PHILIPS
www.philips.com

Dynamic Graph Convolutional Neural Networks

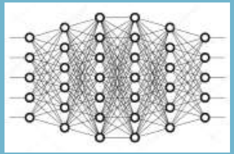
From *DNN* → *CNN & RNN* → *Spectral GCN* → *DGCN*

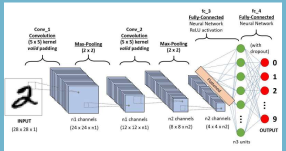
Shuyue Jia
shuyuej@ieee.org
Research Intern @ Tencent & Philips Research
November 2020

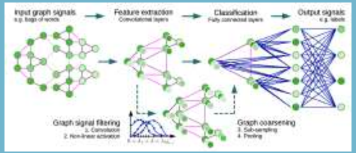
innovation + you

Learning Objectives: From Static to Dynamic Networks

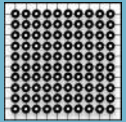
- From DNN → CNN → GCN



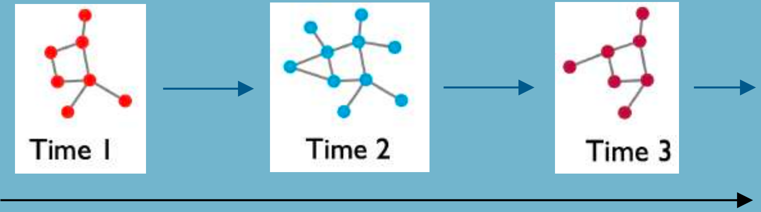




- How to extend CNNs to graph-structured data?
  (Traditional Approach, *Structural Patterns*)

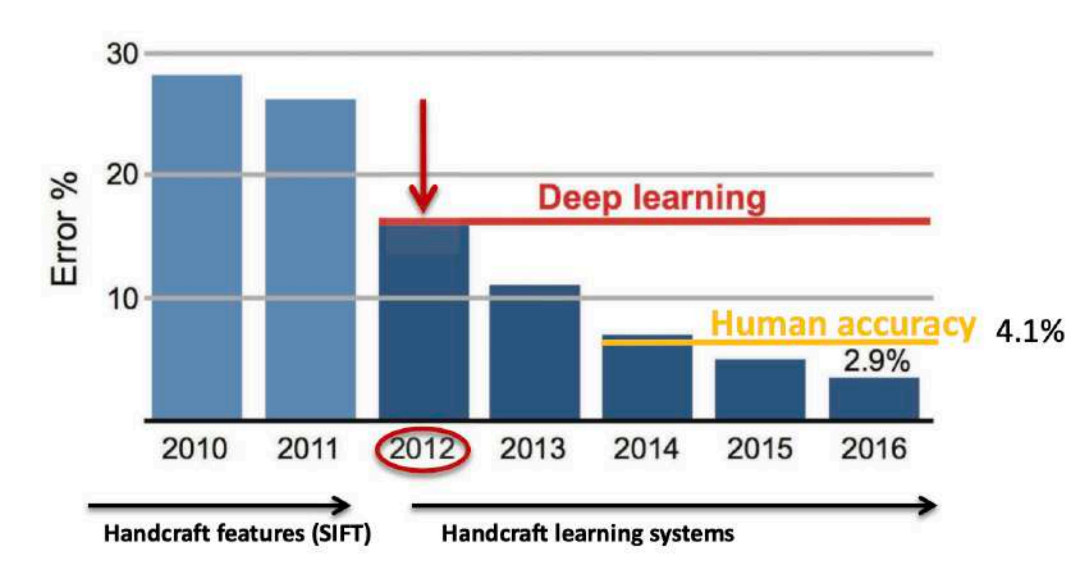


- How to dynamically evolve / learn Graphs through timestamps?
  (Latest Approach, *Structural + Temporal Patterns*)

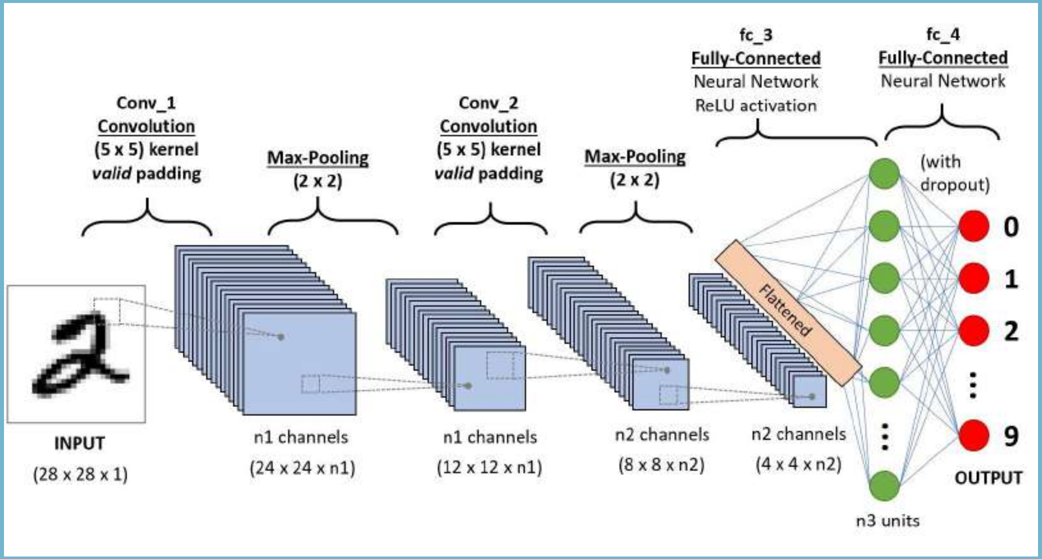


Timestamps

Recall: Deep Learning (DL) - Neural Networks,
Convolutional Neural Networks (CNNs) - for Local-matter Signals,
Supervised Learning - Features Mapped to Labels

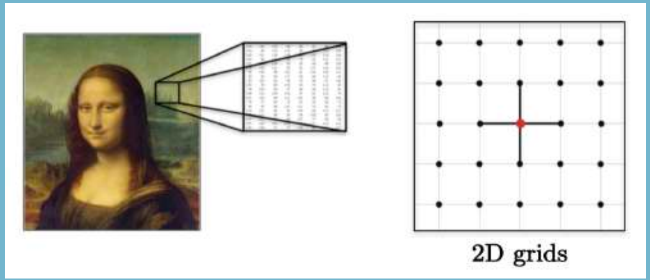


Recall: Traditional *CNNs* (Local Matter)
Automatic Feature Extraction for *Signals in the* ***Euclidean Domain***

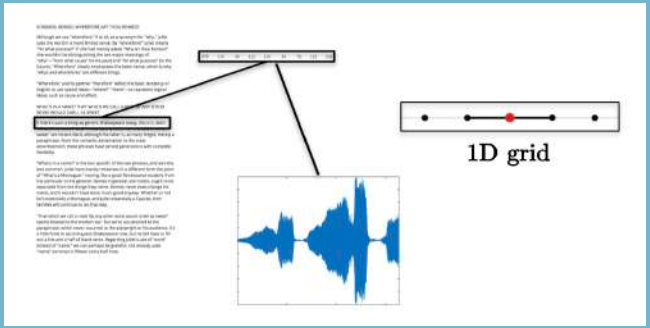


A **Convolutional Neural Network (CNNs)** Architecture includes:

1. Convolutional Layer (Conv)
2. Pooling Layer (Pool)
3. Fully-connected Layer (FC)



Image, volume, video: 2D, 3D → Euclidean domain



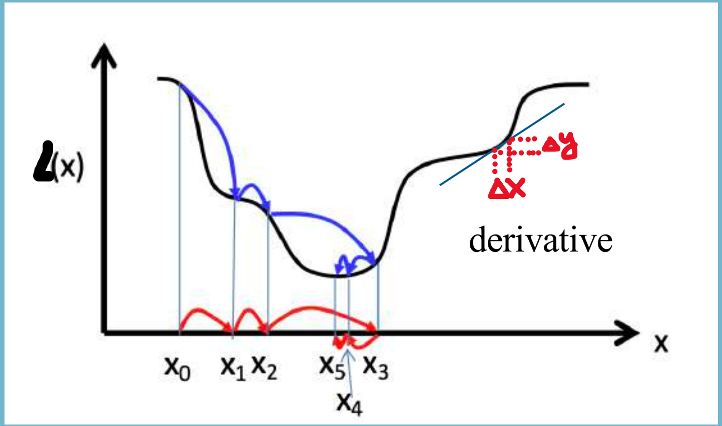
Sentence, word, sound: 1D → Euclidean domain

These domains have nice regular spatial structures.

Recall: CNNs' Fully-connected Layer (Gradient Descent Algorithm to update model parameters)

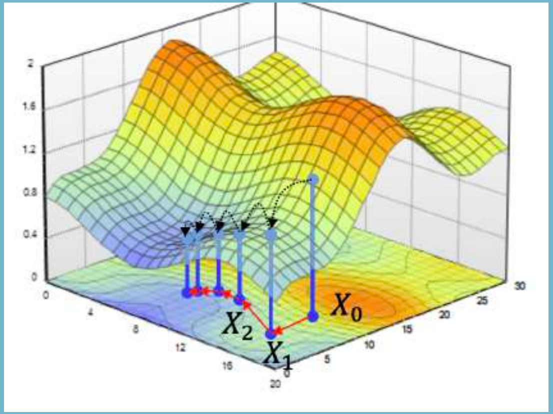
Recall Gradient Descent Algorithm:

Loss Function = $| g(x) - f(x) |_{minimize}$

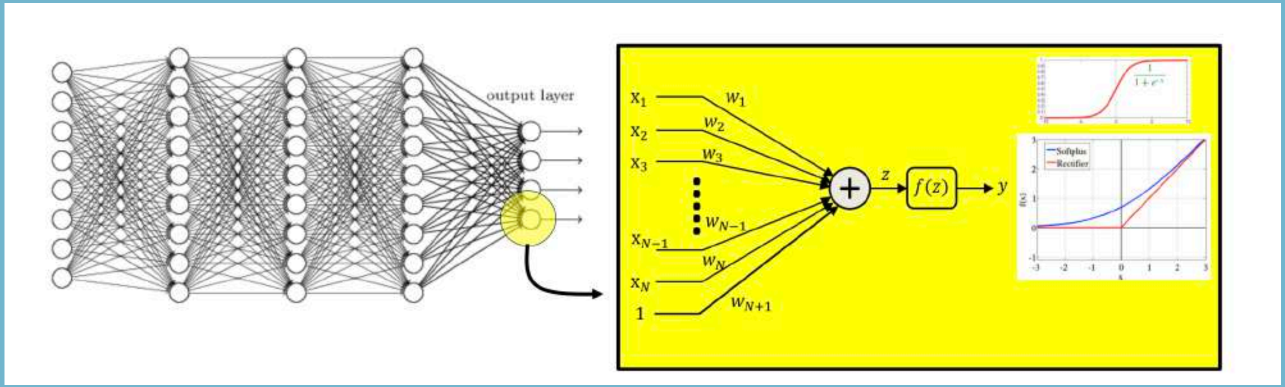


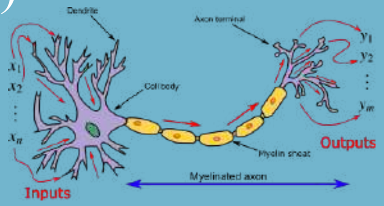
$$\alpha = \frac{\partial(|g(x) - f(x)|)}{\partial x}$$

$$x^{k+1} = x^k - \eta\alpha$$



Recall: CNNs' Fully-connected Layer (**Multi-layer Perceptron**) (Gradient Descent Algorithm to update model parameters)

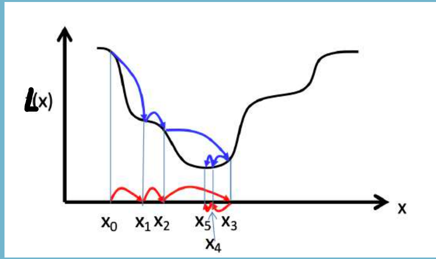




$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1 + e^{-y}}$$

The parameters that we are training are W (weights) and b (biases).



