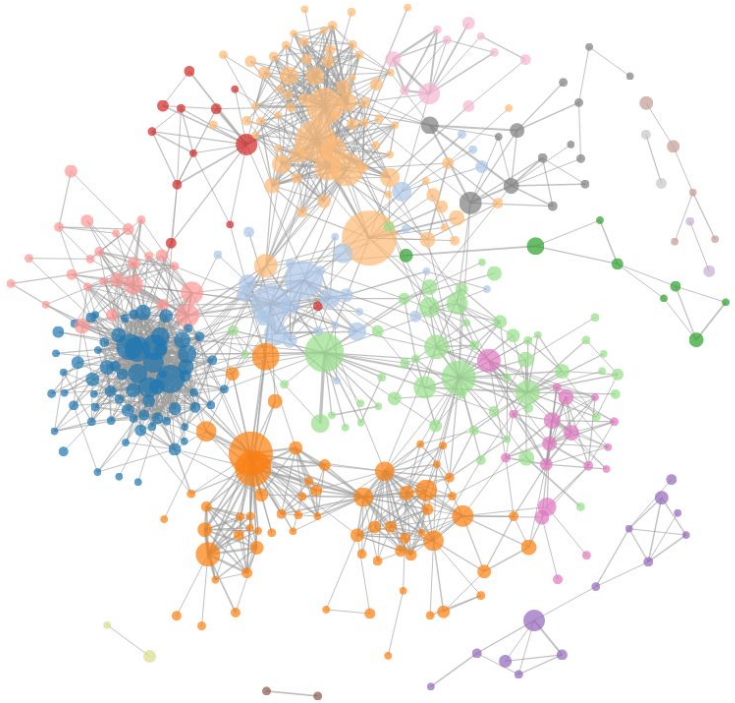
UofT Computer Vision Club



Paper Breakdown Series

Graph Neural Networks

youtube.com/@uoftcv

github.com/ut-cvdg/paper-breakdown

Follow us on social media

UNIVERSITY OF TORONTO

SUPPORTED BY STUDENT INITIATIVE FUND
The Division of Student Life

UofT CV is sponsored by GitHub

Sign up for the GitHub Student Developer Pack:
https://education.github.com/benefits

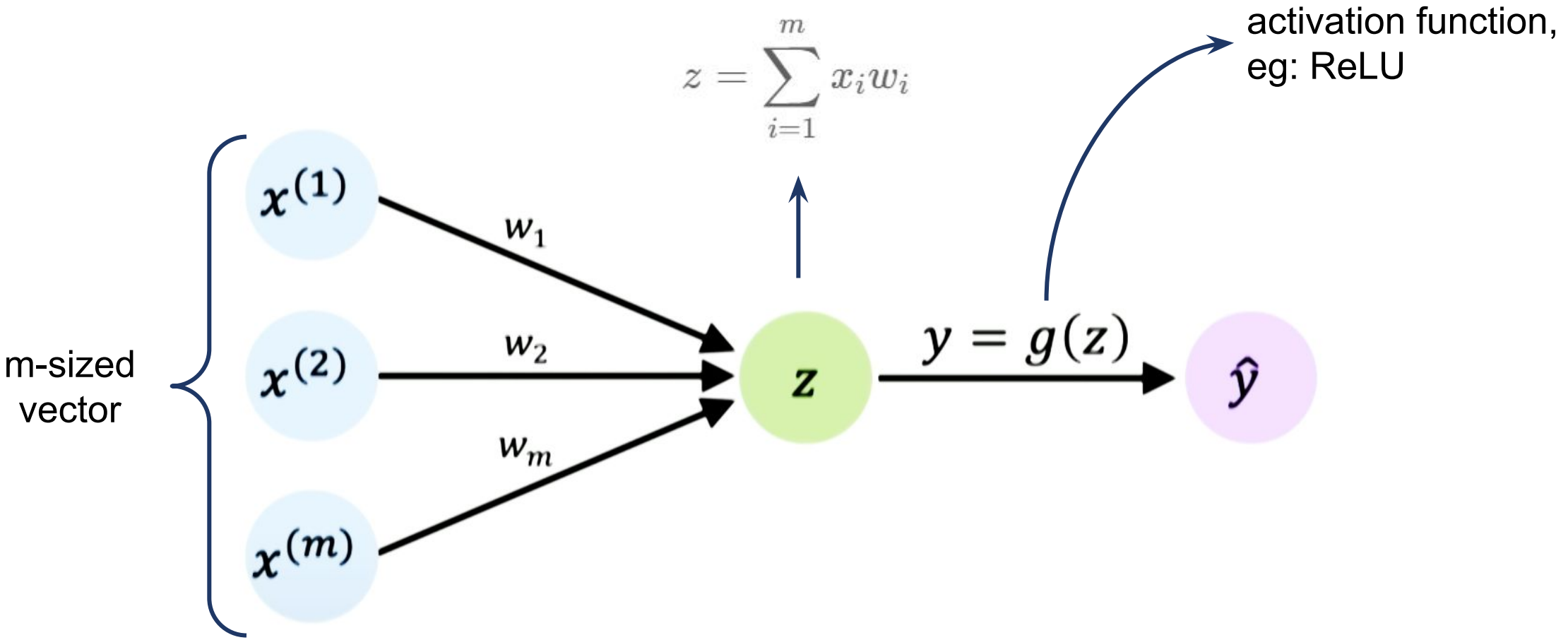
UNIVERSITY OF TORONTO | CVDG

About the Club

- We are a student club for Computer Vision and Machine Learning affiliated with the University of Toronto.
- We run:
  - paper “discussions” for paper in the past 1 week
  - conference-style research orals for pre-prints
  - And, paper breakdown series

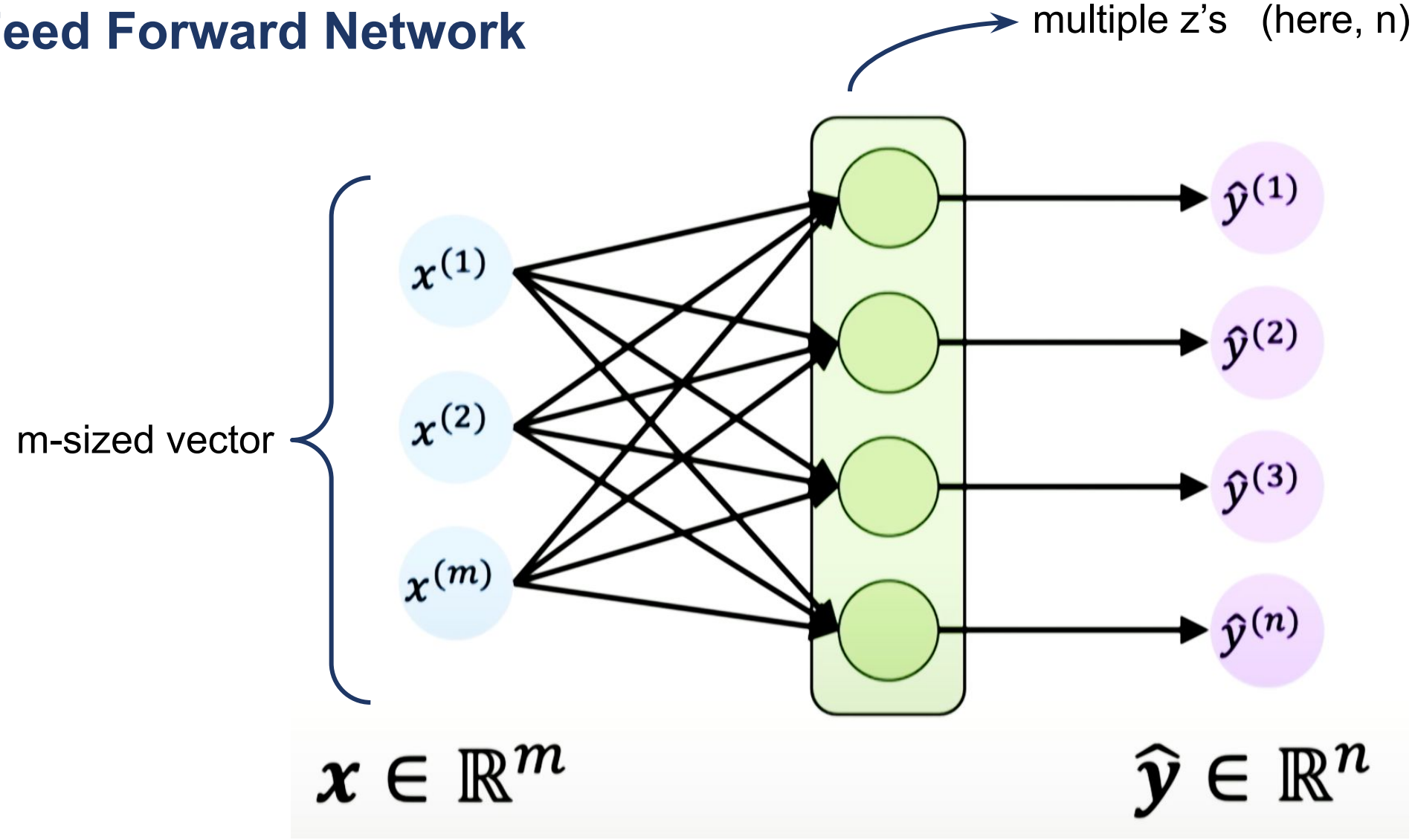
If you want to help run our club, talk to any of us or drop an email contact@cvdg.tech.

