Controlling the Structure of Neural Networks that Grow and Shrink

Norbert Jankowski

Department of Computer Methods
Nicholas Copernicus University

ul. Grudziądzka 5, 87–100 Toruń, Poland
phone: +48 56 6113307 fax: +48 56 621543
e-mail: Norbert.Jankowski@phys.uni.torun.pl
http://www.phys.uni.torun.pl/~norbert

The Goal

The main goal is to build a network/system which would be able to preserve information in as complex net as the incoming data.

Methods of complexity control

Merging: Checking an overlap integral it is possible to answer whether or not (two) neurons can be replaced (merged) by another neuron.

Flexible Transfer Functions: It is important to use flexible transfer functions which can estimate more complex density of data using small number of parameters.

Growing: If the **novelty criterion** is satisfied then a new neuron is added to the hidden layer.

Pruning: Algorithm checks whether or not a neuron should be removed. If yes, then the neuron with the smallest saliency is removed.

Learning: Using an efficient learning algorithm.

Neurons Merging for RBF-like networks

Many pruning methods were described in the last decade. Put pruning leads to the removal of the network connections and unnecessary neurons. Frequently many neurons contribute to decision borders that could be represented by smaller network without decreasing of accuracy. Therefore one should merge two (or even more – it may be more complicated computationally) neurons keeping the current shape of the decision surface unchanged as much as possible.

Two neurons can be replaced by another one if the ratio:

$$\frac{\int_{\mathbf{d}\subseteq\mathcal{D}^n} |\phi_i(\mathbf{x}) + \phi_j(\mathbf{x}) - \phi_{new}(\mathbf{x})| \quad \mathrm{d}\mathbf{x}}{\int_{\mathbf{d}\subseteq\mathcal{D}^n} |\phi_i(\mathbf{x}) + \phi_j(\mathbf{x})| \quad \mathrm{d}\mathbf{x}} < \alpha \tag{1}$$

is smaller than some confidence parameter α. Here d is the subspace in which localized neuron transfer functions (scaled by the networks weights) $\phi_i(\mathbf{x})$ and $\phi_j(\mathbf{x})$ have values greater than a small threshold and $\phi_{new}(\mathbf{x})$ is the new neuron whose transfer function replaces the combination of neurons i and j.

The equation **??** is hard to compute in general case, but if the transfer functions used in hidden layer are dimensionally separable (for example gaussian or bi-radial function) then we can check another inequality

$$\frac{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left(\phi_i(\mathbf{x}) + \phi_j(\mathbf{x}) - \phi_{new}(\mathbf{x})\right)^2 \quad \mathrm{d}\mathbf{x}}{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left(\phi_i(\mathbf{x}) + \phi_j(\mathbf{x})\right)^2 \quad \mathrm{d}\mathbf{x}} < \alpha \tag{2}$$

which can be computed analitycally or numerically.

In other cases the above criterion can be simplified through sampling the space around neurons i and j (using adequate distribution function for the density of neurons i and j) and computing **weighted** mean squared error for a given number of points:

$$\frac{\sum_{d\in\mathbf{d}} \left(\phi_i(\mathbf{x}) + \phi_j(\mathbf{x}) - \phi_{new}(\mathbf{x})\right)^2}{\sum_{d\in\mathbf{d}} \left(\phi_i(\mathbf{x}) + \phi_j(\mathbf{x})\right)^2} < \alpha \tag{3}$$

For the bi-radial transfer function parameters of new neuron can be calculated as below

$$\begin{aligned}
\mathbf{w}_{new} &= (\mathbf{w}_i \cdot \phi(\mathbf{t}_i, \mathbf{t}_i) \cdot \bar{\mathrm{P}}_i + \mathbf{w}_j \cdot \phi(\mathbf{t}_j, \mathbf{t}_j) \cdot \bar{\mathrm{P}}_j)/\phi(\mathbf{t}_{new}, \mathbf{t}_{new}) && (4)\\
\mathbf{t}_{new,k} &= \frac{1}{\mathbf{M}} \int_{d\in\mathbf{D}} x_k(\phi(\mathbf{x}, \mathbf{t}_i) + \phi(\mathbf{x}, \mathbf{t}_j)) \quad \mathrm{d}\mathbf{x} && (5)\\
\mathbf{t}_{new} &= \mathbf{t}_i \cdot \bar{\mathrm{P}}_i + \mathbf{t}_j \cdot \bar{\mathrm{P}}_j && (6)\\
\mathbf{s}_{new} &= \mathbf{s}_i \cdot \bar{\mathrm{P}}_i + \mathbf{s}_j \cdot \bar{\mathrm{P}}_j && (7)\\
\mathbf{b}_{new} &= \mathbf{b}_i \qquad \text{if neuron } j \text{ in } i && (8)\\
&= \mathbf{b}_j \qquad \text{if neuron } i \text{ in } j && (9)\\
&= (\mathbf{b}_i + \mathbf{b}_j + abs(\mathbf{t}_i - \mathbf{t}_j))/2 && (10)
\end{aligned}$$

