GraphSAGE : Inductive Representation Learning on Large Graphs

Mar 23, Jeong Hyun Jae, tAILab Journal Club

Hyunjae Jeong (tAILab, CCIDS)

Yonsei University, Medical Life Systems Information Center (TAIL Lab)
Severance Hospital, Center for Clinical Imaging Data Science (CCIDS)
Severance Hospital, Radiology

CCIDS 의료영상데이터사이언스센터 Center for Clinical Imaging Data Science

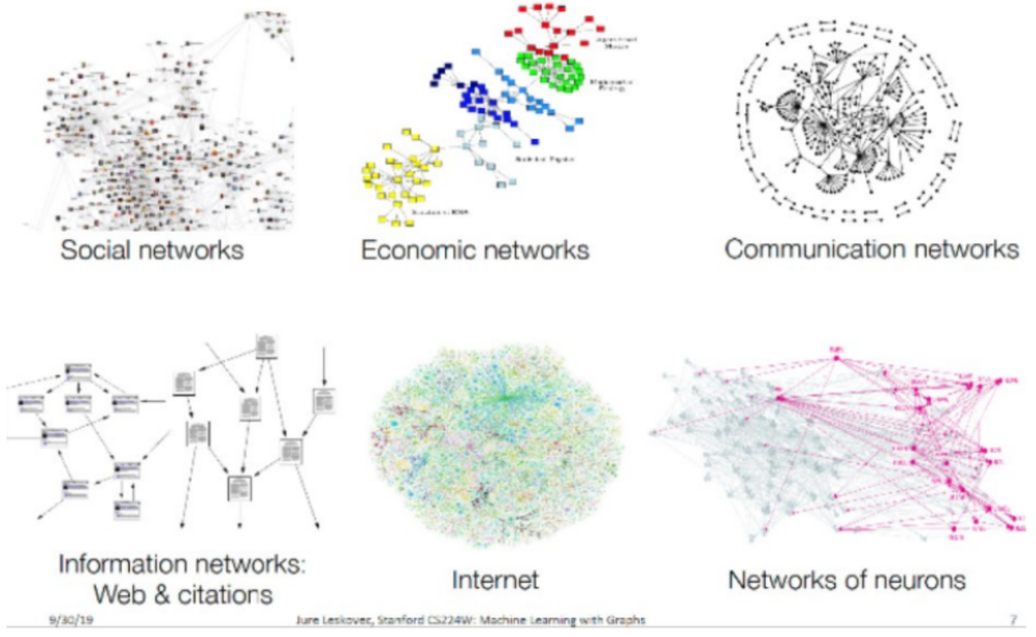
tAILab.

01 Introduction

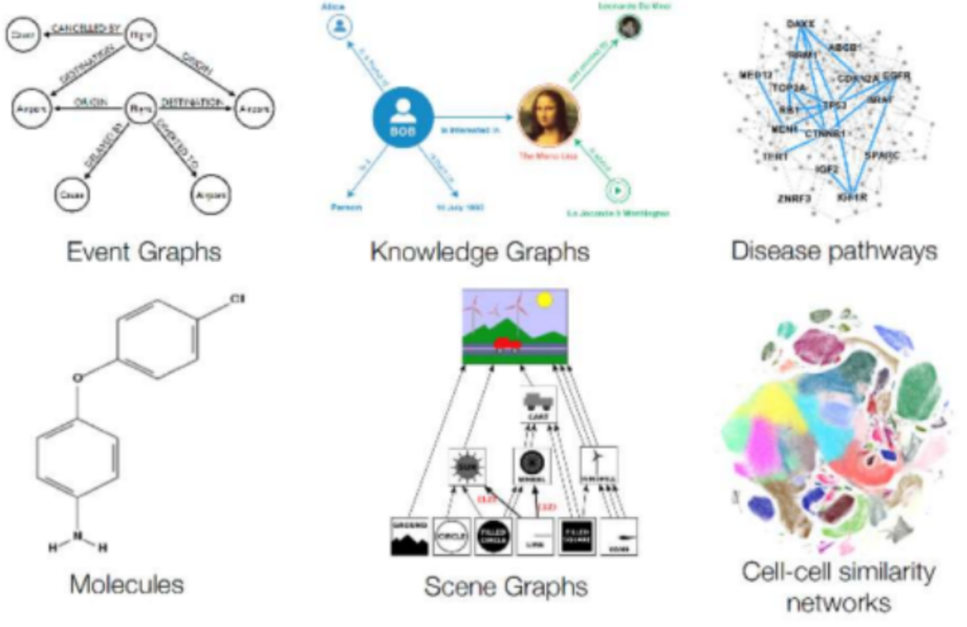
tAILab.

