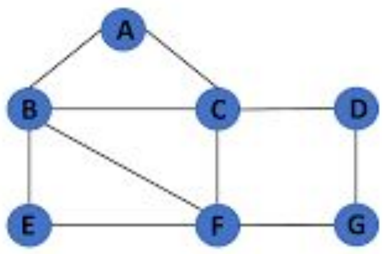
Deep Learning on Graph Data Structures

Graph Overview

Graphs are all around us, often things that we see are interconnected to each other that network with sequence is called



Graph Data Structure

Let’s see what Graph Consist of

G(V,E)

Where V is the node and E is edge

They can also store attributes/features like labels, edge weight etc.

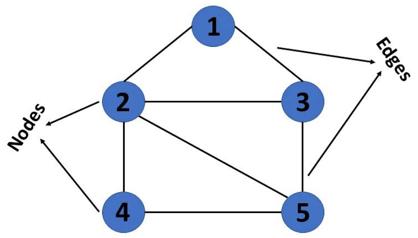


Image as Graph

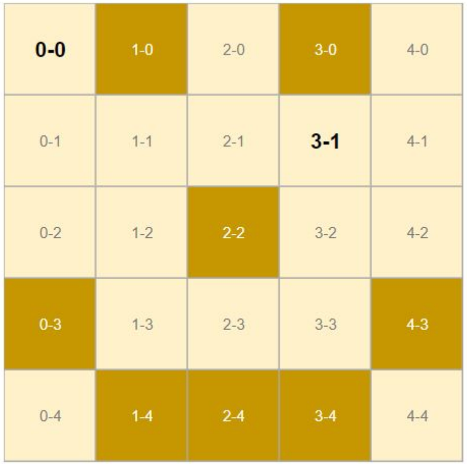
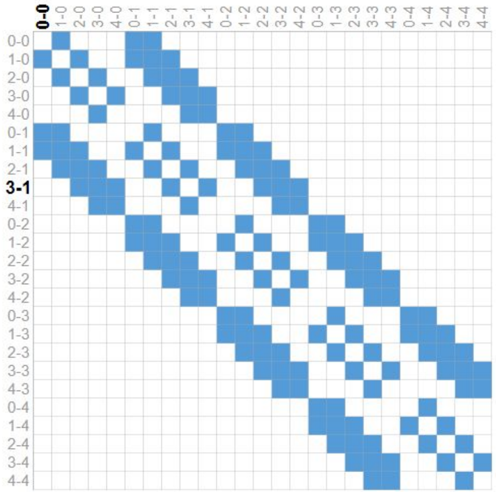
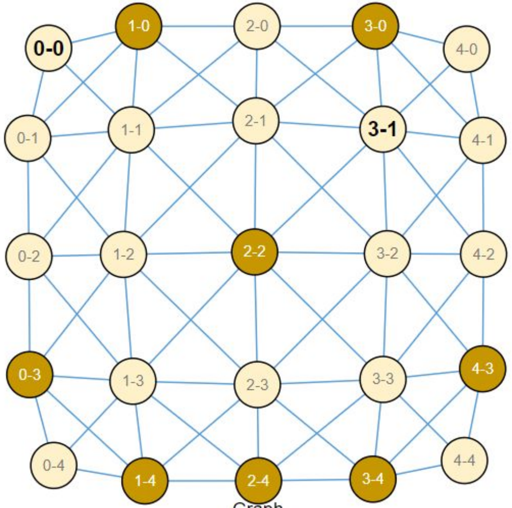


Image Pixels

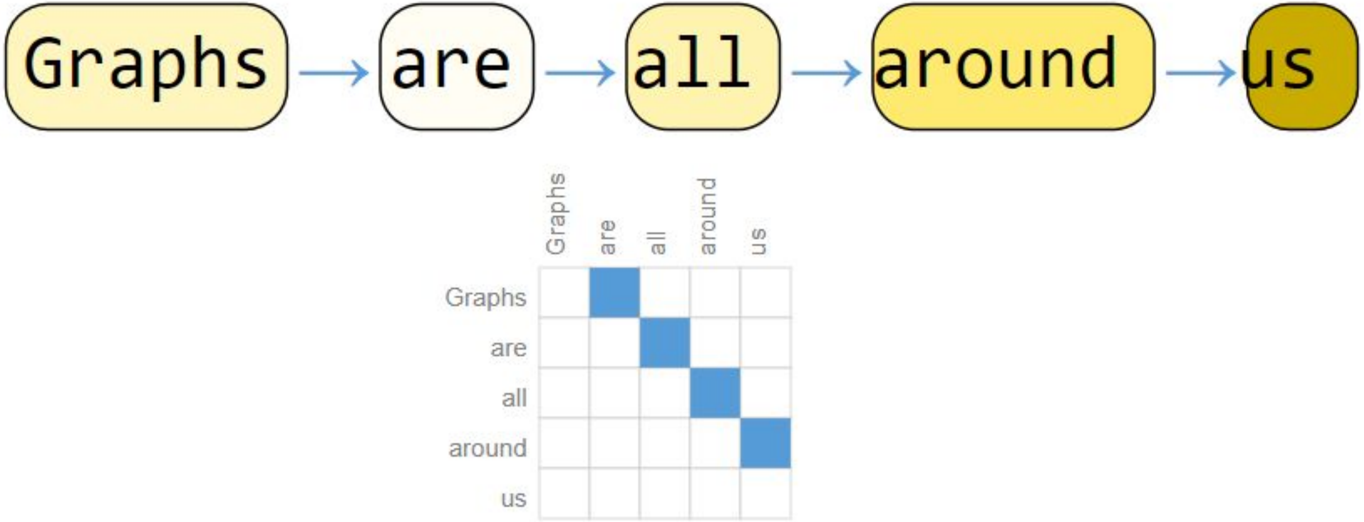


Adjacency Matrix

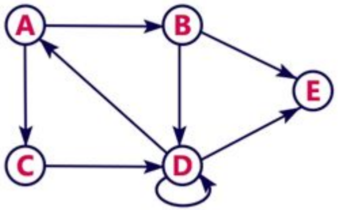


Graph

Text as Graph

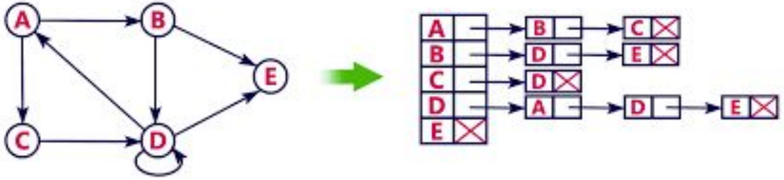


Representation of Graph



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

Adjacency Matrix



Adjacency List

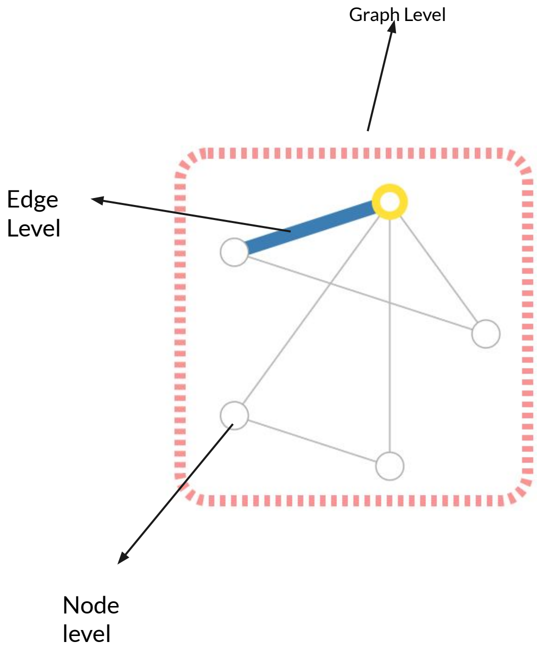
Different Types of Task with Graph

There are three level task we can perform on Graph Data Structure

1. Node Level Task
2. Edge Level Task
3. Graph Level Task

Applications

1. Molecule Sequence prediction Task
2. Physics Simulation
3. Fake news detection
4. Recommender System

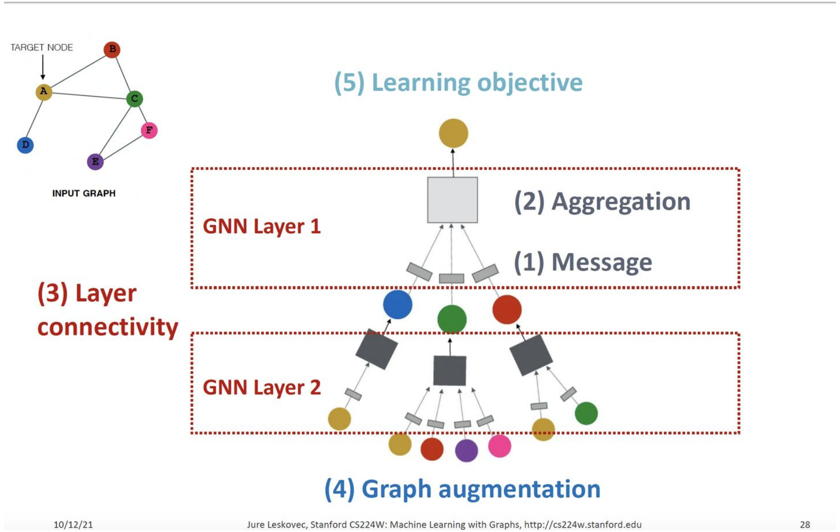


Overview of GNN

A GNN is an optimizable transformation on all attributes of the graph (nodes, edges, global-context) that preserves graph symmetries (permutation invariances).

Five Steps for building any model:

1. Message Passing
2. Aggregation
3. Layers and its Connectivity
4. Graph Augmentation
5. Learning Objective

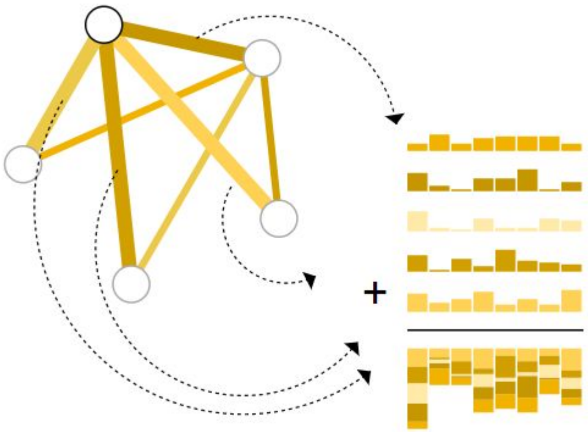


Message

In a GNN, messages are used to propagate information through the graph. In each layer of the network, each node receives updates from its neighbors. These updates are based on the messages that are passed between the nodes. The updates are then used to update the state of the nodes. This process is repeated until the network converges.

Aggregation

It is used to combine the features of a node's neighbors into a single representation for the node. This representation can then be used to make predictions about the node, such as its label or its importance in the graph.



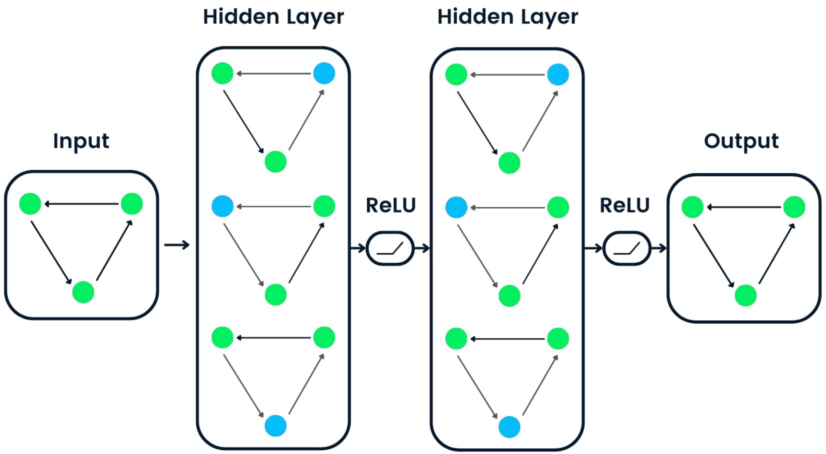
Aggregate information
from adjacent edges

Layers and its Connectivity

After getting information from aggregating we name it as input layer and GNN layers are connected to get the prediction or learning objective.

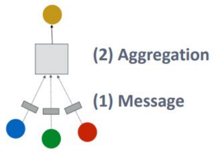
Different GNN models:

1. GCN(Graph Convolution Network)
2. GAT(Graph Attention Networks)
3. GraphSAGE
4. GGNN(Gated Graph Neural Network)



GCN(Graph Convolution Network)

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in N(v)} \mathbf{W}^{(l)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|}\right)$$



- **Message:**
  - Each Neighbor: $\mathbf{m}_u^{(l)} = \frac{1}{|N(v)|} \mathbf{W}^{(l)} \mathbf{h}_u^{(l-1)}$

Normalized by node degree
(In the GCN paper they use a slightly different normalization)

- **Aggregation:**
  - **Sum** over messages from neighbors, then apply activation
  - $\mathbf{h}_v^{(l)} = \sigma\left(\mathrm{Sum}\left(\left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}\right)\right)$

In GCN graph is assumed to have self-edges that are included in the summation.