$$x^{k+1} = x^k - \eta\alpha$$

$$\alpha = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial x}$$

$$y = \sum_i w_i x_i + b$$

$$f(y) = \frac{1}{1 + e^{-y}}$$

Derivative (Gradient)

Derivative of w and b w.r.t. Loss Function (error)

$$dw = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial w}$$

$$db = \frac{\partial(\frac{1}{2}(g(x) - f(x))^2)}{\partial b}$$

Gradient Descent

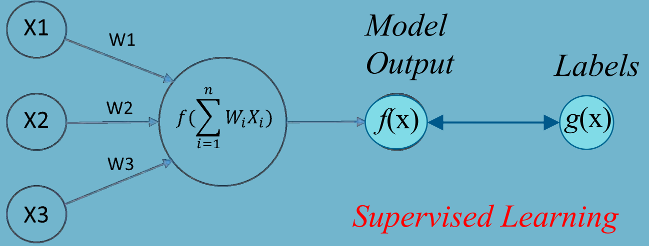
Update Parameters

$$w^{k+1} = w^k - \eta dw$$

$$b^{k+1} = b^k - \eta db$$

Learning Rate

Recall: CNNs' Fully-connected Layer
(Back-propagation (error) Algorithm for model converge)



$$y = \sum_i w_i x_i + b$$

$$f(\mathrm{y}) = \frac{1}{1 + e^{-\mathrm{y}}}$$ Sigmoid Activation Function

Derivative of w and b w.r.t. Loss Function (error)

$$\mathrm{dw} = \frac{\partial(\frac{1}{2}(\mathrm{g}(x) - f(x))^2)}{\partial \mathrm{w}}$$

$$\mathrm{db} = \frac{\partial(\frac{1}{2}(\mathrm{g}(x) - f(x))^2)}{\partial \mathrm{b}}$$

$$\mathrm{L} = \frac{1}{2}(\mathrm{g}(x) - f(x))^2$$

$$y = \sum_i w_i x_i + b$$

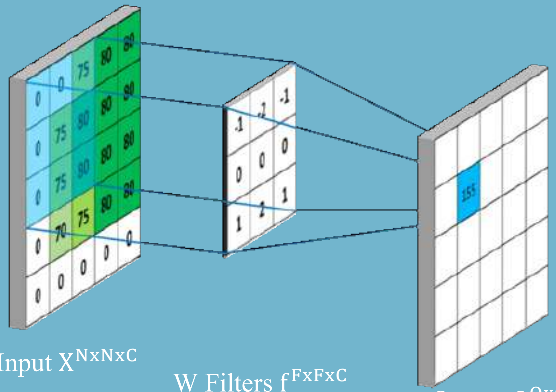
$$f(y) = \frac{1}{1 + e^{-\mathrm{y}}}$$

Chain Rule

$$\mathrm{dw} = \frac{\partial \mathrm{L}}{\partial f(y)} \times \frac{\partial f(y)}{\partial y} \times \frac{\partial y}{\partial w}$$

$$= [\mathrm{g}(x) - f(\mathrm{y})] \times [f(\mathrm{y}) \times (1 - f(\mathrm{y}))] \times X$$

Recall: CNNs' Convolutional Layer (**Weighted Sum**)



Input X^{NxNxC}

W Filters f^{FxFxC}
(kernel)

Output O^{OxOxW}

Convolution

Convolution Output Shape:

$$O = \frac{N + 2P - F}{S} + 1$$

N: Input 2D Signals Size
P: Padding (Zero) Size
F: Filter Size
S: Stride Size

Active Conv (CVRR2017), Deformable Conv (ICCV2017)

Cross Correlation Function, implemented by FFT, O(nlog(n)):

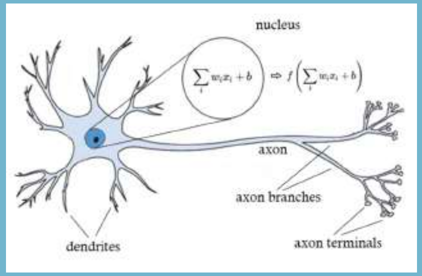
$$O(i, j) = (X * f)(i, j) = \sum_{m}\sum_{n} X(i + m, j + n) f(m, n)$$

Activated by *Rectified Linear Unit (ReLu)* with *Batch Normalization*:

$$\text{neuron} = O(i, j) + b$$

$$BN = \frac{\text{neuron} - \text{batch mean}}{\text{batch Standard Deviation}}$$

$$\text{ReLu} = \max(BN, 0)$$



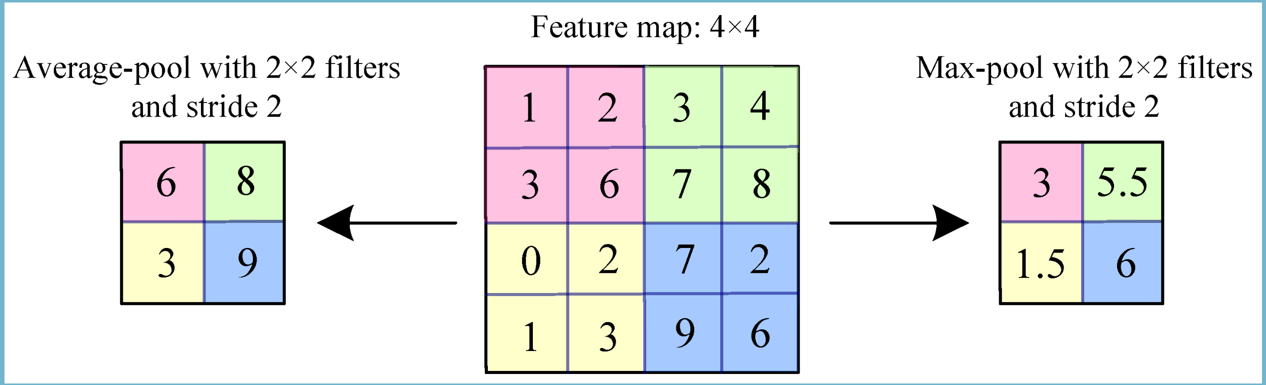
Mathematical Biomedical Neuron

Why Convolutions?

1. Translation & Shift Invariance
2. Weights Sharing and Sparse Connectivity
3. Multi-scale (Hierarchical)

Num of Model Parameters: $F \times F \times C \times W + W = F^2 \times C \times (W + 1)$

Recall: CNNs' Pooling Layer



Averaged Pooling:

$$f_{X,Y} = \text{mean}_{a,b=0}^{1}(S_{2X+a,2Y+b})$$

Max Pooling:

$$f_{X,Y} = \max_{a,b=0}^{1}(S_{2X+a,2Y+b})$$

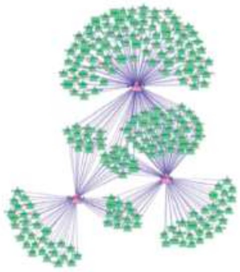
Why Pooling?

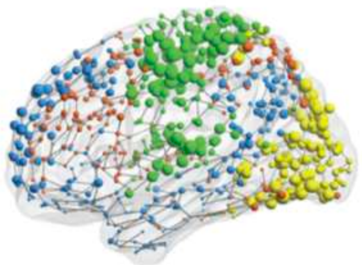
1. Down-sampling + Dimensionality Reduction
2. Enlarge Receptive Fields
3. Enhance Translation Invariance

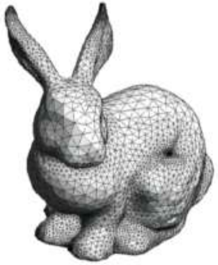
Non-Euclidean data
Graph in the *Non-Euclidean Domain*

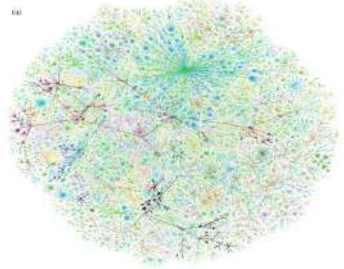
Limitation of Traditional CNN: Cannot handle Graph-structured Signals


Social networks


Regulatory networks

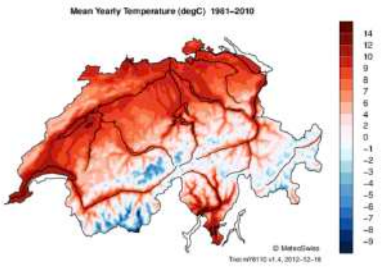

Functional networks


3D shapes

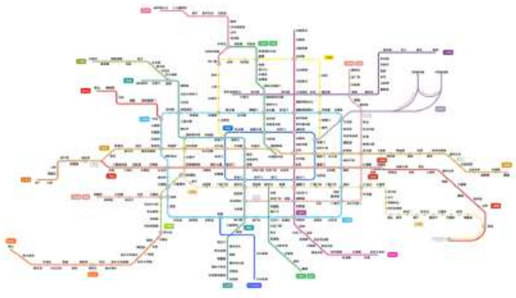

Internet

Electrical data


Traffic data

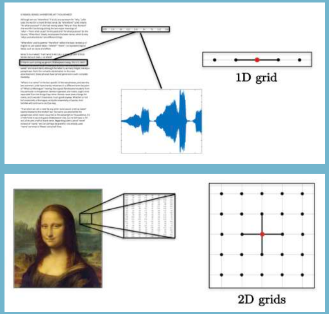


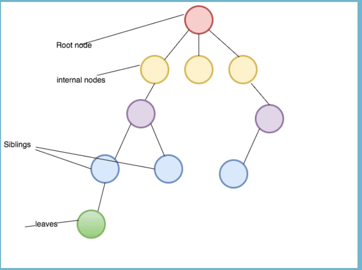
Temperature data

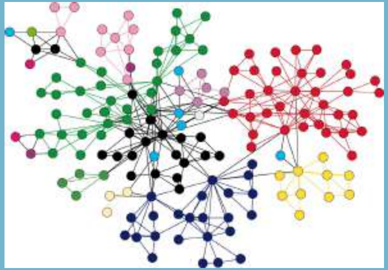

Transportation

Why Graphs?

1/2/3D tensor → Tree (Treebank) → Graph







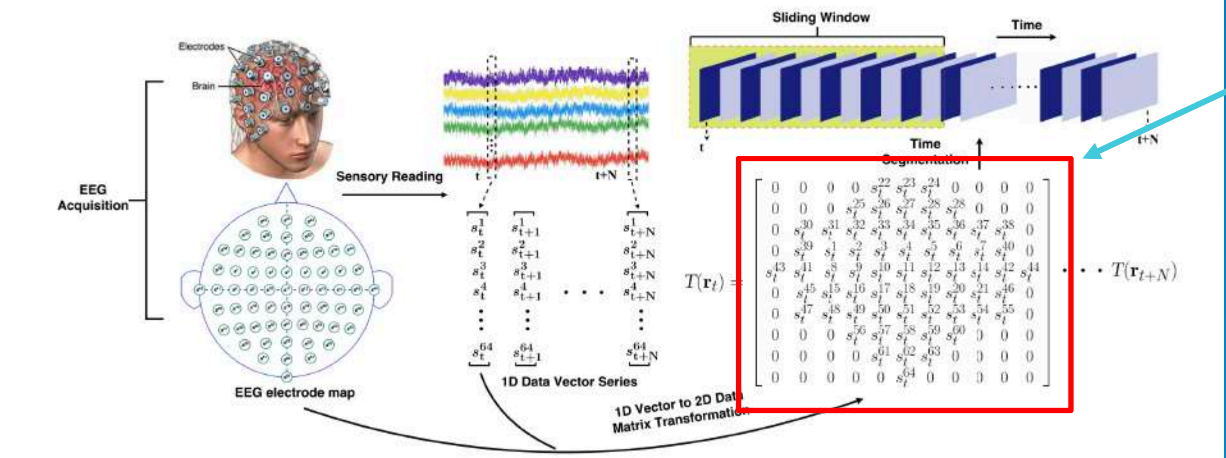
1. Represent *complex relationships* of data
2. Contain *more features* of data
3. Contain *topology information* of data

Key Question:

→ Can we use Traditional CNNs on Graphs directly?

