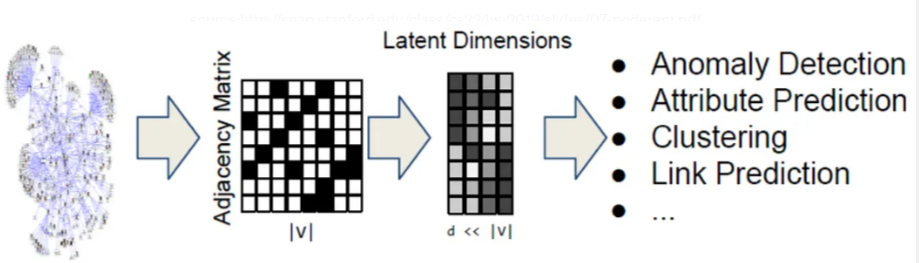
Graph Representation Learning

# Why Graph Representation Learning?

The deep learning models for computer vision work on images, NLP on text data, and predictive analytics on tabular data; all are designed for simple sequences or fixed-size grids.

The main challenge for machine learning on graphs is to find an easy, efficient and reliable task-independent feature learning for machine learning with graphs that can incorporate the graph structure into an ML model.

For example, in the case of Item recommendations using link predictions on graphs, you need to find a way to encode pairwise properties between Items and users and the strength of their relationship.

Graph are complex, irregular, non-Euclidean topographical structers

# Graph Representation Learning

Graph representations learning addresses the challenges with graphs by mapping the nodes or entire graphs or subgraphs as points in a low dimensional vector space.

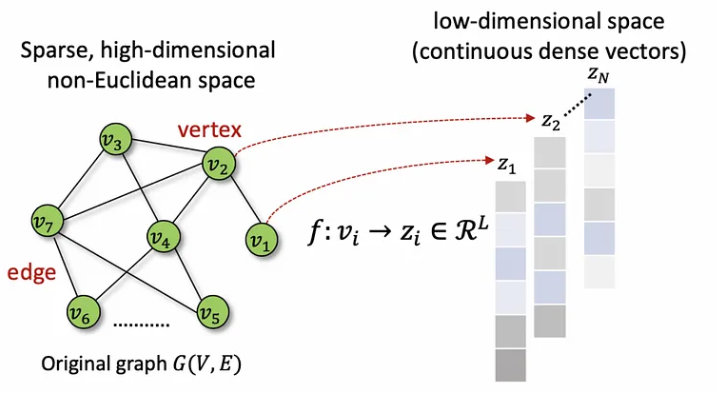
The goal of Graph Representational Learning is to optimize the graph embedding mapping so that geometric relationships in this learned space reflect the structure of original graph

The graph representation learning approach treats the problem of capturing structural information about the graph as a machine learning task and not as a pre-processing step.

The graph representational learning needs to address

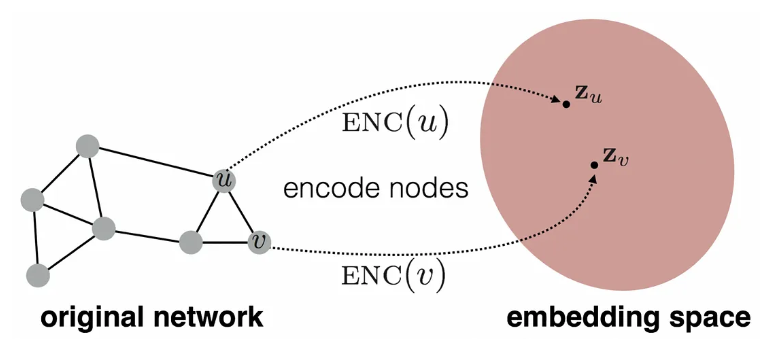
* Graph Structural Property Preservation
* Node Similarity Measurement in both the original and latent space
* An Encoder

# Node Embeddings

The goal of node embedding is to learn an encoder that maps the graph nodes to a low dimensional dense space such that the similarity in the embedding space approximates the similarity in the original graph.

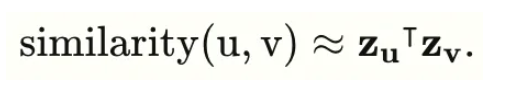
The node embeddings are optimized such that the nodes in the embedding space reflect the relative positions of the nodes in the original graph.

Embeddings compress the graph into compact data structures and hence become easy to compare in real-time using low-cost parallel computing.

To learn the node embeddings following things are required.

1. Define the encoder (ENC): A mapping from the nodes to the embedding. A shallow encoder is the simplest encoder that maps each node to a low dimensional vector using an embedding lookup where each node is assigned a unique embedding vector.
2. Define the node similarity function: Specifies how the relationship in vector space maps to the relationship in the original graph. The similarity in nodes can be characterized by

* Presence of an Edge
* Overlapped neighborhood
* Reachable by k-hops
* Reachable through random walks
* Similar node attributes

1. Optimized the parameters of the encoder such that the similarity of u and v in the network approximates the dot product between node embedding.