where
$\mathbf{M} = \int_{d\in\mathbf{D}} (\phi(\mathbf{x}, \mathbf{t}_i) + \phi(\mathbf{x}, \mathbf{t}_j))\ \mathrm{d}\mathbf{x}$,
$\bar{\mathrm{P}}_i = \mathrm{P}_i/(\mathrm{P}_i + \mathrm{P}_j)$ and $\bar{\mathrm{P}_j} = \mathrm{P}_j/(\mathrm{P}_i + \mathrm{P}_j)$,
P_i and P_j can be defined as

$$\mathrm{P}_i = \int_{d\in\mathbf{D}} \phi(\mathbf{x}, \mathbf{t}_i) \quad \mathrm{d}\mathbf{x} \qquad \mathrm{P}_j = \int_{d\in\mathbf{D}} \phi(\mathbf{x}, \mathbf{t}_j) \quad \mathrm{d}\mathbf{x}$$

or

$$\mathrm{P}_i = \phi(\mathbf{x}, \mathbf{t}_i) \cdot \mathbf{w}_i \cdot \prod_k \mathbf{b}_k \qquad \mathrm{P}_j = \phi(\mathbf{x}, \mathbf{t}_j) \cdot \mathbf{w}_j \cdot \prod_k \mathbf{b}_k$$

Bi-radial Transfer Functions

$$Bi(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(e^{s_i}\cdot(x_i - t_i + e^{b_i}))(1-\sigma(e^{s_i}\cdot(x_i - t_i - e^{b_i}))) \qquad (11)$$

where $\sigma(x) = 1/(1+e^{-x})$.

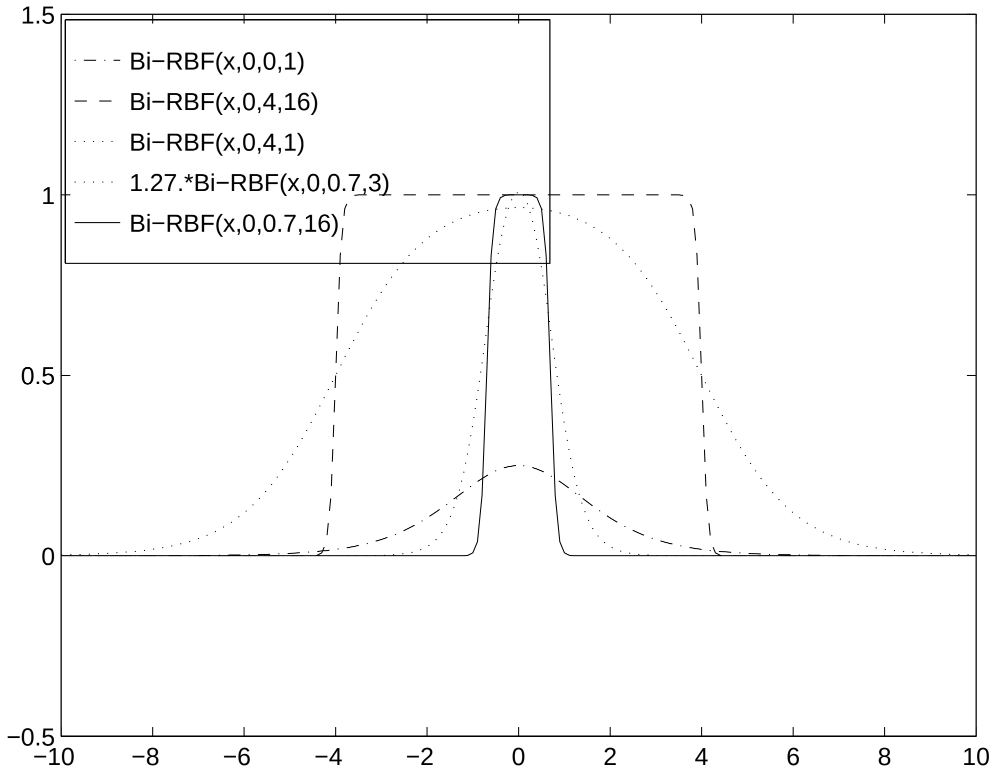


Figure 1: A few shapes of the bi-radial functions in two dimensions.