Let D be diagonal matrix where

$$D_{v,v} = \mathrm{Deg}(v) = |N(v)|$$

- The inverse of D: D^{-1} is also diagonal:

  $D_{v,v}^{-1} = 1/|N(v)|$

$$\boxed{\sum_{u \in N(v)} \frac{h_u^{(k-1)}}{|N(v)|}} \Rightarrow H^{(k+1)} = D^{-1} A H^{(k)}$$

We normalized A and $D^{-1}A = D^{-0.5}(A+I)D^{-0.5}$ by Associative Rule, I is added to make sure that training node v feature vector also added.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$$

where

$W^{(l)}$ is trainable parameter

$$\tilde{A} = A + I_N$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

Graph Augmentation

Raw input ≠computational Graph

Reasons:

1. Input graph may lacks features
2. Graph can be too sparse
3. Graph can be too dense
4. Graph can be too large for computation

That’s why it's important to Augment the Graph.

There are lot of techniques that can Augment Graph Data but these are majorly use:

1. Node augmentation: This involves adding new nodes to the graph. New nodes can be created by randomly sampling from the feature space or by copying existing nodes.

2. Edge augmentation: This involves adding new edges to the graph. New edges can be created by randomly connecting two existing nodes or by connecting a node to itself.

3. Feature augmentation: This involves adding new features to the nodes in the graph. New features can be created by randomly sampling from the feature space or by combining existing features.

Learning Objective

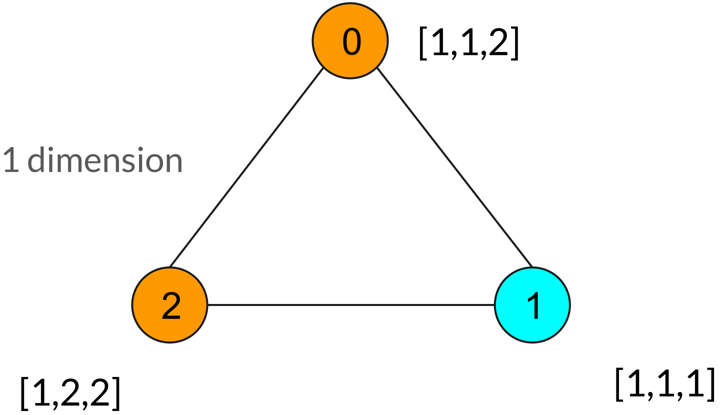
After getting embedding from the model as final output we have to use the output layer such as LinearClassifier, Softmax etc, based on the task that we have to get, we have to perform Regression, Classifier model on it.

Approach of GCN on simple graph

This is Graph G(V,E):

Labels:[0,1,0]

Feature vector x of 3*1 dimension



Adjacency Matrix

A =

0	1	1
1	0	1
1	1	0

$X=$

1	1	2
1	1	1
1	2	2

$\bar{A}=A+I=$

1	1	1
1	1	1
1	1	1

$D=$

3	0	0
0	3	0
0	0	3

$D^{-1}=$

1/3	0	0
0	1/3	0
0	0	1/3

$D^{-½}=$

$1/\sqrt{3}$	0	0
0	$1/\sqrt{3}$	0
0	0	$1/\sqrt{3}$

$D^{-1/2}\bar{A}=$

$1/\sqrt{3}$	$1/\sqrt{3}$	$1/\sqrt{3}$
$1/\sqrt{3}$	$1/\sqrt{3}$	$1/\sqrt{3}$
$1/\sqrt{3}$	$1/\sqrt{3}$	$1/\sqrt{3}$

$D^{-1/2}\bar{A}D^{-1/2}=$

1/3	1/3	1/3
1/3	1/3	1/3
1/3	1/3	1/3

$D^{-1/2}\bar{A}D^{-1/2}X=$

1	4/3	5/3
1	4/3	5/3
1	4/3	5/3

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

where

$$\tilde{A} = A + I_N$$

$$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

Now we have our input layer let’s run our input from one GCN layer with four hidden channels.

$H^{[0]}$=X

$H^{[1]}$=**σ**($D^{-0.5}AD^{-0.5}XW$)

W=

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{44}

this is a random value of the paramters

$D^{-1/2}\bar{A}D^{-1/2}XW=$

$a_{11}+4/3a_{21}+5/3a_{31}$	$a_{12}+4/3a_{22}+5/3a_{32}$	$a_{13}+4/3a_{23}+5/3a_{33}$	$a_{14}+4/3a_{24}+5/3a_{34}$
$a_{11}+4/3a_{21}+5/3a_{31}$	$a_{12}+4/3a_{22}+5/3a_{32}$	$a_{13}+4/3a_{23}+5/3a_{33}$	$a_{14}+4/3a_{24}+5/3a_{34}$
$a_{11}+4/3a_{21}+5/3a_{31}$	$a_{12}+4/3a_{22}+5/3a_{32}$	$a_{13}+4/3a_{23}+5/3a_{33}$	$a_{14}+4/3a_{24}+5/3a_{34}$

By taking parameter value = 1

$\boldsymbol{\sigma}(D^{-1/2}\bar{A}D^{-1/2}XW)=$

0.95	0.95	0.95	0.95
0.95	0.95	0.95	0.95
0.95	0.95	0.95	0.95

After getting final embedding of nodes we go for prediction task

- We first define what task we have to perform, if we have to perform classification we use classifier model(Linear Classifier, SVM Classifier etc.), If we have to do Regression then we will connect with Fully connected layer and then get output with softmax
- After that the result we got we check error or loss. Loss function we will use for Classification is Cross Entropy Loss and for regression we will use

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Cross Entropy Loss(Classification)

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

L2 loss function(Regression)

After getting Loss we will go to train our model, by changing parameters of our model through Backpropagation.
In this we differentiate our Loss function w.r.t parameters by chain rule then update the value of parameters by getting minimum loss function.

$$W_{new} = W_{old} - \alpha \underbrace{\frac{dJ}{dW}}_{\text{gradient}}$$

Implementation of GCN on Dataset(Karate Club)

Karate Club is well developed dataset that has been imported from networkx.

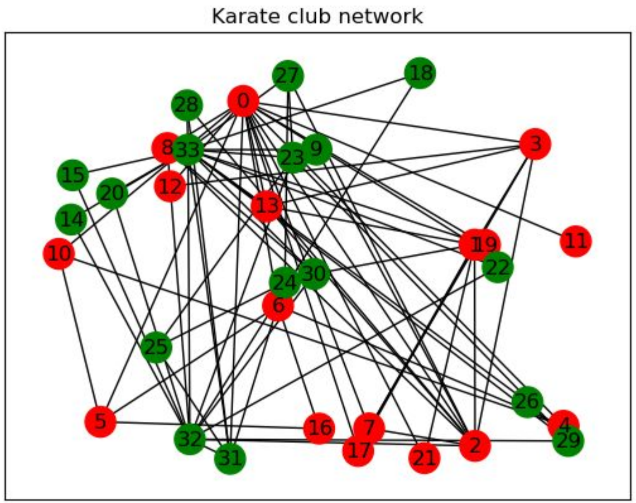
Graph: G(V,E)

G has 34 nodes and 78 edges

Feature vector is taken as I of shape (34,34)

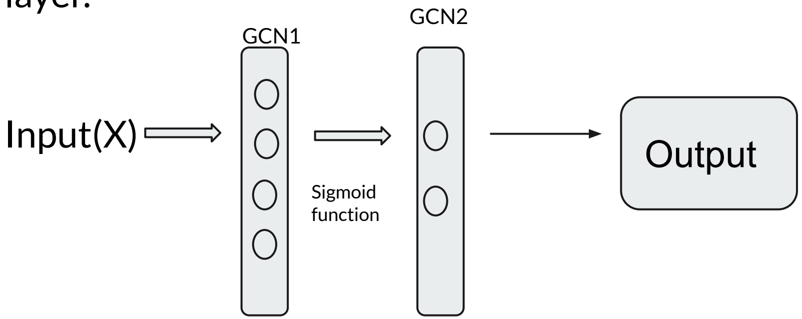
It has two classes :

Red = ‘Mr. Hi’
Green = ‘Officier’

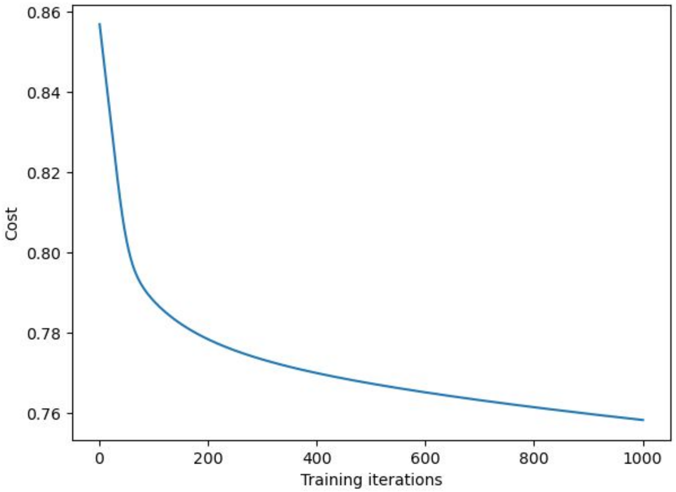


NodeDataView({0: {'club': 'Mr. Hi'}, 1: {'club': 'Mr. Hi'}, 2: {'club': 'Mr. Hi'}, 3: {'club': 'Mr. Hi'}, 4: {'club': 'Mr. Hi'}, 5: {'club': 'Mr. Hi'}, 6: {'club': 'Mr. Hi'}, 7: {'club': 'Mr. Hi'}, 8: {'club': 'Mr. Hi'}, 9: {'club': 'Officer'}, 10: {'club': 'Mr. Hi'}, 11: {'club': 'Mr. Hi'}, 12: {'club': 'Mr. Hi'}, 13: {'club': 'Mr. Hi'}, 14: {'club': 'Officer'}, 15: {'club': 'Officer'}, 16: {'club': 'Mr. Hi'}, 17: {'club': 'Mr. Hi'}, 18: {'club': 'Officer'}, 19: {'club': 'Mr. Hi'}, 20: {'club': 'Officer'}, 21: {'club': 'Mr. Hi'}, 22: {'club': 'Officer'}, 23: {'club': 'Officer'}, 24: {'club': 'Officer'}, 25: {'club': 'Officer'}, 26: {'club': 'Officer'}, 27: {'club': 'Officer'}, 28: {'club': 'Officer'}, 29: {'club': 'Officer'}, 30: {'club': 'Officer'}, 31: {'club': 'Officer'}, 32: {'club': 'Officer'}, 33: {'club': 'Officer'}})

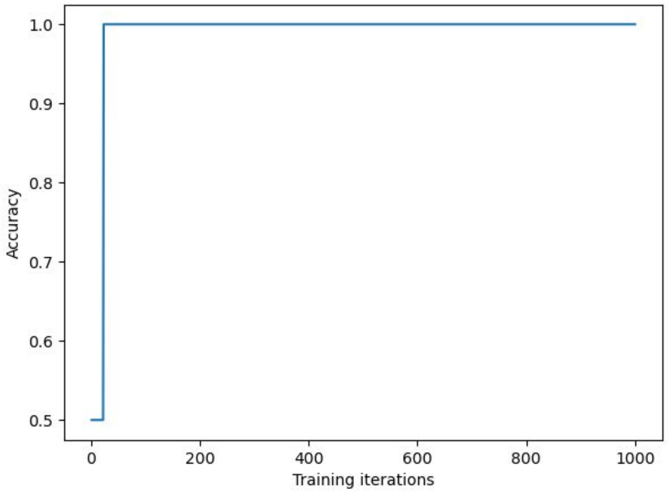
This feature vector X= I of shape(34,34) act as input vector for GCN model of two layer.



