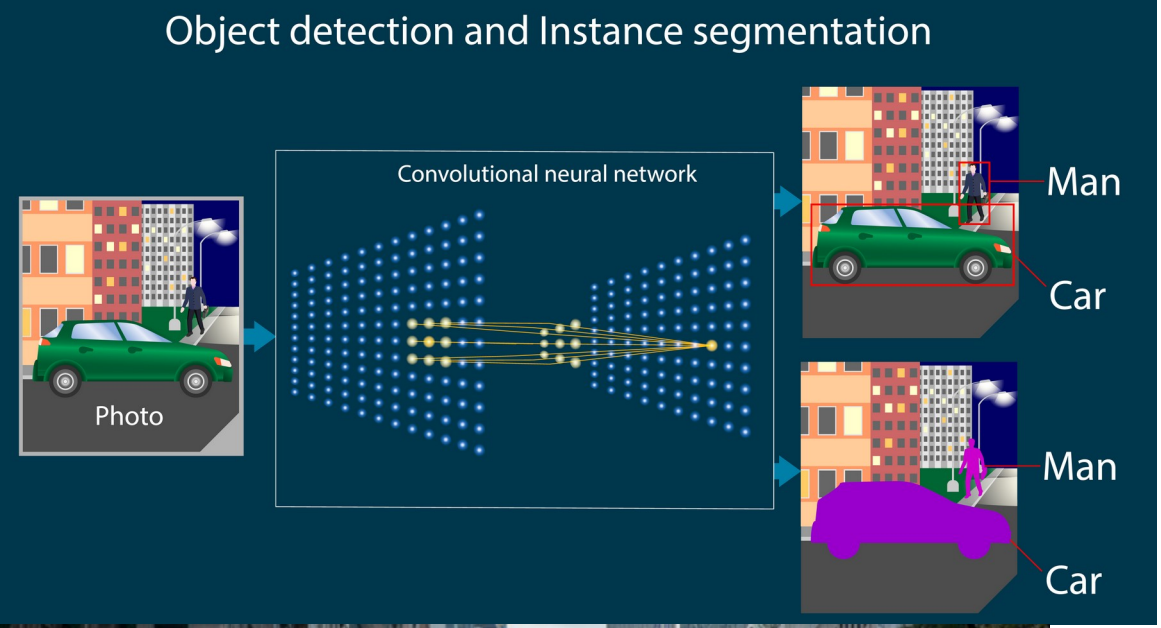
Graph Neural Networks

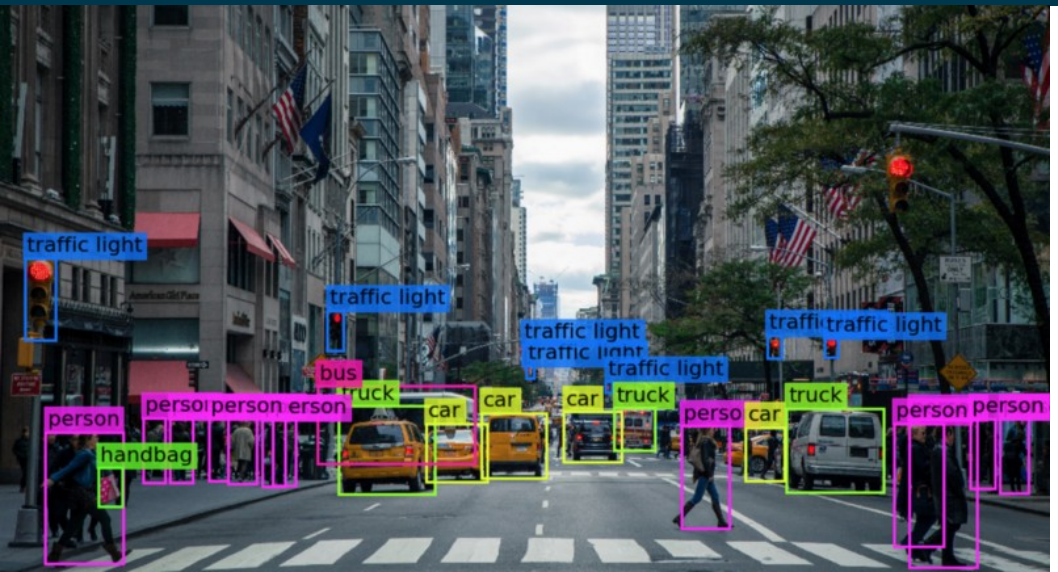
Alikhan Anuarbekov
anuarali@fel.cvut.cz

Outline

- What is a Geometrical Deep Learning
- Types of GNNS, their architecture
- Application of GNN on molecular data

Motivation: What is special about graphs?





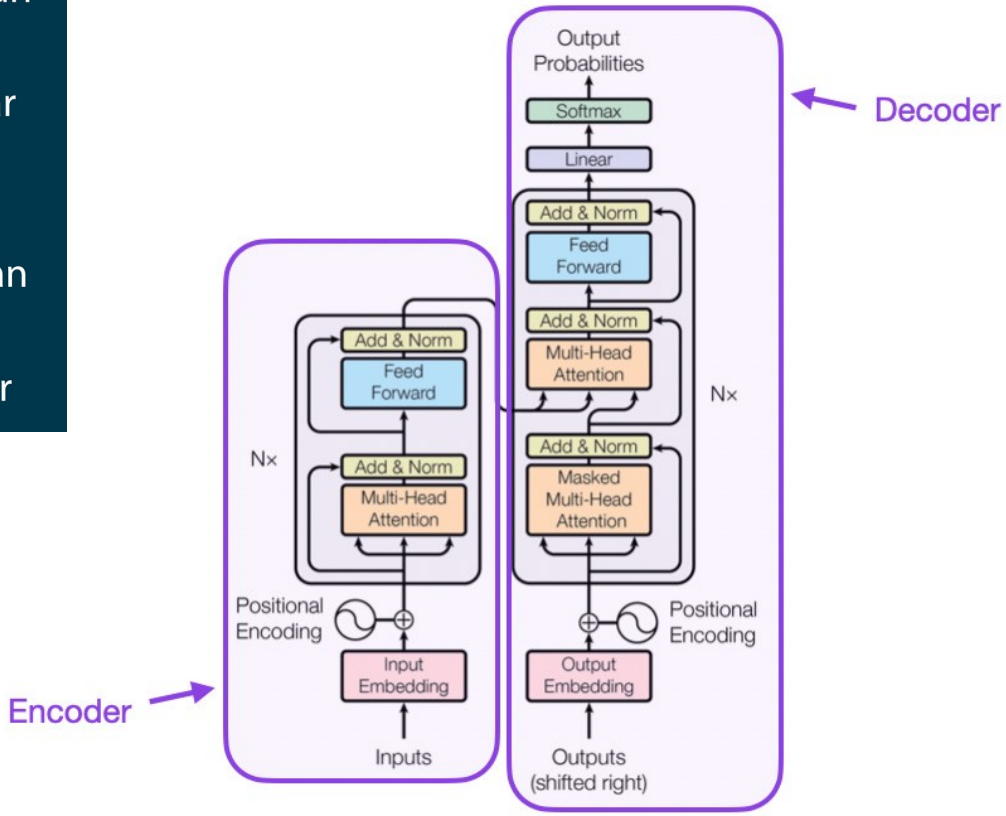
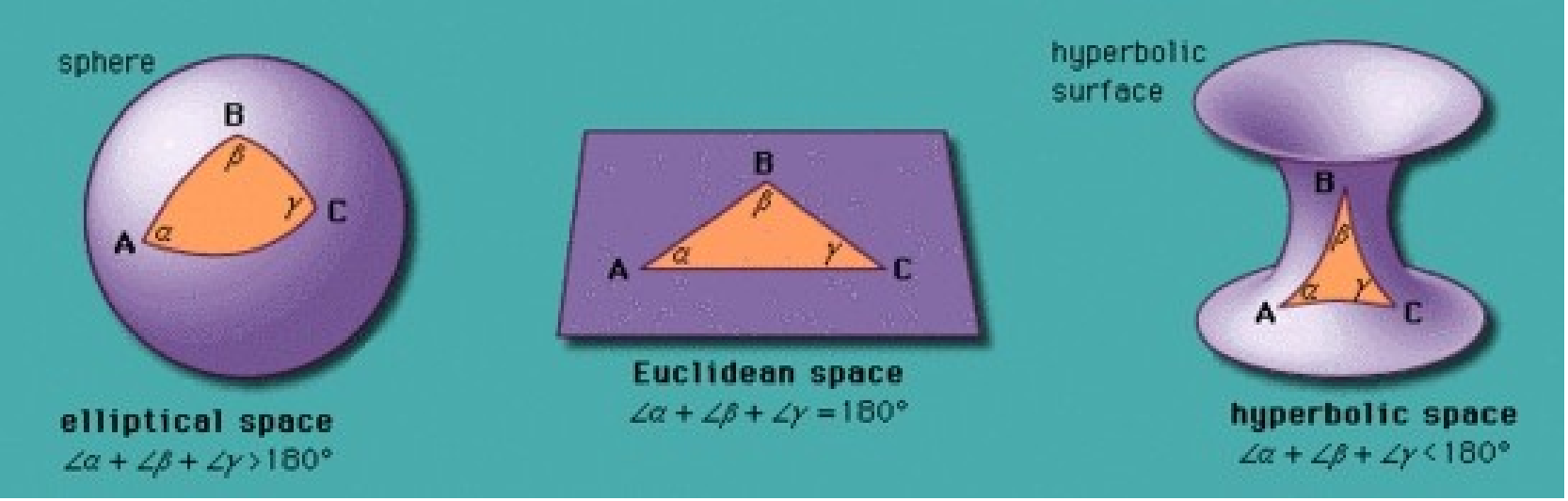
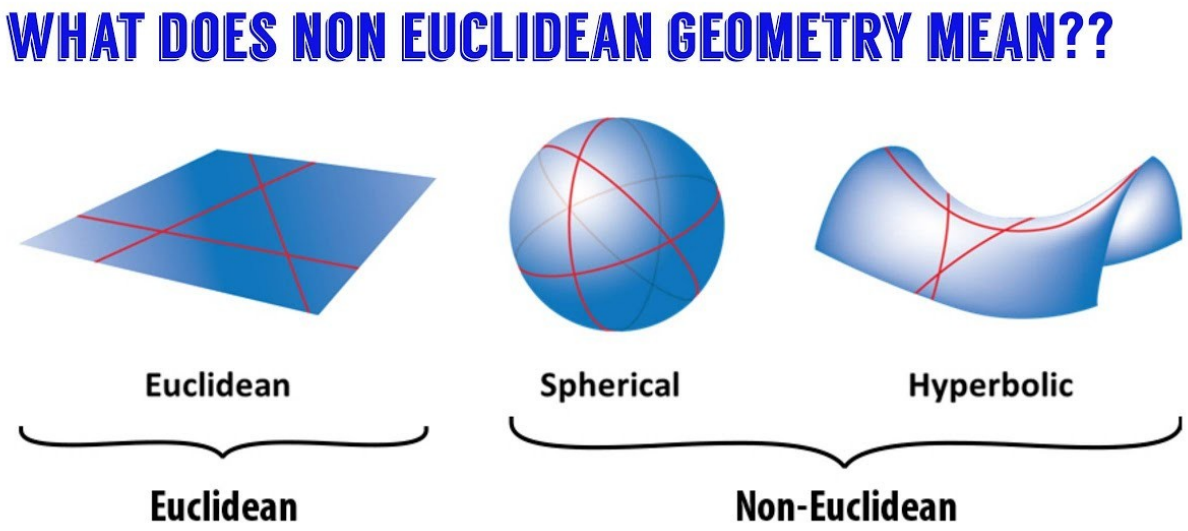


Figure 1: The Transformer - model architecture.



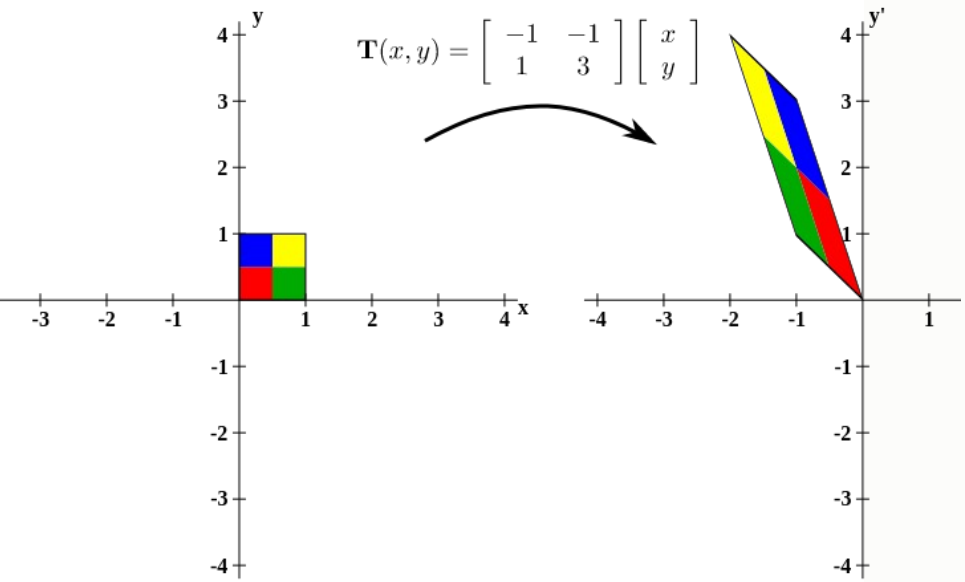
All of the previous approaches are so-called Euclidean

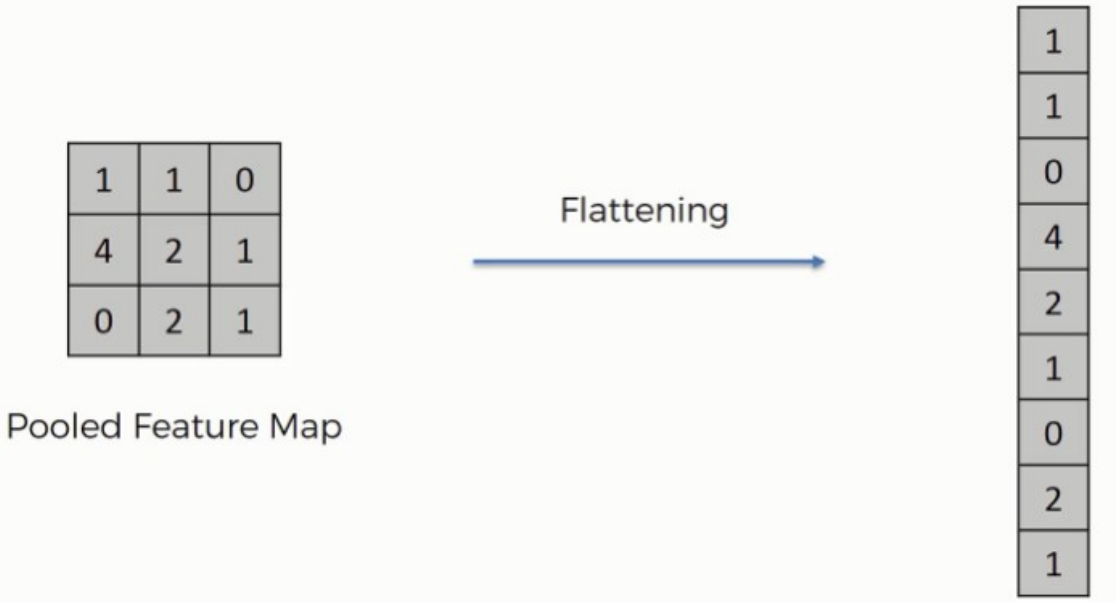


Motivation: What is special about graphs?

