An Introduction to Recursive Neural Networks and Kernel Methods for Cheminformatics

Alessio Micheli[1], Alessandro Sperduti[2], Antonina Starita[1]

[1]Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo 3, 56127 Pisa, Italy. E-mail:
micheli@di.unipi.it, starita@di.unipi.it
[2]Dipartimento di Matematica Pura ed Applicata, Università degli Studi di Padova, Via G.B.
Belzoni7, 35131 Padova, Italy. E-mail: sperduti@math.unipd.it

Abstract

The aim of this paper is to introduce the reader to new developments in Neural Networks and Kernel Machines concerning the treatment of structured domains.
Specifically, we discuss the research on these relatively new models to introduce a novel and more general approach to QSPR/QSAR analysis. The focus is on the computational side and not on the experimental one.

Keywords: QSAR/QSPR, Recursive Neural Networks, Kernel for Structures, Kernel Methods, Learning in Structured Domains, Machine Learning.

1. Introduction

Many biological and chemical problems are characterized by quite complex domains where the managing of relationships and structures is relevant to achieve suitable modeling of the solutions. Specifically, the Quantitative Structure-Property/Activity Relationship (QSPR/QSAR) approach allows an analysis of the interactions between chemicals and biological systems abstracted by a quantitative mathematical model, where the treatment of chemical structures (molecular graphs) plays a major role.

The treatment of these complex domains via a standard algorithmic approach is problematic since the current knowledge is insufficient to formulate a well defined algorithmic solution: data is noisy, uncertain, and partial. Moreover the computational complexity of many tasks in these domains is so huge that only approximate solutions can be sought. The Machine Learning (ML) approach is ideally suited to tackle complex problems where there is a lack of a clear theory explaining the molecular interactions, and the presence of noisy experimental data. In particular, ML has contributed with powerful data modeling tools that are able to approximate complex non-linear input/output dependencies. However, standard ML techniques are often restricted to the treatment of finite dimensional vectors of features. Traditional QSPR/QSAR approaches inherited the same assumptions that lead to a case-dependent extraction of features, guided by the knowledge of experts, to obtain a finite set of suitable structural descriptors of the molecules, i.e. a flat representation of the problem domain where a complete representation of the molecular structures is difficult.

Since molecules are more naturally described via a varying size structured representation, the study of general approaches to the processing of structured information has become the topic of many studies in ML. This research is aimed at proposing new methodologies (with respect to the traditional descriptor based approaches) that are able to directly exploit the molecular structures as input to the ML models.

The main aim of this paper is to review some new results concerning mathematical methodologies for *adaptive processing of structured domains*, discussing whether it is worth to predict biological activity and other molecular properties directly from structure. The main contribution offered by learning directly in structured domains is the shift of the paradigm from the property-to-property prediction to a real structure-to-property prediction. A direct structure-to-property prediction can be valuable when poor information is available on the properties of the molecules due to experimental or computational constrains or to the lack of problem knowledge.

Different ML approaches for structured domains have been explored, such as Graphical Models [66,47,1,24], Inductive Logic Programming [64,48,23], Recursive Neural Networks (RNN) [75,28,4], and lately Kernel Methods [79,18,73]. The specific aim of this paper is to give a brief tutorial on the last two approaches. Indeed, these are relatively young research branches, which are rapidly developing (see for example the special issue recently appeared on the Neural Networks Journal [36]). These approaches allow us to cover new methodological trends for the treatment of structured data and, specifically, to cover regression tasks, which are quite useful in drug design. Actually, due to our direct involvement in the development of Recursive Neural Networks, the presentation will be biased towards the neural network approach, even if basic concepts concerning kernel methods will be covered, and examples of simple spectral kernels for structures will be given.

In Section 2 we start by introducing basic concepts in Machine Learning, and by giving concrete examples of neural networks and kernel methods for dealing with “flat” vectorial representations in a supervised learning setting.

In Section 3 we discuss why structured representations are useful and what are the main issues in representing structured objects of varying size and shape. First we review the general notion of structure in a Computer Science framework. Then, the representation of chemical compounds is considered as an instance of such general framework where the specific representation is tailored to the specific application task. Since the examples of representation of chemical compounds given in Section 3.2 will help to ease the presentation of the methodologies introduced in the subsequent sections, we mainly follow an historical perspective, discussing some relevant issues concerning how to devise a structured representation of chemical compounds amenable to the application of the presented neural network models, basically resorting to subclasses of graphs. Many of these considerations, however, hold also for several kernels for structures. In this respect, new trends are also discussed.

Classes of structural transductions, i.e. functions that take as input structured objects, are described in Section 4, where we show how different computational tasks involving chemical compounds require different structural transductions. In the same section, we also show that a specific but useful type of structural transduction, i.e. supersource transduction, can be decomposed into two sub-functions, one performing the encoding of the input structure in an extended feature space (*encoding function*) and one computing the desired output from the obtained representation in the feature space (*output function*). This decomposition allows us to elucidate the role of neural networks and kernel methods in the implementation of a transduction system (Section 5). Basically, while kernel methods define a priori a static encoding function via the definition of a kernel for structures, and then learn the output function separately, neural networks simultaneously learn encoding and output functions. Through the adaptive encoding of structured input, the learning model can compute structural descriptors that are specific for the data and computational problem at hand. In other words, the map of (chemical) similarities among molecules is automatically adapted by the model to the task, according to the property/activity target values. Note that the process can consider both the 2D graph topology and the atom types (or the chemical functionalities). Hence, exploiting the richness of a structured molecular representation and avoiding the use of any fixed structure-coding (or similarity) scheme, a new perspective to the QSPR/QSAR studies is offered.

Implementations of other relevant types of structural transductions are also discussed, with particular emphasis on neural contextual transductions. How to learn structural transductions is discussed in Section 6, where the Recursive Cascade Correlation algorithm for neural networks is discussed as an example.

A structured representation is a quite flexible and powerful tool capable of including in a single item both atom-groups and their relationships (connections). It is also natural, according to the 2D molecular structure of the studied compound and to conventional rules used in chemistry, to give a unique representation to the molecular compounds. The recursive neural approach, however, imposes constraints for the data representation. In particular the direct treatment of undirected and cyclic graphs poses efficiency issues. Recently, the research in this field has been oriented to address the problem of dealing with general classes of graphs, including undirected and cyclic graphs (Section 7).

Some experimental results, recently appeared in the literature, supporting the efficacy of the discussed models are summarized in Section 8. The aim is to put in perspective the general methodology presented in the paper, showing that learning methods for structured domains not only are competitive with ad hoc feature-based learning methods, but they exhibit a much more flexible and general applicability to many different chemical domains and tasks. However it should be noticed that if the knowledge of the problem is sufficient to devise really relevant (structural) features and such features are available to build a model, the traditional feature-based approach still represents the best approach, especially due to interpretability issues. For new tasks, on the contrary, e.g. when relevant features are not know, the models presented in this paper offer the opportunity to predict directly from the structure.

Finally, in Section 9 we draw some conclusions, summarizing the main differences between the neural and kernel-based approaches.

2. Introduction to Machine Learning

In the following we present a brief introduction to some basic concepts and terms of Machine Learning (ML) theory.

ML deals with the problem of inferring general functions from known data. The ML methods are particularly useful for poorly understood domains where the lack of a clear theory describing the underlying phenomena, or the presence of uncertain and noisy data, hampers the development of effective algorithms for sollving the specific problems.

Let us describe the ML topics of our interest in terms of a few key design choices concerning the data, the computational tasks, the models, and the learning algorithms:

- *Data* is the set of “facts” available for the problem at hand. Examples from the application domain are represented in a formalism that should be able to capture the structure of the analyzed objects.
- The *task* defines the aim of the application. Tasks can be roughly partitioned into predictive (classification and regression) or descriptive (cluster analysis, association analysis, summarization, etc.). In the case of predictive tasks, associated to the available “raw” data there is a desired output, i.e. the values of the *target function* to be learned. The available desired output is in general called *supervision*, and it is either obtained through direct (and usually very expensive) measurements or from a domain expert. Learning involving this type of tasks is usually referred to as *supervised learning*.

As an example of this type of task, let us consider the prediction of the biological activity of a family of compounds with respect to some specific receptor. In this case, the available *training* data will be any compound belonging to this family for which the associated biological activity (supervision), encoded according to some relevant measure (e.g., the IC_{50}), is known. The predictive task will consist in predicting the biological activity for compounds of the family for which the biological activity is not known. We will discuss later a very important issue concerning the “correct” use of data with supervision. It turns out that it is not wise to use all the data for training, since, in this way, no guarantee about the quality of learning can be obtained. In general, it is wiser to keep some data with supervision out of the training process, so to use it to evaluate the quality of the training result.

When considering descriptive tasks, no explicit *target function* is given, however the choice of a specific learning model implicitly defines a functional that is minimized (or maximized) during learning. Unfortunately, for some descriptive learning models the exact form of this functional is not clear, and so it is not possible to describe the outcome of learning in a rigorous way. Learning in this context is usually referred to as *unsupervised learning*.

As an example of this type of tasks, let us consider the problem of discovering common chemical fragments within a patented family of compounds that are known to be active with respect to a specific receptor. Knowing these fragments could ease the task to produce a new family of drugs by devising variants of them that are both active and not covered by patent. In this case, the task is reduced to automatically extracting statistically significant (structural) features of the compounds in the family.

Note that the aim of learning for this type of tasks is not to induce the general form of a function knowing its value on a set of given inputs, but to discover correlations among inputs, i.e., in the instance of the task indicated above as “the set of common fragments”.

- The *model* (or *hypothesis space*) is characterized by the type of *hypothesis* used to describe the “solution” for the task. For example, in prediction tasks, an hypothesis is a function fitting the data of interest, while for descriptive tasks a hypothesis is an expression in a given language that describes the relationships among data.

A ML model is supposed to be used in two different states (the two phases can be interleaved): the *learning phase*, and the *prediction phase* (or *operational phase*). The learning phase corresponds to building the model. An hypothesis is constructed on the basis of a set of known data, the *training data set*. The training data constitutes the “experience” from which the model tries to learn from. Therefore, this data set should be a representative sample of the real distribution of the problem at hand. The prediction phase is the operational one, where the model is used to compute an evaluation of the learned function over novel samples of data.

Summarizing, a hypothesis is a specific function or description which can be implemented by the model, while the hypothesis space can be understood as the “portfolio” of functions (descriptions) from which the *learning algorithm* is allowed to pick a “solution”, to be used in the operational phase.

- A *learning algorithm* is used to learn the best hypothesis according to the data, the task and the class of considered models. It can be understood as a search, within the hypothesis space, for the hypothesis that returns the “best” fit with the data. Of course the meaning of “best” fit should be understood in terms of all the data that can be potentially observed, and not restricted to the data available for training.

2.1. Data Representation

According to the specific application domain and task at hand, data can be represented in different ways:

Flat data: any object in the input domain is represented by a fixed-size vector of properties (or *features*), according to an attribute-value language. The features can have real values (continuous attribute) or nominal values (discrete or categorical attributes).

For example, in traditional QSAR studies, chemical compounds are usually represented through a set of predefined (and hopefully relevant) numerical descriptors such as *physico-chemical properties* (e.g. polarizability, molar refractivity, hydrophobicity, etc.) and/or *topological indices* encoding specific morphological properties of the molecular graph (e.g. adjacency or distance matrix based indices, *chi* indices, etc.). It should be noted that in this case, while this kind of representation is quite easy to produce and to process, it does not fully reflect the “true nature” of the data, which naturally presents itself in a structured form. Of course, this may not be true in other application domains, where the data is naturally representable, through a fixed-size vector and it would be not appropriate, if not even detrimental, to use a more complex representation.

Structured data: objects in the input domain are sequences (e.g. temporal events, signals, proteins, DNA), hierarchical structures (e.g. trees), graphs (e.g. molecular graphs), relational data, multi-table representations, and so on.

2.2. Tasks

In the framework of supervised learning, two main families of tasks are commonly distinguished according to the different learning formulations and approaches adopted:

- *Classification*: means to approximate a discrete-valued function. The function maps an input into a M -dimensional decision space, where M is the number of categories or *classes*. For $M > 2$ we speak of multi-class problems. A special case is given by binary classification, with $M = 2$, also called "concept learning", within the *pattern recognition* framework. Examples of classification tasks within the drug design field are the discrimination between active/nonactive drugs, the prediction of toxicity categories, and the evaluation of positive/negative outcomes of bioassays.
- *Regression*: means to approximate a real-valued target function. The function maps an input into a continuous space. The problem may also be viewed as "curve fitting" (or "sub-space fitting" for multivariate outputs) in a multidimensional space. Examples concerning QSPR/QSAR analysis are the prediction of continuous values properties, as a temperature measured in Celsius or Kelvin degrees, a logP or an IC_{50} value.

Very common unsupervised tasks are:

- *Clustering*: or to determine "useful" subsets of an unclassified set of data sample, in the sense that members of the same subset share common features.
- *Dimensionality reduction*: or to reduce the dimensionality of data (represented through fixed-size vectors) by preserving "useful" information, such as topology or invariance with respect to some other predefined property of data.

2.3. Models

The panorama of ML models is quite large. The classes of hypotheses that can be considered include: equations, classification and regression trees, predictive rules (for association or prediction), distance-based models, probabilistic models (e.g. Bayesian networks), neural networks and kernel-based learning machines, e.g. Support Vector Machines (SVMs). Here, we briefly describe how the hypothesis space is defined for the last two models mentioned above.

Neural Networks

A neural network is a powerful data modeling tool that is able to approximate complex input/output relationships. According to the definition of hypothesis space given, a neural network realizes a mathematical parametrized hypothesis in the form $h(u) = g_W(u)$, where $u \in \mathbb{R}^n$ is the input pattern in the form of fixed-size vector, W is a set of real-valued parameters (*the weights*) whose values determine the specific computed function, and $g_W : \mathbb{R}^n \to \mathbb{R}^z$, were z is the number of output components. For instance, a two-layer feedforward neural network with a single output (i.e., z = 1) and m "hidden" units, computes a function of the following form

$$h(u) = \sigma(\sum_{j=1}^{m} w_j^{out} \underbrace{\sigma(\sum_{q=1}^{n} w_{jq}^{hidden} u_q)}_{x_j}) = \sigma(w^{out} \cdot \underbrace{\hat{\sigma}(W^{hidden} u)}_{X}) \qquad (1)$$

where $W^{hidden} \in \mathbb{R}^m \times \mathbb{R}^n$ is the weight matrix from the input to the so called hidden layer, $w^{out} \in \mathbb{R}^m$ is the weight vector from the hidden layer to the output layer (recall that in general we have $z > 1$),

$\sigma(\cdot)$ is a sigmoidal nonlinear *activation function*, e.g. $\sigma(u) = \frac{1}{1+\exp^{-u}}$, $\hat{\sigma}(\cdot)$ represents the component-wise application of $\sigma(\cdot)$ to the components of the input vector, and $x \in \mathbb{R}^m$ (or more precisely, for the $\sigma(\cdot)$ defined in the example, $x \in (0,1)^m$) is the so called hidden activity vector. In this case, the set of parameters W is constituted by both the elements of W^{hidden} and the elements of w^{out}.

Hence, the hypothesis space that characterizes the neural model is the *continuous* space of all functions that can be represented by assigning specific values to the parameters in W, i.e. the weight values of the given neural network architecture. This allows the model to represent a rich space of non-linear functions, making neural networks a good choice for learning discrete and continuous functions whose general form is unknown in advance. In particular, approximation theory (Theorem by Cybenko, see [38]) formally supports the generality of this class of hypotheses: this class of functions is sufficient to approximate continuous and derivable functions with an arbitrary small error, provided that a sufficient number of hidden units are available.

The computation of the function in eq. 1 is traditionally described in terms of a data-flow process performed by an interconnected set of computing units in the form of a network. Specifically, neural networks models can be characterized by the design of the *neural units* and the topology of the network of units, i.e. the layout of the *architecture*.

In eq. 1 each function of the type

$$out_j = \sigma(\sum_q w_{jq} i_q) \qquad (2)$$

can be seen as computed by an independent processing element j, with output out_j, called artificial neuron or simply *unit*. For the sake of notation, the bias parameters (a linear offset θ) is often included in the weight vector of each unit. Moreover the input vector is extended by a constant value, so that i_0=1 and $w_{j0}=\theta$.

The computation of each unit can be further decomposed as in the following:

$$net_j = \sum_q w_{jq} \cdot i_q \qquad (3)$$

$$out_j = \sigma(net_j). \qquad (4)$$

A graphical representation of the above equations is shown in Fig. (**1**).

