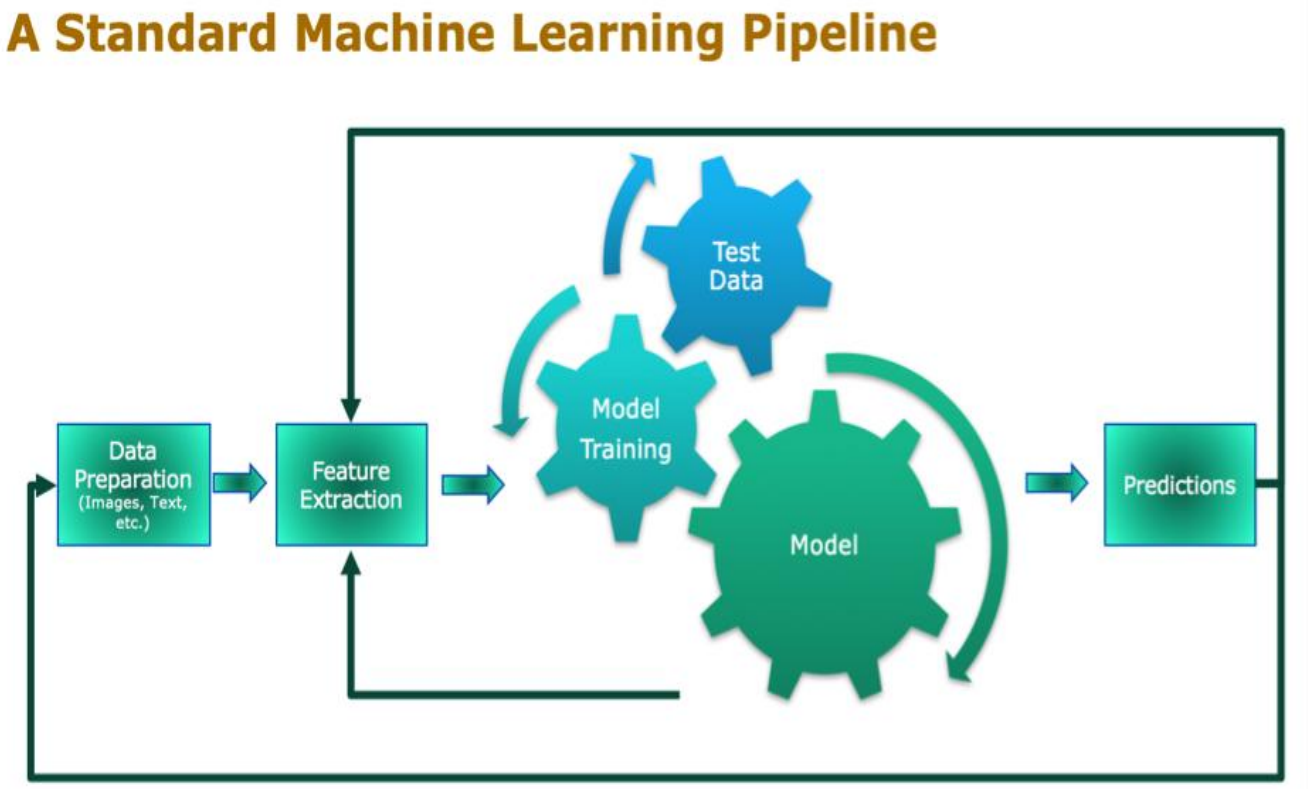
Social Network Analysis

GRAPH REPRESENTATION LEARNING

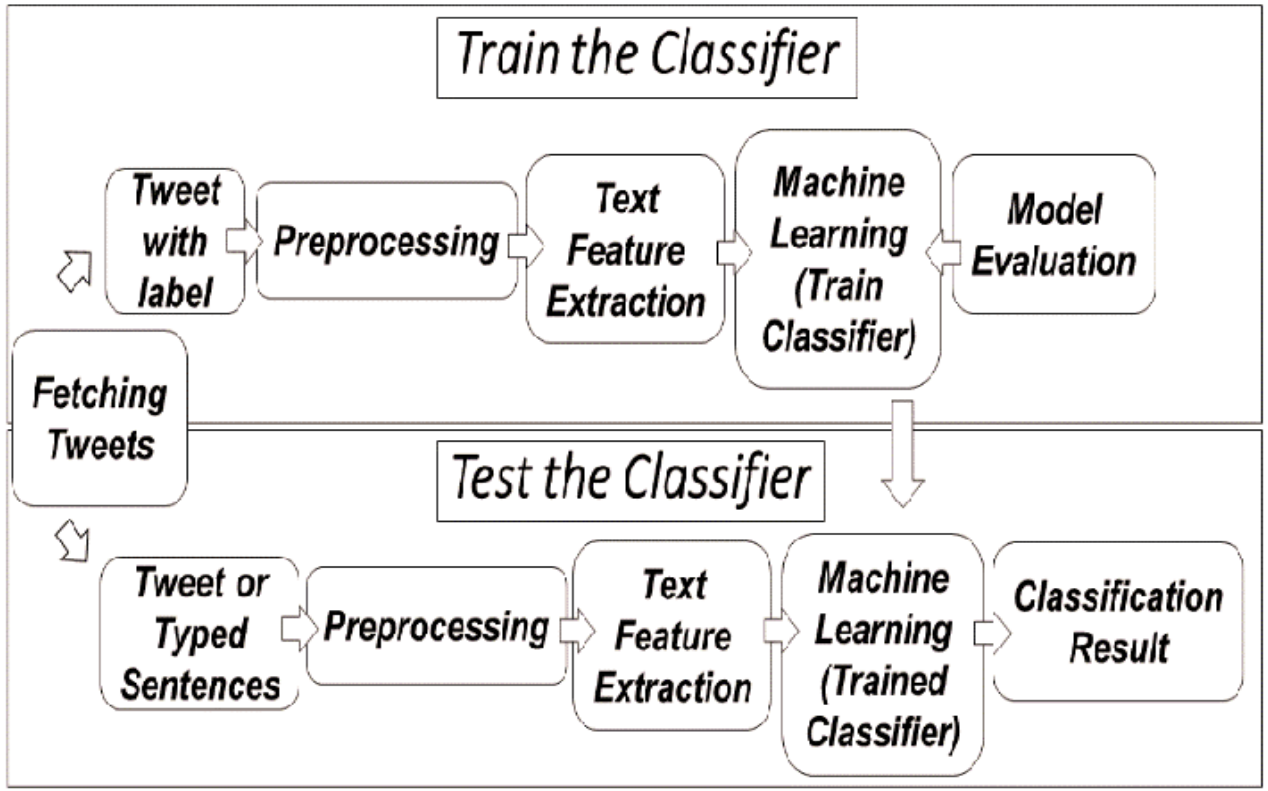
Machine Learning Pipelines



https://www.datanami.com/2018/09/05/how-to-build-a-better-machine-learning-pipeline/

- Data preparation:
  - collecting and annotating data according to requirements
- Data pre-processing:
  - collected data is often noisy and unstructured
  - mandatory cleaning and organizing of the data
- Feature extraction:
  - extract relevant features from our processed data
  - several statistical measures (mean, standard deviation, entropy, etc.) are used as features
  - domain specific features also extracted
- Learning algorithm:
  - features sent as input to ML algorithm for prediction
  - with ground-truth labels (supervised learning)
  - without ground-truth labels (unsupervised learning)

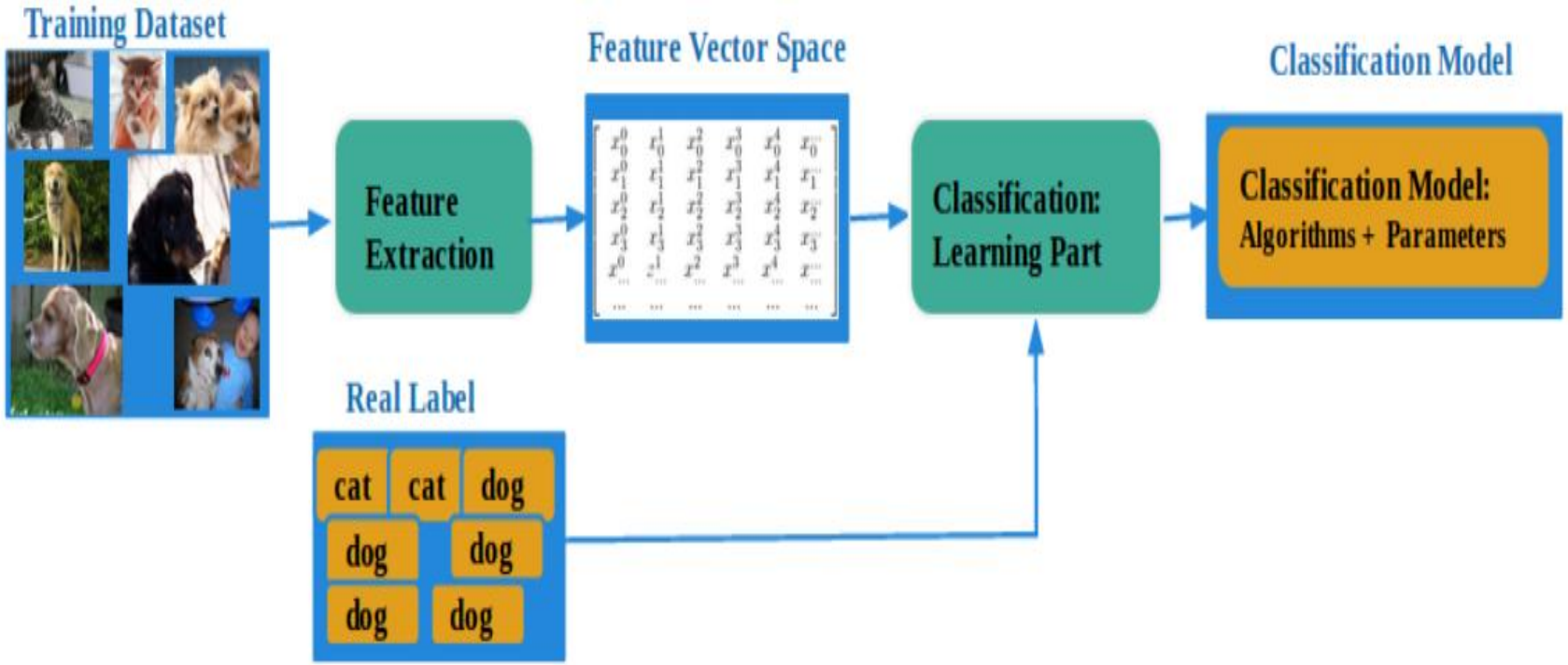
Example: Feature Extraction from Texts



Processes Inside a Tweets Classification Application
https://bit.ly/2OaHSqr

- Several important features can be extracted from the given set of tweets
  - co-occurrence of certain words in the text
  - number of times an author publishes a tweet
  - number of likes on a given tweet
  - number of retweets on a given tweet
  - etc.