Findings



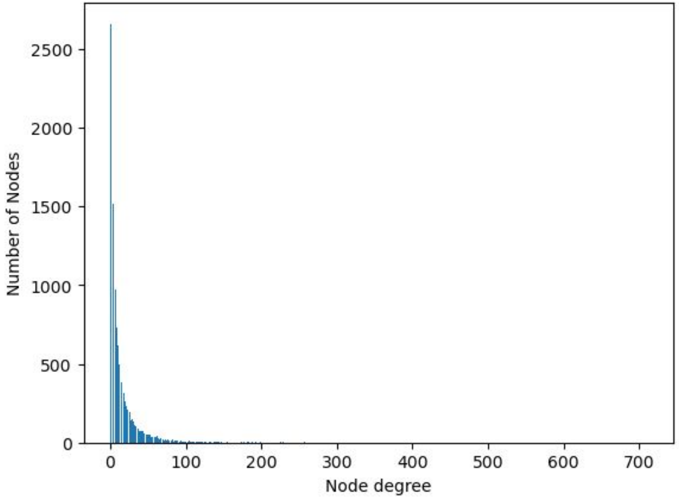
0.86
0.84
0.82
0.80
0.78
0.76
Cost
0
200
400
600
800
1000
Training iterations



1.0
0.9
0.8
0.7
0.6
0.5
Accuracy
0
200
400
600
800
1000
Training iterations

Node Classification with GCN

```
Dataset: FacebookPagePage()
-----------------------
Number of graphs: 1
Number of nodes: 22470
Number of features: 128
Number of classes: 4
```



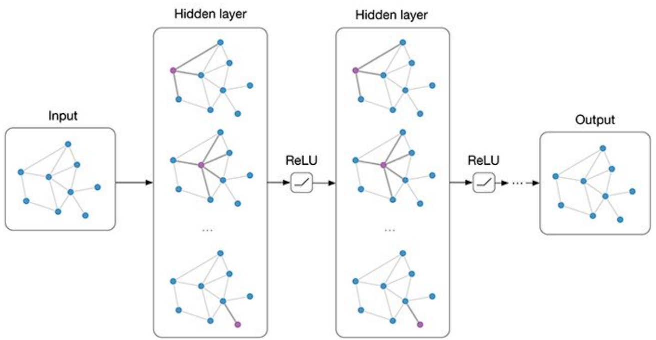
Model:

GCN(

  (gcn1): GCNConv(128, 16)

  (gcn2): GCNConv(16, 4)

)

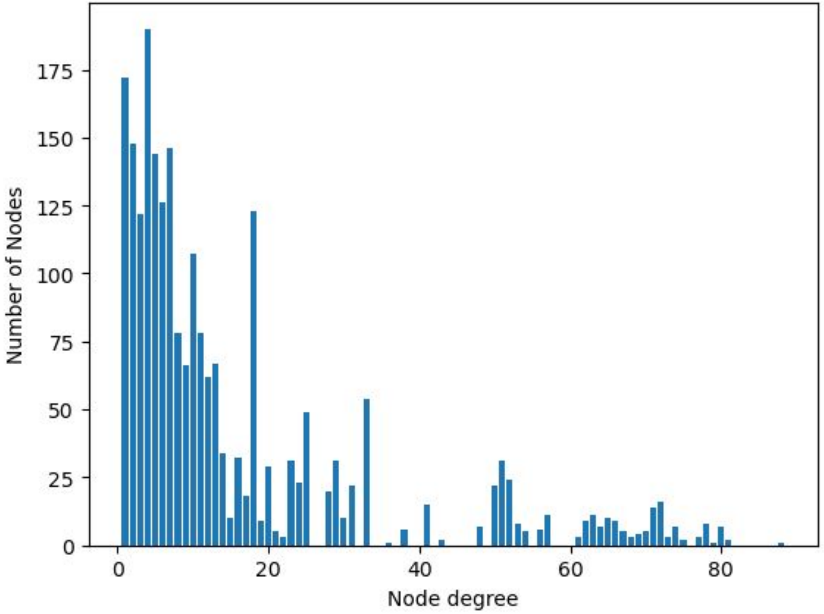


Results: Accuracy score on test set of this model is 91.33%

Node Regression with GCN

```
Dataset: WikipediaNetwork()
-----------------------
Number of graphs: 1
Number of nodes: 2277
Number of unique features: 2325
Number of classes: 5

Graph:
------
Training nodes: 1577
Evaluation nodes: 200
Test nodes: 500
Edges are directed: True
Graph has isolated nodes: False
Graph has loops: True
```



Model: GCN(

(gcn1): GCNConv(2325, 512)

(gcn2): GCNConv(512, 256)

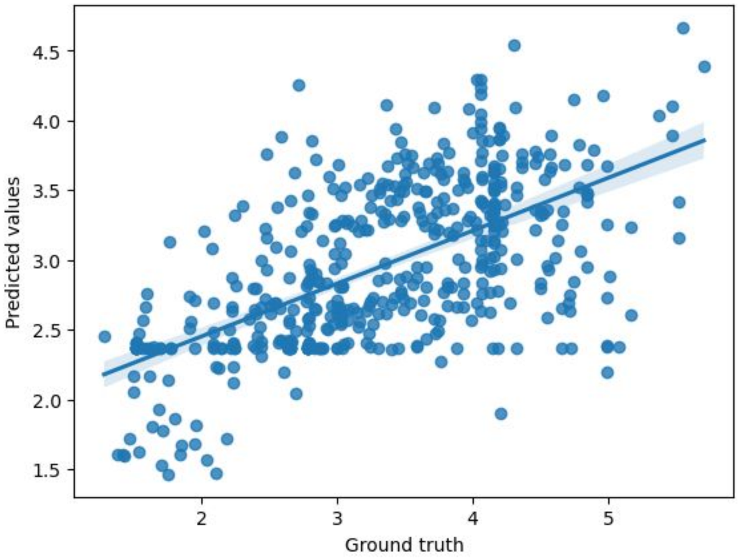
(gcn3): GCNConv(256, 128)

(linear): Linear(128, 1, bias=True)

)

Results:

```
---------------------------------------------
MSE = 0.6993 | RMSE = 0.8362 | MAE = 0.6463
---------------------------------------------
```



GAT

Attention Score

$$h_i = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W} x_j$$

Linear Transformation:

$$a_{ij} = W_{att}^T [\mathbf{W} x_i \,||\, \mathbf{W} x_j]$$

Activation function:

$$e_{ij} = LeakyReLU(a_{ij})$$

Softmax Normalization:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} exp(e_{ik})}$$

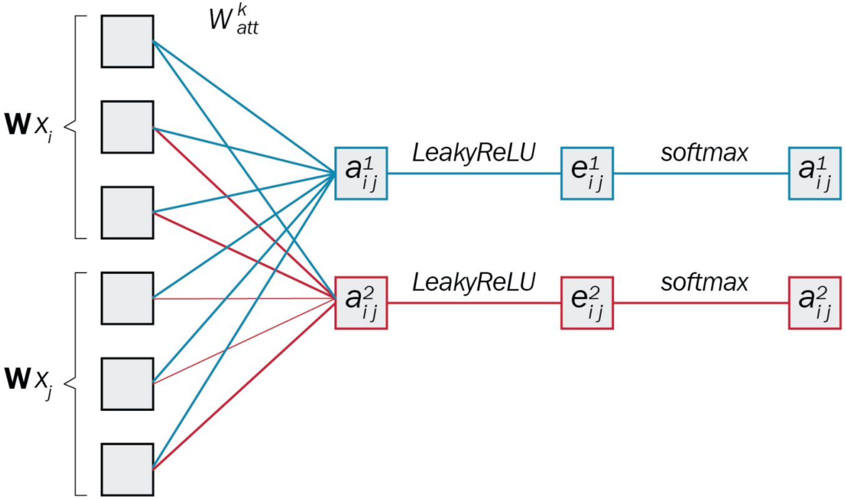
Multihead Attention:

Averaging:

$$h_i = \frac{1}{n}\sum_{k=1}^{n} h_i^k = \frac{1}{n}\sum_{k=1}^{n}\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k x_j$$

Concatenation:

$$h_i = \|_{k=1}^{n} h_i^k = \|_{k=1}^{n} \sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k x_j$$



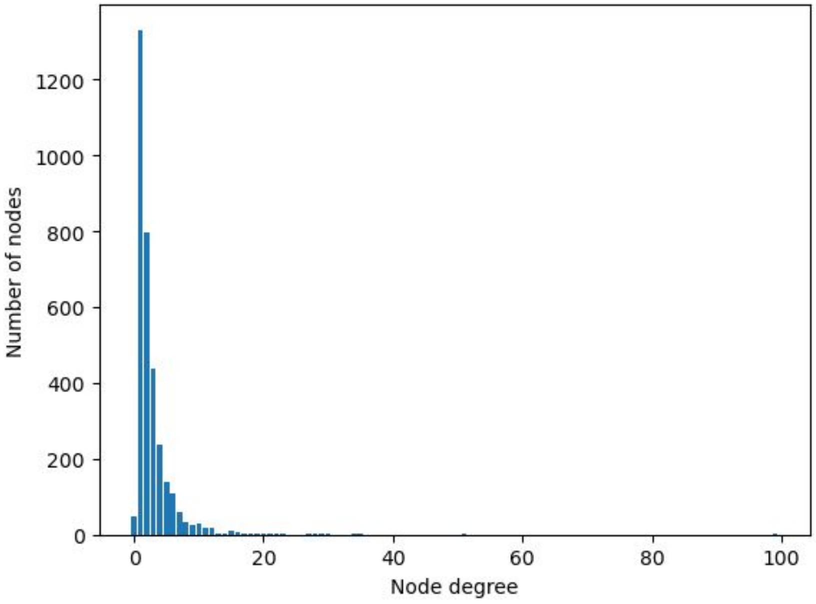
$$\alpha_{ij} = \frac{exp\,(W^t_{att}\ LeakyReLU(\mathbf{W}[x_i \mid\mid x_j]))}{\sum_{k\in\mathcal{N}_i} exp(W^t_{att}\ LeakyReLU(\mathbf{W}[x_i \mid\mid x_k]))}$$

$$h_i = \sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}x_j \qquad\qquad H = \tilde{A}^T W_\alpha X \mathbf{W}^T$$

Implementation of GAT

```
Dataset: CiteSeer()
--------------------
Number of graphs: 1
Number of nodes: 3327
Number of features: 3703
Number of classes: 6
```

```
Graph:
------
Training nodes: 120
Evaluation nodes: 500
Test nodes: 1000
Edges are directed: False
Graph has isolated nodes: True
Graph has loops: False
```



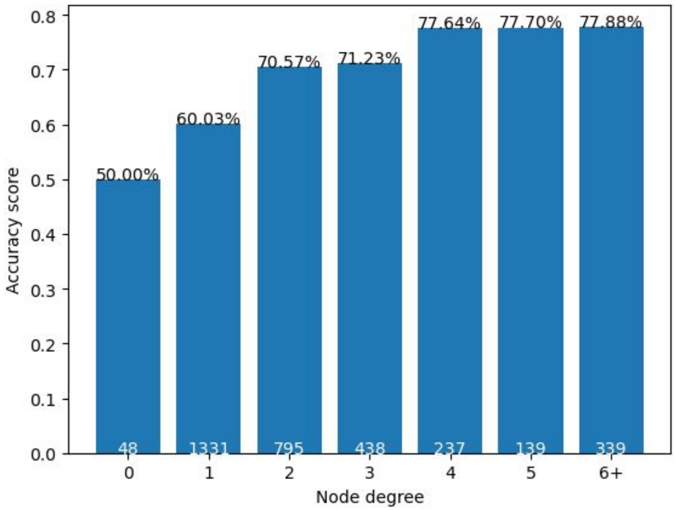
Model: GAT(

(gat1): GATv2Conv(3703, 64, heads=8)

(gat2): GATv2Conv(512, 6, heads=1)

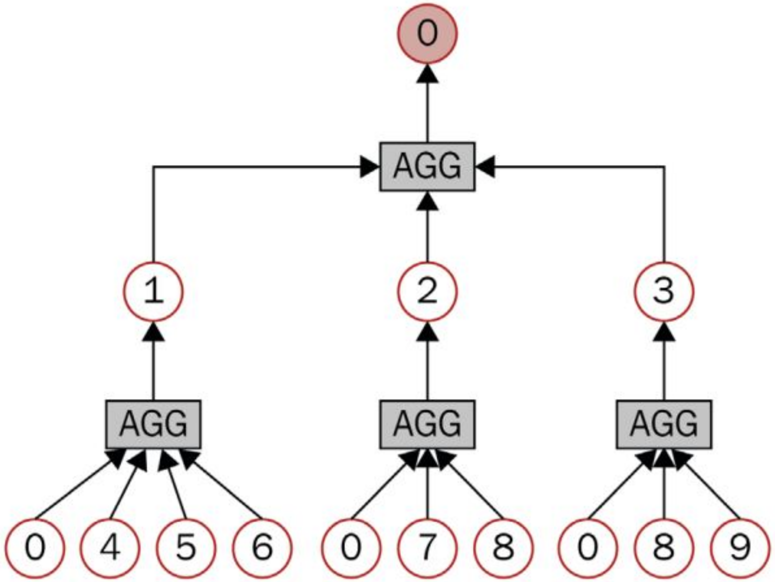
)

