# The graph neural network model
(2009)

Scarselli, Franco Gori, Marco Hagenbuchner, Markus Monfardini, Gabriele
**Resume**

December 17, 2018

---

**Abstract**

Many relationships among data in several areas, such as computer vision and molecular biology, can be represented in terms of graphs. In this paper, they propose a new neural network, called graph neural network, that extends existing NN methods for processing the data prepresented in graph domains.

## 1 Introduction

In machine learning, structured data is often associated with the goad of learning from examples a function $\tau$ that maps a graph $G$ and one of its nodes $n$ to a vector if reals (for regression) or a vector of integers (for classification) $\tau(\boldsymbol{G}, n) \in \mathbb{R}^m$.

In the application of graphical domain, we have two classes:

- graph focused : the function $\tau$ is independent of the node $n$ and implements a classifier or a regressor on a graph structured dataset (for a graph of a chemical component, the mapping $\tau(\boldsymbol{G})$ estimate the propbability of the component causing a given disease)
- node focused: $\tau$ depends on the node $n$, so that the classification of regression depends on the properties of each node, Object detection is an example of this class of applications.

Traditionnal machine learning applications cope with graph structured data using a preprocessing step, which maps graph structured data to a simpler representation, such as vectors of reals, this squashing of the original representation may result in the loss of important and relevent information, like the topological dependency.

In this paper, they present a supervised neural network model, which is suitable for both graoh and node focused applications, GNN is an extension of both recursive neural networks and random walks.

GNNs are based on an information diffusion mechanism. A graph is processed by a set of units, each one corresponding to a node of the graph, which are linked according to the graph connectivity. The units update their states and exchange information until they reach a stable equilibrium. The output of a GNN is then computed locally at each node on the base of the unit state.

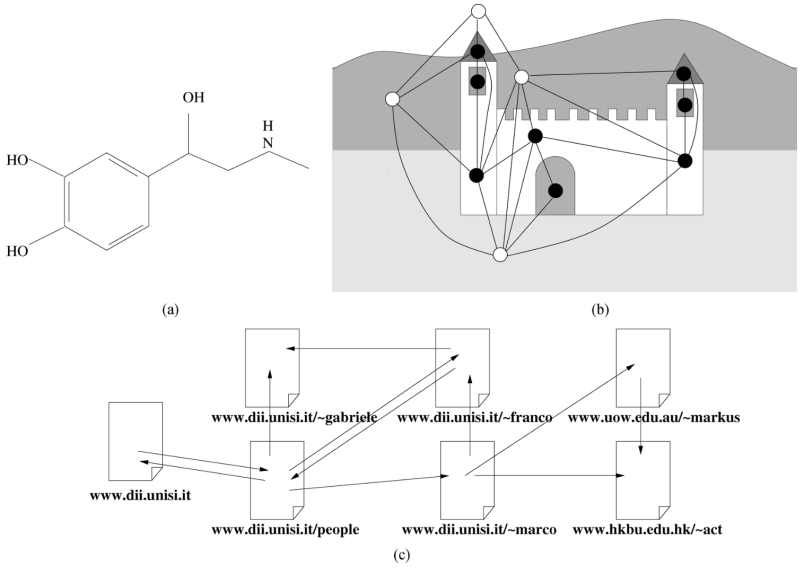


Figure 1: Example of graphs, (a) can be considered as graph focused, and (b) as node focused

# 2 GNN : Graph Neural Network

## 2.1 Notations

- A graph $\boldsymbol{G}$ is a pair $(\boldsymbol{N}, \boldsymbol{E})$, where $\boldsymbol{N}$ is the set of nodes and $\boldsymbol{E}$ is the set of edges.
- The labels attached to node $n$ and edge $(n_1, n_2)$ will be represented by $\boldsymbol{l}_n \in \mathbb{R}^{l_N}$ and $\boldsymbol{l}_{(n_1,n_2)} \in \mathbb{R}^{l_E}$, respectively.
- $\boldsymbol{l}$ is the vector obtained by stacking together all the labels of the graph.
- $\boldsymbol{l}_{\text{ne}[n]}$ is the vector containing the labels of all the neighbors of $n$.
- $\boldsymbol{l}_{\text{co}[n]}$ is the vector containing the edges that have $n$ as a vertex.
- For positionnal graphs, a unique integer identifier is assigned to each neighbors of a node $n$ to indicate its logical position, for each node $n$, there exists $\nu_n : \text{ne}[n] \rightarrow \{1, \dots |\text{N}|\}$ which assigns to each neighbor $u$ of $n$ a position $\nu_n(u)$.
- $\mathcal{D}$ is a set of pairs of a graph and a node $n$, $\mathcal{D} = \mathcal{G} \times \mathcal{N}$, where $\mathcal{G}$ is a set of the graphs and $\mathcal{N}$ is a subset of their nodes.
- A supervised learning framework with the learning set:
  $\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j}) \,|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G} \;\; n_{i,j} \in \boldsymbol{N}_i; \boldsymbol{t}_{i,j} \in \mathbb{R}^m, 1 \leq i \leq p, 1 \leq j \leq q_i\}$,
  Where $n_{i,j} \in N_i$ denotes the $j$th node in the set and $\boldsymbol{t}_{i,j}$ is the desired target associated to $n_{i,j}$. Finally, $p \leq |\mathcal{G}|$ and $q_i \leq |\boldsymbol{N}_i|$.