Perceptron



Source: MIT 6.S191

Feed Forward Network

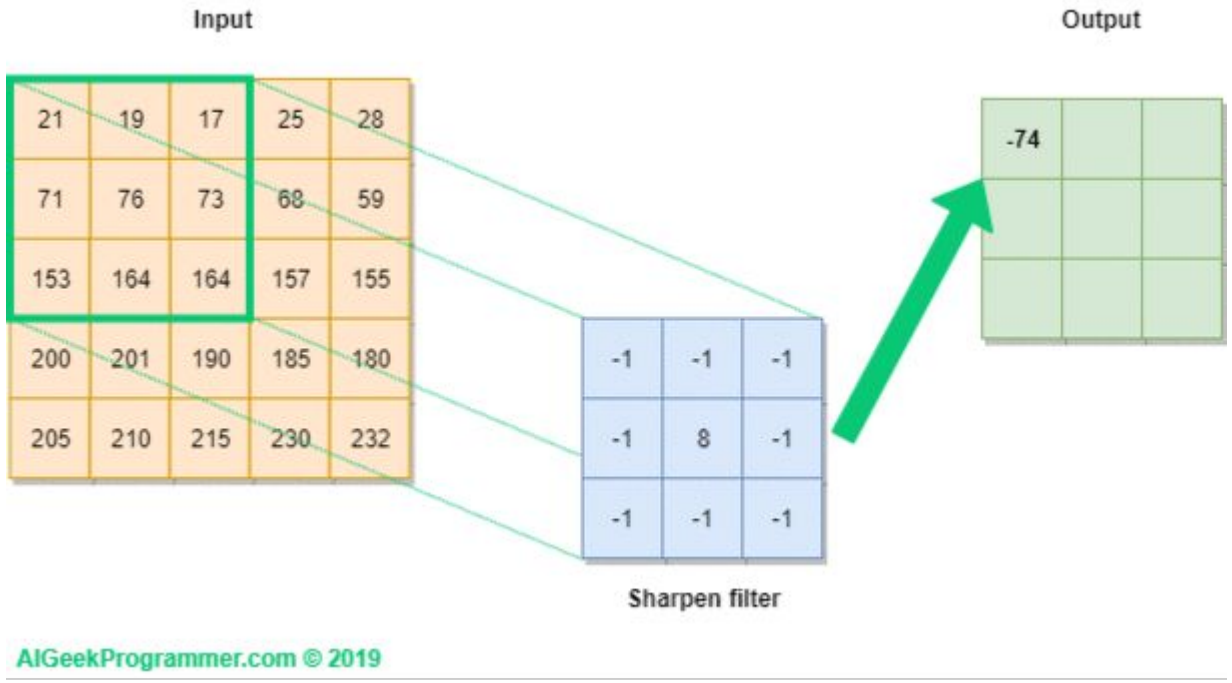


Source: MIT 6.S191

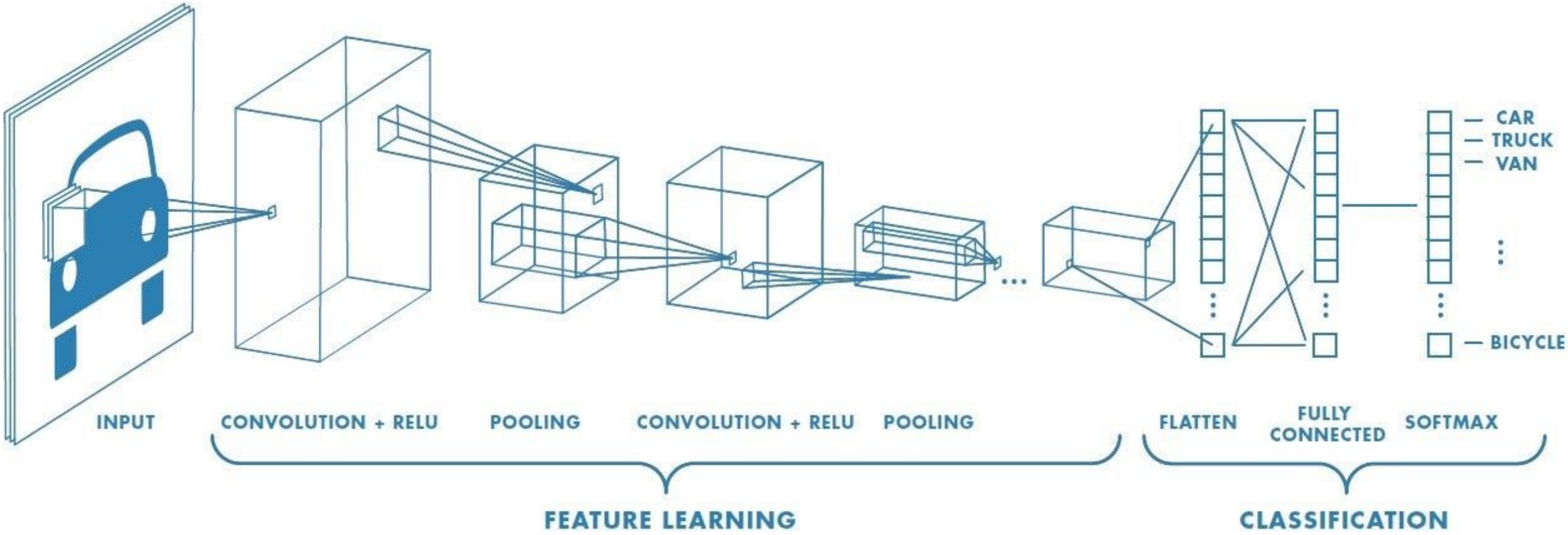
CNN - Convolutional Neural Network

What does it mean to "apply" a filter to some input:

1. Element-wise multiplication
2. Summation



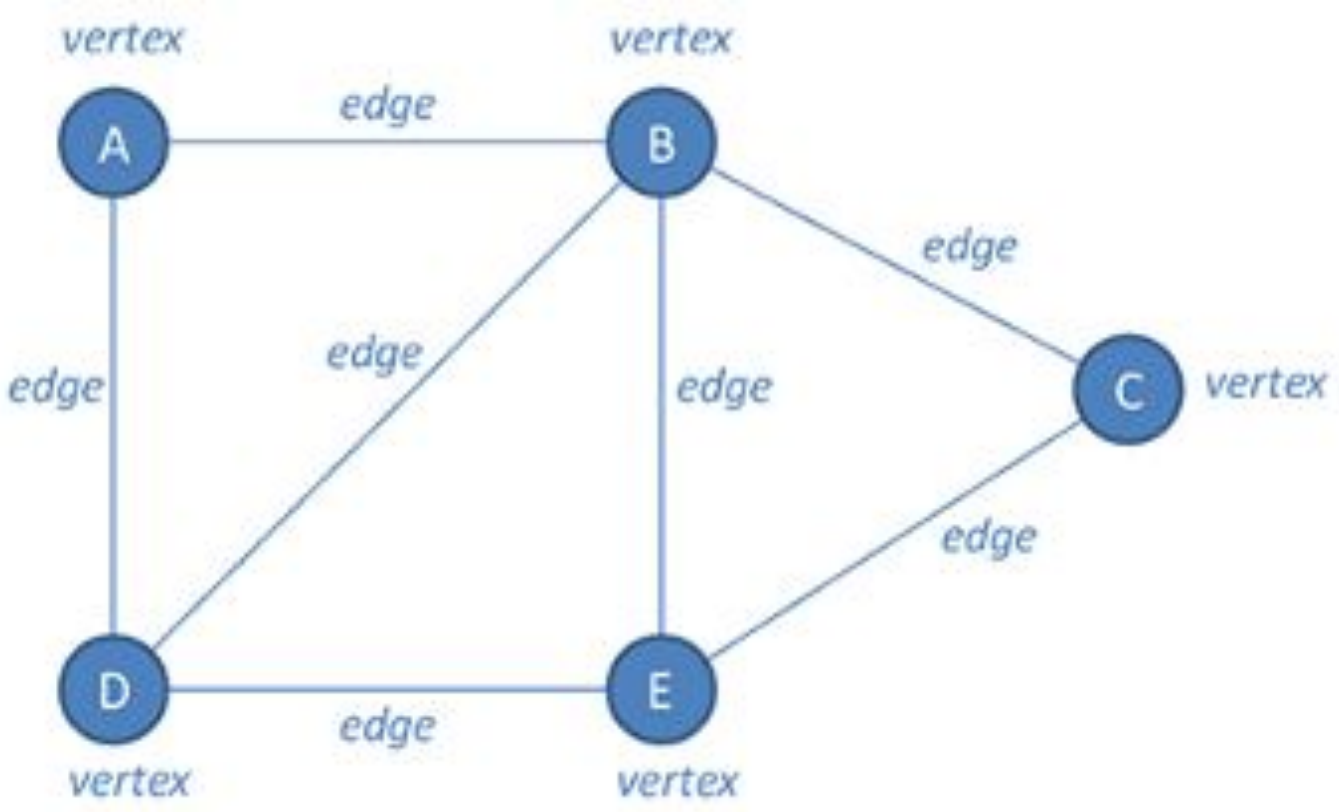
CNN



Source: MIT 6.S191

Basics of Graphs

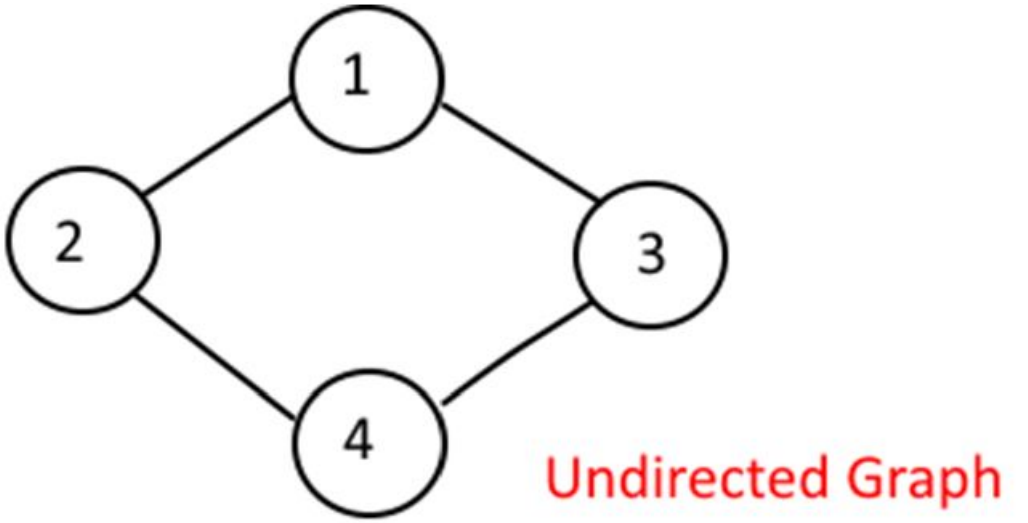
Node is also called vertex.
Edge is also called link.



Source: Nebula graph

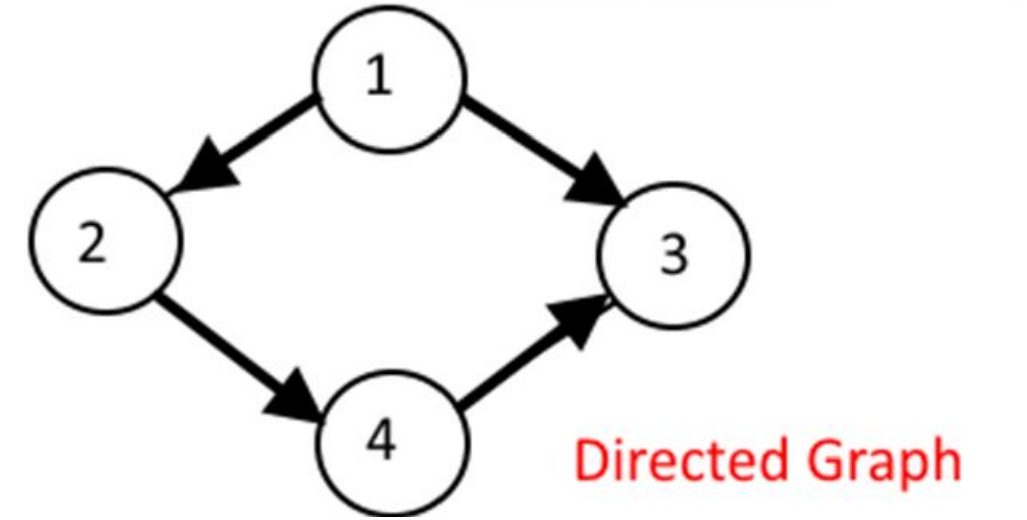
Undirected graph:

We can traverse both ways between node 1 and node 2.

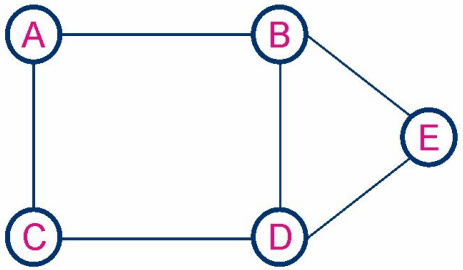


Directed graph:

We can go from node 1 to node 2, but not from node 2 to node 1.



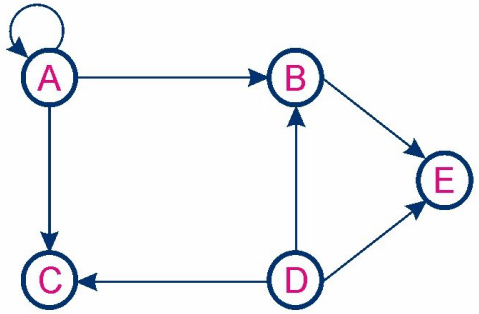
Source: Research gate



Undirected Graph

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

Each element of the adjacency matrix indicates whether there is an edge between 2 nodes or not.
1 means edge is present, and 0 means absent.