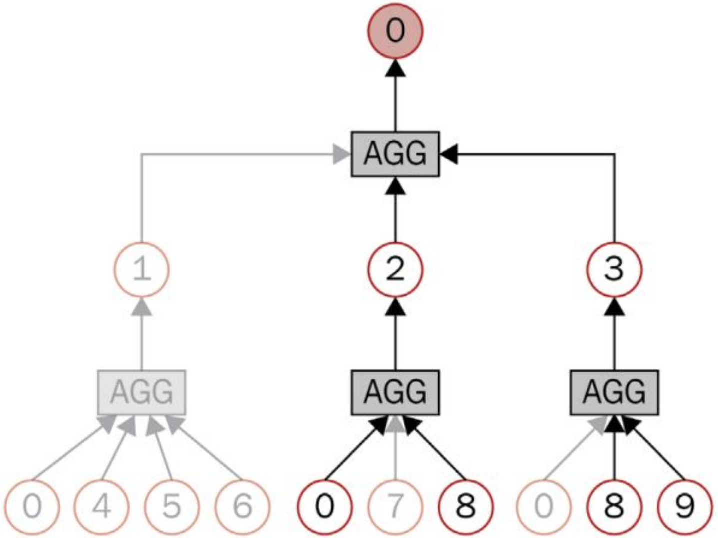
Results: GAT test accuracy: 68.00%



GraphSAGE

Sampling





Aggregation

- **Mean:** Take a weighted average of neighbors

$$\mathrm{AGG} = \boxed{\sum_{u \in N(v)}} \frac{\mathbf{h}_u^{(l-1)}}{\boxed{|N(v)|}}$$

Aggregation Message computation

- **Pool:** Transform neighbor vectors and apply symmetric vector function $\mathrm{Mean}(\cdot)$ or $\mathrm{Max}(\cdot)$

$$\mathrm{AGG} = \boxed{\mathrm{Mean}}(\{\boxed{\mathrm{MLP}}(\mathbf{h}_u^{(l-1)}), \forall u \in N(v)\})$$

Aggregation Message computation

- **LSTM:** Apply LSTM to reshuffled of neighbors

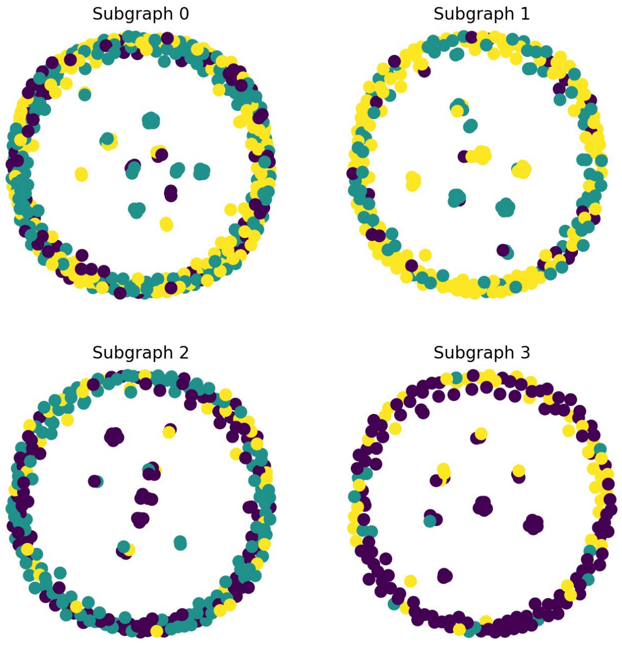
$$\mathrm{AGG} = \boxed{\mathrm{LSTM}}([\mathbf{h}_u^{(l-1)}, \forall u \in \pi(N(v))])$$

Aggregation

Implementation of GraphSAGE

```
Dataset: Pubmed()
---------------------
Number of graphs: 1
Number of nodes: 19717
Number of features: 500
Number of classes: 3
```

```
Graph:
------
Training nodes: 60
Evaluation nodes: 500
Test nodes: 1000
Edges are directed: False
Graph has isolated nodes: False
Graph has loops: False
```



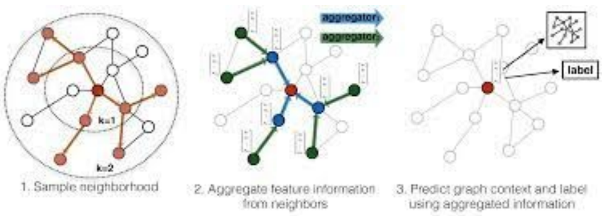
Models: GraphSAGE(

(sage1): SAGEConv(500, 64, aggr=mean)

(sage2): SAGEConv(64, 3, aggr=mean)

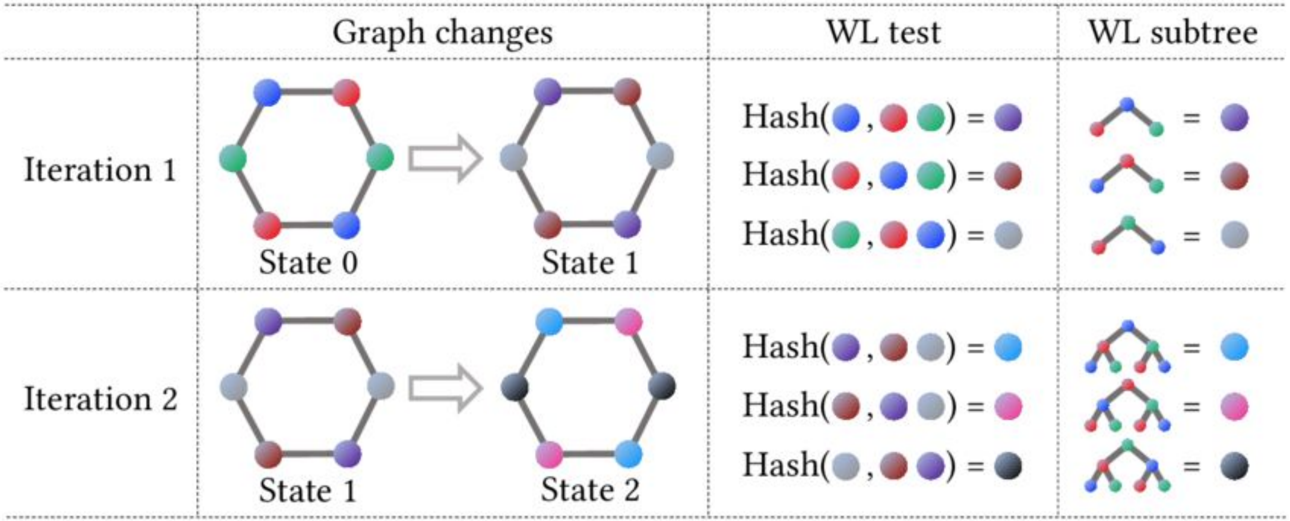
)

Results: GraphSAGE test accuracy: 74.30%



GIN

Weisfeiler-Leman (WL) test

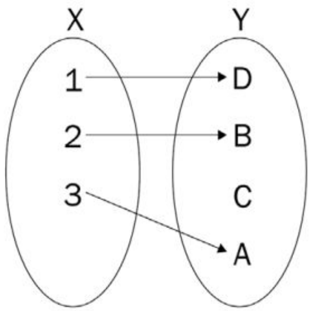


Aggregate: This function, f, selects the neighboring nodes that the GNN considers

Combine: This function, ϕ, combines the embeddings from the selected nodes to produce the new embedding of the target node

Embedding of ith node $$h_i' = \phi\left(h_i, f(\{h_j : j \in \mathcal{N}_i\})\right)$$

Universal Approximation Theorem $$h_i' = MLP\left((1+\varepsilon)\cdot h_i + \sum_{j \in \mathcal{N}_i} h_j\right)$$



Need injective function for node embedding

We require two or more these MLP layer to get node embedding of the neighbour.

Global Pooling

$$h_G = \frac{1}{N}\sum_{i=0}^{N} h_i$$

Mean

$$h_G = \max_{i=0}^{N}(h_i)$$

Max

$$h_G = \sum_{i=0}^{N} h_i$$

Sum

Concatenation of the node embedding produce by every layer

$$h_G = \sum_{i=0}^{N} h_i^0 \,||\, \dots \,||\, \sum_{i=0}^{N} h_i^k$$

Implementation of GIN

```
Dataset: PROTEINS(1113)
------------------------
Number of graphs: 1113
Number of nodes: 14
Number of features: 3
Number of classes: 2
```

```
Training set   = 890 graphs
Validation set = 111 graphs
Test set       = 112 graphs
```

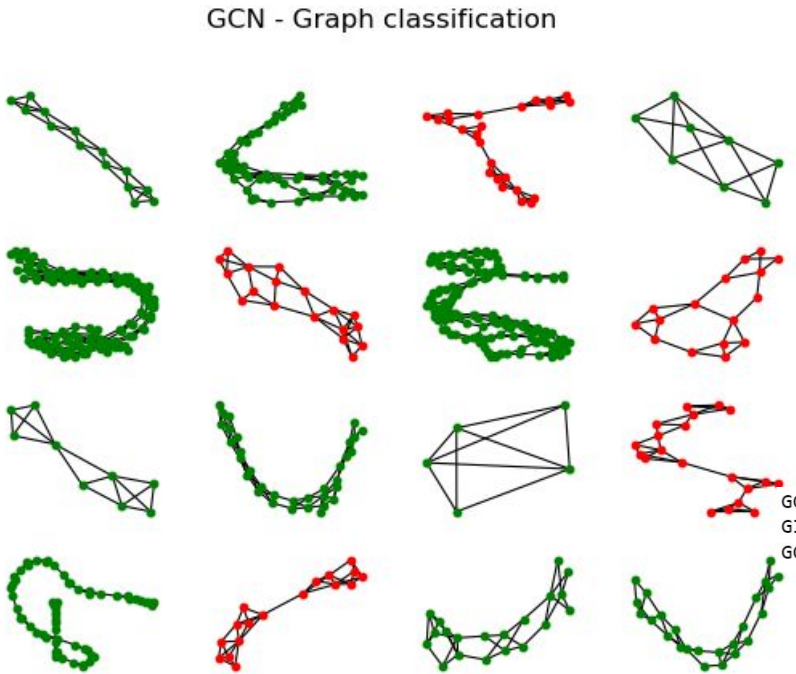
Batch size = 64

Models:

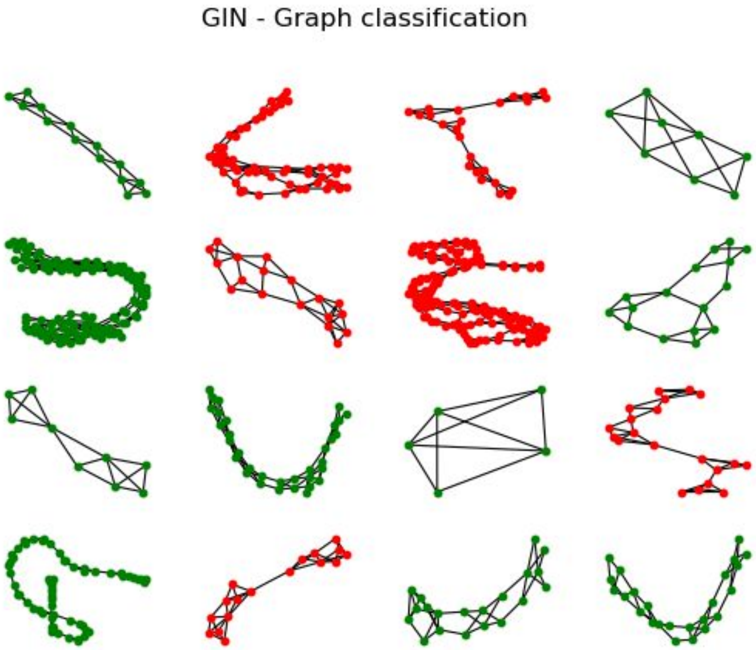
```
GCN(
  (conv1): GCNConv(3, 32)
  (conv2): GCNConv(32, 32)
  (conv3): GCNConv(32, 32)
  (lin): Linear(in_features=32, out_features=2, bias=True)
)
```

```
GIN(
  (conv1): GINConv(nn=Sequential(
    (0): Linear(in_features=3, out_features=32, bias=True)
    (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (2): ReLU()
    (3): Linear(in_features=32, out_features=32, bias=True)
    (4): ReLU()
  ))
  (conv2): GINConv(nn=Sequential(
    (0): Linear(in_features=32, out_features=32, bias=True)
    (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (2): ReLU()
    (3): Linear(in_features=32, out_features=32, bias=True)
    (4): ReLU()
  ))
  (conv3): GINConv(nn=Sequential(
    (0): Linear(in_features=32, out_features=32, bias=True)
    (1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
    (2): ReLU()
    (3): Linear(in_features=32, out_features=32, bias=True)
    (4): ReLU()
  ))
  (lin1): Linear(in_features=96, out_features=96, bias=True)
  (lin2): Linear(in_features=96, out_features=2, bias=True)
)
```

