SIMGNN

Similarity Computation via Graph Neural Networks

Introduction

- Graph similarity search is among the most important graph-based applications, e.g. finding the chemical compounds that are most similar to a query compound

- Similarity Metrics :
  GED - Graph edit distance (Number of edit operations required to convert one graph to other)
  MCS - Maximum common subgraph

Till now the problem is addressed using :

1. Pruning verification framework
2. Approximation using fast and heuristic ways.

SimGNN address it as a **learning problem**

Goals:

Similarity computation which should be :

Representation Invariant : even different ways or representation should give same results.

Inductive : Should also work on unseen graph

Learnable : Adjust to any similarity metric like MCS or Ged

Approach as a learning problem

- Step 1 : Find embeddings using GCN
- Step 2 : Two strategies
  - Graph level embeddings
    - Global context aware attention
    - Finding similarity between two embeddings using Neural tensor networks
  - Pairwise node comparison
- Step 3 : Concat the output of both and feed into Fully connected layer

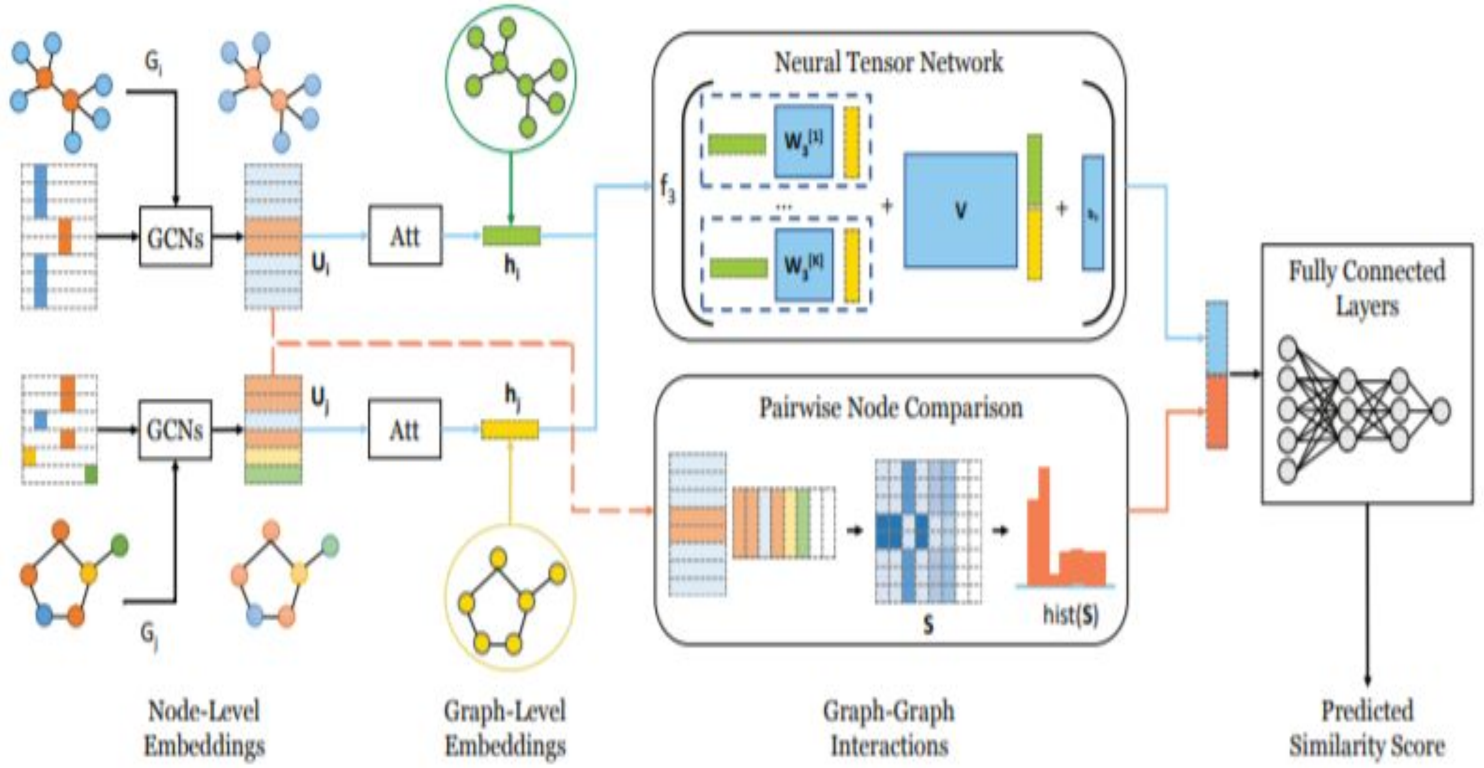


Figure 3: An overview illustration of SimGNN. The blue arrows denote the data flow for Strategy 1, which is based on graph-level embeddings. The red arrows denote the data flow for Strategy 2, which is based on pairwise node comparison.

Graph Conv. Network