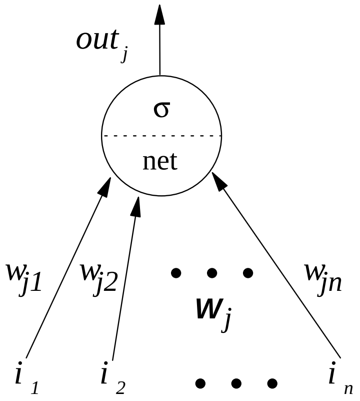


Fig. (1). A neural computational unit.

Adopting different designs for the neural units, i.e. using a $\sigma(\cdot)$ with a "step" shape, or a linear function, we obtain different neural models.

Note that, depending on the class of values produced by the network output units, discrete or continuous, the model can deal, respectively, with classification or regression tasks.

The *architecture* of a neural network defines the topology of the connections among the units. The two-layer feedforward neural network described in eq. 1 belongs to the well-know MLP (multi-layer perceptron) architecture: the units are connected by weighted links and they are organized in the form of layers. The input layer is simply the source of the input u that projects onto the hidden layer of units. The hidden layer projects onto the output layer (feedforward computation of a two-layer network). A graphical representation of this network, with two layers and one output unit (see eq. 1) is given in Fig. (**2**).

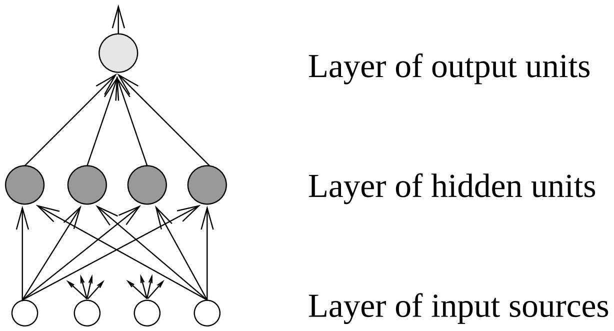


Fig. (2). An example of two-layers fully connected feedforward neural network with an input layer, a hidden layer, and one unit in the output layer.

It is known that the *expressive power* of neural networks is strongly influenced by two aspects: the number of units [1] and their configuration (architecture).

In the connectionist area a variety of different architectures have been studied (see e.g. [38]). The problem of optimizing the architecture has been addressed using different approaches, such as constructive learning algorithms, which grow the network topology during training, or pruning techniques, which after training try to prune "useless" units or connections. An approach related to pruning is Automatic Relevance Determination, which has been applied to neural networks, trained within a Bayesian learning framework, in QSAR studies [12]. Various techniques (such as the Cascade Correlation algorithm [27]) have been proposed to dynamically modify the network topology and its dimension as part of the learning algorithm. Constructive methods, also called growing methods, allow to start with a minimal network configuration and to add units and connections progressively, performing an automatic adaptation of the architecture to the computational tasks. Constructive learning algorithms are particularly suited to deal with structured inputs, where the complexity of learning is so high that an incremental approach is to be preferred.

A very good reference for neural network models and learning algorithms is the book by Haykin [38].

Kernel Methods

Kernel Methods can be described as a large family of models sharing the use of a *kernel function*. The idea underpinning the use of a kernel function is to focus on the representation and processing of pairwise comparisons rather than on the representation and processing of a single object. In other words, when considering several objects, instead of representing each object via a real-valued vector, a real-valued function $K\ (obj_1,obj_2)$ is defined in order to "compare" the two objects obj_1 and obj_2. In this way, a set of n objects is fully described by the $n\times n$ matrix of pairwise comparisons K $K_{i,j}=(obj_i,obj_j)$ denominated *kernel matrix*. Not all functions can take the role of $K(\cdot,\cdot)$, since specific mathematical conditions should be satisfied to get a "comparison function". For example, comparing object obj_i versus object obj_j should return the same result as comparing object obj_j versus object obj_i. Thus $K(\cdot,\cdot)$ should be a *symmetric* function. In the majority of the kernel methods it is also required for $K(\cdot,\cdot)$ to be *positive definite*, i.e. for any $n > 0$, any choice of n objects, $obj_{1,\ldots,}$ obj_n, and any choice of real numbers, $c_{1,\ldots,}c_n$, the following condition $\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j K(obj_i, obj_j) \geq 0$ holds. As we will see in the following, this additional condition allows to ease the learning process. Moreover, under the above conditions it is possible to show that

[1] Actually the network capabilities are influenced by the number of parameters and recent studies report also the dependencies on their value sizes [5]. See discussion on VC-dimension in Section 2.4.

for any kernel K defined on an object space O, there exist a (Hilbert) space F (usually called *feature space*) and a mapping $\Phi : O \rightarrow F$ such that K is the inner product in F of the images of the objects via the Φ mapping, i.e. $K(obj_i, obj_j) = \Phi(obj_i) \cdot \Phi(obj_j)$, for *any* $obj_i, obj_j \in O$.

As an example of kernel function, let us consider objects that are real-valued vectors $u, u' \in \mathbb{R}^n$. Then the Gaussian radial basis kernel function is defined as

$$K_{RBF}(u,u') = \exp\left(-\frac{d(u,u')^2}{2\nu^2} \right)$$

where ν is a parameter and $d(\cdot,\cdot)$ is the Euclidean distance.

In kernel methods, the hypothesis space H is typically defined on the basis of a finite set of reference objects, $obj_{1,\ldots,}\ obj_n$, as the set of functions that can be computed according to

$$H = \{h(\cdot) \mid h(obj) = \sum_{i=1}^{n} \gamma_i K(obj, obj_i), \gamma_i \in \mathbb{R}\}.$$

The strategy to focus on pairwise comparisons shared by all kernel methods leads to some advantages that are worth mentioning here. First of all, it is typically easier to compare two objects instead of defining some abstract space of features where to represent a single object. This is especially true if the objects are complex and possibly of different size. A second advantage is the fact that the hypothesis space is defined on the basis of the kernel matrix, which is always $n \times n$, i.e. independent from the complexity and size of the objects involved. Finally, when considering positive defined kernels, it is possible to exploit the so called *kernel trick*: any (learning) method defined for vectorial data which just exploits dot products of the input vectors, can be applied implicitly in the feature space associated with a kernel, by replacing each dot product by a kernel evaluation. This allows to “transform” linear methods into non-linear methods.

Models for Structured Data

Most of the known ML models are designed for data represented in flat form. However an attribute-value language is not able to capture in a natural way the complexity of a structured domain. ML models have been specifically investigated to handle structured data. This area of ML is usually referred to as “relational learning”, “relational data mining”, or “learning in structured domains”, according to the specific representation formalism and learning model adopted to represent data and to perform training, respectively. For example, Inductive Logic Programming (ILP) [64,48,23] has been proved successful in relational data mining tasks involving chemical compounds (e.g. [45,76]), while Recurrent and Recursive Neural Networks, as well as Kernel Methods for Sequences and Structures, have obtained state of the art results in structured domains within Bioinformatics and Cheminformatics, as we will see in the following.

2.4. Learning

Learning algorithms perform a (heuristic) search through the hypothesis space looking for the “best” hypothesis (or hypotheses) explaining the (training) data. For many models this search is eased by the parameterization of the hypothesis space, which is fundamental for hypothesis spaces of infinite cardinality, e.g. connection weights in a neural network.

The definition of “best” hypothesis should be understood with respect not only to the available data but also to novel samples. In the framework of supervised learning it is easy to evaluate the quality of an hypothesis with respect to the training data since it is sufficient to evaluate, according to some predefined metrics, how far the output generated by the hypothesis is from the available desired value for each input. Unfortunately, selecting the “best” hypothesis only on the basis of the training data does not guarantee that the hypothesis is the best one also for novel samples. In fact, a hypothesis selected in this way may specialize too much on features of the training data that are not shared by samples extracted from the full input space. Thus, the main goal of machine learning is to devise learning algorithms, or more in general learning methodologies, which give guarantees on the quality of the selected hypothesis on the full input domain and not only on the training data.

More formally in the ML framework a supervised learning problem is viewed as a function approximation problem (approximation of an unknown target function), where the available data set *D* (the *training set*) is represented as a set of pairs $\{(\imath_i, d_i)\}_{i=1}^{l}$, called examples, where $\imath_i$ is an input sample (called instance; either a vector or a structured input) and d_i is the corresponding desired value given by a teacher to represent the observed values of a function *f*() that models the real source of data. The intrinsic error $(d_i - f(\imath_i))$ is the noise term carried by the teacher.

The learning problem consists in selecting a hypothesis *h* that approximates the desired responses d_i over the data set in an optimal fashion, e.g. minimizing some risk functional. In particular, we want the function *h* to be a reasonable estimate of the functional relation between unseen pairs $(\imath, f(\imath))$ (prediction or *generalization* property). The error of a hypothesis *h*, i.e. the risk associated to it, is given by a loss or cost function *L*() that measures the distance between *d* and $h(\imath)$. A common example of loss is given by the quadratic error function:

$$L(h(\imath), d) = (d - h(\imath))^2 . \qquad (5)$$

The average error over all *D* is the, so called, *expected risk R*. If we assume that a probability distribution $P(\imath,d)$ exists and is known governing the data generated and the underlying functional dependences, *R* can be expressed as

$$R = \int L(d, h(\imath)) dP(\imath, d) . \qquad (6)$$

In this framework

$$h^{opt} = arg\ min_{h \in H} R . \qquad (7)$$

The problem in minimizing *R* is that the only available information to the learner is the training set, which just constitutes a small finite set. How a learner uses the training set in order to minimize *R* is called the *inductive principle*. The inductive principle is a general prescription for obtaining the estimate of h^{opt} from the training set. It tells *what* to do with data. Given an inductive principle, the learning algorithm tells *how* to use the data to obtain the estimate.

One of the most popular inductive principles is Empirical Risk Minimization [79]. It ammount to searching a solution minimizing the functional

$$R_{emp} = 1/l \sum_{i=1}^{l} (d_i - h(\imath_i))^2 \qquad (8)$$

where R_{emp} is the *empirical risk*, or *training error*, by depending on the values of free parameters (omitted to simplify the notation).

The search is performed by the training algorithms looking for the best values for the free parameters of the model, i.e. searching through *H*: the hypothesis that minimizes R_{emp} (fitting of the training examples). The goodness of the resulting hypothesis *h* is measured by the generalization error, given by the distance of $h(\imath)$ from $f(\imath)$ for any possible $\imath$, according to the underling probability distribution. An estimation of the generalization error (accuracy) is given sampling *D* to extract a finite set, called *test set*, using the same probability distribution used to generate the training set.

Learning in Feed-forward Neural Networks

Training of neural networks provides a first example of a learning strategy based on parameters optimization. In fact, for a neural network the hypothesis space is a continuously parametrized space. Moreover, the standard loss or error function for a neural network, defined as in Equation 5, is differentiable with respect to the continuous parameters of the hypothesis function. As a result, the learning algorithm of a neural network can be based on the minimization of the error function via gradient descent techniques. For multi-layer neural networks, an efficient implementation of gradient descent is obtained by *back-propagation* [70], the most popular among the supervised training algorithms for this class of networks. In particular, the back-propagation training algorithm is an iterative procedure where the values of the errors are back propagated from the output neural units to the input units, changing the weights proportionally to the influence they had on the total error R_{emp} (eq. 8). At each iteration the weight values, w_{ij}, are updated according to the learning rule $w_{ij} = w_{ij} + \Delta\, w_{ij}$, where

$$\Delta w_{ij} = -\eta \frac{\partial R_{emp}}{\partial w_{ij}} \qquad (9)$$

and $\eta > 0$ is the gradient descent step size, also called learning rate.

Despite of its implementation simplicity and usefulness, back-propagation has the classical disadvantages of any gradient descent method (local minima, dependency of the final hypothesis on the initial weights condition, choice of the "right" value for the gradient descent step, etc.).

Learning in Support Vector Machines

Another notable example of learning algorithm is given by the most popular approach within kernel methods, i.e. Support Vector Machines. Here we briefly introduce the Support Vector Regression (SVR) method for estimating real-valued functions when objects are vectors $u_i \in \mathbb{R}^n$, as described in [79]. It is based on the solution of a quadratic optimization problem that represents a tradeoff between the minimization of the empirical error and the maximization of the smoothness of the regression function.

Given a regression task, described through a training set $\{(u_i, d_i)\}_{i=1}^{l}$ where $d_i \in \mathbb{R}$, let us consider as regressor (hypothesis) a linear function $h_w(u_i)= w \cdot u_i+b$, where $w \in \mathbb{R}^n$ is the weight vector which parametrizes the hypothesis space, i.e., each specific vector w describes a specific hypothesis $h_{\hat{w}}()$.

As with neural networks, the empirical error is minimized. However, holding the same global error, in SVR it is preferred to have a "reasonable" approximation of the desired values d_i on as many inputs as possible instead of exactly reproducing the desired values for some inputs and obtaining a poor approximation of the desired values for the remaining inputs. The motivation for this choice is rooted in results of Statistical Learning Theory [79] that state that the probability that the empirical risk R_{emp} will converge to the true risk R with the increase in the number of training samples is higher, provided that the norm of the weight vector is minimized. In SVR, the requirement for a "reasonable" approximation is coded by introducing the constraint that for each input u_i we should have $| d_i - h_w(u_i) | \leq \varepsilon$, where ε is a small positive constant representing the degree of error tolerance we allow. A nice aspect of this requirement is that it can be represented by two linear constraints, i.e., $(d_i - w \cdot u_i - b) \leq \varepsilon$ and $(w \cdot u_i + b - d_i) \leq \varepsilon$, which allows us to define the learning problem via the following quadratic optimization problem:

$$\min_{w,b} \frac{1}{2} \| w \|^2$$

$$\text{subject to:}$$

$$\forall i \in \{1, \ldots, l\}$$

$$d_i - w \cdot u_i - b \leq \varepsilon$$

$$w \cdot u_i + b - d_i \leq \varepsilon$$

where the minimization of the (square of the) norm of the weight vector is used to seek a "smooth" regressor. This procedure is called *regularization* (see [38,79]).

Unfortunately, the above problem may not have solutions in the case in which the desired function we wish to approximate cannot be fully described by a linear function, because of noise in the observations or because of its particular nature. Thus, in order to render practical the above formulation, we have to allow for some violation of the above constraints, i.e. we may allow that for some inputs, the ε-approximation requirement is not satisfied. However, we would like to minimize both the number and the “amount” of such violations. This can be done in different ways. A very popular way of doing it is by introducing a loss function that penalizes an error, i.e. a violation of the ε-approximation, according to the difference between the actual approximation error and the tolerance ε. In detail, if for some u_i we have $| d_i - h_w(u_i) | > \varepsilon$, the weight vector w is penalized by the quantity $\xi_i =| d_i - h_w(u_i) | - \varepsilon$. Adapting this idea to the above problem, we get the standard SVR model for a 1 -norm ε-insensitive loss function (see [17]):

$$\begin{gathered}
\min_{W,b,\xi,\xi^*} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \\
\text{subject to}: \\
\forall i \in \{1, \ldots, l\} \\
d_i - w \cdot u_i - b \le \varepsilon + \xi_i \\
w \cdot u_i + b - d_i \le \varepsilon + \xi_i^* \\
\xi_i, \xi_i^* \ge 0
\end{gathered}$$

where the ξ_i are the so-called *slack variables* accounting for underestimation of the desired values, while the ξ_i^* account for overestimation of desired values. Such variables are introduced in the linear constraints so to guarantee an admissible solution in any case. It is however required to minimize their values in order to minimize the violations of the ε-approximation. Because of the introduction of this additional minimization requirement, a new constant C is introduced in the objective function to define the tradeoff between minimization of the empirical error, $\left(\sum_{i=1}^{n} (\xi_i + \xi_i^*) \right)$, and maximization of the smoothness of the regression function obtained by minimization of the weights vector norm $\left(\frac{1}{2} \| w \|^2 \right)$.

The solution of the above problem is obtained by solving its dual formulation that can be obtained by the application of the Lagrangian multipliers technique (see [17]):

$$\begin{gathered}
\max_{\alpha,\alpha^*} - \varepsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} d_i (\alpha_i - \alpha_i^*) + \\
- \frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) u_i \cdot u_j \\
\text{subject to}: \\
\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0 \\
\alpha_i, \alpha_i^* \in [0, C].
\end{gathered}$$

It is important to note that the couple of *dual variables*, α_i and α_i^*, are introduced for each input vector and that the weight vector, w, can be expressed in function of these variables, i.e. $w = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) u_i$. Moreover, the optimal solution of the dual problem[2] very often requires that only few of the dual variables are nonzero. In fact, α_i and α_i^* will be nonzero only for the u_i which do not strictly satisfy the ε-approximation, i.e. $| d_i - h_w(u_i) | \ge \varepsilon$. The input vectors corresponding to these nonzero variables are called *support vectors* since these are the only contributing to define the

[2] The optimal value for b can be derived from the Karush-Kuhn-Tucker (KKT) conditions on optimality.

optimal weight vector *w*. Thus, given an optimal solution $\bar{\alpha}_i, \bar{\alpha}_i^*$, the SVR output function can be written as $\bar{h}(u) = \sum_{i=1}^{l} (\bar{\alpha}_i - \bar{\alpha}_i^*) u_i \cdot u + b$.

