The 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS2009)
Cincinnati, Ohio, USA - June 14 - 17, 2009

IReNNS:
a recurrent neural network with independent neurons and its application in bioinformatics

Giuseppina Gini
DEI
Politecnico di Milano
Milan, Italy
gini @elet.polimi.it

Antonino Trovato
DEI
Politecnico di Milano
Milan, Italy

Thomas Ferrari
DEI
Politecnico di Milano
Milan, Italy

***Abstract*—We propose a simplified architecture for a recurrent neural network designed for learning from structures. We describe the architecture and the implementation and show the performances of the net. Two examples from the science domain are discussed: the first uses a synthetic data set and the second illustrates a chemical problem. We discuss about the results and compare them to other applications, in terms of performances and computational complexity**

Keywords-ANN; structural learning; recurrent networks; biological and chemical modelling.

I. INTRODUCTION

In the general domain of data mining many problems arise if the data of interest have a structure. In fact, most of machine learning application are designed to work with *flat* data, i.e. data with a fixed dimension and no structured or hierarchical information. A standard feed-forward neural network, for example, can process data which can be represented in fixed-dimension vectors. In many applications, however, this can be a limit, because a fixed-dimension encoding of input data can result in loosing information, or it forces some hypothesis on data, such as their maximum size or their configuration. This is the case of structures, i.e. data that can be represented as sets of elementary units connected by some relation.

The neural model we propose derives from research in learning from structures [1] - [5]. We designed it to work with an important subclass of structures: directed positional acyclic graphs (DPAGs) with a limited in-degree. In particular, our neural network architecture is thought to compute scalar functions on structures (DPAGs). This can be useful in many research fields, from chemistry to natural language, from pattern-recognition to software engineering.

Our approach has been more specifically designed for chemistry and bioinformatics. Chemical compounds can be represented using DPAGs: nodes can represent single atoms or molecular fragments; edges can represent bonds or other kinds of relation between nodes. QSPR (Quantitative Structure-Property Relationship) and QSAR (Quantitative Structure-Activity Relationship) applications have the goal to learn the relationship between the structure of chemical compounds and their physical/chemical properties or their biological activity (i.e. their interaction with other compounds, biological receptors, etc.) [6.].

Traditional approaches to QSPR/QSAR problems use molecular descriptors: structures are pre-processed in order to extract a vector of values which becomes the input for a traditional learning machine. A priori we do not know which descriptors are really useful for the specific application, so this classical approach leads to build very complex models, with lots of input variables, while some optimizations can be done only when a significant model has been built. It would be desirable to have a learning machine which can directly process structures, avoiding all these problems.

DPAGs and subclasses of them, such as trees, are traditionally used in Artificial Intelligence to represent concepts, patterns, ontologies, trees of solutions. So learning from structures can be very useful for pattern-recognition tasks. DPAGs are also used in software engineering to represent modules, dependencies, the flow of an algorithm. Learning from those structures could be useful to get estimations of some software properties.

In the following sections we describe the available methods in learning from structures, we propose our algorithm, we illustrate examples of use, and finally we compare it with other available models. In the Appendix we give details of the data used in learning.

II. INPUT FORMAT AND DPAGS

A DPAG is a particular kind of graph, i.e. a set of *nodes* and *edges*. A node is an elementary unit identified by its *label*. The label could contain categorical and/or quantitative attributes, such as the name of the node (or the entity the node represents) and some properties. An edge is a line which connects two nodes. In a DPAG edges are directed lines, so they connect two nodes in a precise order. The *in-degree* of a node is the number of edges which enter a node, while the *out-degree* is the number of edges coming out from a node.

A path is a sequence of edges connecting a node to another, following the direction of edges. A cycle is a circular path, i.e. it starts and ends on the same node.

A positional graph is a graph in which the connections to nodes have a precise position: if we consider, for example, a binary tree, we can distinguish the left-wise from the right-wise subtree and we can give a different meaning to those connections. A family tree, for example, can be represented through a binary positional tree in which left connections come from male ancestors while right connections come from female ones.

A DPAG is a directed, positional and acyclic graph, chosen as a way to represent structures of data. For our application there are three further requirements for input structures. The first one is to limit the in-degree: our computational model requires to know the maximum number of sub-trees which can be connected to a node. The second requirement is that the graph has to be connected: if we ignore the direction of edges, we should be able to find a path which connects two arbitrary nodes. The third is that a graph must have a super-source, i.e. a node *s* such that every other node of the graph can be reached by a direct path starting from *s*.

III. Neurons and recurrent networks

The most known and used approaches to the exploration of structured data is the recursive approach.