Example: Feature Extraction from Images



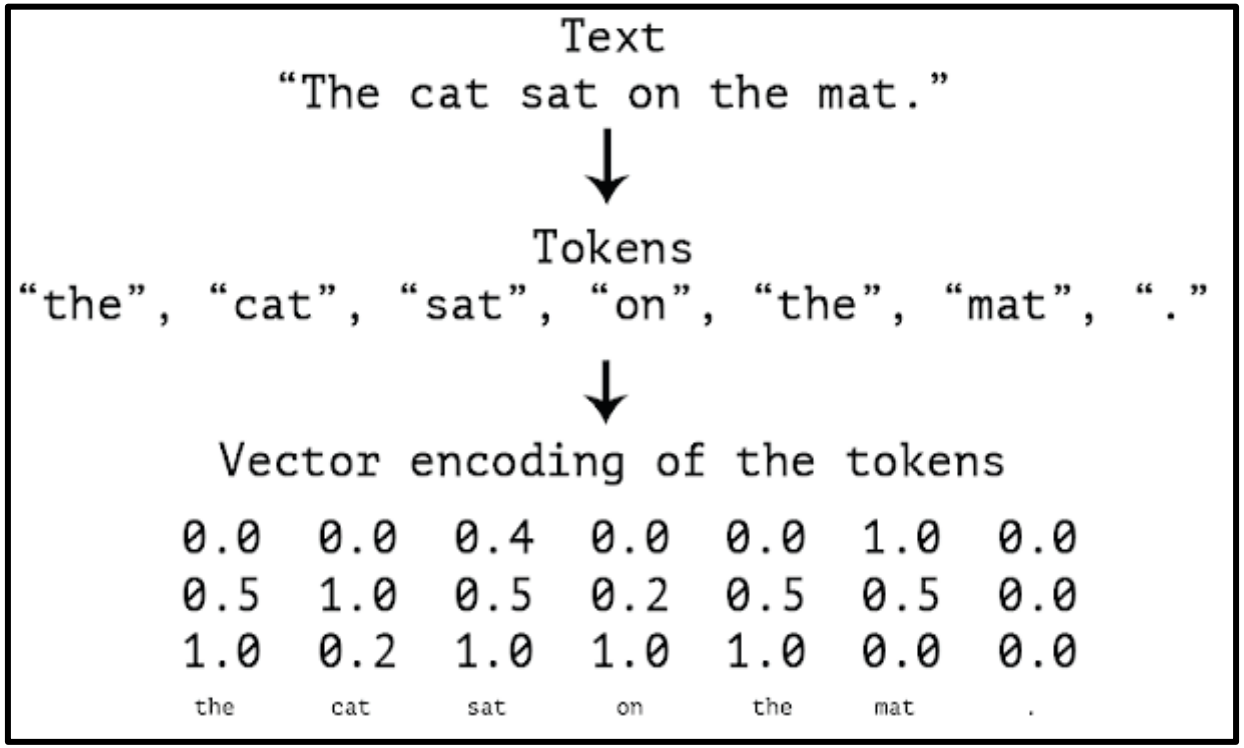
- Given images of dogs and cats, several features can be extracted
  - coordinates of ears
  - number of whiskers in cats
  - shape of the eyes
  - etc.

Image Classification for Dogs and Cats
http://www-labs.iro.umontreal.ca/~liubang/files/DogCat_report.pdf

Feature Extraction: Challenges

- Given a situation, there are a large number of possible features you can extract
  - How should one choose which features to select from this set?
  - Should she take all the features from the pool?
  - Should she take only few out of them?
  - How to make a decision in such a situation?
- Is it possible to encapsulate the feature extraction process with the learning algorithm?
- Can it be ensured to extract features that give the best possible results?
- The answer is Representation Learning

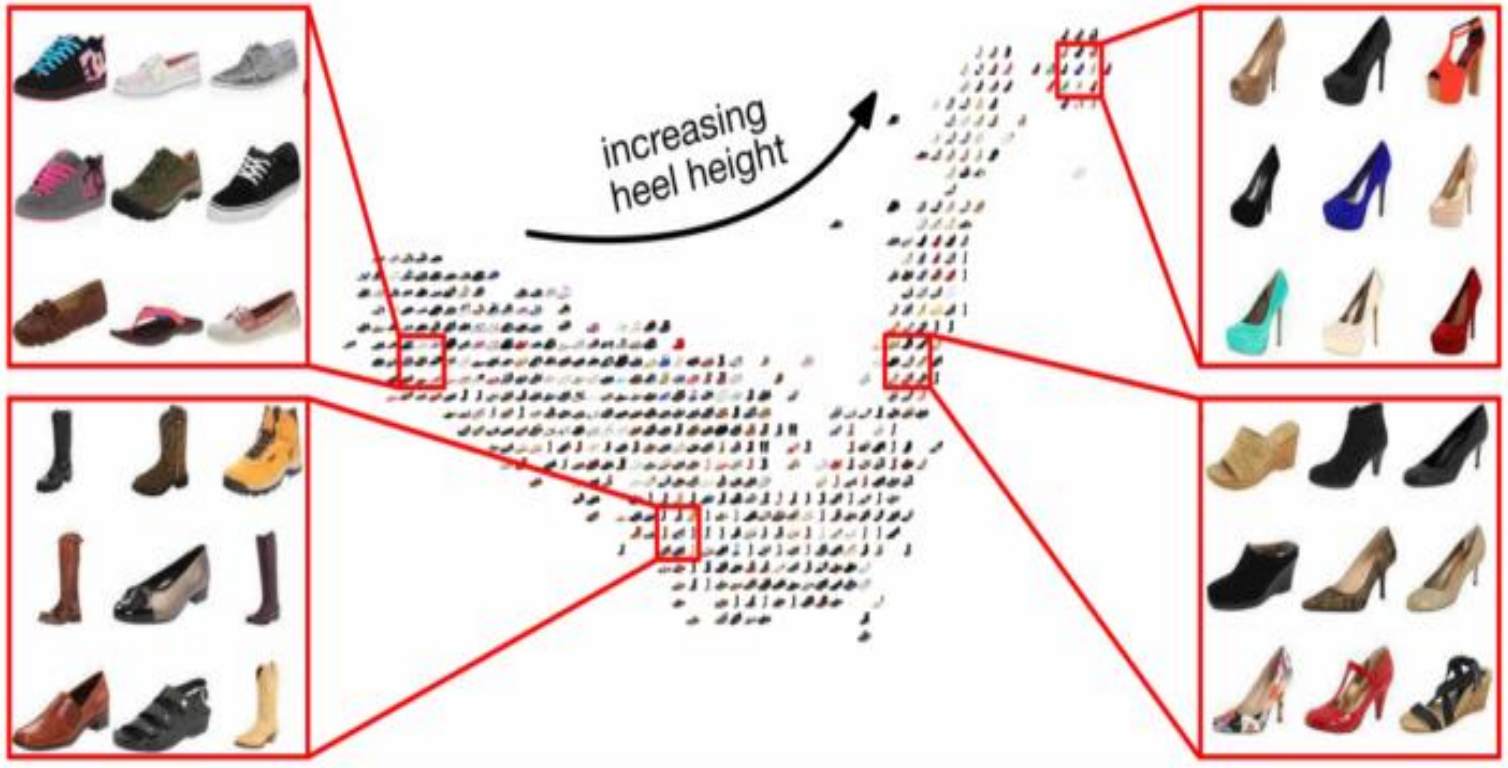
Representation Learning



Representation of words in texts
https://bit.ly/39yvFDs

- The field of machine learning, concerned with automatic computation of features from a given data
- The representation can further be used with various machine learning models
- Also known as feature learning
- Most representation learning algorithms depend on learning a vector
- representations are mostly task-dependent

