Synaptic Noise in Dynamically-driven Recurrent Neural Networks: Convergence and Generalization†

Kam Jim C. Lee Giles* Bill G. Horne

NEC Research Institute, Inc.
4 Independence Way
Princeton, NJ 08540
{kamjim,giles,horne}@research.nj.nec.com

*Also with
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

† Technical Report
UMIACS-TR-94-89 and CS-TR-3322
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

May 1994

Abstract

There has been much interest in applying noise to feedforward neural networks in order to observe their effect on network performance. We extend these results by introducing and analyzing various methods of injecting *synaptic noise* into dynamically-driven recurrent networks during training. By analyzing and comparing the effects of these noise models on the error function, we found that applying a controlled amount of noise during training can improve convergence time and generalization performance. In addition, we analyze the effects of various noise parameters (additive vs. multiplicative, cumulative vs. non-cumulative, per time step vs. per sequence) and predict that best overall performance can be achieved by injecting additive noise at each time step. Noise contributes a second-order gradient term to the error function which can be viewed as an *anticipatory agent* to aid convergence. This term appears to find promising regions of weight space in the beginning stages of training when the training error is large and should improve convergence on error surfaces with local minima. Synaptic noise also enhances the error function by favoring internal representations where state nodes are operating in the saturated regions of the sigmoid discriminant function, thus improving generalization to longer sequences. We substantiate these predictions by performing extensive simulations on learning the dual parity grammar from grammatical strings encoded as temporal sequences with a second-order fully recurrent neural network.

1 INTRODUCTION

There has been a great deal of work on enhancing neural network performance. Two important performance parameters are convergence and generalization. Convergence is the amount of time, measured in CPU operations or training epochs, required to find an acceptable solution during training. Generalization, a term borrowed from psychology, measures the ability to correctly classify new unseen data.

Among some of the important methods that have been introduced to improve convergence is the notion of searching the coarse regions of state space before searching finer regions. *Simulated annealing* is one such method, and was shown in [12] to be very effective on combinatoric problems like graph partitioning. Similarly, in [8] noise was used on Boltzmann machines to escape from local minima. Specifically, it was indicated that at high noise temperatures Boltzmann machines will search the overall structure of state space for a coarse minimum, while subsequently lowering the noise temperature will allow a better minimum to be found within the coarse-scale minimum.

It has been shown that generalization can be improved by removing the redundancies and irrelevancies in a network. For example, *optimal brain damage* improves generalization ability and speed of learning by using second-derivative information to remove unimportant weights from the network [3]. *Weight decay* was shown to improve generalization on feed-forward networks by suppressing irrelevant components of the weight vector [13]. Still another network simplification method is *pruning*, which has demonstrated [7] improvement in generalization in recurrent neural networks.

1.1 Previous Work on Training with Noise

Previous research has investigated the effects of *noise* on feedforward neural networks. Training with noise can lead to more realistic biological models. For example in [8] noise temperature in the Boltzmann machine was compared to the addition of Gaussian noise at neuronal membrane potentials, and [11] introduced a *stochastic neuron* model which uses Monte Carlo techniques to stochastically bias the firing decision of neurons. More relevant to this paper however, are the experiments that use noise injection during training to enhance network performance.

Much work in training with noise has been motivated by the desire to improve *fault tolerance*, i.e. the graceful degradation of performance in the presence of faults. Judd and Munro [10] showed that using noisy hidden nodes during training can result in error-correcting codes which increase the tolerance of feedforward nets to unreliable nodes. Similarly, Sequin and Clay [18] showed that randomly disabling hidden nodes (i.e. clamp to zero) during the training phase increases the tolerance of multilayer perceptrons to node failures. In [15], Minnix also demonstrated improved fault tolerance by training on noisy inputs, and argued that such a method helps prevent *pattern memorization*. Murray and Edwards [16] utilized *synaptic noise* during training to improve fault tolerance and training quality.

With the exception of [16], these methods generally indicate convergence as a trade-off for enhanced performance. Recently, stochastic techniques, such as Nodal Perturbation, Weight Perturbation, and Summed Weight Neuron Perturbation, have been introduced which in some cases may converge more quickly than pure gradient descent [9, 5, 2, 4]. These methods in general require more epochs to converge but less computation per epoch, mainly because calculation of the true gradient is not required.

To date, there has been little work on applying these noise insertion methods to recurrent neural networks. In this paper, we extend these results by exploring the effects of synaptic noise on recurrent neural networks. We introduce several ways to inject noise during training, and demonstrate that descending these *noise-enhanced* error functions can improve convergence and generalization. Previous work on improving these two performance measures have been focused on ways of first searching the coarse regions of state space and simplifying the network. The function of synaptic noise in recurrent nets is novel in the sense that synaptic noise can improve convergence by searching for *promising regions* of state space, and at the same time enhance generalization by favoring *saturated* state nodes. Promising regions of state space do not necessarily correspond to coarsely-spaced local minima, and saturated state nodes are in general different from a simplified network.

1.2 Outline of paper

After the introduction we give in section 2 a brief motivation of our problem and choice of neural network. Section 3 defines the second-order recurrent neural networks and introduces several methods of implementing synaptic noise. We discuss different noise models and compare their effects on the error function in section 4. From this analysis, predictions are made regarding their effects on convergence and generalization performance. Section 5 presents simulation results on the dual parity problem. These results are general to all incremental weight updating methods and should extend easily to other recurrent networks.

2 PROBLEM CHOICE AND RECURRENT NETWORK SELECTION

In order to perform many experiments in a reasonable amount of computation time, we attempt to learn the simple dual parity grammar from sample strings encoded as temporal sequences. Dual parity is a 4-state automaton that recognizes binary strings containing an even number of ones and zeroes (see figure 1) and is a classic example of a "hidden-state" problem. We also made the problem very learnable in the sense that we gave the neural network ample strings for effective training. For network architecture we chose a second-order discrete-time recurrent network since such networks have demonstrated good performance on such problems [6, 14, 19]. As such our experiments consist of 500 simulations for each data point and achieve useful (90%) confidence levels.

3 NOISE INJECTION IN RECURRENT NEURAL NETWORKS

3.1 Noiseless Network Architecture and Training

We briefly review the second-order fully recurrent neural network, string (sequence) encoding, error function, training algorithm and data presentation. For more details see the references.

The second-order discrete-time recurrent network architecture consists of N recurrent state neurons S_j and L non-recurrent input neurons I_k. The states are related by a set of N^2L weights W_{ijk}. The recurrent network operation resembles that of a finite state machine with the state process: {input, current state} $\Longrightarrow$ {next state}, as defined by the following equations:

$$S_i^{t+1} = g(\Theta_i^{t+1}) \tag{1}$$

$$\Theta_i^{t+1} = \sum_{j,k} W_{ijk} S_j^t I_k^t \tag{2}$$

where Θ_i is the net input to unit i, and $g(x)$ is a predetermined sigmoid discriminant function. In the simulations described in section 5, $g(x)$ is defined as

$$g(x) = \frac{1}{1+e^{-x}} \tag{3}$$

.

