Graph Convolutional Networks summary

N. Kilis, Prof. Ioannis Pitas,
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr

Version 5.7.4
Date: 10/7/2021

Artificial Intelligence & Information Analysis Lab

Graph Convolutional Networks

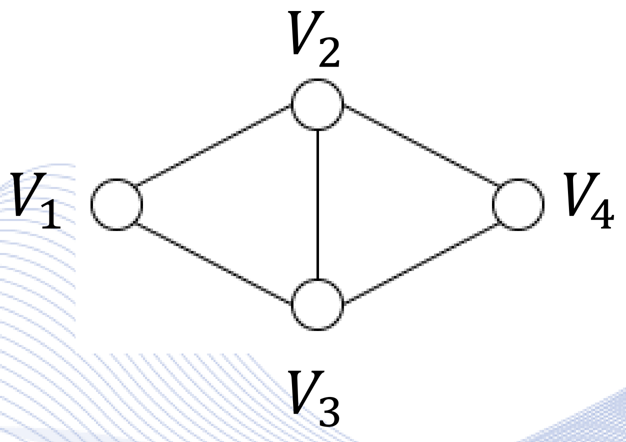
- Graph Convolutions
- Empirical Risk Minimization with Graph Signals
- Learning with Graph Convolutional Filters
- Learning with Graph Perceptrons
- GCN Types
- GCN general architecture

- Two ways to define Convolution:
1. Spectral Graph Convolution
    - Simple Spectral GCN, Spline GCN, LapGCN, ChebNet, CayleyNet
2. Spatial Graph Convolution
    - Simple Spatial GCN, GraphSage, GIN, MoNet, GAT, GatedGCN
- GCN from scratch with numpy
- Spatio-Temporal GCN

Graph Convolutions

Graph definition: $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$

- $\mathcal{V}$: set of nodes,
- $\mathcal{E}$: set of edges,
- $\mathcal{W}$: set of edge weights.
- N: number of nodes
- E: number of edges



Graph types:

- Directed / Undirected or Symmetric,
- Weighted / Unweighted.

Graph Convolutions

Graph-Shift Operator (***GSO***):

$$\boldsymbol{S} \in \mathbb{R}^{N \times N}, \qquad S_{ij} \neq 0 \text{ if } i = j \text{ and/or } (i, j) \in \mathcal{E}.$$

- It enables matrix representations of graphs.
- It captures the local graph structure.
- If the graph is symmetric, $\boldsymbol{S}$ is also symmetric.

Graph Convolutions

- Various algebraic choices of $\boldsymbol{S}$:
  - Adjacency matrix: $\mathbf{S} = \mathbf{A}$,
  - Graph Laplacian matrix (Directed Graphs):

$$\mathbf{S} = \mathbf{L}_{in} = \mathbf{D}_{in} - \mathbf{A}, \qquad \mathbf{S} = \mathbf{L}_{out} = \mathbf{D}_{out} - \mathbf{A}$$

$$[\mathbf{D}_{in}]_{ii} = \sum_{j=1}^{N} \mathbf{A}_{ji}\,, \qquad [\mathbf{D}_{out}]_{ii} = \sum_{j=1}^{N} \mathbf{A}_{ij}$$

  - Symmetric Graph Laplacian (Undirected Graphs):

$$\mathbf{S} = \mathbf{L} = \mathbf{D} - \mathbf{A}, \qquad \mathbf{D} = \mathbf{D}_{in} = \mathbf{D}_{out}$$

- The choice matters in practice, however ***the analysis results hold for any selection***.

Graph Convolutions

- ***Vertex signal***:

$$x_i : \mathcal{V} \to \mathbb{R}.$$

- ***Vectorial vertex signal***:

$$\mathbf{x}_i : \mathcal{V} \to \mathbb{R}^n.$$

- ***Graph signal***:

For notation simplification, it can be described by a vector:

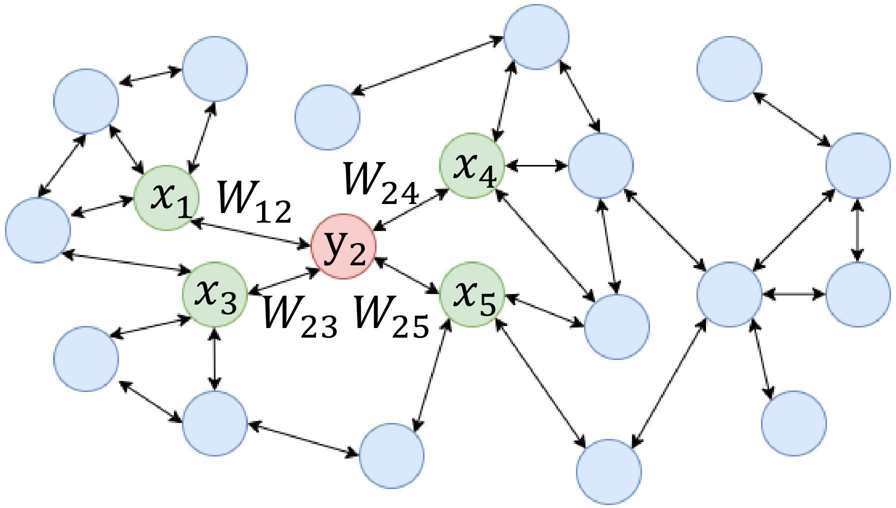
$$\mathbf{x} = [x_1, x_2, \dots, x_N]^T \in \mathbb{R}^N,$$

residing on the vertex set $\mathcal{V}$ of graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$.

Graph Convolutions

- ***Diffusion*** of a Graph Signal: $\mathbf{y} = \boldsymbol{S}\mathbf{x}$.
  - Component i of $\mathbf{y}$ is affected by the set of nodes $j \in \mathcal{N}_i$:

$$y_i = \sum_{j \in \mathcal{N}_i} W_{ij} x_j$$



  - Stronger weights contribute more the diffusion.
  - Local operation where components are mixed with components of neighboring nodes.

Graph Signal Diffusion

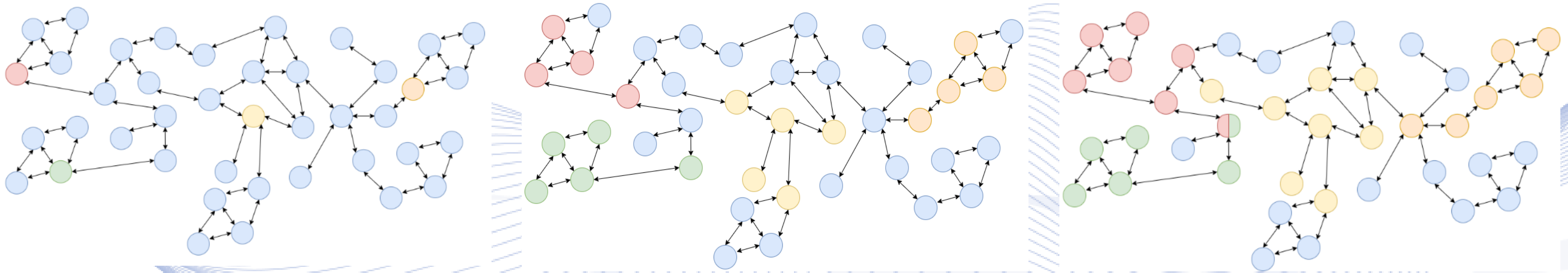
- ***Diffusion sequence*** → Recursive application of Diffusion:

$$\mathbf{x}^{(k+1)} = \mathbf{S}\mathbf{x}^{(k)},$$

$$\mathbf{x}^{(0)} = \mathbf{x}.$$

- We can also write the diffusion sequence as the power sequence:

$$\mathbf{x}^{(k)} = \mathbf{S}^{(k)}\mathbf{x}$$



$\mathbf{x}^{(0)} = \mathbf{x} = \mathbf{S}^{(0)}\mathbf{x}$ $\quad$ $\mathbf{x}^{(1)} = \mathbf{S}\mathbf{x}^{(0)} = \mathbf{S}^{(1)}\mathbf{x}$ $\quad$ $\mathbf{x}^{(2)} = \mathbf{S}\mathbf{x}^{(1)} = \mathbf{S}^{(2)}\ \mathbf{x}$

- Always implement the recursive version. Power version only for analysis.

Artificial Intelligence & Information Analysis Lab

Graph Convolutions

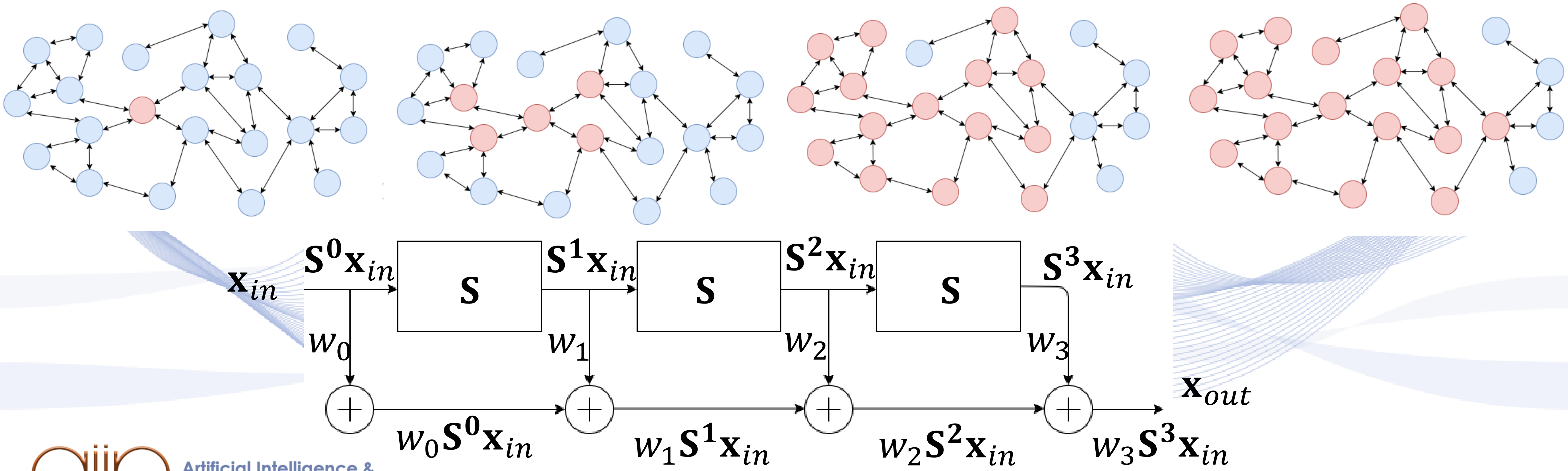
- Implementation of a convolutional filter with coefficients w_k and order K.