Rotation of Densities in Transfer Functions

Next step towards even greater flexibility requires individual rotation of densities provided by each unit. Of course one can introduce a rotation matrix operating on the inputs $\mathbf{Rx}$, but in practice it is very hard to parametrize this $N \times N$ matrix with $N-1$ independent angles (for example, Euler's angles) and calculate the derivatives necessary for the backpropagation procedure. We have found two ways to obtain rotated densities in all dimensions using transfer functions with just N additional parameters per neuron. In the first approach product form of the combination of sigmoids is used

$$\begin{aligned} C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{R}) &= \prod_i \Big(\sigma(\mathbf{R}_i\mathbf{x}+t_i) - \sigma(\mathbf{R}_i\mathbf{x}+t'_i)\Big) \quad (12) \\ SC_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{p},\mathbf{r},\mathbf{R}) &= \prod_i \Big(p_i \cdot \sigma(\mathbf{R}_i\mathbf{x}+t_i) + r_i \cdot \sigma(\mathbf{R}_i\mathbf{x}+t'_i)\Big) \end{aligned}$$

where $\mathbf{R}_i$ is the i-th row of the rotation matrix $\mathbf{R}$ with the following structure:

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0 \\ 0 & s_2 & \alpha_2 & 0 & & \\ & & & & & \\ \vdots & & & \ddots & & \vdots \\ & & & & s_{N-1} & \alpha_{N-1} \\ 0 & \cdots & & & 0 & s_N \end{bmatrix} \quad (13)$$

If $p_i = 1$ and $r_i = -1$ then SC_P function is localized and gives similar densities as the biradial functions (except for rotation). Choosing other values for the p_i and r_i parameters non-local transfer functions are created.

In the second approach the density is created by the sum of a "window-type" combinations of sigmoids $L(x; t, t') = \sigma(x + t) - \sigma(x + t')$ in $N - 1$ dimensions and a combination rotated by a vector $\mathbf{K}$:

$$C_K(\mathbf{x}; \mathbf{t}, \mathbf{t}', \mathbf{W}, \mathbf{K}) = \sum_{i=1}^{N-1} W_i L(x_i, t_i, t'_i) + W_N L(\mathbf{K}\mathbf{x}, t, t') \qquad (14)$$

The last density is perpendicular to the $\mathbf{K}$ vector. Treating $C_K(\cdot)$ as the activation function and using sigmoidal output function with a proper threshold leaves only the densities in the direction perpendicular to $\mathbf{K}$. An alternative is to use the product form:

$$C_{PK}(\mathbf{x}; \mathbf{t}, \mathbf{t}', \mathbf{K}) = L(\mathbf{K}\mathbf{x}, t, t') \prod_{i=1}^{N-1} L(x_i, t_i, t'_i) \qquad (15)$$

as the transfer function – the output sigmoid is not needed in this case. Rotation adds only $N - 1$ parameters for $C_P(\cdot)$ function and N parameters for $C_K(\cdot)$ function.

There is an obvious tradeoff between the flexibility of the processing units increasing with the number of adjustable parameters and the complexity of the training process of the whole network. Biradial and rotated transfer functions ($C_P(\cdot)$, $C_S(\cdot)$) are flexible but still rather simple, therefore we intend to use them also in the FSM and other networks.

The typical goal of ANN used for approximation or classification

the mapping between the input and output[a] space for given data sets $\mathcal{S} = \{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_n, y_n \rangle\}$, where $\langle \mathbf{x}_i, y_i \rangle$ is input – output pair ($\mathbf{x}_i \in \mathcal{R}^N, \quad y_i \in \mathcal{R}$). The underlying mapping $F(\cdot)$ we can write as below