All of the previous approaches and data are so-called Euclidean

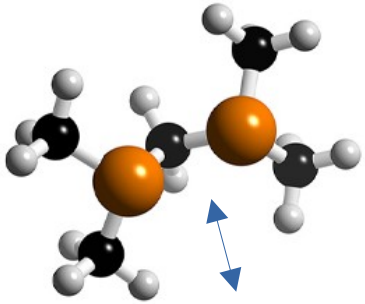
- The data lies in some multidimensional linear vector space
- The distance between two points is a straight line and has a distance metric



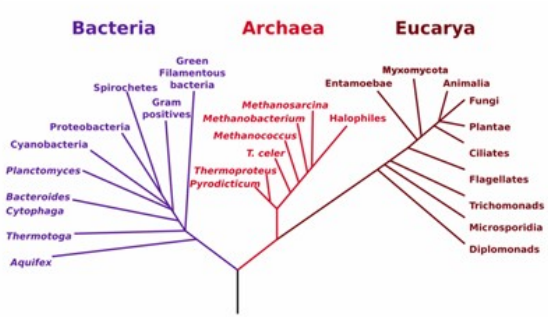


Motivation: What is special about graphs?

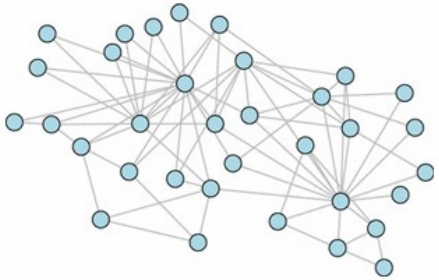
But there are data representations that do not follow the Euclidean vector space representation



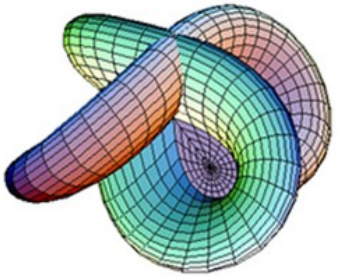
Molecules



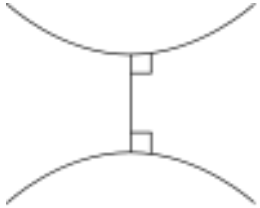
Trees



Networks

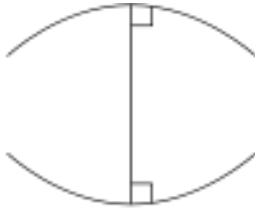


Manifolds



Hyperbolic

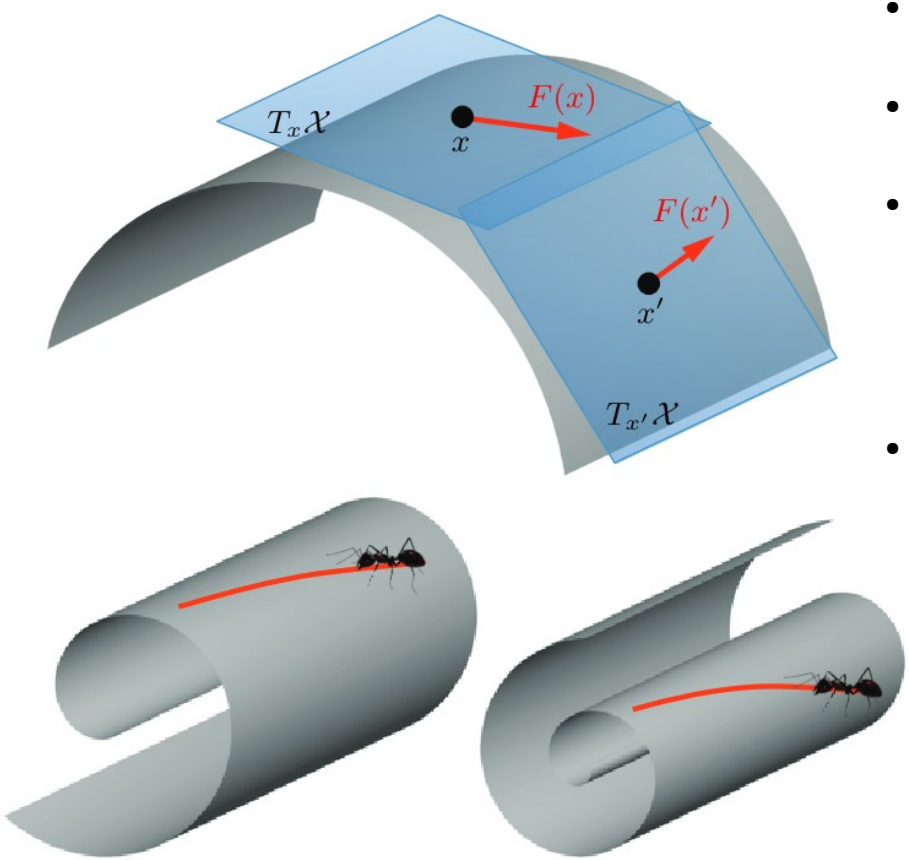
Euclidean



Elliptic

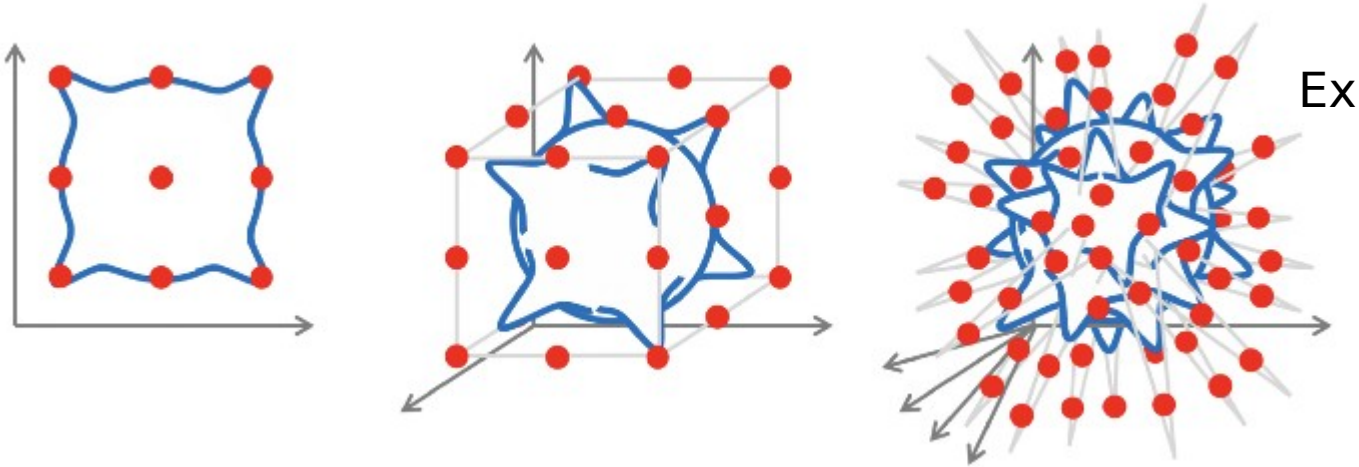
Motivation: What is special about graphs?

The main concern here – the need for local distance preservation

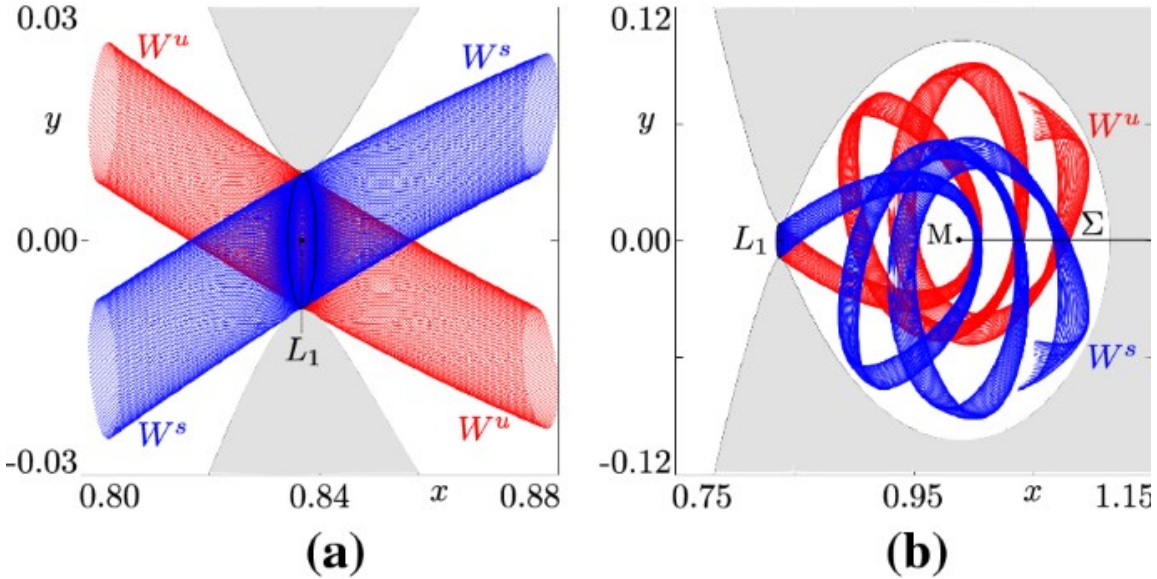


- Manifold = local Euclidean distance relations
- But the global Euclidean distances aren't fulfilled
- Problems: curse of dimensionality
  - Too hard to learn such such complex shape
  - Exponentially many examples
  - Many layers of standard linear FC layers
- Machine learning alternative approaches
  - Try dimension reduction: 3D -> 2D
  - Not all manifolds can be reduced to lower dimensions!

Motivation: What is special about graphs?



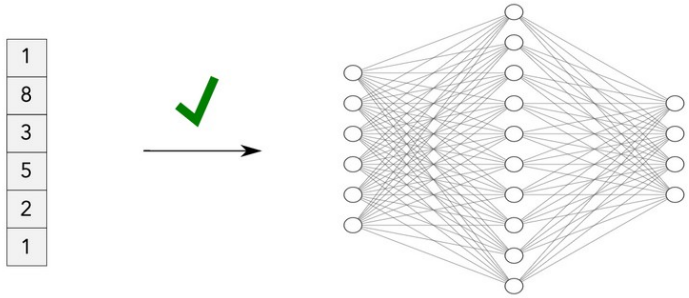
Exponentially complex shape

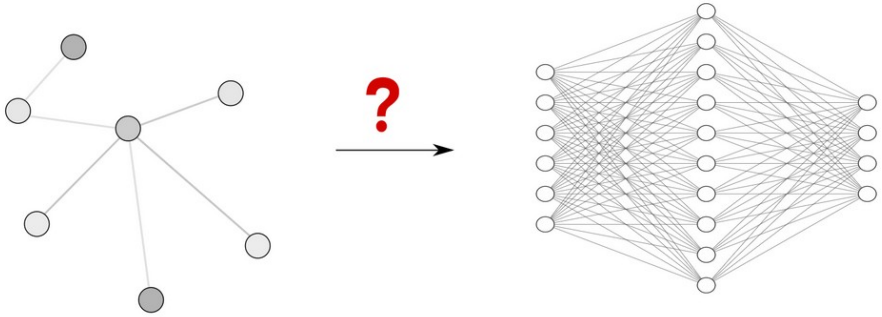


3D manifold geometrical structure

Motivation: What is special about graphs?

All of the previously learned Deep Learning concepts are not easily applicable to non-Euclidean data





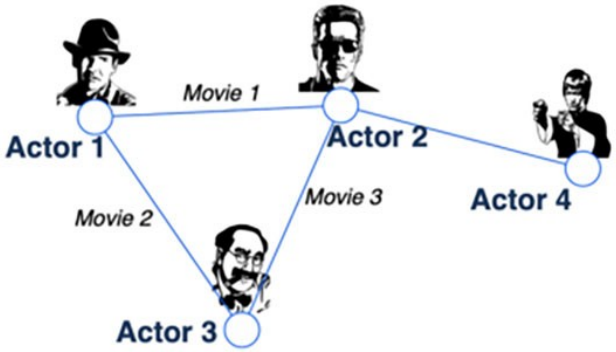
- Since we cannot efficiently represent the data in vector space without the loss of information
- We need to define some new operations to operate in this non-Euclidean space
- To put it shortly:
  ”We want to use the topology as input as well”
- This is the domain of ***Geometrical Deep Learning***

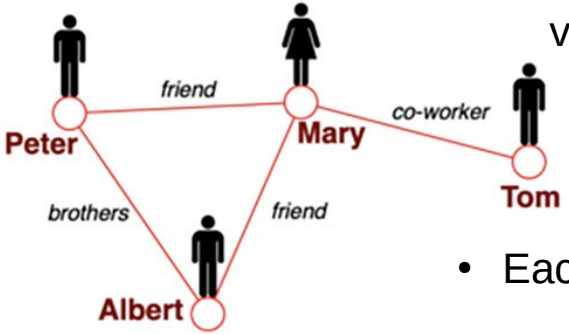
Graph Neural Networks (GNNs):

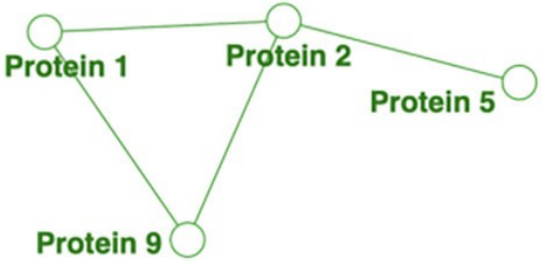
How to deal with non-Euclidean graphs?

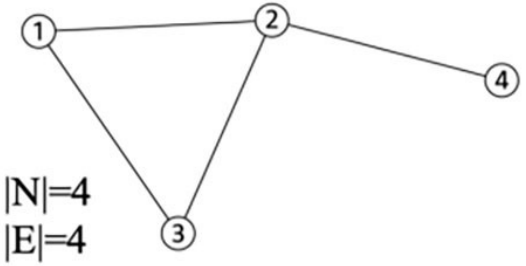
GNNs: How to deal with non-Euclidean graphs?

