SNIWD: Simultaneous Weight Noise Injection With Weight Decay for MLP Training

John Sum[1] and Kevin Ho[2]

[1] Institute of Technology Management, National Chung Hsing University
Taichung 402, Taiwan. pfsum@nchu.edu.tw
[2] Department of Computer Science and Communication Engineering,
Providence University, Sha-Lu, Taiwan. ho@pu.edu.tw

Abstract. Despite noise injecting during training has been demonstrated with success in enhancing the fault tolerance of neural network, theoretical analysis on the dynamic of this noise injection-based online learning algorithm has far from complete. In particular, the convergence proofs for those algorithms have not been shown. In this regards, this paper presents an empirical study on the non-convergence properties of injecting weight noises during training a multilayer perceptron, and an online learning algorithm called SNIWD (simultaneous noise injection and weight decay) to overcome such non-convergence problem. Simulation results show that SNIWD is able to improve the convergence and enforce small magnitude on the network parameters (input weights, input biases and output weights). Moreover, SNIWD is able to make the network have similar fault tolerance ability as using pure noise injection approach.

1 Introduction

Improve tolerance of a neural network towards random node fault, stuck-at node fault and weight noise have been researching for almost two decades [4, 6, 5, 7, 9, 12, 13, 16, 17, 19], Many methods such as injecting random node fault [18, 3], injecting weight noise during training (for multilayer perceptrons (MLP) [14, 15], a recurrent neural network (RNN) [11], or a pulse-coupled neural networks (PCNN) [8]) or node noise (response variability) during training [2] (for PCNN) during training have been developed and demonstrated with success via intensive computer simulations. Despite the idea of injecting weight noise during training is straight forward and its implementation is extremely elegant, theoretical analysis regrading their convergence and the objective functions in which the algorithms are minimizing is scarce [1, 2, 14, 15].

Murray & Edward although have found that injecting multiplicative weight noise can enhance the fault tolerance of a MLP [15], they have not put forward the objective function for this algorithm. While G.An in [1] has attempted to derive an objective function for injecting weight-noise during training (see Section 4 in [1]), he failed to prove the convergence of this algorithm and nevertheless the objective function derived is not the true one. In terms of Murray & Edward's terminology, the objective derived by G.An is essentially the prediction error of a MLP if weight noise is injected after training. Until very recent, Ho *et al* [10] have shown the first complete analysis on the convergence of injecting output weight noise (either multiplicative or additive) during training a radial basis function (RBF) network.

In view of lacking understand on injecting weight noise during training a MLP, simulated experiments have been conducted. We found that pure noise injection during training might lead to non-convergence of network parameters, even the training error has been converged. Rather, adding weight decay together with noise injection during training is able to overcome such non-convergence problem. In this paper, we will present this comparative study based on purely noise injection training algorithm and simultaneous weight noise injection with weight decay (SINWD).

In the next section, the online weight noise injection algorithms will be presented. Their convergence properties, in terms of training error and network parameters, and their fault tolerance abilities will be shown by a simple example in Section 3. Section 4 gives the conclusions of this paper.

2 Noise Injection During Training

Let $\mathbf{f}(\cdot,\cdot) \in R^l$ be a single output multilayer perceptron (MLP) consisting of m hidden nodes, n input nodes and l linear output nodes.

$$\mathbf{f}(\mathbf{x},\mathbf{w}) = \mathbf{D}^T\mathbf{z}(\mathbf{A}^T\mathbf{x}+\mathbf{c}), \tag{1}$$

where $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_l] \in R^{m\times l}$ is the hidden to output weight vector, $\mathbf{z} = (z_1, z_2, \cdots, z_m)^T \in R^m$ is the output of the hidden nodes, $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_m] \in R^{n\times m}$ is the input to hidden weight matrix, $\mathbf{a}_i \in R^n$ is the input weight vector of the i^{th} hidden node and $\mathbf{c} \in R^m$ is the input to hidden bias vector.

