Readout Layer

Architecture

Contents

[I. Abstract 3](#_Toc150604548)

[II. Introduction 4](#_Toc150604549)

[III. Definition and Notation 6](#_Toc150604550)

[IV. Neural Networks for graphs 7](#_Toc150604551)

[1. Graph Convolution 7](#_Toc150604552)

# Abstract

Several machine learning problems can be naturally defined over graph data. Recently, many researchers have been focusing on the definition of neural networks for graphs. The core idea is to learn a hidden representation for the graph vertices, with convolutive or recurrent mechanism. When considering discriminative tasks on graphs such as classification or regression, one critical component to design is the readout function, i.e., **the mapping from the set of vertex representations to a fixed-size vector** (or the output). Different approaches tend to be complex, making the training of the whole network harder. In this paper, we frame the problem in the setting of learning over sets. Adopting recently proposed theorems over functions defines on sets, we propose a simple but powerful formulation for a readout layer that can encode or approximate arbitrarily well any continuous permutation-invariant function over sets. Experimental results on real-world graph datasets show that, compared to other approaches, the proposed readout architecture can improve the predictive performance of Graph Neural Networks while being computationally more efficient.

# Introduction

In the last years, there has been an increasing interest in learning from structured data, especially graphs. Graphs are a neural way to describe relationships between entities such as atoms in a chemical compound or proteins in protein-protein interaction networks. When dealing with learning tasks on graphs, a classical and well-known approach is to use graph kernels that couple with kernel machines, can be employed to build classification and regression models. A disadvantage of using graph kernels-based methods is that their performances strongly depend on the feature space induced by the kernel function, that is defined in advance, i.e., the feature space is not shaped by a learning process.

Even though some work has been done in the direction of learning the representation combining different graph kernels, neural-based approaches are receiving an increasing attention. Neural networks have emerged as the leading methods in different domains, from image recognition to natural language processing. Most existing neural networks take input data in tensor form whose elements are organized in a fixed order. When it comes to graphs such order is a problem, since different representations of the same graph, obtained by any permutation of the order of vertices of the graph, are admissible. This is the well-known graph isomorphism problem, i.e., to recognize when 2 graph representations that are syntactically different are in fact different representations of the same graph. Thus, forcing an order for the vertices of a graph representation does not guarantee that different representations of the same graph can be recognized as such by a neural network. The lack of this property requires the definition of novel neural network architectures, which allow to extract features which are invariant with respect to the order of presentation of the vertices of a graph.

Concerning neural networks for graphs, many generative and neural methods have been recently proposed. The different networks are in general composed of 2 main processing steps.

* Vertex relabeling --- in this step, a recurrent or convolutive architecture is exploited to learn new representations for graph vertices, based on the local neighborhood.
* Readout --- this step computes the mapping from the set of vertex representations to the output.

Despite the existence of different proposals for both the aforementioned processing steps, the predictive performances of graph neural networks are still below kernel methods, at least on some benchmark datasets, so there is a need to propose more effective approaches. In this paper, we focus on the second processing step, i.e., on the definition of readout functions.

Traditionally, the readout is implemented by simple average or sum operations over vertices representations, or the representation of a single vertex is selected according to some criteria (a-priori), while representations of the remaining vertices are discarded. Note that it is not generally recognized that the readout is in fact an operator defined over sets. This consideration emerged only recently, where a readout architecture based on an instance of Neural Turing Machines is proposed. This readout is definitely more expressive than earlier proposals, but it is hard and slow to train (it adopts LSTM, so it is difficult to parallelize). Another approach was proposed more recently in 2018, referred to as SortPooling. The idea is to sort the vertex representations computed by the final graph convolution layer. Then the representations are concatenated (according to their ordering), trimming the last part or adding padding up to a fixed dimension, defined beforehand. We note that this method is also hard to train, since very time the vertex representations change in such a way that the vertex ordering changes, the subsequent layers of the network become meaningless, and the weights have to be re-adapted. Moreover, the ordering phase makes the complexity of this readout super-linear.

In this paper, we build on recent results from “Deep Sets, NIPS, 2017”, proposing a general formation for a readout function that can, in principle, express any (under reasonable assumptions) function defined over sets. Our method does not require the input set to be ordered, thus it has a linear-time complexity. Experimental results on real-world datasets show the performance of our proposed readout operator is competitive with respect to state of the art readout functions for graphs.