- $\mathbf{x}_{in}$, $\mathbf{x}_{out} \in \mathbb{R}^N$: input, output signals of a convolution filter (each signal value residing on a graph node).

- Linear combination of diffuse versions of the input signal $\mathbf{x}_{in}$ scaled by w_k.

Graph Convolutions

- Graph Convolutional filters perform linear processing of graph signals.



Empirical Risk Minimization with Graph Signals

Machine Learning (***ML***) on graphs is equivalent to ***Empirical Risk Minimization*** (***ERM***) on graph signals.

- In ERM, we are given:
  - A training set $\mathcal{D}$ with observation graph signal pairs $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}, i = 1, \ldots, |\mathcal{D}|$ of equal length: $\mathbf{x}_i, \mathbf{y}_i \in \mathbb{R}^N$, residing on the nodes of graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W})$.
  - A loss function $J(\mathbf{y}, \hat{\mathbf{y}})$ to evaluate the similarity between $\mathbf{y}$ and $\hat{\mathbf{y}}$,
  - A function class $\boldsymbol{f} \in \mathcal{C}$, $\hat{\mathbf{y}} = \boldsymbol{f}(\mathbf{x}; \boldsymbol{\theta})$, the degree of freedom available to the designer.

Artificial Intelligence & Information Analysis Lab

Empirical Risk Minimization with Graph Signals

- ***Learning***:
  - find the optimal parameter vector $\boldsymbol{\theta}$ of a function $\boldsymbol{f}^*(\mathbf{x};\boldsymbol{\theta}) \in \mathcal{C}$ that minimizes $J(\mathbf{y},\hat{\mathbf{y}})$ averaged over $\mathcal{D}$:

$$\boldsymbol{f}^* = \underset{\boldsymbol{f}\in\mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} J(\mathbf{y},\boldsymbol{f}(\mathbf{x};\boldsymbol{\theta})).$$

Learning with Graph Convolutional Filters

- ***Graph Filter*** of order K supported by $\boldsymbol{S}$:

$$\mathbf{x} \rightarrow \boxed{\boldsymbol{z}(\mathbf{x};\boldsymbol{\theta}) = \sum_{k=0}^{K-1} w_k \mathbf{S}^k \mathbf{x}} \rightarrow \boldsymbol{z}(\mathbf{x};\mathbf{S},\mathbf{w})$$

- In this case, the learnable parameter vector $\boldsymbol{\theta}$ is the graph convolution kernel coefficient vector $\mathbf{w} = [w_0, \ldots, w_{K-1}]$:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\epsilon\mathcal{D}} J\big(\mathbf{y}, \boldsymbol{f}(\mathbf{x};\mathbf{S},\mathbf{w})\big).$$

Learning with Graph Perceptrons

- A GCN composed of several Graph Perceptrons ($\mathbf{W} = [\mathbf{w}_1^T | \dots | \mathbf{w}_L^T]^T$):

$\mathbf{x}$ →

$$\boldsymbol{z}_1(\mathbf{x}) = \sum_{k=0}^{K-1} w_{1k} \mathbf{S}^k \mathbf{x}_0$$

$\boldsymbol{z}_1(\mathbf{x}_0; \mathbf{S}, \mathbf{w}_1)$ →

$$\mathbf{x}_1 = f(\boldsymbol{z}_1)$$

• • •

$$\boldsymbol{z}_L(\mathbf{x}) = \sum_{k=0}^{K-1} w_{Lk} \mathbf{S}^k \mathbf{x}_{L-1}$$

$\boldsymbol{z}_L(\mathbf{x}; \mathbf{S}, \mathbf{w}_L)$ →

$$\boldsymbol{x}_L = f(\boldsymbol{z}_L)$$

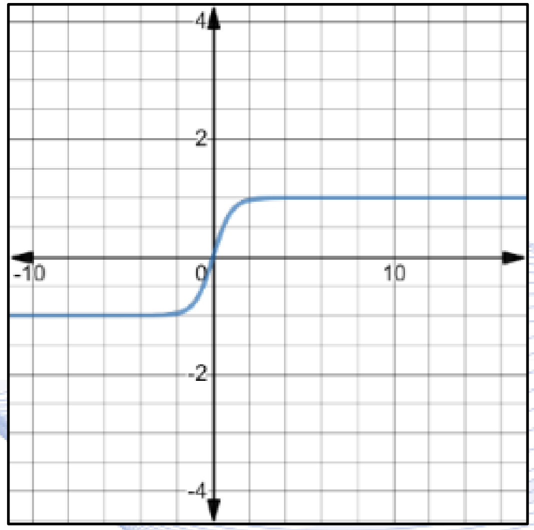
→ $\mathbf{y} = \boldsymbol{f}(\mathbf{x}; \mathbf{S}, \mathbf{W})$

$$\mathbf{C}^* = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y}) \epsilon \mathcal{D}} J\big(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \mathbf{S}, \mathbf{W})\big).$$

Learning with Graph Perceptrons

Tanh:

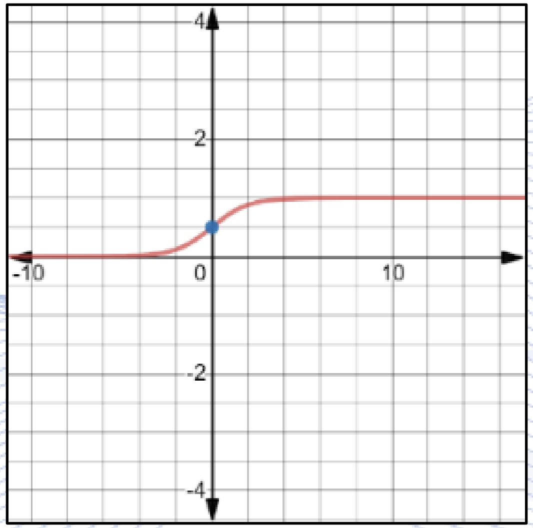
$$f(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}}$$



Example of activation functions

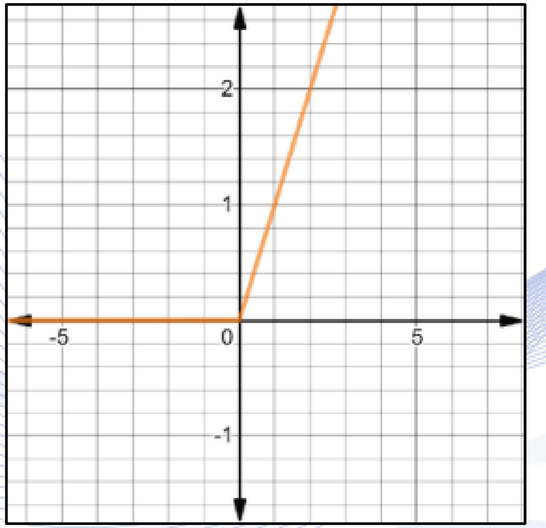
Sigmoid:

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



ReLU:

$$f(x) = max(0, x)$$



Activation functions.

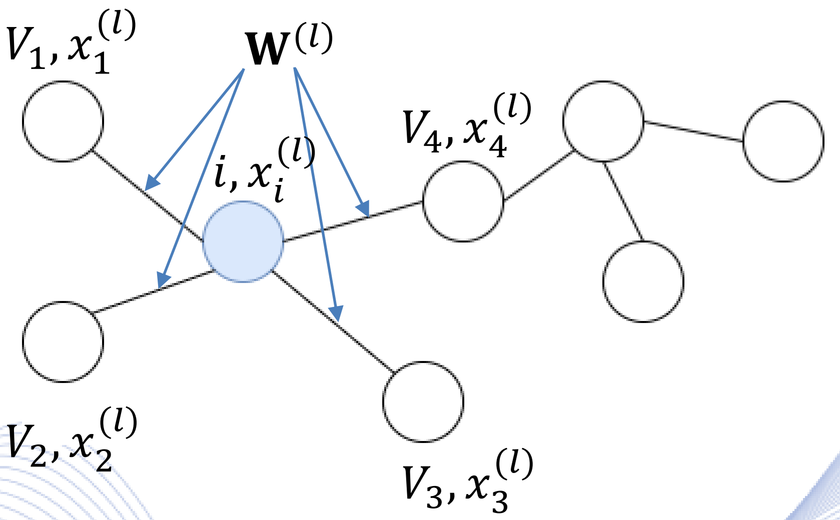
Plots are generated by Desmos : https://www.desmos.com/calculator

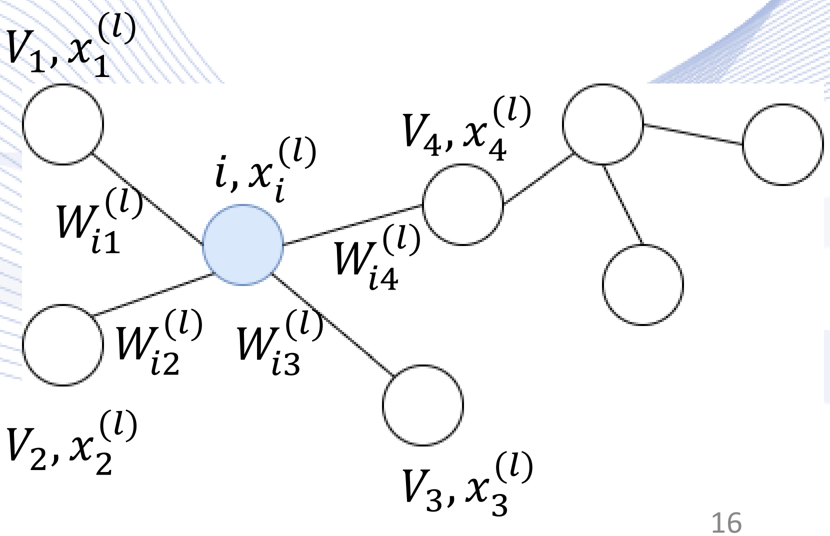
GCN Types

- An isotropic filter treats all neighbors equally, with no particular bias towards certain neighbors.

