國立中興大學科技管理學系
碩士學位論文

在多層感知器訓練時加入權重雜訊和權重衰減的研究

Empirical studies on the online learning algorithms based on combining weight noise injection and weight decay

指導教授: 沈培輝　博士
研 究 生: 梁晏綸

中華民國99年05月

國立中興大學科技管理學系
碩士學位論文

在多層感知器訓練時加入權重雜訊和權重衰減的研究

Empirical studies on the online learning algorithms based on combining weight noise injection and weight decay

指導教授: 沈培輝　博士
研 究 生: 梁晏綸

中華民國99年05月

Institute of Technology Management
National Chung Hsing University

Thesis for the Degree of Master

在多層感知器訓練時加入權重雜訊和權重衰減的研究

Empirical studies on the online learning algorithms based on combining weight noise injection and weight decay

Advisor: Dr. John Sum
Student: Yen-Lun Liang

中華民國99年05月

國立中興大學科技管理學研究所

碩 士學位論文

題目：Empirical studies on the online learning algorithm based on combining weight noise injection and weight decay

姓名：梁昇綸　　　　學號：79725302

經　口　試　通　過　特　此　證　明

論文指導教授	[illegible]
論文考試委員	[illegible]
	[illegible]

中華民國 99 年 6 月 1 日

中文摘要

在類神經網路中, 爲了改善容錯效果而在訓練時加入權重雜訊(weight noise) 已經被廣泛的採用, 但是在理論上及實證上皆未獲得證實。在本論文中, 我們將從兩個方面來討論 - 在演法中加入權重雜訊 (weight noise) 和權重衰減 (weight decay)。我們把 multiplicative weight noise 及 additive weight noise 兩種情況分開探討。爲了得到收斂情況和容錯能力的表現, 我們透過大量的電腦模擬來得到所需的結果。

實驗結果顯示:(一) 在訓練時加入權重雜訊 (weight noise) 將不會使權重收斂。(二) 同時加入權重雜訊 (weight noise) 和權重衰減 (weight decay) 的收斂情況較只加入權重雜訊 (weight noise) 來得更好。(三) 同時加入權重雜訊 (weight noise) 和權重衰減 (weight decay) 的容錯能力較只加入權重雜訊 (weight noise) 來得更好。

本論文有以下兩個貢獻: 第一, 這些研究結果的一部分, 補充了在最近由 Ho, Leung & Sum 三人在訓練時加入權重雜訊 (weight noise) 的 收斂情況的研究結果。第二, 另外一部分結果關於容錯的部分在類神經網路裡是一個新的領域。

最後, 本論文也帶出一個在訓練時加入權重衰減 (weight decay) 的重要訊息。加入權重衰減 (weight decay) 不僅可以提高權重的收斂, 也可以改善類神經網路的容錯效果。

關鍵字: 類神經網路, 多層感知器, 容錯, 權重衰減, 權重雜訊

SUMMARY

While injecting weight noise during training have been widely adopted in attaining fault tolerant neural newtorks, theoretical and empirical studies on the online algorithms developed based on these strategies have yet to be complete. In this thesis, we will investigate two important aspects in regard to the online learning algorithms based on combining weight noise injection and weight decay. Multiplicative weight noise and additive weight noise are considered seperately. The convergence behaviors and the performance of those learning algorithms are investigated via intensive computer simulations.

It is found that (i) the online learning algorithm based on purely multiplicative weight noise injection does not converge, (ii) the algorithms combining weight noise injection and weight decay exhibit better convergence behaviors than their pure weight noise injection counterparts, and (iii) the neural networks attained by these algorithms combining weight noise injection and weight decay showing better fault tolerance abilities than the neural networks attained by the pure weight noise injection-based algorithms.

The contributions of these results are two folds. First, part of these empirical results complement the recent findings from Ho, Leung & Sum on the convergence behaviors of the weight noise injection-based learning algorithms. Second, another part of the results which is in regard to the fault tolerance ability are new in the area.

Finally, one should note that the results presented in this thesis also bring out an important message adding weight decay during training. Weight decay is not just can improve the convergence of an algorithm, but also can improve the weight noise tolerance ability of a neural network that is attained by these online algorithms.

Keywords: Neural Networks, Multilayer Perceptron (MLP), Fault Tolerance, Weight Decay, Weight Noise

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Background

In conventional learning theory, neural network are trained to achieve good generalization. Which could be accomplished by adding regularizer [23, 24, 25, 26, 36] or prunning [16, 22, 23, 29, 31], so as to reduce the weights' magnitudes model complexity. These methods work well under the assumption that neural network after training can be ideally implemented (i.e. fault-free implementation). It is ture if a neural network is hard-corded in a program that is running in a computer with very high precision data representation. However, it is not true for electronic implementations, like FPGA [17]. Component failure, sign bit change, open circuit [30], finite precision [32], and even exposure to radiation [14] could degrade the performance of such an implementation drastically. In such case, the performance of a neural network will be questionable even if it has been trained to achieve very good generalization.

In this regard, many online learning algorithms have been developed throughout the last two decades to improve the tolerance of a neural network against random node fault, stuck-at node fault and weight noise. Some of these include injecting node fault [5, 34] or weight noise [27, 28] during back-propagation training, injecting weight noise during training a recurrent neural network [19] and pulse-coupled neural networks [11], and injecting node noise during training [2]. While controlling the weight magnitude is able to reduce the output sensitivity, some researchers applied weight decay training [12], and others developed online training algorithms based on back-propagation training and bounding weight magnitude [7, 13, 15].

Apart from online training algorithms, some researchers proposed synthesizing methods to obtain a fault tolerant neural network. Some of these include adding network redundancy in a well-trained neural network [30], formulating the training as a nonlinear constraint optimization problem [9, 33], and adding regularizer in the objective function and then obtaining a neural network by batch model training [3, 21, 38].

Amongst all, the algorithms based on online back-propagation training with weight noise injection are of least theoretical study, [1, 18, 27, 28]. Murray & Edward in [10, 28] derived the prediction error of a MLP if multiplicative weight noise is injected (see Section II.A and II.B in [28]). Later, G.An [1] presented the objective functions for the online back-propagation training with additive weight noise injection (see Section 4 in [1]), in which the noise is defined as either (i) mean zero Gaussian noise and (ii) mean zero uniform noise.

Here, we should point out that these results [1, 28] are derived entirely from the prediction error point of view. They are not derived by analyzing the update equations of the training algorithms. Their relations with the underlying online training algorithms are yet to be investigated. Even in the same paper (see Section II.C in [28]), they have presented an analysis on the dynamics of the weight vector that is updated based on online back-propagation training with multiplicative weight noise injection, the corresponding objective function has not been presented.

In a recent study, Ho *et al* [18] have pointed out that the derivation of the objective function based on analyzing the prediction error could be misleading. Take radial basis function (RBF) network as an example. By analyzing the weight update equation of the online training algorithm with either multiplicative or additive weight noise injection, Ho *et al* [18] have shown that the true objective function being minimized consists only the mean square error (MSE). Clearly, it is not the same as those derived from the prediction error point of view [3, 37].

In the past research, researches only focused on the convergence of MSE, but not on the convergence of the network weights. *In this thesis, we will point out that the weight might not converge if only weight noise is injected during training.* The work done in the last two decades are questionable. Therefore, to improve the convergence and fault tolerance, we propose combining weight noise injection and weight decay during training, while the weight noise injection is able to improve the fault tolerance of a MLP, the weight decay is able to regulate the weight magnitude.

1.2 Contributions

Following points are what we verified by empirical study in this thesis:

1. Online back-propagation training with combining multiplicative weight noise injection and weight decay could lead to better convergence in weights.

2. Online back-propagation training with combining additive weight noise injection and weight decay could lead to better convergence in weights.

3. Online back-propagation training with combining multiplicative weight noise injection and weight decay could improve fault tolerant ability of a MLP.

4. Online back-propagation training with combining additive weight noise injection and weight decay could improve fault tolerant ability of a MLP.

1.3 Outline of thesis

The aim of the rest of this thesis is to (1) derive the update equations of the algorithms developed based on online back-propagation training with combining weight noise injection and weight decay and (2) use simulations to study the properties of these algorithms. In the Chapter 3, eight different online weight noise injection and weight decay training algorithms will be shown. The simulation results will be presented in Chapter 4. Conclusion will be given in the last chapter. Detail results and the program code for simulation are added in the Appendix.

Chapter 2

Literature Review

To obtain a fault tolerance neural network, many methods have been proposed in the last 17 years. For example, in 1993, Marry and Edwards [27, 28] has modified BPA by injecting weight noise during training for MLP. By simulations, they showed their algorithms can have better converge, and network generated can have better fault tolerance ability. By theoretical analysis, they only find the effect of weight noise on the prediction error of a MLP.

In 1996, Jim, Giles and Horne [19] has modified real time recurrent learning by injecting weight noise during training for recurrent neural network (RNN). By simulation, they showed their algorithms can have better converge, and better convergence. By theoretical analysis, they only find the effect of weight noise on the prediction error of a RNN

Apart from weight noise injection, regularization is another approach. In 2000, Bernier and co-workers [4, 3] has proposed adding explicit regularizer to training MSE as the objective function to be minimized. Their online learning algorithm is developed by the idea of gradient descent, and no noise is injected during training.

In 2009, Ho, Leung and Sum [38] also proposed adding regularizer term to training MSE as the objective function. It is similar to Bernier *et al* approach. But, the weighting factor for the regularizer can be determined by the noise variance. Their online learning is developed by the idea of gradient descent, and no noise is injected during training From 2009, Ho, Leung and Sum find a misconception about convergence in [37, 38]. They show that the work by G. An [1] is incomplete. Essentially, his work is identical to the works done by Murray, Edwards [27, 28] and Bernier *et al* [4, 3]. Only the effect of weight noise on the prediction error of a MLP has been derived. By theoretical analysis, Ho *et al* find injecting weight noise during training a RBF has no use. By simulation, they find only MSE converges but weights might not converge, and injecting weight noise and weight decay during training can improve convergence

Chapter 3

LEARNING ALGORITHMS

In this chapter, the learning algorithms based on weight noise injection will be introduced. In Section 1 and 2, the backpropagation learning algorithm for the linear output multilayer perceptron will be presented. These algorithms are usually applied to approximation and regression problems. In Section 3 and 4, the backpropagation learning algorithm for the sigmoidal output multilayer perceptron will be presented. These algorithms are usually applied to classification problems. Then, the corresponding weight noise injection algorithms are described in Section 5.

Section	Algorithm	Objective Function	Network Structure
2.1	BPA	MSE	Linear output MLP
2.2	BPA w/ WD	MSE+WD	Linear output MLP
2.3	BPA	CEE	Sigmoid output MLP
2.4	BPA w/ WD	CEE + WD	Sigmoid output MLP
2.5.1	WNI - BPA	Unknown	Linear output MLP
2.5.2	WNI - BPA w/ WD	Unknown	Linear output MLP
2.5.3	WNI - BPA	Unknown	Sigmoid output MLP
2.5.4	WNI - BPA w/ WD	Unknown	Sigmoid output MLP

MSE: Mean Square Errors, CEE: Cross Entropy Errors

Table 3.1: Algorithms to be defined in this chapter.

3.1 Back propagation algorithm 1 (BPA.1)

Fig. 3.1 shows the multilayer perceptron (MLP) structure, we assume that there are m input nodes, n hidden nodes,and 1 linear output node.

Given a set of data, $D = \{\underline{x}_k, y_k\}_{k=1}^{N}$, $\underline{x}_k = \begin{pmatrix} x_{k1} \\ \vdots \\ x_{km} \end{pmatrix}$, we have

$$f\left(\mathbf{x}, \underline{\mathbf{w}}\right) = w_0 + \sum_{j=1}^{n} w_j h_j \left(\mathbf{x}, \mathbf{v}_j, v_{j0}\right), \tag{3.1}$$

where $\underline{\mathbf{w}}$ in Equation (3.1) is given by

$$\underline{\mathbf{w}} = \left(w_0, w_1, \cdots, w_n, v_{10}, \mathbf{v}_1, \cdots, v_{n0}, \mathbf{v}_n\right)^T,$$

and h_j is the output of hidden node

$$h_j\left(\underline{x}, \underline{v}_j, \underline{v}_{j0}\right) = \frac{1}{1 + \exp\left(-\left(\underline{v}_j^T \underline{x} + \underline{v}_{j0}\right)\right)}, \tag{3.2}$$

for $j = 1, 2, ..., n$.

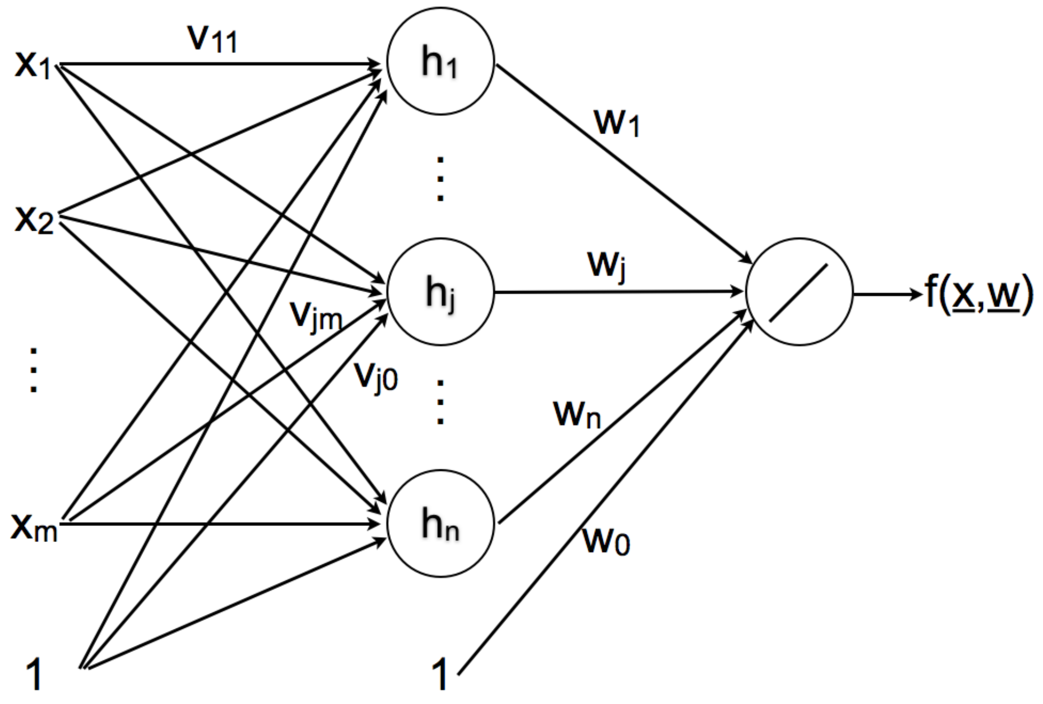

Fig. 3.1: MLP (1).

3.1.1 Objective function

Given N data, $(x_1, y_1), (x_2, y_2), \cdots, (x_{N,} y_N)$, the objective function is defined as follows:

$$E(\underline{\mathbf{w}}) = \frac{1}{N} \sum_{k=1}^{N} (y_k - f(\underline{x}_k, \underline{\mathbf{w}}))^2 . \tag{3.3}$$

3.1.2 Update equations

After we defined the objective function, we can derive the update equations from objective function. The weight vector can then be obtained by the following gradient descent algorithm.

$$\underline{\mathbf{w}}\,(t+1) = \underline{\mathbf{w}}\,(t) - \mu \frac{\partial E\,(\underline{\mathbf{w}}\,(t))}{\partial \underline{\mathbf{w}}}, \tag{3.4}$$

where

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial \underline{\mathbf{w}}} = \left[\frac{\partial E(\underline{\mathbf{w}})}{\partial w_0}, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_1}, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_n}, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{10}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_1}}^T, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{n0}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_n}}^T \right]^T,$$

$\underline{\mathbf{w}}\,(0)$ is initialized as a random vector close to $\mathbf{0}$, and μ is the step size.

In Equation 3.4,

$$\begin{aligned}
\frac{\partial E\,(\underline{\mathbf{w}})}{\partial w_0} &= \frac{1}{N} \sum_{k=1}^{N} \frac{\partial}{\partial w_0} \left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right)^2 \\
&= \frac{1}{N} \sum_{k=1}^{N} 2\left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right) \left(-\frac{\partial f\,(\underline{x}_k, \underline{\mathbf{w}})}{\partial w_0} \right) \\
&= -\frac{2}{N} \sum_{k=1}^{N} \left(y_k - f\,(\underline{x}_k \underline{\mathbf{w}})\right) \left(\frac{\partial f\,(\underline{x}_k, \underline{\mathbf{w}})}{\partial w_0} \right) \\
&= -\frac{2}{N} \sum_{k=1}^{N} \left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right). \qquad (3.5)
\end{aligned}$$

For $j = 1, 2, ..., n$,

$$\begin{aligned}
\frac{\partial E\,(\underline{\mathbf{w}})}{\partial w_j} &= \frac{1}{N} \sum_{k=1}^{N} \frac{\partial}{\partial w_j} \left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right)^2 \\
&= \frac{1}{N} \sum_{k=1}^{N} 2\left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right) \left(-\frac{\partial f\,(\underline{x}_k, \underline{\mathbf{w}})}{\partial w_j} \right) \\
&= -\frac{2}{N} \sum_{k=1}^{N} \left(y_k - f\,(\underline{x}_k, \underline{\mathbf{w}})\right) \left(h_j\left(\underline{x}_k, \underline{v}_j, v_{j0}\right)\right). \qquad (3.6)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{j0}} &= \frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))(\frac{-\partial f(\underline{x}_k, \underline{\mathbf{w}})}{\partial v_{j0}}) \\
&= -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))w_j \frac{\partial h_j(\underline{x}_k, \underline{v}_j, \underline{v}_{j0})}{\partial v_{j0}} \\
&= -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))w_j h_j(1-h_j). \qquad (3.7)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{ji}} &= \frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))(\frac{-\partial f(\underline{x}_k, \underline{\mathbf{w}})}{\partial v_{ji}}) \\
&= -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))w_j \frac{\partial h_j(\underline{x}_k, \underline{v}_j, \underline{v}_{j0})}{\partial v_{ji}} \\
&= -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))w_j h_j(1-h_j)x_{ki}. \qquad (3.8)
\end{aligned}$$

3.2 BPA.1 with weight decay

For online weight noise injection with weight decay, the objective function are similar except that a weight decay term is added.

3.2.1 Objective function

Similarly, given N data, $(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$, the objective function is defined as follows:

$$E(\underline{\mathbf{w}}) = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))^2 + \alpha \underline{\mathbf{w}}^T \underline{\mathbf{w}}. \qquad (3.9)$$

3.2.2 Update equations

After we defined the objective function, we can derive the update equations from objective function. The weight vector can then be obtained by the following gradient descent algorithm.

$$\underline{\mathbf{w}}\left(t+1\right) = \underline{\mathbf{w}}\left(t\right) - \mu \frac{\partial E\left(\underline{\mathbf{w}}\left(t\right)\right)}{\partial \underline{\mathbf{w}}}, \tag{3.10}$$

where

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial \underline{\mathbf{w}}} = \left[\frac{\partial E(\underline{\mathbf{w}})}{\partial w_0}, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_1}, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_n}, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{10}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_1}}^T, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{n0}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_n}}^T\right]^T,$$

$\underline{\mathbf{w}}\left(0\right)$ is initialized as a random vector close to $\mathbf{0}$, and μ is the step size.

In Equation 3.10,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_0} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right) + 2\alpha w_0. \tag{3.11}$$

For $j = 1, 2, ..., n$,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right)\left(h_j\left(\underline{x}_k, \underline{v}_j, v_{j0}\right)\right) + 2\alpha w_j. \tag{3.12}$$

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{j0}} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right) w_j h_j \left(1 - h_j\right) + 2\alpha v_{j0}. \tag{3.13}$$

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{ji}} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right) w_j h_j \left(1 - h_j\right) x_{ki} + 2\alpha v_{ji}. \tag{3.14}$$

3.3 BPA.2

Fig. 3.2 is similar to Fig. 3.1 except that the output node is sigmoid.

Given a set of data, $D = \{\underline{x}_k, y_k\}_{k=1}^{N}$, $\underline{x}_k = \begin{pmatrix} x_{k1} \\ \vdots \\ x_{km} \end{pmatrix}$, we have

$$f(\mathbf{x}, \underline{\mathbf{w}}) = \frac{1}{1 + exp(-g(\mathbf{x}, \underline{\mathbf{w}}))}, \tag{3.15}$$

where $g(\mathbf{x}, \underline{\mathbf{w}})$ in Equation (3.15) is given by

$$g(\mathbf{x}, \underline{\mathbf{w}}) = w_0 + \sum_{j=1}^{n} w_j h_j \left(\mathbf{x}, \mathbf{v}_j, v_{j0}\right), \tag{3.16}$$

where $\underline{\mathbf{w}}$ in Equation (3.15,3.16) is given by

$$\underline{\mathbf{w}} = \left(w_0, w_1, \cdots, w_n, v_{10}, \mathbf{v}_1, \cdots, v_{n0}, \mathbf{v}_n\right)^T,$$

and h_j is the output of hidden node

$$h_j\left(\underline{x}, \underline{v}_j, \underline{v}_{j0}\right) = \frac{1}{1 + \exp\left(-\left(\underline{v}_j^T \underline{x} + \underline{v}_{j0}\right)\right)}, \tag{3.17}$$

for $j = 1, 2, ..., n$.

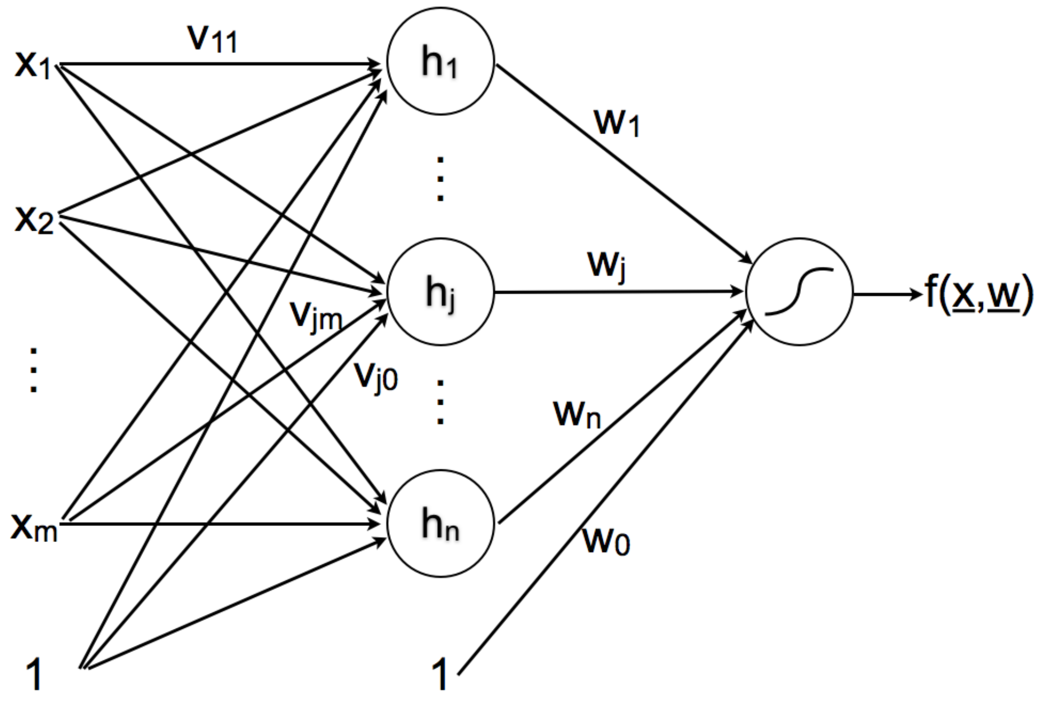

Fig. 3.2: MLP (2).

3.3.1 Objective function

Similarly, given N data, $(x_1, y_1), (x_2, y_2), \cdots, (x_{N,} y_N)$, the objective function is defined as follows:

$$E(\underline{\mathbf{w}}) = \frac{1}{N}\sum_{k=1}^{N}\left\{y_k \ln f(\underline{x}_k, \underline{\mathbf{w}}) + (1 - y_k)\ln(1 - f(\underline{x}_k, \underline{\mathbf{w}}))\right\}. \tag{3.18}$$

3.3.2 Update equations

After we defined the objective function, we can derive the update equations from objective function. The weight vector can then be obtained by the following gradient descent algorithm.

$$\underline{\mathbf{w}}\,(t+1) = \underline{\mathbf{w}}\,(t) - \mu \frac{\partial E\left(\underline{\mathbf{w}}\,(t)\right)}{\partial \underline{\mathbf{w}}}, \tag{3.19}$$

where

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial \underline{\mathbf{w}}} = \left[\frac{\partial E(\underline{\mathbf{w}})}{\partial w_0}, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_1}, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial w_n}, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{10}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_1}}^T, \cdots, \frac{\partial E(\underline{\mathbf{w}})}{\partial v_{n0}}, \frac{\partial E(\underline{\mathbf{w}})}{\partial \mathbf{v_n}}^T \right]^T,$$

$\underline{\mathbf{w}}\,(0)$ is initialized as a random vector close to $\mathbf{0}$, and μ is the step size.

In Equation (3.19),

$$\begin{aligned}
\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_0} &= \frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial f(\underline{x}_k, \underline{\mathbf{w}})} \frac{\partial f(\underline{x}_k, \underline{\mathbf{w}})}{\partial w_0} \\
&= \frac{1}{N} \sum_{k=1}^{N} \left\{ y_k \frac{1}{f(\underline{x}_k, \underline{\mathbf{w}})} + (1 - y_k) \frac{-1}{1 - f(\underline{x}_k, \underline{\mathbf{w}})} \right\} \cdot f(\underline{x}_k, \underline{\mathbf{w}}) \left[1 - f(\underline{x}_k, \underline{\mathbf{w}})\right] \\
&= \frac{1}{N} \sum_{k=1}^{N} \left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right). \qquad (3.20)
\end{aligned}$$

For $j = 1, 2, ..., n$,

$$\begin{aligned}
\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j} &= \frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial f(\underline{x}_k, \underline{\mathbf{w}})} \frac{\partial f(\underline{x}_k, \underline{\mathbf{w}})}{\partial w_j} \\
&= \frac{1}{N} \sum_{k=1}^{N} \left\{ y_k \frac{1}{f(\underline{x}_k, \underline{\mathbf{w}})} + (1 - y_k) \frac{-1}{1 - f(\underline{x}_k, \underline{\mathbf{w}})} \right\} \cdot h_j \cdot f(\underline{x}_k, \underline{\mathbf{w}}) \left[1 - f(\underline{x}_k, \underline{\mathbf{w}})\right] \\
&= \frac{1}{N} \sum_{k=1}^{N} \left(y_k - f\left(\underline{x}_k, \underline{\mathbf{w}}\right)\right) h_j. \qquad (3.21)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{j0}} &= \frac{\partial E(\underline{\mathbf{w}})}{\partial f(\underline{x}_k,\underline{\mathbf{w}})}\frac{\partial f(\underline{x}_k,\underline{\mathbf{w}})}{\partial v_{j0}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\left\{y_k\frac{1}{f(\underline{x}_k,\underline{\mathbf{w}})}+(1-y_k)\frac{-1}{1-f(\underline{x}_k,\underline{\mathbf{w}})}\right\} \\
&\quad \cdot w_j\cdot h_j(1-h_j)\cdot f(\underline{x}_k,\underline{\mathbf{w}})\left[1-f(\underline{x}_k,\underline{\mathbf{w}})\right] \\
&= \frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)w_jh_j\left(1-h_j\right). \qquad (3.22)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{ji}} &= \frac{\partial E(\underline{\mathbf{w}})}{\partial f(\underline{x}_k,\underline{\mathbf{w}})}\frac{\partial f(\underline{x}_k,\underline{\mathbf{w}})}{\partial v_{ji}} \\
&= \frac{1}{N}\sum_{k=1}^{N}\left\{y_k\frac{1}{f(\underline{x}_k,\underline{\mathbf{w}})}+(1-y_k)\frac{-1}{1-f(\underline{x}_k,\underline{\mathbf{w}})}\right\} \\
&\quad \cdot w_j\cdot h_j(1-h_j)\cdot f(\underline{x}_k,\underline{\mathbf{w}})\left[1-f(\underline{x}_k,\underline{\mathbf{w}})\right]\cdot x_{ki} \\
&= \frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)w_jh_j\left(1-h_j\right)x_{ki}. \qquad (3.23)
\end{aligned}$$

3.4 BPA.2 with weight decay

For online weight noise injection with weight decay, the objective function are similar except that a weight decay term is added.

3.4.1 Objective function

Similarly, given N data, $(x_1,y_1),(x_2,y_2),\cdots,(x_{N,}y_N)$, the objective function is defined as follows:

$$E(\underline{\mathbf{w}}) = \frac{1}{N}\left\{y_k\ln f(\underline{x}_k,\underline{\mathbf{w}})+(1-y_k)\ln(1-f(\underline{x}_k,\underline{\mathbf{w}}))\right\}+\alpha\underline{\mathbf{w}}^T\underline{\mathbf{w}}. \qquad (3.24)$$

3.4.2 Update equations

After we defined the objective function, we can derive the update equations from objective function. The weight vector can then be obtained by the following gradient descent algorithm.

$$\underline{\mathbf{w}}\left(t+1\right)=\underline{\mathbf{w}}\left(t\right)-\mu\frac{\partial E\left(\underline{\mathbf{w}}\left(t\right)\right)}{\partial\underline{\mathbf{w}}}, \tag{3.25}$$

where

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial\underline{\mathbf{w}}}=\left[\frac{\partial E(\underline{\mathbf{w}})}{\partial w_0},\frac{\partial E(\underline{\mathbf{w}})}{\partial w_1},\cdots,\frac{\partial E(\underline{\mathbf{w}})}{\partial w_n},\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{10}},\frac{\partial E(\underline{\mathbf{w}})}{\partial\mathbf{v_1}}^T,\cdots,\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{n0}},\frac{\partial E(\underline{\mathbf{w}})}{\partial\mathbf{v_n}}^T\right]^T,$$

$\underline{\mathbf{w}}\left(0\right)$ is initialized as a random vector close to $\mathbf{0}$, and μ is the step size.

In Equation (3.25),

$$\frac{\partial E}{\partial w_0}=\frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)+2\alpha w_0. \tag{3.26}$$

For $j=1,2,...,n$,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j}=\frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)h_j+2\alpha w_j. \tag{3.27}$$

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{j0}}=\frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)w_jh_j\left(1-h_j\right)+2\alpha v_{j0}. \tag{3.28}$$

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{ji}}=\frac{1}{N}\sum_{k=1}^{N}\left(y_k-f\left(\underline{x}_k,\underline{\mathbf{w}}\right)\right)w_jh_j\left(1-h_j\right)x_{ki}+2\alpha v_{ji}. \tag{3.29}$$

3.5 Weight noise injection training algorithm

From online weight noise injection, the update equations are similar except that a weight noise term is added.

$$\underline{\mathbf{w}}\,(t+1) = \underline{\mathbf{w}}\,(t) - \mu \frac{\partial E\left(\underline{\mathbf{w}}\,(t)\right)}{\partial \underline{\mathbf{w}}}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)}. \tag{3.30}$$

For multiplicative weight noise injection, $\underline{\tilde{\mathbf{w}}}$ in Equation (3.30) is given by

$$\underline{\tilde{\mathbf{w}}}(t) = \underline{\mathbf{w}}(t) + \mathbf{b}(t) \otimes \underline{\mathbf{w}}(t). \tag{3.31}$$

For additive weight noise injection, $\underline{\tilde{\mathbf{w}}}$ in Equation (3.30) is given by

$$\underline{\tilde{\mathbf{w}}}(t) = \underline{\mathbf{w}}(t) + \mathbf{b}(t). \tag{3.32}$$

In Equation (3.31),

$$\mathbf{b}(t) \otimes \underline{\mathbf{w}}(t) = (b_1(t)w_1(t), b_2(t)w_2(t), \cdots, b_M(t)w_M(t))^T, \tag{3.33}$$

where $b_i(t)$ is a mean zero Gaussian distribution with variance denoted by S_b.

3.5.1 Weight noise injection for BPA.1

From Equation (3.5), (3.6), (3.7) and (3.8), we have

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_0}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} \quad = \quad -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right). \tag{3.34}$$

For $j = 1, 2, ..., n$,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \tilde{\underline{\mathbf{w}}}\right)\right)\left(h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)\right). \quad (3.35)$$

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{j0}}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \underline{\mathbf{w}}))w_j\tilde{h}_j(1-\tilde{h}_j), \quad (3.36)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{ji}}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}}))w_j\tilde{h}_j(1-\tilde{h}_j)x_{ki}, \quad (3.37)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

3.5.2 Weight noise injection for BPA.1 with weight decay

Similarly, from Equation (3.11), (3.12), (3.13) and (3.14), we have

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_0}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \tilde{\underline{\mathbf{w}}}\right)\right) + 2\alpha w_0. \quad (3.38)$$

For $j = 1, 2, ..., n$,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \tilde{\underline{\mathbf{w}}}\right)\right)\left(h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)\right) + 2\alpha w_j. \quad (3.39)$$

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{j0}}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}}))\, w_j \tilde{h}_j \left(1-\tilde{h}_j\right) + 2\alpha v_{j0}, \quad (3.40)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{ji}}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = -\frac{2}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}}))\, w_j \tilde{h}_j \left(1-\tilde{h}_j\right) x_{ki} + 2\alpha v_{ji}, \quad (3.41)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

3.5.3 Weight noise injection for BPA.2

Similarly, from Equation (3.20), (3.21), (3.22) and 3.23, we have

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial w_0}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}})). \quad (3.42)$$

For $j = 1, 2, ..., n$,

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial w_j}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}}))\, \tilde{h}_j, \quad (3.43)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E(\underline{\mathbf{w}})}{\partial v_{j0}}|_{\underline{\mathbf{w}}(t)=\tilde{\underline{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}(y_k - f(\underline{x}_k, \tilde{\underline{\mathbf{w}}}))\, w_j \tilde{h}_j \left(1-\tilde{h}_j\right), \quad (3.44)$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \tilde{\underline{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{ji}}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right) w_j \tilde{h}_j \left(1-\tilde{h}_j\right) x_{ki}, \tag{3.45}$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \underline{\tilde{v}}_j, \tilde{v}_{j0}\right)$.

3.5.4 Weight noise injection for BPA.2 with weight decay

Similarly, from Equation (3.26), (3.27), (3.28) and (3.29), we have

$$\frac{\partial E}{\partial w_0}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right) + 2\alpha w_0. \tag{3.46}$$

For $j = 1, 2, ..., n$,

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial w_j}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right) \tilde{h}_j + 2\alpha w_j, \tag{3.47}$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \underline{\tilde{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{j0}}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right) w_j \tilde{h}_j \left(1-\tilde{h}_j\right) + 2\alpha v_{j0}, \tag{3.48}$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \underline{\tilde{v}}_j, \tilde{v}_{j0}\right)$.

$$\frac{\partial E\left(\underline{\mathbf{w}}\right)}{\partial v_{ji}}|_{\underline{\mathbf{w}}(t)=\underline{\tilde{\mathbf{w}}}(t)} = \frac{1}{N}\sum_{k=1}^{N}\left(y_k - f\left(\underline{x}_k, \underline{\tilde{\mathbf{w}}}\right)\right) w_j \tilde{h}_j \left(1-\tilde{h}_j\right) x_{ki} + 2\alpha v_{ji}, \tag{3.49}$$

where $\tilde{h}_j = h_j\left(\underline{x}_k, \underline{\tilde{v}}_j, \tilde{v}_{j0}\right)$.

Chapter 4

EXPERIMENTS

4.1 Date sets

In this chapter, simulation results based on the algorithms presented in Chapter 3 will be elucidated. Six datasets are used to examine the convergence properties of the algorithms and the performance of the neural networks being generated by these algorithms. The datasets include (i) 2D mapping, (ii) Mackey-Glass, (iii) NAR, (iv) Astrophysical data, (v) XOR, (vi) Character recognition. In which four of them are for regression and two of them are for classification. A summary of the properties of the datasets are depicted in Table 4.1.

	Training data	Testing data
2D mapping	100	100
Mackey-Glass	500	500
NAR	500	500
Astrophysical	309	309
XOR	100	100
Character recognition	800	793

Table 4.1: Data sets.

2D mapping:

It is as artificial dataset consisting of 200 data points. Let x_{k1}, x_{k2} be the input and y_k be the target output.

$$y_k = \sin(x_{k1})\sin(x_{k2}) + e_k,$$

where e_k is a mean zero Gaussian noise with variance 0.01. Amongst these 200 data points, 100 of them are randomly selected as the training dataset, and the other 100 data points are the testing dataset. Fig. 4.1 shows the data set.

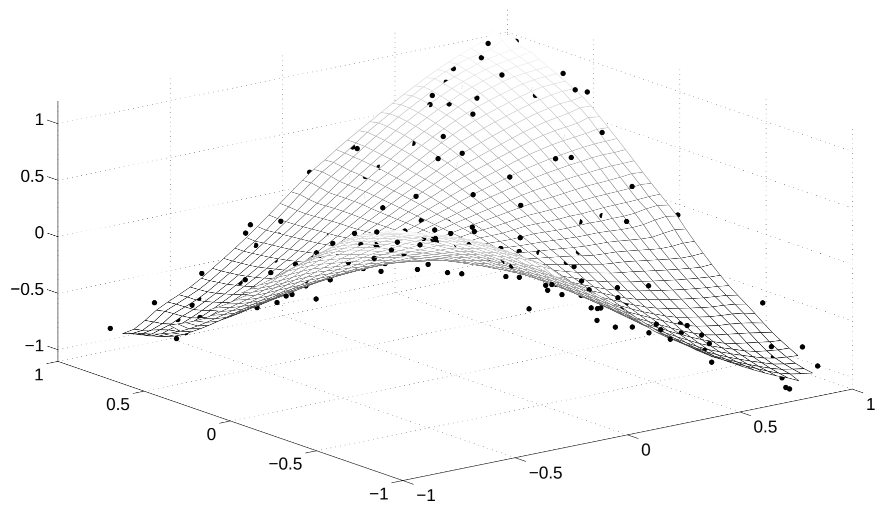

Fig. 4.1: Data set of 2D mapping.

Mackey-Glass:

It is a benchmark time-series dataset available on the Internet. The data is generated by the differential equation

$$dx(t)/dt = 0.2x(t-\tau)/(1 + x(t-\tau)^{10}) - 0.1x(t),$$

with $x(0) = 1.2$ and $\tau = 17$. There are 1000 points in the time series, shown in Fig. 4.2. The task is to train a network to predict the current value $y(k)$ based on the past observation $y(k-1), y(k-2), y(k-3), y(k-4)$. In other words, $x_k = [y(k-1), y(k-2), y(k-3), y(k-4)]^T$. In this dataset, we picked the first 500 points (from $k = 5$ to 504) as the training set and remanding the points are the testing set.

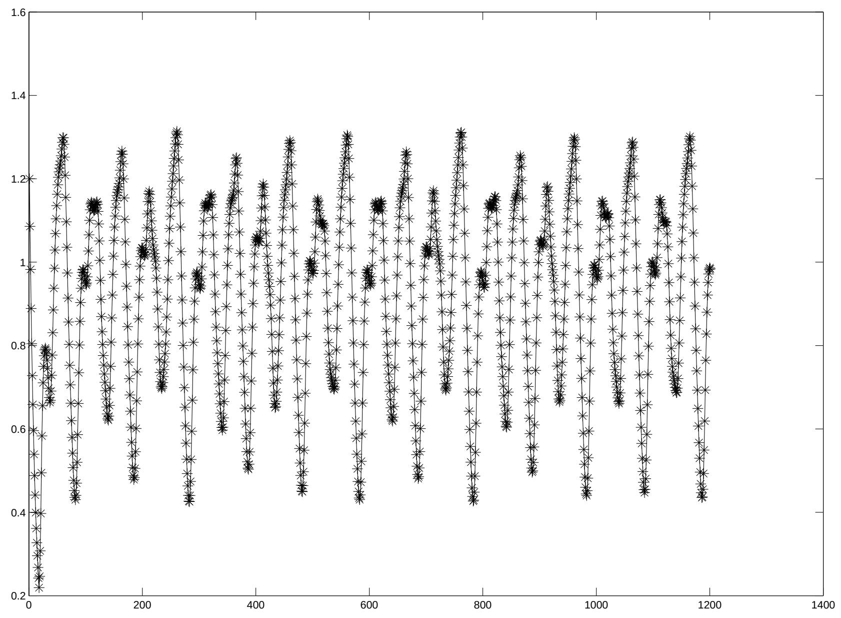

Fig. 4.2: Data set of Mackey-Glass.

NAR:

We consider the following nonlinear autoregressive time series [8, 21], given by

$$\begin{aligned} y(k) &= (0.8 - 0.5\exp(-y^2(y-k)))y(i-1) - (0.3 + 0.9\exp(-y^2(i-1)))y(i-2) \\ &\quad +0.1\sin(\pi y(i-1)) + e(i), \end{aligned}$$

where $e(i)$ is a mean zero Gaussian random variable with variance equals to 0.09. One thousand samples were generated given $y(0) = y(1) = 0.1$, as shown in Fig. 4.3. The first 500 data points were used for training and the other 500 points were used for testing. The task is used to predict $y(k)$ based on the past observation $y(k-1)$ and $y(k-2)$. In other words, $x_k = [y(k-1), y(k-2)]^T$.

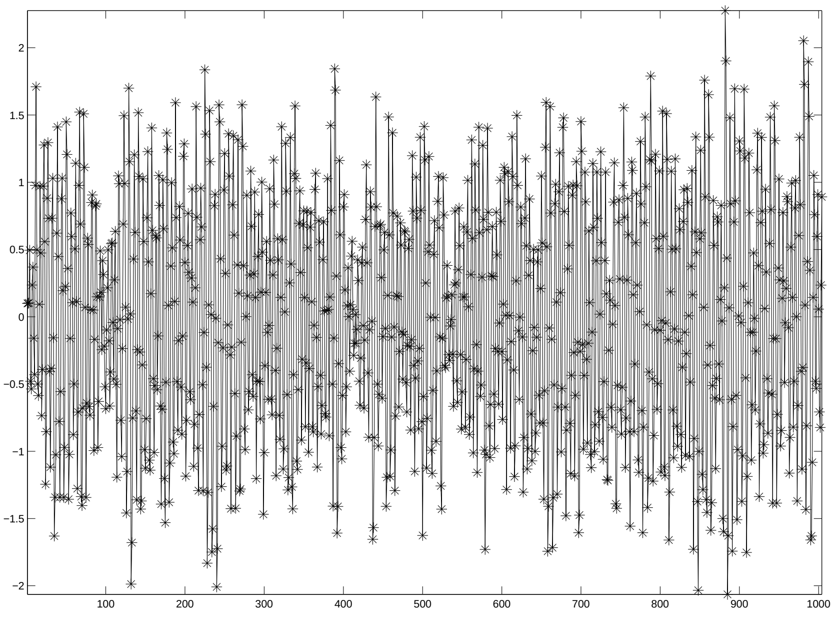

Fig. 4.3: Data set of NAR.

Astrophysical data:

We consider the time-series prediction of a real data set. The series is the time variation of the intensity of the white dwarf star PG1159-035 during March 1989 [20, 21, 35]. The data samples are noisy and nonlinear in nature.[1] Part one of this data set, shown in Fig. 4.4, is selected. There are 618 data samples.

Our task is to train network to predict the current value $y(k)$ based on five past values $y(k-1), ..., y(k-5)$. That means the network has five input and one output. The first 309 pairs are the training data and the remaining pairs are the testing data. In other words, $x_k = [y(k-1), ..., y(k-5)]^T$.

[1]It can be downloaded from http://www-psych.stanford.edu/∼andreas/Time-Series/SantaFe.html

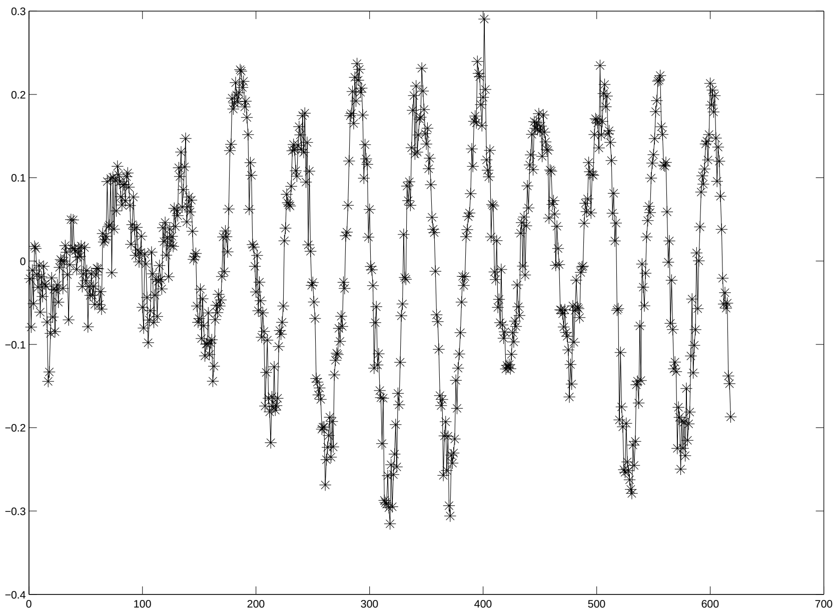

Fig. 4.4: Data set of Astrophysical.

XOR:

It is an artificial data set with two inputs and one output. Fig. 4.5 shows the shape of the function. The black dots in the figure are the data generated. Let x_{k1}, x_{k2} be the input and y_k is the target output.

$$y_k = sign(x_{k1})sign(x_{k1}).$$

Amongst these 200 data points, 100 of them are randomly selected as the training dataset, and the other 100 data points are the testing dataset.