GCN is used for finding the embeddings
Inputs :

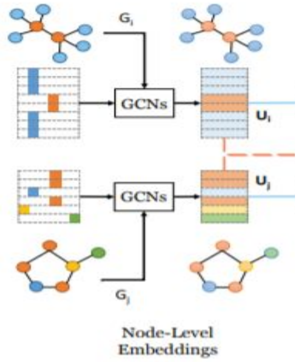
- Nodes features
- Adjacency matrix

Every node have a state H(l) and number of updates of this state depends upon hyperparameter given.

Graph level embedding

Node embeddings using GCN

- Inputs one-hot encodings of nodes that represents node type.
- Inputs graph as adjacency matrix for message flow to neighbours
- Uses two graphs for later comparing the similarity



Attention mechanism

Attention mechanism is used to give heavy weights to some nodes while less to others

$$\boldsymbol{h} = \sum_{n=1}^{N} f_2(\boldsymbol{u}_n^T \boldsymbol{c})\boldsymbol{u}_n = \sum_{n=1}^{N} f_2(\boldsymbol{u}_n^T \tanh(\frac{1}{N} W_2 \sum_{m=1}^{N} \boldsymbol{u}_m))\boldsymbol{u}_n$$

Here **h** is the final graph embedding

Decomposing Graph embedding

- We initially have **n x d** node embeddings , where each row is embedding for a node call it **u** with dimension as (1 X d)
- Now, we calculate a vector **c** , graph context as

C = tanh(f(W2,u)) , W2 is d x d learnable parameter

$$\tanh\left(\frac{1}{N}\boldsymbol{W_2}\sum_{m=1}^{N}\boldsymbol{u_m}\right.$$

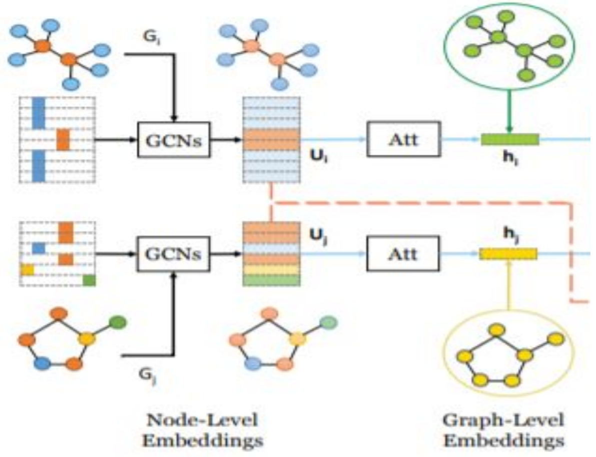
- Now attention , a(n) is calculated as inner product of **c** with **u** and feeding it to a non-linearising activation i.e sigmoid.
-

$$f_2\left(\boldsymbol{u_n^T}\tanh\left(\frac{1}{N}\boldsymbol{W_2}\sum_{m=1}^{N}\boldsymbol{u_m}\right)\right)$$