- ***Isotropic GCNs***:
  - Use same matrix $\mathbf{W}^{(l)}$, for neighborhood $\mathcal{N}_i$.

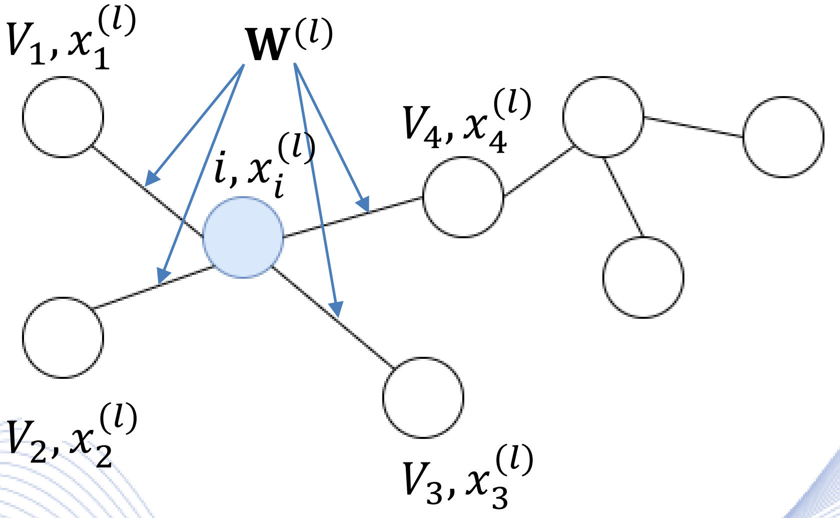
- ***Anisotropic GCNs***:
  - Different neighbors of node i, (V_1, V_2, V_3, V_4) are treated differently $(W_{i1}^{(l)}, W_{i2}^{(l)}, W_{i3}^{(l)}, W_{i4}^{(l)})$.



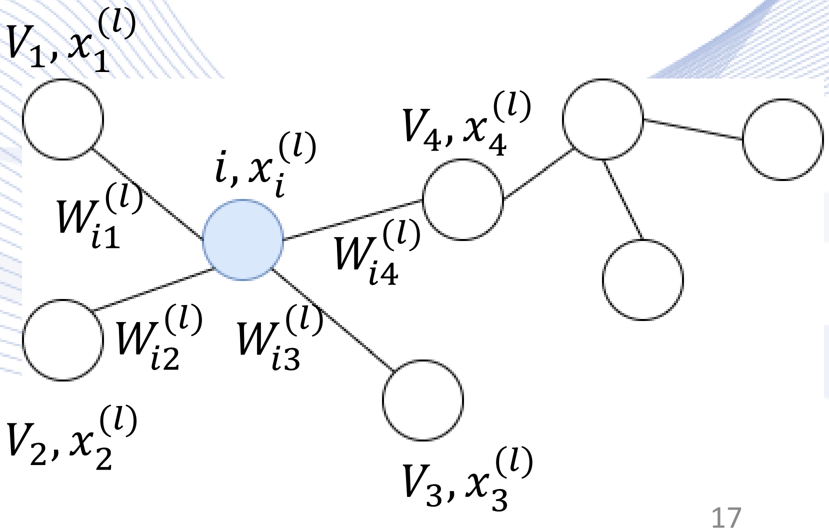


GCN Types

- ***Isotropic GCNs***:
  - ChebNet
  - CayleyNet
  - Simple Spatial GCN
  - GraphSage
  - GIN

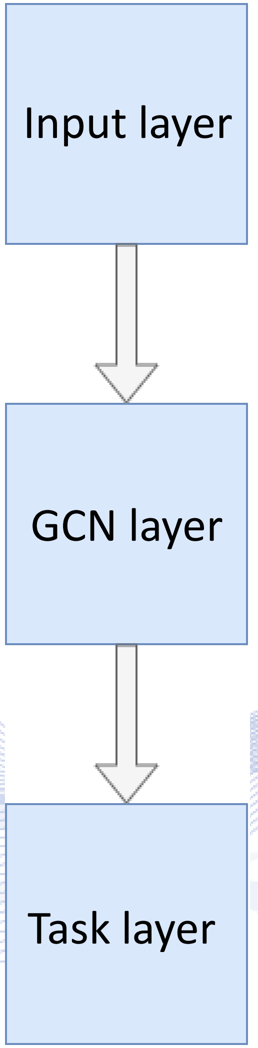


- ***Anisotropic GCNs***:
  - MoNet
  - GAT
  - GatedGCN



GCN general architecture

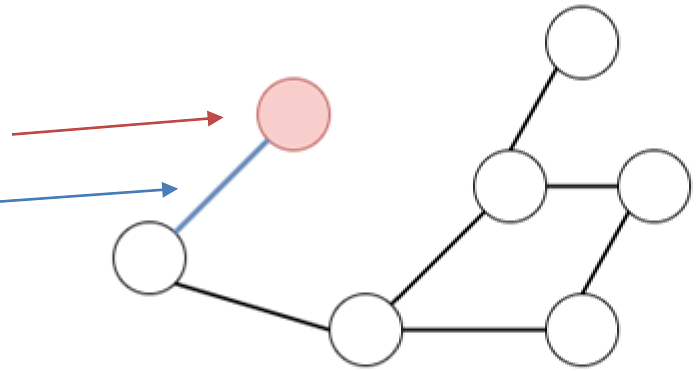
1. ***Input layer***:
   - Linear embedding of input node features.
   - Linear embedding of input edge features.
2. ***GCN layer***:
   - Application of a GCN architecture, L times.
3. ***Task layer***:
   - Graph prediction layer.
   - Node prediction layer.
   - Edge prediction layer.



Artificial Intelligence &
Information Analysis Lab

GCN general architecture

- ***Input layer***:
  - Input node feature vectors $\mathbf{x}_{i,in}$.
  - Input edge features $\mathbf{e}_{ij,in}$.



- Embedding layer of input node/edge features:

$$\mathbf{x}_i^{(l=0)} = \mathbf{x}_{i,in} \in \mathbb{R}^n, \qquad i = 1, \dots, N.$$

$$\mathbf{e}_{ij}^{(l=0)} = \mathbf{e}_{ij,in} \in \mathbb{R}^{n'},\ i = 1, \dots, N \text{ and } j = 1, \dots, E.$$

  - For notation simplicity, we assume $n' = n$.

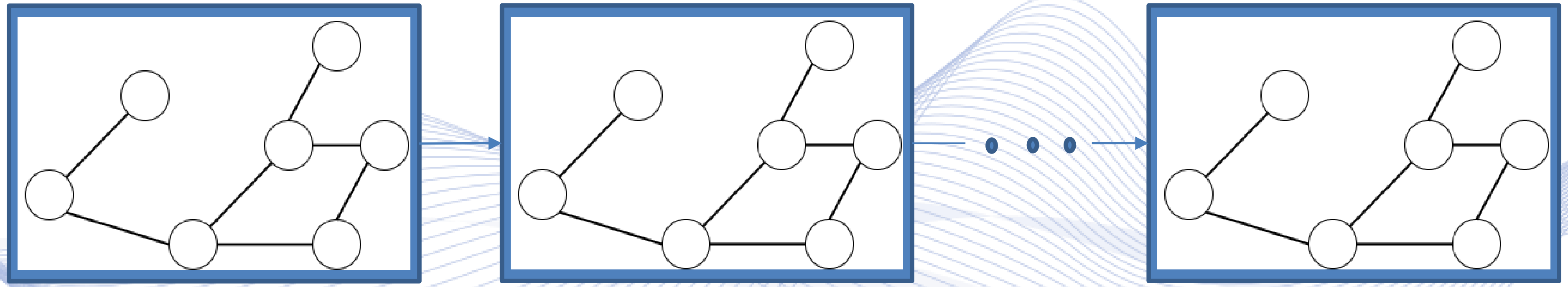
- Output matrix with n features for N nodes: $\mathbf{X}^{(l=0)} \in \mathbb{R}^{N \times n}$.
- Output matrix with n features for E edges: $\mathbf{E}^{(l=0)} \in \mathbb{R}^{E \times n}$.

GCN general architecture

- ***GCN layer***:
  - Input node and edge features embedded into a n-dimensional space:

$$\mathbf{X}^{(l=0)} \in \mathbb{R}^{N\times n}.$$
$$\mathbf{E}^{(l=0)} \in \mathbb{R}^{E\times n}.$$

  - L GCN layers ($l = 1, \dots, L$). Their structure is defined subsequently.



  - L-th layer GCN output:

$$\mathbf{X}^{(l=L)} \in \mathbb{R}^{N\times n}.$$
$$\mathbf{E}^{(l=L)} \in \mathbb{R}^{E\times n}.$$

Two ways to define Convolution

Spatial / Vertex domain:

- A graph is considered as a set of nodes connected by edges.
- Information on one node is aggregated from through its neighbors.
- ***Spatial Graph Convolution*.**

Spectral domain:

- A graph is a discrete manifold [GEOM].
- Discretize manifold and do Spectral Convolution using the Laplacian matrix.
- ***Spectral Graph Convolution*.**

Simple Spectral GCN

- Proposed by [BRU2013].
- Spectral Graph Convolutional layer:

$$\mathbf{X}^{(l+1)} = f\left(\widehat{\mathbf{H}}(\mathbf{L})^{(l)}\mathbf{X}^{(l)}\right) = f\left(\mathbf{U}\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{(l)}\right),$$

$$\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)} = \mathrm{diag}[\hat{\mathbf{h}}] = \begin{bmatrix} \hat{h}(\lambda_1) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \hat{h}(\lambda_N) \end{bmatrix}.$$

- Goal: Learn $\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)}$ via Backpropagation.