Results

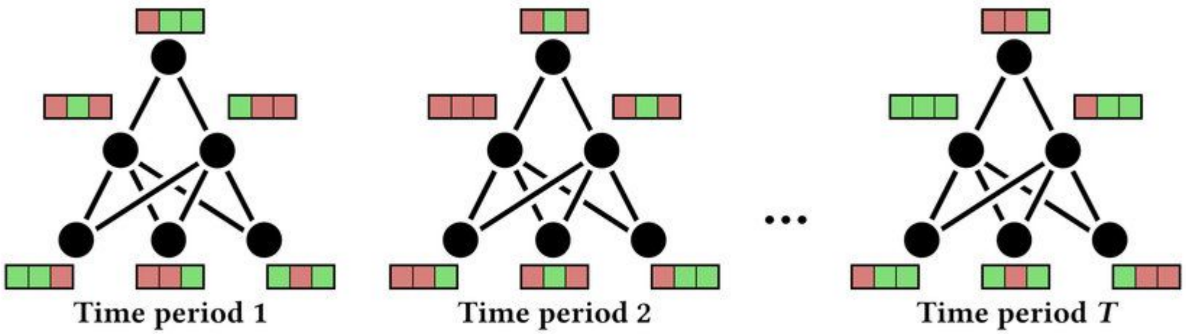



```
GCN accuracy:     67.45%
GIN accuracy:     76.30%
GCN+GIN accuracy: 71.88%
```

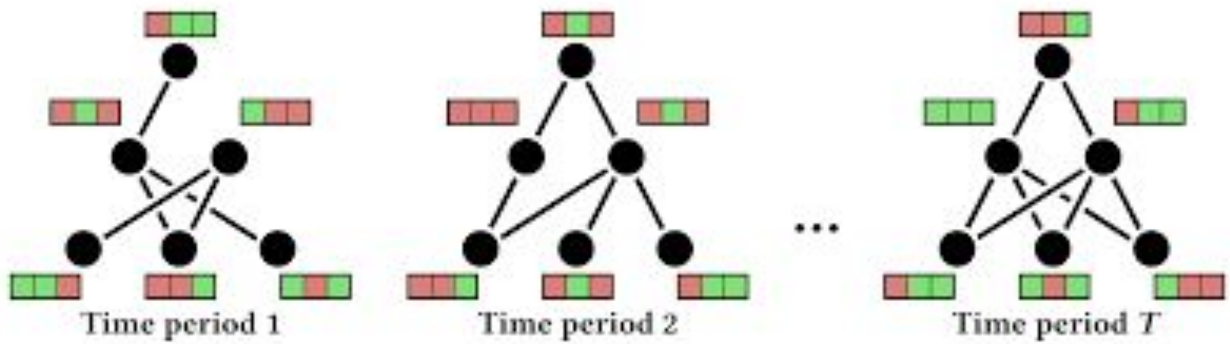


Dynamic Graph

Static graphs with temporal signals: The underlying graph does not change, but features and labels evolve over time.



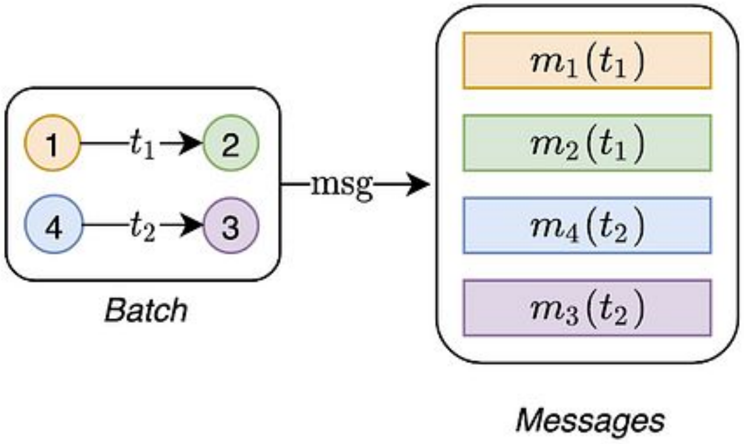
Dynamic graphs with temporal signals: The topology of the graph (the presence of nodes and edges), features, and labels evolve over time



Temporal GNN

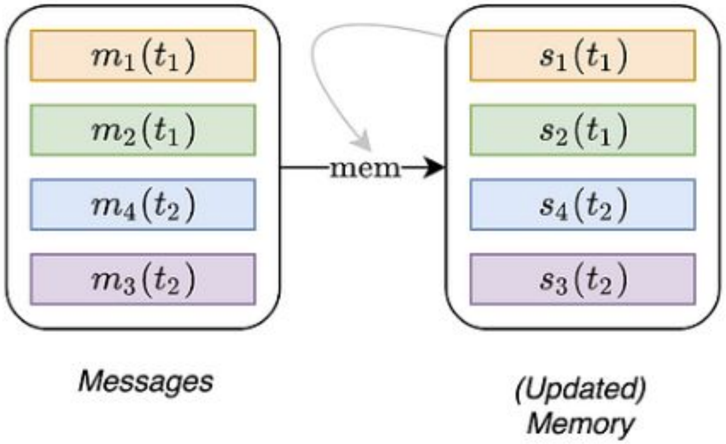
Message Function: Given an interaction between nodes i and j at time t, the message function computes two messages (one for i and one for j), which are used to update the memory.

$$\mathbf{m}_i(t) = \mathrm{msg}\left(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), t, \mathbf{e}_{ij}(t)\right)$$
$$\mathbf{m}_j(t) = \mathrm{msg}\left(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), t, \mathbf{e}_{ij}(t)\right)$$



Memory Updater:is used to update the memory with the new messages. This module is usually implemented as an RNN.

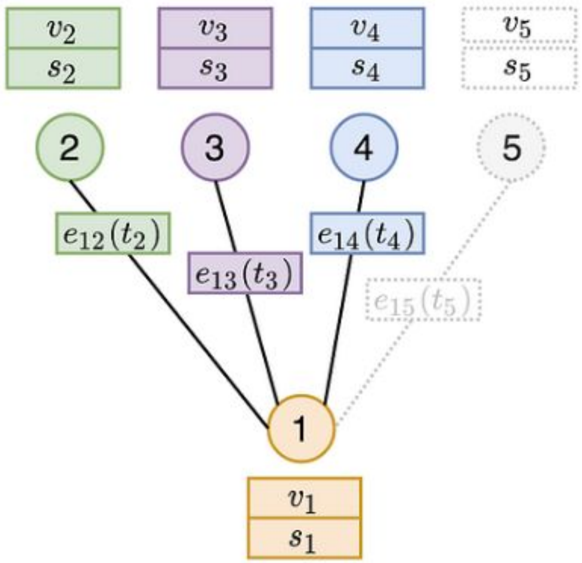
$$\mathbf{s}_i(t) = \text{mem}\left(\mathbf{m}_i(t), \mathbf{s}_i(t^-)\right)$$



Staleness Problem: using these has a direct embedding creates a staleness problem.

Example: If a node doesn’t have interaction over a long interval of time then its node embedding will be outdated.

Solution: Using Spatio temporal neighbours, by performing aggregation of embedding form neighbours over past time intervals.

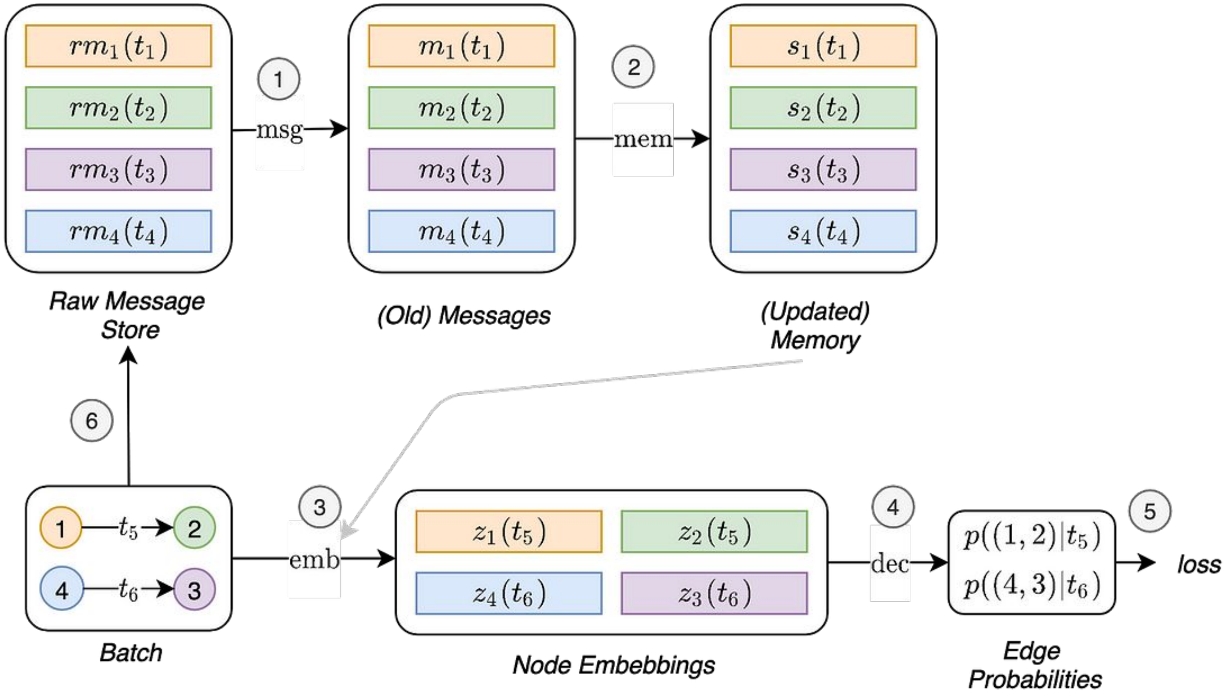


$$
\begin{aligned}
\mathbf{h}_i^{(l)}(t) &= \mathrm{MLP}^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)), \\
\tilde{\mathbf{h}}_i^{(l)}(t) &= \mathrm{MultiHeadAttention}^{(l)}(\mathbf{q}^{(l)}(t), \mathbf{K}^{(l)}(t), \mathbf{V}^{(l)}(t)), \\
\mathbf{q}^{(l)}(t) &= \mathbf{h}_i^{(l-1)}(t) \,\|\, \boldsymbol{\phi}(0), \\
\mathbf{K}^{(l)}(t) &= \mathbf{V}^{(l)}(t) = \mathbf{C}^{(l)}(t), \\
\mathbf{C}^{(l)}(t) &= [\mathbf{h}_1^{(l-1)}(t) \,\|\, \mathbf{e}_{i1}(t_1) \,\|\, \boldsymbol{\phi}(t-t_1), \ldots, \mathbf{h}_N^{(l-1)}(t) \,\|\, \mathbf{e}_{iN}(t_N) \,\|\, \boldsymbol{\phi}(t-t_N)].
\end{aligned}
$$

Temporal Graph Sum:

$$
\begin{aligned}
\mathbf{h}_i^{(l)}(t) &= \mathbf{W}_2^{(l)}(\mathbf{h}_i^{(l-1)}(t) \,\|\, \tilde{\mathbf{h}}_i^{(l)}(t)), \\
\tilde{\mathbf{h}}_i^{(l)}(t) &= \mathrm{ReLu}\Big(\sum_{j \in n_i([0,t])} \mathbf{W}_1^{(l)}(\mathbf{h}_j^{(l-1)}(t) \,\|\, \mathbf{e}_{ij} \,\|\, \boldsymbol{\phi}(t-t_j))\Big).
\end{aligned}
$$