$\mathbf{w}$ in (1) is a vector augmenting all the parameters, i.e.

$$\mathbf{w} = (\mathbf{d}_1^T, \mathbf{d}_2^T, \cdots, \mathbf{d}_l^T, \mathbf{a}_1^T, \mathbf{a}_2^T, \cdots, \mathbf{a}_m^T, \mathbf{c}^T)^T.$$

For $i = 1, 2, \cdots, m$, $z_i(\mathbf{x}, \mathbf{a}_i, c_i) = \tau(\mathbf{a}_i^T\mathbf{x}+c_i)$, where $\tau(\cdot)$ is the neuronal transfer function. Training dataset is denoted by $\mathcal{D} = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^N$. The random noise vector is denoted by $\mathbf{b}$. For simplicity, we assume that there is only one output node, i.e. $l = 1$. In such case, the gradient of $f(\mathbf{x},\mathbf{w})$ with respect to $\mathbf{w}$ is denoted by $g(\mathbf{x}_t, \mathbf{w}(t))$. The Hessian matrix of $f(\mathbf{x},\mathbf{w})$ is denoted by $g_{\mathbf{w}}(\mathbf{x}_t, \mathbf{w}(t))$.

2.1 Pure Weight Noise Injection

The online **weight noise injection** training algorithm for $f(\mathbf{x},\mathbf{w})$ given a dataset $\mathcal{D}$ can be written as follows :

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu_t(y_t - f(\mathbf{x}_t, \tilde{\mathbf{w}}(t)))\mathbf{g}(\mathbf{x}_t, \tilde{\mathbf{w}}(t)). \tag{2}$$

$$\tilde{\mathbf{w}}(t) = \mathbf{w}(t) + \mathbf{b} \odot \mathbf{w}(t). \quad \text{(multiplicative weight noise)} \tag{3}$$

$$\tilde{\mathbf{w}}(t) = \mathbf{w}(t) + \mathbf{b}. \quad \text{(additive weight noise)} \tag{4}$$

	Pure noise injection	With weight decay
Add. weight noise (Fig 1)	$S_b = .01$, $\alpha = 0$	$S_b = .01$, $\alpha = .00001$
Mul. weight noise (Fig 3)	$S_b = .01$, $\alpha = 0$	$S_b = .01$, $\alpha = .00001$

Table 1. Settings of the experiments.

Here $\mathbf{b} \odot \mathbf{w} = (b_1 w_1, b_2 w_2, \cdots, b_M w_M)^T$ and b_i, for all i, is a mean zero Gaussian distribution with variance S_b.

2.2 SNIWD

For simultaneous weight noise injection and weight decay (SNIWD), the update equations are similar except the decay term is added.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \mu_t \left\{ (y_t - f(\mathbf{x}_t, \tilde{\mathbf{w}}(t)))\mathbf{g}(\mathbf{x}_t, \tilde{\mathbf{w}}(t)) - \alpha \mathbf{w}(t) \right\}. \quad (5)$$

$$\tilde{\mathbf{w}}(t) = \mathbf{w}(t) + \mathbf{b} \odot \mathbf{w}(t). \quad \text{(multiplicative weight noise)} \quad (6)$$

$$\tilde{\mathbf{w}}(t) = \mathbf{w}(t) + \mathbf{b}. \quad \text{(additive weight noise)} \quad (7)$$

Clearly, the difference between pure noise injection during training, and the one with weight decay lies in the last term of the update equation, i.e. $-\alpha \mathbf{w}(t)$, which can limit the growth of $\|\mathbf{w}(t)\|$ to infinity.

3 Simulation Study

To illustrate the effect of injection noise during training MLP with and without adding weight decay, a training dataset consisting of 100 samples that are generated from an XOR function is used for the MLP training.

Let $(\mathbf{x}_k, y_k)$ be the k^{th} input and output pair. $\mathbf{x}_k$ is uniformly random selected from $[-1, 1] \times [-1, 1]$. The output vector $y_k \in \{0, 1\}$ is generated by the following equation.

$$y_k = \mathbf{sign}(x_{k1})\mathbf{sign}(x_{k2}). \quad (8)$$

Then, an MLP consisting of 2 input nodes, 10 hidden nodes and 1 linear output nodes is trained. Four experiments are carried out. The values of S_b and α are depicted in Table 1. The step size for all eight experiments is set to 0.05. The change of parameters during training are shown in Figure 1 and Figure 3. To validate the fault tolerance ability, each network that is trained with online additive (multiplicative) weight noise injection will be injected S'_b additive (multiplicative) weight noise after training and then the testing error is evaluated. The last step is repeated 100 times, the statistics of the testing errors are displayed in box-plot form and shown in Figure 2 and Figure 4 respectively for additive weight noise injection and multiplicative weight noise injection. The range of S'_b is from 0 to 0.04.