Why Graphs?

- A graph is a general purpose for describing a complex system of interactions between objects. it is language In the current deep learning, data such as image and text/speech are simple. It is designed with sequences/grids.
- But all information, It cannot be expressed in sequences/grids, and the relationship between multiple data Through modeling, the performance for accurate prediction can be improved.



Networks (Natural Graphs)

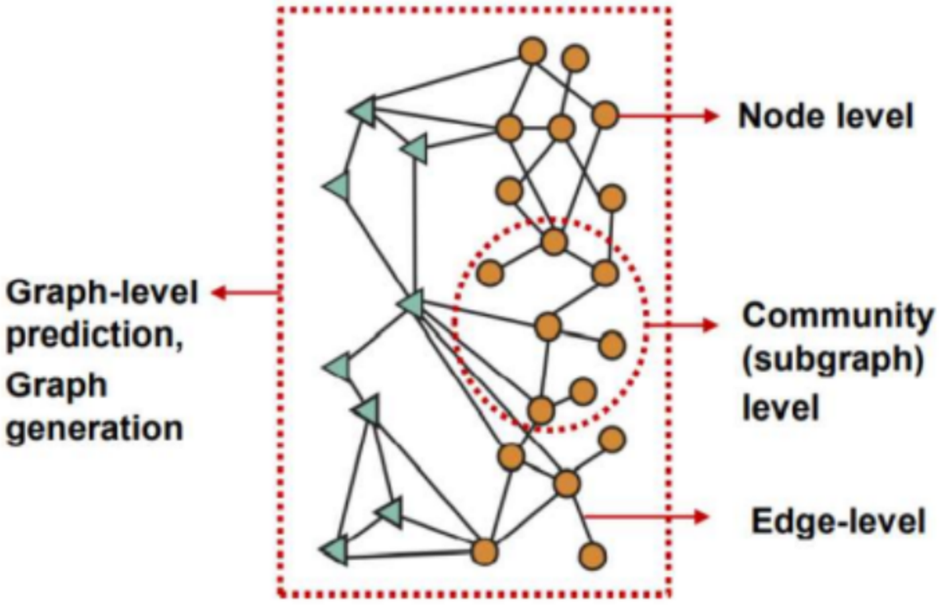


Graphs (Representations)

tAILab.

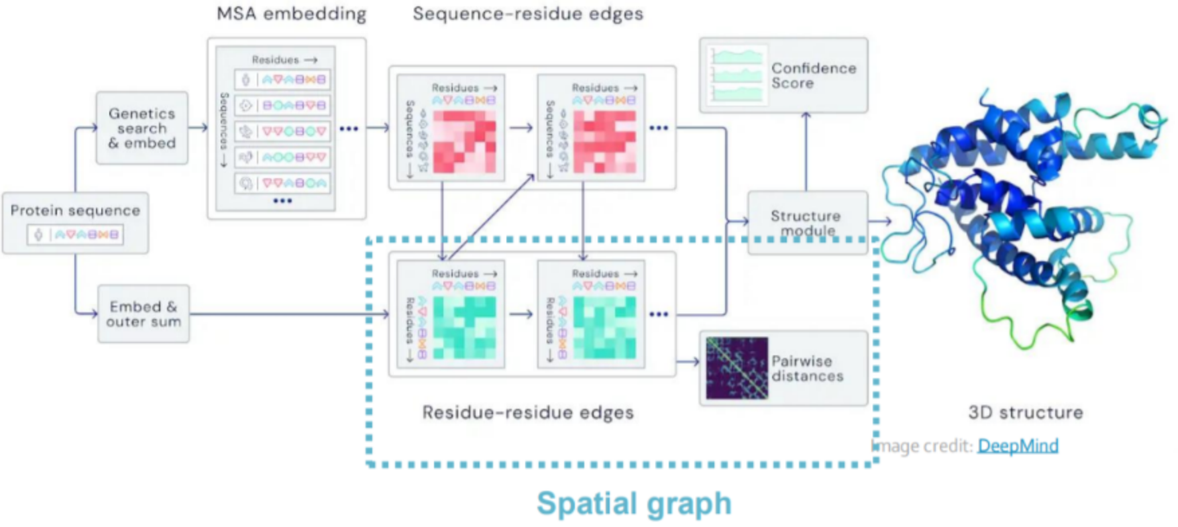
Applications of Graphs ML

- Research using graphs can be done at various levels of graph construction. From the smallest node/edge level to the subgraph level, the entire graph-level Prediction and generation are used.
- Node classification, Link predictions, Graph classification, Clustering, Graph generation, Graph evolution



