Graphs and graph theory

A set V of vertices and a set E of unordered and ordered pairs of vertices; denoted by $G(V, E)$. An unordered pair of vertices is said to be an **edge**, while an ordered pair is said to be an **arc**. A graph containing edges alone is said to be **non-oriented**; a graph containing arcs alone is said to be **oriented**. An arc (or edge) can begin and end at the same vertex, in which case it is known as a **loop**.

One says that an edge u, v connects two vertices u and v, while an arc (u, v) begins at the vertex u and ends at the vertex v. Vertices connected by an edge or a loop are said to be **adjacent**. Edges with a common vertex are also called **adjacent**. An edge (arc) and any one of its two vertices are said to be **incident**.

There are various ways of specifying a graph. Let $u_1, ..., u_n$ be the vertices of a graph $G(V, E)$ and let $e_1, ..., e_m$ be its edges. The **adjacency matrix** corresponding to G is the matrix $A = (a_{i,j})$ in which the element $a_{i,j}$ equals the number of edges (arcs) which join the vertices u_i and u_j (go from u_i to u_j) and $a_{i,j} = 0$ if the corresponding vertices are not adjacent. A sequence of edges $(u_0, u_1), ..., (u_{r-1}, u_r)$ is called an **edge progression** connecting the vertices u_0 and u_r. An edge progression is called a **chain** if all its edges are different and a simple chain or path if all its vertices are different. A closed (simple) chain is also called a (simple) **cycle**.

Matrice di adiacenza. Wolfram MathWorld

The **degree** of a vertex u_i of a graph G, denoted by d_i, is the number of edges incident with that vertex. The **length** of an edge progression (chain, simple chain) is equal to the number of edges in the order in which they are traversed. The length of the shortest simple chain connecting two vertices u_i and u_j in a graph G is said to be the **distance** $d(u_i, u_j)$ between u_i and u_j. The quantity $min_{u_i} max_{u_j} d(u_i, u_j)$ is called the **diameter**, while a vertex u_0 for which $max_{u_j} d(u_i, u_j)$ assumes its minimum value is called a centre of G. A graph can contain more than one centre or no centre at all.
Graph. Encyclopedia of Mathematics

Geometric Deep Learning

Graph embeddings

Graph embeddings are the transformation of property graphs to a vector or a set of vectors. Embedding should capture the graph topology, vertex-to-vertex relationship, and other relevant information about graphs, subgraphs, and vertices.

The **DeepWalk** method uses random walks to produce graph embeddings. The random walk starts in a selected node then we move to the random neighbor from a current node for a defined number of steps. The method basically consists of three steps: 1. **Sampling**: A graph is sampled with random walks. Few random walks from each node are performed. Authors show that it is sufficient to perform from 32 to 64 random walks from each node. They also show that good random walks have a length of about 40 steps. 2. **Training skip-gram**: Random walks are comparable to sentences in word2vec approach. The skip-gram network

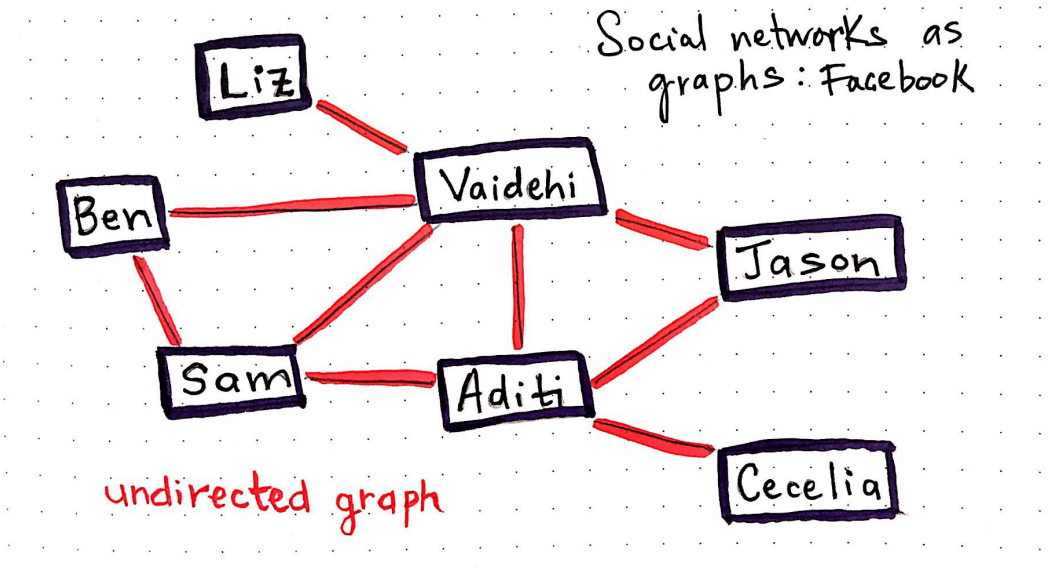

Figure 1:

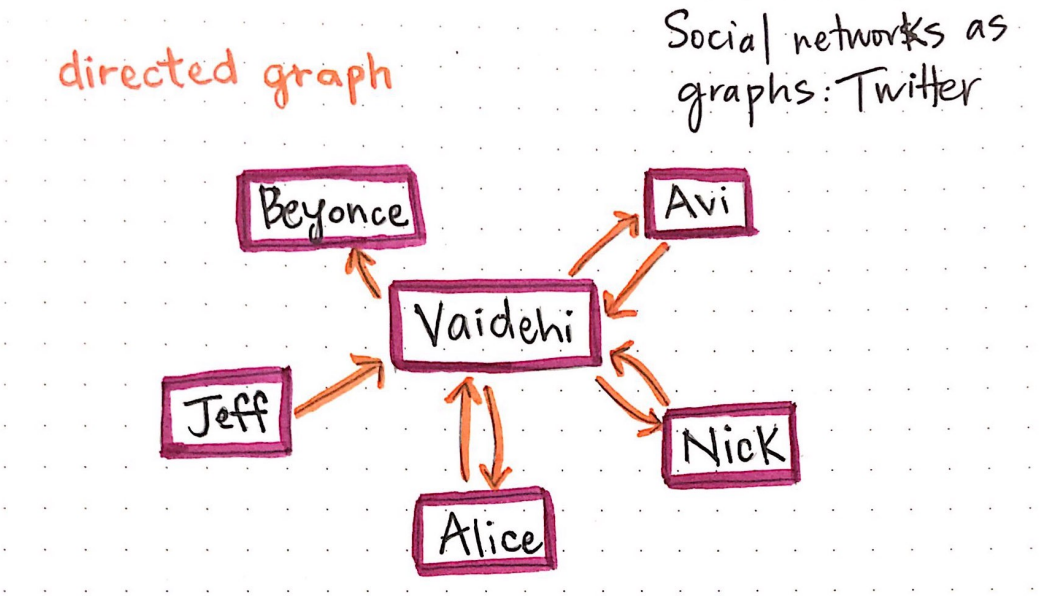

Figure 2:

accepts a node from the random walk as a one-hot vector as an input and maximizes the probability for predicting neighbor nodes. It is typically trained to predict around 20 neighbor nodes - 10 nodes left and 10 nodes right. 3. **Computing embeddings**: Embedding is the output of a hidden layer of the network. The DeepWalk computes embedding for each node in the graph.

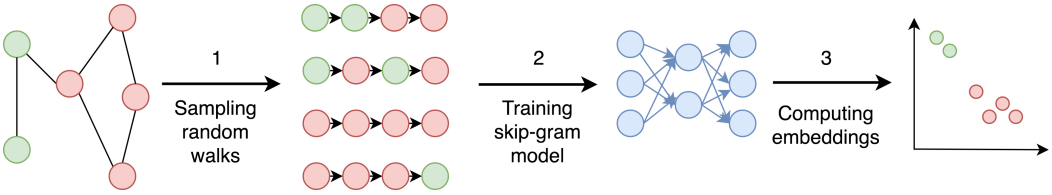

Figure 3:

DeepWalk method performs random walks randomly what means that embeddings do not preserve the local neighborhood of the node well. Node2vec approach fixes that.

Node2vec is a modification of DeepWalk with the small difference in random walks. It has parameters P and Q. Parameter Q defines how probable is that the random walk would discover the undiscovered part of the graph, while parameter P defines how probable is that the random walk would return to the previous node. The parameter P control discovery of the microscopic view around the node. The parameter Q controls the discovery of the larger neighborhood. It infers communities and complex dependencies.

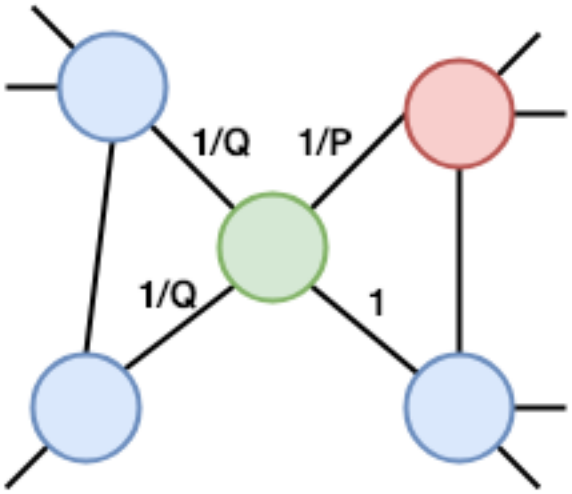

Figure 4:

Structural Deep Network Embedding (**SDNE**) is designed so that **embeddings preserve the first and the second order proximity**. The first-order proximity is the local pairwise similarity between nodes linked by edges. It characterizes the **local network structure**. Two nodes in the network are similar if they are connected with the edge. When one paper cites other paper, it means that they address similar topics. The second-order

proximity indicates the similarity of the nodes' neighborhood structures. It captures the **global network structure**. If two nodes share many neighbors, they tend to be similar.

Authors present an autoencoder neural network which has two parts. **Autoencoders** (left and right network) accept node adjacency vector and are trained to reconstruct node adjacency. These autoencoders are named vanilla autoencoders and they **learn the second order proximity**. Adjacency vector (a row from adjacency matrix) has positive values on places that indicate nodes connected to the selected node.

