Graph Neural Networks

Thomas Bonald and Tiphaine Viard

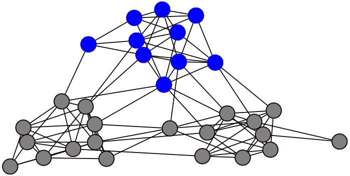
2020 – 2021

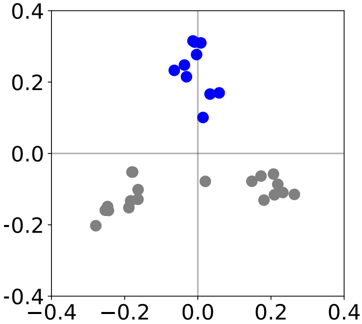
Outline

1. Sparse matrices
2. PageRank
3. Clustering
4. **Embedding**

Graph embedding

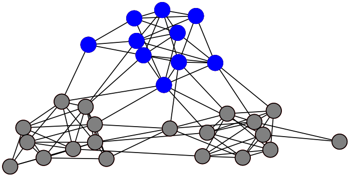
How to transform **graph data** into **vector data**, so as to preserve the **proximity** between nodes?

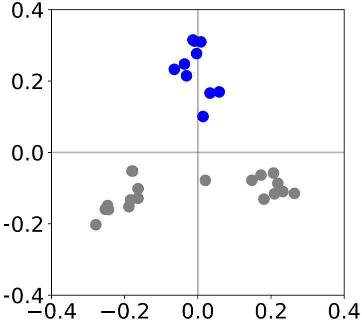




Graph embedding

How to transform **graph data** into **vector data**, so as to preserve the **proximity** between nodes?





We first assume that the graph is **undirected**

Getting inspiration from language processing: word2vec

Goal: Predict **contextual words**

How ? Extract vector representations of words in a text

$$\cos(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \in [-1, 1]$$

Two models: CBOW vs skip-gram

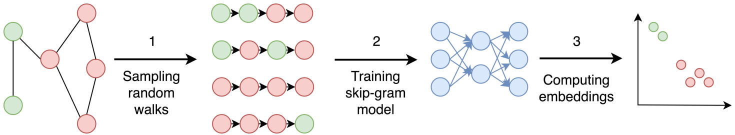
Trick: train a neural net, but without an end task

On graphs: node2vec

Text: A special graph

Voyez $\rightarrow$ ce $\rightarrow$ koala $\rightarrow$ fou $\rightarrow$ qui $\rightarrow$ mange $\rightarrow$ des $\rightarrow$ journaux...

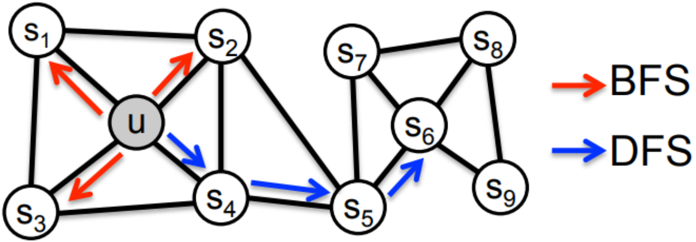
Solution: random walks, again!



Node2vec

Actually, **biased** random walks

- Walk length: How many nodes are in each random walk
- p: return parameter
- q: Breadth-depth parameter



Node2vec

Actually, **biased** random walks

- Walk length: How many nodes are in each random walk
- p: return parameter
- q: Breadth-depth parameter

Objective:

$$\max_f \sum_u \log Pr(N(u)|f(u))$$

i.e. similar nodes will be in each others' neighbourhood

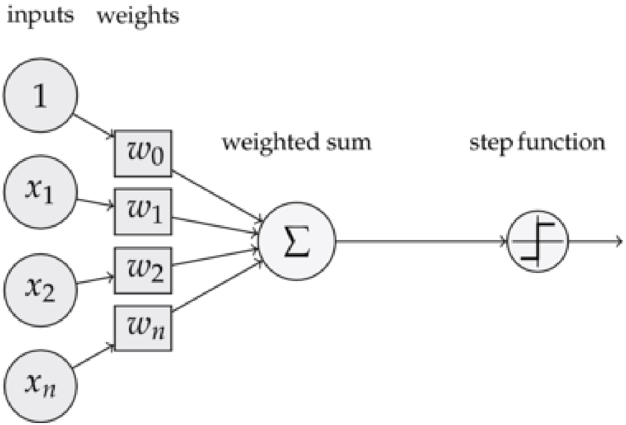
Motivation for graph neural networks

Why are embeddings not enough?