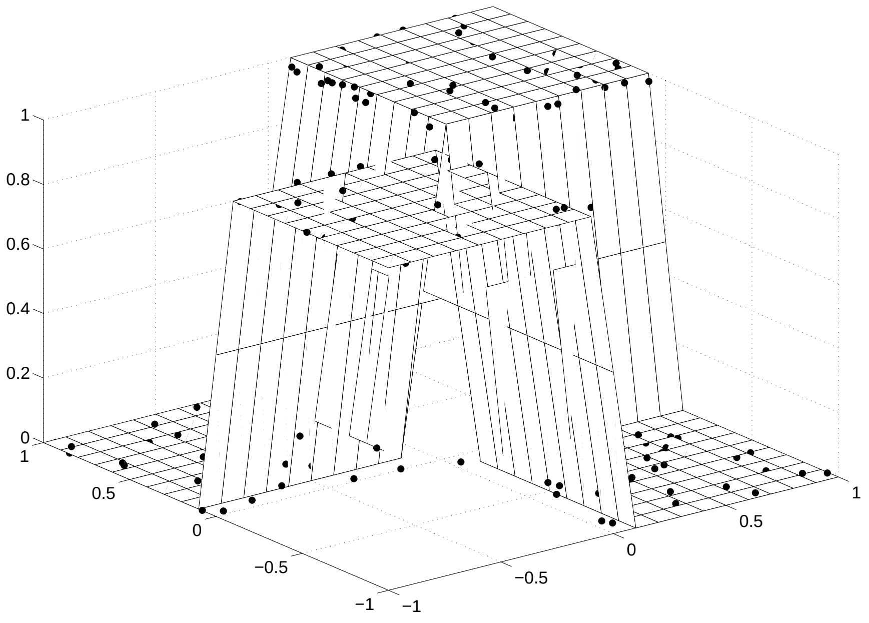

Fig. 4.5: Data set of XOR.

Character recognition:

Character recognition data set [6] [2] consists of 1593 handwritten digits collected from around 80 people. Each person wrote on a paper all the digits from 0 to 9, twice. The digits were scanned and stretched in a 16x16 rectangular box in a gray scale of 256. Then each pixel of each image was scaled into a binary (1/0) value using a fixed threshold. The commitment was to write the digit the first time in the normal way (trying to write each digit accurately) and the second time in a fast way (with no accuracy). Fig. 4.6 shows the sample data set. The 800 data set is as training data and the rest 793 data as testing data.

[2]It can be downloaded from http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit

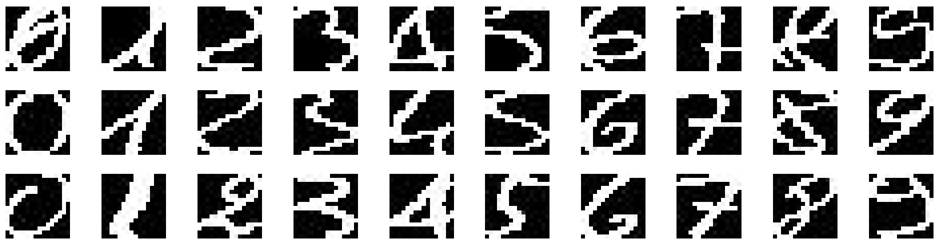

Fig. 4.6: Samples of Character recognition.

4.2 Methodology

Table 4.2 summarizes the structure of the networks for simulations. For the first four simulations, we use linear output MLPs. For the last two simulations, we use sigmoid output MLPs.

	Input nodes	Hidden nodes	Output nodes
2D mapping	2	10 S.N.	1 L.N.
Mackey-Glass	4	10 S.N.	1 L.N.
NAR	2	10 S.N.	1 L.N.
Astrophysical	5	10 S.N.	1 L.N.
XOR	2	6 S.N.	1 S.N.
Characteristic recognition	256	20 S.N.	10 S.N.

* S.N. : Sigmoid node

L.N. : Linear node

Table 4.2: Structures of the networks for simulations.

The hardware and software of computer we use as follows:

CPU	AMD Athlon II X4 630
Ram	4 Gb
OS	Windows XP SP2
Software	Matlab R2008b

Table 4.3: Hardware and software of computer.

In all simulations, we trained the network at least 100000 epochs to make sure the weight and MSE are stable, means they are convergent. If they are not convergent, we still have enough data to observe they are not convergent. Here one epoch is that we trained the network from the first training data and then the second training data until the last training data. When we trained the network, we consider four cases during training: (1) pure online back-propagation, (2) online back-propagation with weight noise injection that contains multiplicative weight noise and additive weight noise, (3) online back-propagation with adding weight decay, (4) online back-propagation with weight noise injection with weight decay . The variance of weight noise are $0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$, and the weight decay are $0, 10^{-5}, 10^{-4}, 10^{-3}$, a total of 20 kinds of combination in training.

Table 4.4 shows the summation of the network setting.

	Train WD	Train Var(WN)	(μ,epoch)	Test Var(WN)
2D mapping	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.1, 10^5)$	[0,0.04]
Mackey-Glass	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.1, 10^5)$	[0,0.04]
NAR	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.1, 10^5)$	[0,0.04]
Astrophysical	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.01, 10^6)$	[0,0.04]
XOR	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.1, 10^5)$	[0,0.04]
C.R.	$0, 10^{-5}, 10^{-4}, 10^{-3}$	$0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}$	$(0.1, 10^5)$	[0,0.04]

Table 4.4: Parameter setting for training and testing-MWN & AWN.

4.3 Results

After training, we exam the MSE and convergence of weights. Appendix B shows all the results. Here we summarizes the major results. We consider two kinds of situations: (1) If we injected multiplicative weight noise during training, we also injected multiplicative weight noise during testing; (2) If we injected additive weight noise during training, we also injected additive weight noise during testing. The variance of S_b in testing are $[0, 0.002, 0.004, ..., 0.04]$, a total of 21 cases, as shown in Table 4.4. For each S_b, we repeated 100 times to get the distribution of the testing errors. The results are shown in Appendix A.

The following sub-sectors show the average performance. Note that the parameter a and S_b in all the MSE plots correspond to weight decay parameter and the noise variance.

For the regression problems, we measure the performance of a MLP by prediction error given by

$$MSE = \frac{1}{N_{test}} \sum_{k=1}^{test} (y_k - f(x_k, \mathbf{w} + \triangle\mathbf{w}))^2, \tag{4.1}$$

where $\triangle\mathbf{w}$ is the weight noise existing of the training. It depends on the value of S_b.

For the classification problems, we measure the performance of a MLP by prediction error given by

$$CE = \frac{1}{N_{test}} \sum_{k=1}^{test} |(y_k - sign(f(x_k, \mathbf{w} + \triangle\mathbf{w}) - 0.5))|, \tag{4.2}$$

where $\triangle\mathbf{w}$ is the weight noise existing of the training. It depends on the value of S_b.

4.3.1 2D mapping

Fault tolerance (Fig 4.7 & Fig 4.8): For MWN, the MLP generated by the algorithm based on combining weight noise injection with weight decay during training gives the best performance. The MLP generated by the algorithm based on pure back-propagation gives the worst performance. For AWN, the MLP generated by the algorithm based on combining weight noise injection with weight decay during training also gives the best performance. The MLP generated by the algorithm based on only weight noise injection gives the worst performance.

Convergence (Fig 4.9 & Fig 4.10): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge. However, when $a = 0$ the weights do not converge.

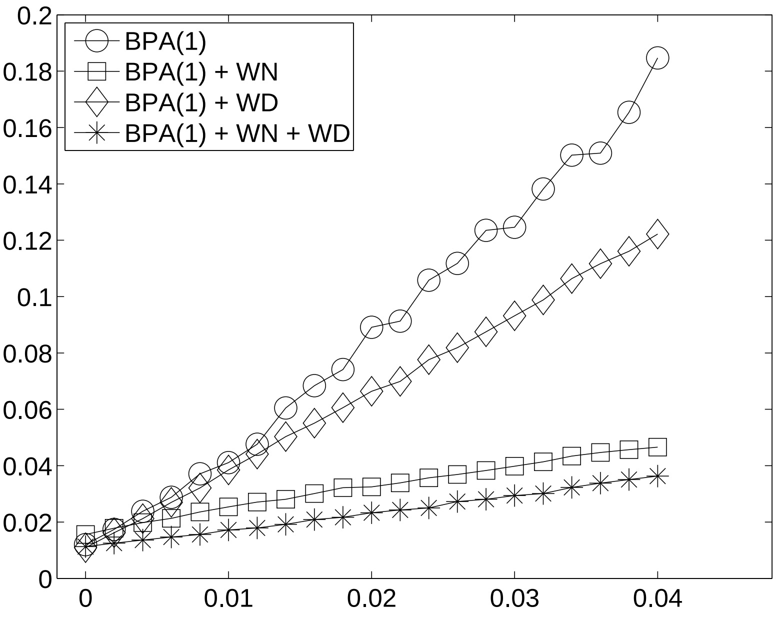

Fig. 4.7: Testing MSE of 2D mapping-MWN.

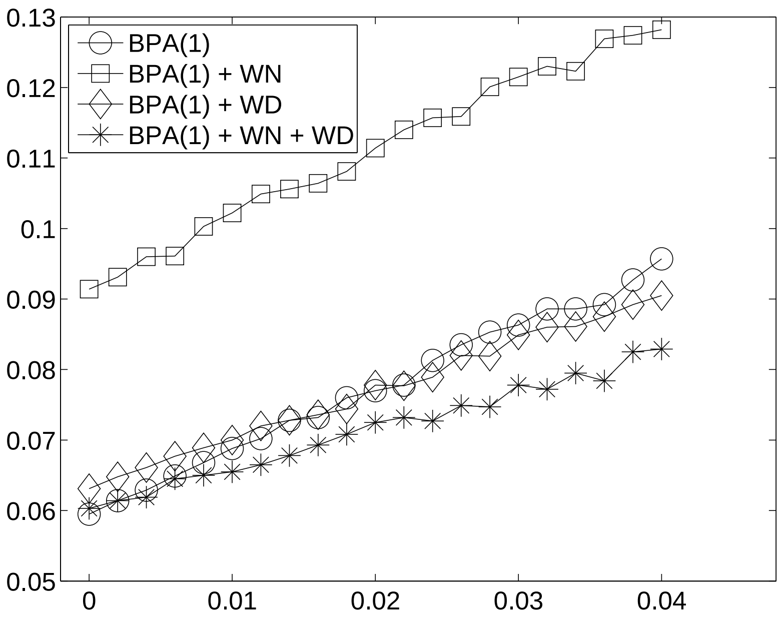

Fig. 4.8: Testing MSE of 2D mapping-AWN.

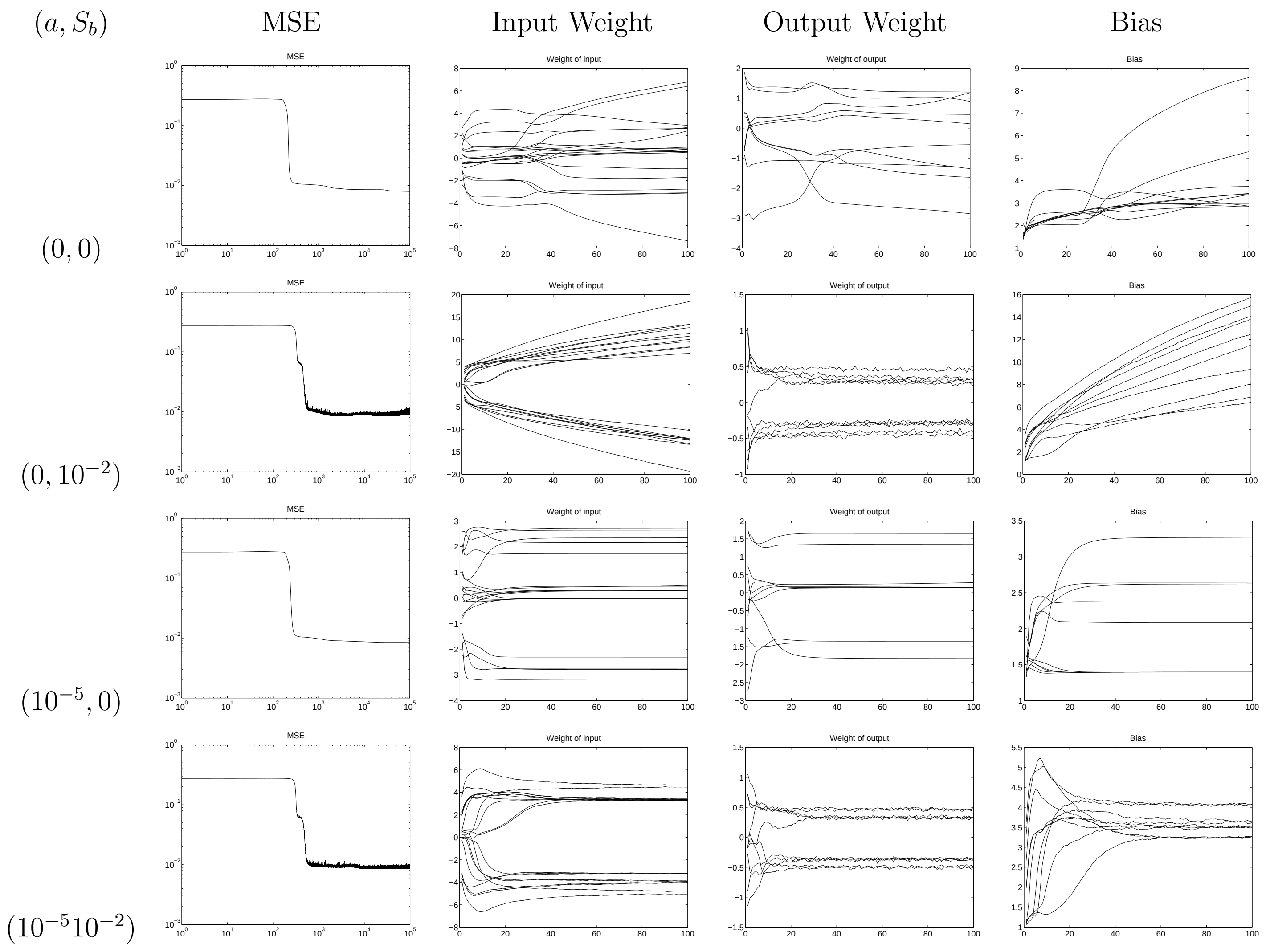

Fig. 4.9: MSE & Weight of 2D mapping-MWN.

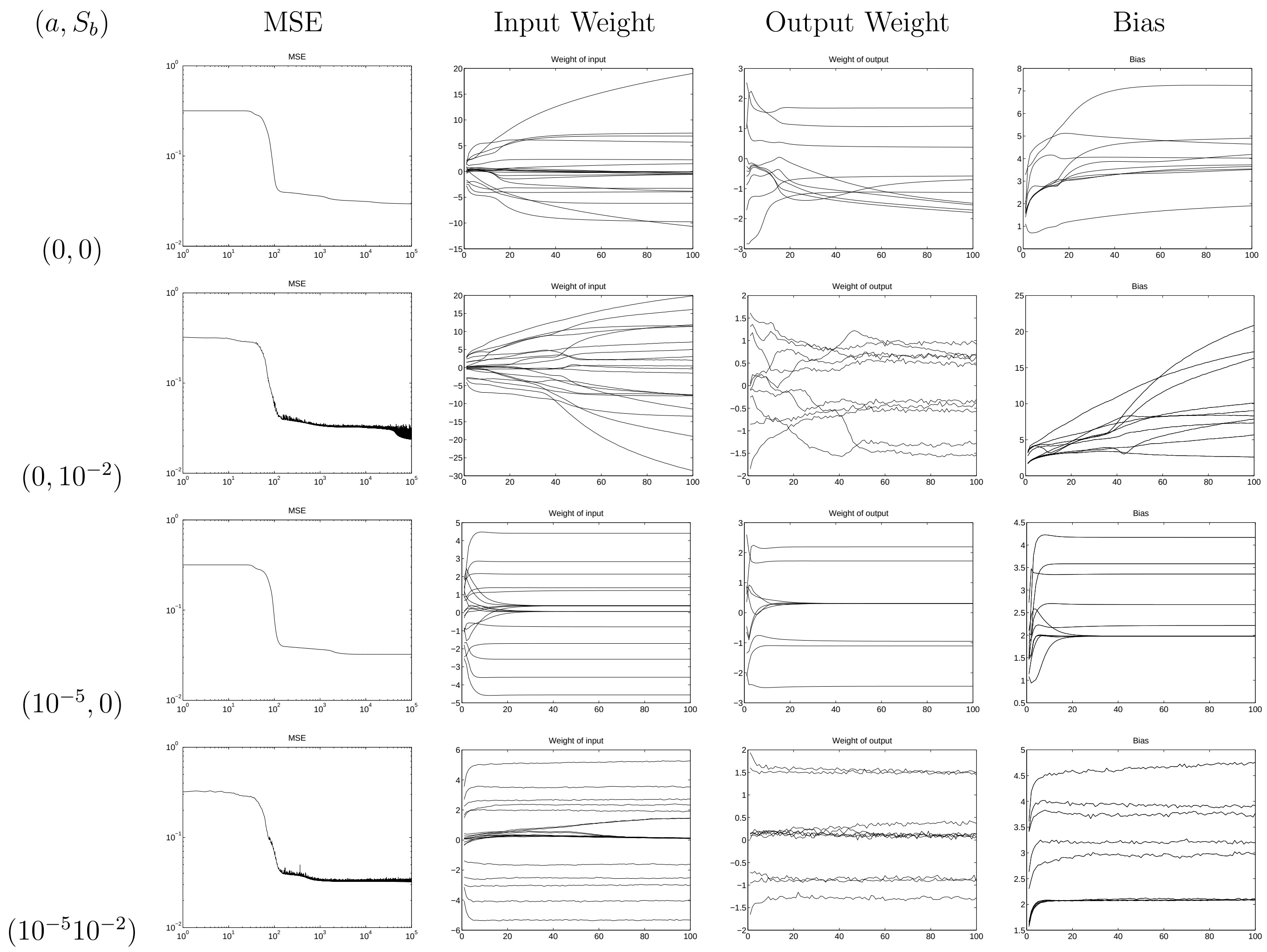

Fig. 4.10: MSE & Weight of 2D mapping-AWN.

4.3.2 Mackey-Glass

Fault tolerance (Fig 4.11 & Fig 4.12): For both cases of MWN and AWN, the MLP generated by the algorithm based on (1) combining weight noise injection with weight decay during training, or (2) only weight noise injection give the best performance. For MWN, the MLP generated by the algorithm based on pure back-propagation gives the worst performance. For AWN, the MLP generated by the algorithm based on only weight noise injection gives the worst performance.

Convergence (Fig 4.13 & Fig 4.14): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge. However, when $a = 0$ the weights do not converge.

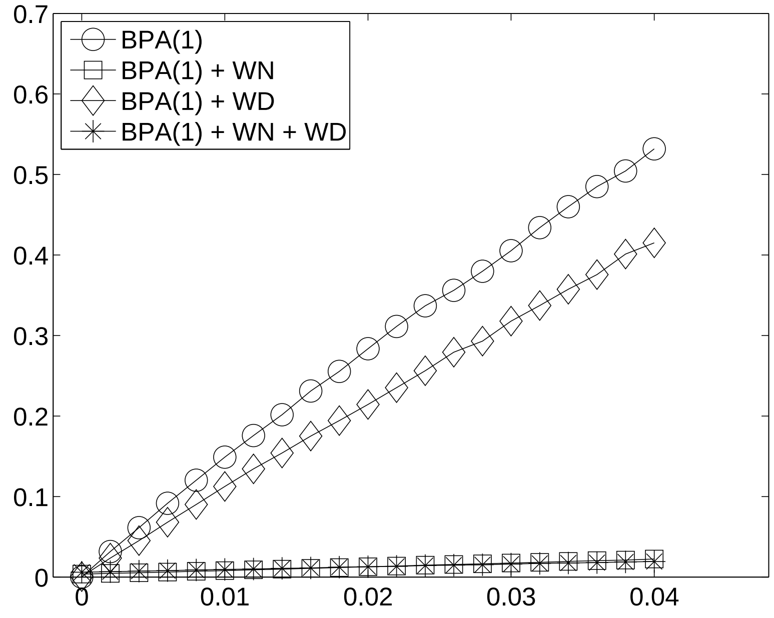

Fig. 4.11: Testing MSE of Mackey-Glass-MWN.

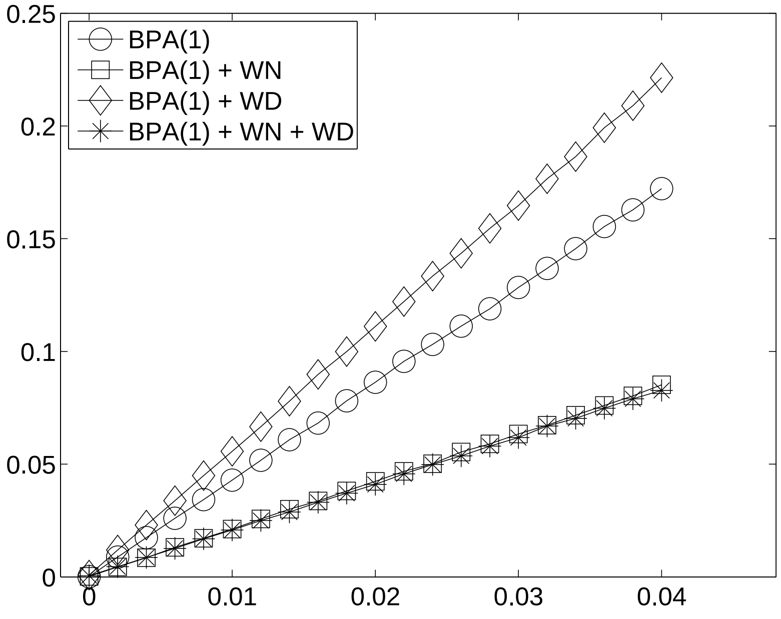

Fig. 4.12: Testing MSE of Mackey-Glass-AWN.

(a, S_b) MSE Input Weight Output Weight Bias

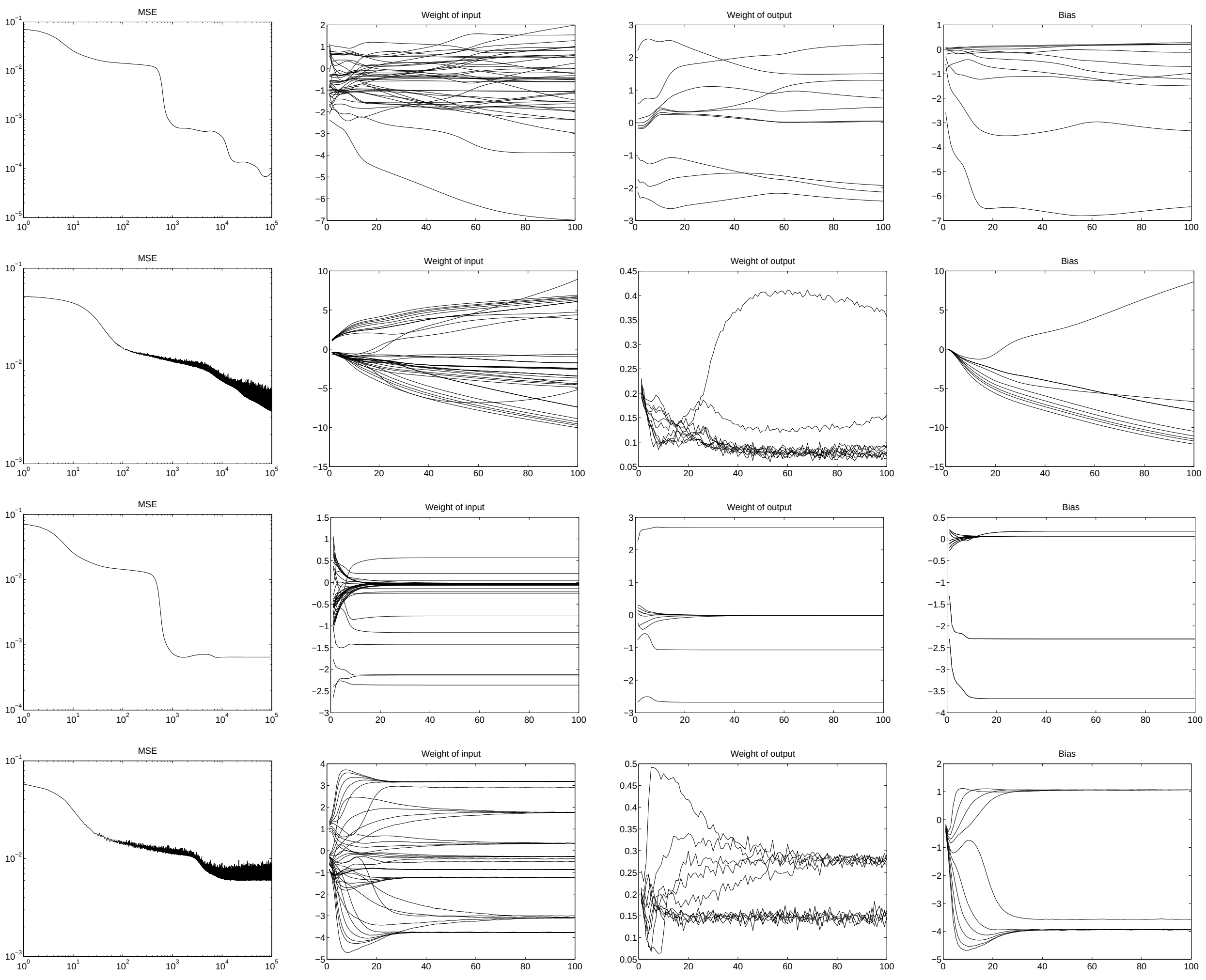

Fig. 4.13: MSE & Weight of Mackey-Glass-MWN.

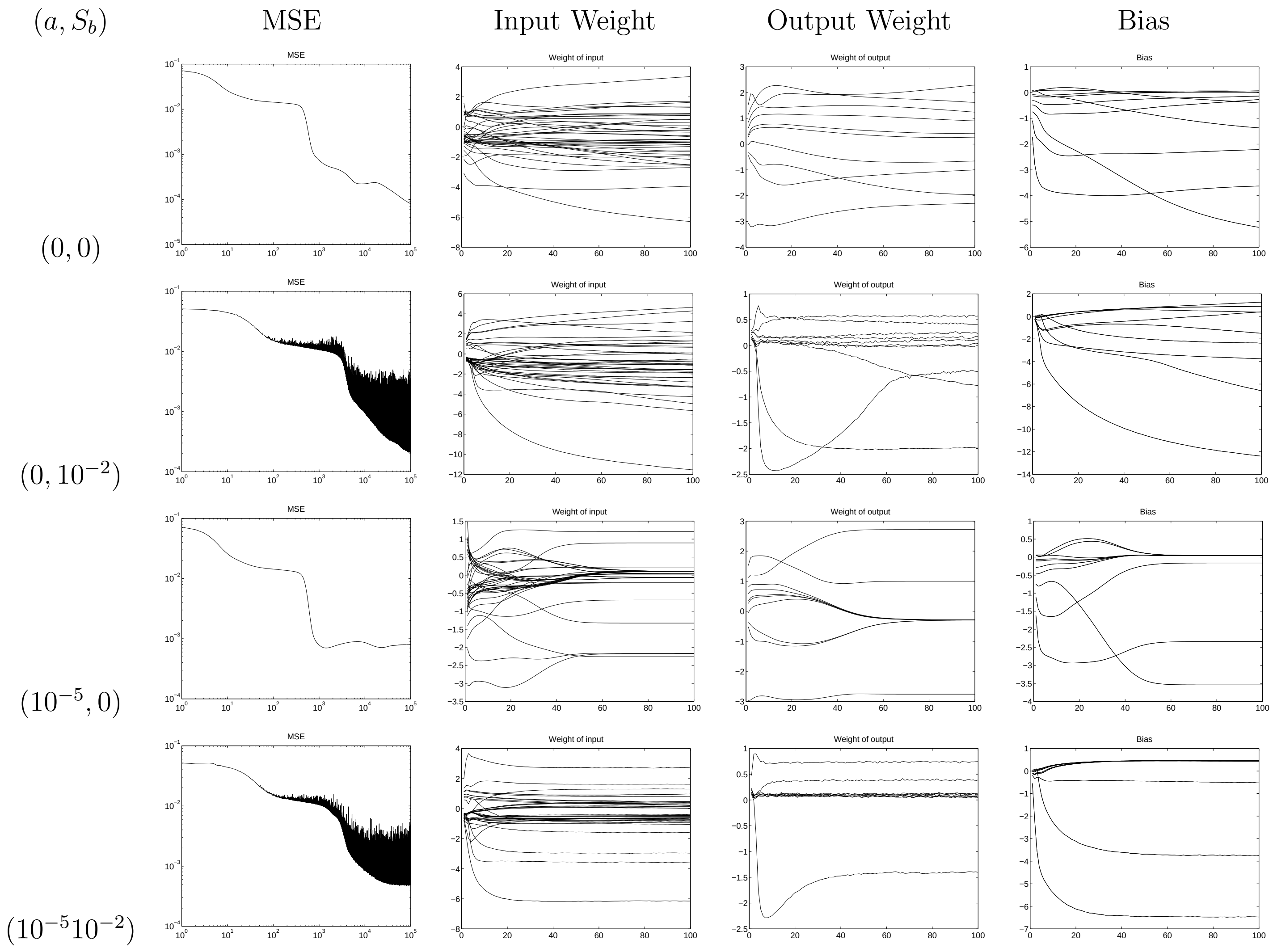

Fig. 4.14: MSE & Weight of Mackey-Glass-AWN.

4.3.3 NAR

Fault tolerance (Fig 4.15 & Fig 4.16): For both cases of MWN and AWN, the MLP generated by the algorithm based on combining weight noise injection with weight decay during training gives the best performance. The MLP generated by the algorithm based on pure back-propagation gives the worst performance.

Convergence (Fig 4.17 & Fig 4.18): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge.

However, when $a = 0$ the weights do not converge.

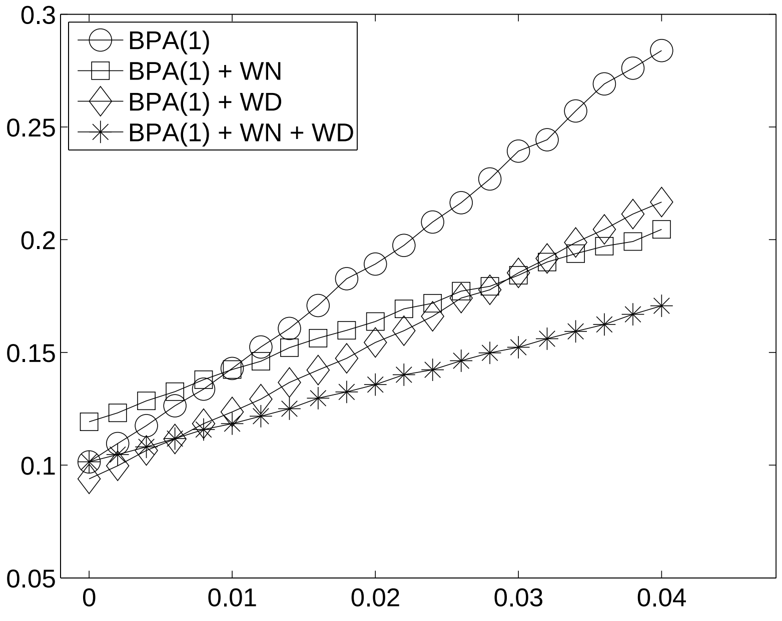

Fig. 4.15: Testing MSE of NAR-MWN.

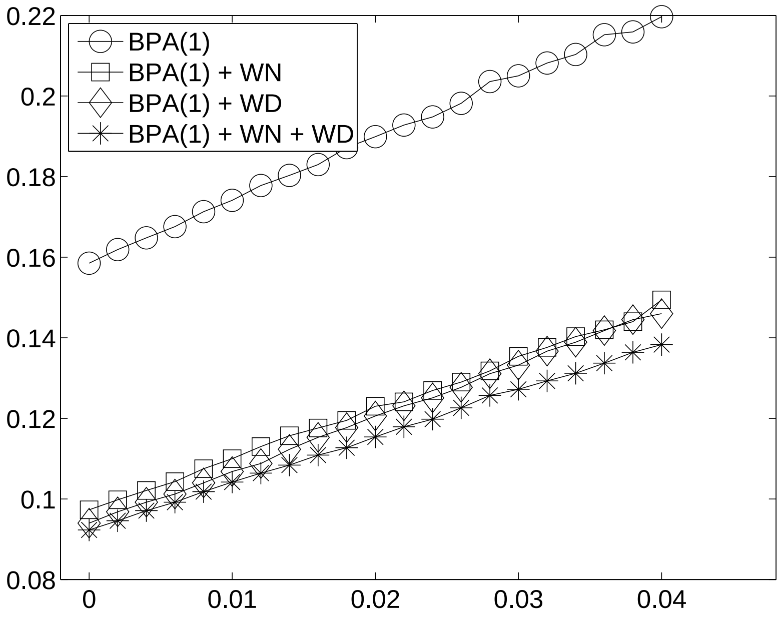

Fig. 4.16: Testing MSE of NAR-AWN.

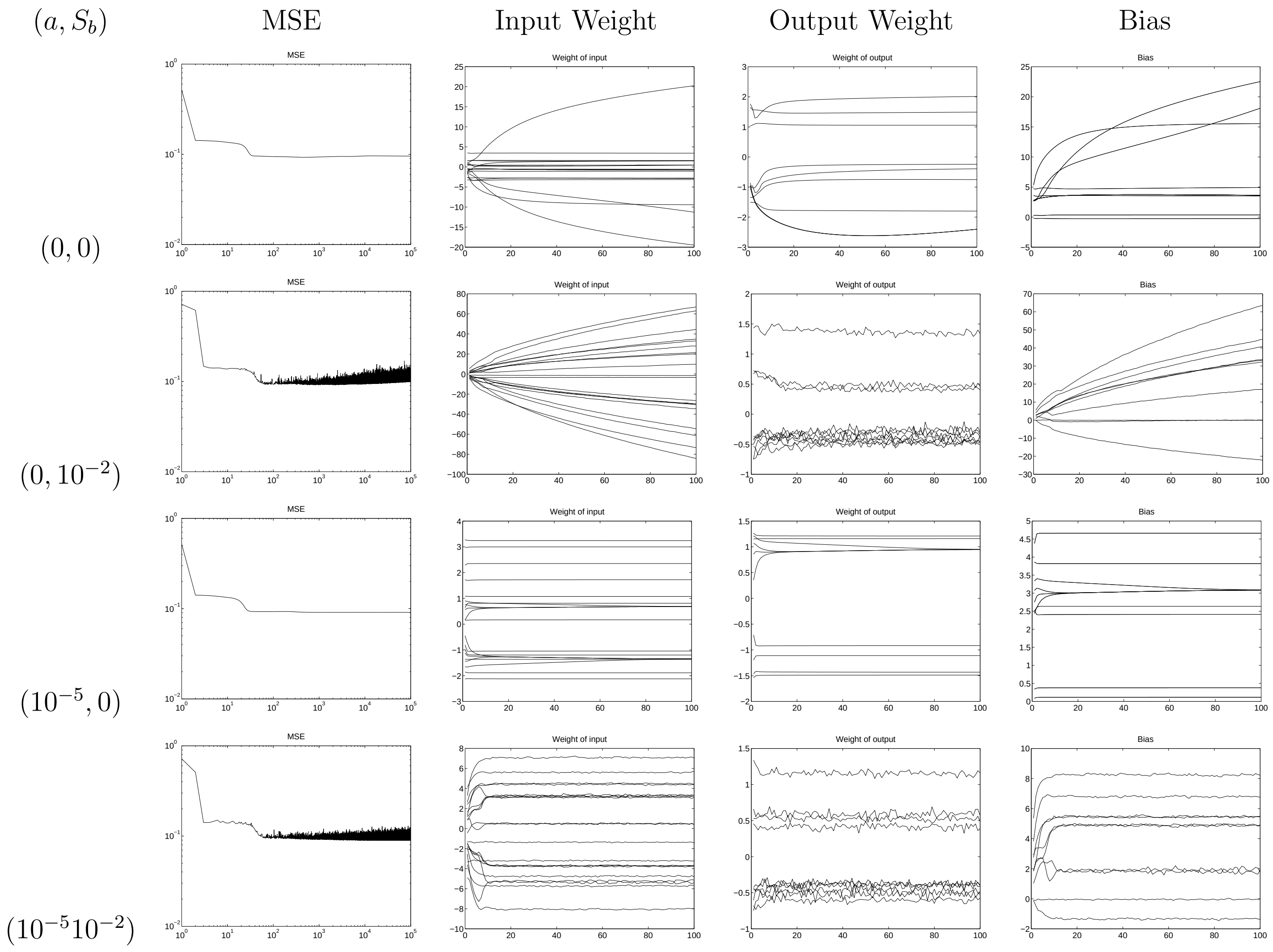

Fig. 4.17: MSE & Weight of NAR-MWN.

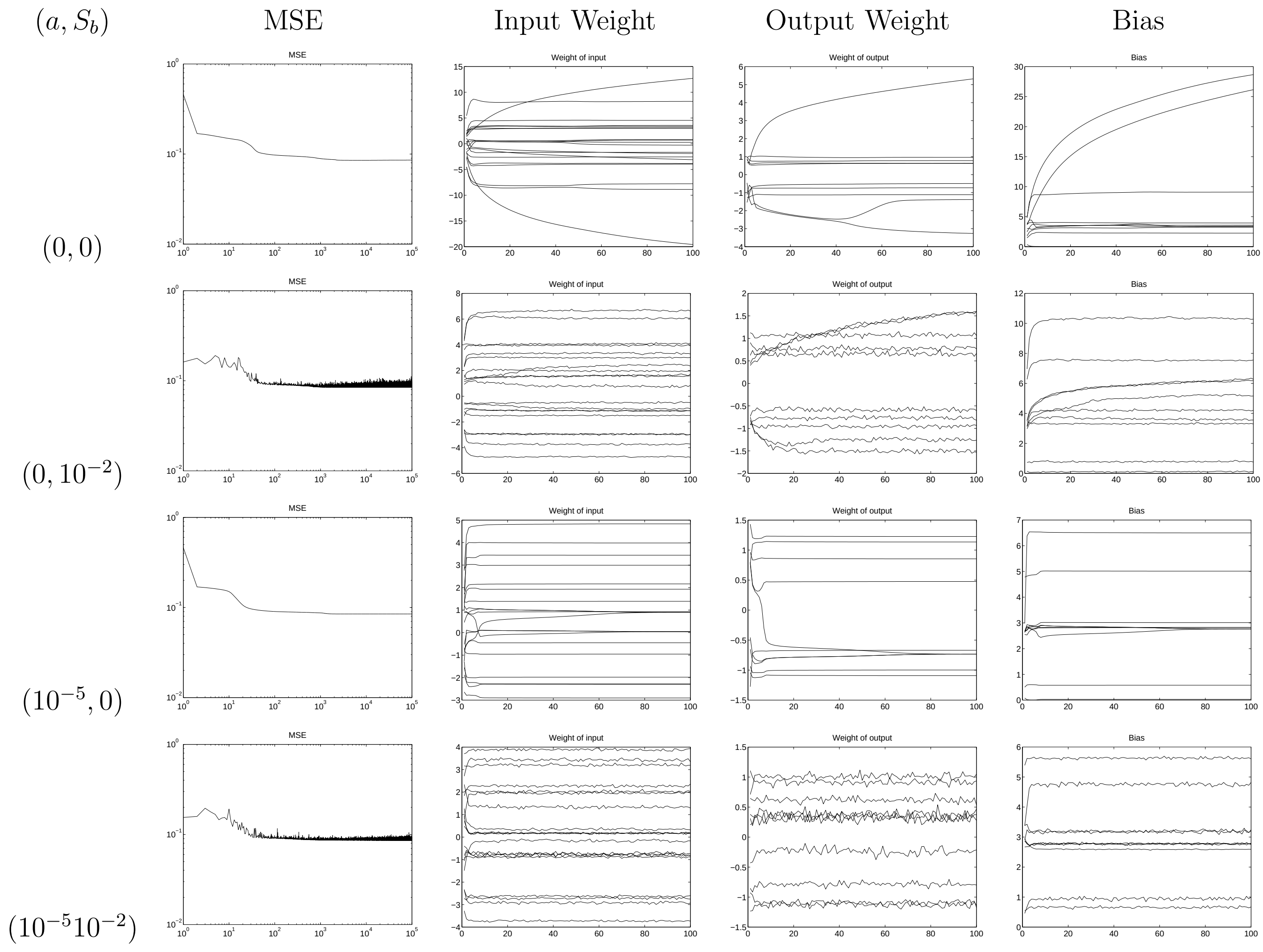

Fig. 4.18: MSE & Weight of NAR-AWN.

4.3.4 Astrophysical data

Fault tolerance (Fig 4.19 & Fig 4.20): For both cases of MWN and AWN, the MLP generated by the algorithm based on only weight noise injection during training gives the best performance. The MLP generated by the algorithm based on adding only weight decay gives the worst performance.

In this simulations, we have found that adding weight decay does not improve the fault tolerance ability. To investigate this problem, we have tried different network structures as shown in Table **??**.

	Input nodes	Hidden nodes	Output nodes
	5	10 S.N.	1 L.N.
	5	15 S.N.	1 L.N.
Astrophysical	5	20 S.N.	1 L.N.
data	6	10 S.N.	1 L.N.
	7	10 S.N.	1 L.N.
	5	5/5 S.N. (2 hidden layers)	1 L.N.

* S.N. : Sigmoid node

L.N. : Linear node

Table 4.5: Astrophysical data network structures.

Beside, we also changed the step size from 0.1 to 0.01. In so far, we still could not find the reason to explain this exceptional finding.

Convergence (Fig 4.21 & Fig 4.22): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge. However, when $a = 0$ the weights do not converge.

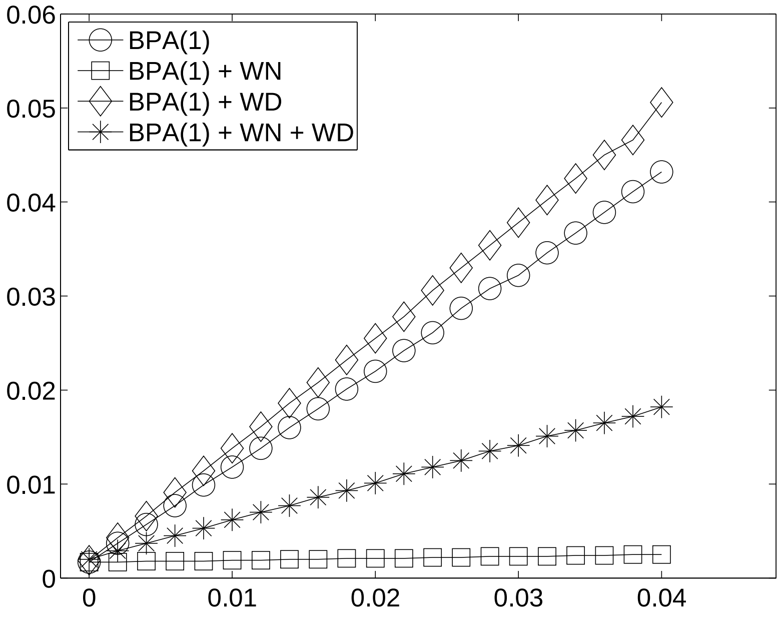

Fig. 4.19: Testing MSE of Astrophysical-MWN.

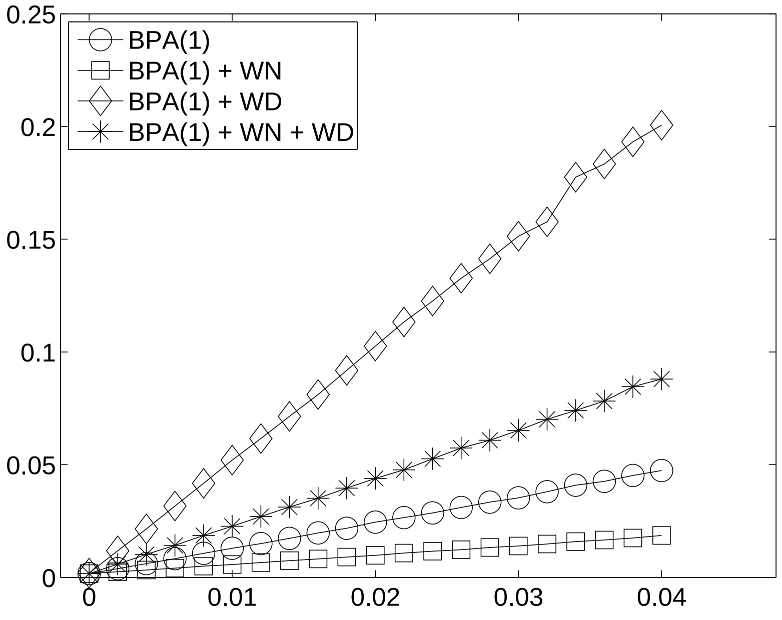

Fig. 4.20: Testing MSE of Astrophysical-AWN.

