Kernel Base Learning Methods: Regularization Networks and RBF Networks

Petra Kudová*, Roman Neruda

Institute of Computer Science,
Academy of Sciences of the Czech Republic,
18207 Prague, Czech Republic
petra@cs.cas.cz

Abstract. Kernel based learning methods are subject of great interest at present. We discuss two kernel based learning methods, namely the Regularization Networks (RN) and the Radial Basis Function Network (RBF networks).
The RNs are derived from the regularization theory, had been studied thoroughly from a function approximation point of view, and therefore have very good theoretical background.
The RBF networks represent a model of artificial neural networks with both neuro-physiological and mathematical motivation. In addition they may be treated as a generalised form of Regularization Networks, i.e. RN with increased number of kernel functions.
We demonstrated the performance of both approaches on experiments, including both benchmark and real-life learning tasks. We claim that the performance of RN and RBF network is comparable in terms of generalisation error. The RN approach usually leads to solutions with higher model complexity (high number of base units). In this situations, the RBF networks can be used as a 'cheaper' alternative.

1 Introduction

The problem of *learning from examples* (also called *supervised learning*) is a subject of great interest. Systems with the ability to autonomously learn a given task, would be very useful in many real life applications, namely those involving prediction, classification, control, etc.

The problem can be formulated as follows. We are given a set of examples $\{(\boldsymbol{x}_i, y_i) \in R^d \times R\}_{i=1}^N$ that was obtained by random sampling of some real function f, generally in the presence of noise. To this set we refer as *a training set*. Our goal is to recover the function f from data, or find the best estimate of it. It is not necessary that the function exactly interpolates all the given data points, but we need a function with good *generalisation*. That is a function that gives relevant outputs also for the data not included in the training set.

* The work was supported by the project 1ET100300419 of the Program Information Socienty (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation".

The problem of learning from examples is studied as a function approximation problem. Given the data set, we are looking for the function that approximate the unknown function f. It can be done by *Empirical Risk Minimization*, i.e. minimizing the functional $H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}_i) - y_i)^2$ over a chosen *hypothesis space*. In section 2 we will study the problem of learning from examples as a function approximation problem and show how a regularization network (RN) is derived from regularization theory. In 3 we will discuss a learning algorithm for RNs.

The learning problem can be also handled by artificial neural networks (ANNs). There is a good supply of network architectures and corresponding supervised learning algorithms (see [1]). In this case the model, that is a particular type of neural network, is chosen in advance and its parameters are tuned during learning so as to fit the given data. In terms of function approximation, the Empirical Risk is minimized over the hypothesis space defined by the chosen type of ANN, i.e. the space of functions representable by this type of ANN. In section 4 we will describe one type of neural network – an RBF network, which is closely related to RN.

In section 5 the performances of RBF network and RN are compared on experiments, including both benchmark and real learning tasks.

2 Derivation of Regularization Networks

In this section we will study the problem of learning from examples by means of regularization theory.

We are given a set of examples $\{(\boldsymbol{x}_i, y_i) \in R^d \times R\}_{i=1}^{N}$ obtained by random sampling of some real function f and we would like to find this function.

Since this problem is ill-posed, we have to add some a priori knowledge about the function. It is usually assumed that the function is *smooth*, in the sense that two similar inputs corresponds to two similar outputs and the function does not oscillate too much. This is the main idea of the regularization theory, where the solution is found by minimizing the functional (1) containing both the data and smoothness information.

$$H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(\boldsymbol{x}_i) - y_i)^2 + \gamma\Phi[f], \tag{1}$$

where Φ is called a *stabilizer* and $\gamma > 0$ is *the regularization parameter* controlling the trade off between the closeness to data and the smoothness of the solution. The regularization scheme (1) was first introduced by Tikhonov [2] and therefore it is often called a Tikhonov regularization.

Poggio, Girrosi and Jones in [3] proposed a form of a smoothness functional based on Fourier transform:

$$\Phi[f] = \int_{R^d} d\boldsymbol{s} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})}, \tag{2}$$

where $\tilde{f}$ indicates the Fourier transform of f , $\tilde{G}$ is some positive function that goes to zero for $||s|| \to \infty$ (i.e. $1/\tilde{G}$ is a high-pass filter). The stabiliser (2) measures the energy in the high frequency and so penalises the functions with high oscilations.

