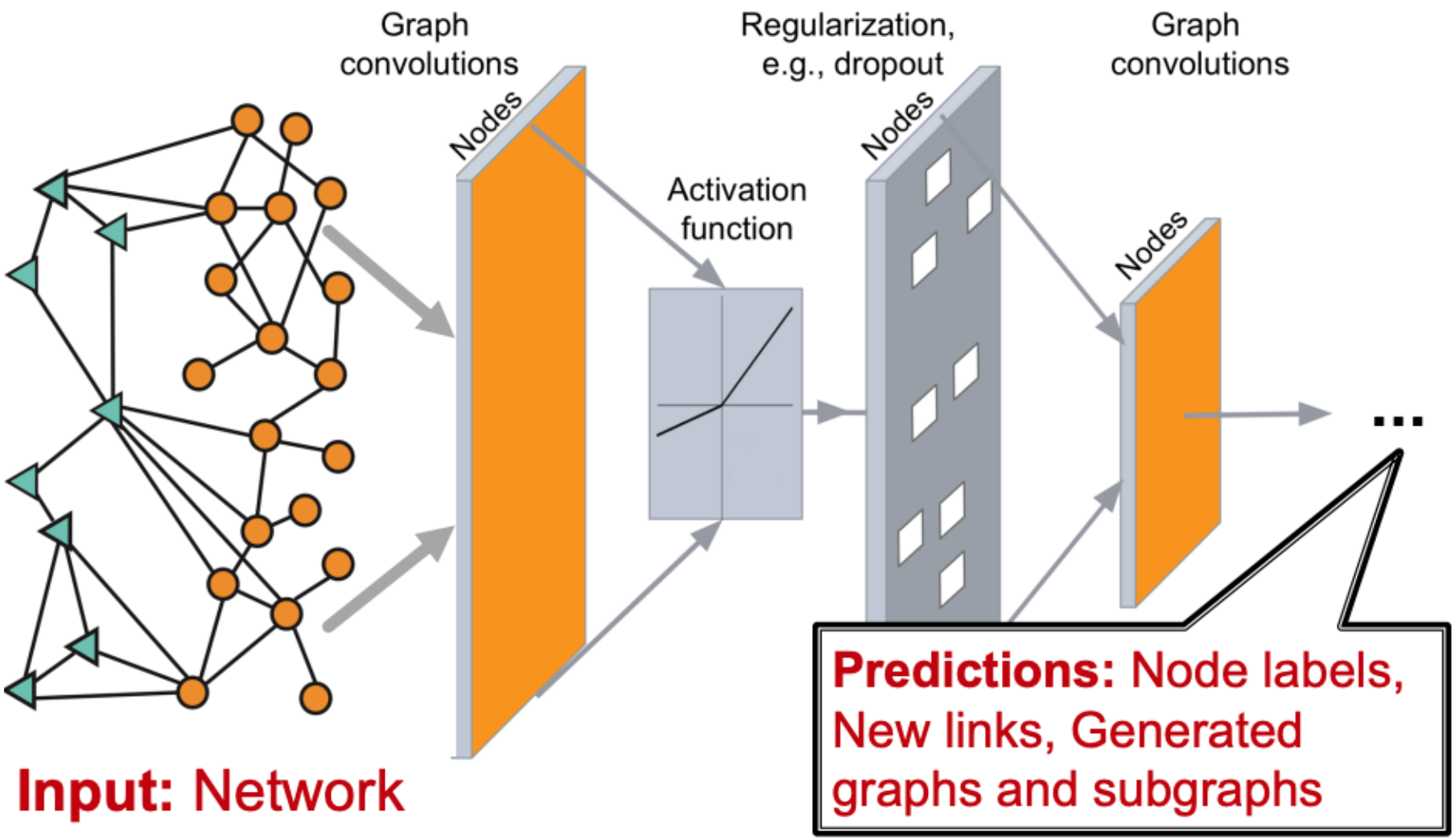
- Main Target:

  By explicitly modeling relationships we achieve better performance!

- How can we develop neural networks that are much more broadly applicable?

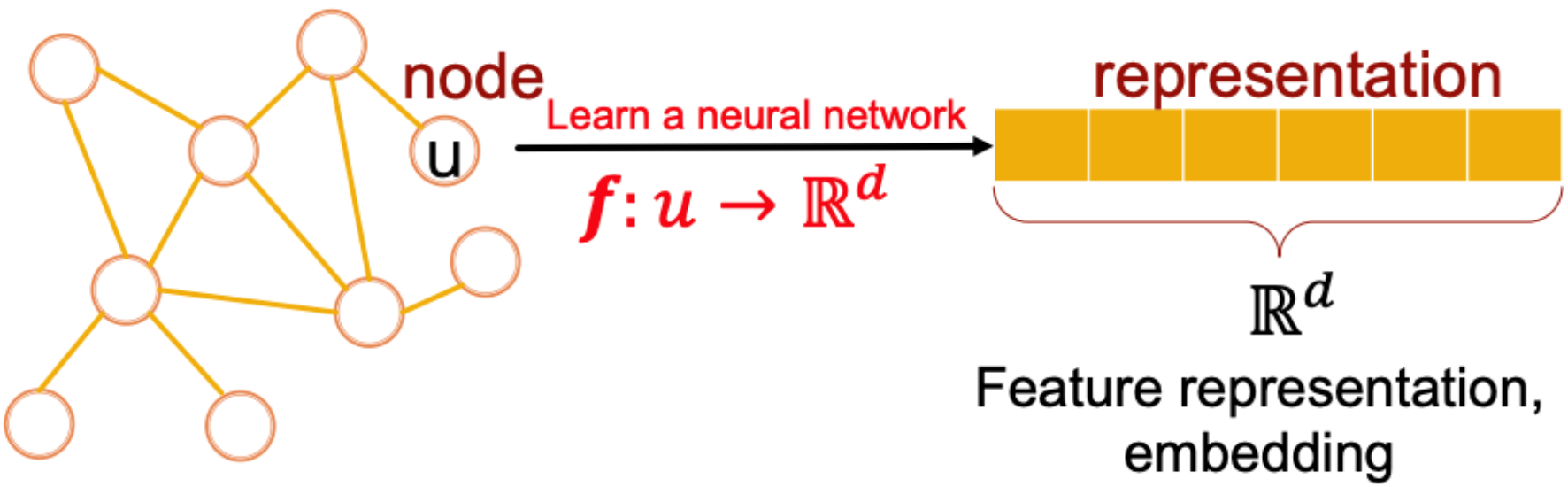
  Graphs are the new frontier of deep learning