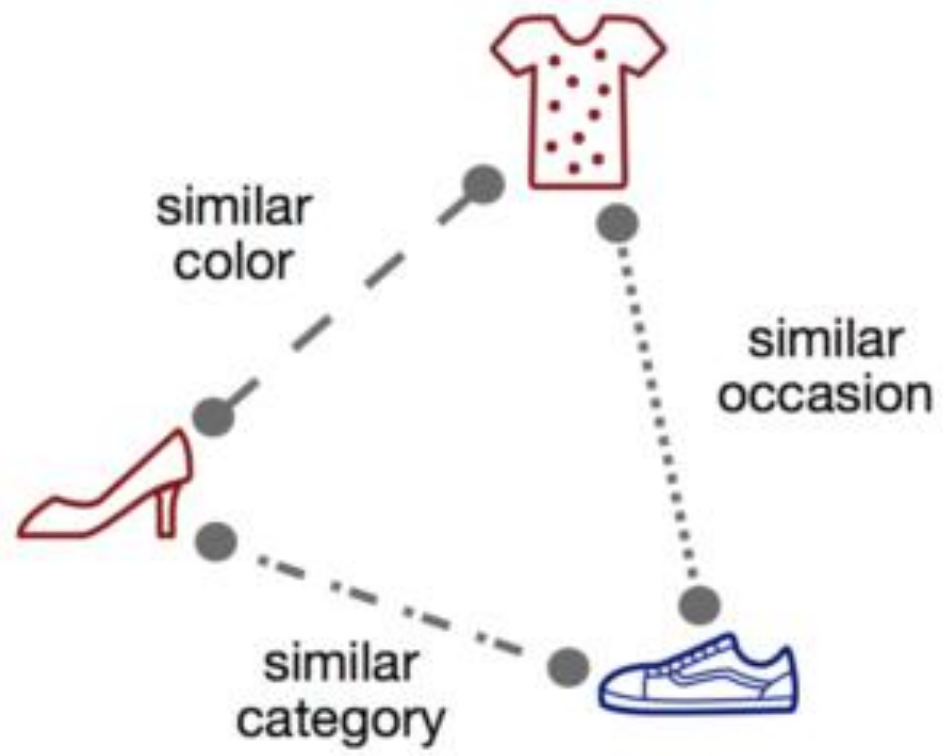
Similarity between Entities



- What makes images similar?
- Images are typically embedded in a feature vector space
- Their distance in feature space preserve the relative dissimilarities
- Notion of cosine similarity in vector spaces is good metric
- The above is the key to representation learning

https://vision.cornell.edu/se3/embeddings-and-metric-learning/

Similarity Assumption



- Simplified assumption regarding similarity is often required to be made
- Images are usually compared against a unique measure of similarity in a situation
- Fine-grained categorization suffers due to
  - lack of training data
  - large number of fine-grained categories
  - high intra-class vs. low inter-class variance

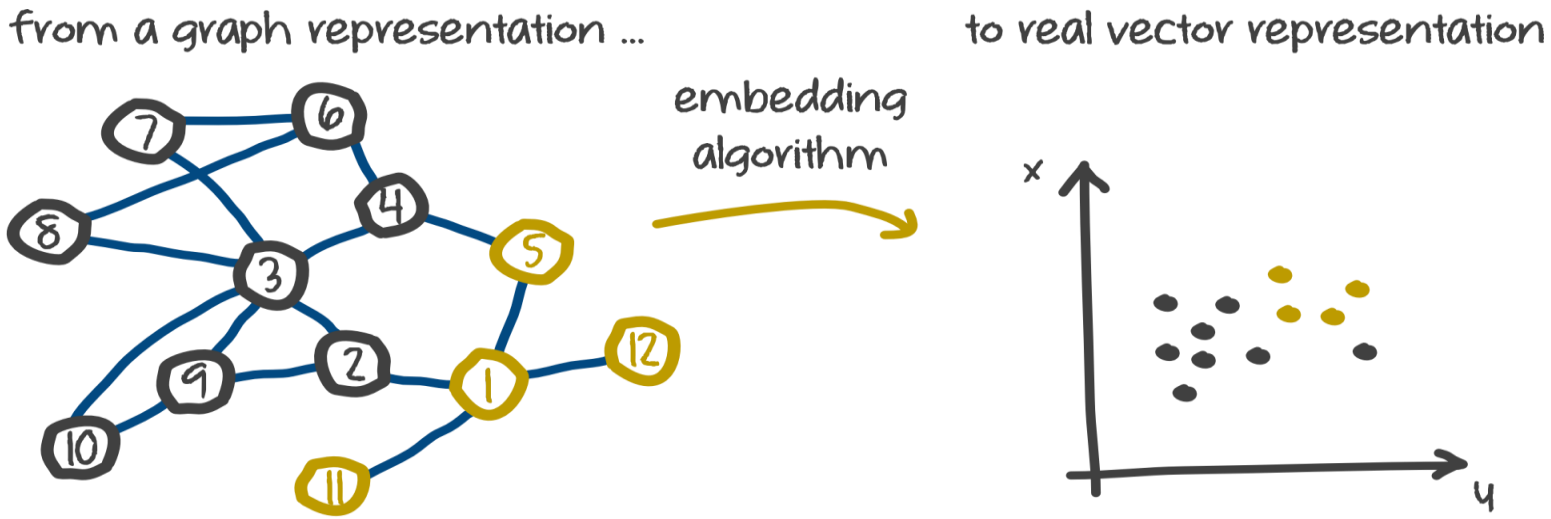
https://vision.cornell.edu/se3/embeddings-and-metric-learning/

Graph Representation Learning

- Graph-theoretic algorithms require to manually tune certain attributes
- Graph Representation Learning is all about employing machine learning algorithms which reduce human intervention significantly
- Has the same end goal as representation learning
- Help us solve most of the graph problems on their own
- Need to devise a method to incorporate a graph as an input into the algorithm

Graph Representation Learning

- Example:
  - map different components of a graph (nodes, edges, sub-graphs, communities) to an embedding space
  - embeddings of similar types of nodes come closer
  - embeddings of dissimilar nodes move away from each other



https://gearons.org/posts/mage

Graph Representation Learning: Challenges

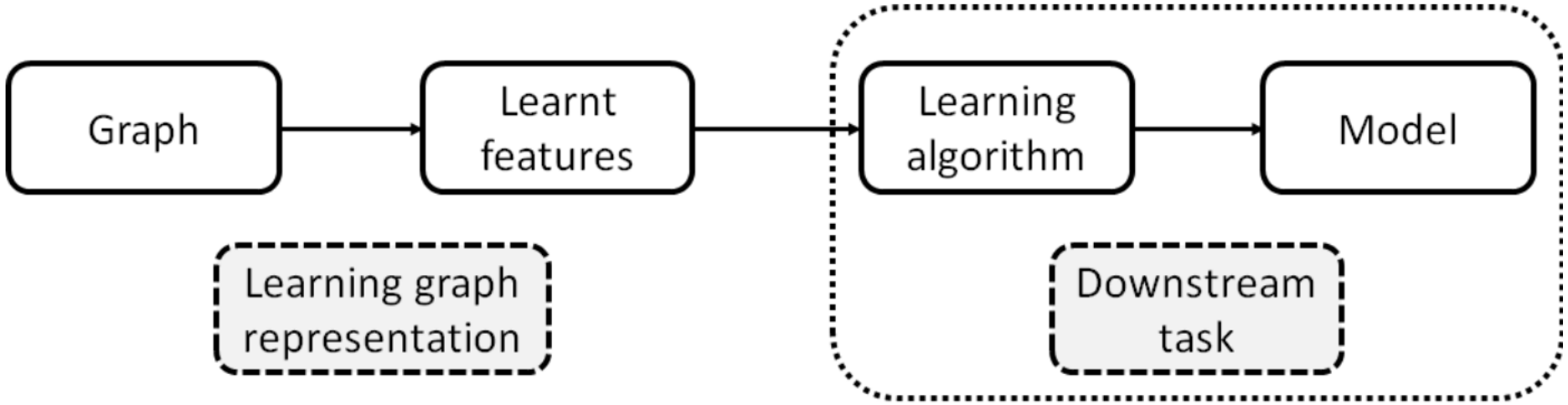
- However, it is more interesting and challenging than other kinds of data
  - Graphs are not sequential in nature
  - Rename the nodes in a graph,
  - Entries in the adjacency matrix will change; but the graph structure will remain same
- Graphs can represent enormously complex data
- Given the complex structure of graphs, what should one encode?
- Roughly define the problem as learning vector representations of various components of a graph
- Depending on use cases, it is possible to find
  - vectors that encode nodes of a graph
  - vectors that encode edges of a graph
  - vectors that encode the entire graph
  - vectors that encode paths in a graph, and so on

Graph Representation Perspectives

- Node Similarity:
  - Nodes that are similar in the graph should have close-by representations in the embedding space
  - Closeness of two nodes may refer shorter path length between them
  - Closeness in the embedding vectors space may be:
    - Euclidean distance
    - cosine similarity, or
    - any other suitable similarity metrics

- Neighbourhood structure Similarity:
  - node representations would contain information about the nodes that are connected to it