Artificial Intelligence & Information Analysis Lab

SplineGCN

- Proposed by [HEN2015].
- Spectral Graph Convolutional layer:

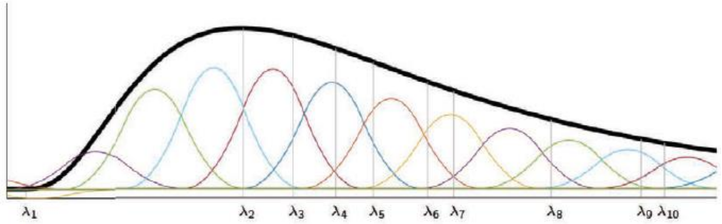
$$\mathbf{X}^{(l+1)} = f\left(\widehat{\mathbf{H}}(\mathbf{L})^{(l)}\mathbf{X}^{(l)}\right) = f\left(\mathbf{U}\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)}\mathbf{U}^{\mathrm{T}}\mathbf{X}^{(l)}\right),$$

$$\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)} = \mathrm{diag}[\mathbf{B}\hat{\mathbf{h}}^{(l)}],$$

$$\widehat{\mathbf{H}}(\mathbf{\Lambda})^{(l)} \in \mathbb{R}^{N\times N} \quad \mathbf{B} \in \mathbb{R}^{N\times S} \quad \hat{\mathbf{h}}^{(l)} \in \mathbb{R}^{S}.$$

SplineGCN

- If smooth in Spectral domain:



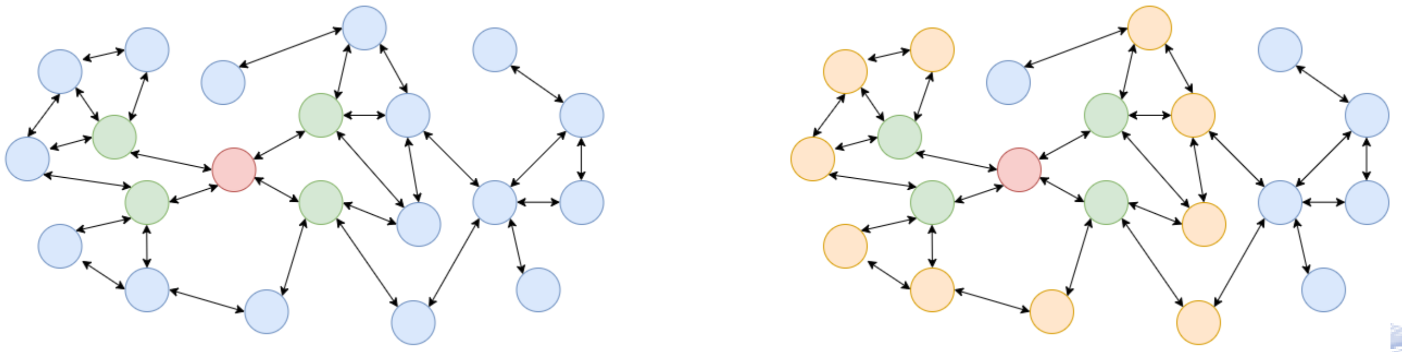
- Then localized in Spatial domain:

- Related publication [SHU2016].

LapGCN

- Obtain exactly localized filters with k-hop support:

$$\hat{\mathbf{H}}(\mathbf{L}) \triangleq \sum_{k=0}^{K-1} w_k \mathbf{L}^k$$



1-hop neighborhood ($\mathbf{L}^1$)   2-hop neighborhood ($\mathbf{L}^2$)

Artificial Intelligence & Information Analysis Lab

ChebNets

- A filter can be parametrized as the truncated expansion:

$$\hat{\mathbf{H}}(\tilde{\mathbf{L}}) = \sum_{k=0}^{K-1} w_k T_k(\tilde{\mathbf{L}}).$$

- Where w_k are the Chebyshev coefficients and

- $T_k(\tilde{\mathbf{L}}) \in \mathbb{R}^{N \times N}$ is the Chebyshev polynomial evaluated at the scaled Laplacian matrix:

$$\tilde{\mathbf{L}} \triangleq 2\lambda_{max}{}^{-1}\mathbf{L} - \mathbf{I}.$$

CayleyNets

- Proposed by [LEV2018].

- Choose an orthonormal basis like the Cayley rationals:

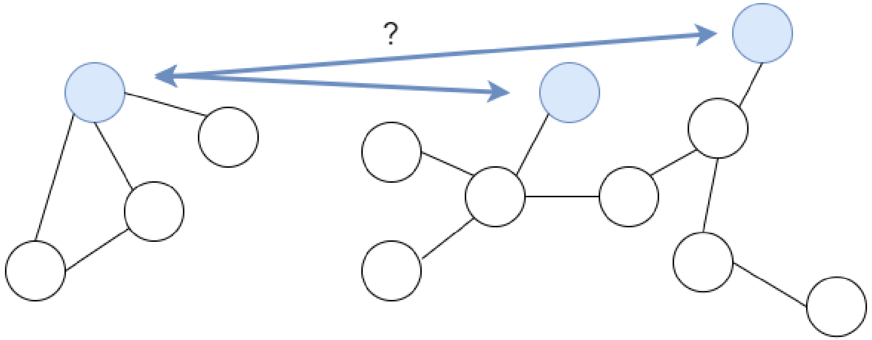
- Benefits:
  - Same properties like ChebNets.
  - Localized in frequency (with spectral zoom).
  - Provide a richer class of filters for the same order K.

- Limitations:
  - Isotropic model.

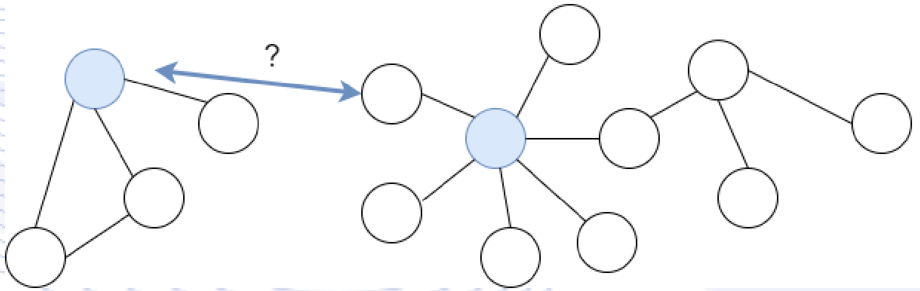
Template matching in graphs

Limitations:

- ***Lack of node ordering***:
  - Can not match the template features with the data features.
  - The nodes do not have a well-defined position, but only an arbitrary index.



- ***Heterogeneous neighborhoods***:
  - Can not deal with nodes that have a different number of neighbors.



Spatial Graph Convolution

Absence of node ordering solution:

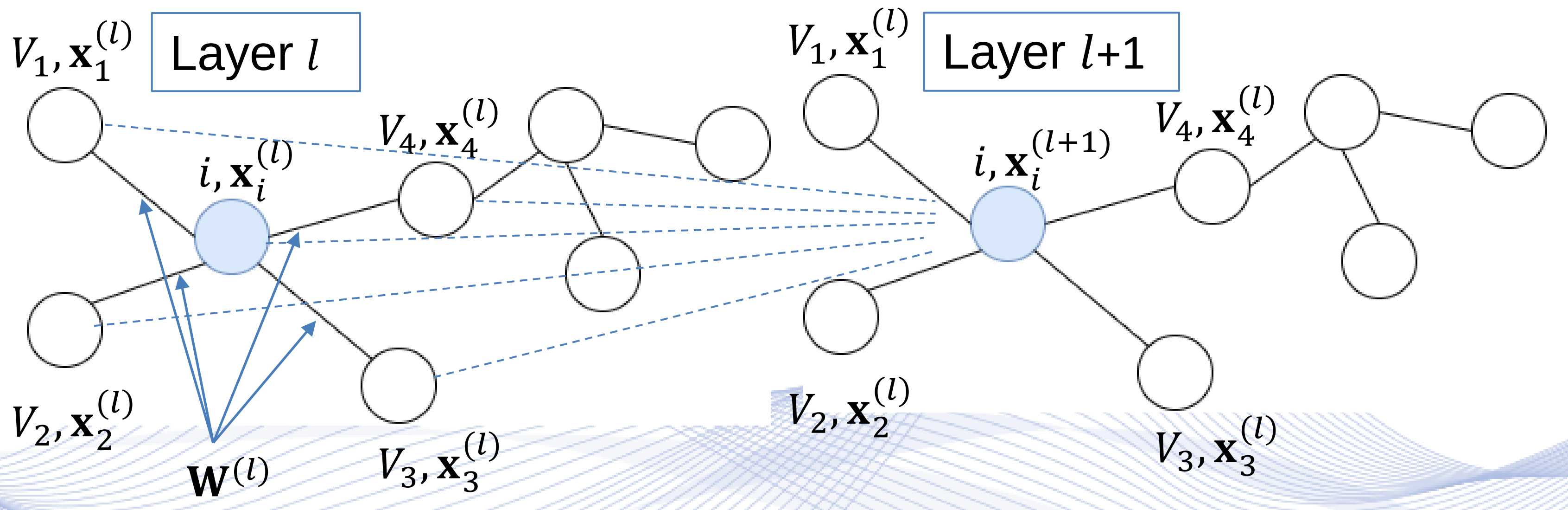
- Use the ***same*** template matrix for all neighbors.

Heterogeneous neighborhoods solution:

- Compute the ***average*** value of all neighbors.