"One-hot" or "unary" input encoding is used. The input symbols are encoded as orthonormal unit vectors in L-dimensional space, so the inputs are in the form $I_k = \delta_{k\alpha}$ for $(0 \leq \alpha \leq L)$. For the dual parity DFA, $L = 3$, in order to represent the binary input characters 0, 1, and the empty string **e**. Hence, the possible symbols are (0,0,1), (0,1,0), and (1,0,0).

Each input string is presented to the network as a temporal sequence one character per discrete time step t. At time $t = T$, when the entire input string has been presented to the network, the value at a selected neuron S_O is defined to be the output. An input string is accepted if $S_O > 1 - \epsilon$, or rejected if $S_O < \epsilon$, where ϵ is the *error tolerance* and is a positive real number. Weights are updated at the end of each string.

During training, a squared-error function is computed from

$$E_p = \frac{1}{2}\epsilon_p^2 = \frac{1}{2}(S_O^T - d_p)^2 \tag{4}$$

where d_p is the target output value for pattern p (either 1 or 0 depending on the DFA response of the input string), and ϵ_p is the raw error.

We used the Back-Propagation-Through-Time (BPTT) training algorithm [20]. Incremental learning, as described in [6], is used to descend the error function. Basically, the network is trained on a subset of the training set, called the *working set*, which gradually increases in size until the network is able to correctly classify the entire training set. Strings from the working set are presented to the network in *alphabetical order*. Incremental learning reduces the training time required because often not all the strings in the training set need to be seen before the network can classify the entire set correctly. Training stops when all strings in the training set are correctly classified.

3.2 Noise Injection

Murray and Edwards [16] simulated synaptic noise on multilayered perceptrons (MLPs) by applying noise to the weights of each layer during training. Applying this method to recurrent networks is not straightforward because effectively the same weights are propagated forward in time. This can be seen by recalling the BPTT representation of *unrolling* a recurrent network in time into T layers with identical weights, where T is the length of the temporal sequence. As in [16] we will only focus on noise injection in synaptic weights and ignore other methods of noise insertion.

Because of its temporal nature, synaptic noise injection of recurrent networks introduces two issues which do not pertain to feedforward nets:

1. What is the frequency of noise injection?
2. Is the noise injection cumulative or non-cumulative?

This is a direct result of the fact that recurrent networks effectively propagate the same weights forward in time. Should one set of noise values also be propagated forward, or should a different set of noise values be applied at each time step? Or, should noise be applied only at the beginning of each input sequence? If noise is propagated forward in time, should it accumulate in the weights over all time steps, or can the noise be inserted independently each time step?

In this paper we consider the cases where noise is propagated forward in time. We introduce a *per time step* method of injecting synaptic noise into recurrent networks which inserts a new random noise value at each time step of the forward pass during training. On the other hand, a *per sequence* model can be viewed as a special case of the per time step model such that the same set of noise values is injected for all time steps of each given input string.

Non-cumulative models apply a new noise value Δ_{ijk} to a noise-free weight at each instance of noise injection, while cumulative models apply a new noise to a previously perturbed weight at each instance. Non-cumulative noise can be represented as

$$W_{t,ijk} = N(W^*_{ijk}), \tag{5}$$

where $\{W^*_{ijk}\}$ is the noise-free weight set and N is the noise process applied to weight W^*_{ijk}. Cumulative noise be represented by

$$\begin{aligned} W_{0,ijk} &= N(W^*_{ijk}), & (6)\\ W_{t,ijk} &= N(W_{t-1,ijk}), t > 0. & (7) \end{aligned}$$

An issue common to both feedforward and recurrent networks is whether noise is additive or multiplicative. Additive noise perturbs a weight parameter such that

$$W_{ijk} \longrightarrow W_{ijk} + \Delta_{ijk}, \tag{8}$$

where Δ_{ijk} is a random noise value. On the other hand, multiplicative noise perturbs each weight such that

$$W_{ijk} \longrightarrow W_{ijk} + W_{ijk}\Delta_{ijk}. \tag{9}$$

Table 1 clarifies these issues by illustrating the noise injection steps for all additive noise processes. The Δ's are the various noise terms, and W^* is the noise free-weight. For per time step non-cumulative noise insertion a different noise is added at each time step and the noise from the previous time step does not accumulate in time. For the per time step cumulative noise insertion all previously added noise is carried over to the next time step. For per sequence non-cumulative, the same noise is inserted at the beginning and used throughout the temporal updates. For per sequence cumulative, the initial noise accumulates for each time step. Similar results would be seen for the multiplicative noise case.

We report experiments which explore all combinations of additive vs. multiplicative, cumulative vs. non-cumulative, and per sequence vs. per time step noise models. In all noise models the actual weight updates ΔW_{ijk}'s descend the noise-enhanced error term E_p. These updates are applied to the noise-free weight set. This prevents "noisy updates", allows simpler mathematical analysis, and for the purpose of these experiments has the desirable effect of limiting the effect of noise to the error function.

For all cases the noise source is chosen to be a random, zero-mean white noise process uniformly distributed between (-**a**, **a**), where **a** is a positive constant parameter.

	t_1	t_2	t_3	$\cdots$
per time step non-cumulative	$W^* + \Delta_1$	$W^* + \Delta_2$	$W^* + \Delta_3$	$\cdots$
per time step cumulative	$W^* + \Delta_1$	$W^* + \Delta_1 + \Delta_2$	$W^* + \Delta_1 + \Delta_2 + \Delta_3$	$\cdots$
per sequence non-cumulative	$W^* + \Delta_1$	$W^* + \Delta_1$	$W^* + \Delta_1$	$\cdots$
per sequence cumulative	$W^* + \Delta_1$	$W^* + 2\Delta_1$	$W^* + 3\Delta_1$	$\cdots$

Table 1: Sample noise injection steps for all additive noise models at time steps t_1, t_2, and t_3. W^* is the noise-free weight. Δ's are the noise terms for each time step.

3.3 Memory Requirements for Noise Insertion

Memory requirements are a function of the frequency of noise injection and whether cumulative or non-cumulative noise is used. They are also a function of the training algorithm. Noise-free, second-order fully recurrent networks require N^2L weights. This section presents the additional memory requirements of each noise model assuming the back-propagation-through time training algorithm. This analysis is simple for any real-time training algorithm such as real-time recurrent learning or the extended Kalman estimation [17].

In general, the network sees a newly perturbed set of weights at each time step t during the forward pass of training. These weights need to be stored in memory for gradient calculations on the backward pass. Injecting a new set of noise values at each time step increases memory requirements linearly by a factor of $T + 1$, in order to store the set of noise-free weights and the T sets of noisy weights. Hence, $N^2L(T + 1)$ weights need to be remembered, where T is taken to be the length of the longest training pattern.

Per sequence noise injection can be viewed as a special case where the noise sets are equivalent for all time steps. For cumulative models, $N^2L(T + 1)$ weights can be stored to represent the newly perturbed weight sets at each time step. Alternatively, N^2L weights and N^2L noise values can be stored, with the weights at each time step recalculated from these two sets. Non-cumulative per sequence models require only $2N^2L$ weights, to store the single set of noisy weights along with the set of noise-free weights. Since the noisy weights are equivalent at each time step, no additional calculations are required.