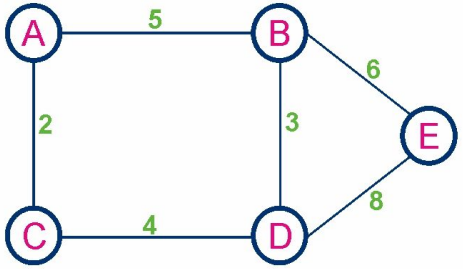
Directed Graph

Ending position

Starting Position

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

Since we can go from A to B, so the element in the purple box is 1.
However, we cannot go from B to A, so the element in orange box is 0.



Weighted Graph

	A	B	C	D	E
A	0	5	2	0	0
B	5	0	0	3	6
C	2	0	0	4	0
D	0	3	4	0	8
E	0	6	0	8	0

This is similar to the adjacency matrix of an undirected graph. The only addition is that if an edge is present then instead of 1, we put the weight of that edge.

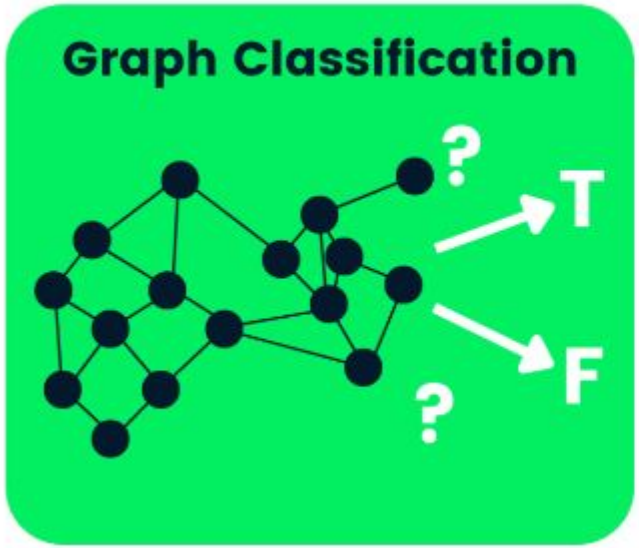
UNIVERSITY OF TORONTO | CVDG

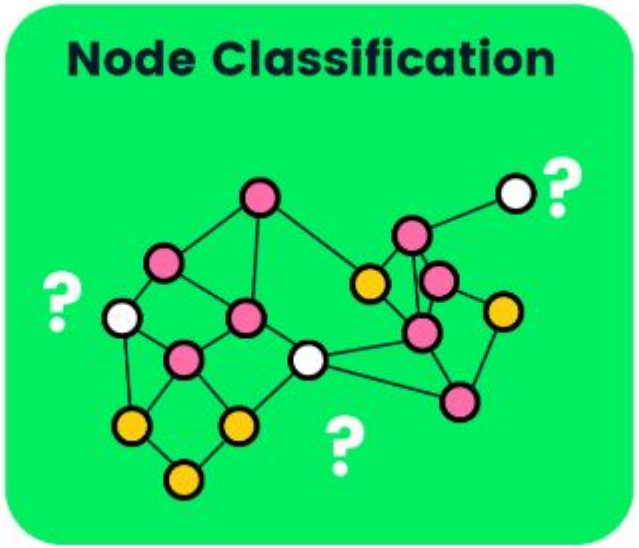
Adjacency Matrices

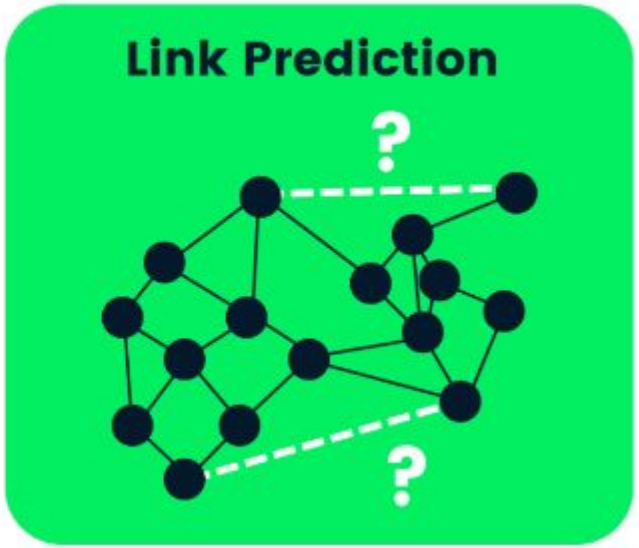
Source: Study Glance

Applications Areas of GNNs

- **Graph classification:** classify graphs into various categories. Eg: social network analysis.
- **Node classification:** use neighboring node labels to predict missing node labels in a graph.
- **Link prediction:** predict the link between a pair of nodes in a graph with an incomplete adjacency matrix.



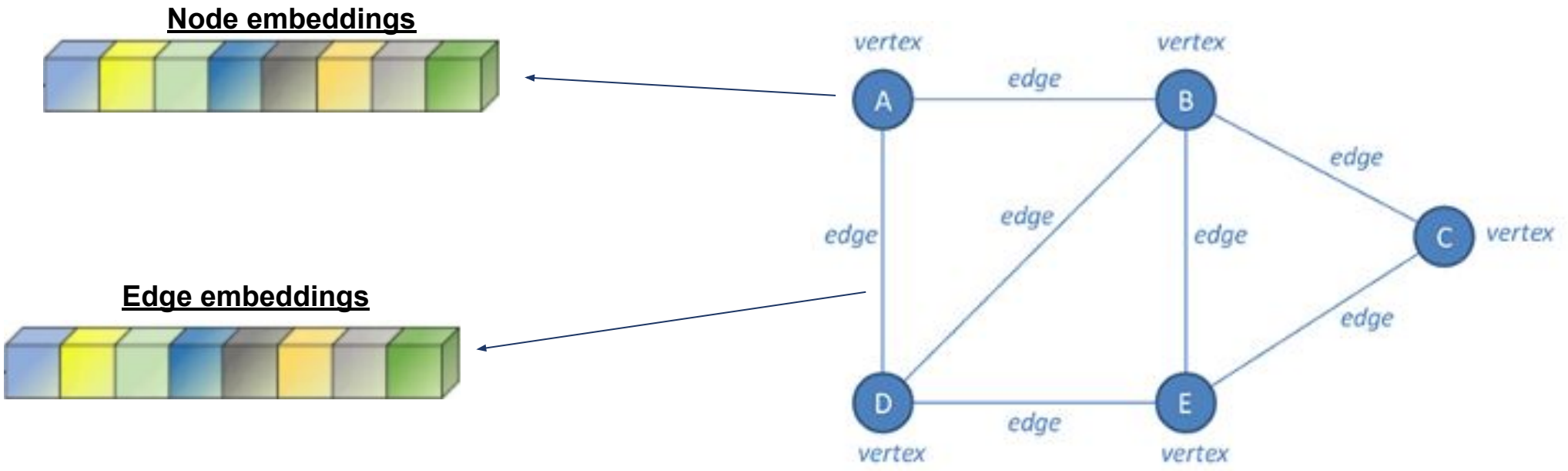




Source: DataCamp

Node & Edge Embeddings

The nodes and edges both have their own features, called node features (or node embeddings) and edge features (or edge embeddings), respectively.



Node & Edge Embeddings (contd.)

But why do we need node and edge embeddings?

Node & Edge Embeddings (contd.)

But why do we need node and edge embeddings?

So that we can better represent the information contained in them.

Node & Edge Embeddings (contd.)

But why do we need node and edge embeddings?

So that we can better represent the information contained in them.

Q. How do we preprocess words before feeding them to neural network?

Node & Edge Embeddings (contd.)

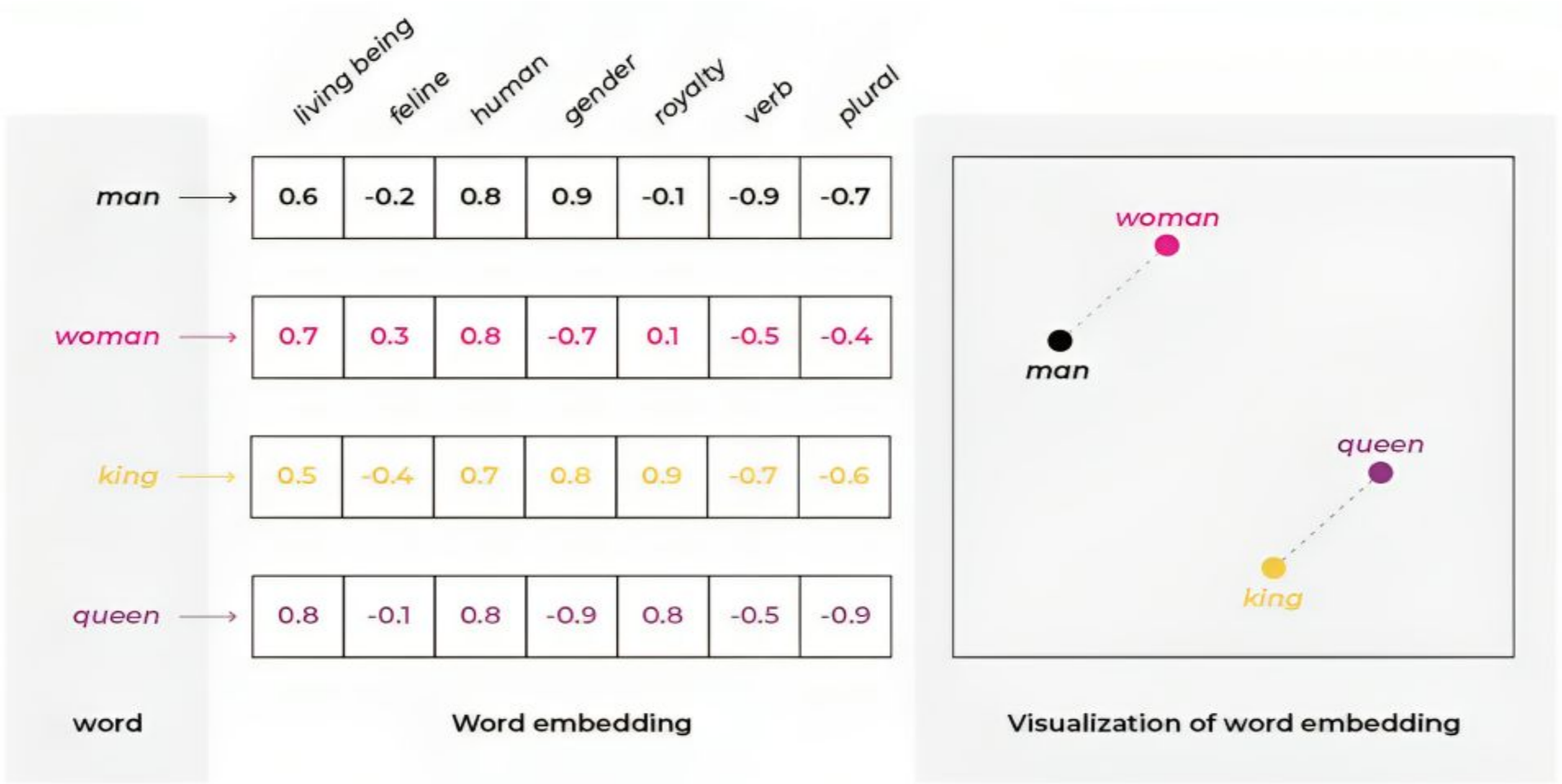
But why do we need node and edge embeddings?

So that we can better represent the information contained in them.

Q. How do we preprocess words before feeding them to neural network?

A. Word embeddings.

Word Embeddings



[Source: Francisco Castillo]

Graph linear layer

Recall the equation of a linear layer:

$$Y = XW^T$$

, where:

- Y = output matrix
- X = input matrix (or node features)
- W = weights matrix

Graph linear layer

Recall the equation of a linear layer:

$$Y = XW^T$$

, where:

- Y = output matrix
- X = input matrix (or node features)
- W = weights matrix

Since we are dealing with graphs, we will need a way to incorporate information about how each node is connected.