Deep Walk and Node2Vec are the two popular node embedding algorithms based on random walks performed on the graph.

The graph embedding methods are classified into 3 major categories:

* Matrix factorization-based methods
* Random walk-based methods
* Neural network-based methods

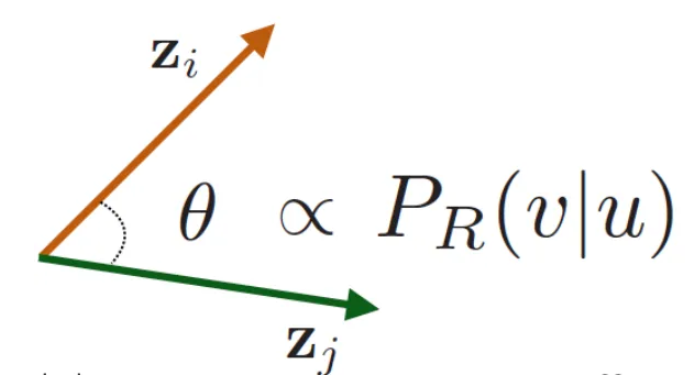
# Random walks

A Random walk is a sequence of nodes selected randomly during an iterative process.

If you play GeoGuessr, a geography game where you find yourself dropped randomly in any part of the world, you need to recognize your surroundings to know which part of your world. You randomly walk from one neighborhood to another using a random strategy like a Random Walk.

A Random walk uses a random walk strategy R where a random walker moves through the graph starting from node u and visiting node v after a fixed number of steps k.

Note: Random walks are expressive as they incorporate local and higher-order and neighborhood information. Random Walks are also efficient as they do not consider all node pairs as they only need to consider pairs that co-occur on random walks.

Embeddings using Random walk estimate the probability by picking a node randomly, starting from a node u using k steps to reach the node v to optimize embedding. The encoding is optimized such that the cosine of the angle between the two vectors is proportional to or similar to the dot product between the vector u and v.

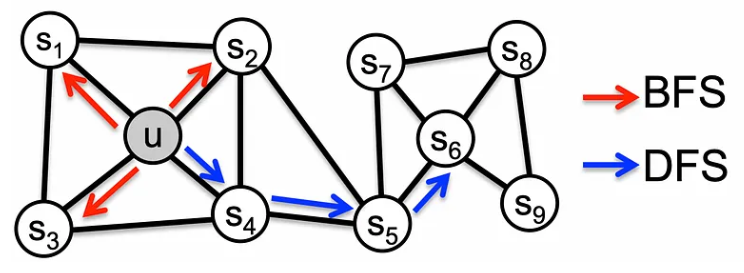
The Random Walk strategy helps derive node embedding such that nearby nodes are close together in the network.

Node2Vec uses flexible, biased random walks to trade-off between local and global views of the network using BFS (Breadth First Strategy) and DFS (Depth First Strategy).

# Breadth-First Strategy --- BFS

* Finds the shortest path in the graph to provide the local view of the neighborhood.
* BFS explores and examines the edges of the graph to find every node reachable from the source.

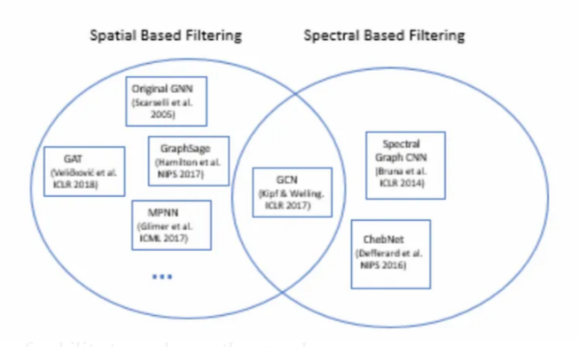
# Depth First Strategy --- DFS

* Provides the global view of the neighborhood by traversing through more edges to reach a destination node from the source node.
* Explores large network parts by moving further away from the source node.

Deep Walk is another approach for learning latent representation of Nodes in a graph. The Deep Walk Algorithm consists of 2 main components.

* A random walk generator: Takes a graph G and samples uniformly from the neighbors of the last node visited until the maximum length t is reached to generate a representation.
* An update procedure: SkipGram algorithm is used to update representations obtained from node pairs of weighted combinations due to the random walk generator.