4 ANALYSIS OF SYNAPTIC NOISE

The effects of each noise model is analyzed by taking the Taylor expansion on the error function around the noise-free weight set. By truncating this expansion to second order and lower terms, we can interpret the effect of noise as a set of regularization terms applied to the error function. From these terms predictions can be made about the effects on generalization and convergence. A similar analysis was performed by Murray and Edwards [16] on MLP's to demonstrate the effects of synaptic noise on fault tolerance and training quality. Tables 2 and 3 list the noise injection step and resulting first and second order Taylor expansion terms for all noise models. These results are derived in Appendix A.

4.1 Generalization Improvement due to Synaptic Noise

We (and others) propose that two ways to improve generalization are

- Reduce the raw error, and
- Improve the internal representation of information in the network.

Reducing the raw error ϵ_p improves the tolerance of the network to perturbations in the output nodes. This tolerance provides a buffer from which slight variations in the output node will lead to the same classification, thus improving generalization. At the same time, it has been shown that on networks with too many parameters over-training can lead to poor generalization, since reducing the raw error to zero causes the network to effectively memorize the training set [1].

One common cause of poor generalization in recurrent networks that are trained on hidden-state problems is the presence of unsaturated state representations [21]. Typically, a network cannot revisit the exact same point in state space, but tends to wander away from its learned state representation. One approach to alleviate this problem is to encourage state nodes to operate in the saturated regions of the sigmoid. The

	NON-CUMULATIVE	
	Additive	Multiplicative
Noise step	$W_{t,ijk} = W^*_{ijk} + \Delta_{t,ijk}$	$W_{t,ijk} = W^*_{ijk} + W^*_{ijk}\Delta_{t,ijk}$
$1st$ order	$\frac{1}{2}\sigma^2 \sum_{t=0}^{T-1}\sum_{ijk}\left(\frac{\partial S_O^T}{\partial W_{t,ijk}}\right)^2$	$\frac{1}{2}\sigma^2 \sum_{t=0}^{T-1}\sum_{ijk}\left(W_{t,ijk}\frac{\partial S_O^T}{\partial W_{t,ijk}}\right)^2$
$2nd$ order	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t=0}^{T-1}\sum_{ijk}\frac{\partial^2 S_O^T}{\partial (W_{t,ijk})^2}$	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t=0}^{T-1}\sum_{ijk}(W_{t,ijk})^2\frac{\partial^2 S_O^T}{\partial (W_{t,ijk})^2}$
	CUMULATIVE	
	Additive	Multiplicative
Noise step	$W_{t,ijk} = W^*_{ijk} + \sum_{\tau=0}^{t}\Delta_{\tau,ijk}$	$W_{t,ijk} = W^*_{ijk}\prod_{\tau=0}^{t}(1+\Delta_{\tau,ijk})$
$1st$ order	$\frac{1}{2}\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} v\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,ijk}}$	$\frac{1}{2}\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} vW_{t,ijk}W_{u,ijk}\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,ijk}}$
$2nd$ order	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} v\frac{\partial^2 S_O^T}{\partial W_{t,ijk}\partial W_{u,ijk}}$	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} vW_{t,ijk}W_{u,ijk}\frac{\partial^2 S_O^T}{\partial W_{t,ijk}\partial W_{u,ijk}}$

Table 2: *Per time step* noise injection terms and order of error expansion terms for the error function for cumulative and non-cumulative noise for both the additive and multiplicative cases. $v = min(t+1, u+1)$. W^* is the noise-free weight set.

	NON-CUMULATIVE	
	Additive	Multiplicative
Noise step	$W_{t,ijk} = W^*_{ijk} + \Delta_{ijk}$	$W_{t,ijk} = W^*_{ijk} + W^*_{ijk}\Delta_{ijk}$
$1st$ order	$\frac{1}{2}\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk}\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,ijk}}$	$\frac{1}{2}\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} W_{t,ijk}W_{u,ijk}\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,ijk}}$
$2nd$ order	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk}\frac{\partial^2 S_O^T}{\partial W_{t,ijk}\partial W_{u,ijk}}$	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk} W_{t,ijk}W_{u,ijk}\frac{\partial^2 S_O^T}{\partial W_{t,ijk}W_{u,ijk}}$
	CUMULATIVE	
	Additive	Multiplicative
Noise step	$W_{t,ijk} = W^*_{ijk} + (t+1)\Delta_{ijk}$	$W_{t,ijk} = W^*_{ijk}(1+\Delta_{ijk})^t$ $= W^*_{ijk} + \sum_{\tau=0}^{t} c_{t,\tau}(\Delta_{ijk})^{\tau+1}$
$1st$ order	$\frac{1}{2}\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk}(t+1)(u+1)\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,ijk}}$	$\frac{1}{2}\sum_{t,u=0}^{T-1}\sum_{ijk,lmn}\Psi_{t,ijk}\Psi_{u,lmn}\frac{\partial S_O^T}{\partial W_{t,ijk}}\frac{\partial S_O^T}{\partial W_{u,lmn}}$ $+2\epsilon_p^2\sum_{t=0}^{T-1}\sum_{ijk}\Psi_{t,ijk}\frac{\partial S_O^T}{\partial W_{t,ijk}}$
$2nd$ order	$\frac{1}{2}\epsilon_p\sigma^2 \sum_{t,u=0}^{T-1}\sum_{ijk}(t+1)(u+1)\frac{\partial^2 S_O^T}{\partial W_{t,ijk}\partial W_{u,ijk}}$	$\frac{1}{2}\epsilon_p\sum_{t,u=0}^{T-1}\sum_{ijk,lmn}\Psi_{t,ijk}\Psi_{u,lmn}\frac{\partial^2 S_O^T}{\partial w_{t,ijk}\partial W_{u,lmn}}$

Table 3: *Per sequence* noise injection terms and order of error expansion terms for the error function for cumulative and non-cumulative noise for both the additive and multiplicative cases. $\Psi_{t,ijk} = W_{t,ijk}\sum_{\tau=0}^{t} c_{t,\tau}(\Delta_{ijk})^{\tau+1}$. W^* is the noise-free weight set.

first order error expansion terms of most noise models considered are capable of encouraging the network to achieve saturated states. This can be shown by applying the chain rule to the partial derivative in the first order expansion terms:

$$\frac{\partial S_O^T}{\partial W_{t,ijk}} = \frac{\partial S_O^T}{\partial \Theta_O^T} \sum_l \left[\left(\frac{\partial \Theta_O^T}{\partial S_l^{T-1}} \frac{\partial S_l^{T-1}}{\partial \Theta_l^{T-1}} \right) \cdots \sum_n \left(\frac{\partial \Theta_m^{t+1}}{\partial S_n^t} \frac{\partial S_n^t}{\partial W_{t,ijk}} \right) \right], \tag{10}$$

where Θ_i^t is the net input to state node i at time step t. The partial derivatives $\frac{\partial S}{\partial \Theta}$ favor internal representations such that the effects of perturbations at the net inputs Θ_i^t on the states S_i^t are minimized.

