LEARNING NUMBERS FROM GRAPHS

Aurélie Goulon[1], Arthur Duprat[1,2], Gérard Dreyfus[1]

École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris (ESPCI-ParisTech)
[1]Laboratoire d'Électronique, [2]Laboratoire de Chimie Organique (CNRS UMR 7084)
10, rue Vauquelin
75005 PARIS, France
(e-mail agoulon@libertysurf.fr, Arthur.Duprat@espci.fr, Gerard.Dreyfus@espci.fr)

Abstract. The recent developments of statistical learning focused mainly on *vector machines,* i.e. on machines that learn from examples described by a vector of features. There are many fields where structured data must be handled; therefore, it would be desirable to learn from examples described by *graphs*. The presentation describes *graph machines*, which learn real numbers from graphs. Applications in the field of Quantitative Structure-Activity Relations (QSAR), which aim at predicting properties of molecules from their (graph) structures, are described.

1 Introduction

The present paper describes graph machines, i.e. machines that learn numbers from structured data, which can be described by graphs, in contrast to conventional approaches such as neural networks, kernel machines, support vector machines, which handle vectors. Unlike recursive neural networks, graph machines can handle any type of graph, whether cyclic or not. The first part of the paper is devoted to definitions. The second part is devoted to examples of applications; first, academic validations are described, showing that graph machines are indeed able to learn numbers related to the graph structure itself, such as graph diameters or Wiener indices. We proceed to show that graph machines are very efficient for QSAR and QSPR applications; comparisons with results obtained by other authors on the same data show that graph machines outperform standard machine learning techniques and recursive neural networks, with the computational advantage of exempting the model designer from performing the steps of computing and selecting descriptors, which are generally at least as costly as the training of the machine.

2 Graph machines

Before describing graph machines, some facts and definitions pertaining to vector machines are described cursorily.

2.1 Vector machines

Conventional numerical machine learning methods aim at learning applications from $\Re^n$ to $\Re^m$: data is in the form of pairs of vectors, the input vector being of dimension n, and the output vector of dimension m. In all the following we consider that $m = 1$

without loss of generality. When the task to be learnt is a classification task, the output is often binary; for process modeling, whether static or dynamic, the output is real. The techniques of machine learning for static modeling are very similar to statistical regression techniques: the main difference is the fact that statistical regression is essentially interested in the values of the *parameters* of the models, whereas modeling by machine learning is essentially interested in the *predictions* of the models. Support vector machines and neural networks are typical vector machines; support vector machines were designed mainly for classification tasks, with excellent performances; neural networks are more suitable for modeling, whether static (feedforward neural networks, also termed Multilayer Perceptrons), or dynamic (recurrent neural networks).
In all the following, we focus on static modeling, i.e. learning from examples an application of $\Re^n$ to $\Re$. The model is sought within a family of parameterized functions $g_{\boldsymbol{\theta}}(\mathbf{x})$, where $\mathbf{x}$ is the vector of variables (of dimension n) and $\boldsymbol{\theta}$ is the vector of parameters (of dimension p). Training is performed by minimizing a cost function, which is usually the least squares cost function, with respect to the parameters:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left(y_p^i - g_{\boldsymbol{\theta}}(\mathbf{x}_i) \right)^2 \tag{1}$$

where the summation runs on all N examples of the training set, y_p^i is the value of the quantity to be modeled for example i, and $\mathbf{x}_i$ is the vector of variables for example i.

2.2 Graph machines

2.2.1 Definition

We turn now to the problem of learning an application between a set of graphs and a corresponding set of real (or possibly binary) numbers. To start with, we consider directed acyclic graphs only. A natural idea is to build a model whose mathematical structure is the same as the structure of the input graph: each node of the graph is a parameterized function, and the model is a composition of that function, which reflects the structure of the graph. In the field of neural networks, such models are known as *folding networks* (the function present at each node is a feedforward neural network), but the idea can be extended to other types of machines (for a review see [Hammer, 2003]). Consider, as an illustration, the graphs shown on Figure 1:

- the graph machine associated to graph 1 is:
  $f^1_{\boldsymbol{\theta},\Theta} = G_{\Theta}\left\{ g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(\mathbf{x}) \right\}$;
- the graph machine associated to graph 2 is:
  $f^2_{\boldsymbol{\theta},\Theta} = G_{\Theta}\left\{ g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(\mathbf{x})), 0), g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(\mathbf{x})))) \right\}$;
- the graph machine associated to graph 3 is:
  $f^3_{\boldsymbol{\theta},\Theta} = G_{\Theta}\left\{ g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), 0), g_{\boldsymbol{\theta}}(g_{\boldsymbol{\theta}}(\mathbf{x}), g_{\boldsymbol{\theta}}(\mathbf{x}))))), 0), 0 \right\}$.