Deep Learning in Graphs

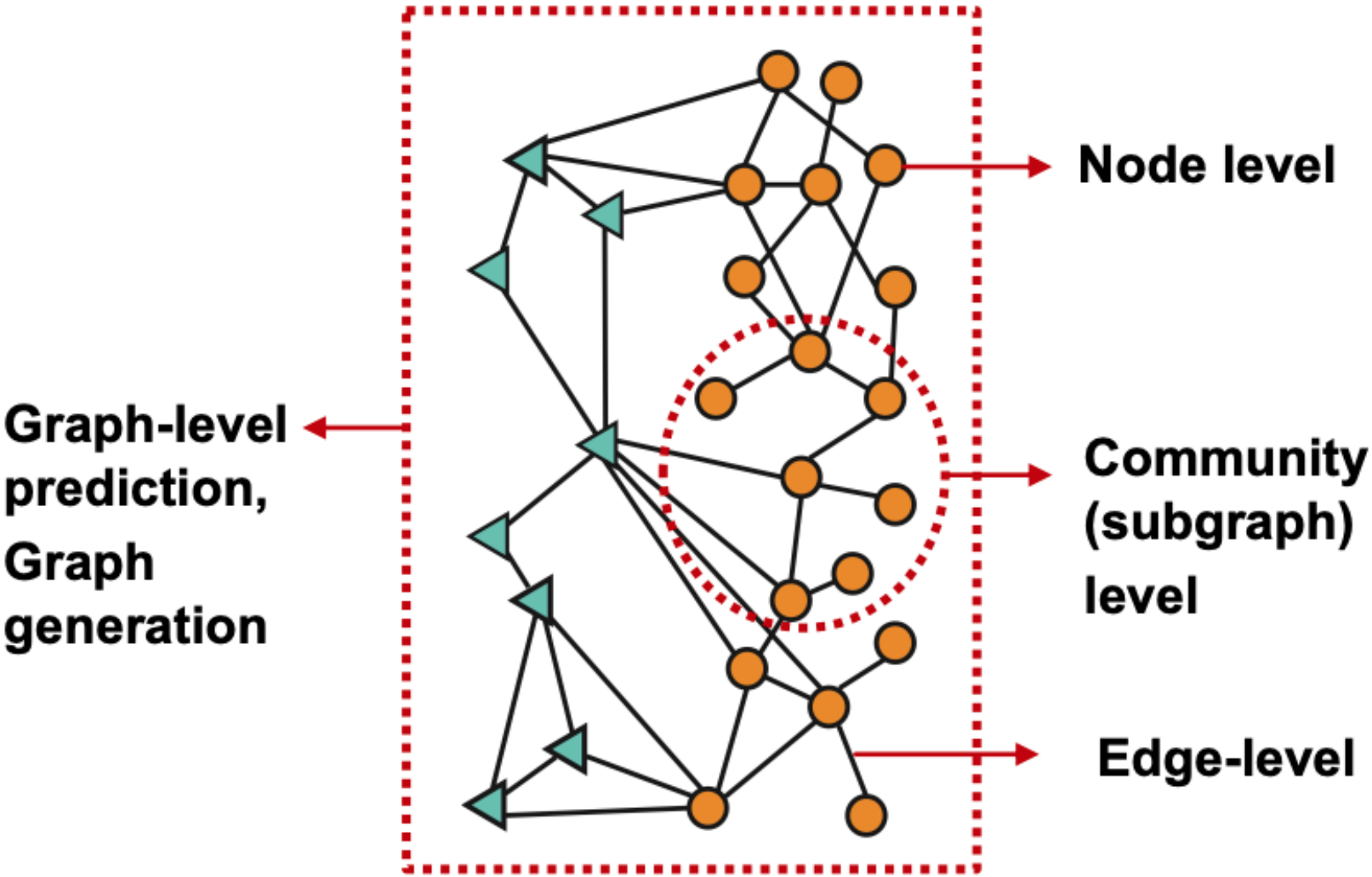


Important Idea

- Map nodes to *d*-dimensional embeddings such that similar nodes in the graph are embedded close together

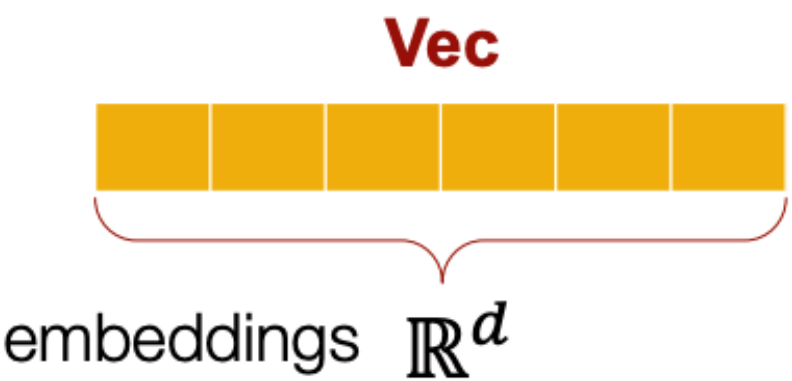


Different Types of Tasks



Why Embedding?

- Task: map nodes into an embedding space
  - Similarity of embeddings between nodes indicates their similarity in the network. For example:
    - Both nodes are close to each other (connected by an edge)
- Encode network information
- Potentially used for many downstream predictions



Tasks

- Node classification
- Link prediction
- Graph classification
- Anomalous node detection
- Clustering
- ....

Classic Graph ML Tasks

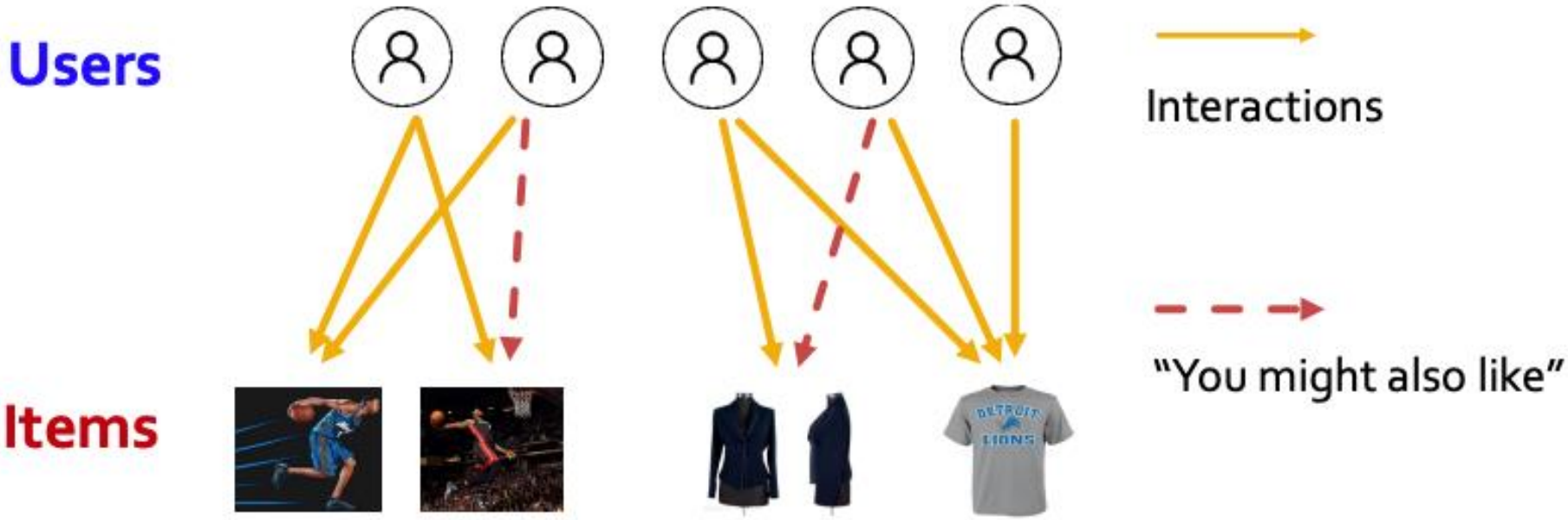
- **Node classification**: Predict a property of a node
  - **Example**: Categorize online users / items
- **Link prediction**: Predict whether there are missing links between two nodes
  - **Example**: Knowledge graph completion
- **Graph classification**: Categorize different graphs
  - **Example**: Molecule property prediction
- **Clustering**: Detect if nodes form a community
  - **Example**: Social circle detection
- **Other tasks**:
  - **Graph generation**: Drug discovery
  - **Graph evolution**: Physical simulation