Multiplicative noise implements a form of weight decay because the error expansion terms include the weight products $W_{t,ijk}^2$ or $W_{t,ijk}W_{u,ijk}$. Although weight decay has been shown to improve generalization on feedforward networks [13], we hypothesize that in general weight decay will not always improve generalization for recurrent networks. that are learning finite state automata (FSA) problems. Large weights are necessary to saturate the state nodes to the upper and lower limits of the sigmoid discriminant function. Therefore, we predict additive noise will allow better generalization because of its absence of weight decay.

Noise models whose first order error term contain the expression $\frac{\partial S_O^T}{\partial W_{t,ijk}} \frac{\partial S_O^T}{\partial W_{u,lmn}}$ will favor saturated states for those partials whose sign correspond to the sign of a majority of all partials. These noise models will favor unsaturated states, operating in the linear region of the sigmoid, for partials whose sign is in the minority. Such *sign-dependence* is not optimal because not all the weights are encouraged to be saturated.

The error terms for cumulative per time step noises sum a product with the expression $v\frac{\partial S_O^T}{\partial W_{t,ijk}} \frac{\partial S_O^T}{\partial W_{u,lmn}}$, where $v = min(t+1, u+1)$. The effect of cumulative noise increases more rapidly because of v and thus optimal generalization and detrimental noise effects will occur at lower amplitudes than non-cumulative noise.

For cumulative per sequence noise models, the products $(t+1)(u+1)$ and $\Psi_{t,ijk}\Psi_{u,lmn}$ in the expansion terms rapidly overwhelm the raw error term. Generalization improvement is not expected for these models.

We also reason that all generalization enhancements will be valid only for a range of noise values, above which noise overwhelms the raw error information. The differences between the noise models are summarized in Table 4, where a check indicates that the corresponding enhancement is predicted. It is important to realize that although all noise models are theoretically capable of improving convergence and generalization by searching for promising weights and saturated states, Table 4 lists the *additional* enhancements to the search. From this table we see that additive per time step noise models should yield the best generalization performance.

4.2 Convergence Improvement due to Synaptic Noise

Convergence can be represented by the number of epochs required to learn the training set. This measure is dependent on the error trajectory followed during the training phase. On error surfaces filled with local minima many epochs may be required before the training set is learned, because the learning trajectory may enter an unacceptable local minima and face a difficult or impossible task of exiting from it.

Intuitively, synaptic noise may alleviate this problem by allowing the network to make noisy jumps out of local minima. Obviously, this will be beneficial only to learning problems whose error surfaces contain local minima. On simple error surfaces synaptic noise may worsen generalization performance by perturbing a straightforward training trajectory which would have otherwise led directly to a solution.

Synaptic noise can improve convergence by favoring *promising* weights in the beginning stages of training. This can be demonstrated by examining the second order error expansion term for non-cumulative, multiplicative, per time step noise:

$$\frac{1}{2}\epsilon_p \sigma^2 \sum_{t=0}^{T-1} \sum_{ijk} (W_{t,ijk})^2 \left(\frac{\partial^2 S_O^T}{\partial (W_{t,ijk})^2} \right).$$

When ϵ_p is negative, solutions with a negative second order state-weight partial derivative will be destabilized. In other words, when the output S_O^T is too small the network will favor updating in a direction such that the first order partial derivative is increasing. A corresponding relationship can be observed for

	Generalization Improvement	Generalization & Convergence Improvement
	Sign-independence favoring of saturated states	No additional weight terms
Per Timestep Cumulative		
Additive		$\surd$
Multiplicative		
Per Timestep Non-cumulative		
Additive	$\surd$	$\surd$
Multiplicative	$\surd$	
Per Sequence Cumulative		
Additive		$\surd$
Multiplicative		
Per Sequence Non-cumulative		
Additive		$\surd$
Multiplicative		

Table 4: Generalization and Convergence Improvement: Checks indicate additional enhancements to the search for saturated states/promising weights due to each noise model. Sign-independent enforcement of saturated states improves generalization, and the absence of weight terms shown in Table 3 improves convergence/generalization by not constraining the weights to small amplitudes.

the case when ϵ_p is positive. Thus the second order term of the error function will allow a higher raw error ϵ_p to be favored if such an update will place the weights in a more promising area, i.e. a region where weight changes are likely to move S_O^T in a direction to reduce the raw error. The *anticipatory effect* of this term is more important in the beginning stages of training where ϵ_p is large, and will become insignificant in the finishing stages of training as ϵ_p approaches zero.

Similar to arguments in Section 4.1, the absence of weight decay will make the learning task easier and improve convergence (see Table 4).

5 SIMULATION RESULTS

5.1 Experimental Methodology

Two sets of experiments which explore all combinations of cumulative vs. non-cumulative, additive vs. multiplicative, and per sequence vs. per time step noise are reported in this paper. One set of experiments are performed on recurrent networks with 3 state neurons, and the other on recurrent networks with 4 state neurons.

Second order recurrent networks with 3 input neurons are trained on the dual parity problem for all simulations. The finite state automaton (FSA) that recognizes the dual parity grammar is shown in figure 1. The data consists of 8191 strings of lengths 0 to 12. The training set consists of the first 1023 strings of lengths 0 to 9, while the initial working set consists of 31 strings of lengths 0 to 4. Generalization performance of a trained network was obtained by testing on the entire data set of 8191 strings. Training was stopped when all of the training set was correctly classified to the previously discussed error criterion. If training persisted beyond 5000 epochs, training was halted and the trained network was not used. Thus, only networks which were perfectly trained on the training set were tested for generalization. For runs where the noise amplitude was small, the networks usually converged in less than 5000 epochs. For higher noise values, up to a quarter of the runs did not correctly classify the training set in the required number of epochs and were discarded.

As mentioned before the noise source is chosen to be a random, zero-mean white noise process uniformly distributed between (-**a**, **a**), where **a** is the amplitude.

To ensure consistency and fairness across all experiments, the learning rate and momentum were set to 0.5, and the weights were initialized to random values between [-1.0, 1.0] in all simulations. The empirical results presented below represent 90% confidence levels using a 2-tailed Student's t-distribution, and were obtained by running 500 simulations for each data point.

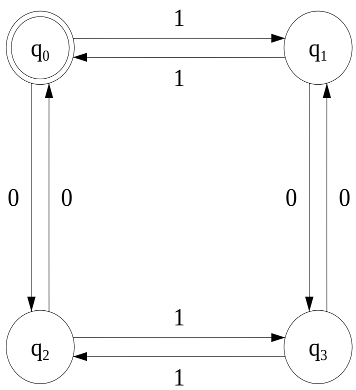

Figure 1: The Dual Parity Finite State Automaton. State 1 is both start and final state.

5.2 Generalization Performance

In figures 2 and 3, we see generalization performance as a function of noise amplitude for both the 3-state and 4-state neuron recurrent nets. For each net we see generalization for various noise combinations. For all simulations the zero-noise case is where $a = 0$. Generalization performance mirrors some of our predictions. However, all curves exhibit the same general shape of generalization performance. The generalization improves as noise increases from the zero-noise point for all noise cases and then starts to degrade after a certain noise-threshold amplitude. After this noise-threshold amplitude, generalization performance eventually becomes worse than the zero-noise case. (The curve in Figure 3a does increase as a increases, but is not shown.) The cumulative per sequence noise cases for both additive and multiplicative noise failed to converge for all runs.