Graph linear layer (contd.)

Thus, we will make use of adjacency matrices. WHY?

Graph linear layer (contd.)

Thus, we will make use of adjacency matrices. WHY?

Because:

- An adjacency matrix contains the connections between every node in the graph, and by multiplying the input matrix by this adjacency matrix, we will directly sum up the neighboring node features.

Graph linear layer (contd.)

Thus, we will make use of adjacency matrices. WHY?

Because:

- An adjacency matrix contains the connections between every node in the graph, and by multiplying the input matrix by this adjacency matrix, we will directly sum up the neighboring node features.
- We can add self loops to the adjacency matrix so that the central node (or the node in consideration) is also considered in this operation.

Graph linear layer (contd.)

Thus, we will make use of adjacency matrices. WHY?

Because:

- An adjacency matrix contains the connections between every node in the graph, and by multiplying the input matrix by this adjacency matrix, we will directly sum up the neighboring node features.
- We can add self loops to the adjacency matrix so that the central node (or the node in consideration) is also considered in this operation.

Updated adjacency matrix = $\tilde{A} = A + I$

, where:

- A = adjacency matrix
- I = identity matrix

Graph linear layer (contd.)

Initially, the equation of our linear layer was: $Y = XW^T$

Graph linear layer (contd.)

Initially, the equation of our linear layer was: $Y = XW^T$

Now, we can modify it and write the equation of our graph linear layer as: $Y = \tilde{A}^T X W^T$

, where:

- Y = output of the graph linear layer
- A = updated adjacency matrix (includes self-loops)
- X = input matrix (or node features)
- W = weights matrix

Message Passing Neural Network (MPNN) – Generalizing the GNN

The idea behind the graph linear layer was to take the linear combination of features from the neighboring nodes (including the target node) with a weight matrix. Then take the sum of this linear combination.

Message Passing Neural Network (MPNN) – Generalizing the GNN

The idea behind the graph linear layer was to take the linear combination of features from the neighboring nodes (including the target node) with a weight matrix. Then take the sum of this linear combination.

We can generalize the above using a message passing neural network (MPNN) framework.

Message Passing Neural Network (MPNN) – Generalizing the GNN

The idea behind the graph linear layer was to take the linear combination of features from the neighboring nodes (including the target node) with a weight matrix. Then take the sum of this linear combination.

We can generalize the above using a message passing neural network (MPNN) framework.

MPNN consists of 3 main operations

- **Message** – a node creates a message for its neighbors
- **Aggregate** – each node aggregates the messages from its neighboring nodes
- **Update** – each node's features are updated using the aggregated messages

MPNN (contd.)

- **Message**: Every node uses a function to create a message for each of its neighbors. The message can simply consist of its own features or also consider the neighboring node's features and the edge features.

MPNN (contd.)

- **Message**: Every node uses a function to create a message for each of its neighbors. The message can simply consist of its own features or also consider the neighboring node's features and the edge features.
- **Aggregate:** Every node uses a permutation-equivariant function (eg: concatenation) to aggregate the messages from its neighboring nodes. A function is said to be permutation-equivariant if the output changes in the same way as the input when the order of the input is permuted. Eg: f(x,y) = xy; f(y,x) = yx

MPNN (contd.)

- **Message**: Every node uses a function to create a message for each of its neighbors. The message can simply consist of its own features or also consider the neighboring node's features and the edge features.
- **Aggregate:** Every node uses a permutation-equivariant function (eg: concatenation) to aggregate the messages from its neighboring nodes. A function is said to be permutation-equivariant if the output changes in the same way as the input when the order of the input is permuted. Eg: f(x,y) = xy; f(y,x) = yx
- **Update:** Every node uses a function to update its features by combining the current features and the aggregated messages. For example, a self-loop can be used to update a node's features by combining the node's current features and the aggregated messages.

MPNN (contd.)

- **Message**: Every node uses a function to create a message for each of its neighbors. The message can simply consist of its own features or also consider the neighboring node's features and the edge features.
- **Aggregate:** Every node uses a permutation-equivariant function (eg: concatenation) to aggregate the messages from its neighboring nodes. A function is said to be permutation-equivariant if the output changes in the same way as the input when the order of the input is permuted. Eg: f(x,y) = xy; f(y,x) = yx
- **Update:** Every node uses a function to update its features by combining the current features and the aggregated messages. For example, a self-loop can be used to update a node's features by combining the node's current features and the aggregated messages.

Let's discuss this more mathematically…

MPNN (contd.)

Let:

- h_i be the node embedding of node i
- h_j be the node embedding of node j
- $e_{j,i}$ be the edge embedding of the edge $j \rightarrow i$
- h_i' be the updated node embedding of node i

MPNN (contd.)

Let:

- h_i be the node embedding of node i
- h_j be the node embedding of node j
- $e_{j,i}$ be the edge embedding of the edge $j \rightarrow i$
- h_i' be the updated node embedding of node i

Message: Suppose node *i* creates a message using its own node features, the node features of node *j*, and the edge features of the edge between nodes *i* and *j* using a message function ϕ:

$$\phi\left(h_i,\ h_j,\ e_{j,i}\right)$$

MPNN (contd.)

Let:

- h_i be the node embedding of node i
- h_j be the node embedding of node j
- $e_{j,i}$ be the edge embedding of the edge $j \rightarrow i$
- h_i' be the updated node embedding of node i

Message: Suppose node *i* creates a message using its own node features, the node features of node *j*, and the edge features of the edge between nodes *i* and *j* using a message function ϕ:

$$\phi\left(h_i,\ h_j,\ e_{j,i}\right)$$

This was just one message. Node *i* creates one message for each of its neighboring nodes. Let's denote the set of node *i* ' s neighbors by $\mathcal{N}_i$.

MPNN (contd.)

Aggregate: Now suppose that node *i* created and received one message from each of its neighboring nodes. So, node *i* aggregates all the messages it received from its neighbors along with its own features using an aggregate function $\oplus$:

$$\bigoplus_{j \in \mathcal{N}_i} \left(\phi(h_i, h_j, e_{j,i}) \right)$$

MPNN (contd.)

Aggregate: Now suppose that node *i* created and received one message from each of its neighboring nodes. So, node *i* aggregates all the messages it received from its neighbors along with its own features using an aggregate function $\oplus$:

$$\bigoplus_{j \in \mathcal{N}_i} \left(\ \phi(h_i, h_j, e_{j,i}) \ \right)$$

Update: Now node *i* updates its features by combining its current features and the aggregated messages using an update function γ as follows:

$$h'_i \ = \ \gamma(h_i, \bigoplus_{j \in \mathcal{N}_i} \left(\ \phi(h_i, h_j, e_{j,i}) \ \right))$$

MPNN (contd.)

The MPNN equation:

$$h_i' = \gamma(h_i, \bigoplus_{j \in \mathcal{N}_i}(\ \phi(h_i, h_j, e_{j,i}) \ ))$$

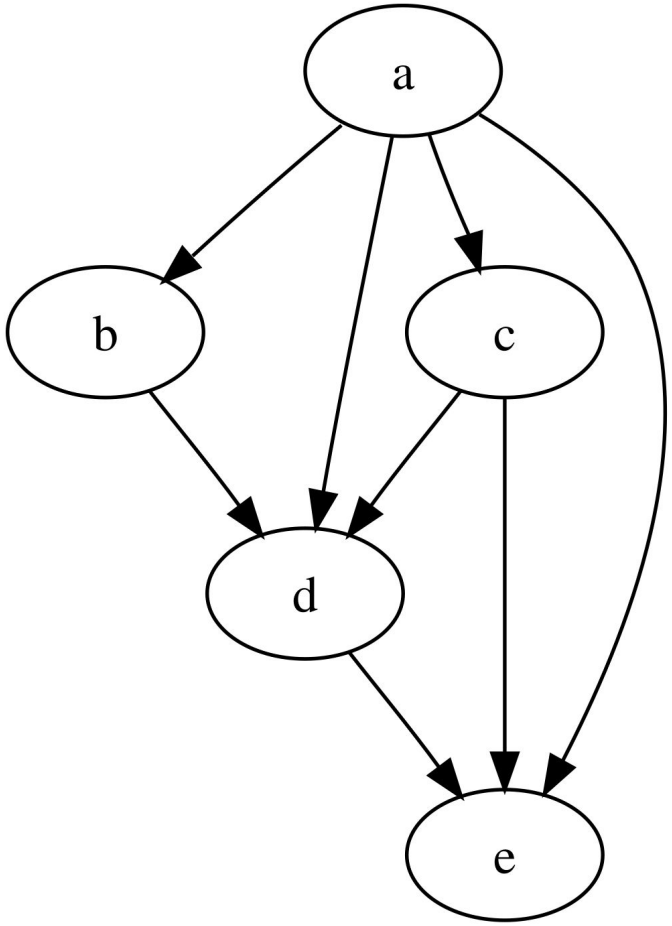
, where:

- h_i = node embedding of node i
- h_j = node embedding of node j
- $e_{j,i}$ = edge embedding of the link/edge $j \rightarrow i$
- ϕ = message function
- $\mathcal{N}_i$ = neighbors of node i
- $\bigoplus$ = aggregate function
- γ = update function
- h_i' = updated node embedding of node i

Problem Statement

Citation Network

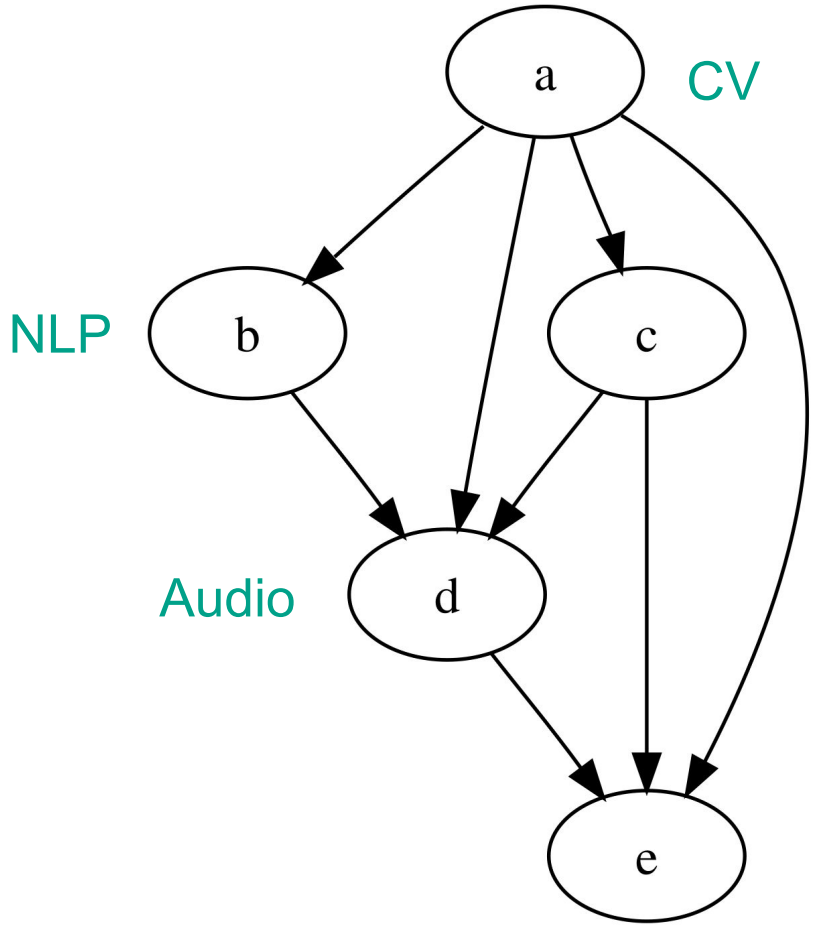
- Directed Graph
- Node = document
- Edge(a, b) = a cites b
- Application:
  - Search tools
  - Legal Judgements
  - Citation Analysis



In this example, document b cites document d, and is cited by document a.