Graph ML Application

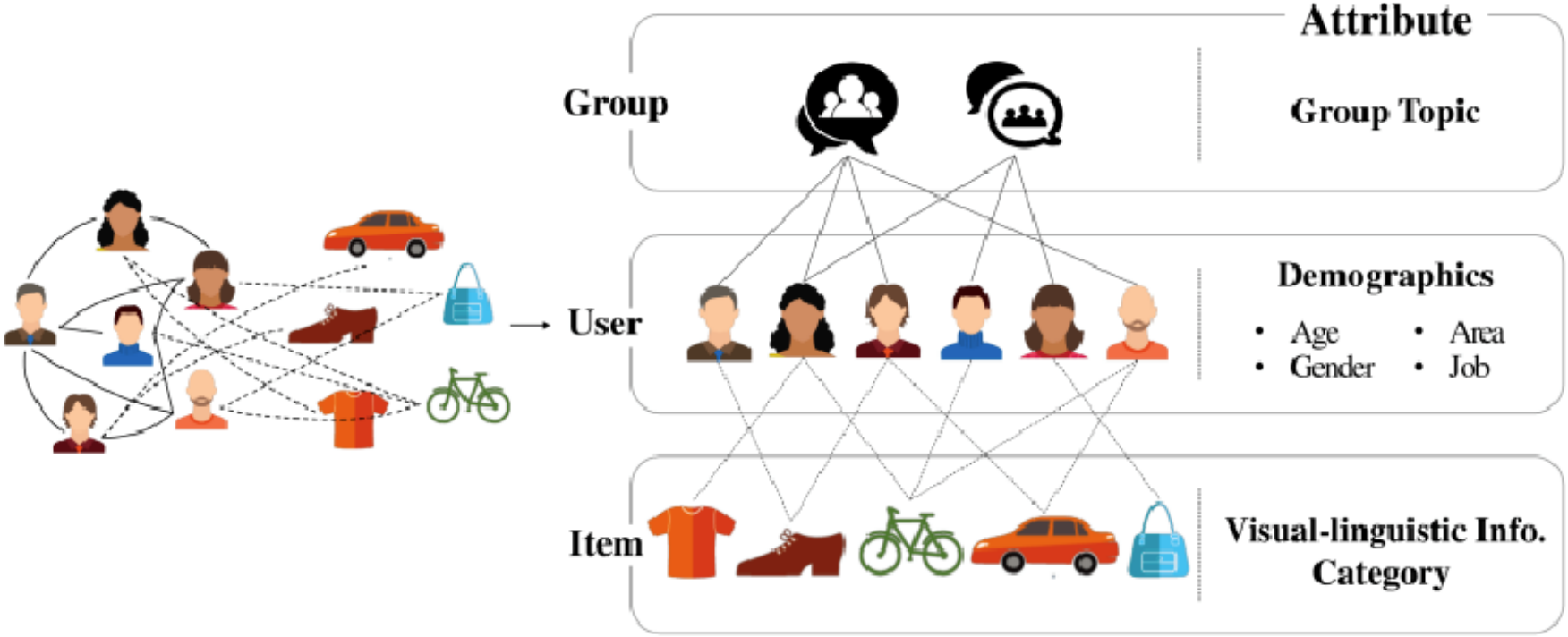
- Graph ML tasks leads to high-impact applications!
  - Most cases:
    - decide on graph ML task < – important to – ML itself

Node&Edge-level ML Tasks in Recommender Systems

- **Users interacts with items**
  - Watch movies, buy product, listen to music
  - Nodes: Users and items
  - Edges: User-item interactions
- **Goal: Recommend items users might like**

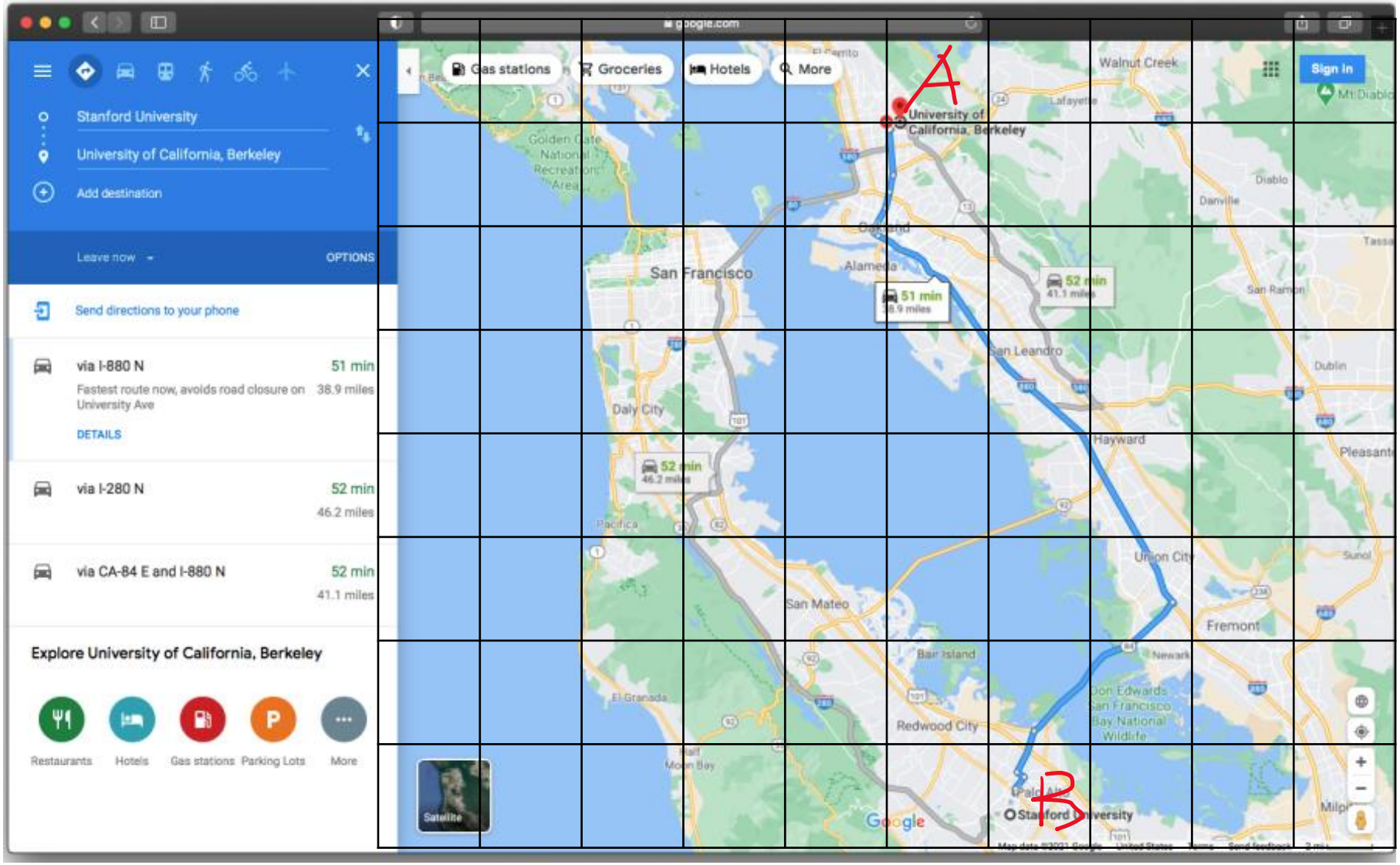


AS Graph



(a) Traditional social graph (b) Tripartite attributed multiplex heterogeneous graph