It was shown that for a wide class of stabilizers in form of (2) the solution has a form of feed-forward neural network with one hidden layer, called *Regularization Network*, and that different types of stabilizers lead to different types of Regularization Networks [3, 4].

Poggio and Smale in [4] studied the Regularization Networks derived using a Reproducing Kernel Hilbert Space (RKHS) as the hypothesis space.

Let $\mathcal{H}_K$ be an RKHS defined by a symmetric, positive-definite kernel function $K_{\boldsymbol{x}}(\boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$. Then if we define the stabiliser by means of norm in $\mathcal{H}_K$ and minimise the functional

$$\min_{f \in \mathcal{H}_K} H[f], \text{where } H[f] = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(\boldsymbol{x}_i))^2 + \gamma ||f||_K^2 \tag{3}$$

over the hypothesis space $\mathcal{H}_K$, the solution of minimisation (3) is unique and has the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} c_i K_{\boldsymbol{x}_i}(\boldsymbol{x}), \qquad (N\gamma I + K)\boldsymbol{c} = \boldsymbol{y}, \tag{4}$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(\boldsymbol{x_i}, \boldsymbol{x_j})$, and $\boldsymbol{y} = (y_1, \ldots, y_N)$.

Girrosi in [**?**] showed that for positive definite functions of the form $K(\boldsymbol{x}-\boldsymbol{y})$ (such as Gaussian function) the norm in RKHS defined by K is equivalent to stabilizer (2):

$$||f||_K^2 = \int_{R^d} d\boldsymbol{s} \frac{|\tilde{f}(\boldsymbol{s})|^2}{\tilde{G}(\boldsymbol{s})}. \tag{5}$$

3 Learning with Regularization Networks

Input: `Data set` $\{\boldsymbol{x}_i, y_i\}_{i=1}^N \subseteq X \times Y$ **Output:** `Function` f.

1. `Choose a symmetric, positive-definite function` $K_{\boldsymbol{x}}(\boldsymbol{x}')$, `continuous on` $X \times X$.
2. `Create` $f : X \to Y$ `as` $f(\boldsymbol{x}) = \sum_{i=1}^{N} c_i K_{\boldsymbol{x}_i}(\boldsymbol{x})$ `and compute` $\boldsymbol{c} = (c_1, \ldots, c_N)$ `by solving`

$$(N\gamma I + K)\boldsymbol{c} = \boldsymbol{y}, \tag{6}$$

`where` I `is the identity matrix, K is the matrix` $K_{i,j} = K(\boldsymbol{x_i}, \boldsymbol{x_j})$, `and` $\boldsymbol{y} = (y_1, \ldots, y_N)$, $\gamma > 0$ `is real number.`

Algorithm 3.1

The form of Regularization Network in (4) leads in the learning algorithm (3.1).

The power of this algorithm is in its simplicity and effectiveness, the drawback is the size of the model (that is a number of kernel functions), which corresponds to the size of the training set, and so the tasks with huge data sets lead to solutions of implausible size.

The algorithm suppose that the type of kernel function and regularization parameter γ are chosen in advanced.

Let us discuss closely the case of Gaussian kernel $K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\left(\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|}{b}\right)^2}$, which is widely used.

Once the width b and the regularization parameter γ are fixed, the algorithm reduces to the problem of solving linear system of equations (6).

Since the system has N variables, N equations, K is positive-definite and $(N\gamma I + K)$ is strictly positive, it is well-posed, i.e. is has a unique solution and the solution exists. But we would also like it to be well-conditioned, i.e. insensitive to small perturbations of the data. In other words, we would like the condition number of the matrix $(N\gamma I + K)$ to be small, which is fulfilled if $N\gamma$ is large. Note that we are not entirely free to choose γ, because with too large γ we loose the closeness to data. See figure 3.

The second parameter b determines the width of the Gaussians, and should reflect the density of data points. Suppose that the distances between the data points are high or the widths are small, than the matrix K has 1s on diagonal and small numbers everywhere else and therefore is well-conditioned, but if the widths are too small the matrix goes to identity and contains almost no information. On the other hand, if the widths are too large, all elements of the matrix K are close to 1 and its condition number tends to be high.