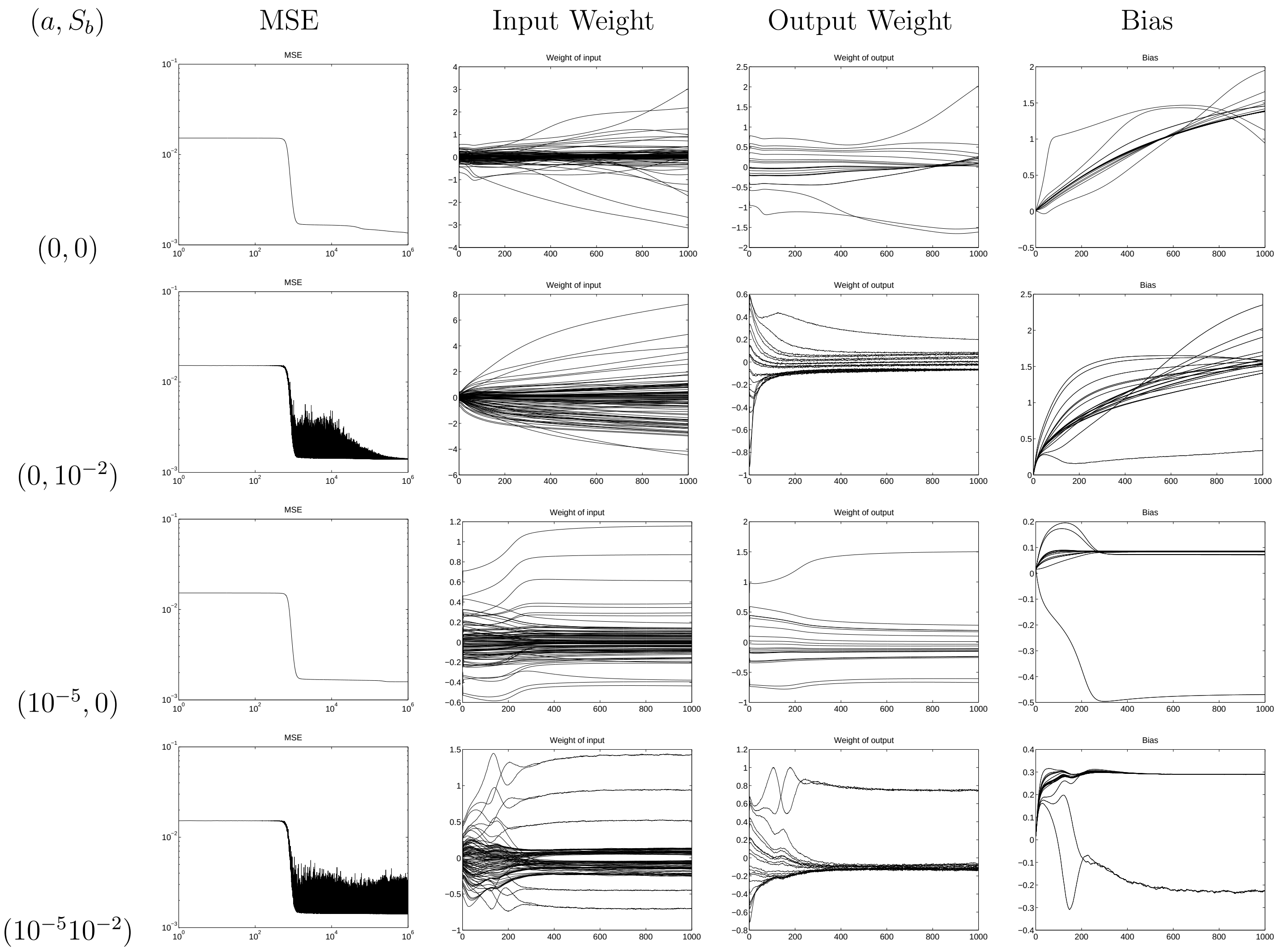

Fig. 4.21: MSE & Weight of Astrophysical data-MWN.

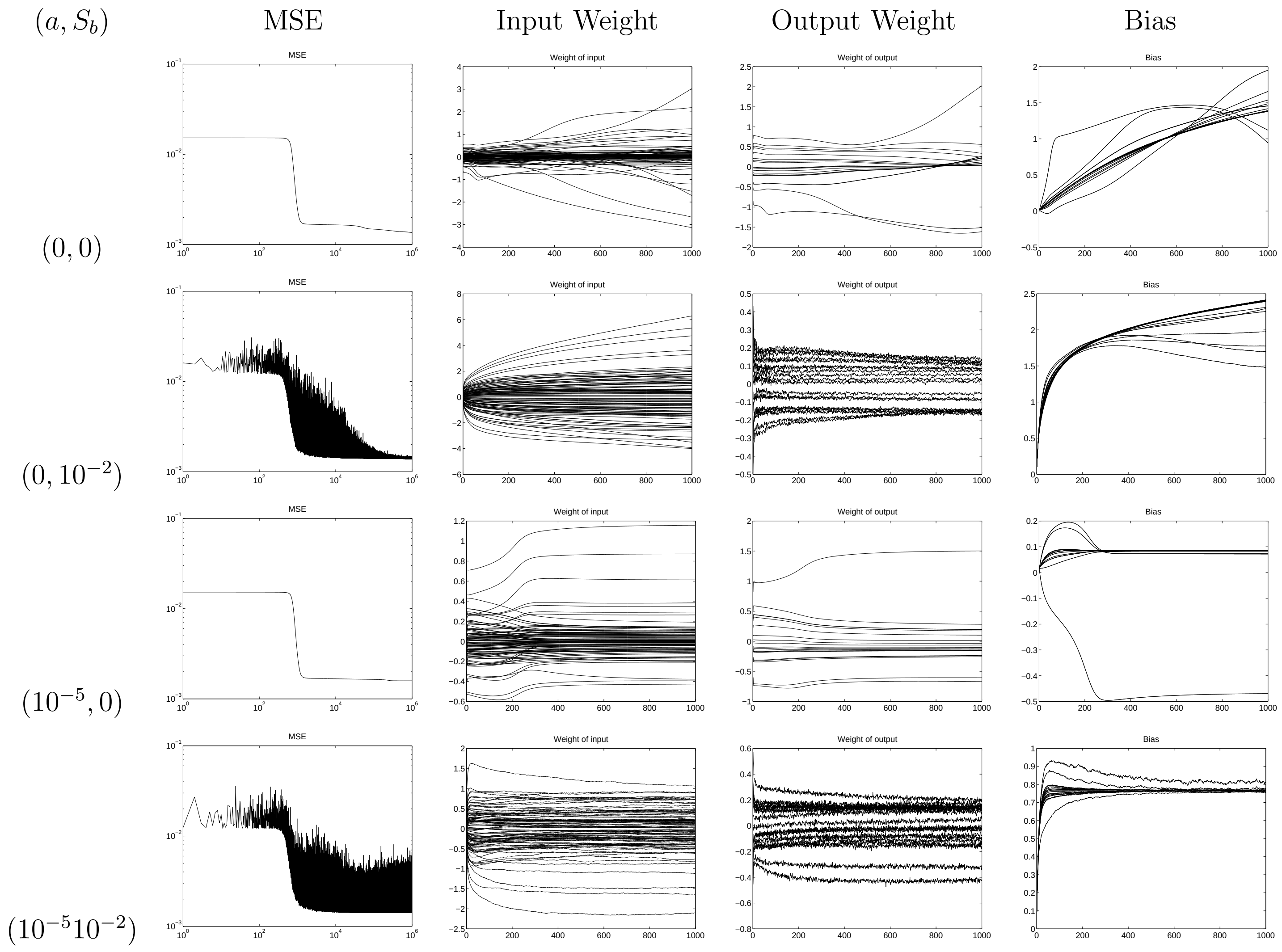

Fig. 4.22: MSE & Weight of Astrophysical data-AWN.

4.3.5 XOR

Fault tolerance (Fig 4.23 & Fig 4.24): For MWN, the MLP generated by the algorithm based on (1) combining weight noise injection with weight decay during training ,or (2) only adding weight decay give the best performance. These two cases seem the same. The MLP generated by the algorithm based on pure back-propagation gives the worst performance. For AWN, the MLP generated by the algorithm based on combining weight noise injection with weight decay during training gives the best performance. The MLP generated by the algorithm based on only adding weight decay gives the worst performance.

Convergence (Fig 4.25 & Fig 4.26): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge. However, when $a = 0$ the weights do not converge.

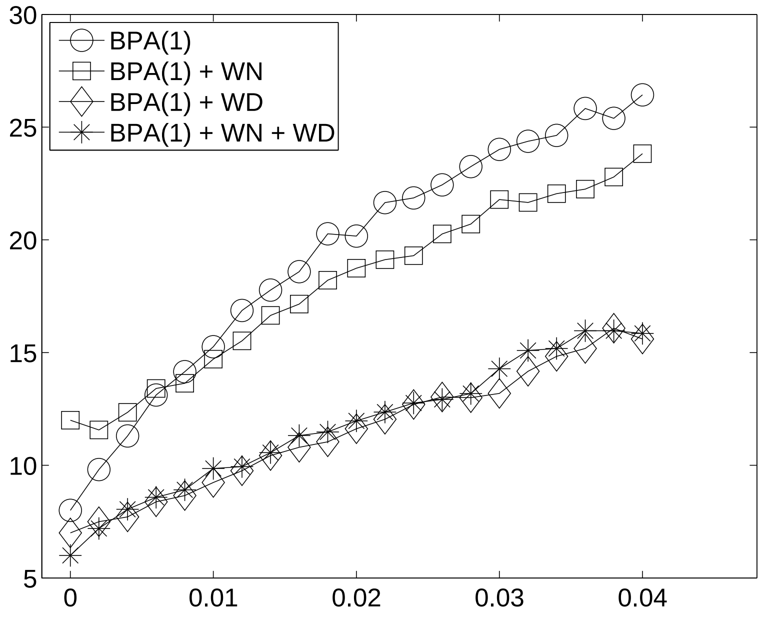

Fig. 4.23: Classification error of XOR-MWN.

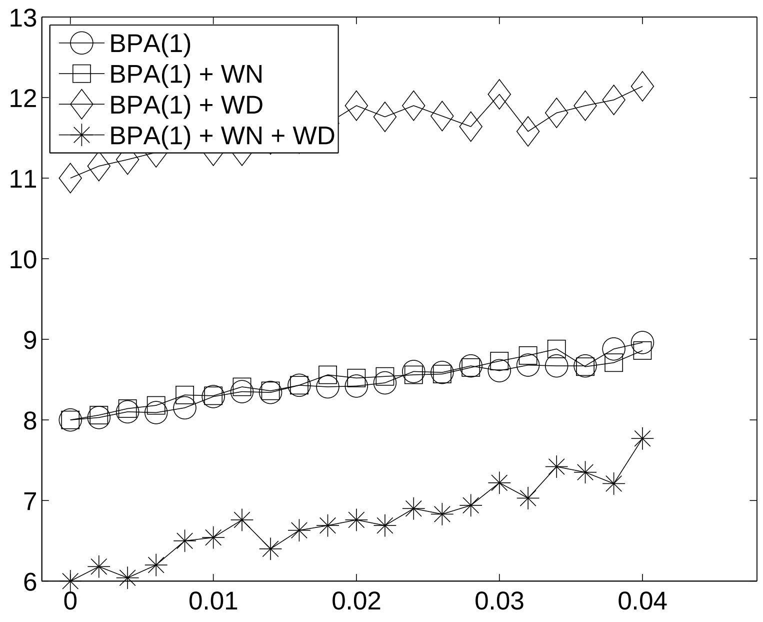

Fig. 4.24: Classification error of XOR-AWN.

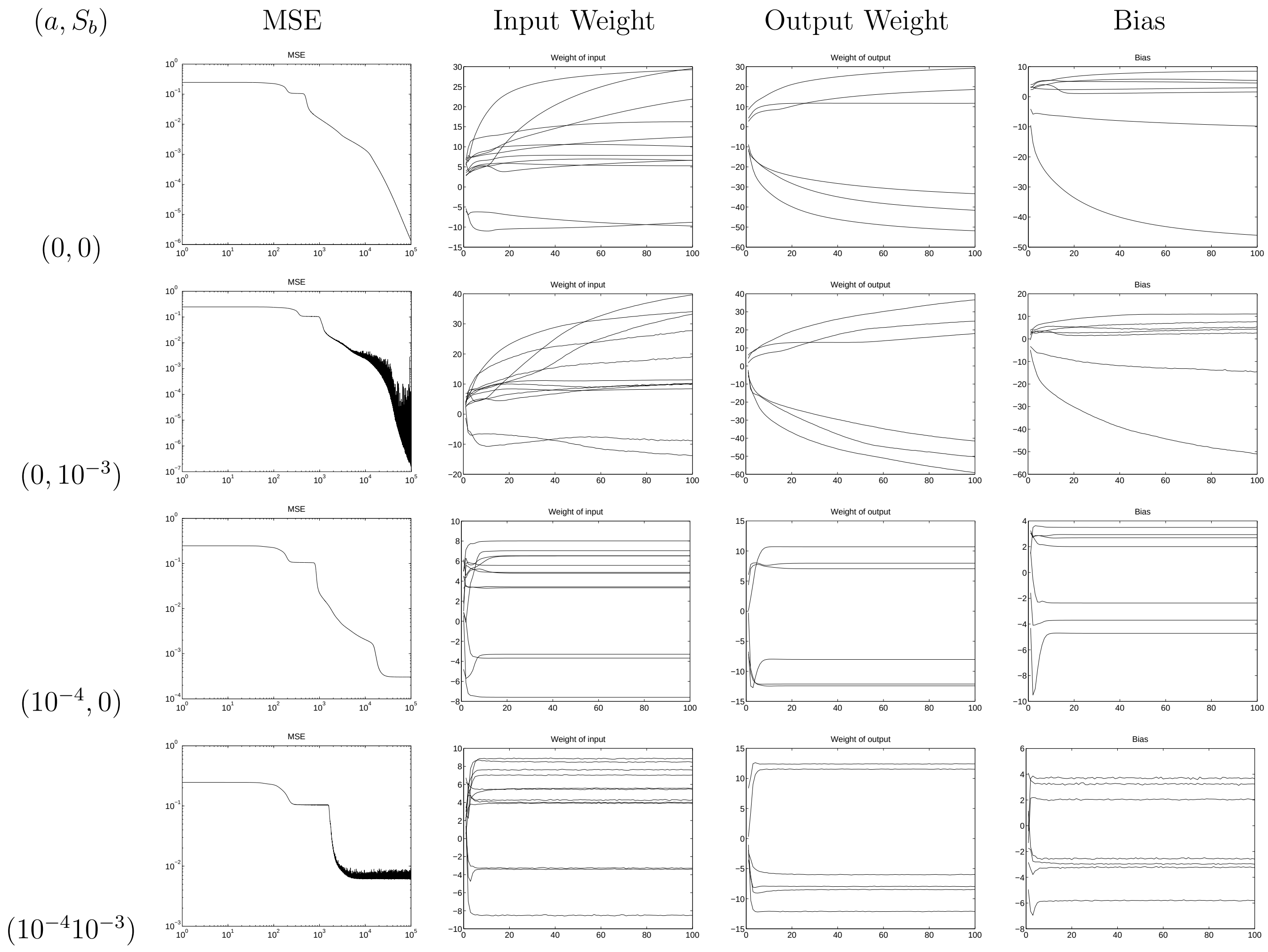

Fig. 4.25: MSE & Weight of XOR-MWN.

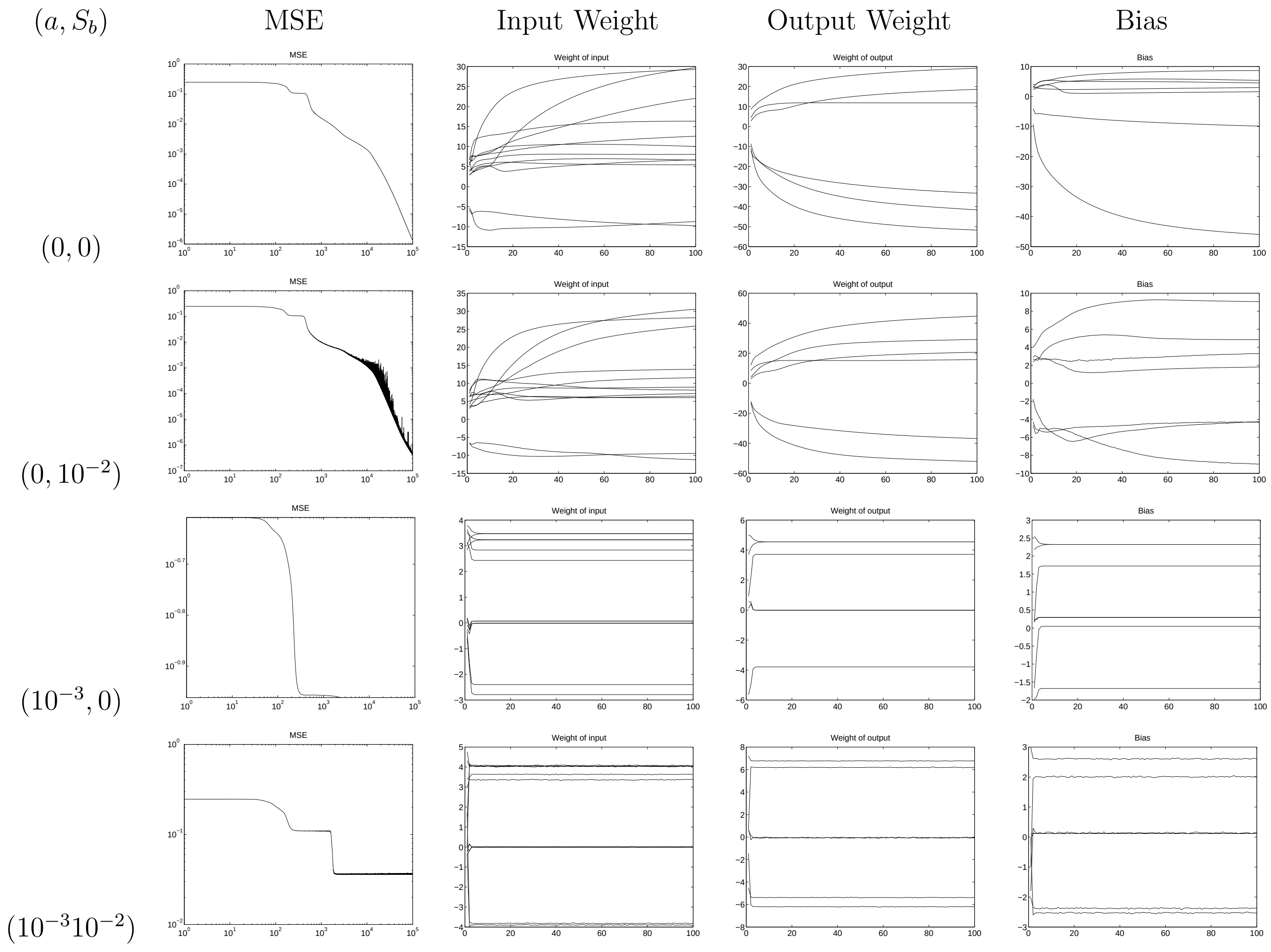

Fig. 4.26: MSE & Weight of XOR-AWN.

4.3.6 Handwritten recognition

Fault tolerance (Fig 4.27 & Fig 4.28): For MWN, there are two situations: (1)When variance was less than 0.02, the MLP generated by the algorithm based only adding weight decay during training gives the best performance. (2) When variance was greater than 0.02, the MLP generated by the algorithm based combining weight noise injection with weight decay during training gives the best performance. The MLP generated by the algorithm based on weight noise injection gives the worst performance. For AWN, there are three situations: (1) When variance was less than

0.01, the MLP generated by the algorithm based on adding weight decay during training gives the best performance. (2) When variance was between 0.01 and 0.03, the MLP generated by the algorithm based on combining weight noise injection with weight decay during training gives the best performance. (3) When variance was greater than 0.03, the MLP generated by the algorithm based on only weight noise injection during training gives the best performance.

Convergence (Fig 4.29 & Fig 4.30): For both cases of MWN and AWN, the MSE converge during training even if $a = 0$. When $a >= 10^{-5}$ the weights converge. However, when $a = 0$ the weights do not converge.

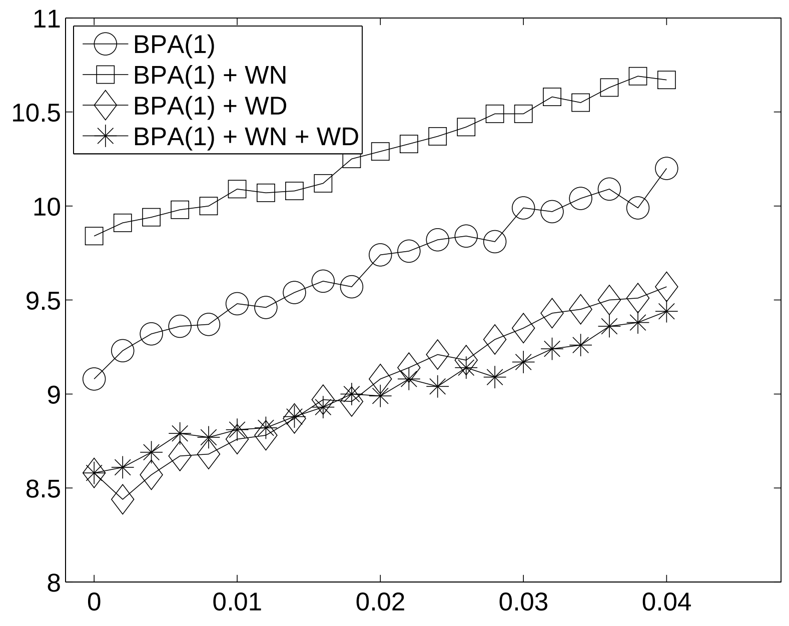

Fig. 4.27: Classification error of Handwritten recognition-MWN.

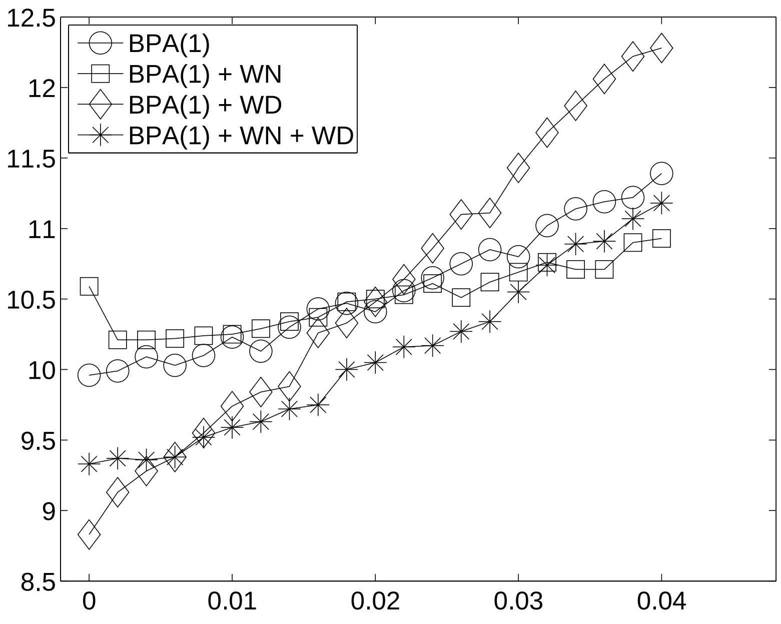

Fig. 4.28: Classification error of Handwritten recognition-AWN.

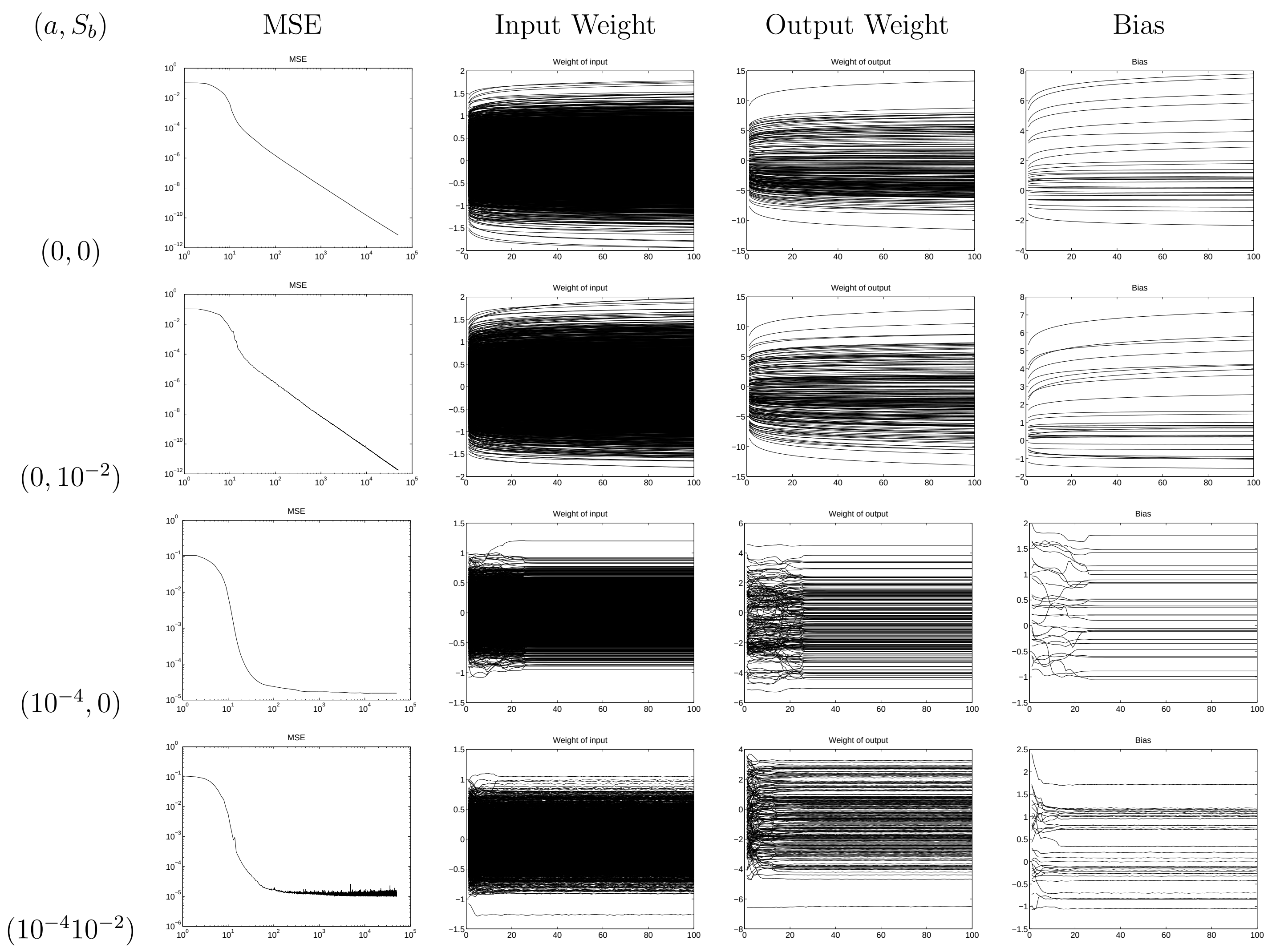

Fig. 4.29: MSE & Weight of Handwritten recognition-MWN.

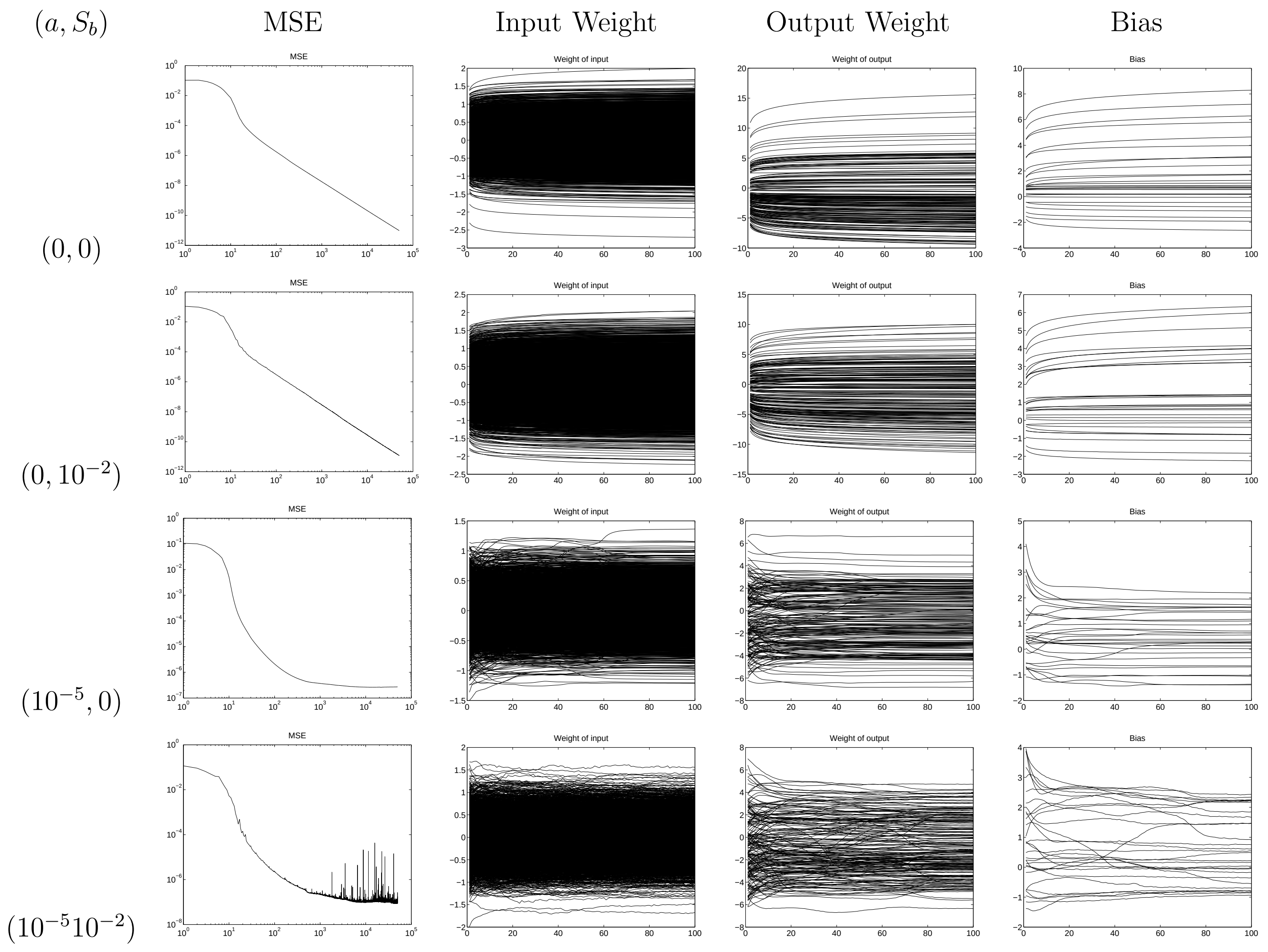

Fig. 4.30: MSE & Weight of Handwritten recognition-AWN.

4.3.7 Summary

In overall, the results presented in the previous sections can be summarized as in Table 4.6.

	F.T. MWN	F.T. AWN	Convergence MWN	Convergence AWN
2D mapping	$\surd$	$\surd$	$\surd$	$\surd$
Mackey-Glass	$\surd$	$\surd$	$\surd$	$\surd$
NAR	$\surd$	$\surd$	$\surd$	$\surd$
Astrophysical	$-$	$-$	$\surd$	$\surd$
XOR	$\surd$	$\surd$	$\surd$	$\surd$
Character recognition	$\surd$	$\surd$	$\surd$	$\surd$

Table 4.6: Summary of results.

1. (Fault Tolerance) The MLP obtained by the algorithms based on combining weight noise injection with weight decay could have better fault tolerance ability, except for the Astrophysical dataset.

2. (Convergence) The algorithms based on combining weight noise injection with weigh decay are able to converge.

Table 4.7 shows the fault tolerance ability of BPA, BPA with weight decay, and BPA with weight noise injection.

	MWN	AWN
2D mapping	$WN > WD > BPA$	$BPA \doteqdot WN \doteqdot WD$
Mackey-Glass	$WN > WD > BPA$	$WN > WD > BPA$
NAR	$WD > WN > BPA$	$WN \doteqdot WD > BPA$
Astrophysical	$WN > WD > BPA$	$WN > WD > BPA$
XOR	$WD > WN > BPA$	$WN > WD > BPA$
Character recognition	$BPA \doteqdot WN \doteqdot WD$	$WN > WD \doteqdot BPA$

Table 4.7: Summary of results (II).

Chapter 5

CONCLUSION

In this thesis, we have altogether presented eight different types of online weight noise injection-based algorithms.

MWN	AWN	Weight Decay	Output node	Noise-Free Objective
√	–	–	Linear	MSE
–	√	–	Linear	MSE
√	–	√	Linear	MSE
–	√	√	Linear	MSE
√	–	–	Sigmoid	CEE
–	√	–	Sigmoid	CEE
√	–	√	Sigmoid	CEE
–	√	√	Sigmoid	CEE

Table 5.1: Algorithms studied in this thesis.

In which, four of them are extended from the original back-propagation algorithm based on minimizing mean square errors (MSE). While the other four are extended from the original back-propagation algorithm based on minimizing cross entropy errors (CEE). For those algorithms based on MSE, the output neuron is defined as a linear neuron. For those algorithms based on CEE, the output neuron is defined as a sigmoid neuron.

To study the convergence behaviors of these algorithms and the fault tolerance abilities of the multilayer perceptron (MLP) models attained by these learning algorithms, extensive computer simulations based on six datasets are conducted. The datasets include (1) an artificial 2D mapping, (2) the Mackey-Class time series data, (3) a nonlinear time series data from Chen, (4) an astrophysical dataset, (5) XOR problem, (6) a handwritten character dataset. The first four datasets are regression datasets. They are used for testifying the algorithms based on MSE. The last two datasets are classification datasets. They are used for testifying the algorithms based on CEE.

First, simulation results show that the online learning algorithm based on purely multiplicative weight noise injection does not converge. Our results complement a recent analytical results analyzed by Sum *et al* regarding the objective functions of the weight noise injection-based algorithms. In their paper, it is shown that the objective function of pure multiplicative weight noise injection-based learning algorithm is with an extra integral term. In which, the property of the extra term is not easily analyzed. Our results suggest that the extra term might play a role pushing the weight vector diverges.

Second, simulation results show that the algorithms combining weight noise injection and weight decay exhibit better convergence behaviors than their pure weight noise injection counterparts. For multiplicative weight noise injection-based algorithms, the benefit of adding weight decay during training is much clear. Adding weight decay during training is able to alleviate the divergence effect due to multiplicative weight noise injection.

Third, our simulation results have also shown that the neural networks attained by the algorithms combining weight noise injection and weight decay could have better fault tolerance abilities than the neural networks attained by the pure weight noise injection-based algorithms. For the six datasets, five of them show positive results, except the astrophysical dataset.

For the astrophysical dataset, the performance of the neural networks attained by combining weight noise injection and weight decay does not show significant improvement in terms of their weight noise tolerance. Even though different architectures of MLPs have been experimented (including $[5, 10, 1]$, $[5, 15, 1]$, $[5, 20, 1]$, $[6, 10, 1]$, $[7, 10, 1]$ and $5, 5, 5, 1]$), no significant change in the performance has been observed. As when the time this thesis is submitted, we still did not have an explanation for this exception case. We leave it open for further investigation.

Owing to make this thesis self-contained, we have included an appendix with diagrams showing the fault tolerance abilities of the neural networks attained by each of these algorithms and the changes of the weight values during training. The Matlab programs for running such simulations are also included in the end of the Appendix.

Appendix A

MSE & FAULT RATE

In this appendix, the deatil results on the performance of the networks generated are presented. The variance of S_b in testing are $[0, 0.002, 0.004, ..., 0.04]$, a total of 21 cases. For each S_b, we repeated 100 times to get the distribution of the testing errors and used box-and-whisker plot the distribution. In following figures, x-axis represents the variance of S_b in testing and , y-axis represents MSE (for regression problems) or classification error (for classification problems).

A.1 2D mapping

A.1.1 Multiplicative Weight Noise (MWN)

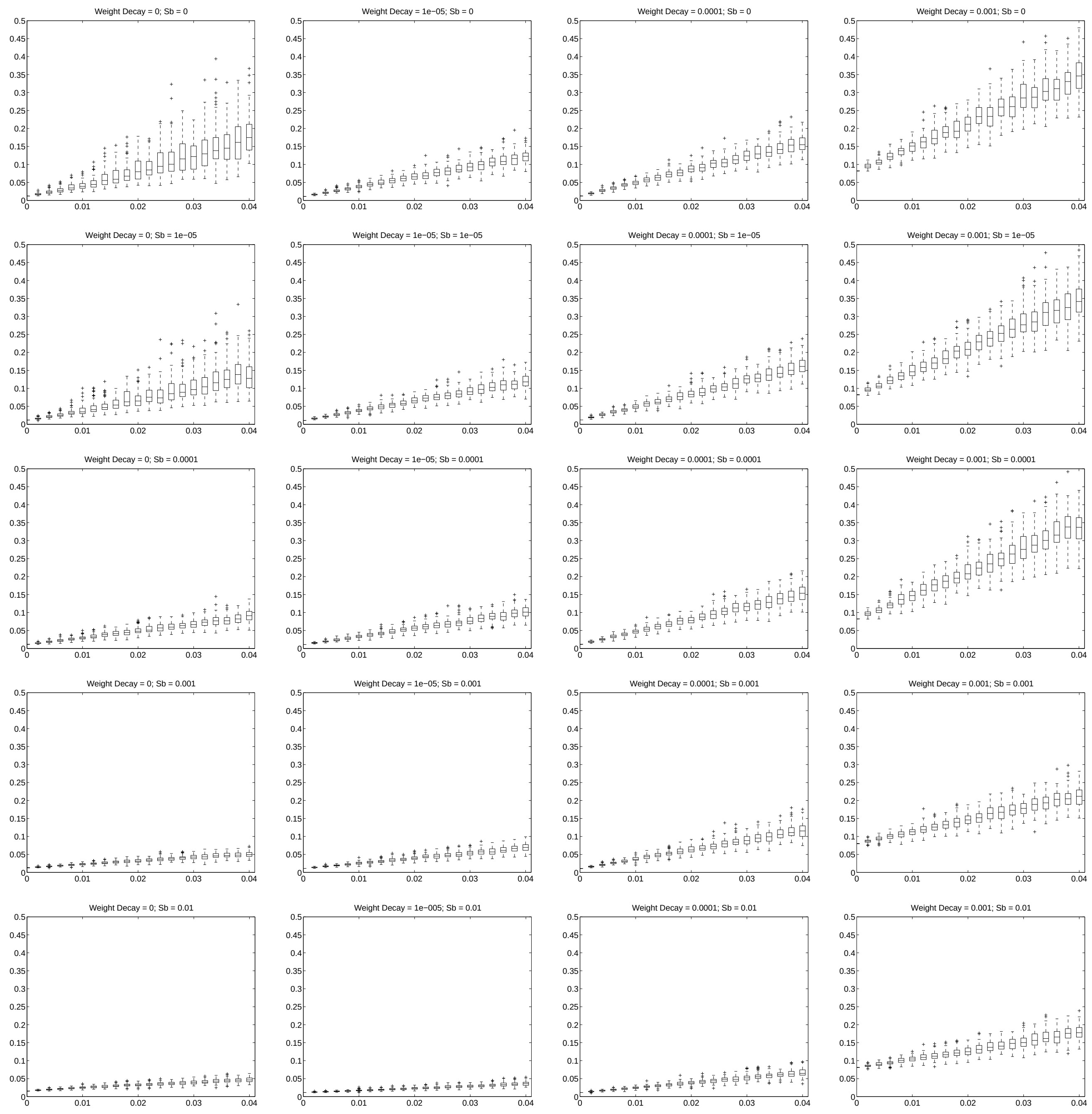

Fig. 1.1: MSE, Multiplicative Weight Noise - 2D mapping.

A.1.2 Additive Weight Noise (AWN)

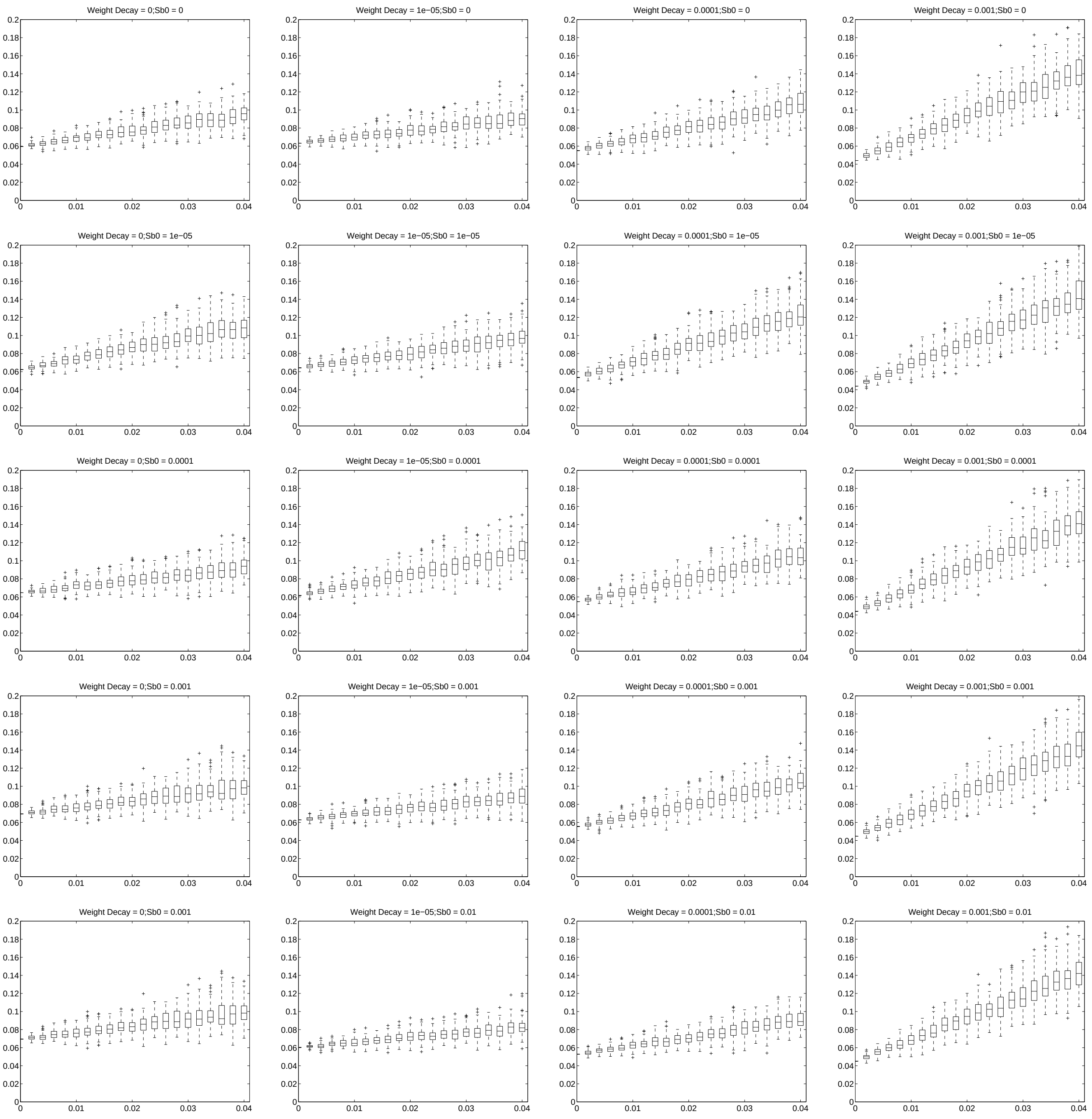

Fig. 1.2: MSE, Additive Weight Noise - 2D mapping.

A.2 Mackey-Glass

A.2.1 MWN

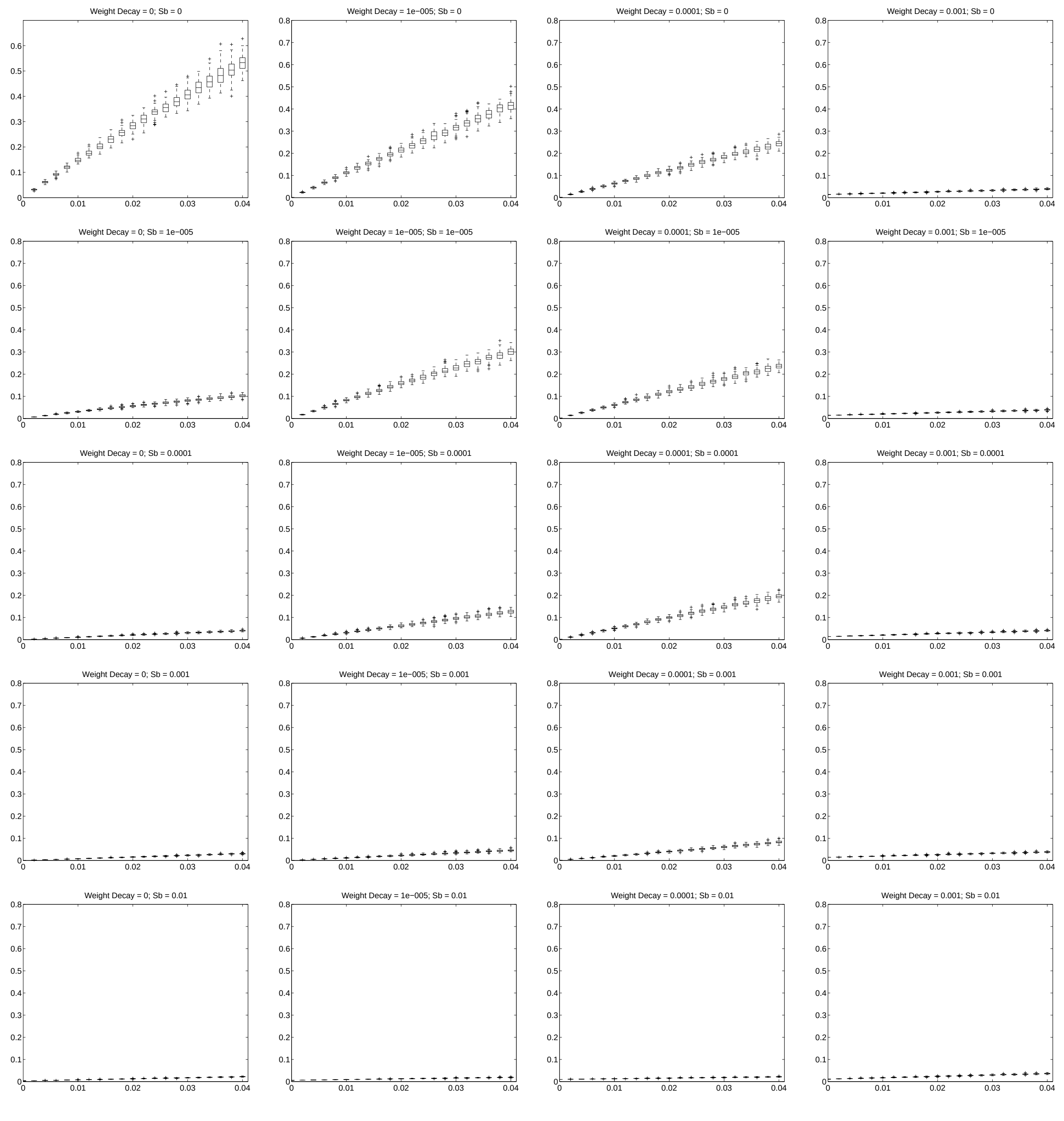

Fig. 1.3: MSE, MWN - Mackey-Glass.

