Graph Machines and Their Applications to Computer-Aided Drug Design: A New Approach to Learning from Structured Data

Conference Paper *in* Lecture Notes in Computer Science · September 2006

DOI: 10.1007/11839132_1 · Source: DBLP

3 authors, including:

Aurélie Goulon
Lavoix
12 PUBLICATIONS **90** CITATIONS
SEE PROFILE

Gérard Dreyfus
École Supérieure de Physique et de Chimie Industrielles
226 PUBLICATIONS **5,005** CITATIONS
SEE PROFILE

Lecture Notes in Computer Science, vol. 4135, pp. 1 – 19, Springer (2006)

Graph Machines and Their Applications to Computer-Aided Drug Design: a New Approach to Learning from Structured Data

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

[1]Laboratoire d'Électronique, [2]Laboratoire de Chimie Organique,
(CNRS UMR 7084)
École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris
(ESPCI-ParisTech)
10 rue Vauquelin, 75005 PARIS, France

Agoulon@libertysurf.fr, Arthur.Duprat@espci.fr, Gerard.Dreyfus@espci.fr

Abstract. The recent developments of statistical learning focused on *vector machines,* which learn from examples that are described by vectors of features. However, there are many fields where structured data must be handled; therefore, it would be desirable to learn from examples described by *graphs*. *Graph machines* learn real numbers from graphs. Basically, for each input graph, a separate learning machine is built, whose algebraic structure contains the same information as the graph. We describe the training of such machines, and show that virtual leave-one-out, a powerful method for assessing the generalization capabilities of conventional vector machines, can be extended to graph machines. Academic examples are described, together with applications to the prediction of pharmaceutical activities of molecules and to the classification of properties; the potential of graph machines for computer-aided drug design is highlighted.

Introduction

Whether neural networks still fall in the category of "unconventional" computational methods is a debatable question, since that technique is well understood and widely used at present; its advantages over conventional regression methods are well documented and mathematically proven. Neural networks are indeed conventional in that they learn from *vector* data: typically, the variables of the neural model are in the form of a vector of numbers. Therefore, before applying learning techniques to neural networks, or any other conventional learning machine (Support Vector Machine, polynomial, multilinear model, etc.), the available data must be turned into a vector of variables; the learning machine then performs a mapping of a set of input vectors to a set of output vectors. In most cases, the output is actually a scalar, so that the mapping is from $\mathbb{R}^n$ to $\mathbb{R}$, where n is the dimension of the input vectors. When modeling a physical process for instance, the factors that have an influence on the quantity to be modeled are known from prior analysis, so that the construction of the vector of vari-

ables is straightforward, requiring simply normalization, and possibly variable selection by statistical methods.

In many cases of interest, however, encoding the data into a vector cannot be performed without information loss. Such is the case whenever the information to be learnt from is structured, i.e. is naturally encoded into a graph. In scene analysis for instance, a scene can be encoded into a graph that describes the relationships between the different parts of the scene. In computer-aided drug design, the purpose of learning is a mapping of the space of molecules to the space of pharmaceutical activities; in most cases, the structure of the molecule explains, to a large extent, its activity. Since molecular structures are readily described by graphs, QSAR (Quantitative Structure-Activity Relationships) aims at mapping the space of the graphs of molecular structures to the space of molecular activities or properties.

In the present paper, we describe an approach to learning that can be termed unconventional insofar as its purpose is a mapping of graphs to real numbers (or vectors) instead of a mapping of vectors to real numbers. The latter quantities may be either real-valued (graph regression) or binary (graph classification). The idea of learning from graphs (and generally structured data) can be traced back to the early days of machine learning, when Recursive Auto-Associative Memories (RAAMs) were designed for providing compact representations of trees [1]. It evolved subsequently to Labeled RAAMs [2], recursive networks [3], and graph machines (for a review of the development of numerical machine-learning from structured data, see [4]).

The first part of the paper is devoted to a description of graph machines and of some didactic, toy problems. It will also be shown that model selection methods that are proved to be efficient for conventional machine learning can be extended to graph machines. The second part of the paper will describe novel applications of graph machines to the prediction and classification of the properties or activities of molecules, a research area known as QSAR/QSPR (Quantitative Structure-Activity/Structure-Property Relationships). We show that graph machines are particularly powerful in that area, because they avoid a major problem of that field: the design, computation and selection of molecular descriptors.

Graph machines

We first provide the definitions and notations for handling acyclic graphs, and the construction of graph machines from general graphs (possibly cyclic). Academic problems are described as illustrations. It is shown that the training and model selection methods developed for vector machines can be extended to graph machines.

Handling directed acyclic graphs

Definitions: we consider the mapping from a set of acyclic graphs G to a set of real-valued numbers.

For each acyclic graph G_i of G, a parameterized function $g^i\ \mathbb{R}^n \to \mathbb{R}$ is constructed, which is intended (i) to encode the structure of the graph [5], and (ii) to provide a prediction of the quantity of interest, e.g. a property or an activity of the molecule, from G_i. It is constructed as follows. A parameterized function $f_{\boldsymbol{\theta}}$ ("node function") is associated to each node. $\boldsymbol{\theta}$ denotes the vector of parameters of the node function. All nodes, except the root node, have the same node function $f_{\boldsymbol{\theta}}$; those functions are combined in such a way that g^i has the same structure as graph G_i: if an edge from node k to node l exists in the graph, then the value of the node function associated to node k is a variable of the node function associated to node l. The root node may be assigned a different function, denoted by $F_{\boldsymbol{\Theta}}$, where $\boldsymbol{\Theta}$ is the vector of parameters of $F_{\boldsymbol{\Theta}}$. If the node functions are neural networks, the g^i's are termed recursive neural networks [3].

Notations: the following notations are used throughout the paper.
We denote by $\mathbf{x}_j$ the (optional) vector of labels that provide information about node j of graph G_i. The size of the label vector is denoted by X_i; it is the same for all nodes of a given graph. Therefore, the parameterized function associated to G_i will be denoted as $g^i_{\boldsymbol{\theta},\boldsymbol{\Theta}}\left(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{v_i}\right)$, where v_i is the number of nodes of graph G_i. If no specific information about the node is necessary, $g^i_{\boldsymbol{\theta},\boldsymbol{\Theta}}$ has no variable: its value depends only on the structure of graph G_i.