First – we will define the framework for our upcoming







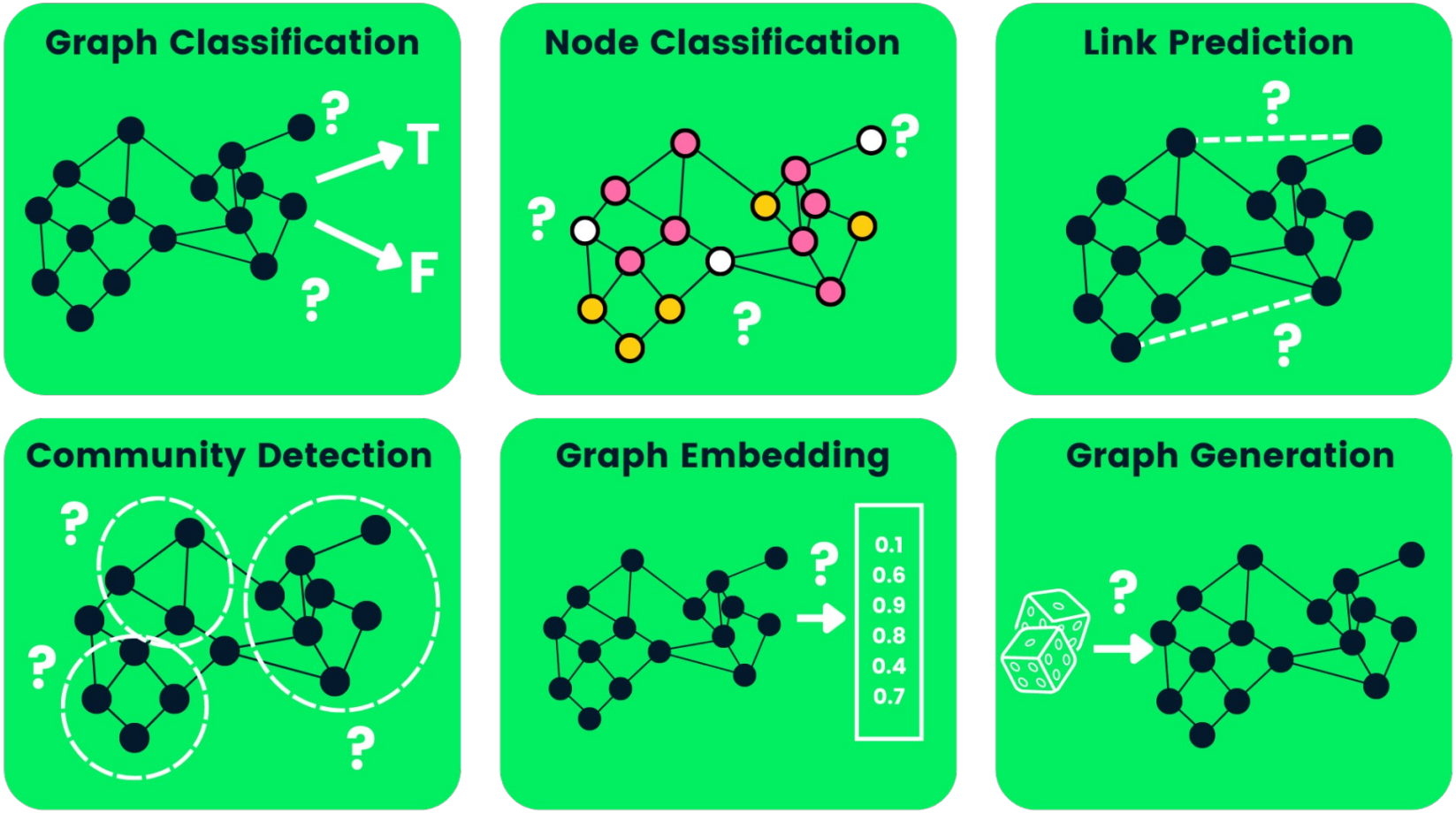


- Graph G = (V, E)
  - Can be directed or undirected
- Each node has its own features in form of vector $v \in V \rightarrow \vec{x}_v = (x_1, x_2, .., x_n)$
- Each edge can have its own feature vector

$$e = (v, u) \in E \rightarrow \vec{x}^{e}_{v,u} = (x_1, x_2, .., x_k)$$

GNNs: How to deal with non-Euclidean graphs?

Second – we define the type of tasks that we want to solve



https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial

Graph Neural Networks (GNNs):

Recurrent GNN approach

GNNs: How to deal with non-Euclidean graphs?

General recurrent algorithm on Graphs (information diffusion/message passing)

$$\vec{h}_v^{(t)} = \sum_{u \in N(v)} f\left(\vec{x}_v, \vec{x}_{v,u}^{e}, \vec{x}_u, \vec{h}_u^{(t-1)}\right) \qquad \forall v : \vec{h}_v^{(0)} = randomly$$

$$\vec{h}_{uv}^{(t)} = f\left(\vec{x}_u, \vec{x}_{uv}, \sum_{w \in N(u)/v} \vec{h}_{wu}^{(t-1)}\right) \qquad \vec{h}_{uv}^{(0)} = \vec{0}$$

(It can be applied to edge feature messages also)

- Regardless of number of neighbors aggregate (collect) their hidden states
- Run until equilibrium (convergence)
- F(.) satisfy conditions of contraction (convergence)
  - Before Neural Networks (Label propagation):
    - Vector of classification probabilities per node
    - Simple recurrent update algorithm

$$f(t+1) = \alpha S f(t) + (1-\alpha) Y$$

$$S = \text{adjacency matrix}$$

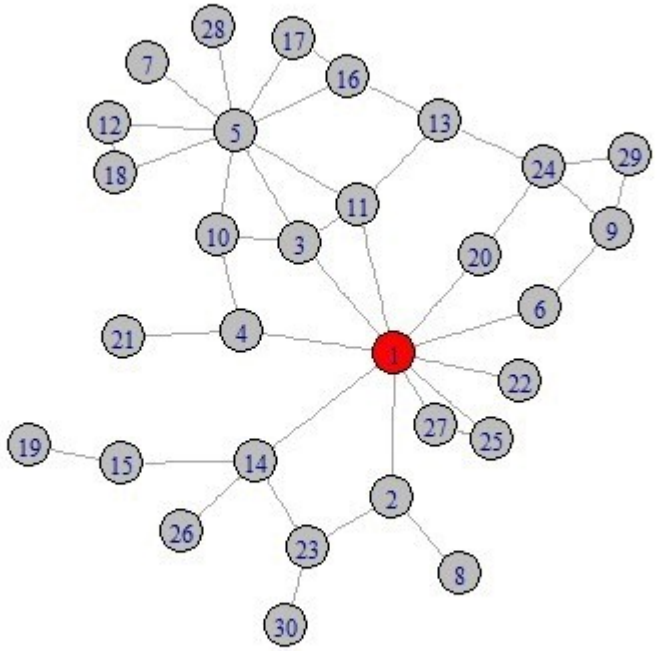
$$Y = \text{initial states of nodes}$$

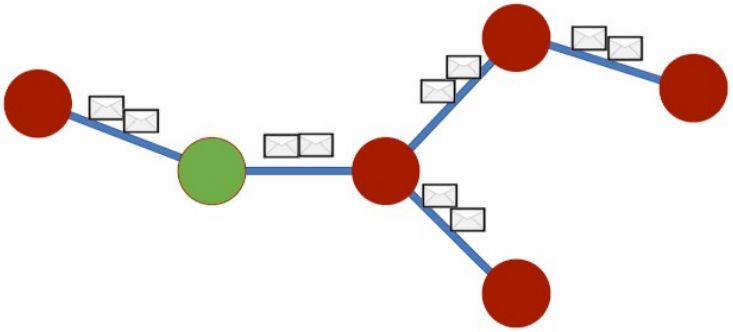
$$\alpha \in (0,1)$$

GNNs: How to deal with non-Euclidean graphs?

General recurrent algorithm on Graphs (information diffusion/message passing)

Label propagation = recurrent information diffusion algorithm





https://medium.com/@mohammadsharique.cse/a-classical-graph-neural-network-gnn-graphs-tell-stories-b80152a725d9

https://makeagif.com/gif/network-diffusion-9t0NVQ

GNNs: How to deal with non-Euclidean graphs?

Recurrent Graph Neural Network (RecGNN) from General recurrent algorithm

$$\vec{h}_v^{(t)} = \sum_{u \in N(v)} f\left(\vec{x}_v, \vec{x}_{v,u}^{e}, \vec{x}_u, \vec{h}_u^{(t-1)}, W\right) \quad \forall v : \vec{h}_v^{(0)} = randomly$$

- Instead of probabilities = label output per node, use a feature vector

- Try to incorporate some weights and non-linearities in F(.) function:

  - W in our case is the weight matrix
  - Sigmoid/tanh/Softmax can be applied as the activation/output functions
  - Can have multiple weight matrices combined in layers
  - Should have specific regularization and normalization terms to satisfy convergence

- Training:
  1) Run the recurrent chain until convergence
  2) Use back-propagation on final converged output to update W
  3) Repeat

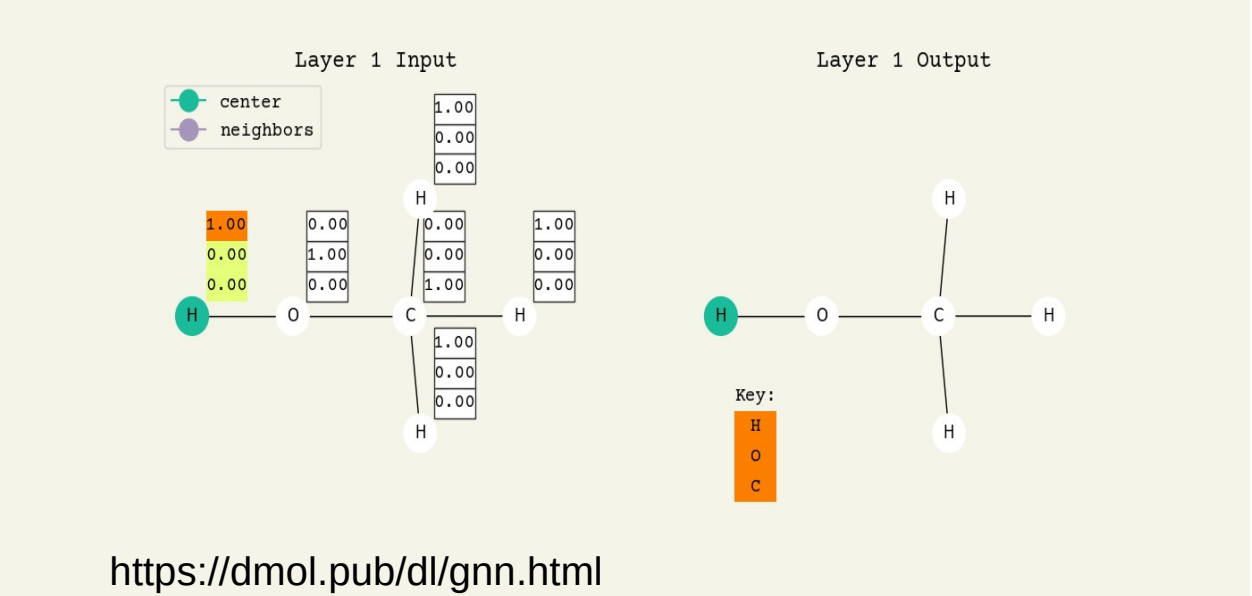
GNNs: How to deal with non-Euclidean graphs?

Recurrent Graph Neural Network (RecGNN)

$$\vec{h}_v^{(t)} = F\left(W_1 \vec{h}_v^{(t-1)} + \sum_{u \in N(v)} W_2 \vec{h}_u^{(t-1)}\right)$$

$$F = \text{activation function} = sigmoid / \tanh / ReLU / \ldots$$

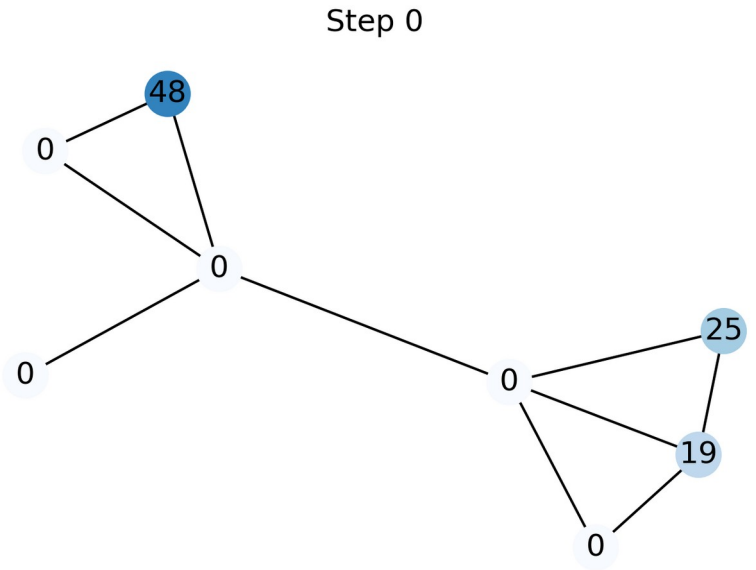
- Single layer instead of constants
- Can be generalized to multiple layers
- Weights are shared across time iterations



https://dmol.pub/dl/gnn.html

GNNs: How to deal with non-Euclidean graphs?