A.2.2 AWN

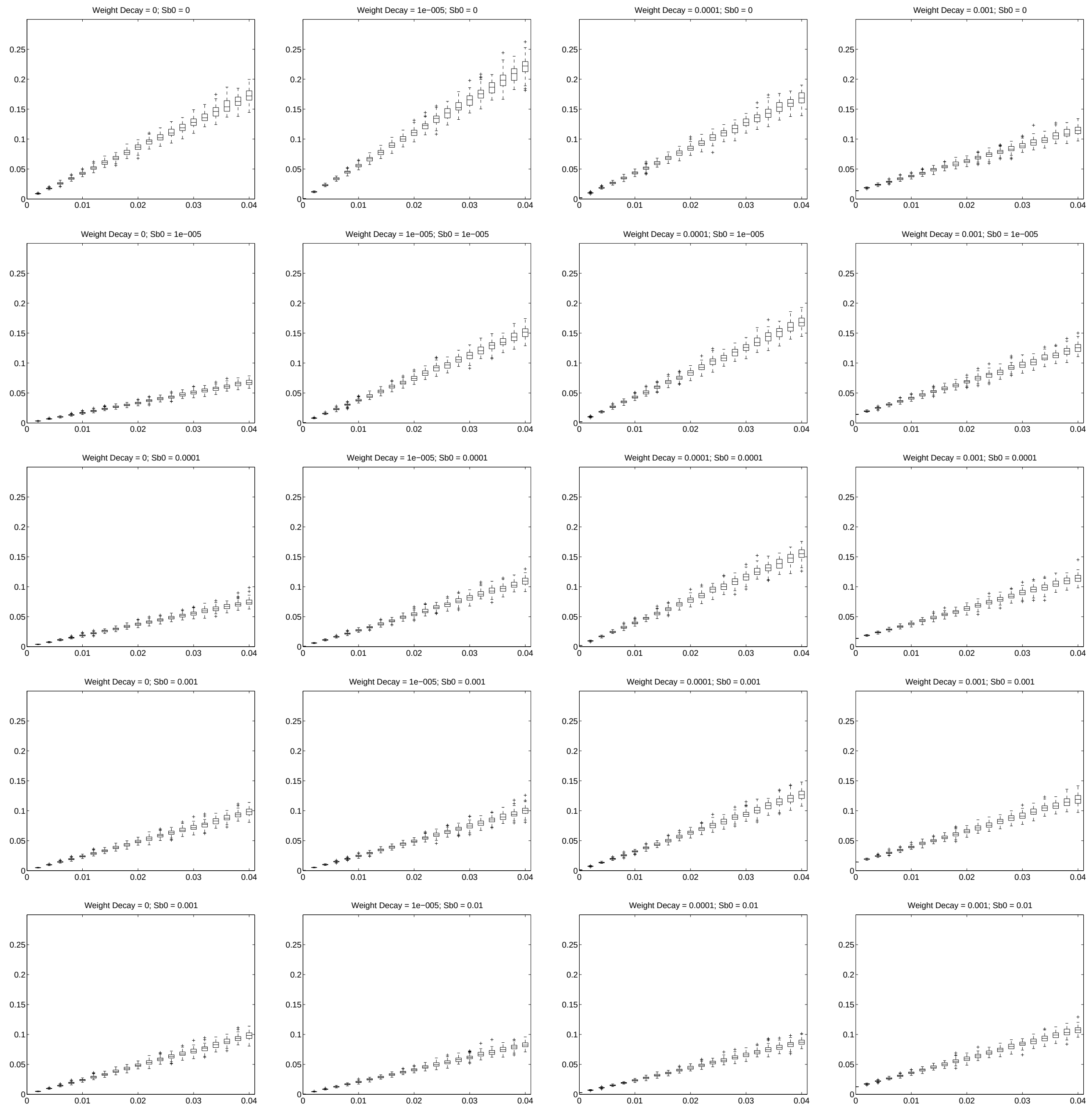

Fig. 1.4: MSE, AWN - Mackey-Glass.

A.3 Nonlinear autoregressive time series (NAR)

A.3.1 MWN

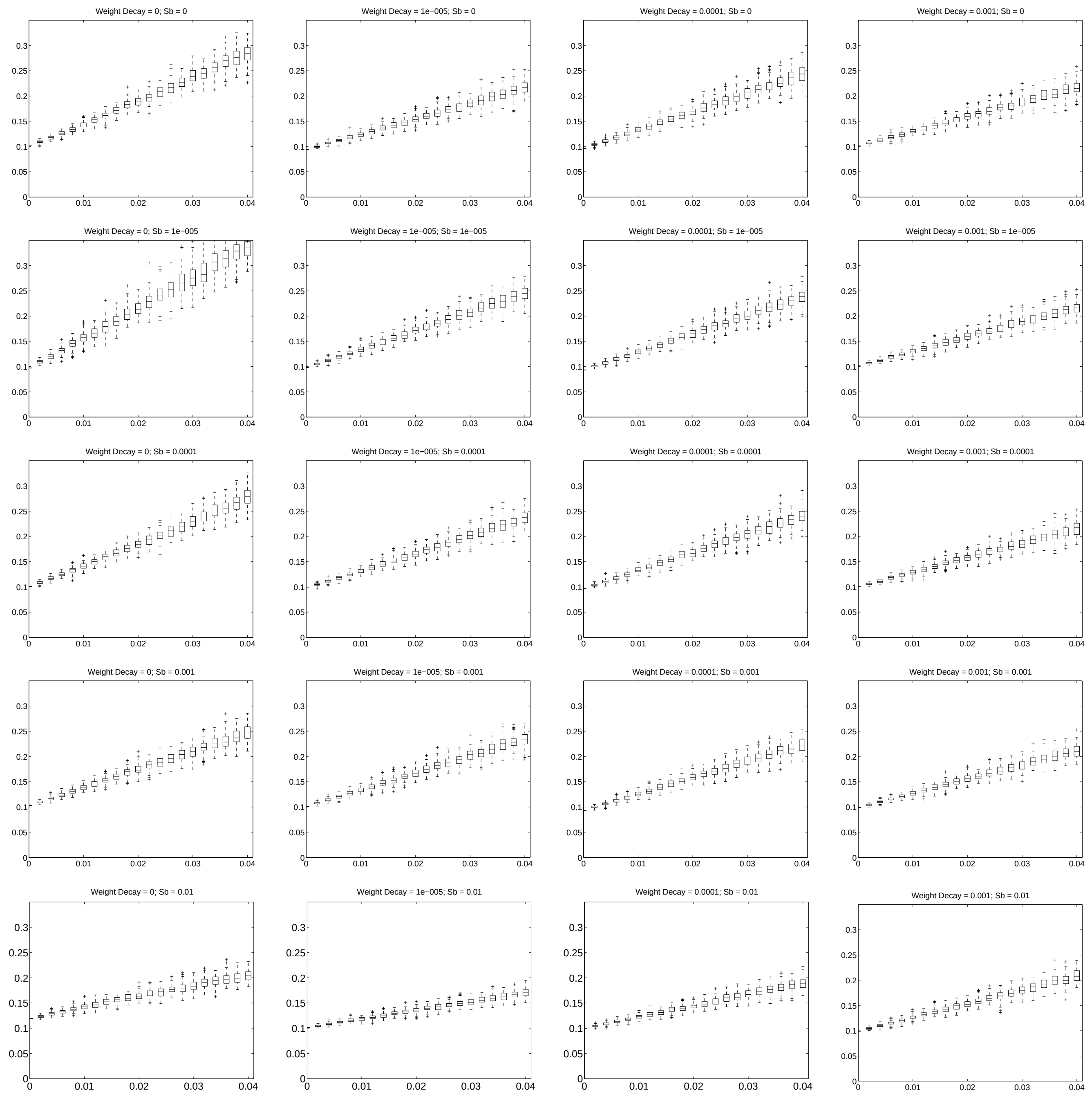

Fig. 1.5: MSE, MWN - NAR.

A.3.2 AWN

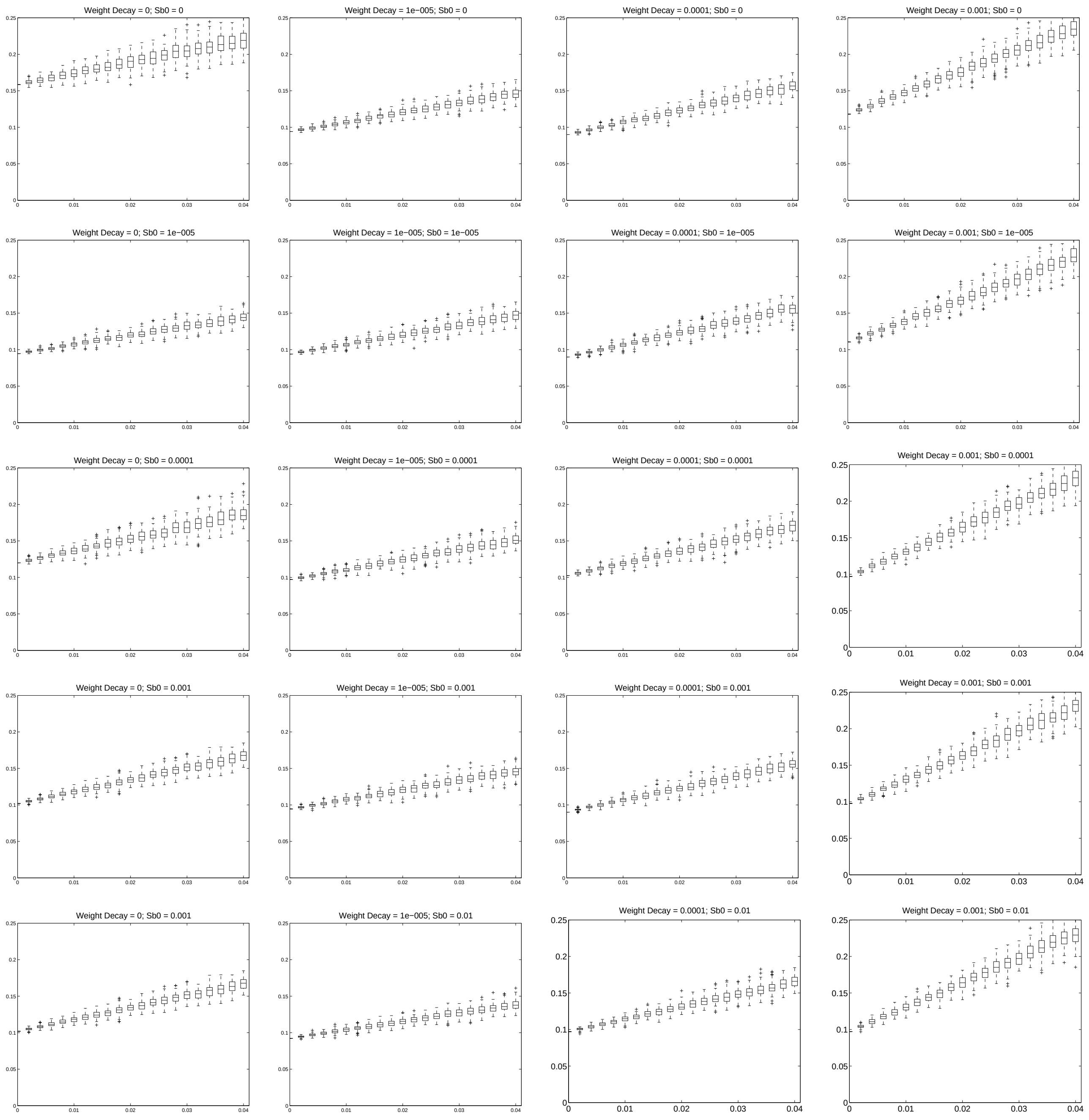

Fig. 1.6: MSE, AWN - NAR.

A.4 Astrophysical data

A.4.1 MWN

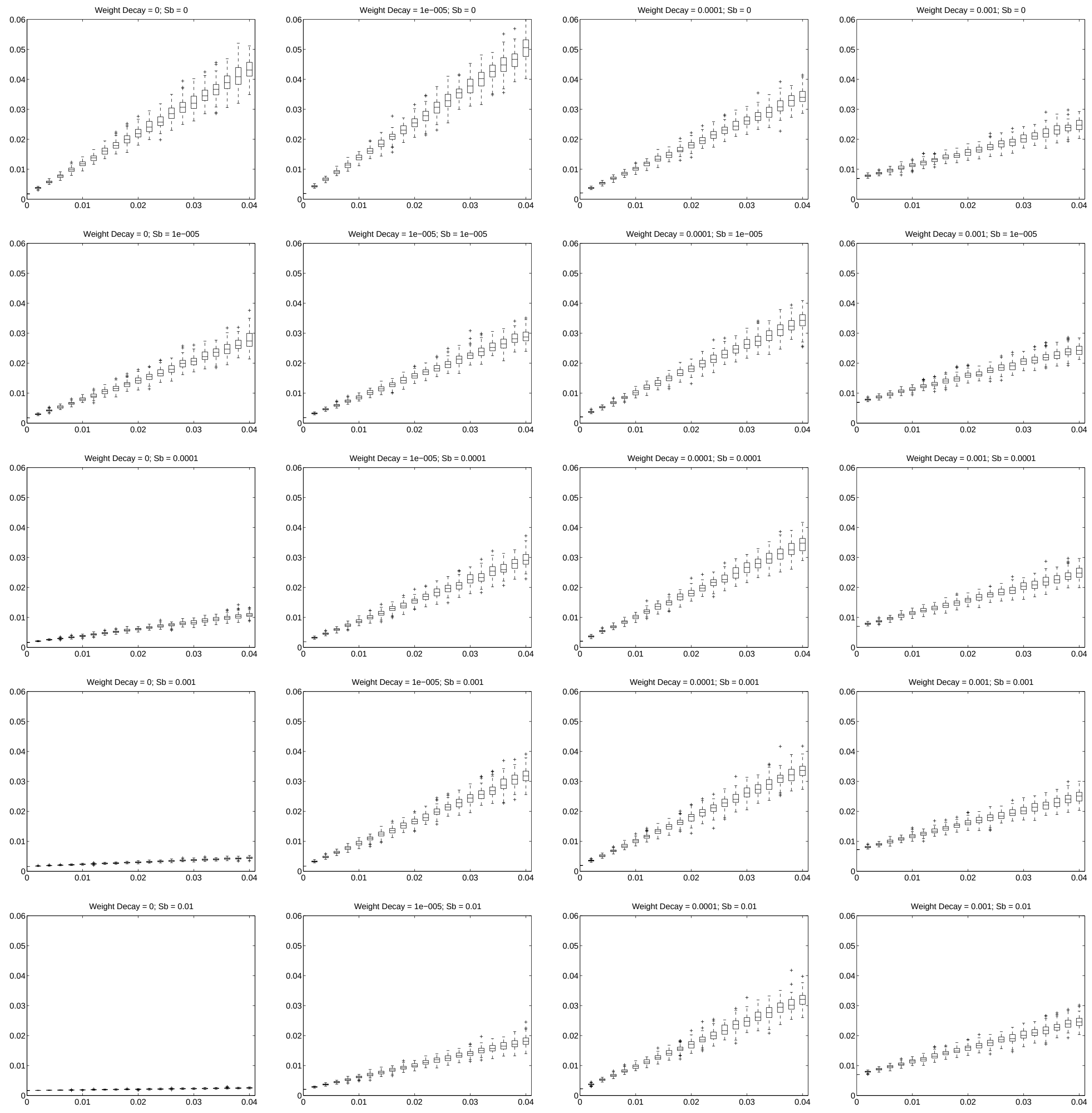

Fig. 1.7: MSE, MWN - Astrophysical data.

A.4.2 AWN

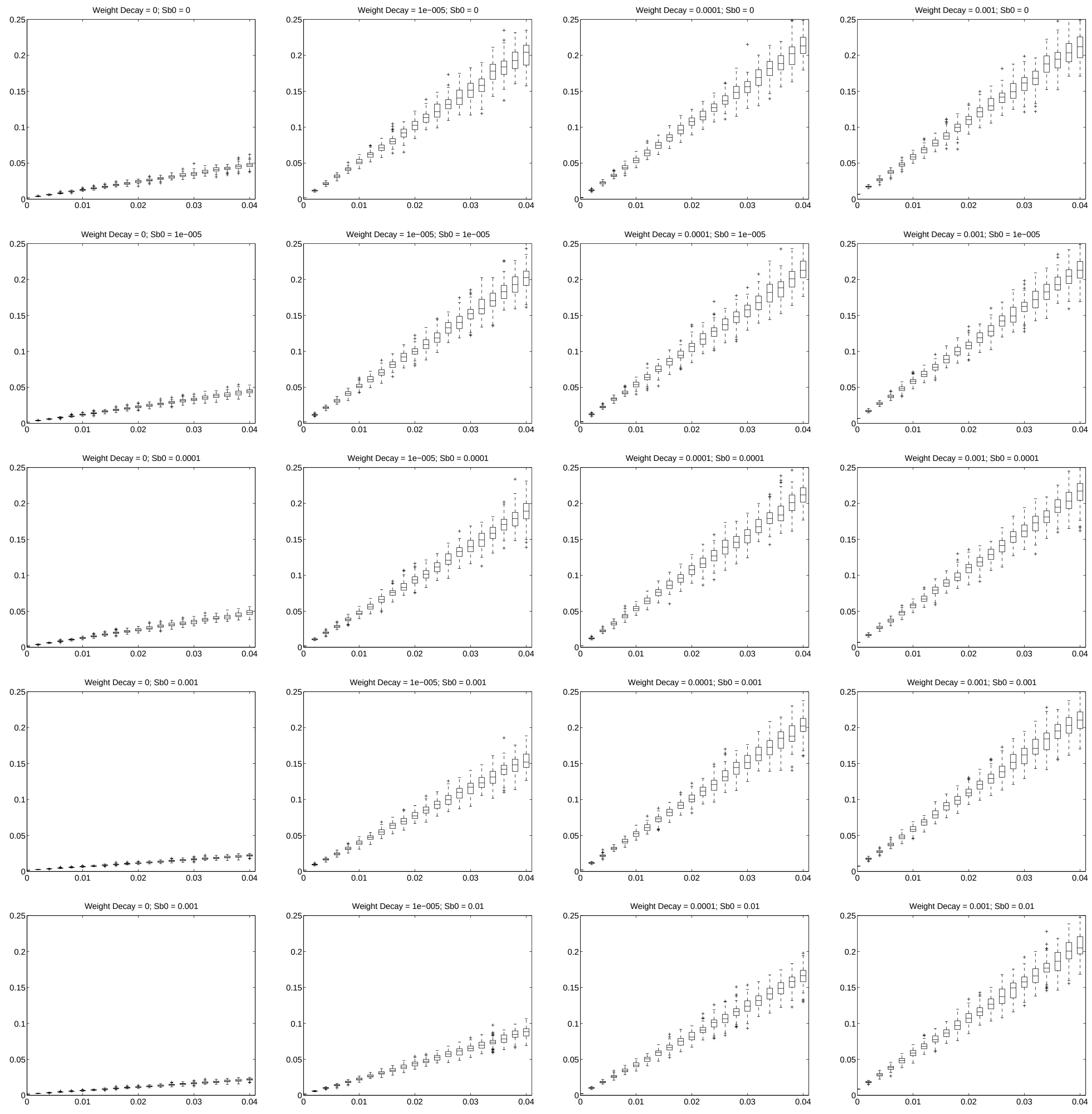

Fig. 1.8: MSE, AWN - Astrophysical data.

A.5 XOR

A.5.1 MWN

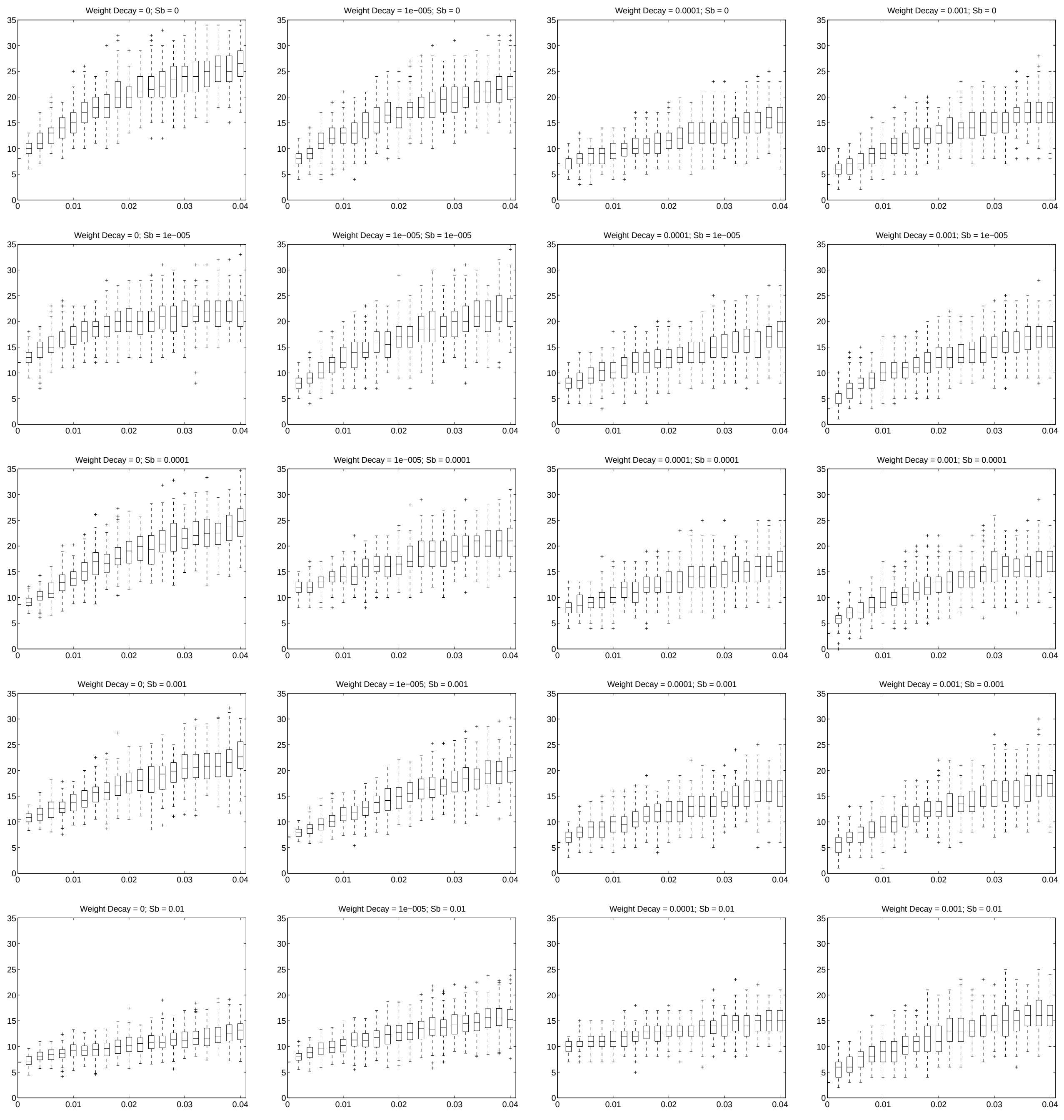

Fig. 1.9: Fault rate, MWN - XOR.

A.5.2 AWN

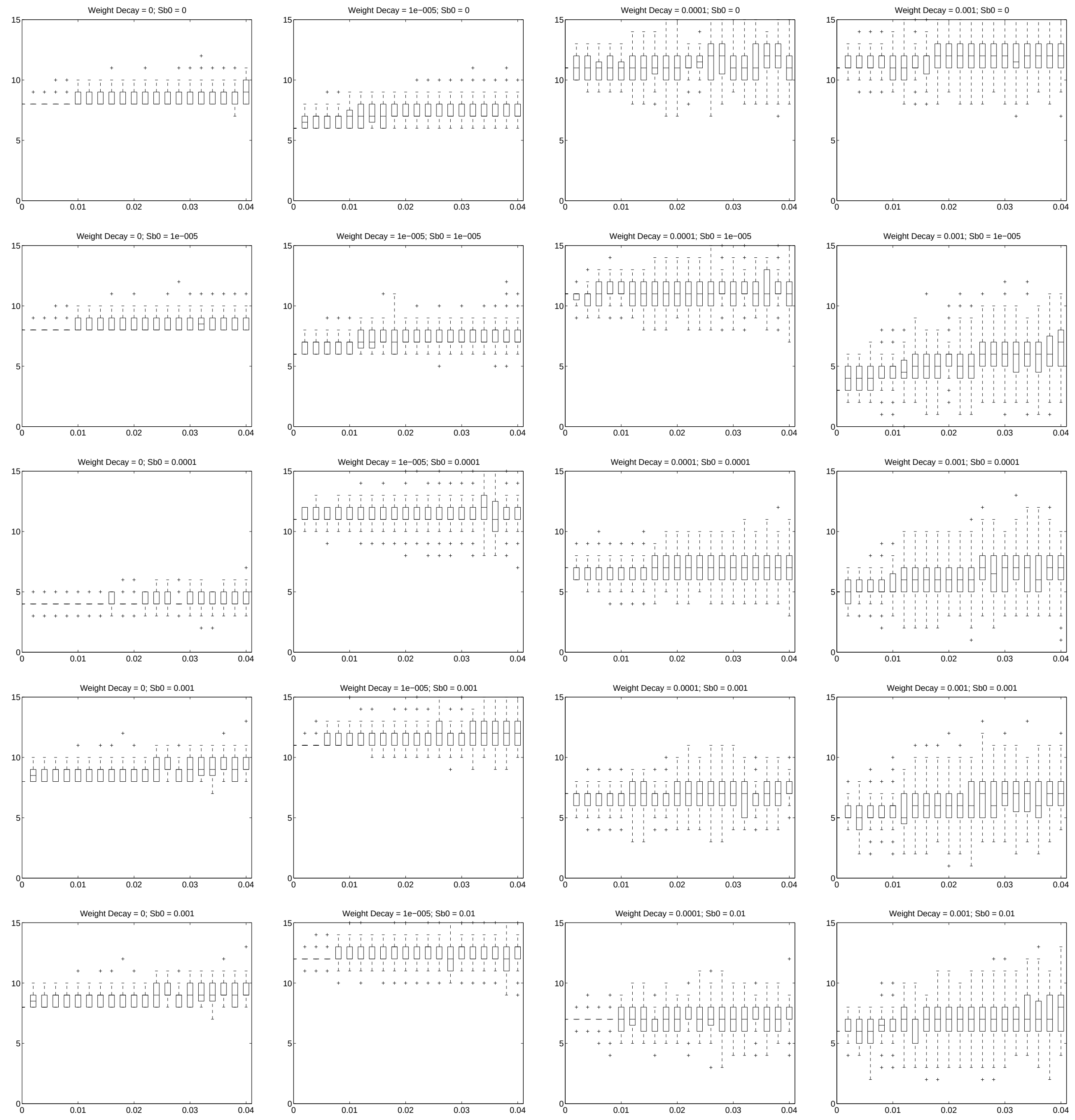

Fig. 1.10: Fault rate, AWN - XOR.

A.6 Semeion handwritten digital recognition

A.6.1 MWN

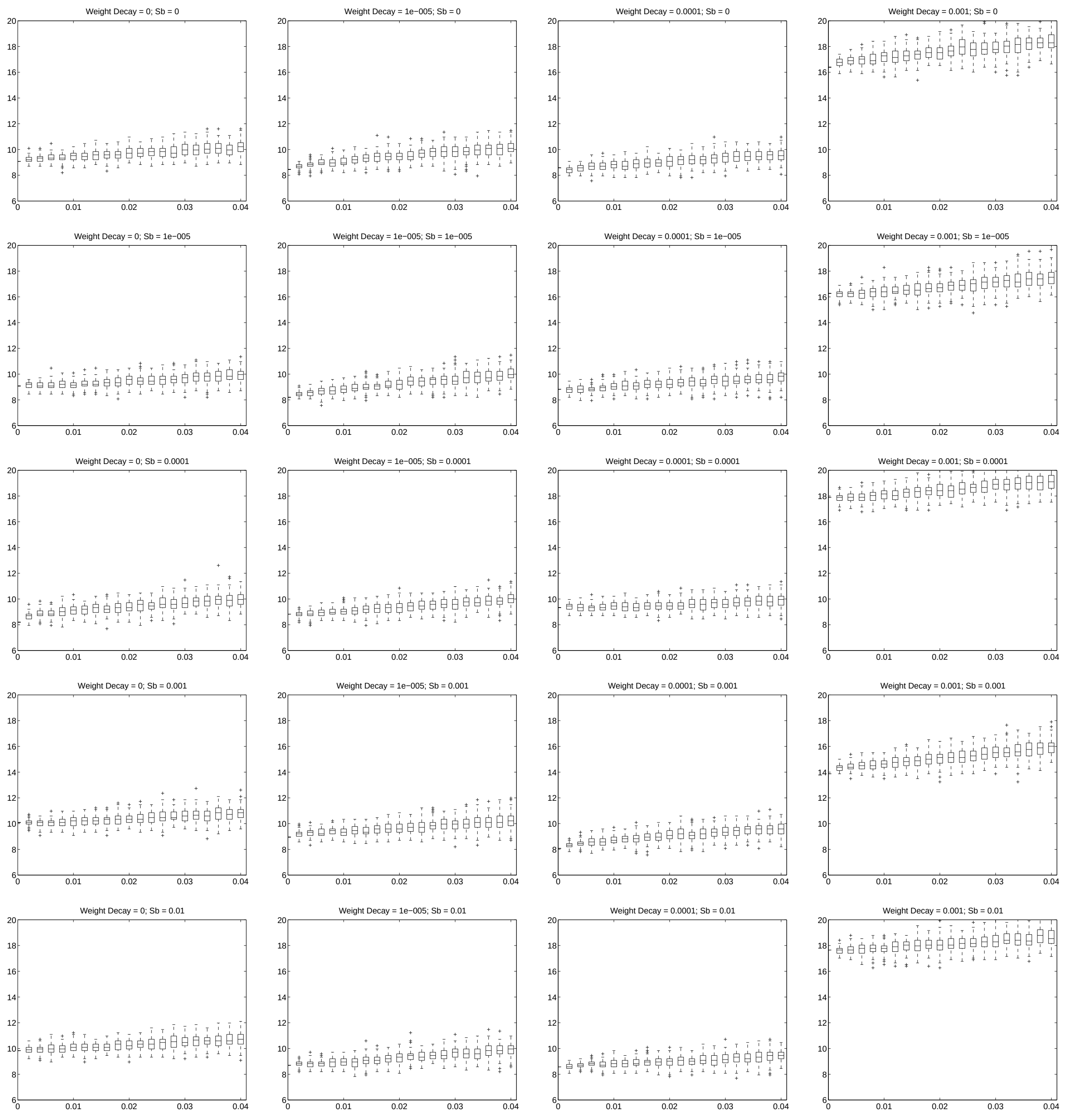

Fig. 1.11: Fault rate, MWN - Semeion.

A.6.2 AWN

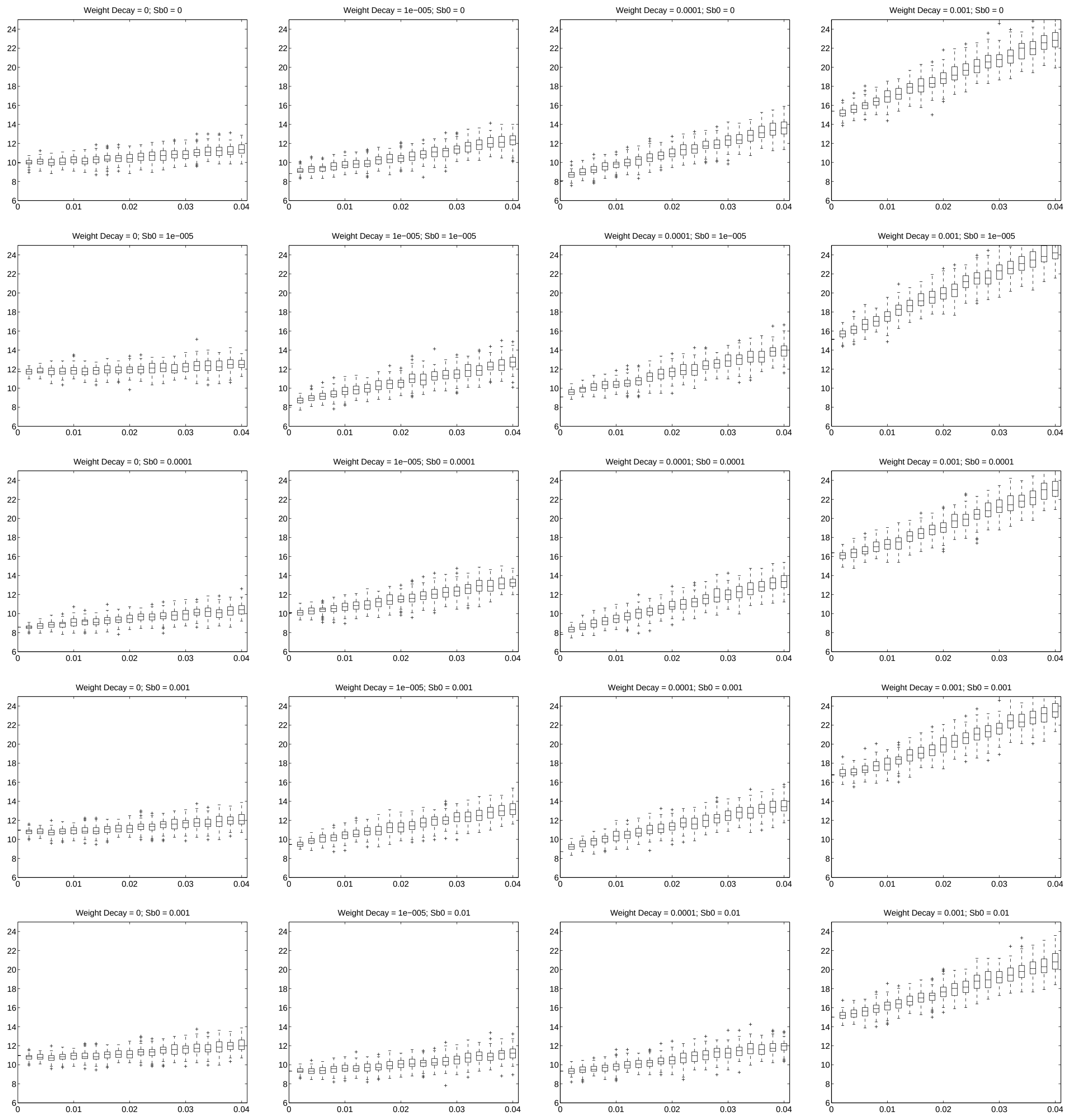

Fig. 1.12: Fault rate, AWN - Semeion.

Appendix B

CONVERGENCE OF MSE AND WEIGHT

In this appendix, the changes of the training MSE and weights against training steps will be shown. The MSEs were recored each epoch, and weights were recorded once per 1000 time. In following figures, x-axis represents epoch, and y-axis represents MSE or weight magnitude. In first column , S_b represents the variance of weight noise in training.

B.1 2D mapping

B.1.1 MWN

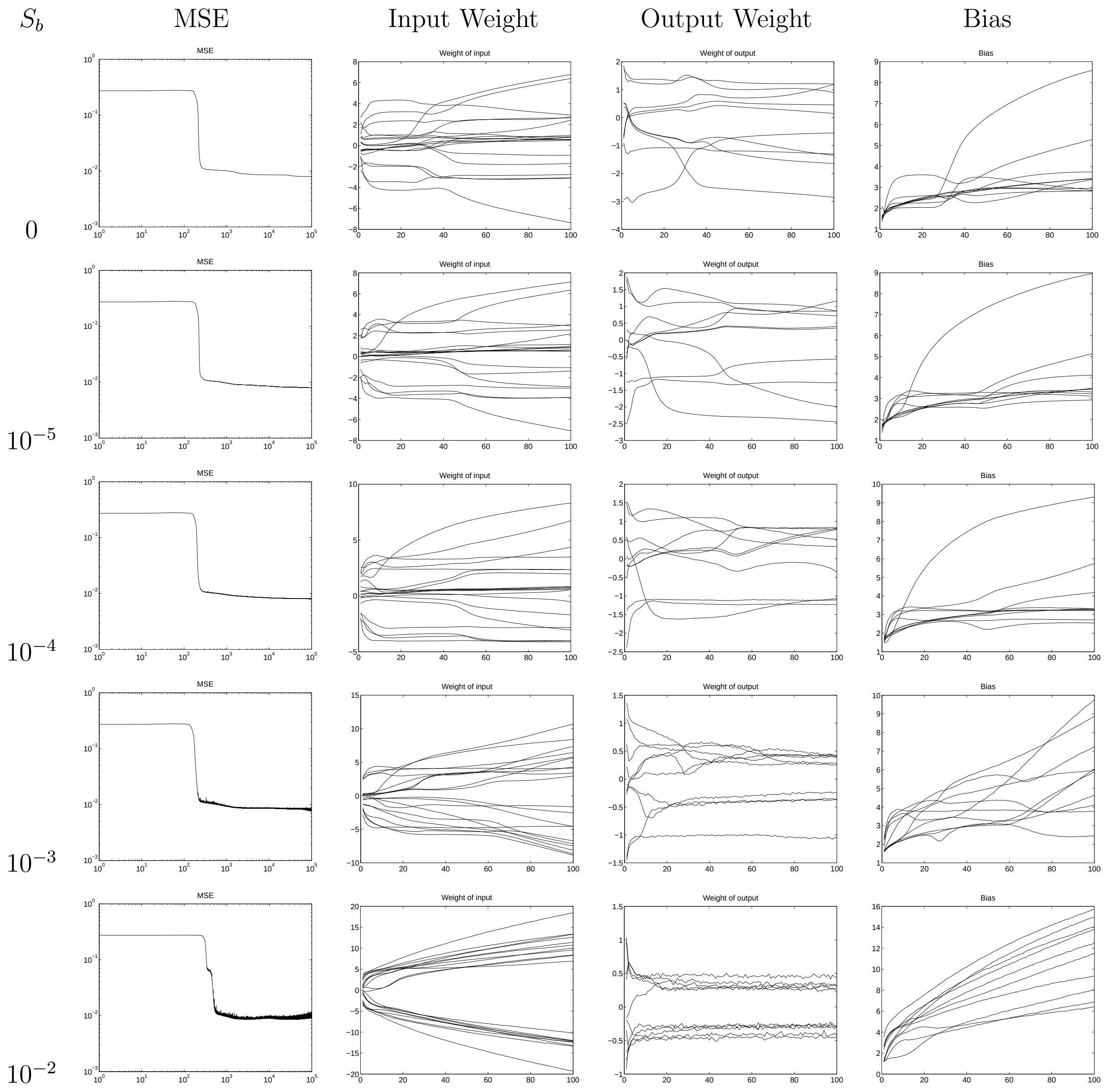

Fig. 2.1: $a = 0$, MWN - 2D mapping.

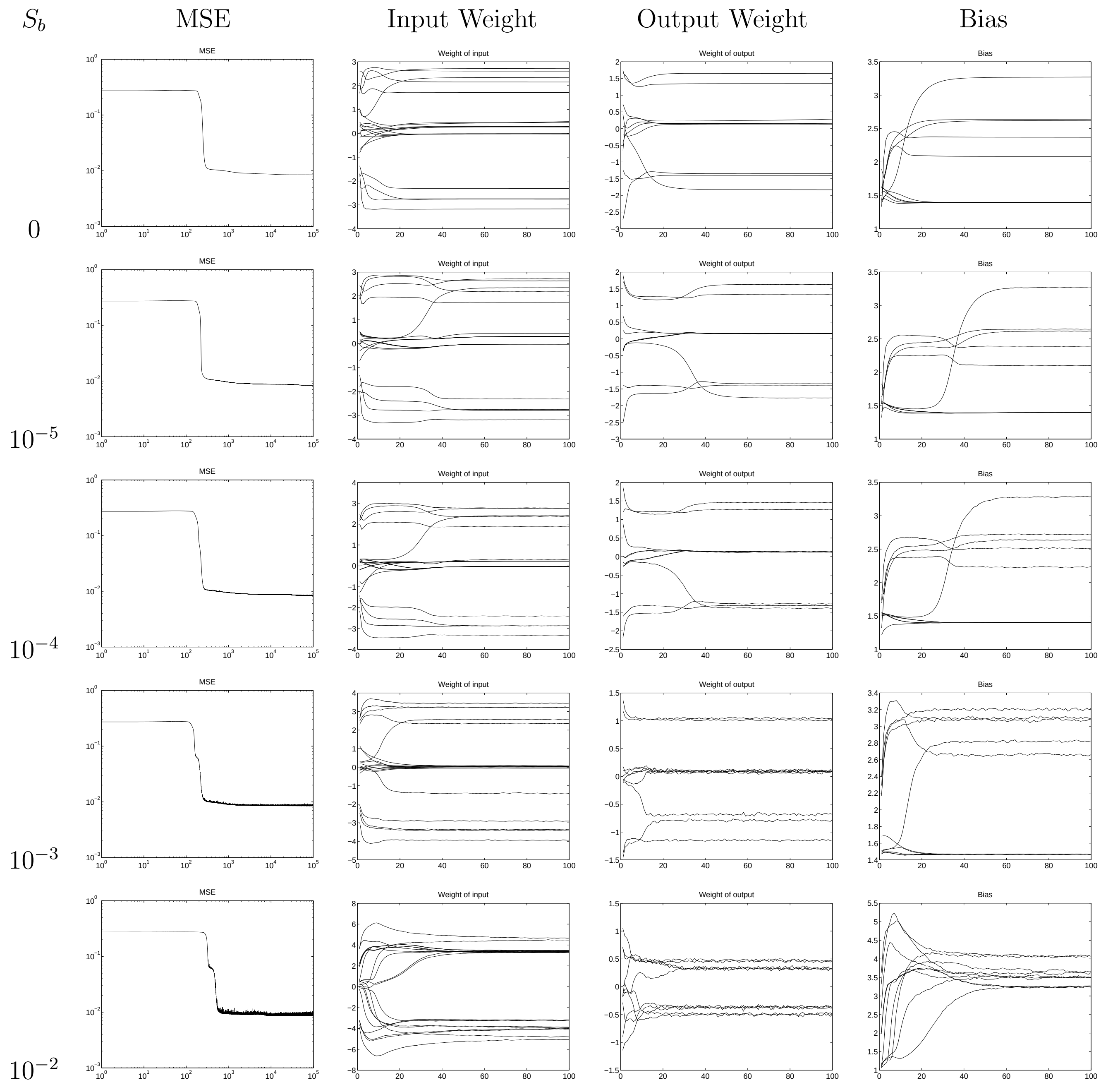

Fig. 2.2: $a = 10^{-5}$, MWN - 2D mapping.

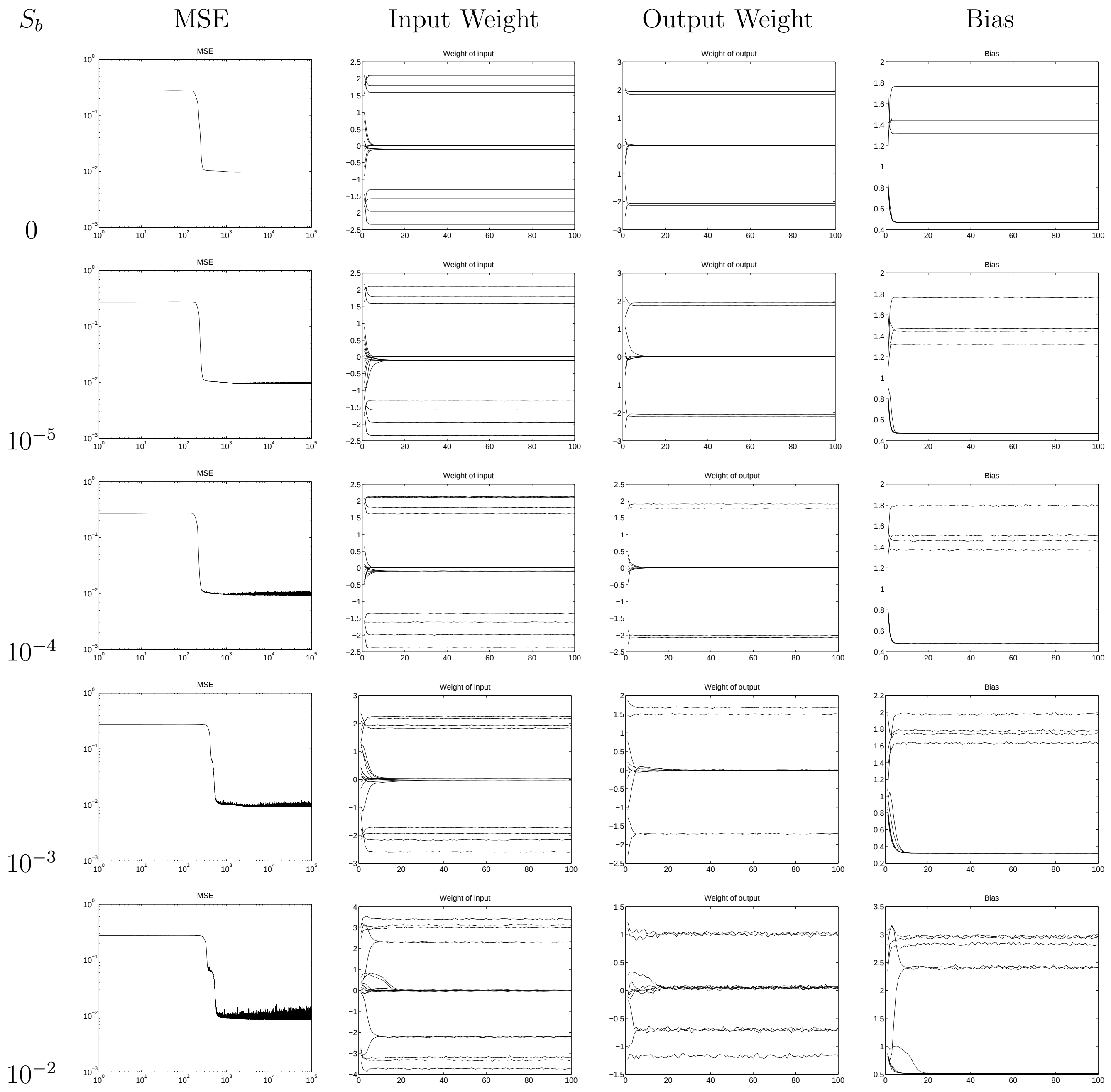

Fig. 2.3: $a = 10^{-4}$, MWN - 2D mapping.