$$F(\mathbf{x}_i) = y_i + \eta, \qquad i = 1, \ldots, n \tag{16}$$

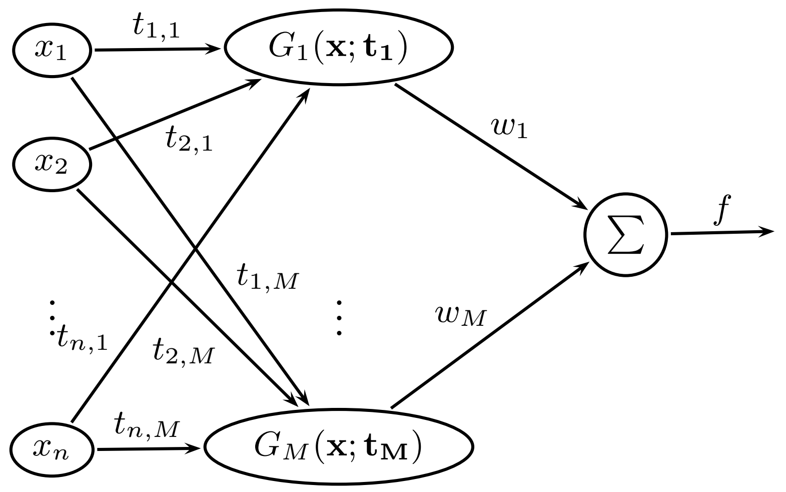
η is a zero mean white noise with variance σ_{ns}^2.

Radial Basis Function Networks

- designed as a solution to a curve–fitting (approximation) problem over a multi–dimensional space — the surface reconstruction
- typical form of the RBF network can be written as

$$f(\mathbf{x}; \mathbf{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i). \tag{17}$$

[a]Only supervised networks are considered.



- typical RBFs

$$h_1(\mathbf{x};\mathbf{t}) = ||\mathbf{x}-\mathbf{t}|| \tag{18}$$

$$h_2(\mathbf{x};\mathbf{t},b) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0 \tag{19}$$

$$h_3(\mathbf{x};\mathbf{t},b) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1 \tag{20}$$

$$h_4(\mathbf{x};\mathbf{t},b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2} \tag{21}$$

$$h_5(\mathbf{x};\mathbf{t},b) = (b||\mathbf{x}-\mathbf{t}||)^2\ln(b||\mathbf{x}-\mathbf{t}||) \tag{22}$$

Resource–Allocating Network

- is able to grow (to add new neurons to the hidden layer)
- if the growth criterion given below is satisfied then a new neuron is added.

$$\begin{aligned} \mathbf{y}_n - f(\mathbf{x}_n) = e_n &> e_{min} \quad (23) \\ ||\mathbf{x}_n - \mathbf{t}_c|| &> \epsilon_{min} \quad (24) \end{aligned}$$

- newly added k-th neuron has weight w_k (see Eq. **??**) equal to $e_n = \mathbf{y}_n - f(\mathbf{x}_n)$, the center of basis function is positioned at $\mathbf{x}_n$. Thus, now the network can be described by

$$f^{(n)}(\mathbf{x}, \mathbf{p}) = \sum_{i=1}^{k-1} w_i G_i(\mathbf{x}, \mathbf{p}_i) + e_n G_k(\mathbf{x}, \mathbf{p}_k) = \sum_{i=1}^{k} w_i G_i(\mathbf{x}, \mathbf{p}_i) \quad (25)$$

  where $\mathbf{p}_k$ consists of the center – $\mathbf{x}_n$ – and others parameters which are set up with some initial values.

- smoothness constraint must be imposed on function $G_n(\cdot)$ which has the following property: $G_k(\mathbf{x}_n) = 1$ and $G_k(\mathbf{x}_n + \mathbf{a}) = 0$ for any $||\mathbf{a}|| \neq 0$

Extended Kalman Filter

$$\begin{tikzcd} y(t) \arrow[r] & \boxed{\text{State estimator}} \arrow[r] & \hat{p}(t|t) \end{tikzcd}$$

$$\begin{tikzcd} y(t) \arrow[r] & \boxed{\text{Measurement filter}} \arrow[r] & \hat{y}(t|t) \end{tikzcd}$$

$$\begin{tikzcd} y(t) \arrow[r] & \boxed{\text{Whitening filter}} \arrow[r] & e(t|t) \end{tikzcd}$$

The EKF equations can be written as below:

$$\begin{aligned}
e_n &= y_n - f(\mathbf{x}_n;\mathbf{p}_{n-1}) \\
\mathbf{d}_n &= \tfrac{\partial f(\mathbf{x}_n;\mathbf{p}_{n-1})}{\partial \mathbf{p}_{n-1}} \\
R_y &= R_n + \mathbf{d}_n^T \mathbf{P}_{n-1} \mathbf{d}_n \\
\mathbf{k}_n &= \mathbf{P}_{n-1}\mathbf{d}_n / R_y \\
\mathbf{p}_n &= \mathbf{p}_{n-1} + e_n \mathbf{k}_n \\
\mathbf{P}_n &= [\mathbf{I} - \mathbf{k}_n \mathbf{d}_n^T]\mathbf{P}_{n-1} + Q_0(n)\mathbf{I}
\end{aligned} \tag{26}$$