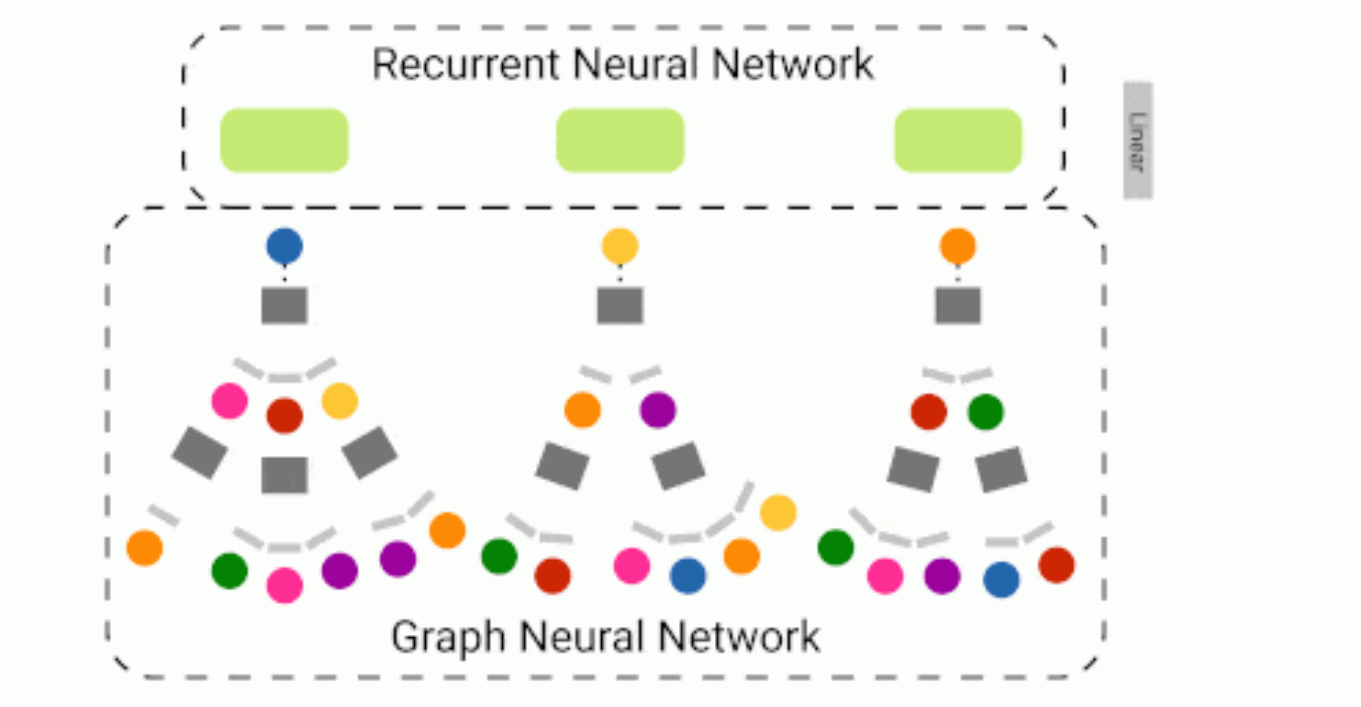
Recurrent Graph Neural Network (RecGNN), Training process



https://blog.twitter.com/engineering/en_us/topics/insights/2022/graph-machine-learning-with-missing-node-features

GNNs: How to deal with non-Euclidean graphs?

Recurrent Graph Neural Network (RecGNN), Training process



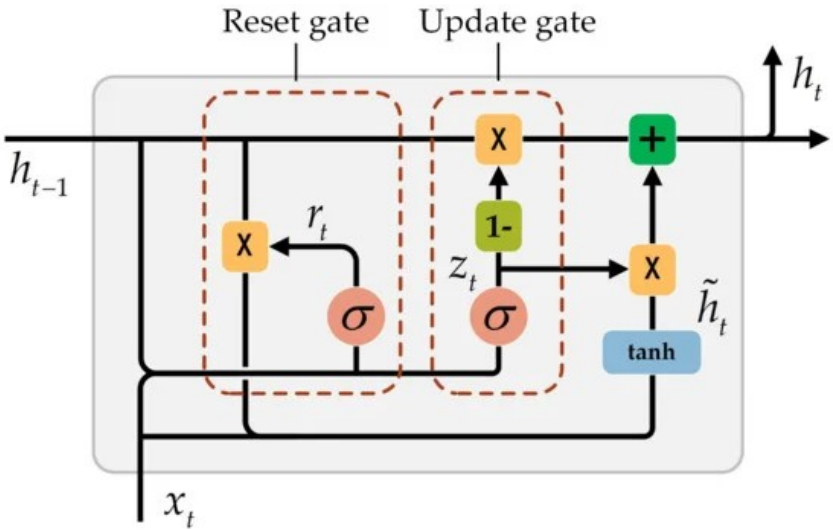
https://medium.com/stanford-cs224w/wikinet-an-experiment-in-recurrent-graph-neural-networks-3f149676fbf3

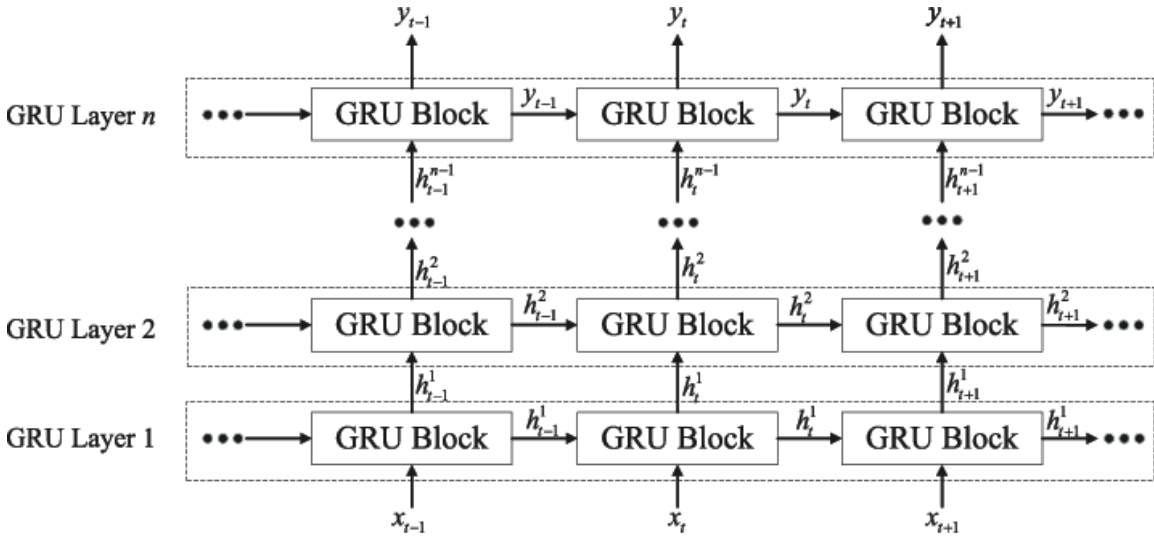
GNNs: How to deal with non-Euclidean graphs?

Recurrent Graph Neural Network (RecGNN)

$$\vec{h}_v^{(t)} = GRU\left(\vec{h}_u^{(t-1)}, \sum_{u \in N(v)} W \vec{h}_u^{(t-1)}\right)$$

- ***Gated RecGNN*** – instead of convergence just do a fixed number of different linear layers
- More parameters, more memory and computation requirement
- Less time to run, no need for global convergence guarantees/constraints





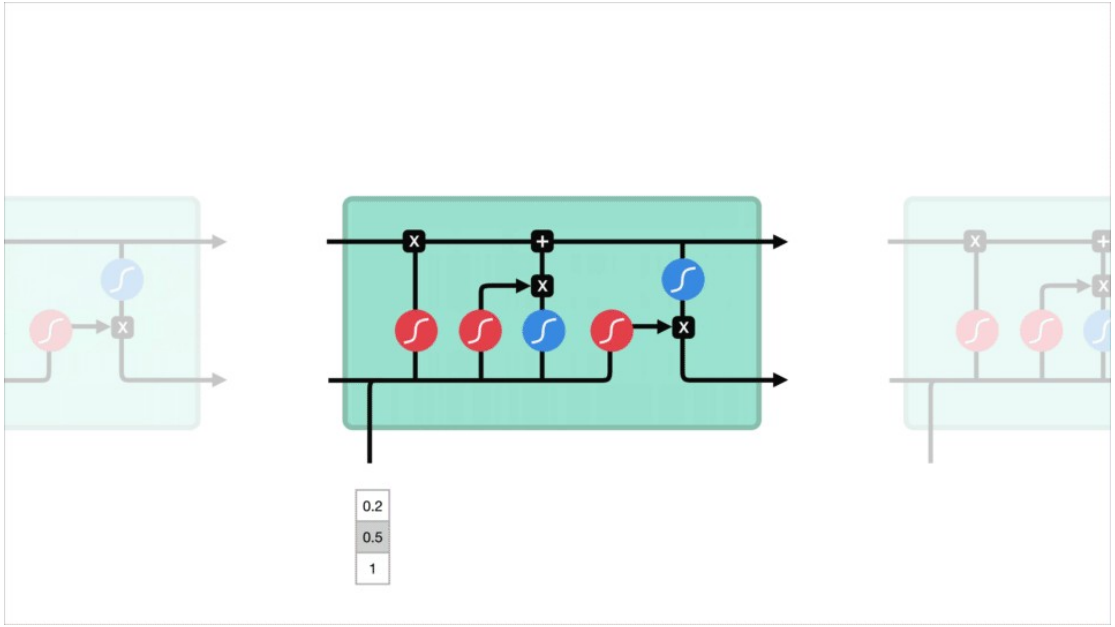
https://www.researchgate.net/figure/The-architecture-of-a-multi-layer-gated-recurrent-neural-network_fig5_330723201

GNNs: How to deal with non-Euclidean graphs?

Recurrent Graph Neural Network (RecGNN)

$$\vec{h}_v^{(t)} = GRU\left(\vec{h}_u^{(t-1)}, \sum_{u \in N(v)} W \vec{h}_u^{(t-1)}\right)$$

- ***Gated RecGNN*** – instead of convergence just do a fixed number of different linear layers



https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

Graph Neural Networks (GNNs):

Convolution GNNs

GNNs: How to deal with non-Euclidean graphs?

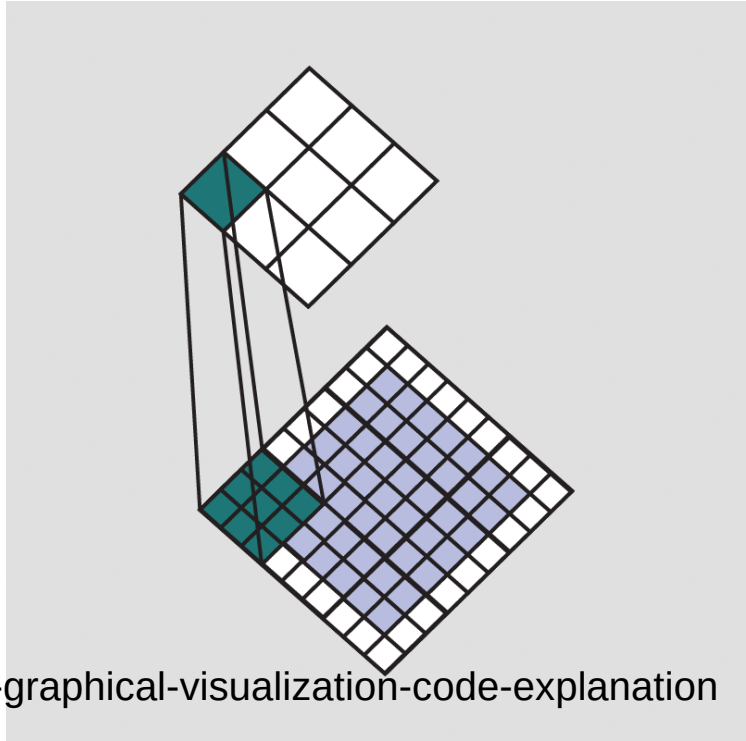
Convolution Neural Network (CNN) on image data (Spatial-based)

$$X * g_\theta^l = Y \qquad y_{ij} = \sigma\left(\sum_m^b \sum_n^d w_{mn}^l x_{i+m,\, j+n} + b_{ij}^l\right)$$

- Locality, Shift invariance, highly reduce parameter count
- Need to generalize this approach to graphs
  - Since graphs have different number of neighbors
- There exists two approaches to it:

1) Spectral-based Convolution Neural Networks

2) Spatial-based Convolution Neural Networks

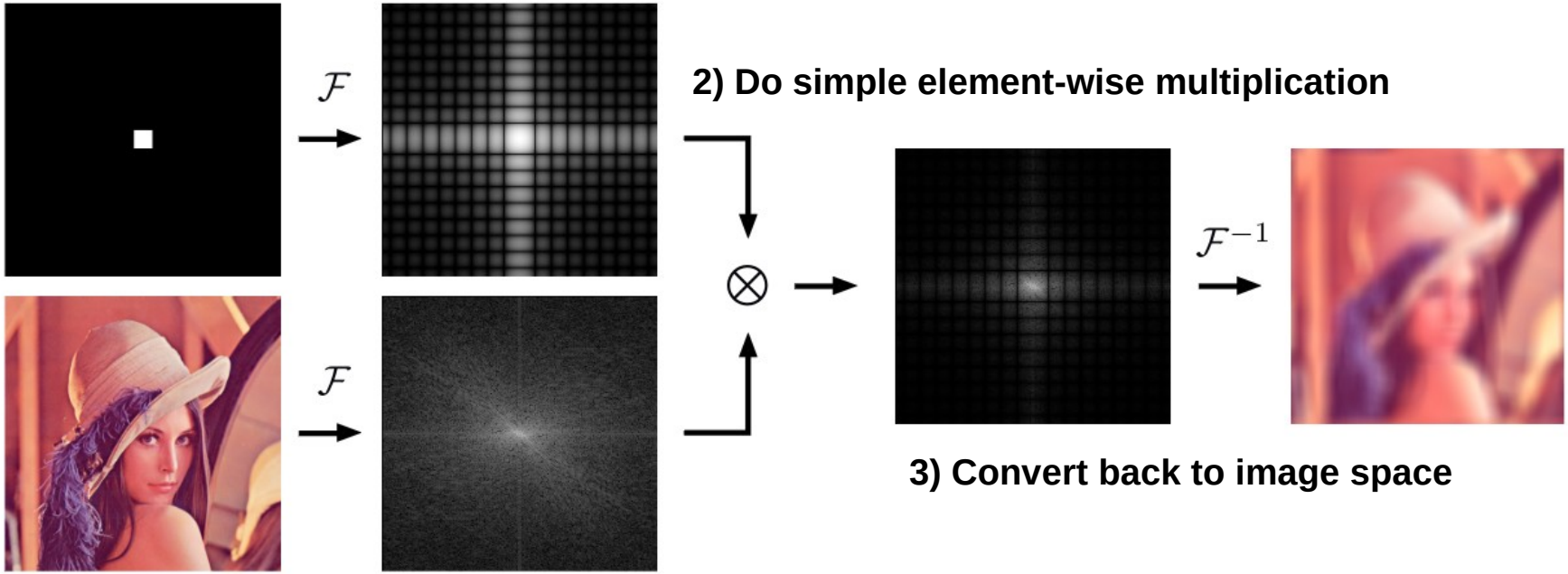


https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation

GNNs: How to deal with non-Euclidean graphs?