At this point, it is not difficult to recognize that the obtained solution can be cast in the form given for kernel methods, i.e. given the vectors $u_1 \equiv \begin{bmatrix} u_1 \\ 0 \end{bmatrix}, \ldots, u_l \equiv \begin{bmatrix} u_l \\ 0 \end{bmatrix}$ and the additional vector $u_0 \equiv \begin{bmatrix} 0 \\ b \end{bmatrix} \in \mathbb{R}^{d+1}$, and that it is possible to express $\bar{h}(u)$ as $\sum_{i=0}^{l} \gamma_i K(u_i, u)$, where $K(u_i, u) = u_i \cdot u$, $u \equiv \begin{bmatrix} u \\ 1 \end{bmatrix}$, $\gamma_i = (\bar{\alpha}_i - \bar{\alpha}_i^*)$ for $i > 0$, and $\gamma_0 = 1$. Moreover, observing that the formulation and solution of the dual problem only depend on dot products, it is possible to learn a *nonlinear regressor* by substituting the dot products between vectors with suitably defined kernels functions, e.g. the polynomial kernels $K(u_i, u) = (u_i \cdot u)^p$ with degree $p \in N$ ($p > 1$). Thus, the formulation for a general kernel $K(\cdot,\cdot)$ becomes

$$\max_{\alpha,\alpha^*} - \varepsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} d_i (\alpha_i - \alpha_i^*) +$$
$$- \frac{1}{2} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) K(u_i, u_j)$$
$$\text{subject to :}$$
$$\sum_{i=1}^{l} (\alpha_i - \alpha_i^*) = 0$$
$$\alpha_i, \alpha_i^* \in [0, C].$$

One problem with Support Vector Machines is how to choose the “right” value for *C*. In practice, different values for *C* are tested and the one returning the best performance according to some criterion (see below) is selected.

Other Learning Issues

Statistical Learning Theory, mentioned above, allows to formally study the condition under which the empirical risk R_{emp} can be used to drive an approximation of *R* and allows to derive some bounds on the generalization error, i.e. the expected risk. Without including any details, in the framework of binary classification, we can say that, with probability $1-\delta$, the following inequality holds (VC-bonds):

$$R \leq R_{emp} + \varepsilon(l, VC, \delta) \tag{10}$$

where ε is a function of the number of training examples *l* and of δ, and *VC* is a non-negative integer called the Vapnik Chervonenkis dimension (*VC-dimension*) [79]. The VC-dimension measures the capacity, or complexity, or expressive power of *H* by the number of distinct instances of data set that can be completely discriminated using *H*. *H*, of the type seen above, including neural networks, show a VC-dimension that is proportional to the number of parameters; however this is not true in general. The intuition behind the VC bound is that the deviation between expected risk and empirical risk decreases with the number of data points *l* (dimension of the training set) but increases with the VC-dimension. On the other hand, the training error decreases as the capacity or VC-dimension is increased. Given a fixed number of training examples, the general result is that the increase of the model capacity can reduce R_{emp} at the expenses of the generalization performance (*overfitting*). Therefore, the challenge in solving a supervised learning problem is to obtain the best generalization performance by matching the machine capacity to the available amount of training data for the problem at hand. It is worth to note that the VC-dimension is a quantitative capacity measure. It measures the richness of the class of functions but it cannot delineate the class of functions implemented by a model in terms of representation of the input domain. Clearly, if the target function cannot be described in the hypothesis space due to the initial language choice, the model cannot yield a consistent solution. Two classes of functions can have the same VC-dimension however they may be able to properly represent different objects. In this view we cannot use the above concept to fully describe the computational properties of the proposed models.

The knowledge acquired in the learning phase should allow the model to predict with reasonable accuracy the correct response for previously unseen data. The estimation of this accuracy is the critical aspect of each ML application and the rational basis for the appropriate measures of the model performance. The *holdout* method is the most used approach for this estimation: a set of known data independent of the training set, called *validation data set* is used to evaluate the quality of prediction. An appropriate measure of the performance over the validation set can estimate the accuracy (or the generalization capacity) of the model. Different statistical techniques can be used to improve the estimation accuracy: e.g. *k-fold-cross validation*, *bootstrapping*, etc. If the performance on the validation set is used to choose among different models or different configurations of the current model (i.e. it is used to drive the building of the predictive model), another set of data called *test set* is used to assess the final prediction accuracy of the system.

Committee and ensemble techniques can be used to improve the performance of the single models (e.g. [11]).

Another related discussion regards a crucial trade-off concerning the hypothesis space expressivity. The choice of the hypothesis space involves some basic question entailing what the model can compute and what the model can learn. If the model cannot compute (represent) the target function it also cannot learn it. On the other hand it is often the case that models that are capable of a certain kinds of computation cannot easily be trained from examples to perform it. Since the model defines the class of hypothesis considered in the learning procedure, it specifies the computational power of the method. Anyway, a model with a high expressive power can lead also to problems of efficiency in the search of solutions. The search of a balance between the expressive power of the model and the computational complexity of the learning algorithm is a critical issue of many ML methods.

To further characterize the search of the best hypothesis, it is possible to describe for each method the constraints imposed to the learner, referred to as *inductive bias* (or simply *bias*). The representation of the hypotheses, entailed by a specific ML model, defines the hypothesis space that the learner can explore to find *h* (*language bias*). The strategy adopted by the algorithm to search *h* over the hypothesis space is the *search bias*. Furthermore another decision concerns adding a strategy to control overfitting for a specific training dataset (*overfitting-avoidance bias*). More formally, the inductive bias is the set of assumptions made by the learner in order to constrain the problem: the addition of such assumptions allows to obtain, deductively, the inductive result of the learner on the training set. It must be noted that these assumptions are necessary to extract regularities, i.e. a learner without bias cannot extract any regularities from data and reduces itself to a lookup-table system with loss of generalization properties [63].

As seen above, there are benefits by limiting the complexity of the model. Practical approaches can be characterized by the bias that constrains the problem. The reduction of the hypothesis space, imposing language bias, can be well-motivated by a knowledge-driven process. If there is an a priori knowledge on the expected solution, it is possible to reduce effectively the search space. For example “regularization theory” is a nice framework in which the search of hypothesis is subject to a smoothness constraint.

On the other side, different ML methods (models and algorithms) correspond to different inductive biases. There is no universal “best” learning method. The lack of inherent superiority of any ML methods is formally expressed in the so called “no free lunch theorem” [22]. Each method is more or less suitable according to the specific task at hand. In particular, the language can be more or less appropriate to describe the relationships among the data of the application. Moreover, comparison parameters, such as predictive accuracy, speed (generating and using the model), robustness, scalability and interpretability can be considered to evaluate the quality of the method.

In the following section we will argue that tasks involving chemical compounds are best addressed by naturally representing compounds as structured objects and we will elaborate on which and how structured information can be encoded. Then we will introduce different types of functions defined on structured domains, that we will call *structural transductions*, and we will describe how each of them can cover a different task. Finally, we will present two quite effective approaches for implementing and learning structural transductions: Recursive Neural Networks and Kernel Methods for Structures.

3. Structured Domains

As we already mentioned above, computational tasks involving chemical compounds are best described in the framework of structured domains, i.e., domains where each entity is composed of several components related to each other in a non trivial way, and the specific nature of the task to be performed is strictly related not only to the information stored at each component, but also to the structure connecting the components.

In this section we discuss, from a very general point of view, i.e. disregarding the specific nature of chemical compounds, several issues that arise when considering the representation of structured objects. Such issues are important since they contribute to the definition of specific classes of computational models, which are not specific for chemical compounds, but which can profitably be applied to chemical compounds, provided that the relevant chemical structural information is represented accordingly to the representational constraints imposed by the specific models. E.g., as we will see in the following, it is in fact possible to successfully apply general computational models defined on trees to prediction tasks involving families of chemical compounds which turn out to be representable as sets of trees with no relevant loss of structural information.

In general, an instance of a structured domain is not simply represented by a real-valued vector, but by a more complex data structure. For example, we may consider a single vertex (labeled by a symbol or more in general by a vector), sequences of real vectors or strings of symbols, trees (in the form of free trees or k -ary trees, see Section 4), directed acyclic graphs and directed positional acyclic graphs (DPAG), directed cyclic graphs, undirected graphs (see Fig. (**3**)).

Note that in the k -ary tree the directions on the edges are given starting from a particular vertex, the root, and a position (up to degree k) is given to each edge leaving from the nodes.

Sequences, rooted k -ary trees and DPAGs are directed and hierarchical structures. A supersource vertex can be defined for the directed structures as a vertex from which all the other vertexes can be reached via a directed path. For the sequence, the rooted k -ary tree and the DPAG in Fig. (**3**), the supersource is the vertex with label “A”.

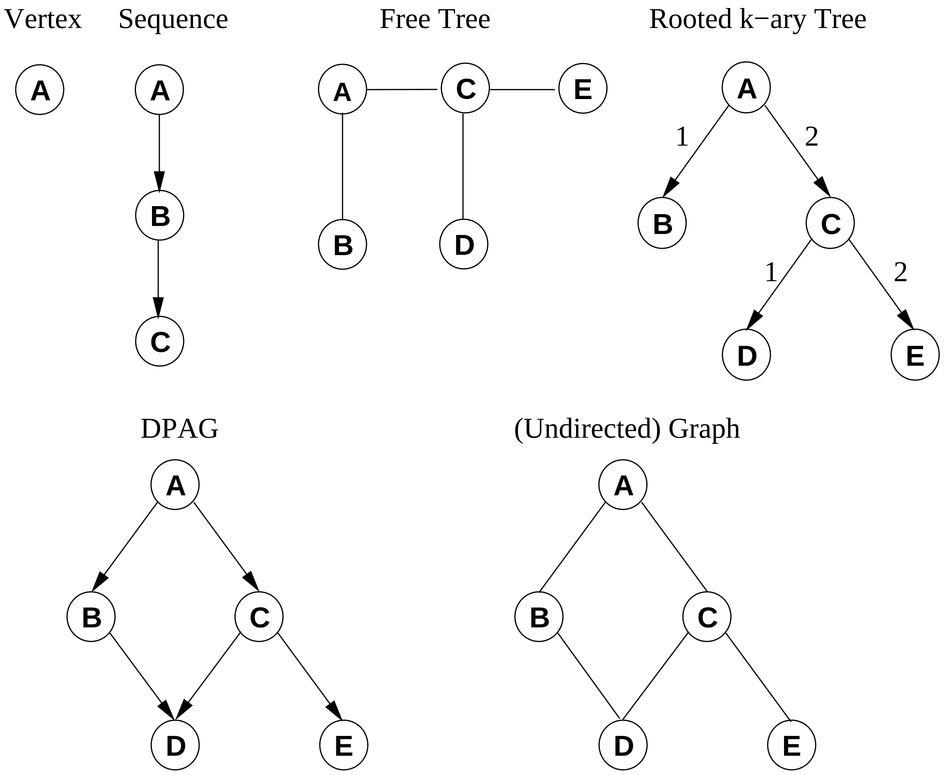


Fig. (3). Examples of some structured representations.

Of course, in our view, a non-structured domain can always be considered as a special case (restriction) of a structured domain. Thus a domain of real valued vectors can be seen as a restriction of a domain of sequences of real valued vectors (e.g., it is a domain of sequences, all of size 1), which in turn can be considered as a restriction of a domain of trees (i.e., a sequence can be seen as a tree where all the nodes have a single child, except for the single leaf which has no child and it corresponds to the last element of the sequence), which in turn can be considered as a restriction of a domain of directed acyclic graphs, and so on. This means that, focusing on structured domains, does not rule out the possibility to exploit the traditional learning models defined for vectorial data: traditional vector-based approaches are just specific instances of a more general structure-based framework. In the following we discuss, from a machine learning perspective, the main issues concerning the representation of structural information.

3.1. Issues about Representation of Structural Information for Machine Learning

The first design choice for a learning system is the collection and representation of the problem data. The set of represented data constitutes the training experience from which the system will learn. The type of information available can have a significant impact on the qualitative response of the learner. The instance of data entry for the system is a quantitative or structural description of an object referred to as *pattern*. The most frequent, and often convenient, way to represent the data is in the form of a pattern vector with a fixed dimension (or tuple of constants). The dimension of the vector corresponds to the number of measurements, or features, or attributes. Here, this form is referred to as “flat” because there is no hierarchical arrangement of the elements of the vector. The language used to describe patterns is called attribute-value language. Primitive elements occurring in a variable number of instances, with a sequential ordered structure, are in principle representable in a flat form. However, the characteristics of sequential components are best described by a string (or list) of symbols, i.e. the simplest case of structure relationships. Unlike the flat description, this type of representation yields patterns composed by primitive elements whose meaning is related to the basic structure of the input data. A more powerful approach is obtained through the use of tree representations. Tree structure is the common form in which to convey hierarchical information.

More complete ways to represent relationships between primitives lead to the general concept of graph. Labeled graphs may represent in the same structure both vector patterns and their relationships. The choice of the appropriate representation, considering both the selection of the representational schemes, e.g. among the ones presented above, and the selection of the set of distinguishing features that convey information, is strongly problem oriented. In the case of structured representations, besides the domains where the patterns are naturally described by resorting to structural relationships, there are problems in which the structured nature of the data is not immediately recognizable but a structural representation is convenient. The reason may be that of efficiency: for example, an improved representation can be obtained reducing the amount of data required when the structure allows to consider the repetitive nature inherent in data or when it is possible to extract invariant elements, like in image analysis applications. Other reasons to adopt structured representations are related to the possibility to increase the information contents by using relationships. A basic problem is however to define a suitable strategy to extract the structure from data. Even in the case of domains that can be naturally described by structures we have to decide the level information of details, for example which type of information can influence the topology or can be included in the graph vertexes, which information is represented by the label of the node, and so on. The case of Chemistry gives us a natural example of a problem characterized by a set of structured objects, i.e. the 2D graph of chemical molecules. In that case, the graph represents a suitable model to represent in a very general and abstract form, in the same graphical paradigm, both the information at atom and bond level and, potentially, the background information of experts that allows us to identify functional groups in the selected compounds.

When using structured data to represent the input patterns, learning has to take into account:

Information type and contents:

- information related to each vertex (typically encoded by a fixed-size label);
- single connections between vertexes of the structure;
- local information, i.e. the relevant information which can be extracted by a specific part of the whole structure. To give an example, we may consider the occurrence of a specific vertex or specific substructures in the pattern, or even a specific substructure occurring in specific positions, etc.
- global information, i.e. the information that inherently depends (also) on the whole topology of the structure;
- target information, i.e. the information that collects the desired response of the learning system. Such information can be available (supervised learning) or not explicitly available (unsupervised learning); moreover it can be defined at different levels, i.e. the target can be used to label the global structure, or some substructures or each vertex of the structure. Discrete or continuous values can characterize different tasks.

Variability:

- size of the structure, which depends on the number of vertexes, the out-degree, the depth, etc.
- relative arrangements of single vertexes inside the structure topology.

In order to achieve a suitable processing of structure information, a careful evaluation of the previous aspects should be considered. The main consequences can be briefly sketched in the following points:

1. The representation of structures using a fixed-size attribute-values language is generally inadequate: it is unnatural, difficult, incomplete or, otherwise, inefficient. Typically, the representation scheme is decided a priori and cannot change automatically according to the task, i.e. it is not adaptive.
2. Learning is influenced by the growth of the proportion between the huge combinatorial number of possible data examples (and thus the need to use a huge hypothesis space for learning) and the presence of a finite and small training set.

First attempts to deal with structured domains used a predefined mapping from structure to attribute-value representations. This mapping often destroys the explicit information carried by the relationships among the components of data described in the structure. Moreover the design of a predefined mapping of structure has drawbacks. In fact, since the mapping is problem dependent, it is valid only for specific tasks. In computational chemistry, for example, topological indexes (see, for example [9]) are mathematically defined for chemical graphs. This approach implies the proper treatment of the following issues: *i)* the uniqueness of representation, *ii)* the relevance of the selected attributes, and *iii)* the efficiency of the procedure to select and to compute the relevant features on which the mapping is based on. Moreover, the selection of features is strongly task-dependent, and often requires the availability of an expert in the field or an expensive trial and error approach. More in general, any a priori definition of such mapping is constrained to be either specific and knowledge-dependent, and therefore not extendible to different problems, where the relevance of the established features may change dramatically; or as general as possible, and therefore requiring different structures to be mapped into different representations. Unfortunately, the property of uniqueness is not a trivial problem. For example, it is related to the isomorphism problem between graphs, and it may lead to representations scarcely correlated with the relevant properties, making the learning task difficult.

