ADAPTIVE MULTIPLE OPTIMAL LEARNING FACTORS FOR NEURAL NETWORK TRAINING

Jeshwanth Challagundla
Supervising Prof: Dr. Michael T. Manry
April 10 2015

UNIVERSITY OF TEXAS ARLINGTON

Outline

I. Multilayer perceptron review
II. Review of neural network training algorithms
III. Goals
IV. Proposed algorithm
V. Simulations
VI. Conclusions

I. MLP Review

- Widely used for function approximation and pattern recognition
  - In function approximation a function that maps between input and output is approximated
  - In pattern recognition a received signal is assigned to one of a prescribed number of classes
- Known for its special properties of approximating Bayes discriminants and universal approximation
- Function approximation is the focus of this thesis

$\mathbf{W} \rightarrow$ Input weights

$\mathbf{W_{oi}} \rightarrow$ Bypass weights

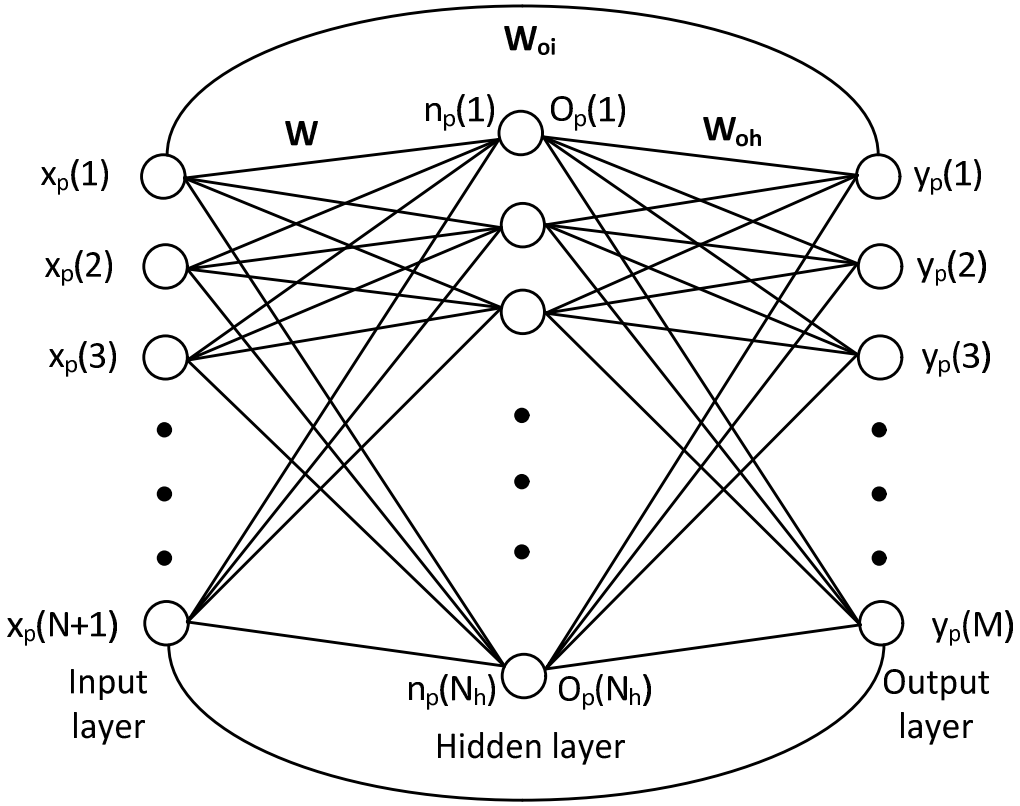
$\mathbf{W_{oh}} \rightarrow$ Output to hidden weights

$$\mathbf{n_p} = \mathbf{W} \cdot \mathbf{x_p}$$

$$O_p(k) = \frac{1}{1 + e^{-n_p(k)}}$$

$$\mathbf{y_p} = \mathbf{W_{oi}} \cdot \mathbf{x_p} + \mathbf{W_{oh}} \cdot \mathbf{O_p}$$

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[t_p(i) - y_p(i) \right]^2$$



II. Review of NN training algorithms

- Conjugate Gradient

Let, $\mathbf{w} = \mathrm{vec}\{\mathbf{W}, \mathbf{W}_{oi}, \mathbf{W}_{oh}\}$, $\mathbf{g} = \mathrm{vec}\{\mathbf{G}, \mathbf{G}_{oi}, \mathbf{G}_{oh}\}$

$$\mathbf{p} = \mathrm{vec}\{\mathbf{P}, \mathbf{P}_{oi}, \mathbf{P}_{oh}\}$$

$$\boldsymbol{p}_{k+1} = \mathbf{g}_k + \mathrm{B}_l . \boldsymbol{p}_k$$

$$\mathrm{B}_l = \frac{\mathbf{g}_{k+1}^{\mathrm{T}} \mathbf{g}_{k+1}}{\mathbf{g}_k^{\mathrm{T}} \mathbf{g}_k} \quad , \quad \mathbf{w}_{k+1} = \mathbf{w}_k + z \cdot \boldsymbol{p}_k$$

- Minimizes quadratic error functions of *n* variables in *n* steps
- No Hessian computation is required

- Levenberg Marquardt

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{e}$$

$$\mathbf{e} = \left[\mathbf{H}' + \lambda \mathbf{I}\right]^{-1} \mathbf{g}'$$

$$h'(m,n) = \frac{\partial^2 E}{\partial w(m) \partial w(n)} \qquad g'(m) = \frac{-\partial E}{\partial w(m)}$$

- By changing λ the algorithm interpolates between steepest descent and Gauss-Newton methods
- Computationally very expensive
- Mostly used for small networks

Newton-related improved training

Newton-related algorithms usually converge faster per iteration

- Output Weight Optimization

$$\mathbf{R} = \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{X_p} \mathbf{X_p}^T$$

$$\mathbf{C} = \frac{1}{N_v} \sum_{p=1}^{N_v} \mathbf{X_p} \mathbf{t_p}^T$$

Augmented input:

$$X_p = \begin{bmatrix} x_p \\ O_p \end{bmatrix}$$

$$\mathbf{W_o}^T = \mathbf{R}^{-1} \cdot \mathbf{C}$$

- Equivalent to Newton's method for output weights
- Computationally less expensive

- OWO-MOLF

In each iteration:

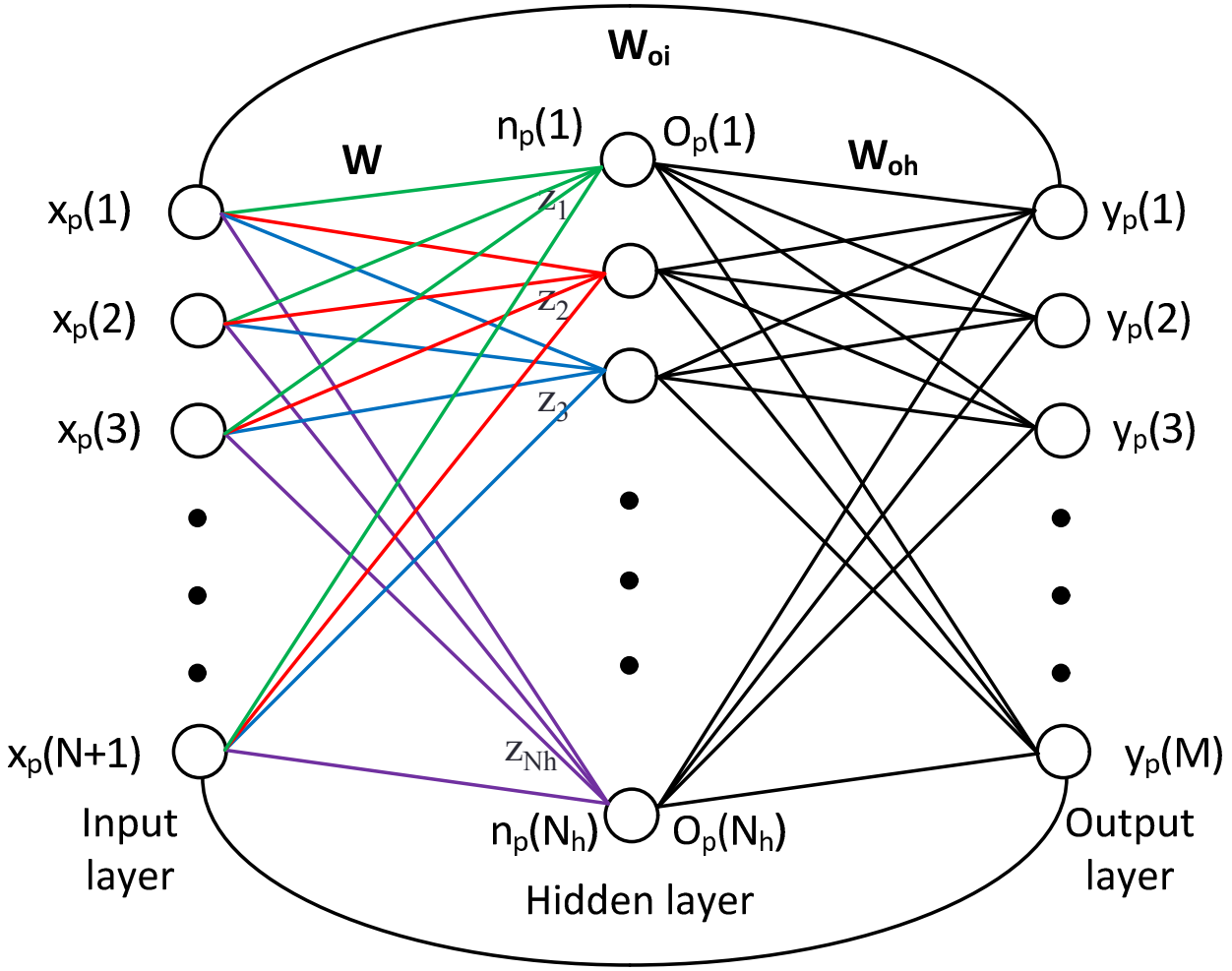
- Input weights are trained using
  - Learning factor vector **z** of size $N_h \times 1$
  - Negative gradient vector **g**

- Output weights are trained using OWO

- z_k , k^{th} element of vector **z** used to update all the weights connected to hidden unit k

$$y_p(i) = \sum_{n=1}^{N+1} w_{oi}(i,n)\, x_p(n) + \sum_{k=1}^{N_h} w_{oh}(i,k)\, f\left(\sum_{n=1}^{N+1} \left(w(k,n) + z_k\, g(k,n) \right) x_p(n) \right)$$

MOLF demonstration



- Learning factor vector **z** is computed as

$$\mathbf{z} = \mathbf{H}_{\mathbf{molf}}^{-1}\mathbf{g}_{\mathbf{molf}}$$

- Element of matrix $\mathbf{H}_{\mathbf{molf}}$,

$$h_{molf}(k, j) = \frac{\partial^2 E}{\partial z_k \partial z_j}$$

- $\mathbf{g}_{\mathbf{molf}} = \left[-\partial E/\partial z_1, -\partial E/\partial z_2 ...., -\partial E/\partial z_{N_h}\right]^T$

- OWO-MOLF trains much faster than BP

- OWO-Newton

In each iteration:

- Newton's method for input weights

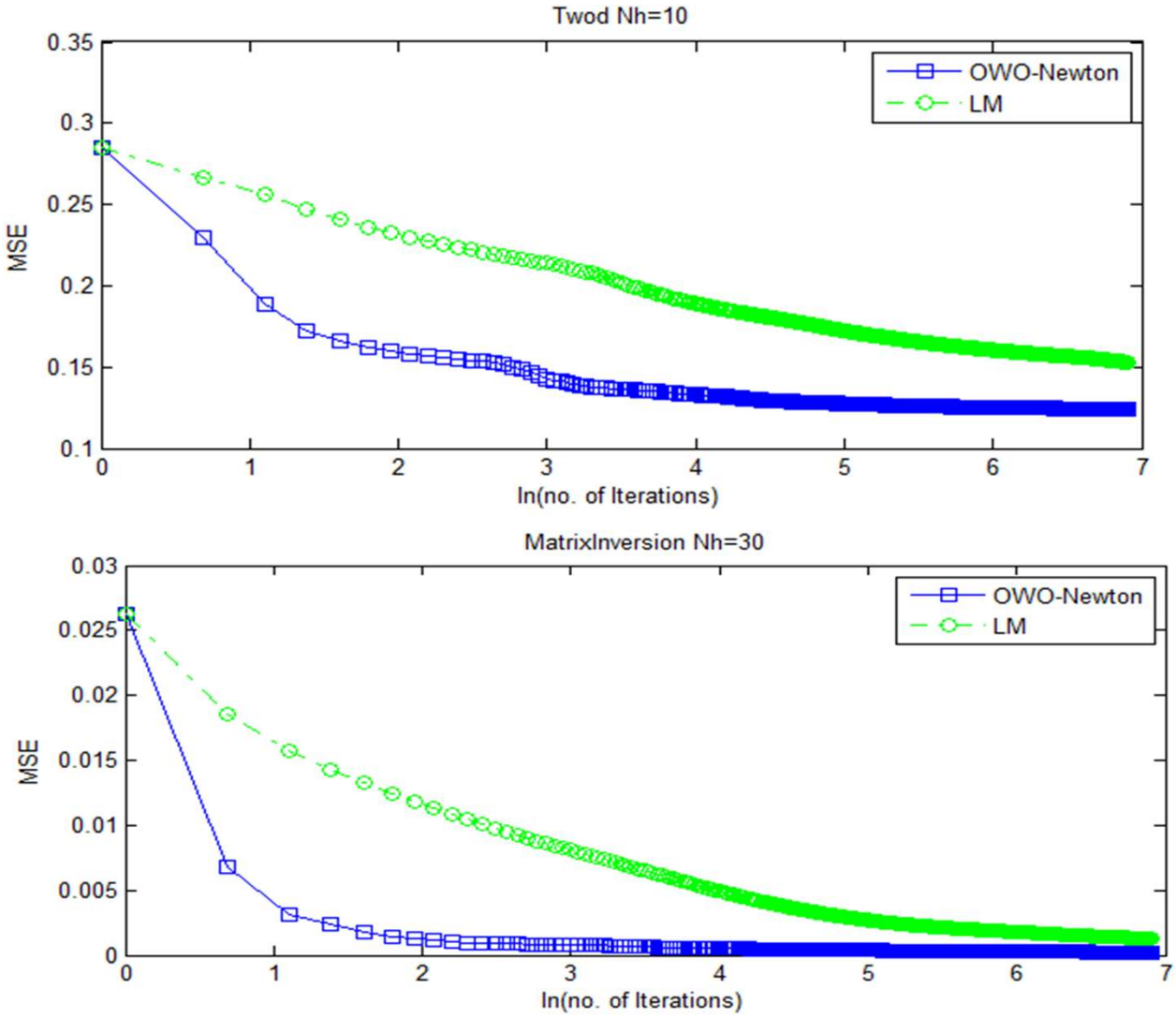
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{e} \qquad\qquad \mathbf{w} = \text{vec}\{\mathbf{W}\}$$

$$\mathbf{e} = \mathbf{H}^{-1} \cdot \mathbf{g}$$

- $\mathbf{H}$ $\longrightarrow$ Hessian of error with respect to input weights in the network
- $\mathbf{g}$ $\longrightarrow$ negative input weight gradient vector

- Output weights are solved using OWO

- OWO-Newton has simpler and faster code than LM for similar performance



- Problems

- Little theoretical justification for using small second order modules in first order algorithms

- Second order training is computationally very expensive

- Error decrease per iteration is larger in OWO-Newton, number of multiplies per iteration is less for OWO-MOLF

- Lemmas
  - Lemma 1: Assume for quadratic E(**w**), **w** is divided into k partitions $\mathbf{w_k}$ If we minimize E with respect to **z** producing an error $E_k = E(\mathbf{w_1} + z_1\mathbf{g_1}, \mathbf{w_2} + z_2\mathbf{g_2}, ...\mathbf{w_k} + z_k\mathbf{g_k})$ and k increases by splitting an existing partition, then $E_{k+1} \leq E_k$
  - Implications
    - More learning factors means more error decrease
    - Not guaranteed since E(**w**) is not quadratic

III. Goals

- Develop a new algorithm that adapts between OWO-MOLF and OWO-Newton
- Maximize error decrease per multiply
- Simulations

IV. Adaptive Multiple Optimal Learning Factors

In each iteration:

- Input weights are trained using
  - Learning factor vector **z**
  - Negative gradient vector **g**

- Output weights are trained using OWO

- Size of **z** can vary between N_h and $N_h \times (N+1)$ in each iteration

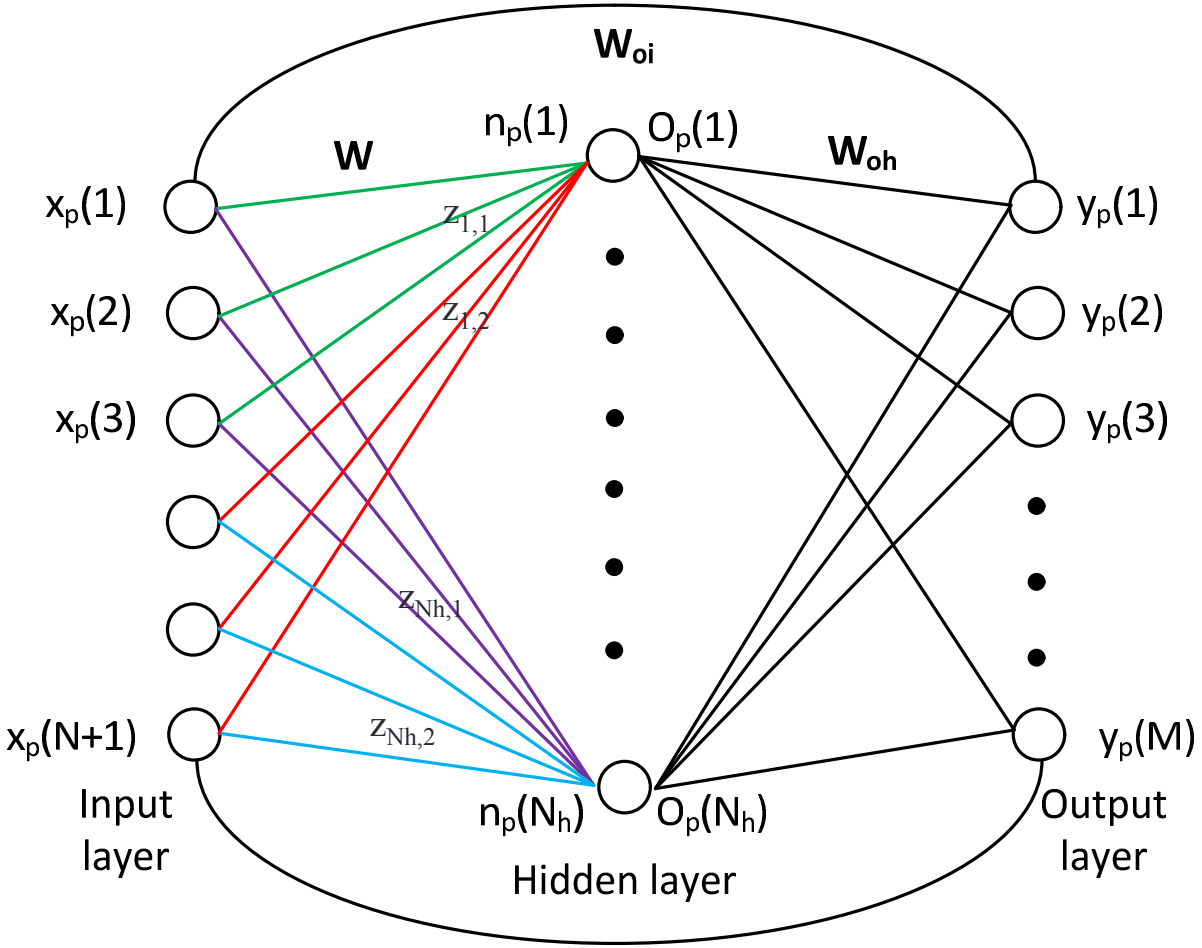
- This algorithm adapts between OWO-MOLF and OWO-Newton