Fast EKF

we can simplify the matrix $\mathbf{P}_n$ using $\widetilde{\mathbf{P}}_n$ which consists of a chain of matrices $\widetilde{\mathbf{P}}_n^k$ on diagonal

$$\widetilde{\mathbf{P}}_n = \begin{bmatrix} \widetilde{\mathbf{P}}_n^1 & 0 & \cdots & 0 \\ 0 & \widetilde{\mathbf{P}}_n^2 & \cdots & 0 \\ \vdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & \widetilde{\mathbf{P}}_n^M \end{bmatrix} \tag{27}$$

$\mathbf{P}_n$	$m \cdot M \times m \cdot M$	$O(M^2)$
$\widetilde{\mathbf{P}}_n$	$m^2 M$	$O(M)$

m is constant in $\mathcal{P}$

Novelty Criterion

- the variance of network output — a measure of uncertainty:

$$\sigma_y^2(\mathbf{x}) = \mathrm{Var}[f(\mathbf{x};\mathbf{p})]$$

- null hyphotesis for the statistical inference of model sufficiency is stated as follows:

$$\mathcal{H}_0 : \quad \frac{e^2}{\mathrm{Var}[f(\mathbf{x};\mathbf{p}) + \eta]} = \frac{e^2}{\sigma_y^2(\mathbf{x}) + \sigma_{ns}^2} < \chi^2_{n,\theta} \tag{28}$$

where $\chi^2_{n,\theta}$ is θ% confidence on χ^2 distribution for n degree of fredom. e is error given by $y - f(\mathbf{x};\mathbf{p})$ (see Eq. **??**).

- As we use the EKF algorithm R_y estimates the total uncertainty in the expected output:

$$R_y = \mathrm{Var}[f(\mathbf{x};\mathbf{p}) + \eta], \tag{29}$$

- and the null hypothesis can be written as follows:

$$\mathcal{H}_0 : \quad \frac{e_n^2}{R_y} < \chi^2_{n,\theta} \tag{30}$$

- new neuron $(M+1)$-th. Example for Gaussian function $G_{M+1}(\cdot)$:

$$w_{M+1} := e_n, \quad \mathbf{t}_{M+1} := \mathbf{x}_n \quad b_{M+1} := b_0, \quad \mathbf{P}_n := \begin{bmatrix} \mathbf{P}_n & 0 \\ 0 & P_0\mathbf{I} \end{bmatrix},$$

Pruning

Let´s sort the vector $\mathbf{p}_n$ as follows

$$\mathbf{p}_n = [w_1, \ldots, w_M, \ldots]^T$$

then the covariance matrix $\mathbf{P}_n$ look like:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_w & \mathbf{P}_{wv} \\ \mathbf{P}_{wv}^T & \mathbf{P}_v \end{bmatrix} \tag{31}$$

It is possible to define pruning method for the IncNet network by checking for each neuron:

$$L_1 = \min_i w_i^2 / [\mathbf{P}_w]_{ii} \tag{32}$$

We have to prune a neuron i corresponding to L_1.

By checking the inequality below we can decide whether to prune or not to prune:

$$\frac{L_1}{R_y} < \chi^2_{1,\vartheta} \tag{33}$$

where $\chi^2_{n,\vartheta}$ is ϑ% confidence on χ^2 distribution for 1 degree of fredom.

Results

Hermite function approximation.

$$f_{her}(x) = 1.1(1 - x + 2x^2)\exp(-1/2x^2) \tag{34}$$

The training data consists of 40 random points in $[-4, 4]$ interval, and testing data consists of 100 uniformly distributed points from the same interval. All nets was running over 800 iterations (not epoch).

RMSE errors			
IncNet Pro	IncNet S.G.	RAN-EKF	RAN
0.015	0.054	0.09	0.15

By IncNet S.G. we mean IncNet with Gaussian nodes (the center and bias is adapted); RAN-EKF is a RAN net with EKF as learning algorithm.