In accordance with the simulation results, it is clear that the network parameters do not converge for pure weight noise injection cases. Even the training

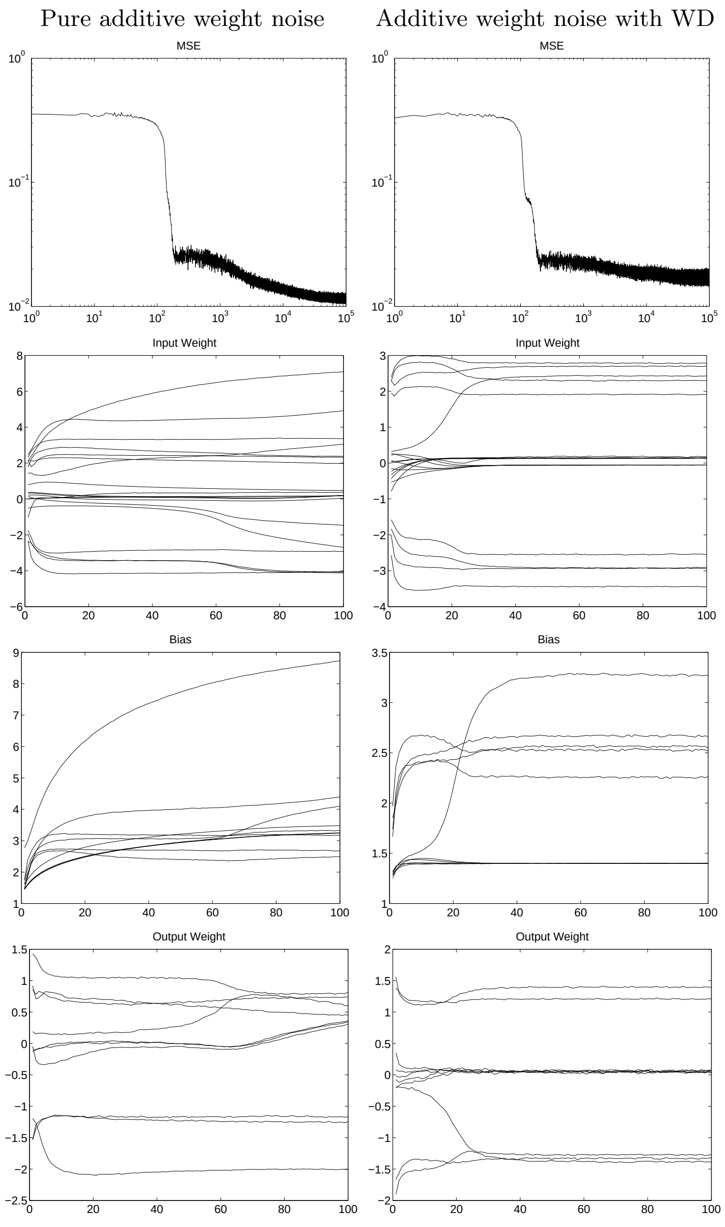

Fig. 1. Dynamical changes of the network parameters while additive weight noise is injected during training. Note that the total number of training steps is 100×1000. Every two consecutive points are taken at an interval of 1000 steps.

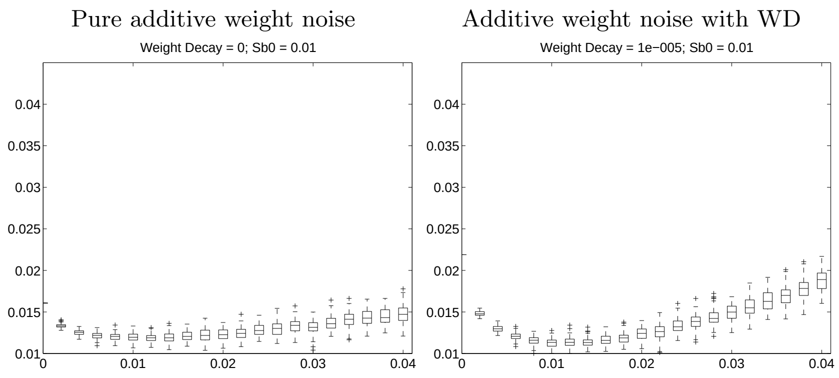

Fig. 2. Testing error versus different values of S_b', for the networks obtained in Figure 1.