5.3 Convergence Time

Convergence times in epochs as a function of noise amplitude are plotted in figures 4 and 5. As mentioned above, networks injected with cumulative per sequence noise do not converge. For all noise models, convergence for added noise initially decreases up to a noise-threshold and then starts to increase and finally exceeds that for the no-noise case. Similar behavior for feedforward networks has been observed [16]. This is more apparent for the 3-state neuron simulations than the 4-state neuron ones; however, close inspection will show there is a convergence mean less than that for the zero-noise case for the 4-state neuron case.

5.4 Effects of each Noise Parameter on Performance

In order to make fair assessments of the effects of each noise parameter on convergence time and generalization performance, it is reasonable to compare the best mean performance due to each noise parameter. For example, to compare the effects of additive versus multiplicative noise on generalization, we should compare the mean generalization curves of networks injected with additive cumulative per time step noise with the mean generalization curves of networks injected with multiplicative non-cumulative per time step noise. These noises represent the additive and multiplicative noise combinations yielding the best generalization. By looking at the best noise combinations for each parameter, comparisons can be made which are independent of other noise issues. Using the above case as an example, if the best additive noise show better generalization than the best multiplicative noise, then this is a fair indication that additive noise yields better generalization than multiplicative noise. It cannot be argued that perhaps it was because cumulative noise allows better generalization, since if this was the case then we would be comparing additive cumulative per time step noise with multiplicative cumulative per time step noise – which would be fair if such was the case.

Simulated performance closely mirror our predictions. Improvements were observed for all noise models except for cumulative per sequence noises which failed to converge for all runs. Generalization improvement was more emphasized on networks with 4 states, while convergence enhancement was more noticeable on 3-state networks. The simulations show the following results:

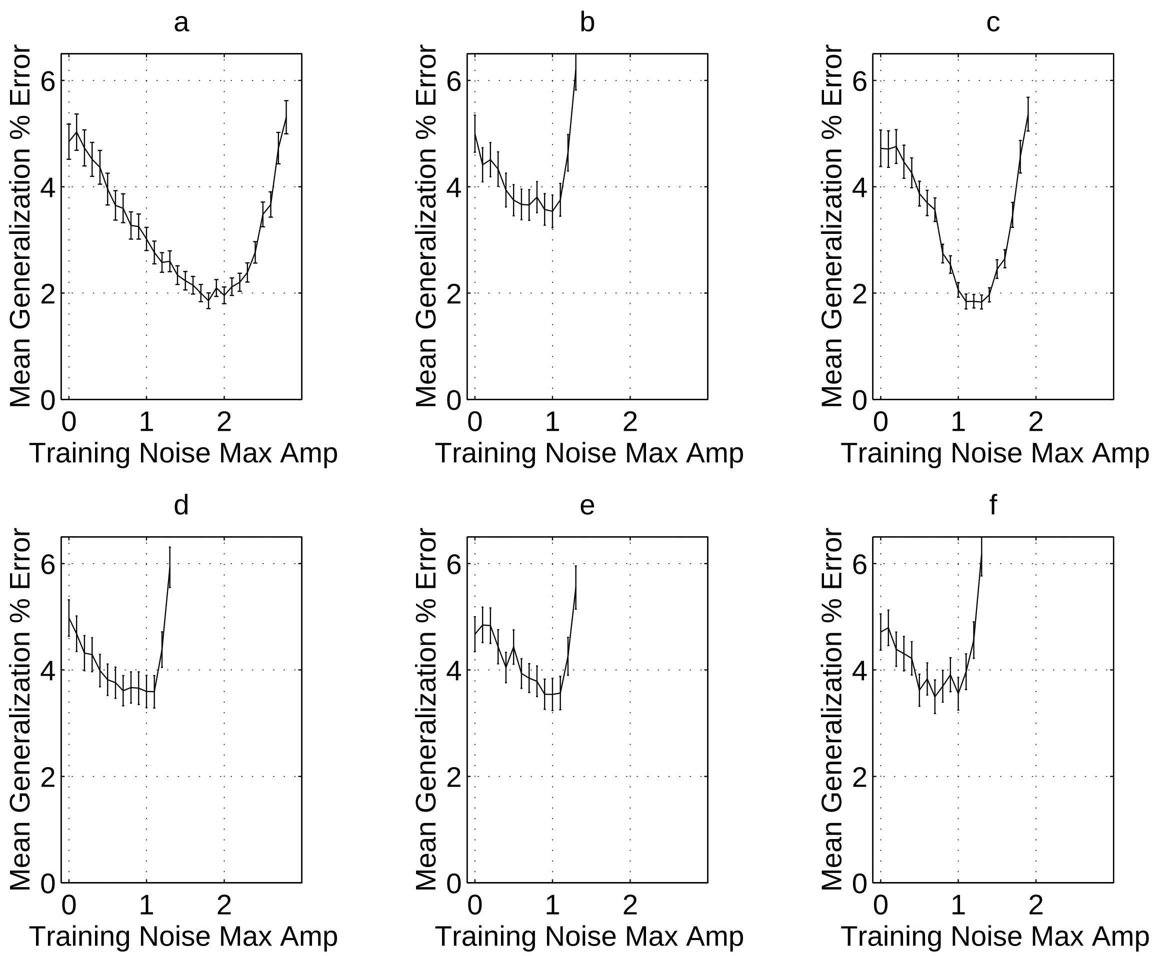

Figure 2: Mean generalization percent error as a function of noise amplitude a for recurrent networks with 4-state neurons. The error bars show 90% confidence intervals, obtained by running 500 simulations for each point. All 3 curves in the top row are for additive noise and all 3 in the bottom row for multiplicative noise. The generalization curves are for (a) non-cumulative additive per time step; (b) non-cumulative additive per sequence; (c) cumulative additive per time step; (d) non-cumulative multiplicative per time step; (e) non-cumulative multiplicative per sequence; (f) cumulative multiplicative per time step. Experiments with cumulative per sequence noise do not converge.