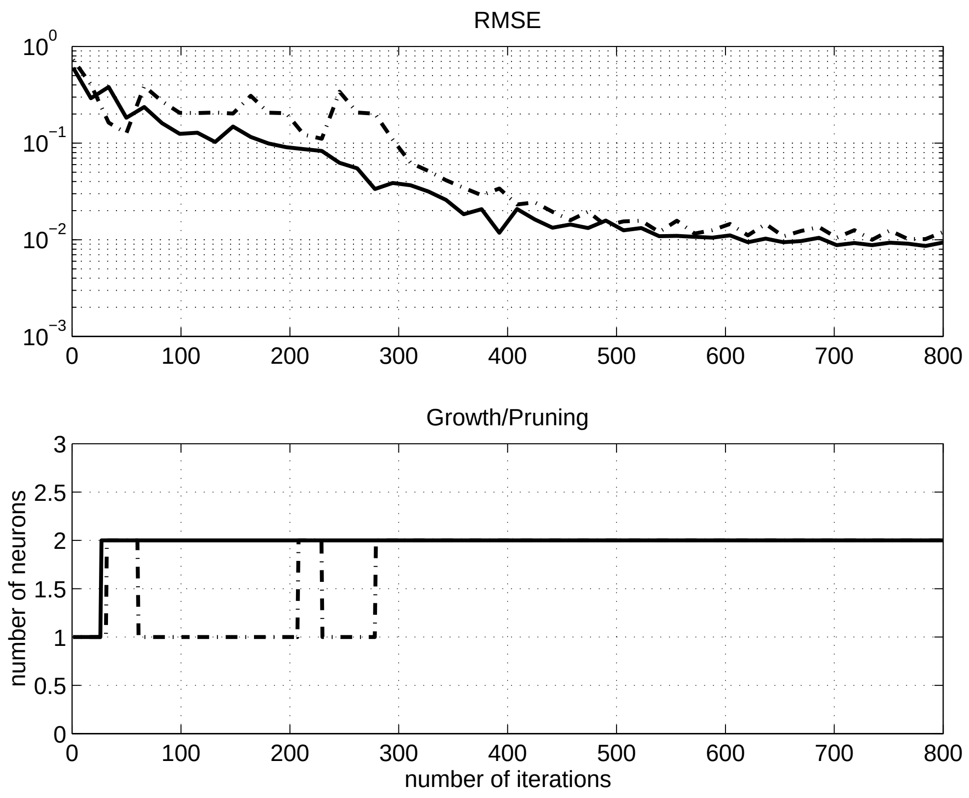


Figure 2: Comparison of IncNet with Gaussian (dashdot line) and Bi-radial (solid line) nodes on 2D approximation problem of Hermite function. In all methods pruning is on.

Sugeno function approximation.

$$f_{sug}(x,y,z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \tag{35}$$

216 training points was randomly choose from $[1, 6]$ interval for all variable, and 125 testing points from $[1.5, 5.5]$ interval.

Average percentage error (APE):

$$APE = \frac{1}{N} \sum_{i=1}^{N} \left| \frac{f(\mathbf{x}_i) - y_i}{y_i} \right| * 100\% \tag{36}$$

Model	APE_{TRS}	APE_{TES}
GMDS model Kongo	4.7	5.7
Fuzzy model 1 Sugeno	1.5	2.1
Fuzzy model 2 Sugeno	0.59	3.4
FNN Type 1 Horikawa	0.84	1.22
FNN Type 2 Horikawa	0.73	1.28
FNN Type 3 Horikawa	0.63	1.25
M - Delta model	0.72	0.74
Fuzzy INET	0.18	0.24
Fuzzy VINET	0.076	0.18
IncNet Pro	**0.1564**	**0.1565**

Final networks had 9 neurons in hidden layer, and the time of learning was about 5600 second[a] (40000 iterations). The learning process was presented in Fig. **??**.

[a]The computations were made on Pentium 150MHz.