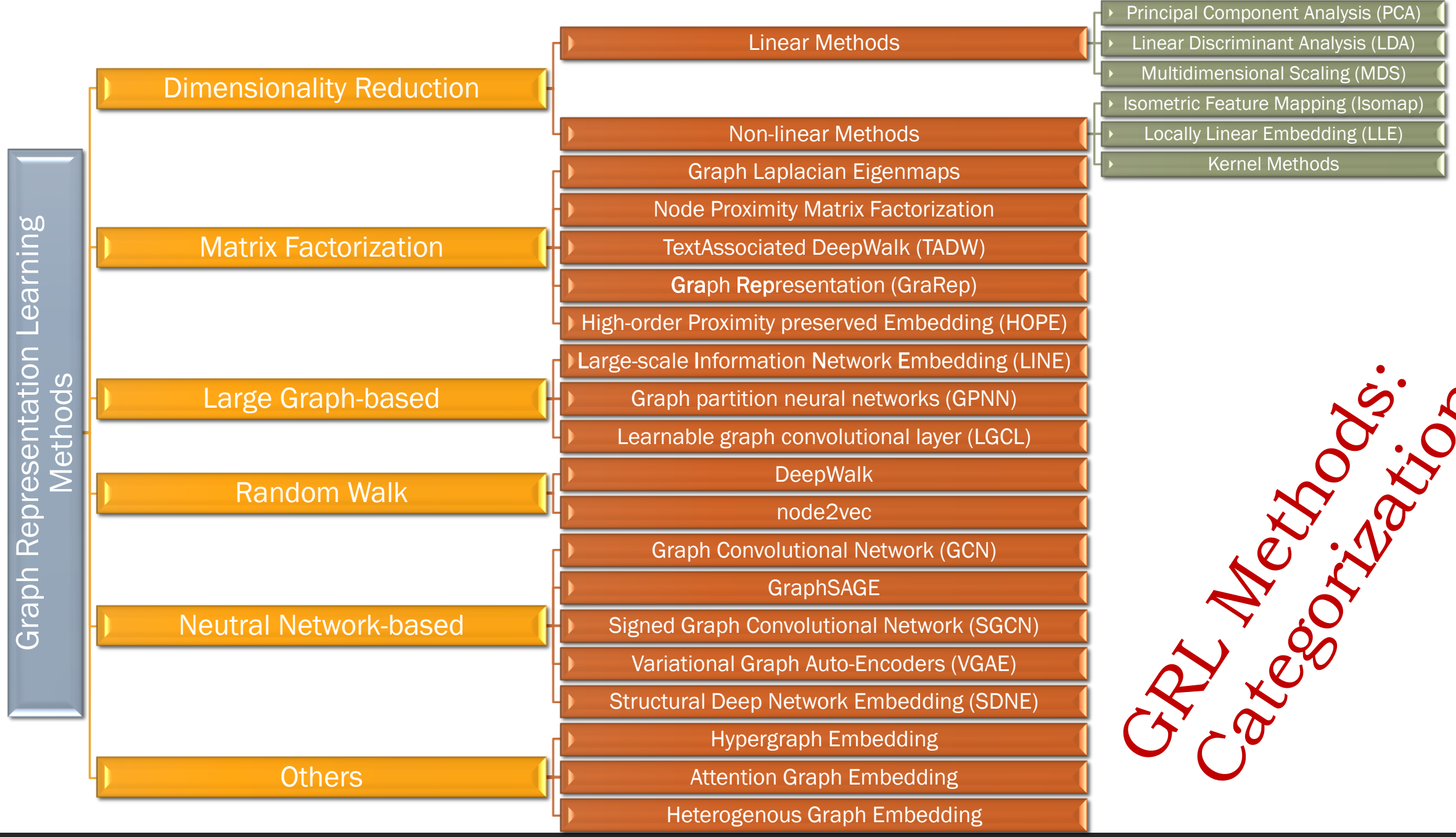
GRL Pipeline



- GRL is used to learn the features
- Learning algorithm is often different from GRL
- The same is called the downstream task
- Support Vector Machine or a similar ML algorithms may be used as the learning algorithm
- Ideally, GRL is independent of the downstream task
- An end-to-end learning encapsulates the learning of features and the classification task into one entire learning task

GRL Pipeline: Components

- Input to GRL pipeline:
  - Homogeneous or heterogeneous graphs
  - Auxiliary information about nodes and edges
- Output to GRL pipeline:
  - Node Embedding
  - Edge Embedding
  - Graph Embedding: only makes sense when we have more than one graph to embed
  - Hybrid Embedding: complex representations of hybrid combinations of nodes and edges


Graph Representation Learning Methods
Dimensionality Reduction
Linear Methods
Principal Component Analysis (PCA)
Linear Discriminant Analysis (LDA)
Multidimensional Scaling (MDS)
Non-linear Methods
Isometric Feature Mapping (Isomap)
Locally Linear Embedding (LLE)
Kernel Methods
Matrix Factorization
Graph Laplacian Eigenmaps
Node Proximity Matrix Factorization
TextAssociated DeepWalk (TADW)
Graph Representation (GraRep)
High-order Proximity preserved Embedding (HOPE)
Large Graph-based
Large-scale Information Network Embedding (LINE)
Graph partition neural networks (GPNN)
Learnable graph convolutional layer (LGCL)
Random Walk
DeepWalk
node2vec
Neutral Network-based
Graph Convolutional Network (GCN)
GraphSAGE
Signed Graph Convolutional Network (SGCN)
Variational Graph Auto-Encoders (VGAE)
Structural Deep Network Embedding (SDNE)
Others
Hypergraph Embedding
Attention Graph Embedding
Heterogenous Graph Embedding
GRL Methods: Categorization

GRL Method Categories: Dimensionality Reduction based

- Reduce the dimension of a high-dimensional graph data into low dimension
- Preserve as many properties of the original data as possible
- Extremely General Methods
- Popular methods in this category
  - Principle Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)

GRL Method Categories: Matrix Factorization based

- Older paradigm of learning graph features
- adjacency matrix of a graph is a good representative of its connectivity
- Large dimensions of the adjacency matrix restricts its direct use in representation learning
- Factorize adjacency matrix of the graph keeping structure that needs to be highlighted preserved after the factorization
- Provides some key insights on network embedding
- Slower compared to random walk based or neural network based methods

GRL Method Categories: Random Walk based

- Do not enforce traversing all the nodes in a graph
- Only a small neighborhood of a node in the graph is traversed with the help of random walks
- Performs extremely well on large networks

GRL Method Categories: Neural Network based

- To design graph representation learning algorithms based on neural networks
- Recently gained popularity due to the rapid rise of computing power
- GPU computing methodologies enable neural network based methods to run efficiently
- Neural network methods specialize in
  - abstracting a lot of details of the problem description, and
  - implicitly representing complex mathematical functions

GRL Method Categories: Large Graph based

- Large graphs have vast real-life existence
- Several space and time complexity restrictions
- Need to develop more efficient, yet accurate, representation methods

Matrix Factorisation based GRL Method: Node Proximity Matrix Factorization

- Each node is representation using a d-dimensional embedding ($d \ll |V|$)
- Resultant Matrix $X \in \mathbb{R}^{|V| \times d}$
- context matrix, X^c, for the graph is defined on the basis of a property
- Example: If one need encoding neighborhood information, context matrix of a source node is a combined polynomial matrix that contains all the representations of its corresponding neighbors
- Problem statement: Given a higher dimensional matrix W, the aim is to produce a low-dimensional representation $X \in \mathbb{R}^{|V| \times d}$, given the context matrix X^c
- Find closeness of matrix W with representation X using L^2 norm:

$$min \| W - X(X^c)^T \|$$

Matrix Factorisation based GRL Method: Node Proximity Matrix Factorization

- Singular Value Decomposition (SVD): an approach to obtain X from W:

$$W = \sum_{i=1}^{|V|} \sigma_i u_i (u_i^c)^T \approx \sum_{i=1}^{d} \sigma_i u_i (u_i^c)^T$$

$\sigma_i : i = 1, \cdots, N$ are the singular values of W in descending order

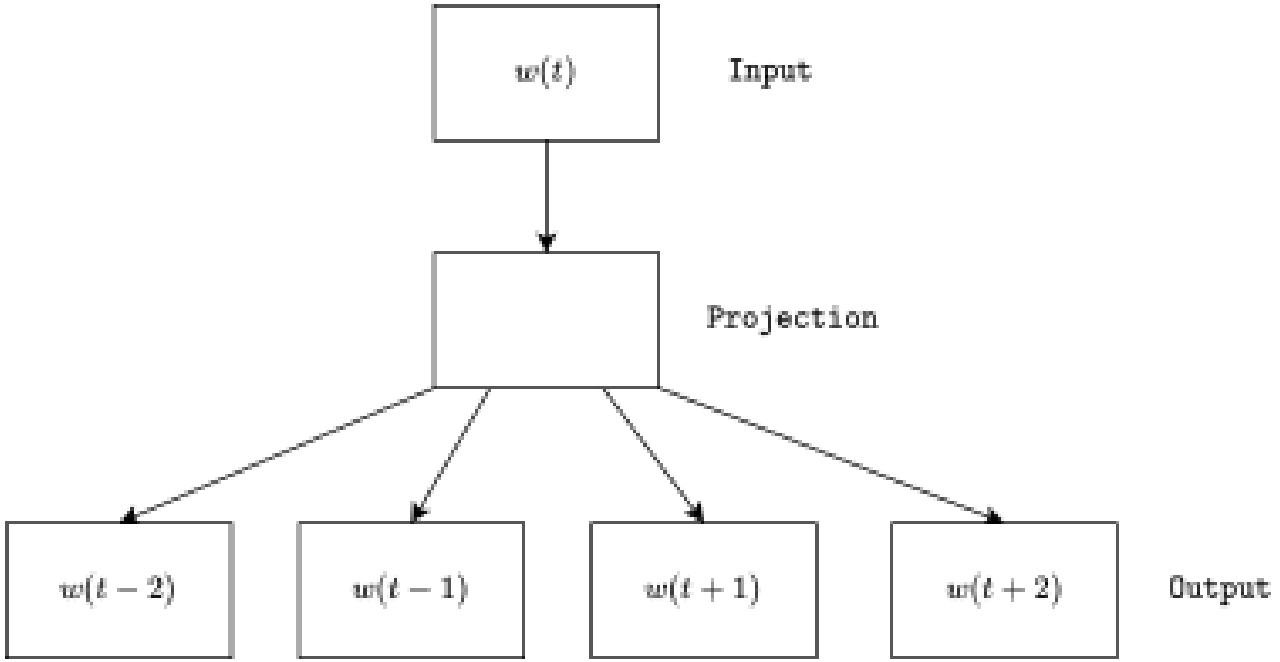
u_i and u_i^c: singular vectors of σ_i

- Optimal embedding:

$$X = [\sqrt{\sigma_1} \cdot u_1^s, \cdots, \sqrt{\sigma_d} \cdot u_d^s]$$
$$X^c = [\sqrt{\sigma_1} \cdot v_1^t, \cdots, \sqrt{\sigma_d} \cdot v_d^t]$$

Matrix Factorisation based GRL Method: **Gra**ph **Rep**resentation (GraRep)

- Skip-gram model in Word2Vec is used to predict context words given a source word



- GraRep uses skip-gram model to capture different *k*-step relational information between vertices in distinct sub spaces

Word2Vec and Skip-gram

- Word2vec: a word-representation technique used to represent words as vectors of a given size
- skip-gram:
  - an algorithm used by Word2vec to construct the vectors
  - to predict the 'context' given an input word
  - to find words in the context so that the probability of the surrounding context is maximized
- The log likelihood:

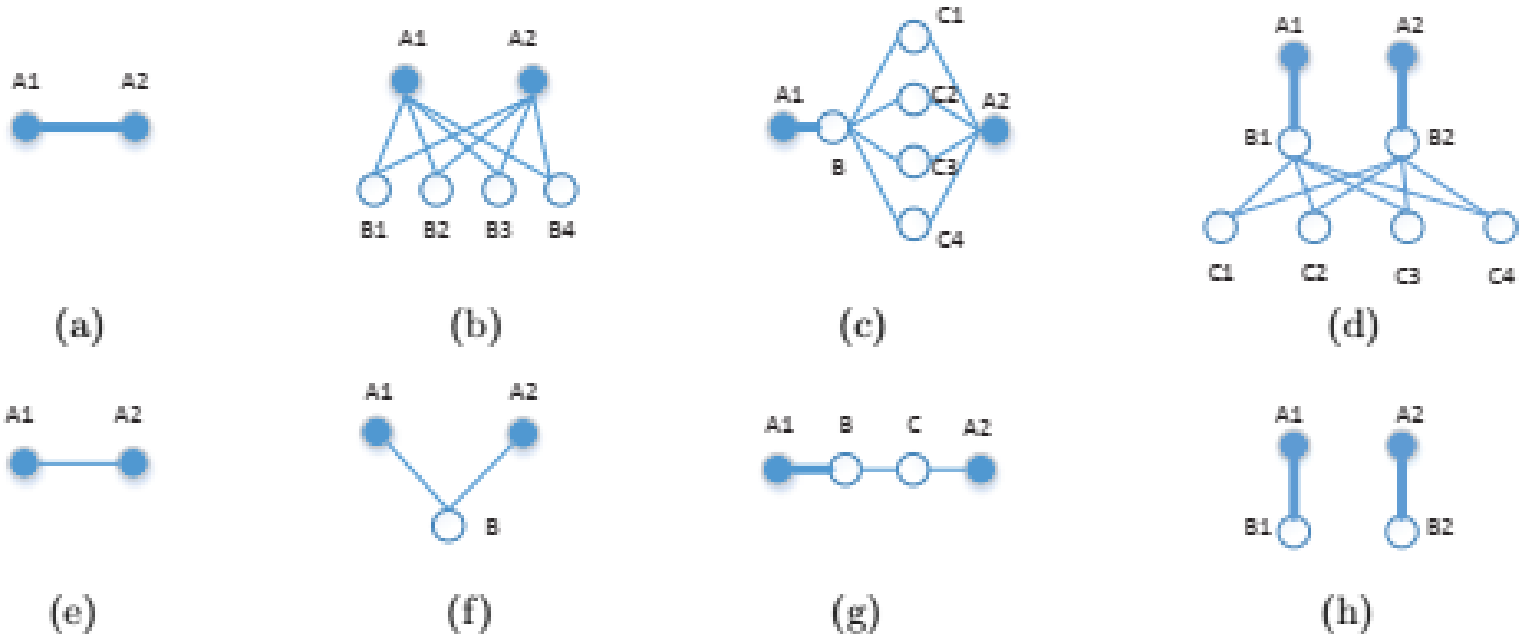
$$maximize\ J = \log P[\omega(c-m), \cdots, \omega(c-1), \omega(c+1), \cdots, \omega(c+m)|\omega(c)]$$

- Using Markov Assumption:

$$minimize\ J = -\log \prod_{\substack{j=0 \\ j \neq m}}^{2m} P[\omega(c-m+j)|\omega(c)]$$

- In case of graphs, to replace the sequence of words by a sequence of nodes obtained by a random walk

Matrix Factorisation based GRL Method: **Gra**ph **Rep**resentation (GraRep)



- Example graphs highlight the importance of capturing k-step relational information
- An increment in k value captures an increasing global structural information of the graph relative to a particular node
- Probability of transition from current vector ω to context vector c in exactly k steps: $p_k(c|\omega) = A^k_{\omega,c}$

Graph **Rep**resentation (GraRep): Positive and Negative Sampling

- Current vector ω and context vector c are connected via a path of maximal path length of k
- Current vector ω and negatively sampled context vector c' are not connected via a path of maximal path length of k
- Negatively sampled context vector at step k is a context node which is not at a distance of k from the current vertex
- for a particular k, the loss function, motivated by the skip-gram model and negative sampling is:

$$L_k(\omega) = \left(\sum_{c \in V} p_k(c|\omega) \cdot \log\left(\sigma(\vec{\omega} \cdot \vec{c})\right) \right) + \lambda \mathbb{E}_{c' \sim p_k(V)} \left[\log \sigma\left(-\vec{\omega} \cdot \vec{c'}\right) \right] \cdots\cdots\cdots (*)$$

$p_k(V)$: distribution over the vertices in the graph

λ: hyper parameter for the number of negative samples

Graph **Rep**resentation (GraRep): Positive and Negative Sampling

- Rewrite equation $(*)$ as:

$$L_k(\omega, c) = p_k(c|\omega) \cdot \log\left(\sigma(\vec{\omega} \cdot \vec{c})\right) + \lambda \cdot p_k(c) \cdot \log \sigma\left(-\vec{\omega} \cdot \vec{c'}\right) \cdots\cdots\cdots (**)$$

- Assuming a normal distribution for the probability of selecting ω' as the seed vertex, $(**)$ takes the form:

$$L_k(\omega, c) = A^k_{\omega,c} \cdot \log\left(\sigma(\vec{\omega} \cdot \vec{c})\right) + \frac{\lambda}{N} \cdot \sum_{\omega'} A^k_{\omega',c} \cdot \log \sigma\left(-\vec{\omega} \cdot \vec{c'}\right) \cdots\cdots\cdots (***)$$

- Setting $a = (\vec{\omega} \cdot \vec{c})$ and letting $\frac{\partial L_k}{\partial a} = 0$, we get

$$\vec{\omega} \cdot \vec{c} = \log\left(\frac{A^k_{\omega,c}}{\sum_{\omega'} A^k_{\omega',c}}\right) - \log\left(\frac{\lambda}{N}\right)$$

Graph **Rep**resentation (GraRep): Algorithm

- We need to factorize matrix X into two matrices W and C such that

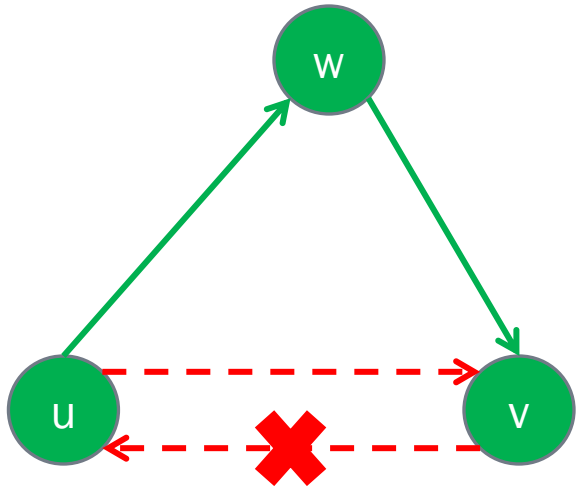
$$X_{n_1,n_2}^k = W_{n_1}^k \cdot C_{n_2}^k = \log\left(\frac{A_{n_1,n_2}^k}{\sum_n A_{n,n_2}^k}\right) - \log\left(\frac{\lambda}{N}\right)$$

  - Step 1: Compute k-step transition probability matrix A^k for $k = 1, \cdots, K$, the maximal path length of the graph
  - Step 2: Compute k-step log probability matrix X^k and subtract by the normalized constant hyperparameter λ. Replace negative entries by 0
  - Step 3: Apply SVD to get the final representation vector

Graph **Rep**resentation (GraRep): Observations

- Consistent performance regardless of graph size
  - good performance for small graphs as well
- Performance increases with an increasing k
  - performance is shown to saturate after a particular k
- Exponential increase in running time
  - primarily due to SVD which is needed to be performed inside the loop

Higher Order Proximity preserved Embedding (HOPE)



- Transitivity relation between nodes in an undirected graph:
  - if nodes u and v are individually connected to a node w, then probability that there exists a relationship between u and v is high
- Transitivity is asymmetric in directed graph
- If there is an edge from nodes u to w and w to v, then the probability of an edge from u to v increases, not v to u
- HOPE aims to generate lower order representations of nodes that can preserve the asymmetric transitivity in directed graphs
- Proposed by Ou et al. in 2016

Higher Order Proximity preserved Embedding (HOPE)

- Find two representations of each node:
  - target representation: U^t
  - Source representation: U^s
- If there exists an edge from u to v without a reverse link from v to u, then
  - source representation of u would have a similar value to the target representation of v
  - the target representation of u and source representation of v would contain different values
- attempts to minimize the L_2 loss function as the objective loss function such that

$$min\|S - U^s \cdot (U^t)^T\|_F^2$$

S: A similarity-based matrix calculated on the basis of higher-order proximity measurements such as Katz Centrality, Adamic Adar, common neighbors, etc.

$$S = M_g \cdot M_l^{-1}$$

M_g and M_l are polynomial matrices that capture different aspects of the proximity

Higher Order Proximity preserved Embedding (HOPE)

- Consider Katz Index as:

$$S^{Katz} = \sum_{l=1}^{\infty} \beta \cdot A^l = \beta \cdot A \cdot S^{Katz} + \beta \cdot A$$

decay factor β is is used to reduce the influence of far away nodes in the path

- On convergence:

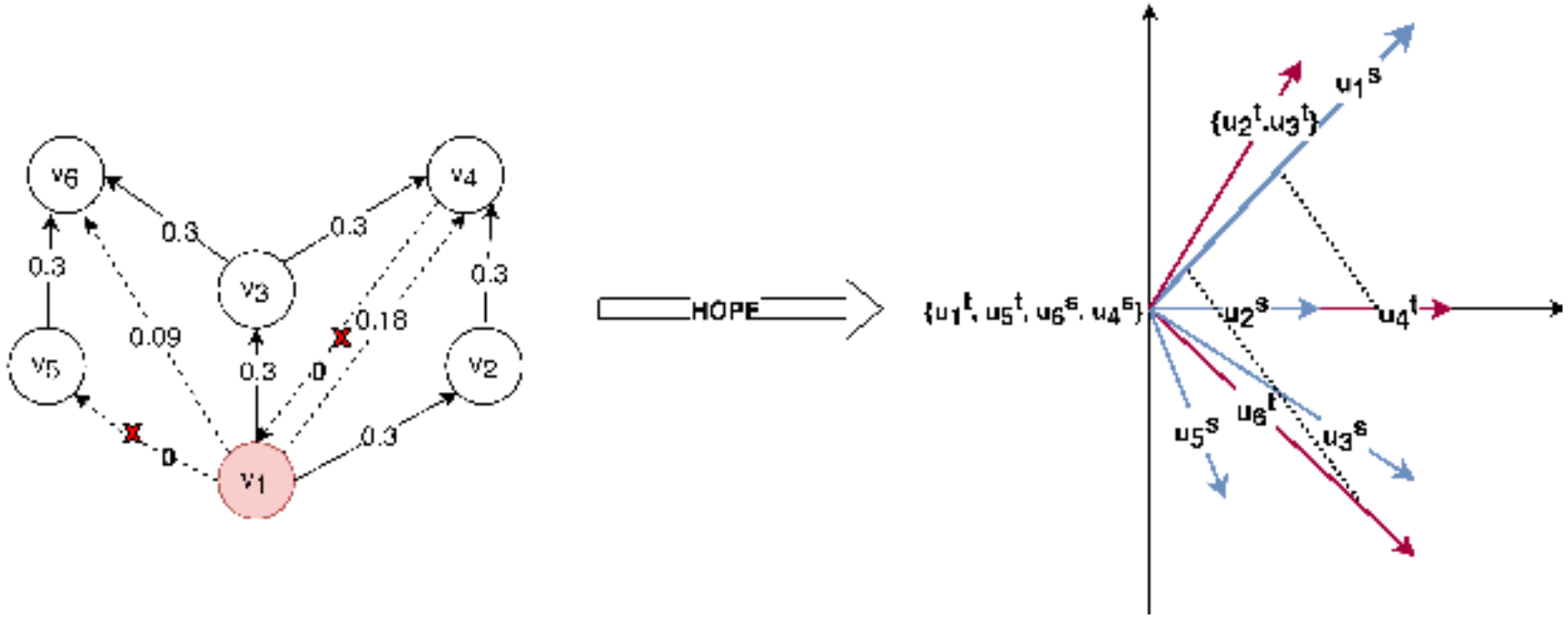
$$S^{Katz} = (I - \beta \cdot A)^{-1} \cdot \beta \cdot A$$

- On comparison:

$$M_g = (I - \beta \cdot A);\ M_l = \beta \cdot A$$

Higher Order Proximity preserved Embedding (HOPE)

- Example figure demonstrates asymmetric transitivity preservation by HOPE in the embedding space



Higher Order Proximity preserved Embedding (HOPE)

- Apply SVD, and then use the largest k singular values and singular vectors to make the embedding vectors

$$U^s = [\sqrt{\sigma_1} \cdot v_1^s, \cdots, \sqrt{\sigma_K} \cdot v_K^s];\ U^t = [\sqrt{\sigma_1} \cdot v_1^t, \cdots, \sqrt{\sigma_K} \cdot v_K^t]$$

σ_i: singular values in descending order and v_i^s and v_i^t: singular vectors σ_i

- Time complexity of the entire process is extremely high. HOPE removes the requirement of calculating the similarity matrix S, based on the following:

Theorem: If we have the singular value decomposition of the general formulation $M_g \cdot M_l = V_s \cdot \Sigma \cdot V_t$, where V_t and V_s are two orthogonal matrices and $\Sigma = diag(\sigma_1, \cdots, \sigma_N)$, hen there exists a non-singular matrix X and two diagonal matrices $\Sigma^l$and Σ^g satisfying the followings:

$$V^{t^T} \cdot M_l^T X = \Sigma^l V^{s^T} \cdot M_g^T X = \Sigma^g$$

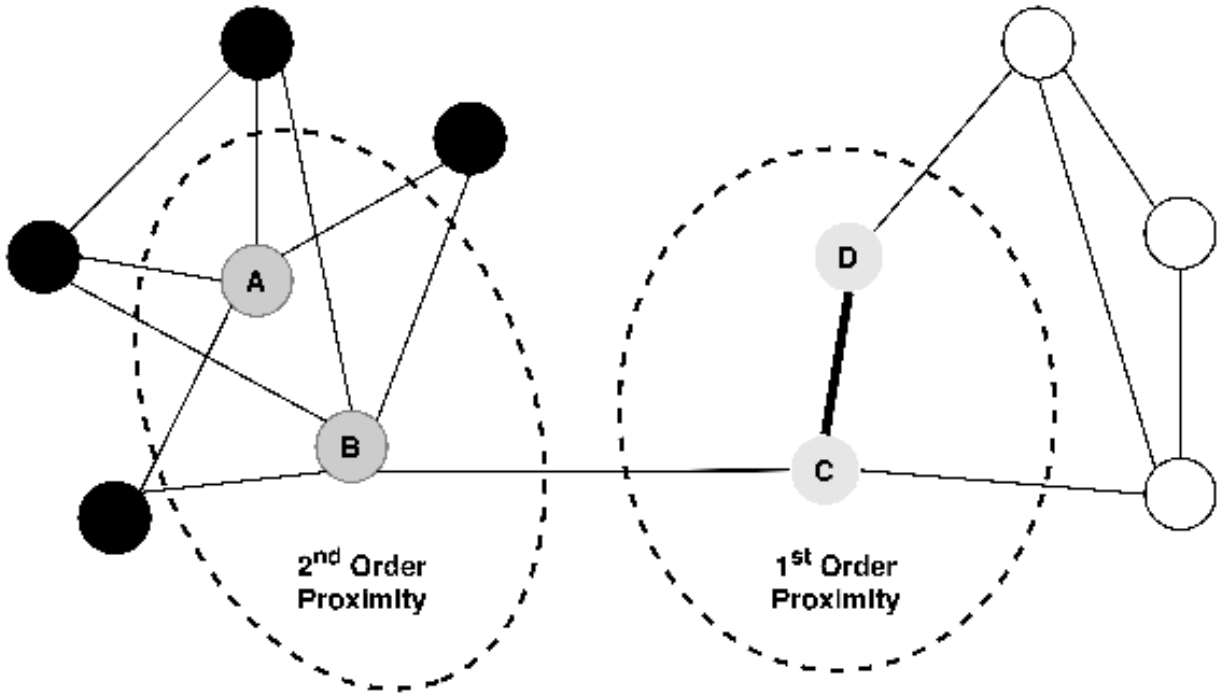
$$\Sigma^l = diag(\sigma_1^l, \cdots, \sigma_N^l);\ \Sigma^g = diag(\sigma_1^g, \cdots, \sigma_N^g); \sigma_1^l \geq \sigma_2^l \geq \cdots \geq \sigma_k^l \geq 0; \sigma_1^g \geq \sigma_2^g \geq \cdots \geq \sigma_l^g \geq 0$$

$$\sigma_i^{l^2} + \sigma_i^{g^2} = 1\ \forall i;\ \ and\ \ \sigma_i = \frac{\sigma_i^l}{\sigma_i^g}\ \forall i$$

Large-scale Information Network Embedding (LINE)

- Tang et al. introduced LINE in 2015 aiming to find large graph embeddings
- incorporates first-order and second-order proximities to compute embeddings

- First-order proximity:
  - The nodes which are directly connected to the source node
  - preserves the local network neighborhood of the node
- Second-order proximity:
  - The nodes which have many common neighbors
  - preserves the global structure of the graph



Large-scale **I**nformation **N**etwork **E**mbedding (LINE)

- Second-order performs better on graphs which are dense
- First-order performs better on nodes with less degree
- Combined first-order and second-order perform better than both individually

Random Walk Based Methods

- Use random walks to learn the low-dimensional latent embedding of each node
- Captures the local neighborhood and structure of a node by performing enough random walks with a seed node as the starting node
- Output of multiple random walks is combined together and used in an optimization function
- The method is fast
  - Multiple random walks on different seed nodes could be computed in parallel
  - If a change occurs in large real-world networks, only the effected nodes needs to be recomputed
- Popular algorithms from the category:
  - DeepWalk
  - node2vec

DeepWalk

- Uses uniform random walks based on a seed node aiming to identify the local neighborhood of the node and capture local community information
- Generating random walk:
  - a seed vertex is sampled
  - a set of multiple random walks based upon this seed vertex are computed
  - random walks are uniform and have fixed length
  - length of the random walk could be set randomly
  - a teleport probability could be assigned to random walks
- Updating the parameters:
  - a skip-gram language model is used to maximize the co-occurrence probability among the nodes
  - the skip-gram probability minimization function:

$$min_U - \log\left(P\left(n_{i-\omega}, \cdots, n_{i-1}, n_{i+1}, \cdots, n_{i+\omega} | U(n_i)\right)\right)$$