Source: Wikipedia

Citation Network - Node Classification



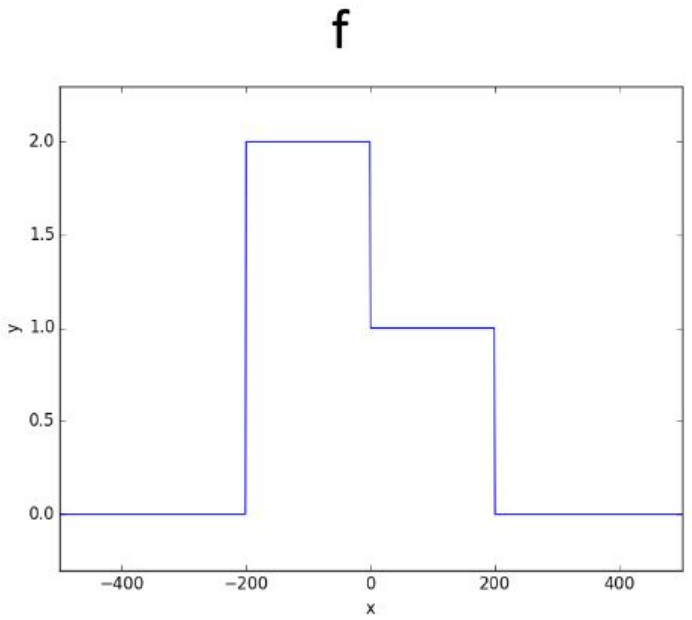
In this example, document b cites document d, and is cited by document a.

Source: Wikipedia

Mathematical Foundations

Convolution

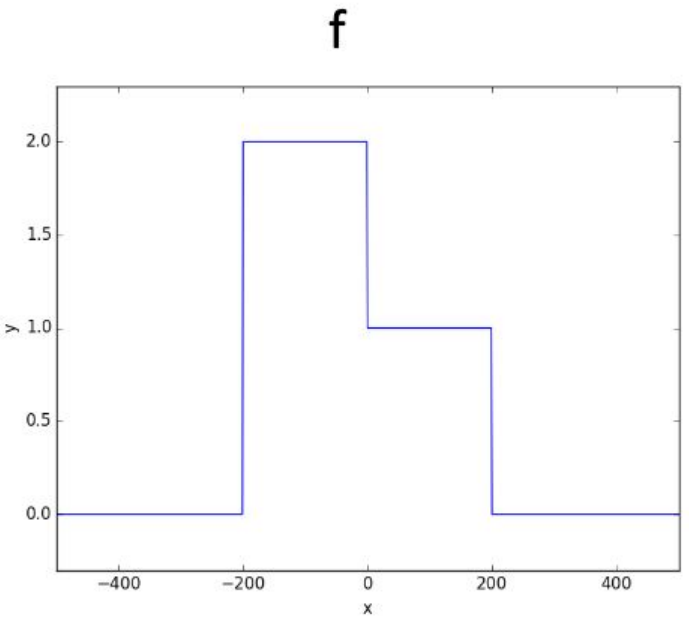
- We are given an array of numerical values



Source: Professor Dror

Convolution

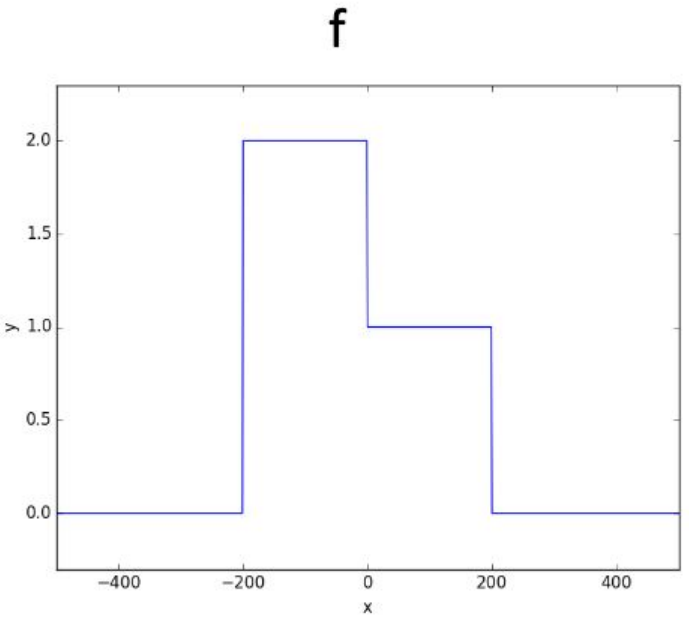
- We are given an array of numerical values
- To compute a moving average, we replace each value in the array with the average of several values that precede and follow it (i.e., the values within a window)



Source: Professor Dror

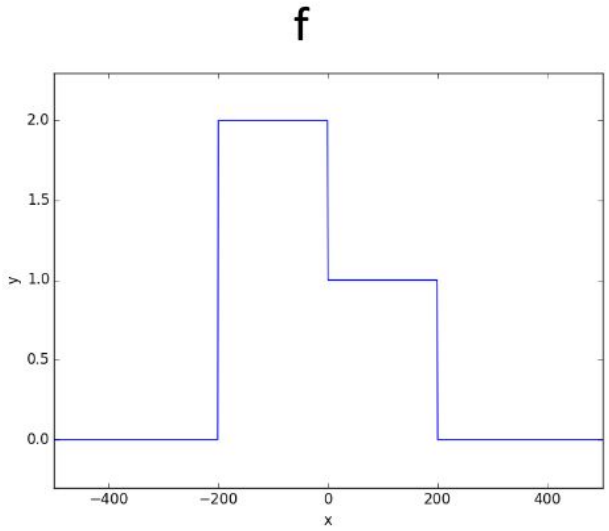
Convolution

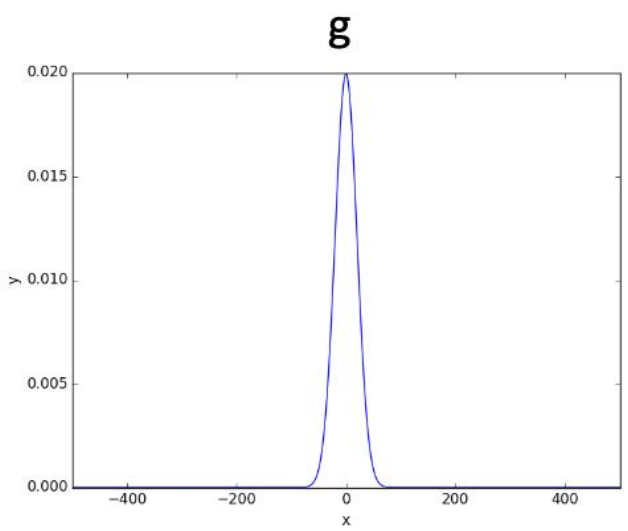
- We are given an array of numerical values
- To compute a moving average, we replace each value in the array with the average of several values that precede and follow it (i.e., the values within a window)
- We might choose instead to calculate a weighted moving average, where we again replace each value in the array with the average of several surrounding values, but we weight those values differently



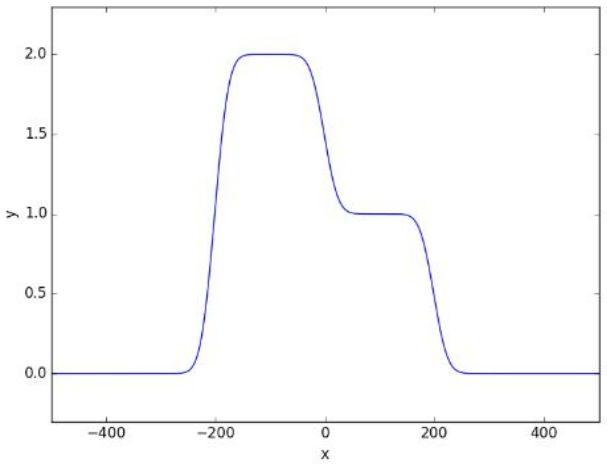
Source: Professor Dror

Convolution



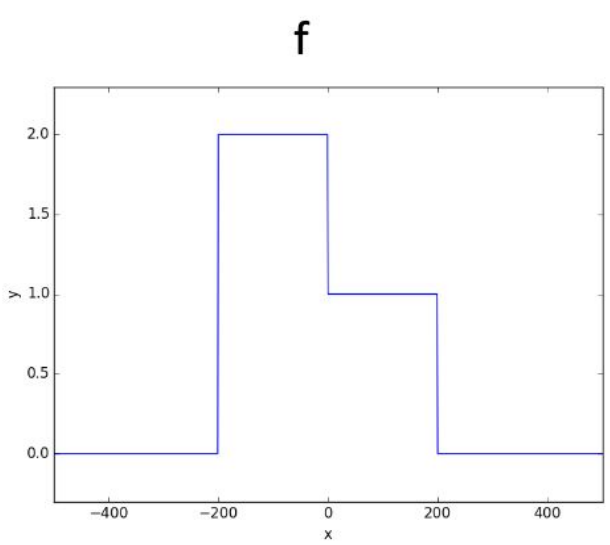


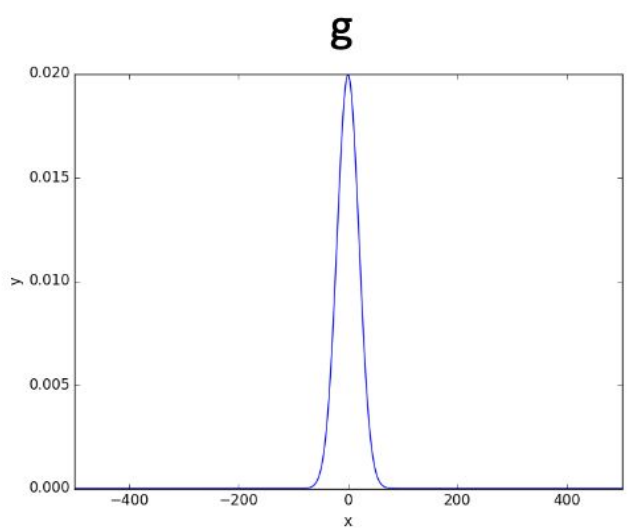
f convolved with g (written f∗g)

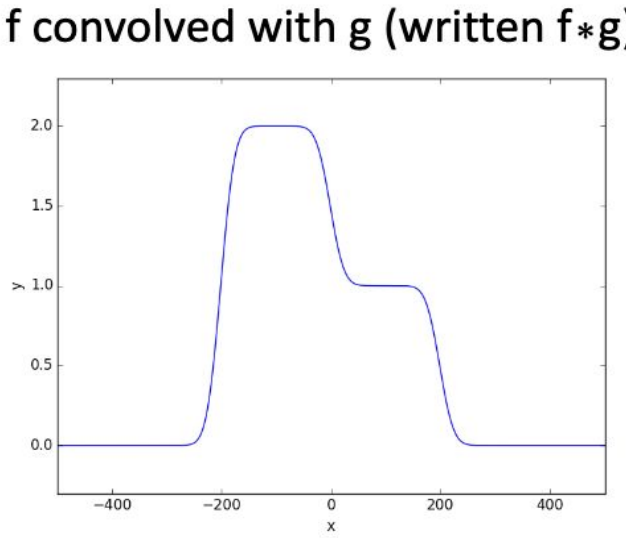


Source: Professor Dror

Convolution







If f and g are functions defined at evenly spaced points, their convolution is given by:

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Source: Professor Dror

Fourier Transforms

- First, some visuals: https://www.jezzamon.com/fourier/

Fourier Transforms

- First, some visuals: https://www.jezzamon.com/fourier/
- Basically transforms a vector in the normal space to different vector in the fourier space

Fourier Transforms

- First, some visuals: https://www.jezzamon.com/fourier/
- Basically transforms a vector in the normal space to different vector in the fourier space
- Calculate convolutions fast!