We denote by $\mathbf{z}_j$ the vector of variables of the node function $f_{\boldsymbol{\theta}}(\mathbf{z}_j)$ of the non-root node j of graph G_i. Denoting by d_j the in-degree of non-root node j, and defining $M_i = \arg\max_j d_j$, the size of vector $\mathbf{z}_j$ is equal to $D_i = M_i + X_i + 1$. The vectors of variables of the node functions $f_{\boldsymbol{\theta}}(\mathbf{z}_j)$ are constructed as follows: for all j, the first component z_j^0 is equal to 1 (the "bias" if $f_{\boldsymbol{\theta}}(\mathbf{z}_j)$ is a neural network, the constant term if $f_{\boldsymbol{\theta}}(\mathbf{z}_j)$ is an affine function); for node j, of in-degree d_j, components z_j^1 to $z_j^{d_j}$ are the values of the node functions assigned to the parent nodes of node j; if $d_j < M_i$, components $z_j^{d_j+1}$ to $z_j^{M_i}$ are equal to zero; if $X_i \neq 0$, components $z_j^{M_i+1}$ to $z_j^{M_i+X_i}$ are the labels of node j.

We denote by $\mathbf{y}_i$ the vector of variables of the node function $F_{\boldsymbol{\Theta}}\left(\mathbf{y}_i\right)$ of the root node of graph G_i. The size of $\mathbf{y}_i$ is $\Delta_i = d_r + X_r + 1$, where d_r denotes the in-degree of the root node and X_r the size of its vector of labels. y_i^0 is equal to 1 (bias), y_i^1 to $y_i^{d_r}$ are the values of the node functions assigned to the parent nodes of the root node, $y_i^{d_r+1}$ to $y_i^{d_r+X_r}$ are the labels assigned to the root node.

As an example, Fig. 1 shows an acyclic graph G_1 with maximum in-degree $M_1 = 2$; the corresponding graph machine is:

$$g^1_{\boldsymbol{\theta},\boldsymbol{\Theta}}\left(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_8\right) = F_{\boldsymbol{\Theta}}\left(\mathbf{y}_1\right) = F_{\boldsymbol{\Theta}}\left(\mathbf{x}_8, f_{\boldsymbol{\theta}}\left(\mathbf{x}_7, f_{\boldsymbol{\theta}}\left(\mathbf{z}_6\right), 0\right), f_{\boldsymbol{\theta}}\left(\mathbf{x}_5, f_{\boldsymbol{\theta}}\left(\mathbf{z}_4\right), f_{\boldsymbol{\theta}}\left(\mathbf{x}_3, f_{\boldsymbol{\theta}}\left(\mathbf{z}_2\right), f_{\boldsymbol{\theta}}\left(\mathbf{z}_1\right)\right)\right)\right)\right) \quad (1)$$

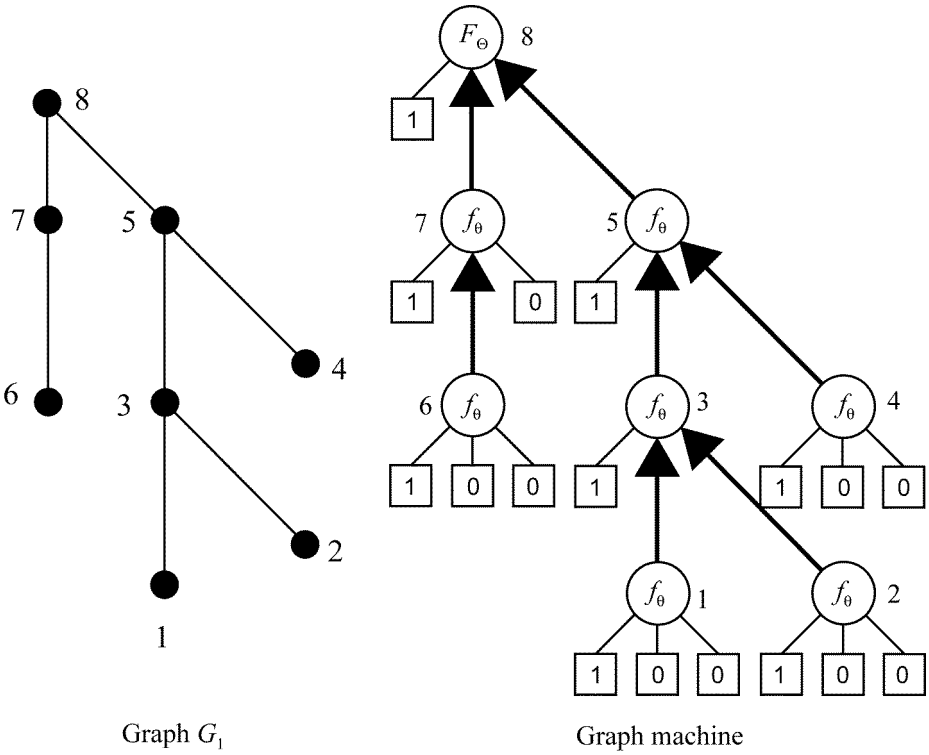

Fig. 1. An acyclic graph and its graph machine

If no information about the nodes is required by the problem at hand ($X_1 = 0$), one has $D = 3$, and:

$$\mathbf{z}_1 = \mathbf{z}_2 = \mathbf{z}_4 = \mathbf{z}_6 = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}^T,\ \mathbf{z}_3 = \begin{pmatrix} 1 & f_{\boldsymbol{\theta}}(\mathbf{z}_1) & f_{\boldsymbol{\theta}}(\mathbf{z}_2) \end{pmatrix}^T,$$

$$\mathbf{z}_5 = \begin{pmatrix} 1 & f_{\boldsymbol{\theta}}(\mathbf{z}_3) & f_{\boldsymbol{\theta}}(\mathbf{z}_4) \end{pmatrix}^T,\ \mathbf{z}_7 = \begin{pmatrix} 1 & f_{\boldsymbol{\theta}}(\mathbf{z}_6) & 0 \end{pmatrix}^T,\ \mathbf{y}_1 = \begin{pmatrix} 1 & f_{\boldsymbol{\theta}}(\mathbf{z}_5) & f_{\boldsymbol{\theta}}(\mathbf{z}_7) \end{pmatrix}^T.$$

Cyclic graphs

Graph machines handle cyclic graphs and parallel edges. To that effect, the initial graph is preprocessed by deleting a number of edges equal to the number of cycles, and all parallel edges but one; moreover, a label is assigned to each node: it is equal to the degree of the node, thereby retaining the information about the original graph structure. Finally, a root node is chosen and the edges are assigned orientations, according to an algorithm described in [6].

The training of graph machines

Graph machines are trained in the usual framework of empirical risk minimization. A cost function $J(\boldsymbol{\Theta},\boldsymbol{\theta})$ is defined, and its minimum with respect to the parameters is sought, given the available training data. The cost function takes into account the discrepancy between the predictions of the models and the observations present in the training set, and may include regularization terms, e.g.:

$$J\left(\boldsymbol{\theta},\boldsymbol{\Theta}\right)=\sum_{i=1}^{N}\left(y^i-g^i_{\boldsymbol{\theta},\boldsymbol{\Theta}}\right)^2+\lambda_1\left\|\boldsymbol{\theta}\right\|+\lambda_2\left\|\boldsymbol{\Theta}\right\|, \tag{2}$$

where N is the size of the training set, y^i is the value of the i-th observation of the quantity to be modeled, and λ_1 and λ_2 are suitably chosen regularization constants.