Subgraph-level ML Tasks



Road Network as a Graph

- **Nodes**: Road segments
- **Edges**: Connectivity between road segments

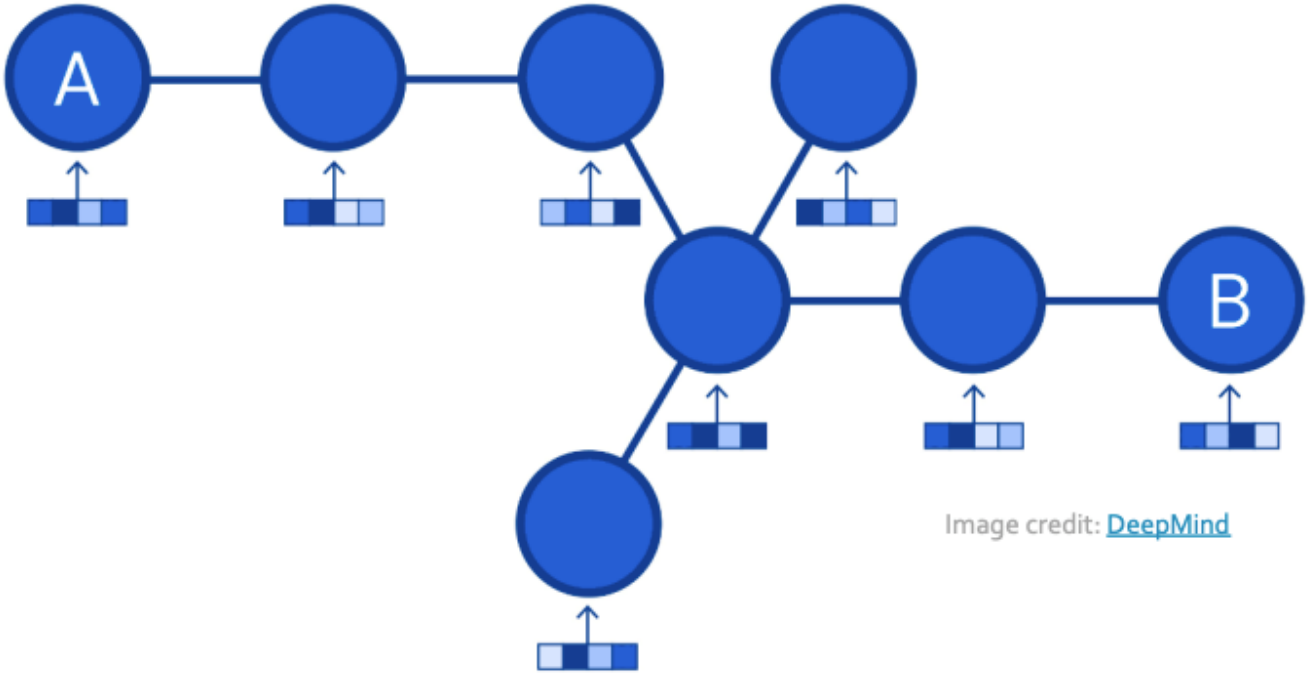


Image credit: DeepMind

Node2vec: Random Walk based Unsupervised Feature Learning

Part3

Overview of node2vec

- **Goal**: Embed nodes with similar network neighborhoods close in the feature space.
- We frame this goal as prediction-task independent maximum likelihood optimization problem.
- **Key observation**: Flexible notion of network neighborhood $N_S(u)$ of node u leads to rich features.
- Develop biased **2nd order random walk** procedure S to generate network neighborhood $N_S(u)$ of node u.

Unsupervised Feature Learning

- **Intuition**: Find embedding of nodes to d-dimensions that preserves similarity
- **Idea**: Learn node embedding such that nearby nodes are close together
- Given a node u, how do we define nearby nodes?
  - $N_S(u)$ … neighbourhood of u obtained by some strategy S

Feature learning as optimization

- Given $G = (V, E)$,
- Our goal is to learn a mapping $f: u \rightarrow \mathbb{R}^d$.
- Log-likelihood objective:
  $\max_f \sum_{u \in V} \log \Pr(N_S(u) | f(u))$
  – where $N_S(u)$ is neighborhood of node u.
- Given node u, we want to learn feature representations predictive of nodes in its neighborhood $N_S(u)$.

Feature learning as optimization

$$\max_f \sum_{u \in V} \log \Pr(N_S(u) | f(u))$$

- **Assumption**: Conditional likelihood factorizes over the set of neighbors.

$$\log \Pr(N_S(u) | f(u)) = \sum_{n_i \in N_S(u)} \log \Pr(f(n_i) | f(u))$$

- Softmax parametrization:

$$\Pr(f(n_i) | f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

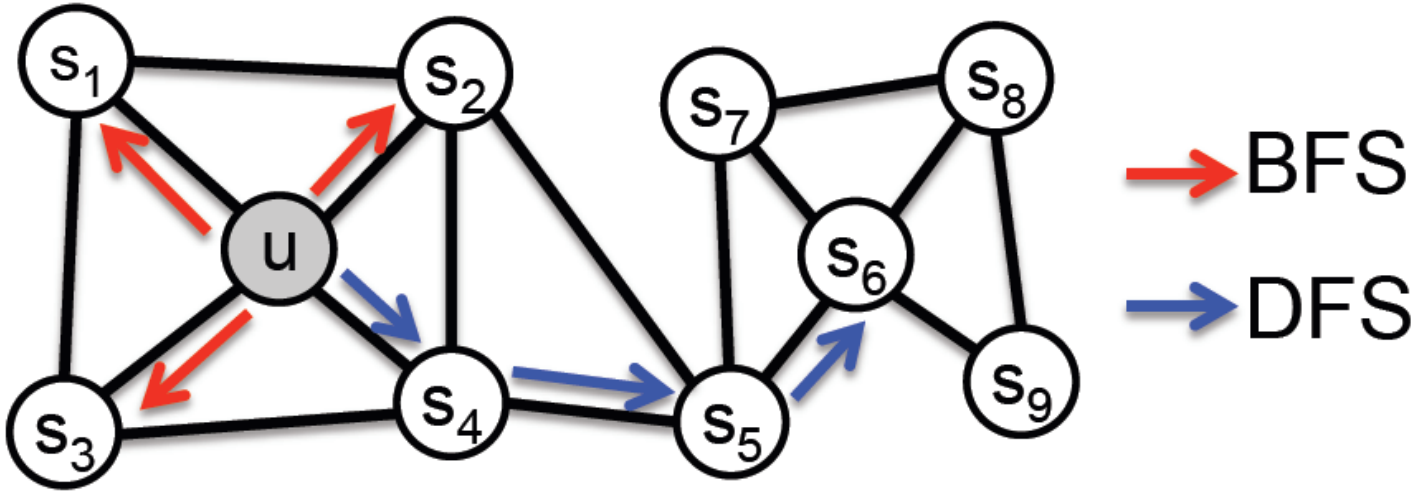
Negative Sampling

$$\max_f \sum_{u \in V} \sum_{n \in N_S(u)} \log \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