Flow operation of TGN

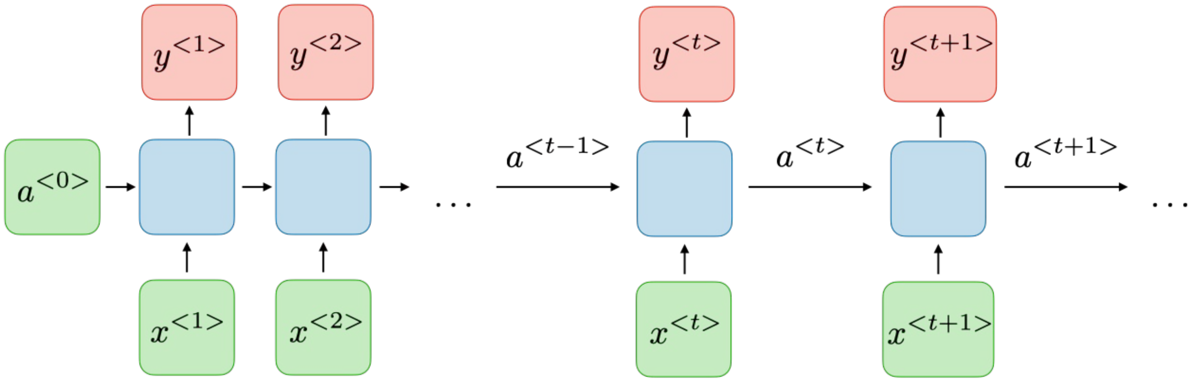


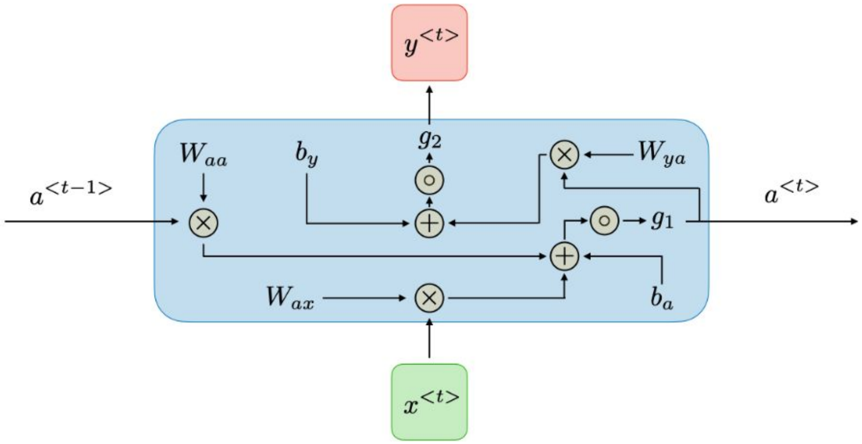
	Mem.	Mem. Updater	Embedding	Mess. Agg.	Mess. Func.
Jodie	node	RNN	time	—[†]	id
TGAT	—	—	attn (2l, 20n)*	—	—
DyRep	node	RNN	id	—[‡]	attn[‖]
TGN-attn	node	GRU	attn (1l, 10n)	last	id
TGN-2l	node	GRU	attn (2l, 10n)	last	id
TGN-no-mem	—	—	attn (1l, 10n)	—	—
TGN-time	node	GRU	time	last	id
TGN-id	node	GRU	id	last	id
TGN-sum	node	GRU	sum (1l, 10n)	last	id
TGN-mean	node	GRU	attn (1l, 10n)	mean	id

Now let's learn more about the memory updater.

RNN (Recurrent Neural Network)

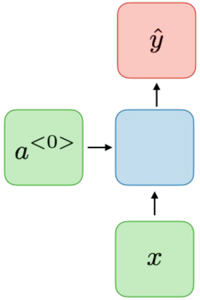
This a architecture design to capture sequence and can do forecasting over a sequence



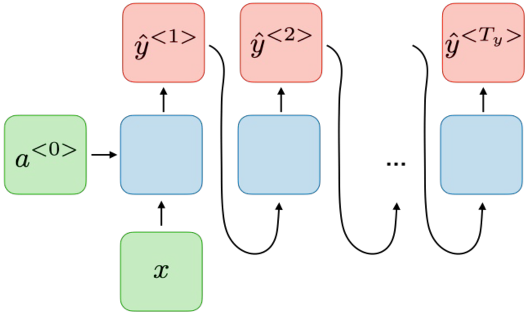


$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

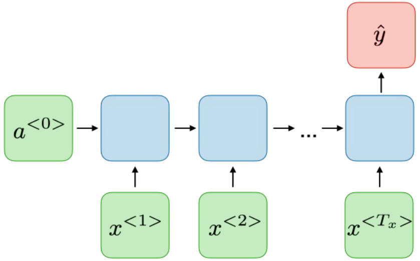
Types of RNN



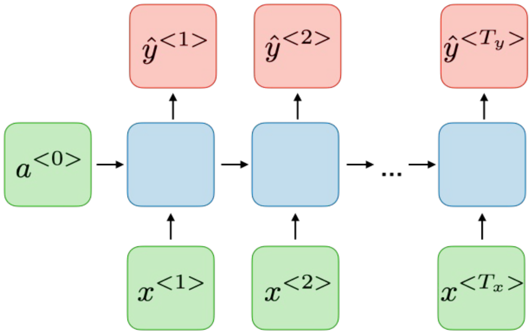
One to
One



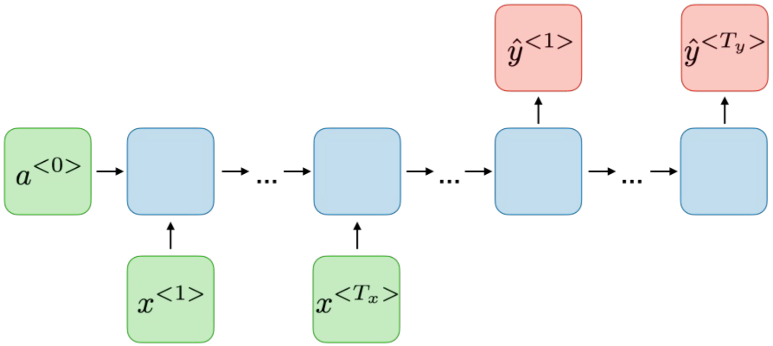
One to many



Many to One



Many to many(Tx=Ty)



Many to many(Tx≠Ty)

Loss Function: In the case Recurrent neural network the loss function L of all time steps is defined based on the loss at every time steps

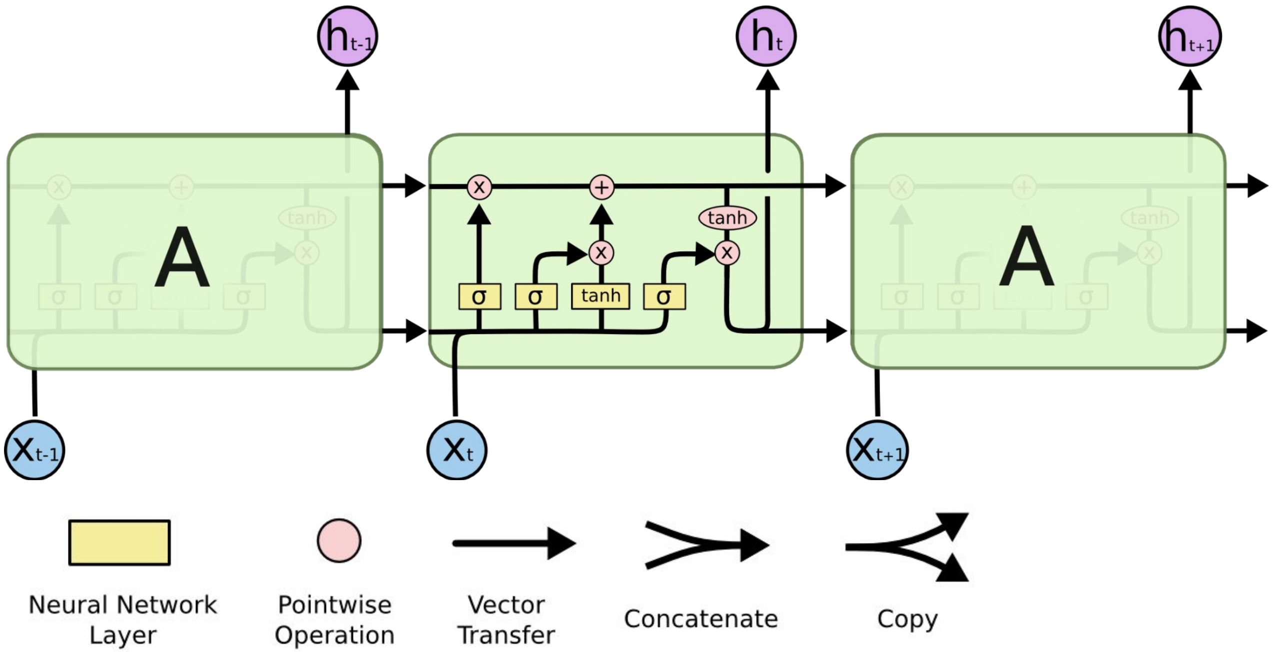
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}(\hat{y}^{<t>}, y^{<t>})$$

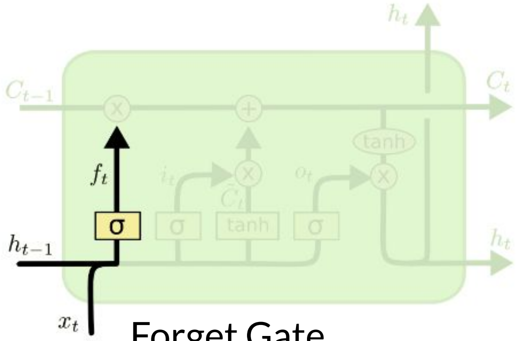
Backpropagation: Backpropagation is done at each point in time. At timestep T, the derivative of the loss L with respect to weight matrix W

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left. \frac{\partial \mathcal{L}^{(T)}}{\partial W} \right|_{(t)}$$

Traditional RNN is computationally inefficient and also have vanishing and exploding gradient problem. By gradient clipping we can solve exploding gradient problem and for vanishing gradient we will use Gated architecture LSTM and GRU

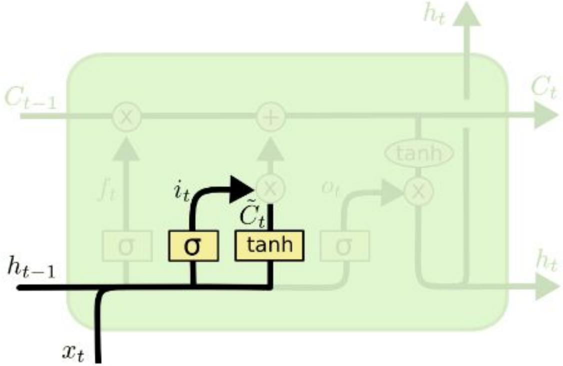
LSTM (Long Short Term Memory)





Forget Gate

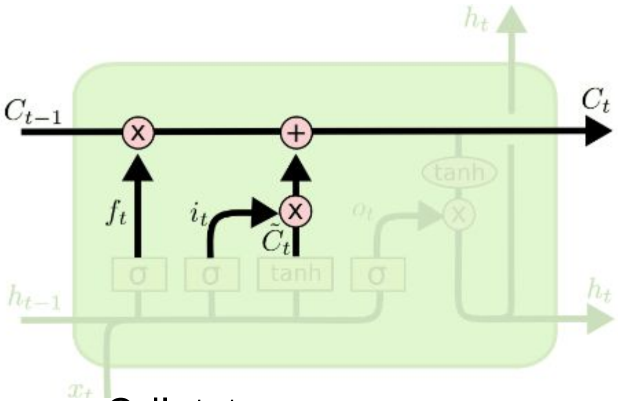
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



Input gate

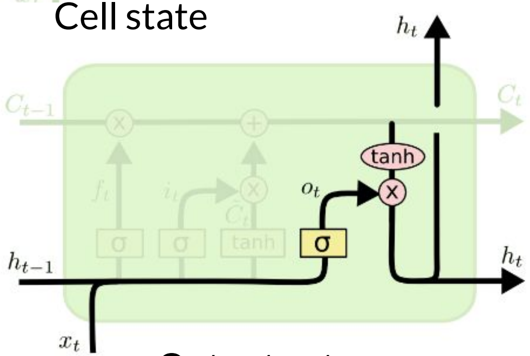
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



Cell state

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

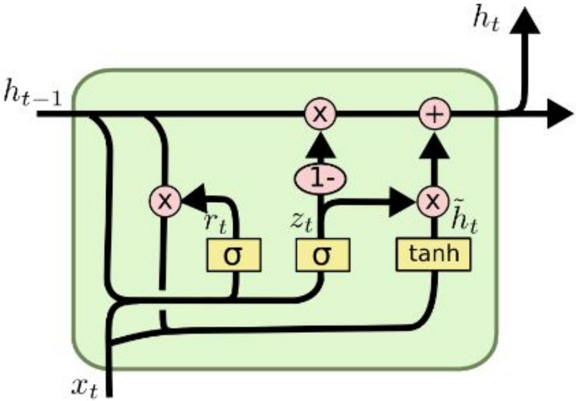


Output gate

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

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

GRU (Gated Recurrent Unit)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

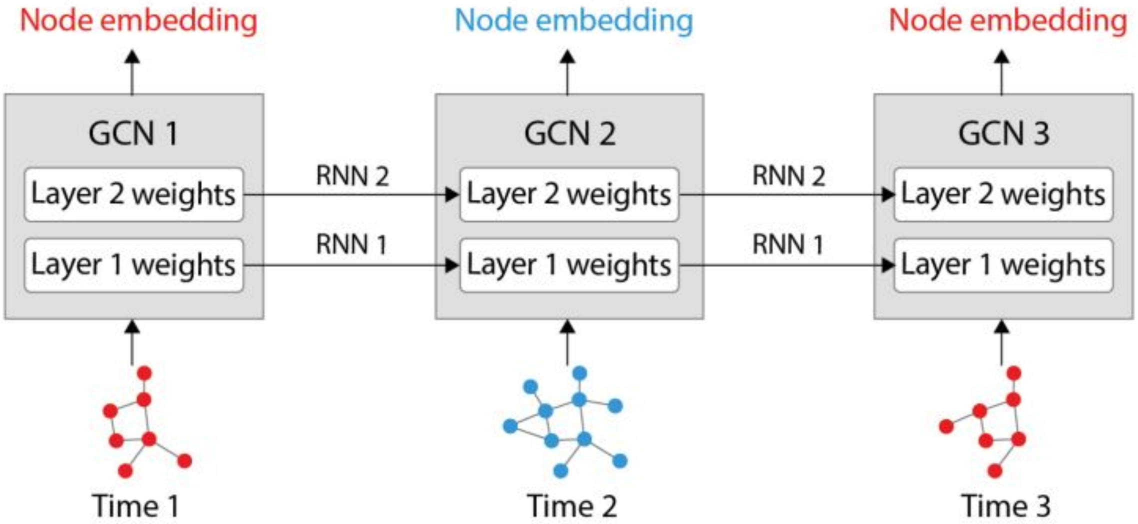
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

It is a variation in LSTM with combining forget and input gate into a single update gate.