Convolution theorem

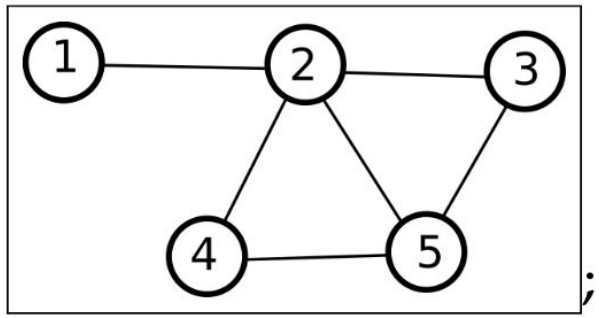
$$f(x,y) * h(x,y) \quad \Leftrightarrow \quad F(u,v)H(u,v)$$

Space convolution = frequency multiplication

Graph Laplacian

$$\mathbf{L} = \mathbf{D} - \mathbf{A}$$

A is the adjacency matrix and **D** is a diagonal matrix with
$d_{ii} = \sum_j a_{ij}$



;

$$\underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ 0 & -1 & 2 & 0 & -1 \\ 0 & -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & -1 & 3 \end{bmatrix}}_{\mathbf{L}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}}_{\mathbf{D}} - \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}}_{\mathbf{A}}$$

Source: Professor Nonato

Normalized Graph Laplacian

Definition 1 *The* **normalized adjacency matrix** *is*

$$\mathscr{A} \equiv D^{-1/2} A D^{-1/2},$$

where A is the adjacency matrix of G and $D = diag(d)$ for $d(i)$ the degree of node i.

For a graph G (with no isolated vertices), we can see that

$$D^{-1/2} = \begin{pmatrix} \frac{1}{\sqrt{d(1)}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{d(2)}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sqrt{d(n)}} \end{pmatrix}.$$

Definition 2 *The* **normalized Laplacian matrix** *is*

$$\mathscr{L} \equiv I - \mathscr{A}.$$

Source: Professor Williamson

Eigendecomposition

$$\begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1/2 \\ 1 \end{bmatrix} = 4 \begin{bmatrix} 1/2 \\ 1/2 \\ 1 \end{bmatrix}$$

A — *eigenvalue* — *eigenvector*

Eigendecomposition

$$\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \mathbf{A}.$$

Eigenvector Matrix

Eigenvalue Matrix

Transpose of Eigenvector Matrix

Spectral Graph Convolutions

Eigendecomposition of the Normalized Graph Laplacian

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

- L = Normalized graph laplacian
- In = Identity matrix of size n x n
- D = Diagonal matrix of degrees of the nodes
- A = Adjacency Matrix

Eigendecomposition of the Normalized Graph Laplacian

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

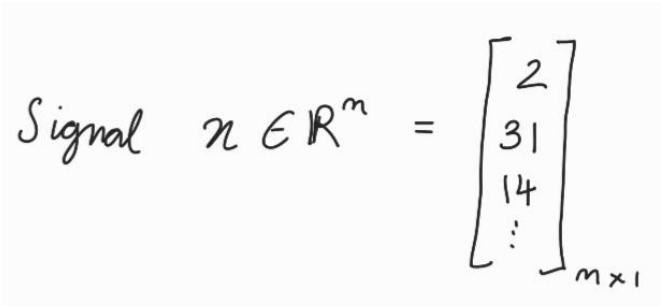
- L = Normalized graph laplacian
- In = Identity matrix of size n x n
- D = Diagonal matrix of degrees of the nodes
- A = Adjacency Matrix
- U = Eigenvector matrix of L
- Λ= Diagonal eigenvalue matrix of L

Signal and Filter

- Assign a scalar to each subject
  - CV = 1, NLP = 2, Audio = 3, and so on

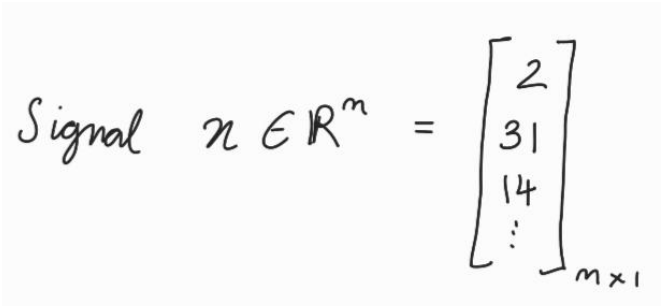
Signal and Filter

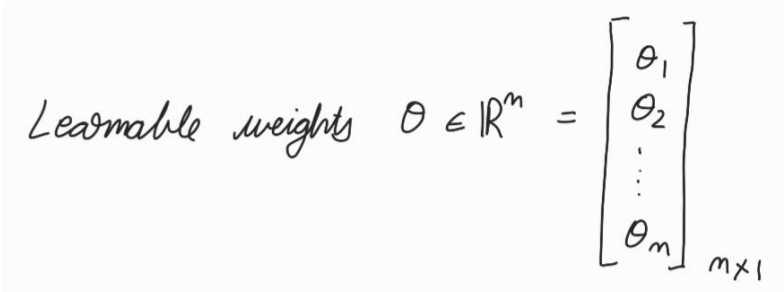
- Assign a scalar to each subject
  - CV = 1, NLP = 2, Audio = 3, and so on



Signal and Filter

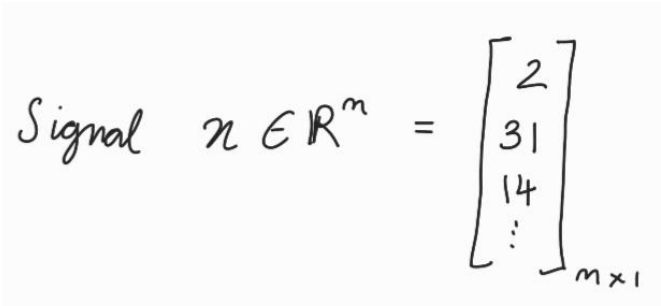
- Assign a scalar to each subject
  - CV = 1, NLP = 2, Audio = 3, and so on

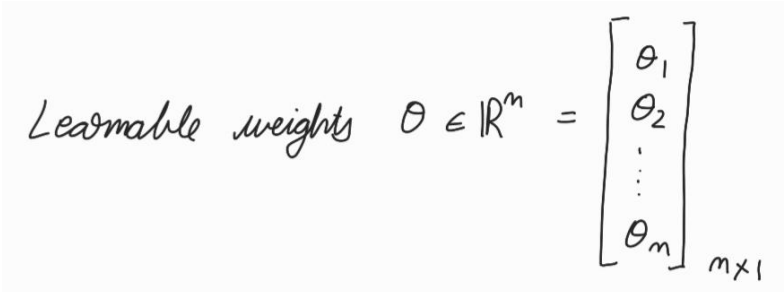


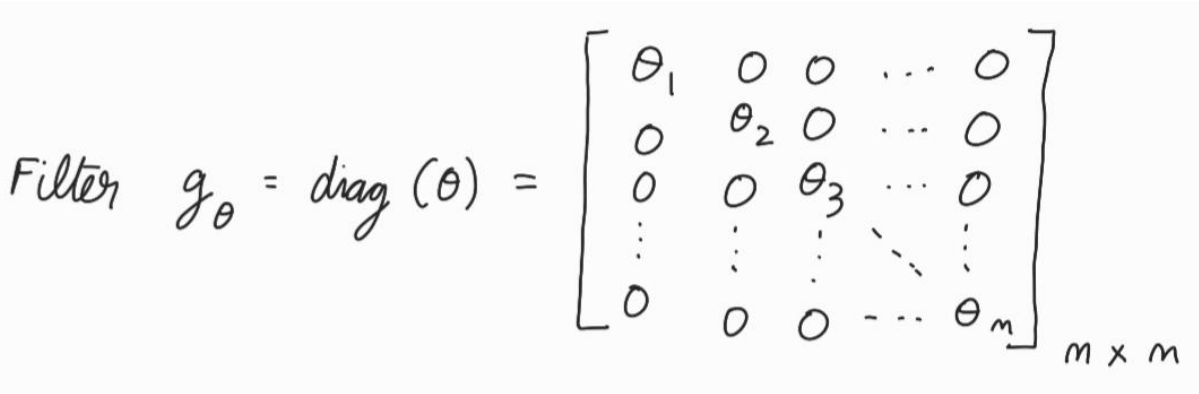


Signal and Filter

- Assign a scalar to each subject
  - CV = 1, NLP = 2, Audio = 3, and so on

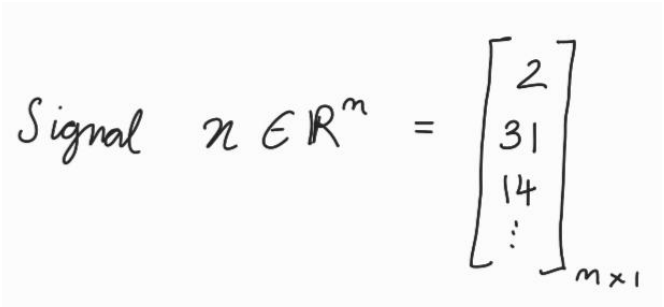


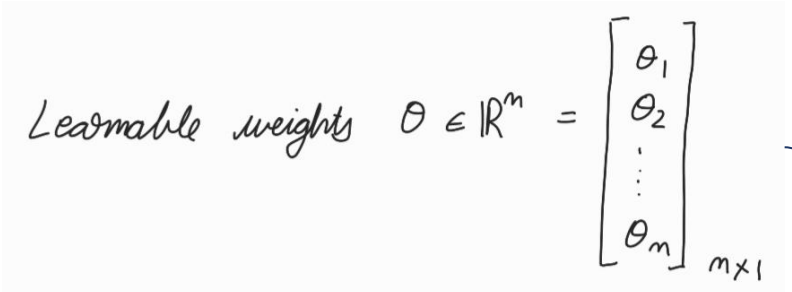


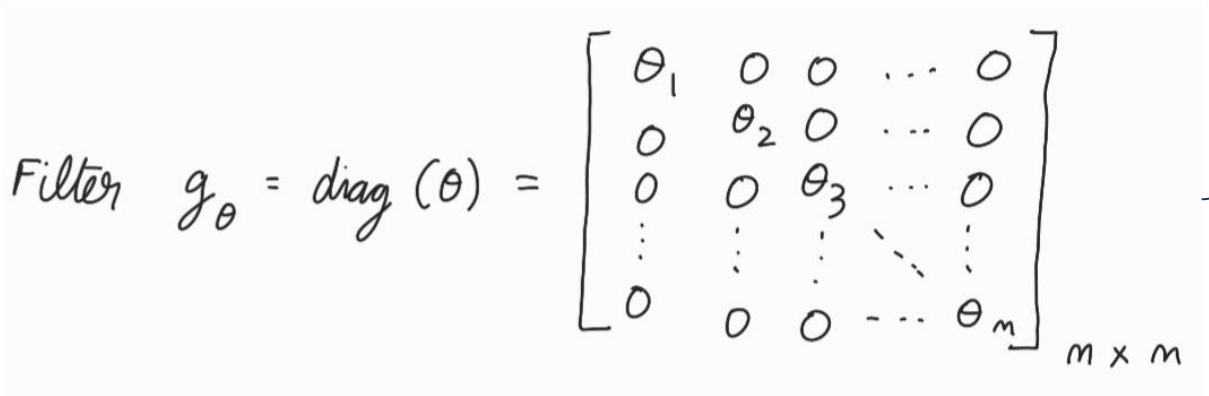


Signal and Filter

- Assign a scalar to each subject
  - CV = 1, NLP = 2, Audio = 3, and so on







Defined in Fourier space

Signal and Filter

- We want to convolve signal x with filter g

Signal and Filter

- We want to convolve signal x with filter g
- To extract important features, properties and patterns