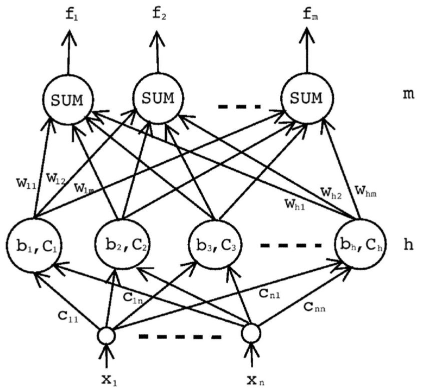
The real performance of the algorithm depends significantly on the choice of parameters γ and b. The optimal choice of these parameters depends on a particular data set. See figure 2.

We estimate both parameters by adaptive grid search and k-fold crossvalidation. Adaptive grid search starts with a coarse grid of pairs (γ, b) defined by user and for each pair computes the crossvalidation error. Then finer grid is evaluated only in the smaller region containing the pair with the lowest crossvalidation error. The process is repeated until the crossvalidation error stops decreasing. Then the parameters with the lowest crossvalidation error are picked up and used for evaluation of the algorithm on the whole training set.

4 RBF neural networks

An RBF neural network (RBF network) represents a relatively new model of neural network. On the contrary to classical models (multilayer perceptrons, etc.) it is a network with local units which was motivated by the presence of many local response units in human brain. Other motivation came from numerical mathematics, radial basis functions (RBF) were first introduced in the solution of real multivariate problems [5].

In the framework of regularization networks, the RBF networks belong to the family of generalised regularization networks. Generalized regularization networks are RN with lower number of kernels than data points and also it is not necessary that the kernels are uniform (so for example the network with gaussian kernels may use kernels with different widths).



$$y(\boldsymbol{x}) = \varphi\left(\frac{\parallel \boldsymbol{x} - \boldsymbol{c} \parallel_C}{b}\right) \quad (7)$$

$$f_s(\boldsymbol{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\parallel \boldsymbol{x} - \boldsymbol{c}_j \parallel_{C_j}}{b_j}\right) \quad (8)$$

Fig. 1. a) RBF network architecture b) RBF network function

An RBF network is a standard feed-forward neural network with one hidden layer of RBF units and linear output layer (fig. 1). By an RBF unit we mean a neuron with n real inputs and one real output, realising a radial basis function (7), usually Gaussian. Instead of the Euclidean norm we use the *weighted norm* $\| \cdot \|_C$, where $\|\boldsymbol{x}\|_C^2 = (C\boldsymbol{x})^T(C\boldsymbol{x}) = \boldsymbol{x}^T C^T C\boldsymbol{x}$.

The network computes a function $\boldsymbol{f} = (f_1, \ldots, f_m)$ as linear combination of outputs of the hidden layer (see (8)).

The goal of RBF network learning is to find the parameters (i.e. centers $\boldsymbol{c}$, widths b, norm matrices C and weights w) so as the network function approximates the function given by the training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i) \in R^n \times R^m\}_{i=1}^N$.

There is a variety of algorithms for RBF network learning, in our past work we studied their behaviour and possibilities of their combinations [6, 7].

The two most significant algorithms,*Three step learning* and *Gradient learning*, are sketched in Algorithm 2.1 and Algorithm 2.2. See [6] for details.

Input: `Data set` $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ **Output:** $\{\boldsymbol{c}_i, b_i, C_i, w_{ij}\}_{i=1..h}^{j=1..m}$

1. `Set the centers` $\boldsymbol{c}_i$ `by a k-means clustering.`
2. `Set the widths` b_i `and matrices` C_i`.`
3. `Set the weights` w_{ij} `by solving` $\Phi W = D$`.`

$$D_{ij} = \sum_{t=1}^{N} y_{tj} e^{-\left(\frac{\|\boldsymbol{x}_t - \boldsymbol{c}_i\|_{C_i}}{b_i}\right)^2}, \Phi_{qr} = \sum_{t=1}^{N} e^{-\left(\frac{\|\boldsymbol{x}_t - \boldsymbol{c}_q\|_{C_q}}{b_q}\right)^2} e^{-\left(\frac{\|\boldsymbol{x}_t - \boldsymbol{c}_r\|_{C_r}}{b_r}\right)^2}$$