In the above examples, the size of **x** must be at least equal to the maximal in-degree d_m of the nodes of the graph. For a node of in-degree $d < d_m$, $d_m - d$ components of **x** are arbitrary, and may be taken equal to 1 for instance.

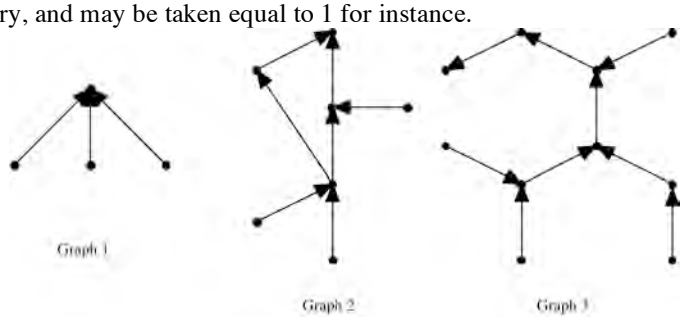

Figure 1

For generality, in the above examples, the function $G_{\boldsymbol{\Theta}}$ associated to the final root is different from the function $g_{\boldsymbol{\theta}}$ of the other nodes. That is by no means necessary; in all examples described in the present paper, all nodes including the root were assigned the same function.
The size of vector $\boldsymbol{\theta}$ (and the size of $\boldsymbol{\Theta}$) depends on the complexity of the mapping, just as for vector machines.

Definition: a graph machine is a set G of parameterized functions, constructed as described above from the same functions $g_{\boldsymbol{\theta}}(\mathbf{x})$ (and $G_{\boldsymbol{\Theta}}(\mathbf{x})$), which are representations of the graphs of the training set. The size of **x** is lower bounded by the maximal in-degree of the nodes of the graphs.

2.2.2 The training of a graph machine

The training of a graph machine is performed by minimizing a cost function with respect to the parameters $\boldsymbol{\theta}$ (and $\boldsymbol{\Theta}$); in all the examples described below, the least squares cost function was used:

$$J(\boldsymbol{\theta},\boldsymbol{\Theta}) = \sum_{i=1}^{N} \left(y_p^i - f_{\boldsymbol{\theta},\boldsymbol{\Theta}}^i \right)^2 \tag{2}$$

where y_p^i is the quantity to be learnt, associated to graph *i*. Note the difference with the cost function (1) that is minimized during the training of a vector machine:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left(y_p^i - g_{\boldsymbol{\theta}}(\mathbf{x}_i) \right)^2 .$$

Instead of training a single parameterized function with different input output-pairs, different parameterized functions, *sharing the same set of parameters*, are trained with a single example each.
As mentioned above, the fact that two sets of parameters, $\boldsymbol{\theta}$ and $\boldsymbol{\Theta}$, are used in graph machines is unimportant. A single parameter vector $\boldsymbol{\theta}$ is often sufficient.
In practice, training is performed much in the same way as vector machines. One has

$$\frac{\partial J}{\partial \theta_k} = \sum_{i=1}^{N} \frac{\partial J^i}{\partial \theta_k} \quad \text{where } J^i = \left(y_p^i - f_{\boldsymbol{\theta},\boldsymbol{\Theta}}^i \right)^2 \tag{3}$$

and θ_k denotes the k-th component of vector $\boldsymbol{\theta}$. For neural networks, the gradient of J^i with respect to each parameter $\frac{\partial J^i}{\partial \theta_{k_j}}$ is computed by backpropagation on the network that represents graph i; θ_{k_j} denotes the j-th occurrence of parameter θ_k in graph i. Denoting by $n_{\theta_k}^i$ the number of occurrences of parameter θ_k in graph i, the shared weight trick consists in setting

$$\frac{\partial J^i}{\partial \theta_k} = \sum_{j=1}^{n_{\theta_k}^i} \frac{\partial J^i}{\partial \theta_{k_j}} \tag{4}$$

(if the root node has the same parameters as the other nodes, then $n_{\theta_k}^i$ is equal to the number of nodes in graph i). Therefore, one obtains:

$$\frac{\partial J}{\partial \theta_k} = \sum_{i=1}^{N} \sum_{j=1}^{n_{\theta_k}^i} \frac{\partial J^i}{\partial \theta_{k_j}} \tag{5}$$

Finally, the cost function (2) is minimized by any appropriate gradient optimization method (Levenberg-Marquardt, BFGS, conjugate gradient, etc.), using gradient (5).