There is also a **supervised part of the network** — the link between the left and the right wing. It **computes the distance between embedding from the left and the right part** and includes it in the common loss of the network. The network is trained such that left and right autoencoder get all pairs of nodes which are connected by edges on the input. The loss of a distance part helps in preserving the first order proximity.

The total loss of the network is calculated as a sum of the losses from the left and the right autoencoder combined with the loss from the middle part.

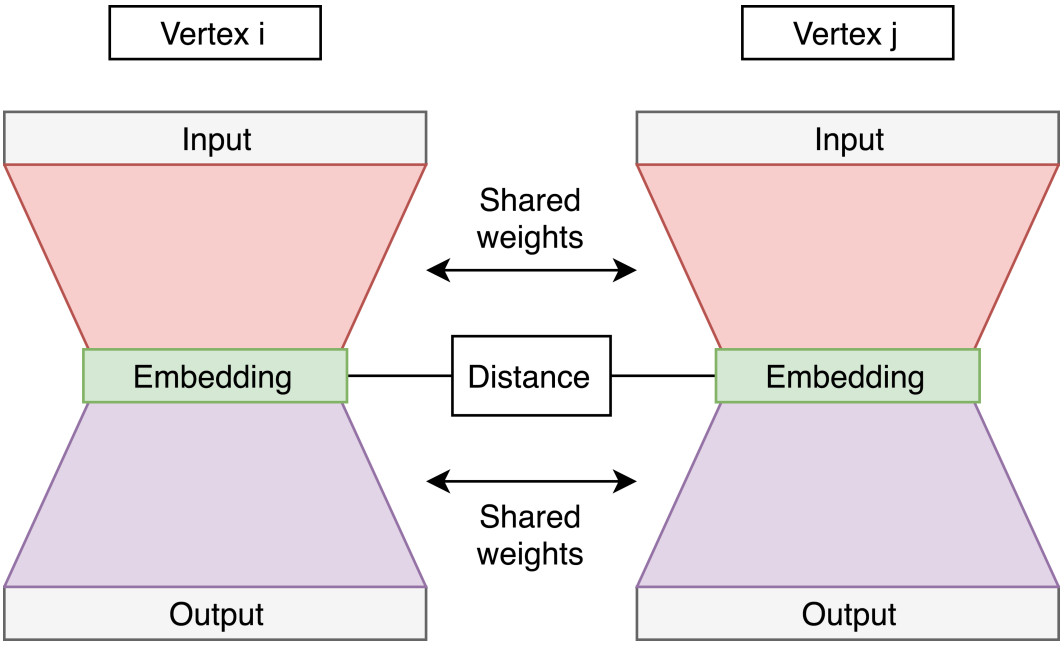

Figure 5:

Graph Embeddings — The Summary

Non-Euclidean Spaces and Data

The vast majority of deep learning is performed on Euclidean data (i.e. Images, text, audio, etc. . .) and the core object is the vector, or, alternatively, its multidimensional generalization, the tensor. **Non-euclidean data can represent more complex items and concepts** with more accuracy than 1D or 2D representation (i.e. graphs or manifolds). This means

we are working with an euclidean space with all of its inherent properties, among the **most critical of which is its flatness**.

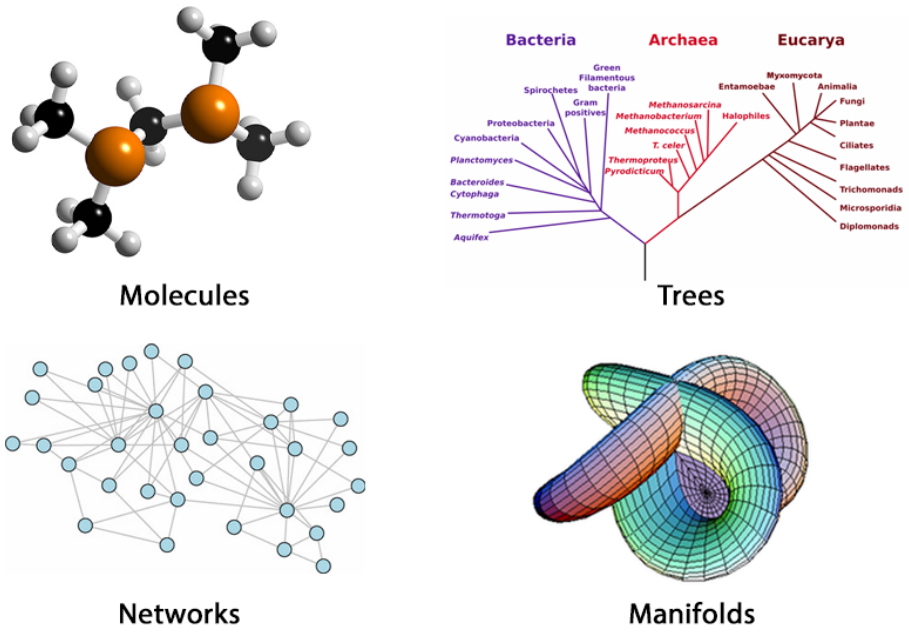

Figure 6:

As described by *Bronstein et al.* in their seminal description of the geometric deep learning paradigm, "many scientific fields study data with an underlying structure that is a non-Euclidean space.". Some reasons to abandon vector spaces: - **Better representations**: Euclidean space simply doesn't fit many types of data structures that we often need to work with. More flexible spaces - like mixed-curvature products - provide enough expressivity for other, less regular, types of data. - **Unlocking the full potential of models**: there are a proliferation of increasingly sophisticated models coming out daily. We argue that the space the points live in can be a limit to the performance of these models, and that if we lift them (both the space and functions) to the non-Euclidean versions, in certain cases we can **break through these limits**. - **More flexible operations**: The flatness of Euclidean space means that certain operations require a large number of dimensions and complexity to perform—while non-Euclidean spaces can potentially perform these operations in a more flexible way, with fewer dimensions.

So, instead of using euclidean space embeddings such as node2vec, which maps a graph in a n-dimensional tensor, we can use hyperbolic space embeddings, which are more suited to model hierarchical data. (*On the WordNet hypernym graph reconstruction, the embedding of trees into a Poincare disk obtains a nearly perfect mean average precision (MAP) of 0.989 using* ***just 2 dimensions****. The best published numbers for WordNet in Nickel & Kiela (2017) range between 0.823 and 0.87 for 5 to 200 dimensions*).

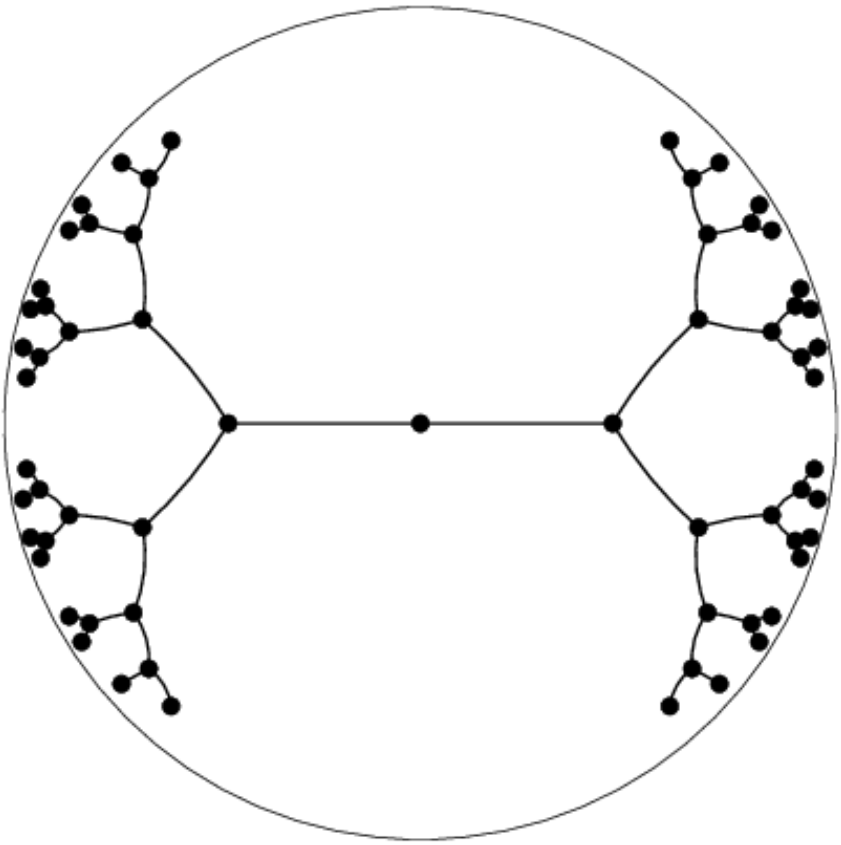

Figure 7:

Hyperbolic space embedding

Hyperbolic embeddings have captured the attention of the machine learning community through two exciting recent proposals [1] [2]. The motivation is to combine structural information with continuous representations suitable for machine learning methods. The big goal when embedding a space into another is to **preserve distances and more complex relationships**. It turns out that hyperbolic space can better embed graphs (particularly hierarchical graphs like trees) than is possible in Euclidean space.

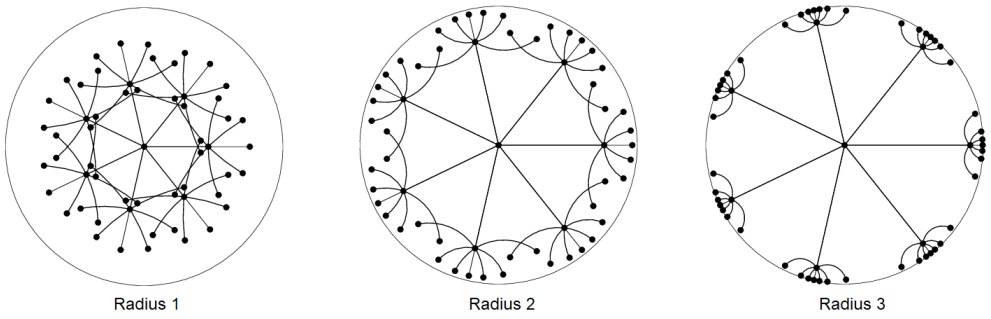

Figure 8:

In a recent NeurIPS paper on hyperbolic graph convolutional networks (HGCN), non-Euclidean ideas are used in a popular family of neural networks, graph neural networks (GNNs). This work develops practical techniques that can improve the predictive performance of recently-developed graph convolutional networks (GCNs) using ideas from hyperbolic geometry. These new hyperbolic versions of GNNs work well when the underlying graph is more hyperbolic - reaching state-of-the-art on PubMed!