Algorithm 4.1

Input: `Data set` $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ **Output:** $\{\boldsymbol{c}_i, b_i, C_i, w_{ij}\}_{i=1..h}^{j=1..m}$

1. `Put the small part of data aside as an evaluation set` ES`,`
   `keep the rest as a training set` TS `.`

2. $\forall j\ \boldsymbol{c}_j(i) \leftarrow$ `random sample from` TS_1, $\forall j\ b_j(i), \Sigma_j^{-1}(i) \leftarrow$ `small random value,` $i \leftarrow 0$
3. $\forall j, p(i)$ `in` $\boldsymbol{c}_j(i), b_j(i), \Sigma_j^{-1}(i)$:
   $\Delta p(i) \leftarrow -\epsilon \frac{\delta E_1}{\delta p} + \alpha \Delta p(i-1), \qquad p(i) \leftarrow p(i) + \Delta p(i)$
4. $E_1 \leftarrow \sum_{\boldsymbol{x} \in TS_1} (f(\boldsymbol{x}) - y_i)^2$, $E_2 \leftarrow \sum_{\boldsymbol{x} \in TS_2} (f(\boldsymbol{x}) - y_i)^2$
5. `If` E_1 `and` E_2 `are decreasing,` $i \leftarrow i+1$`, go to 3, else STOP. If` E_2 `started to increase, STOP.`

Algorithm 4.2

5 Experimental results

- uceno na bladovi (jaky procesor + pamet) - rn uceno jak bylo popsano - rbf uceno gradientem, prumer a std z 10 vypoctu, pocitano pro 10, 15, 20, 30 jednotek ... uvedena nejlepsi - v tabulkach je vzdycky error percentage viz. itat clanek - uceno na treninkove mnozine (vcetne crossvalidace), chyba na testovaci

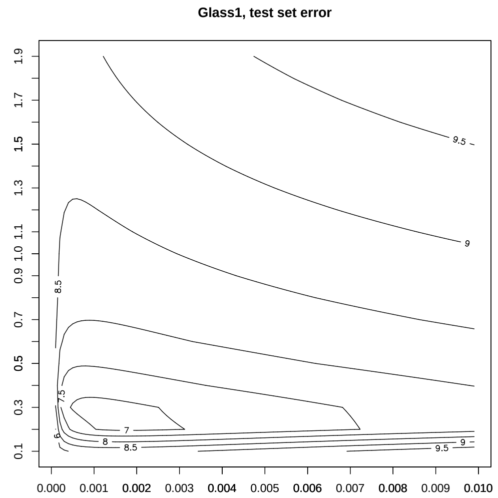


Fig. 2. Dependancy of error on testing data set on regularization parameter γ and width b.