The simplest structure we can consider is a sequence, i.e. a directed graph in which each node can't have more than one predecessor and one successor. The variable length of the sequence makes it difficult to define a standard way to compute functions on it. Anyway, we can consider a recursive function that takes as input some information on the current node and the results of the elaboration on its predecessors.

The Fahlman recurrent neuron [1] receives new data about the current node from standard input, while the recurrent connection makes the neuron to compute information on the "past", as illustrated in Fig. 1.

For our model we used the same definition of generalized recurrent neuron from [5]. A generalized recurrent neuron is a standard artificial neuron whose output connection is connected to some of its inputs. Each recurrent connection brings information computed on each sub-tree.

Given a graph, if we unfold all the recursions we get a feed-forward neural network which has the same topology of the input graph, as shown in Figure 2. A similar recurrent approach to structures has been studied by [7] using vector machines.

If the parametric function to train is g_θ, the graph in Figure 2. could be represented as:

$$f(graph) = g_\theta(g_\theta(g_\theta, g_\theta), g_\theta)$$

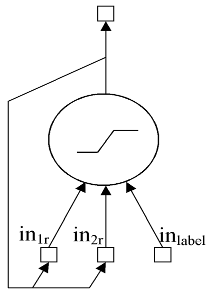

Figure 1. . A recurrent neuron

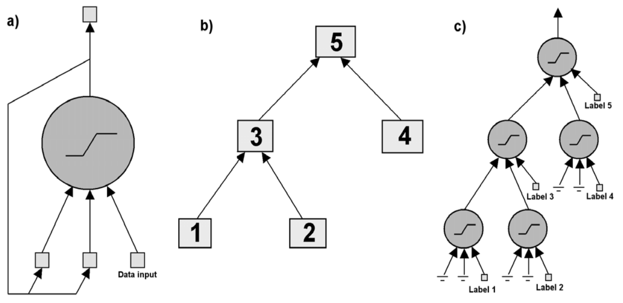

Figure 2. Figure 2 a) a recurrent neuron; b) a tree structure and c) the corresponding feed forward network that can be obtained unfolding the recursion.

IV. Network model in IReNNS

We developed our neural network from the cascade-correlation architecture [8]. The architecture of the neural network can be primarily divided in two main blocks: a *hidden layer* and an *output layer*, as illustrated in Figure 3.

The output layer can be constituted by a single neuron or a standard feed-forward neural network, while the hidden layer consists in a number of hidden units of recursive neurons. The structure of the neural network is progressively modified during the training, adding one after one new hidden units, while trying to make the output of the network more and more precise.

The training algorithm executes the following steps:

1. the initial network is constituted by the only output layer, which is trained;
2. a new hidden unit is added to the structure and it is trained in order to maximize the correlation between its output and the error of the previous network. While training, the new hidden unit is not connected to the output layer;
3. the new hidden unit is connected to the output layer, its weights are frozen and the output layer is trained again with this new input. If the results are satisfactory the algorithm stops, otherwise a new hidden unit can be introduced (as in step 2).

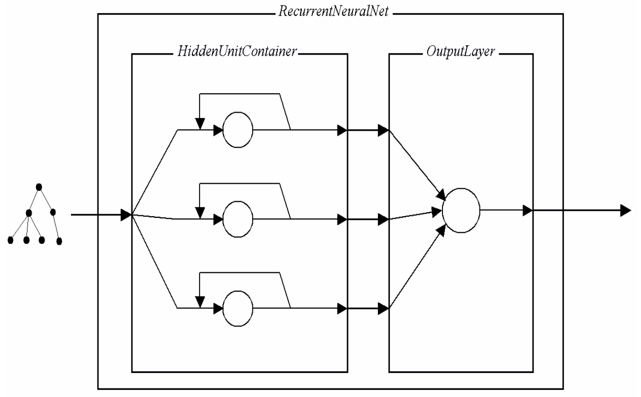

Figure 3. Schema of the network

The simplest hidden unit that can be used is a recursive neuron. A recursive neuron is characterized by a set of inputs, a net function which combines inputs (for example, a weighted sum) and an activation function (for example, a sigmoid). Inputs can be divided in three categories: a bias input, with a conventional value (usually 1); some data inputs; some recursive inputs, which let to compute the results of previous computations.

While processing a tree structure, the data inputs give the network information about the current node (the node label); the recursive inputs bring information coming from sub-trees.

For example, in processing the tree in Figure 2, the network starts from the leaves 1, 2, 4. They have no sub-trees, so the recursive neuron will not have any significant input from the recursive connections and the only information it has to compute is included in data inputs (information about the nodes 1, 2, 4) and bias. The network can now process node 3: the recursive inputs bring the results of elaboration in node 1 and 2 (sub-trees of 3), while the data input give information about the current node (3). Finally the root of the tree (5) can be processed. Recursive inputs bring the results from the sub-tree constituted by nodes 1, 2, 3 and the sub-tree 4, while the data input gives information about the current node (5).

