

# Constructive Algorithms for Feed-forward Neural Networks

Enrique Romero

Advanced Topics in Artificial Intelligence  
Master in Artificial Intelligence  
Soft Computing Group  
Departament de Llenguatges i Sistemes Informàtics  
Universitat Politècnica de Catalunya, Barcelona, Spain

- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

Several names:

- Constructive
- Sequential
- Incremental
- Growing

- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

# The Bias-Variance decomposition

The Bias-Variance decomposition [Geman et al., 1992]

$$\begin{aligned} E_D [ R(\mathcal{L}^2, f_D^o) ] = & E_X [ E_Y [ (y - E_Y [ y|x ]) ^2 ] ] + \\ & E_X [ ( E_D [ f_D^o(x) ] - E_Y [ y|x ]) ^2 ] + \\ & E_X [ E_D [ ( f_D^o(x) - E_D [ f_D^o(x) ] ) ^2 ] ]. \end{aligned}$$

- Model:  $f_D^o$  (output function trained with a data set  $D$ )
- $R(\mathcal{L}^2, f_D^o) = E_{X \times Y} [ (y - f_D^o(x))^2 ]$  (expected risk)
- First term: **independent of the model**
- Second term: **bias**, related to the approximation capability of the model: high approximation capability may have small bias
- Third term: **variance**, related to the complexity of the model: high complexity may have large variance
- **Bias and variance go in opposite directions**

# Different ways of controlling the complexity of FNNs

Different ways of controlling the complexity of FNNs:

- Weight constraints (limiting the size,...)
- Weight sharing
- Weight decay
- Early stopping
- Adding noise (to the weights, activities,...)
- Selecting the optimal number of hidden units:
  - Too few will lead to high bias
  - Too many will lead to large variance

# Motivation of constructive algorithms

- Selection of the minimum ( $\implies$  optimal?) number of hidden units for FNNs
- Low model complexity
- Computational cost (on-line applications)
- Minimum storage

- 1 Motivation
- 2 General Description**
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines



- Dynamically construct the network, starting from scratch, without setting *a priori* the architecture
- Start with a small network (usually with no hidden units)
- Sequentially add hidden units (usually one at every step) by selecting their hidden-layer and output-layer weights
- Stop when a satisfactory solution is found

---

## Algorithm

### repeat

Increase by 1 the number of hidden units  $N$

Obtain the hidden-layer weights of the new hidden unit

Compute the output-layer weights

Optionally, fix the new hidden unit in the network

**until** a certain stopping criterion is satisfied

### end Algorithm

---

- Over standard BP:
  - The number of hidden units is automatically selected
  - The proof of the universal approximation capability is “constructive” (but not very practical)
- Over pruning methods:
  - It is straightforward to specify an initial network
  - Pruning algorithms spend most of the time training networks larger than necessary

- Obtaining the hidden-layer weights
  - Set of candidates or search in a continuous space?
  - Hard non-linear problem
- Obtaining the output-layer weights
  - When?
  - Easy (many times linear) problem
- Stopping criterion
  - Approximation
  - Generalization

# A review of the literature

- A direct BP approach: Dynamic Node Creation [Ash, 1989]
- A sophisticated BP approach: Cascade-Correlation [Fahlman and Lebiere, 1990]
- Matching the residue:
  - Statistics: Projection Pursuit [Friedman and Stuetzle, 1981]
  - Signal Processing: Matching Pursuit [Mallat and Zhang, 1993]
  - Feed-forward Neural Networks:
    - Projection Pursuit Learning Network [Hwang et al., 1994]
    - Incremental Linear Quasi-Parallel Network [Kůrková and Beliczyński, 1995]
- Interacting hidden-layer weights:
  - Orthogonal Least Squares Learning [Chen et al., 1991]
  - Kernel Matching Pursuit [Vincent and Bengio, 2002]
- Universal approximation capability:
  - Projection Pursuit Regression: [Jones, 1987]
  - Other objective functions: [Kwok and Yeung, 1997]
  - Interacting hidden-layer weights: [Romero and Alquézar, 2006]

- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem**
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