2.2.3 Model selection

All the tricks-of-the-trade that are usually applied to vector machines can be applied to graph machines as well: validation, cross-validation, leave-one-out, bootstrap estimates of the generalization error, etc. In the following examples, cross-validation is used for model selection; the root mean square error on a set (training or validation) is defined as:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(y_p^i - f_{\boldsymbol{\theta},\boldsymbol{\Theta}}(\mathbf{x}_i) \right)^2} \tag{6}$$

where N is the size of the set.

3 Examples

3.1 Neural-network-based graph machines

In all examples described below, the function $g_{\boldsymbol{\theta}}$ is a neural network. Therefore, a machine is made of identical neural networks, connected with the same structure as the nodes in the graph. For example, consider a node A with n parent vertices B_i, $i = 1, \ldots, n$. An elementary neural network (A) is assigned to that node: its inputs are (i) the outputs of the networks (B_i), and (ii) additional inputs that provide information on the node (e.g. its degree). If the graph is cyclic, the degree of the node is provided by one such input, so that a cyclic graph is first turned into a

directed acyclic graph by deleting as many edges as necessary, while retaining the information about the original graph structure.

3.2 Learning graph properties

In order to validate the approach in an academic way, graph machines were trained to learn graph properties. For all examples described below, a data base of 150 randomly generated graphs, featuring 2 to 15 nodes and 0 to 9 cycles was created. Various splits between training and validation sets were performed on that data base.

3.2.1 Learning the number of nodes and cycles of a graph

The easiest problem consists in learning the number of nodes of a graph, since it is a linear problem. The number of vertices N is equal to the number of elementary functions $g_{\boldsymbol{\theta}}$ in the graph machine, and the number of cycles of a connected graph is given by:

$$C = E - N + 1$$

where E is the number of edges. Therefore, graph machines with *linear* elementary functions

$$g_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_i \theta_i \mathbf{x}_i$$

should learn those tasks. As expected, for all splits between training and validation sets, the task was perfectly learnt and the error was equal to zero.

3.2.2 Learning the diameter of a graph

The diameter of a graph is the length of the shortest path between its most distant nodes:

$$D = \max_{u,v} d(u,v)$$

where $d(u, v)$ is the distance (the shortest path) between nodes u and v. In the database under investigation, that index ranges from 1 to 9. That is clearly a non-linear property; therefore, the elementary function was a neural network with four hidden neurons. The RMS error (relation (6)) on the training set is 0.36, and the RMS validation error (10-fold cross-validation) is 0.53. Since the index is an integer ranging from 1 to 9, the prediction is excellent given the complexity of the graphs.

3.2.3 Learning the Wiener Index of a graph

The Wiener Index of a graph G is the sum of the distances between the vertices of G. That index was first defined by the chemist H. Wiener, in order to investigate the relationships between the structure of chemicals and their properties:

$$W(G) = \frac{1}{2} \sum_{u,v} d(u,v)$$

In our database, that index ranges from 1 to 426.
10-fold cross-validation was performed with a 4-hidden neuron elementary neural network, leading to a RMS validation error of 7.9.

The above examples (together with other examples not reported here) show the ability of graph machines to learn from the sole data structure, without any need for extraneous descriptors.
In addition, they prove that indices such as the Wiener index need not be used as descriptors (e.g. in QSAR as described in the next section) since the information is present in the structure of the machine.

3.3 Application to the prediction of chemical properties of molecules

3.3.1 Graph machines for QSPR/QSAR

Graph machines are particularly appropriate for the prediction of molecular properties in QSPR (Quantitative Structure-Property Relations) and QSAR (Quantitative Structure-Activity Relations). A molecule can be described as a directed graph by associating each non-hydrogen atom to a node and each bond to an edge. The original graphs are preprocessed in order to turn them to acyclic graphs as described in section 3.1. Provision is made for extraneous inputs that code (in a one-out-of-*n* code) for the nature of the atoms, their degree or their stereochemistry for example. Therefore, any type of molecule can be handled, be it acyclic, cyclic or even aromatic. Figure 2 shows how an aromatic compound can be described as a graph; the digits are the degrees of the nodes.

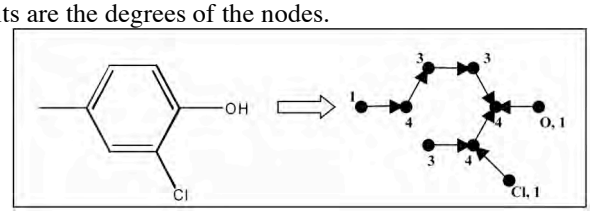

Figure 2

3.3.2 Learning and predicting boiling points of alkanes

Graph machines were first tested on the prediction of the boiling points of a set of linear or branched acyclic alkanes. Table 1 compares the results obtained by graph machines to those found in the literature.