We can imagine to unfold all the recursions during the elaboration of an input graph. If we do so, we get a feed-forward neural network constituted by the same kind of neurons and with the same topology of the input graph (figure 1,2 (c)).

Our model, IReNNS (Independent Recursive NN Structure), has *independent hidden units*. This means that hidden units are not mutually connected: their inputs come all from input graphs and from their own recursive connections; no input comes from previously introduced hidden units.

This is an important difference from the cascade-correlation architecture [8], and significantly speeds up the training algorithm. Another difference from cascade-correlation is the possibility to define more complex hidden units (for example small multi-level neural networks).

V. Training in IReNNS

IReNNS is a neural network composed by two main blocks: a *hidden unit container* and an *output layer*. The construction and the training process of the network involves operations on both these components.

Figure 4. shows how the network structure is modified during training. When the algorithm evaluates results (2, 8, 14), it checks if they are satisfactory (for example, the error is enough small or there is the risk of over-fitting) and in this case it stops; otherwise a new hidden unit is added and trained. When the new hidden unit has been trained, it is connected to the output layer as a new input and the output layer is trained again. In out implementation the output layer is constituted by a single neuron.

For hidden units we firstly have to choose a *performance function*, which can be the correlation function or some other function related to the effectiveness of the solution. Then, as hidden units are recursive elements, we have to compute the gradient of this performance function through the recursions.

Our neurons compute the following function (1):

$$x_k(n) = F(net_k(n)) = F\left(w_\theta + \sum_{i=1}^{nDataInput} w_i l_i(n) + \sum_{j=1}^{nFeedbacks} \hat{w}_j x_k(ch_j(n)) \right) \tag{1}$$

where

- F is the activation function (i.e. a sigmoid...),
- n is a node of a graph,
- $l(n)$ is the label of that node and
- $ch_j(n)$ is its j-th child.

The partial derivatives of the output of this kind of neuron have the following expressions:

$$\frac{\partial x_k(n)}{\partial w_\theta} = F'(net_k(n)) \cdot \left(1 + \sum_{j=1}^{m} \hat{w}_j^k \frac{\partial x_k(ch_j(n))}{\partial w_\theta} \right) \tag{2}$$

$$\frac{\partial x_k(n)}{\partial w_i} = F'(net_k(n)) \cdot \left(l_i(n) + \sum_{j=1}^{m} \hat{w}_j^k \frac{\partial x_k(ch_j(n))}{\partial w_i} \right) \tag{3}$$

$$\frac{\partial x_k(n)}{\partial \hat{w}_i} = F'(net_k(n)) \cdot \left(x_k(ch_i(n)) + \sum_{j=1}^{m} \hat{w}_j^k \frac{\partial x_k(ch_j(n))}{\partial \hat{w}_i} \right) \tag{4}$$

where

- F' is the first derivative of the activation function,
- wθ is the bias, wi are weights associated to data inputs
- $\hat{w}_i$ are weights associated to feedback connections.

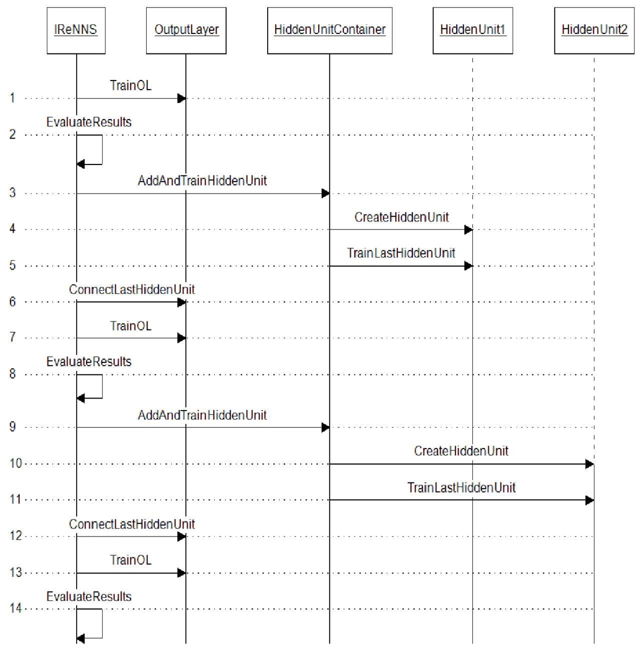

Figure 4. Steps in training IReNNS networks.

VI. Examples and Validation

The model so far described has been implemented in *java* as a standalone application, available for download [9].