- Maximize the objective using Stochastic Gradient descent with **negative sampling**.
  - Computing the summation is expensive
  - **Idea**: Just sample a couple of "negative nodes"
  - This means at each iteration only embeddings of a few nodes will be updated at a time
  - Much faster training of embeddings

How to determine $N_S(u)$

- Two classic strategies to define a neighborhood $N_S(u)$ of a given node u:
  - Breadth-first search (BFS): Micro-view of neighbourhood
  - Depth-first search (DFS): Macro-view of neighbourhood

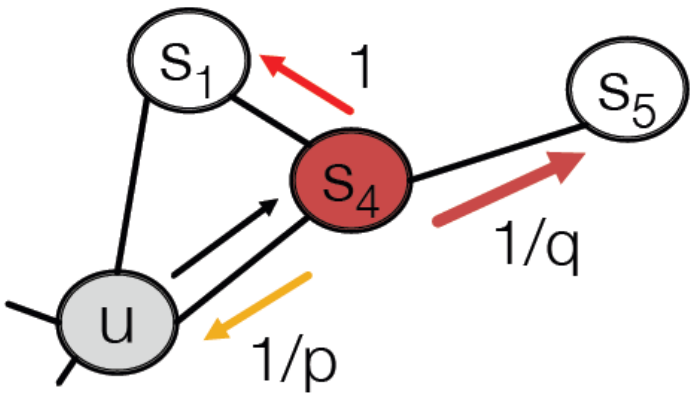


Interpolating BFS and DFS

- Biased random walk S that given a node u generates neighborhood $N_S(u)$
  - Two parameters:
    - Return parameter p:
      - Return back to the previous node
    - In-out parameter q:
      - Moving outwards (DFS) vs. inwards (BFS)

Biased Random Walks

$N_S(u)$: Biased 2nd-order random walks explore network neighborhoods:



$u \rightarrow s_4 \rightarrow ?$

u
s_1
s_5

- BFS-like: low value of p
- DFS-like: low value of q

p, q can learned in a semi-supervised way

Node2vec algorithm

- 1) Compute random walk probs.
- 2) Simulate r random walks of length l starting from each node u
- 3) Optimize the node2vec objective using Stochastic Gradient Descent

Linear-time complexity.

All 3 steps are individually parallelizable

node2vec: Discussion

- General-purpose feature learning in networks:
  - An explicit locality preserving objective for feature learning.
  - Biased random walks capture diversity of network patterns.
  - Scalable and robust algorithm with excellent empirical performance.
  - Future extensions would involve designing random walk strategies entailed to network with specific structure such as heterogeneous networks and signed networks.

Semi-supervised Classification with Graph Convolutional Networks (GCN)

Thomas N. Kipf, Max Welling 2017

Part4

Problem setting

- GNNs are NNs that operate on graph-structured data.
  - To capture correlation inside sample.
  - To capture correlation across samples.
- Undirected graph $G = (V, E)$, where $|V| = N$.
- Node $v_i \in V$ has a feature vector $X_i \in \mathbb{R}^d$.
- Edge $(v_i, v_j) \in E$ can be weighted or unweighted.
- Some nodes are labeled, and the task is to make predictions to those unlabeled.

GCN Layer

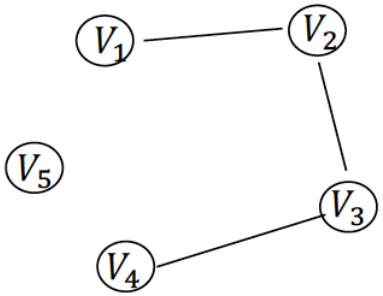
- GCN Layer:

$$\sigma(\hat{A}XW)$$

- σ: activation function
- X: input matrix $\in \mathbb{R}^{N\times d}$
- W: parameter matrix $\in \mathbb{R}^{d\times d'}$
- $\hat{A}$: Graph Laplacian $\hat{A} = \widetilde{D}^{-1/2}\,\tilde{A}\,\widetilde{D}^{-1/2}$
  - $\tilde{A} = A + I$
  - $\widetilde{D}$ is a diagonal matrix with $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Example of GCN Layer

$\hat{A} = \widetilde{D}^{-1/2}\tilde{A}\widetilde{D}^{-1/2}$, where $\tilde{A} = A + I$, $\widetilde{D}$ is a diagonal matrix with $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.



$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\widetilde{D}^{-1/2} = \begin{bmatrix} 0.71 & 0. & 0. & 0. & 0. \\ 0. & 0.58 & 0. & 0. & 0. \\ 0. & 0. & 0.58 & 0. & 0. \\ 0. & 0. & 0. & 0.71 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

$$\hat{A} = \begin{bmatrix} 0.5 & 0.41 & 0. & 0. & 0. \\ 0.41 & 0.33 & 0.33 & 0. & 0. \\ 0. & 0.33 & 0.33 & 0.41 & 0. \\ 0. & 0. & 0.41 & 0.5 & 0. \\ 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \Rightarrow \hat{A}X = \begin{bmatrix} 0.5x_1 + 0.41x_2 \\ 0.41x_1 + 0.33x_2 + 0.33x_3 \\ 0.33x_2 + 0.33x_3 + 0.41x_4 \\ 0.41x_4 + 0.5x_5 \\ x_5 \end{bmatrix}$$

Why GCNs work?

- In essence, GCN layer is an approximated spectral convolution.
- Two main streams of GNN architectures:
  - 1. Spectral-based.
    - spectral graph theory, eigendecomposition, matrix multiplication, sound theory but inefficient implementation
  - 2. Spatial-based.
    - Message passing among nodes, lack of theory but efficient implementation
- GCN is at the intersection of these two main streams!

GCN structure in the paper

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

- Why only tow layers?
  - Because deep GCNs do not perform well. Why?
    - An intuitive explanation is, graph convolution can be viewed as information exchange between neighbors, and if we keep doing this, all nodes' features will become more and more similar.
    - Graph Laplacian $\hat{A}$ has a smoothing effect. [1] proves that if we apply the graph Laplacian enough times, all nodes' features will converge to the same value. Hence the name *over-smoothing*.
    - Still an open question. Researchers try to explain it from various perspectives, such as Markov Process [2], Neural Tangent Kernel [3].

[1] Qimai Li. et al. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. 2018
[2] Kenta Oono. et al. GRAPH NEURAL NETWORKS EXPONENTIALLY LOSE EXPRESSIVE POWER FOR NODE CLASSIFICATION, 2020
[3] Wei Huang. et al. Towards Deepening Graph Neural Networks: A GNTK-based Optimization Perspective. 2022

GCN structure in the paper

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

- What if we need deep GCNs?
  - Just stack two GCN layers after a deep neural network.