Convolution on signal analysis (Spectral-based)

$$(f*g)(s,t)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s,t)\,g(s-x,t-y)\,dxdy \quad \in O(|f||g|)$$



2) Do simple element-wise multiplication

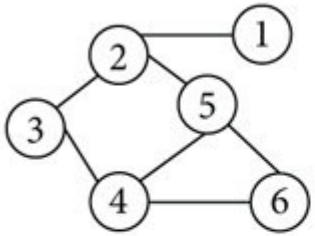
3) Convert back to image space

1) First convert to Frequency Space
Fourier Transform (FT)

$$F^{-1}F(f*g)=F^{-1}(F(f)\odot F(f))\in O(|f||\log f|)$$

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

Labeled graph	Degree matrix
	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$
Adjacency matrix	**Laplacian matrix**
$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$

$$L = I_n - D^{-1/2} A D^{-1/2}$$

$$D = \text{Degree matrix}$$

$$A = \text{Adjacency matrix}$$

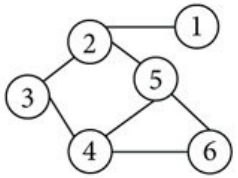
$$I_n = \text{Identity matrix}$$

- Assume undirected graphs only
- L = real, symmetric, semi-definite (all eigenvalues are >= 0)
- Has a property of Eigen decomposition
  - Diagonalization in orthonormal space

https://www.researchgate.net/figure/Example-of-degree-adjacency-and-Laplacian-matrices-for-an-undirected-graph_fig3_362345733

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

Labeled graph	Degree matrix
	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$
Adjacency matrix	**Laplacian matrix**
$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$

$$L = U \Lambda U^T$$

$$U = \text{orthonormal matrix}$$

$$\Lambda = \text{Diagonal matrix}, \ \lambda_i \geq 0$$

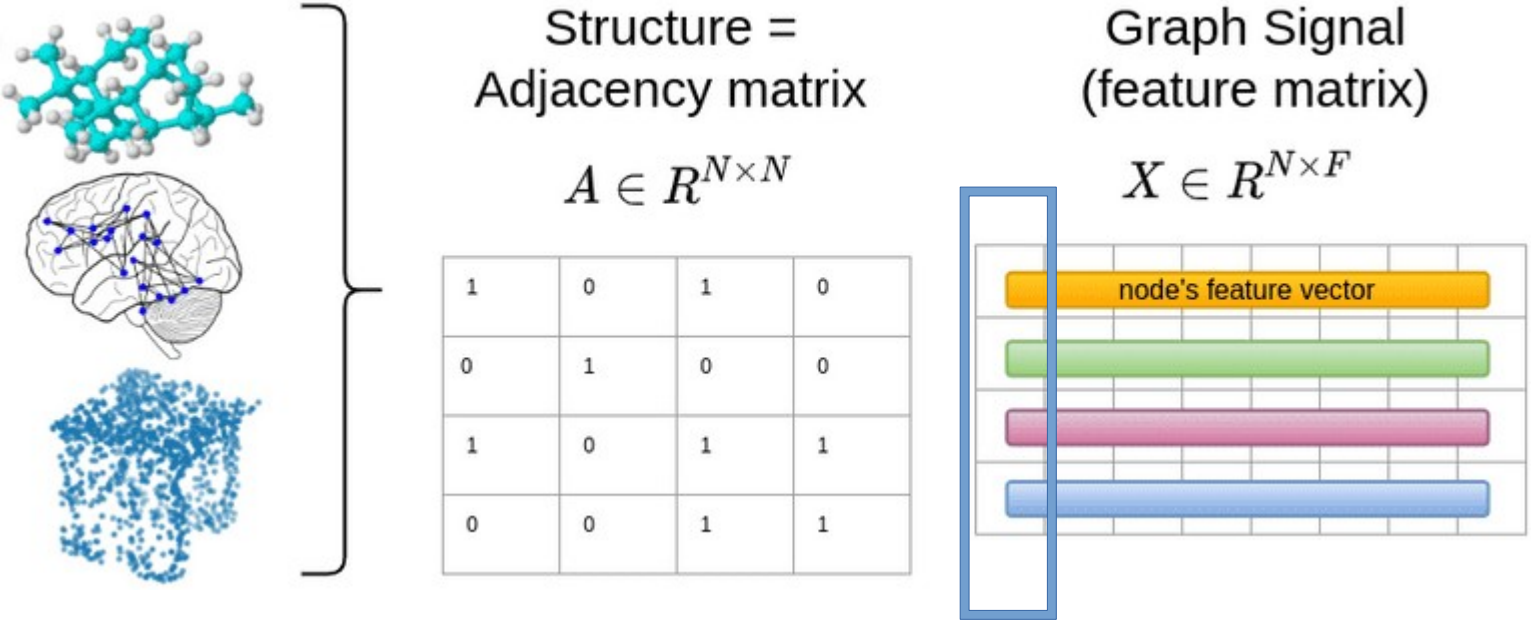
- This orthonormal space is exactly the Spectral Space!

$$F(\vec{x}) = U^T \vec{x}$$

$$F^{-1}(\vec{x}) = U \vec{x}$$

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach



$$g = \text{filter/kernel} \in R^N$$

$$\vec{x} * g_\theta = F^{-1}\left(F(\vec{x}) \odot F(g_\theta)\right) = U U^T \vec{x} \odot U^T g_\theta = U\, diag\left(U^T g_\theta\right) U^T \vec{x}$$

https://theaisummer.com/graph-convolutional-networks/

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

$$H_{:,j}^{(k)} = \sigma\left(\sum_{i=0}^{f_{\text{in}}} U \Theta_{i,j}^{k} U^T H_{:,i}^{(k-1)}\right) \quad j = 1, 2, .., f_{\text{out}}$$

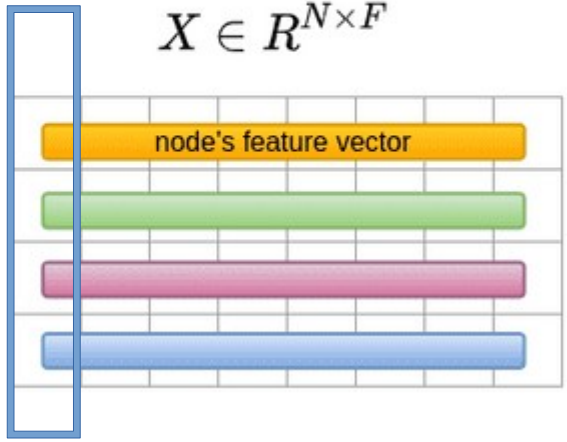
$$\Theta = diag\left(U^T g_\theta\right)$$

$$H^{(0)} = X, \quad H^{(k)} \in R^{f_{\text{out}} x N}, \quad H^{(k-1)} \in R^{f_{\text{in}} x N}$$

- So, we are apply the same kernel across the feature axis

Graph Signal
(feature matrix)

$X \in R^{N \times F}$



GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

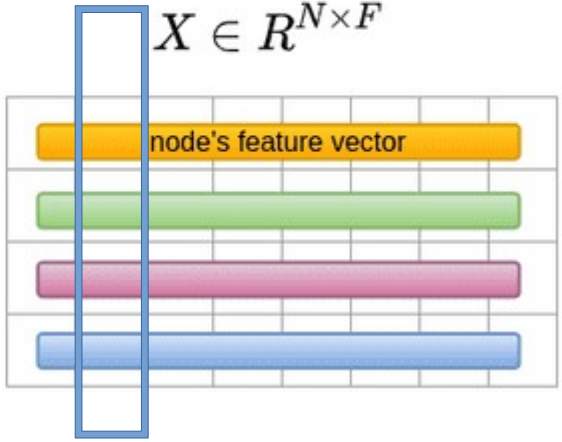
$$H_{:,j}^{(k)} = \sigma\left(\sum_{i=0}^{f_{\text{in}}} U \Theta_{i,j}^{k} U^T H^{(k-1)}\right) \quad j = 1, 2, .., f_{\text{out}}$$

$$\Theta = diag\left(U^T g_\theta\right)$$

$$H^{(0)} = X, \quad H^{(k)} \in R^{f_{\text{out}} x N}, \quad H^{(k-1)} \in R^{f_{\text{in}} x N}$$

- So, we are apply the same kernel across the feature axis

Graph Signal
(feature matrix)



GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

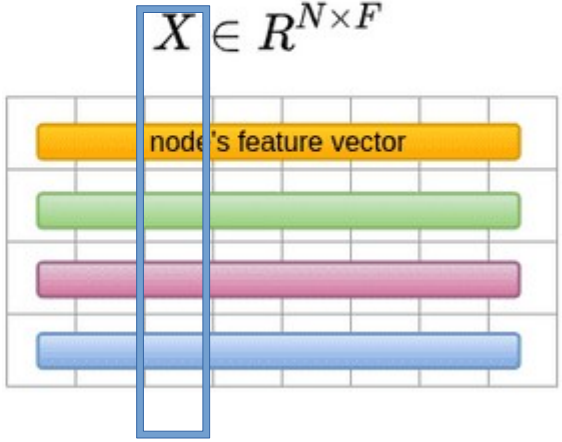
$$H_{:,j}^{(k)} = \sigma\left(\sum_{i=0}^{f_{\text{in}}} U \Theta_{i,j}^{k} U^{T} H_{:,i}^{(k-1)}\right) \quad j = 1, 2, .., f_{\text{out}}$$

$$\Theta = diag\left(U^{T} g_{\theta}\right)$$

$$H^{(0)} = X, \quad H^{(k)} \in R^{f_{\text{out}} x N}, \quad H^{(k-1)} \in R^{f_{\text{in}} x N}$$

- So, we are apply the same kernel across the feature axis

Graph Signal
(feature matrix)



GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

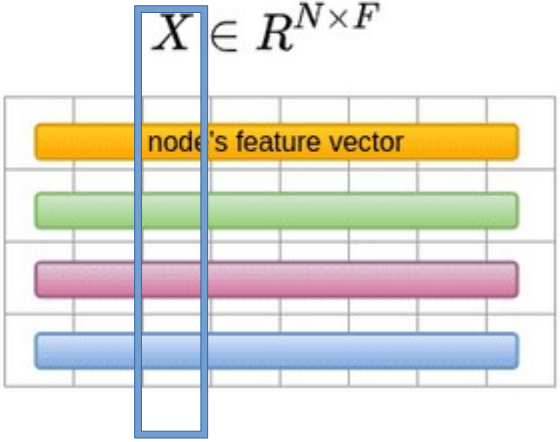
$$H^{(k)}_{:,j} = \sigma\left(\sum_{i=0}^{f_{\text{in}}} U \Theta^{k}_{i,j} U^{T} H^{(k-1)}_{:,i}\right) \quad j = 1,2,..,f_{\text{out}}$$

$$\Theta = diag\left(U^{T} g_{\theta}\right)$$

$$H^{(0)} = X, \quad H^{(k)} \in R^{f_{\text{out}} x N}, \quad H^{(k-1)} \in R^{f_{\text{in}} x N}$$

- But:
  - After every change to structure – train all over
  - Convolution not local, a global sliding window
  - Eigen-decomposition is O(n^3)
  - Not practical, just a theory…..

Graph Signal
(feature matrix)



GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spectral-based approach

- ***Chebyshev Spectral CNN (ChebNet):***
  - If Laplacian is sparse, can decompose it
  - Use decomposition by Chebyshev polynomials instead of Eigen
  - No need for O(n^3), but only approximately for O(n)
  - Further layers are reduced even to O(1)

$$\Theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{\Lambda})$$

$$\tilde{\Lambda} = 2\Lambda / \lambda_{max} - I_n$$

- ***Graph Convolution Network (GCN, 2018)***:
  - Use even more approximation
    - Same principle as computing first elements of Taylor series
    - Adding up to K is exact value, compute only i=0, i=1
    - First successful Spectral-based ConvGNN

- Both methods are still unusable for directed or changeable graphs

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spatial-based approach

Neural Network for Graphs (NN4G)

- Finally, the same approach as in standard ConvNN, by applying a sliding window
- Different number of neighbors per node is solved by **Aggregation**

$$\vec{h}_v^{(k)} = f\left(W^{(k)^T} \vec{x}_v + \sum_{u \in N(v)} \Theta^{(k)^T} \vec{h}_u^{(k-1)}\right)$$

$$H_v^{(k)} = f\left(X W^{(k)} + A H^{(k-1)} \Theta^{(k)^T}\right)$$

$$\vec{h}_v^{(0)} = \vec{0}$$

$$f = \text{activation function}$$

- Efficient to compute
- Can be applied to directed graphs
- Can change a graph as long as nodes are the same/shared

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spatial-based approach

Neural Network for Graphs (NN4G)

- Finally, the same approach as in standard ConvNN, by applying a sliding window
- Different number of neighbors per node is solved by **Aggregation**

$$\vec{h}_v^{(k)} = f\left(W^{(k)^T} \vec{x}_v + \boxed{\sum_{u \in N(v)} \Theta^{(k)^T} \vec{h}_u^{(k-1)}}\right)$$

$$H_v^{(k)} = f\left(X W^{(k)} + A H^{(k-1)} \Theta^{(k)^T}\right)$$

$$\vec{h}_v^{(0)} = \vec{0}$$

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spatial-based approach

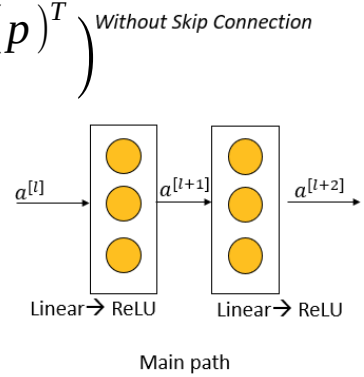
Neural Network for Graphs (NN4G)

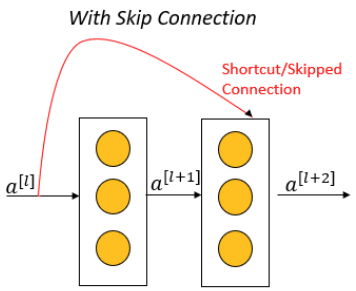
- Finally, the same approach as in standard ConvNN, by applying a sliding window
- Different number of neighbors per node is solved by **Aggregation**
- Can also enhance – introduce *Skip connection* (ResNet)

$$\vec{h}_v^{(k)} = f\left(W^{(k)^T} \vec{x}_v + \sum_{p=1}^{k} \sum_{u \in N(v)} \Theta^{(p)^T} \vec{h}_u^{(p-1)}\right)$$

$$H_v^{(k)} = f\left(X W^{(k)} + \sum_{p=1}^{k} A H^{(p-1)} \Theta^{(p)^T}\right)$$

$$\vec{h}_v^{(0)} = \vec{0}$$





- Shortcut added before non-linearity (ReLU) to the main path
- Stack these Residual blocks to form much deeper neural network

https://www.pluralsight.com/guides/introduction-to-resnet

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spatial-based approach

Message Passing Neural Network (MPNN)

- Generalizes all of preivous Spatial-based ConvGNNs (Not really, a lot of extensions afterward)
- Same mechanism as in RecGNNs, but implemented as ***Spatial Graph Convolution*** layer