The software lets import datasets from CSV files, configure training parameters, train the network, compute results and produce reports about the model performance. A module for n-fold-cross-validation is also provided.

The performances of this model have been tested on two datasets. The first trial is on a synthetic data set, according to the recommendation in [10] for testing machine learning algorithms.

This first data set represents family trees and the target function is the logarithm of the probability of transmission of a genetic illness (the autosomic recessive disease). This is an artificial dataset of 63 elements, where the target function has been determined using Mendel's laws [11]. Although the presence of 8 outliers, the validation performances made with ten fold cross validation are good with quite small models (about 30 hidden neurons). The correlation and determination coefficients are very near to 1, while the mean absolute error is 1.2993 and, removing outliers, it goes down to 0.9742.

The second dataset is a QSPR(Quantitative Structure Property Relationship) application [8], [12]: the input structure is a simplified representation of 150 Alkanes molecules, while the target function is their boiling point. A set of alkanes with their experimental boiling point has been provided in [8] and already used by [13] to develop a neural model. See the dataset in Appendix 1.

Alkanes are saturated acyclic hydrocarbons, as illustrated in Fig. 5. Those structures can be represented as strings using the SMILES codes [14]. In this case, the smiles representation is given in equation 5:

ch3(c (ch3, ch3, ch2 (ch(ch3, ch3)))) (5)

Since the hydrogens are fully distributed and all the bonds are single bonds, we can remove the hydrogens, as in Fig 6, and obtain the simplified representation of equation 6:

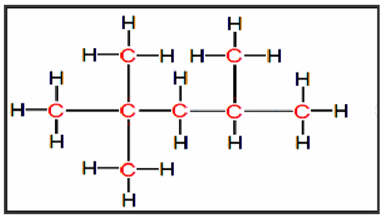

Figure 5. An alkane molecule, the 2,2,4 - trimethil pentan.

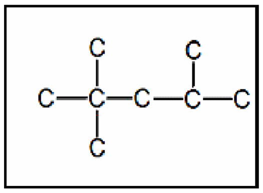

Figure 6. The reduced representation

C(C (C, C, C (C(C, C)))) (6)

Using the simplified structure representation of alkanes we trained IReNNS models to predict their boiling point. The results have been tested using ten-fold cross-validation.

We may observe, from the experimental data, that each new Carbon atom increases the boiling point of about 20 to 30 degrees, but this rule is not linear. In particular the small molecules (as methane) have a different behaviour and are omitted in the results of the Cherqaoui model. We consider instead all the molecules with a maximum of 10 Carbons, for a total of 150 compounds as above indicated.

Our models have a variable complexity from 25 to 63 hidden neurons and, due to the high variability of the target function (from –164 °C to 174 °C) and the presence of outliers, the mean absolute error of our models was 5.37 °C, while the correlation and determination coefficients had the value of 0.989 and 0.969.

We see in Figure 7 the results of the chosen IReNS model, and in Table 1 a comparison with the other mentioned models. The comparison of the results in Table 1 is based on four parameters: mean square error, mean absolute error, correlation coefficient, coefficient of determination (see in Appendix 2 the definitions of the used parameters).

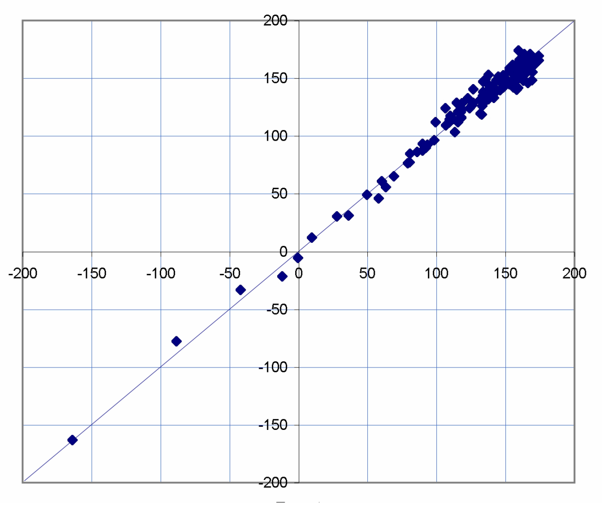

Figure 7. .Predicted boiling point versus experimental boiling point as obtained from our IReNNS model on Alkanes

TABLE I. Comparison of different models to predict the alkanes boiling point

Model	ε_{sq}	ε_{abs}	R	R^2
Cherqaoui	59,56	3,09	0,997	0,966
Cascade-	59,53	3,28	0,989	0,865
IReNNS	50,18	5,37	0,989	0,969

VII. Discussion and Conclusion

The three models before compared have similar performances. With respect to IReNNS, the cascade correlation model has a lower R^2 but also a lower absolute error. The Cherqaoui model, a multilayer feed-forward NN, reaches the same R^2 at the cost of computing a vector of molecular characteristics to be used as input to the network, and omitting small molecules that have a peculiar behavior. The explanation of the worse result in the mean absolute error is in the small number of neurons that we have in the network when it is stopped. This number can be reduced a bit with a greater number of neurons, but with computational costs

The advantage of our IReNNS architecture is the simplicity, the reduction in the number of the hidden neurons, and the limited computer time necessary.