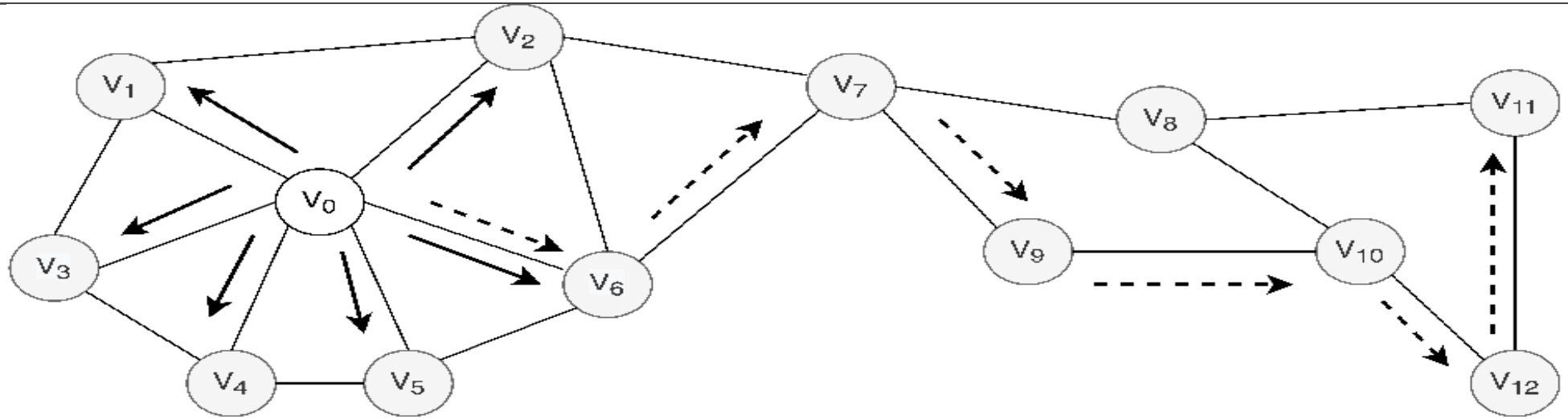
DeepWalk

- Different classifiers could be used for learning the distribution
- Original architecture uses a hierarchical softmax
- Softmax layers are typical in neural networks for computing the probabilities of the input lying in each of the possible output classes
- Softmax function is given by:

$$softmax(x) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} [e^{x_1} e^{x_2} \cdots e^{x_n}]^T; \quad x \in \mathbb{R}^n$$

- To reduce the number of computations in Softmax, in hierarchical softmax
  - classes are grouped in a binary tree formation, the classes form the leaf nodes
  - calculate the probability of the path from the root node to the leaf node

BFS vs DFS on Graphs



- Breadth First Search (BFS): All the k^{th} proximity neighbors of a node are visited before proceeding to $(k+1)^{th}$ proximity neighbors
  - Solid Arrow shows the BFS execution on the example graph

- Depth First Search (DFS): All the neighbors of the currently visiting node in a level are explored before proceeding to the next node from that level
  - Dashed Arrow shows the DFS execution on the example graph

node2vec

- Expands upon DeepWalk
  - random walks conducted in node2vec is biased
  - bias the second-order random walk by defining two parameters – p and q
- Random walker sourced at node t decides the next node x that it would visit from currently visiting node v on the basis of a transition probability from v to x given by

$$\pi_{v,x} = \alpha_{p,q}(t,x) \cdot \omega_{v,x}$$

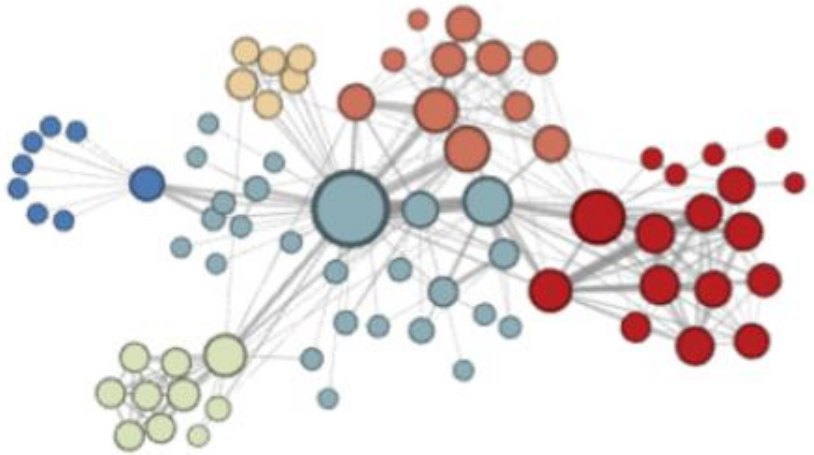
Where $\omega_{v,x}$ is the weight of the edge between v and x, and

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p} & if\ d_{t,x} = 0 \\ 1 & if\ d_{t,x} = 1 \\ \frac{1}{q} & if\ d_{t,x} = 2 \end{cases}$$

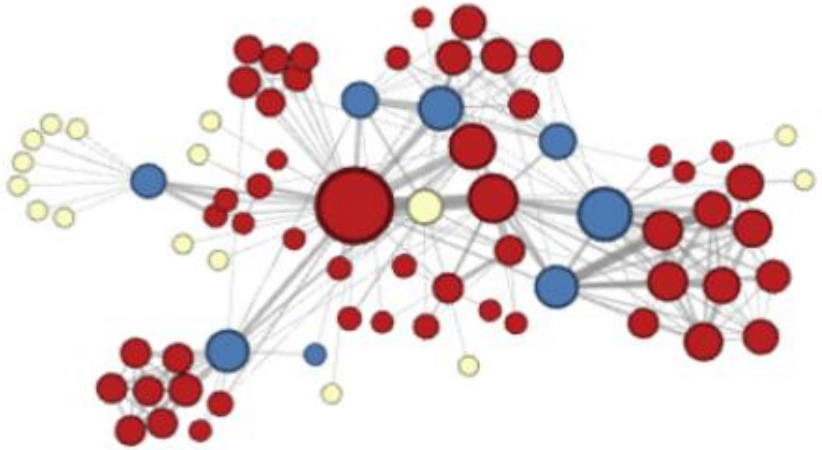
Les Misérables Network: Impact of p and q in node2vec

- Return Parameter (p): determines how far the random node is to explore from the source node. A high value of p indicate that random walker is more likely to walk 'away' from the source node

- In-out Parameter (q): guides the random walker between wither the inward or the outward nodes.
  - If $q < 1$, then the walker is more biased to move away from t (more like DFS)
  - if $q > 1$, the walker is more likely to move inwards or towards t (more like BFS)



(a) $p = 1,\ q = 0.5$



(b) $p = 1,\ q = 2$

Linear Neural Networks

- Input Dataset (X): 100 grayscale images of size 32 × 32
- Final Objective:
  - To reduce the images to some kind of features to feed this information into a neural network
  - To perform binary classification of these images
- Task:
  - flatten out every image by placing every row of pixels linearly ⇛ A single image ≡ 1024 dimensional vector
  - X: A 100 × 1024 matrix
  - Output Y: A 100 × 2 matrix
  - To learn a weight matrix W such that multiplication of W with the input matrix X yield output matrix Y
  - However, W should not be dependent on the number of images

Linear Neural Networks

- $Y \in \mathbb{R}^{100\times 2}$, $X \in \mathbb{R}^{100\times 1024}$,
  - If $Y = W \times X \Longrightarrow$ No W can satisfy the equation
  - If $Y = X \times W \Longrightarrow W \in \mathbb{R}^{1024\times 2}$
- Weight matrix W can be thought of as a complete bipartite graph
  - number of nodes on one partition (partition 1) is equal to the number of features in our input
  - number of nodes on the other partition (partition 2) is the number of classes in our output
  - entry W_{ij} in W is the weight on the edge connecting node i of partition 1 to node j of partition 2
- Challenge: Learning the weights W_{ij} in W
  - several optimization techniques
  - Most common ones is Gradient Descent

Linear Neural Networks: Gradient Descent

- Measures the derivative of the optimization function for given inputs
- Determines the direction of maximum increase
- Take a “step” in the opposite direction, in order to minimize the objective function
- Objective Function: a quantitative measure of the error made by the network
- Popular Objective functions:
  - one-half mean squared error: $L = \frac{\sqrt{\sum_{i=1}^{m}(\widehat{y_i}-y_i)^2}}{2m}$
  - cross-entropy of predictions with the ground-truth: $L = \sum_{i=1}^{m} y_i \log \widehat{y_i} + (1 - y_i) \log(1 - \widehat{y_i})$

Linear Neural Networks: Backpropagation Algorithm

- Appears exactly like chain rule of derivatives
- Evenly distribute the error that the algorithm made amongst all the entries of the weight matrix
- updating the entries to reflect the "step" in the gradient descent
- if the error is measured by mean squared error function, then the error is:

$$\frac{\partial L}{\partial W_{ij}} = \frac{1}{m}\left(\sum_{i=1}^{m} \widehat{y_i} - y_i\right) \times \frac{\partial \widehat{y_i}}{\partial W_{ij}} = \frac{1}{m}\left(\sum_{i=1}^{m} \widehat{y_i} - y_i\right)$$

- subtract this error from W_{ij} to update it according to the gradient descent algorithm
- To protect our network against outlier data points, some $\alpha \in (0, 1]$ fraction of this error is used for update

$$W_{ij} = W_{ij} - \alpha \frac{\partial L}{\partial W_{ij}}$$

Convolutional Neural Networks

- Use a convolution operator on the images to aggregate information from several surrounding pixels together in a single pixel
- Done via a learnt kernel matrix
- Moved over the original image and corresponding elements multiplied in order to obtain one element of the output matrix
- Challenge: To learn the convolutional kernel matrix

Convolution on Graphs

- Challenge in performing convolution on a graph
  - Graph data is not as structured as the image data
  - Perturbing the adjacency matrix of a graph (relabeling the nodes) will not change the graph structure
  - No fixed label for an individual node in a graph

- Convolution operation in GCN is supposed to
  - Transform neighbourhood information
  - Aggregate this information to form output