Simple Spatial GCN

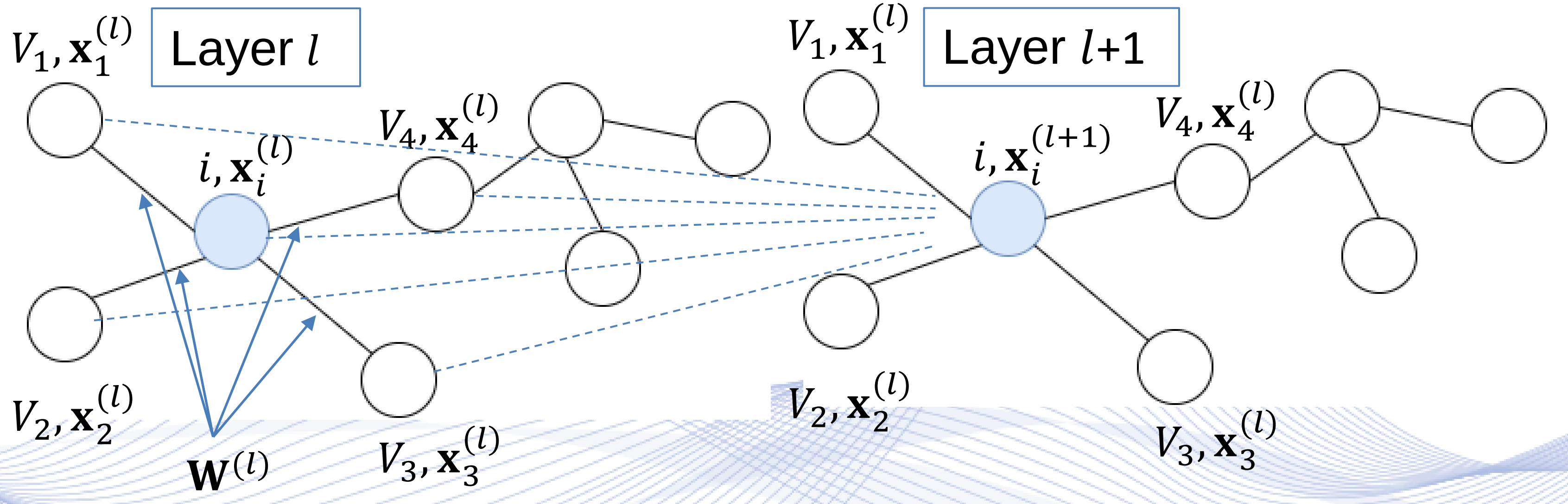
Matrix representation: $\mathbf{X}^{(l+1)} = f\left(\mathbf{D}^{-1}\mathbf{A}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right)$



$$\mathbf{x}_i^{(l+1)} = f_{GCN}\left(\mathbf{x}_i^{(l)}, \{\mathbf{x}_j^{(l)} : j \rightarrow i\}\right)$$

Simple Spatial GCN

Matrix representation: $\mathbf{X}^{(l+1)} = f\left(\mathbf{D}^{-1}\mathbf{A}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right)$



$$\mathbf{x}_i^{(l+1)} = f_{GCN}(\mathbf{x}_i^{(l)}, \{\mathbf{x}_j^{(l)} : j \rightarrow i\})$$

Different families of NNs

GraphSage

- Proposed by [HAM2017].
- A modification of Simple Spatial GCN:

$$\mathbf{x}_i^{(l+1)} = f\left(\frac{1}{d_i}\sum_{j\in\mathcal{N}_i} A_{ij}\mathbf{W}^{(l)}\,\mathbf{x}_{ij}^{(l)}\right)$$

For connected nodes: A_{ij} values are equal to 1.

$$\mathbf{x}_i^{(l+1)} = f\left(\frac{1}{d_i}\sum_{j\in\mathcal{N}_i} \mathbf{W}^{(l)}\,\mathbf{x}_{ij}^{(l)}\right)$$

Graph Isomorphism Networks

- Proposed by [XU2018].
- The architecture of GINs can discriminate Graphs that are not isomorphic:

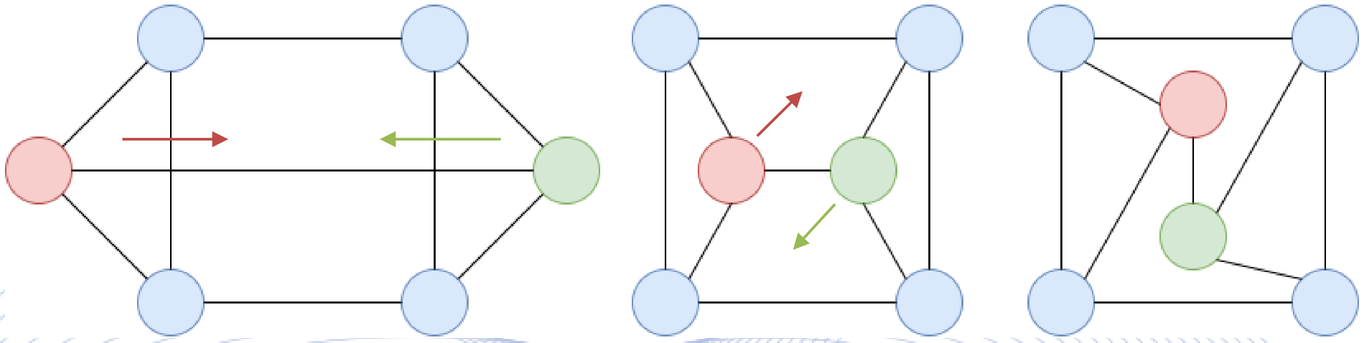
$$\mathbf{x}_i^{(l+1)} = f\left(\mathbf{W}_2^{(l)} f\left(BN\left(\mathbf{W}_1^{(l)} \tilde{\mathbf{x}}_i{}^{(l)}\right)\right)\right)$$

$$\tilde{\mathbf{x}}_i{}^{(l)} = (1+\varepsilon)\mathbf{x}_i^{(l)} + \sum_{j \in \mathcal{N}_i} \mathbf{x}_i^{(l)}$$

- $\mathbf{W}_1^{(l)} \in \mathbb{R}^{n \times n}$, $\mathbf{W}_2^{(l)} \in \mathbb{R}^{n \times n}$.
- f: $ReLU$ activation function.
- BN: Batch Normalization.
- ε : can be either a learnable parameter or a fixed scalar.

Graph Isomorphism Networks

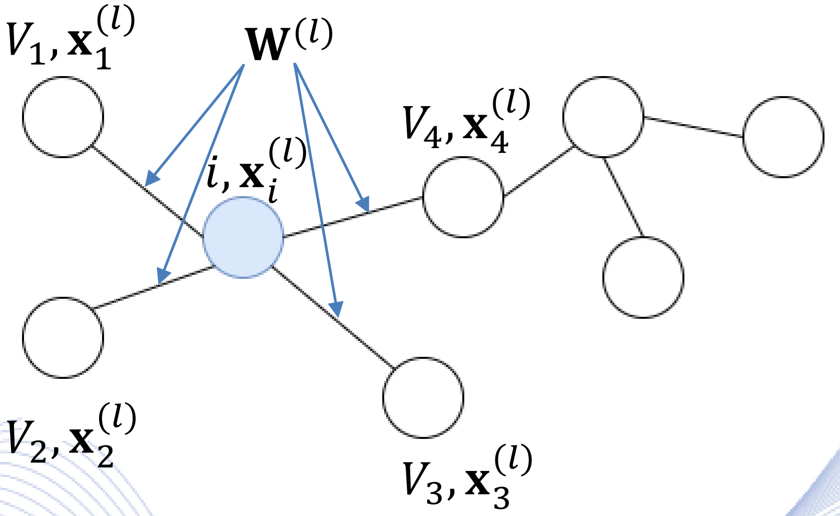
- Graph isomorphism example:



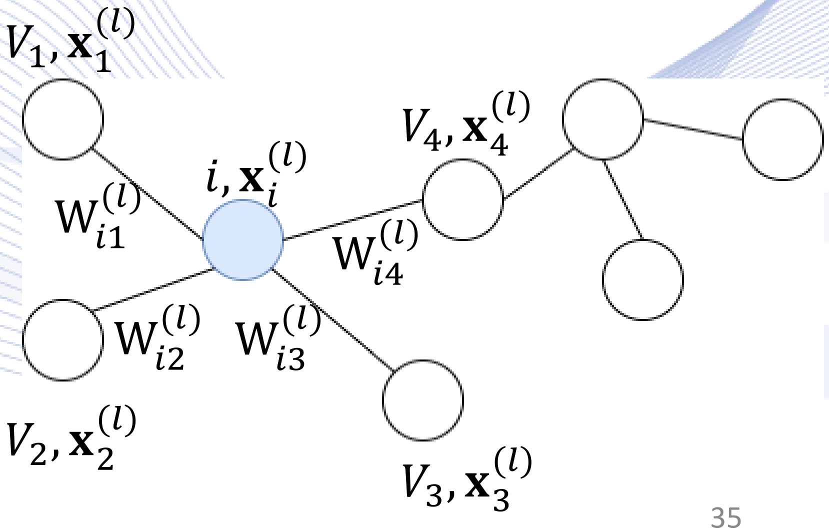
- Limitations:
  - Isotropic model.

GNN Types

- How can we can get back anisotropy?
  - Natural edge features (if available).
  - Anisotropic mechanism independent of node parametrization.