- The input weights connected to each hidden unit are grouped into N_g groups

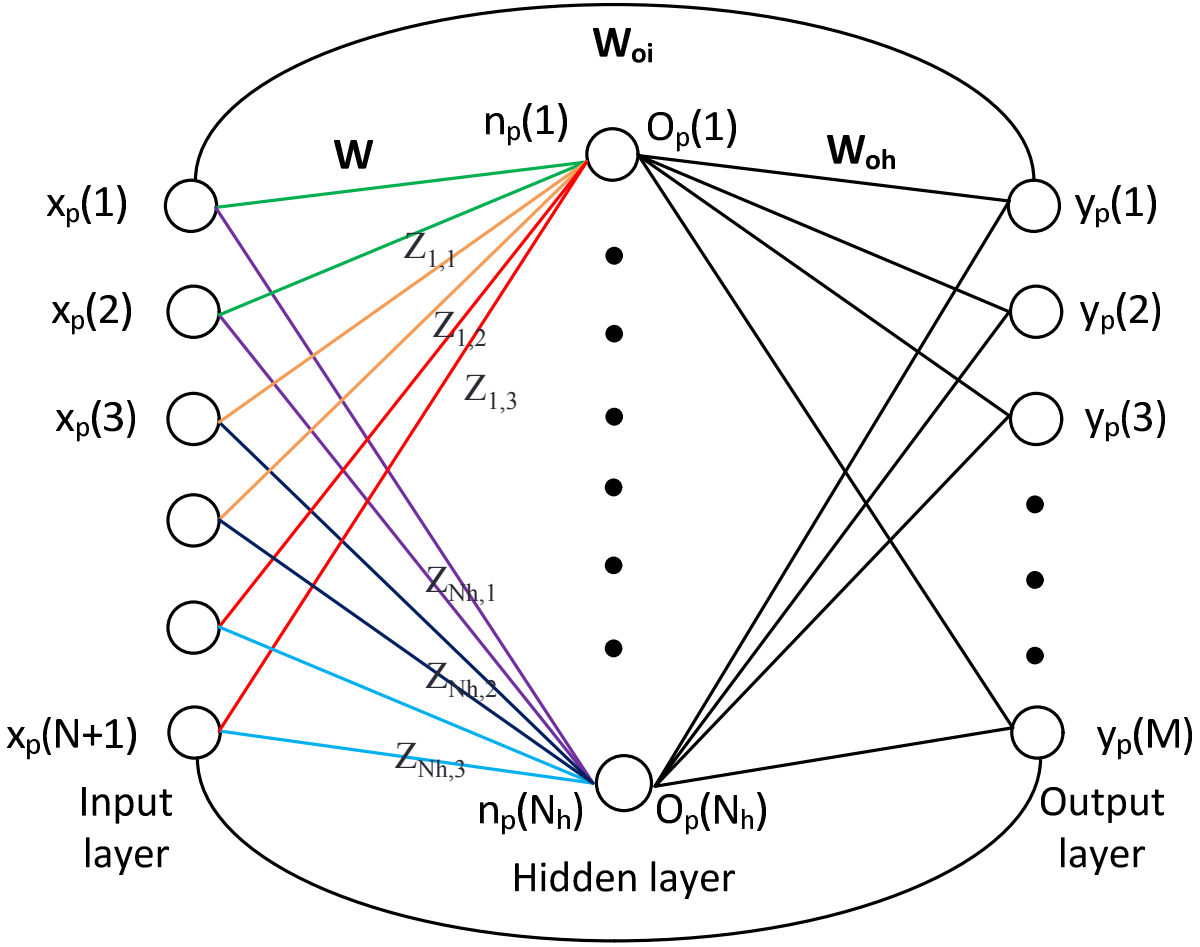
- An unique learning factor is computed for each of this groups

- $z_{k,C}$ is an element of **z** used to update all the input weights that belong to the group C of hidden unit k.

Adaptive MOLF with N_g=2



Adaptive MOLF with N_g=3



- Grouping of input weights

- Grouping is done based on the curvature $\mathbf{H_w}$ of the error function calculated with respect to input weights

- The elements of $\mathbf{H_w}$ can be calculated as,

$$h_w(k,n) = \frac{\partial^2 E}{\partial w(k,n)^2} = \frac{2}{N_v} \sum_{i=1}^{M} w_{oh}(i,k)^2 \sum_{p=1}^{N_v} f'(n_p(k))^2 x_p(n)^2$$

- How is the number of groups changed

- Goal is to vary number of groups so that the error change per multiply is maximized
- Number of groups per hidden unit increases as the error change per multiply (EPM) increases and vice versa.

$$EPM(i_t) = \frac{E(i_t - 1) - E(i_t)}{M(i_t)}$$

- $M(i_t)$ stands for number of multiplies in iteration i_t
- $EPM(i_t)$ is the error per multiply in iteration i_t
- $E(i_t)$ it the error computer in iteration i_t
- i_t is the current iteration number.