Answer 1: YES, we can!

- *Represent Graph Signals as **2D Mesh** ← Signals in the Euclidean Domain*
- *then use Traditional CNNs or RNNs*



2D Mesh

Fig. 1. Process of EEG data acquisition and spatio-temporal information preserving conversion. EEG signals are first captured using a BCI headset with multiple electrodes and recorded as time series vectors. These vectors are then converted to 2-D data meshes according to the electrode map of the BCI headset. The converted 2-D meshes are finally segmented to clips using the sliding window technique.

Reference: Making Sense of Spatio-Temporal Preserving Representations for EEG-Based Human Intention Recognition, *IEEE Transactions on Cybernetics*, 2019.

Key Question:

→ Can we use Traditional CNNs on Graphs directly?

Answer 2: YES, we can!

- *Use Graph Theory to represent Graph Signals ← Signals in the Euclidean Domain*
- *then use Traditional CNNs or RNNs*

e.g., Adjacency Matrix, Laplacian Matrix

Graph Representation

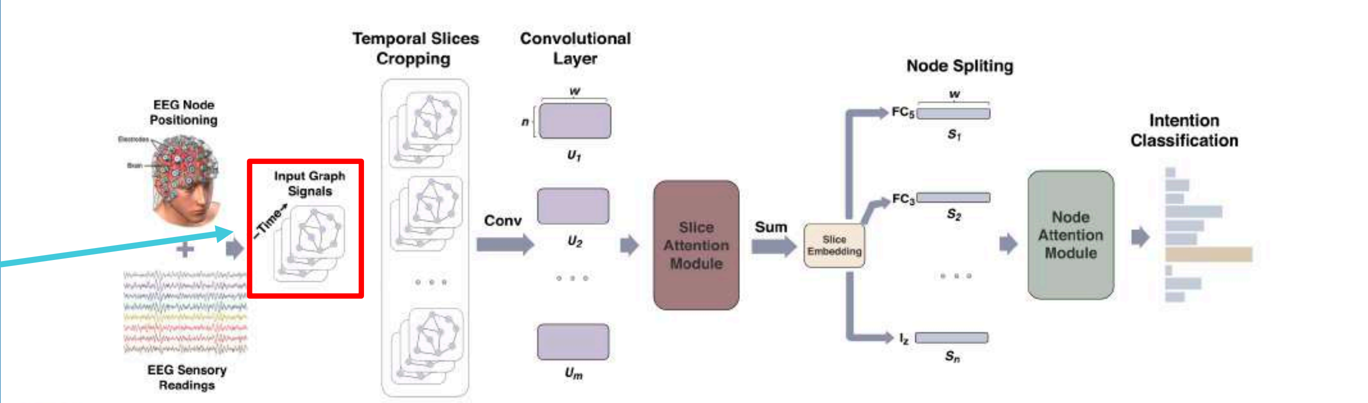


Fig. 1. Overview of the Graph Hierarchical Attention Model (G-HAM) on EEG-based intention recognition. We first embed the raw EEG signal with the node positioning graph; then we apply a sliding window technique to crop continuous EEG sequences into temporal slices and utilize a CNN to extract features of each slice. The first-level attention mechanism is applied to focus on the most discriminative temporal slices; the second-level attention layer targets the most discriminative EEG node and lastly the extracted features are classified to the target intentions using a dense layer with a softmax function.

Reference: **A Graph-Based Hierarchical Attention Model for Movement Intention Detection from EEG Signals**, ***IEEE Transactions on Neural Systems and Rehabilitation Engineering***, 2019.

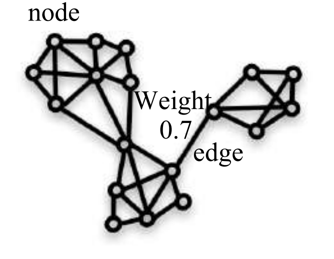
Key Question:

→ Can we use Traditional CNNs on Graphs directly?

Answer 3: NO, we cannot!

- *Graphs are irregular! (1. unordered 2. vary in size)*

→ *Convolution cannot keep **translation invariance** on the non-Euclidean signals*



OUR QUESTION

Can we *intrinsically* and *mathematically* implement CNNs on Graph to learn the node(s) and edge(s) representations?

That's why we discuss GCN here!!!

Problem Definition

Definitions

- *Graph Representation* → Graph Laplacian
- *Graph Convolution* → Spectral Graph Theory
- Vertex-focused V.S. Graph-focused

Temporally fine-grained Model Taxonomy:

- *Static Networks* → Static Network without temporal information
- *Edge Weighted Networks* → Static Network with temporal information as labels on the edge(s) / Node(s)
- *Discrete Networks* → Dynamic Networks in discrete time intervals
- *Continuous Networks* → Dynamic Networks without temporal aggregation

Measurements (for Classification)

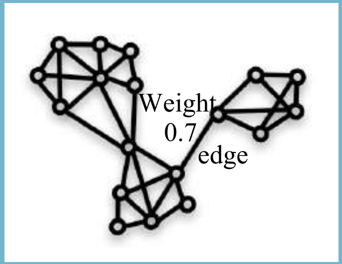
- *Metrics:* Accuracy, Precision, Recall, F1-Score, Confusion Matrix, ROC Curve, AUC, Kappa Coefficient, ……
- *Loss function:* Cross-entropy, Negative Log-likelihood (NLL), ……

Keywords

- Graph Convolutional Neural Networks, Graph CNN, GCN, GNN, …
- Dynamic Graph Convolutional Neural Networks, Dynamic GCN, Dynamic GNN, DGNN, DGCN, …

Graph Representation: **Laplacian Matrix** in Graph Theory

Graph Description: Undirected and Weighted Graph: $G = \{V, E, A\}$



- V: nodes (vertices), $|V| = N$
- E: edges (links) that connected nodes
- A: weights / correlations between nodes

1. Weights
2. Degrees

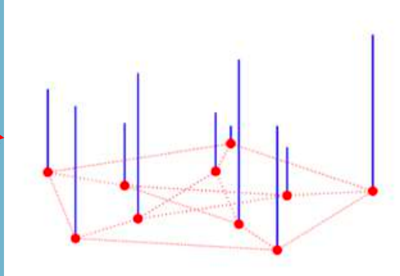
Nodes: different sensors, observations, or data points.
Edges: connections, similarities, or correlations among those points.

Correlations representation: Pearson Matrix

- Measure the linear correlations between nodes
- Below, μ is the expectation, σ is the standard deviation, and $P_{x,y}$ is the Pearson Correlation Coefficient (PCC) between two nodes

$$P_{x,y} = \frac{E((x-\mu_x)(y-\mu_y))}{\sigma_x \sigma_y}$$

- Absolute Pearson Matrix: $|P_{x,y}| \Longrightarrow X \in R^{|v| \times d}$ (Vertex-features Matrix)



Graph Weights representation: Adjacency Matrix: $A \in R^{|v| \times |v|} = |P_{x,y}| - I$, I is an Identity Matrix

Graph Degrees representation: Degree Matrix

$$D_{ii} = \sum_{j=1}^{N} A_{ij}$$

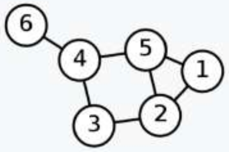
Graph representation: Graph Laplacian (Laplacian Matrix, Combinatorial Laplacian)

$$L = D - A$$

Normalized Graph Laplacian:

$$L = I_N - D^{-\frac{1}{2}} A D^{\frac{1}{2}}$$

Why use **Laplacian Matrix**?

Labelled graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\ 0&3&0&0&0&0\\ 0&0&2&0&0&0\\ 0&0&0&3&0&0\\ 0&0&0&0&3&0\\ 0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\ 1&0&1&0&1&0\\ 0&1&0&1&0&0\\ 0&0&1&0&1&1\\ 1&1&0&1&0&0\\ 0&0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 2&-1&0&0&-1&0\\ -1&3&-1&0&-1&0\\ 0&-1&2&-1&0&0\\ 0&0&-1&3&-1&-1\\ -1&-1&0&-1&3&0\\ 0&0&0&-1&0&1 \end{pmatrix}$

Intuitively

- Contain Graph Weights and Degrees → Represent Graph
- Non-zero: central node and its 1-hop neighbors; The others are all zeros!
- Laplacian Matrix = Discrete Laplace Operator $\vec{\nabla}^2 f = \vec{\nabla} \cdot (\vec{\nabla} f)$

Mathematically

Semi-definite Matrix

» n^{th} orthogonal eigenvectors → **Spectral Decomposition** → Extract graph' Spatial Info from Spectral domain

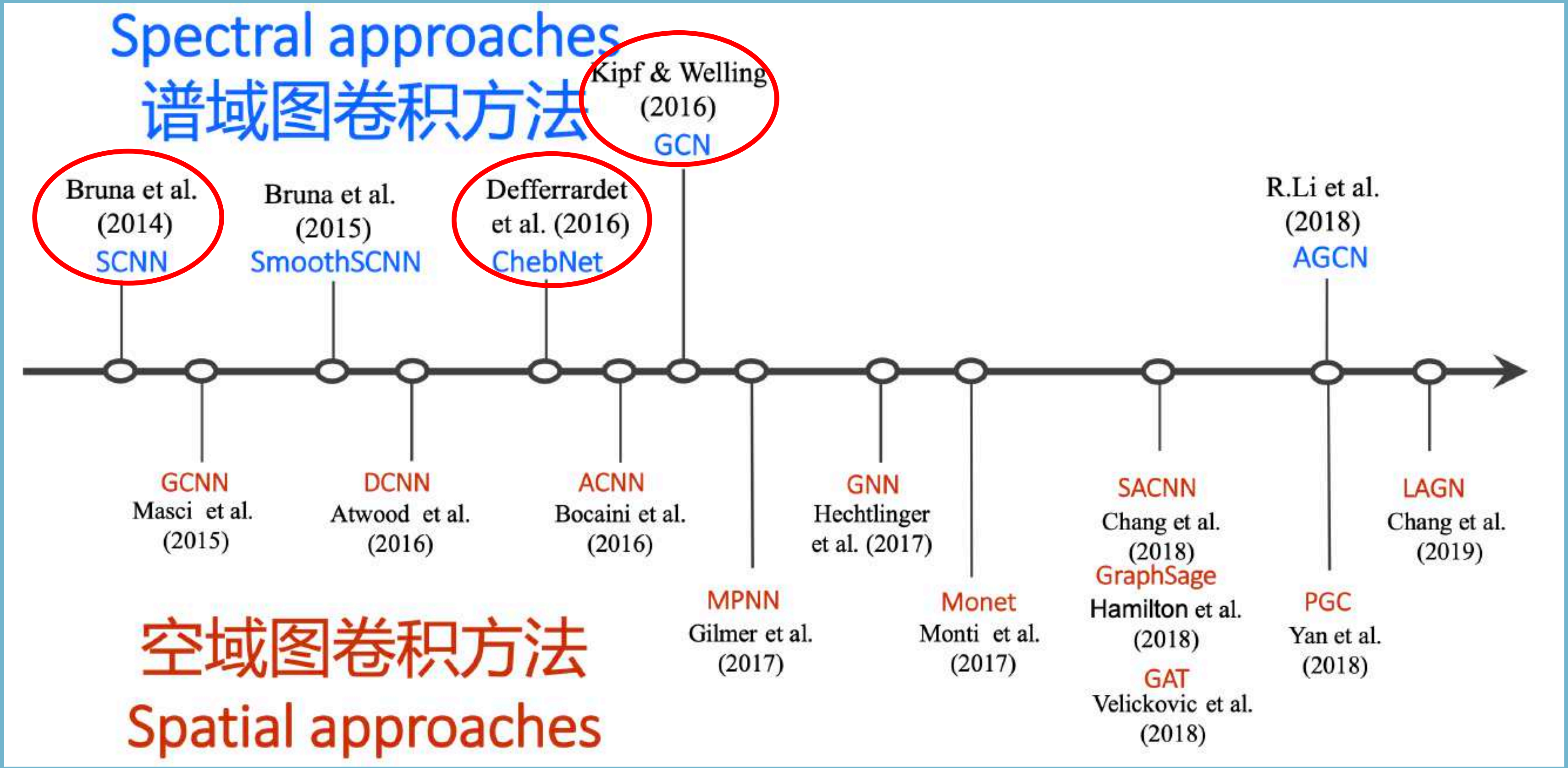
» Eigenvectors = Discrete Laplace Operator's characteristic function (Ch.f.): e^{-iwt}