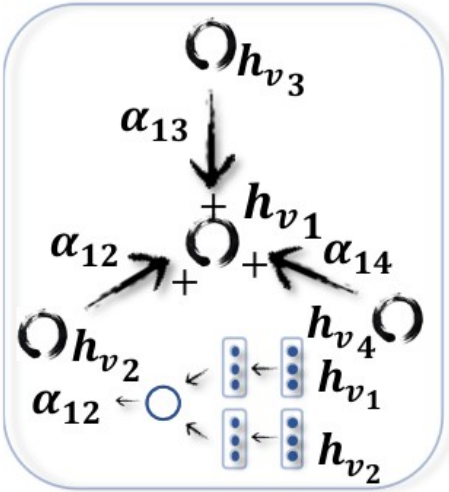
$$\vec{h}_v^{(k)} = U_k\left(\vec{h}_v^{(k-1)} + \sum_{u \in N(v)} M_k\left(\vec{h}_v^{(k-1)}, \vec{h}_u^{(k-1)}, \vec{x}_{uv}^{e}\right)\right)$$

$$\vec{h}_v^{(0)} = \vec{0}$$

- Particular implementations are then choosing the U_k and M_k
  - ***Graph Attention Network (GAT)***

$$\mathbf{h}_v^{(k)} = \sigma\Big(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}\Big)$$

$$\alpha_{vu}^{(k)} = softmax\big(g(\mathbf{a}^T[\mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)} || \mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)}])\big)$$

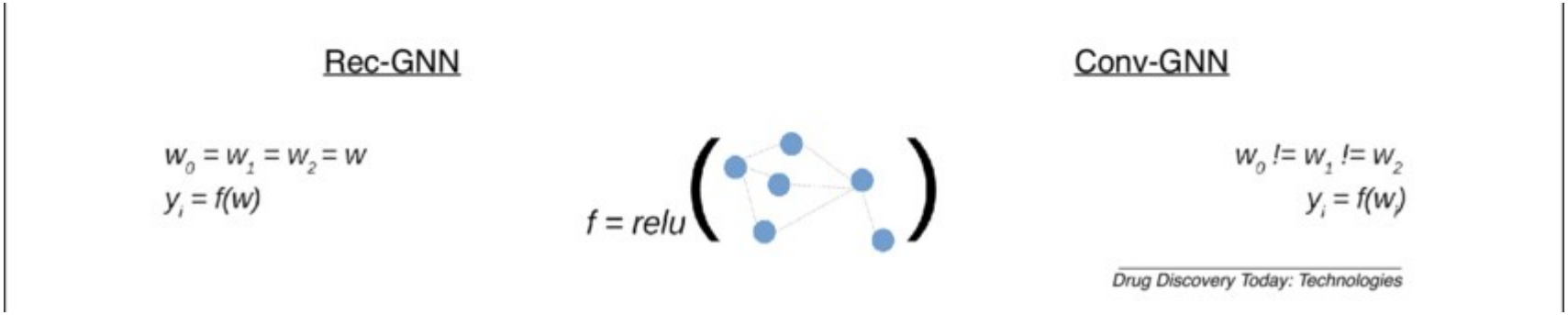


(b) GAT [43] implicitly captures the weight a_{ij} via an end-to-end neural network architecture, so that more important nodes receive larger weights.

GNNs: How to deal with non-Euclidean graphs?

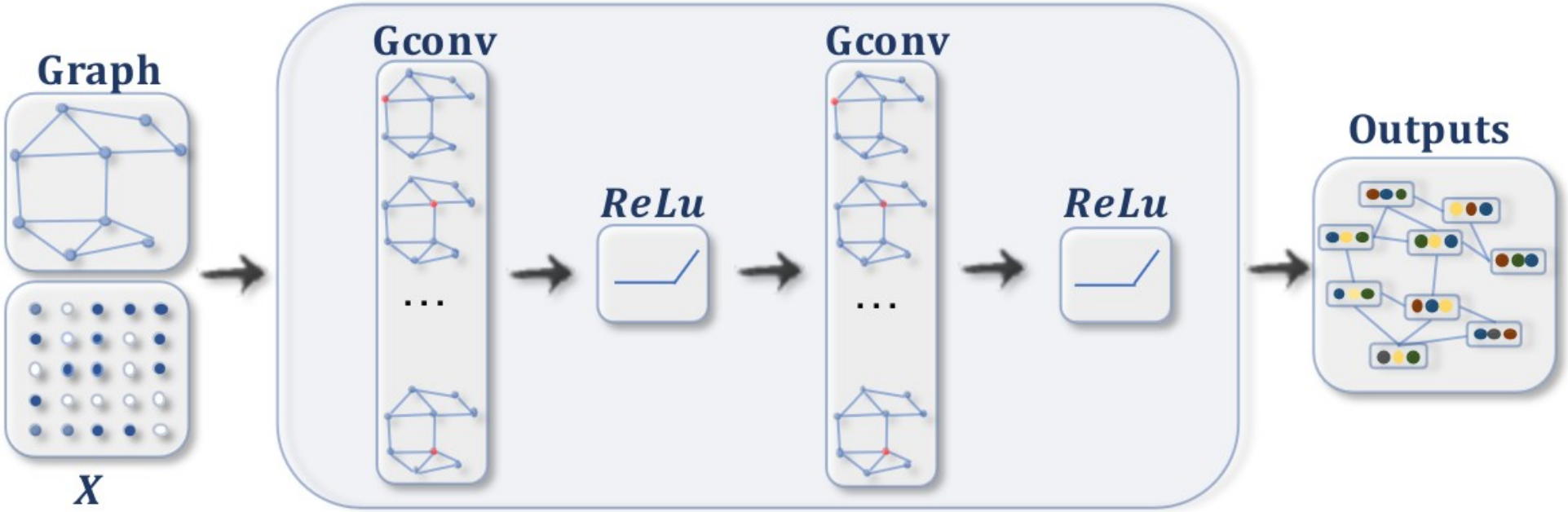
Convolution Graph Neural Network (ConvGNN), Spatial-based approach

But isn't the ***MPNN*** ConvGNN the same as Gated RecGNN?



- Rec-GNN apply the same set of weights until a criterion is met, whereas Conv-GNNs apply different weights at each iteration
- So, the index (k) means layers in ConvGNN, whereas in Gated RecGNN it is a time/iteration!

https://www.researchgate.net/figure/Difference-between-recurrent-Rec-GNN-and-convolution-Conv-GNN-based-graph-neural_fig2_347681450



(a) A ConvGNN with multiple graph convolutional layers. A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the resulted outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood.

GNNs: How to deal with non-Euclidean graphs?

Convolution Graph Neural Network (ConvGNN), Spatial-based approach

Message Passing Neural Network (MPNN)

- Generalizes all of the Spatial-based ConvGNNs
- 1) ***Readout*** function to convert multiple node features into single vector for non-graph output

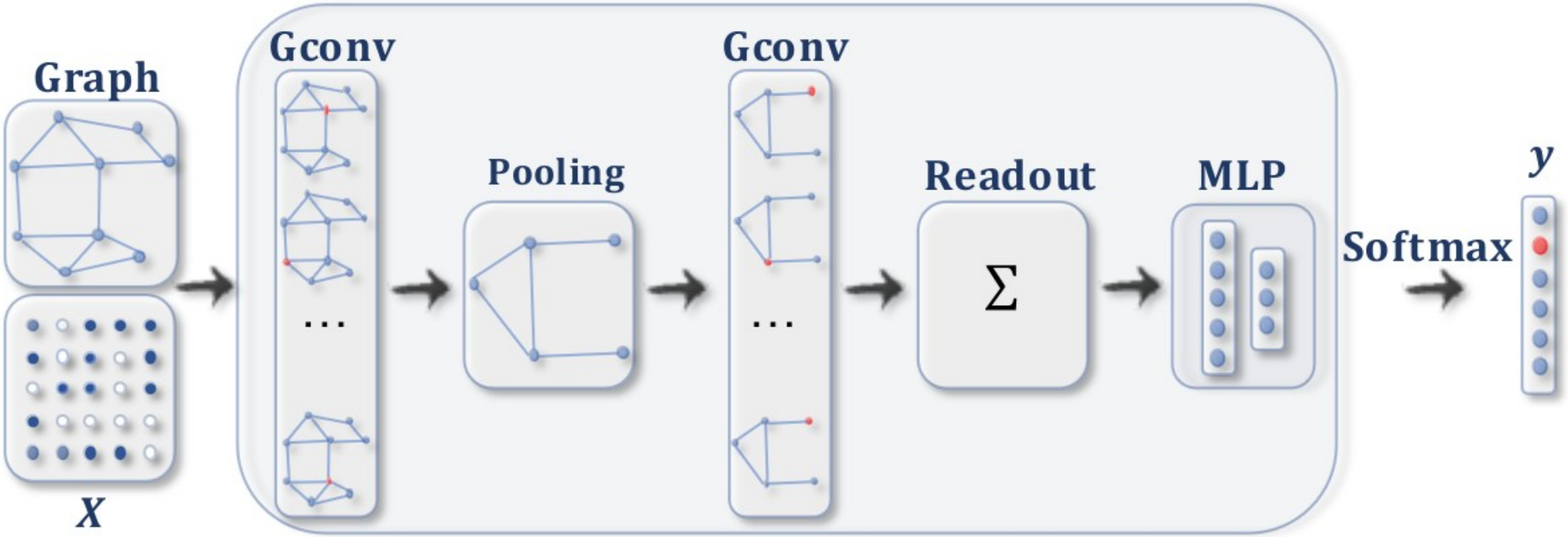
$$\vec{h}_G = R\left(\vec{h}_v^{(K)} \middle| v \in G\right)$$

- 2) **Pooling** function to reduce the number of nodes in a graph – convert to subgraph
  - Convert feature matrices by matrix S
    S function can either be a deterministic algorithm
    or a trainable layer

$$S = S(A, X)$$

$$H_{\text{pooled}} = S^T H$$

$$A_{\text{pooled}} = S^T A S$$



(b) A ConvGNN with pooling and readout layers for graph classification [21]. A graph convolutional layer is followed by a pooling layer to coarsen a graph into sub-graphs so that node representations on coarsened graphs represent higher graph-level representations. A readout layer summarizes the final graph representation by taking the sum/mean of hidden representations of sub-graphs.

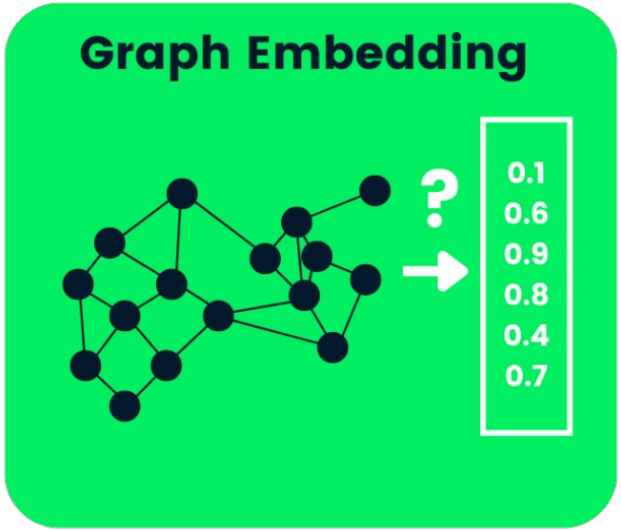
Graph Neural Networks (GNNs):

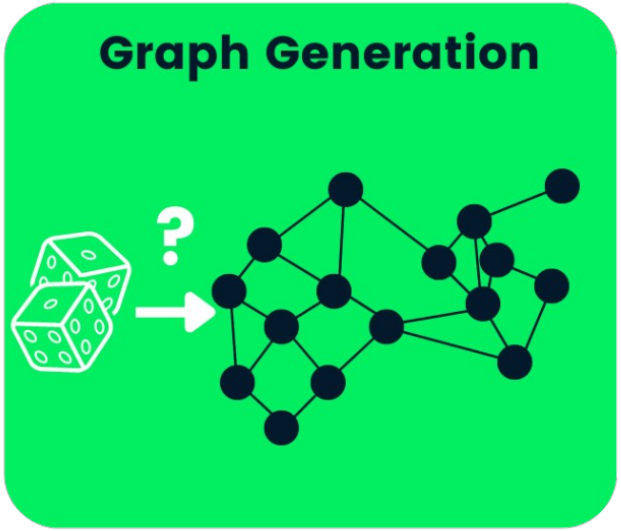
Graph Autoencoders

GNNs: How to deal with non-Euclidean graphs?

But what about the topology inference?

- Previous approaches were targeting the node/edge/vector feature extraction from a graph and used topology as a feature.
- But what about topology prediction/generation?
- Use latent space and AutoEncoders to do it!



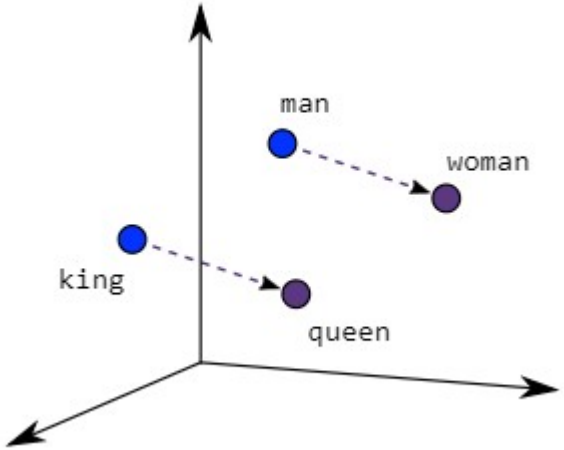


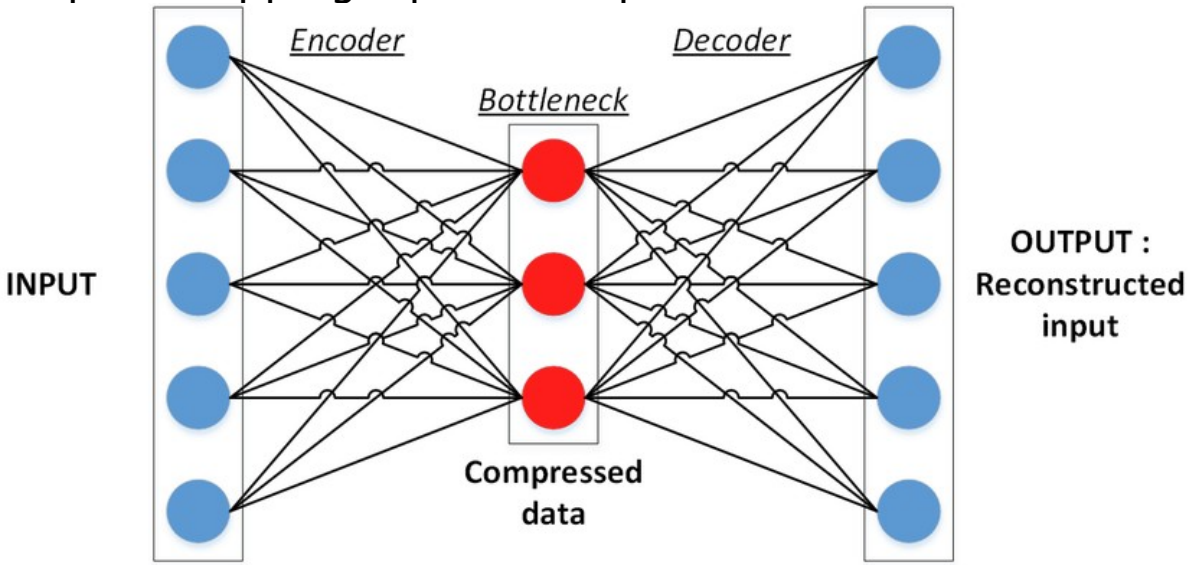
GNNs: How to deal with non-Euclidean graphs?