Since the parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\Theta}$ must be identical within each function g^i and across all those functions, one must resort to the so-called *shared weight* trick; the k-th component of the gradient of the cost function can be written as

$$\frac{\partial J\left(\boldsymbol{\theta},\boldsymbol{\Theta}\right)}{\partial\theta_k}=\sum_{i=1}^{N}\frac{\partial J^i}{\partial\theta_k}, \tag{3}$$

where J^i is the contribution of example i to the cost function. We denote by $n^i_{\theta_k}$ the number of occurrences of parameter θ_k in acyclic graph G_i; if the root is assigned the same parameterized function as the other nodes, then $n^i_{\theta_k}$ is equal to the number of nodes in graph G_i. The shared weight trick consists in setting

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

so that one has finally:

$$\frac{\partial J\left(\boldsymbol{\theta},\boldsymbol{\Theta}\right)}{\partial\theta_k}=\sum_{i=1}^{N}\sum_{j=1}^{n^i_{\theta_k}}\frac{\partial J^i}{\partial\theta_{k_j}}. \tag{5}$$

Relation (5) is subsequently used for minimizing cost function (2) by any suitable gradient descent algorithm (Levenberg-Marquardt, BFGS, conjugate gradient, …).

If functions $f_{\boldsymbol{\theta}}$ and $F_{\boldsymbol{\Theta}}$ are neural networks, the usual backpropagation algorithm may be conveniently used for computing the gradient; otherwise, one resorts to numerical estimations thereof.

Didactic examples: learning the number of nodes and the number of cycles of a graph

In the present section, two simple examples are provided, whose solutions can be worked out analytically because they are linear. In both cases, we consider the training set made of three graphs, shown on Fig. 2.

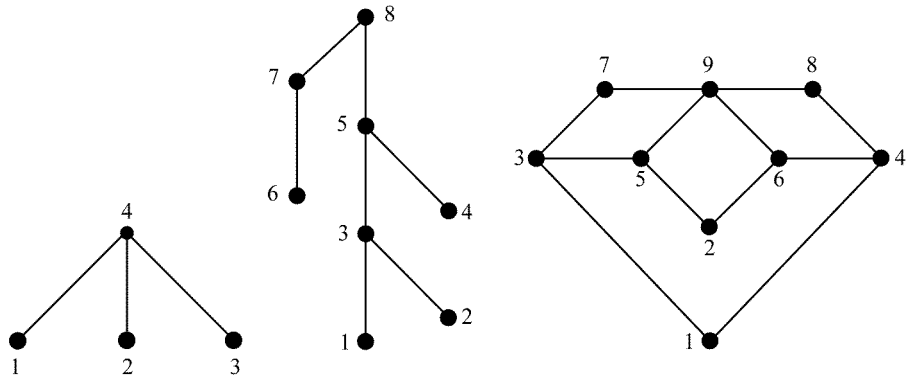

Fig. 2. A training set

Learning the number of nodes of a graph: first, assume that it is desired to learn, from examples, the number of nodes of a graph. Then the desired mapping is: $G_1 \rightarrow 4$; $G_2 \rightarrow 8$; $G_3 \rightarrow 9$. Moreover, generalization should be performed by using the node functions thus obtained in any other graph machine, i.e. to compute the number of nodes of any graph.

The first step consists in constructing directed acyclic graphs (DAGs) from the initial graphs. The construction of the DAGs is obvious for G_1 and G_2. Since graph G_3 has four cycles, four edges must be deleted. Fig. 3 shows the directed acyclic graphs on which the graph machines will be based.

The node function $f_{\boldsymbol{\theta}}$ is sought in the family of affine functions $f_{\boldsymbol{\theta}}(\mathbf{z}) = \sum_{j=0}^{D-1} \theta_j z_j$, and $F_{\boldsymbol{\Theta}}$ is taken identical to $f_{\boldsymbol{\theta}}$. Since the presence or absence of an edge is irrelevant for the computation of the number of nodes, no label is necessary: $X_1 = X_2 = X_3 = 0$. The node functions being the same for all graphs of the training set, we take $D = \max_i M_i + 1 = 5$. Since all edges are equivalent, one has $\theta_1 = \theta_2 = \theta_3 = \theta_4 = \theta$. Therefore, there are actually two independent parameters only.

The obvious solution is $\theta_0 = \theta = 1$. For graph G_1 for instance, one has:

$$g^1_{\boldsymbol{\theta},\boldsymbol{\Theta}}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = f_{\boldsymbol{\theta}}(1, f_{\boldsymbol{\theta}}(\mathbf{z}_1), f_{\boldsymbol{\theta}}(\mathbf{z}_2), f_{\boldsymbol{\theta}}(\mathbf{z}_3), 0) = \theta_0 + 3\theta\theta_0 = 4,$$

where $\mathbf{z}_1 = \mathbf{z}_2 = \mathbf{z}_3 = (1 \quad 0 \quad 0 \quad 0 \quad 0)^{\mathrm{T}}$.

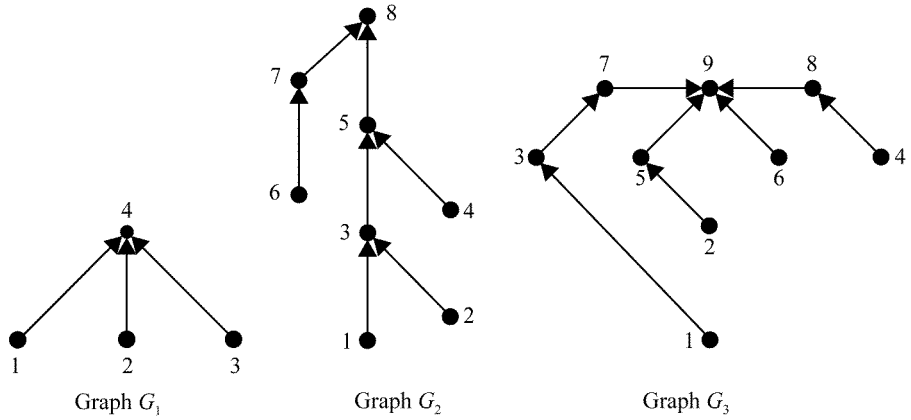

Fig. 3. The acyclic graphs derived from the training set shown on Fig. 2.

Similarly, one has, for graph G_2: $g^2_{\boldsymbol{\theta},\boldsymbol{\Theta}}\left(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_8\right)=2\theta_0\left(1+\theta+\theta^2+\theta^3\right)=8$, and, for graph G_3: $g^3_{\boldsymbol{\theta},\boldsymbol{\Theta}}\left(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_9\right)=\theta_0\left(1+4\theta+3\theta^2+\theta^3\right)=9$.