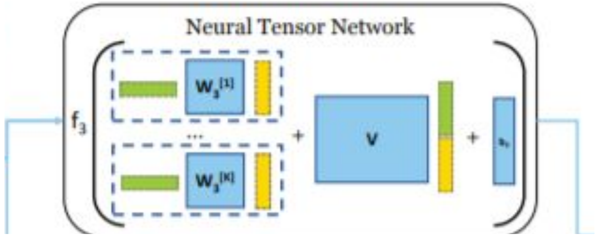
Decomposing Graph embedding

Finally, sum(a(n)*u(n)) to get graph embedding gives :

$$\boldsymbol{h} = \sum_{n=1}^{N} f_2(\boldsymbol{u}_n^T \boldsymbol{c})\boldsymbol{u}_n = \sum_{n=1}^{N} f_2(\boldsymbol{u}_n^T \tanh(\frac{1}{N}\boldsymbol{W}_2 \sum_{m=1}^{N} \boldsymbol{u}_m))\boldsymbol{u}_n$$

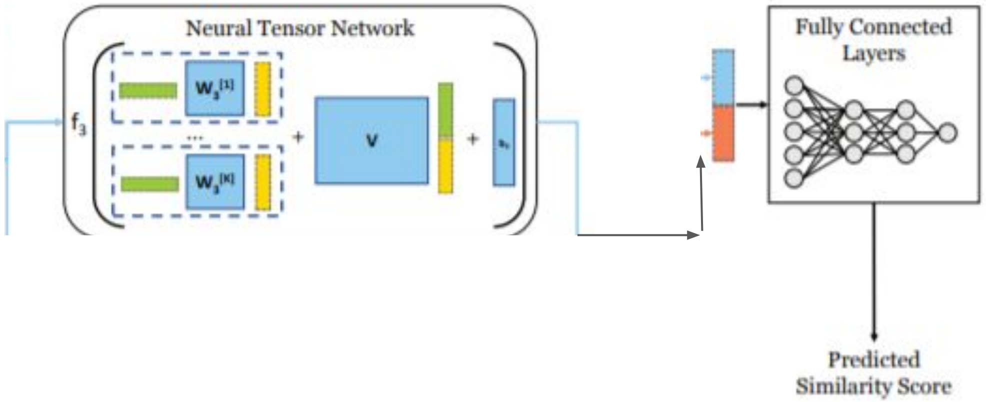


Neural Tensor Network to find relationship



The graph embeddings of 2 graphs are input to NTN to get a relation vector among them

Fully connected layer



Finally this relation vector is fed into a fully Connected layer to get the predicted similarity score.