# Graph Embedding

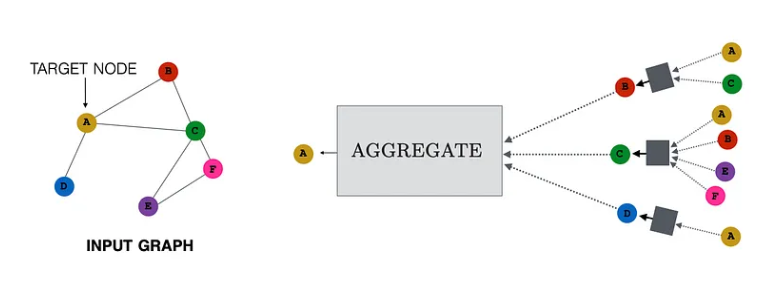
The Graph Embedding goal is to encode a set of nodes and edges into a low dimensional vector embedding. This technique can embed both subgraphs as well as full graphs.

The graph embedding approaches are used for subgraph classification, where the goal is to predict a label associated with a particular subgraph.

Note: Spectral based graph filtering is based on an Eigen decomposition of the Laplacian Matrix of the graph allowing the graph Fourier transformation. Graph signal processing is a spectral based graph filtering that extracts information based on frequency.

Note: Spatial graph filtering is simple and works on the graph topology, by aggregating local node neighborhood information. They are preferred due to less computational complexity, their localized property, and their transferability to apply on other graphs.

# Graph Neural Network (GNN)

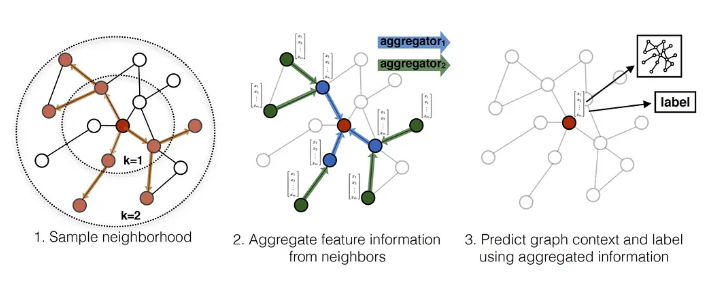
The basic intuition behind the GNN message-passing framework is a two-key step approach: aggregation and updating at each iteration. Every node aggregates information from its local neighborhood and then updates the aggregated information in the graph to define a computation graph. As the iterations progress, each node embedding contains more and more information from further reaches of the graph.

# Graph Convolutional Neural Network (GCN)

A diagram of a device

Description automatically generatedThe Graph Convolutional Neural Network (GCN) model is a generalization of convolutions to non-Euclidean data. GCN generalizes the operation of convolution from grid data to graph data. GCN is based on graph convolutions built by stacking multiple convolutional layers, and a point-wise non-linearity function follows each layer.

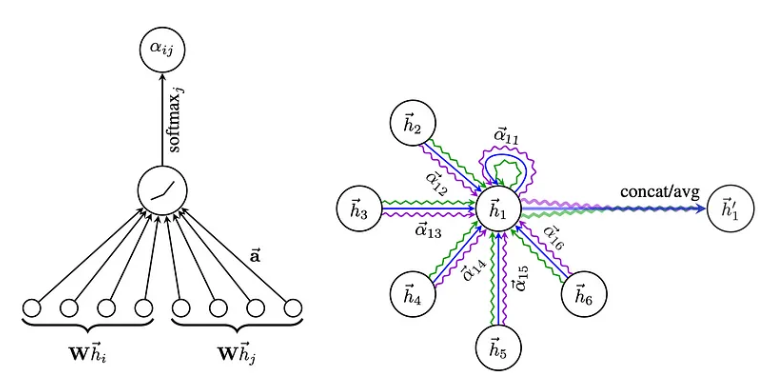
GraphSAGE

The GraphSAGE is a general inductive framework that leverage node attribute information like text attributes, node profile information, and node degrees to efficiently generate representations for previously unseen data. GraphSAGE is trained on an example graph or set of graphs. Once the training of GraphSAGE is done, the model can be used to generate node embeddings on unseen nodes or new input graphs as long as the unseen nodes or graphs have the same attribute schema as the training data.

GraphSAGE learns to aggregate feature information from a node’s local neighborhood at training time. Each aggregator function aggregates information from a different number of hops, or search depth, aways from a given node. GraphSAGE generates embeddings for nodes at test or inference time, assuming that the GraphSAGE model parameters are already learned.

# Graph Attention Network (GAT)

GAT is an inductive learning strategy that operates on graph-structed data by leveraging masked self-attentional layers. GAT is based on the idea of computing the hidden representations of each node in the graph by attending over its neighbors using a self-attention strategy.

The attention mechanism utilized throughout GAT is computationally efficient and parallelizable across all graph nodes. GAT allows for implicitly assigning different importance to different nodes within a neighborhood. It deals with different-size neighborhoods and does not depend on knowing the entire graph structure upfront.

The critical difference between GAT and GCN is how the information from the one-hop neighborhood is aggregated.

Conclusion:

Node or Graph embeddings aim to represent a graph into a low-dimensional vector representation while preserving both network topology structure and node content information to perform graph analytics tasks such as classification, clustering, and recommendations.