error has shown converge, many network parameters are still increasing. Adding weight decay is able to control the growth of the network parameters, especially the input weights. If weight decay is added, their magnitudes converge to below 4. Without weight decay, their magnitude can diverge to as large as 10, see Figure 3.

Moreover, as observed from Figure 2 and Figure 4 that the fault tolerance abilities of a network trained by pure noise injection and SNIWD are quite similar. Except that, SNIWD gives slightly better performance when S_b' is close to 0.01. For S_b' is larger than 0.02, the situation is reverse.

4 Conclusion

In this paper, we have presented simulation results comparing the convergence of network parameters (including input weights, input biases and output weights) that are obtained by purely noise injection and simultaneous noise injection with weight decay. We have found that purely injecting weight noise during training a MLP might not be able to improve its fault tolerance, as the some of network parameters might diverge. By simulations, we have found that adding weight decay simultaneously with weight noise injection during training is able to overcome such problem. For a network that is trained by SNIWD approach, its network parameters are with smaller magnitude compared with pure weight noise injection approach. Convergence of network parameters is almost guaranteed. The fault tolerance ability of that network is comparable to that is trained by purely noise injection approach. Due to page limit, we are not able to derive the objective functions in which those algorithms are minimizing in this paper. Those theoretical results will be presented in our future papers.

