FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Graph Neural Networks

Rui Valente de Almeida

May 3, 2019

FCT NOVA

2019-05-03 Graph Neural Networks

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Graph Neural Networks

Rui Valente de Almeida
May 3, 2019
FCT NOVA

Structure

This presentation was done as one of the assignments for the ISM PhD course. It is focused around Graph Neural Networks, on the theoretical and practical side.

Most of the presentation was built using two articles:

- Zonham et al.: *A comprehensive survey on Graph Neural Networks*;
- Zhou et al.: *Graph Neural Networks: a review of methods and applications*

Structure

This presentation was done as one of the assignments for the ISM PhD course. It is focused around Graph Neural Networks, on the theoretical and practical side.

Most of the presentation was built using two articles:

- Zonham et al.: *A comprehensive survey on Graph Neural Networks*;
- Zhou et al.: *Graph Neural Networks: a review of methods and applications*

Test 2

Structure

The presentation is structured as follows:

- Theoretical background: graphs, Artificial Neural Networks and Graph Neural Networks;
- GNN applications;
- Conclusions.

Structure

The presentation is structured as follows:

- Theoretical background: graphs, Artificial Neural Networks and Graph Neural Networks;
- GNN applications;
- Conclusions.

Graphs

A graph is an ordered pair (V(G), E(G)) and an incidence function ψ_G

Graphs

A graph is an ordered pair (V(G), E(G)) and an incidence function ψ_G

Graphs

A graph is an ordered pair (V(G), E(G)) and an incidence function ψ_G

- V(G) is the set of vertices (or nodes);
- E(G) is the set of edges;

Graphs

A graph is an ordered pair (V(G), E(G)) and an incidence function ψ_G

- V(G) is the set of vertices (or nodes);
- E(G) is the set of edges;

Graphs

The incidence functions creates relations between V and E. Let $e \in E$ and $u, v \in V$. Then if $\psi_G(e) = uv$, e is said to join u and v.

Graphs

The incidence functions creates relations between V and E. Let $e \in E$ and $u, v \in V$. Then if $\psi_G(e) = uv$, e is said to join u and v.

Defining a Graph

We can use a purely mathematical (analytical) form to write a graph:

$$G = (V(G), E(G))$$

where

$$\begin{aligned} V(G) &= \{u, v, w, x, y\} \\ E(G) &= \{a, b, c, d, e, f, g, h\} \end{aligned}$$

and ψ_G is defined by

$$\begin{array}{llll} \psi_G(a) = uv & \psi_G(b) = uu & \psi_G(c) = vw & \psi_G(d) = wx \\ \psi_G(e) = vx & \psi_G(f) = wx & \psi_G(g) = ux & \psi_G(h) = xy \end{array}$$

Defining a Graph
We can use a purely mathematical (analytical) form to write a graph:

Defining a Graph

Sure it is rigorous, but also cumbersome!

$$G = (V(G), E(G))$$

where

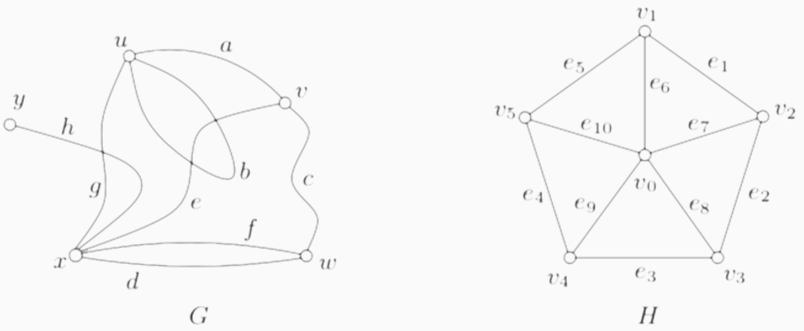
$$\begin{aligned} V(G) &= \{u, v, w, x, y\} \\ E(G) &= \{a, b, c, d, e, f, g, h\} \end{aligned}$$

and ψ_G is defined by

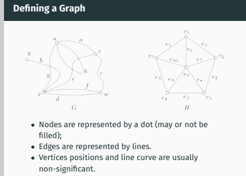
$$\begin{array}{llll} \psi_G(a) = uv & \psi_G(b) = uu & \psi_G(c) = vw & \psi_G(d) = wx \\ \psi_G(e) = vx & \psi_G(f) = wx & \psi_G(g) = ux & \psi_G(h) = xy \end{array}$$

Defining a Graph
Sure it is rigorous, but also cumbersome!