Graph Learning Pipeline

- **Nodes:** Amino acids in a protein sequence
- **Edges:** Proximity between amino acids (residues)



Deepminds' AlphaFolds Model

tAILab.

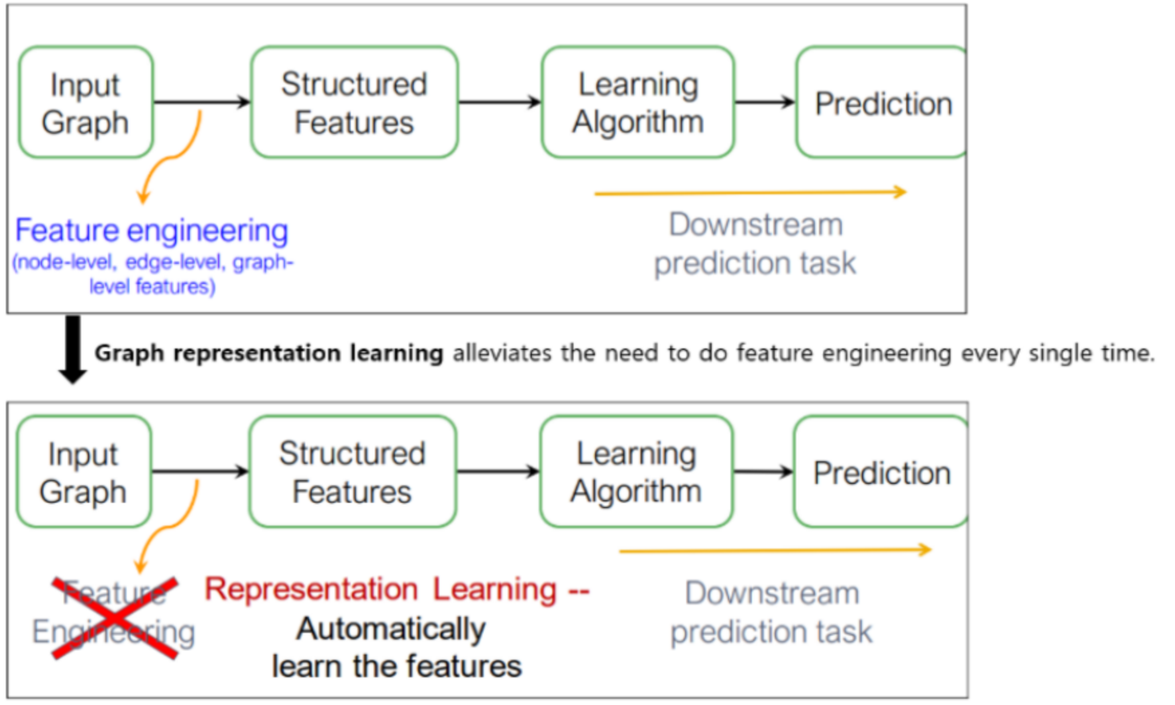
Graph Representation Learning

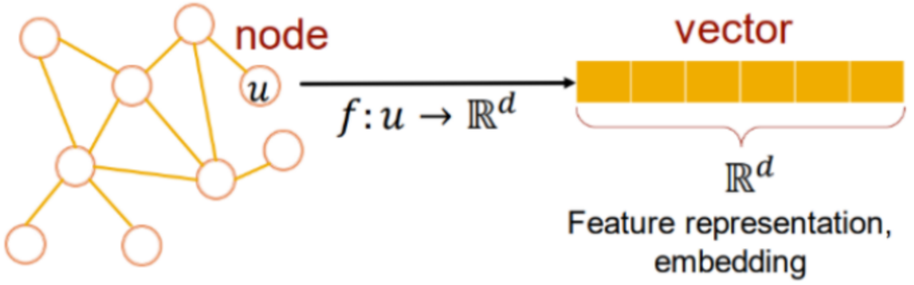
- Elements of Graphs (node, link, graph)