In principle, the encoding of a complex object by an object of a lower degree of complexity can preserve all the information but at the expenses of a remarkable increase in the size of the representation: in order to be complete, a fixed-size vectorial representation should be dimensioned to the maximum size allowed for the graph, and therefore it is potentially wasteful. We can also mention that the vectorial representation should preserve the information on the edges, for example by using an incidence matrix, which requires a quadratic increase in the dimension of the representation. Another representation scheme may be obtained by encoding graphs through sequences. For instance, in the case of binary-trees, a representation based on nested parenthesis is a way of creating a sequential representation that makes it possible to reconstruct the tree. Since the number of vertexes explodes exponentially with the height of trees, even small trees give rise to long sequences. Furthermore, the sequential mapping of data structures is likely to break some nice regularity inherently associated with the data structure (that can be relevant for the learning task). In the ML context, these inefficiencies of the representation exacerbate learning problems connected with input dimension, especially for the models whose input dimension is related to the number of free-parameters, such as in the neural network models.

Point 2. concerns an important issue of ML theory, i.e. the search for the “right” trade-off between the expressiveness of hypothesis space and efficiency, i.e. computation time and number of examples required to learn. The high combinatorial number of possible data examples cannot always be easily compensated by a high cardinality of the training set. This induces a stronger need to impose conditions and constraints in the ill-posed problem to find the correct mapping in a potentially large hypothesis space. This need for parsimony is solved in different ways by the different approaches to structured domain. A specific solution provided by Recursive Neural Networks, “solving” the parsimony issue by a recursive weight-sharing technique, will be discussed in the following.

3.2. Representing Chemical Compounds

The representation of molecules by graphs is a current research issue. Chemical graphs are clearly recognized as a flexible vehicle for the rich expression of chemical structural information. In fact, the representation mapping between molecules and graphs is (almost) bijective and most of the molecules used for experimental purposes can be described by simple molecular graphs. Of course care must be taken not to rely on too simple graph representations, as in general chemicals show also relevant 3D features, flexibility, tautomerisms, and even incompletely defined structures, whose explicit representation in the graph representation, however, does introduce items which for the majority of the compounds are not used. For this reason, in order to optimize the approach, the class of graphs to be used should be tailored to the specific application domain at hand and to the constraints of the ML models. In order to give concrete examples of this issue, let us introduce a specific approach for representing chemical compounds, based on the experience of our research group using Recursive Neural Networks, which shows how it is actually possible to reach a trade-off between the expressiveness of the structured representation of the chemical compounds and the complexity of the ML model.

We investigated the feasibility of a structured representation in Cheminformatics starting from the class of directed positional acyclic graphs (DPAGs), which can be naturally treated by Recursive Neural Networks (RNN). This choice introduces also an historical perspective, showing the advantages and limitations of the first approaches for the direct treatment of structures for Cheminformatics, which progressively leads to further approaches for the treatment of general classes of graphs. To this aim we discuss in the following the characteristics of a molecular representation based on this subclass of graphs.

Rules

The basic ingredients of a molecular representation based on k -ary trees and DPAGs (see Fig. (**3**)) can be summarized as in the following:

- we need to devise a set of atomic groups, usually made by single atoms or functional groups. The groups are represented by the graph vertexes and they are connected to the other vertexes according to the molecular topology. Each devised group is labeled by a numerical tuple. Usually the label are categorical attributes that distinguish among different atomic/group symbols;
- we need to specify a supersource (or start) vertex[3]: this can be done according to the standard conventional rules used for the chemical nomenclature [41], and/or according to a priority scale on the functional groups based on the designer preferences;
- we need to specify an order or position of substructures: this can be done according to the nomenclature rules (e.g. for the position of the substituents in a homologue series), or according to a priority scale that reflects chemical information or, finally, according to some rules devised by the designer to convey information related to the 3D molecular configuration (including stereochemical configurations);
- finally cycles have to be represented, usually as a specific labeled atomic group, or rewriting the original graph into a hierarchical structure (see e.g. the convention used by SMILES [83]) or, in the form of DPAG (cycles with oriented edges).

Rules that allow us to define the structure representation according to the above ideas have been specified for different application domains [9,61,62,54,6,7,20,21]. This should prove the flexibility of such rules to adapt to drastically different problems, allowing the designer to exploit the highly abstract and graphical description offered by the structured domain. Of course, defining a data structure is a more general task than dealing with data formats. Hence, the issue on data formats can be left to a further phase without losing the generality of the structured data. In particular, the designer can decide the amount of molecular details that need to be explicitly described (including prior knowledge in such representation). Defining the information represented by the vertexes (e.g. atoms and chemical groups), the designer can easily specify the "granularity" (or the levels of abstraction) of the structured representation. A further degree of flexibility is offered by the label encoding. Specifically, giving a numerical code for each vertex label of the domain it is possible to specify the degree of similarity among the different type of vertexes (assuming some prior knowledge): e.g., orthogonal vectors can be used when we like to distinguish labels without taking any assumption for properties shared among the vertexes. Besides, the label can be also used to convey other information corresponding to features selected by the designer, including descriptors and other properties of interest or a graph representation based on the 2D molecular structure with

[3] A supersource is a vertex of the graph from which all the other vertexes can be reached via a directed path.

various degree of abstraction, which can be tailored by the designer to define the amount of molecular structure that needs to be explicitly described.

Examples

In order to clarify the above discussion, here we consider the representational issues concerning two paradigmatic QSPR and QSAR applications dealt with the RNN approach [9,61]. The applications concern two regression tasks consisting in the prediction of the boiling point for a group of acyclic hydrocarbons (alkanes) and the prediction of the non-specific activity (affinity) towards the benzodiazepine/GABA receptor of a group of benzodiazepines (classical 1,4-benzodiazepin-2-ones).

In order to obtain a unique structured representation of each compound, and their substituent fragments, as labeled positional rooted trees (k -ary trees), in [9,61] a set of representation rules have been defined.

It is worth to note that alkanes (acyclic hydrocarbons molecules) *are* trees. In order to represent them as labeled k -ary rooted trees, carbon-hydrogens groups are associated with vertexes, and bonds between carbon atoms are represented by edges; the root of the tree can be determined by the first carbon-hydrogens group according to the IUPAC nomenclature system [41] and a total order over the edges can be based on the size of the sub-compounds.

In the case of benzodiazepines, the major atom group that occurs unchanged throughout the class of analyzed compounds (common template) constitutes the root of the tree. Note that, an alternative representation would have been to explicitly represent each atom in the major atom group (by a graph based representation). However, since this group occurs in all the compounds, no additional information is conveyed to the learning model by adopting this representation. Finally, each substituent fragment is naturally represented as a tree, once cycles are treated as replicated atom groups and described by the label information.

As a result the use of labeled trees (namely labeled k -ary rooted trees) does not imply the loss of relevant information for these classes of compounds, which are representative of a large class of QSPR and QSAR problems. In particular, the representation of compounds is strictly related to the molecular topology and also conveys detailed information about the presence and types of the bonds, the atoms, and the chemical groups/chemical functionalities. Examples of representations for alkanes are shown in Fig (**4**). Examples of representations of a generic compound from a congeneric series (as in the case of benzodiazepines) with a common template used as nucleus-vertex are shown in Fig. (**5**). For these latter cases the substituents are represented by subtrees with a position on the root corresponding to the position of the substituents on the molecular template. Note that the phenyl ring representation used in the example ([62]) includes the orto, meta, para positions on the cycle as children of the “Ph” vertex.

Summarizing, the representation rules (that are fully discussed in [61,9,54] for these sets of compounds) allow us to give a unique labeled k -ary rooted tree representation of various sets of compounds through a standardized representation of cycles, by giving direction to edges, and by defining a total order over the edges. Since the rules are defined according to the IUPAC nomenclature, they retain the standard representational conventions used in Chemistry.

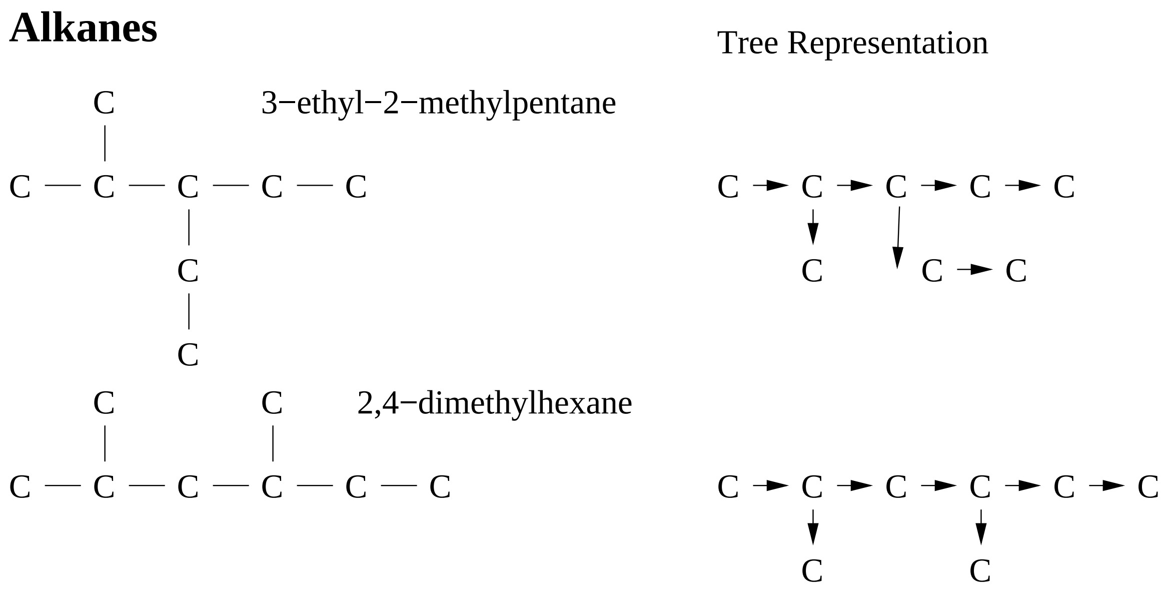


Fig. (4). Examples of tree based representation of alkanes.

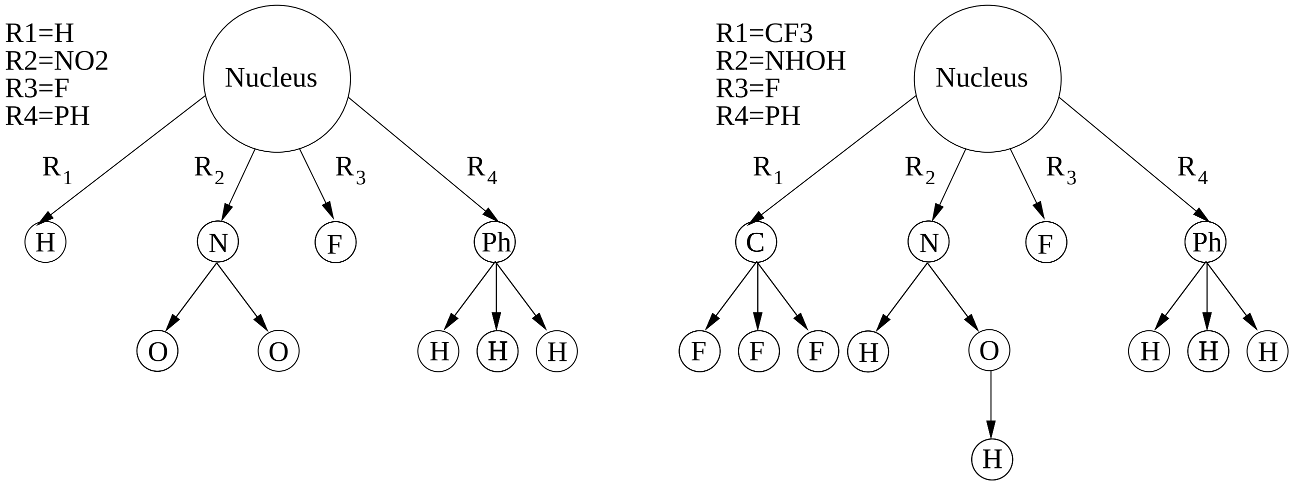


Fig. (5). Examples of representations for compounds with a common template and their substituents.

Discussion of the Examples

The two regression tasks covered by these examples, are meant to address two radically different problems in QSPR/QSAR with the aim of showing the flexibility of the proposed approaches to tackle different molecular structures, while using the same computational approach. They have provided a first assessment of the RNN approach by comparison to the results obtained in the respective specific fields [14,33]. However, these two examples are also meaningful and representative of a wider class of problems. In fact, simple structure often characterizes a wide set of problems where phisico-chemical properties have to be predicted for organic compounds (QSPR tasks). In such studies, typically the problem is to find a compromise between the possibility to fully characterize the topology of the compound and the necessity to explicitly convey into the representation information concerning the occurrence of single atoms or groups. A tree representation with labeled nodes can naturally tackle both problems, allowing to input much more information into the model than traditional approaches, e.g. topological indexes, group contribution methods, etc. (see [9] for a short review). In particular, in the approaches presented the actual selection of the relevant information is left to the learning machinery. For QSAR tasks it is very common to collect congeneric series of compounds that have the same mode of biological action, but with different quantitative levels, that medical-chemistry researchers would like to study. In

these cases, it is typical to find a common template of the congeneric series and therefore to identify a nucleus-vertex at which the structure can be rooted.
Note that in both cases, the convention used in Chemistry, as for the standard IUPAC nomenclature, follows similar approaches to get unique representation of compounds. Moreover, a tree-based representation of molecular data characterizes also other recent well-know standard representation methodology; see e.g. SMILES [83]. For instance, in the SMILES representation, rules are defined to rewrite a connected cyclic molecular graph as a connected acyclic graph by breaking one bond per ring in the original molecular graph.

Recently, a generalized set of rules for a wider set of compounds was introduced for different tasks encompassing the prediction of thermodynamical properties [6,7,54,20] of small molecules and polymers [21].
However, it is worth to note that such examples are not intended to cover all the possible structures that can be found in the chemical domains but they provide a base to show how subclasses of structured data can match the characteristics of QSPR/QSAR problems starting only from the molecular structures. A discussion on how to represent, in general, chemical structures deserves specific attention, as it has, although at an early stage of development, in [54,62,6,20]. In particular, in [54,62,6,20], it is shown that, with a proper treatment, complex chemical structures, including for instance steroisomerism (geometric isomerism or optical isomerism, i.e. Cis/trans and enantioners cases), cycles and position of cycles substituents, and even tautomeric configurations, can be represented in the proposed framework for the purpose of the QSPR/QSAR studies.
The main concept we would like to convey here is that the representation should not exclude a priori basic information such as the topology and the label content of the structure of a chemical compound. In this way, the learning tool for structure domains can exploit as much information as needed for the task at hand. The only goal of the representation rule is to find a unique “useful” representation for each molecule.
It is finally worth to note that for RNN models, since the model is adaptive and it can modify the encoding process of the structured input according to the training data (i.e. to the task), the *arbitrariness* that can result from the representation rules can be partially or totally *compensated* by the learning process. Specifically, for the RNN approach, theoretical support to the generality of the encoding performed by the model is given by the universal approximation theorem showing that RNN can approximate arbitrarily well any function from labeled trees to real values [34]. For the kernel approach the representation choice can be a stronger bias, as the similarity measures for data defined by the kernel should reflect the characteristics of the problem at hand.

4. Structural Transductions

In the previous section, we have informally introduced examples of structured domains and examples of how most of the relevant structural information concerning specific families of chemical compounds can be represented by tree structured representations. In this section we discuss about generic families of functions (denoted here as structural transductions) from a structured domain to a structured domain, and we discuss special cases that are relevant in Bioinformatics and Chemioinformatics. Since this and the following sections are a bit more technical, we need to introduce in the following some formal notations to describe a specific class of structures, i.e. DPAGs, which are relevant for Neural Networks. The reader just interested in grasping the main ideas about structural transductions may try to consider the subsection about notation as a reference manual, skipping it for the moment, and returning to it only to find out the meaning of definitions which are not readily clear from the context.

Notation

A DPAG is a directed acyclic graph G with vertex set vert(G) and edge set edg(G), where it is assumed that for each vertex $v \in$ vert(G), a bijection P_v: edg(G) $\rightarrow N$ is defined on the edges leaving from v. The *in-degree* of node v is the number of incoming edges to v, whereas the *out-degree* of v is the number of outgoing edges from v. Note that an (inefficient) undirected version of a DPAG, i.e. a PG, can be obtained by adding for each directed arc from node u to node v, a directed arc from v to u.