Defining a Graph



- Nodes are represented by a dot (may or not be filled);
- Edges are represented by lines.
- Vertices positions and line curve are usually non-significant.



- Nodes can also be represented by vectors of features;
- When this representation is chosen, we can then write all the nodes as a matrix of features

Defining a Graph

Most definitions in graph theory are built according to these notation rules. For instance:

- The ends of an edge are said to be incident with the edge;

2019-05-03 Graph Neural Networks

└─Defining a Graph

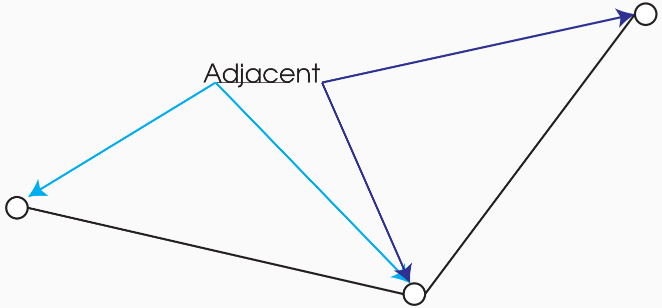
Defining a Graph

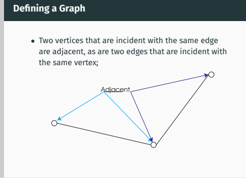
Most definitions in graph theory are built according to these notation rules. For instance:

- The ends of an edge are said to be incident with the edge;

Defining a Graph

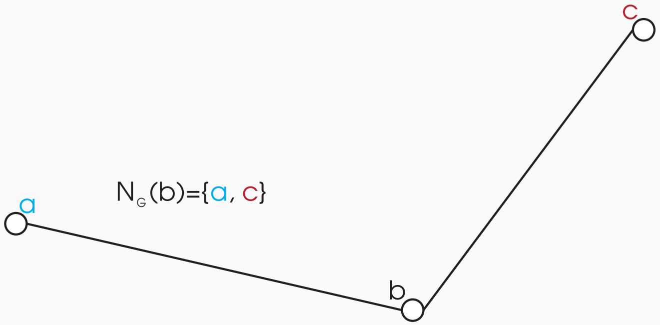
- Two vertices that are incident with the same edge are adjacent, as are two edges that are incident with the same vertex;





Defining a Graph

- Two distinct adjacent vertices are called neighbours. The set of neighbours of vertex v in graph G is denoted $N_G(v)$.



Defining a Graph

- Two distinct adjacent vertices are called neighbours. The set of neighbours of vertex v in graph G is denoted $N_G(v)$.

Graph digital representation

As humans, we prefer graphical representations (an image is worth a thousand words). This is useless for a computer!

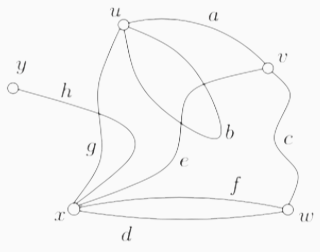
Graph digital representation

As humans, we prefer graphical representations (an image is worth a thousand words). This is useless for a computer!

Graph digital representation

As humans, we prefer graphical representations (an image is worth a thousand words). This is useless for a computer!

But we can also represent a graph in matricial form:



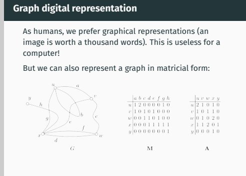
G

	a	b	c	d	e	f	g	h
u	1	2	0	0	0	0	1	0
v	1	0	1	0	1	0	0	0
w	0	0	1	1	0	1	0	0
x	0	0	0	1	1	1	1	1
y	0	0	0	0	0	0	0	1

$\mathbf{M}$

	u	v	w	x	y
u	2	1	0	1	0
v	1	0	1	1	0
w	0	1	0	2	0
x	1	1	2	0	1
y	0	0	0	1	0

$\mathbf{A}$



•

Graph digital representation

Let $\boldsymbol{M}_G \in \mathbb{R}^{m \times n}$ be the incidence matrix of G. Then $\boldsymbol{M}_G := (m_{ve})$, where m_{ve} is the number of times (0, 1 or 2) that vertex v and edge e are incident.