» All eigenvalues are positive

Symmetric Matrix

» Eigenvectors U → Definite Matrix $U^T U = E$

Graph Convolution Timeline



Spatial Convolution V.S. *Spectral* Convolution

Spatial Convolution (Vertex / Spatial Domain) → *Mainstream* (Until 10/07/2020)

- Applied to *Nodes' Neighbors directly in the Spatial domain* to aggregate features
- Cons:
  1. No static neighbors' structure
  2. Nodes unordered
  3. Output dimension changed, hard to process later
- Representation Model: *GNN*, *GraphSAGE*, *GAT*, *PGC*

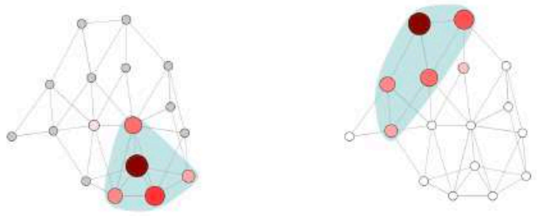


Figure 1: Visualization of the graph convolution size 5. For a given node, the convolution is applied on the node and its 4 closest neighbors selected by the random walk. As the right figure demonstrates, the random walk can expand further into the graph to higher degree neighbors. The convolution weights are shared according to the neighbors' closeness to the nodes and applied globally on all nodes.

Spatial Convolution V.S. *Spectral* Convolution

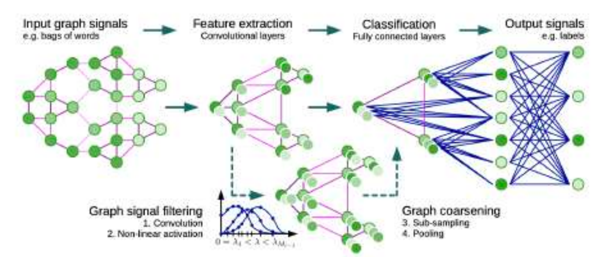


Figure 1: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

Spectral Convolution (Spectral / Frequency Domain)

- Signals (*Spatial*) → Signals (*Frequency*) → Signals (*Spatial*)
- Cons:

1. Only *undirected graphs* are applicable → Cannot use Spectral Convolution

   Lots of scenarios are directed graphs → $W_{ij} \neq W_{ji}$

2. Cannot change *Graph Structure* (Graph Laplacian) during Training
3. SCNN high Time Complexity $O(n^3)$, and ChebNet and GCN few parameters weaken model performance

- Representation Model: *SCNN*, *ChebNet*, *GCN*

Recall: *Spectral Theorem*

Let $A \in R^{nxn}$ be symmetric, and $\lambda_i \in R$ $(i = 1,2,3,\ldots,n)$, n be the eigenvalues of A. There exists a set of orthonormal vectors $u_i \in R_n$ $(i = 1,2,3,\ldots,n)$, such that $Au_i = \lambda_i u_i$. Equivalently, there exists an orthonormal matrix $U = [u_1, u_2, \ldots, u_n] \in R^{nxn}$, such that $UU^T = U^T U = I_n$

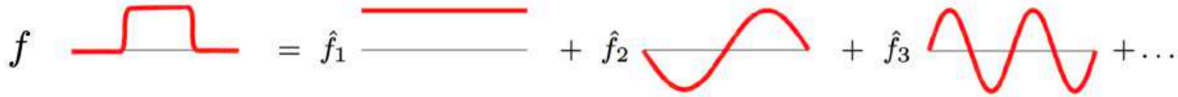
$$A = U\Lambda U^T = \sum_{i=1}^{n} \lambda_i u_i u_i^T$$

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Recall: *Fourier Transform* $\mathrm{F}(\mathrm{w}) = \mathrm{F}[f(t)] = \int f(t)\mathrm{e}^{-iwt}\mathrm{d}t$

- A function $f : [-\pi, \pi] \rightarrow \mathbb{R}$ can be written as **Fourier series:**

$$f(x) = \sum_{k\geq 0} \underbrace{\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x')e^{-ikx'}dx'}_{\hat{f}_k = \langle f, e^{-ikx}\rangle_{L^2([-\pi,\pi])}} e^{-ikx}$$



- Fourier basis e^{-ikx} = **Laplace-Beltrami eigenfunctions:**

$$-\Delta\phi_k = k^2\phi_k$$

$$\begin{cases} \phi_k & = & \text{Fourier mode} \\ k & = & \text{frequency of Fourier mode} \end{cases}$$

Spectral Theorem for Graph Laplacian

$$\mathrm{L} = \mathrm{U}\Lambda\mathrm{U}^{\mathrm{T}}$$

$$\mathrm{LU} = \Lambda\mathrm{U}$$

- U: Fourier modes, which are ***real*** and ***orthonormal*** eigenvectors of L (self-adjointness)
- Λ : Fourier Frequencies, where the diagonal is the ***ordered real nonnegative*** eigenvalues of L (positive-semidefiniteness)

Graph Fourier Transform

can be seen as e^{-iwt}

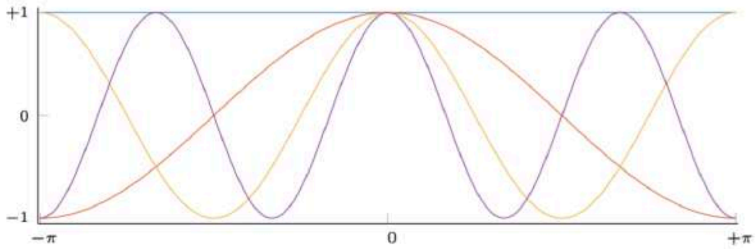
$$\mathrm{F}[f(\lambda_k)] = \hat{f}(\lambda_k) = \langle f, \mathrm{U}_k \rangle = \sum_{i=1}^{n} f(i) * \mathrm{U}_k(i)$$

$$\hat{f}(\lambda) = \mathrm{U}^{\mathrm{T}} f \Leftrightarrow f = \mathrm{U}\hat{f}(\lambda)$$

$\hat{f}(\lambda_k)$ is the projection value of Fourier basis U_k w.r.t. f

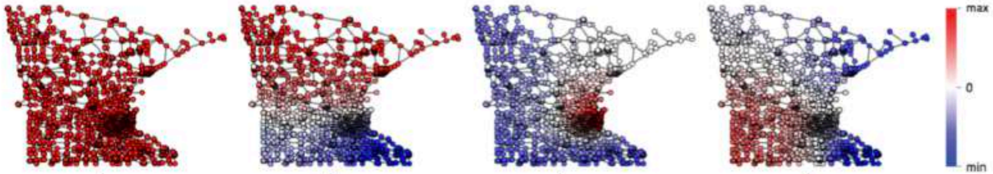
Illustration Fourier Basis

- Euclidean domain:



First eigenvectors of 1D Euclidean Laplacian = standard Fourier basis

- Graph domain:



ϕ_0 ϕ_1 ϕ_2 ϕ_3

First Laplacian eigenvectors of a graph

Lap eigenvectors related to graph geometry
(s.a. communities, hubs, etc), spectral clustering[10]

[10] Von Luxburg 2007

Graph Convolution $F((f * h)_G) = \hat{f}(w) \times \hat{h}(w)$

$$(f * h)_G = F^{-1}(\hat{f}(w) \times \hat{h}(w))$$

$$\hat{f}(\lambda) = U^T f$$

Hamada Product
Element-wise Multiplication

$$(f * h)_G = F^{-1}((U^T f) \odot (U^T h))$$

$$f = U\hat{f}(\lambda)$$

$$(f * h)_G = U((U^T f) \odot (U^T h))$$

If d=1:
Output Shape: [n x n]
otherwise:
Output Shape: [n x d] or [n x n x d]

[n x n] [n x n] [n x n] [n x d]

$$(f * h)_G = U \,\mathrm{diag}[\hat{h}(\lambda_1), \hat{h}(\lambda_2), \ldots, \hat{h}(\lambda_n)] U^T f$$

Graph Convolution "prototype"

Spectral Graph Convolution Timeline

$$(f * h)_G = U\,\mathrm{diag}[\hat{h}(\lambda_1), \hat{h}(\lambda_2), \dots, \hat{h}(\lambda_n)]U^T f$$

NIPS 2014 — NIPS 2016 — ICLR 2017 → Timeline

1st *Generation: SCNN*

$$\mathcal{Y} = \sigma(U g_\theta U^T \chi)$$
$$g_\theta = \mathrm{diag}[\theta_1, \theta_2, \dots, \theta_n]$$

2nd *Generation: ChebNet*

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K} \theta_k L^k \chi\right)$$

3rd *Generation: ChebNet*

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K-1} \theta_k T_k(\hat{L}) \chi\right)$$
$$\hat{L} = \frac{2}{\lambda_{max}} L - I_N$$

4th *Generation: GCN*

$$\mathcal{Y} = \sigma\left(\theta D^{-\frac{1}{2}} \hat{A} D^{-\frac{1}{2}} \chi\right)$$

1^{st} *Generation Graph Convolution: SCNN*

$$(f * h)_G = U\,\mathrm{diag}[\hat{h}(\lambda_1), \hat{h}(\lambda_2), \ldots, \hat{h}(\lambda_n)]\,U^T f$$

Activation Function

$$\mathcal{Y} = \sigma(U g_\theta U^T \chi)$$
$$g_\theta = \mathrm{diag}[\theta_1, \theta_2, \ldots, \theta_n]$$

Cons:

1. Global Convolution → No Local Connection, no Weights Sharing
2. $O(n^3)$ Spectral Decomposition

Model Parameters: n×Input_size×num_filter + num_filter = n×Input_size×(num_filter + 1)

Reference: **Spectral Networks and Locally Connected Networks on Graphs**, ***NIPS***, **2014.**

2^{nd} *Generation Graph Convolution*

$$(f * h)_G = U\,\mathrm{diag}[\hat{h}(\lambda_1), \hat{h}(\lambda_2), \ldots, \hat{h}(\lambda_n)]\,U^T f$$

Activation Function

$$\mathcal{Y} = \sigma(U g_\theta U^T \chi)$$

$$\mathcal{Y} = \sigma(U g_\theta(\Lambda) U^T \chi)$$

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

Approximate

K^{th} Polynomial Function

$$g_\theta(\Lambda) = \sum_{k=0}^{K} \theta_k \Lambda^k$$

$$\mathcal{Y} = \sigma\left(U \sum_{k=0}^{K} \theta_k \Lambda^k U^T \chi\right) = \sigma\left(\sum_{k=0}^{K} \theta_k (U \Lambda^k U^T) \chi\right) = \sigma\left(\sum_{k=0}^{K} \theta_k (U \Lambda U^T)^k \chi\right) = \sigma\left(\sum_{k=0}^{K} \theta_k L^k \chi\right)$$

$$\boxed{\mathcal{Y} = \sigma\left(\sum_{k=0}^{K} \theta_k L^k \chi\right)}$$

Reference: **Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering**, *NIPS*, 2016.