In some relevant cases, we can require the DPAG to possess a supersource[4]. Given a DPAG G and $v \in$ vert(G), we denote by ch[v] the set of children of v, and by $\text{ch}_k[v]$ the k -th child of v.

In addition, very often, we may require that each node of our DPAG is associated to an element of a set L representing numerical or categorical variables, denoted as the label set. Labels can be sets of symbols, e.g. the alphabet used to generate a set of strings, or a set of real valued vectors which may represent, for example, the results of some measurement related to the objects represented by each node of the structure. Given a DPAG G whose vertexes are associated to elements of L, we use subscript notation when referencing the labels attached to vertexes in a data structure.

Free trees are a class of connected undirected graphs characterized by the absence of cycles. *Rooted ordered trees* are free trees with a distinctive vertex (or node) called the *root*. Direction on trees can be naturally assigned to edges following the path from the root to each node of the tree. In the case of trees, its root node always defines the supersource. Rooted positional trees are a subclass of DPAGs, with supersource, formed by prohibiting cycles in the undirected version of the graph. Positional trees with bounded out-degree K are also called K *-ary trees*. In the following we often use the term *tree* referring to the class of labeled K *-ary* trees.

The *skeleton* of a graph G is the unlabeled version of the graph G, denoted as skel (G). The skeleton of a graph retains the topology of the graph. Hence, the term *structure*, in this paper, refers to (and is characterized by) the node labels L and their relationships, expressed in skel (G). See Fig. (**3**) for examples of these classes of structures.

In the following, we shall denote by $L^{\#^{(i,o)}}$ the class of DPAGs with maximum in-degree i and maximum out-degree o and labels in the set L. A generic class of DPAGs with labels in L and bounded (but unspecified) in-degree and out-degree, will simply be denoted by $L^{\#}$, which is in turn a sub-class of a generic structure space I. The void DPAG will be denoted by the special symbol ξ.

4.1. Classes of Structural Transductions

The most general structural transduction we may think of is the one where there is no restriction both on the domain, co-domain, and the function relating elements from the domain to elements from the co-domain. Learning this kind of transductions is very hard, since there is no bound on their complexity, although some attempts have been made in the past using a connectionist approach (e.g., [15]), and more recently within the kernel methods framework [78]. Thus, we prefer to turn our attention to subclasses of transductions that still preserve a strong relevance for real-world applications in bioinformatics and chemioinformatics.

For example, we can constrain the general form of the transduction in such a way that the skeleton of the input structure is the same as the skeleton of the output structure. We call this type of transductions *IO-isomorph* transductions. Notice that an IO-isomorph transduction does not transform the structure, but only the labels attached to each vertex of the input structure. Thus, the basic idea, underpinning the definition of functions in this class, is that structural information (including the labels) is used to generate a new label for each vertex of the input structure. A well

[4] If no supersource is present, a new node connected with all the nodes of the graph with null *in-degree* can be added.

known prediction task in Bioinformatics that involves this type of functions is the prediction of the secondary structure from the primary structure of a protein:

$$\tau_{secondary} : (amino\_acids)^{*} \rightarrow (structural\_motifs\_types)^{*}.$$

In this case, given an amino acid in the input sequence, there will be a corresponding prediction of the kind of secondary structure (i.e., structural motif) it belongs to in the output sequence. So the skeleton of the input sequence is not modified, but the labels attached to each element of the sequence are transformed from amino acids to types of secondary structures. It should be stressed, however, that the way each amino acid is transformed into a structural motif can in general depend on the whole input sequence.

Within the class of IO-isomorph transductions, there is a very important subclass characterized by the causality assumption. The definition of this subclass is easy in our framework, where we are considering DPAGs. In fact, the causality assumption states that, given an input DPAG *U*, the output $Y = \tau(U)$ of a causal transduction at vertex $v \in$ vert(*Y*) only depends on *v* (including the label) and all its descendants (including the labels), i.e. the vertexes in *Y* which can be reached from *v* by following the directed arcs. A (stationary) causal IO-isomorph transduction can be implemented by resorting to a recursive state representation, i.e. by defining a structure space *X* such that for each input structure *U*, belonging to the input structure space *I*, and output structure $Y = \tau(U)$, belonging to the output space *O*, there exist an "internal structure" $X \in X$ with skel(*X*) = skel(*U*) = skel(*Y*) and two functions *f*() and *g*() such that for each $v \in$ vert (*U*)

$$X_v = f(X_{\mathrm{ch}[v]}, U_v) \qquad (11)$$

$$Y_v = g(X_v, U_v) \qquad (12)$$

where $X_{\mathrm{ch[v]}}$ is a fixed size array of labels (sets of variables) attached to the children of *v* in the internal state DPAG. In the above definition, *f* is referred to as the *state transition function* and *g* is referred to as the *output function*. It can be observed that in the special case of sequences, each node *v* corresponds to an element of the sequence and ch[*v*] contains a single node that corresponds to the predecessor of *v* in the sequence. Notice that the above defined $\tau_{secondary}()$ is reasonably believed not to be causal, since the secondary structure does not depend only on the portion of the sequence to the left (or right) of the i-th amino acid in the input sequence. Thus, we will say that $\tau_{secondary}()$ is a *contextual* IO-isomorph transduction, since the output for the i-th amino acid in the input sequence in general depends on the amino acids that are located 'around' it, up to eventually involving all the amino acids belonging to the sequence.

Another important class of transductions is the one where the co-domain is some 'flat' domain O_{flat}, i.e. a set of numerical or categorical variables:

$$\tau : I \rightarrow O_{flat}.$$

It is not difficult to recognize that all the (multi-class) classification and (multivariate) regression tasks, involving structured objects, require learning a transduction from this class. For example, the design of a prediction function for a specific biological activity of a family of chemical compounds with respect to some receptor can be obtained by resorting to a Quantitative Structure Activity Relationship (QSAR) study. The basic idea of this approach is that the biological activity is mainly related to the structural characteristics of the chemical compounds. So the aim of QSAR is to design a function able to predict the biological property of interest

$$\tau_{QSAR} : Chemical\_compounds \rightarrow R$$

where *R* is the set of real numbers. Notice that, for this class of transductions, there is no requirement to generate an output for each vertex of the input structure, but only an element of the co-domain for the whole structure. For this reason, we will denote this type of transductions as *supersource* transductions, since the generation of the output is performed only when the whole structure has been processed, i.e., after the processing of the supersource.

Notice that a causal IO-isomorph transduction $\tau_1()$ can be implemented via a suitable supersource transduction $\tau_2()$. In fact, for each vertex *v* in the input structure *Y*, the output label generated by $\tau_1()$ in correspondence of that vertex, can be reproduced by a $\tau_2()$ that takes as input the subgraph in *Y* rooted in *v* (and including all and only its descendants) and returns the output label generated by $\tau_1()$ for *v*. This implementation scheme, however, does not work for contextual IO-isomorph transductions since the subgraph rooted in *v* does not constitute a complete context for *v*.

Finally, because of its definition, a supersource transduction can be decomposed in two parts, i.e.

$$\tau = g \circ \hat{\tau},$$

where $\hat{\tau} : I \to F$ is the *encoding* function and $g : F \to O$ is the *output* function, where *F* is a domain, that we call the *feature space* and it has the same role of the feature space introduced in the description of kernel methods, where input structures are mapped. The basic idea underpinning this decomposition is to make explicit the fact that not all the structural information contained in the input structure may be relevant to realize $\tau()$, and so the role of $\hat{\tau}()$ is to extract and to represent only the features that are relevant for $\tau()$. In this way, the output function $g()$ can focus only on relevant information. Typical examples for *F* are $\mathbb{R}^m$, or $[0,1]^m$, or $[-1,1]^m$.

In the following, we will discuss which class of transductions Recurrent Neural Networks (RNN) and Kernel Methods can implement, and also what are the differences between these two approaches.

5. Implementing Structural Transductions: Recursive Neural Networks and Kernel Methods

In this section we discuss how the different types of structured transductions can be implemented by two specific learning models: Neural Networks and Kernel Methods. We focus on these models since they are particularly effective for tasks involving chemical compounds.

Let us start our analysis from the less complex class of transductions, i.e. supersource transductions.

5.1. Supersource transductions: $\tau = g \circ \hat{\tau}$

Neural Networks rely on a recursive and adaptive realization of the encoding function $\hat{\tau}$. Free parameters of the model equip the realization of τ, allowing the learning algorithm to adapt the encoding function to the given task. Recursive Neural Networks are an instance of this class of models that assume *causality* (see Section 4.1) of the structural transduction. Because of this assumption, and because of the problem to assign a given set of free parameters to each structural entity, the input structures it can deal with in a natural way are DPAGs (Directed Positional Acyclic Graphs).

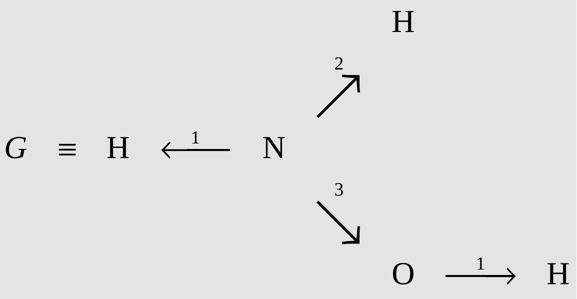
Kernel Methods rely on a (implicit) map $\Phi()$ that allows to represent the input structure $G \in A^{\#^{(i,o)}}$ in some dot product space (the *feature* space). The dot product in the feature space (computed via a *kernel* function $K(\cdot,\cdot)$ defined in the input space) can be used as similarity measure to compare two input structures *G* and G', i.e. $K(G,G') = \Phi(G) \cdot \Phi(G')$ and large margin methods, such as Support Vector Machines, can be used to learn the supersource transduction. In our framework, the function $\Phi()$ plays the role of encoding an input structure, i.e. $\hat{\tau}(G) = \Phi(G)$. As a result, in such an approach, the function $\tau()$ is chosen a priori by the kernel designer and it is crucial, for success, that the choice properly reflects the characteristic of the problem at hand.

5.1.1. Recursive Neural Networks

As we already mentioned, Recursive Neural Networks (RNNs) [75] are naturally defined on DPAGs. Recent developments are actually enlarging the class of graphs that can be processed by these models; however, for the sake of explanation, here we focus on DPAGs.
Given a DPAG G, a RNN can be defined via the following recursive definition of $\hat{\tau}$:

$$\hat{\tau}(G) = \begin{cases} 0 \ \ \text{(the null vector in } \mathbb{R}^m) & \text{if } G = \xi \\ f(l(s), \hat{\tau}(G^{(1)}), \ldots, \hat{\tau}(G^{(o)})) & \text{otherwise} \end{cases} \tag{13}$$

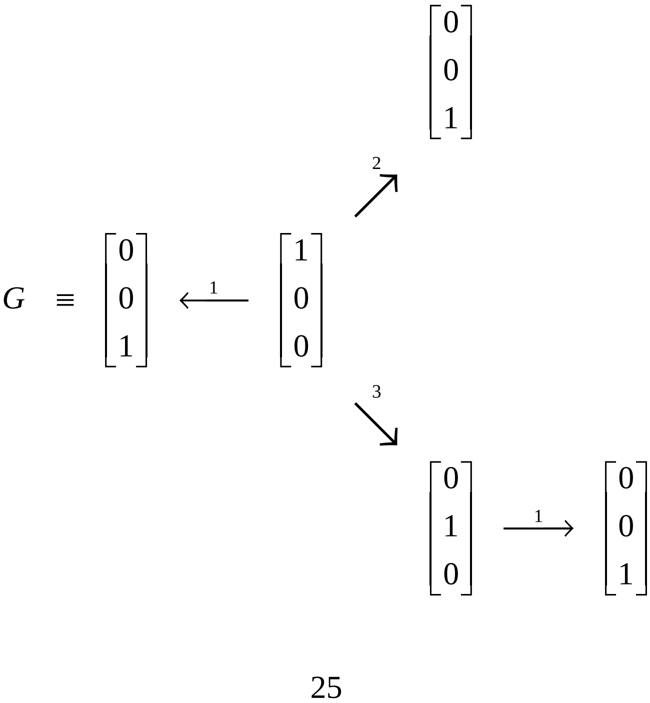
where a (*stationary*) $f()$ can be defined as $f : \mathbb{R}^n \times \underbrace{\mathbb{R}^m \times \cdots \times \mathbb{R}^m}_{\text{o times}} \to \mathbb{R}^m$, $\mathbb{R}^n$ is the label space, the remaining domains represent the encoded subgraphs up to the maximum out-degree of the input domain $(\mathbb{R}^n)^{\#^{(i,o)}}$, s=*source*(G), $l(s) \in \mathbb{R}^n$ is the label attached to the source of G, and $G^{(1)}, \ldots, G^{(o)}$ are the subgraphs pointed by s, i.e. *source*$(G^{(i)})$ = ch$_i[s]$.
As an example, let us consider a small compound which can be used to illustrate the above formula, i.e. hydroxylamine (NH2OH). Following the linear notation, we can represent its molecular graph as a rooted tree, where the root is N and the children of a vertex are sorted according to the order in the linear representation:



Thus G has out-degree 3 and in-degree 1. Given the following orthogonal vectorial encodings for the atoms

$$\text{N} \equiv \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{O} \equiv \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{H} \equiv \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

the final representation becomes as follows



Let us now consider the encoding of G by eq.(13) in a feature space of dimension $m = 2$ (i.e., using a number of hidden neurons equal to 2), where we have $\mathbf{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$:

$$\hat{\tau}(G) = f\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \hat{\tau}\left(G^{(1)}\right), \hat{\tau}\left(G^{(2)}\right), \hat{\tau}\left(G^{(3)}\right) \right)$$

where

$$G^{(1)} \equiv G^{(2)} \equiv \left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right), \qquad G^{(3)} \equiv \left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)$$

and applying recursively eq.(13) we also get

$$\hat{\tau}\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) = f\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right)$$

$$\hat{\tau}\left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) = f\left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \hat{\tau}\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right), \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right).$$

By proper substitutions, the encoding can finally be written as

$$\hat{\tau}(G) = f\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, f\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right), f\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right), f\left(\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, f\left(\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right), \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \right).$$

It should be noted that the position attached to edges is actually used above to decide which argument of the function *f*() should take the encoding of a given subgraph. Moreover, it should be stressed that the above derivation basically remains the same for a feature space of different dimension, e.g. m = 5: the only differences are in the size of the null vector $\mathbf{0}$ which should be equal to m, as well as the dimension of vectors encoding the DPAG and its subcomponents. Finally, it should be noted that the vectorial encodings for the label of atoms (or functional groups) could also include more informative numerical descriptors, such as properties of the atoms (or of the functional groups). Moreover, if needed, such labels can also be used to represent 3D information of the molecular components. The labeling of atoms or groups is a further point of flexibility of a structured representation.

A possible neural realization for *f*(), applied to a vertex *v*, is as follows

$$f\,(l(v), x(\mathrm{ch}_1[v]), \ldots, x(\mathrm{ch}_o[v])) = \sigma(Wl(v) + \sum_{j=1}^{o} \widehat{W}_j x(\mathrm{ch}_j[v]) + \theta),$$

where $\sigma(\cdot)$ is as already defined, $l(v) \in \mathbb{R}^n$ is the label attached to the vertex, $\theta \in \mathbb{R}^m$ is the bias vector, $W \in \mathbb{R}^{m \times n}$ is the weight matrix associated with the label space, $x(\mathrm{ch}_j[v]) \in \mathbb{R}^m$ are defined as $x(\mathrm{ch}_j[v]) = \hat{\tau}(G^{(j)})$, where *source* $(G^{(j)})$ = ch $_j[v]$, i.e. they are the feature vectors (in neural networks language, the hidden activation vectors) obtained by applying the encoding function $\hat{\tau}$ to the subgraphs $(G^{(j)})$ pointed by *v*, and $\widehat{W}_j \in \mathbb{R}^{m \times m}$ is the weight matrix associated with the *j* th subgraph space.

Note that in the example above, *f*() computes 5 feature vectors of dimension 2, one for each subcomponent of *G*, three of which are equal, and one for the full tree *G*.

There are different ways to implement recursive neural networks [75,54]. Following the discussion in Section 2.3, constructive approaches can be considered. Constructive approaches allow the training algorithm to progressively add (hidden) recursive neurons during the training phase. Recursive Cascade Correlation (RCC) [74,75,54] is an extension of Cascade Correlation algorithms [27,26] allowing to deal with structured data. Specifically, in RCC, hidden units are added incrementally to the network, so that their functional dependency can be described as follows:

$$\begin{aligned}
x_1(v) &= f_1(l(v), x_1(\mathrm{ch}[v])) \\
x_2(v) &= f_2(l(v), x_2(\mathrm{ch}[v]), x_1(\mathrm{ch}[v])) \\
&\vdots \\
x_m(v) &= f_m(l(v), x_m(\mathrm{ch}[v]), x_{m-1}(\mathrm{ch}[v]), \cdots, x_1(\mathrm{ch}[v]))
\end{aligned} \tag{14}$$

where, as notational shortcuts, we use $x_j(\mathrm{ch}[v]) \equiv x_j(\mathrm{ch}_1[v]), \ldots, x_j(\mathrm{ch}_o[v])$. The number of hidden units defining the *f*() function depends on the training process. Hence, the dimension *m* of the feature space *F* is automatically computed by the training algorithm, thus allowing an adaptive computation of the number and type of (numerical) descriptors needed for a specific transduction task.