# Definition and Notation

We denote matrices with bold uppercase letters, vectors with uppercase letters, and variables with lowercase letters. Given a matrix **M**, *Mi* denotes the *i*-th row of the matrix, and *mij* is the element in *i*-th row and *j*-th column. Given the vector V, *vi* refers to its *i*-th element.

Let’s consider G=(VG, EG, **X**G) as a graph, where:

* VG = {*v1*, …, *vn*} --- is the set of vertices
* --- is the set of edges
* --- is the vertex label matrix, where each row is the label (a vector of size d) associated to each vertex , i.e., .

Note that, in this paper, we will not consider edge labels. When the reference to the graph *G* is clear from the context, for the sake of notation we discard the superscript referring to the specific graph. We define the adjacency matrix as , **0** otherwise. We also define the neighborhood of a vertex *v* as the set of vertices connected to *v* by an edge, i.e., **.** Note that *N(v)* is also a set of vertices at shortest path distance exactly one from *v*, i.e., , where *sp* is a function computing the shortest-path distance between 2 vertices in a graph.

# Neural Networks for graphs

The first definition of neural network for graphs has been proposed in 1997. More recent models have been proposed in 2009. Both works are based on an idea that has been re-branded later as graph convolution or neural message passing. The idea is to define the neural architecture following the topology of the graph. Then a transformation is performed from the neurons corresponding to a vertex and its neighborhood to a new hidden representation, that is associated to the same vertex (possibly in another layer of the network). This transformation depends on some parameters, that are shared among all the vertices. Similarly, to convolutional networks for images, graph convolutions can possibly be alternated with graph pooling layers. In “Hierarchical Graph Representation Learning with Differentiable Pooling --- NIPS, 2018) a different graph pooling or (coarsening) operator is defined; however, this operator sacrifices the sparsity of input graphs, generating weighted fully connected graphs as output of the pooling operation. This idea has been improved in “Towards Sparse Hierarchical Graph Classifier --- Relational Representation Learning Workshop, 2018”, where a simpler approach selecting a subset of nodes is introduced. After a certain number of transformations, a readout layer transforms the graph and its vertex representations to a fixed-size vector, from which a fully connected layer can be attached to compute the output. In the following, we revise these 2 basic components of graph neural networks.

## Graph Convolution

In the following, for the sake of simplicity we ignore the bias terms. In “The Graph Neural Network Model --- IEEE Transactions on Neural Networks, 2009”, when considering non-positional graphs, i.e., the most common definition, and the one considered in this parper, a transition function on a graph vertex v at time is defined as:

Where:

* --- is a parametric function whose parameters have to be learned (e.g., neural network) and are shared among all the vertices. Note that, if edge labels are available, they can be included in the equation.

In fact, in the original formulation, depends also on the label of the edge between v and u. This transition function is part of a recurrent system. It is defined as a contraction mapping; thus, the system is guaranteed to converge to a fixed point, i.e., a representation, that does not depend on the particular initialization of the weight matrix **H**O. The final representation for each vertex is computed form the last representation and the original vertex labels as follows:

Where is another neural network. From this representation, a graph readout function can be applied to generate a representation for the whole graph. In “*Gated Graph Sequence Neural Networks --- 2016*” the work in “*The Graph Neural Network Model --- 2009*) is extended, removing the constraint for the recurrent system to be a contraction mapping, and replacing the recurrent units with GRUs. As a side node, recently it has been shown in “*An Experimental Study of Neural Networks for Variable Graphs --- 2018.*” That stacked graph convolutions are superior to graph recurrent architectures in terms of both accuracy and computational cost.

In “*Neural network for graphs: A contextual constructive approach --- 2009.*”, a model referred to as Neural Network for graph (NN4G) is proposed. In the first layer, a transformation over vertex labels is computed:

Where are weights connecting the previous hidden layers to the current neuron (shared). Note that in this formulation, skip connections are present, to the (*i-*1)-th layer, from layer 1 to layer *i*.

There is an interesting recent work about the parallel between skip-connections (residual networks in that case) and recurrent networks. However, since in the formulation in above equation, every layer is connected to all subsequent layers, it is not possible to reconduct it to a (vanilla) recurrent model. Let us consider the (i+1)-th graph convolutional layer, that comprehends ci+1 graph convolutional filters. We can rewrite above equations for the whole layer as:

Where:

* *i* = 0, …, *l*-1 (and *l* is the number of layers)