# Formalization of the problem

- State space search problem
  - State space
  - Initial state
  - Evaluation criterion: approximation, generalization
  - Termination of the search
  - Search algorithm: forward selection
  - How to obtain the new state (how to obtain the hidden-layer weights of the new hidden unit)
- The new hidden-layer weights can be obtained:
  - Without keeping the previously selected ones fixed: Dynamic Node Creation
  - Keeping the previously selected hidden-layer weights fixed:
    - The new hidden-layer weights are selected to optimize a certain objective function, usually depending on the previous residue: Cascade-Correlation and Matching the residue
    - Orthogonalization: Interacting hidden-layer weights

# Formalization of the problem

- Given  $D = \{d_1, \dots, d_L\}$  with  $L$  patterns
- Objective: minimize the sum-of-squares error function  $\|Y - X_N\|^2$ 
  - $\|\cdot\|$  is the 2-norm
  - $Y = (y_1, \dots, y_L)$  is the target vector
  - $N$  is the number of hidden units
  - $X_0 = 0$
  - The output  $X_N$  is a vector in  $\mathbb{R}^L$  (output units are linear)

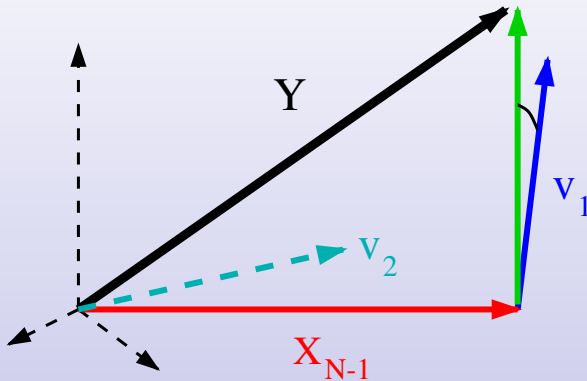
$$X_N = \sum_{k=1}^{N-1} \lambda_k^N v_{\omega_k} + \lambda_N^N v_{\omega_N}$$

- $v_{\omega_k} = (\varphi_k(\omega_k, d_1, b_k), \dots, \varphi_k(\omega_k, d_L, b_k))$  is the output vector of the  $k$ -th hidden unit (with hidden-layer weights  $\omega_k$ )
- $\lambda_k^N$  are the output-layer weights of that hidden unit at step  $N$

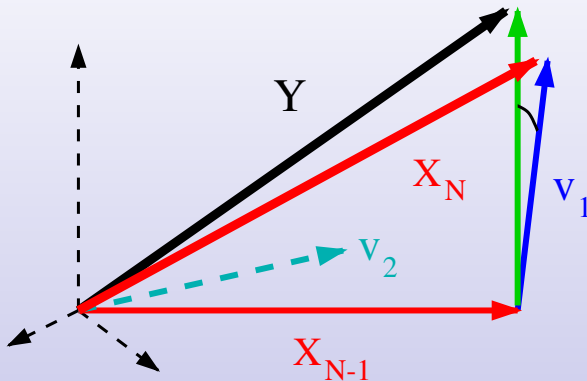


- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

# Matching the residue



# Matching the residue



# Matching the residue

- The new hidden-layer weights  $\omega_N$  are such that  $v_{\omega_N}$  matches the previous residue as best as possible:

$$(\omega_N, \lambda_N^N) = \arg \min_{(\omega, \lambda)} \|(Y - X_{N-1}) - \lambda v_{\omega}\|^2$$

- Given  $\omega_N$ , the (optimal) output-layer weight of the new hidden unit is

$$\lambda_N^N = \langle Y - X_{N-1}, v_{\omega_N} \rangle / \|v_{\omega_N}\|^2$$

- The difficult problem is to obtain  $\omega_N$

# Matching the residue

Proof:

$$\lambda_N^N = \arg \min_{\lambda} \|(Y - X_{N-1}) - \lambda v_{\omega_N}\|^2$$

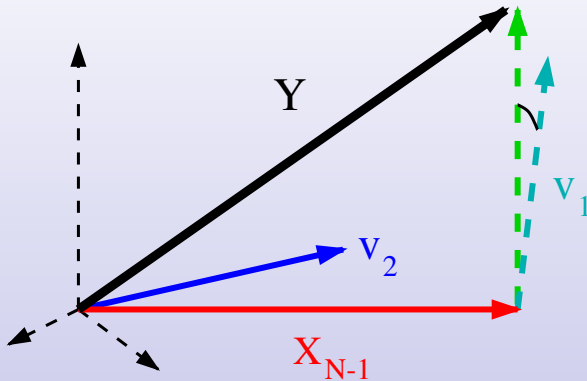
$$\frac{\partial}{\partial \lambda} \|(Y - X_{N-1}) - \lambda v_{\omega_N}\|^2 = 0 \implies \langle Y - X_{N-1} - \lambda v_{\omega_N}, v_{\omega_N} \rangle = 0$$

Therefore,

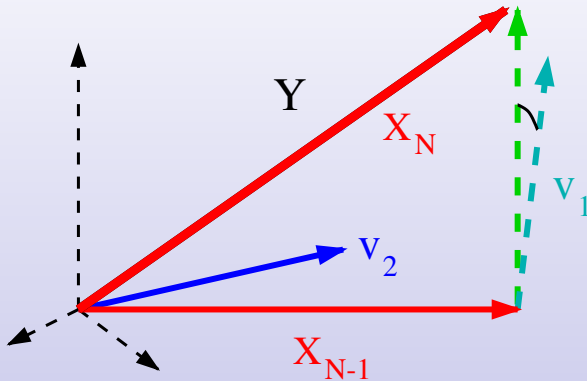
$$\lambda = \langle Y - X_{N-1}, v_{\omega_N} \rangle / \langle v_{\omega_N}, v_{\omega_N} \rangle$$

**Geometrical interpretation:** the new residue is orthogonal to the vector  $v_{\omega_N}$  of the approximation

# Interacting hidden-layer weights



# Interacting hidden-layer weights



# Interacting hidden-layer weights

- $\omega_N^1$  is considered better than  $\omega_N^2$  if  $v_{\omega_N^1}$  allows, **together with the previously selected hidden-layer weights** (and after computing the optimal output-layer weights of the whole network), a better approximation of  $Y$  than  $v_{\omega_N^2}$ . We look for

$$(\omega_N, \lambda_1^N, \dots, \lambda_N^N) = \arg \min_{(\omega, \lambda)} \left\| Y - \left( \sum_{k=1}^{N-1} \lambda_k v_{\omega_k} + \lambda_N v_{\omega} \right) \right\|^2$$

- Given  $\omega_N$ , the optimal output-layer weights of the whole network can be obtained by **solving a linear equations system**:

$$\mathbf{A}_N \cdot (\lambda_1^N, \dots, \lambda_{N-1}^N, \lambda_N^N)^t = (\langle Y, v_{\omega_1} \rangle, \dots, \langle Y, v_{\omega_{N-1}} \rangle, \langle Y, v_{\omega_N} \rangle)^t$$

where  $\mathbf{A}_N[i, j] = \langle v_{\omega_i}, v_{\omega_j} \rangle$  for  $1 \leq i, j \leq N$

- **The difficult problem is to obtain  $\omega_N$**



# Interacting hidden-layer weights

Proof:

$$(\lambda_1^N, \dots, \lambda_N^N) = \arg \min_{\lambda} \left\| Y - \sum_{k=1}^N \lambda_k v_{\omega_k} \right\|^2$$

$$\frac{\partial}{\partial \lambda_j} \left\| Y - \sum_{k=1}^N \lambda_k v_{\omega_k} \right\|^2 = 0 \implies \left\langle Y - \sum_{k=1}^N \lambda_k v_{\omega_k}, v_{\omega_j} \right\rangle = 0$$

Therefore,

$$\sum_{k=1}^N \lambda_k \langle v_{\omega_k}, v_{\omega_j} \rangle = \langle Y, v_{\omega_j} \rangle \quad \forall j \in \{1, \dots, N\}$$

**Geometrical interpretation:** the new residue is orthogonal to the hyperplane containing the vectors  $v_{\omega_j} \quad j = 1 \dots, N$