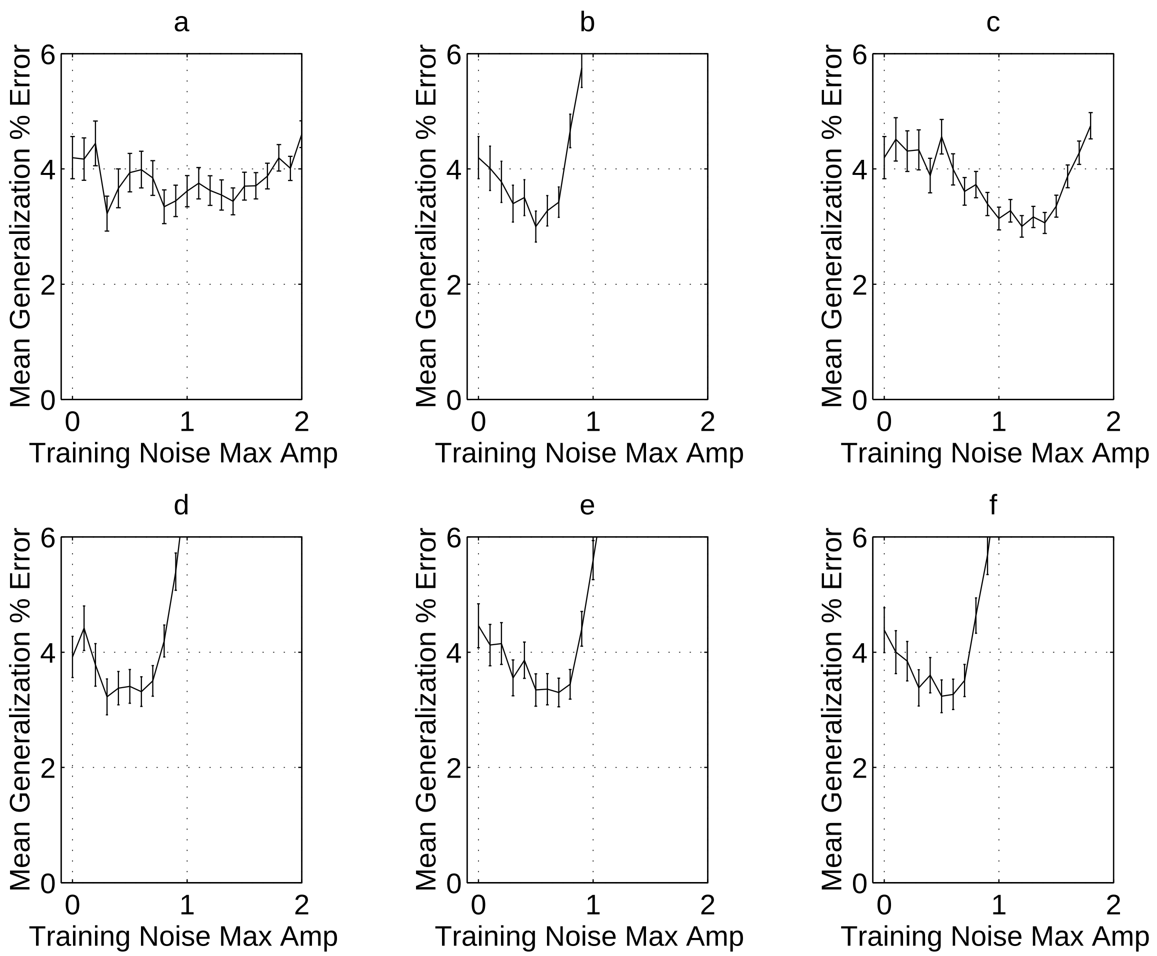

Figure 3: Mean generalization percent error as a function of error magnitude a for recurrent networks with 3-state neurons. The error bars show 90% confidence intervals, obtained by running 500 simulations for each data point. All 3 curves in the top row are for additive noise and all 3 in the bottom row for multiplicative noise. The generalization curves are for (a) non-cumulative additive per time step; (b) non-cumulative additive per sequence; (c) cumulative additive per time step; (d) non-cumulative multiplicative per time step; (e) non-cumulative multiplicative per sequence; (f) cumulative multiplicative per time step. Experiments with cumulative per sequence noise do not converge.

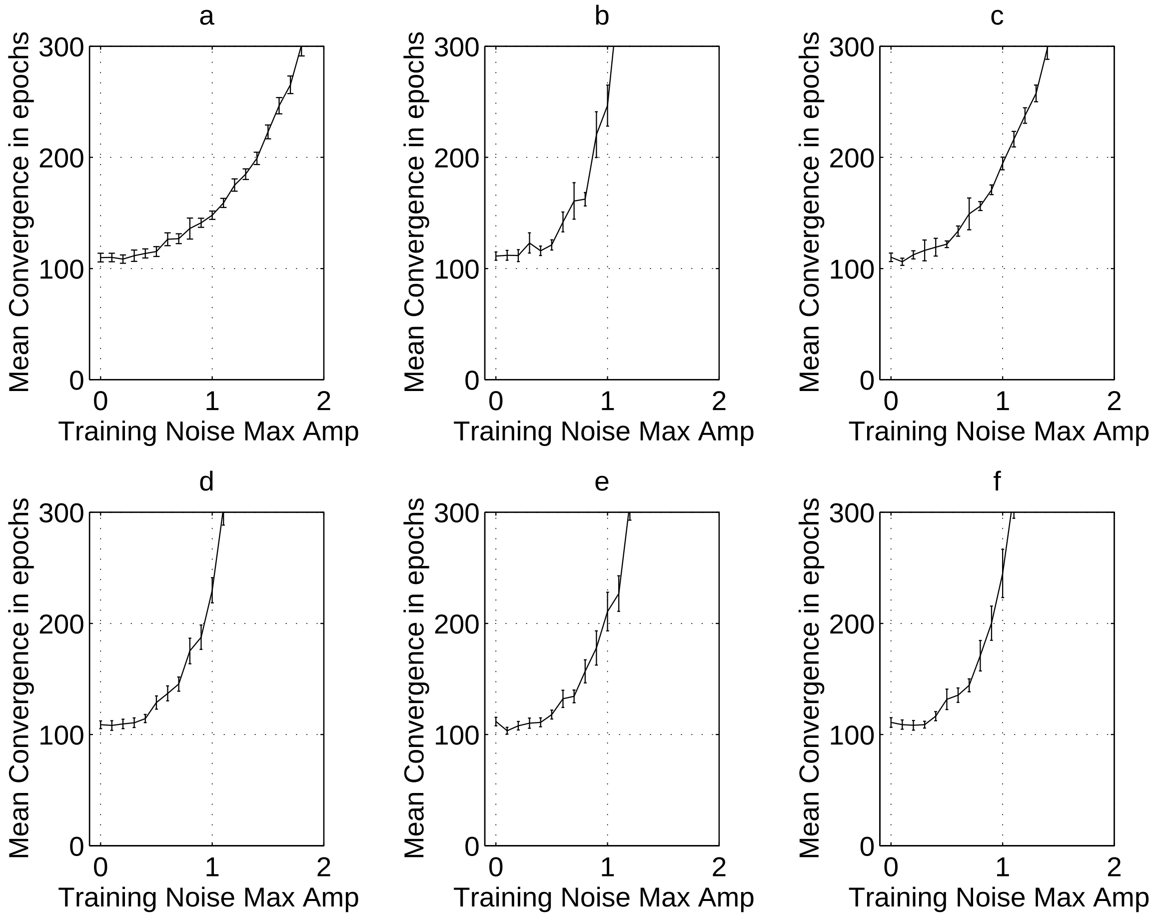

Figure 4: Mean convergence in epochs as a function of noise amplitude a for recurrent networks with 4-state neurons. The error bars show 90% confidence intervals, obtained by running 500 simulations for each data point. The convergence curves are for (a) non-cumulative additive per time step; (b) non-cumulative additive per sequence; (c) cumulative additive per time step; (d) non-cumulative multiplicative per time step; (e) non-cumulative multiplicative per sequence; (f) cumulative multiplicative per time step. Experiments with cumulative per sequence noise do not converge.