- Derivation of adaptive MOLF algorithm

$$y_p(i) = \sum_{n=1}^{N+1} x_p(n) w_{oi}(k,n) + \sum_{k=1}^{N_h} w_{oh}(i,k) f\left(\sum_{C=1}^{N_g} \sum_{a=R(C-1)+1}^{R(C)} x_p(i_k(a)) \left[w(k,i_k(a)) + z_{k,C} g(k,i_k(a)) \right] \right)$$

$$R(C) = C \times Gs(C)$$

$$Gs(0) = 0$$

- C is the group index
- **Gs** is an array of $N_g \times 1$ elements, these elements contains the sizes of all group the belong to a hidden unit
- $\mathbf{I_k}$ contains input indices ordered in such a way that the index n of input to which weight w(k,n) with higher curvatures are connected will come first.

- $\mathbf{I_k} = [n_1, n_2, n_3 \ldots\ldots, n_{N+1}]$ where $n_1, n_2, n_3 \ldots\ldots, n_{N+1}$ are input indices such that $h_w(k,n_1) \geq h_w(k,n_2) \geq h_w(k,n_3) \ldots\ldots\ldots\ldots \geq h_w(k,n_{N+1})$

- Hessian and negative gradient elements

$$g_{Amolf}(k,C) = \frac{-\partial E}{\partial z_{k,C}} = \frac{2}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left(t_p(i) - y_p(i)\right) \frac{\partial y_p(i)}{\partial z_{k,C}}$$

$$\frac{\partial y_p(i)}{\partial z_{k,C}} = w_{oh}(i,k) f'\left(n_p(k)\right) \Delta n_p(k,C)$$

$$\Delta n_p(k,C) = \sum_{a=R(C-1)+1}^{R(C)} x_p\left(i_k(a)\right) g\left(k, i_k(a)\right)$$

$$h_{Amolf}(k,C_1,j,C_2) = \frac{2}{N_v} \sum_{i=1}^{M} w_{oh}(i,k) w_{oh}(i,j) \sum_{p=1}^{N_v} f'\left(n_p(k)\right) f'\left(n_p(j)\right) \Delta n_p(k,C_1) \Delta n_p(j,C_2)$$

- The elements of the 2 dimensional Hessian $\mathbf{H_{Amolf}}$ are found from $h_{Amolf}(k,C_1,j,C_2)$ as

$$h_{Amolf}\left((k-1)N_g + C_1, (j-1)N_g + C_2\right) = h_{Amolf}\left(k, C_1, j, C_2\right)$$

- Elements of negative gradient column vector $\mathbf{g_{Amolf}}$ are calculated from $g_{Amolf}(k,C)$ as,

$$g_{Amolf}\left((k-1)N_g + C\right) = g_{Amolf}\left(k, C\right)$$

$$\mathbf{z} = \mathbf{H}_{\mathbf{Amolf}}^{-1} \cdot \mathbf{g}_{\mathbf{Amolf}}$$

- The input weights **W** are updated as follows,

$$w\left(k, i_k\left(a\right)\right) = w\left(k, i_k\left(a\right)\right) + z_{k,C}\, g\left(k, i_k\left(a\right)\right)$$

where,

$$[(C\text{-}1) \times Gs(C\text{-}1)] + 1 \leq a \leq [C \times Gs(C)]$$

- Lemma 2: $E\text{-}E_{MOLF}$ and $E\text{-}E_{aMOLF}$ denote the error decrease due to the Newton steps of OWO-MOLF and adaptive MOLF respectively, then $E\text{-}E_{MOLF} \leq E\text{-}E_{aMOLF}$

- Lemma 3: OWO-Newton is a limiting case of the adaptive MOLF algorithm as the k groups of adaptive MOLF are split until $k=N_h \cdot (N+1)$

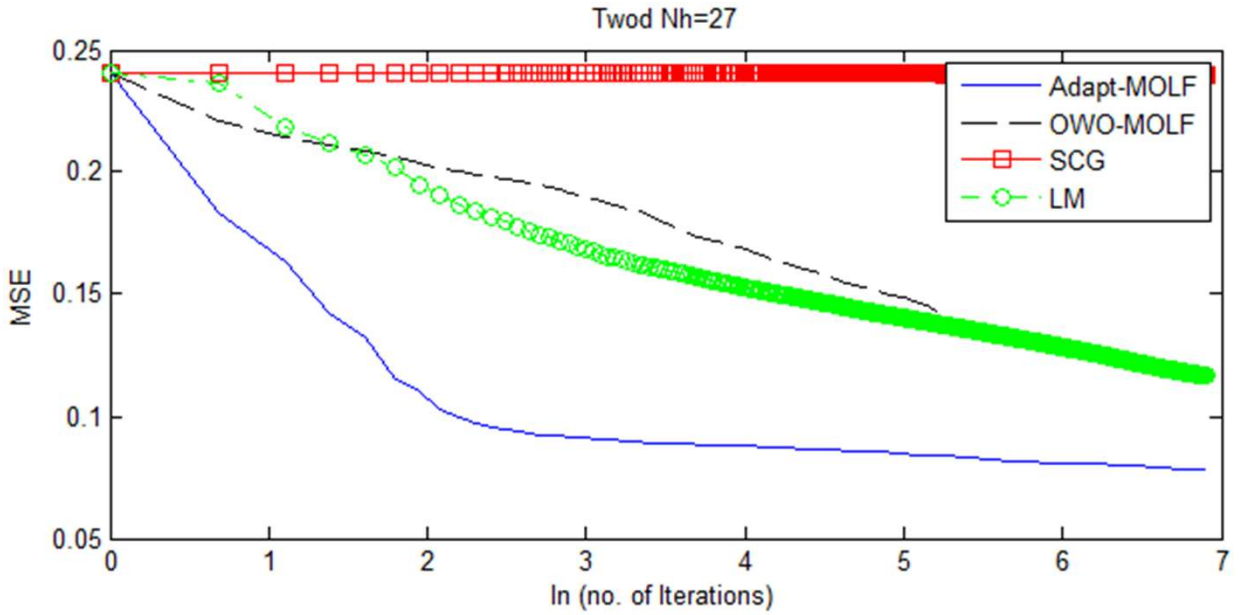
- Adaptive MOLF Hessian and gradients can be generated from Gauss-Newton Hessian and gradients as follows