Graph digital representation

Let $\boldsymbol{M}_G \in \mathbb{R}^{m \times n}$ be the incidence matrix of G. Then $\boldsymbol{M}_G := (m_{ve})$, where m_{ve} is the number of times (0, 1 or 2) that vertex v and edge e are incident.

Graph digital representation

Let $\boldsymbol{M}_G \in \mathbb{R}^{m \times n}$ be the incidence matrix of G. Then $\boldsymbol{M}_G := (m_{ve})$, where m_{ve} is the number of times (0, 1 or 2) that vertex v and edge e are incident.

Let $\boldsymbol{A}_G \in \mathbb{R}^{n \times n}$ be the adjacency matrix of G. Then $\boldsymbol{A}_G := (a_{uv})$, where a_{uv} is the number of edges joining vertices u and v.

Graph digital representation

Let $\boldsymbol{M}_G \in \mathbb{R}^{m \times n}$ be the incidence matrix of G. Then $\boldsymbol{M}_G := (m_{ve})$, where m_{ve} is the number of times (0, 1 or 2) that vertex v and edge e are incident.

Let $\boldsymbol{A}_G \in \mathbb{R}^{n \times n}$ be the adjacency matrix of G. Then $\boldsymbol{A}_G := (a_{uv})$, where a_{uv} is the number of edges joining vertices u and v.

There are other, more compact, versions of digital representatin of graphs. This falls out of the scope of this presentation.

Graph Signal Processing

As long as it's undirected, a graph can be represented by its normalised Laplacian matrix:

$$\boldsymbol{L}^{sym} = \boldsymbol{I}_n - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$$

In this equation, $\boldsymbol{A}$ is the adjacency matrix and $\boldsymbol{D}$ is a diagonal matrix of node degrees, $\boldsymbol{D}_{ii} = \sum_j (A_{ij})$.

Graph Signal Processing

As long as it's undirected, a graph can be represented by its normalised Laplacian matrix:

$$\boldsymbol{L}^{sym} = \boldsymbol{I}_n - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$$

In this equation, $\boldsymbol{A}$ is the adjacency matrix and $\boldsymbol{D}$ is a diagonal matrix of node degrees, $\boldsymbol{D}_{ii} = \sum_j (A_{ij})$.

- An undirected graph is one for which the edges have no particular direction.

Graph Signal Processing

This matrix is real symmetric positive semidefinite, so it can be factorised as:

$$\boldsymbol{L}^{sym} = \boldsymbol{U}\Lambda\boldsymbol{U}^T$$

Where $\boldsymbol{U} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-1}] \in \mathbb{R}^N$ is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues, $\Lambda_{ii} = \lambda_i$

Graph Signal Processing

This matrix is real symmetric positive semidefinite, so it can be factorised as:

$\boldsymbol{L}^{sym} = \boldsymbol{U}\Lambda\boldsymbol{U}^T$

Where $\boldsymbol{U} = [\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{n-1}] \in \mathbb{R}^N$ is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues, $\Lambda_{ii} = \lambda_i$

- To this we call the eigendecomposition
- $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$

Convolution

Convolution is a mathematical operation on two functions, that tells us how the shape of the first function modifies the shape of the second.

Convolution

Convolution is a mathematical operation on two functions, that tells us how the shape of the first function modifies the shape of the second.

- cross correlation is the adjoint operator of convolution;

Convolution

Here is the mathematical definition:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau$$

We can look at this expression as a weighted average of $f(\tau)$ at the moment t where the weighting is given by $g(-\tau)$.

Convolution

Here is the mathematical definition:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau) \cdot g(t - \tau) d\tau$$

We can look at this expression as a weighted average of $f(\tau)$ at the moment t where the weighting is given by $g(-\tau)$.