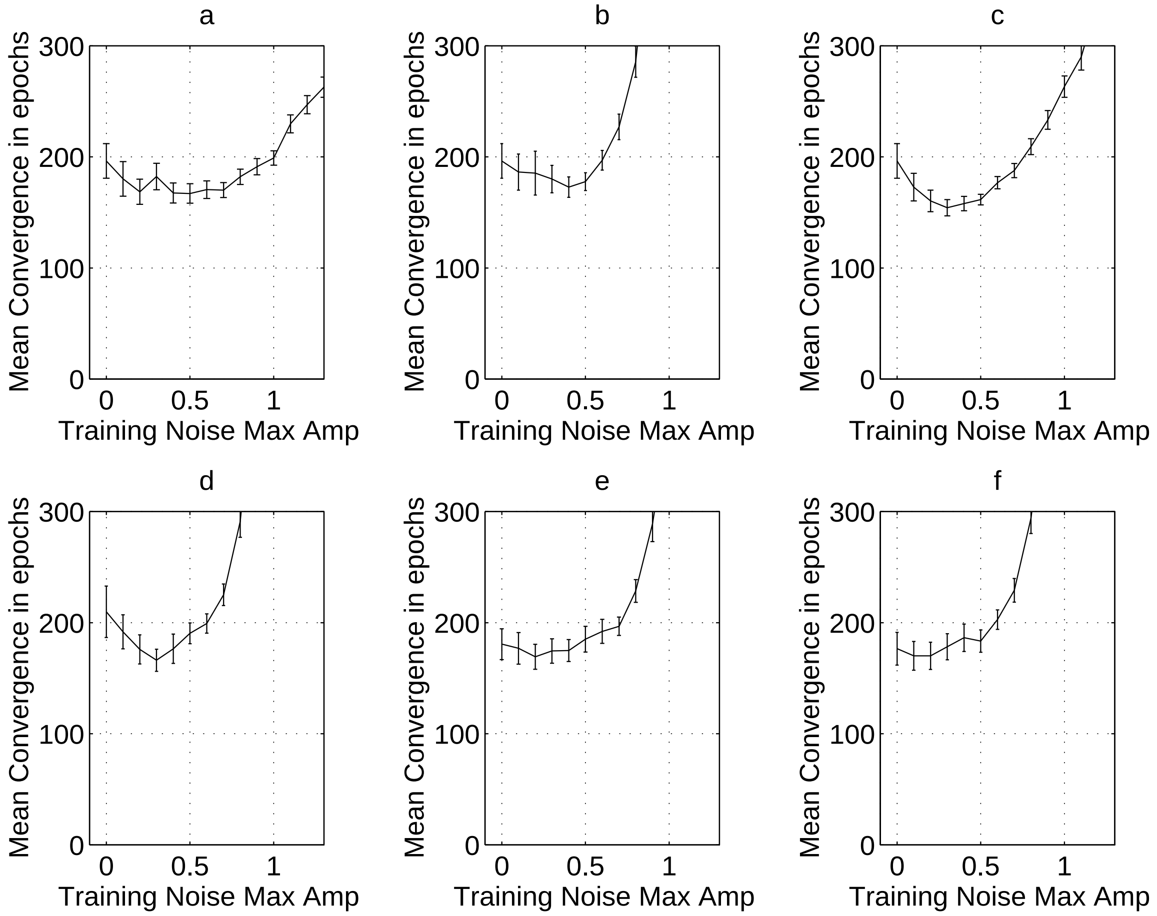

Figure 5: Mean convergence in epochs as a function of noise amplitude a for recurrent networks with 3-state neurons. The error bars show 90% confidence intervals, obtained by running 500 simulations for each data point. The convergence curves are for (a) non-cumulative additive per time step; (b) non-cumulative additive per sequence; (c) cumulative additive per time step; (d) non-cumulative multiplicative per time step; (e) non-cumulative multiplicative per sequence; (f) cumulative multiplicative per time step. Experiments with cumulative per sequence noise do not converge.

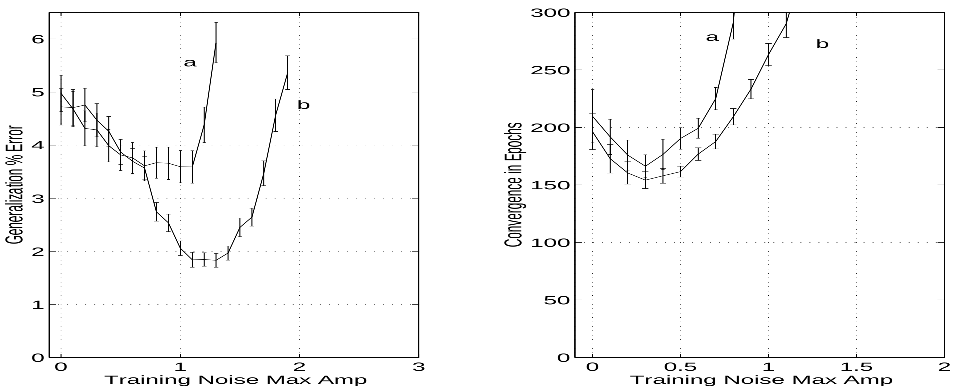

Figure 6: Best Convergence/Generalization for Additive and Multiplicative Noises. (a) multiplicative non-cumulative per time step; (b) additive cumulative per time step.

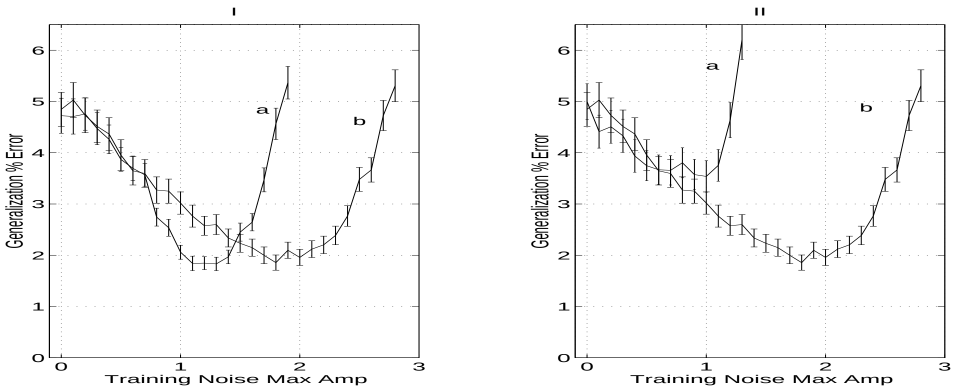

Figure 7: (I) Best Generalization for Cumulative and Non-Cumulative Noises: a) cumulative additive per time step; b) non-cumulative additive per time step. (II) Best Generalization for non-cumulative Per Time Step and Per String Noises: a) per sequence additive; b) per time step additive.

- Additive noise is better tolerated than multiplicative noise, achieves better generalization, and yields slightly better convergence (Figure 6).
- Cumulative noise achieves optimal generalization and convergence at lower amplitudes than non-cumulative noise. Cumulative noise also has a narrower range of beneficial noise, which is defined as the range of noise amplitudes which yields better performance than that of a noiseless network (Figure 7a illustrates this for generalization).
- All non-cumulative per time step noises result in better convergence than non-cumulative per sequence noises on 3–state networks. Non-cumulative additive per time step noise yields better generalization and is better tolerated than non-cumulative additive per sequence noise (Figure 7b).