Q & A

1. After reading the paper, someone claims: graph convolution is matrix multiplication, which takes $O(n^3)$, so it won't work for large-scale problems. Do you agree?
   - If the graph is large, a common practice is not to load the whole graph into the memory. Instead, we sample some subgraph in each iteration.
2. Is there any connection between graph neural networks and the self-attention mechanism?
   - with specific choice of hyper-parameters, Graphormer can represent popular GNN models including GCN.
3. GCN still relies on the assumption that connected nodes tend to be in the same class, though not as heavily as previous methods. What if we want to apply GCN when this assumption is violated?

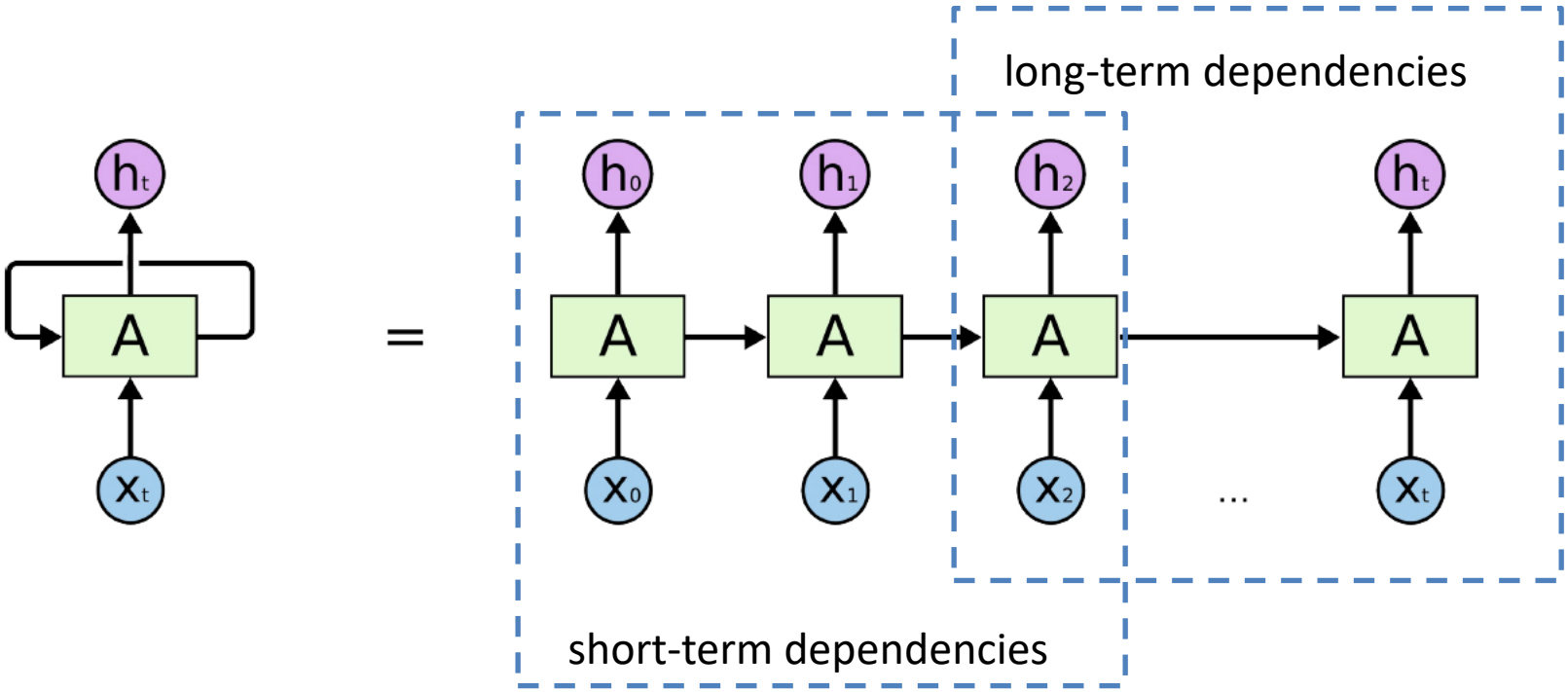
Long Short-Term Memory model (LSTM)

Part4

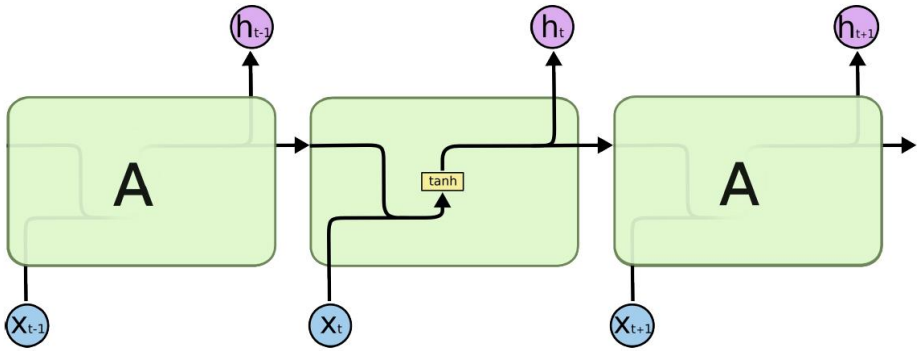
Overview

- LSTM is a class of recurrent neural network (RNN).
  - It is a class of neural networks tailored to deal with temporal data. The neurons of RNN have a cell state/memory, and input is processed according to this internal state, which is achieved with the help of loops with in the neural network. There are recurring module(s) of ‘tanh’ layers in RNNs that allow them to retain information. However, not for a long time, which is why we need LSTM models.
- Most popular model in time series domain

RNN

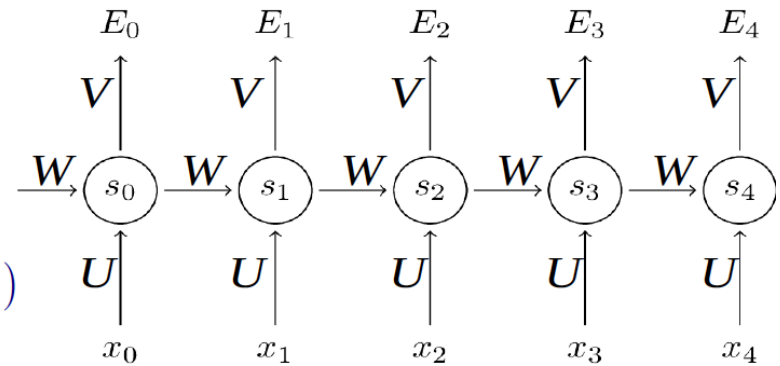

long-term dependencies
h_t
A
x_t
=
h_0
h_1
h_2
h_t
A
A
A
A
x_0
x_1
x_2
...
x_t
short-term dependencies

Standard RNN



- RNN forward pass

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$



$$E(y, \hat{y}) = -\sum_t E_t(y_t, \hat{y}_t)$$

- RNN Backpropagation through time

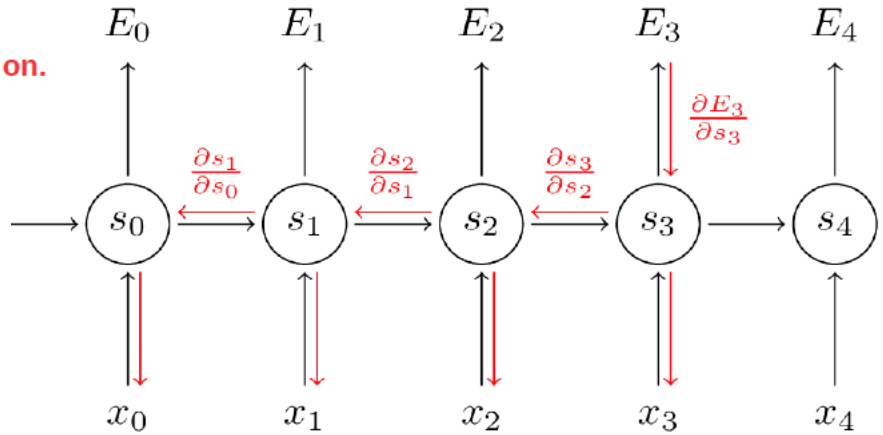
$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

S_3 depends on s_2, which depends on W and s_1, and so on.