Convolution

The convolution theorem simplifies the calculation of the convolution. It's written as:

$$\mathcal{F}\{f \star g\} = k \cdot \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

This theorem holds for graphs as well, and we can write it theough the use of the eigendecomposition of the Adjacency matrix.

Convolution

The convolution theorem simplifies the calculation of the convolution. It's written as:

$\mathcal{F}\{f \star g\} = k \cdot \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$

This theorem holds for graphs as well, and we can write it theough the use of the eigendecomposition of the Adjacency matrix.

- The convolution of f and g is the inverse Fourier transform of the multiplication of the Fourier transforms of f and g.

Graph Convolution

The graph's signal Fourier transform is written $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{U}^T\boldsymbol{x}$, with the inverse being $\mathcal{F}^{-1}(\hat{\boldsymbol{x}}) = \boldsymbol{U}\hat{\boldsymbol{x}}$, with $\hat{\boldsymbol{x}}$ being the transformed signal.

Graph Convolution

The graph's signal Fourier transform is written $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{U}^T\boldsymbol{x}$, with the inverse being $\mathcal{F}^{-1}(\hat{\boldsymbol{x}}) = \boldsymbol{U}\hat{\boldsymbol{x}}$, with $\hat{\boldsymbol{x}}$ being the transformed signal.

Graph Convolution

Now, using the previously stated convolution theorem, we can write the convolution between a graph node's signal $\boldsymbol{x}$ and a filter $\boldsymbol{g}$, which is also a signal:

$$\boldsymbol{x} \star \boldsymbol{g} = \mathcal{F}^{-1}\left(\mathcal{F}\left(\boldsymbol{x}\right) \odot \mathcal{F}\left(\boldsymbol{g}\right)\right)$$

Where $\odot$ denotes element-wise product (instead of matrix product).

Graph Convolution

Now, using the previously stated convolution theorem, we can write the convolution between a graph node's signal $\boldsymbol{x}$ and a filter $\boldsymbol{g}$, which is also a signal:

$$\boldsymbol{x} \star \boldsymbol{g} = \mathcal{F}^{-1}\left(\mathcal{F}\left(\boldsymbol{x}\right) \odot \mathcal{F}\left(\boldsymbol{g}\right)\right)$$

Where $\odot$ denotes element-wise product (instead of matrix product).

Graph Convolution

Now, the previous equation can be decomposed with the Fourier transform definition for graph signals:

$$\boldsymbol{x} \star \boldsymbol{g} = \boldsymbol{U}\left(\boldsymbol{U}^T\boldsymbol{x} \odot \boldsymbol{U}^T\boldsymbol{g}\right)$$

Graph Convolution
Now, the previous equation can be decomposed with the Fourier transform definition for graph signals:
$$\boldsymbol{x} \star \boldsymbol{g} = \boldsymbol{U}\left(\boldsymbol{U}^T\boldsymbol{x} \odot \boldsymbol{U}^T\boldsymbol{g}\right)$$

This is the definition of convolution that is used in both papers and in the GNN community at large, as far as I had the opportunity to see.

This definition of convolution has a problem: is is computationally very heavy and many times prohibitively so. To circumvent this problem, some authors have suggested using polynomial expansions, such as Chebyshev's, to represent the matrices

Artificial Neural Networks

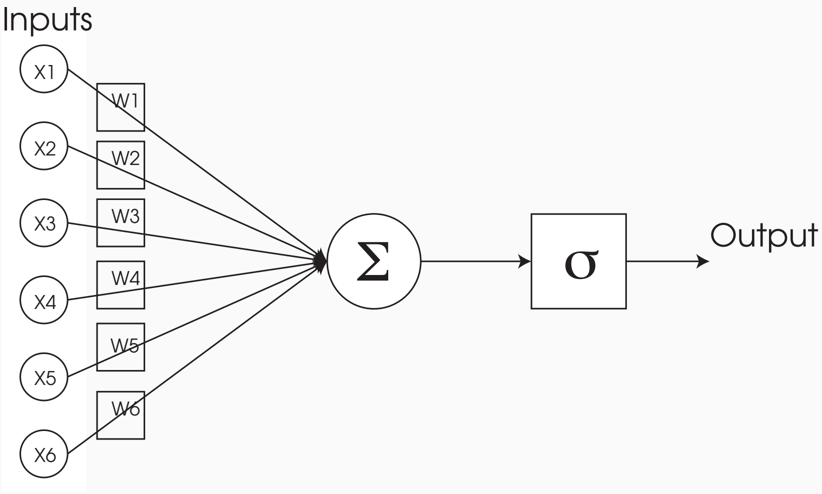
Artificial Neural Networks are today's most popular artificial intelligence tools.