- We want to recreate deep learning, but for graphs
- Embeddings are costly to compute (no shared parameters!)
- Embeddings cannot generalize to new graphs
- Can we use regular neural networks?
- Features vs propagating on the graph
- Some say reasoning uses a graph-like structure

But first, let's take a step back...

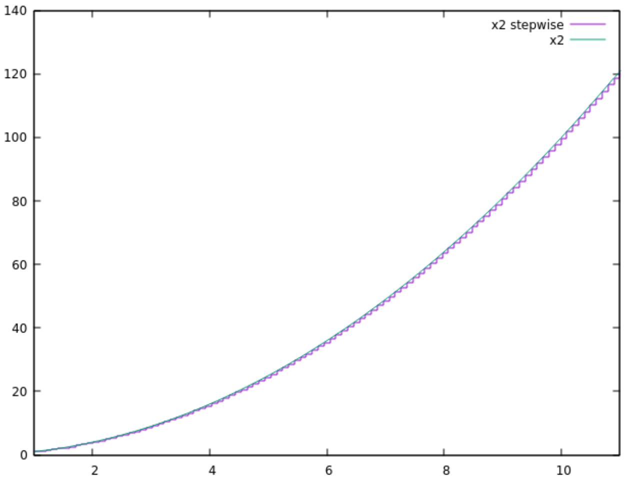
The perceptron



$$y = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

(Rosenblatt, 1957)

Multilayer perceptron

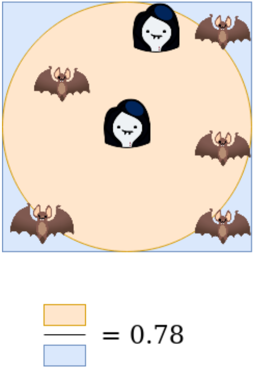


"universal approximator"

- proof of existence: it does not say **how**
- curse of dimensionality

Curse of dimensionality

aka how much data do I need?



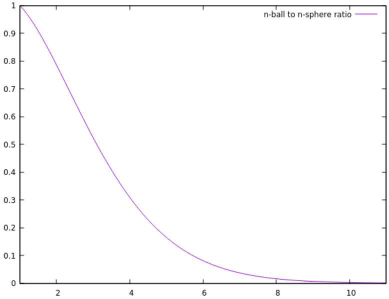
$$V = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)\cdot 2^d} \approx \mathcal{O}(\epsilon^{-d})$$

In general, to approximate a function $f : \mathbb{R}^d \to \mathbb{R}$, you need $\mathcal{O}(\epsilon^{-d})$ points

But we are not *in general*

Curse of dimensionality

aka how much data do I need?



$$V = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)\cdot 2^d} \approx \mathcal{O}(\epsilon^{-d})$$

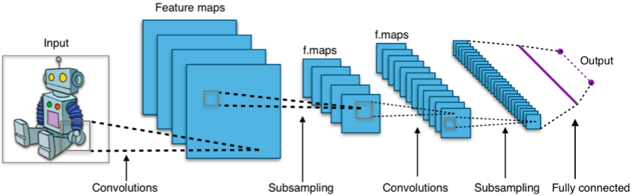
In general, to approximate a function $f : \mathbb{R}^d \to \mathbb{R}$, you need $\mathcal{O}(\epsilon^{-d})$ points

But we are not *in general*

Let's forget a bit about general neural networks...

Convolutional Neural Networks

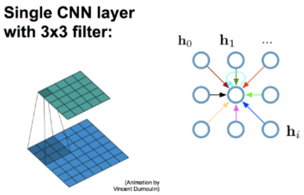
CNNs are great



Two ideas: convolution and **pooling**

Fixed number of neighbours for each node, **strong locality**, very **scalable**
Shift invariance, **shared weights**
Able to find **mesoscale structures**

From graphs to images



One update:

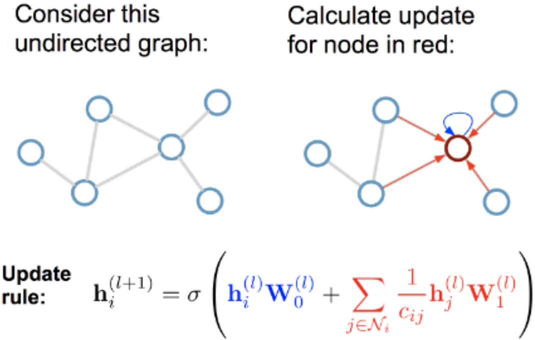
- Transform each pixel $\boldsymbol{W_i h_i}$
- Sum it up $\sum_i \boldsymbol{W_i h_i}$