Appendix 1: The Alkanes data set

Id	Name	Structure	C°
1	methane	C	-164
2	ethane	C(C)	-88,6
3	propane	C(C(C))	-42,1
4	2-methylpropane	C(C(C,C))	-11,7
5	butane	C(C(C(C)))	-0,5
6	2,2-dimethylpropane	C(C(C,C,C))	9,5
7	2-methylbutane	C(C(C,C(C)))	27,8
8	pentane	C(C(C(C(C))))	36,1
9	2,2-dimethylbutane	C(C(C,C,C(C)))	49,7
10	2,3-dimethylbutane	C(C(C,C(C,C)))	58
11	2-methylpentane	C(C(C,C(C(C))))	60,3
12	3-methylpentane	C(C(C(C,C(C))))	63,3
13	hexane	C(C(C(C(C(C)))))	69
14	2,2,3-trimethylbutane	C(C(C,C,C(C,C)))	80,9
15	2,2-dimethylpentane	C(C(C,C,C(C(C))))	79,2
16	3,3-dimethylpentane	C(C(C(C,C,C(C))))	86,1
17	2,3-dimethylpentane	C(C(C,C(C,C(C))))	89,8
18	2,4-dimatilpentane	C(C(C,C(C(C,C))))	80,5
19	2-methylhexane	C(C(C,C(C(C(C)))))	90
20	3-methylhexane	C(C(C(C,C(C(C)))))	92
21	3-ethylpentane	C(C(C(C(C),C(C))))	93,5
22	heptane	C(C(C(C(C(C(C))))))	98,4
23	2,2,3,3-tetramethylbutane	C(C(C,C,C(C,C,C)))	106,5
24	2,2,3-trimethylpentane	C(C(C,C,C(C,C(C))))	110
25	2,3,3-trimethylpentane	C(C(C,C(C,C,C(C))))	114,7
26	2,2,4-trimethylpentane	C(C(C,C,C(C(C,C))))	99,2
27	2,2-dimethylhexane	C(C(C,C,C(C(C(C)))))	106,8
28	3,3-dimethylhexane	C(C(C(C,C,C(C(C)))))	112
29	3-ethyl-3-methylpentane	C(C(C(C,C(C),C(C))))	118,2
30	2,3,4-trimethylpentane	C(C(C,C(C,C(C,C))))	113,4
31	2,3-dimethylhexane	C(C(C,C(C,C(C(C)))))	115,6
32	3-ethyl-2-methylpentane	C(C(C,C(C(C),C(C))))	115,6
33	3,4-dimethylhexane	C(C(C(C,C(C,C(C)))))	117,7
34	2,4-dimethylhexane	C(C(C,C(C(C,C(C)))))	109,4
35	2,5-dimethylhexane	C(C(C,C(C(C(C,C)))))	109
36	2-methylheptane	C(C(C,C(C(C(C(C))))))	117,6
37	3-methylheptane	C(C(C(C,C(C(C(C))))))	118
38	4-methylheptane	C(C(C(C(C,C(C(C))))))	117,7
39	3-ethylhexane	C(C(C(C(C),C(C(C)))))	118,5
40	octane	C(C(C(C(C(C(C(C)))))))	125,7
41	2,2,3,3-tetramethylpentane	C(C(C,C,C(C,C,C(C))))	140,27
42	2,2,3,4-tetramethylpentane	C(C(C,C,C(C,C(C,C))))	133
43	2,2,3-trimethylhexane	C(C(C,C,C(C,C(C(C)))))	131,7
44	2,2-dimethyl-3-ethylpentane	C(C(C,C,C(C(C),C(C))))	133,83
45	3,3,4-trimethylhexane	C(C(C(C,C,C(C,C(C)))))	140,5
46	2,3,3,4-tetramethylpentane	C(C(C,C(C,C,C(C,C))))	141,5
47	2,3,3-trimethylhexane	C(C(C,C(C,C,C(C(C)))))	137,7
48	2,3-dimethyl-3-ethylpentane	C(C(C,C(C,C(C),C(C))))	141,6
49	2,2,4,4-tetramethylpentane	C(C(C,C,C(C(C,C,C))))	122,7
50	2,2,4-trimethylhexane	C(C(C,C,C(C(C,C(C)))))	126,5
51	2,4,4-trimethylhexane	C(C(C,C(C(C,C,C(C)))))	126,5
52	2,2,5-trimethylhexane	C(C(C,C,C(C(C(C,C)))))	124
53	2,2-dimethylheptane	C(C(C,C,C(C(C(C(C))))))	132,7
54	3,3-dimethylheptane	C(C(C(C,C,C(C(C(C))))))	137,3
55	4,4-dimethylheptane	C(C(C(C(C,C,C(C(C))))))	135,2
56	3-ethyl-3-methylhexane	C(C(C(C,C(C),C(C(C)))))	140,6
57	3,3-dimethylpentane	C(C(C(C(C),C(C),C(C))))	146,2
58	2,3,4-trimethylhexane	C(C(C,C(C,C(C,C(C)))))	139
59	2,4-dimethyl-3-ethylpentane	C(C(C,C(C(C),C(C,C))))	136,73
60	2,3,5-trimethylhexane	C(C(C,C(C,C(C(C,C)))))	131,3
61	2,3-dimethylpentane	C(C(C,C(C,C(C(C(C))))))	140,5
62	3-ethyl-2-methylhexane	C(C(C,C(C(C),C(C(C)))))	138
63	3,4-dimethylheptane	C(C(C(C,C(C,C(C(C))))))	140,1
64	3-ethyl-4-methylhexane	C(C(C(C(C),C(C,C(C)))))	140,4
65	2,4-dimethylheptane	C(C(C,C(C(C,C(C(C))))))	133,5
66	4-ethyl-2-methylhexane	C(C(C,C(C(C(C),C(C)))))	133,8