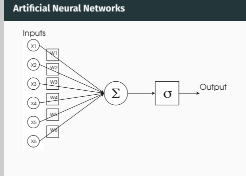
Artificial Neural Networks

Artificial Neural Networks are today's most popular artificial intelligence tools.

For all the misticism around them, ANNs are remarkably simple mathematical entities, especially given the fact that they are (really) capable of solving any problem that can be expressed as a function.

Artificial Neural Networks





- ANNs stem from mathematical theories developed in the 1940s;
- The most basic form of modern ANN, the perceptron, was created in the 1950s;
- Back then, the structure had some problems, namely the fact that it could not compute the exclusive-or operation.
- Everything changed in the 1980s, when the backpropagation algorithm was invented.

Artificial Neural Networks

The backpropagation algorithm appeared in the 1980's:

$$W^{t+1} = W^t - \eta \nabla_E^t$$

In which t is hte iteration, W represents the weights, η the learning rate and ∇_E the gradient of the loss function (which normally is MSE).

Artificial Neural Networks

The backpropagation algorithm appeared in the 1980's:

$$W^{t+1} = W^t - \eta \nabla_E^t$$

In which t is hte iteration, W represents the weights, η the learning rate and ∇_E the gradient of the loss function (which normally is MSE).

- First models of neural network could not implement the exclusive-or;
- For that, and for the comuptational cost, they were less used than they promised;

Artificial Neural Networks

The backpropagation algorithm appeared in the 1980's:

$$W^{t+1} = W^t - \eta \nabla_E^t$$

In which t is hte iteration, W represents the weights, η the learning rate and ∇_E the gradient of the loss function (which normally is MSE).

Artificial Neural Networks

The backpropagation algorithm appeared in the 1980's:

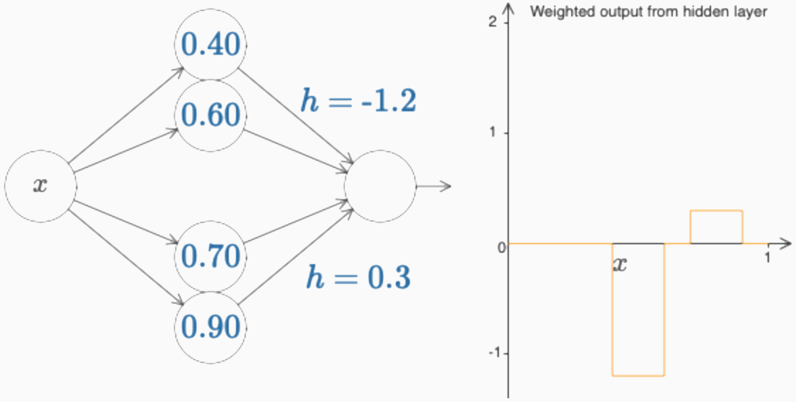
$$W^{t+1} = W^t - \eta \nabla_E^t$$

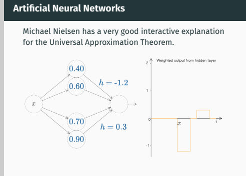
In which t is hte iteration, W represents the weights, η the learning rate and ∇_E the gradient of the loss function (which normally is MSE).

- gradient descent is used to minimize the loss function;
- the learning rate parameterises GD so that we can escape or avoid local minima;
- with this algorithm, and the introduction of non-linearities like the sigmoid or the ReLu functions in the network, ANNs can literally approximate any function.