Learning the number of cycles in a graph: similarly, consider learning, from examples, the number of cycles in a graph.
By contrast to the previous example, the presence or absence of edges is highly relevant, so that each node must be labeled by its degree: $X_1 = X_2 = X_3 = 1$. Therefore, one has $D=\max_i M_i+2=5$.

$f_{\boldsymbol{\theta}}$ is sought in the family of affine functions $f_{\boldsymbol{\theta}}\left(\mathbf{z}\right)=\sum_{j=0}^{D-1}\theta_j z_j$, and the root node is assigned a different affine function $F_{\boldsymbol{\Theta}}\left(\mathbf{y}_i\right)$. Therefore, the size of vectors $\mathbf{y}_i$ is $\max_i \Delta_i+1=7$, because the present problem requires an additional label, equal to 1, for the root nodes.

An obvious solution to the problem is the following: θ_0 = -1, $\theta_1 = \theta_2 = \theta_3 = 1$, $\theta_4 = 1/2$, Θ_0 = -1, $\Theta_1 = \Theta_2 = \Theta_3 = \Theta_4 = 1$, $\Theta_5 = 1/2$, $\Theta_6 = 1$; the additional label assigned to the root node is equal to 1.

Consider graph G_1: $f_{\boldsymbol{\theta}}(\mathbf{z}_1) = f_{\boldsymbol{\theta}}(\mathbf{z}_2) = f_{\boldsymbol{\theta}}(\mathbf{z}_3)$ = -1/2,
$\mathbf{y}_1=\left(1\quad -1/2\quad -1/2\quad -1/2\quad 0\quad 3\quad 1\right)^{\mathrm{T}}$, so that: $F_{\boldsymbol{\Theta}}(\mathbf{y}_1) = -1 - 3/2 + 3/2 + 1 = 0$, as expected.

Similarly, for graph G_3, one has: $f_{\boldsymbol{\theta}}(\mathbf{z}_1) = f_{\boldsymbol{\theta}}(\mathbf{z}_2) = 0$, $f_{\boldsymbol{\theta}}(\mathbf{z}_4) = f_{\boldsymbol{\theta}}(\mathbf{z}_6) = 1/2$, $f_{\boldsymbol{\theta}}(\mathbf{z}_3) = f_{\boldsymbol{\theta}}(\mathbf{z}_5) = 1/2$, $f_{\boldsymbol{\theta}}(\mathbf{z}_7) = f_{\boldsymbol{\theta}}(\mathbf{z}_8) = 1/2$, $\mathbf{y}_3=\left(1\quad 1/2\quad 1/2\quad 1/2\quad 1/2\quad 4\quad 1\right)^{\mathrm{T}}$, hence $F_{\boldsymbol{\Theta}}(\mathbf{y}_3) = 4$.

Some nonlinear learning tasks

The above two problems have linear solutions that can be obtained by inspection. In general, graph regression or classification problems cannot be solved in the framework of linear models, so that one has to resort to training, as described above. The two examples described below are examples of graph machines being trained to learn graph properties as in the previous section, but the solutions are not linear. A database of 150 randomly generated graphs, featuring 2 to 15 nodes and 0 to 9 cycles, was created. Model selection was performed by cross-validation.

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

$$D = \max_{u,v} d(u,v)\,, \tag{6}$$

where $d(u, v)$ is the distance (the length of the shortest path) between nodes u and v. In the database under investigation, the index ranges from 1 to 9. That is clearly a non-linear property; therefore, the node function was a neural network; model selection resulted in a neural network with four hidden neurons. The root mean square error on the training set is 0.36, and the root mean square 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.

Learning the Wiener index of a graph: the Wiener index of a graph G is the sum of the distances between its vertices. That index was first defined by H. Wiener [7], in order to investigate the relationships between the structure of chemicals and their properties. It is defined as:

$$W(G) = \frac{1}{2}\sum_{u,v} d(u,v)\,. \tag{7}$$

In our database, that index ranges from 1 to 426.

Model selection by 10-fold cross-validation resulted in a 4-hidden neuron node function, leading to a RMS validation error of 7.9. The scatter plot is shown on Fig. 4, illustrating the accuracy of the results obtained by training *without having to compute any feature for describing the graph structure*. The problem of feature design and selection, which is central in conventional machine learning with vector machines, is irrelevant for graph machines. This is very important for the applications described below.

Model selection

Similarly to vector machines, usual model selection techniques such as hold-out, K-fold cross-validation, leave-one-out, can be applied to recursive networks and to graph machines. In the present section, we show how virtual leave-one-out, a power-

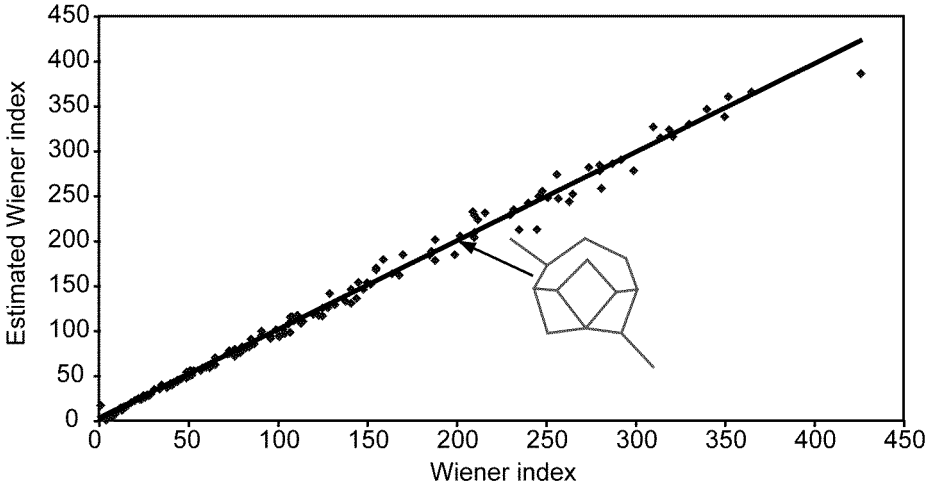

Fig. 4. Learning the Wiener index of graphs

ful method for estimating the generalization capability of a vector machine, can be extended to graph machines.

Virtual leave-one-out for vector machines: leave-one-out is known to provide an unbiased estimation of the generalization error of a model [8]. However, it is very demanding in terms of computation time: it involves training N different models, where N is the number of examples; each model is trained from $N - 1$ examples, and the modeling error on the left-out example is computed; the estimator of the generalization error is

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(R_i^{-i}\right)^2}\ , \tag{8}$$

where R_i^{-i} is the modeling error on example i when the latter is left out of the training set.