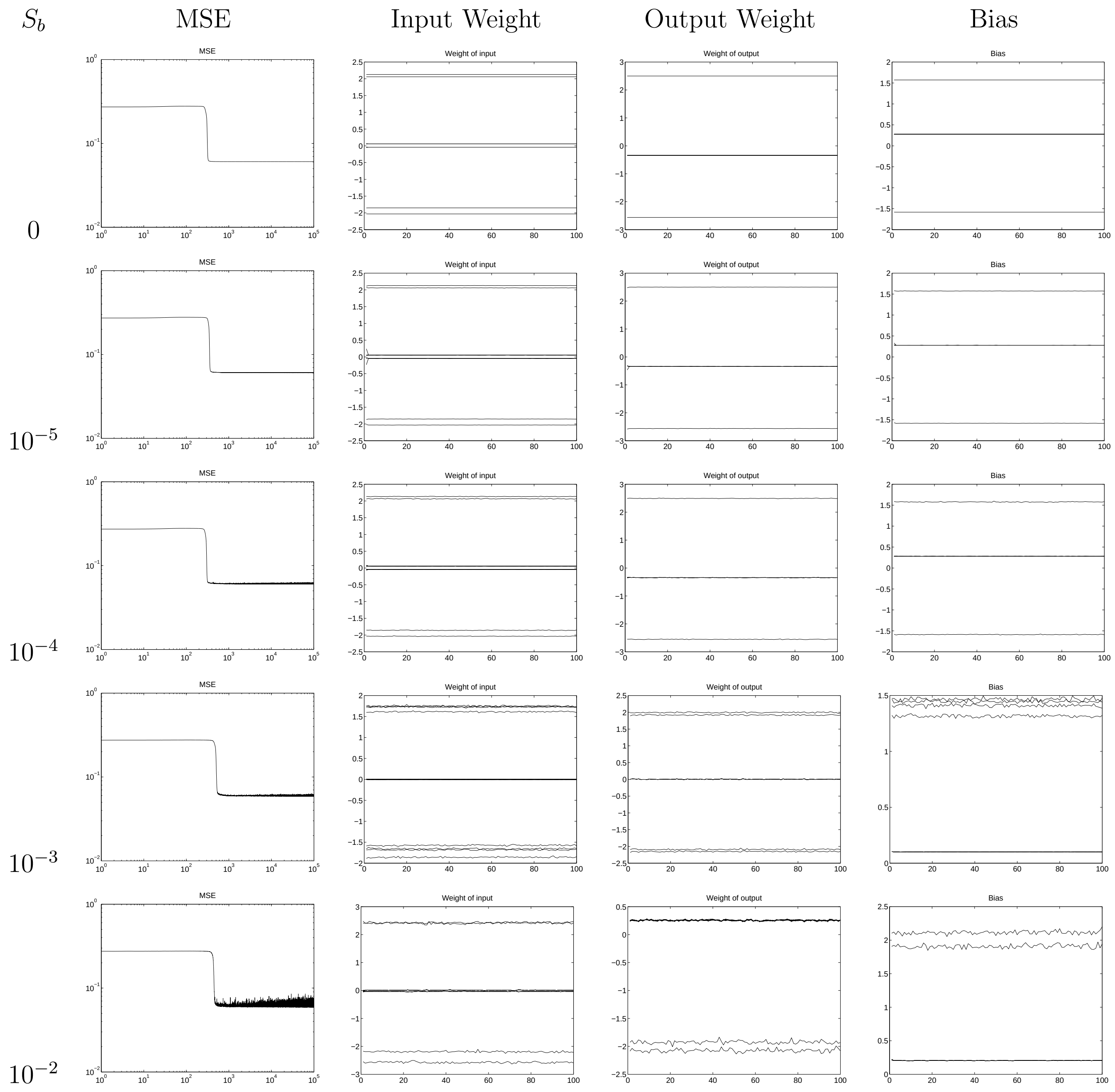

Fig. 2.4: $a = 10^{-3}$, MWN - 2D mapping.

B.1.2 AWN

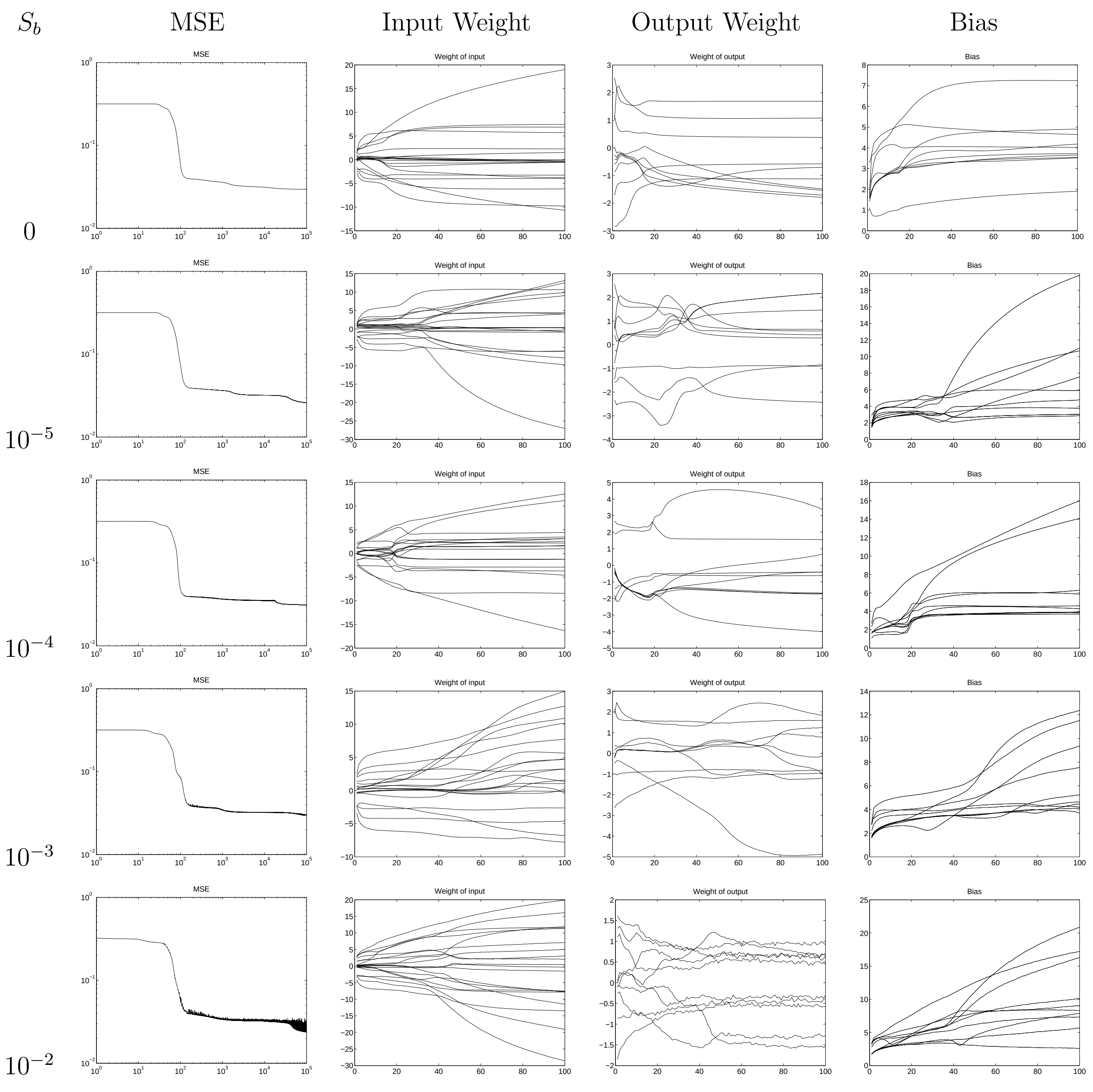

Fig. 2.5: $a = 0$, AWN - 2D mapping.

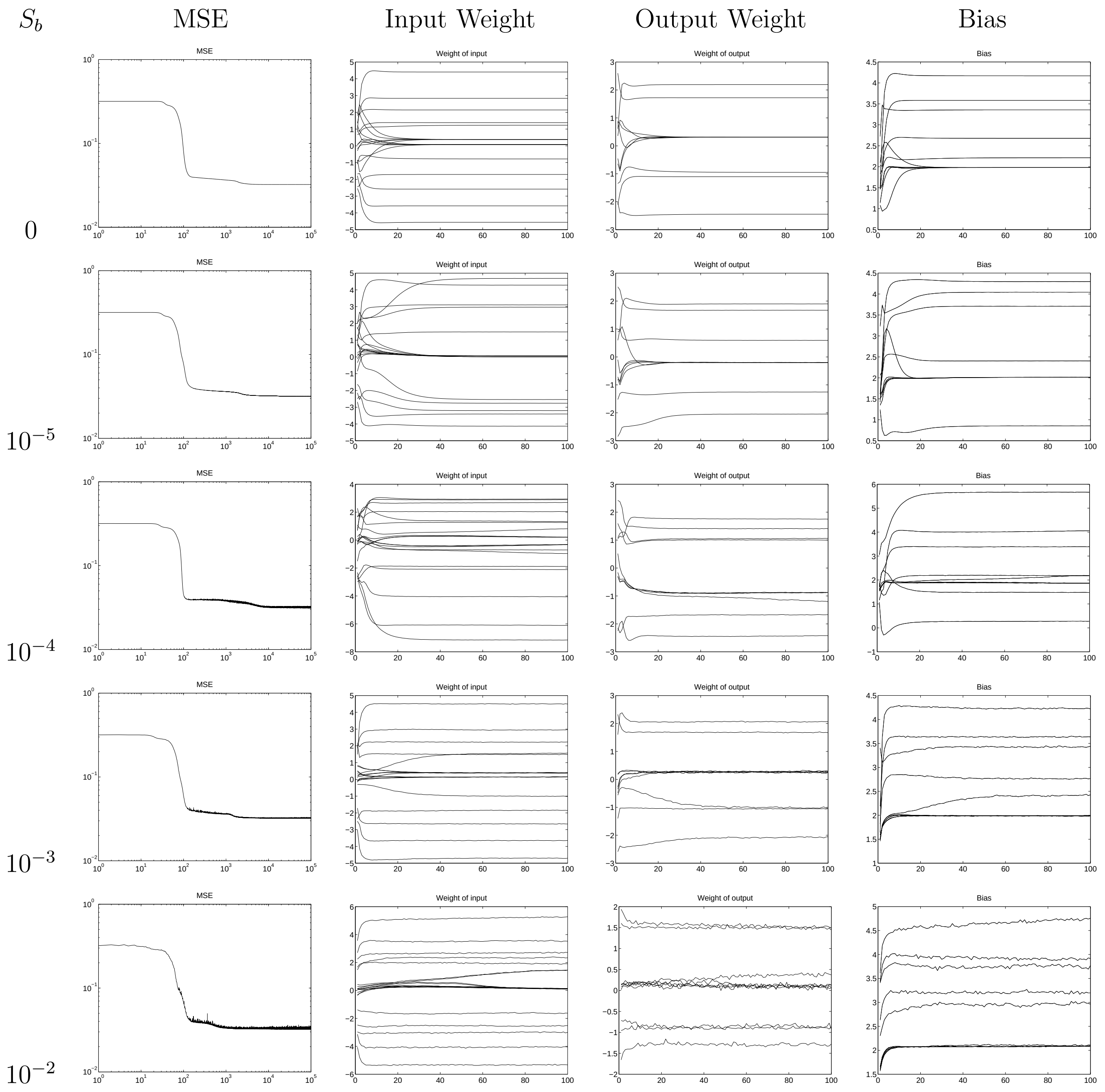

Fig. 2.6: $a = 10^{-5}$, AWN - 2D mapping.

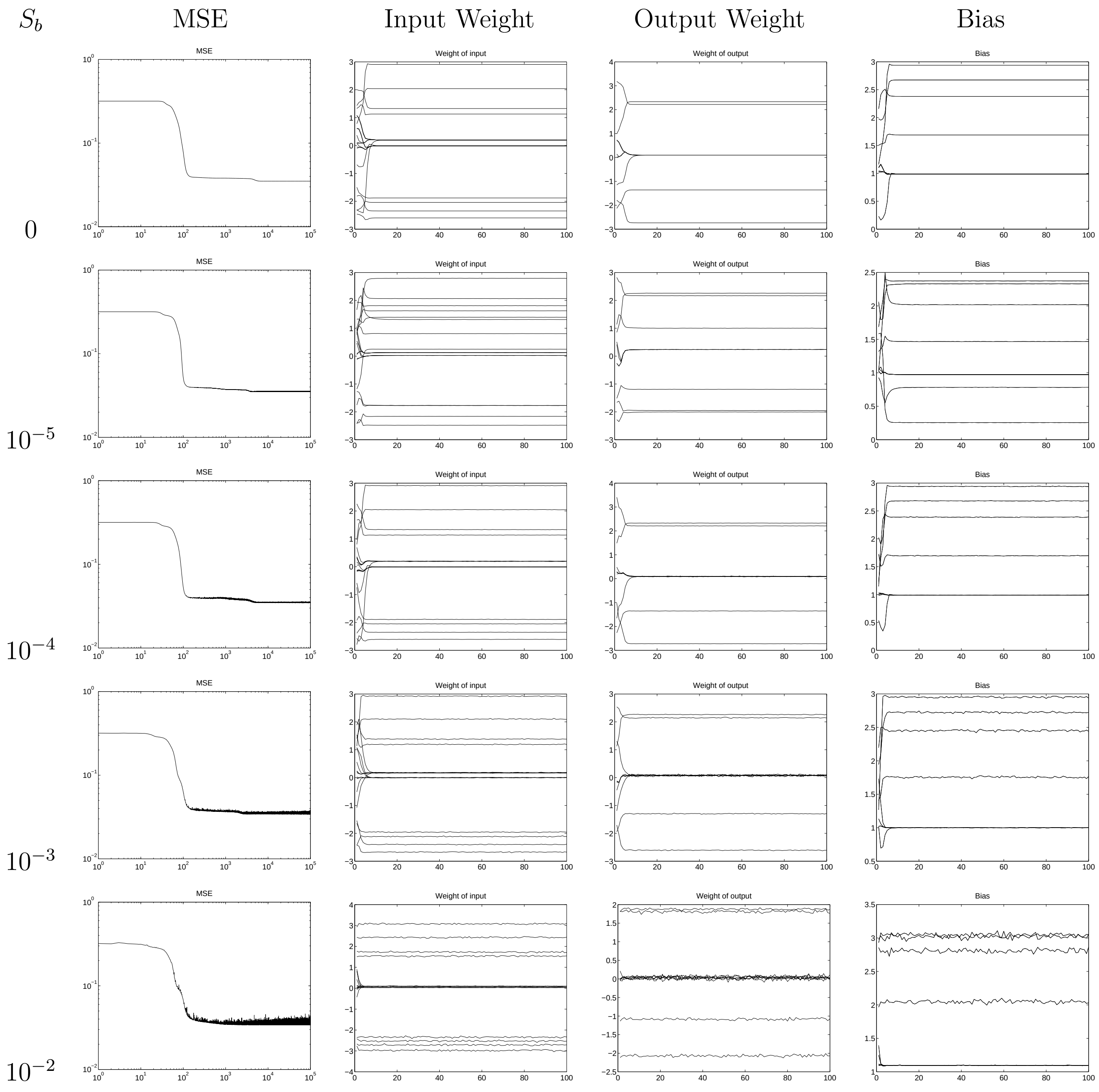

Fig. 2.7: $a = 10^{-4}$, AWN - 2D mapping.

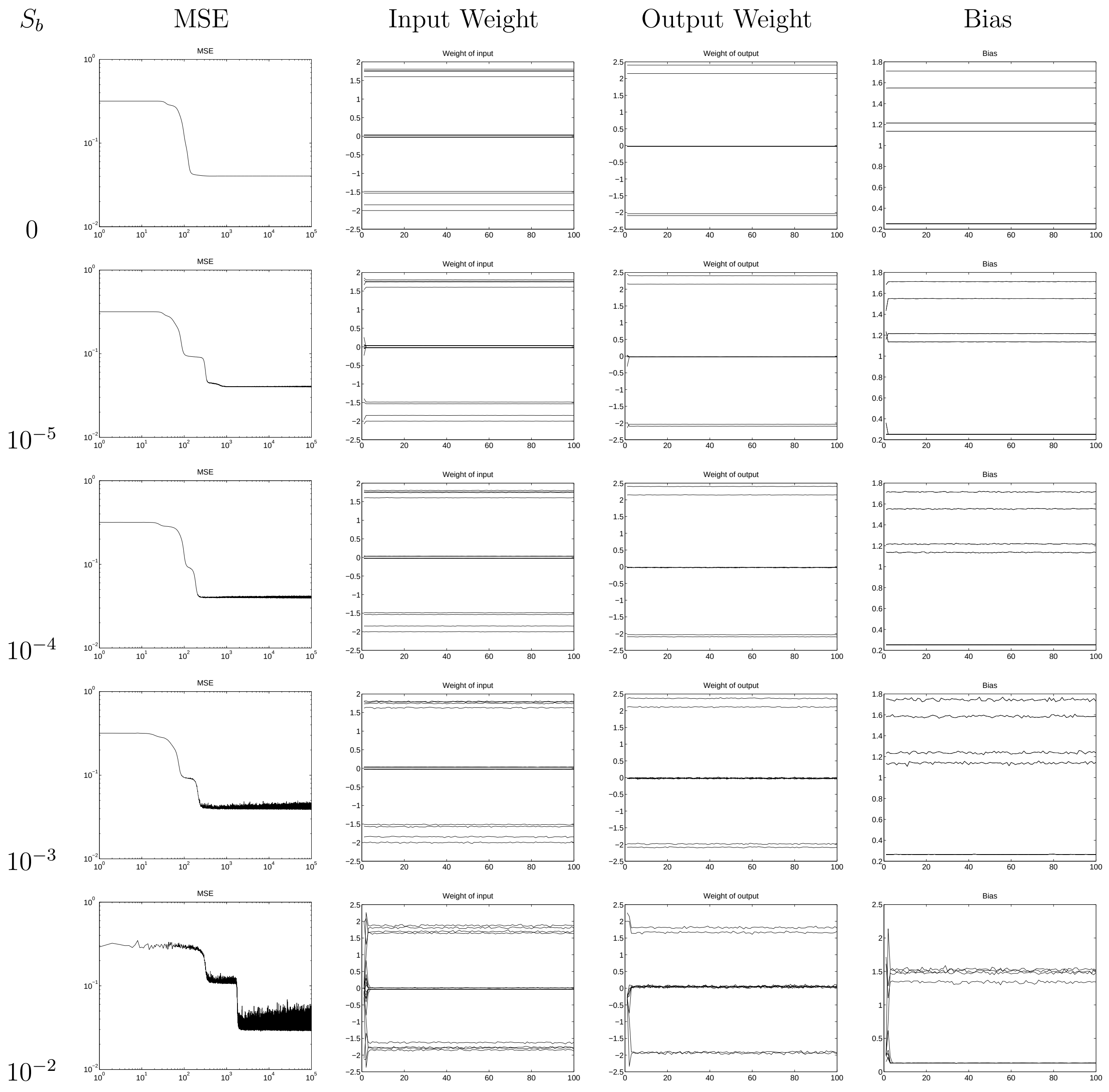

Fig. 2.8: $a = 10^{-3}$, AWN - 2D mapping.

B.2 Mackey-Glass

B.2.1 MWN

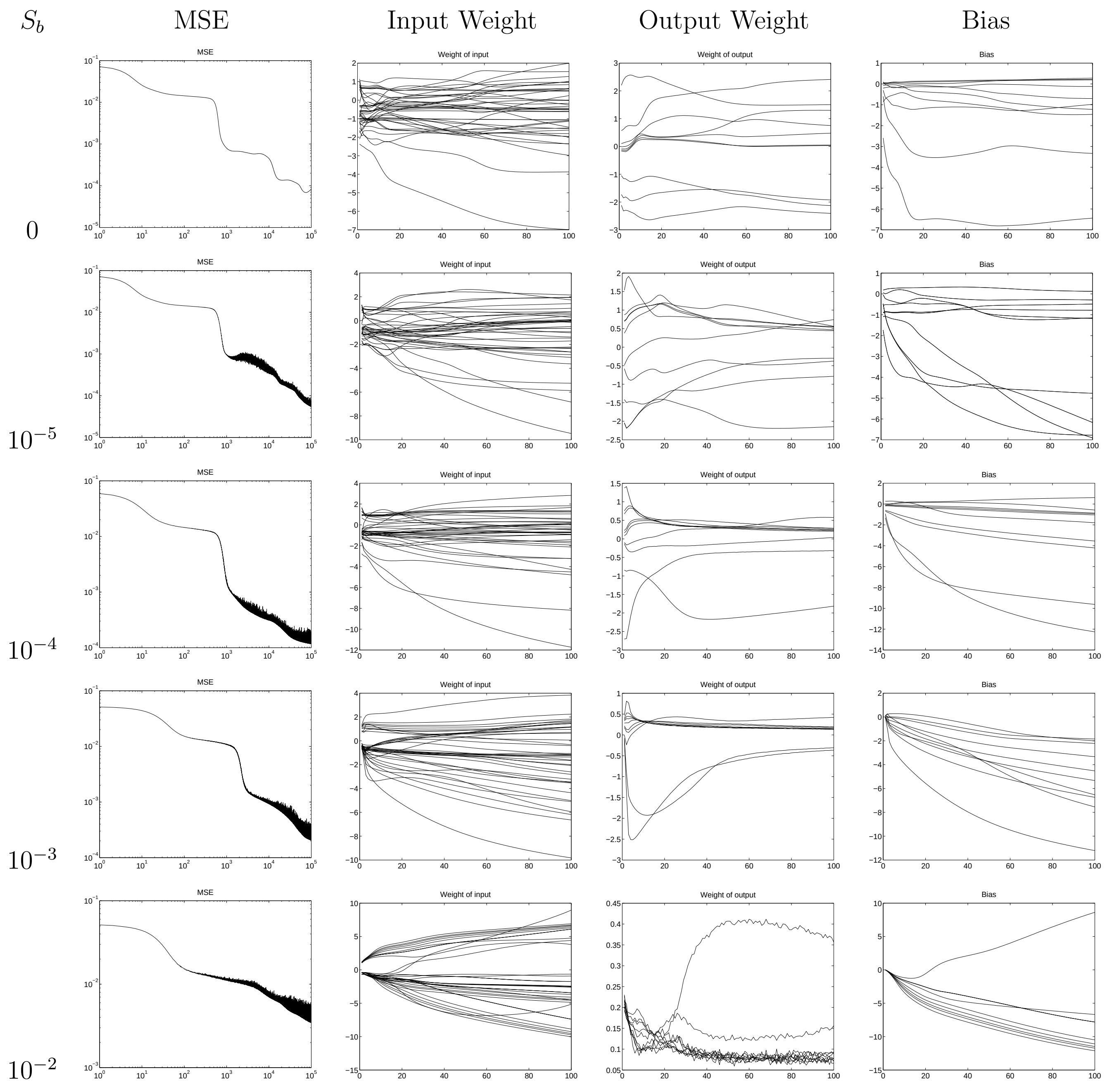

Fig. 2.9: $a = 0$, MWN - Mackey-Glass.

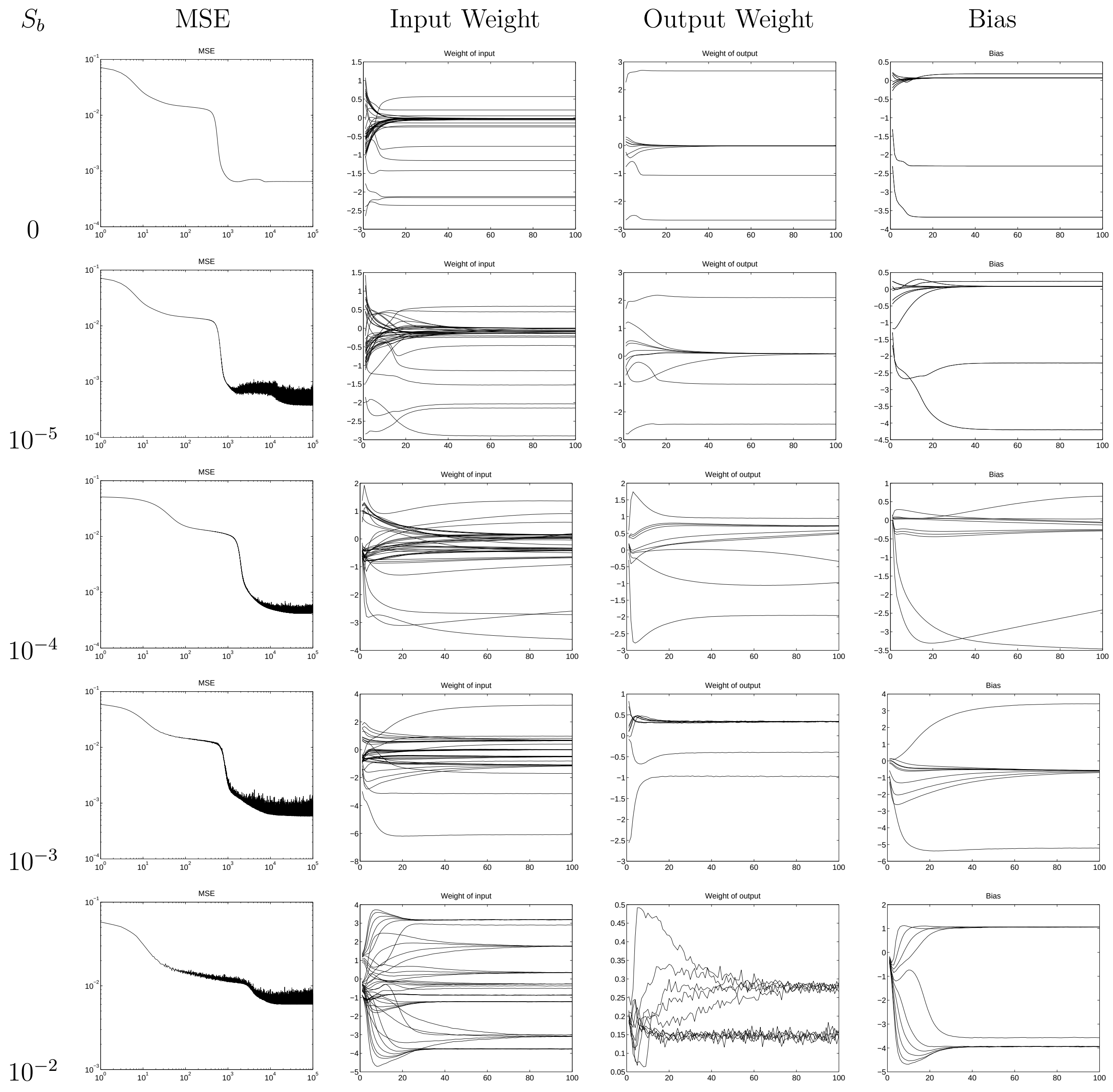

Fig. 2.10: $a = 10^{-5}$, MWN - Mackey-Glass.

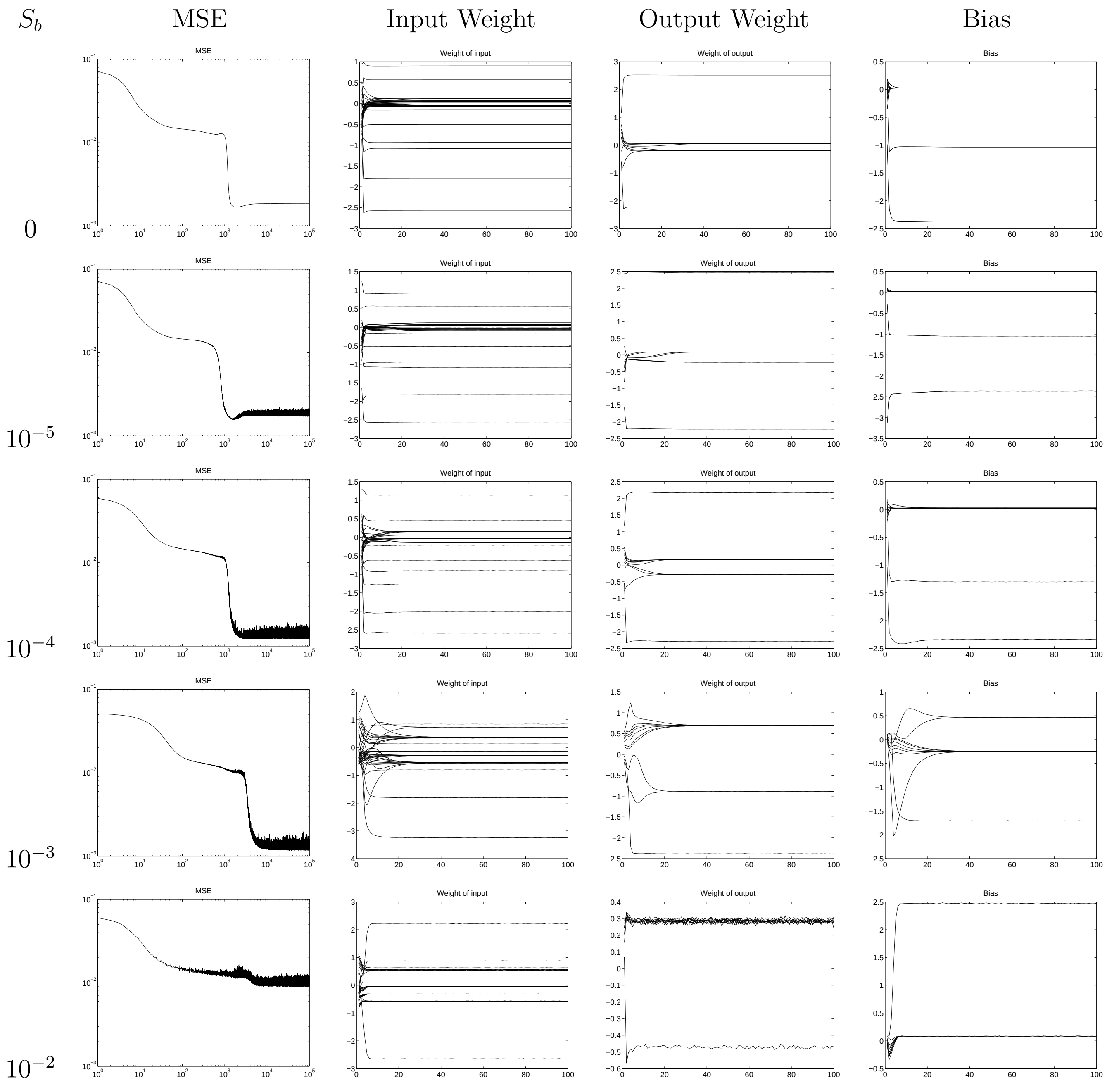

Fig. 2.11: $a = 10^{-4}$, MWN - Mackey-Glass.

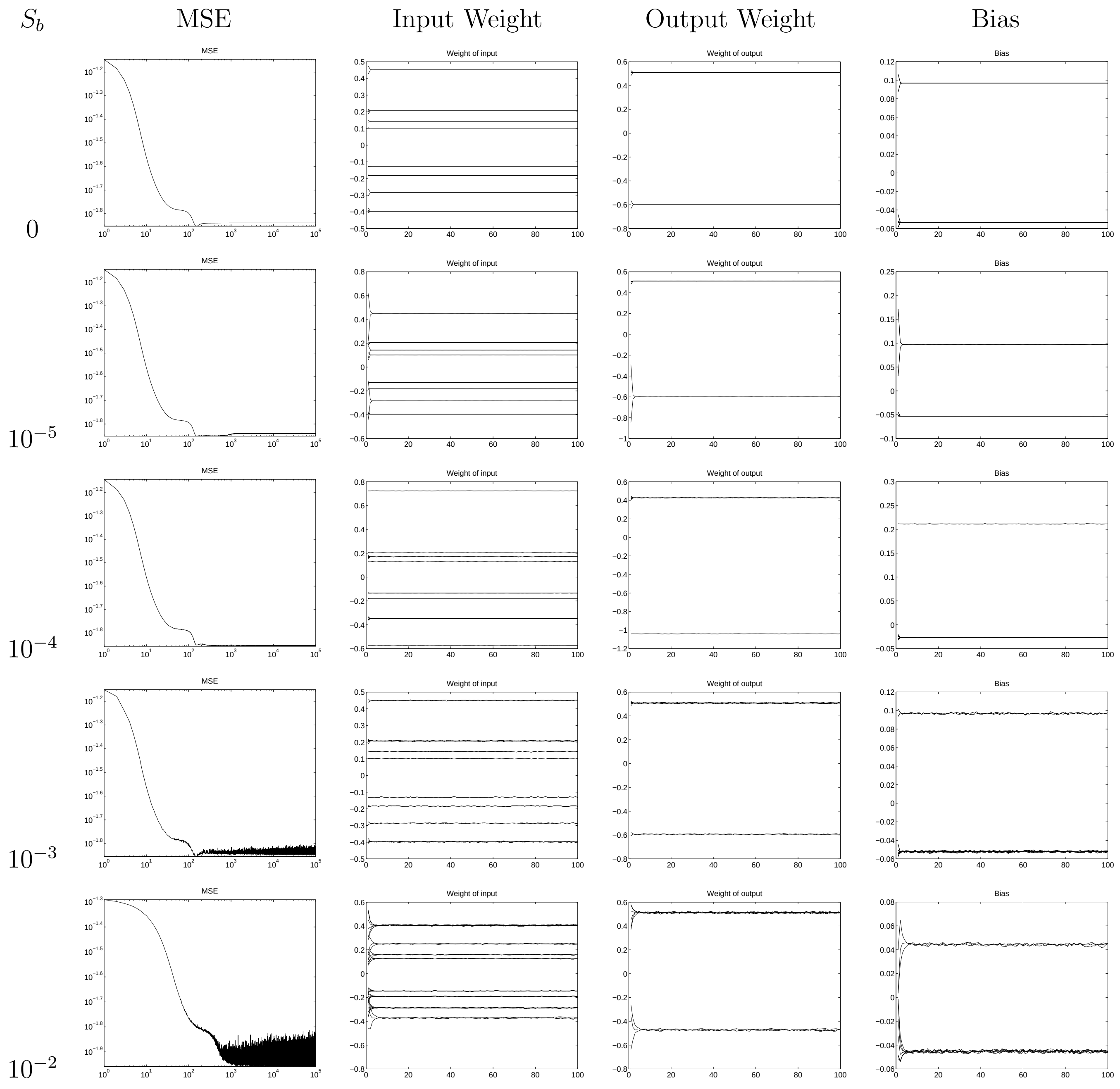

Fig. 2.12: $a = 10^{-3}$, MWN - Mackey-Glass.

B.2.2 AWN

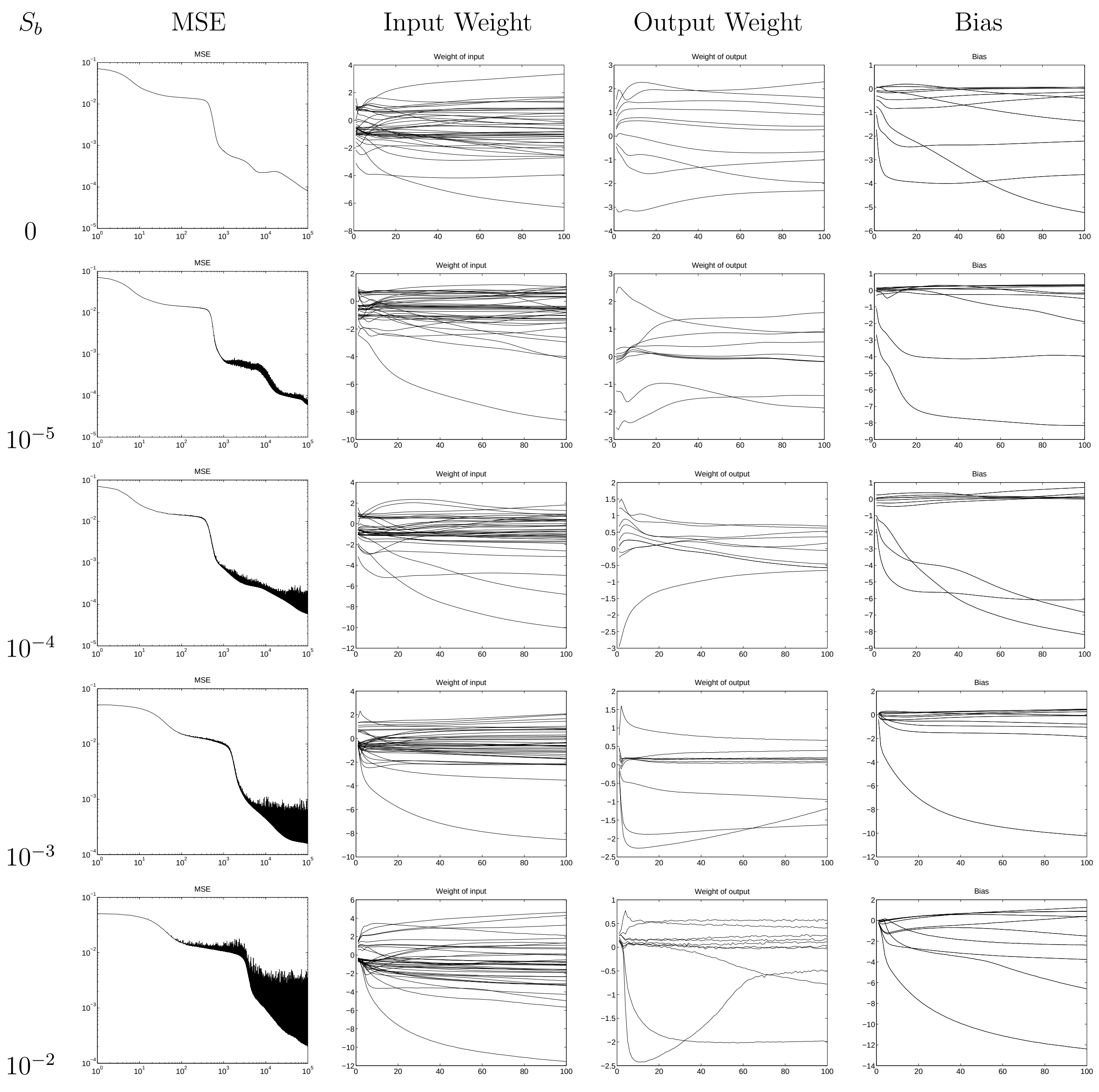

Fig. 2.13: $a = 0$, AWN - Mackey-Glass.

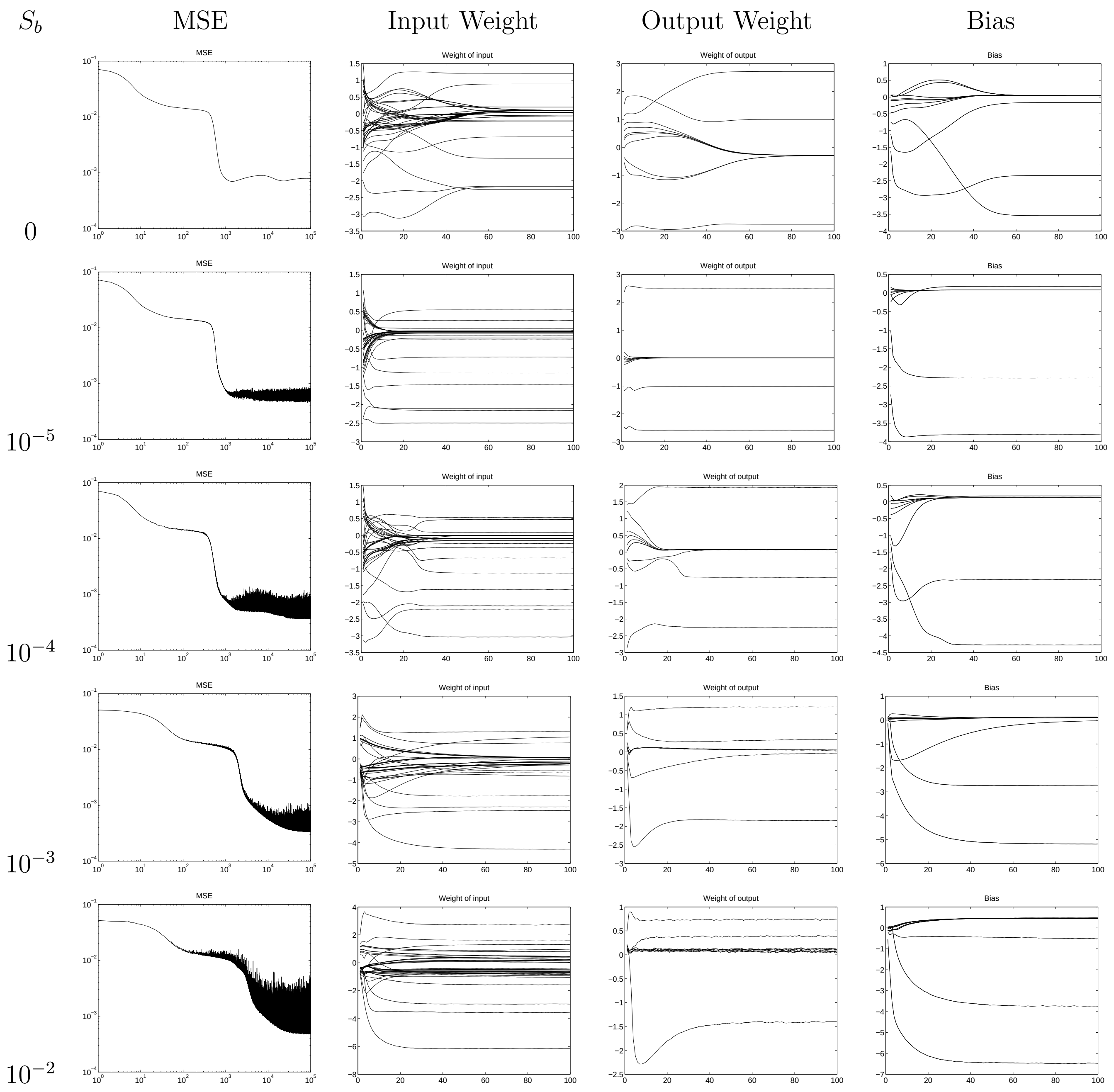

Fig. 2.14: $a = 10^{-5}$, AWN - Mackey-Glass.

S_b MSE Input Weight Output Weight Bias

0

10^{-5}

10^{-4}

10^{-3}

10^{-2}

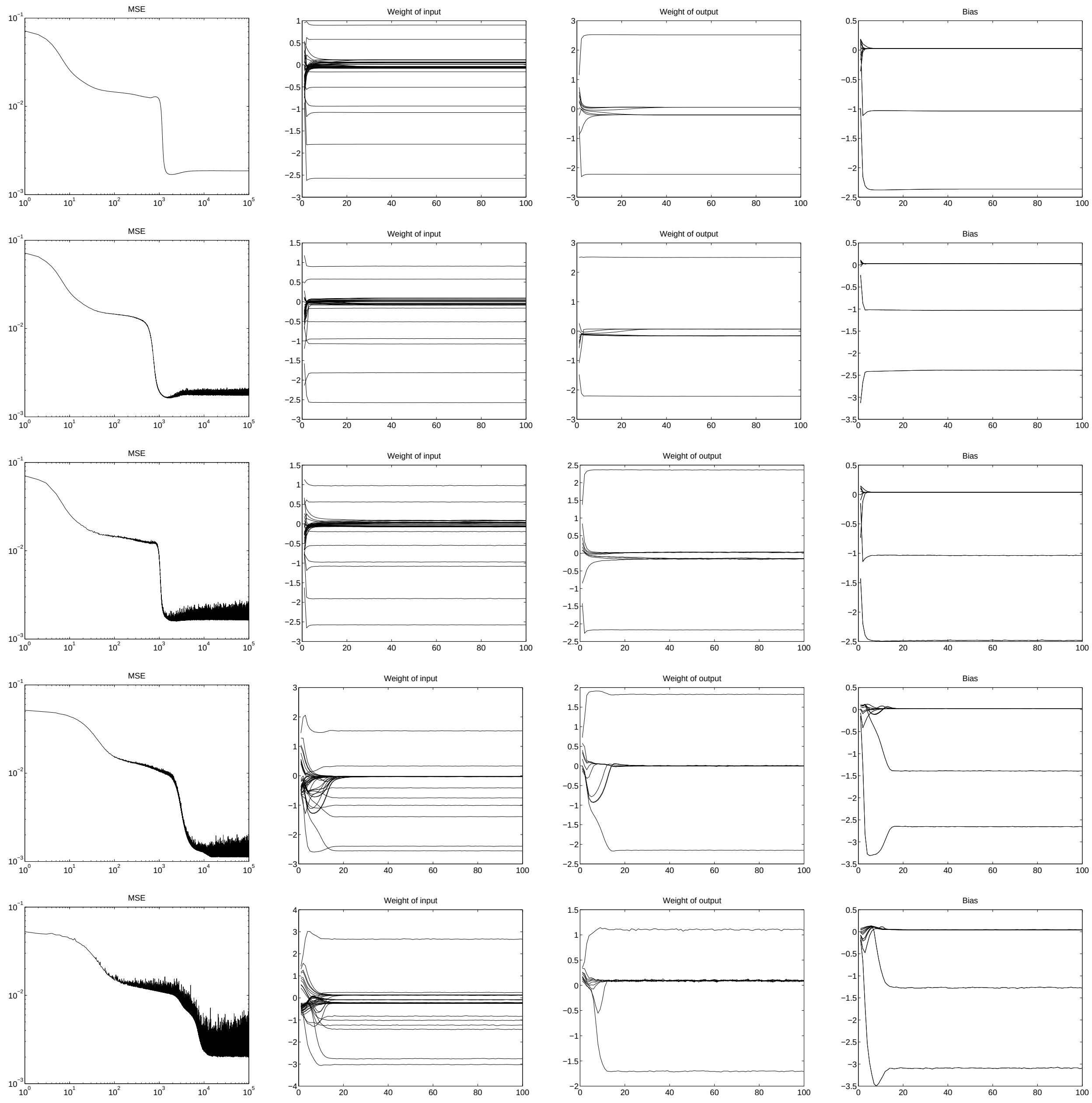

Fig. 2.15: $a = 10^{-4}$, AWN - Mackey-Glass.