## 2.2 The model

The intuitive idea underlining the proposed approach is that nodes in a graph represent objects or concepts, and edges represent their relationships. Each concept is naturally defined by its features and the related concepts. Thus, we can attach a state $\boldsymbol{x}_n \in \mathbb{R}^s$ to each node $n$ that is based on the information contained in the neighborhood of $n$, and can be used to produce the output.

- Let $f_{\boldsymbol{\omega}}$ be a parametric function, called local transition function, that expresses the dependence of a node $n$ on its neighborhood :

$$\boldsymbol{x}_n = f_{\boldsymbol{w}} \left( \boldsymbol{l}_n, \boldsymbol{l}_{\text{co}[n]}, \boldsymbol{x}_{\text{ne}[n]}, \boldsymbol{l}_{\text{ne}[n]} \right)$$

- Let $g_{\boldsymbol{\omega}}$ be the local output function that describes how the output is produced :

$$\boldsymbol{o}_n = g_{\boldsymbol{w}}\left(\boldsymbol{x}_n, \boldsymbol{l}_n\right)$$

- Let $\boldsymbol{x}, \boldsymbol{O}, \boldsymbol{l}$ and $\boldsymbol{l_N}$ be the vectors constructed by stacking all the states, all the outputs, all the labels, and all the node labels, respectively. Then, the above equations can be rewritten in a compact form as

$$\boldsymbol{x} = F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l})$$
$$\boldsymbol{o} = G_{\boldsymbol{w}}\left(\boldsymbol{x}, \boldsymbol{l}_N\right)$$

**Notes**

- When dealing with directed graphs, the function $f_{\boldsymbol{w}}$ can also accept as input a representation of the direction of the arcs. For example, $f_{\boldsymbol{w}}$ may take as input a variable $d_l$ for each arc $l \in \mathrm{co}[n]$ such that $d_l = 1$, if is directed towards $n$ and 0, if comes from $n$.
- In general, the transition and the output functions and their parameters may depend on the node. In fact, it is plausible that different mechanisms (implementations) are used to represent different kinds of objects. In this case, each kind of nodes $k_n$ has its own transition function $fk_n$.
- We are interested in the case when $\boldsymbol{x}, \boldsymbol{o}$ are uniquely defined and $\boldsymbol{o} = G_{\boldsymbol{w}}\left(\boldsymbol{x}, \boldsymbol{l}_N\right)$ defines a map, which takes a graph as input and returns an output for each node. According to Banachs theorem , $\boldsymbol{o} = G_{\boldsymbol{w}}\left(\boldsymbol{x}, \boldsymbol{l}_N\right)$ has a unique solution provided that $F_{\boldsymbol{w}}$ is a contraction map with respect to the state,

  i.e., there exists $\mu, 0 \leq \mu < 1$, such that holds $\|F_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{l}\|) - F_{\boldsymbol{w}}(\boldsymbol{y}, \boldsymbol{l})\| \leq \mu \|\boldsymbol{x} - \boldsymbol{y}\|$ for any $\boldsymbol{x}, \boldsymbol{y}$.

**Implementation** In order to implement the GNN model, the following items must be provided:

1. a method to solve $\boldsymbol{o}_n = g_{\boldsymbol{w}}\left(\boldsymbol{x}_n, \boldsymbol{l}_n\right)$.
2. a learning algorithm to adapt $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$ using examples from the training data set
3. an implementation of $f_{\boldsymbol{w}}$ and $g_{\boldsymbol{w}}$.

**Computation** Banachs fixed point theorem suggests the following classic iterative scheme for computing the state:

$$x(t+1) = F_w(x(t), l)$$

where $x(t+1)$ denotes the th iteration of $x$. The dynamical system converges exponentially fast to the solution of $\boldsymbol{o} = G_{\boldsymbol{w}}\left(\boldsymbol{x}, \boldsymbol{l}_N\right)$ for any initial value. We can, therefore, think of as the state that is updated by the transition function. Thus, the outputs and the states can be computed by iterating:

$$\boldsymbol{x}_n(t+1) = f_{\boldsymbol{w}}\left(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}, \boldsymbol{l}_{\mathrm{ne}[n]}\right)$$
$$\boldsymbol{o}_n(t) = g_{\boldsymbol{w}}\left(\boldsymbol{x}_n(t), \boldsymbol{l}_n\right), \quad n \in \boldsymbol{N}$$