Virtual leave-one-out is a very effective method for obtaining an approximation of the above estimate [9], [10]. It consists in training the candidate model with all examples, and computing the virtual leave-one-out score as:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{R_i}{1-h_{ii}}\right)^2}\ , \tag{9}$$

where R_i is the modeling error on example i when the latter is in the training set. h_{ii} is the *tangent-plane leverage* of example *i:* it is the i-th diagonal element of the *hat matrix*:

$$\mathbf{H} = \mathbf{Z}\left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z}\right)^{-1}\mathbf{Z}^{\mathrm{T}} . \tag{10}$$

Z is the Jacobian matrix of the model, i.e. the matrix whose columns are the values of the partial derivative of the model $g_{\boldsymbol{\theta}}(\mathbf{x})$ with respect to its parameters, for the examples of the training set:

$$z_{ij} = \left(\frac{\partial g_{\boldsymbol{\theta}}\left(\mathbf{x}\right)}{\partial \theta_j} \right)_{\mathbf{x}=\mathbf{x}_i} . \tag{11}$$

For models that are linear in their parameters, relation (9) is exact: it is known as the PRESS (Predicted REsidual Sum of Squares) statistics. For models that are not linear in their parameters, such as neural networks, it is a first-order approximation of the estimator.

Leverages have the following properties:

$$\begin{aligned} &0 < h_{ii} < 1 \\ &\sum_{i=1}^{N} h_{ii} = q \end{aligned} \tag{12}$$

where q is the number of parameters of the model. Therefore, the leverage of example i can be viewed as the proportion of the parameters of the model that is devoted to fitting example i. If h_{ii} is on the order of 1, the model has devoted a large fraction of its parameters to fitting example i, so that the model is probably strongly overfitted to that example. Therefore, virtual leave-one-out is a powerful tool for overfitting detection and model selection.

Virtual leave-one-out for graph machines: in the present section, we show how virtual leave-one-out can be extended to graph machines. We give a simplified proof of the result, which provides a flavor of the full derivation. For simplicity, consider a model with a single parameter θ; moreover, assume that, for all graph machines, the node function of the root node is identical to the node function of non-root nodes $f_\theta(\mathrm{x})$. We denote by y_p^j the measured value of the property of interest for example j: the modeling error on example j is $R_j = y_p^j - g_\theta^j$; we denote by R_j^{-i} the modeling error on example j when example i has been withdrawn from the training set: $R_j^{-i} = y_p^j - g_{\theta^{-i}}^j$, where θ^{-i} is the parameter computed from the training set from which example i has been withdrawn. Therefore, one has:

$$R_j^{-i} = R_j + g_\theta^j - g_{\theta^{-i}}^j . \tag{13}$$

Assuming that the withdrawal of example i from the training set does not affect the parameters of the model to a large extent, a first-order Taylor expansion of the model, in parameter space, can be performed:

$$g^j_{\theta^{-i}} = g^j_\theta + \frac{\partial g^j_\theta}{\partial \theta}\left(\theta^{-i} - \theta\right). \tag{14}$$

Therefore, one has:

$$R^{-i}_j = R_j - \frac{\partial g^j_\theta}{\partial \theta}\left(\theta^{-i} - \theta\right). \tag{15}$$

The first derivative of the model can similarly be expanded to:

$$\frac{\partial g^j_{\theta^{-i}}}{\partial \theta} = \frac{\partial g^j_\theta}{\partial \theta} + \frac{\partial^2 g^j_\theta}{\partial \theta^2}\left(\theta^{-i} - \theta\right). \tag{16}$$

As defined above, the least squares cost function (without regularization terms), is

$$J\left(\theta\right) = \sum_{i=1}^{N}\left(y^i - g^i_\theta\right)^2, \tag{17}$$

which is minimum for θ. Therefore, one has

$$0 = \frac{\partial J\left(\theta\right)}{\partial \theta} = \sum_j R_j \frac{\partial g^j_\theta}{\partial \theta}, \tag{18}$$

and, similarly

$$0 = \frac{\partial J^{-i}\left(\theta\right)}{\partial \theta} = \sum_{j\neq i} R^{-i}_j \frac{\partial g^j_{\theta^{-i}}}{\partial \theta}, \tag{19}$$

where J^{-i} is the cost function after training with the dataset from which i was withdrawn. Substituting relations (15) and (16) into relation (19) gives:

$$0 = \sum_j R_j \frac{\partial g^j_\theta}{\partial \theta} - R_i \frac{\partial g^i_\theta}{\partial \theta} - \left[\sum_{j\neq i}\left(\frac{\partial g^j_\theta}{\partial \theta}\right)^2 - \sum_{j\neq i} R_j \frac{\partial^2 g^j_\theta}{\partial \theta^2}\right]\left(\theta^{-i} - \theta\right). \tag{20}$$

The first term on the right-hand side is equal to zero. Neglecting the second derivatives with respect to the squared first derivatives (the usual Levenberg-Marquardt approximation), one gets, to first order:

$$\left(\theta^{-i} - \theta\right) = -\frac{R_i \dfrac{\partial g^i_\theta}{\partial \theta}}{\displaystyle\sum_{j\neq i}\left(\frac{\partial g^j_\theta}{\partial \theta}\right)^2}. \tag{21}$$

Substituting into (15) with $j = i$, one obtains:

$$R_i^{-i} = \frac{R_i}{1 - h_{ii}}\,, \tag{22}$$

with:

$$h_{ii} = \frac{\left(\frac{\partial g_\theta^i}{\partial \theta}\right)^2}{\sum_j \left(\frac{\partial g_\theta^j}{\partial \theta}\right)^2}\,. \tag{23}$$

The above relation is similar to relation (10), which, for a single-parameter model, reduces to:

$$h_{ii} = \frac{\left(\frac{\partial g_\theta(\mathbf{x})}{\partial \theta}\right)^2_{\mathbf{x}=\mathbf{x}_i}}{\sum_j \left(\frac{\partial g_\theta(\mathbf{x})}{\partial \theta}\right)^2_{\mathbf{x}=\mathbf{x}_j}}\,. \tag{24}$$

Thus, virtual leave-one-out can provide an estimate of the generalization error of graph machines, in much the same way as for conventional vector machines: the matrix whose general term is $z_{ij} = \frac{\partial g_{\boldsymbol{\theta}}^i}{\partial \theta_j}$ plays exactly the same role as the Jacobian matrix $\mathbf{Z}$ (of general term $z_{ij} = \left(\frac{\partial g_{\boldsymbol{\theta}}(\mathbf{x})}{\partial \theta_j}\right)_{\mathbf{x}=\mathbf{x}_i}$) for conventional vector machines. In the case of neural networks, it can easily be computed by backpropagating an error equal to ½, after the completion of training.