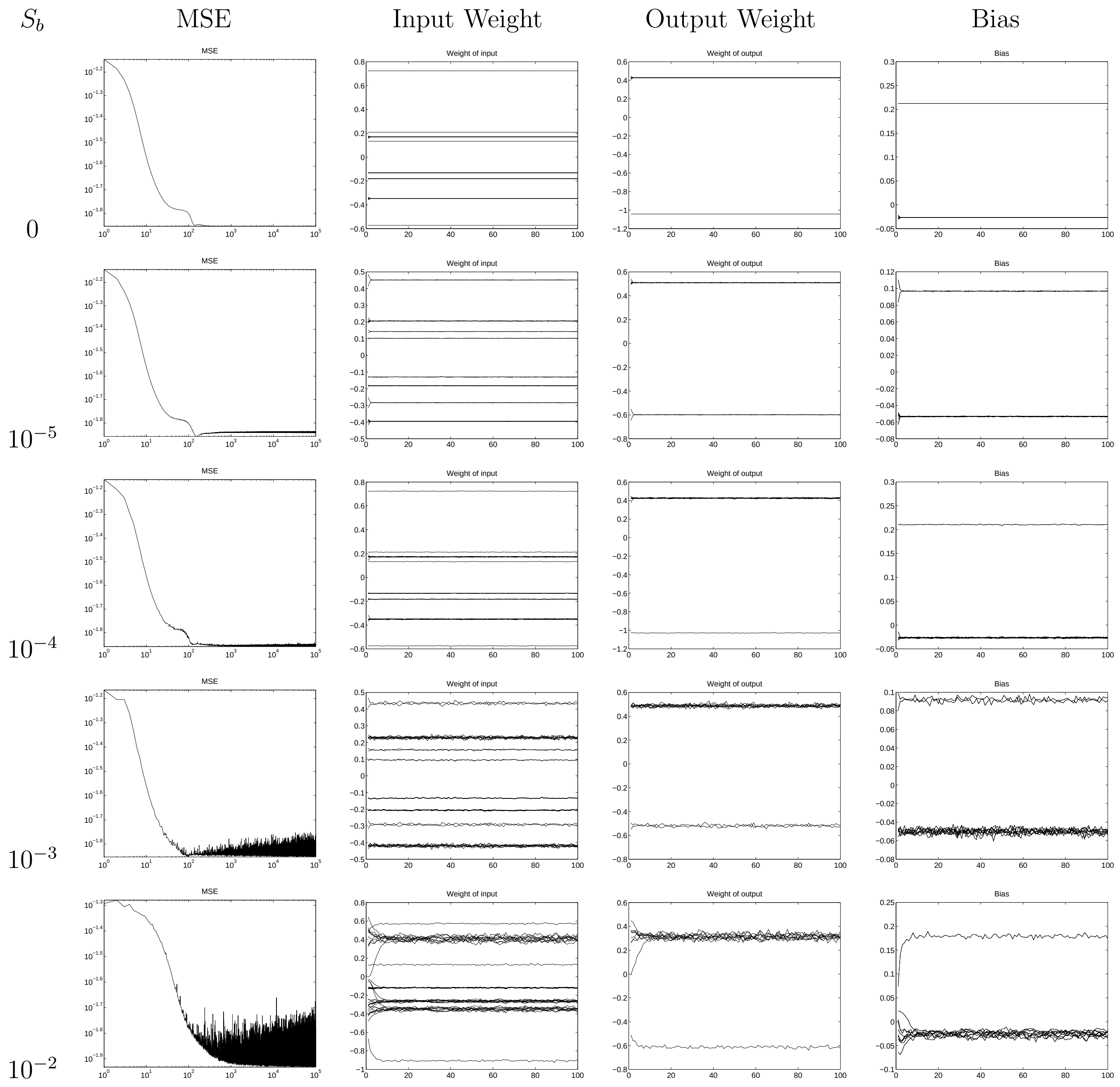

Fig. 2.16: $a = 10^{-3}$, AWN - Mackey-Glass.

B.3 NAR

B.3.1 MWN

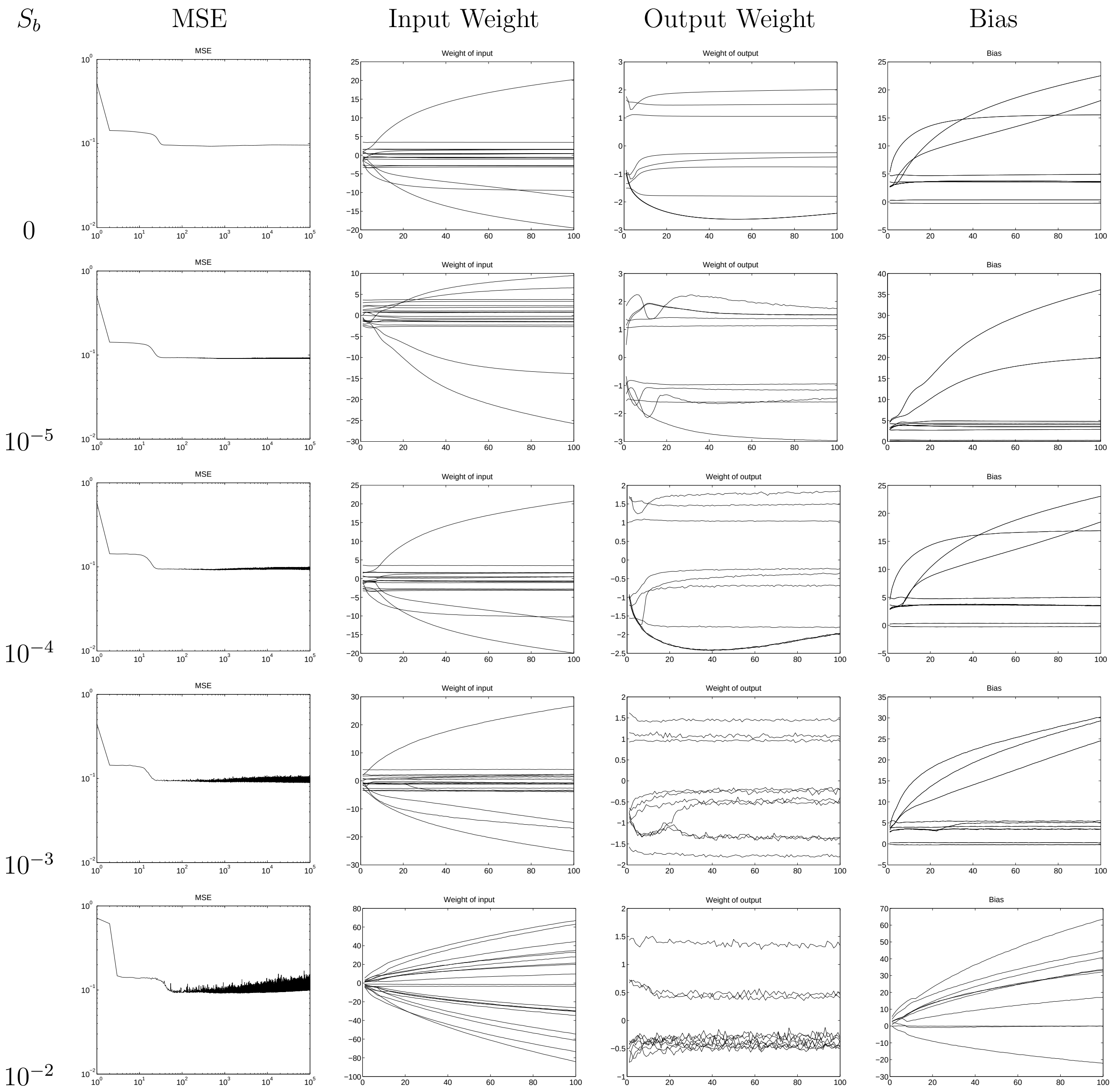

Fig. 2.17: $a = 0$, MWN - NAR.

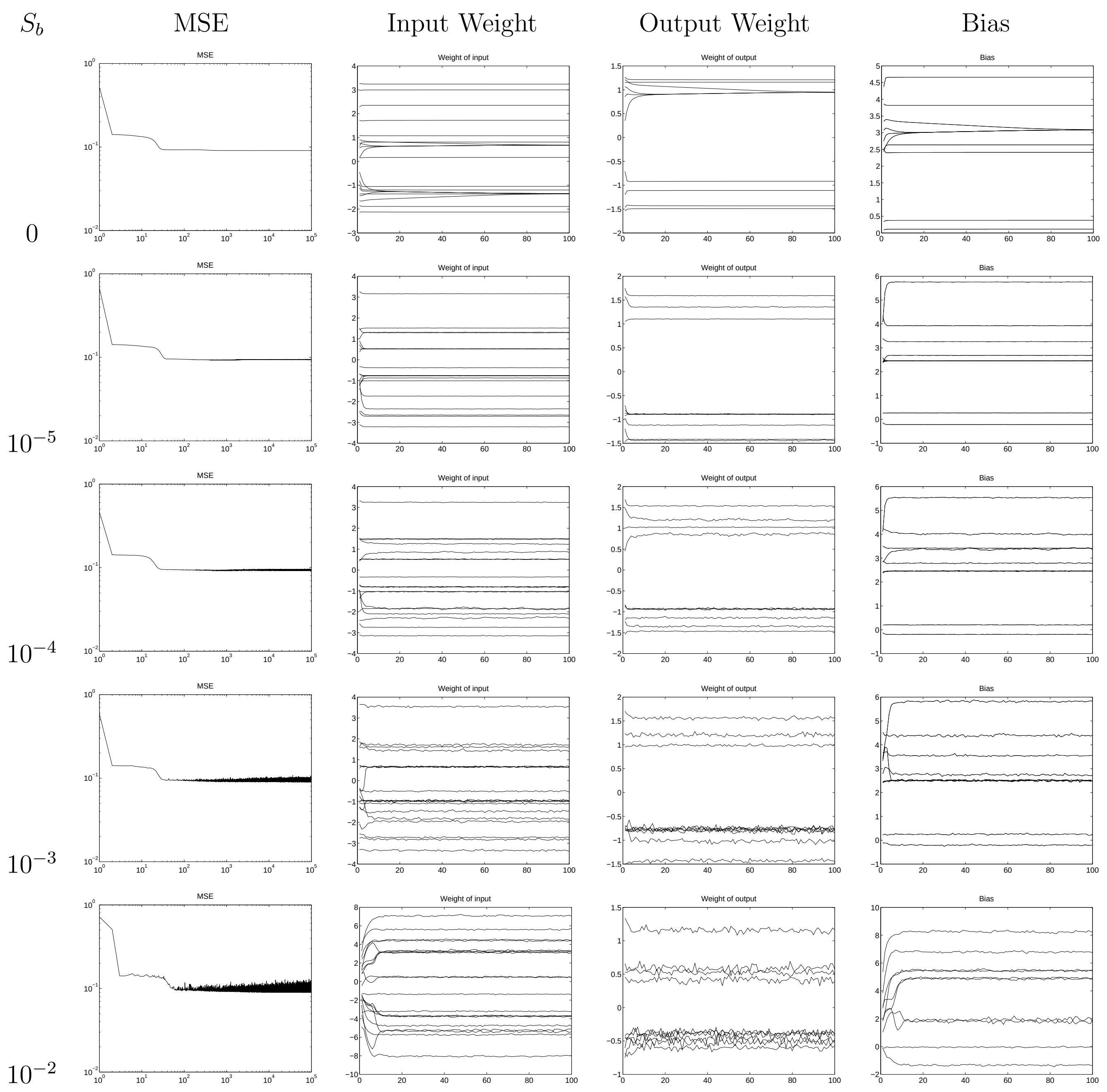

Fig. 2.18: $a = 10^{-5}$, MWN - NAR.

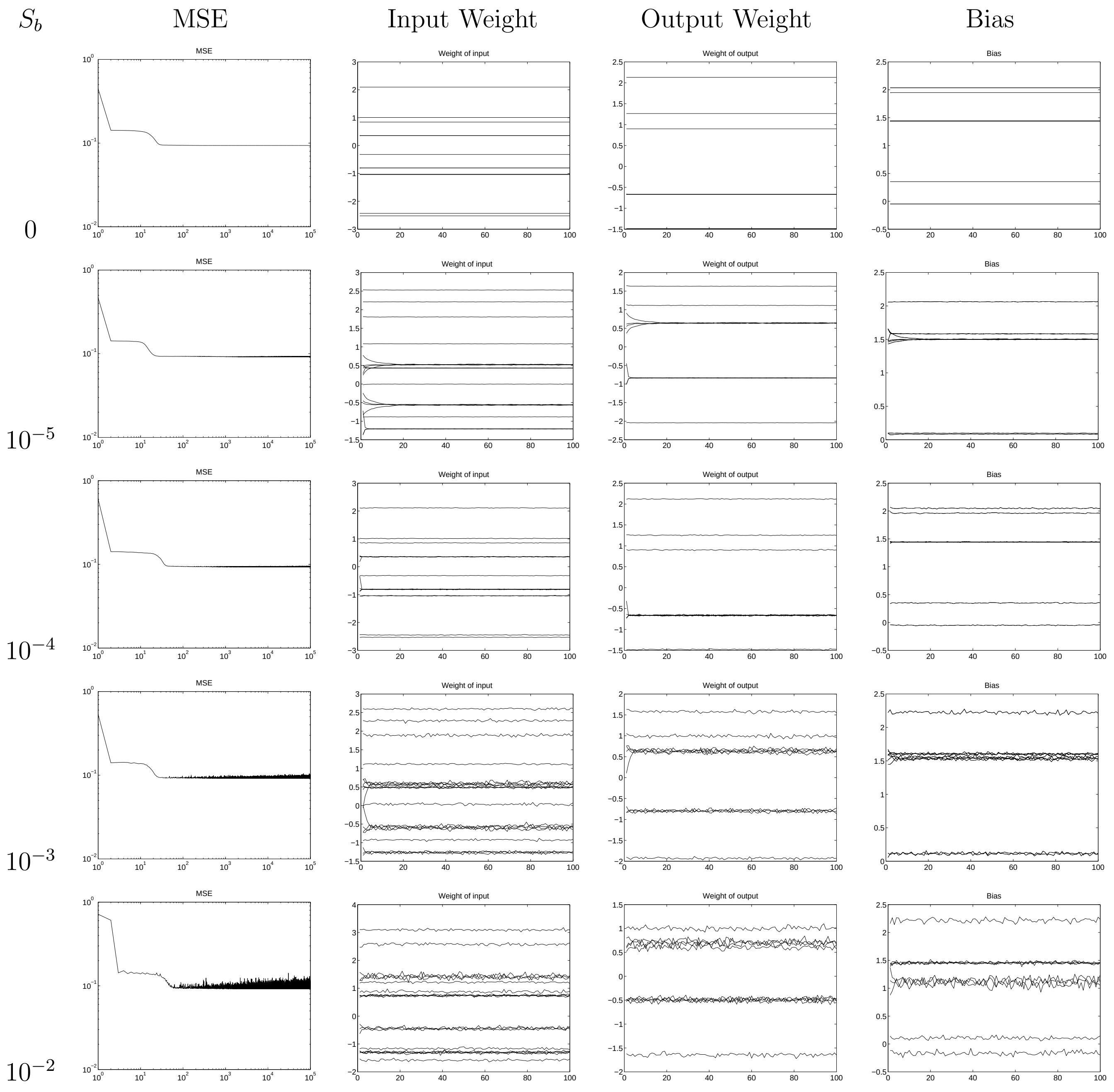

Fig. 2.19: $a = 10^{-4}$, MWN - NAR.

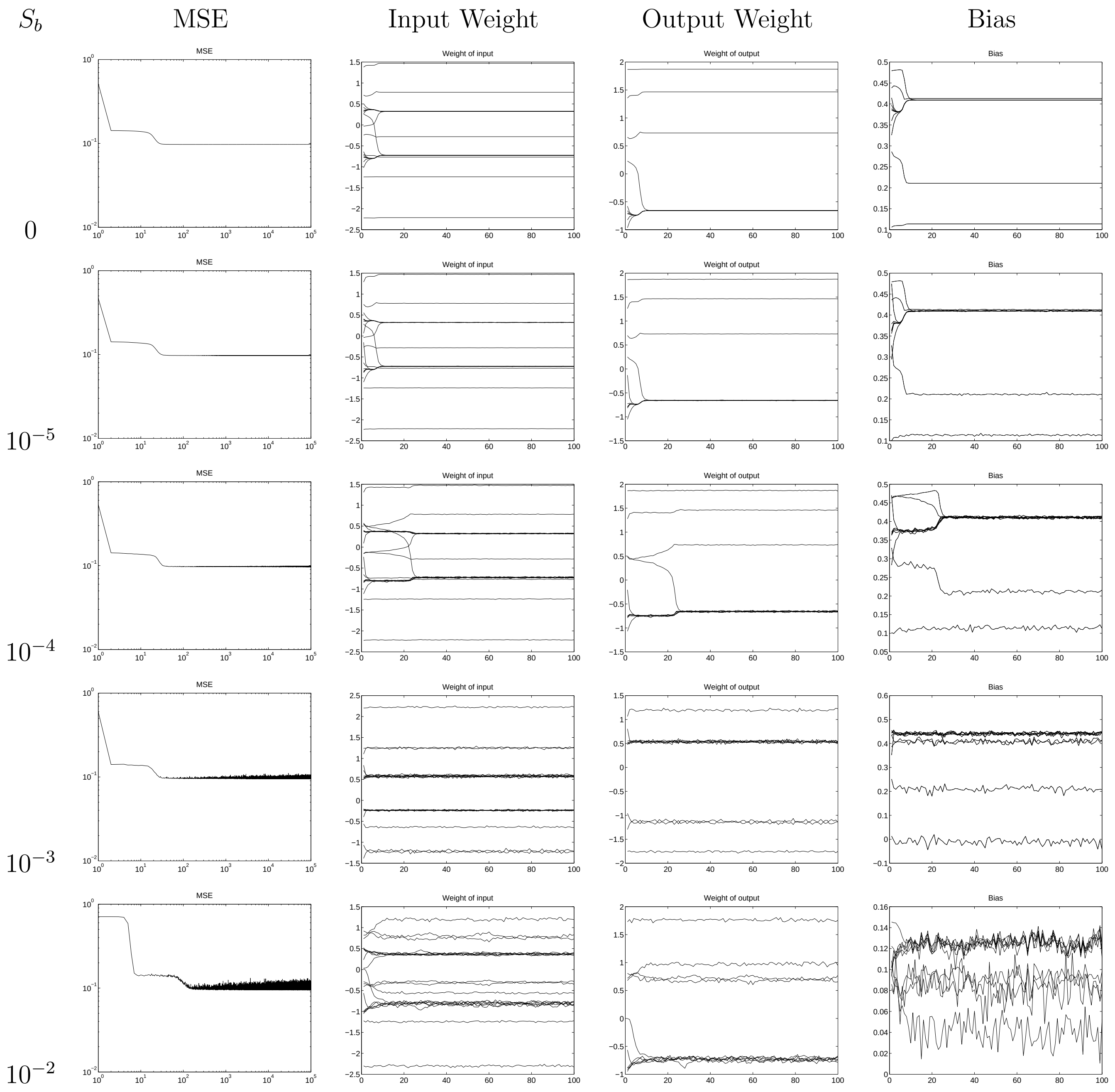

Fig. 2.20: $a = 10^{-3}$, MWN - NAR.

B.3.2 AWN

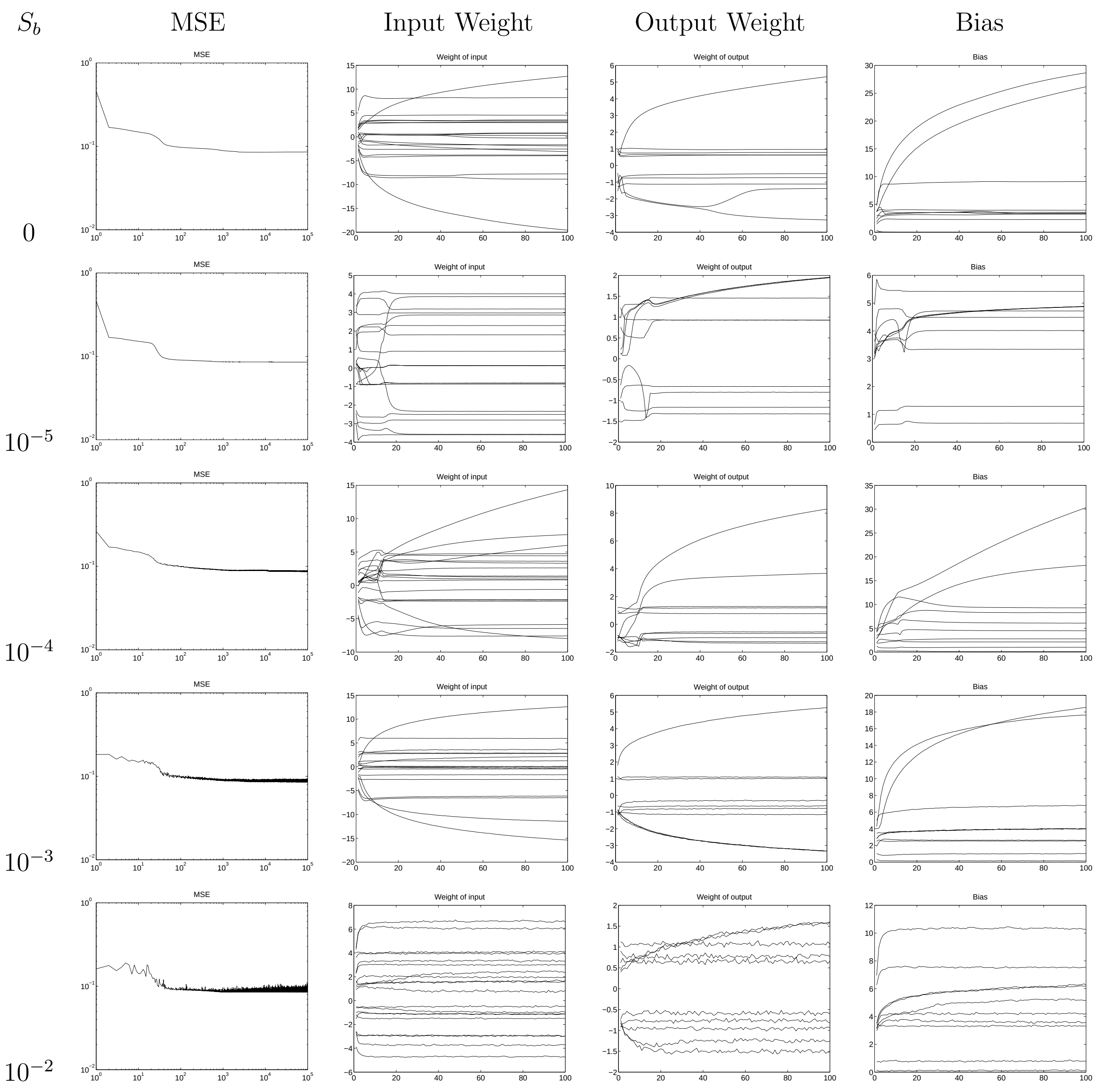

Fig. 2.21: $a = 0$, AWN - NAR.

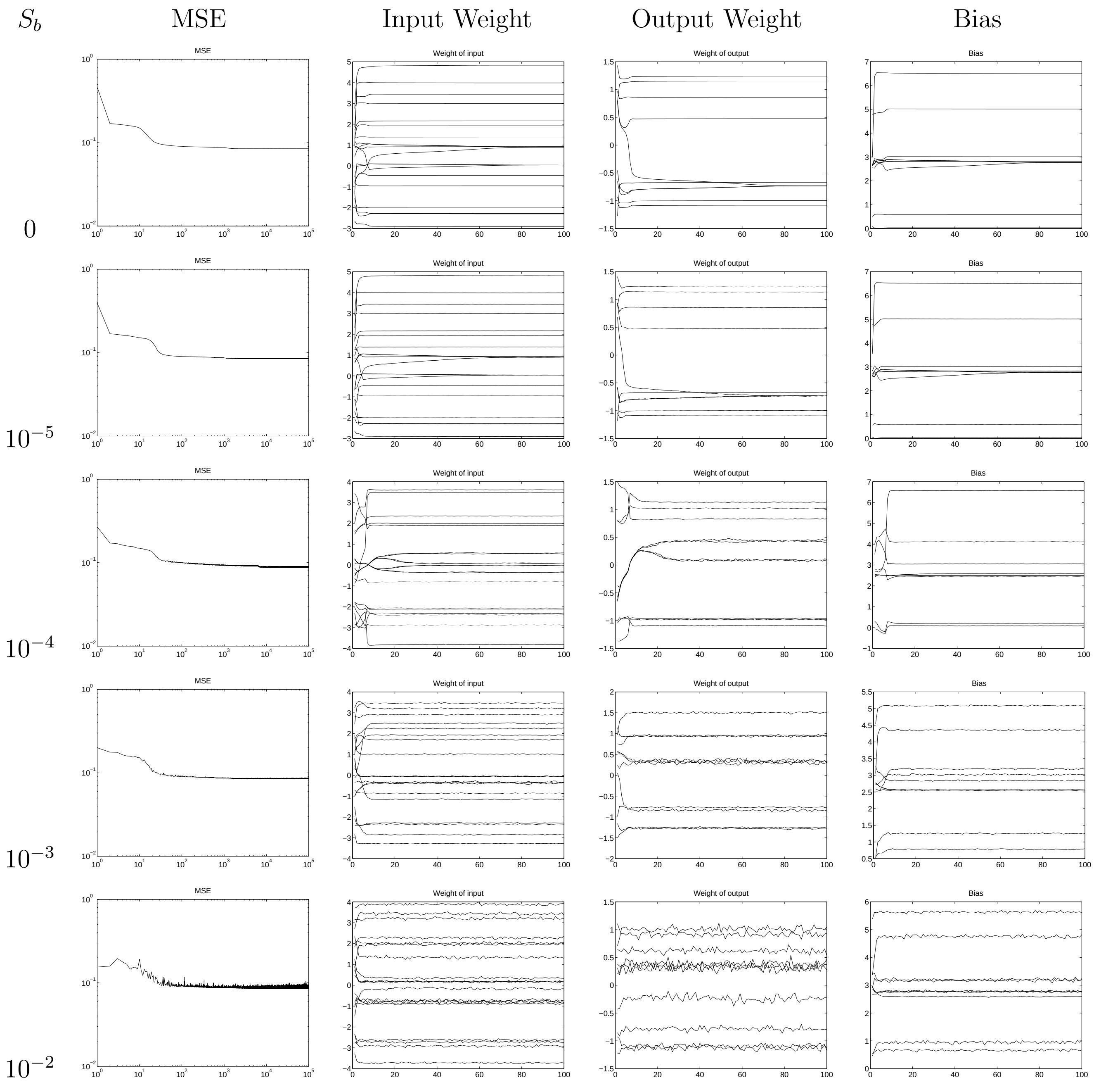

Fig. 2.22: $a = 10^{-5}$, AWN - NAR.

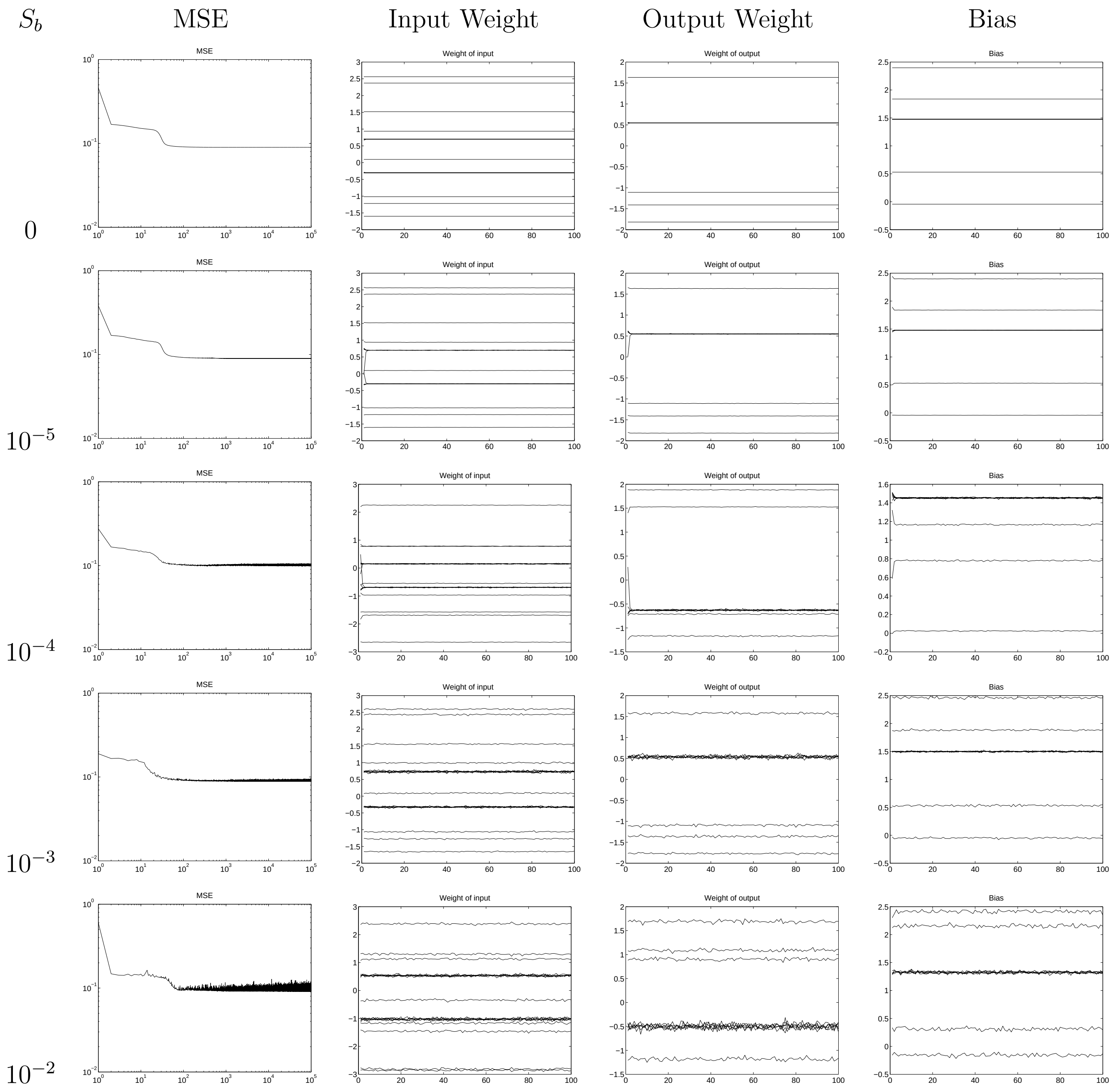

Fig. 2.23: $a = 10^{-4}$, AWN - NAR.

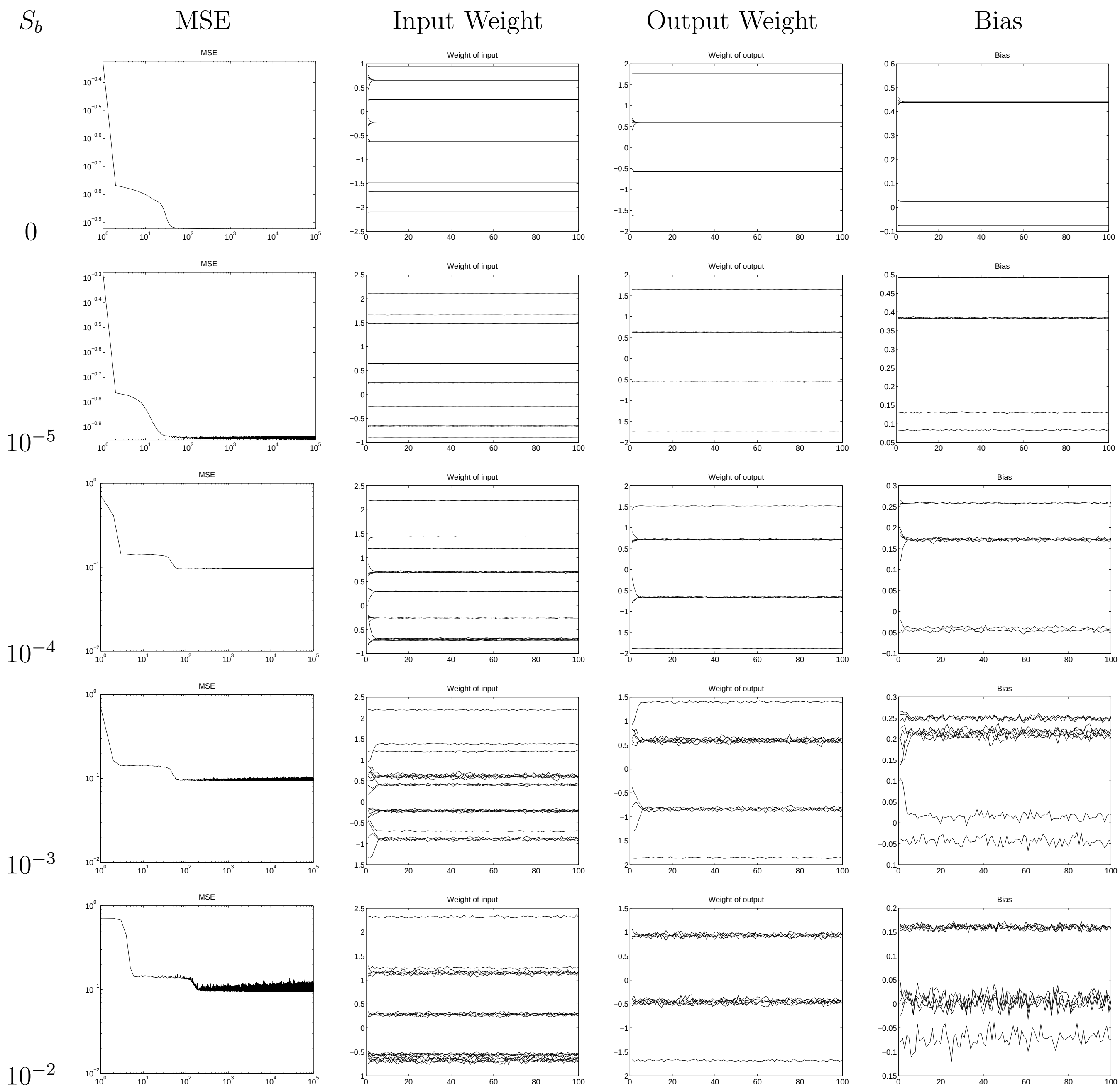

Fig. 2.24: $a = 10^{-3}$, AWN - NAR.

B.4 Astrophysical data

B.4.1 MWN

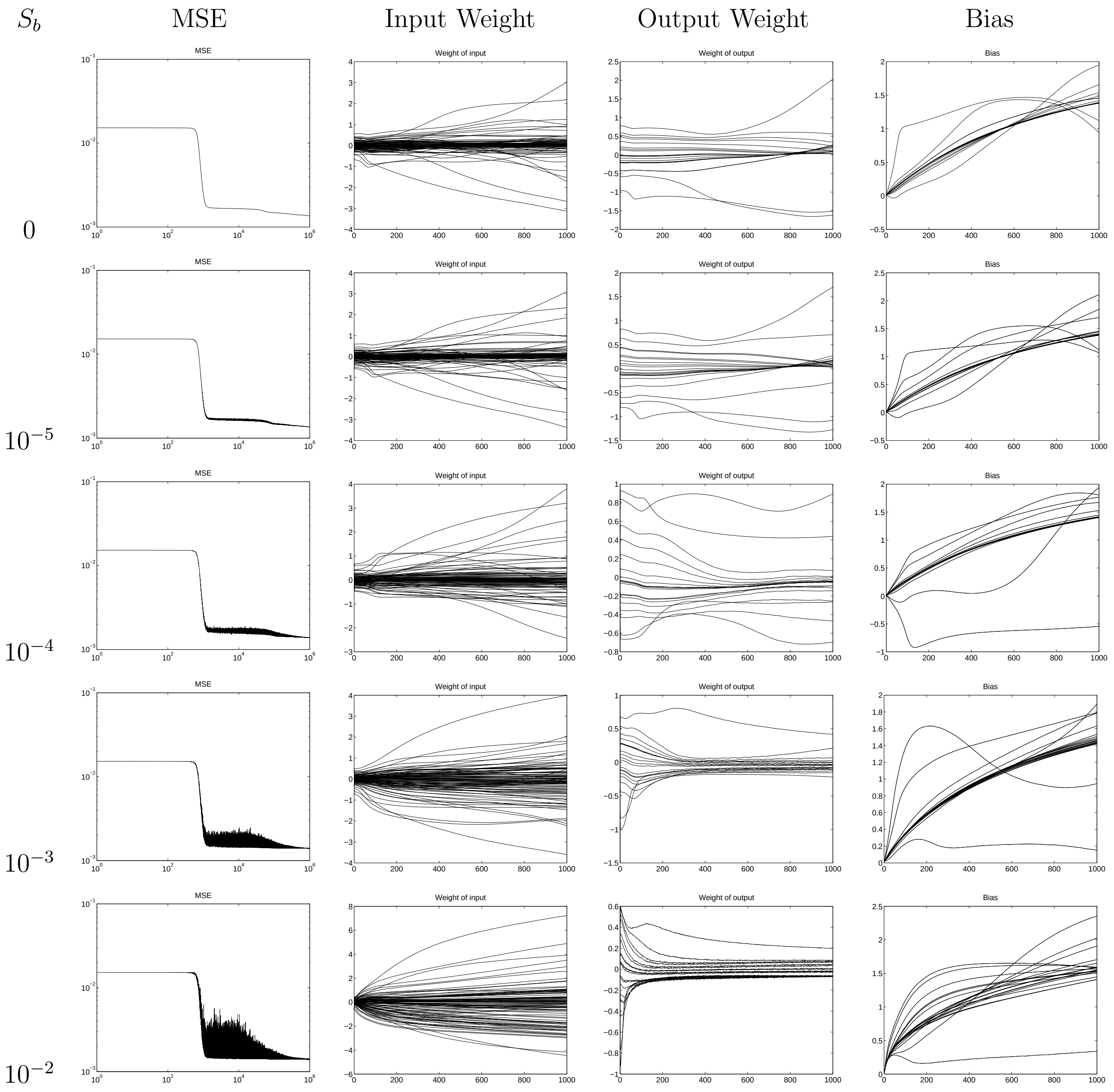

Fig. 2.25: $a = 0$, MWN - Astrophysical data.

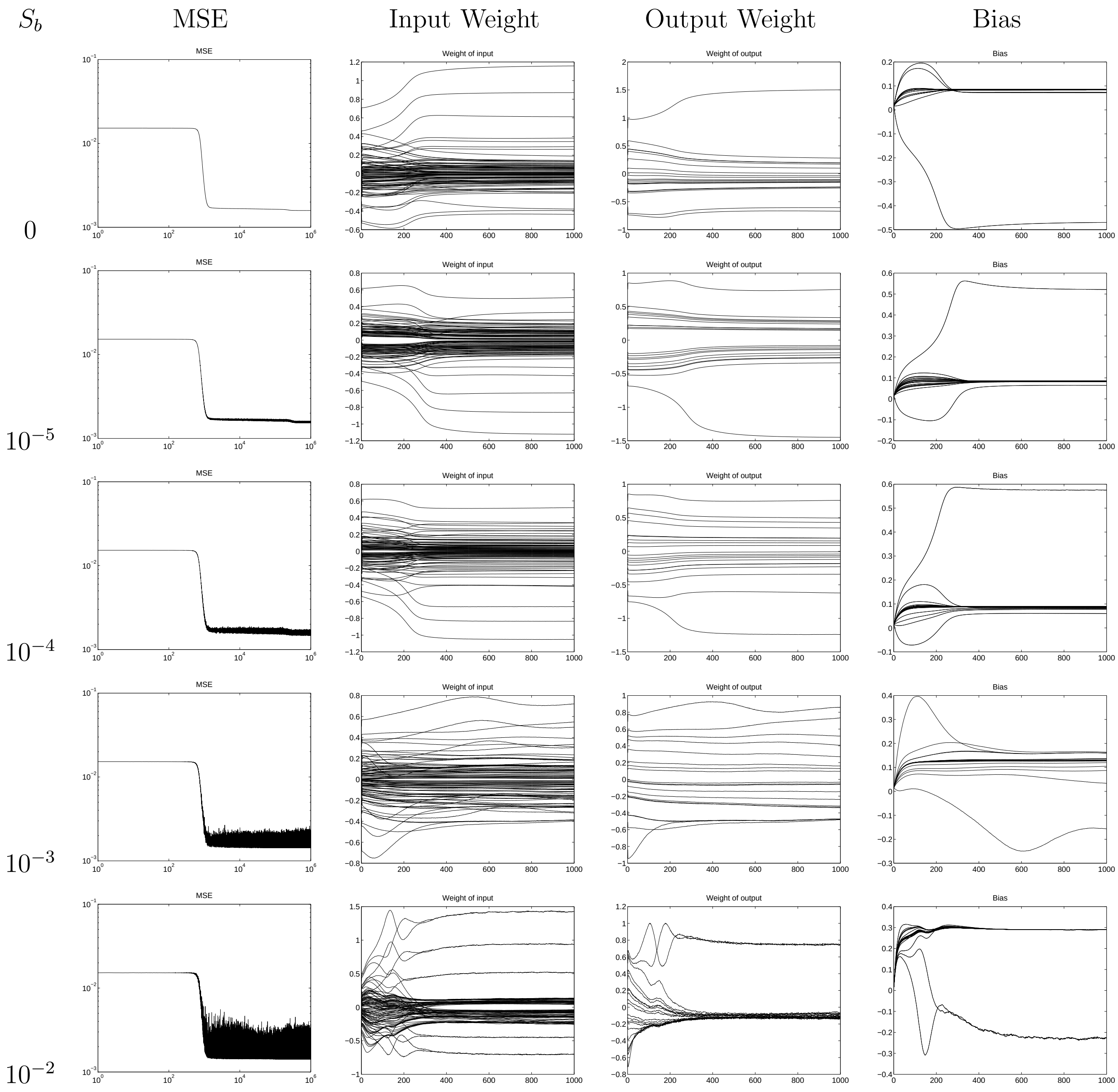

Fig. 2.26: $a = 10^{-5}$, MWN - Astrophysical data.

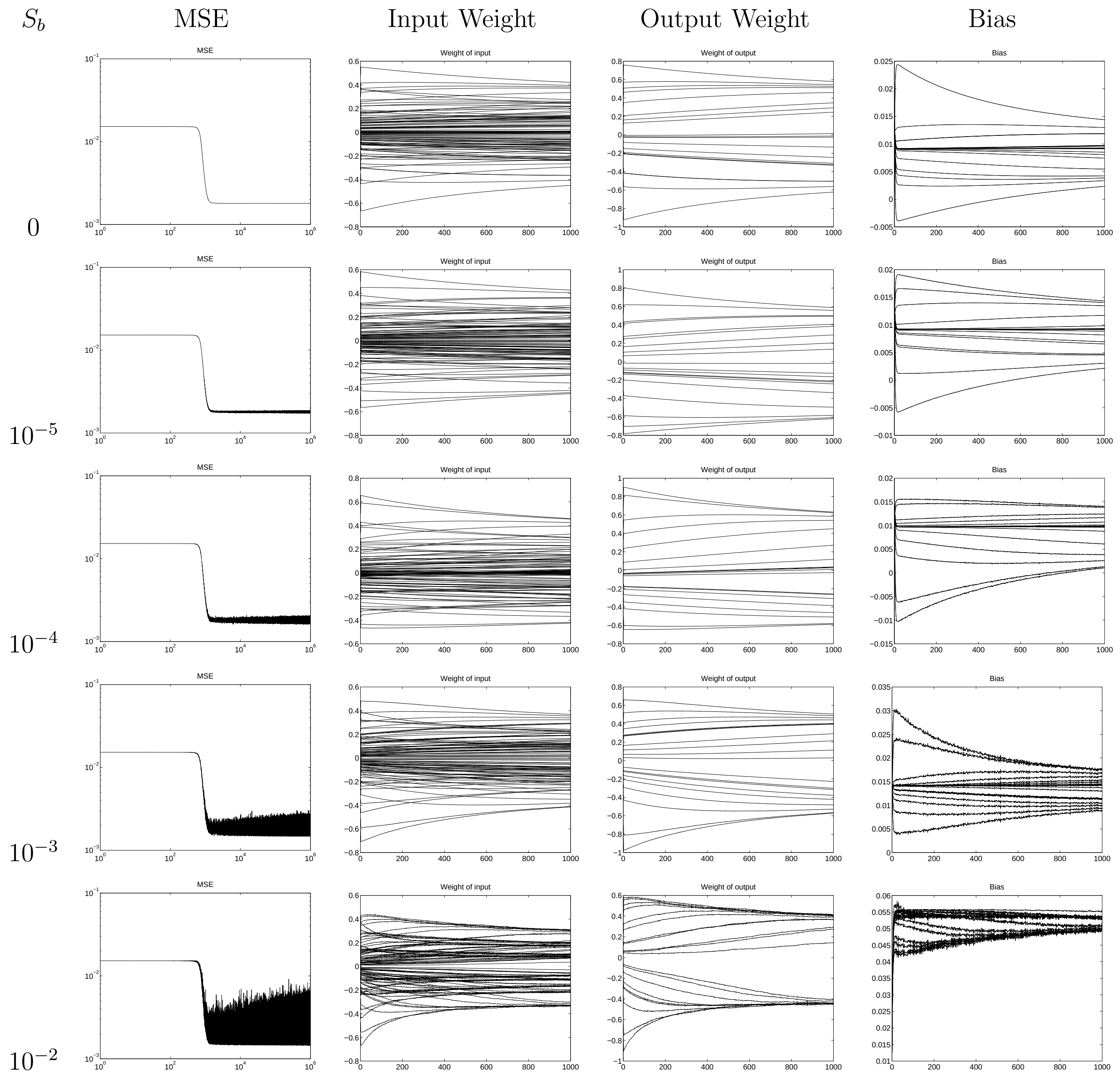

Fig. 2.27: $a = 10^{-4}$, MWN - Astrophysical data.