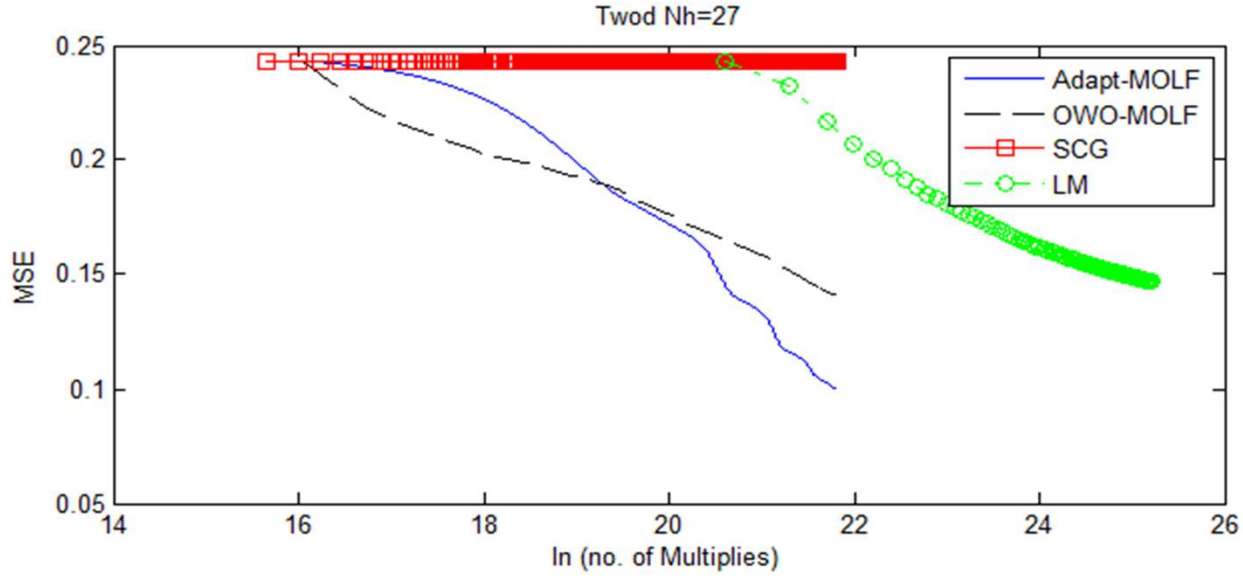
$$g_{\text{Amolf}}(k,C) = \sum_{a=R(C-1)+1}^{R(C)} g(k,i_k(a))^2$$

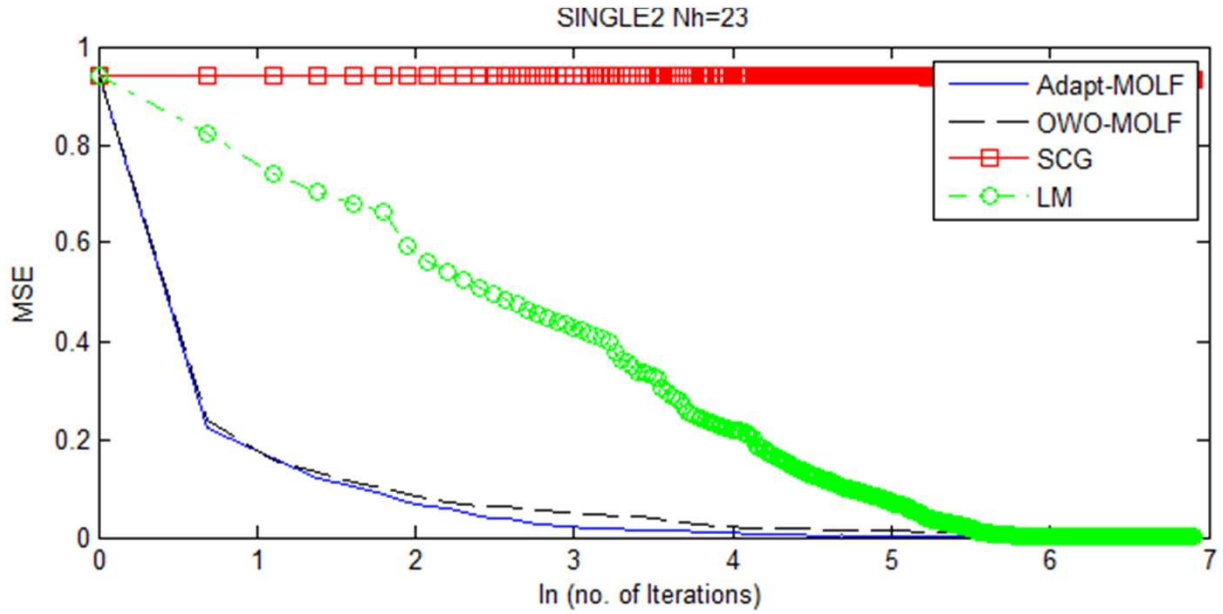
$$h_{\text{Amolf}}(k,C_1,j,C_2) = \sum_{a=R(C_1-1)+1}^{R(C_1)} \sum_{b=R(C_2-1)+1}^{R(C_2)} h(k,i_k(a),j,i_j(b)) g(k,i_k(a)) g(j,i_j(b))$$