Standard AutoEncoders in Deep Learning

- When the task is given as a generative one – e.g. to operate with something between given training examples
  - To create meaningful sentences from words
  - To generate images that combine concepts from training samples

- Instead of trying to accomplish it with complex mapping input to output, we introduce:
  - The **latent space** of embeddings (remember vector embedding logic?)





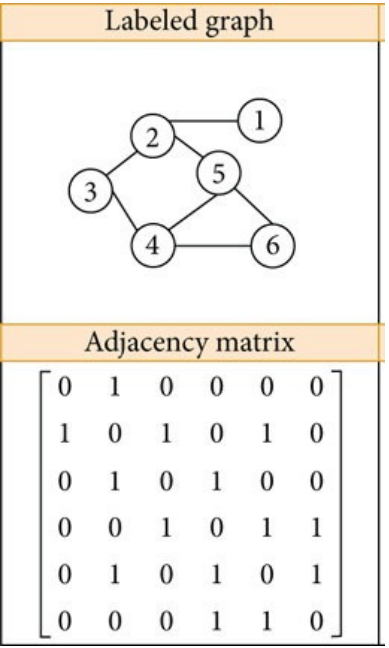
https://www.researchgate.net/figure/Basic-architecture-of-a-single-layer-autoencoder-made-of-an-encoder-going-from-the-input_fig3_333038461

https://medium.com/@krithiqkrish/from-words-to-vectors-decoding-the-intuition-behind-word-embedding-5a4f83fbf920

GNNs: How to deal with non-Euclidean graphs?

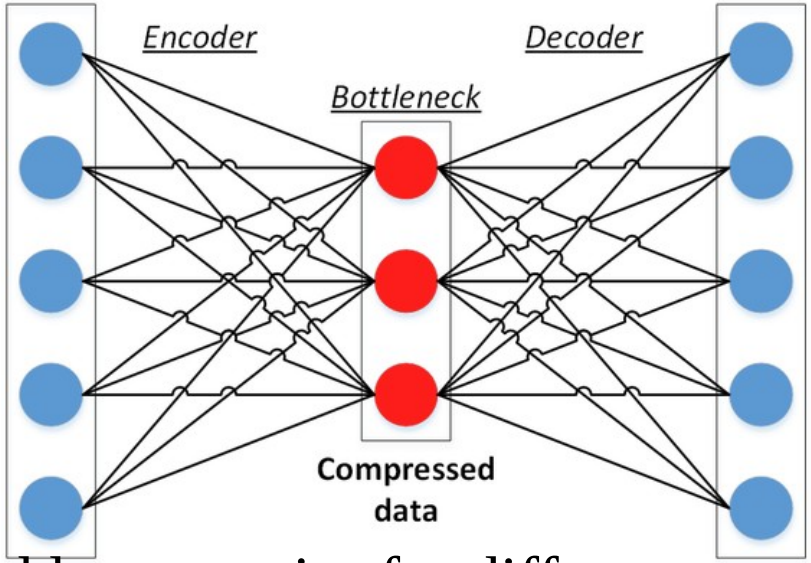
Apply AutoEncoders to Graph data

Deep Neural Network for Graph Representations (DNGR)



INPUT

$\vec{Z}_{\text{in}} \quad \vec{Z}_{\text{out}}$



OUTPUT : Reconstructed input

- Use simple FC linear layers
- Or maybe some Convolution

Add some noise for different output:

$$\vec{Z}_{\text{out}} \sim N\left(\vec{z}_{\text{in}}, I\right)$$

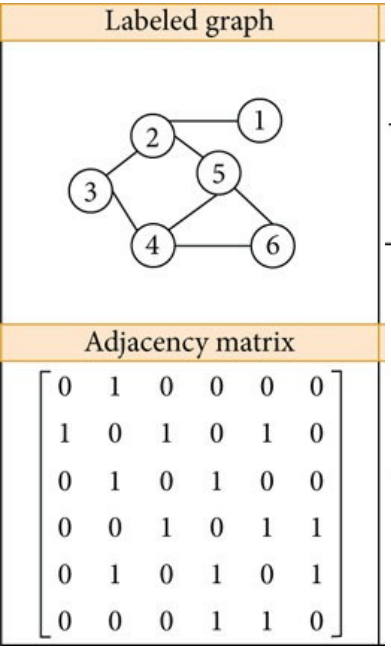
+ train to denoise

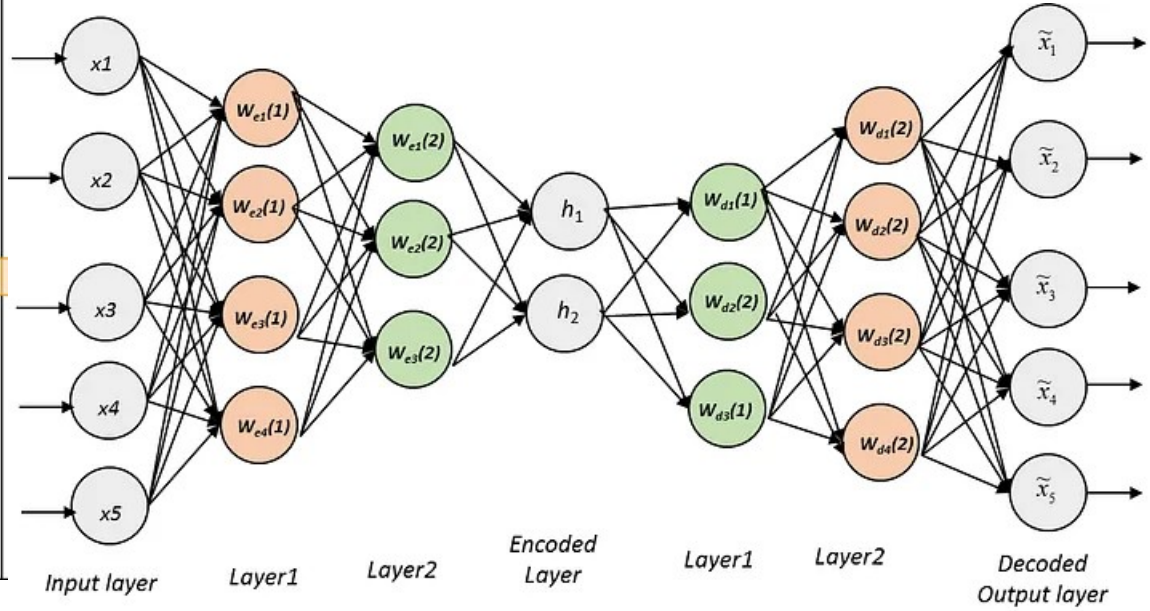
GNNs: How to deal with non-Euclidean graphs?

Apply AutoEncoders to Graph data

Structural Deep Network Embedding (SDNE)

- Stack multiple layers
- More complex latent space





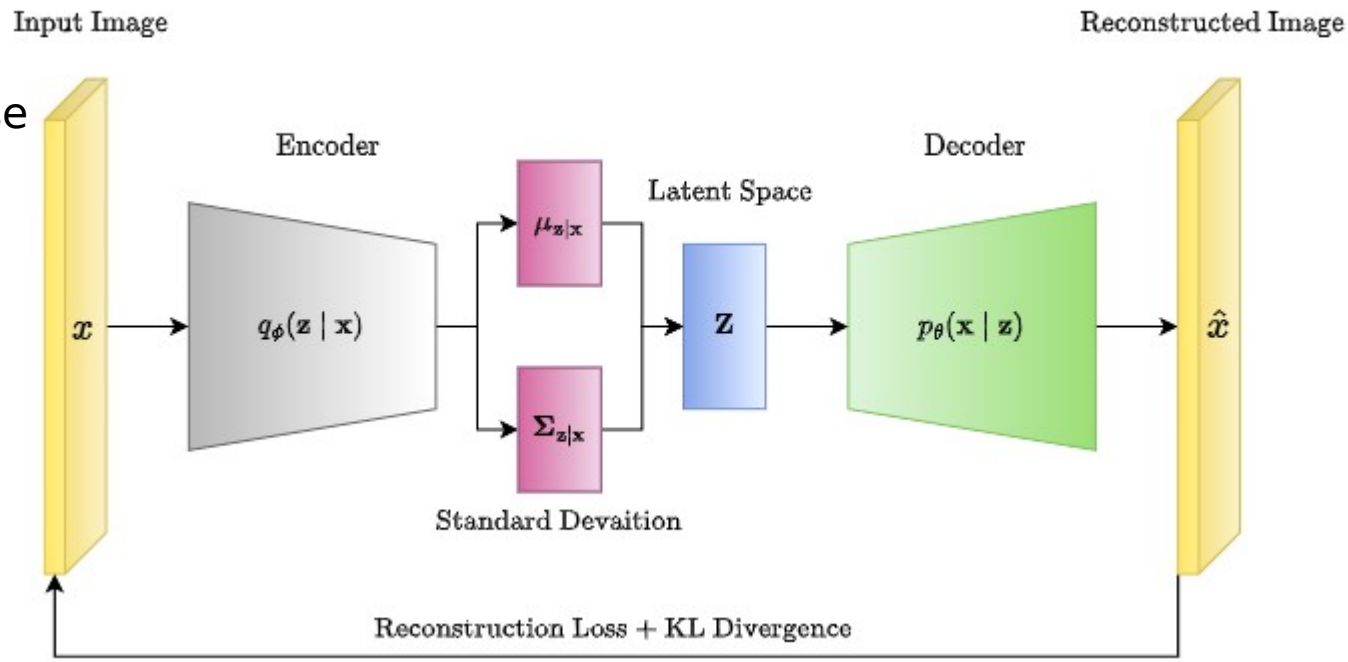
https://medium.com/@evertongomede/unveiling-the-power-of-stacked-autoencoders-675de2ce4273

GNNs: How to deal with non-Euclidean graphs?

Apply AutoEncoders to Graph data

Variational Autoencoders

- Generalize the AutoEncoders
- What optimal loss function to use
- What type of noise to introduce
- *Learn distribution too!*
  - *Not only the noise*
  - *Can generate new samples*



https://medium.com/@rushikesh.shende/autoencoders-variational-autoencoders-vae-and-%CE%B2-vae-ceba9998773d

GNNs: How to deal with non-Euclidean graphs?

Graph AutoEncoders (GAEs)

- But what if I want to combine features and topology?
- Let us start with the most successful feature extraction architecture, ConvGNNs
- Use AutoEncoders to it to combine features and topology

$$Z_{\text{in}} = encoder(X, A) = G_{\text{conv}}(f(G_{\text{conv}}(A, X; \Theta_1)); \Theta_2)$$

1 latent vector per node! But compute all at once with matrices

$$\tilde{A}_{u,v} = decoder(\vec{z}_u, \vec{z}_v) = \sigma(\vec{z}_u^{\,T} \vec{z}_v)$$

GNNs: How to deal with non-Euclidean graphs?

Graph Variational AutoEncoders (GVAs)

- But what if I want to combine features and topology?
- Let us start with the most successful feature extraction architecture, ConvGNNs
- Use Variational AutoEncoders to learn the distribution of features + topology
  - Introduce the loss that compute the distribution-wise error, not only sample-wise (Kullback-Leibler divergence)

$$L = E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - KL[q(\mathbf{Z}|\mathbf{X},\mathbf{A})||p(\mathbf{Z})]$$

$$P(Z) = \text{Gaussian prior} = \prod N(\vec{z}_i|0, I)$$

$$P(A|Z) = \text{Noise likelihood} = \prod p(A_{ij}|Z) = \sigma(\vec{z}_i^{\,T}\vec{z}_j)$$

$$q(Z|X, A) = \text{Empirical learned distribution} = \prod q(z_i|A, X) = N(z_i|mu_i, \sigma_i \cdot I)$$

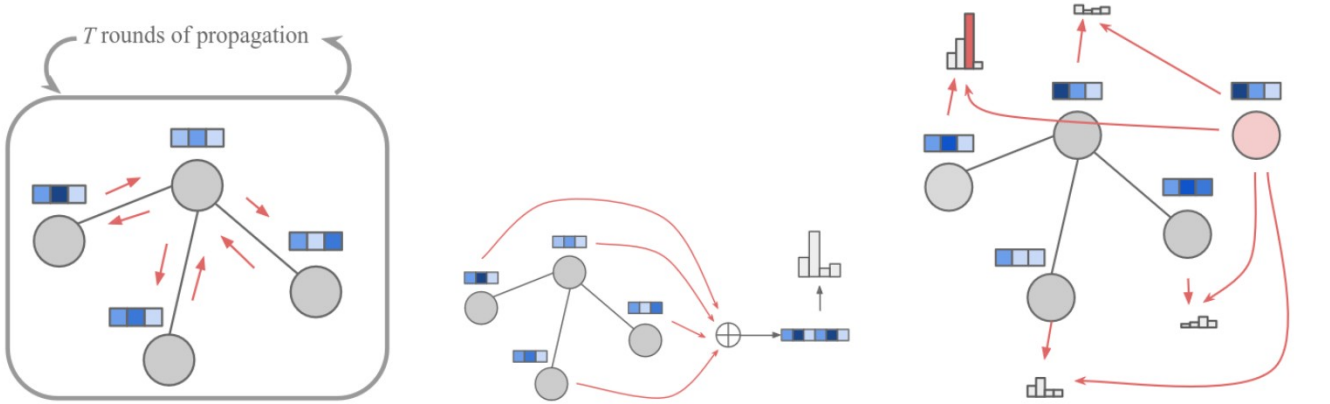
- Other usages: Robustness
  - **Adversarially Regularized Variational Graph Autoencoder (ARVGA)**
    Use generator to try to train to distinguish between fake and real samples

GNNs: How to deal with non-Euclidean graphs?

But what about different size graph generation?