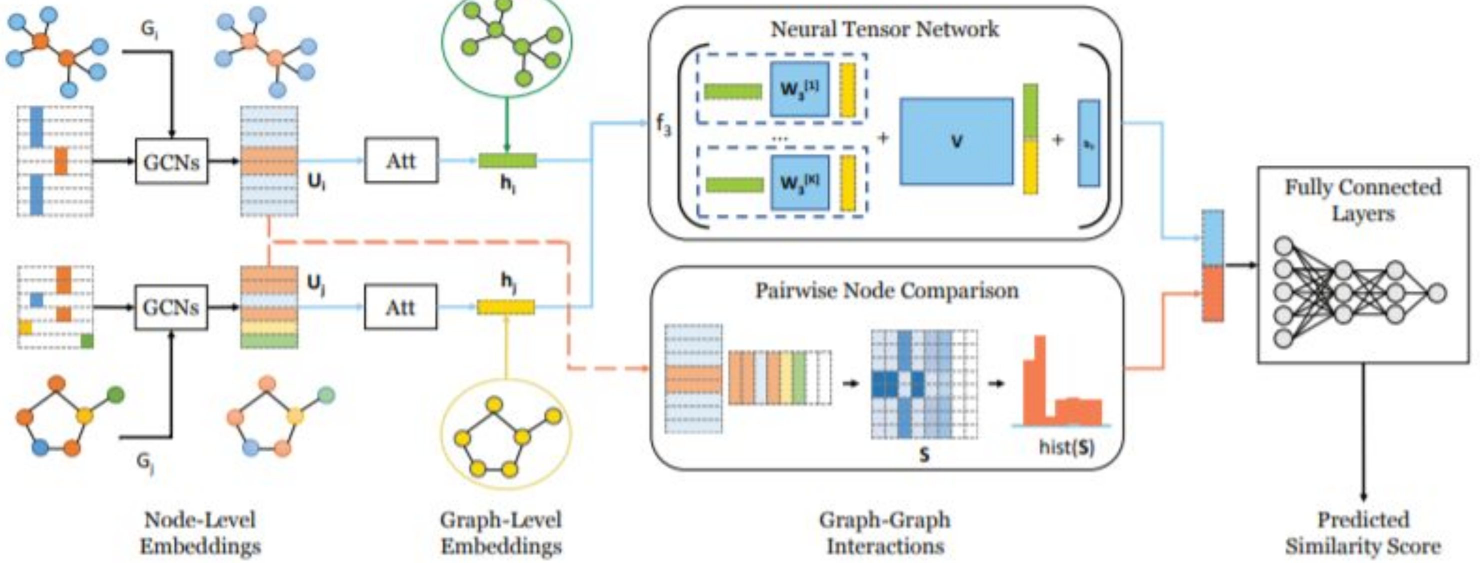


Figure 3: An overview illustration of SimGNN. The blue arrows denote the data flow for Strategy 1, which is based on graph-level embeddings. The red arrows denote the data flow for Strategy 2, which is based on pairwise node comparison.

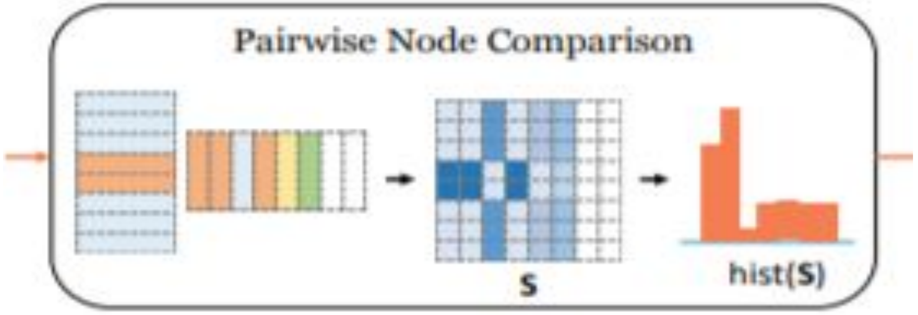
Pairwise node comparison

Compute the pairwise similarity matrix using the node level embeddings generated in previous method using the below equation -

$$\boldsymbol{S} = \sigma(\boldsymbol{U}_{\boldsymbol{i}} \boldsymbol{U}_{\boldsymbol{j}}^{T}),$$

Then we extract the histogram feature vector from the pairwise similarity matrix.

The histogram feature vector is concatenated with similarity scores obtained from graph level embeddings and fed to the fully connected layer to compute the final similarity score.



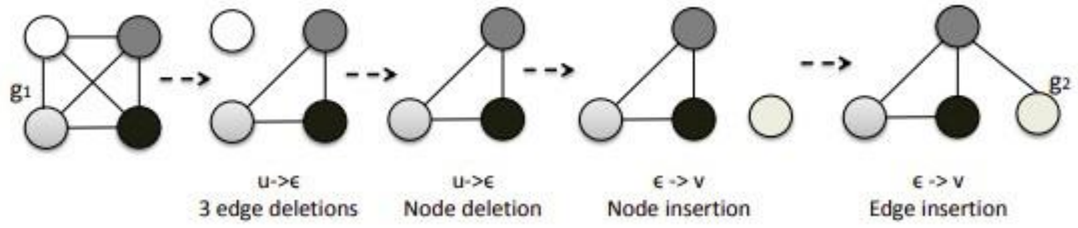
Time Complexity Analysis

The time complexity for generation of node level and graph level embeddings is O(E), where E = number of edges.

The time complexity of strategy 1 is $O(D^2K)$, where D is dimension of graph level embedding and K is feature map dimension of NTN. (D is dimension of node level embedding as well)

The time complexity of strategy 2 is $O(DN^2)$, where N is number of nodes in graph.

GED Example



g1
u->ϵ
3 edge deletions
u->ϵ
Node deletion
ϵ -> v
Node insertion
ϵ -> v
Edge insertion
g2