EvolveGCN



Two variants of EvolveGCN:

1. EvolveGCN-H: uses Gated Recurrent Unit (GRU)
2. EvolveGCN-O: uses LSTM networks

EvolveGCN-H

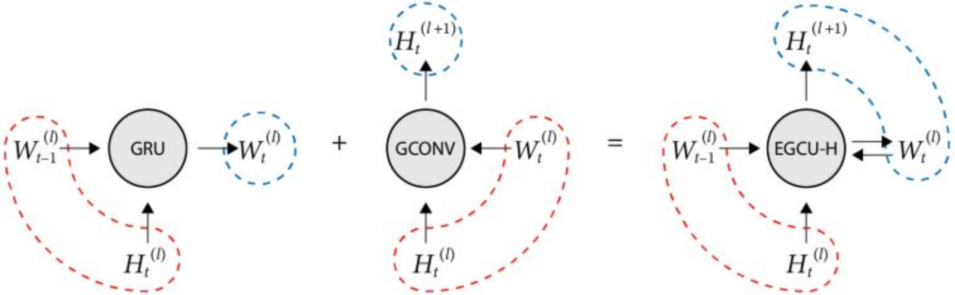
GRU updates the GCN's weight matrix for the layer l at time t

$$W_t^{(l)} = \mathrm{GRU}\left(H_t^{(l)}, W_{t-1}^{(l)}\right)$$

This resulting weight matrix is used to calculate the next layer's node embedding

$$H_t^{(l+1)} = \mathrm{GCN}\left(A_t, H_t^{(l)}, W_t^{(t)}\right)$$

$$= \tilde{D}^{-\frac{1}{2}} \tilde{A}^T \tilde{D}^{-\frac{1}{2}} H_t^{(l)} {W_t^{(t)}}^T$$



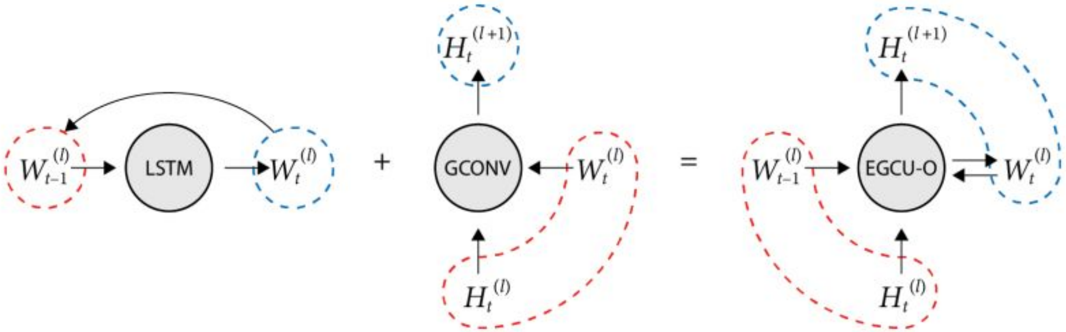
EvolveGCN-O

LSTM update the weight matrix W for layer l at time t

$$W_t^{(l)} = \mathrm{LSTM}\,(W_{t-1}^{(l)})$$

This resulting weight matrix is used to calculate the next layer's node embedding

$$H_t^{(l+1)} = \mathrm{GCN}\,(A_t, H_t^{(l)}, W_t^{(t)})$$
$$= \tilde{D}^{-\frac{1}{2}}\,\tilde{A}^T\,\tilde{D}^{-\frac{1}{2}}\,H_t^{(l)}\;{W_t^{(t)}}^T$$



Implementation of EvolveGCN

Dataset:

The WikiMaths dataset is comprised of 1,068 articles represented as nodes.

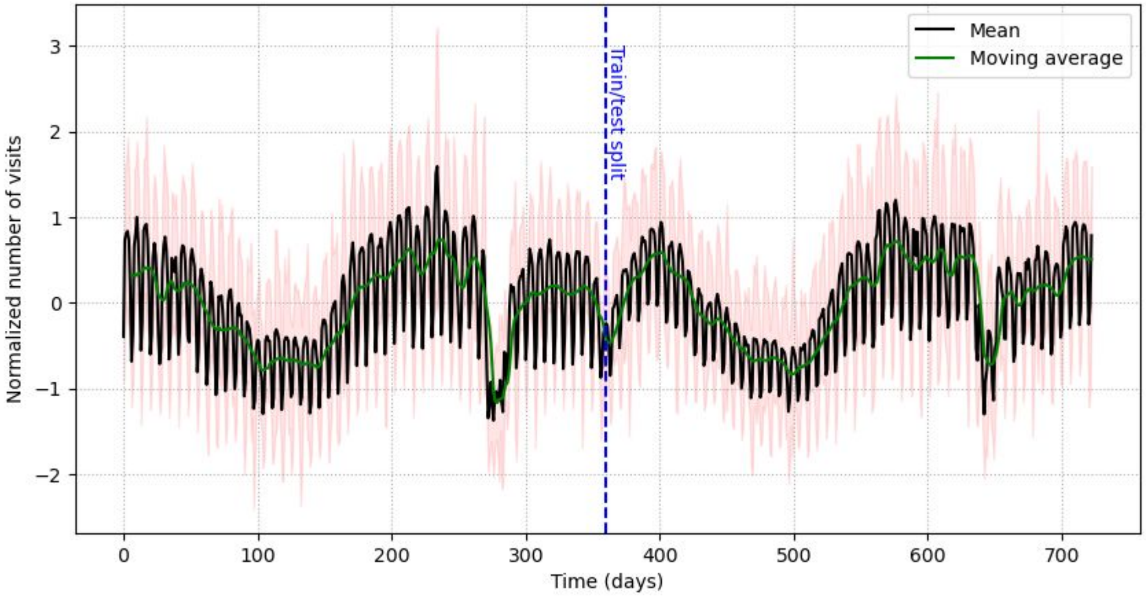
Node features correspond to the past daily number of visits (eight features by default).

Edges are weighted, and weights represent the number of links from the source page to the destination page.

We want to predict the daily user visits to these Wikipedia pages between March 16, 2019, and March 15, 2021, which results in 731 snapshots.

Each snapshot is a graph describing the state of the system at a certain time

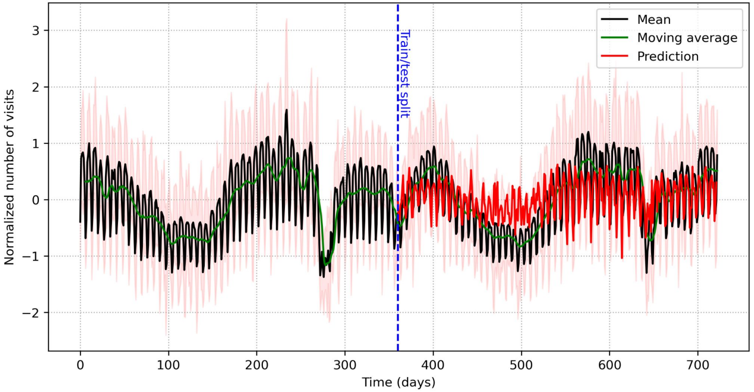
However for visualizing the data is hard that’s why we have taken mean and std deviation of every snapshot

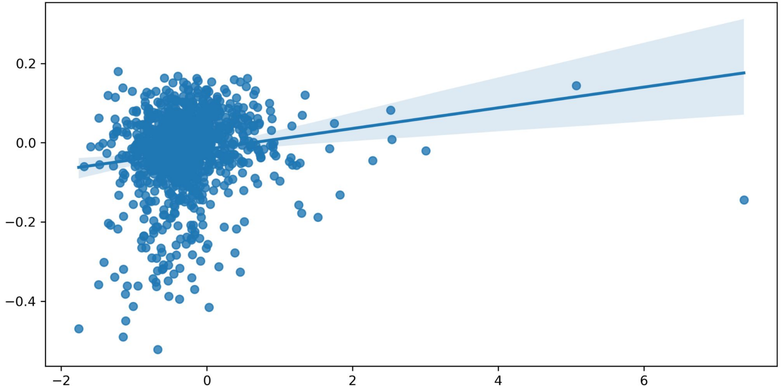


Model:

```
TemporalGNN(
  (recurrent): EvolveGCNH(
    (pooling_layer): TopKPooling(8, ratio=0.00749063670411985, multiplier=1.0)
    (recurrent_layer): GRU(8, 8)
    (conv_layer): GCNConv_Fixed_W(8, 8)
  )
  (linear): Linear(in_features=8, out_features=1, bias=True)
)
```

Results:

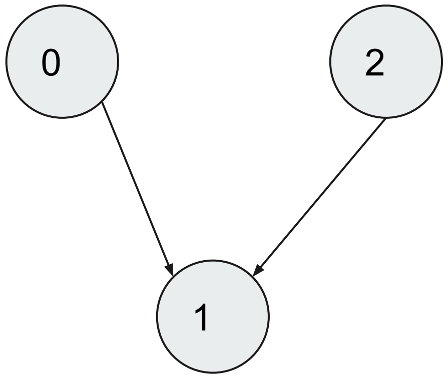




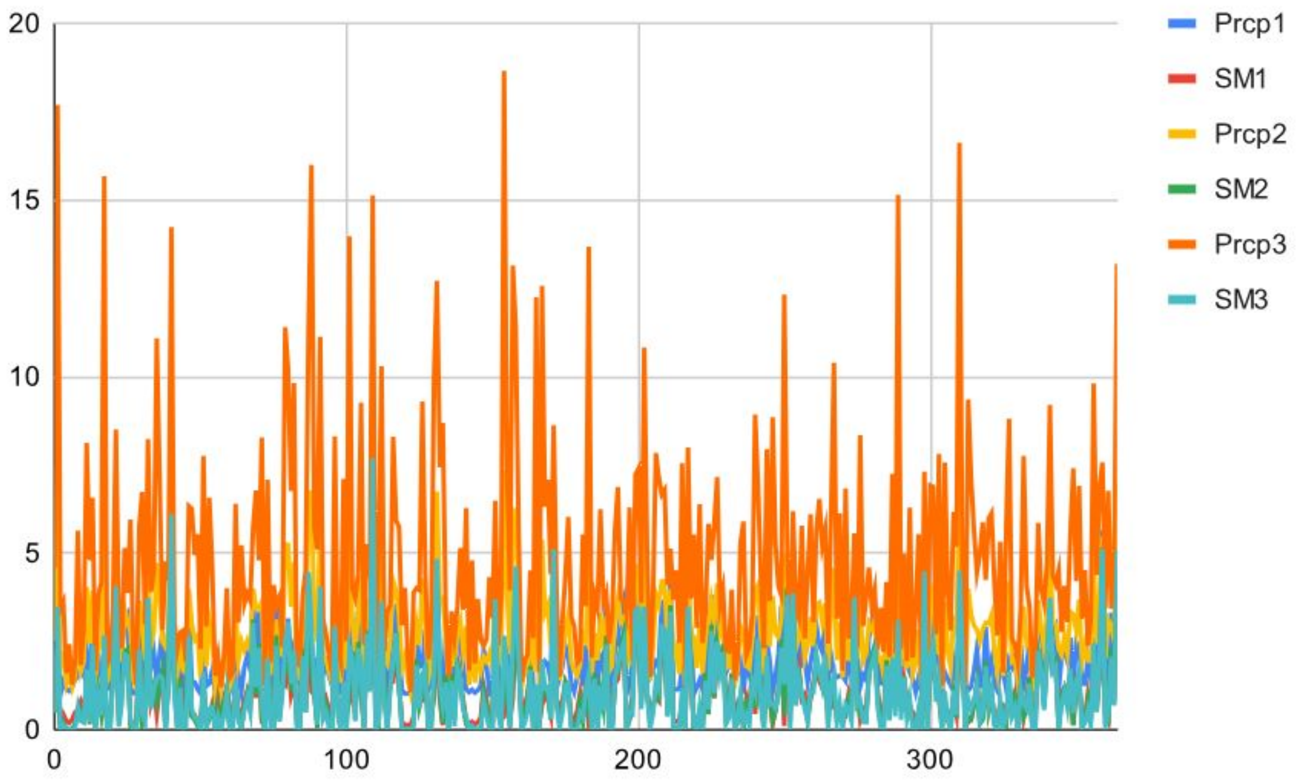
MSE = 0.8194

Intro to Problem

Given three node representing three different locations and every node has two features Soil Moisture and Precipitation, we have given node’s labels as flood(1) or not flood(2) we have to forecast its label using it spatial and temporal Relation