**The Learning Algorithm** Learning in GNNs consists of estimating the parameter $\boldsymbol{w}$ such that $\varphi_{\boldsymbol{w}}$ approximates the data in the learning data set:

$$\mathcal{L} = \{(\boldsymbol{G}_i, n_{i,j}, \boldsymbol{t}_{i,j}) \,|, \boldsymbol{G}_i = (\boldsymbol{N}_i, \boldsymbol{E}_i) \in \mathcal{G}\ n_{i,j} \in \boldsymbol{N}_i; \boldsymbol{t}_{i,j} \in \mathbb{R}^m, 1 \leq i \leq p, 1 \leq j \leq q_i\}$$

The learning task can be posed as the minimization of a quadratic cost function

$$e_{\boldsymbol{w}} = \sum_{i=1}^{p} \sum_{j=1}^{q_i} \left(\boldsymbol{t}_{i,j} - \varphi_{\boldsymbol{w}}\left(\boldsymbol{G}_i, n_{i,j}\right)\right)^2$$

The learning algorithm is based on a gradient-descent strategy and is composed of the following steps:

1. The states are iteratively updated by $\boldsymbol{x}_n(t+1) = f_{\boldsymbol{w}}\left(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}(t), \boldsymbol{l}, \boldsymbol{l}_{\mathrm{ne}[n]}\right)$ until at time $T$ they approach the fixed point solution of $\boldsymbol{o} = G_{\boldsymbol{w}}\left(\boldsymbol{x}, \boldsymbol{l}_N\right)$: $x(T) \approx x$

2. The gradient $\partial e_{\boldsymbol{w}}(T)/\partial \boldsymbol{w}$ is computed

3. The weights $\boldsymbol{\omega}$ are updated according to the gradient computed in step b)

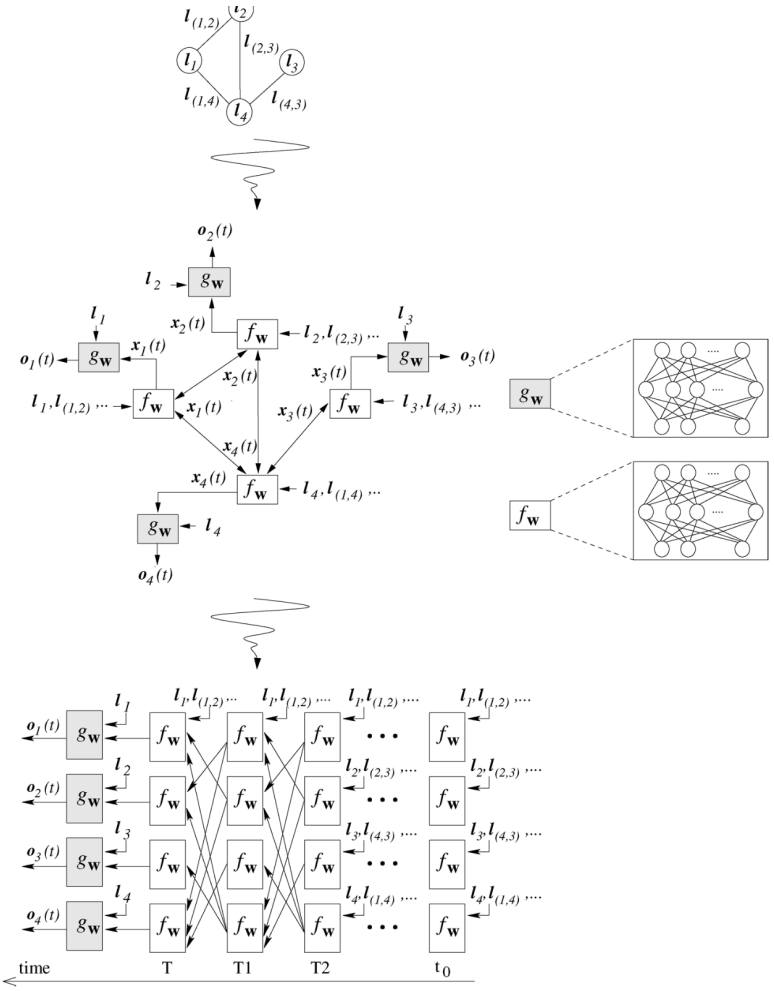


Figure 2: Graph (on the top), the corresponding encoding network (in the middle), and the network obtained by unfolding the encoding network (at the bottom). The nodes (the circles) of the graph are replaced, in the encoding network, by units computing $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ (the squares). When $F_{\boldsymbol{w}}$ and $G_{\boldsymbol{w}}$ are implemented by feedforward neural networks, the encoding network is a recurrent neural network. In the unfolding network, each layer corresponds to a time instant and contains a copy of all the units of the encoding network. Connections between layers depend on encoding network connectivity