# Interacting hidden-layer weights

In matrix notation:

$$\lambda = (\lambda_1^N, \dots, \lambda_N^N)^t \in \mathbb{R}^N$$

$$Y, v_{\omega_1}, \dots, v_{\omega_N} \in \mathbb{R}^L$$

$$\mathbf{V} = (v_{\omega_1}, \dots, v_{\omega_N}) \in \mathbb{R}^L \times \mathbb{R}^N$$

$$\mathbf{A}_N = \mathbf{V}^t \mathbf{V}$$

The previous linear equations system is equivalent to

$$\mathbf{V}^t \mathbf{V} \lambda = \mathbf{V}^t Y$$

and the solution is

$$\lambda = (\mathbf{V}^t \mathbf{V})^{-1} \mathbf{V}^t Y = \mathbf{V}^\dagger Y$$

$\mathbf{V}^\dagger$  is the (Moore-Penrose) pseudo-inverse of the hidden-layer output matrix  $V$

# Interacting hidden-layer weights

There is a much more efficient way to compute the solution of this linear equation system, since the proposed system at step  $N$  is equal to the system at step  $N - 1$  ( $\mathbf{A}_{N-1} \cdot x = b$ ), but with a new row and a new column (bordered systems):

$$\left( \begin{array}{c|c} \mathbf{A}_{N-1} & w \\ \hline w^t & \gamma \end{array} \right) \cdot \left( \begin{array}{c} x \\ \alpha \end{array} \right) = \left( \begin{array}{c} b \\ \beta \end{array} \right)$$


The above system is equivalent to

$$\begin{cases} \mathbf{A}_{N-1} \cdot x + \alpha w = b \\ w^t \cdot x + \alpha \gamma = \beta \end{cases}$$

Therefore,  $x = \mathbf{A}_{N-1}^{-1} \cdot b - \alpha \mathbf{A}_{N-1}^{-1} \cdot w$ , and  $\alpha = \frac{\beta - w^t \cdot \mathbf{A}_{N-1}^{-1} \cdot b}{\gamma - w^t \cdot \mathbf{A}_{N-1}^{-1} \cdot w}$

Profits from the solution of the system at the previous step

# Advantages and disadvantages

- Approximation capability
- Computacional cost
- Although both models have been proved to have the convergence property, these theoretical results cannot be directly applied in practice: the optimization problem posed cannot be solved analytically in the general case
- A suboptimal solution is searched heuristically
  - Evolutionary algorithms [Romero, 2004]
  - Input data set [Romero, 2004], link with **Support Vector Machines** 
  - Randomly [Romero, 2004], [Huang et al., 2006b], [Romero and Alquézar, 2007], [Feng et al., 2009], link with **Extreme Learning Machines**

- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

# Comparison of constructive algorithms with Support Vector Machines

- Work described in [Romero and Toppo, 2007]
- Comparison of
  - Support Vector Sequential FNNs (SVSFNN), a constructive algorithm with
    - Interacting hidden-layer weights
    - Selection of hidden-layer weights: Input data set
    - Output-layer weights bounded by a constant  $B$
  - Standard 1-norm soft Support Vector Machines (SVM)
  - $\nu$ -Support Vector Machines ( $\nu$ -SVMs)
    - The  $\nu \in [0, 1]$  controls a trade-off between the margin errors and the number of support vectors
  - Support Vector Machines with Reduced Complexity (SVMRC) [Keerthi et al., 2006]
    - Forward selection of the support vectors (explicit control)

# Algorithm for SVSFNN

---

## Algorithm

### repeat

Increase by 1 the number of hidden units  $N$

**for every** example  $\omega$  in the training set

Assign  $\omega$  to the weights of the  $N$ th hidden unit

Compute the optimal output-layer weights

Set  $\omega_N := \omega$  if the output-layer weights are  
valid (their norm is bounded by a constant  $B$ )  
and the training error is minimized