- Proposed methods:
  - Edge degrees: MoNets
  - Edge gates: GatedGCNs
  - Attention mechanism: GATs

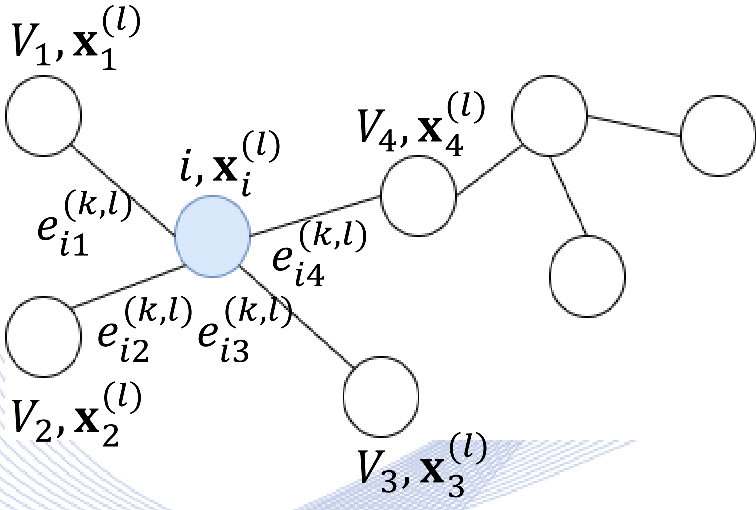


Artificial Intelligence & Information Analysis Lab

MoNet

- Proposed by [MON2017].
- MoNets exploit the Graph degree to learn a Bayesian Gaussian Mixture Model (GMM):

$$\mathbf{x}_i^{(l+1)} = f(\sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} e_{ij}^{(k,l)} \mathbf{W}_1^{(k,l)} \mathbf{x}_j^{(l)})$$

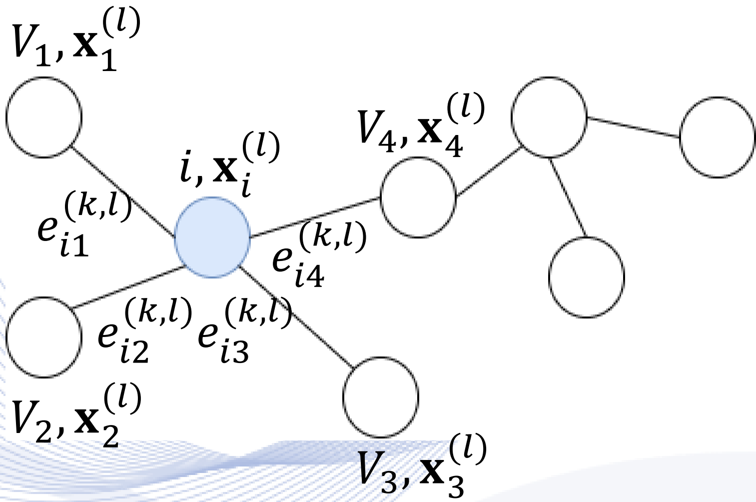


- Where:
  - $f: ReLU$ activation function,
  - $\mathbf{W}_1^{(k,l)} \in \mathbb{R}^{n \times n}$.

Graph Attention Networks

- Proposed by [VEL2017].
- GATs exploit the attention mechanism to increase the impact of some neighbors in the Graph neighborhoods with a multi-headed architecture:

$$\mathbf{x}_i^{(l+1)} = Concat_{k=1}^{K}(f\left(\sum_{j \in \mathcal{N}_i} e_{ij}^{(k,l)} \mathbf{W}_1^{(k,l)} \mathbf{x}_j^{(l)}\right))$$

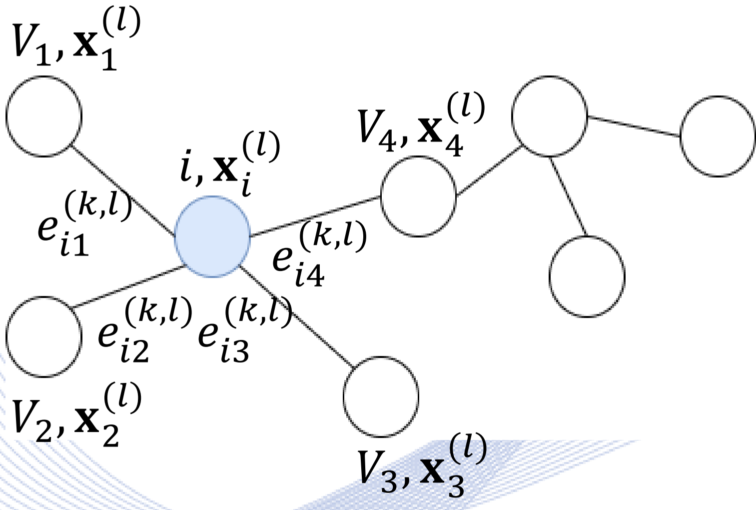


- Where:
  - f: ELU activation function.
  - $Concat_{k=1}^{K}$: K independent attention mechanisms, whose features are concatenated.

Gated Graph ConvNets

- Proposed by [BRE2017].
- GatedGCNs employ a gating mechanism on the edges (soft attention):

$$\mathbf{x}_i^{(l+1)} = \mathbf{x}_i^{(l)} + f\left(BN\left(\mathbf{W}_1^{(l)}\mathbf{x}_i^{(l)} + \sum_{j\in\mathcal{N}_i} \mathbf{e}_{ij}^{(l)} \otimes \mathbf{W}_2^{(l)}\mathbf{x}_j^{(l)}\right)\right)$$



- Where:
  - f: $ReLU$ activation function.
  - BN: Batch Normalization.

GCN from scratch with numpy

1. *Message passing*:

- Matrix multiplication of the Adjacency matrix and the feature vector:
  - Mask out all the values, except the ones that the examined node has a connection with.

  - Final result:
    - new feature vector (same shape as the original),
    - each value now represents the sum of the connected neighborhoods of each node.

GCN from scratch with numpy

1. ***Message passing***:
   - Matrix multiplication of the Adjacency matrix and the feature vector:
     - Message: Feature vectors,
     - Aggregation function : ***Summation***.

   - Alternative aggregation function (***Average***):

$$\mathbf{D}^{-1}\mathbf{A} = \mathbf{A}_{avg}$$

GCN from scratch with numpy

- Self connections – modified Adjacency matrix:

$$\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$$

- Normalized Adjacency matrix (scale with each node's degree):

$$\widehat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2} \qquad \widehat{\mathbf{A}}_{i,j} = \frac{\tilde{A}_{i,j}}{\sqrt{\tilde{d}_i \tilde{d}_j}}$$

- Diffusion mechanism visualized with an animation.

GCN from scratch with numpy

2. *GCN from scratch*:

- Message passing (multiplication with the Adjacency matrix of the Graph):

GCNLayer forward: $\text{self.\_X} = (\text{A @ X}).\text{T}$

- Computation of a linear projection with **W** followed by an activation function:

$$\mathbf{X}^{(l+1)} = f\left(\mathbf{A}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right)$$

- Backpropagation : ***independent of the Graph*** (same as in other NNs).

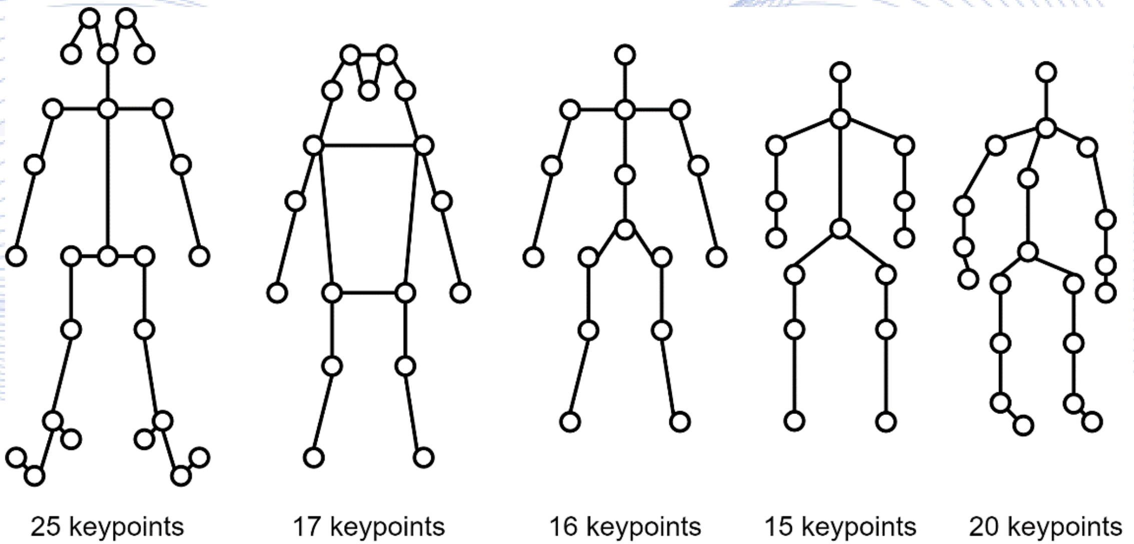
Spatio-Temporal GCN

- Proposed by [YAN2018].
- Applied in skeleton-based ***Human Action Recognition*** from video frames:
  - Important topic in Computer Vision,
  - Identification of ***actions*** that take place in a video:
    - Primitive action, elementary body part motion (e.g., Hand raising).
    - Action, incorporates multiple temporally organized primitive actions (e.g., Running).
    - Activity, high-level motion that includes several actions (e.g., Playing tennis).
  - Other applications: Robotics, Medicine, Supervised physical training, Human-computer interaction.

Spatio-Temporal GCN

- Human skeleton:
  - Keypoints: Nodes in the Graph,
  - Connections: Edges in the Graph.

- Representation with graphs:
  - ***Invariant to view point and appearance***.



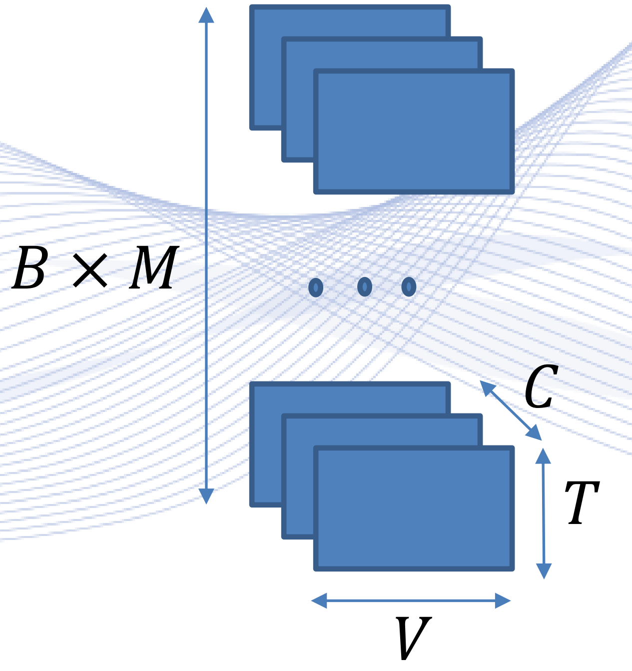
Spatio-Temporal GCN

- Human skeleton as ***ST-GCN input***:
  1. Data: Skeleton Spatial Coordinates,
  2. Graphical connections: Adjacency matrix.

- Input data tensor: $[B \times C \times T \times V \times M]$.
  - B = batch size,
  - C = number of channels,
  - T = number of video frames,
  - V = number of nodes,
  - M = number of skeletons in a frame.