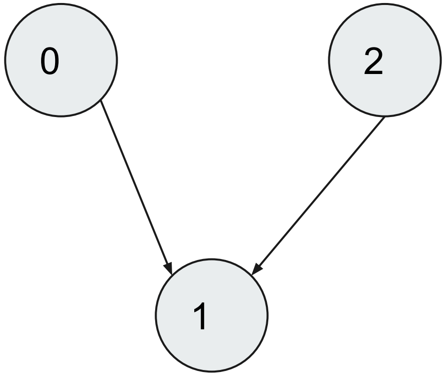


Spatial Relation



Temporal Relation

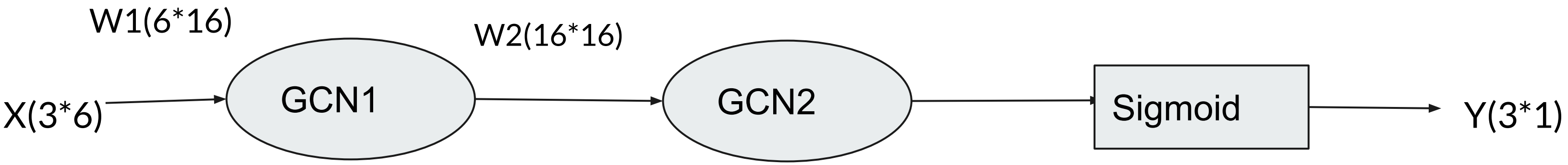
GCN with time window



Time window=3

X=

$a^{[t]}1$	$a^{[t+1]}1$	$a^{[t+2]}1$	$b^{[t]}1$	$b^{[t+1]}1$	$b^{[t+2]}1$
$a^{[t]}2$	$a^{[t+1]}2$	$a^{[t+2]}2$	$b^{[t]}2$	$a^{[t+1]}2$	$a^{[t+2]}2$
$a^{[t]}3$	$a^{[t+1]}3$	$a^{[t+2]}3$	$b^{[t]}3$	$b^{[t+1]}3$	$b^{[t+2]}3$



Evolve GCNO

Input X of shape (3*2)

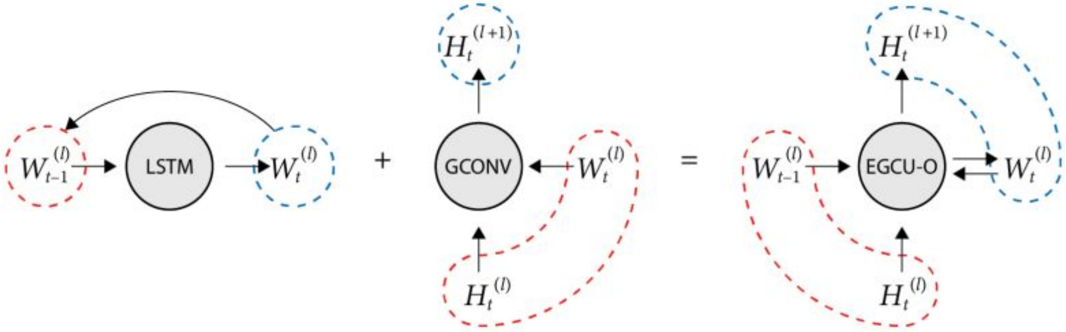
LSTM update the weight matrix W for layer l at time t

$$W_t^{(l)} = \text{LSTM}\,(W_{t-1}^{(l)})$$

This resulting weight matrix is used to calculate the next layer's node embedding

$$H_t^{(l+1)} = \text{GCN}\,(A_t, H_t^{(l)}, W_t^{(t)})$$

$$= \tilde{D}^{-\frac{1}{2}} \tilde{A}^T \tilde{D}^{-\frac{1}{2}} H_t^{(l)} {W_t^{(t)}}^T$$



Evolve GCNH

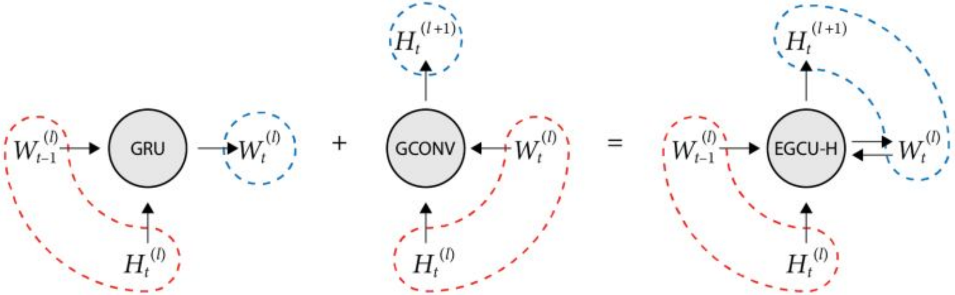
Input X of shape (3*2)
GRU updates the GCN's weight matrix for the layer l at time t

$$W_t^{(l)} = \mathrm{GRU}\left(H_t^{(l)}, W_{t-1}^{(l)}\right)$$

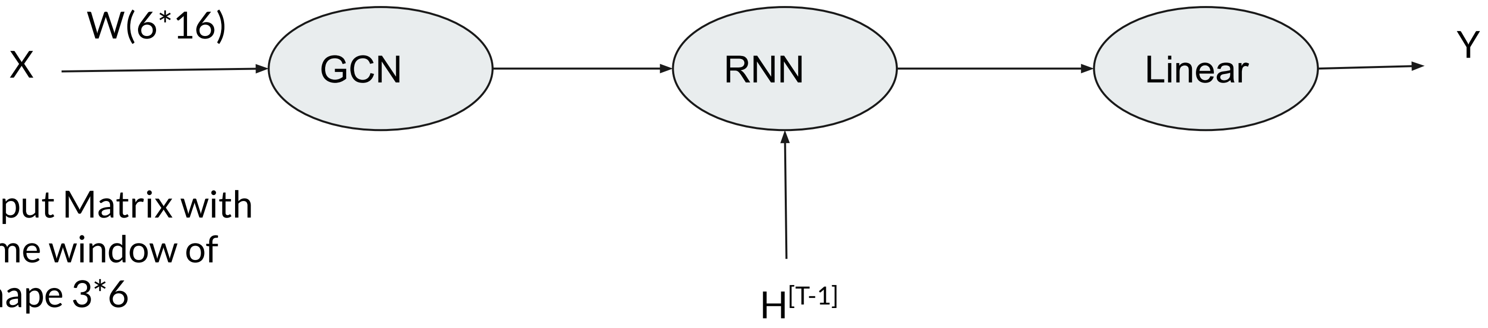
This resulting weight matrix is used to calculate the next layer's node embedding

$$H_t^{(l+1)} = \mathrm{GCN}\left(A_t, H_t^{(l)}, W_t^{(t)}\right)$$

$$= \tilde{D}^{-\frac{1}{2}} \tilde{A}^T \tilde{D}^{-\frac{1}{2}} H_t^{(l)} {W_t^{(t)}}^T$$

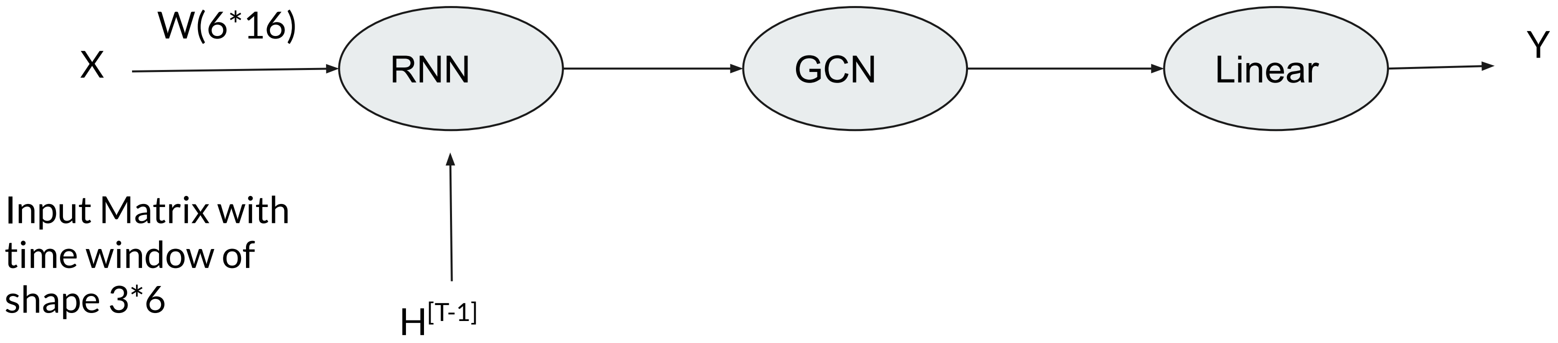


GCN->RNN



Input Matrix with time window of shape 3*6

RNN->GCN



W(6*16)
X
RNN
GCN
Linear
Y
Input Matrix with time window of shape 3*6
H[T-1]

GConvLSTM

$$
\begin{aligned}
i &= \sigma(W_{xi} *_{\mathcal{G}} x_t + W_{hi} *_{\mathcal{G}} h_{t-1} + w_{ci} \odot c_{t-1} + b_i),\\
f &= \sigma(W_{xf} *_{\mathcal{G}} x_t + W_{hf} *_{\mathcal{G}} h_{t-1} + w_{cf} \odot c_{t-1} + b_f),\\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} *_{\mathcal{G}} x_t + W_{hc} *_{\mathcal{G}} h_{t-1} + b_c),\\
o &= \sigma(W_{xo} *_{\mathcal{G}} x_t + W_{ho} *_{\mathcal{G}} h_{t-1} + w_{co} \odot c_t + b_o),\\
h_t &= o \odot \tanh(c_t).
\end{aligned}
$$

GConvGRU

$$
\begin{aligned}
z &= \sigma(W_{xz} *_{\mathcal{G}} x_t + W_{hz} *_{\mathcal{G}} h_{t-1}), \\
r &= \sigma(W_{xr} *_{\mathcal{G}} x_t + W_{hr} *_{\mathcal{G}} h_{t-1}), \\
\tilde{h} &= \tanh(W_{xh} *_{\mathcal{G}} x_t + W_{hh} *_{\mathcal{G}} (r \odot h_{t-1})), \\
h_t &= z \odot h_{t-1} + (1 - z) \odot \tilde{h}.
\end{aligned}
$$

RESULTS(Accuracy Scores)

Model	epochs=10	epochs=20	epochs=50	epochs=100
GCN	91.21	91.94	92.31	95.60
GCN->RNN	91.58	93.04	94.14	94.87
RNN->GCN	95.24	96.70	95.97	97.80
EvolveGCNH	89.13	89.49	90.22	90.30
EvolveGCNO	83.70	85.87	86.59	86.23
GConvGRU	92.39	92.03	93.12	93.39
GConvLSTM	91.78	93.15	93.35	93.69

CONCLUSION

- RNN -> GCN architecture achieved the highest accuracy of 97.80% at 100 epochs, outperforming other models.
- GCN model showed consistent improvement, reaching 95.60% accuracy at 100 epochs.
- GCN -> RNN, GConvGRU, and GConvLSTM also displayed competitive performance with accuracies between 92.03% to 94.87% at 100 epochs.
- EvolveGCNH and EvolveGCNO models had lower accuracies ranging from 83.70% to 90.30%.
- The study highlights the potential of GNNs in flood prediction for Water Resource Engineering, with RNN -> GCN emerging as a promising architecture.

Thank You

References

1. Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." *arXiv preprint arXiv:1609.02907* (2016).
2. Léonard, Nicholas, et al. "rnn: Recurrent library for torch." *arXiv preprint arXiv:1511.07889* (2015).
3. Pareja, Aldo, et al. "Evolvegcn: Evolving graph convolutional networks for dynamic graphs." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. No. 04. 2020.
4. Seo, Youngjoo, et al. "Structured sequence modeling with graph convolutional recurrent networks." *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*. Springer International Publishing, 2018.