2^{nd} *Generation Graph Convolution*

Convolution: Weighted Sum

Weights Sharing → Translation Invariance

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K} \theta_k L^k \chi\right)$$

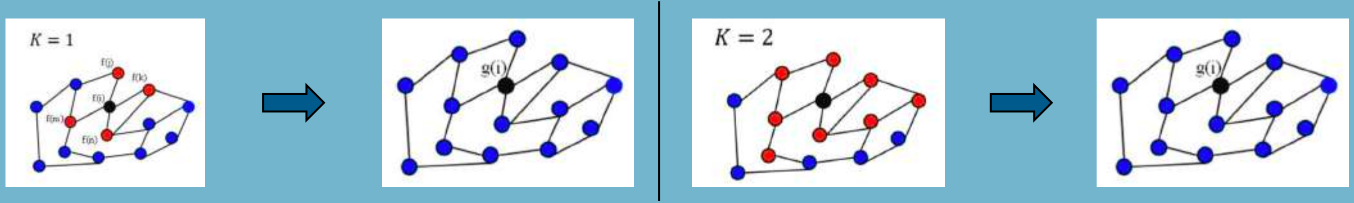
Local connectivity
No need for Fourier

"Node Aggregation"
K is Filter Size

"Laplace Operator"

$$x_{new} \leftarrow Lx_i = \sum_j A_{ij}(x_i - x_j)$$

GCN Key Idea: Use "edge information" to "aggregate" "node information" to generate a new "node representation"



Pros:

1. No need for Spectral Decomposition
2. Less number of parameters (decrease model complexity) → $K \ll n$

Cons:

Need to compute L^k, $O(n^2)$

Model Parameters:

$$K \times Input\_size \times num\_filter + num\_filter = K \times Input\_size \times (num\_filter + 1)$$

3rd *Generation Graph Convolution (ChebNet)*

$$(f * h)_G = U\,\mathrm{diag}[\hat{h}(\lambda_1), \hat{h}(\lambda_2), \dots, \hat{h}(\lambda_n)]\,U^T f$$

$$\mathcal{Y} = \sigma(U g_\theta U^T \chi)$$

$$g_\theta \approx \sum_{k=0}^{K-1} \theta_k T_k(\hat{\Lambda})$$

$$\hat{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I_N$$

Approximate

K^{th} Chebyshev polynomial

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$
$$T_0 = 1$$
$$T_1 = x$$

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K-1} \theta_k T_k(\hat{L})\chi\right) \qquad \hat{L} = \frac{2}{\lambda_{max}} L - I_N$$

Reference: **Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering**, ***NIPS***, **2016.**

3rd *Generation Graph Convolution: ChebNet*

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K-1} \theta_k \, \mathrm{T}_k(\hat{\mathrm{L}})\chi\right)$$

$$\hat{\mathrm{L}} = \frac{2}{\lambda_{\max}}\mathrm{L} - \mathrm{I}_N$$

Pros:

1. No need for Spectral Decomposition
2. No need for L^k
3. Less number of parameters (decrease model complexity) → $K \ll n$
4. O(n) Time Complexity

Model Parameters: K×Input_size×num_filter + num_filter = K×Input_size×(num_filter + 1)

4^{th} *Generation Graph Convolution: GCN*

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{K-1} \theta_k \, T_k(\hat{L})\chi\right)$$

$$\hat{L} = \frac{2}{\lambda_{max}} L - I_N$$

K^{th} Chebyshev polynomial

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$
$$T_0 = 1$$
$$T_1 = x$$

Assume K=1: ← Only consider 1^{th} order Chebyshev Approximation

→ *Two Parameters per filter*

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{1} \theta_k \, T_k(\hat{L})\chi\right) = \sigma(\theta_0 T_0(\hat{L})\chi + \theta_1 T_1(\hat{L})\chi) = \sigma(\theta_0\chi + \theta_1\hat{L}\chi)$$

Assume λ_{max} =2:

$$\mathcal{Y} = \sigma(\theta_0\chi + \theta_1\hat{L}\chi) = \sigma(\theta_0\chi + \theta_1(L - I_N)\chi)$$

$$L = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$$

Reference: **Semi-Supervised Classification with Graph Convolutional Networks**, ***ICLR***, **2017.**

4^{th} *Generation Graph Convolution: GCN*

Assume K=1:

$$\mathcal{Y} = \sigma\left(\sum_{k=0}^{1} \theta_k \mathrm{T}_k(\hat{\mathrm{L}})\chi\right) = \sigma(\theta_0 \mathrm{T}_0(\hat{\mathrm{L}})\chi + \theta_1 \mathrm{T}_1(\hat{\mathrm{L}})\chi) = \sigma(\theta_0\chi + \theta_1\hat{\mathrm{L}}\chi)$$

Assume λ_{max} =2:

$$\mathcal{Y} = \sigma(\theta_0\chi + \theta_1\hat{\mathrm{L}}\chi) = \sigma(\theta_0\chi + \theta_1(\mathrm{L} - \mathrm{I_N})\chi)$$

$$\mathrm{L} = \mathrm{D}^{-\frac{1}{2}}(\mathrm{D} - \mathrm{A})\mathrm{D}^{-\frac{1}{2}}$$

$$\rightarrow \quad \mathcal{Y} = \sigma\left(\theta_0\chi + \theta_1(-\mathrm{D}^{-\frac{1}{2}}\mathrm{A}\mathrm{D}^{-\frac{1}{2}})\chi\right)$$

Assume $\theta = \theta_0 = -\theta_1$: *One Parameter*

Eigenvalues $\in [0, 2]$

$$\mathcal{Y} = \sigma\left(\theta(\mathrm{I_N} + \mathrm{D}^{-\frac{1}{2}}\mathrm{A}\mathrm{D}^{-\frac{1}{2}})\chi\right)$$

Assume $\hat{\mathrm{A}} = \mathrm{I_N} + \mathrm{A}$:
(renormalization trick)

$$\mathcal{Y} = \sigma\left(\theta\mathrm{D}^{-\frac{1}{2}}\hat{\mathrm{A}}\mathrm{D}^{-\frac{1}{2}}\chi\right)$$

$$\mathrm{H}^{(l+1)} = \mathrm{D}^{-\frac{1}{2}}\hat{\mathrm{A}}\mathrm{D}^{-\frac{1}{2}}\mathrm{H}^{(l)}\mathrm{W}^{(l)}$$

4^{th} *Generation Graph Convolution: GCN*

$$H^{(l+1)} = D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}H^{(l)}W^{(l)}$$

Pros:

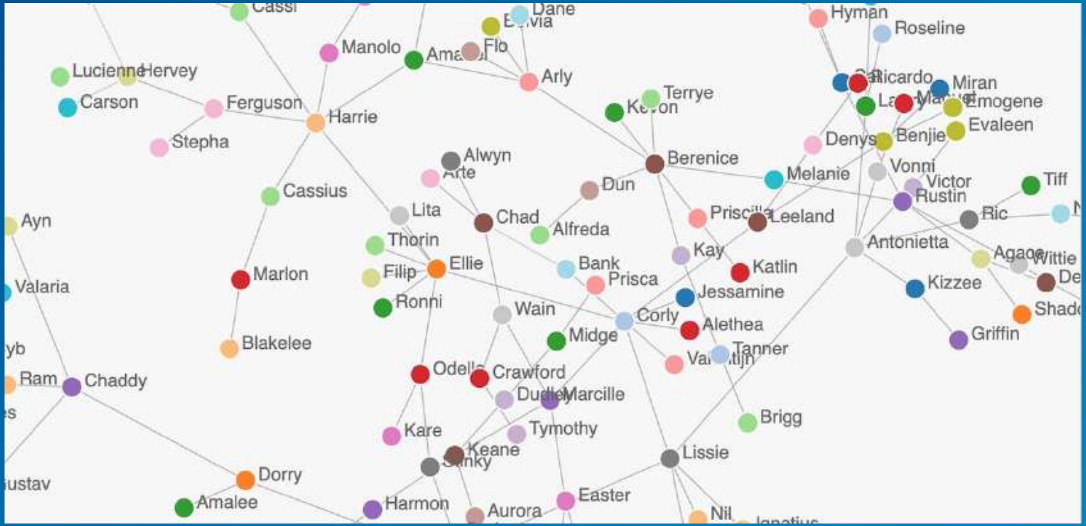
1. Few trainable parameters: *one parameter per filter*
2. Only concern *one-hop neighbor*: Stacked GCN layer → enlarge receptive fields

Cons:

Few trainable parameters → Weaken the capability of the model

Model Parameters: Input_size×num_filter + num_filter = Input_size×(num_filter + 1)

The GCN Models are AWESOME for Graph Signals!!!!!



(Traditional GCN Model) Cons:

1. inability to manage *dynamic vertex features*
2. inability to manage *dynamic edge connections*

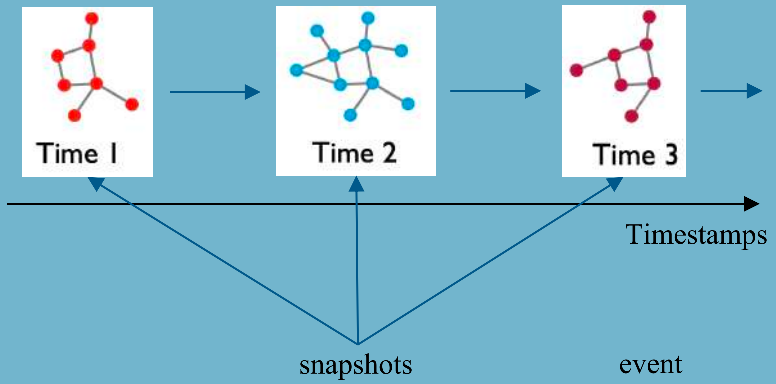
As for real-life scenarios, the Graph should be *dynamically changed through time*!!!

Problem Definition

Definitions

- *Dynamic Network*: a network that changes over time (Time-Varying)
- *Dynamic Graph Neural Networks*: Graph Nodes (Node Dynamics) and Edges (Link Duration) appear and/or disappear over time

Exploit graph spatial and dynamic (temporal) information about data



Problem Definition

Dynamic GNN: Graph Nodes (Node Dynamics) and Edges (Link Duration) appear and/or disappear over time

Graph Description: Undirected and unweighted Graph: $G = \{V, E\}$

– V: nodes (vertices), $V = \{(v, t_s, t_e)\}$

– E: edges (links) that connected nodes, $E = \{(u, v, t_s, t_e)\}$

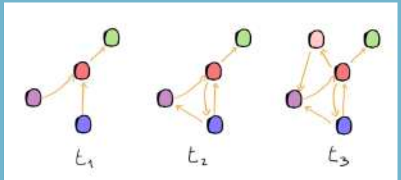
– t_s: start timestamp, t_e: end timestamp

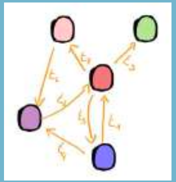
– $u, v \in V$

Taxonomy

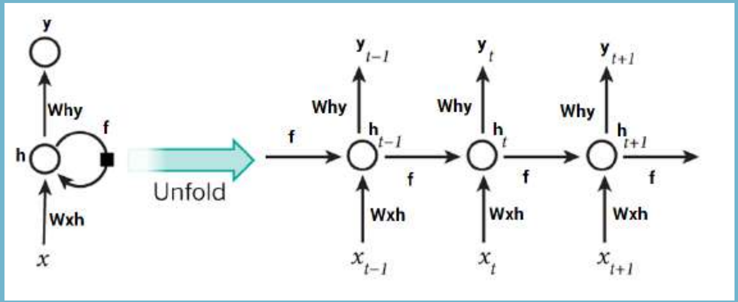
1. *Temporal Networks*: highly dynamic
2. *Evolving Networks:* Links persist longer

1. *Continuous Networks:* sequence of snapshots
2. *Discrete Networks:* sequence of time-events