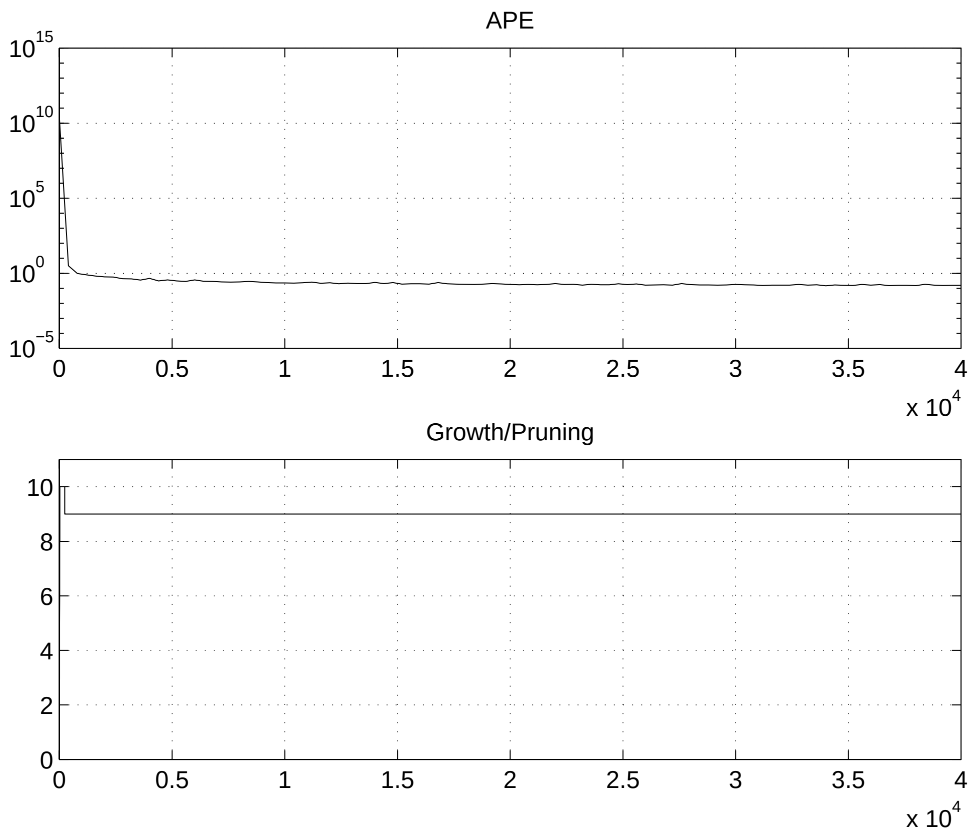


Figure 3: Approximation of Sugeno function.

Gabor and Girosi functions.

$$f_{gab}(x,y) = e^{-\|\mathbf{x}\|^2}\cos(.75\pi(x+y)) \tag{37}$$

$$f_{gir}(x,y) = sin(2\pi x) + 4(y-0.5)^2) \tag{38}$$

For learning only 20 points were used from uniformly distributed interval $[-1,1]\times[-1,1]$ for Gabor function (Eq. **??**) and from $[0,1]\times[0,1]$ interval for additive function (Eq. **??**). 10,000 points were used in testing phase.

Model1	$\sum_{i=1}^{20} c_i[e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)} + e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1 = \sigma_2 = 0.5$
Model2	$\sum_{i=1}^{20} c_i[e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)} + e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1 = 10, \sigma_2 = 0.5$
Model3	$\sum_{i=1}^{20} c_i[e^{\frac{(x-x_i)^2}{\sigma}} + e^{-\frac{(y-y_i)^2}{\sigma}}]$	$\sigma = 0.5$
Model4	$\sum_{\alpha=1}^{7} b_\alpha e^{-\frac{(x-t_x^\alpha)^2}{\sigma}} + \sum_{\beta=1}^{7} c_\beta e^{-\frac{(y-t_y^\beta)^2}{\sigma}}$	$\sigma = 0.5$
Model5	$\sum_{\alpha=1}^{n} c_\alpha e^{-(\mathbf{W}_\alpha \cdot \mathbf{X} - t_\alpha)^2}$	
Model6	$\sum_{i=1}^{20} c_i[\sigma(x-x_i)+\sigma(y-y_i)]$	
Model7	$\sum_{\alpha=1}^{7} b_\alpha \sigma(x - t_x^\alpha) + \sum_{\beta=1}^{7} c_\beta \sigma(y - t_y^\beta)$	
Model8	$\sum_{\alpha=1}^{n} c_\alpha \sigma(\mathbf{W}_\alpha \cdot \mathbf{X} - t_\alpha)$	

Additive function — MSE train/test								
IncNet	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
.0013	.000036	.000067	.000001	.000001	.000170	.000001	.000003	.000743
.0079	.011717	.001598	.000007	.000009	.001422	.000015	.000020	.026699
Gabor function — MSE train/test								
.000000	.000000	.000000	.000000	.345423	.000001	.000000	.456822	.000044
0.0298	.003818	.344881	67.9523	1.22211	.033964	98.4198	1.39739	.191055

- IncNet model is not always the best one
- but it is good on average by adapting more flexibly

Note that for Gabor function IncNet used 5 neurons and for additive Girosi function used only 3 neurons.

On Fig. **??** we can see that in the first phase pruning is used quite often — unnecessary neurons are killed as soon as possible.