Artificial Neural Networks

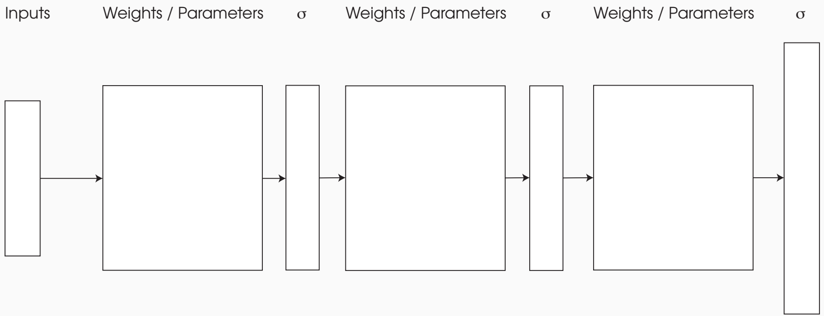
Michael Nielsen has a very good interactive explanation for the Universal Approximation Theorem.

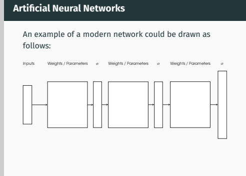




Artificial Neural Networks

An example of a modern network could be drawn as follows:





- ANNS are just a stack of linear and non linear operations that can approximate any problem in function form to any precision;
- They rely on simple matrix operations, like multiplication and convolution

Graph Neural Networks

Graphs are data just like any other...but they are non Euclidean!

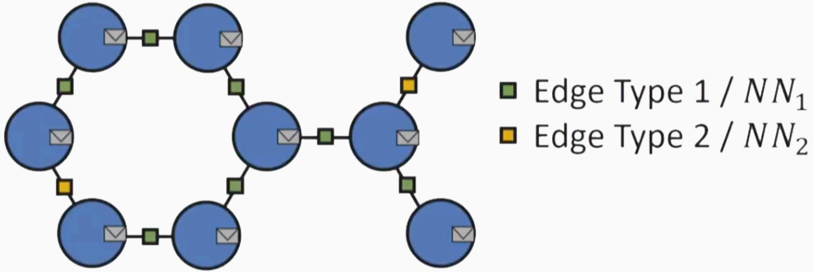
Graph Neural Networks

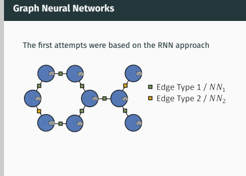
Graphs are data just like any other...but they are non Euclidean!

- In Euclidean data, spatial structure is well and hard defined;
- We always know the distance between one ”node” and the next;
- in graphs and non-euclidean data, this is not true –it is highly irregular
- So if we want to classify graphs..how do we do it?

Graph Neural Networks

The first attempts were based on the RNN approach

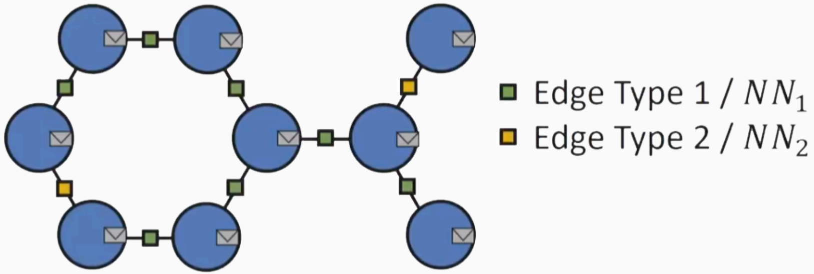


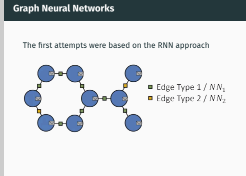


- Consider a feature vector for each node;
- Consider each feature as a message (remember envelope);
- Each ”message” is passed to the next iteration order neighbour of every node;
- In the end, messages are summed and classified;

Graph Neural Networks

The first attempts were based on the RNN approach





- In this model, information decays too rapidly with node distance (exponential deacy)
- Next models were Gated Recurrent Units, which addressed this deacying problem
- In the mean time, some models appeared which instead of RNNs, were based in convolution and CNN approach.
- In fact, Gilmer et al claim that GCNs and GRNN are the same thing, and generalise it using two operating moments;

Graph Convolutional Networks

In 2016, a new concept called Message Passing Neural Networks appeared, which unified both approaches.

Graph Convolutional Networks
In 2016, a new concept called Message Passing Neural Networks appeared, which unified both approaches.