	RMSE (K) (training)	*RMSE* (K) 10-fold cross-validation
Graph machines	1.0	1.5
Recursive neural networks [Bianucci *et al.*, 2000]	2.0	3.0
Conventional neural networks [Cherqaoui and Villemin, 1994]	2.2	2.7

Table 1

3.3.3 Predicting the toxicity of phenols

Phenols are a family of chemicals that are of current industrial use as biocides or disinfectants. Most synthetic phenols are toxic and considered as dangerous pollutants. We studied a set of 153 of these phenols, whose toxicity to a particular kind of cells, Tetrahymena pyriformis, was available ([Schultz, 1997]). This database is especially interesting since it contains complex molecules with 6 kinds of heteroatoms, and it deals with a property that is close to pharmacological properties.

In order to compare the performances of graph machines to those obtained with other methods - Multiple Linear Regression (MLR), Support Vector Machines (SVM) and Radial Basis Function Neural Networks (RBFNN) - the same protocol as used in [Yao *et al.*, 2004] was implemented: the database was split into training and validation sets of 131 and 22 examples respectively. The results obtained with graph machines built with 4 and 5 hidden neuron (GM-4N and GM-5N) elementary neural networks are summarized in Table 2, where they are compared to those obtained with the previously cited methods.

	Method	Learning	Validation
RMSE	GM-4N	0.17	0.29
	GM-5N	0.09	0.32
	MLR	0.30	0.46
	RBFNN	0.19	0.29
	SVM	0.22	0.36

Table 2

For a more rigorous assessment, 7-fold cross-validation was also performed on the same set with a 4 hidden neuron network. The RMSE obtained were then respectively 0.16 and 0.29 in learning and validation. No test set was provided in the referenced articles.

The above results show that graph machines compare favorably with other QSAR methods for the prediction of that biological activity, with an accuracy that is at least as good as the accuracy of the methods investigated by other authors, be it on the learning or on the validation sets. However, whereas the other methods require the prior selection and measurement or computation of descriptors such as hydrophobicity (log Kow), acidity constant (pKa), and frontier orbital energies (HOMO and LUMO), the structure of the molecules is the only information required for graph machines to perform accurate predictions. This is a twofold advantage. First, graph machines are much less computationally expensive than methods that require the design, selection and computation of descriptors. Furthermore, since no specific descriptors are selected, a graph machine implemented for a given molecule can be used for the prediction of *any* property of that molecule: the only requirement is a re-training of the machine, whereas conventional vector machines require the

selection and computation of an appropriate set of descriptors for each property to be predicted.

3.3.4 A classification task: classification of the molecules as aromatics/ non aromatics

Graph machines can perform classification tasks, just as neural networks or SVM's do. As a test example, 240 molecules were classified into two classes: molecules that feature no aromatic cycle and molecules that feature at least one aromatic cycle. 10-fold cross-validation was performed on that database, leading to a training classification error rate of 0% and a validation error rate of 2% with a 4 hidden neuron elementary neural network. A test set of 40 examples lead to an error rate of 0%. Again, no descriptor whatsoever was computed prior to performing classification.

4 Conclusion

In the present paper, graph machines have been described, and some of their applications have been outlined. The results presented here show that graph machines outperform vector machines and recursive neural networks. The prediction of the properties of molecules from their structure is obviously an important field of application of our approach, but it can be conjectured that graph machines may be beneficial in all fields where learning must be performed from structured data.

References

[Bianucci *et al*., 2000]A.M. Bianucci, A. Micheli, A. Sperduti, and A. Starita. Application of cascade correlation networks for structures to chemistry. *Applied Intelligence*, pages 115-145, 2000.

[Cherqaoui and Villemin, 1994]D. Cherqaoui and D. Villemin. Use of a neural network to determine the boiling point of alkanes. *Journal of the Chemical Society, Faraday Transactions*, pages 97-102, 1994.

[Hammer, 2003]B. Hammer, Perspectives on Learning Symbolic Data with Connectionistic Systems. In R.Kühn, R.Menzel, W.Menzel, U.Ratsch, M.M.Richter, I.-O.Stamatescu, eds., , *Adaptivity and Learning,* pages 141-160, Springer, 2003.

[Schultz, 1997]T.W. Schultz. TETRATOX: The Tetrahymena pyriformis population growth impairment endpoint - A surrogate for fish lethality. *Toxicological Methods 7*, pages 289-309, 1997.

[Yao *et al*., 2004]X. J. Yao, A. Panaye, J.P. Doucet, R.S. Zhang, H.F. Chen, M.C. Liu, Z.D. Hu, and B.T. Fan. Comparative Study of QSAR/QSPR Correlations Using Support Vector Machines, Radial Basis Function Neural Networks, and Multiple Linear Regression. *Journal of Chemical Information and Computer Sciences*, pages 1257-1266, 2004.