67	3,5-dimethylheptane	C(C(C(C,C(C(C,C(C))))))	136
68	2,5-dimethylheptane	C(C(C,C(C(C(C,C(C))))))	136
69	2,6-dimethylheptane	C(C(C,C(C(C(C(C,C))))))	135,2
70	2-methyloctane	C(C(C,C(C(C(C(C(C)))))))	142,8
71	3-methyloctane	C(C(C(C,C(C(C(C(C)))))))	143,3
72	4-methyloctane	C(C(C(C(C,C(C(C(C)))))))	142,4
73	3-ethylheptane	C(C(C(C(C),C(C(C(C))))))	143
74	4-ethylheptane	C(C(C(C(C(C),C(C(C))))))	141,2
75	nonane	C(C(C(C(C(C(C(C(C))))))))	151,77
76	2,2,3,3,4-pentamethylpentane	C(C(C,C,C(C,C,C(C,C))))	166,05
77	2,2,3,3-tetramethylhexane	C(C(C,C,C(C,C,C(C(C)))))	158
78	3-ethyl-2,2,3-trimethylpentane	C(C(C,C,C(C,C(C),C(C))))	168
79	3,3,4,4-tetramethylhexane	C(C(C(C,C,C(C,C,C(C)))))	170,5
80	2,2,3,4,4-pentamethylpentane	C(C(C,C,C(C,C(C,C,C))))	159,29
81	2,2,3,4-tetramethylhexane	C(C(C,C,C(C,C(C,C(C)))))	154,9
82	3-ethyl-2,2,4-trimethylpentane	C(C(C,C,C(C(C),C(C,C))))	155,3
83	2,3,4,4-tetramethylhexane	C(C(C,C(C,C(C,C,C(C)))))	162,2
84	2,2,3,5-tetramethylhexane	C(C(C,C,C(C,C(C(C,C)))))	148,4
85	2,2,3-trimethylheptane	C(C(C,C,C(C,C(C(C(C))))))	158
86	2,2-dimethyl-3-ethylhexane	C(C(C,C,C(C(C),C(C(C)))))	159
87	3,3,4-trimethylheptane	C(C(C(C,C,C(C,C(C(C))))))	164
88	3,3-dimethyl-4-ethylhexane	C(C(C(C,C,C(C(C),C(C)))))	165
89	2,3,3,4-tetramethylhexane	C(C(C,C(C,C,C(C,C(C)))))	164,59
90	3,4,4-trimethylheptane	C(C(C(C,C(C,C,C(C(C))))))	164
91	3,4-dimethyl-3-ethylhexane	C(C(C(C,C(C),C(C,C(C)))))	170
92	3-ethyl-2,3,4-trimethylpentane	C(C(C,C(C,C(C),C(C,C))))	169,44
93	2,3,3,5-tetramethylhexane	C(C(C,C(C,C,C(C(C,C)))))	153
94	2,3,3-trimethylheptane	C(C(C,C(C,C,C(C(C(C))))))	160,1
95	2,3-dimethyl-3-ethylhexane	C(C(C,C(C,C(C),C(C(C)))))	169
96	3,3-diethyl-2-methylpentane	C(C(C,C(C(C),C(C),C(C))))	174
97	2,2,4,4-tetramethylhexane	C(C(C,C,C(C(C,C,C(C)))))	153,3
98	2,2,4,5-tetramethylhexane	C(C(C,C,C(C(C,C(C,C)))))	148,2
99	2,2,4-trimethylheptane	C(C(C,C,C(C(C,C(C(C))))))	147,7
100	2,2-dimethyl-4-ethylhexane	C(C(C,C,C(C(C(C),C(C)))))	147
101	3,3,5-trimethylheptane	C(C(C(C,C,C(C(C,C(C))))))	155,68
102	2,4,4-trimethylheptane	C(C(C,C(C(C,C,C(C(C))))))	153
103	2,4-dimethyl-4-ethylhexane	C(C(C,C(C(C,C(C),C(C)))))	158
104	2,2,5,5-tetramethylhexane	C(C(C,C,C(C(C(C,C,C)))))	137,46
105	2,2,5-trimethylheptane	C(C(C,C,C(C(C(C,C(C))))))	148
106	2,5,5-trimethylheptane	C(C(C,C(C(C(C,C,C(C))))))	152,8
107	2,2,6-trimethylheptane	C(C(C,C,C(C(C(C(C,C))))))	148,2
108	2,2-dimethyloctane	C(C(C,C,C(C(C(C(C(C)))))))	155
109	3,3-dimethyloctane	C(C(C(C,C,C(C(C(C(C)))))))	161,2
110	4,4-dimethyloctane	C(C(C(C(C,C,C(C(C(C)))))))	157,5
111	3-ethyl-3-methylheptane	C(C(C(C,C(C),C(C(C(C))))))	163,8
112	4-ethyl-4-methylheptane	C(C(C(C(C,C(C),C(C(C))))))	167
113	3,3-diethylhexane	C(C(C(C(C),C(C),C(C(C)))))	166,3
114	2,3,4,5-tetramethylhexane	C(C(C,C(C,C(C,C(C,C)))))	161
115	2,3,4-trimethylheptane	C(C(C,C(C,C(C,C(C(C))))))	163
116	2,3-dimethyl-4-ethylhexane	C(C(C,C(C,C(C(C),C(C)))))	164
117	2,4-dimethyl-3-ethylhexane	C(C(C,C(C(C),C(C,C(C)))))	164
118	3,4,5-trimethylheptane	C(C(C(C,C(C,C(C,C(C))))))	164
119	2,4-dimethyl-3-isopropylpentane	C(C(C,C(C(C,C),C(C,C))))	157,04
120	3-isopropyl-2-methylhexane	C(C(C,C(C(C,C),C(C(C)))))	163