### end for

Assign  $\omega_N$  to the weights of the  $N$ th hidden unit

**until** the maximum number of hidden units is added

Compute the optimal output-layer weights

### end Algorithm

---

- **Kernel and activation function:** Gaussian and 2-degree polynomial kernels
- **Parameters and model selection:**
  - The  $\gamma$  parameter of the kernel can be considered equivalent to the MLP-gain factor or RBF-width of the activation function
  - The  $C$  parameter for SVMs and SVMRC is similar to the  $B$  parameter of SVSFNN; For  $\nu$ -SVM,  $C$  is replaced by  $\nu$

A *grid search* was performed to get adequate  $\gamma$  and  $B/C/\nu$  parameters: The parameters related to the best 10-fold CV accuracy on the training set were kept to build the model

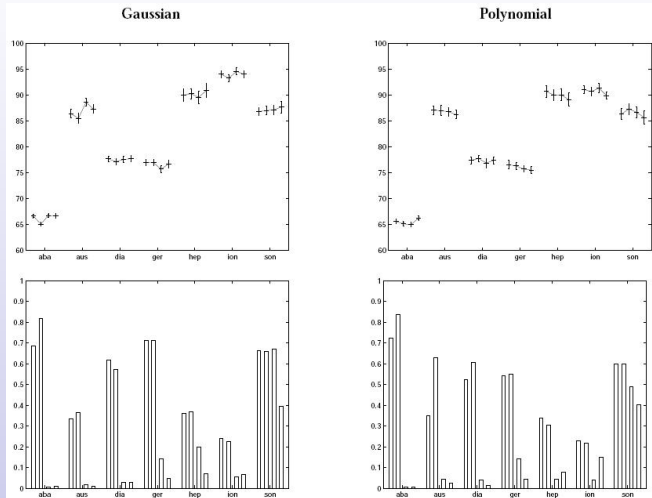
- **Model training and testing:** 30 training-test different random partitions (90%-10%) of the whole data set; **In all cases, the training and test data sets were exactly the same for all the methods**



<b>Data Set</b>	<b>#Var</b>	<b>#Cla</b>	<b>#Exa</b>
<i>Abalone</i>	8	3	4177
<i>Australian Credit</i>	15	2	690
<i>Pima Indians Diabetes</i>	8	2	768
<i>German Credit</i>	20	2	1000
<i>Hepatitis</i>	19	2	155
<i>Ionosphere</i>	33	2	351
<i>Sonar</i>	60	2	208

**Table:** Description of the data sets. Columns '#Var', '#Cla' and '#Exa': number of variables, classes and examples, respectively

# Results



**Figure:** Top: Error bars with mean accuracies and standard errors. Bottom: average ratio of support vectors/hidden units (wrt the number of training examples). Order: SVM,  $\nu$ -SVM, SVMRC, SVSFNN

- Accuracies obtained were very similar for all the models (specially for the Gaussian kernel, which yielded the best results)
- Regarding the support vectors / hidden units, SVSFNN obtained models with much less hidden units than the number of support vectors found by standard and  $\nu$ -SVMs, and similar to SVMRC
- In general, the computational training times were lower for SVMs than for SVSFNN

- 1 Motivation
- 2 General Description
- 3 Formalization of the Problem
- 4 Matching the Residue vs. Interacting Hidden-layer Weights
- 5 Comparison with Support Vector Machines
- 6 Comparison with Constructive Extreme Learning Machines

# Extreme Learning Machines

- [Huang et al., 2006a] proved that one hidden-layer FNNs with **random weights** in the hidden layer are universal approximators with different activation functions, including the most usual ones
- Based on this result, the Extreme Learning Machines (ELMs) algorithm was proposed [Huang et al., 2006b] for the construction of FNNs with  $N$  hidden units

- The algorithm is very simple:

---

## Algorithm

**Randomly** assign  $N$  hidden-unit parameters  $\omega_1, \dots, \omega_N$

Compute the hidden-layer output matrix

$$\mathbf{V} = (v_{\omega_1}, \dots, v_{\omega_N}) \in \mathbb{R}^L \times \mathbb{R}^N$$

Compute the output-layer weights  $\lambda = \mathbf{V}^\dagger Y = (\mathbf{V}^t \mathbf{V})^{-1} \mathbf{V}^t Y$