Graph machines for the prediction of properties and/or activities of molecules

The prediction of the physico-chemical properties and pharmaceutical activities of molecules is a critical task in the drug industry for shortening the development times and costs. Typically, one synthesized molecule out of 10,000 becomes a commercial drug, and the development time of a new drug is approximately 10 years. Therefore, predicting the activity of a hitherto non-existent molecule may lead to tremendous savings in terms of development time and cost. Hence, over the past few years, QSPR/QSAR has become a major field of research in the chemical industry.

In a typical QSAR/QSPR scenario, a database of measured properties or activities of molecules is available, and it is desired to infer, from those data, the property/activity of molecules that have not yet been synthesized. Therefore, machine learning is a natural context for solving such problems. Linear, polynomial, neural, and SVM regression have been used extensively. For all such techniques, the design and the selection of relevant features, for the prediction of a given activity, are a major problem.

In the following, we show that graph machines solve that problem by exempting the model designer from finding and computing elaborate features, because the structure of the molecule is embodied into the learning machine itself. We show that, for the problems described here as well as for other problems, graph machines provide predictions that are at least as good as (and generally better than) predictions made by conventional machine learning, without the need for designing, computing and selecting features.

Encoding the molecules

Molecules are usually described in databases in a representation called SMILES (Simplified Molecular Input Line Entry System), which provides a description of the graph structure of the molecule as a character string. In the applications described here, the functions $g_{\mathbf{\theta}}^{i}$ were generated from the SMILES files of the molecules by the following procedure: the molecules, described by these files, were converted into labeled graphs by the association of each non-hydrogen atom to a node, and each bond to an edge. The nodes were also assigned labels describing the atoms they were related to (e.g. their nature, their degree or stereoisomery …). Then, the adjacency matrices associated to those labeled graphs were generated. In the subsequent step, the matrices were cast into a canonical form, by an algorithm that ranks the nodes according to criteria such as their degree, the fact that they belong to a cycle… [6]. This canonical form allowed the choice of the root nodes, and the conversion of the graphs into directed acyclic graphs. Fig. 5 illustrates the processing of a molecule from its SMILES representation into a directed acyclic graph.

Graph machines were then built for each graph of the data set; node functions were feedforward neural networks with a single layer of hidden neurons whose complexity (i.e. the number of neurons in the hidden layer) was controlled by cross-validation. The graph machines were then trained, with the shared weight condition, using the software package NeuroOne™[1], which computes the gradient of the cost function by backpropagation and minimizes the cost function by the Levenberg-Marquardt algorithm.

[1] NeuroOne is a product of Netral S.A. (http://www.netral.com)

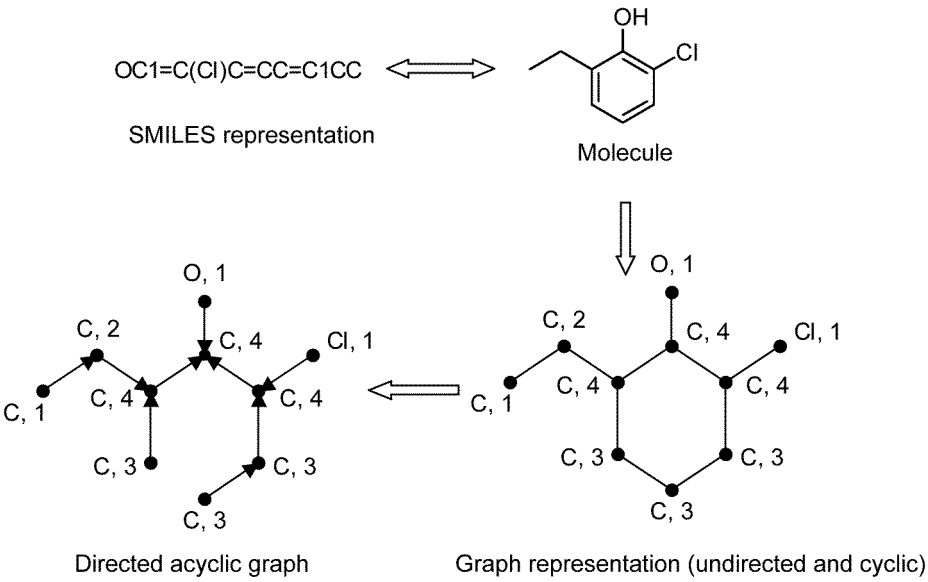

Fig. 5. Encoding a molecule into a graph machine

Predicting the Boiling Points of Halogenated Hydrocarbons

The volatility of halogenated hydrocarbons is an important property, because those compounds are widely used in the industry, for example as solvents, anaesthetics, blowing agents, and end up in the environment, where they can damage the ozone layer or be greenhouse gases. The volatility of a molecule can be assessed by its boiling point, a property measured only for a small proportion of possible halogenated hydrocarbons.

We studied a data set of 543 haloalkanes, whose boiling points were previously predicted by Multi Linear Regression (MLR) [11] [12]. This regression required the computation of numerous molecular descriptors, including arithmetic descriptors, topological indices, geometrical indices, and counts of substructures and fragments. The best feature subset was then selected, and generally comprised 6 or 7 descriptors. To provide a comparison with the results obtained by this method, we used the same training and test sets as Rücker et al. [12]. They feature 507 and 36 haloalkanes respectively, whose boiling points range from -128 °C to 249 °C.

In order to select the number of neurons required by the complexity of the problem, we first performed 10-fold cross-validation on the 507 examples of the training set. The results suggested the use of neural networks with 4 hidden neurons.

We then trained the selected graph machines, and predicted the boiling points of the test set molecules. The results of this study are shown in Table 1, where they are compared to the results obtained by Rücker et al. [12] on the same sets, using a 7-regressor MLR model. The predictions of the model on the test set are also displayed on Fig. 6.

Table 1. Prediction of the boiling points of haloalkanes by multi-linear regression and graph machines

	MLR (7-descriptor)		**4N-GM**	
	RMSE (°C)	R^2	RMSE (°C)	R^2
Training	6.607	0.9888	3.92	0.9960
10-fold CV	-	-	4.70	0.9942
Test	7.44	0.9859	5.07	0.9921