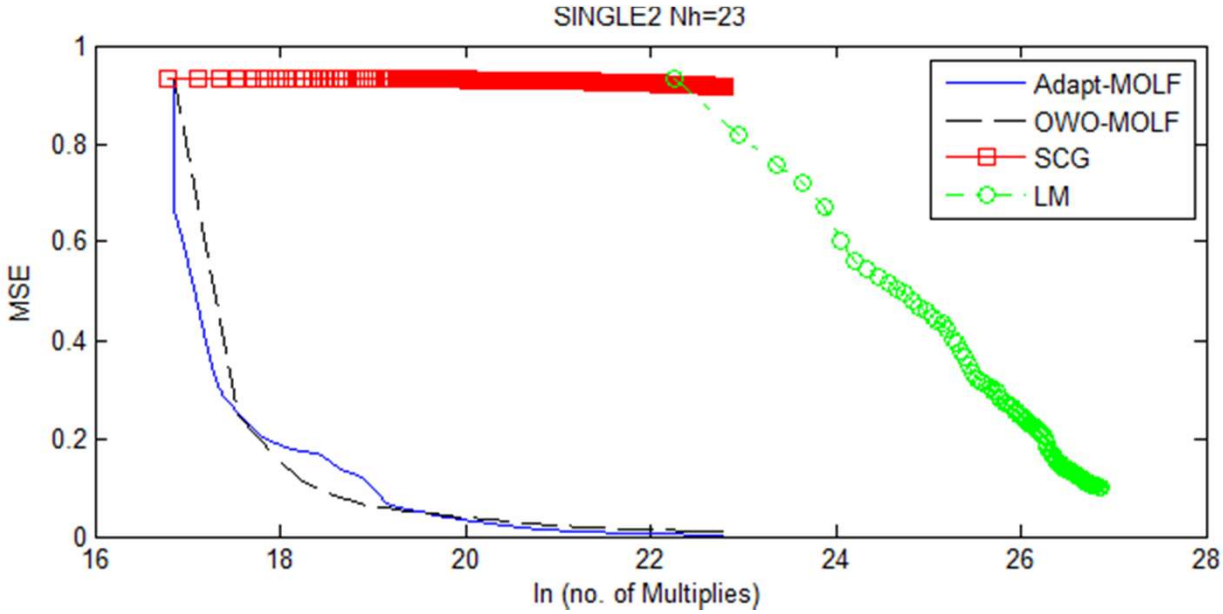
V. Simulations

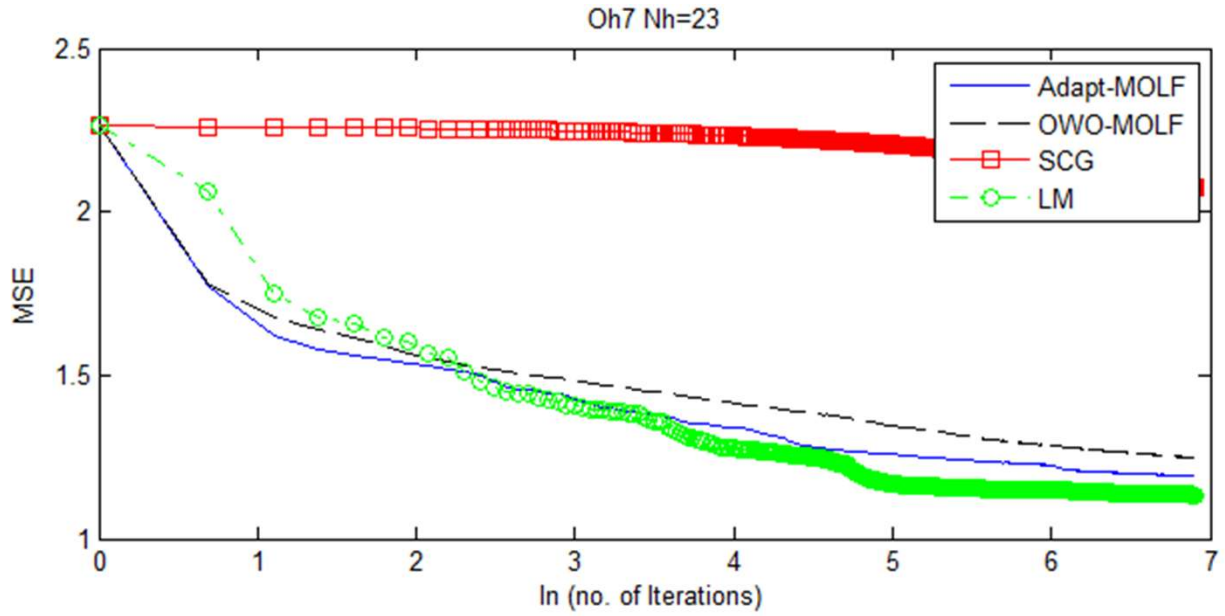
- Training is done on the entire data set 10 times with 10 different initial networks. The average MSE from this 10-fold training is shown in the plots below.

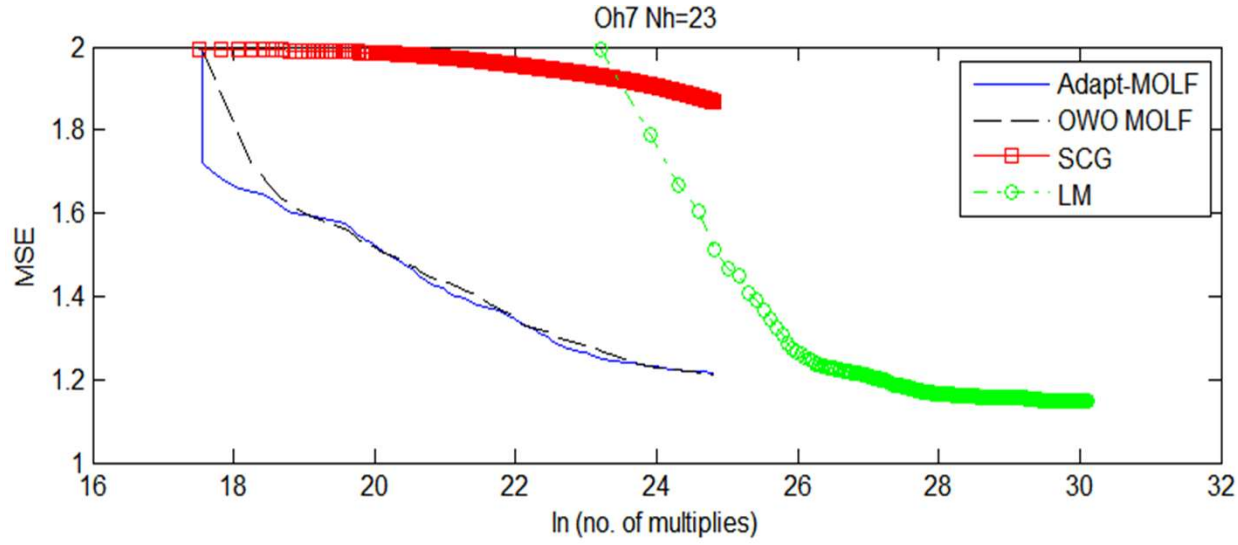

Twod Nh=27
MSE
ln (no. of Iterations)
Adapt-MOLF
OWO-MOLF
SCG
LM
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
5
6
7