Deep Generative Model of Graphs (DeepGMG)

- Generate a graph sequentially
- Start by one node and perform
- Use a RecGNN to do it
  - Classify as 0/1 output



- Alternatively:
  - **Variational GAE (GraphVAE)**

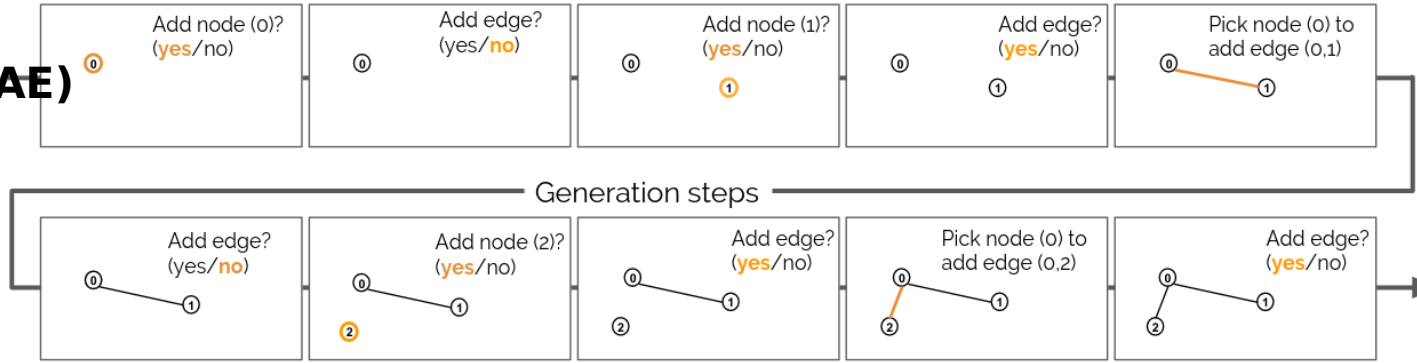


Figure 1. Depiction of the steps taken during the generation process.

https://arxiv.org/pdf/1803.03324.pdf

Applications: What GNNS are good for?

Applications: What GNNS are good for?

Molecular mapping in chemistry

- Given a set of pair {(pAct1, pAct2)} graphs, we define a known chemical reaction/transition from one molecule to another by a single change
- Such transformation improves some phys/chem properties, represented by some value (For example effectiveness on target disease or activity in certain environment)
- Task – given some other molecule pActX, generate an improved molecule
- MMPA – previous approach, use all know pairs to learn some general transformation rules
- GNN approach – see this problem as graph-to-graph mapping
  - Analogy from machine translation: Paraphrase of one sentence to a better one: ”buy sandwich cheese” -> ”I’d like to buy some sandwich with cheese”

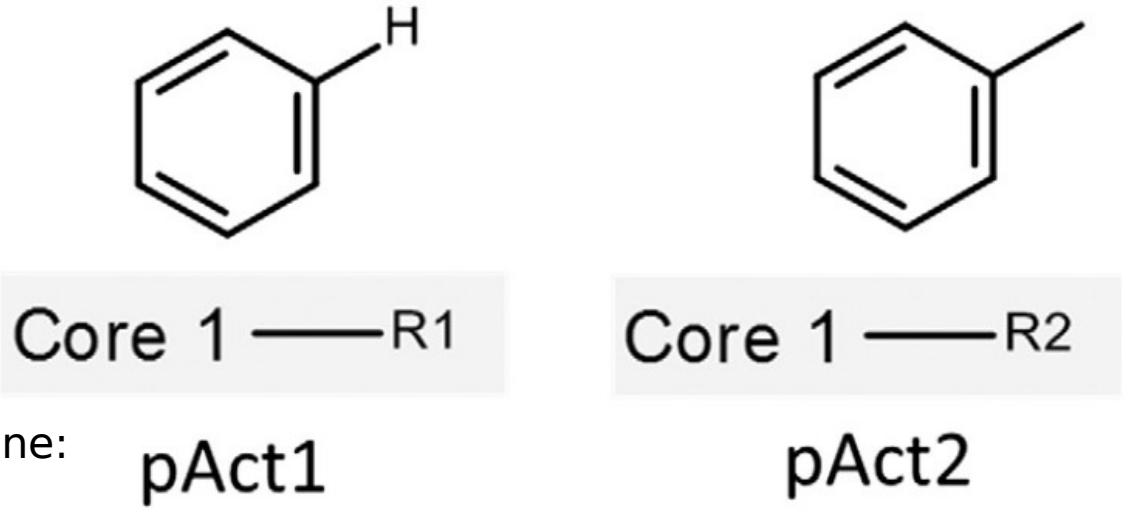
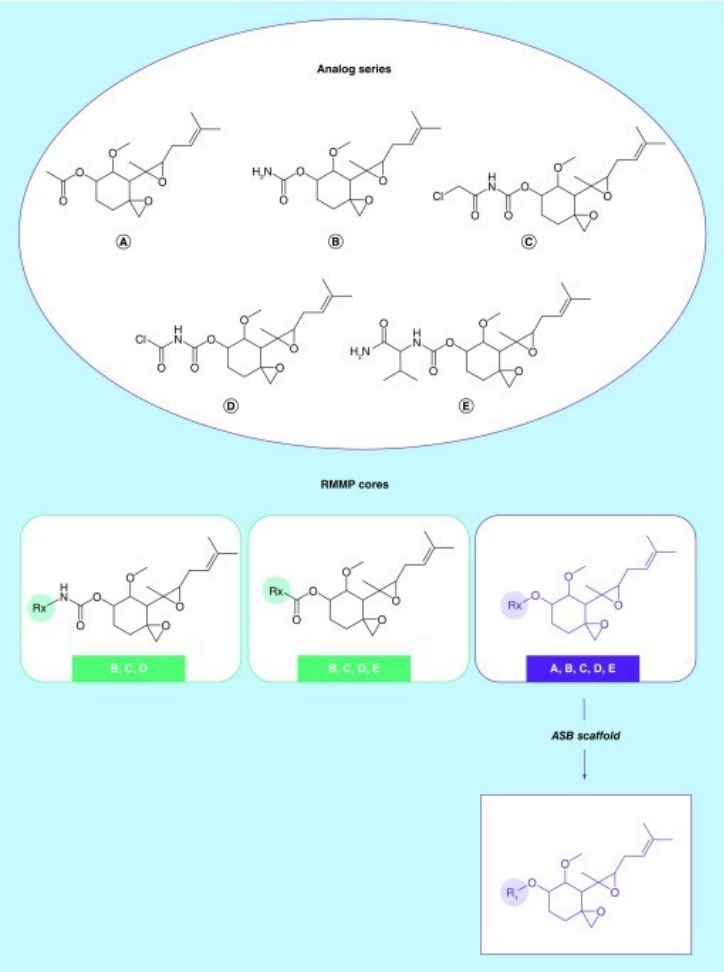


Fig. 1. Example of a matched molecular pair (MMP).

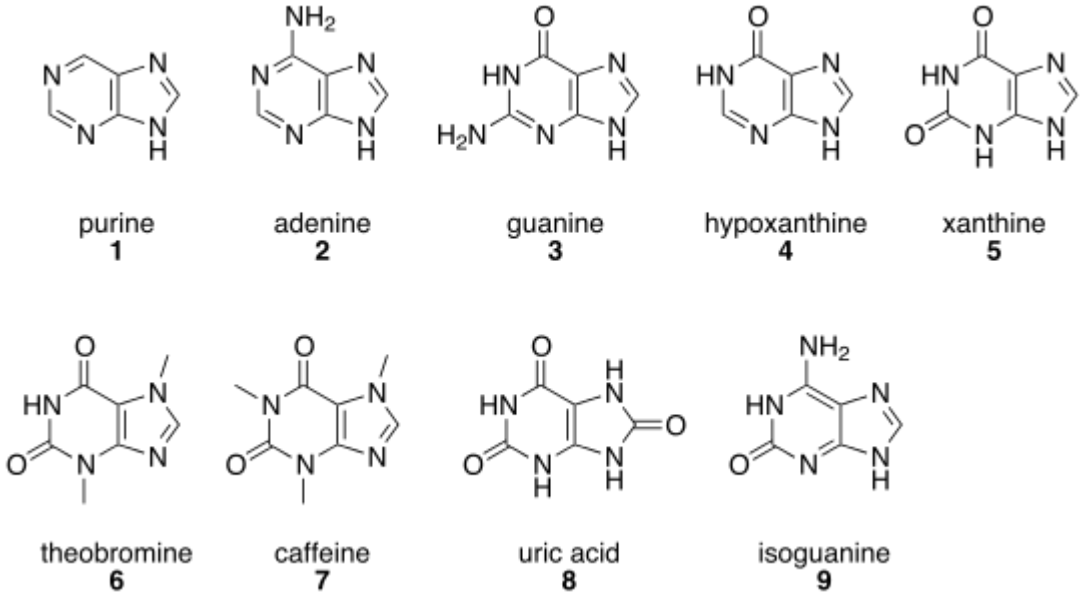
Applications: What GNNS are good for?

Molecular mapping in chemistry



- ***Scaffold*** – a family of molecules share the same core structure
- Intuitive example:

  A and G in DNA are from the same purine family



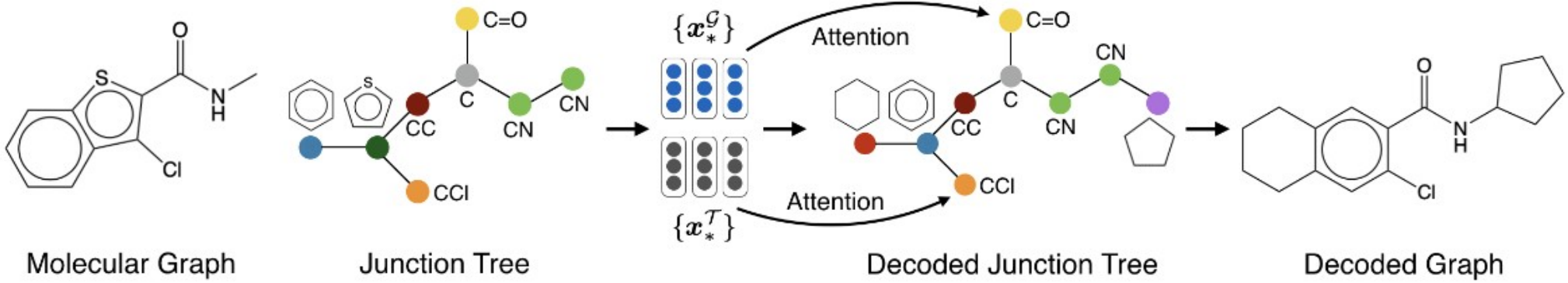
Applications: What GNNS are good for?

Molecular mapping in chemistry

2 Graphs input:

Atom-per-node graph

scaffold multi-atom-node graph



Compute from vocabulary of known scaffolds

atom = node v → $\vec{f}_v$ = atom type, valence, etc..

edge uv → $\vec{f}_{uv}$ = interaction features...

scaffold node w → $\vec{f}_w$ = one-hot id in vocabulary

- Run RecGNN on edge for T iterations only

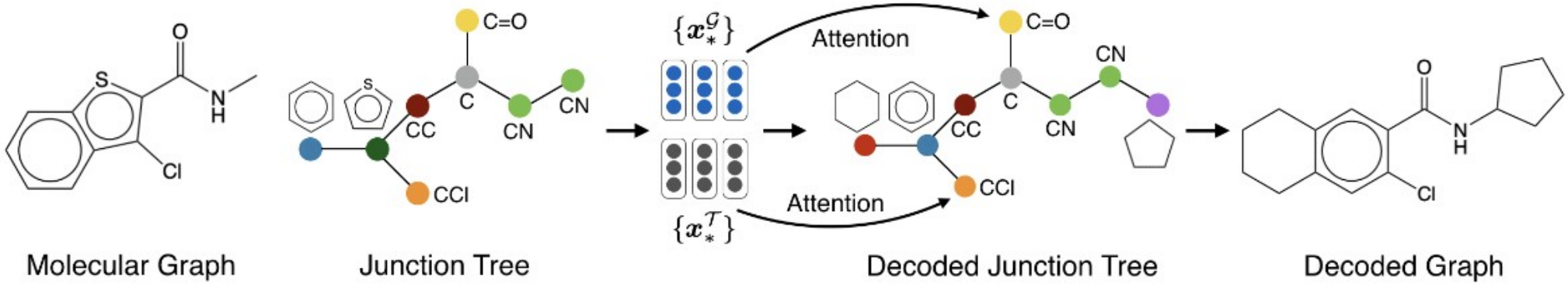
$$\vec{v}_{uv}^{(t)} = g_1\left(\vec{f}_u, \vec{f}_{uv}, \sum_{w \in N(u)/v} \vec{v}_{wu}^{(t-1)}\right) \qquad \vec{v}_{uv}^{(0)} = \vec{0}$$

- Run RecGNN with previous messages

$$\vec{x}_u = g_2\left(\vec{f}_u, \sum_{w \in N(u)} \vec{v}_{wu}^{(T)}\right) \qquad \vec{v}_{uv}^{(0)} = \vec{0}$$

Applications: What GNNS are good for?

Molecular mapping in chemistry



- Use Graph Attention Network and Gated RecGNN
  - Traverse the tree in DFS manner
  - For every visit do a binary classification by GRU
  - If 1 – split and expand node

- Use MPNN, Spatial-based ConvGNN
  - Use Decoded Junction Tree feature vectors
  - For every possible molecular graph compute its score by MPNN
  - Output the probabilities of all possible molecular graphs

Applications: What GNNS are good for?

Molecular mapping in chemistry

Method	QED			DRD2		
	Success	Diversity	Novelty	Success	Diversity	Novelty
MMPA	32.9%	0.236	99.9%	46.4%	**0.275**	99.9%
JT-VAE	8.8%	-	-	3.4%	-	-
GCPN	9.4%	0.216	100%	4.4%	0.152	100%
VSeq2Seq	58.5%	0.331	99.6%	75.9%	0.176	79.7%
VJTNN	59.9%	0.373	98.3%	77.8%	0.156	83.4%
VJTNN+GAN	**60.6%**	**0.376**	99.0%	**78.4%**	0.162	82.7%

Papers used:

[A Comprehensive Survey on Graph Neural Networks]
https://arxiv.org/pdf/1901.00596.pdf

[Geometric deep learning: going beyond Euclidean data]
https://arxiv.org/pdf/1611.08097.pdf

[Deep recurrent graph neural networks]
https://www.research.unipd.it/handle/11577/3366866

[Learning Multimodal Graph-to-Graph Translation for Molecular Optimization]
https://openreview.net/pdf?id=B1xJAsA5F7

[Understanding Pooling in Graph Neural Networks]
https://arxiv.org/pdf/2110.05292.pdf

Additional materials used:

https://www.youtube.com/watch?v=2KRAOZIULzw

https://dataroots.io/blog/a-gentle-introduction-to-geometric

https://cw.fel.cvut.cz/b221/courses/b4m33dzo/start