pl1 jeden den zpatky pl2 dva dni zpatky s znamena, srazky z aktualniho dne

6 Conclusion

References

1. Haykin, S.: Neural Networks: a comprehensive foundation. 2nd edn. Tom Robins (1999)

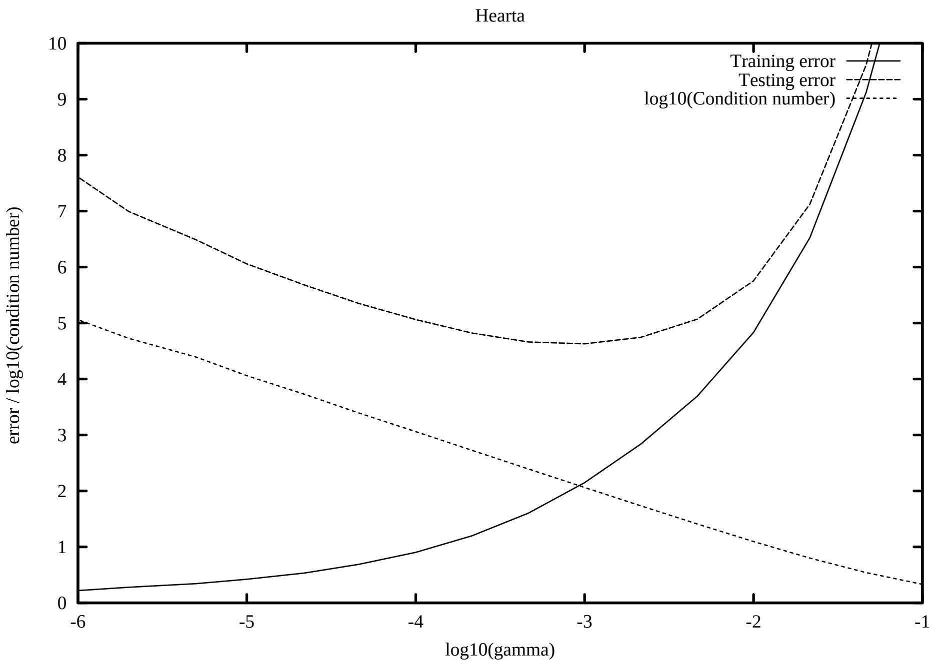


Fig. 3. Dependancy of errors(on training and testing data sets) and condition number of the linear system 6 on regularization parameter γ.

task name	inputs	outpus	training set	testing set	type
cancer	9	2	525	174	class
card	51	2	518	172	class
flare	24	3	800	266	approx
glass	9	6	161	53	class
heartac	35	1	228	75	approx
hearta	35	1	690	230	approx
heartc	35	2	228	75	class
heart	35	2	690	230	class
horse	58	3	273	91	class
soybean	82	19	513	170	class

Table 1. Overview of Proben1 tasks. Number of inputs, number of outputs, number of samples in training and testing sets.

	E_{train}	E_{test}	γ	b	time
cancer1	2.29	1.76	0.2690×10^{-3}	1.63	4:5:49
cancer2	1.82	3.01	0.2642×10^{-3}	1.46	3:30:13
cancer3	2.12	2.80	0.4958×10^{-3}	1.58	4:22:27
card1	8.80	10.00	1.5963×10^{-3}	4.46	3:36:37
card2	7.63	12.53	1.2864×10^{-3}	4.31	3:8:30
card3	6.58	12.32	0.3078×10^{-3}	4.43	4:10:19
diabetes1	13.94	16.04	1.4590×10^{-3}	1.00	5:29:3
diabetes2	13.85	16.81	1.9810×10^{-3}	0.97	5:24:10
diabetes3	13.75	15.93	0.2943×10^{-3}	1.42	4:42:47
flare1	0.36	0.54	3.6517×10^{-3}	5.70	6:19:53
flare2	0.43	0.27	3.6517×10^{-3}	4.07	7:26:6
flare3	0.41	0.34	2.5483×10^{-3}	4.85	9:2:17
glass1	3.26	6.95	2.4472×10^{-3}	0.30	0:31:18
glass2	4.26	7.91	2.1480×10^{-3}	0.51	0:24:30
glass3	4.06	7.33	2.3607×10^{-3}	0.42	0:26:42
heartac1	4.19	2.78	1.6144×10^{-3}	6.51	1:12:13
heartac2	3.47	3.86	0.8467×10^{-3}	6.00	0:56:2
heartac3	3.32	5.01	1.0413×10^{-3}	6.50	0:55:14
hearta1	3.49	4.40	0.2618×10^{-3}	5.74	7:30:12
hearta2	3.59	4.05	0.2996×10^{-3}	5.72	8:43:32
hearta3	3.47	4.43	0.3398×10^{-3}	5.48	6:11:4
heartc1	9.90	16.02	1.9832×10^{-3}	6.51	1:31:35
heartc2	12.48	6.10	1.1665×10^{-3}	6.51	1:29:34
heartc3	8.88	12.66	1.9810×10^{-3}	3.37	0:47:22
heart1	9.57	13.65	1.5679×10^{-3}	2.89	6:37:13
heart2	9.37	13.80	1.3824×10^{-3}	3.09	7:30:9
heart3	9.27	15.99	0.9647×10^{-3}	3.90	7:29:45
horse1	7.55	11.90	3.7855×10^{-3}	3.40	1:10:59
horse2	7.84	15.18	3.7855×10^{-3}	3.87	1:6:2
horse3	4.81	13.58	2.4144×10^{-3}	2.94	1:27:51
soybean1	0.12	0.66	0.1075×10^{-3}	3.04	3:18:12
soybean2	0.23	0.49	0.1433×10^{-3}	3.60	3:17:22
soybean3	0.24	0.58	0.1334×10^{-3}	3.88	2:4:27

Table 2. Overview of results obtained by Regularization Network. Error on the training set E_{train}, error on the testing set E_{test}, winning regularization parameter γ, winning width b and time needed for the computation.