GNNs not only embed structure, but also preserve semantic information that might come with the graph in the form of node features. Most GNNs models are based on a message passing algorithm, where nodes aggregate information from their neighbors at each layer in the GNN network. Generalizing message passing algorithms to non-Euclidean geometry is a challenge: it's done by using the tangent space. With the aggregation being performed at the local tangent space of each point, it better approximates the local hyperbolic geometry - giving our performance boost when using hyperbolic-like datasets.

Hyperbolic Embeddings with a Hopefully Right Amount of Hyperbole
Into the Wild: Machine Learning In Non-Euclidean Spaces

Geometric Deep Learning (GDL)

Geometric Deep Learning is significant because it allows us to **take advantage of data with inherent relationships, connections, and shared properties**.

- In traditional Deep Learning, **dimensionality is directly correlated with the number of features in the data** whereas in Geometric Deep Learning, it refers to the **type of the data itself, not the number of features it has**.

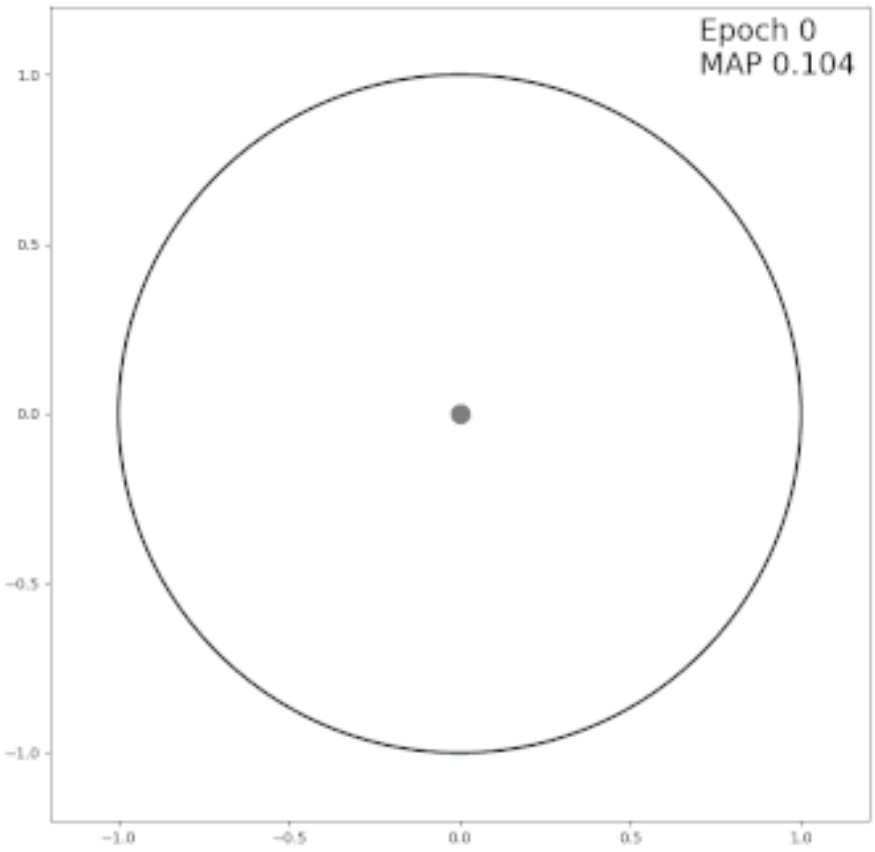

Figure 9:

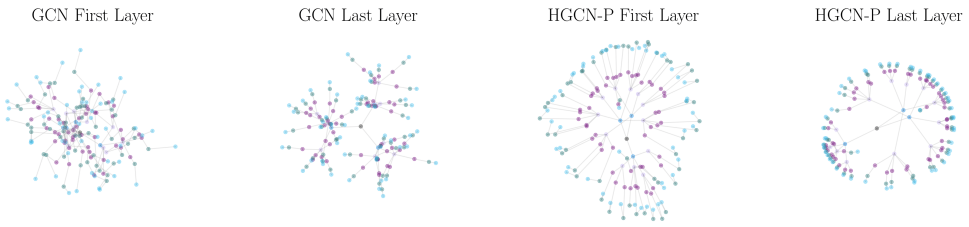

Figure 10:

- The Euclidean domain and non-euclidean domain have different rules that are followed; **data in each domain specializes in certain formats** (image, text vs graphs, manifolds) and convey differing amounts of information

  "Graph convolutional networks are the best thing since sliced bread because they allow algos to analyze information in its native form rather than requiring an arbitrary representation of that same information in lower dimensional space which destroys the relationship between the data samples thus negating your conclusions." - Graham Ganssle, Head of Data Science at Expero

What is Geometric Deep Learning?