Signal and Filter

- We want to convolve signal x with filter g
- To extract important features, properties and patterns
- Using the Convolution theorem, we know how to shorten this task

Convolution theorem

$$f(x,y) * h(x,y) \quad \Leftrightarrow \quad F(u,v)H(u,v)$$

Space convolution = frequency multiplication

Signal and Filter

- We want to convolve signal x with filter g
- To extract important features, properties and patterns
- Using the Convolution theorem, we know how to shorten this task
- Without proof, for a signal y, Fourier(y) = $U^T y$, Inverse Fourier(y) = Uy
  - Source: Trust me bro

Convolving

$$g_\theta \star x = U g_\theta U^\top x$$

Inverse Fourier

Multiply with filter

x to Fourier

Convolving

$$g_\theta \star x = U g_\theta U^\top x$$

- Eigendecomposition is computationally expensive, $O(n^2)$

Convolving

$$g_\theta \star x = U g_\theta U^\top x$$

- Eigendecomposition is computationally expensive, $O(n^2)$
- So we will approximate this

Convolving

$$g_\theta \star x = U g_\theta U^\top x$$

- Eigendecomposition is computationally expensive, $O(n^2)$
- So we will approximate this
- Using some optimization techniques, assumptions, theorems - outlined in '[Wavelets on Graphs via Spectral Graph Theory]' by Hammond et al. - particularly Chebyshev polynomials, we get…

Approximate Expression

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

Approximate Expression

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \right) x$$

Remember,

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

- In = Identity matrix of size n x n
- D = Diagonal matrix of degrees of the nodes
- A = Adjacency Matrix
- U = Eigenvector matrix of L
- Λ= Diagonal eigenvalue matrix of L

- x = vector of n scalar features
- θ = learnable vector of size n
- gθ = diag(θ) = filter to be convolved

Issues with this approximation

Problem 1

There are no self loops in the adjacency matrix. The model doesn't know that each document is related to itself.

Problem 2

$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is no longer normalized. Multiplying these terms continuously may lead to exploding terms.

Issues with this approximation

Problem 1

There are no self loops in the adjacency matrix. The model doesn't know that each document is related to itself.

Problem 2

$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is no longer normalized. Multiplying these terms continuously may lead to exploding terms.

Solution: Simply add self-loops to A

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

Issues with this approximation

Problem 1

There are no self loops in the adjacency matrix. The model doesn't know that each document is related to itself.

Solution: Simply add self-loops to A

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

Problem 2

$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is no longer normalized. Multiplying these terms continuously may lead to exploding terms.

Solution: *Renormalization trick*

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$

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

Updated Approximate Expression

$$g_\theta \star x \approx \theta \left(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \right) x$$

Remember,

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

- In = Identity matrix of size n x n
- D = Diagonal matrix of degrees of the nodes
- A = Adjacency Matrix
- U = Eigenvector matrix of L
- Λ= Diagonal eigenvalue matrix of L

- x = vector of n scalar features
- θ = learnable vector of size n
- gθ = diag(θ) = filter to be convolved
- $\tilde{A} = A + I_N$
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Final Expression

We can generalize this definition to a signal $X \in \mathbb{R}^{N \times C}$ with C input channels (i.e. a C-dimensional feature vector for every node) and F filters or feature maps as follows:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \,, \tag{8}$$

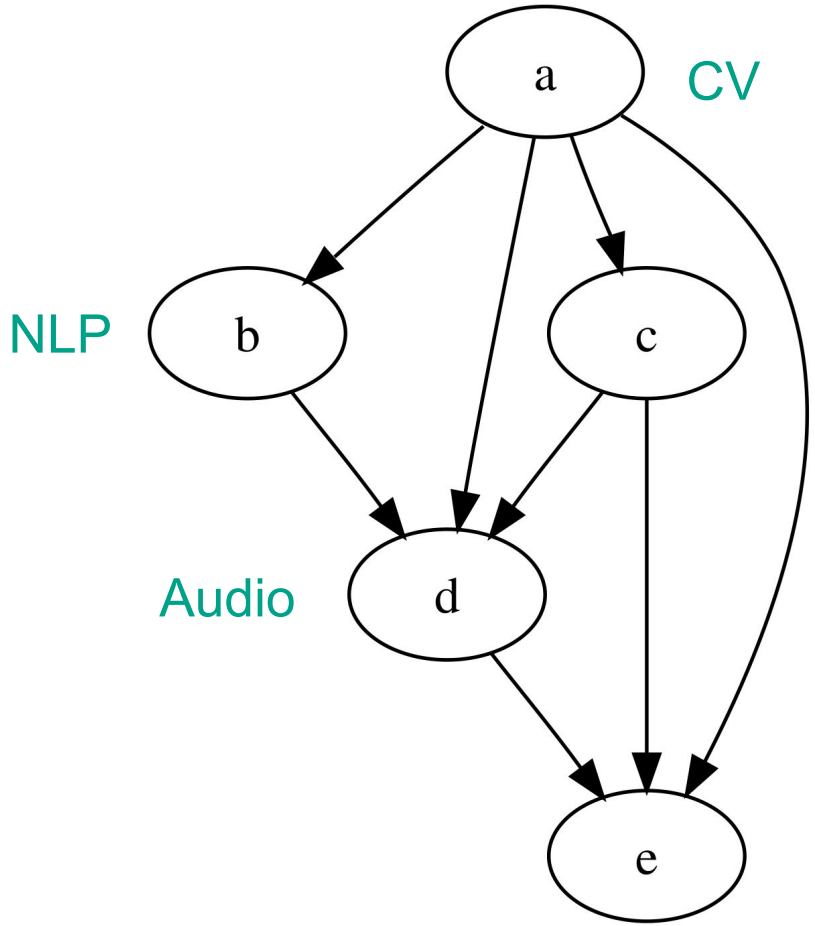
where $\Theta \in \mathbb{R}^{C \times F}$ is now a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the convolved signal matrix.

- In = Identity matrix of size n x n
- D = Diagonal matrix of degrees of the nodes
- A = Adjacency Matrix
- U = Eigenvector matrix of L
- Λ= Diagonal eigenvalue matrix of L

- x = vector of n scalar features
- θ = learnable vector of size n
- gθ = diag(θ) = filter to be convolved
- $\tilde{A} = A + I_N$
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

Semi-Supervised Node Classification

Citation Network - Node Classification

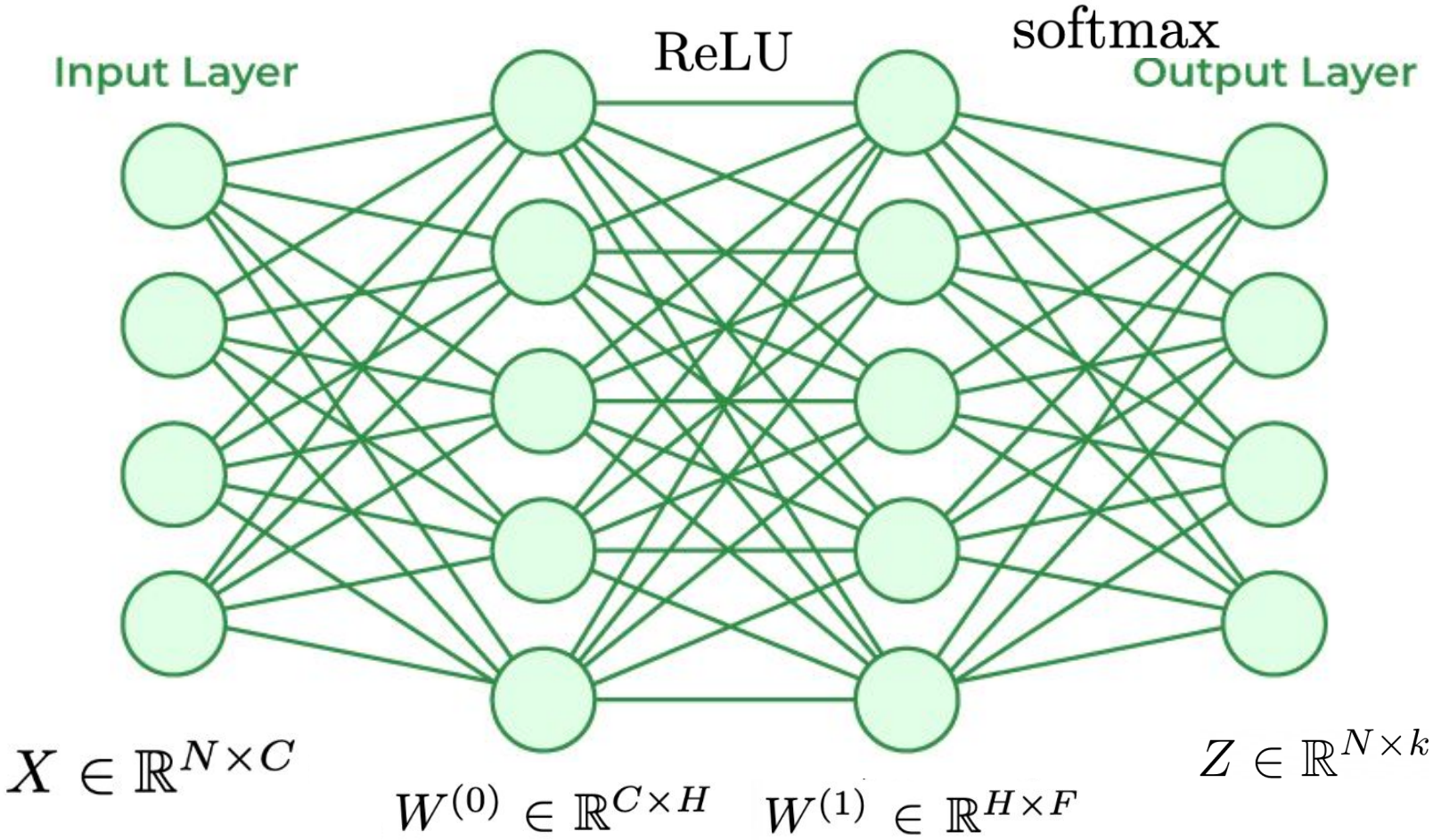


In this example, document b cites document d, and is cited by document a.