Concerning the output function *g*, it can be defined as a map $g : \mathbb{R}^m \rightarrow \mathbb{R}^z$ implemented, for example, by a standard feed-forward network.

5.1.2. Kernel Methods

As in the previous section, the goal is to obtain a supersource transduction $\tau = g \circ \hat{\tau}$, where $\hat{\tau}$ is the encoding function and *g* is the output function. As already said at the beginning of Section 5.1, the encoding function $\hat{\tau}()$ is implemented via an implicit function $\Phi()$, while the output function $g()$ is typically realized by a Support Vector Machine, or a regularized large margin approach. Since the function $\Phi()$ is implicitly defined by specifying the kernel, we briefly talk about the different approaches that have been proposed to devise a kernel for structured data. Surveys of kernel for graphs can be found in [30,73].

First of all, it should be said that many of the kernels defined on structured domains assume discrete labels, usually represented by symbols. This assumption allows to define efficient procedures for computing the kernels, which otherwise would require much more computation. In fact, when comparing two structures, only vertexes with the same label contribute to the definition of the kernel.

Kernels can be classified considering the different methods employed by the kernel to implicitly map the structured information into a vectorial space. Here we distinguish kernel functions between Structure based kernels and Kernels based on probabilistic models. Structure based kernels are kernels which directly refer to parts of the structure and compute distance measures comparing parts of the structures (see for example [37,82,51,50,80,49,72,46,16,29,31,39,40,77,71]. In fact, the main approaches in this family are convolution kernels, and spectral kernels, where a structure (either string, tree, or graph) is represented via the “amount” of presence of every possible substructure in it, and the output of the kernel is obtained by the standard dot product between representations of this type.

Kernels based on probabilistic models start from a probabilistic model of the data and either derive a kernel classifier directly from probabilistic decision models or use parameters of the probabilistic model instead of the original data. Deriving a probabilistic model incorporating information about data, however, is in general computationally very expensive. These models include [44,43,42,52,65,19]. A very important kernel in this class is the Fisher kernel [44], which actually is a general way on how to generate a kernel using a probabilistic model describing the data, such as a Hidden Markov Model (used, for example, in Bioinformatics to model protein families) or Stochastic Context-Free Grammars (which have been used to model RNA sequences). In the following, in order to give concrete examples of kernel for structures, we briefly describe two examples of structure-based kernels, i.e. a string kernel and a tree kernel.

A String Kernel

Let A be a finite set of *characters* called *alphabet*. A *string* is an element $\iota \in A^k$ for $k = 0,1,2\ldots$. Let $|\iota|$ be the length of ι and let $s,\iota \in A^k$, we say that $s \sqsubseteq \iota$ if s is a substring of ι. Let $s \in A^k$, then we denote with $\mathrm{num}_s\ (\iota)$ the number of occurrences of s in ι. Let $\Phi(\iota)$ be a function that maps $\iota \in A^k$ into a feature space $\mathbb{R}^m$ with $m = \sum_{i=1}^{k} |A|^i$, where each dimension represents a particular string of A^k. Defining $\phi_i(\iota) = \mathrm{num}_{s_i}(\iota)\sqrt{|s_i|}$, we can consider a dot product between vectors in this feature space [80]:

$$K(\iota,\iota') = \sum_{s_i \in A^*} \phi_i(\iota)\phi_i(\iota') . \qquad (15)$$

Note that, by definition, this is a kernel function since it represents a dot product in $\mathbb{R}^m \times \mathbb{R}^m$. Therefore, the kernel function $K(\iota,\iota')$ depends on the co-occurrences of substrings s_i both in ι and in ι'. A match is then weighted with the length of the common substring s_i. The function described in eq. (15) can be computed in time $O(|\iota|+|\iota'|)$ building the matching statistics with the Suffix Tree algorithm [80]. String kernels, including variations of the above scheme defined for dealing with inexact matchning, are typically used in Bioinformatics' tasks such as remote protein homology detection.

A Tree Kernel

Concerning a kernel operating directly on trees, we present the kernel proposed in [16]. It is based on counting matching subtrees between two input trees. Given an input rooted tree t, let $s^{(t)}$ be a subtree of t if $s^{(t)}$ is rooted in a node of t and the set of branches of $s^{(t)}$ is a subset of connected branches of t (note that, with this definition, leaves do not need to be necessarily included in the subtree). We assume that each of the m subtrees in the whole training data set is indexed by an integer between 1 and m. Then $\phi_s(t)$ is the number of times the tree indexed with s occurs in t as a subtree. We represent each tree t as a feature vector $\Phi(t) = [\phi_1(t), \phi_2(t), \ldots]$ (see Fig. (**6**)). The inner product between two trees under the representation $\Phi(t) = [\phi_1(t), \phi_2(t), \ldots \phi_m(t)]$ is: $K(t,t') = \Phi(t)\cdot\Phi(t') = \sum_{s=1}^{m} \phi_s(t)\phi_s(t')$.

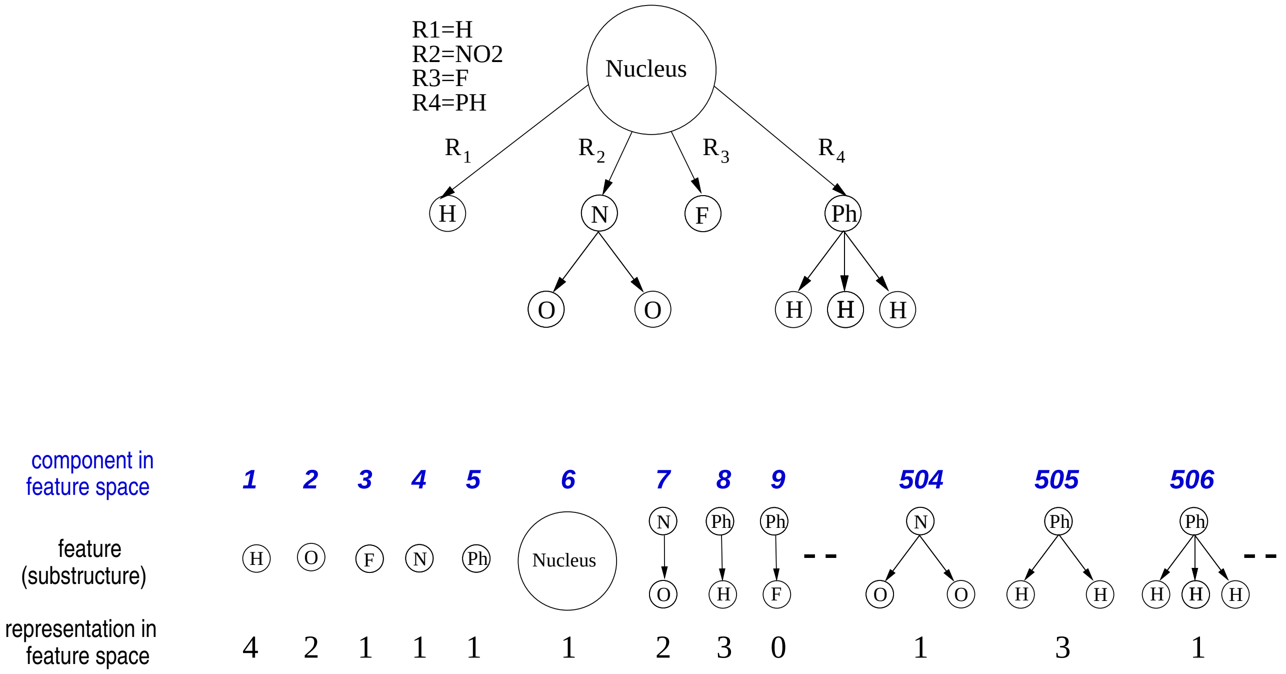


Fig. (6). Example of representation in feature space of a rooted tree.

Thus, this tree kernel defines a similarity measure between trees that is proportional to the number of shared sub-trees. Efficient computation of the kernel can be obtained via dynamic programming, leading to a $O(\mathrm{vert}\,|t| \times \mathrm{vert}\,|t'|)$ time algorithm (see [16]).

5.2. Causal IO-isomorph transductions

In this case, RNN can easily be extended to deal with this kind of transductions by just having the output function *g*() to emit an output for every vertex of the input structure. This can be obtained either by resorting to a transformation of the original problem into an equivalent supersource transduction formulation, or by applying *g*() to every vertex of the input structure. Learning proceeds as usual by a gradient descent approach or similar incremental strategies.

For kernel methods, the direct implementation of causal IO-isomorph transductions may become very expensive because of the implicit definition of $\Phi()$ which requires the computation of the kernel on the whole structures and not on parts of it. So the more natural way to proceed, without devising an ad hoc kernel, but reusing an already defined kernel, is to transform the original problem into an equivalent supersource transduction formulation. The more promising approach for this kind of transductions is the Fisher kernel, also with respect to the treatment of real-valued labels.

5.3. Contextual IO-isomorph transductions

Examples of RNN able to exhibit contextual processing ability can be found in the BRNN model [2] for sequences and the CRCC model [60] for DPAGs. The principled design of large-scale recursive neural network architectures denominated DAG-RNN is described in [3], where an application to protein structure prediction is discussed. In this paper, we present the CRCC model.

Concerning kernel methods, even in this case the implicit definition of $\Phi()$ does not help, and, again, the Fisher kernel seems to be the best candidate as basis for the development of an ad hoc kernel.

5.3.1. Contextual Recursive Cascade Correlation

In the following we describe *Contextual Recursive Cascade Correlation* (CRCC) for the processing of directed positional acyclic structures, based on the extension of the Recursive Cascade Correlation model (RCC, Section 5.1.1).

The first idea of CRCC appeared in [58] applied to the sequence domain. The basic idea of CRCC approach appears in [59], and a fully formal description is developed in [60]. A theoretical study of the universal approximation capability of the model appears in [35].

The main idea underlying the CRCC approach is to extend the computational capability of causal recursive neural networks. Due to the causality assumption (see Section 4) a standard RNN cannot represent in its hypothesis space contextual transductions. In particular for a RNN the context for a given vertex in a DPAG is restricted to its descending vertex. In contrast, in a contextual transduction the context is extended to all the vertexes represented in the structured data.

Since we consider the extension of the models for contextual processing of DPAGs, we start extending the notation introduced in Section 4 denoting by $\mathrm{pa}[v]$ the set of parents of v, and by $\mathrm{pa}_j[v]$ the j-th parent of v, with respect to P_v.

Since CRCC is a constructive algorithm, we denote by $x_i(v)$ the i-th component of X_v, i.e., the output of the i-th hidden unit in the network for the vertex v. The computation of each $x_i(v)$ hidden unit is based on frozen units. In other words, the output $x_i(v)$ of each hidden unit i is computed for all the vertexes of the input DPAGs before the training of a new hidden unit $j > i$. The weights of the unit i cannot be retrained in the next steps and therefore the unit is said to be *frozen*. Thus, when training hidden unit k, the state variables $x_1,\ldots, x_{k-1}$ for all the vertexes of all the DPAGs in the training set are already available, and can be used in the definition of x_k. Consequently, equations of causal RNN, and RCC in particular (eq. (17), Section 6.1.1), can be expanded in a contextual fashion by using, where possible, the variables $x_i(\mathrm{pa}[v])$. The equations for the state transition function of a CRCC model are defined as:

$$\begin{aligned} x_1(v) &= f_1(l(v), x_1(\mathrm{ch}[v])) \\ x_2(v) &= f_2(l(v), x_2(\mathrm{ch}[v]), x_1(\mathrm{ch}[v]), x_1(\mathrm{pa}[v])) \\ &\vdots \\ x_m(v) &= f_m(l(v), x_m(\mathrm{ch}[v]), x_{m-1}(\mathrm{ch}[v]), x_{m-1}(\mathrm{pa}[v]), \cdots, x_1(\mathrm{ch}[v]), x_1(\mathrm{pa}[v])) \end{aligned} \tag{16}$$

where, as notational shortcuts, we use $x_j(\mathrm{ch}[v]) \equiv x_j(\mathrm{ch}_1[v]),\ldots, x_j(\mathrm{ch}_o[v])$ and $x_j(\mathrm{pa}[v]) \equiv x_j(\mathrm{pa}_1[v]),\ldots, x_j(\mathrm{pa}_i[v])$ for an input domain $(\mathbb{R}^n)^{\#(i,o)}$, i.e. a DPAG with maximum out-degree o and maximum in-degree i.

Hence, in the CRCC model, the frozen state values of both children and parents of each vertex can be used as input for the new units without introducing cycles in the dynamics of the state computing system.

Following the line developed in Section 5.1.1 it is easy to define neural network models that realize the CRCC approach. In fact, it is possible to realize each f_j of Equation 15 by a neural unit associating each argument of the f_j to the inputs and free parameters W of the neural unit. In particular, given a vertex v, the CRCC model takes in input for each neural unit:

- connections associated to the input label of v, and connections associated to the state variables (both the output of already frozen hidden units and the current j-th hidden unit) of the children of the current vertex v (as for the RNN/RCC model presented in Section 5.1.1);
- connections associated to the state variables of the *parents* of v which are already *frozen* (i.e., the output of already frozen hidden units computed for the parents of v). The connections associated to *parents* add new parameters to the model that are fundamental to the contextual processing.

The major aspect of this computation is that an increasing context is taken into account for a large *m*. In fact, the hidden unit for each layer takes directly a local context (parents and children) of each vertex as input and progressively, by composition of context developed in the previous steps, it extends the context involving other vertexes, up to the vertexes of the whole structure. The context window follows a nested development; as an example, let us consider vertex $v' =$ (pa[v]) and the presence of 3 hidden units. Following the definition given in eq. (15), $x_2(v')$ depends on x_1(pa[v']) and $x_3(v)$ depends on x_2(pa[v]) = $x_2(v')$. Thus, we obtain that also $x_3(v)$ depends on x_1(pa[v']), which means that $x_3(v)$ includes information by $pa[v'] = pa[pa[v]]$ in its "context window".

In particular, adding new hidden units to the CRCC network leads to an increase of the "context window" associated to each vertex *v*. Hence, the size of the context window can grow during model training and we do not need to fix it prior to learning. Given a sufficient number of hidden units the information included in x_i grows to all the context of the structured data according to the structure topology. A formal analysis of the context development of the CRCC model is in [60], showing the extension of the capability allowed by CRCC for tasks that cannot be computed by causal models. In particular, such results include contextual IO-isomorph transductions and supersource transductions on DPAGs.

Beside these results, there are other interesting cases that deserve to be discussed for supersource transductions that both causal and contextual models can compute. In fact, relaxing the causal assumption can be useful also for supersource transductions whenever the meaning of a sub-structure depends on the "context" in which it is found. In such way it is possible to consider in which position, within a larger structure, the given substructure does occur. In order to show the relevance of a contextual processing for this issue, let us consider an example of sub-structure encoding, as shown in Fig. (**7**).

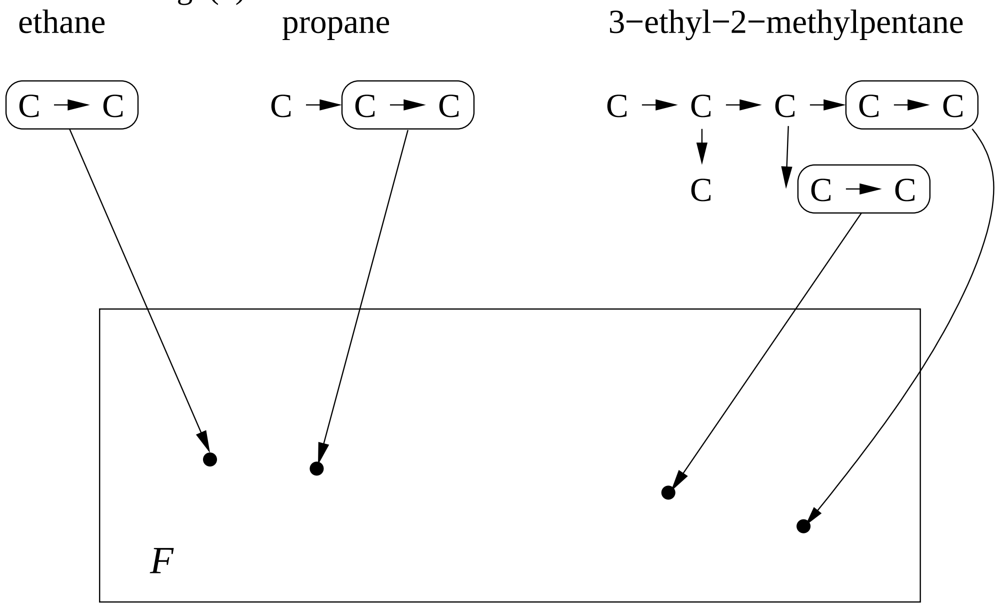


Fig. (7). Different representations of the "C-C" chemical fragment in the F space by a contextual model.