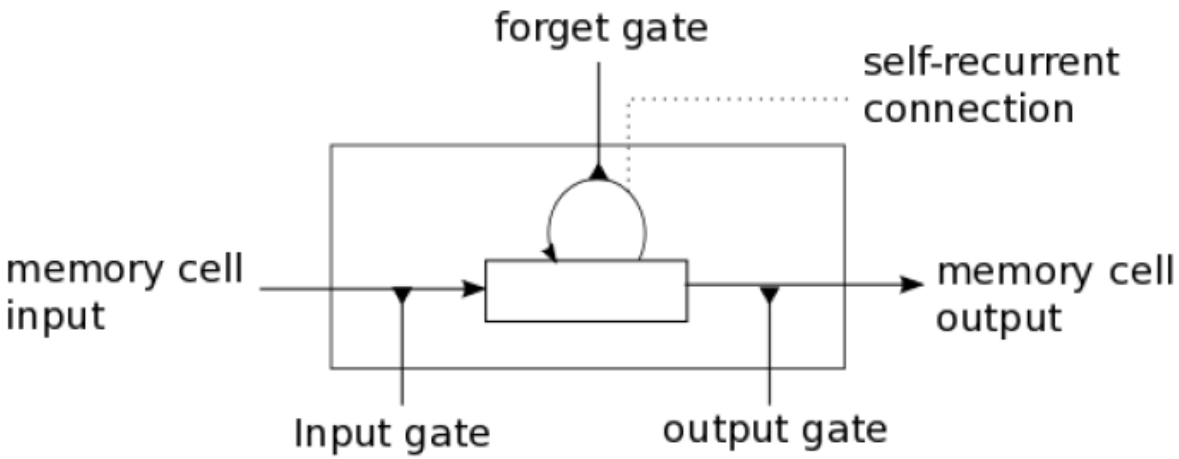
$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

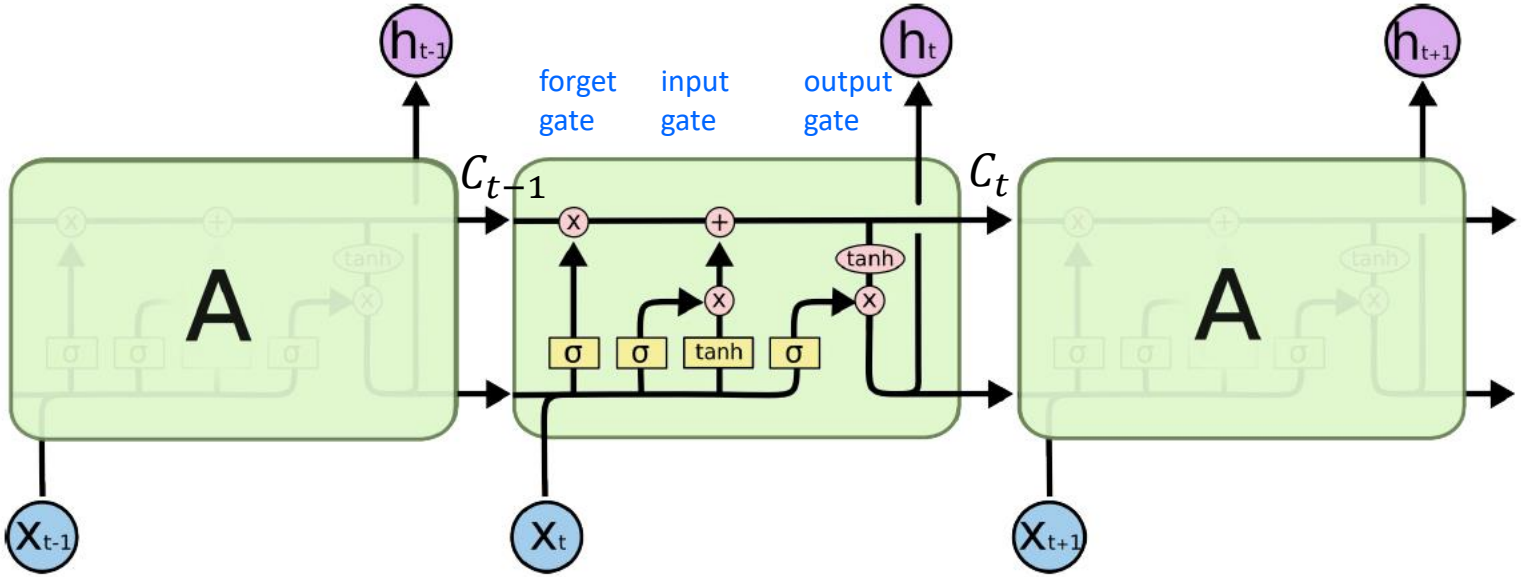
But $s_3 = \tanh(Ux_t + Ws_2)$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

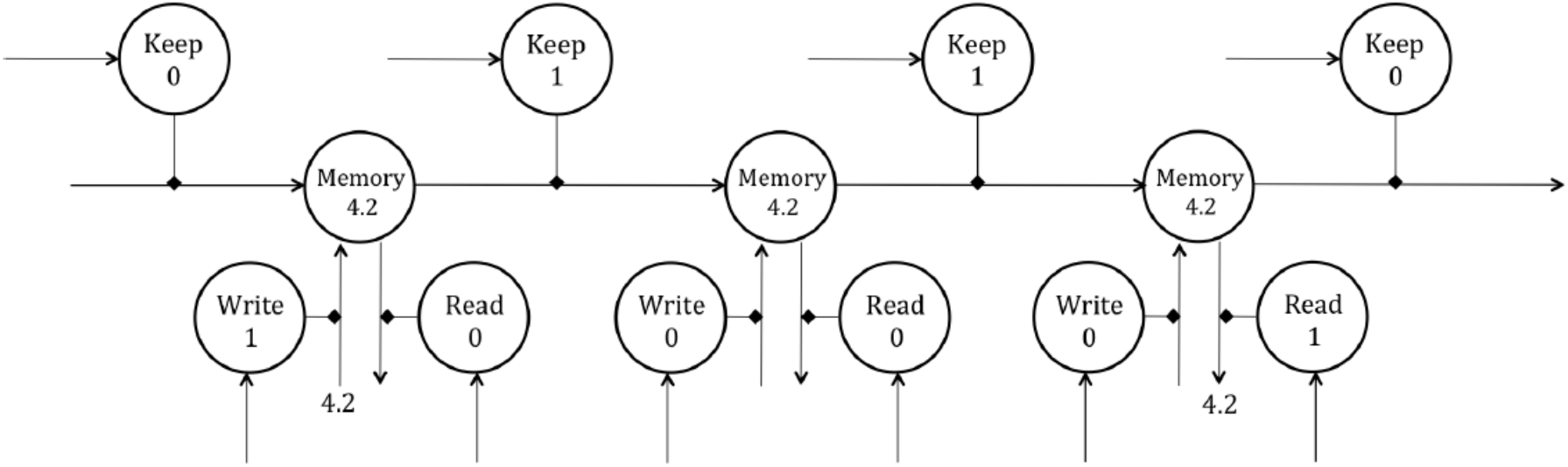


Basic LSTM


forget gate
self-recurrent connection
memory cell input
memory cell output
Input gate
output gate