↓ Traslation : $f: u \rightarrow \mathbb{R}^d$

Convert to d-dimensional vector
(Embedding space)

- Goal: Efficient task-independent feature learning for machine learning with graphs
  - Efficient: Reduce the time required for feature engineering.
  - Task-independent: Create a pipeline for automatic feature e xtraction

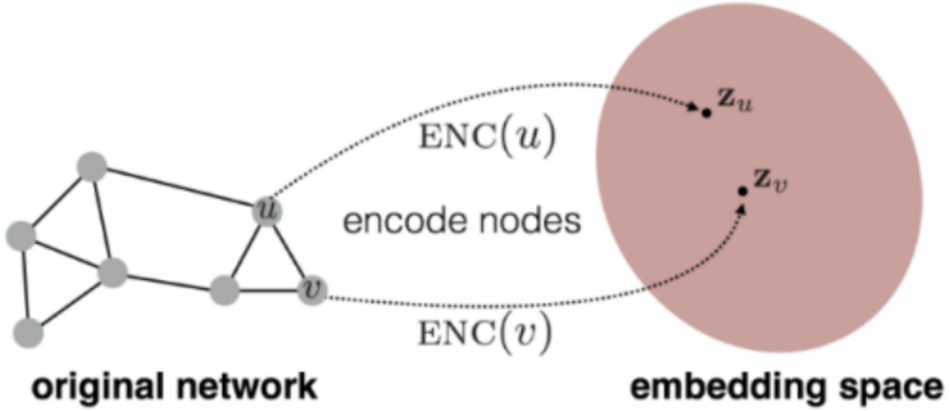




Graph Representation Learning

Node Embedding

- The basic idea of node embedding is to use a dimensionality reduction technique that makes high-dimensional in formation about a node's graph neighbors into dense vector embeddings.
- Such node embedding can be applied to downstream machine learning systems and can help with problems such as node classification, clustering, and link prediction.



- $ENC(v) = \mathbf{z}_v$

  v: node in the input graph

  $\mathbf{z}_v$: d-dimensional embedding

  Node Embedding

- Goal is to encode nodes so that similarity in the embeddin g space approximates similarity in the graph

Node Embedding Framework

1. The encoder maps the nodes to the embedding space and generate

2. We define a node similarity function, and on the original network Measure the similarity between nodes

3. Decoder measures similarity of mapped vectors

4. Optimize the parameters of the encoder so that the similarity meas ured in (2) and (3) is similar.

Problems of Graph Learnings

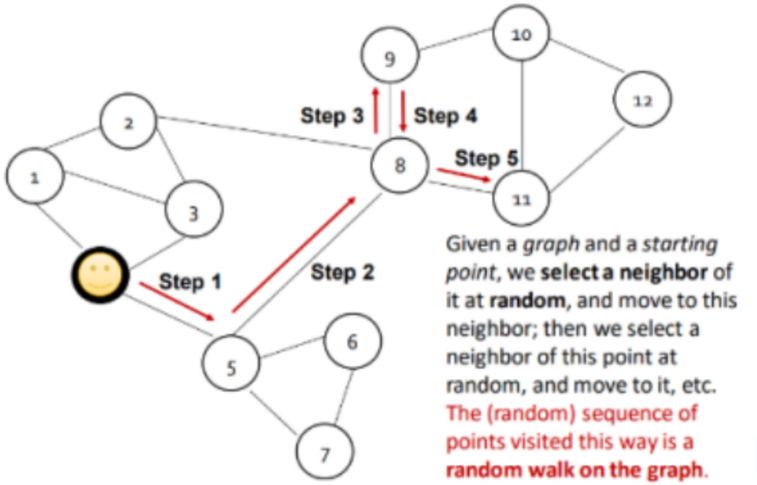
- However, previous works have focused on embedding nodes from a single fixed graph, and many real-world applicat ions require embeddings to be quickly generated for unseen nodes, or entirely new (sub)graphs - Transductive
- This inductive capability is essential for high-throughput, production machine learning systems, which operate on evolving graphs and constantly encounter unseen nodes - Inductive

↓

- The inductive node embedding problem is especially difficult, compared to the transductive setting, because g eneralizing to unseen nodes requires “aligning” newly observed subgraphs to the node embeddings.
- We propose a general framework, called GraphSAGE (SAmple and aggreGatE), for inductive node embedding

Related Work

- 1. Factorization-based embedding approach: Learn low-dimensional embeddings using random walk. statistics and matrix factorization-based learning objectives.