121	2,3,5-trimethylheptane	C(C(C,C(C,C(C(C,C(C))))))	157
122	2,5-dimethyl-3-ethylihexane	C(C(C,C(C(C),C(C(C,C)))))	157
123	2,4,5-trimethylheptane	C(C(C,C(C(C,C(C,C(C))))))	157
124	2,3,6-trimethylheptane	C(C(C,C(C,C(C(C(C,C))))))	155,7
125	2,3-dimethyloctane	C(C(C,C(C,C(C(C(C(C)))))))	164,31
126	3-ethyl-2-methylheptane	C(C(C,C(C(C),C(C(C(C))))))	166
127	3,4-dimethyloctane	C(C(C(C,C(C,C(C(C(C)))))))	166
128	4-isopropylheptane	C(C(C(C(C(C,C),C(C(C))))))	160
129	4-ethyl-3-methylheptane	C(C(C(C,C(C(C),C(C(C))))))	167
130	4,5-dimethyloctane	C(C(C(C(C,C(C,C(C(C)))))))	162,1
131	3-ethyl-4-methylheptane	C(C(C(C(C),C(C,C(C(C))))))	167
132	3,4-diethylhexane	C(C(C(C(C),C(C(C),C(C)))))	162
133	2,4,6-trimethylheptane	C(C(C,C(C(C,C(C(C,C))))))	144,8
134	2,4-dimethyloctane	C(C(C,C(C(C,C(C(C(C)))))))	153
135	4-ethyl-2-methylheptane	C(C(C,C(C(C(C),C(C(C))))))	160
136	3,5-dimethyloctane	C(C(C(C,C(C(C,C(C(C)))))))	160
137	3-ethyl-5-methylheptane	C(C(C(C(C),C(C(C,C(C))))))	158,3
138	2,5-dimethyloctane	C(C(C,C(C(C(C,C(C(C)))))))	156,8
139	5-ethyl-2-methylheptane	C(C(C,C(C(C(C(C),C(C))))))	159,7
140	3,6-dimethyloctane	C(C(C(C,C(C(C(C,C(C)))))))	160
141	2,6-dimethyloctane	C(C(C,C(C(C(C(C,C(C)))))))	158,84
142	2,7-dimethyloctane	C(C(C,C(C(C(C(C(C,C)))))))	159,87
143	2-methylnonane	C(C(C,C(C(C(C(C(C(C))))))))	167
144	3-methylnonane	C(C(C(C,C(C(C(C(C(C))))))))	167,8
145	4-methylnonane	C(C(C(C(C,C(C(C(C(C))))))))	165,7
146	3-ethyloctane	C(C(C(C(C),C(C(C(C(C)))))))	166
147	5-methylnonane	C(C(C(C(C(C,C(C(C(C))))))))	165,1
148	4-ethyloctane	C(C(C(C(C(C),C(C(C(C)))))))	163,64
149	4-propilheptane	C(C(C(C(C(C(C)),C(C(C))))))	162
150	decane	C(C(C(C(C(C(C(C(C(C))))))))))	174,12