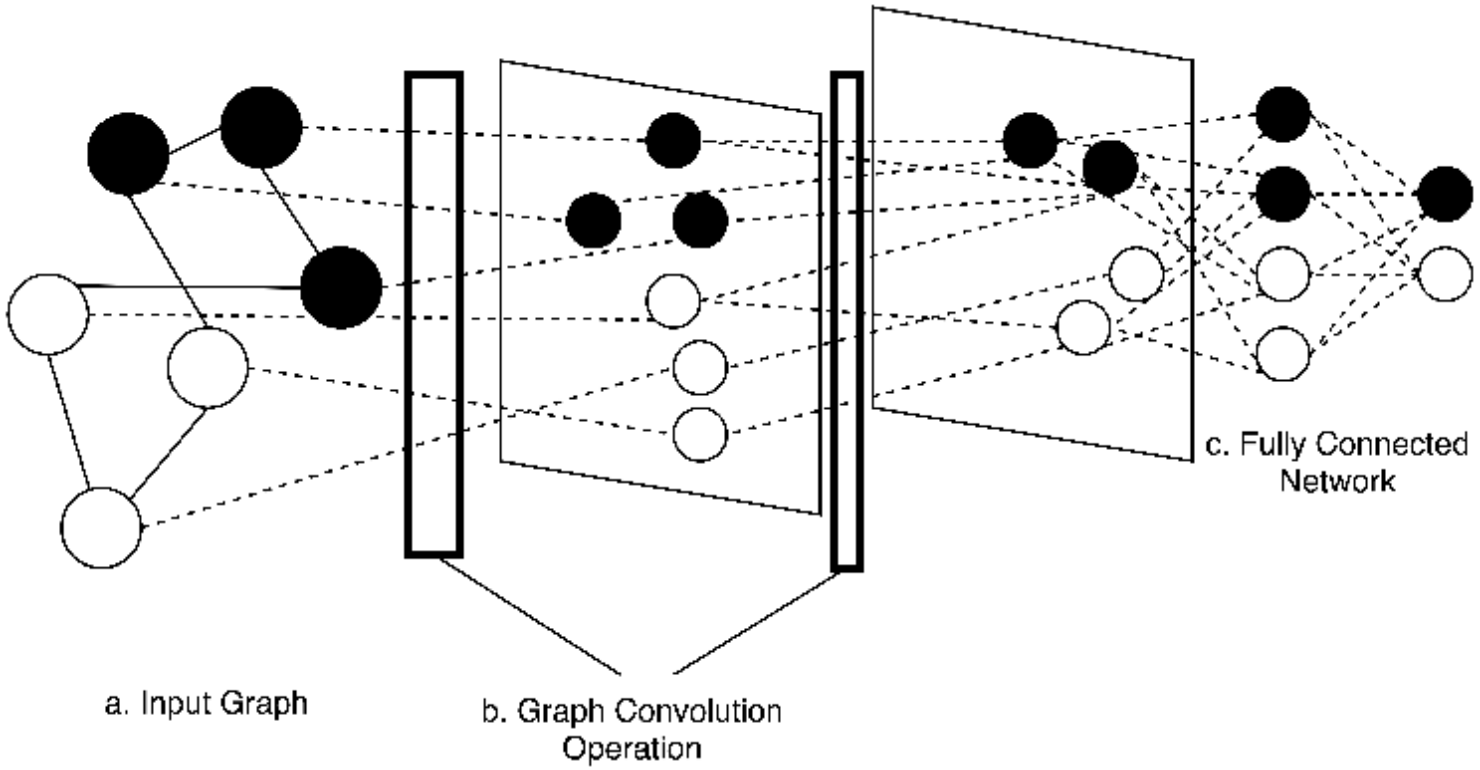
Spectral Graph Convolution

- Proposed by Bruna et al. in 2013
- Spectrum of a graph refers to the eigenvalues of its adjacency matrix
- A graph Laplacian is the adjacency matrix normalized with a diagonal matrix consisting of its eigenvalues
- Spectral graph convolution is to learn a kernel matrix for this graph Laplacian
- It is invariant to the positioning of nodes in adjacency matrix
- Apply normal CNNs on the Laplacian matrix as if it is an image pixel matrix

Graph Convolutional Network

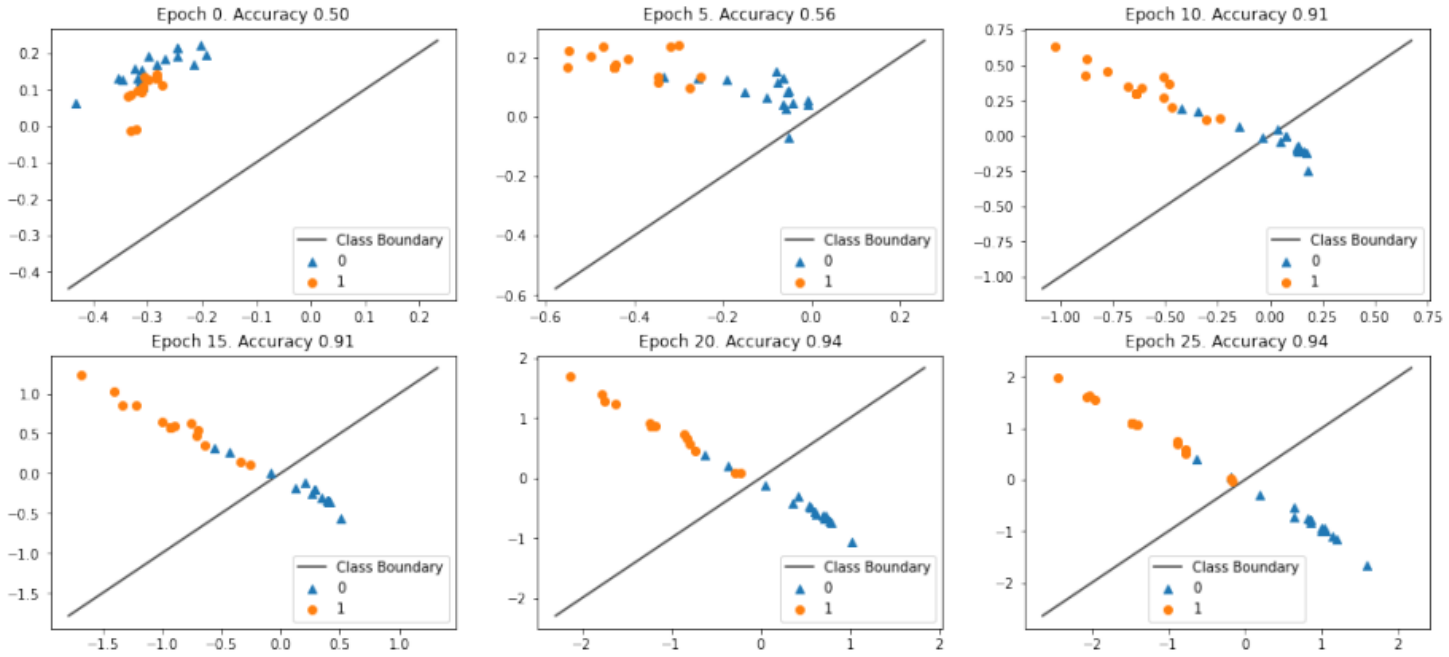
- For every node do
  - Send its own embedding to all its neighbors
  - Receive its neighborhood embeddings and aggregate them
- Embedding of a node is like a message, and hence the method resembles a message passing paradigm
- A formulation for GCN:

$$h_v^k = W^{k-1} \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + h_v^{k-1}$$



Here h_v^0 are essentially the node features from any given graph

Graph Convolutional Network: Case Study



Training GCN on the karate club network that ended up splitting into two factions

- start with random embeddings for all nodes on the karate club graph
- use a two-layer GCN with no fully connected network, with the output embeddings of dimension two
- embeddings slowly separate out on the division line

Variations of GCN: Relational GCN (R-GCN)

- key idea in R-GCN is to learn different matrices for different kinds of edges
- knowledge graphs are heterogeneous triples of the form (A, R, B), where R is a relation from entity A to entity B
- These relations are non-homogeneous
- need to learn different weight matrices for different relations
- R-GCNs follows the following equation:

$$h_v^k = \sum_{r \in R} \sum_{u \in N^r(v)} W_r \frac{h_u^{k-1}}{|N(v)|} + h_v^{k-1}$$

Variations of GCN: Relational GCN (R-GCN)

- Types of edges can be extremely large in number
  - cause a need for a large number of parameters to be learnt
- Instead of learning W_r, the method learns the basis vectors V_b of W_r
- Perform linear combination with different weights and obtain different projection weights W_r at any given recursion step k
- The revised formulation:

$$W_r^k = \sum_{b=1}^{B} a_{rb}^k V_b^k$$

Variations of GCN: Graph Attention Network

- In many cases, it is better to give more weight to, say, more "influential" nodes
  - importance is often called as attention
  - not possible with vanilla GCN
  - In a plain GCN, we give equal weight to all the neighbors of any given node
- The revised formulation:

$$h_v^k = \sum_{u \in N(v)} \alpha_{uv}^{k-1} W^{k-1} \frac{h_u^{k-1}}{|N(v)|} + h_v^{k-1}$$

- Additional Challenge: Learning the attention α

Variations of GCN: Graph Attention Network

- Most commonly used way of learning attention in graphs is:

$$z_u^{k-1} = W_u^{k-1} h_u^{k-1}$$

$$e_{uv}^{k-1} = Nonlinearity\left(\left(\alpha^{k-1}\right)^T \times [z_u^{k-1} z_v^{k-1}]\right)$$

$$\alpha_{uv}^{k-1} = softmax\left(e_{uv}^{k-1}\right) = \frac{exp\left(e_{uv}^{k-1}\right)}{\sum_{u \in N(v)} exp\left(e_{uv}^{k-1}\right)}$$

GraphSAGE

- Develops representation for dynamic graphs, using a paradigm called inductive learning
- Capable of predicting embedding of a new node, without requiring a re-training procedure
- GraphSAGE learns aggregator functions that can induce the embedding of a new node given its features and neighborhood
- Can be divided into two major components:
  - Context construction: assumes that nodes which belong to the same neighborhood should have similar embeddings
  - Information aggregation: apply a weighted combination on each neighbor's embeddings
- As GraphSAGE learns the aggregators rather than the node embeddings itself, it is able to generate the embeddings of 'unseen' nodes from the features derived from its neighborhood
- A learnable aggregator is a single neural network layer, followed by a maxpooling operator

END