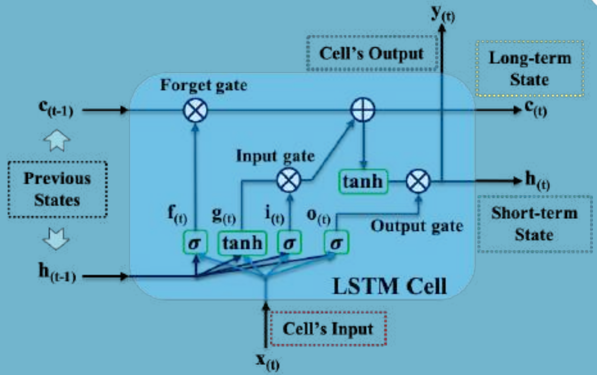




Recall: RNN-based Model (Order Matter) for Time-series (Sequence) Signals



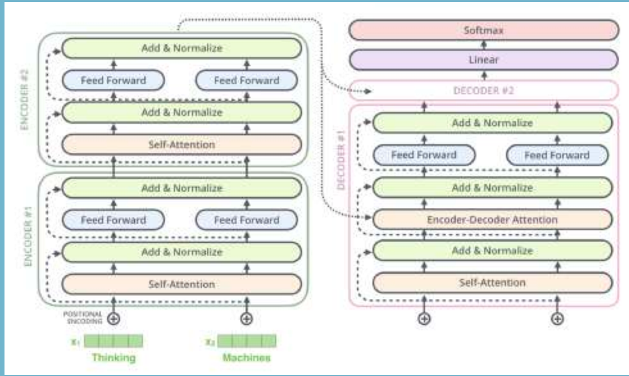
RNN-based Architecture
Input: [max_time x Input_dim]



LSTM Model

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

LSTM Equations
Num of Parameters: $4\times(\text{Input\_Size}\times h+h^2+h)$

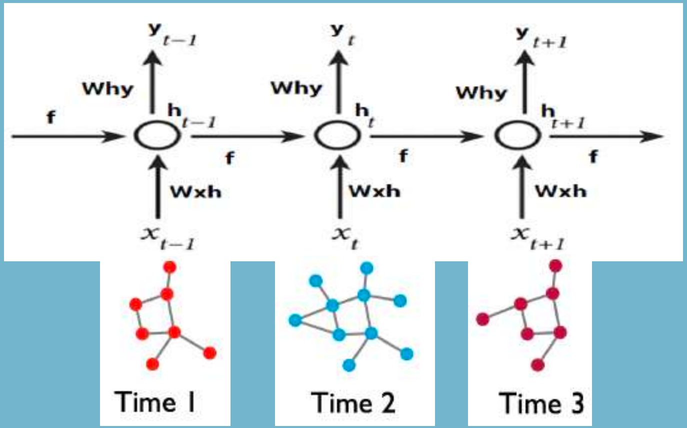


Current Popular Model
Transformer

Pros: *finding long and short range sequence dependencies*
Cons: *lack the ability to explicitly exploit graph-structured information*

Problem Definition

Dynamic GNN: Graph Nodes (Node Dynamics) and Edges (Link Duration) appear and/or disappear over time



Can we combine them??

GCN + RNN

Our Question: Can we use *GCN* ☞ encode graph structures

RNN ☞ process temporal information

→ **Leverage structural and temporal patterns**

Dynamic GCN Models:

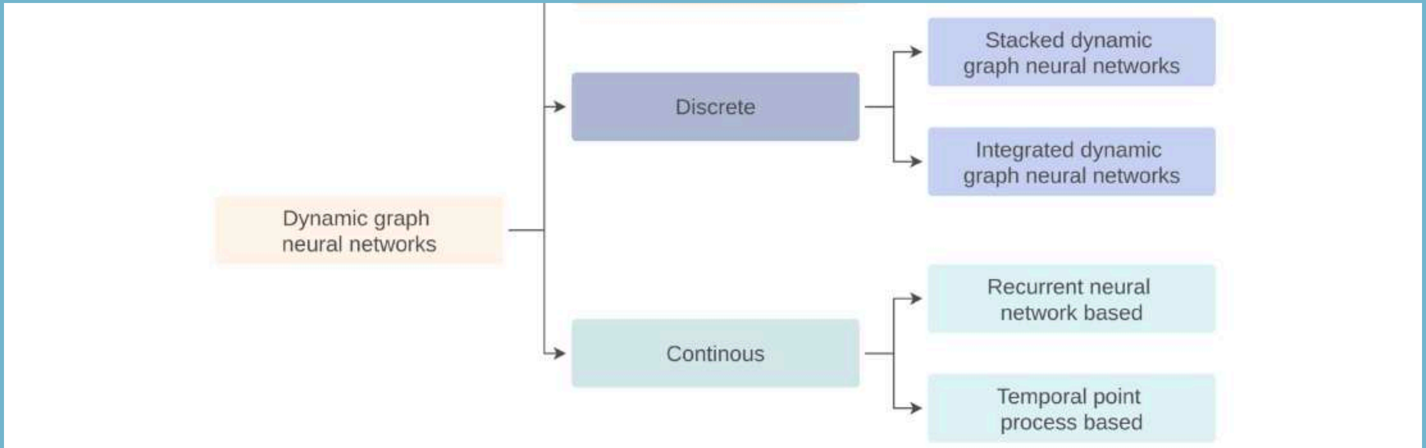
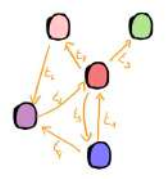
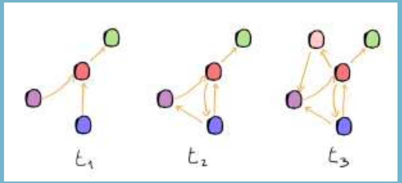


Fig. 6: An overview of dynamic graph neural networks. The main distinction is between discrete and continuous models. This is an extension of Fig 5.



1. *Continuous Networks:* sequence of time-events

2. *Discrete Networks:* sequence of snapshots



Basic Idea:

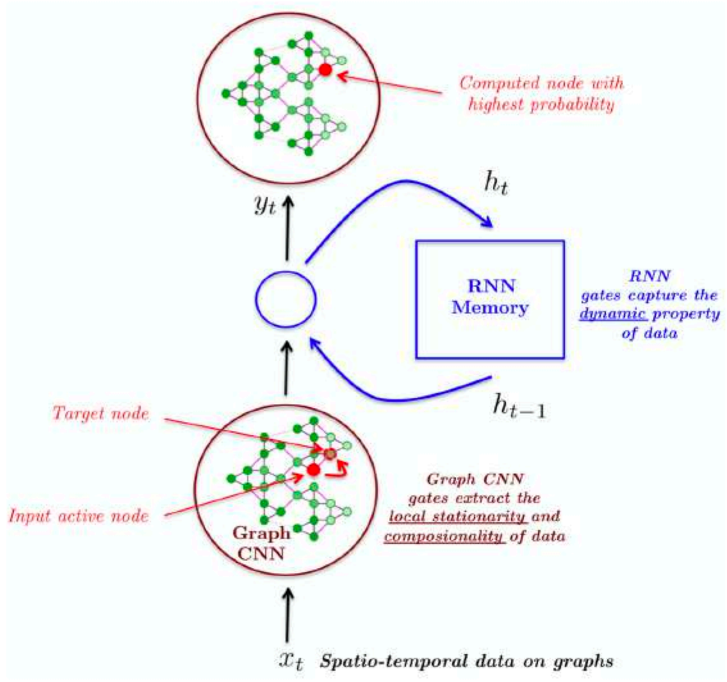


Figure 1: Illustration of the proposed GCRN model for spatio-temporal prediction of graph-structured data. The technique combines at the same time CNN on graphs and RNN. RNN can be easily exchanged with LSTM or GRU networks.

Discrete Model 1:

Be advised that, theoretically,

this "GC" can be any Graph Convolution we discussed before!!

Stacked DGNNs **(Conv + RNN) : GNN** → Graph Structural Patterns; **RNN** → Temporal Patterns

$A \in R^{nxn}$: Adjacency Matrix

$x \in R^{nxd}$: Nodes' Features

Filters are learning the structural patterns of each snapshot!

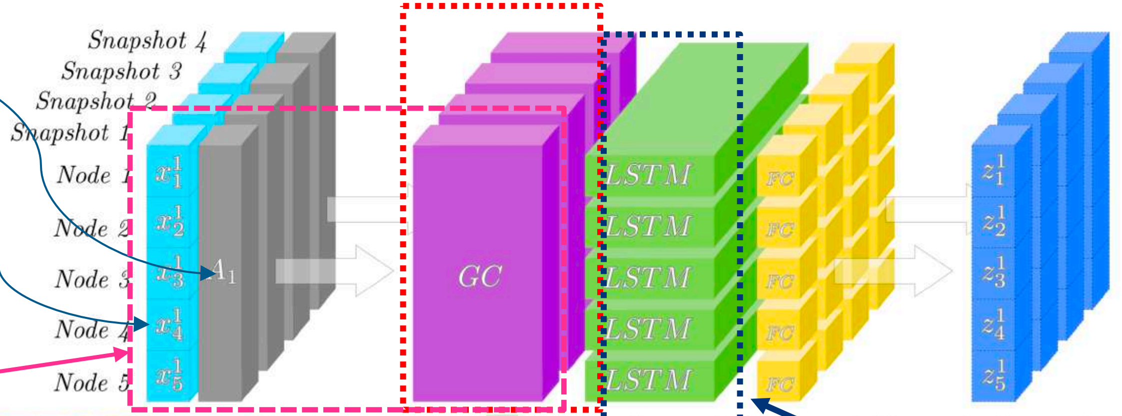


Fig. 7: Stacked DGNN structure from Manessi *et al.* [65]. The graph convolution layer (GC) encode the graph structure in each snapshot while the LSTMs encode temporal patterns.

$Z \in R^{n\,x\,l}$

$$Z_1, \dots, Z_t = \mathrm{GNN}(A_1, X_1), \dots, \mathrm{GNN}(A_t, X_t)$$

$H \in R^{k\,x\,n\,x\,t}$

$$H = \mathrm{vLSTM}_k(Z_1, \dots, Z_t) = \begin{pmatrix} \mathrm{LSTM}_k(V_1{}'Z_1, \dots, V_1{}'Z_t) \\ \dots \\ \mathrm{LSTM}_k(V_n{}'Z_1, \dots, V_n{}'Z_t) \end{pmatrix}$$

RNN Model

(RNN,LSTM, GRU, Transformer,…)

Reference: **Structured Sequence Modeling with Graph Convolutional Recurrent Networks**, *ICLR*, 2017.