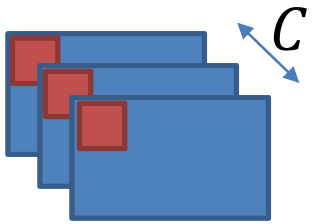
Spatio-Temporal GCN

- Feed the input data tensor into a PyTorch Conv2d module:
  - Need to rearrange axis : $[(B \times M) \times C \times T \times V]$, with batch size $[B \times M]$.
  - Every batch consists of C channels.
  - Each channel is a matrix with T rows and V columns.

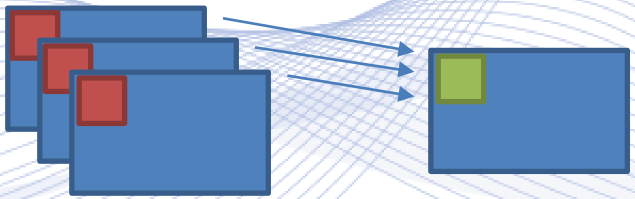


Spatio-Temporal GCN

- ***Spatial Convolution block***:
  - Uses $[1 \times 1]$ kernel, that ensures that features from a frame do not overlap with other frames.



  - Sums all the values from the C channels and returns a single value for each node.

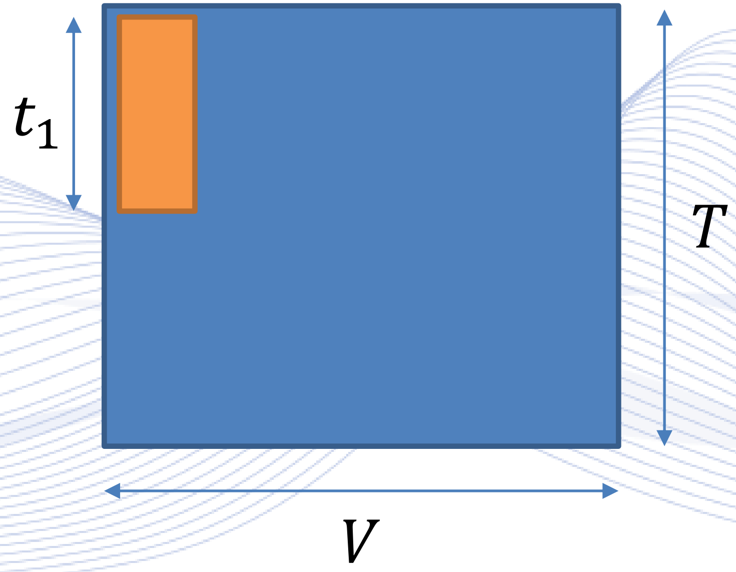


- The spatial convolution output is then ***multiplied with the Adjacency matrix***.

Spatio-Temporal GCN

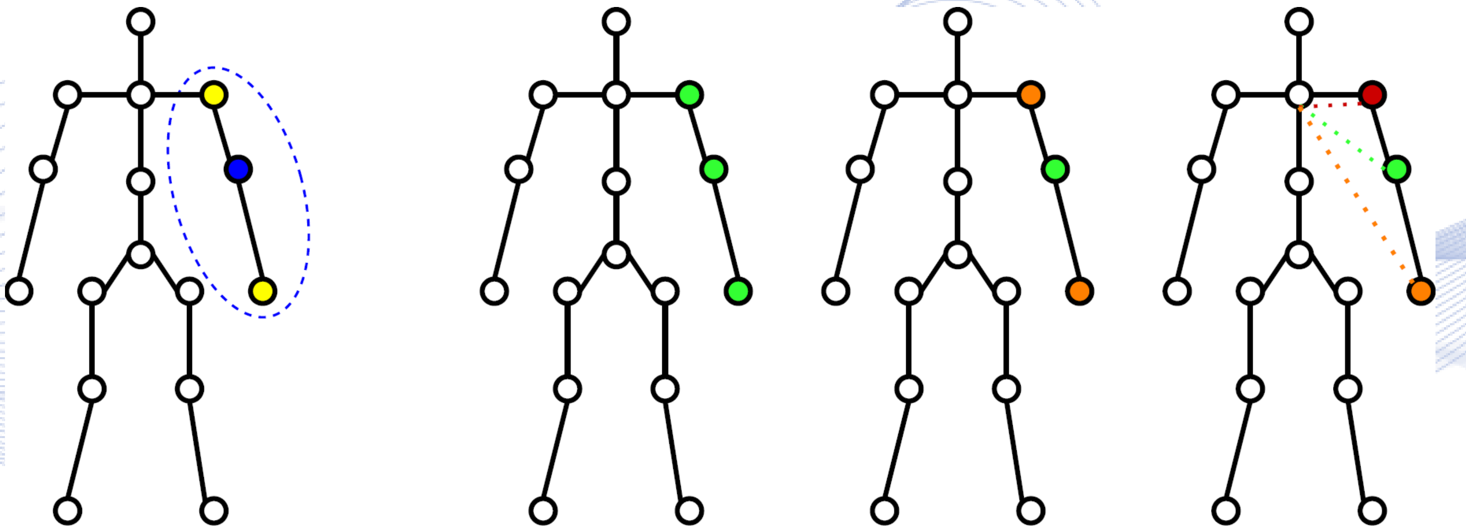
- The multiplication output is fed into a ***Temporal Convolution block***.

- The Temporal Convolution uses a $[t_1 \times 1]$ kernel:

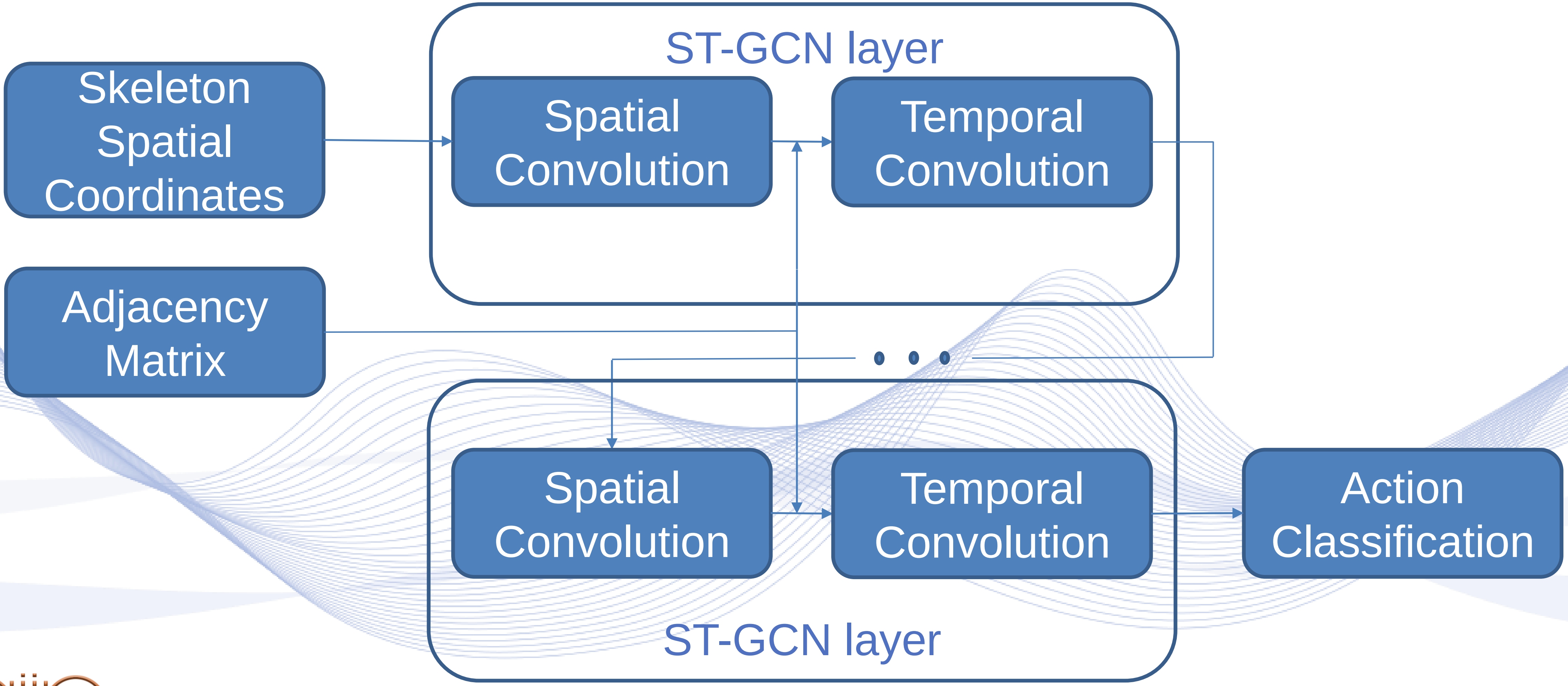


Spatio-Temporal GCN

- Deal with absence of node ordering, introduced by [NIE2016]:
  - Partition Strategies to create subsets:
    - ***Uni-labeling***, all nodes in a neighborhood are treated the same.
    - ***Distance based***, 1st subset: root node, 2nd subset: 1-hop neighborhood.
    - ***Spatial location based*** , 1st subset: root node, 2nd subset: centripetal nodes (closer to center than root), 3rd subset: centrifugal nodes (further away).