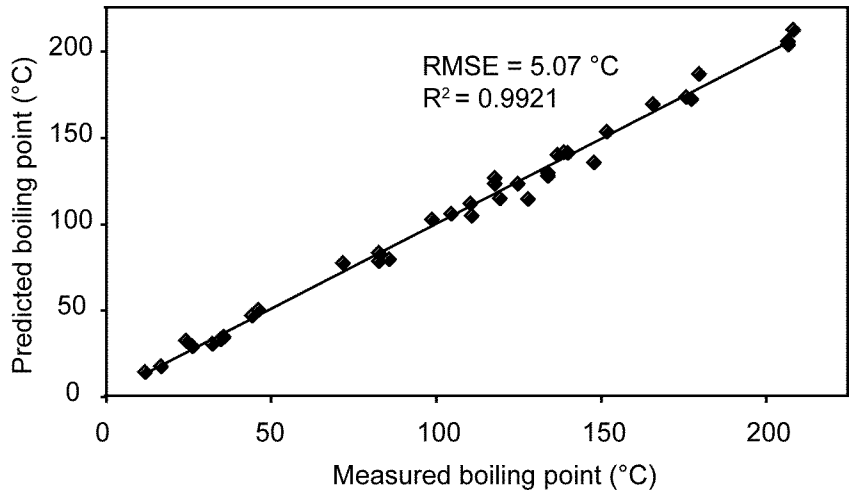

Fig. 6. Scatter plot for the prediction of the boiling point of 36 haloalkanes

The above results show that graph machines are able to model the boiling points of haloalkanes very well, without requiring the computation of any descriptor. Furthermore, this modeling task illustrates the fact that the design of the learning machine from the structure of the molecules avoids the loss of information caused by the selection of descriptors. Actually, Rücker et al. [12] stress the fact that the prediction of the boiling points of fluoroalkanes with their model is not satisfactory, which is presumably due to the lack of a descriptor taking into account the strong dipole interactions. The removal of 7 of these compounds decreased the training error of MLR regression by 0.56 °C (from 6.607 to 6.049 °C). By contrast, in the case of graph machines, the errors on those examples are not particularly high, and their removal from the database decreased the training error by 0.08 °C only (from 3.92 to 3.84 °C).

Predicting the anti-HIV activity of TIBO derivatives

TIBO (Tetrahydroimidazobenzodiazepinone) derivatives are a family of chemicals with a potential anti-HIV activity. They belong to the category of non-nucleoside inhibitors, which block the reverse transcriptase of the retrovirus and prevent its duplication. We studied a data set of 73 of those compounds, whose activity was previously modeled with several QSAR methods, including conventional neural networks [13], multi-linear regression [14], comparative molecular field analysis (CoMFA) [15], and the substructural molecular fragments (SMF) method [16]. The latter approach is based on the representation of the molecules with graphs, which are split into fragments, whose contribution to the modeled activity is then computed with linear or non-linear regression. Those fragments are either atom-bond sequences, or "augmented atoms", defined as atoms with their nearest neighbours.

In order to compare the prediction abilities of graph machines to the performances of the SMF method [16], the data set was split into a training and a test set of 66 and 7 examples respectively, exactly as in [16]. The activity is expressed as $\log(1/IC_{50})$, where IC_{50} is the concentration leading to the inhibition of 50% of the HIV-1 reverse transcriptase enzyme. Since some compounds of the set are stereoisomers, a label that described the enantiomery (*R* or *S*) of the atoms was added when necessary.

We first performed 6-fold cross-validation on the training set with node functions having up to five hidden neurons to select the complexity of the model. Three hidden neurons provided the best cross-validation estimate of the generalization. The results obtained with this model are reported in Table 2 and on Fig. 7.

Table 2. Prediction of the activity of TIBO derivatives by different methods

	SMF		**3N GM**	
	RMSE	R^2	RMSE	R^2
Training set	0.89	0.854	0.28	0.9901
Test set	0.51	0.943	0.45	0.9255

Since the accuracies of the experimental values are not known, the prediction errors cannot be compared to the measurement errors. However, this study demonstrates again that graph machines compare favourably with other methods, without the requirement of computing descriptors. This task also illustrates another advantage of graph machines on some other methods: Solov'ev et al. [16] had to remove several compounds from their original set because they contained "rare" fragments, whereas this problem does not occur with graph machines, insofar as the molecules of the test set do not require labels (atom types or degrees for example) that are not present in the training set.

Additional results on the prediction of the toxicity of phenols, the anti-HIV activity of HEPT analogues, and the carcinogenicity of molecules, are described in [17].

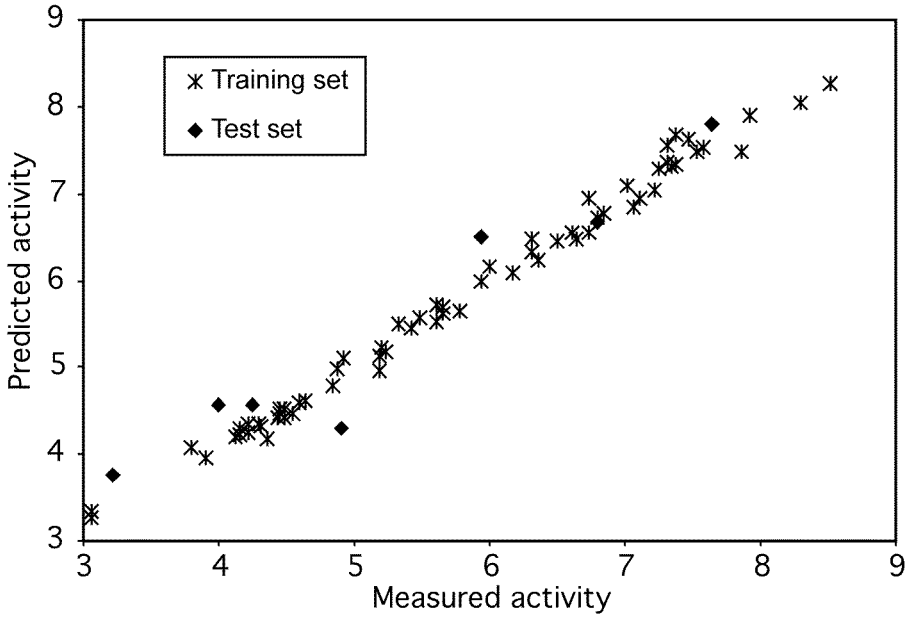

Fig. 7. Scatter plot of the prediction of the activity of TIBO derivatives

Graph machines for classification: discriminating aromatic from non-aromatic molecules

All the above examples are regression problems, in which a mapping is performed from graphs to real numbers. Graph machines can also perform classification, i.e. mappings from graphs to {-1, +1}. As an illustration, we show that discrimination between aromatic molecules (i.e. molecules that contain an aromatic cycle) and non-aromatic molecules can be performed. A cycle is aromatic when it fulfils several criteria: it must be planar, and have a set of conjugated π orbitals, thereby creating a delocalized π molecular orbital system. Furthermore, there must be $4n + 2$ electrons in this system, where n is an integer.

A set of 321 molecules was investigated, with various functional groups taken from [18]; it was divided into a training and a test set of 274 and 47 examples respectively. The proportion of molecules containing at least one aromatic cycle is shown on Fig. 8.

To select the number of hidden neurons required by this problem, 10-fold cross-validation was performed on the set of 273 examples. Table 3 reports the percentage of correct classification obtained with three and four hidden neurons.

The cross-validation error with the graph machines with 4 hidden neurons is due to a single misclassified example: the pipamperone, shown on Fig. 9. That error can be traced to the fact that the main part of the molecule is non-aromatic.