Discrete Model 1: *Stacked DGNNs* (Conv + RNN)

The mathematics of the GC layer [14] and the LSTM [16] are here briefly recalled, since they are the basic building blocks of the contribution of this paper. Given a graph with adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ and vertex-feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times d}$, the GC layer with M output nodes and $\boldsymbol{B} \in \mathbb{R}^{d\times M}$ weight matrix is defined as the function:

$$\mathrm{GC}^{\boldsymbol{B}}_{M,\boldsymbol{A}} : \mathbb{R}^{|\mathcal{V}|\times d} \to \mathbb{R}^{|\mathcal{V}|\times M}$$

Reshape → $\mathrm{R}^{|v|M \times 1}$

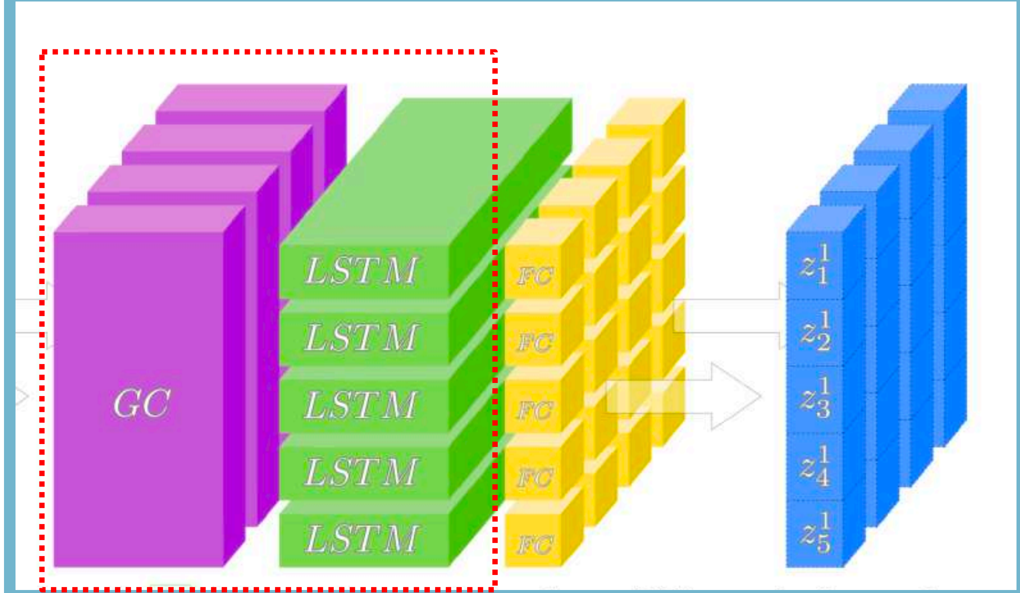
$$\mathrm{GC}^{\boldsymbol{B}}_{M,\boldsymbol{A}}(\boldsymbol{X}) := \mathrm{ReLU}(\hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{B}), \qquad (1)$$

where $\hat{\boldsymbol{A}}$ is the re-normalized adjacency matrix, i.e. $\hat{\boldsymbol{A}} := \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$ with $\tilde{\boldsymbol{A}} := \boldsymbol{A} + \boldsymbol{I}_{|\mathcal{V}|}$ and $[\tilde{\boldsymbol{D}}]_{kk} := \sum_l [\tilde{\boldsymbol{A}}]_{kl}$. Note that the GC layer can be seen as localized first-order approximation of spectral graph convolution [34], with the additional *renormalization trick* in order to improve numerical stability [14].

Given the sequence $(\boldsymbol{x}_i)_{i\in\mathbb{Z}_T}$ with $\boldsymbol{x}_i$ d-dimensional row vectors for each $i \in \mathbb{Z}_T$, a *returning sequence-LSTM* with N output nodes, is the function $\mathrm{LSTM}_N : (\boldsymbol{x}_i)_{i\in\mathbb{Z}_T} \mapsto (\boldsymbol{h}_i)_{i\in\mathbb{Z}_T}$, with $\boldsymbol{h}_i \in \mathbb{R}^N$ and

$$\begin{aligned}
\boldsymbol{h}_i &= \boldsymbol{o}_i \odot \tanh(\boldsymbol{c}_i), & \boldsymbol{f}_i &= \sigma(\boldsymbol{x}_i\boldsymbol{W}_f + \boldsymbol{h}_{i-1}\boldsymbol{U}_f + \boldsymbol{b}_f),\\
\boldsymbol{c}_i &= \boldsymbol{j}_i \odot \tilde{\boldsymbol{c}}_i + \boldsymbol{f}_i \odot \boldsymbol{c}_{i-1}, & \boldsymbol{j}_i &= \sigma(\boldsymbol{x}_i\boldsymbol{W}_j + \boldsymbol{h}_{i-1}\boldsymbol{U}_j + \boldsymbol{b}_j),\\
\boldsymbol{o}_i &= \sigma(\boldsymbol{x}_i\boldsymbol{W}_o + \boldsymbol{h}_{i-1}\boldsymbol{U}_o + \boldsymbol{b}_o), & \tilde{\boldsymbol{c}}_i &= \sigma(\boldsymbol{x}_i\boldsymbol{W}_c + \boldsymbol{h}_{i-1}\boldsymbol{U}_c + \boldsymbol{b}_c),
\end{aligned} \qquad (2)$$

where $\odot$ is the Hadamard product, $\sigma(x) := 1/(1+e^{-x})$, $\boldsymbol{W}_l \in \mathbb{R}^{d\times N}$, $\boldsymbol{U}_l \in \mathbb{R}^{N\times N}$ are weight matrices and $\boldsymbol{b}_l$ are bias vectors, with $l \in \{o, f, j, c\}$.



Discrete Model 1:

Stacked DGNNs **(Conv + RNN) : GNN** → Graph Structural Patterns; **RNN** → Temporal Patterns

Model 1. The most straightforward definition is to stack a graph CNN, defined as (5), for feature extraction and an LSTM, defined as (2), for sequence learning:

$$
\begin{aligned}
x_t^{\text{CNN}} &= \text{CNN}_{\mathcal{G}}(x_t) \\
i &= \sigma(W_{xi} x_t^{\text{CNN}} + W_{hi} h_{t-1} + w_{ci} \odot c_{t-1} + b_i), \\
f &= \sigma(W_{xf} x_t^{\text{CNN}} + W_{hf} h_{t-1} + w_{cf} \odot c_{t-1} + b_f), \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} x_t^{\text{CNN}} + W_{hc} h_{t-1} + b_c), \\
o &= \sigma(W_{xo} x_t^{\text{CNN}} + W_{ho} h_{t-1} + w_{co} \odot c_t + b_o), \\
h_t &= o \odot \tanh(c_t).
\end{aligned}
\tag{8}
$$

In that setting, the input matrix $x_t \in \mathbb{R}^{n \times d_x}$ may represent the observation of d_x measurements at time t of a dynamical system over a network whose organization is given by a graph $\mathcal{G}$. x_t^{CNN} is the output of the graph CNN gate. For a proof of concept, we simply choose here $x_t^{\text{CNN}} = W^{\text{CNN}} *_{\mathcal{G}} x_t$, where $W^{\text{CNN}} \in \mathbb{R}^{K \times d_x \times d_x}$ are the Chebyshev coefficients for the graph convolutional kernels of support K. The model also holds spatially distributed hidden and cell states of size d_h given by the matrices $c_t, h_t \in \mathbb{R}^{n \times d_h}$. Peepholes are controlled by $w_{c\cdot} \in \mathbb{R}^{n \times d_h}$. The weights $W_{h\cdot} \in \mathbb{R}^{d_h \times d_h}$ and $W_{x\cdot} \in \mathbb{R}^{d_h \times d_x}$ are the parameters of the fully connected layers. An architecture such as (8) may be enough to capture the data distribution by exploiting local stationarity and compositionality properties as well as the dynamic properties.

Discrete Model 2:

Integrated DGNNs **(Idea is from ConvLSTM)**

$A \in R^{n \times n}$: Adjacency Matrix

$x \in R^{n \times d}$: Nodes' Features

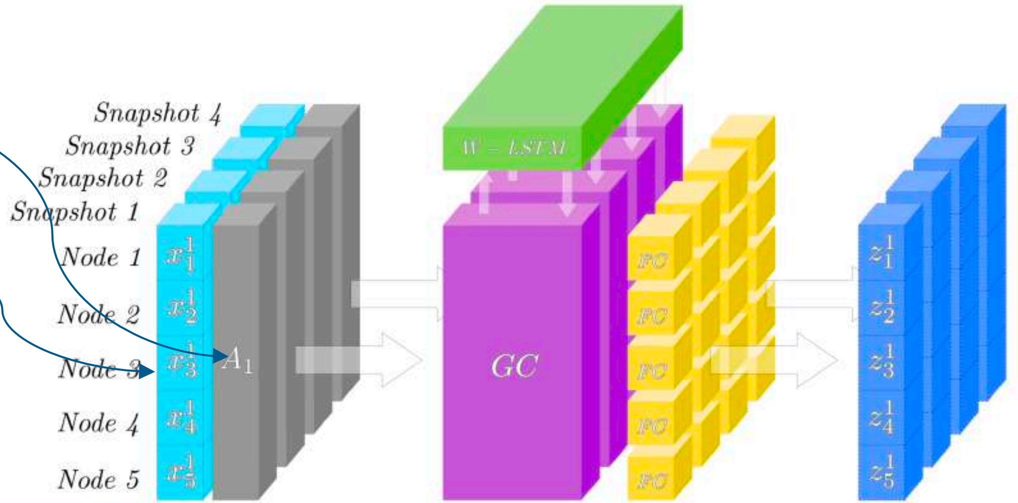


Fig. 8: Integrated DGNN structure of EvolveGCN with a EGCU-O layer [71]. The EGCU-O layer constitute the GC (graph convolution) and the W-LSTM (LSTM for GC weights). W-LSTM is used to initialize the weights of the GC.

Graph Convolution

$$f_t = \sigma(W_f *_G X_t + U_f *_G h_{t-1} + w_f \odot c_{t-1} + b_f)$$
$$i_t = \sigma(W_i *_G X_t + U_i *_G h_{t-1} + w_i \odot c_{t-1} + b_i)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c *_G X_t + U_c *_G h_{t-1} + b_c)$$
$$o_t = \sigma(W_o *_G X_t + U_o *_G H_{t-1} + w_o \odot c_t + b_o)$$
$$h_t = o \odot \tanh(c_t)$$

Reference: **Structured Sequence Modeling with Graph Convolutional Recurrent Networks**, ***ICLR***, **2017.**

Discrete Model 2:

Integrated DGNNs **(Idea is from ConvLSTM)**

Model 2. To generalize the convLSTM model (6) to graphs we replace the Euclidean 2D convolution $*$ by the graph convolution $*_{\mathcal{G}}$:

$$\begin{aligned} i &= \sigma(W_{xi} *_{\mathcal{G}} x_t + W_{hi} *_{\mathcal{G}} h_{t-1} + w_{ci} \odot c_{t-1} + b_i), \\ f &= \sigma(W_{xf} *_{\mathcal{G}} x_t + W_{hf} *_{\mathcal{G}} h_{t-1} + w_{cf} \odot c_{t-1} + b_f), \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc} *_{\mathcal{G}} x_t + W_{hc} *_{\mathcal{G}} h_{t-1} + b_c), \\ o &= \sigma(W_{xo} *_{\mathcal{G}} x_t + W_{ho} *_{\mathcal{G}} h_{t-1} + w_{co} \odot c_t + b_o), \\ h_t &= o \odot \tanh(c_t). \end{aligned} \tag{9}$$

In that setting, the support K of the graph convolutional kernels defined by the Chebyshev coefficients $W_{h\cdot} \in \mathbb{R}^{K \times d_h \times d_h}$ and $W_{x\cdot} \in \mathbb{R}^{K \times d_h \times d_x}$ determines the number of parameters, which is independent of the number of nodes n. To keep the notation simple, we write $W_{xi} *_{\mathcal{G}} x_t$ to mean a graph convolution of x_t with $d_h d_x$ filters which are functions of the graph Laplacian L parametrized by K Chebyshev coefficients, as noted in (4) and (5). In a distributed computing setting, K controls the communication overhead, i.e. the number of nodes any given node i should exchange with in order to compute its local states.

RNN-based Continuous Model: *Streaming GNN*

Update Node Embedding from source to target: Input Node Features $x \in R^{nxd}$ → Node Embedding

1. Compare: *Node Embedding*: Process; *Node Vector*: Result

   (Node Embedding = Node Vector = Node Representation)

1. Compare: *Input Node Features*: Sparse and High dimension; *Node Embedding*: Dense and Low dimension

2. Streaming GNN: maintain and update a Hidden Representation on each node (*Node Embedding*)

   Update Node Embedding

   [source node Embedding →→→ → → → → → target node Embedding]

- Update component → Update Node Embedding
- Propagation component → propagate the update to the involved node neighbors

3. Each Component keeps Three States

- Interact Unit → Interaction Node Embedding
- Update / Propagate Unit
- Merge Unit → Update Node Embedding

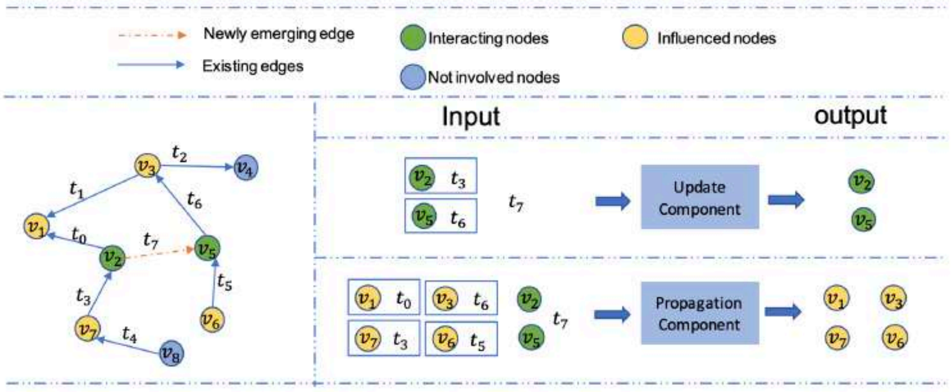


Figure 1: An overview of DGNN when a new interaction happened at time t_7 from v_2 to v_5. The two interacting nodes are v_2 and v_5. The nodes $\{v_1, v_3, v_6, v_7\}$ are assumed to be the influenced nodes.