Full update:

$$\boldsymbol{h}_4^{(l+1)} = \sigma(\boldsymbol{W}_0^{(\boldsymbol{l})}\boldsymbol{h}_0^{(\boldsymbol{l})} + \boldsymbol{W}_1^{(\boldsymbol{l})}\boldsymbol{h}_1^{(\boldsymbol{l})}) + \cdots + \boldsymbol{W}_8^{(\boldsymbol{l})}\boldsymbol{h}_8^{(\boldsymbol{l})}$$

Onto general graphs

Graphs and grids are not *that different*

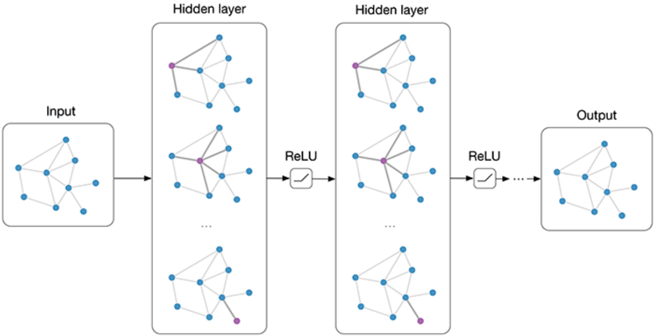


- Shift invariance $\rightarrow$ permutation invariance
- Still scalable
- Still shared weights

Graph Convolutional Networks

Kipf and Welling, ICLR 2017

Main idea: pass messages between nodes and aggregate



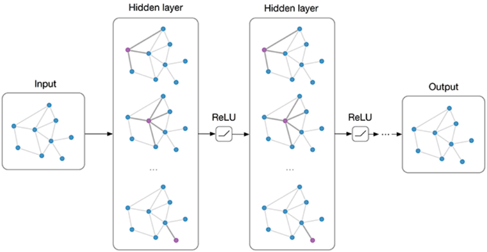
$$\boldsymbol{h}_i^{(l+1)} = \sigma\left(\boldsymbol{h}_i^{(l)}\boldsymbol{W}_0^{(l)} + \sum_{j \in N(i)} \frac{1}{c_{ij}}\boldsymbol{h}_j^{(l)}\boldsymbol{W}_j^{(l)}\right)$$

$N(i)$: neighbours of node i

c_{ij}: constant, trainable

Real-world problems with GNNs

Input: Features $\boldsymbol{X} \in \mathbb{R}^{\mathbb{N} \times \mathbb{E}}$, adjacency matrix $\hat{\boldsymbol{A}}$



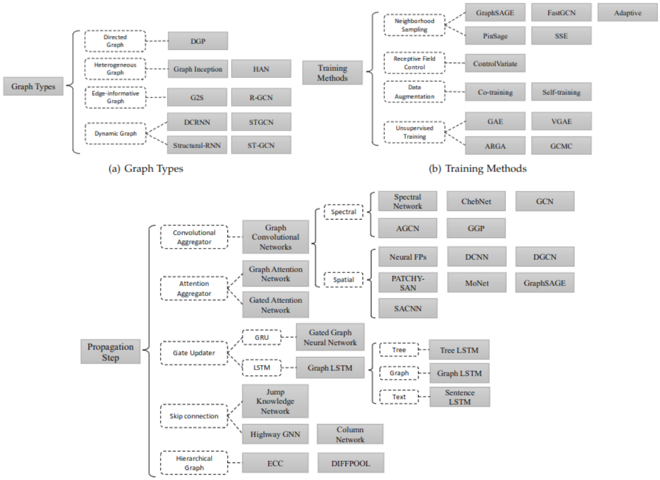
Node classification?
$softmax(\boldsymbol{z}_n)$

Graph classification?
$softmax(\sum_n \boldsymbol{z}_n)$

Link prediction?
$p(A_i j) = \sigma(\boldsymbol{z}_i^T \boldsymbol{z}_j)$

figure by Thomas Kipf

Graph Neural Networks



(a) Graph Types (b) Training Methods

(c) Propagation Steps

(Zhou et al., 2019)

Some perspectives

Graph neural networks are **generalizations** of traditional neural networks

i.e. GCN on a grid is a CNN

They **scale** very well thanks to sampling

What are the consequences?

What happens in the dynamic case?

Should we generalize LSTMs? Bridge with link streams and dynamic graphs?

Summary

Many data have a graph structure, which requires suitable **data structures** and **algorithms**:

- sparse matrices
- PageRank
- Louvain
- spectral embedding
- node embeddings

See `scikit-network` and `Deep Graph Library`