	# units	E_{train}		E_{test}		average time
		mean	std	mean	std	
cancer1	15	1.85	0.85	1.69	0.72	0:49:18
cancer2	15	1.91	0.26	3.12	0.07	1:1:3
cancer3	15	1.66	0.36	3.19	0.13	0:58:8
card1	10	8.12	0.75365	10.16	0.56799	0:23:23
card2	10	8.05	0.10627	12.81	0.01129	0:2:6
card3	10	6.77	0.09258	12.09	0.00857	0:55:12
flare1	10	0.37	0.01051	0.37	0.00011	1:12:33
flare2	10	0.41	0.00775	0.31	0.00006	0:39:3
flare3	10	0.37	0.00816	0.38	0.00007	0:51:34
glass1	20	5.10	0.14506	6.76	0.02104	0:4:31
glass2	20	4.93	0.06963	7.96	0.00485	0:4:51
glass3	20	5.80	0.98584	8.06	0.97188	0:3:24
heartac1	10	2.26	0.28085	3.69	0.07888	0:28:27
heartac2	10	1.78	0.19411	4.98	0.03768	0:28:20
heartac3	10	1.66	0.06073	5.81	0.00369	0:29:31
hearta1	15	3.08	0.08863	4.36	0.00786	0:25:12
hearta2	10	3.36	0.07981	4.05	0.00637	0:20:41
hearta3	10	3.19	0.04009	4.29	0.00161	0:36:2
heartc1	10	6.07	0.25620	16.17	0.06564	0:12:24
heartc2	10	7.99	0.19760	6.49	0.03905	0:21:34
heartc3	10	7.13	0.60961	14.35	0.37163	0:3:57
heart1	10	9.96	0.39903	14.05	0.15923	0:20:45
heart2	20	6.36	5.87046	11.67	34.46230	0:35:8
heart3	15	6.95	6.04203	12.02	36.50611	0:27:46
horse1	10	10.57	0.21522	11.96	0.04632	0:10:51
horse2	10	10.04	0.31862	16.80	0.10152	0:12:19
horse3	10	9.88	0.26721	14.56	0.07140	0:14:16
soybean1	30	0.28	0.06739	0.73	0.00454	0:48:32
soybean2	30	0.15	0.30384	0.24	0.09232	0:20:23
soybean3	30	0.31	0.09	0.72	0.01	0:40:52

Table 3. Overview of results obtained by RBF network.

	correlation with width
min	0.158
max	0.421
mean	0.552
3 nearest neibourhgs	0.357
5 nearest neibourhgs	0.360
10 nearest neibourhs	0.290

Table 4. Correlation coeficients: correlation between width found by crossvalidation (for RN) and minimal, maximal and mean distance between two data points, mean distance of 3, 5, and 10 nearest neibourghs of each data point. Computed over Proben1 taks.