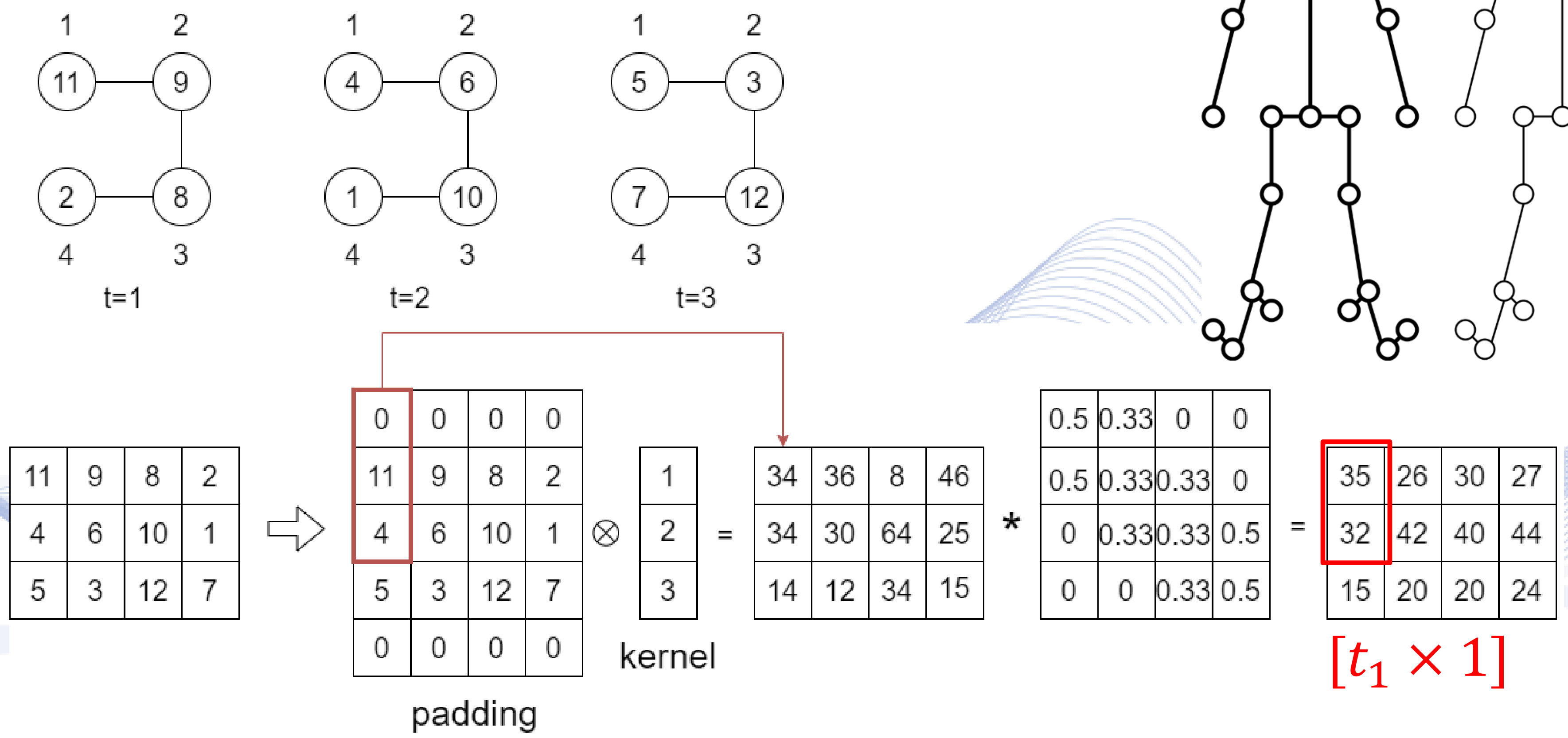


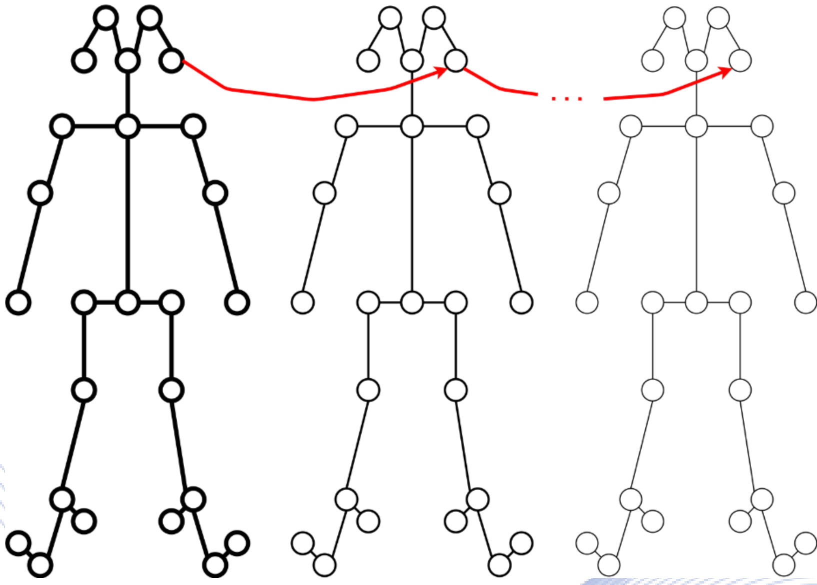
Spatio-Temporal GCN



Spatio-Temporal GCN

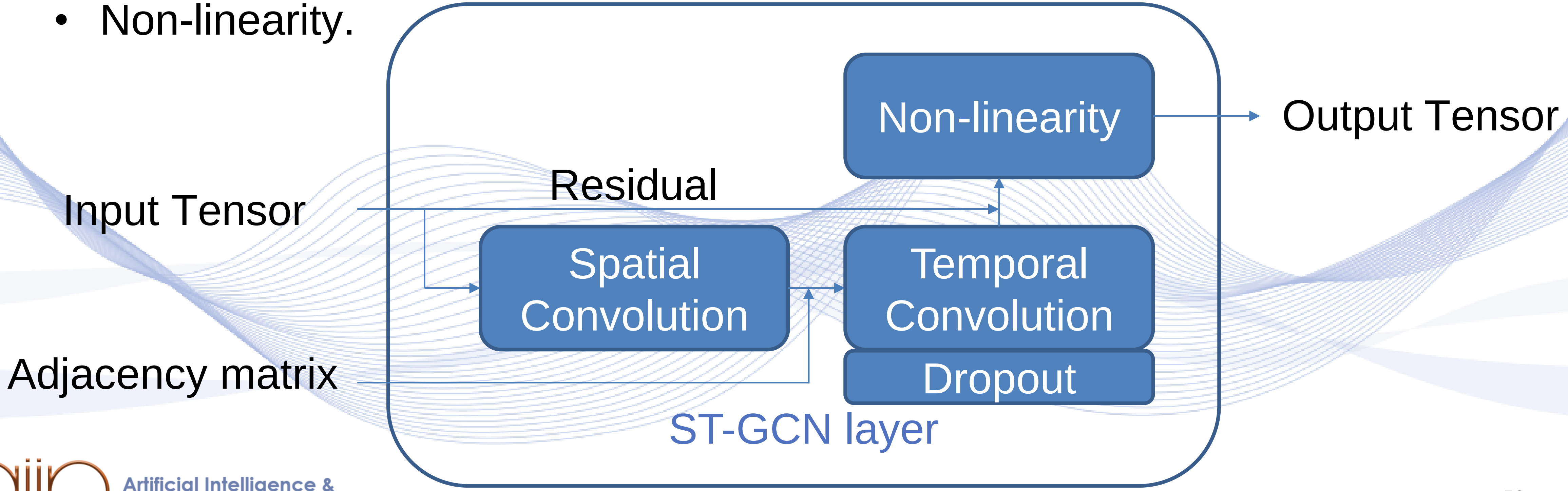
- A sub-graph example of 4 joints and 3 frames:





Spatio-Temporal GCN

- The ***ST-GCN layer*** is also equipped with:
  - A Residual mechanism,
  - Dropout,
  - Non-linearity.



Bibliography

[ACTIV] https://en.wikipedia.org/wiki/Activation_function

[GEOM] http://geometricdeeplearning.com/ (Geometric Deep Learning on Graphs and Manifolds).

[BRU2013] J. Bruna, et al. "Spectral networks and locally connected networks on graphs." *arXiv preprint arXiv:1312.6203* (2013).

[HEN2015] M. Henaff, J. Bruna, and Y. LeCun. "Deep convolutional networks on graph-structured data." *arXiv preprint arXiv:1506.05163* (2015).

[SHU2016] D. I. Shuman, B. Ricaud, and P. Vandergheynst. "Vertex-frequency analysis on graphs." Applied and Computational Harmonic Analysis 40.2 (2016): 260-291.

Bibliography

[DEF2016] M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." arXiv preprint arXiv:1606.09375 (2016).

[LEV2018] R. Levie, et al. "Cayleynets: Graph convolutional neural networks with complex rational spectral lters." IEEE Transactions on Signal Processing 67.1 (2018): 97-109.

[SCA2008] F. Scarselli, et al. "The graph neural network model." IEEE transactions on neural networks 20.1 (2008): 61-80.

[KIPF2016] T. N. Kipf, and M. Welling. "Semi-supervised classiffication with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[SUK2016] S. Sukhbaatar, A. Szlam, and R. Fergus. "Learning multiagent communication with backpropagation." arXiv preprint arXiv:1605.07736 (2016).

Bibliography

[HAM2017] W. L. Hamilton, R. Ying, and J. Leskovec. "Inductive representation learning on large graphs." arXiv preprint arXiv:1706.02216 (2017).

[XU2018] K. Xu, et al. "How powerful are graph neural networks?." arXiv preprint arXiv:1810.00826 (2018).

[MON2017] F. Monti, et al. "Geometric deep learning on graphs and manifolds using mixture model cnns." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[VEL2017] P. Velickovic, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

[BRE2017] X. Bresson, and T. Laurent. "Residual gated graph convnets." arXiv preprint arXiv:1711.07553 (2017).

Bibliography

[YAN2018] S. Yan, Y. Xiong, and D. Lin. "Spatial temporal graph convolutional networks for skeleton-based action recognition." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

Bibliography

[PIT2016] I. Pitas (editor), “Graph analysis in social media”, CRC Press, 2016.

[PIT2021] I. Pitas, “Computer vision”, Createspace/Amazon, in press.

[PIT2017] I. Pitas, “Digital video processing and analysis” , China Machine Press, 2017 (in Chinese).

[PIT2013] I. Pitas, “Digital Video and Television” , Createspace/Amazon, 2013.

[NIK2000] N. Nikolaidis and I. Pitas, “3D Image Processing Algorithms”, J. Wiley, 2000.

[PIT2000] I. Pitas, “Digital Image Processing Algorithms and Applications”, J. Wiley, 2000.

Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

Contact: Prof. I. Pitas
pitas@csd.auth.gr