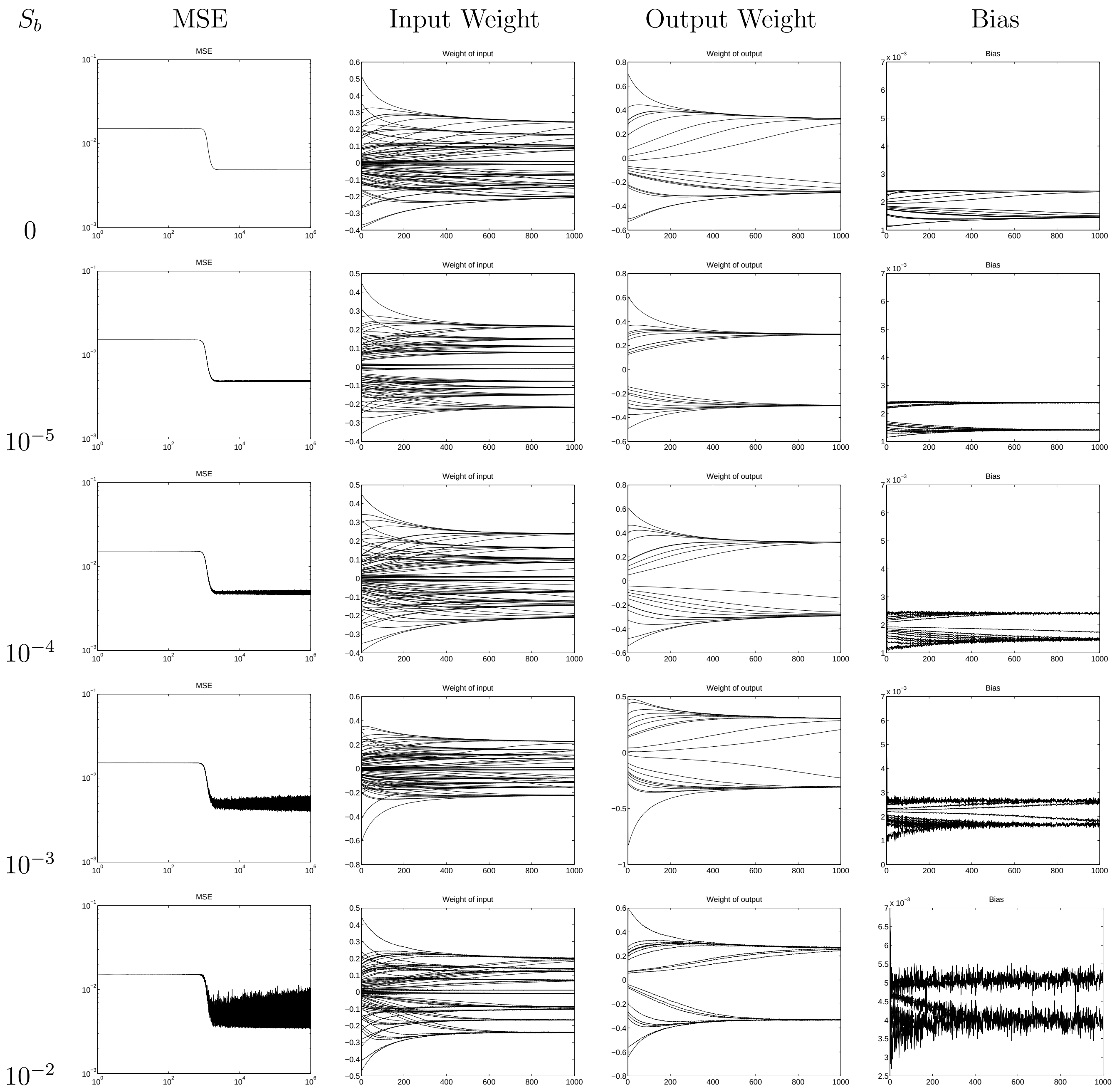

Fig. 2.28: $a = 10^{-3}$, MWN - Astrophysical data.

B.4.2 AWN

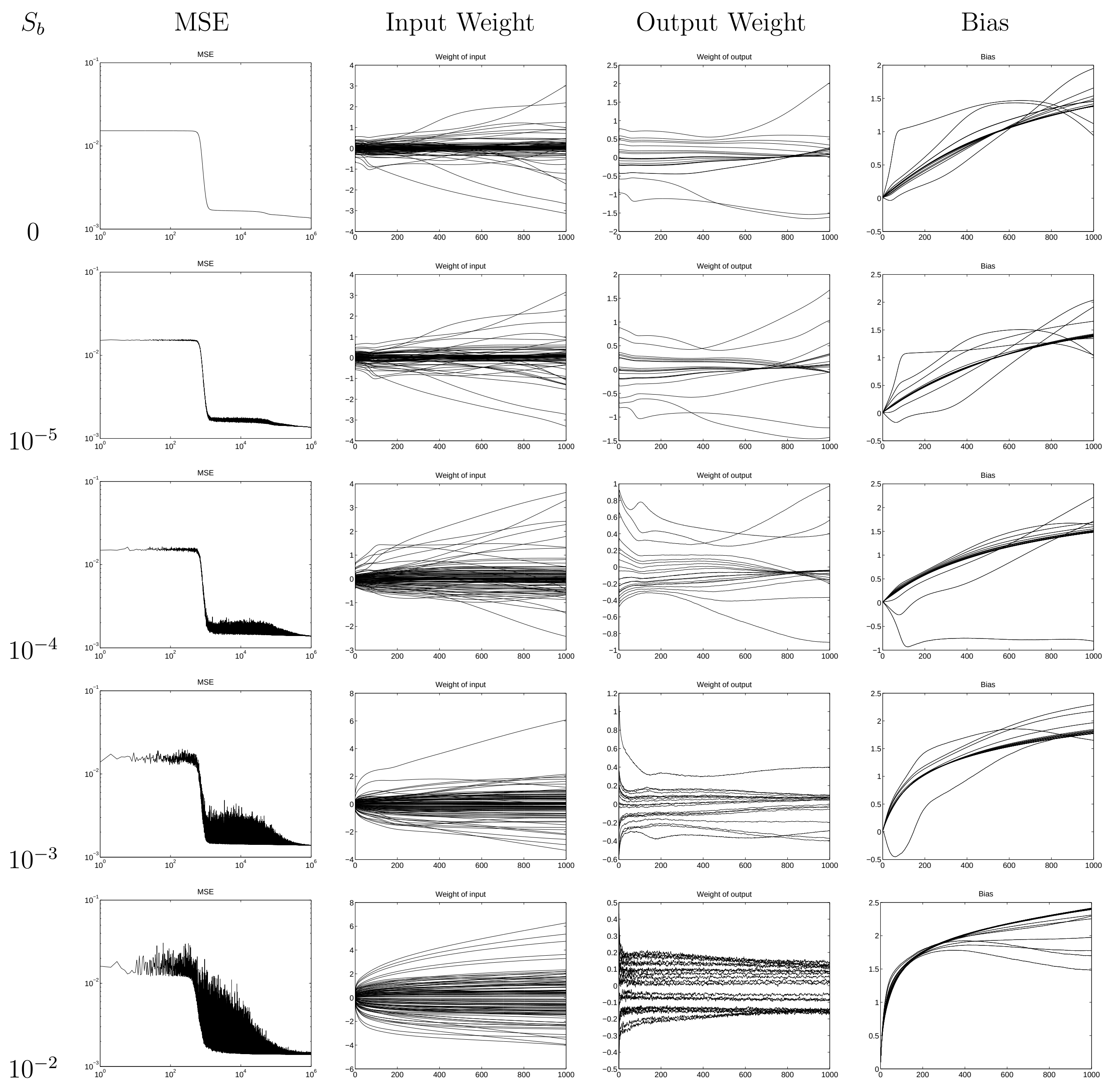

Fig. 2.29: $a = 0$, AWN - Astrophysical data.

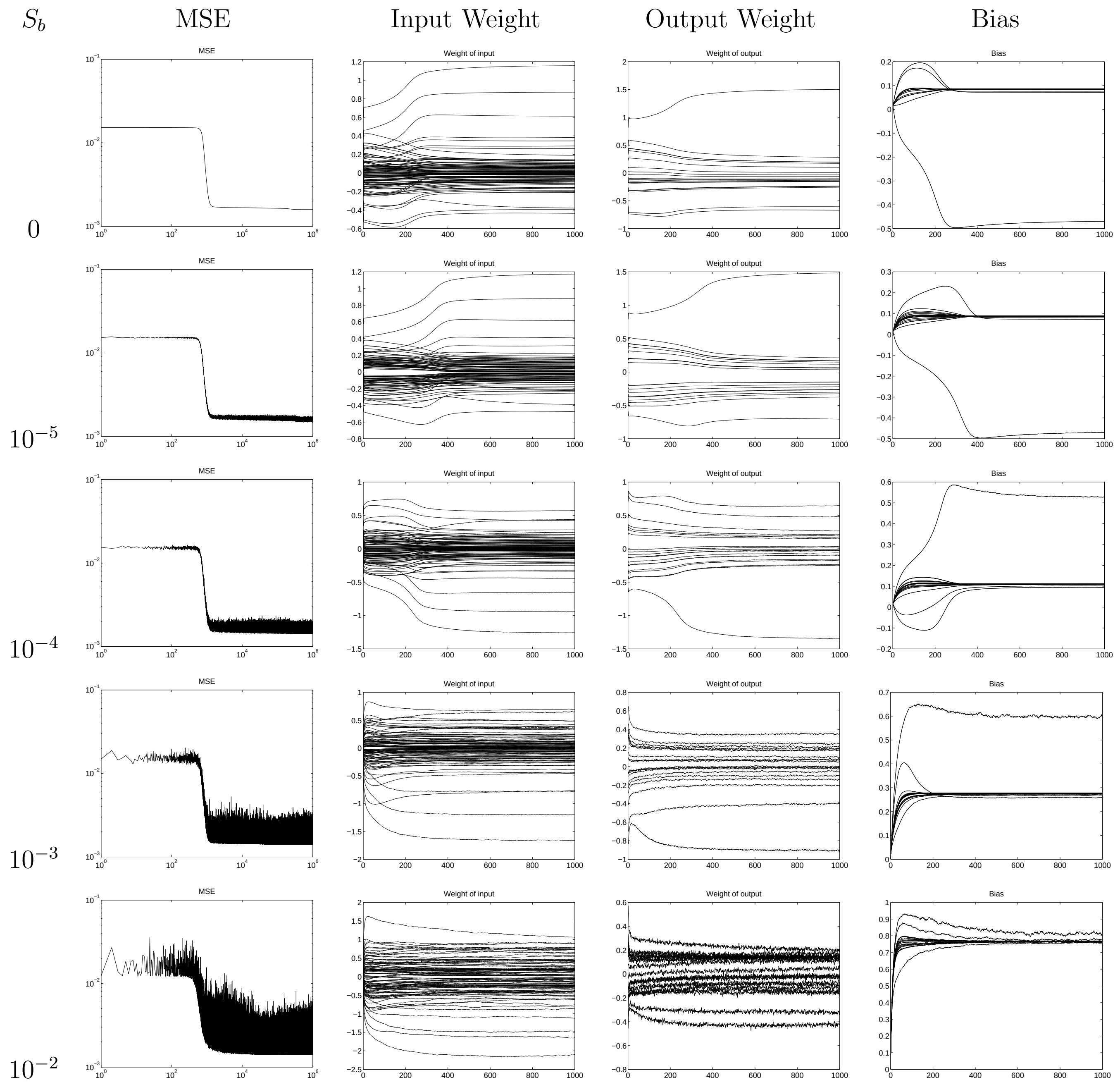

Fig. 2.30: $a = 10^{-5}$, AWN - Astrophysical data.

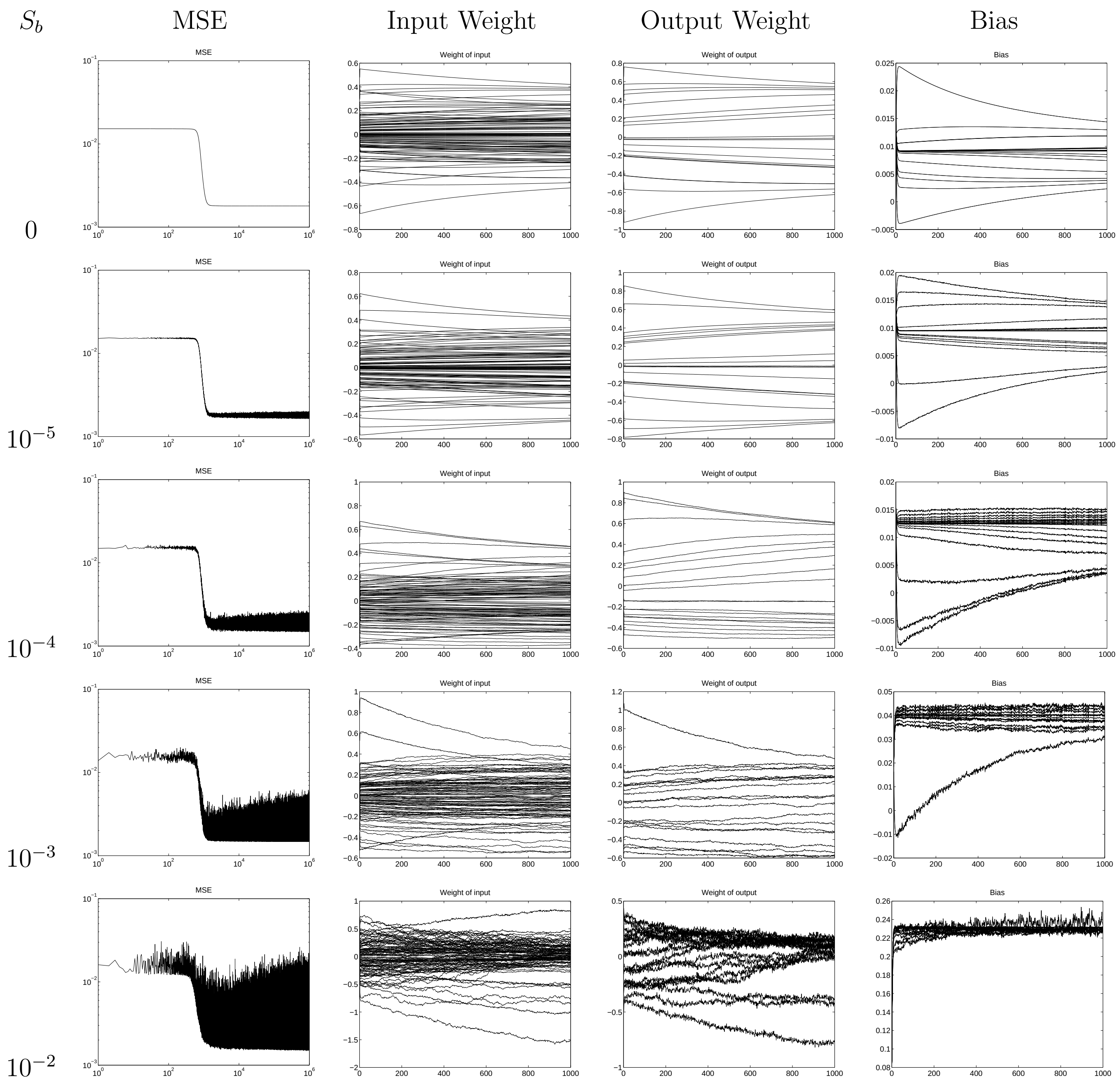

Fig. 2.31: $a = 10^{-4}$, AWN - Astrophysical data.

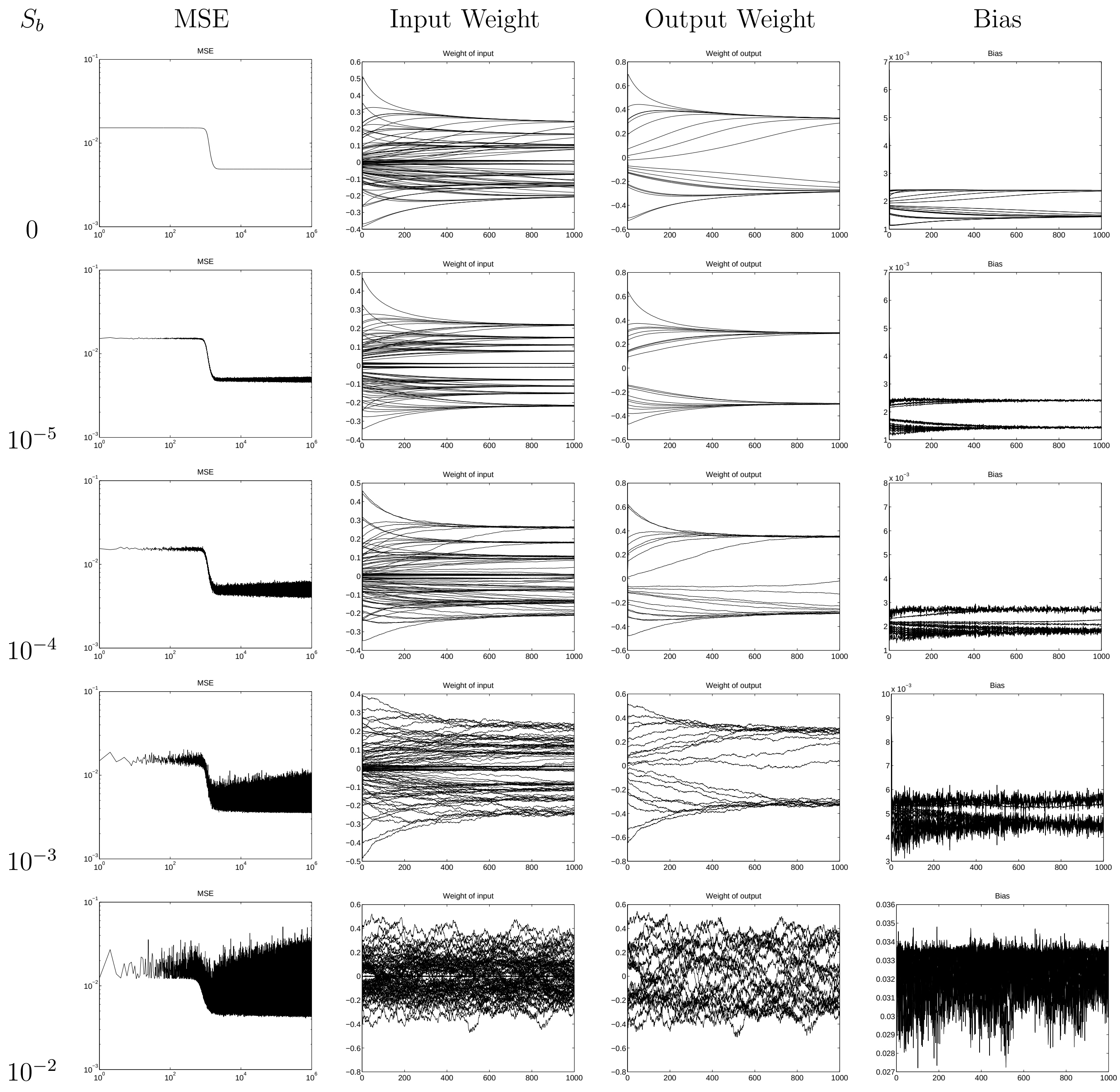

Fig. 2.32: $a = 10^{-3}$, AWN - Astrophysical data.

B.5 XOR

B.5.1 MWN

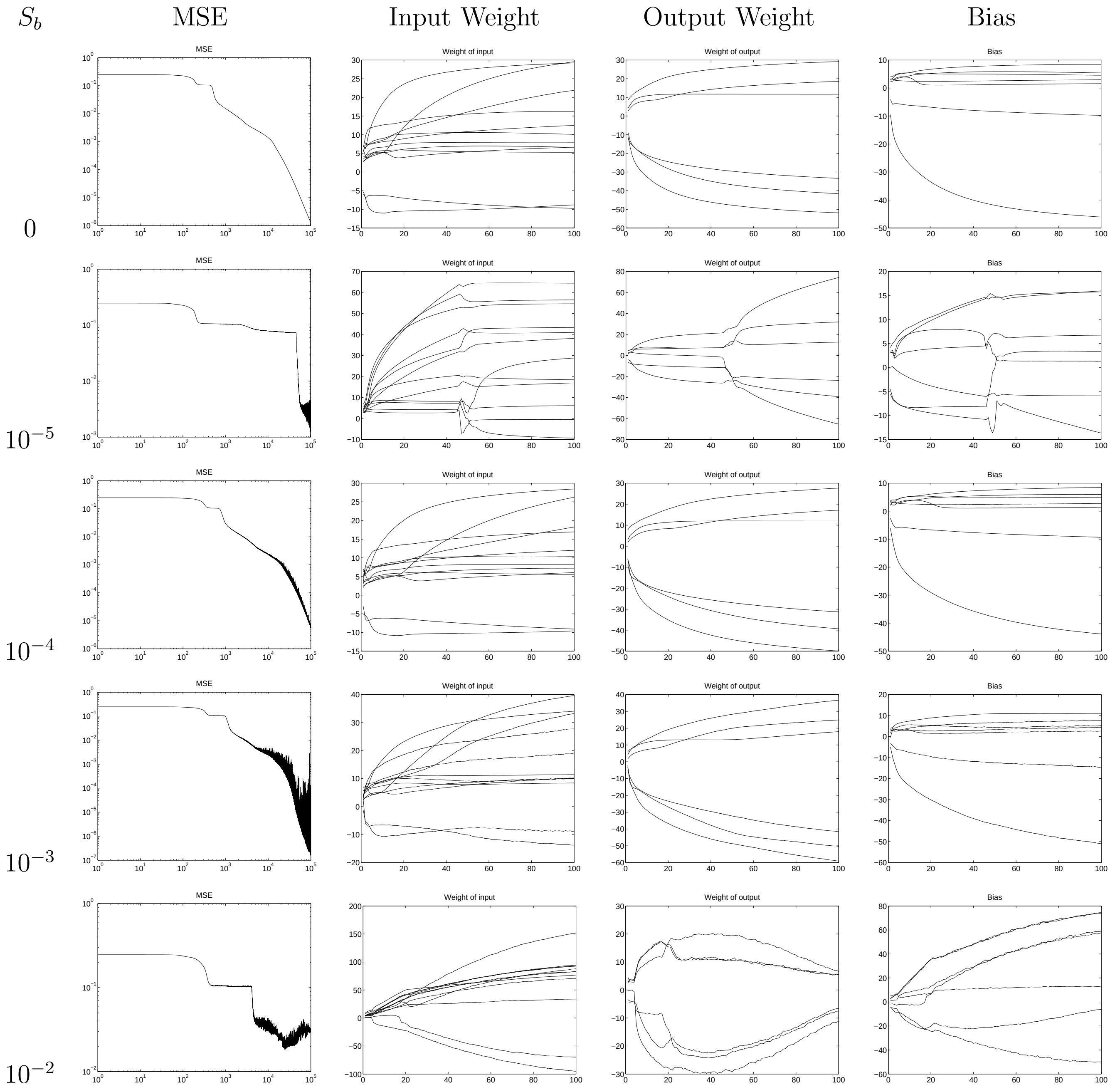

Fig. 2.33: $a = 0$, MWN - XOR.

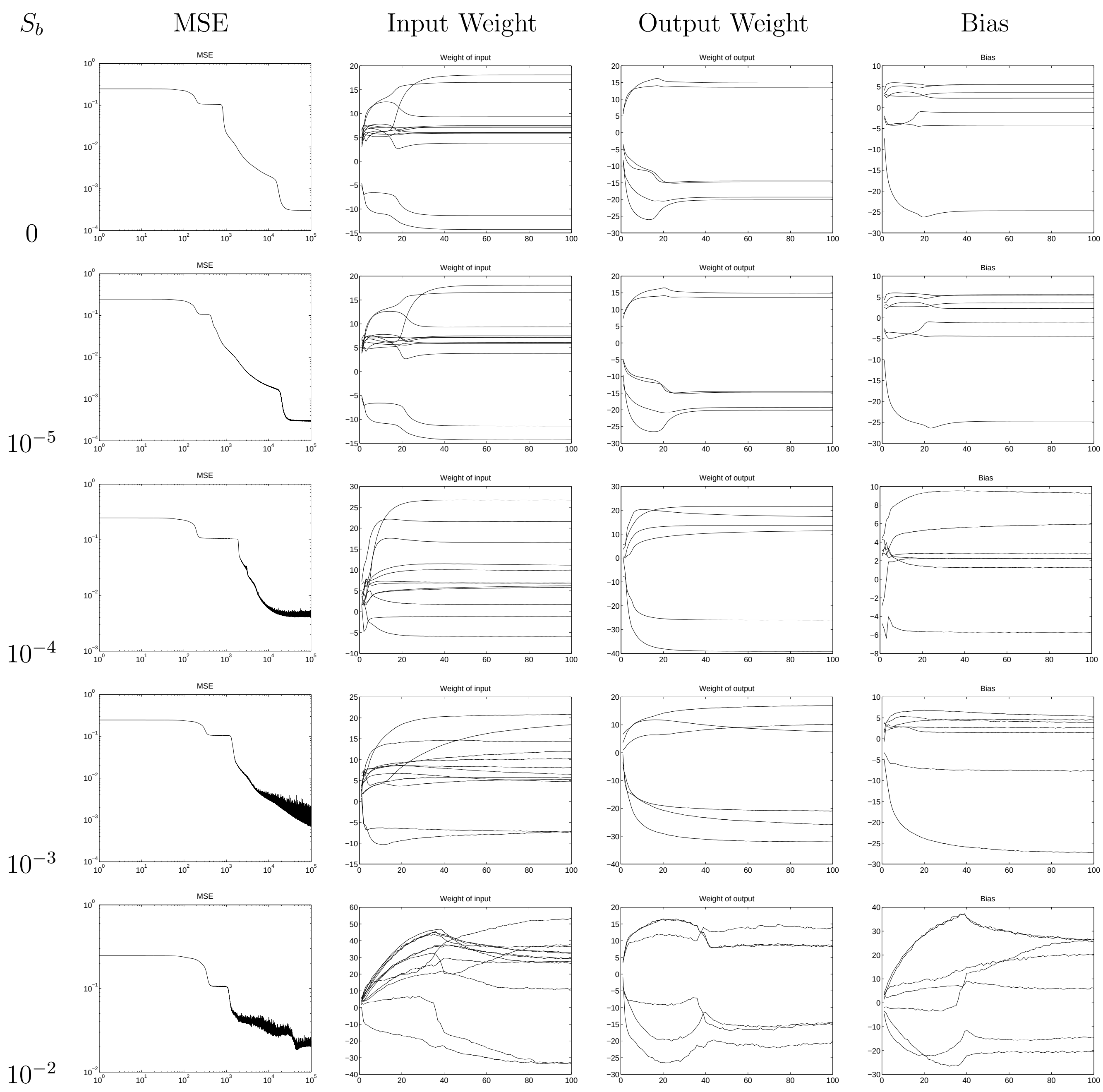

Fig. 2.34: $a = 10^{-5}$, MWN - XOR.

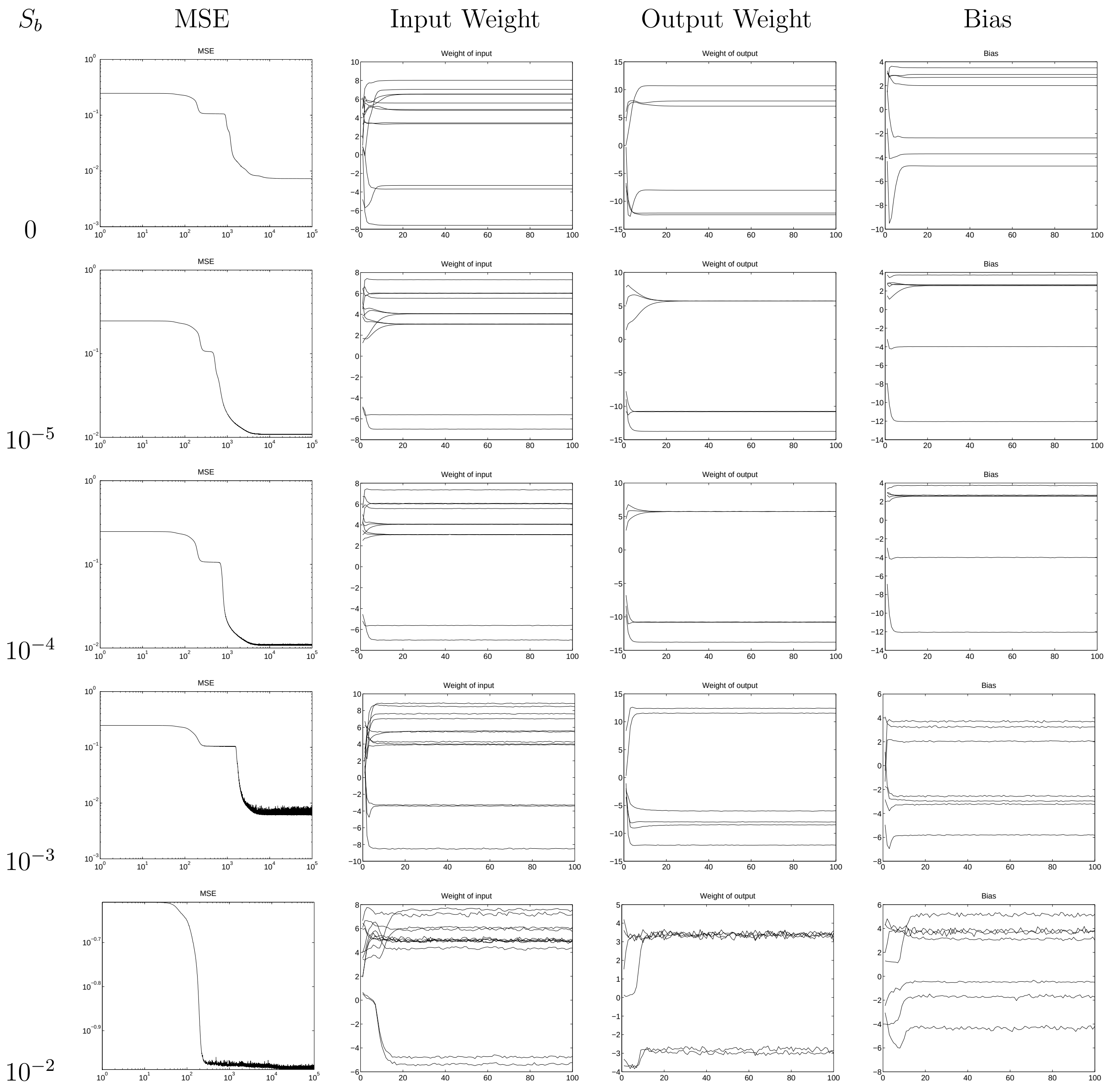

Fig. 2.35: $a = 10^{-4}$, MWN - XOR.

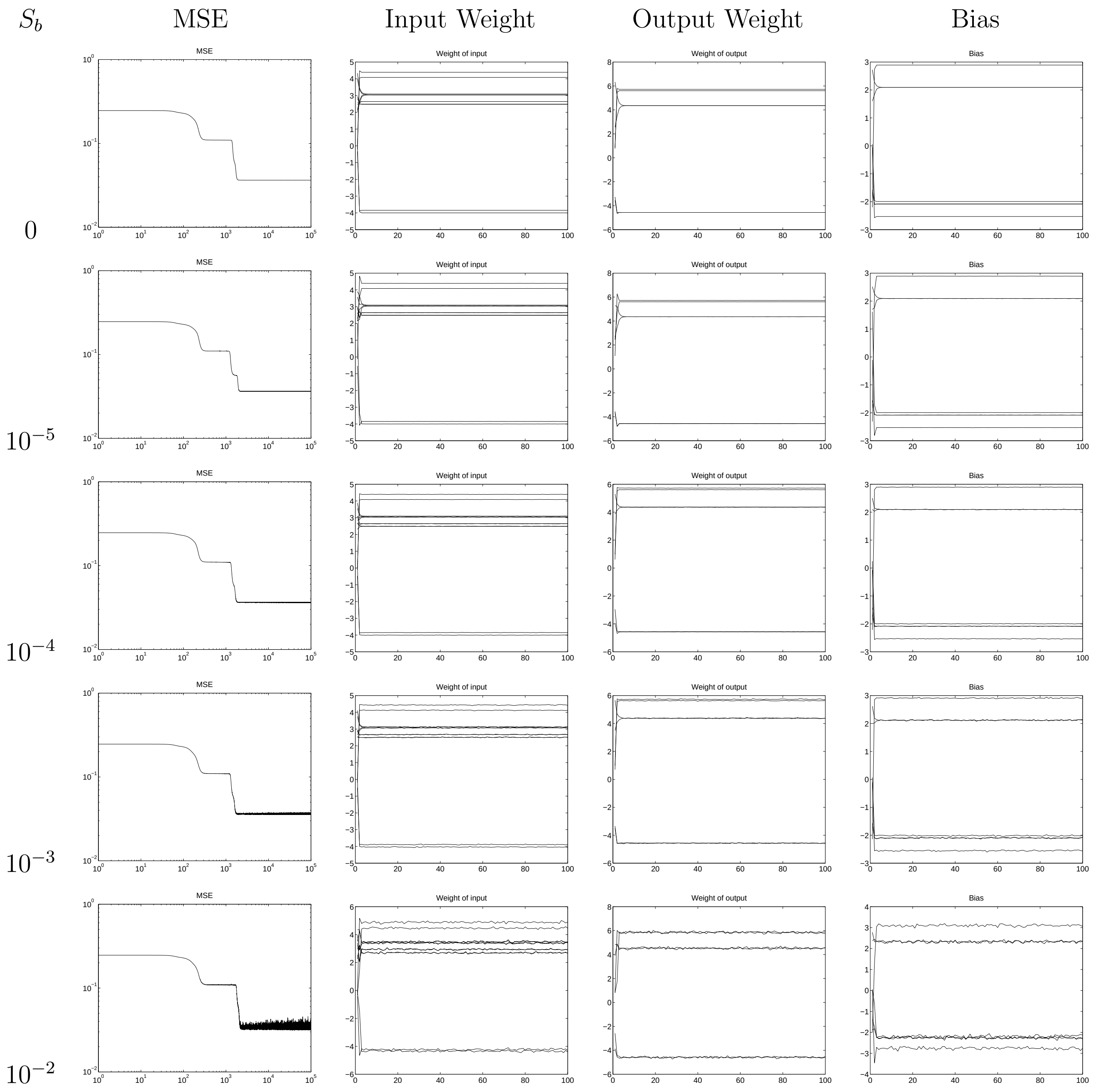

Fig. 2.36: $a = 10^{-3}$, MWN - XOR.

B.5.2 AWN

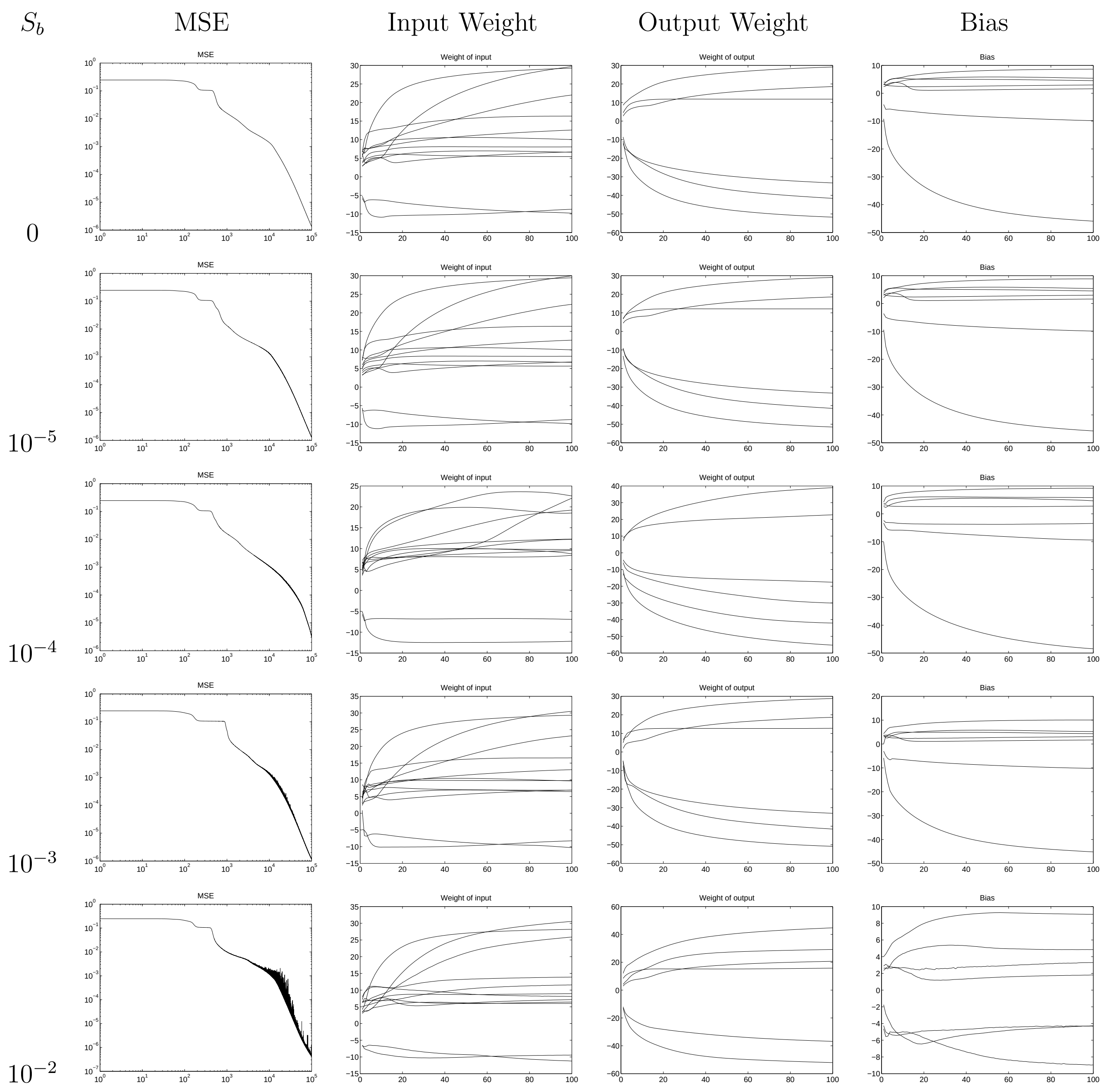

Fig. 2.37: $a = 0$, AWN - XOR.

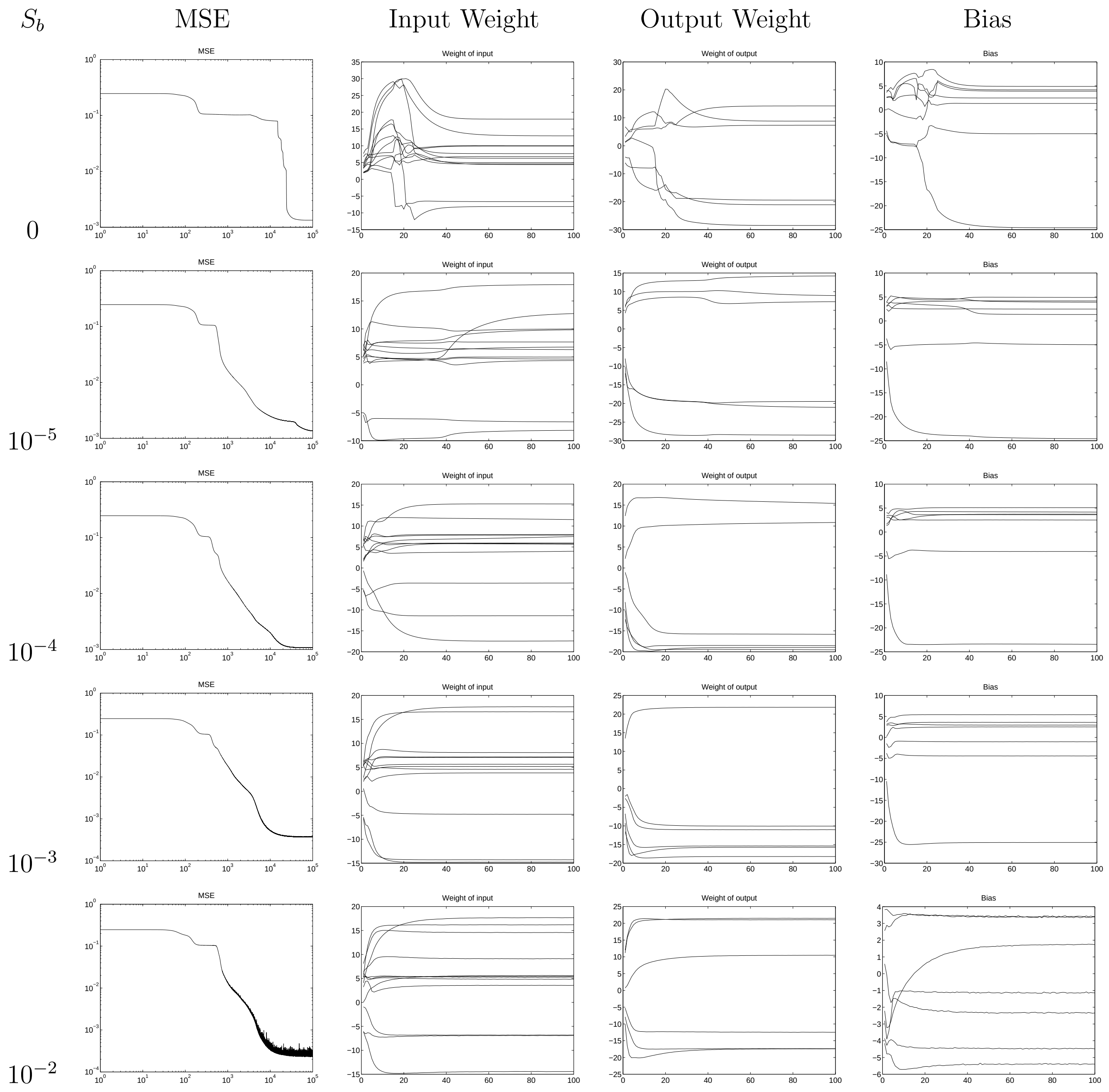

Fig. 2.38: $a = 10^{-5}$, AWN - XOR.

S_b MSE Input Weight Output Weight Bias

0

10^{-5}

10^{-4}

10^{-3}

10^{-2}

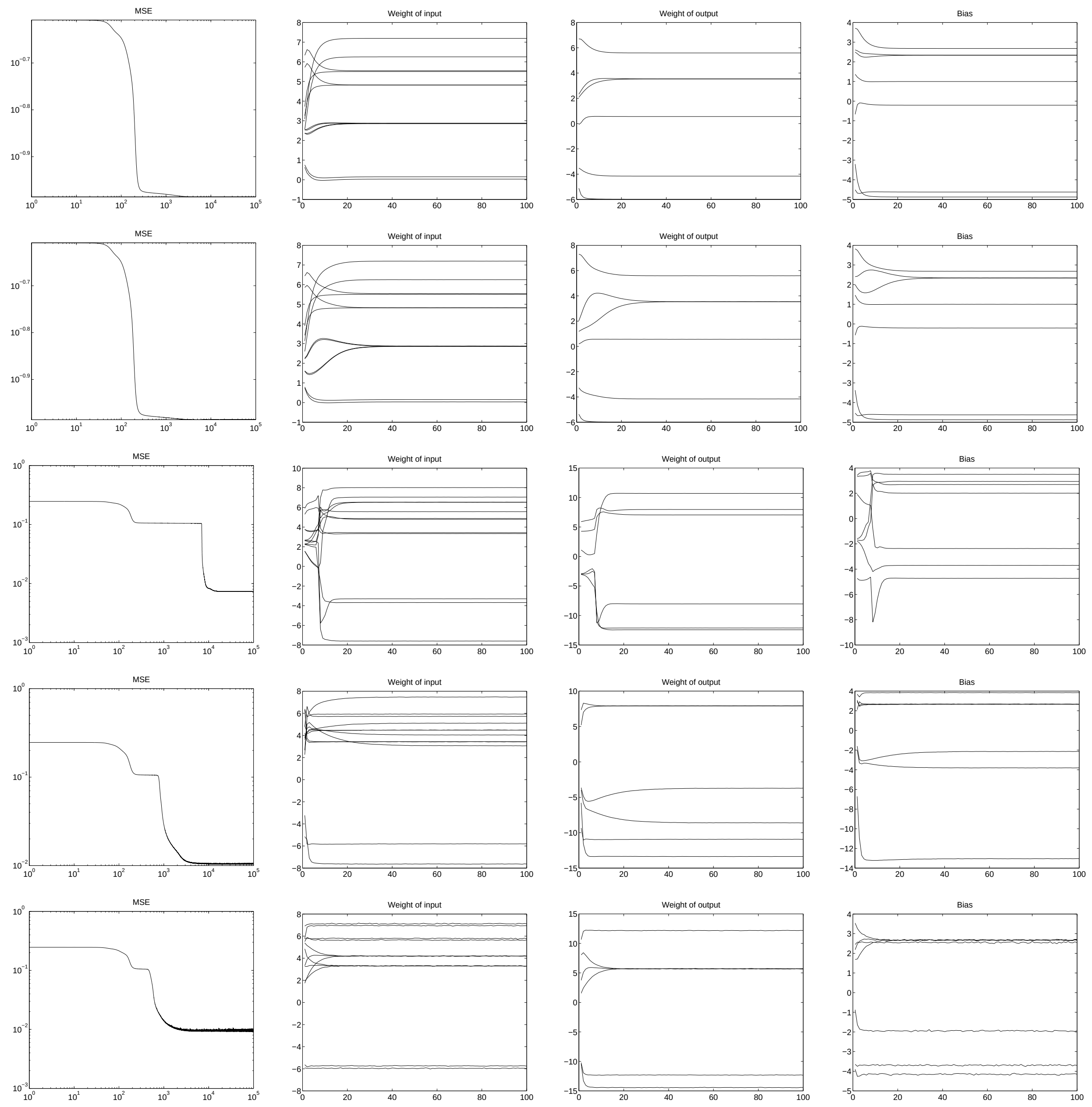

Fig. 2.39: $a = 10^{-4}$, AWN - XOR.