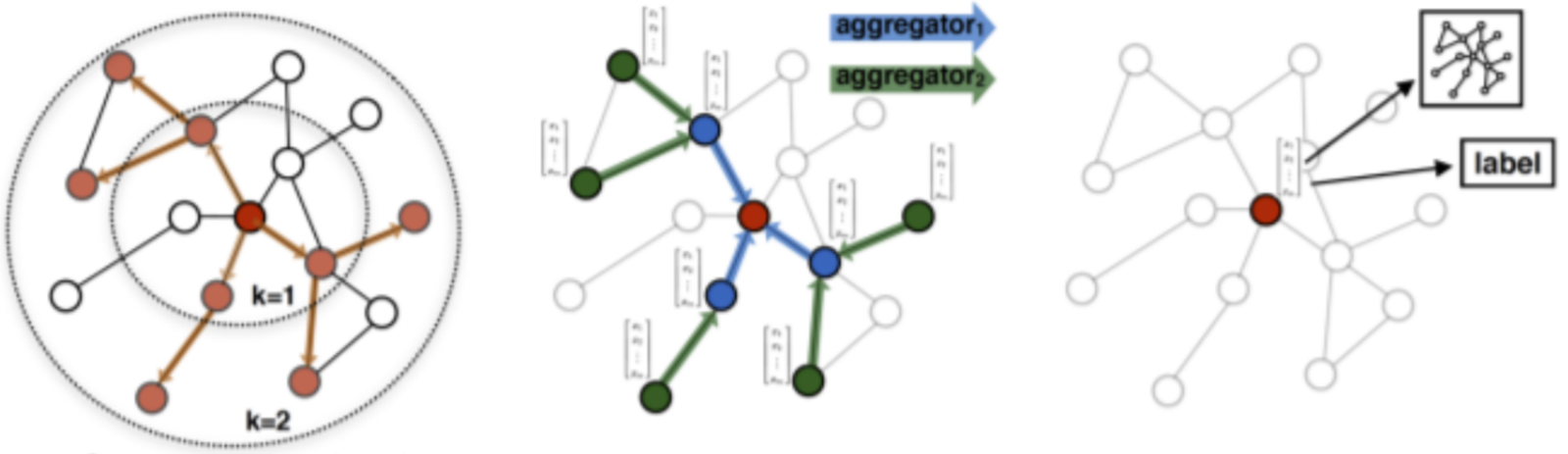
Random Walk

- 2. Supervised learning over graphs: includes a wide variety of kernel-based approaches, where feature vectors for g raphs are derived from various graph kernels.

- 3. Graph Convolutional Network : The original GCN algorithm is designed for semi-supervised learning in a transductive setting, and the exact algorithm requires that the full graph Laplacian is known during training

Proposed method: GraphSAGE

- The key idea behind our approach is that we learn how to aggregate feature information from a node's local neighborhood (SAmple + aggreGatE)
- The detailed architecture of the generator in network. In each of the block, the three number means number of input channels, number of output channels and the image size.



Visual illustration of the GraphSAGE sample and aggregate approach.

tAILab.

Embedding generation (i.e., forward propagation) algorithm

- Assumes that the model has already been trained and that the parameters are fixed.

- There are two types of hyperparameters : 1. $AGGREGATE_k, \forall k \in 1, \ldots, K$
  2. $\mathbf{W}^k, \forall k \in 1, \ldots, K$

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

$AGGREGATE_k, \forall k \in 1, \ldots, K$

Aggregator Function - This function is responsible for integrating information from Node neighbors.

$\mathbf{W}^k, \forall k \in 1, \ldots, K$

Weight Matrices - They are used to pass information between different layers or search depths of the model.

tAILab.