In Fig.(**7**) the internal encoding of some fragments (subgraphs) is shown as represented in a 2-dimensional feature space *F* (see Section 4). Each point in the *F* space represents the numerical code developed by the CRCC model for each fragment in the input structured domain. This can be practically obtained by projecting X_v, usually belonging to a m -dimensional space into a 2-dimensional space that can be easily visualized, e.g. by Principal Component Analysis (PCA). In such a case, a causal mapping yields a unique code for each occurrence of the fragment. By a contextual mapping, each fragment can be represented in different ways depending on the context (position). Hence, clearly, CRCC achieves a more expressive substructure encoding. A PCA plot and analysis of the internal representation of CRCC developed for alkanes data can be found in [60].

In Chemistry a more expressive substructure encoding can result useful because it is quite common that the meaning of a chemical fragment or functional group depends on its molecular position or on its different context in different molecules.

6. Learning in Recurrent Neural Networks and Kernel Methods

Concerning Kernel Methods for structured objects, there is basically no difference with respect to “flat” representations. In fact, since the learning machinery is defined independently of the kernel definition, it is sufficient to define a kernel for structure and to plug it in the kernel method, e.g. Support Vector Machine. Thus learning is the same as described in Section 2.4.

Neural Networks do not share this modularity property with Kernel Methods, and, for this reason, specialized learning procedures must be devised. The parameters of a RNN are typically adapted by a gradient descent approach. Constructive approaches, for example based on Cascade Correlation, have also been proposed and tested on real-world applications, obtaining very good results.

6.1. Neural Networks

Actually, several supervised learning algorithms can be extended to train recursive neural networks. The Back-Propagation Through Structure and RTRL (Real Time- Recursive Learning) are gradient descend based algorithms [75] that extend the BPTT and RTRL algorithms [84] developed for recurrent neural networks. Here we describe Recursive Cascade Correlation [74].

6.1.1. Recursive Cascade Correlation

As already said, the problem of designing optimal neural network architecture for a given task is still theoretically unsolved. Moreover, the results strongly depend on the choices made using an expensive trial and error approach. For these reasons, we present in the following a specific neural realization of the functions *f*() and *g*() based on constructive algorithms, i.e. algorithms that build a network starting with a minimal configuration and that add new units and connections during training.

Recursive Cascade Correlation (RCC) [74,75,54] is the extension of Cascade Correlation algorithms [27,26] to deal with structured data. The Cascade Correlation class of algorithms supplies a method to solve the problem of establishing the number of hidden units, which is typically found when dealing with fixed size architectures. RCC creates a recursive neural network using an incremental approach for the classification or regression tasks.

The main advantages of the RCC approach can be summarized as follows:

- automatic determination of network dimension and topology: RCC allows to deal with hypothesis spaces of flexible size, since the number of recursive hidden units is decided by the learning algorithm;
- training of a single unit for each step.

Specifically, this incremental learning approach has been found to be particularly useful in application settings (e.g. [9]) when no “a priori” information is known on the problem’s complexity.

Model

Each unit, i.e. the function *f*(), is realized by a recursive neuron: the output, x_h, of the *h* th hidden unit over the current vertex *v* can be computed as

$$x_h(v) = \sigma(\mathrm{net}_h(v))$$

$$\mathrm{net}_h(v) = \sum_{i=0}^{n} w_{hi} l_i(v) + \sum_{i=1}^{h} \sum_{j=1}^{K} \hat{w}_{hi}^{j} x_i(\mathrm{ch}_j[v])) \tag{17}$$

where $\sigma(\cdot)$ is a sigmoidal function, w_{hi} is the weight of the i -th input element to the h -th hidden unit, $\hat{w}_{hi}^{j}$ is the weight of the edge connecting the i -th unit to the h -th current unit, which brings the encoded information of the j -th child of the current input vertex.

Learning in RCC

The RCC learning algorithm creates a neural network using an incremental approach for the classification (or regression) of structured patterns. At the beginning, it is assumed that the starting network N_0 is a network without hidden units and one output-layer yielding constant null outputs for each input pattern. In the first network N_1 a hidden recursive unit x_1 is added such that the *correlation* between the output of the unit and the residual error of network N_0 is maximized. Typically, since the maximization of the correlation is obtained using a gradient ascent technique and the maximization of correlation induces an error surface with several maxima, a pool of hidden units is trained and the best one is selected. The weights of x_1 are frozen (they cannot be retrained in the next steps) and the remaining weights (output layer) are retrained. If the obtained network N_1 cannot solve the problem, new hidden recursive units are progressively added and connected with all the inputs and previously installed hidden units. The process continues until the residual errors of the output layer satisfy a specified stopping criteria (e.g. the errors are below a given threshold). Hence, the described method dynamically builds up a (not-fully connected) recursive neural network and terminates once a sufficient number of hidden units has been found to solve the given problem. The resulting network is a *cascade* of units.

Summarizing, learning is performed as in the Cascade Correlation model family according with the recursive nature of the hidden units.

A technical explanation of the learning procedure, which also involves variations of the standard backpropagation algorithm [25,69], can be found in [54,74,75].

7. Discussion about Recursive Neural Networks and Other Recent Developments

Most of the molecules used for experimental purposes can be treated by the set of specifications given in Section 3 (Rules paragraph). However, there are various cases that deserve attention when the set of data does not fit the condition used for a DPAG representation. The need for a more general approach leads to the study of models able to directly deal with general graphs, including either directed/undirected, cyclic/acyclic, ordered/unordered graphs.

The kernel-based approaches offer a concrete example of such possibilities. In fact the kernel is highly flexible in its definition allowing to include a wide set of possible input data, including general classes of graphs. However, in order to retain the advantages deriving from an adaptive encoding of structures, i.e. the $\hat{\tau}$ function, various extensions have been developed for the RNN approach aimed at dealing with general classes of graphs.

A computational characteristic of the RNN approach is the *causality* assumption (see Section 4). Such assumption is used by RNN to realize an encoding that matches the input structures according to their topological order (i.e. transductions on the structured domain). However, assuming causality imposes constraints on the encoding process and thus on the transduction realized by the RNN models. The constraints concern both the computational power of the model ([60]) and the classes of data: in particular the orientation, cyclicity and the need of a supersource vertex are the graph characteristics affected by the causality assumption. In fact, rooted trees or directed acyclic graphs (DAG) are subclasses of graphs for which a topological sort exists. In Section 5.3.1 we discussed a contextual approach able to partially relax the causality assumption for directed acyclic positional graphs (DPAG).

Contextual processing of structured domains has opened new ways to deal with graph data. The main issue is to determine if, besides the state-of-the-art results achieved by the RNN model, the causality assumption can hamper the solution of specific chemical problems. The preliminary results in [60], achieved for the analysis of the alkane data set and some artificial benchmarks, indicate that a more expressive (contextual) encoding of the structures introduces benefits for the QSPR analysis.

The direct processing of cyclic graphs is another relevant issue for the computational chemistry applications. For RNN, extensions to cyclic graphs are studied both by introducing cycles in the state transition system or by rewriting the original graphs into hierarchical structures, e.g. [75,8]. However, using these strategies, although feasible in principle for processing cyclic graphs is paid in terms of efficiency of the learning algorithm. To this aim, the study of efficient solutions to the treatment of general classes of graphs has been recently addressed by different authors. A recursive approach has been exploited in [32] defining a dynamical system based on contractive mappings that can be efficiently solved by relaxation techniques. A different approach based on a constructive contextual processing for fairly general graph structures has been introduced in [57,56]. In contrast to previous neural networks for structures, this new approach, called NN4G, relaxes the causality assumption by relaxing the full power of a recursive approach in favor of the possibility to treat a more general class of graphs, including both cyclic/acyclic, directed/undirected, labeled graphs.

8. Summary of Application Results

For the sake of completeness we report in the following a brief summary of some recent results obtained applying some of the models we have introduced in this paper to benchmark regression tasks. There are many more works dealing with applications of Recursive Neural Networks to tasks involving chemical structures. Just to mention a few in Bioinformatics, Pollastri et al. [67] have shown how contact maps of proteins can be predicted from the primary structure, Vullo & Frasconi [81] predicted disulfide connectivity patterns in proteins, and Ceroni et al. [13] learned protein secondary structure from sequential and relational data. Concerning kernel for structures, many papers have been very recently published. Among these, it is worth to mention applications in Chemistry addressed by Ralaivola et al. [68], classification of chemical compounds by Mahé et al. [53] via marginalized graph kernels, and predictive graph mining of the NCI-HIV database by Horváth et al. [39] involving the use of cyclic pattern kernels.

Although here the focus is on a methodological discussion, the following assessments should give a hint on the potentiality of the models and on their evolution. In fact, the capability of the proposed approach to adaptive processing of structured domains has been progressively supported by the application of the RNN models to various QSPR/QSAR problems, basically involving either congeneric series or simple organic compounds. In particular, let us consider QSPR problems, such as the prediction of the boiling point for alkanes [9] and the prediction of thermodynamical properties of the solvation process (solvation free energy in water) [6,54,7,20], and QSAR problems, such as the prediction of the non-specific activity (affinity) towards the benzodiazepine/GABA receptor [10,9,61], and the prediction of A1 adenosine receptor ligands affinity toward the receptor [62]. As a general result, we found that the generality of the RNN approach, which is able to tackle such radically different problems, is not at the expense of predictive accuracy, since our results outperformed the results achieved by traditional QSPR/QSAR treatments. Moreover, studies of the numerical code developed by the RNN demonstrated that the model is able to discover relevant features for the problem at hand just on the basis of the associations between the molecular morphology and the target property [61]. Specifically, the results in [61] show that the characteristics of many substituents affecting the activity of benzodiazepines, already highlighted by previous QSAR studies, were correctly recognized by the model. This is a further step towards the assessment of the model as a new tool for the rational design of new molecules.

Recently, new applications have been explored addressing the prediction of the glass transition temperature of metha-acrylic polymers, i.e. macromolecules data [21].

In the following, for the sake of uniformity, we report the evolution of the results achieved by the various models based on the RNN approach developed through time by our research group and the tree-kernel based method applied to the alkanes QSPR benchmarks (prediction of the boiling point temperature of alkanes). The data set used is described in [14,9] and comprised all the 150 alkanes (acyclic hydrocarbons) with up to 10 carbon atoms (C_nH_{2n+2}), represented by hydrogens suppressed graphs (see Fig.(**4**)). There is a total of 1331 vertexes in the data set. The target values are measured in Celsius degrees and are in the range [-164 , 174].

The original aim of the application developed in [9] was the assessment of the RNN method by comparison versus standard multilayer feed-forward networks using *ad hoc* vectorial representations of alkanes [14]. Successively, the same benchmarks were used to test the tree-kernel approach [55] and to assess the advantages deriving by the contextual models [60].

Due to the relative simplicity of this class of compounds, the task is particularly useful to test the performance of advanced models for structured domain. In fact, the prediction task is well characterized for this class of compounds, since the boiling points of hydrocarbons depend upon molecular size and molecular shape (number and branching of carbon atoms in particular), and vary regularly within a series of compounds, which means that there is a clear correlation between molecular shape and boiling point. Moreover, the structure of these compounds does not need different atoms and bound type representations and it mainly conveys topological information. Thus, these compounds are amenable to very compact representations by traditional QSPR approaches such as topological indexes and/or vectorial codes, which are capable of retaining the relevant information for prediction. For these reasons, standard multilayer feed-forward networks using “ad hoc” representations yield very good performances. For the same reason, the test is particularly challenging for the new models which do not exploit the background information on the task: the model have to “rediscover” the relevant features just on the basis of the association between the input structure and the target property. In this way, the capability of the models for structured domain to extract topological information directly from the structured data is fairly tested.

The results obtained by the different models are summarized in Table **1**. They include the original results reported in [14] using a standard feed-forward Neural Networks on a *ad hoc N-tuple* vectorial representations of alkanes, and the models presented in the current paper, i.e. the RCC (i.e. an implementation of RNN by a cascade correlation method) [9], the tree-kernel-based methods (TK) [55], and the CRCC [60].

In Table 1, we report the mean of the training mean absolute error (TR Mean Abs. Error) and of the test maximum and mean absolute error (TS Max/Mean Abs. Error) on a 10-fold cross validation.

In the original RCC/CRCC experiments [9,60], learning was stopped when the maximum absolute error for a single compound was below 8^0 C (TR Max AE), or when a maximum number of hidden units (150) was reached. For each fold, 5 different learning trials were performed by the RCC and CRCC models.

For the sake of comparison, we have used exactly the same tree based representation described in Section 7 for both the RCC/CRCC and tree-kernel based approaches. This allows us to investigate the differences among these different approaches. Moreover, the general goal is to find better model solutions without relying on the ad hoc tuning of the input structured representation.

Table 1. Results on 10-fold cross-validation for Alkanes

	TR Mean Abs.Error	TS Max Abs.Error	TS Mean Abs.Error
MLP	2.22	10.42	3.01
RCC	2.15	10.03	2.87
TK	2.43	10.65	2.93
CRCC	1.68	8.29	2.56

In [14], Cherqaoui et al. use a vectorial code representation of alkanes based on the *N-tuple* code for the encoding of trees. So they represent each alkane as a 10 numerical components vector with the last components filled by zeros when the number of atoms of the compound is less than 10. The single component encodes the number of bounds of the corresponding carbon node. In particular, the *N-tuple* code used in [14] is specific for the set of alkanes considered, as it provides a fixed dimensional vector for all the data, and it does not allow the representation of different atom symbols, for instance atoms or groups different from the carbon atom. Moreover, “ad-hoc” features, such as the number of carbon atoms and the branching of the molecular trees, are explicitly reported in the representation. Hence, the information conveyed by the *N-tuple* code corresponds to the features we described as correlated to the boiling point property. As a result, the representation is efficient and the obtained accuracy is very good for the prediction of the boiling points. However, when considering different classes of data, e.g. where various types of atoms occur, and different tasks, the representation assumptions could be useless. In particular, it is possibly required to design a different representation able to convey proper features related to the different predicted properties. Of course, a labeled tree representation can represent variable-size structures and can easily convey labeling information, which can be used to represent various types of atoms or chemical groups. Moreover, the topological aspect of the original tree is fully represented, therefore the topology information are not selected a priori on the basis of the specific target property. Hence, tree based representations result much more general for similar classes of acyclic compounds and they can be used for various tasks without the need to modify them according to the target property. Actually, the same basic RNN approach used to deal with this QSPR problem has been exploited on the other radically different QSPR/QSAR problem described above. Beside this generality, the new models for structured domain introduce performance benefits in the benchmark, as summarized by the mean absolute error results reported in Table **1**. RCC networks seem to perform slightly better than the tree kernel, even if it is difficult to compare the two approaches on a fair ground. Anyway, the experimental results clearly provide a support to the hypothesis that for structured domains it is better to use methods able to deal directly with the structured nature of the domain. We can finally observe that CRCC improves previous results obtained by standard QSPR approaches, causal recursive models, and kernel for tree methods, thus showing that beyond theoretical results, a contextual approach can improve results also for supersource transduction tasks. Such tasks can be computed by both causal and contextual models but the reliability of a causality assumption remains unknown.

9. Discussion and Conclusions

The aim of this paper was to discuss some recently developed ML models for the direct representation of structured objects and direct learning of functions involving structured objects. These models encompass different ML paradigms. Here we focused on the analysis of neural network-based and kernel-based approaches and on their potential for providing new enhanced solutions to many computational tasks involving chemical compounds, which can be cast as classification and regression tasks on structured objects in a natural way. Indeed, the flexibility of the structured representation used so far by RNNs, allowed us to effectively tackle several of these problems, ranging on different families of molecular data. The positive experimental results obtained with respect to traditional approaches seem to confirm the advantage in using a direct correlation between molecular structures and property values, even if only subclasses of graphs, i.e. rooted trees, are used to represent chemical compounds. Similar experimental results are emerging with the definition and application of kernel for structures.