Source: Wikipedia

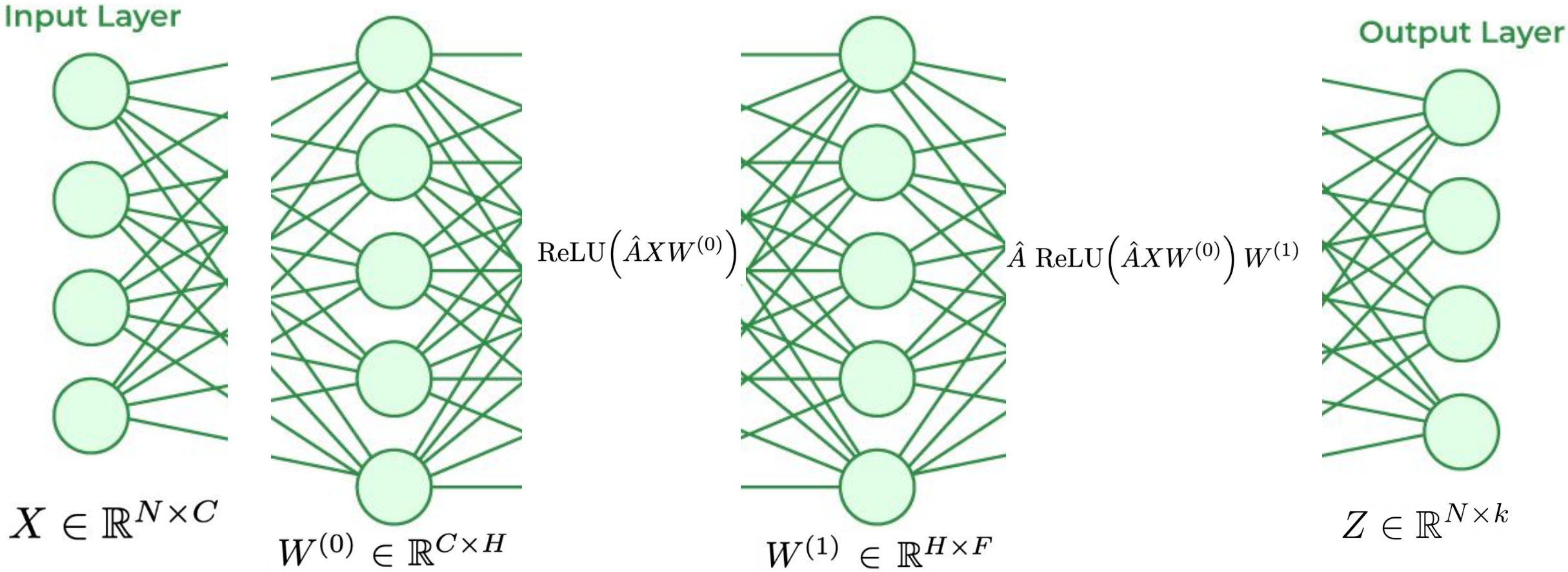
Two-layer GCN

We first calculate $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$



Two-layer GCN

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$$



$$Z = f(X, A) = \mathrm{softmax}\left(\hat{A}\ \mathrm{ReLU}\left(\hat{A}XW^{(0)}\right) W^{(1)}\right)$$

```python
class GCNLayer:
    def __init__(self, in_feat, out_feat):
        self.weight = np.random.randn(in_feat, out_feat)
        self.bias = np.zeros((1, out_feat))
        self.A = None
        self.X = None

    def forward(self, X: np.ndarray, A: np.ndarray):
        self.A = A
        self.X = X
        A_bar = GraphUtils.normalized_graph_laplacien(A)
        return np.dot(np.dot(A_bar, X), self.weight) +
self.bias
    def backward(self, error: np.ndarray, lr):
        L = GraphUtils.normalized_graph_laplacien(self.A)
        feat = np.dot(self.X.T, L)
        grad_weight = np.dot(feat, error)
        grad_bias = np.sum(error, axis=0, keepdims=True)
        grad_input = np.dot(L.T, np.dot(error, self.weight.T))

        self.weight -= lr * grad_weight
        self.bias -= lr * grad_bias
        return grad_input
```

$$Z = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta = \hat{A}XW$$

```python
class GCN:
    def __init__(self, in_feat, hid_feat, out_feat, layers=1):
        self.init_layer = GCNLayer(in_feat, hid_feat)
        self.hidden_layer = GCNLayer(hid_feat, hid_feat)
        self.out_layer = GCNLayer(hid_feat, out_feat)
        self.layers = layers

    def forward(self, X: np.ndarray, A: np.ndarray):
        H = GraphUtils.ReLU(self.init_layer.forward(X, A))
        for i in range(self.layers):
            H = GraphUtils.ReLU(self.hidden_layer.forward(H, A))
        return GraphUtils.softmax(self.out_layer.forward(H, A))

    def backward(self, y, y_hat, alpha=0.01):
        error = (y_hat - y) / y.shape[0]
        gradient_out = self.out_layer.backward(error, alpha)
        gradient_hid = self.hidden_layer.backward(gradient_out,
alpha)  gradient_init = self.init_layer.backward(gradient_hid, alpha)

        return gradient_init, gradient_hid, gradient_out
```

UNIVERSITY OF TORONTO | CVDG

$$Z = f(X, A) = \text{softmax}\Big(\hat{A}\ \text{ReLU}\Big(\hat{A}XW^{(0)}\Big)\ W^{(1)}\Big)$$

PyTorch

PyG

```python
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GCN(nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

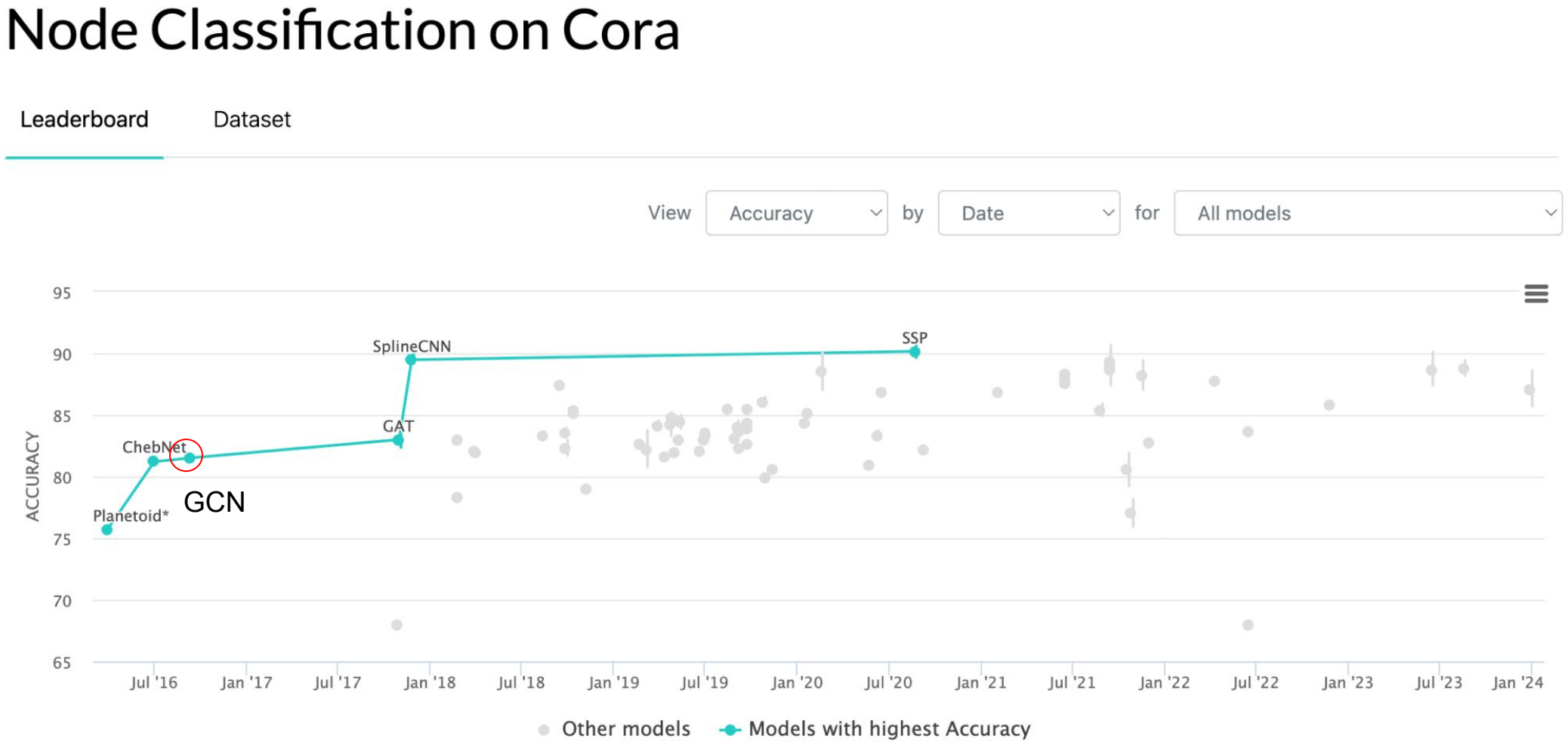
UNIVERSITY OF TORONTO | CVDG

$$Z = f(X, A) = \mathrm{softmax}\Big(\hat{A}\ \mathrm{ReLU}\Big(\hat{A}XW^{(0)}\Big)\,W^{(1)}\Big)$$

Results

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Current Work

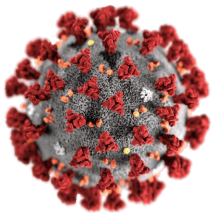


UNIVERSITY OF TORONTO

CVDG

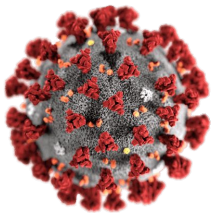
Current Work

- GNNs helped identify existing drugs that could be effective against the virus by analyzing a biomedical knowledge graph called the Drug Repurposing Knowledge Graph (DRKG)



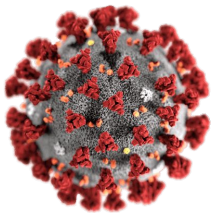
Current Work

- GNNs helped identify existing drugs that could be effective against the virus by analyzing a biomedical knowledge graph called the Drug Repurposing Knowledge Graph (DRKG)
- Platforms like Twitter and Facebook use GNNs to analyze user behavior and improve their social graph algorithms



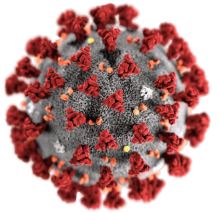
Current Work

- GNNs helped identify existing drugs that could be effective against the virus by analyzing a biomedical knowledge graph called the Drug Repurposing Knowledge Graph (DRKG)
- Platforms like Twitter and Facebook use GNNs to analyze user behavior and improve their social graph algorithms
- Companies like Uber Eats and Pinterest have adopted GNN-based approaches to improve their recommendation engines



Current Work

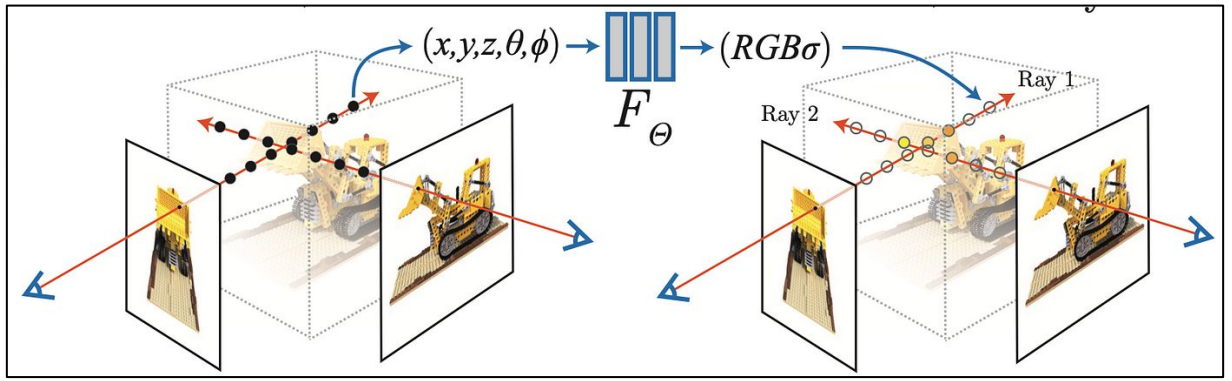
- GNNs helped identify existing drugs that could be effective against the virus by analyzing a biomedical knowledge graph called the Drug Repurposing Knowledge Graph (DRKG)
- Platforms like Twitter and Facebook use GNNs to analyze user behavior and improve their social graph algorithms
- Companies like Uber Eats and Pinterest have adopted GNN-based approaches to improve their recommendation engines
- GNNs help in forecasting traffic flow and congestion, enabling real-time updates and route optimization



Slides on github.com/ut-cvdg/paper-breakdown

Thank You

Up Next: Guest Speaker
Later: NeRF



Follow us on social media

UNIVERSITY OF TORONTO

CVDG