---

Same linear equations than for interacting hidden-layer weights

- Currently, ELMs is an **active area of research**  
[Huang et al., 2011, Huang et al., 2015]

- In [Feng et al., 2009], the Error Minimized ELMs was proposed, a constructive method based on ELMs

---

## Algorithm

$N = 0$

**while**  $N < M_{\max}$  **and** **Error**  $> \epsilon$

**Randomly** add  $\delta N$  hidden-unit parameters  $\omega_{N+1}, \dots, \omega_{N+\delta N}$

Compute the hidden-layer output matrix

$$\mathbf{V} = (v_{\omega_1}, \dots, v_{\omega_N}, v_{\omega_{N+1}}, \dots, v_{\omega_{N+\delta N}}) \in \mathbb{R}^L \times \mathbb{R}^{N+\delta N}$$

Compute the output-layer weights  $\lambda = \mathbf{V}^\dagger \mathbf{Y}$  “efficiently”

$N = N + \delta N$

**end while**

---

# Comparison of constructive algorithms with Constructive Extreme Learning Machines

- Work described in [Romero and Alquézar, 2012]
- Comparison of
  - Support Vector Sequential FNNs (SVSFNN), the constructive algorithm described in [Romero and Toppo, 2007]
    - Interacting hidden-layer weights
    - Selection of hidden-layer weights: Input data set
    - Output-layer weights bounded by a constant  $B$
  - Constructive Extreme Learning Machines
- Similar experimental setting than the previous slides



# Datasets

<b>Data Set</b>	<b>#Var</b>	<b>#Cla</b>	<b>#Exa</b>
<i>Australian Credit</i>	43	2	690
<i>Gene</i>	120	3	3175
<i>German Credit</i>	56	2	1000
<i>Ionosphere</i>	34	2	351
<i>Iris</i>	4	3	150
<i>Satimage</i>	36	6	6435
<i>Segmentation</i>	16	7	2310
<i>Sonar</i>	60	2	208
<i>Vehicle</i>	18	4	846
<i>Wine</i>	13	3	178

**Table:** Description of the **classification** data sets. Columns '**#Var**', '**#Cla**' and '**#Exa**': number of variables, classes and examples, respectively

Data Set	#Var	#Exa	Target Mean	StdDev
<i>Abalone</i>	8	4177	9.93	3.22
<i>Auto Price</i>	15	159	11445.73	5877.85
<i>Boston Housing</i>	13	506	22.53	9.20
<i>Calif. Housing</i>	8	20640	206854.97	115395.58
<i>Census House</i>	8	22784	50073.10	52846.16
<i>Delta Ailerons</i>	5	7129	-0.000007	0.0003
<i>Delta Elevators</i>	6	9517	-0.000133	0.0023
<i>Machine CPU</i>	6	209	105.62	160.83
<i>Servo</i>	4	167	1.39	1.56
<i>Stock</i>	9	950	46.99	6.54

**Table:** Description of the **regression** data sets. Columns '**#Var**', '**#Exa**': number of variables and examples, respectively

# Average test accuracies for **classification** sets

Data Set	Gaussian RBF		Sigmoid MLP	
	Input	Rand.	Input	Rand.
<i>Australian</i>	84.49	83.48	<b>85.02</b>	<u>83.82</u>
<i>Gene</i>	<b>86.60</b>	<u>86.05</u>	86.36	85.92
<i>German</i>	77.13	<b>78.33</b>	<b>78.03</b>	77.37
<i>Ionosphere</i>	<b>93.90</b>	<u>90.67</u>	90.00	88.83
<i>Iris</i>	100	100	100	100
<i>Satimage</i>	<b>86.35</b>	<u>83.31</u>	81.73	77.50
<i>Segmentation</i>	<b>88.56</b>	83.39	87.30	<u>86.83</u>
<i>Sonar</i>	<b>96.50</b>	<u>81.83</u>	75.00	74.17
<i>Vehicle</i>	86.67	85.56	<b>86.75</b>	<b>86.63</b>
<i>Wine</i>	100	100	100	100