Convolutional Neural Networks

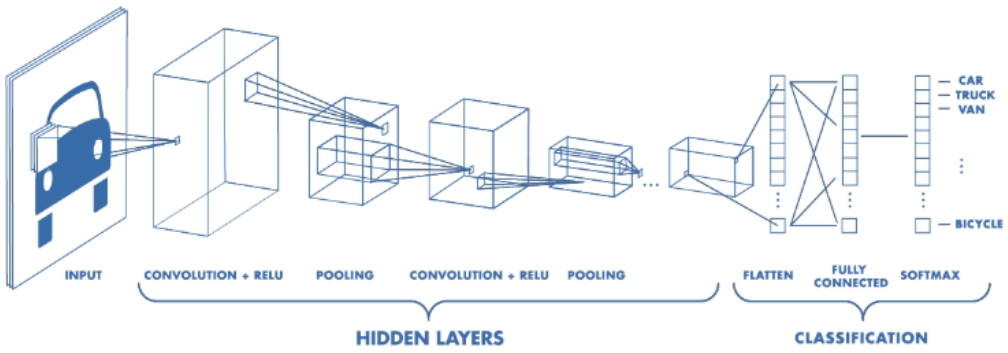

Figure 11:

Per sfruttare al meglio la correlazione spaziale delle informazioni contenute nelle immagini il modello più classico di rete neurale, il Multilayer Perceptron (MLP) è stato abbandonato in favore delle CNN. Questo per diversi motivi: il quantitativo di *pesi* di un immagine diventa velocemente ingestibile al crescere della risoluzione dell'immagine (i.e. un immagine 224 x 224 x 3 ha circa 150000 pesi); un altro problema è che l'MLP reagisce in modo differente se ha in input un immagine o una sua versione traslata: l'**MLP non è translation invariant**.

Nelle CNN l'approccio è diverso: per analizzare l'influenza di (ad esempio) pixel vicini fra loro si usano dei filtri, di una dimensione specificata dall'utente, e si spostano su tutta l'immagine. Per ogni punto dell'immagine viene effettuata un'operazione di convoluzione con il filtro.

Ogni filtro sostanzialmente può essere considerato un *feature identifier*, in quanto viene attivato da una funzione ($ReLU, tanh, ELU$, etc...) quando l'immagine presa dal filtro corrisponde alla feature cercata. Di seguito, la visualizzazione di alcuni filtri:

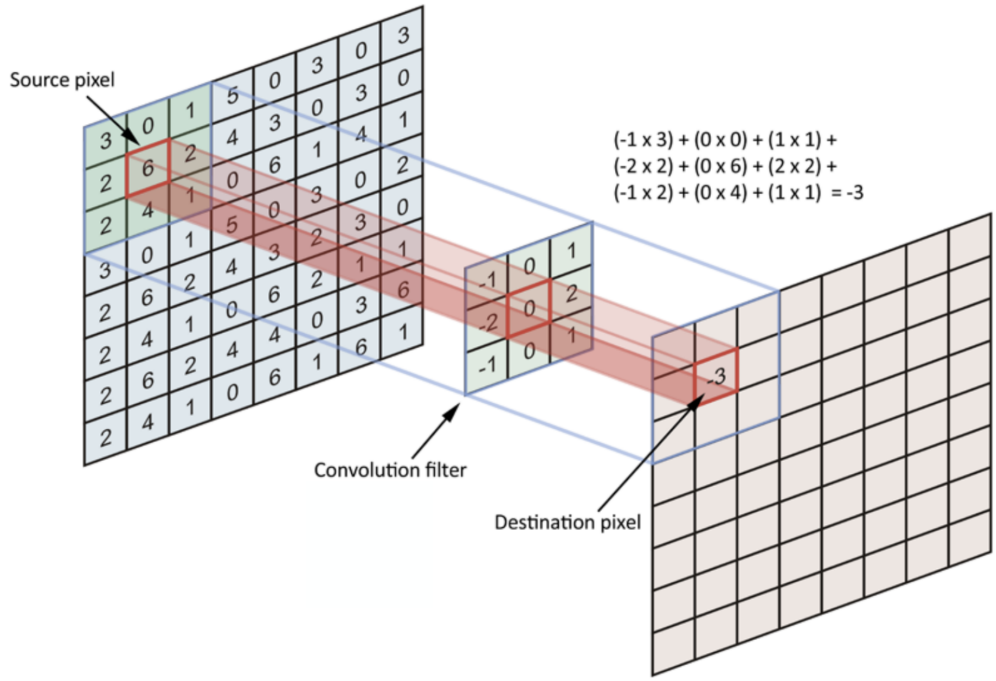

Figure 12:

Un primo livello convolutivo può ad esempio aiutare a identificare feature molto semplici (i.e. curve e linee rette, forme geometriche), un successivo livello può identificare feature più complesse (i.e. occhi, bocca, naso etc...) e altri livelli ancora possono rilevare dei volti completi:

Dopo aver effettuato questa operazione con tutti i filtri (il cui numero è deciso anch'esso dall'utente) il risultato è una **feature map**. I filtri vengono aggiornati durante il training della rete.