	RN		RBF			MLP		
	E_{test}	# units	mean E_{test}	std	# units	mean E_{test}	std	architecture
cancer1	1.76	525	1.69	0.072	15	1.60	0.41	4+2
cancer2	3.01	525	3.12	0.07	15	3.40	0.33	8+4
cancer3	2.80	525	3.19	0.13	15	2.57	0.24	16+8
card1	10.00	518	10.16	0.567	10	10.53	0.57	32+0
card2	12.53	518	12.81	0.011	10	15.47	0.75	24+0
card3	12.32	518	12.09	0.008	10	13.03	0.50	16+8
flare1	0.54	800	0.37	0.00011	10	0.74	0.80	32+0
flare2	0.27	800	0.31	0.00006	10	0.41	0.47	32+0
flare3	0.34	800	0.38	0.00007	10	0.37	0.01	24+0
glass1	6.95	161	6.76	0.02104	20	9.75	0.41	16+8
glass2	7.91	161	7.96	0.00485	20	10.27	0.40	16+8
glass3	7.33	161	8.06	0.97188	20	10.91	0.48	16+8
heartac1	2.78	228	3.69	0.07888	10	2.82	0.22	2+0
heartac2	3.86	228	4.98	0.03768	10	4.54	0.87	8+4
heartac3	5.01	228	5.81	0.00369	10	5.37	0.56	16+8
hearta1	4.40	690	4.36	0.00786	15	4.76	1.14	32+0
hearta2	4.05	690	4.05	0.00637	10	4.52	1.10	16+0
hearta3	4.43	690	4.29	0.00161	10	4.81	0.87	32+0
heartc1	16.02	228	16.17	0.06564	10	17.18	0.79	16+8
heartc2	6.10	228	6.49	0.03905	10	6.47	2.86	8+8
heartc3	12.66	228	14.35	0.37163	10	14.57	2.82	32+0
heart1	13.65	690	14.05	0.15923	10	14.33	1.26	32+0
heart2	13.80	690	11.67	34.46230	20	14.43	3.29	32+0
heart3	15.99	690	12.02	36.50611	15	16.58	0.39	32+0
horse1	11.90	273	11.96	0.04632	10	13.95	0.60	16+8
horse2	15.18	273	16.80	0.10152	10	18.99	1.21	16+8
horse3	13.58	273	14.56	0.07140	10	17.79	2.45	32+0
soybean1	0.66	513	0.73	0.00454	30	1.03	0.05	16+8
soybean2	0.49	513	0.24	0.09232	30	0.90	0.08	32+0
soybean3	0.58	513	0.72	0.01	30	1.05	0.09	16+0

Table 5. Comparision of E_{test} of RN, RBF and MLP.

	RN		RBF	
	E_{train}	E_{test}	E_{train}	E_{test}
ploucnice1	0.057	0.048	0.059	0.049
ploucnice1s	0.0257	0.0891	0.061	0.051
ploucnice2	0.062	0.182	0.088	0.062
ploucnice2s	0.0611	0.167	0.099	0.092

Table 6. Results of RN and RBF on Ploucnice data sets.

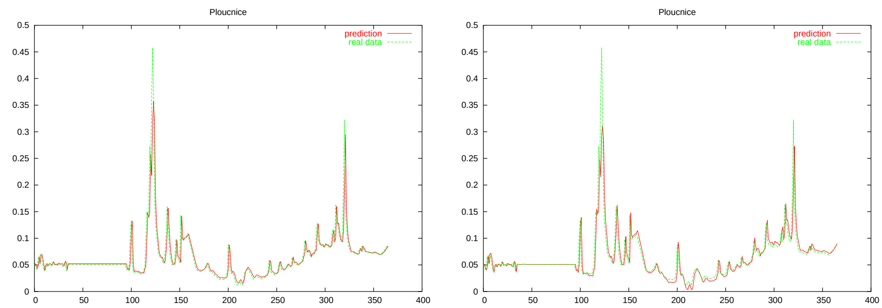


Fig. 4. Prediction of flow rate by a) RN b) RBF

2. Tikhonov, A., Arsenin, V.: Solutions of Ill-posed Problems. W.H. Winston, Washington, D.C (1977)
3. Girosi, F., Jones, M., Poggio, T.: Regularization theory and Neural Networks architectures. Neural Computation **7,No.2** (1995) 219–269
4. Poggio, T., Smale, S.: The mathematics of learning: Dealing with data. Notices of the AMS **50, No.5** (2003) 537–544
5. Powel, M.: Radial basis functions for multivariable interpolation: A review. In: IMA Conference on Algorithms for the Approximation of Functions and Data, RMCS, Shrivenham, England (1985) 143–167
6. Neruda, R., Kudová, P.: Hybrid learning of RBF networks. Neural Networks World **12** (2002) 573–585
7. Neruda, R., Kudová, P.: Learning methods for RBF neural networks. Future Generations of Computer Systems (2004) In press.