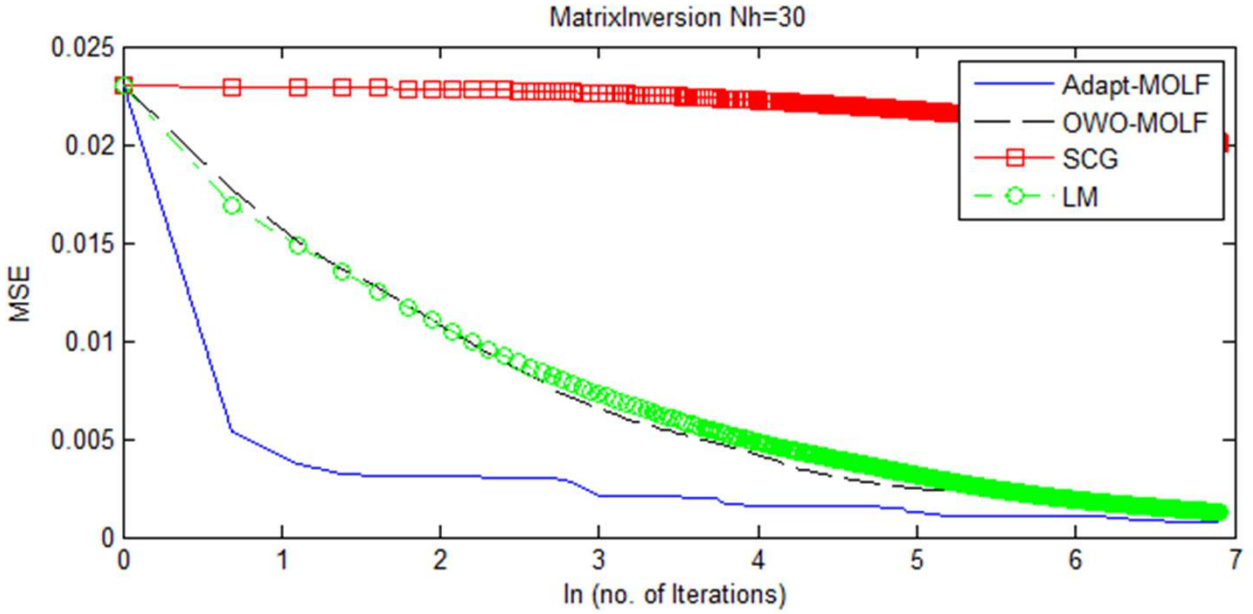

Twod Nh=27
MSE
ln (no. of Multiplies)
Adapt-MOLF
OWO-MOLF
SCG
LM
0.25
0.2
0.15
0.1
0.05
14
16
18
20
22
24
26

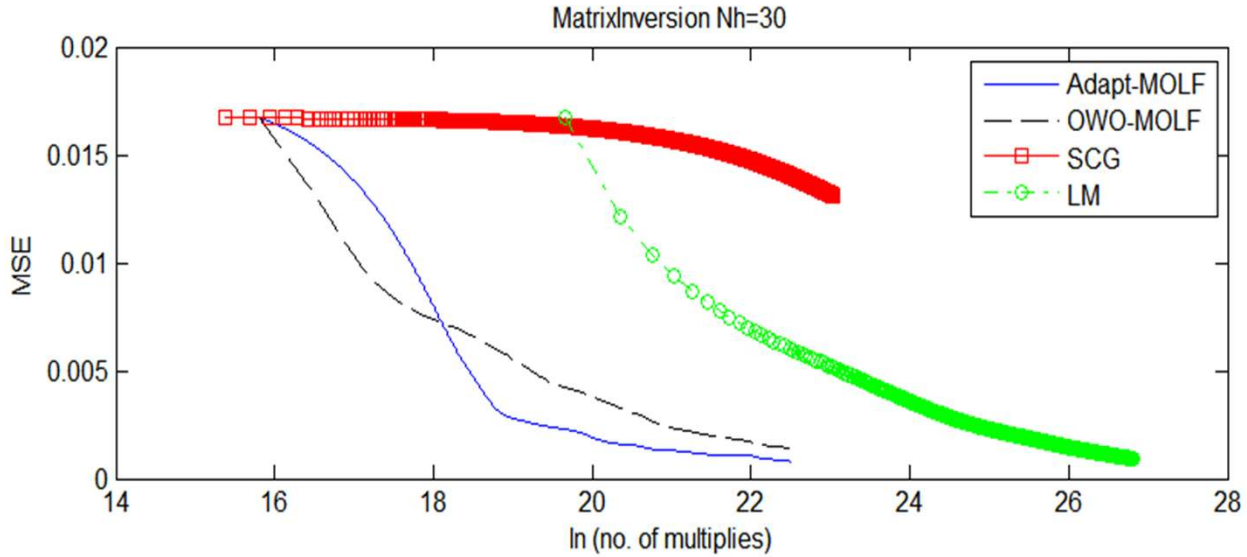

SINGLE2 Nh=23
Adapt-MOLF
OWO-MOLF
SCG
LM
MSE
ln (no. of Iterations)


SINGLE2 Nh=23
Adapt-MOLF
OWO-MOLF
SCG
LM
MSE
ln (no. of Multiplies)


Oh7 Nh=23
MSE
ln (no. of Iterations)
Adapt-MOLF
OWO-MOLF
SCG
LM
2.5
2
1.5
1
0
1
2
3
4
5
6
7


Oh7 Nh=23
MSE
ln (no. of multiplies)
Adapt-MOLF
OWO MOLF
SCG
LM
2
1.8
1.6
1.4
1.2
1
16
18
20
22
24
26
28
30
32


MatrixInversion Nh=30
MSE
ln (no. of Iterations)
Adapt-MOLF
OWO-MOLF
SCG
LM
0.025
0.02
0.015
0.01
0.005
0
0
1
2
3
4
5
6
7


MatrixInversion Nh=30
MSE
ln (no. of multiplies)
Adapt-MOLF
OWO-MOLF
SCG
LM
0.02
0.015
0.01
0.005
0
14
16
18
20
22
24
26
28

K-fold validation and testing

Data Set		Adaptive MOLF	OWO-MOLF	SCG	LM
Twod.tra	E_{trn}	0.0888	0.1554	1.0985	0.2038
	E_{tst}	0.1172	0.1731	1.0945	0.2205
Single2.tra	E_{trn}	0.0042	0.0151	3.5719	0.0083
	E_{tst}	0.2319	0.1689	3.6418	0.0178
Mattrn.tra	E_{trn}	0.0011	0.0027	4.2400	0.0022
	E_{tst}	0.0013	0.0032	4.3359	0.0027
Oh7.tra	E_{trn}	1.2507	1.3205	4.1500	1.1602
	E_{tst}	1.4738	1.4875	4.1991	1.4373

VI. Conclusions

- The number of learning factors needed by a training algorithm is addressed and adaptively optimized
- The adaptive MOLF algorithm is superior to the OWO-MOLF algorithm in terms of error decrease per iteration and often in terms of error decrease per multiply
- The proposed algorithm interpolates between OWO-MOLF and OWO-Newton

Thank you

Questions