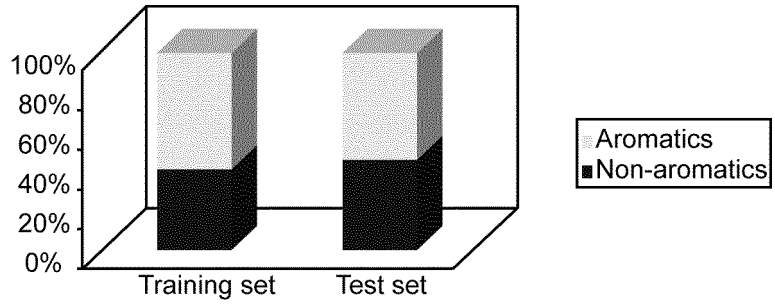

Fig. 8. Distribution of the molecules including at least one aromatic cycle in the data sets

Table 3. 10-fold cross-validation results for the prediction of the aromaticity

	Correct classification (training)	**Correct classification** (10-fold CV)
GM 3N	100%	100%
GM 4N	100%	99.7%

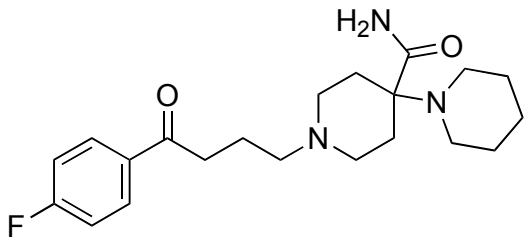

Fig. 9. Structure of the pipamperone, misclassified with 10-fold cross-validation

No example from the test set was misclassified by the graph machines with 3 hidden neurons, which illustrates the ability of graph machines to efficiently encode the structures of the graphs, thus to retain structure-related properties such as aromaticity.

Conclusions

A computational method that allows regression and classification on graphs has been described. Interestingly, it illustrates two principles that turn out to be very useful for solving real-life machine-learning tasks: (i) if a representation of the data for a given problem cannot be found "by hand", it may be advantageous to learn it; (ii) always embed as much prior information as possible into the structure of the learning machine. In agreement with statement (i), a representation of the graph evolves during training, as described in [5]; in agreement with statement (ii), the information about the structure of the data is embedded in the structure of the graph machine itself.

Model selection for recursive networks and graph machines used to be performed by conventional cross-validation, hold-out or leave-one-out; we have shown, in the present paper, that the powerful technique of virtual leave-one-out extends to recursive networks or graph machines in a relatively straightforward fashion; the full derivation of the results, and illustrations, will be provided in a forthcoming paper.

Applications of graph machines to computer-aided drug design have been described. The major asset of graph machines is their ability to make efficient predictions while exempting the model designer from the design, computation and selection of descriptors, which is recognized to be a major burden in QSAR/QSPR tasks. It should be noted, however, that, if the graph structure of the molecule is not sufficient for accurate prediction, descriptors could indeed be implemented as inputs to graph machines, in the form of labels for the nodes.

Scalability is an issue that should be investigated in a principled way. If the method is to be used for scene or text analysis for instance, very large corpuses must be handled. Experimental planning methods, allowing the model designer to use only the most informative data, have recently become available for nonlinear models in conventional machine learning. The extension of those techniques to graph machines should be investigated.

References

1. Pollack, J.B.: Recursive Distributed Representations. Artificial Intell. **46** (1990) 77-106.
2. Sperduti, A.: Encoding Labeled Graphs by Labeling RAAM. Connection Science **6** (1994) 429-459.
3. Frasconi, P., Gori, M, Sperduti, A: A General Framework for Adaptive Processing of Data Structures. IEEE Trans. on Neural Networks **9** (1998) 768-786.
4. Goulon-Sigwalt-Abram, A., Duprat, A., Dreyfus, G.: From Hopfield Nets to Recursive Networks to Graph Machines: Numerical Machine Learning for Structured Data. Th. Comp. Sci. **344** (2005) 298-334.
5. Hammer, B.: Recurrent Networks for Structured Data - a Unifying Approach and its Properties. Cognitive Systems Res. **3** (2002) 145-165.
6. Jochum, C., Gasteiger, J.: Canonical Numbering and Constitutional Symmetry. J. Chem. Inf. Comput. Sci. **17** (1977) 113-117.
7. Wiener, H.: Structural Determination of Paraffin Boiling Point. J. Amer. Chem. Soc. **69** (1947) 17-20.
8. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer-Verlag (2000).
9. Monari, G., Dreyfus, G.: Local Overfitting Control via Leverages. Neural Computation **14** (2002) 1481-1506.
10. Oussar, Y., Monari, G., Dreyfus G.: Reply to the Comments on "Local Overfitting Control via Leverages" in "Jacobian Conditioning Analysis for Model Validation" by I. Rivals and L. Personnaz. Neural Computation **16** (2004) 419-443.
11. Balaban, A.T., Basak S.C., Colburn, T., Grunwald, G.D.: Correlation between Structure and Normal Boiling Points of Haloalkanes C1-C4 Using Neural Networks. J. Chem. Inf. Comput. Sci. **34** (1994) 1118-1121.
12. Rücker, C., Meringer, M., Kerber, A.: QSPR Using MOLGEN-QSPR: The Example of Haloalkane Boiling Points. J. Chem. Inf. Comput. Sci. **44** (2004) 2070-2076.

13. Douali, L., Villemin, D., Cherqaoui, D.: Exploring QSAR of Non-Nucleoside Reverse Transcriptase Inhibitors by Neural Networks: TIBO Derivatives. Int. J. Mol. Sci. **5** (2004) 48-55.
14. Huuskonen, J.: QSAR modeling with the electrotopological state: TIBO derivatives. J. Chem. Inf. Comput. Sci. **41** (2001) 425-429.
15. Zhou, Z., Madura, J.D.: CoMFA 3D-QSAR Analysis of HIV-1 RT Nonnucleoside Inhibitors, TIBO Derivatives Based on Docking Conformation and Alignment. J. Chem. Inf. Comput. Sci. **44** (2004) 2167-2178.
16. Solov'ev, V. P., Varnek, A.: Anti-HIV Activity of HEPT, TIBO, and Cyclic Urea Derivatives: Structure-Property Studies, Focused Combinatorial Library Generation, and Hits Selection Using Substructural Molecular Fragments Method. J. Chem. Inf. Comput. Sci. **43** (2003) 1703-1719.
17. Goulon, A., Picot, T., Duprat, A., Dreyfus, G.: Predicting Activities without Computing Descriptors: Graph Machines for QSAR. Submitted to SAR and QSAR in Environmental Research.
18. Duprat, A.F., Huynh, T., Dreyfus, G.: Toward a Principled Methodology for Neural Network Design and Performance Evaluation in QSAR. Application to the Prediction of LogP. J. Chem. Inf. Comput. Sci. **38** (1998) 586-594.