APPENDIX 2: PARAMETERS FOR MODEL EVALUATION

$$\varepsilon_{sq} = \frac{1}{N}\sum_{i=1}^{N}(t_i - y_i)^2$$

Mean Square error

The squared difference between the target and the predicted output for each element of the data set.

$$\varepsilon_{abs} = \frac{1}{N}\sum_{i=1}^{N}|t_i - y_i|$$

Mean Absolute Error

The absolute difference between the target and the predicted output.

$$R = \frac{\sum_i (t_i - E(t))\cdot(y_i - E(y))}{\sqrt{\left(\sum_i (t_i - E(t))^2\right)\cdot\left(\sum_i (y_i - E(y))^2\right)}}$$

Correlation Coefficient

"E" computes the average.

$$R^2 = \frac{\left[\sum_i (t_i - E(t))\cdot(y_i - E(y))\right]^2}{\left(\sum_i (t_i - E(t))^2\right)\cdot\left(\sum_i (y_i - E(y))^2\right)}$$

Determination Coefficient

For linear models it is the square of R. The definition for non linear models is not unique.

REFERENCES

[1] Fahlman S.E., Lebiere C. (1990). The cascade-correlation learning architecture. In *Advances in neural information processing systems 2*, edited by D.S. Touretzky, San Mateo, CA, Morgan Kaufmann, 524-532.

[2] Frasconi P., Gori M., Sperduti A. (1997). A general framework for processing of data structures. *IEEE Transactions on Neural Networks,* 8 (3), 714-735.

[3] Hammer B., Micheli A., Sperduti A. (2005). Universal Approximation Capability of Cascade Correlation for Structures. *Neural Computation*, 17(5), 1109-1159.

[4] Pollack J. B. (1990). Recursive Distributed Representations. *Artificial Intelligence*, 46(1-2), 77-105.

[5] Sperduti A., Starita A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3), 714-735.

[6] Schultz T. W., Cronin M. T. D., Walker J. D., Aptula A. O. (2003), Quantitative structure–activity relationships (QSARs) in toxicology: a historical perspective. Journal of Molecular Structure (Theochem) 622, 1–22.

[7] Goulon-Sigwalt-Abram A., Duprat A., Dreyfus G. (2005). From Hopfield nets to recursive networks to graph machines: numerical machine learning for structured data. *Theoretical Computer Science*, 344, 298-334.

[8] Bianucci A.M., Micheli A., Sperduti A., Starita A. (2000). Application of Recursive Cascade-Correlation Networks for Structures to Chemistry. *J.l of Applied Intelligence*, 12, 117-146.

[9] IReNNS http://sourceforge.net/project/showfiles.php?group_id=240254&package_id=313096

[10] Langley P., Crafting Papers on Machine Learning, from http://www-csli.stanford.edu/icml2k/craft.html

[11] Sijbrands E., Westendorp R., Defesche J., de Meier P., Smelt A., Kastelein J. (2001). Mortality over two centuries in large pedigree with familial hypercholesterolaemia: family tree mortality study. *BMJ*, 322(7293), 1019–1023.

[12] Bernazzani L., Duce C., Micheli A., Mollica V., Sperduti A., Starita A., Tiné M.R. (2006). Predicting Physical-Chemical Properties of Compounds from Molecular Structures by Recursive Neural Networks. *J. Chem. Inf. Model.*, 46(5), 2030-2042.

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

[14] Weininger D., Weininger A., Weininger J. L. (1989). SMILES. 2. Algorithm for generation of unique SMILES notation. *Journal of Chemical Information and Computer Science*, 29, 97-101.