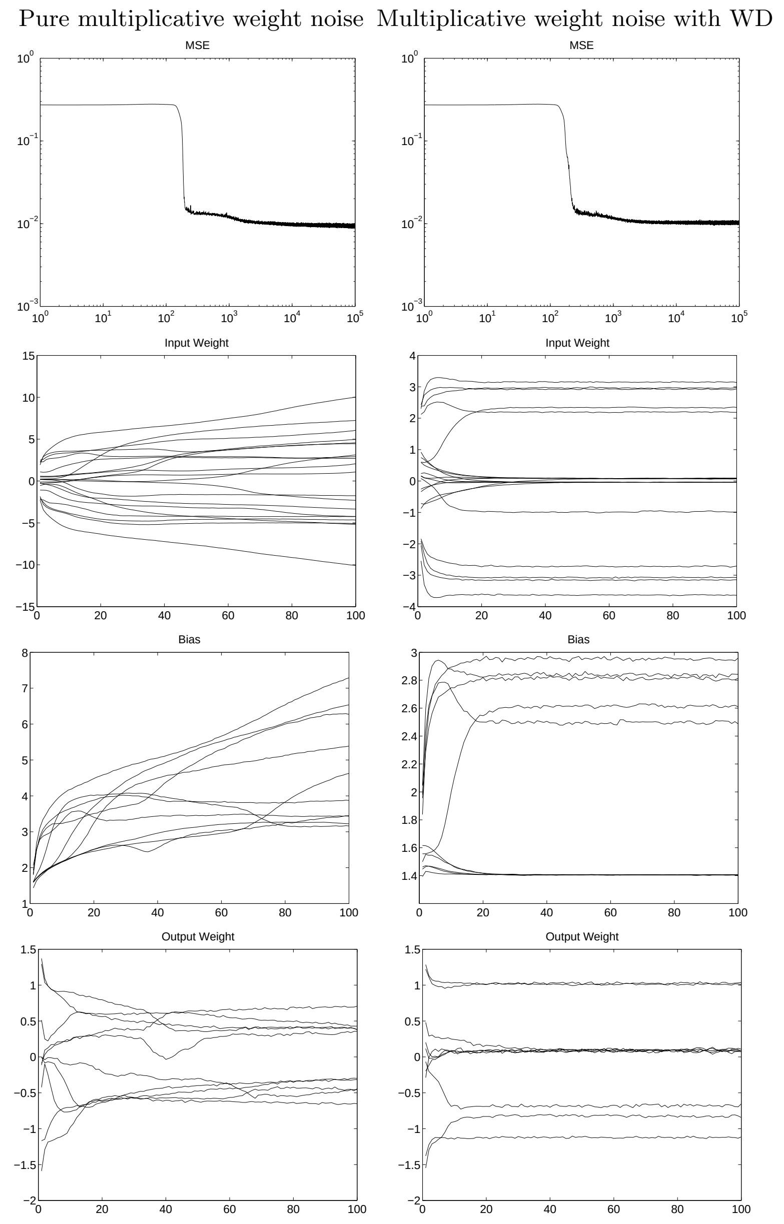

Fig. 3. Dynamical changes of the network parameters while additive weight noise is injected during training. Note that the total number of training steps is 100×1000. Every two consecutive points are taken at an interval of 1000 steps.

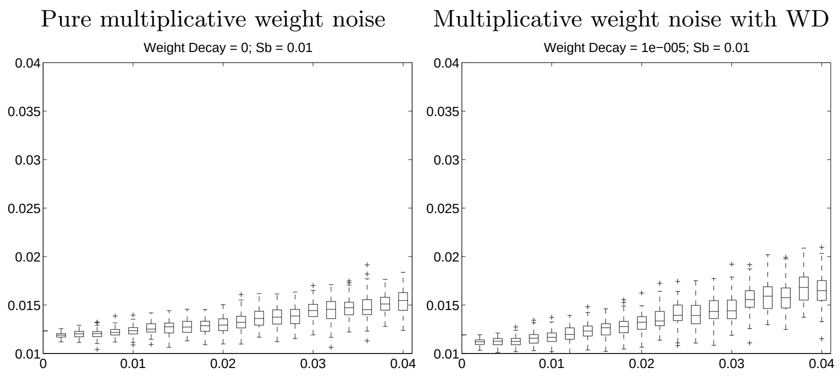

Fig. 4. Testing error versus different values of S'_b, for the networks obtained in Figure 3.

Acknowledgement

The research work reported in this paper is supported in part by Taiwan NSC Research Grant 97-2221-E-005-050.

References

1. An G. The effects of adding noise during backpropagation training on a generalization performance, *Neural Computation*, Vol.8, 643-674, 1996.
2. Basalyga G. and E. Salinas, When response variability increases neural network robustness to synaptic noise, *Neural Computation*, Vol.18, 1349-1379, 2006.
3. Bolt G., *Fault tolerant in multi-layer Perceptrons*. PhD Thesis, University of York, UK, 1992.
4. Bernier J.L. *et al*, Obtaining fault tolerance multilayer perceptrons using an explicit regularization, *Neural Processing Letters*, Vol.12, 107-113, 2000.
5. Cavalieri S. and O. Mirabella, A novel learning algorithm which improves the partial fault tolerance of multilayer NNs, *Neural Networks*, Vol.12, 91-106, 1999.
6. Chiu C.T. *et al.*, Modifying training algorithms for improved fault tolerance, ICNN'94 Vol.I, 333-338, 1994.
7. Deodhare D., 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.
8. Edwards P.J. 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.
9. Hammadi N.C. and I. Hideo, A learning algorithm for fault tolerant feedforward neural networks, *IEICE Transactions on Information & Systems*, Vol. E80-D, No.1, 1997.
10. Ho K., 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. 919V926, 2009.

11. Jim K.C., C.L. Giles and B.G. Horne, An analysis of noise in recurrent neural networks: Convergence and generalization, *IEEE Transactions on Neural Networks*, Vol.7, 1424-1438, 1996.
12. Kamiura N., *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.
13. Leung C.S., J. Sum, A fault tolerant regularizer for RBF networks, *IEEE Transactions on Neural Networks*, Vol. 19 (3), pp.493-507, 2008.
14. Murray A.F. 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.
15. Murray A.F. 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.
16. 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.
17. Phatak D.S. and I. Koren, Complete and partial fault tolerance of feedforward neural nets., *IEEE Transactions on Neural Networks*, Vol.6, 446-456, 1995.
18. Sequin C.H. and R.D. Clay, Fault tolerance in feedforward artificial neural networks, *Neural Networks*, Vol.4, 111-141, 1991.
19. Sum J., 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.