The relationship between Recursive Neural Networks and Kernel for Structures can be understood via the formal definition of structural transduction introduced in Section 4. Specifically, when considering QSPR/QSAR analysis, it can be readily recognized that this definition is fully compatible with the modus operandi used in that context, where the input domain defines a proper structured representation of molecules and the output domain conveys the property/activity values. In particular, when considering the decomposition of a structural transduction introduced for supersource transductions, the function $\hat{\tau}$ can be understood as the process of encoding molecular data into a space of descriptors or structural features and the g function as the mapping from this space to the output values.

In the case of the kernel-based method, $\hat{\tau}$ is realized by the kernel allowing implicit embedding of data into a high-dimensional features space. Since the space exploited by the kernel methods may have very high dimensionality (even infinite), the expressivity of such representation can be very high. However, the mapping performed by the kernel corresponds to the *a priori* definition of an encoding function. Since the kernel defines a similarity measure, it is crucial to assess whether such similarity reflects the characteristics of the problem at hand or not. In addition, the function g is realized by a SVR (or SVM for classification tasks). This leads to an advantage for kernel methods which, thanks to the theoretical guarantees given by the bounds on the generalization performance for SVR (or SVM), may reach very good performances if the kernel is the "right" one.

In the case of RNNs, the encoding of chemical structures in a set of numerical features via $\hat{\tau}$, and the regression functions g are both realized by neural networks and training is performed jointly. Hence, a RNN is able to *learn* a direct map between the input structured domain and the activity/property output space *discovering* numerical features for the chemical structures which are optimized with respect to the prediction task at hand. In other words, the similarity measure on data is, in this case, adaptive. The output of the RNN constitutes the regression output, while the internal representations of the recursive neural network in *F* (i.e., the output of the hidden units) constitutes the neural implementation of the numerical descriptors returned by $\hat{\tau}$, i.e. a "focused" low-dimensional features space.

At this point, we may observe that the main difference between the traditional QSPR/QSAR scheme and the proposed new scheme consists in the automatic definition of the $\hat{\tau}$ function obtained by training the RNN over the regression task. This implies that no selection and/or extraction of *a priori* defined features or properties by an expert, or automatic selection procedure, is needed in the RNN realization of $\hat{\tau}$.

In the paper, we have also reported some preliminary results comparing the two above-mentioned approaches. These results seem to show that RNN are competitive with respect to a spectral tree kernel, however a more extensive set of experiments, involving several different structured domains and kernels, should be performed before drawing any conclusion.

Considering the problem to deal with general classes of graphs, the kernel-based approach offers a relevant tool since kernels for structures are highly flexible in their definition. Moreover, we briefly discussed recent developments in the RNN area that aim to extend the classes of graphs that can be dealt with.

In conclusion, we hope to have convinced the reader that, besides the specific computational characteristics of the various approaches, the ability to treat the proper inherent nature of the input data is the key feature for a successful application of the machine learning methodologies to very difficult prediction and classification tasks within pharmaceutical and medicinal chemistry research.

References

[1] Baldi P, Brunak S. *Bioinformatics: the Machine Learning Approach*. MIT Press, 1998.

[2] Baldi P, Brunak S, Frasconi P, Pollastri G, Soda G. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 1999; 15(11):937–946.

[3] Baldi P, Pollastri G. The principled design of large-scale recursive neural network architectures–DAG-RNNs and the protein structure prediction problem. *Journal of Machine Learning Research*, 2003; 4:575–602.

[4] Baldi P, Pollastri G. The principled design of large-scale recursive neural network architectures-DAG-RNNs and the protein structure prediction problem. *Journal of Machine Learning Research*, 2003; 4:575–602.

[5] Bartelett PL. The sample complexity of pattern classification with neural networks: the size of weights is more important than the size of the network. *IEEE Trans on Inform. Theory*, 1998; 44(2):525–536.

[6] Bernazzani L, Duce C, Micheli A, Mollica V, Sperduti A, Starita A, Tiné MR. Predicting thermodynamic properties from molecular structures by recursive neural networks. Comparison with classical group contributions methods. Technical Report TR-04-16, Università di Pisa, Dipartimento di Informatica, Pisa, October 2004.

[7] Bernazzani L, Duce C, Mollica V, Tiné MR, Micheli A, Sperduti A, Starita A. Recursive neural networks prediction of thermodynamic properties from molecular structures. Application to mono- and poly-functional linear molecules. In *13th International Congress on Thermal Analysis and Calorimetry ICTAC 2004*, September 2004; 96.

[8] Bianchini M, Gori M, Scarselli F. Recursive processing of cyclic graphs. In *Proc. of WCCI-IJCNN'2002*, 2002; 1:154–159.

[9] Bianucci AM, Micheli A, Sperduti A, Starita A. Application of cascade correlation networks for structures to chemistry. *Journal of Applied Intelligence (Kluwer Academic Publishers)*, 2000; 12:117–146.

[10] Bianucci AM, Micheli A, Sperduti A, Starita A. Quantitative structure-activity relationships of benzodiazepines by Recursive Cascade Correlation. In IEEE, editor, *Proceedings of IJCNN '98 - IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, May 1998; 117–122.

[11] Bishop CM. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, Oxford, 1995.

[12] Burden FR, Ford MG, Whitley DC, Winkler DA. Use of Automatic Relevance Determination in QSAR Studies Using Bayesian Neural Networks. *Journal of Chem. Inf. and Comp. Sci.*, 2000; 40:1423–1430.

[13] Ceroni A, Frasconi P, Pollastri G. Learning protein secondary structure from sequential and relational data. *Neural Networks*, 2005; 18(8):1029–1039.

[14] Cherqaoui D, Villemin D. Use of neural network to determine the boiling point of alkanes. *J. Chem. Soc. Faraday Trans.*, 1994; 90(1):97–102.

[15] Chrisman L. Learning recursive distributed representations for holistic computation. *Connection Science*, 1991; 3(4):345–366.

[16] Collins M, Duffy N. Convolution kernels for natural language. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, MIT Press, Cambridge, MA, 2002; 625–632.

[17] Cristianini N, Shawe-Taylor J. *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press, Cambridge, 2000.

[18] Cristianini N, Shawe-Taylor J. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[19] Cuturi M, Vert J-P. The context-tree kernel for strings. *Neural Networks*, 2005; 18(8):1111–1123.

[20] Duce C. *Physical chemical methods in the rational design of new materials: QSAR and calorimetric approaches*. PhD thesis, Dept. of Chemistry and Industrial Chemistry, University of Pisa, Italy, 2005.

[21] Duce C, Micheli A, Solaro R, Starita A, Tiné MR. Prediction of chemical-physical properties by neural networks for structures. XVII Convegno Italiano di Scienza e Tecnologia delle Macromolecole - AIM, 11-15 September 2005, Napoli, Italy. Macromolecular Symposia Journal, Wiley-VCH, Weinheim, Germany, 2006; 234(1):13-19.

[22] Duda RO, Hart PE, Stork DG. *Pattern Classification*. John Wiley & Sons, Inc., New York, second edition, November 2001.

[23] Džeroski S, Lavrač N. *Relational Data Mining*. Springer-Verlag, Berlin, September 2001.

[24] Jordan MI. (Editor). *Learning in Graphical Models*. MIT Press, 1999.

[25] Fahlman SE. Fast-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, 1988; 38–51.

[26] Fahlman SE. The recurrent cascade-correlation architecture. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann Publishers, San Mateo, CA, 1991; 190–196.

[27] Fahlman SE, Lebiere C. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, CA, 1990; 524–532.

[28] Frasconi P, Gori M, Sperduti A. A general framework for adaptive processing of data structures. *IEEE Trans. Neural Networks*, 1998; 9(5):768–786.

[29] Gärtner T. Exponential and geometric kernels for labelled graphs. In *Workshop on Unreal Data: Principles of Modeling Nonvectorial Data, NIPS 2002*, 2002.

[30] Gärtner T. A survey of kernels for structured data. *SIGKDD Explor. Newsl.*, 2003; 5(1):49–58.

[31] Gärtner T, Flach PA, Wrobel S. On Graph Kernels: Hardness Results and Efficient Alternatives. In *Proc. of the 16th Annual Conf. on Computational Learning Theory and 7th Kernel Workshop*, Lecture notes in Computer Science, Springer-Verlag, 2003; 2777:129-143.

[32] Gori M, Monfardini G, Scarselli F. Learning in graphic domains using neural networks. In P. Avesani M. Gori, editor, *Proceedings of the Workshop on "Sub-symbolic paradigms for learning in structured domains", 16th European Conference on Machine Learning (ECML) the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, October 2005; 31–45.

[33] Hadjipavlou-Litina D, Hansch C. Quantitative structure-activity relationships of the benzodiazepines. a review and reevaluation. *Chemical Reviews*, 1994; 94(6):1483–1505.

[34] Hammer B. *Learning with Recurrent Neural Networks*, volume 254 of *Springer Lecture Notes in Control and Information Sciences*. Springer-Verlag, 2000.

[35] Hammer B, Micheli A, Sperduti A. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 2005; 17(5):1109–1159.

[36] Hammer B, Saunders C, Sperduti A. (eds.). Special issue on Neural Networks and Kernel Methods for Structured Domains. *Neural Networks*, 2005; 18(8):1015–1123.

[37] Haussler D. Convolution Kernels on Discrete Structures. Technical Report UCS-CRL-99-10, UC Santa Cruz, 1999.

[38] Haykin S. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, 2nd edition, 1999.

[39] Horváth T, Gärtner T, Wrobel S. Cyclic pattern kernels for predictive graph mining. In *Proc. of the 2004 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2004; 158–167.

[40] Horváth T. Cyclic pattern kernels revisited. In Advances in Knowledge Discovery and Data Mining, 9th Pacific-Asia Conference, PAKDD 2005, 2005; 791-801.

[41] IUPAC. *Nomenclature of Organic Chemistry*. Pergamon Press, Oxford, 1979.

[42] Jaakkola T, Diekhans M, Haussler D. A discriminative framework for detecting remote protein homologies. *J. Comput. Biol.*, 2000; 7(1,2):95–114.

[43] Jaakkola TS, Diekhans M, Haussler D. Using the fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, AAAI Press, 1999; 149–158.

[44] Jaakkola TS, Haussler D. Exploiting generative models in discriminative classifiers. In *Proc. of Tenth Conference on Advances in Neural Information Processing Systems*, 1999; 487-493.

[45] King RD, Srinivasan A, Sternberg MJE. Relating chemical activity to structure: an examination of ILP successes. *New Generation Computing*, 1995; 13:411–433.

[46] Kuang R, Ie E, Wang K, Wang K, Siddiqi M, Freund Y, Leslie C. Profile-based string kernels for remote homology detection and motif extraction. In *Proc. of the IEEE Computational Systems Bioninformatics 2004*, 2004; 152-160.

[47] Lauritzen SL. *Graphical Models*. Oxford University Press, 1998.

[48] Lavrač N, Džeroski S. *Inductive Logic Programming : Techniques and Applications*. Ellis Horwood, 1994.

[49] Leslie C, Eskin E, Cohen A, Weston J, Noble WS. Mismatch string kernels for SVM protein classification. In *Adv. in Neural Information Processing Systems,* 2003; vol. 15.

[50] Leslie C, Eskin E, Noble WS. The spectrum kernel: A string kernel for svm protein classification. In *Proc. Of the Pacific Symposium on Biocomputing, 2002,* 2002; 564-575.

[51] Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C. Text Classification using String Kernels. *Journal of Machine Learning Research*, 2000; 2:419-444.

[52] Mahé P, Ueda N, Akutsu T, Perret J-L, Vert J-P. Extension of Marginalized Graph Kernels. In *Proc. of the 21st Int. Conf. on Machine Learning*, New York, NY, USA, 2004; 70.

[53] Mahé P, Ueda N, Akutsu T, Perret JL, Vert JP. Extensions of marginalized graph kernels. In *Proceedings of the 21 st International Conference on Machine Learning, Banff, Canada, 2004*, 2004; 552–559.

[54] Micheli A. *Recursive Processing of Structured Domains in Machine Learning*. Ph.d. thesis: TD-13/03, Department of Computer Science, University of Pisa, Pisa, Italy, December 2003.

[55] Micheli A, Portera F, Sperduti A. A preliminary empirical comparison of recursive neural networks and tree kernel methods on regression tasks for tree structured domains. *Neurocomputing*, March 2005; 64:73–92.

[56] Micheli A, Sestito AS. Dealing with graphs by neural networks. In P. Avesani M. Gori, editor, *Proceedings of the Workshop on "Sub-symbolic paradigms for learning in structured domains", 16th European Conference on Machine Learning (ECML) the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD),* October 2005; 66–75.

[57] Micheli A, Sestito AS. A new neural network model for contextual processing of graphs. Proceedings of WIRN 2005, June. Lecture notes in Computer Science, Springer-Verlag, 2006; 3931:10–17.

[58] Micheli A, Sona D, Sperduti A. Bi-causal recurrent cascade correlation. In *IJCNN'2000 - Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*, 2000; 3: 3–8.

[59] Micheli A, Sona D, Sperduti A. Recursive cascade correlation for contextual processing of structured data. In *Proc. of the Int. Joint Conf. on Neural Networks - WCCI-IJCNN'2002*, 2002; 1: 268–273.

[60] Micheli A, Sona D, Sperduti A. Contextual processing of structured data by recursive cascade correlation. *IEEE Trans. Neural Networks,* November 2004; 15(6):1396–1410.

[61] Micheli A, Sperduti A, Starita A, Bianucci AM. Analysis of the internal representations developed by neural networks for structures applied to quantitative structure-activity relationship studies of benzodiazepines. *Journal of Chem. Inf. and Comp. Sci.*, January 2001; 41(1):202–218.

[62] Micheli A, Sperduti A, Starita A, Bianucci AM. Design of new biologically active molecules by recursive neural networks. In *IJCNN'2001 - Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, Washington, DC, July 2001; 2732–2737.

[63] Mitchell T. *Machine Learning*. McGraw Hill, New York, 1997.

[64] Muggleton S. *Inductive Logic Programming*, volume 38 of *A.P.I.C. series*. Academic Press Ltd., London, 1992.

[65] Nicotra L, Micheli A, Starita A. Fisher kernel for tree structured data. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2004)*, IEEE Press. 2004; 1917–1922.

[66] Pearl J. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[67] Pollastri G, Baldi P, Vullo A, Frasconi P. Prediction of protein topologies using GIOHMMs and GRNNs. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[68] Ralaivola L, Swamidass SJ, Saigo H, Baldi P. Graph kernels for chemical informatics. *Neural Networks*, 2005; 18(8):1093–1110.

[69] Riedmiller M, Braun H. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *In Proceedings of the IEEE International Conference on Neural Networks*, 1993; 586–591.

[70] Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*. MIT Press, 1986.

[71] Menchetti SS, Costa F, Frasconi P. Weighted decomposition kernels. In Stefan Wrobel (Eds.) Luc De Raedt, editor, *Proceedings of the 22-nd International Conference on Machine Learning, Bonn, Germany*. ACM Press, 2005; 585-592.

[72] Saigo H, Vert J-P, Ueda N, Akutsu T. Protein homology detection using string alignment kernels. *Bioinformatics*, 2004; 20(11):1682–1689.

[73] Scholkopf B, Tsuda K, Vert J-P. *Kernel Methods in Computational Biology*. MIT Press, 2004.

[74] Sperduti A, Majidi D, Starita A. Extended cascade-correlation for syntactic and structural pattern recognition. In P. Perner, P. Wang, and A. Rosenfeld, editors, *Advances in Structural and Syntactical Pattern Recognition*, Lecture notes in Computer Science, Springer-Verlag, 1996; 1121:90–99.

[75] Sperduti A, Starita A. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Networks*, May 1997; 8(3):714–735.

[76] Srinivasan A, Muggleton S, King RD, Sternberg MJE. Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 1996; 85(1,2):277–299.

[77] Swamidass SJ, Chen J, Phung P, Bruand J, Ralaivola L, Baldi P. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics*, 2005; 21 Suppl. 1:359–368.

[78] Tsochantaridis I, Hofmann T, Joachims T, Altun Y. Support vector machine learning for interdependent and structured output spaces. In Carla E. Brodley, editor, *ICML '04: Twenty-first international conference on Machine learning*, New York, NY, USA, 2004. ACM Press.

[79] Vapnik VN. *Statistical Learning Theory*. Wiley, New York, 1998.

[80] Viswanathan SVN, Smola AJ. Fast kernels for string and tree matching. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, MIT Press, Cambridge, MA, 2003; 569–576.

[81] Vullo A, Frasconi P. Disulfide connectivity prediction using recursive neural networks and evolutionary information. *Bioinformatics*, 2004; 20(5):653–9.

[82] Watkins C. Kernels from matching operations. Technical report, Dept. of Computer Science, Royal Holloway, Univ. of London, 1999.

[83] Weininger D. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 1988; 28:31–36.

[84] Williams RJ, Zipser D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989; 1:270–280.