# Average test normalized squared error for regression sets

Data Set	Gaussian RBF		Sigmoid MLP	
	Input	Rand.	Input	Rand.
<i>Abalone</i>	0.512	<b>0.524</b>	<b>0.511</b>	0.537
<i>Auto Price</i>	<b>0.177</b>	0.840	0.299	<u>0.517</u>
<i>Boston Housing</i>	<b>0.657</b>	<b>0.620</b>	0.745	0.921
<i>Calif. Housing</i>	<b>0.332</b>	<u>0.345</u>	0.369	0.365
<i>Census House</i>	<b>0.383</b>	0.508	0.403	<u>0.470</u>
<i>Delta Ailerons</i>	<b>0.293</b>	<b>0.295</b>	0.299	0.301
<i>Delta Elevators</i>	0.375	0.376	<b>0.375</b>	<b>0.375</b>
<i>Machine CPU</i>	<b>0.299</b>	<u>0.348</u>	0.345	0.380
<i>Servo</i>	<b>0.158</b>	<u>0.206</u>	0.171	0.300
<i>Stock</i>	<u>1.714</u>	<b>1.608</b>	3.318	1.807

# Average number of hidden units for **classification** sets

Data Set	Gaussian RBF		Sigmoid MLP	
	Input	Rand.	Input	Rand.
<i>Australian</i>	50.63	23.33	24.93	21.77
<i>Gene</i>	95.23	89.87	88.33	84.80
<i>German</i>	12.20	12.20	13.83	16.90
<i>Ionosphere</i>	65.73	29.03	12.53	12.67
<i>Iris</i>	19.13	4.50	3.97	2.77
<i>Satimage</i>	97.67	89.73	95.40	97.43
<i>Segmentation</i>	95.83	32.00	91.87	68.27
<i>Sonar</i>	88.27	55.43	25.13	21.13
<i>Vehicle</i>	81.73	37.13	58.37	54.27
<i>Wine</i>	4.90	7.13	5.13	6.23

# Average number of hidden units for regression sets

Data Set	Gaussian RBF		Sigmoid MLP	
	Input	Rand.	Input	Rand.
<i>Abalone</i>	34.60	48.17	31.70	56.23
<i>Auto Price</i>	7.37	4.20	3.00	4.87
<i>Boston Housing</i>	11.93	14.90	1.30	8.80
<i>Calif. Housing</i>	88.87	54.00	7.10	14.47
<i>Census House</i>	95.13	37.00	91.23	78.27
<i>Delta Ailerons</i>	54.73	34.43	72.20	72.20
<i>Delta Elevators</i>	40.17	57.90	36.20	29.20
<i>Machine CPU</i>	5.37	3.97	5.27	3.53
<i>Servo</i>	61.57	21.17	57.37	30.63
<i>Stock</i>	4.93	1.00	1.13	1.33

- A Student's t-test showed (**red** vs underline) that
  - In **six of the classification data sets** (Australian Credit, Gene, Ionosphere, Satimage, Segmentation, Sonar)
  - In **five of the regression data sets** (Auto Price, California Housing, Census House, Machine CPU, Servo)

**results of the input strategy were significantly better**, whereas no significant difference was found in the other ones, except for the Stock data set
- (small differences are sometimes more significant than larger ones because the former have very small variances)
- Although no clear trend is observed about the number of hidden units selected by both strategies, the input strategy seems to need more units than the random strategy in the case of Gaussian RBF hidden units

That's it!