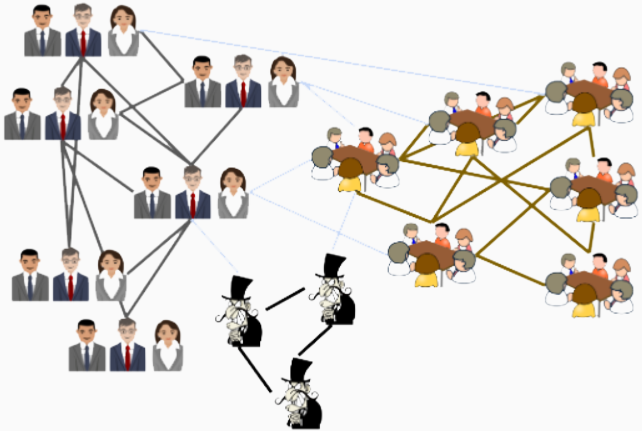
- Two stages: convolution and aggregation, in which convolution is also called message passing.
- This is a nice place to make a small demonstration.

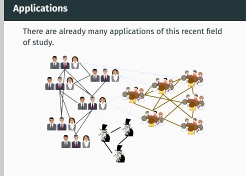
Applications of GNNS

Applications of GNNS

Applications

There are already many applications of this recent field of study.

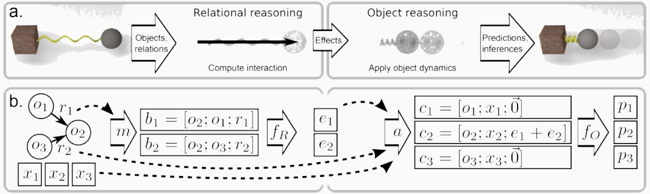


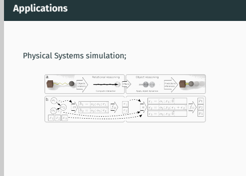


- text;
- image classification;
- Social relationship analysis;
- Semantic segmentation;
- We present some of the main applications of AI in graphs;

Applications

Physical Systems simulation;

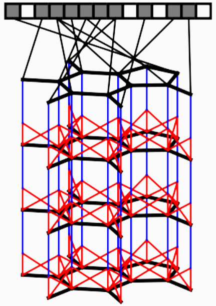


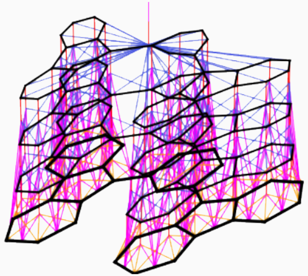


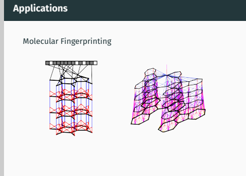
- This system was developed in x and Y;
- It tries to simulate and predict interactions between physical bodies;
-

Applications

Molecular Fingerprinting

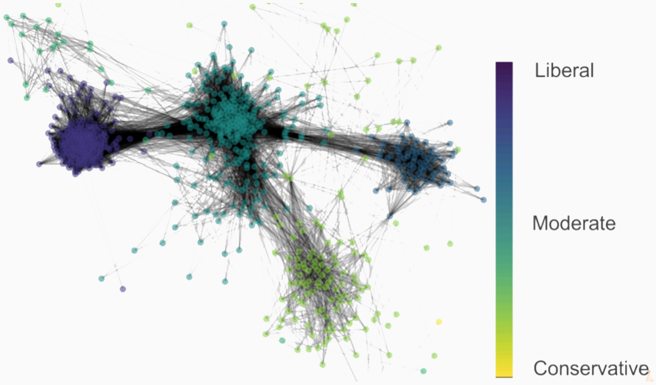


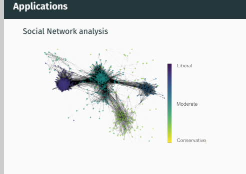




Applications

Social Network analysis





Thank you

rf.almeida@campus.fct.unl.pt
ruivalmeida/ism_gnn_share

Thank you
rf.almeida@campus.fct.unl.pt
ruivalmeida/ism_gnn_share