Graph Transformations and Neural Networks

Ingrid Fischer
Friedrich-Alexander University Erlangen-Nürnberg, Germany

INTRODUCTION

As the beginning of the area of artificial neural networks, the introduction of the artificial neuron of McCulloch and Pitts is considered. They were inspired by the biological neuron. Since then, many new networks or new algorithms for neural networks have been invented with the result that this area is not very clearly laid out. In most textbooks on (artificial) neural networks (Rojas, 2000; Silipo, 2002), there is no general definition on what a neural net is, but rather an example-based introduction leading from the biological model to some artificial successors. Perhaps the most promising approach to define a neural network is to see it as a network of many simple processors (units), each possibly having a small amount of local memory. The units are connected by communication channels (connections) that usually carry numeric (as opposed to symbolic) data called the weight of the connection. The units operate only on their local data and on the inputs they receive via the connections. It is typical of neural networks that they have great potential for parallelism, since the computations of the components are largely independent of each other. Neural networks work best if the system modeled by them has a high tolerance to error. Therefore, one would not be advised to use a neural network to balance one's checkbook. However, they work very well for:

- capturing associations or discovering regularities within a set of patterns;
- any application where the number of variables or diversity of the data is very great;
- any application where the relationships between variables are vaguely understood; or,
- any application where the relationships are difficult to describe adequately with conventional approaches.

Neural networks are not programmed but can be trained in different ways. In supervised learning, examples are presented to an initialized net. From the input and the output of these examples, the neural net learns somehow. There are as many learning algorithms as there are types of neural nets. Also, learning is motivated physiologically. When an example is presented to a neural network that it cannot recalculate, several different steps are possible: changes can be done for a neuron, for the connection's weight or new connections, and neurons can be inserted. For unsupervised learning, the results of an input are not known.

There are many advantages and limitations to neural network analysis, and to discuss this subject properly, one must look at each individual type of network. Nevertheless, there is one specific limitation of neural networks that potential users should be aware of. Neural networks are more or less, depending on the different types, the ultimate black boxes. The final result of the learning process is a trained network that provides no equations or coefficients defining a relationship beyond its own internal mathematics.

Graphs are widely-used concepts within computer science; in nearly every field, graphs serve as a tool for visualization, summarization of dependencies, explanation of connections, and so forth. Famous examples are all kinds of different nets and graphs as semantic nets, petri nets, flow charts, interaction diagrams, or neural networks. Invented first 35 years ago, graph transformations have been expanding constantly. Wherever graphs are used, graph transformations also are applied (Blostein & Schürr, 1999; Ehrig et al., 1999a; Ehrig et al., 1999b; Rozenberg, 1997).

Graph transformations are a very promising method for modeling and programming neural networks. The graph part is automatically given, as the name *neural network* already indicates. Having graph transformations as methodology, it is easy to model algorithms on this graph structure. Structure-preserving and structure-changing algorithms can be modeled equally well. This is not the case for the widely used matrices programmed mostly in *C* or *C++*. In these approaches, modeling structure change becomes more difficult.

This leads directly to a second advantage. Graph transformations have proven useful for visualizing the network and its algorithms. Most modern neural network simulators have some kind of visualization tool. Graph transformations offer a basis for this visualization, as the algorithms are already implemented in visual rules. Also, in nearly all books, neural networks are visualized as graphs.

When having a formal methodology at hand, it is also possible to use it for proving properties of nets and algorithms. Especially in this area, earlier results for graph

transformation systems can be used. Three possibilities are especially promising: first, it is interesting whether an algorithm is terminating. Though this question is undecidable in the general case, the formal methods of graph rewriting and general rewriting offer some chances to prove termination for neural network algorithms. The same holds for the question whether the result produced by an algorithm is useful, whether the learning of a neural network was successful. Then it helps to prove whether two algorithms are equivalent. Finally, possible parallelism in algorithms can be detected and described, based on results for graph transformation systems.

BACKGROUND

A Short Introduction to Graph Transformations

Despite the different approaches to handling graph transformations, there are some properties that all approaches have in common. When transforming a graph *G* somehow, it is necessary to specify what part of the graph, what subgraph *L*, has to be exchanged. For this subgraph, a new graph *R* must be inserted. When applying such a rule to a graph *G*, three steps are necessary:

- Choose an occurrence of *L* in *G*.
- Delete *L* from *G*.
- Insert *R* into the remainder of *G*.

In Figure 1, a sample application of a graph transformation rule is shown. The left-hand side *L* consists of three nodes *(1:, 2:, 3:)* and three edges. This graph is embedded into a graph *G*. Numbers in *G* indicate how the nodes of *L* are matched. The embedding of edges is straightforward. In the next step *L* is deleted from *G*, and *R* is inserted. If *L* is simply deleted from *G*, hanging edges remain. All edges ending/starting at *1:,2:,3:* are missing one node after deletion. With the help of numbers *1:,2:,3:* in the right-hand side *R,* it is indicated how these hanging edges are attached to *R* inserted in *G/L*. The resulting graph is *H*.

Simple graphs are not enough for modeling real-world applications. Among the different extensions, two are of special interest. First, graphs and graph rules can be labeled. When *G* is labeled with numbers, *L* is labeled with variables, and *R* is labeled with terms over *L*'s variables. This way, calculations can be modeled. Taking our example and extending *G* with numbers *1,2,3*, the left-hand side *L* with variables *x,y,z* and the right-hand side with terms $x+y$, $x-y$, $x \times y$, x^y is shown in Figure 1. When *L* is embedded in *G*, the variables are set to the numbers of the corresponding nodes. The nodes in *H* are labeled with the result of the terms in *R* when the variable settings resulting from the embedding of *L* in *G* are used.

Also, application conditions can be added, restricting the application of a rule. For example, the existence of a certain subgraph *A* in *G* can be allowed or forbidden. A rule only can be applied if *A* can be found resp. not found in *G*. Additionally, label-based application conditions are possible. This rule could be extended by asking for $x < y$. Only in this case would the rule be applied.

Figure 1. The application of a graph rewrite rule $L \rightarrow R$ to a graph G