# Bibliography

- ▶ Ash, T. (1989).  
Dynamic Node Creation in Backpropagation Networks.  
*Connection Science*, 1(4):365–375.
- ▶ Chen, S., Cowan, C. F. N., and Grant, P. M. (1991).  
Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks.  
*IEEE Transactions on Neural Networks*, 2(2):302–309.
- ▶ Fahlman, S. E. and Lebiere, C. (1990).  
The Cascade-Correlation Learning Architecture.  
In *Advances in Neural Information Processing Systems*, volume 2, pages 524–532. Morgan Kaufmann.
- ▶ Feng, G., Huang, G. B., Lin, Q., and Gay, R. (2009).  
Error Minimized Extreme Learning Machine with Growth of Hidden Nodes and Incremental Learning.  
*IEEE Transactions on Neural Networks*, 20(8):1352–1357.
- ▶ Friedman, J. H. and Stuetzle, W. (1981).  
Projection Pursuit Regression.  
*Journal of the American Statistical Association*, 76:817–823.
- ▶ Geman, S., Bienenstock, E., and Doursat, R. (1992).  
Neural Networks and the Bias/Variance Dilemma.  
*Neural Computation*, 4(1):1–58.
- ▶ Huang, G., Huang, G. B., Song, S., and You, K. (2015).  
Trends in Extreme Learning Machines: A Review.  
*Neural Networks*, 61:32–48.
- ▶ Huang, G. B., Chen, L., and Siew, C. K. (2006a).  
Universal Approximation using Incremental Constructive Feedforward Networks with Random Hidden Nodes.  
*IEEE Transactions on Neural Networks*, 17(4):879–892.

# Bibliography

- ▶ Huang, G. B., Wang, D. H., and Lan, Y. (2011).  
Extreme Learning Machines: a Survey.  
*International Journal of Machine Learning and Cybernetics*, 2:107–122.
- ▶ Huang, G. B., Zhu, Q. Y., and Siew, C. K. (2006b).  
Extreme Learning Machine: Theory and Applications.  
*Neurocomputing*, 70(1-3):489–501.
- ▶ Hwang, J. N., Ray, S. R., Maechler, M., Martin, D., and Schimert, J. (1994).  
Regression Modelling in Back-Propagation and Projection Pursuit Learning.  
*IEEE Transactions on Neural Networks*, 5(3):342–353.
- ▶ Jones, L. K. (1987).  
On a Conjecture of Huber Concerning the Convergence of Projection Pursuit Regression.  
*The Annals of Statistics*, 15(2):880–882.
- ▶ Keerthi, S. S., Chapelle, O., and DeCoste, D. (2006).  
Building Support Vector Machines with Reduced Classifier Complexity.  
*Journal of Machine Learning Research*, 7:1493–1515.
- ▶ Kúrková, V. and Beliczyński, B. (1995).  
Incremental Approximation by One-Hidden-Layer Neural Networks.  
In *International Conference on Artificial Neural Networks*, volume 1, pages 505–510.
- ▶ Kwok, T. Y. and Yeung, D. Y. (1997).  
Objective Functions for Training New Hidden Units in Constructive Neural Networks.  
*IEEE Transactions on Neural Networks*, 8(5):1131–1148.
- ▶ Mallat, S. G. and Zhang, Z. (1993).  
Matching Pursuits with Time-Frequency Dictionaries.  
*IEEE Transactions on Signal Processing*, 41(12):3397–3415.

- ▶ Romero, E. (2004).  
*Learning with Feed-forward Neural Networks: Three Schemes to Deal with the Bias/Variance Trade-off.*  
PhD thesis, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain.
- ▶ Romero, E. and Alquézar, R. (2006).  
A Sequential Algorithm for Feed-forward Neural Networks with Optimal Coefficients and Interacting Frequencies.  
*Neurocomputing*, 69(13-15):1540–1552.
- ▶ Romero, E. and Alquézar, R. (2007).  
Heuristics for the Selection of Weights in Sequential Feed-forward Neural Networks: An Experimental Study.  
*Neurocomputing*, 70(16-18):2735–2743.
- ▶ Romero, E. and Alquézar, R. (2012).  
Comparing Error Minimized Extreme Learning Machines and Support Vector Sequential Feed-forward Neural Networks.  
*Neural Networks*, 25:122–129.
- ▶ Romero, E. and Toppo, D. (2007).  
Comparing Support Vector Machines and Feed-forward Neural Networks with Similar Hidden-layer Weights .  
*IEEE Transactions on Neural Networks*, 18(3):959–963.
- ▶ Vincent, P. and Bengio, Y. (2002).  
Kernel Matching Pursuit.  
*Machine Learning*, 48(1-3):165–187.  
Special Issue on New Methods for Model Combination and Model Selection.