Nel passare i filtri sull'immagine per mantenere costanti le dimensioni delle feature maps si utilizzano diverse strategie di *padding*:

Successivamente, tra i vari livelli convoluzionali, esistono dei livelli di **Pooling**, volti a ridurre il numero di parametri e di computazione nella rete, riducendone le dimensioni spaziali. In questi livelli si possono svolgere diverse operazioni, sebbene nella maggior parte dei casi si utilizzano dei livelli di **Max-pooling**, che prendono solo il valore massimo da un filtro, che attraverso un meccanismo di sliding windows viene applicato su tutte le porzioni dell'immagine.

Simple Introduction to Convolutional Neural Networks
Basics of the Classic CNN
Basic Overview of Convolutional Neural Network (CNN)

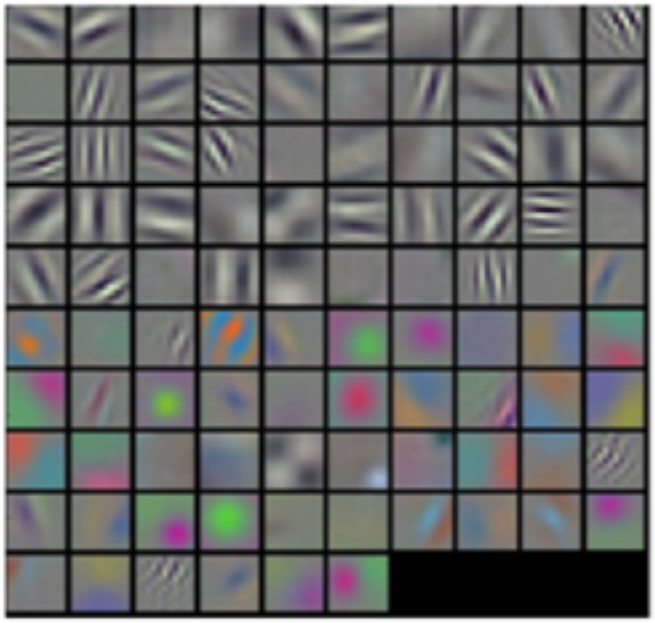

Figure 13:

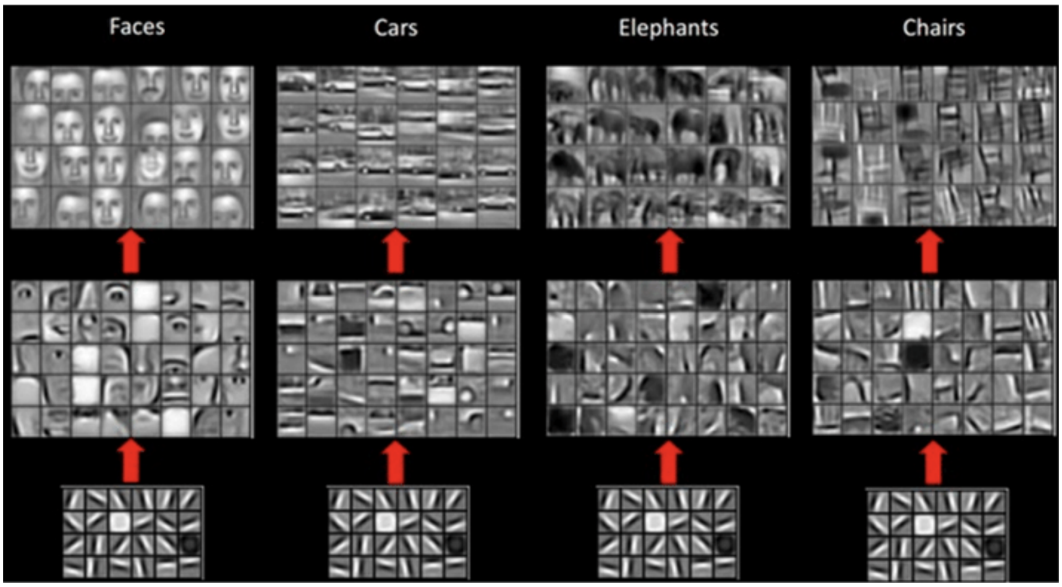

Figure 14:

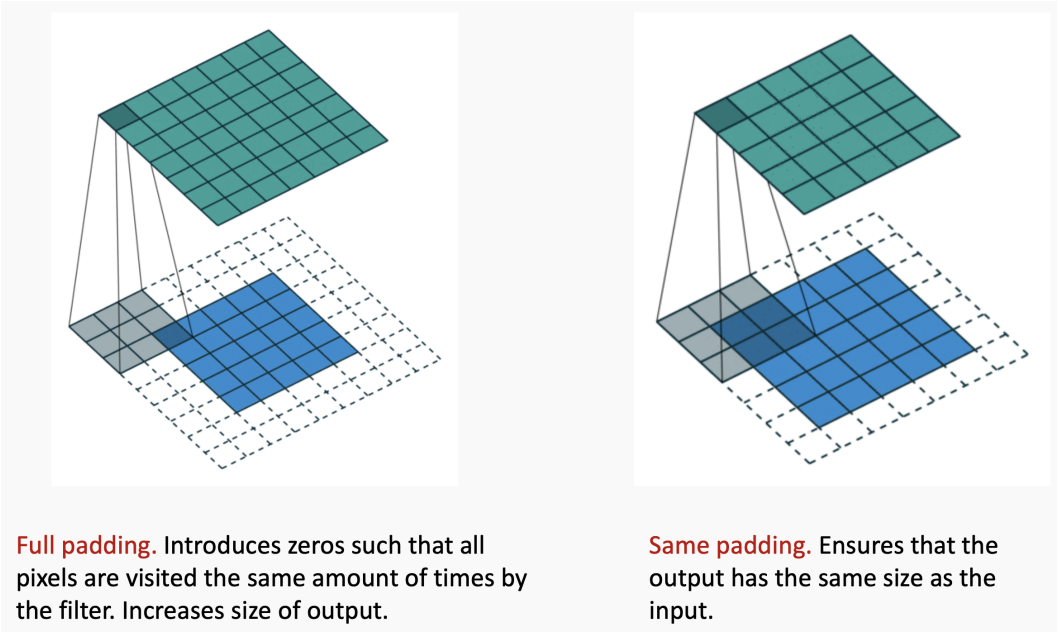

Figure 15:

Graph Neural Networks

Graph neural networks (**GNNs**) **are deep learning based methods that operate on graph domain**.

Based on CNNs and graph embedding, graph neural networks (GNNs) are proposed to **collectively aggregate information from graph structure**. Thus they can model input and/or output consisting of elements and their dependency.

Firstly, the standard neural networks like CNNs and RNNs cannot handle the graph input properly in that they stackthe feature of nodes by a specific order. However, there isn't a natural order of nodes in the graph. To present a graph completely, we should traverse all the possible orders as the input of the model like CNNs and RNNs, which is very redundant when computing. To solve this problem, **GNNs propagate on each node respectively, ignoring the input order of nodes**. In other words, **the output of GNNs is invariant for the input order of nodes**. Secondly, an edge in a graph represents the information of dependency between two nodes. In the standard neural networks, the dependency information is just regarded as the feature of nodes. However, GNNs can do propagation guided by the graph structure instead of using it as part of features. Generally, **GNNs update the hidden state of nodes by a weighted sum of the states of their neighborhood**.

Graph Convolutional Network

Il punto di forza delle CNN e delle RNN è la loro capacità di saper sfruttare al meglio la conoscenza delle interconnessioni fra i dati in input. Ad esempio un filtro convolutivo si basa sul fatto che i dati necessari ad elaborare un singolo pixel (per estrarne una feature) si trovino nei pixel a lui vicini (tipicamente, a due o tre pixel di distanza), mentre i pixel più distanti possano essere ignorati. Da qui si può evidenziare come dietro questo concetto vi sia un grafo, in quanto la *vicinanza* dei pixel equivale a rappresentare l'immagine come un grafo dalla struttura perfettamente regolare:

Wolfram MathWorld

Sfruttare questa informazione di vicinanza tra i pixel è il cuore di una rete convolutiva, sebbene i pixel in un'immagine sono interconnessi in modo estremamente regolare. C'è modo per sfruttare l'informazione contenuta nel grafo senza senza sacrificare efficienza o flessibilità delle architetture nel caso di grafi irregolari (i.e. con nodi quasi isolati, alcuni centrali etc. . .)?

Definito un grafo di N nodi, su ciascuno dei quali è definito un segnale $x_n \in R^C$ (dove C è il numero di "canali" del signale del segnale, i.e. 3 colori per un pixel). Denotata con X la matrice $N \times C$ che colleziona su ogni riga il segnale definito sul rispettivo nodo. Infine, A sarà la matrice $N \times N$ di adiacenza.

Per comprendere il funzionamento delle **GCN**, ignoriamo per un attimo le connessioni fra i vari nodi. In questo caso, una rete neurale "classica" che operasse su ciascun nodo si comporrebbe di strati del tipo:

$$g(X) = ReLU(XW)$$

dove W è la matrice di pesi dello strato e $ReLU(s) = max(0, s)$ è la nostra funzione di attivazione che aggiunge non-linearità ($ReLU$ solo come esempio, andrebbe bene una qualsiasi funzione di attivazione). Questo schema processa ogni nodo in maniera **indipendente**, ignorando completamente le connessioni fra di essi e di conseguenza una quantità potenzialmente molto importante di informazione.

Consideriamo però questa semplice variante:

$$g(X) = ReLU(\mathbf{A}XW)$$

dove A è la matrice di adiacenza introdotta prima. La **logica è simile a quella di una rete convolutiva**: l'informazione elaborata da ciascun "filtro" dipende solo dagli immediati vicini del nodo, mentre le stesse operazioni (definite dalla matrice W) vengono applicate su tutti i nodi del grafo in simultanea.
GEOMETRIC DEEP LEARNING: GRAPH CONVOLUTIONAL NETWORK. IAML

TODO: Qui c'è da esplorare la GRAPH CONVOLUTION E GRAPH FOURIER TRANSFORM