Subsection : Relation to the Weisfeiler-Lehman Isomorphism Test

- Proposed Algorithm can be thought of as an instance of the Weisfeiler-Lehman: WL Isomorphism Test called N aive Vertex Refinement, if set to , the Weight Matrices as the identity matrix, and an appropriate non-linear Ha sh function is used as the aggregator.
- Weisfiler-Lehman Isomorphism Test (WL Isomorphism Test) : Algorithm created to test Graph Isomorphism

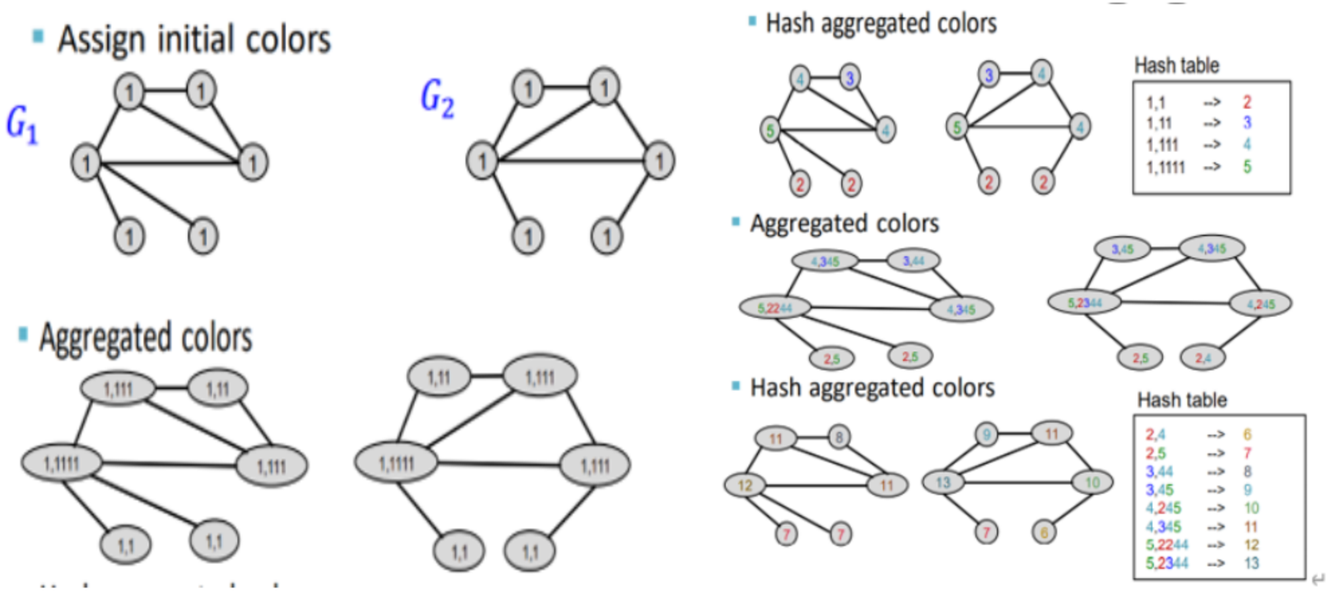
$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right)$$

Neighborhood Definition

In this work, we uniformly sample a fixed-size set of ne ighbors, instead of using full neighborhood sets in Algo rithm 1, in order to keep the computational footprint o f each batch 4 fixed.

$$\mathcal{N}(v) \;\; u \in \mathcal{V} : (u, v) \in \mathcal{E}$$

It can be defined as a sample drawn uniformly wi th a fixed size at iteration k.



Learning the Paremeters of GraphSAGE

- In order to learn useful and predictive representations in a completely unsupervised learning situation, we appl y the Graph-based Loss function to the Output Representation
- We need to tune the parameters of the Aggregator Funciton through Stochastic Gradient Descent and Weight Metrics. Graph-based Loss function makes adjacent nodes have similar representations and distant nodes hav e different representations.

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

v = Nodes occurring at the same time near u
in a fixed-length random walk
Pn = Negative Sampling Distribution
Q = Number of Negative Sample

Aggregator Architectures

- Unlike machine learning over N-D lattices (e.g., sentences, images, or 3-D volumes), a node's neighbors have n o natural ordering; thus, the aggregator functions in Algorithm 1 must operate over an unordered set of vector
- Ideally, an aggregator function would be symmetric (i.e., invariant to permutations of its inputs) while still being trainable and maintaining high representational capacity. There are 3 candidates

Mean aggregator

- Our first candidate aggregator function is the mean operator, where we simply take the elementwise mean of t he vectors $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$
- We call this modified mean-based aggregator convolutional since it is a rough, linear approximation of a localized spectral convoluti

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}).$$

Aggregator Architectures

LSTM aggregator

- Compared to the mean aggregator, LSTMs have the advantage of larger expressive capability. However, it is important to note that LSTMs are not inherently symmetric (i.e., they are not permutation invaria nt), since they process their inputs in a sequential manner.