Reference: **Streaming Graph Neural Networks**, ***SIGIR***, **2020.**

Temporal Point Process (TPP) - Continuous Model: *DyREP*

Temporal Point Process (TPP)

1. dynamics "of the network" (topological evolution)
2. dynamics "on the network" (node communication)

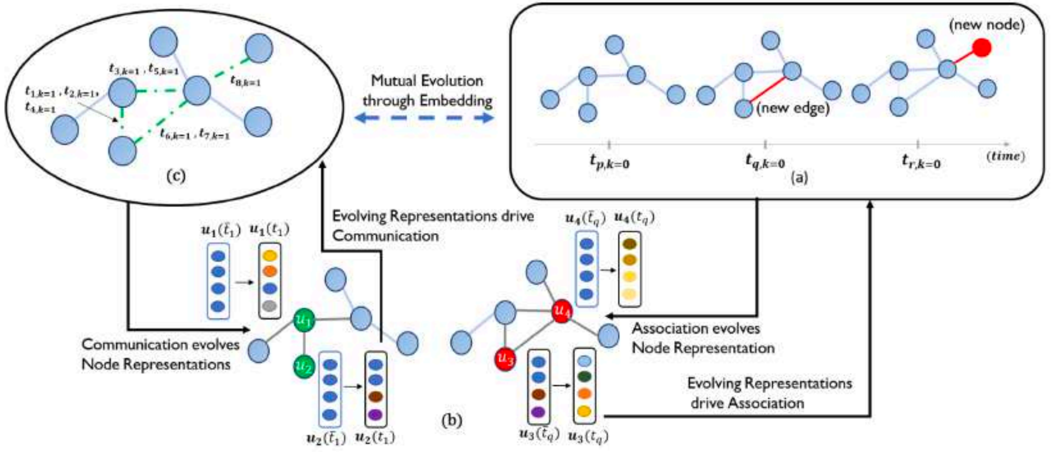


Figure 1: Evolution Through Mediation. **(a)** Association events (k=0) where the node or edge grows. **(c)** Communication Events (k=1) where nodes interact with each other. For both these processes, $t_{p,k=0} < (t_1, t_2, t_3, t_4, t_5)_{k=1} < t_{q,k=0} < (t_6, t_7)_{k=1} < t_{r,k=0}$. **(b)** Evolving Representations.

Reference: DyRep: Learning Representations over Dynamic Graphs, *ICLR*, 2019.

EvolveGCN

RNN: Regulate GCN model (i.e., network parameters)

Weight Evolution for Node Embedding:

$$\underbrace{\overbrace{W_t^{(l)}}^{\text{GCN weights}}}_{\text{hidden state}} = \text{GRU}(\underbrace{\overbrace{H_t^{(l)}}^{\text{node embeddings}}}_{\text{input}}, \underbrace{\overbrace{W_{t-1}^{(l)}}^{\text{GCN weights}}}_{\text{hidden state}})$$

+

$$\underbrace{\overbrace{W_t^{(l)}}^{\text{GCN weights}}}_{\text{output}} = \text{LSTM}(\underbrace{\overbrace{W_{t-1}^{(l)}}^{\text{GCN weights}}}_{\text{input}})$$

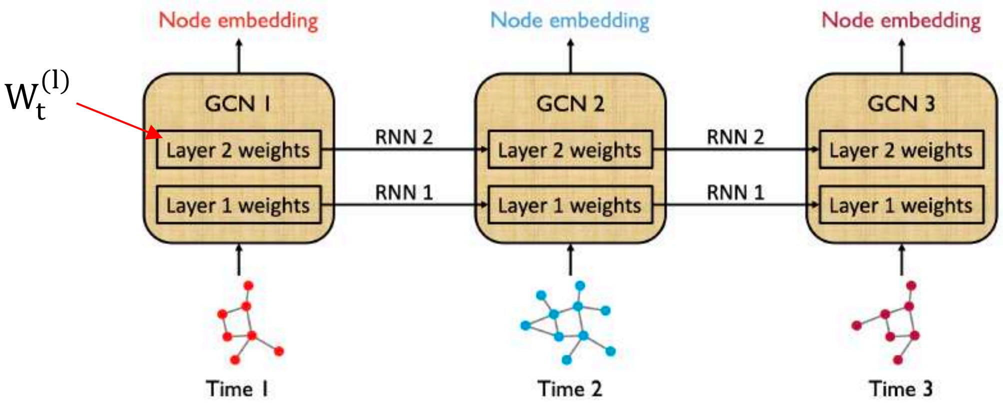


Figure 1: Schematic illustration of EvolveGCN. The RNN means a recurrent architecture in general (e.g., GRU, LSTM). We suggest two options to evolve the GCN weights, treating them with different roles in the RNN. See the EvolveGCN-H version and EvolveGCN-O version in Figure 2.

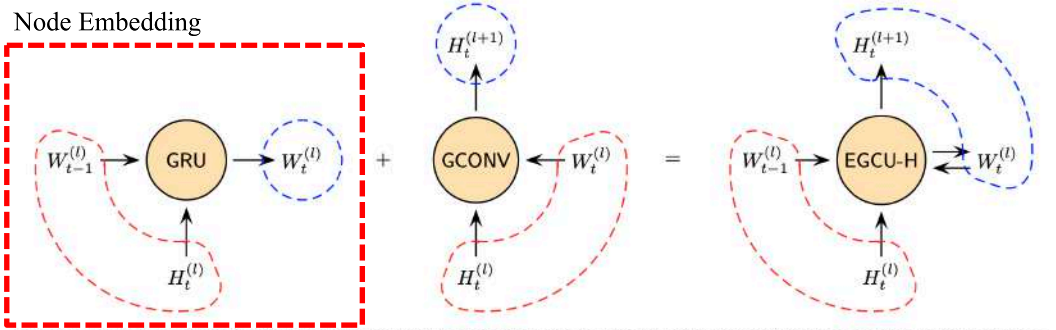
Reference: **EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs**, ***AAAI***, **2020.**

EvolveGCN

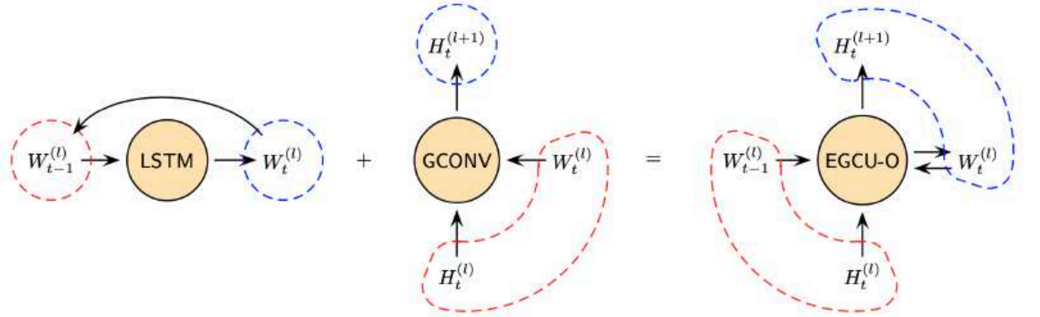
Weight Evolution for Node Embedding:

$$\underbrace{\overbrace{W_t^{(l)}}^{\text{GCN weights}}}_{\text{hidden state}} = \text{GRU}(\underbrace{\overbrace{H_t^{(l)}}^{\text{node embeddings}}}_{\text{input}}, \underbrace{\overbrace{W_{t-1}^{(l)}}^{\text{GCN weights}}}_{\text{hidden state}})$$

$$\underbrace{\overbrace{W_t^{(l)}}^{\text{GCN weights}}}_{\text{output}} = \text{LSTM}(\underbrace{\overbrace{W_{t-1}^{(l)}}^{\text{GCN weights}}}_{\text{input}})$$



(a) EvolveGCN-H, where the GCN parameters are hidden states of a recurrent architecture that takes node embeddings as input.



(b) EvolveGCN-O, where the GCN parameters are input/outputs of a recurrent architecture.

Node features are informative:

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCU-H}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \text{GRU}(H_t^{(l)}, W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

Change of the structure:

1: **function** $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCU-O}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$
2: $\quad W_t^{(l)} = \text{LSTM}(W_{t-1}^{(l)})$
3: $\quad H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$
4: **end function**

EvoNet – Predict the topology of future graphs

Sequence-to-sequence Model (Autoencoder)

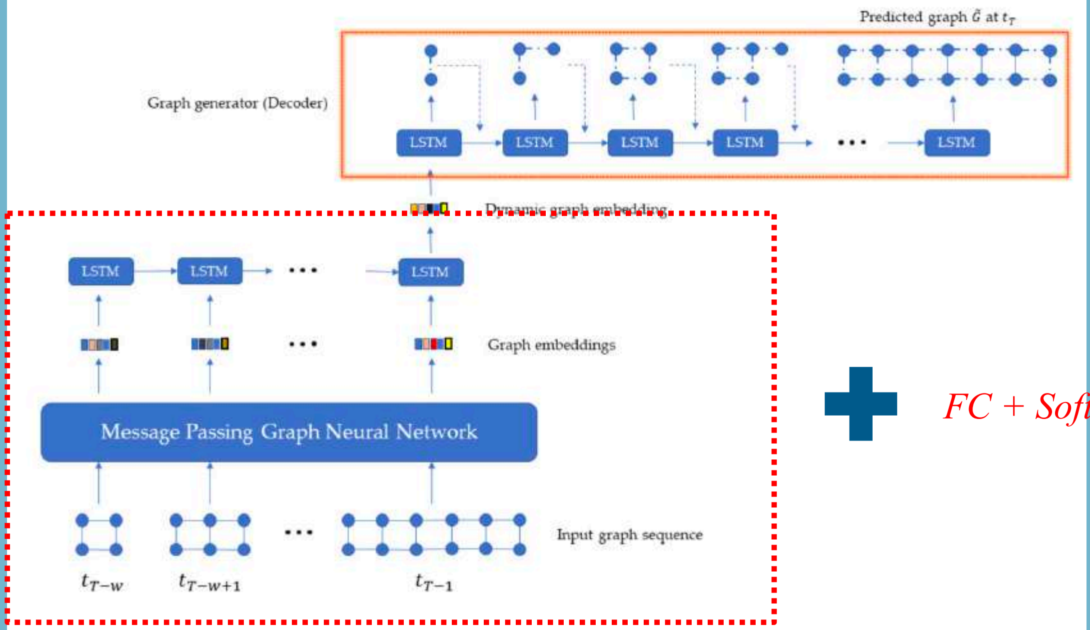


Figure 1: Illustration of the proposed architecture

FC + Softmax

Reference: **EvoNet: A Neural Network for Predicting the Evolution of Dynamic Graphs**, 2020.

Our Perspective

1. Current Loop: [GCN + RNN] → Can we jump the loop?
2. Can we employ Dynamic GCN method to our filed, e.g., EEG Signals Classification
3. Virtualize the dynamic process of the Graphs.

Thanks and any Questions?