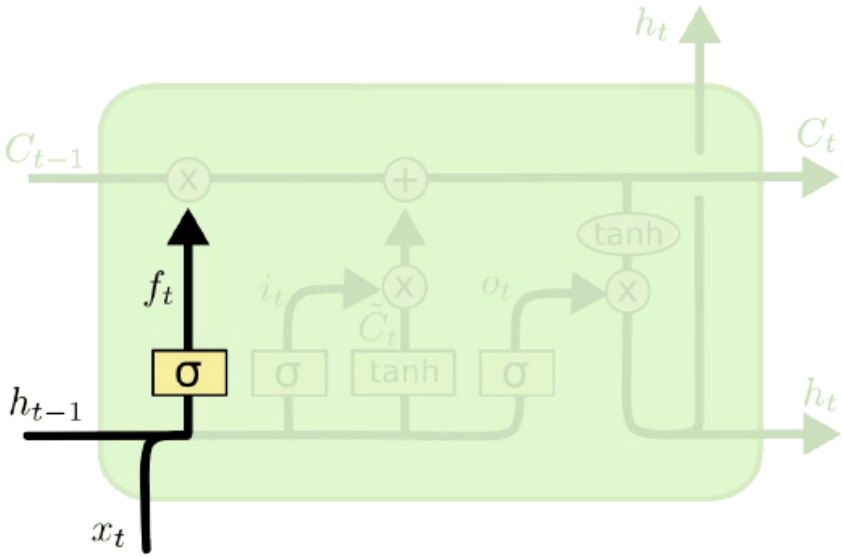

h_{t-1}
h_t
h_{t+1}
forget gate
input gate
output gate
C_{t-1}
C_t
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}

Unrolling the LSTM through time



Step-by-Step LSTM Walk Through

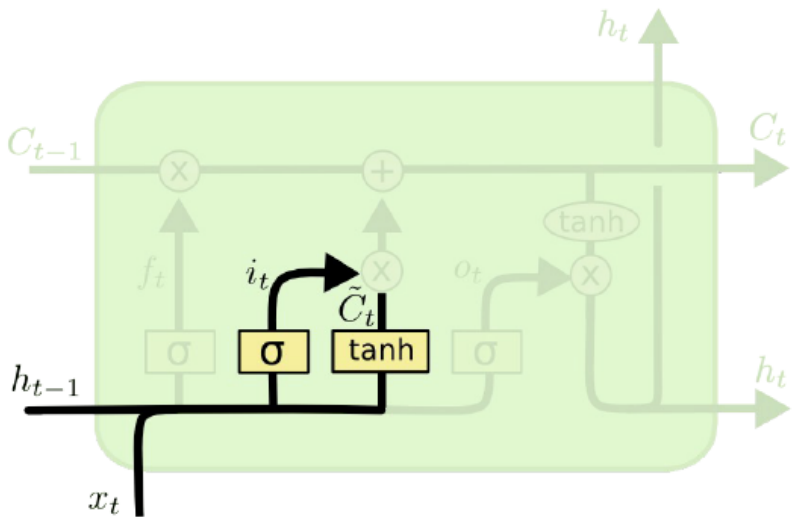
Forget Gate Layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Step-by-Step LSTM Walk Through

Input Gate Layer

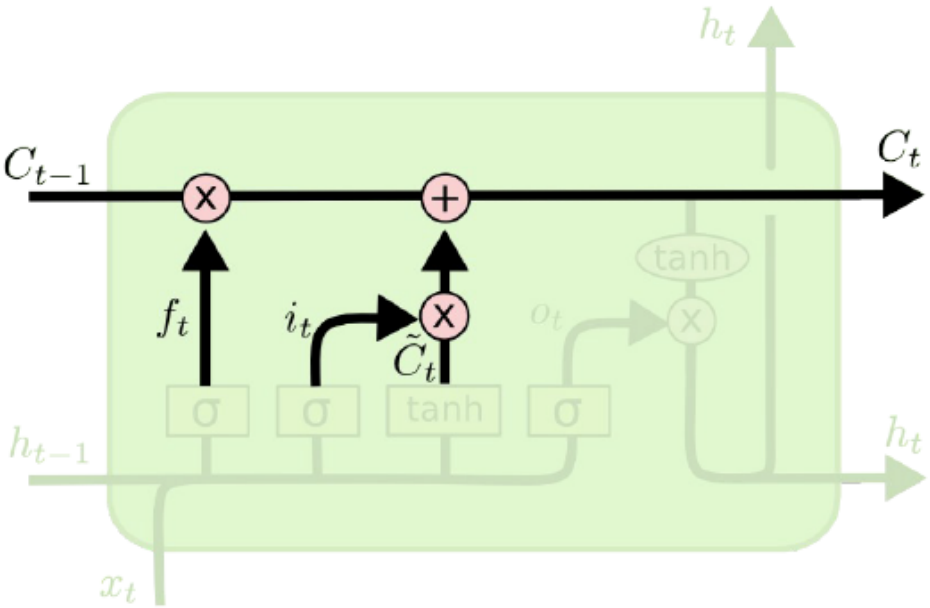


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step-by-Step LSTM Walk Through

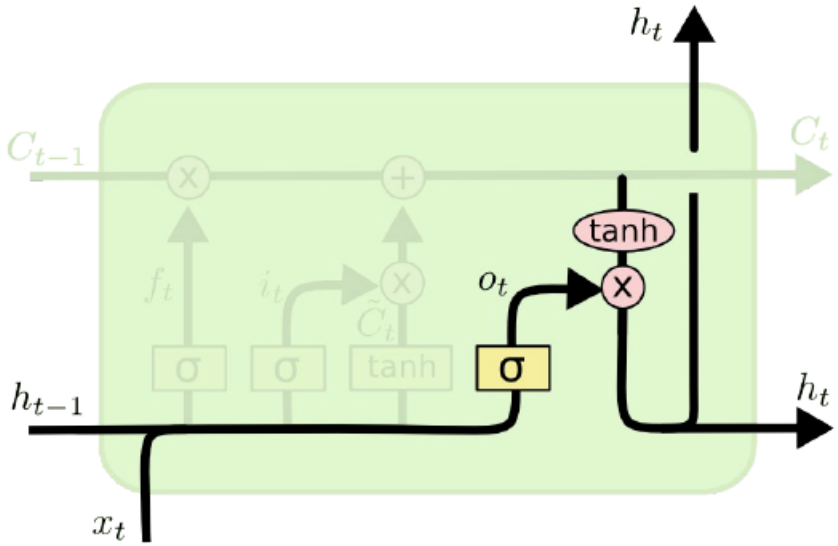
The Current State



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM Walk Through

Output Layer



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Q&A

Images Courtesy of Google Image