Pooling aggregator

- The final aggregator we examine is both symmetric and trainable. In this pooling approach, each neighbor's vec tor is independently fed through a fully-connected neural network
- The function applied before the max pooling can be an arbitrarily deep multi-layer perceptron, but we focus on simple single-layer architectures in this work

$$AGGREGATE_k^{pool} = max(\{\sigma(\mathbf{W}_{pool}\mathbf{h}_{u_i}^k + \mathbf{b})\})$$

$$\forall u_i \in \mathcal{N}(v)$$

03 Experiments

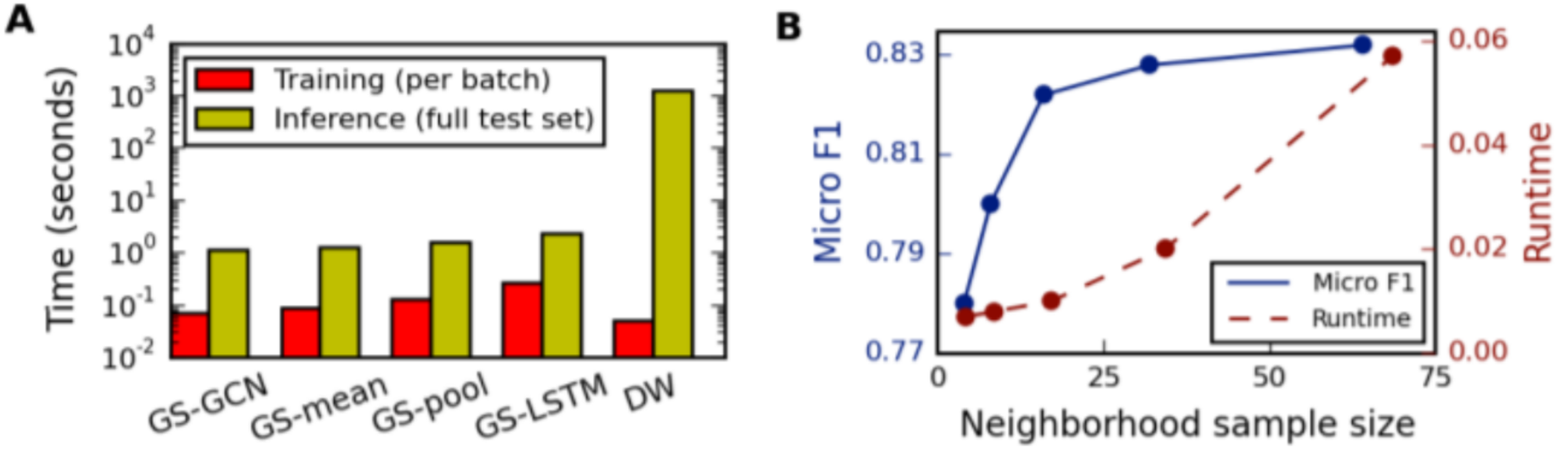
Experimental Set Up

- Classifying academic papers into different subjects using the Web of Science citation datase
- Classifying Reddit posts as belonging to different communities
- Classifying protein functions across various biological protein-protein interaction (PPI) graphs
- Compare against four baselines: a random classifer, a logistic regression feature-based classifier (that ignores gr aph structure), the DeepWalk algorithm as a representative factorization-based approach, and a concatenation o f the raw features and DeepWalk embeddings

03 Experiments

Experiments

Name	Citation		Reddit		PPI	
	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%



Conclusion

- Overall, we found that the LSTM- and pool-based aggregators performed the best, in terms of both average performance and number of experimental settings where they were the top-performing method
- GraphSAGE consistently outperforms state-of-the-art baselines, effectively trades off performance and runtime by sampling node neighborhoods, and our theoretical analysis provides insight into how our approach can learn about local graph structures.
- A particularly interesting direction for future work is exploring non-uniform neighborhood sampling functions, and perhaps even learning these functions as part of the GraphSAGE optimization.