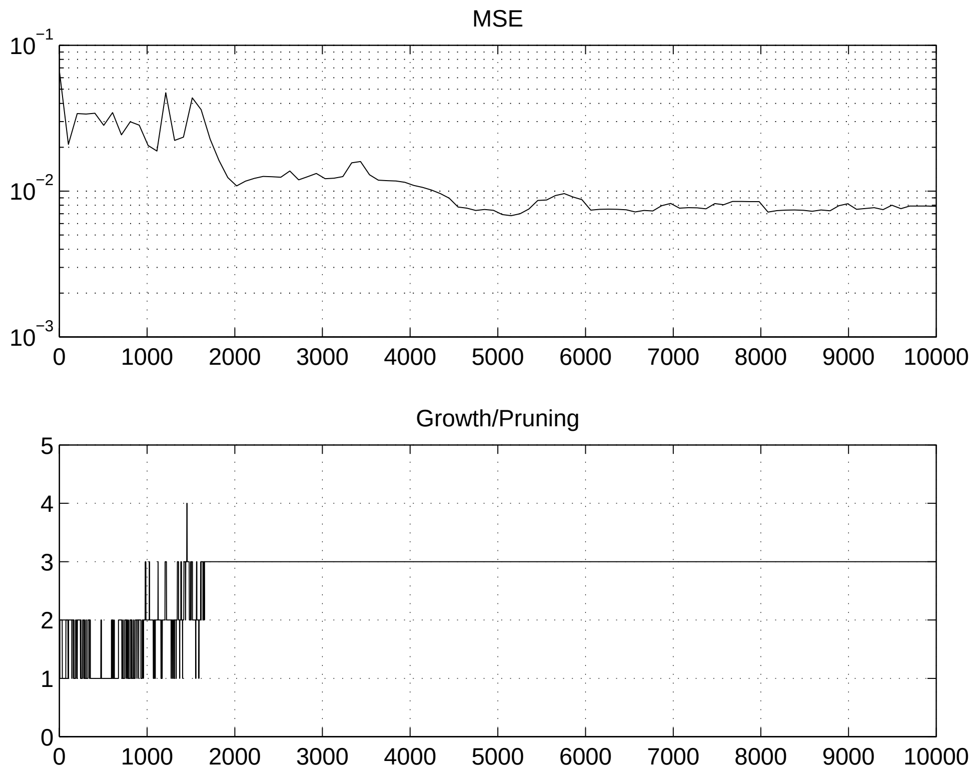
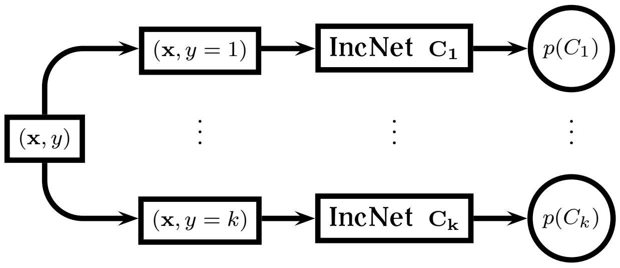


Figure 4: Hopeful pruning.

Conclusions

- Self growth and pruning — discovering of the complexity of underlying system
- Nearly all parameters are controlled automaticaly – EKF controls learning speed, some others parameters are (very) similar for different tasks
- In spite of incremental feature of algorithm, the *pruning time* is determined by theoretical criterion — not in random time moment or by checking the error on whole training data set
- Direct pruning saves the time of computations — in contrast to many other methods pruning AFTER learning
- good generalization, often $E_{TRS} \approx E_{TES}$ and not rarely $E_{TRS} \geq E_{TES}$!

Classification using IncNet Pro



k independed IncNet network are used for k-class problem. Each of them receives input vector $\mathbf{x}$ and 1 if index of i-th IncNet is equal to desired number of class, otherwise 0. The output of i-th IncNet Pro network is equal to probability that the vector belongs to i-th class. See figure on the right.

Classification of medical data

Breast Cancer, Hepatitis, Pima Indians Diabetes, Heart Disease — UCI Machine Learning Repository

Data set	ATTR	EX	E-TRS	E-TES	#N	Iter	Time (s)
B. Cancer	9 – D	699	97.7	97.1	49	3000	5150
Hepatitis	19 – D+C	155	98.6	82.3	97	500	3100
Diabetes	8 – C+D	768	77.2	77.6	100	5000	11200
Heart D.	13 – D+C	303	92.6	90.0	117	100	7400

method	Breast	Hepat.	Diab.	Heart
IncNet	**97.1**	**82.3**	**77.6**	**90.0**
BP	96.7	82.1	76.4	81.3
LVQ	96.6	83.2	75.8	82.9
CART	94.2	82.7	72.8	80.8
Fisher	96.8	84.5	76.5	84.2
LDA	96.0	86.4	77.2	84.5
QDA	34.5	85.8	59.5	75.4
KNN	96.6	85.3	71.9	81.5
LFC	94.4	81.9	75.8	75.1
ASI	95.6	82.0	76.6	74.4

Correctness