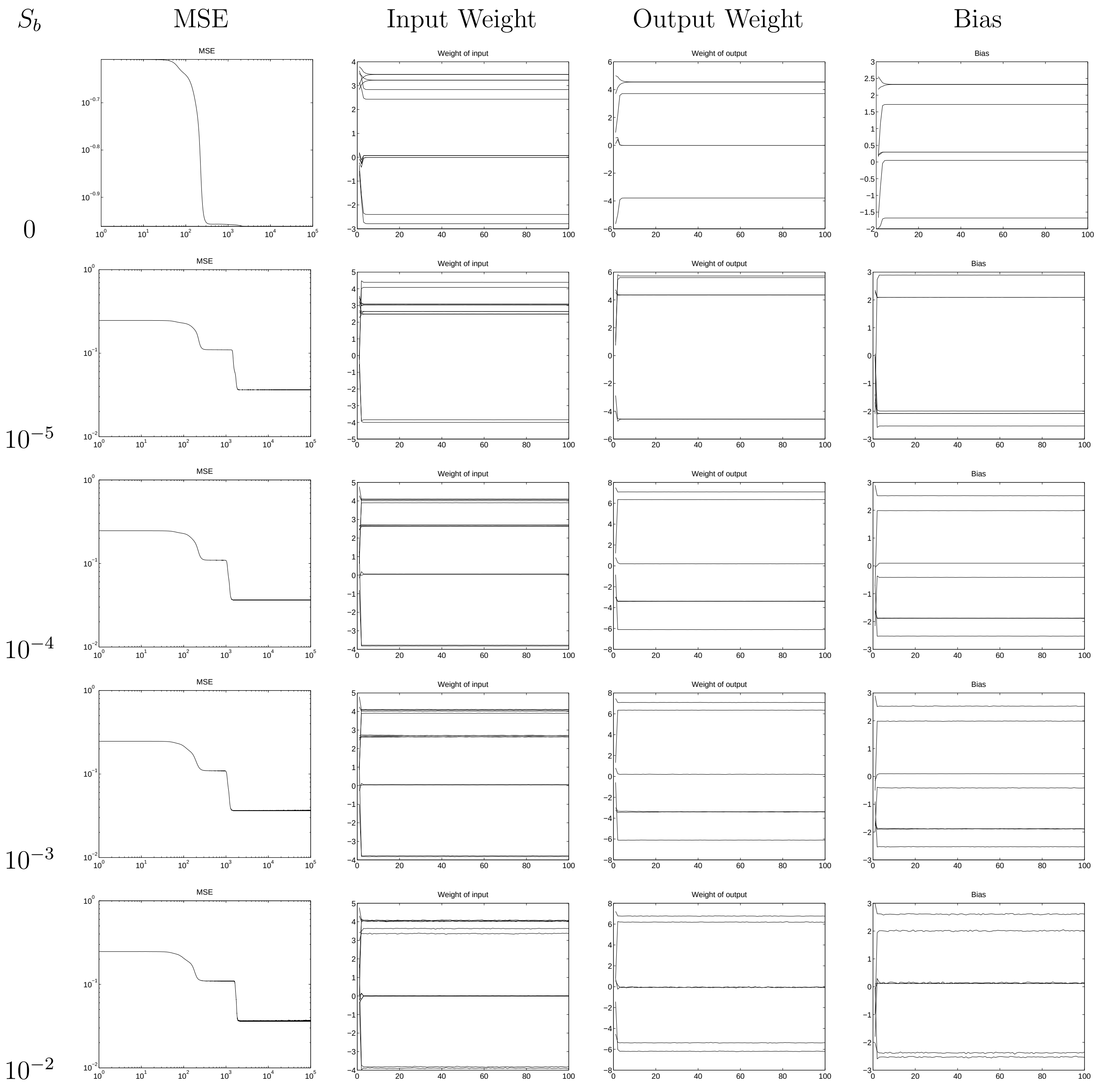

Fig. 2.40: $a = 10^{-3}$, AWN - XOR.

B.6 Semeion handwritten digital recognition

B.6.1 MWN

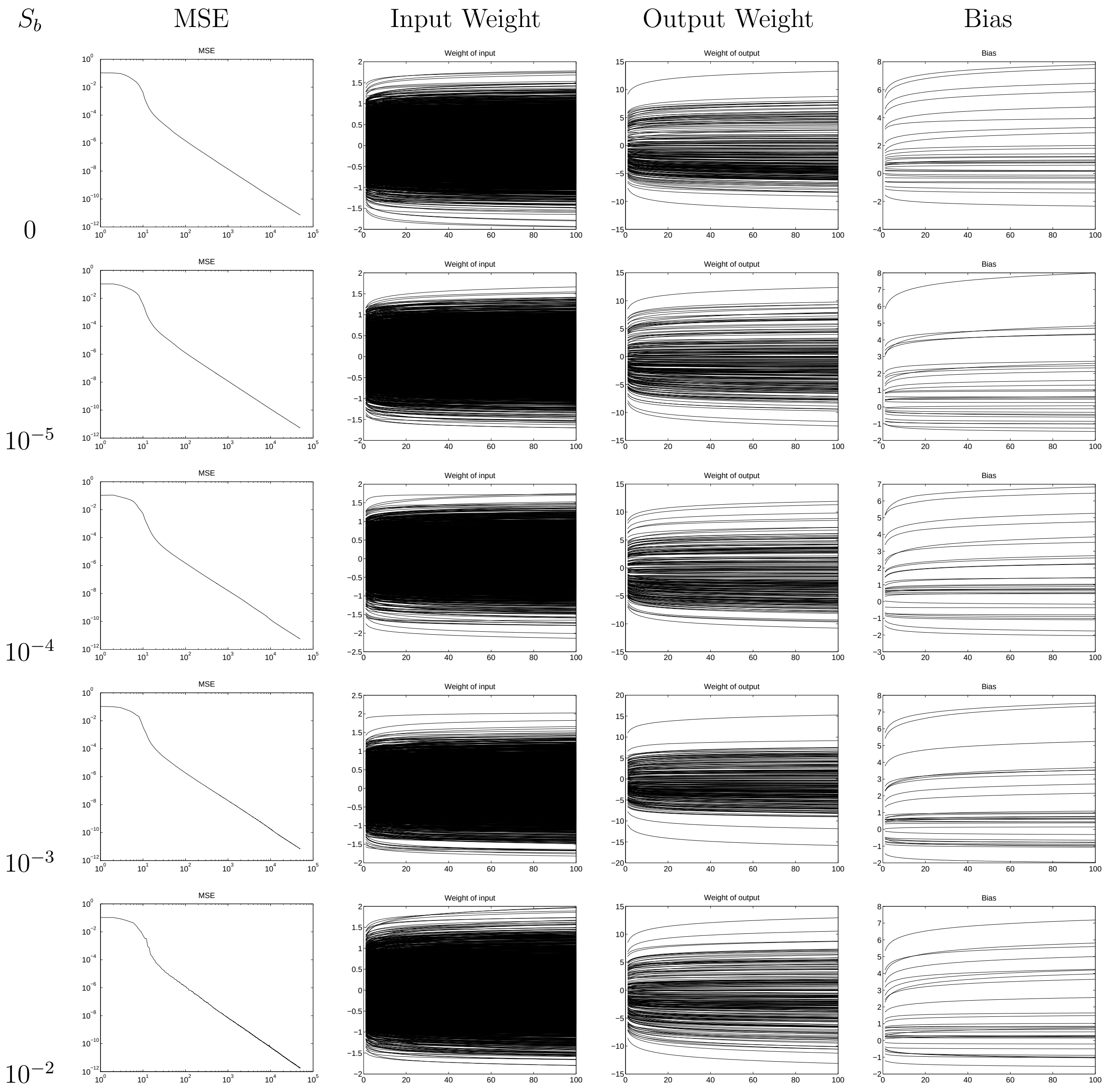

Fig. 2.41: $a = 0$, MWN - Semeion.

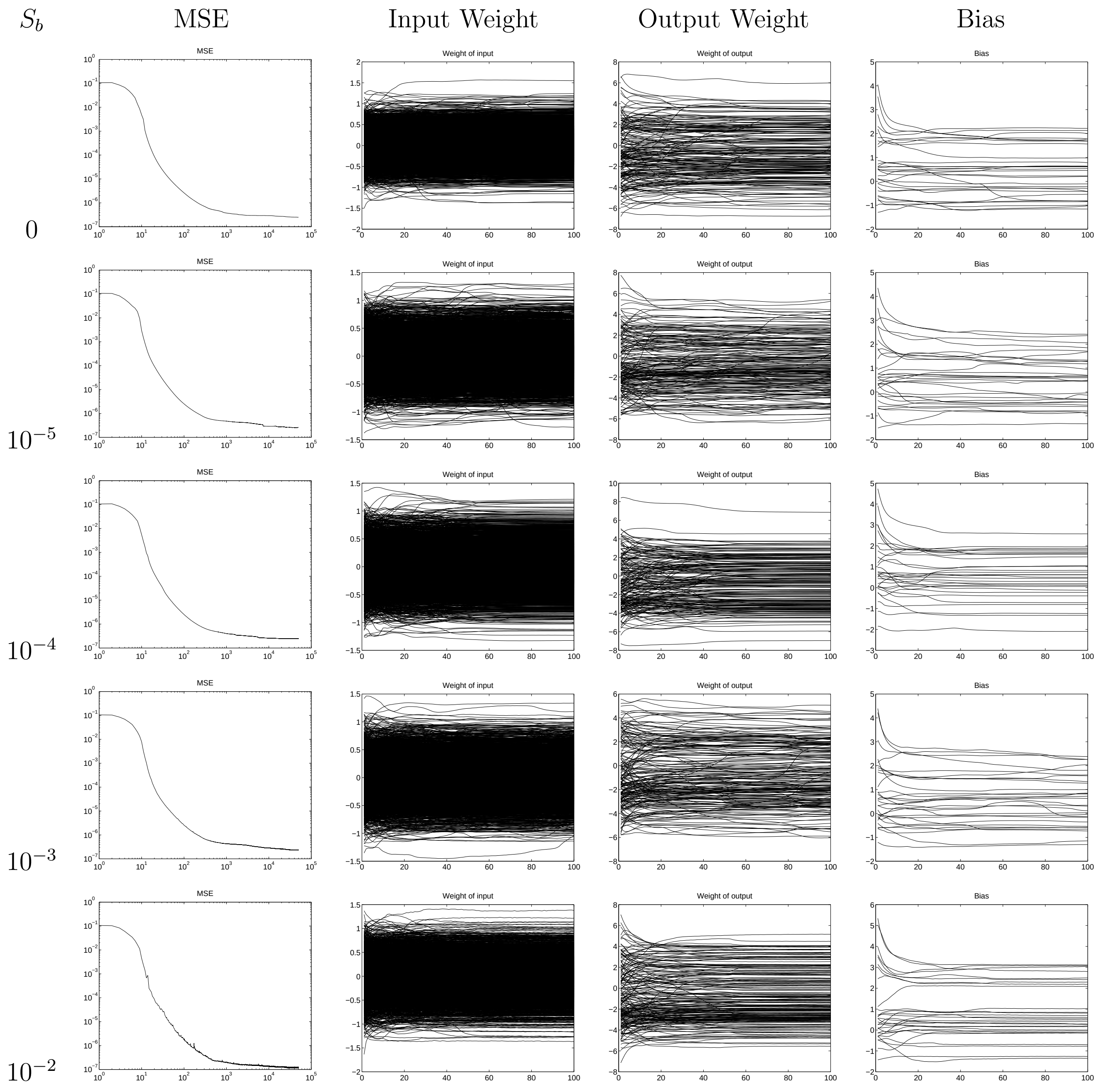

Fig. 2.42: $a = 10^{-5}$, MWN - Semeion.

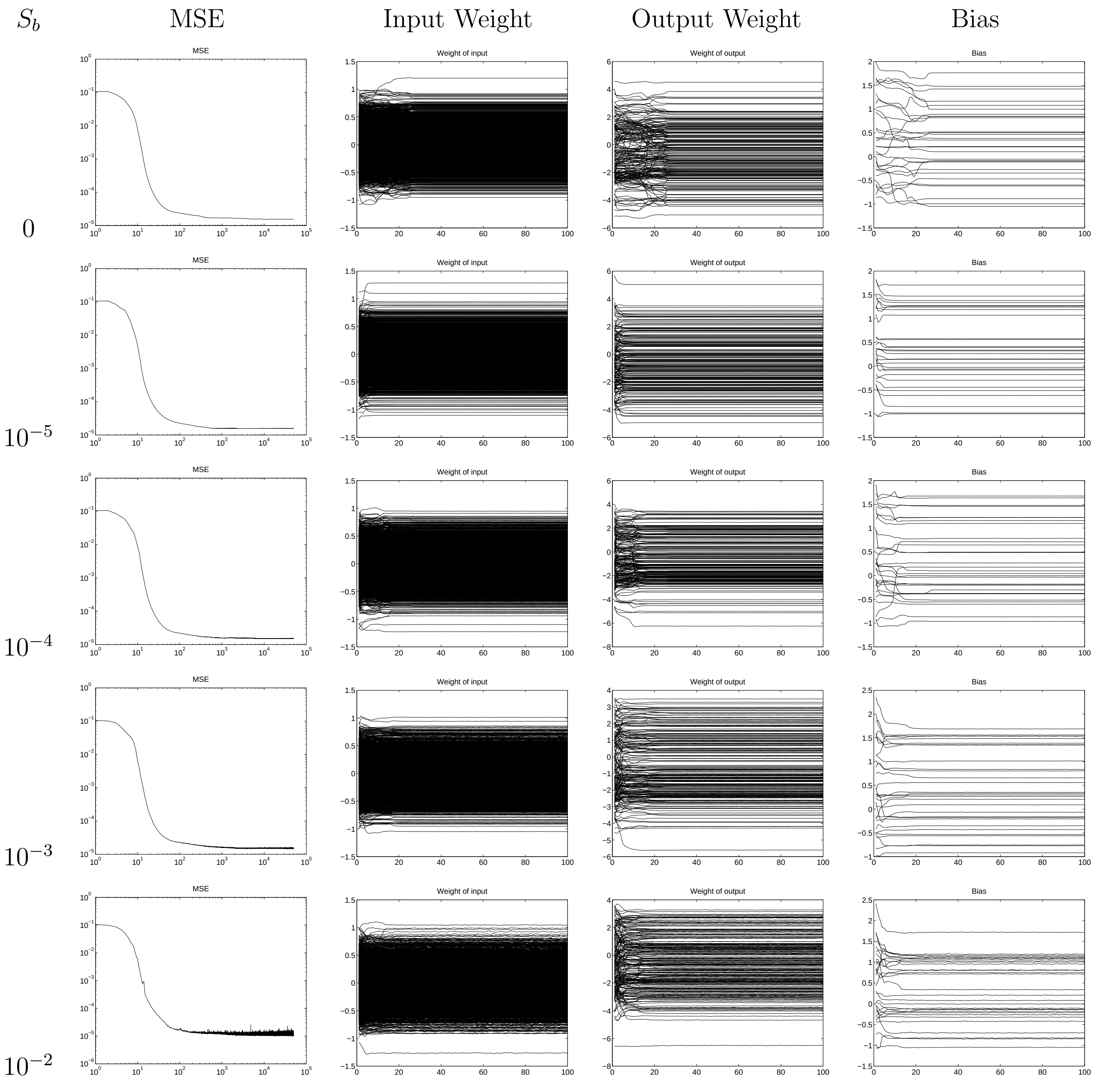

Fig. 2.43: $a = 10^{-4}$, MWN - Semeion.

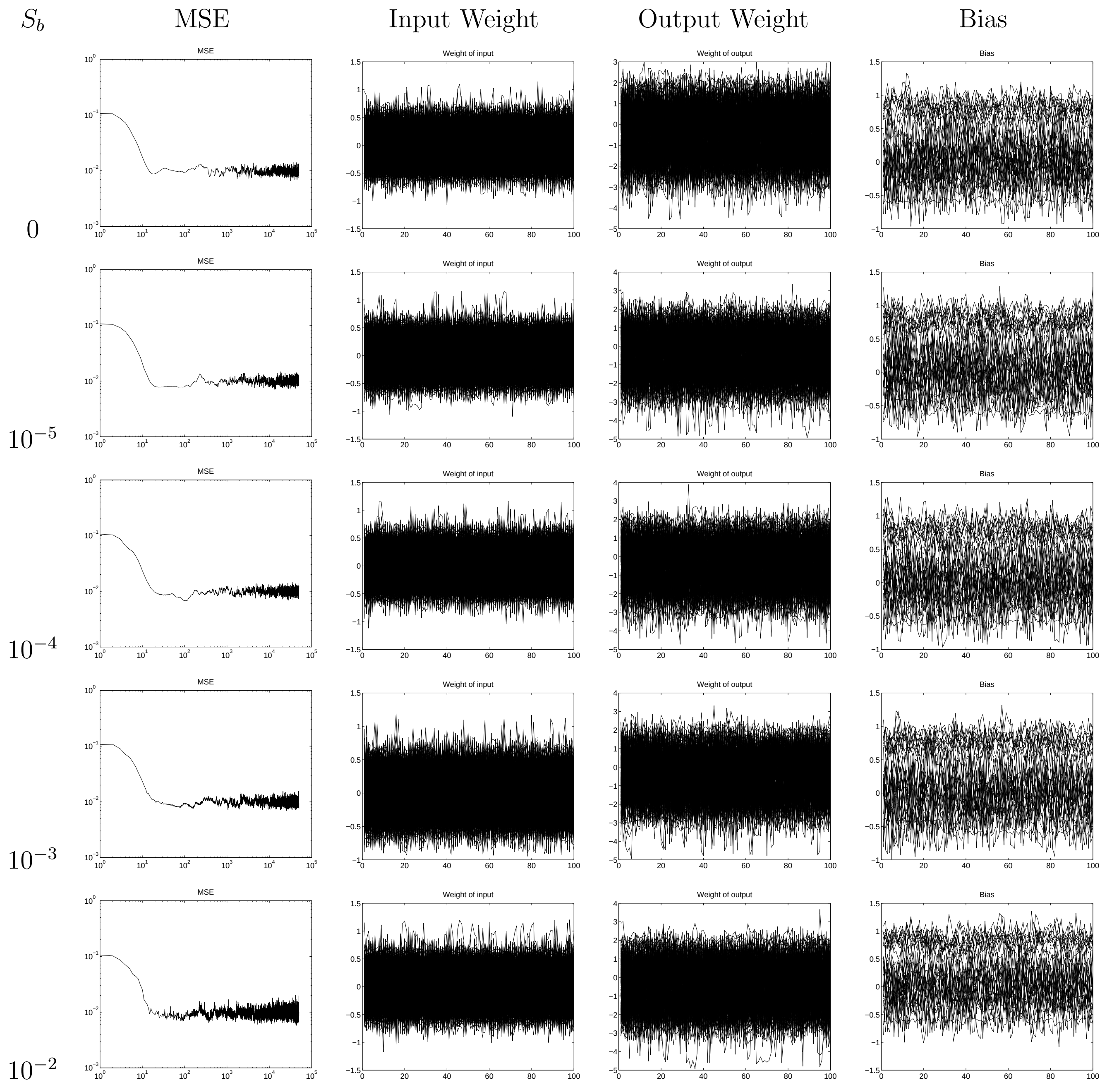

Fig. 2.44: $a = 10^{-3}$, MWN - Semeion.

B.6.2 AWN

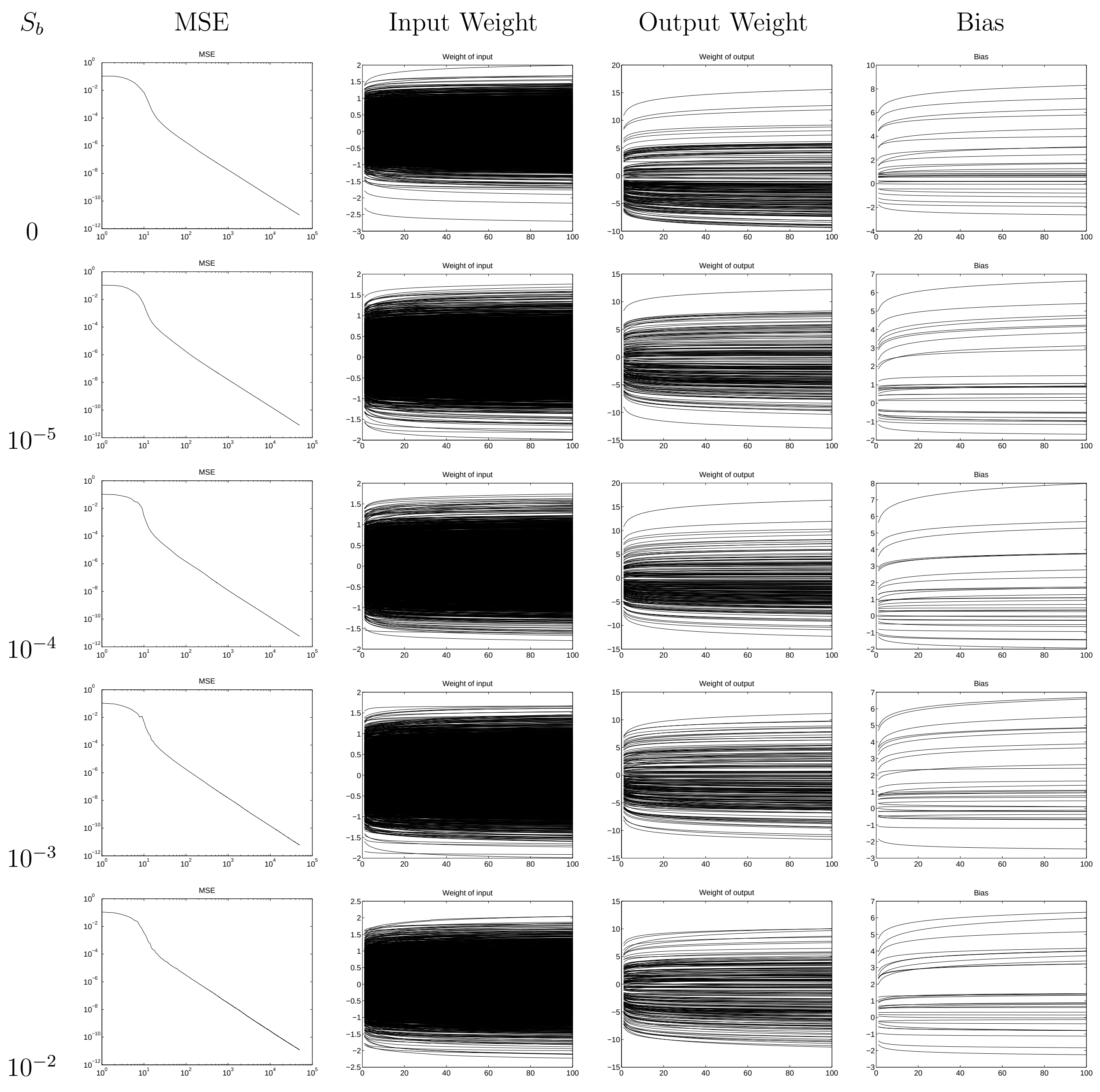

Fig. 2.45: $a = 0$, AWN - Semeion.

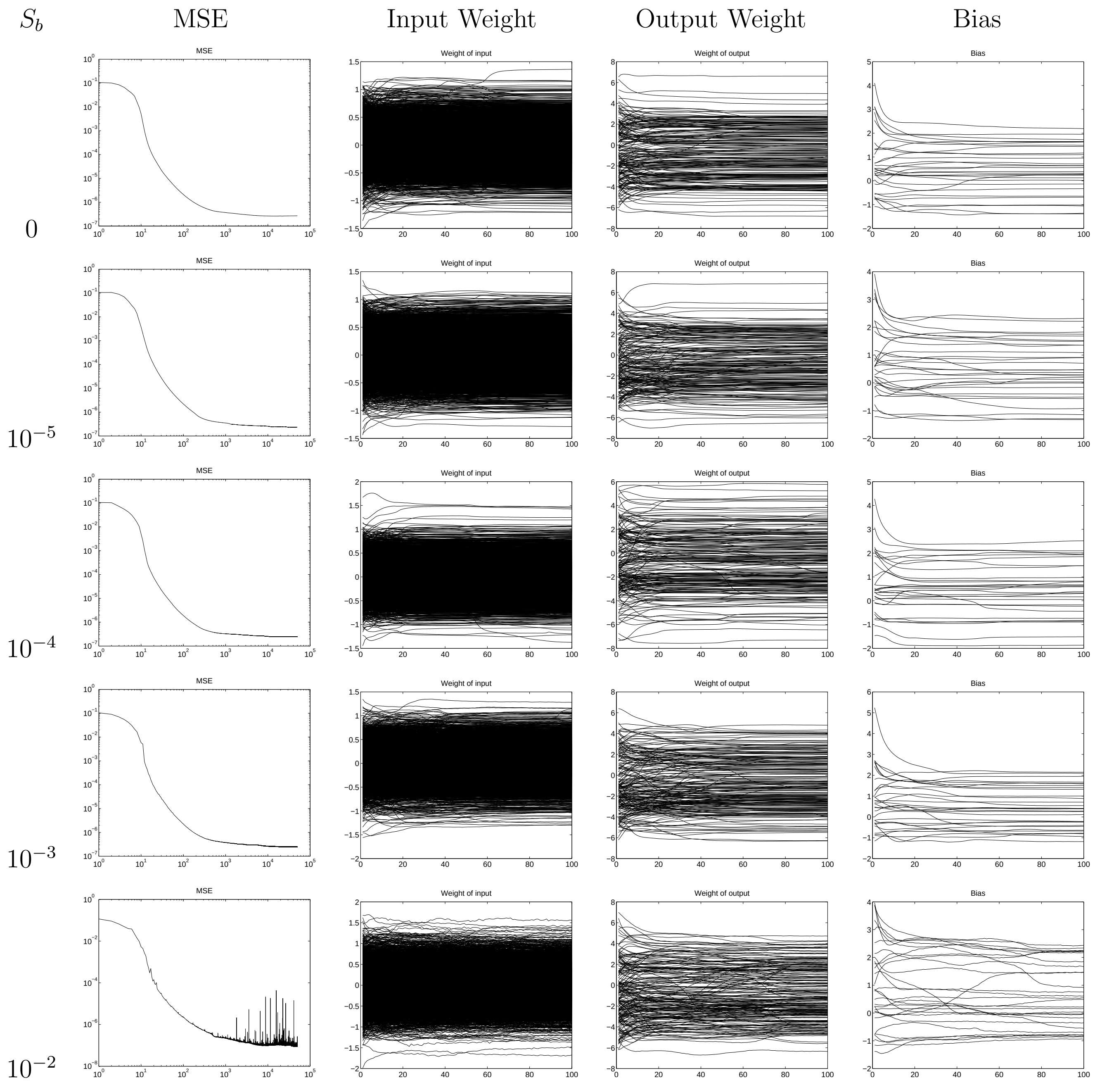

Fig. 2.46: $a = 10^{-5}$, AWN - Semeion.

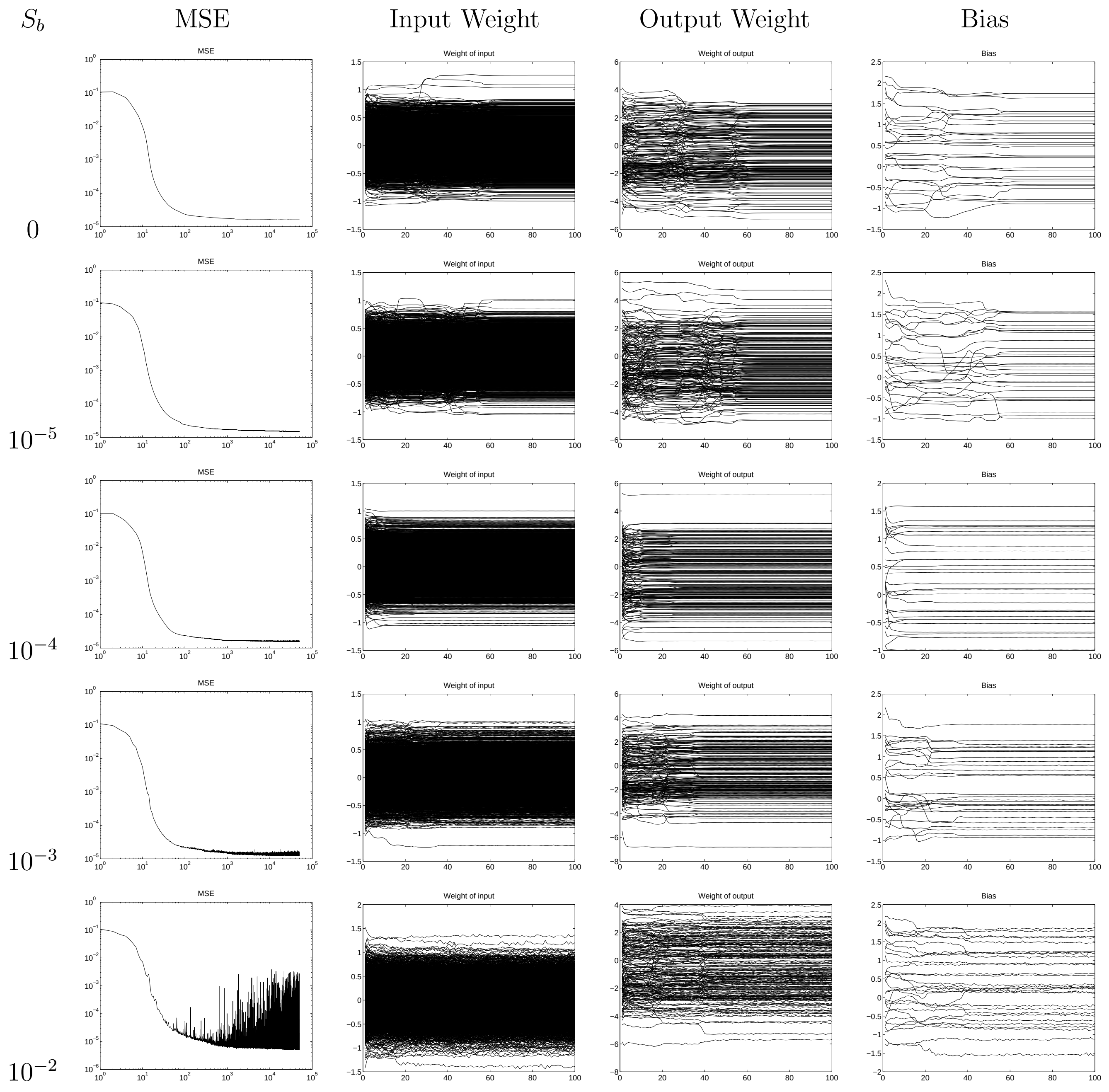

Fig. 2.47: $a = 10^{-4}$, AWN - Semeion.

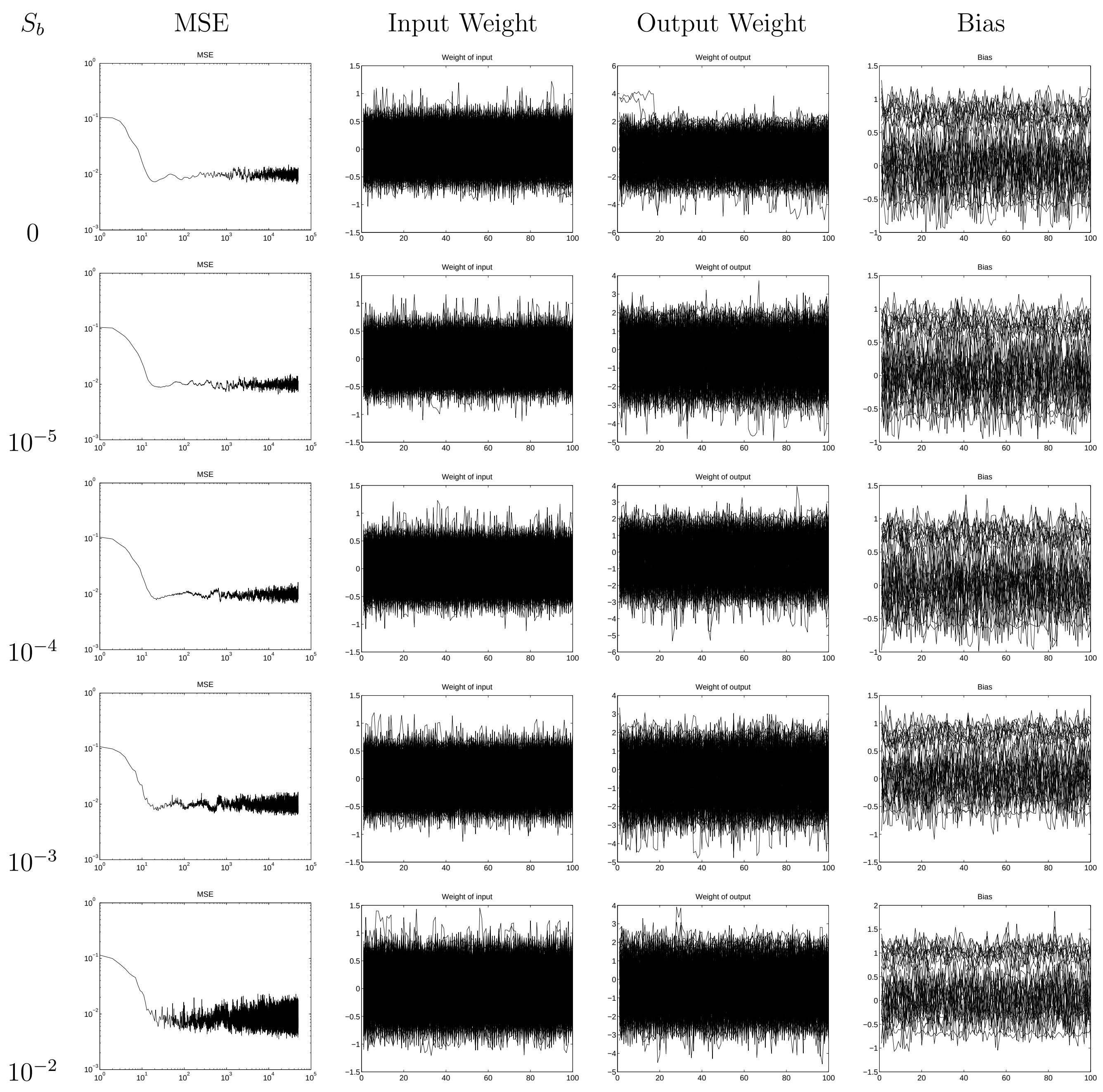

Fig. 2.48: $a = 10^{-3}$, AWN - Semeion.

Appendix C

MATLAB Program for Simulations

```
% user specified values
hidden_neurons = 20;
epochs = 500;
repeat = 100;
lr=0.1;
% a=0;               %weight decay
% SD=0;              %weight noise training
SD1=0:0.002:0.04;   %weight noise testing
de=sqrt(SD);
de1=sqrt(SD1);
ERR=[];

load data;

N = length(xtrain(:,1));
M = length(xtest(:,1));
u = length(xtrain(1,:));
v = length(ytrain(1,:));

% ---------- set weights -----------------
% set initial random weights
weight_input_hidden = ((rand(u,hidden_neurons))-0.5)/500;
weight_hidden_output = ((rand(v,hidden_neurons))-0.5)/500;

% add bias
bias1 = (rand(hidden_neurons,1)-0.5)/500;
bias2 = (rand(1,v)-0.5)/500;

%------------------------------------
%--- Learning Starts Here! ---------
```

```
%------------------------------------
for n = 1:repeat
        n

     tic;

    % do a number of epochs
    for iter = 1:epochs


        % loop through the patterns
        for j = 1:N


            % set the current pattern
            X = xtrain(j,:);
            Y = ytrain(j,:);

            % calculate the current error for this pattern
            weight_hidden_output_noise = ...
            weight_hidden_output.*(1+de*randn(size(weight_hidden_output)));
            weight_input_hidden_noise = ...
            weight_input_hidden.*(1+de*randn(size(weight_input_hidden)));
            bias1_noise = bias1.*(1+de*(randn(size(bias1))));
            bias2_noise = bias2.*(1+de*(randn(size(bias2))));

            net_1 = X*weight_input_hidden_noise-bias1_noise';
            h_1 = 1./(1+exp(-net_1));
            net_2 = weight_hidden_output_noise*h_1'-bias2_noise';
            h_2 = 1./(1+exp(-net_2))';

            % adjust the weights & bias
            delta_h_2 = (Y-h_2);
            delta_h_1 = h_1.*(1-h_1).*(delta_h_2*weight_hidden_output);

            delta_w_2 = lr*delta_h_2'*h_1;
            delta_w_1 = lr*X'*delta_h_1;

            weight_hidden_output = (1-lr*a)*weight_hidden_output + delta_w_2;
            weight_input_hidden = (1-lr*a)*weight_input_hidden + delta_w_1;
            bias1 = (1-lr*a)*bias1 - lr*delta_h_1';
            bias2 = (1-lr*a)*bias2 - lr*delta_h_2;

        end

        % -- another epoch finished

        % plot overall network error at end of each epoch

        for k=1:N
            net_1 = xtrain(k,:)*weight_input_hidden-bias1';
            h_1 = 1./(1+exp(-net_1));
            net_2 = weight_hidden_output*h_1'-bias2';
            h_2_1(k,:) = 1./(1+exp(-net_2))';
        end

        error = h_2_1 - ytrain;
        err(iter) = mse(error);


    end

    ERR = [ERR err];
    WIH(:,:,n) = weight_input_hidden;
    WHO(:,:,n) = weight_hidden_output;
    BS1(:,n) = bias1;
    BS2(n,:) = bias2;
     toc;

end

WIH = reshape(WIH,u*hidden_neurons,repeat);
WHO = reshape(WHO,v*hidden_neurons,repeat);
BS1 = reshape(BS1,hidden_neurons,repeat);


figure(1);
subplot (2,3,1)
plot(ERR)
title ('MSE')
subplot(2,3,2)
plot(WIH')
title ('weight input')
subplot(2,3,3)
plot(WHO')
title ('weight output')
subplot(2,3,4)
plot(BS1')
title ('bias1')
subplot(2,3,5)
plot(BS2)
title ('bias2')



m = length(SD1);
mse_test=[];
count_test=[];
for k=1:100
    for i=1:m
        for k=1:M
            net_1 =...
            xtest(k,:)*(weight_input_hidden...
            .*(1+de1(i)*randn(size(weight_input_hidden))))...
            -(bias1.*(1+de1(i)*(randn(size(bias1)))))';
            h_1 = 1./(1+exp(-net_1));
            net_2 =...
            (weight_hidden_output.*...
            (1+de1(i)*randn(size(weight_hidden_output))))*h_1'...
            -(bias2.*(1+de1(i)*(randn(size(bias2)))))';
            h_2_2(k,:) = 1./(1+exp(-net_2))';
        end
        test_mse(i) = mse(ytest-h_2_2);

        xx = sum(abs(round(h_2_2)-ytest)')';
        xx(find(xx>=1))=1;
        count(i)=sum(xx);
    end
    mse_test = [mse_test;test_mse];
    count_test = [count_test;count];
end
avg_test_mse = mean(mse_test);
avg_count_test = mean(count_test)/M;

figure(3);
boxplot (mse_test,0:0.002:0.04);
title (['Weight Decay = ',num2str(a),'; Sb = ',num2str(SD)],'fontsize',16)
axis ([1 21.5 0 0.1]);
set(gca,'XTick',1:5:21)
set(gca,'XTickLabel',{'0','0.01','0.02','0.03','0.04'})
filename=(['a=', num2str(a), ' Sb=', num2str(SD),'.mat'])
save(filename)
print (figure(3),'-deps',filename)
```

Bibliography

[1] G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation, Vol.8, 643-674*, 1996.

[2] G. Basalyga and E. Salinas. When response variability increases neural network robustness to synaptic noise. *Neural Computation, Vol.18, 1349-1379*, 2006.

[3] J. L. Bernier, J. Ortega, E. Ros I. Rojas, and A. Prieto. Obtaining fault tolerance multilayer perceptrons using an explicit regularization. *Neural Processing Letters, Vol.12, 107-113*, 2000.

[4] J. L. Bernier, J. Ortega, I. Rojas, and A. Prieto. Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations. *Neurocomputing, vol.31, pp.87-103*, Jan. 2000.

[5] G. Bolt. Fault tolerant in multi-layer perceptrons. *PhD Thesis, University of York, UK,*, 1992.

[6] Massimo Buscema. Metanet: The theory of independent judges. *Substance Use and Misuse, Volume 33, Issue 2, pp. 439-461.*, Jan. 1998.

[7] S. Cavalieri and O. Mirabella. A novel learning algorithm which improves the partial fault tolerance of multilayer nns. *Neural Networks, Vol.12, 91-106*, 1999.

[8] S. Chen. Local regularization assisted orthogonal least squares regression. *Neurocomputing, pp.559-585*, 2006.

[9] D. Deodhare, M. Vidyasagar, and S. Sathiya Keerthi. Synthesis of fault-tolerant feedforward neural networks using minimax optimization. *IEEE Transactions on Neural Networks, Vol.9(5), 891-900*, 1998.

[10] P.J. Edwards and A.F. Murray. Can deterministic penalty terms model the effects of synaptic weight noise on network fault-tolerance? *International Journal of Neural Systems, 6(4):401-16*, 1995.

[11] P.J. Edwards and A.F. Murray. Fault tolerant via weight noise in analog vlsi implementations of mlp's – a case study with epsilon. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol.45, No.9, p.1255-1262*, Sep. 1998.

[12] C.T. Chiu *et al.* Modifying training algorithms for improved fault tolerance. *ICNN'94 Vol.I, 333-338*, 1994.

[13] N. Kamiura *et al.* On a weight limit approach for enhancing fault tolerance of feedforward neural networks. *IEICE Transactions on Information & Systems, Vol. E83-D, No.11*, 2000.

[14] R. Velazco *et al.* Seu fault tolerance in artificial neural netwok. *IEEE Transactions on Nuclear Science, Vol.42(6), pp.1856-1862*, 1995.

[15] N.C. Hammadi and I. Hideo. A learning algorithm for fault tolerant feedforward neural networks. *IEICE Transactions on Information & Systems, Vol. E80-D, No.1*, 1997.

[16] B. Hassibi and D.G. Stork. Second order derivatiives for network prunning: Optimal brain surgeon. in hanson *et al. Advances in Neural Information Processing Systems, 164-171*, 1993.

[17] S. Himavathi, D. Anitha, and A. Muthuramalingam. Feedforward neural network implementation in fpga using layer multiplexing for effective resource utilization. *IEEE Transactions on Neural Networks, Vol. 18, pp.880-888*, 2007.

[18] K. Ho, C.S. Leung, and J. Sum. On weight-noise-injection training, m.koeppen, n.kasabov and g.coghill (eds.). *Advances in Neuro-Information Processing, Springer LNCS 5507, pp. 919–926*, 2009.

[19] K.C.Jim, C.L. Giles, and B.G. Horne. An analysis of noise in recurrent neural networks: Convergence. *IEEE Transactions on Neural Networks, Vol.7, 1424-1438*, 1996.

[20] E. W. M. Lee, C. P. Lim, R. K. K. Yuen, and S. M. Lo. A hybrid neural network model for noisy data regression. *IEEE Trans. Syst. Man Cybern. B., Cybern., vol. 34, no. 2, pp. 951-960*, April 2004.

[21] C.S. Leung and J. Sum. A fault tolerant regularizer for rbf networks. *IEEE Transactions on Neural Networks, Vol. 19 (3), pp.493-507*, 2008.

[22] C.S. Leung, K.W. Wong, J. Sum, and L.W. Chan. On-line training and prunning for rls algorithm. *Electronics Letters, Vol.32, No.23, pp.2152-2153*, 1996.

[23] C.S. Leung, K.W. Wong, P.F. Sum, and L.W. Chan. A prunning method for recursive least square algorithm. *Neural Networks*, 2001.

[24] C.S. Leung, G.H. Young, J. Sum, and W.K. Kan. On the regularization of forgetting recusive least square. *IEEE Transactions on Neural Networks, Vol. 10, pp.1842-1846*, 1999.

[25] J.E. Moody. Note on generalization, regularization, and architecture selection in nonlinear learning system. *First IEEE-SP Workshop on Neural Networks for Signal Processing*, 1991.

[26] N. Murata, S. Yoshizawa, and S. Amari. Netwoek information criterion - determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks Vol.5(6), pp.865-872*, 1994.

[27] A.F. Murray and P.J. Edwards. Synaptic weight noise during multilayer perceptron training: fault tolerance and training improvements. *IEEE Transactions on Neural Networks, Vol.4(4), 722-725*, 1993.

[28] A.F. Murray and P.J. Edwards. Enhanced mlp performance and fault tolerance resulting from synaptic weight noise during training. *IEEE Transactions on Neural Networks, Vol.5(5), 792-802*, 1994.

[29] M.W. Pederson, L.K. Hansen, and J. Larsen. Prunning with generalization based weight saliencies: γobd, γobs. *Advances in Neural Information Processing Systems 8, pp.512-528*, 1996.

[30] D.S. Phatak and I. Koren. Complete and partial fault tolerance of feedforward neural nets. *IEEE Transactions on Neural Networks, Vol.6, 446-456*, 1995.

[31] R. Reed. Prunning algorithm - a survey. *IEEE Transactions on Neural Networks Vol.4(5), pp.740-747*, 1993.

[32] Antony W. Savich, Medhat Moussa, and Shawki Areibi. The impact of arithmetic representation on implementing mlp-bp on fpgas: A study. *IEEE Transactions on Neural Networks, Vol. 18, pp.240-252*, 2007.

[33] Neti C. M.H. Schneider and E.D. Young. Maximally fault tolerance neural networks. *IEEE Transactions on Neural Networks, Vol.3(1), 14-23*, 1992.

[34] C.H. Sequin and R.D. Clay. Fault tolerance in feedforward artificial neural networks. *Neural Networks, Vol.4, 111-141*, 1991.

[35] S. Singth. Noise impact on time-series forecasting using an intelligent pattern maching technique. *Pattern Recognit., vol. 32, pp.1389-1398*, 1999.

[36] M. Sugiyama and H. Ogawa. Optimal design of regularization term and regularization parameter by subspace information criterion. *Neural Networks, Vol.15, 349-361*, 2002.

[37] J. Sum and K. Ho. Sniwd: Simultaneous weight noise injection with weight decay for mlp training. *Proc. ICONIP 2009, Bangkok Thailand*, 2009.

[38] J. Sum, C.S. Leung, and K. Ho. On objective function, regularizer and prediction error of a learning algorithm for dealing with multiplicative weight noise. *IEEE Transactions on Neural Networks Vol.20(1), Jan, 2009*, 2009.