Overall, the best performance is obtained by applying cumulative and non-cumulative additive noise at each time step. These results closely match the predictions of section 4.1. The only exceptions are that all multiplicative noise models seem to yield equivalent performance. This discrepancy between prediction and simulation may be due to the detrimental effects of weight decay in multiplicative noise, which can conflict with the advantages of cumulative and per time step noise.

5.5 Generalization vs. Convergence

In most applications both generalization and convergence are important; improving one at the expense of the other becomes a difficult decision. In this section we plot the trade-off between generalization and convergence, and show that in many cases it is possible to improve *both* performance measures simultaneously.

In figure 8 we plot generalization vs. convergence. Decreasing noise amplitudes proceed from the right end-point of each curve to the left end-point. For example, if convergence is not an issue than one would pick the curve that dips the lowest, and choose the corresponding noise amplitude for the noise method corresponding to that curve. These plots clearly illustrate that additive noise methods allow the best generalization and convergence for the 4-state neurons networks.

More importantly, these plots illustrate the cases where both convergence and generalization are improved. In figures 8c and 8d the curves clearly curl down and to the left for lower noise amplitudes before rising to the right at higher noise amplitudes. These lower regions are optimal because they represent improved generalization and improved convergence, i.e., by picking a noise amplitude from this "hook" convergence and generalization do not trade-off.

6 CONCLUSIONS

We have presented several methods of injecting synaptic noise to recurrent neural networks. We give a thorough analysis of these methods and empirically test these methods on the learning dual parity grammar

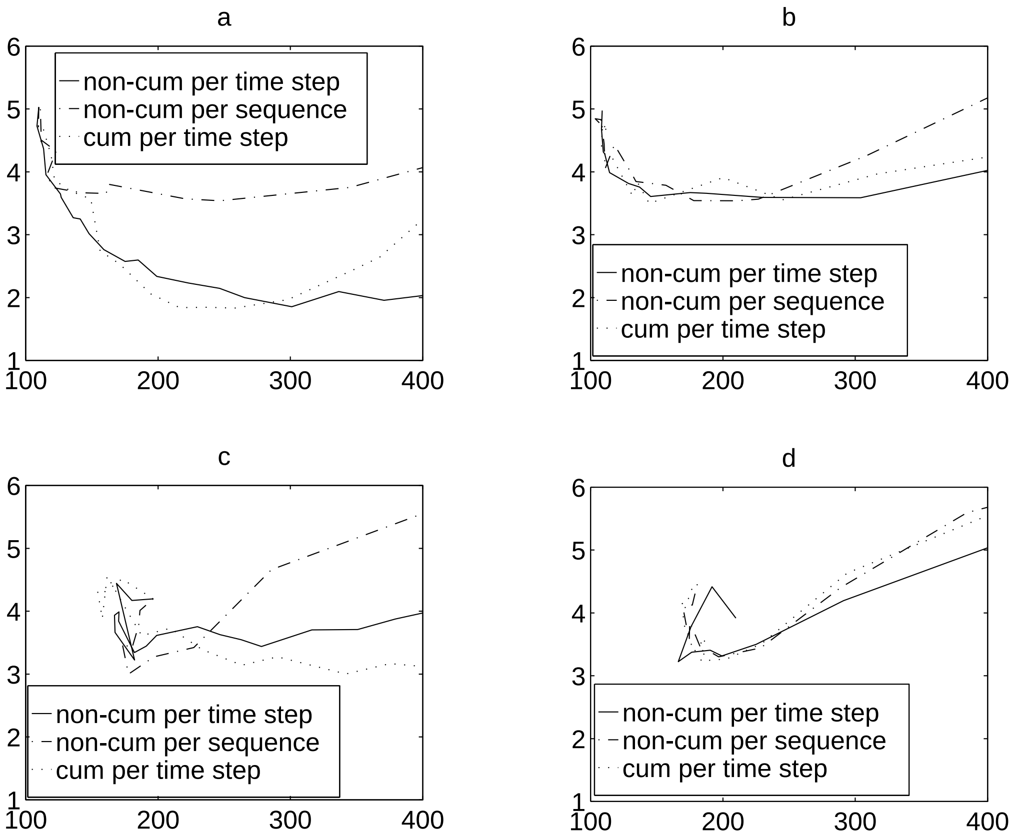

Figure 8: Generalization percent error rate vs. convergence in epochs. The curves are obtained by using the data in previous figures (recall 500 simulations for each data point). (a) Additive noise on 4-state network; (b) Multiplicative noise on 4-state network; (c) Additive noise on 3-state network; (d) Multiplicative noise on 3-state network. Decreasing noise amplitudes increase from right most end-point to the left most end-point of each curve.

from strings encoded as temporal sequences.

Results show that most of these methods offer several benefits on recurrent networks:

- Ability to improve generalization performance.
- Ability to improve convergence performance.
- Ability to improve generalization and convergence simultaneously.

These benefits should apply to other recurrent networks. The ability to improve generalization and convergence simultaneously makes these results novel to other methods in previously discussed for feedforward nets, which in general cast convergence as a cost for improved generalization performance.

These results also offer some insight into what types of synaptic noises are better tolerated when training under noisy environments. If only synaptic noise is applicable, and noise-free updates are allowed, then cumulative multiplicative noise is the most detrimental at high amplitudes. In addition, noise which is uncorrelated in time is less destructive, as illustrated by the fact that injecting a different noise at each time step (for one input string) allows higher detrimental noise thresholds than injecting the same noise at each time step (effectively equivalent to the per sequence noise models). We speculate that these results are appropriate to other problems besides learning grammars from temporal sequences. However, experiments in other problem domains would have to justify this speculation.

It would also be interesting to apply noise insertion to other problem domains to study the effect of problem complexity on the range of beneficial noise. Further research with larger networks could determine the effect of network size on noise tolerance and effectiveness on generalization and convergence. Another important issue is the relationship between the range of optimal noise values to network size and problem complexity. Thus, it could be possible to determine the usefulness of synaptic noise by allowing an optimal range of noise values to be predicted for each application.

7 ACKNOWLEDGEMENTS

We would like to acknowledge Li Chun An for help in the simulations and useful discussions.

APPENDIX

A Series Expansions of the Error Functions

The Taylor series expansion terms of the error functions of all noise models are derived in this section. In all derivations below the noise is assumed to be zero-mean white and uncorrelated in time.

A.1 Non-Cumulative Noise

The multiplicative, per time step noise case is analyzed in detail below, followed by straightforward extensions to other non-cumulative noise cases.

A.1.1 Multiplicative Per Time Step

Each weight $W_{t,ijk}$ is perturbed at each time step such that

$$W_{t,ijk} \longrightarrow W^*_{ijk} + W^*_{ijk}\Delta_{t,ijk} \tag{11}$$

where $\{W^*_{ijk}\}$ is the noise-free weight set. Performing a Taylor series expansion on the output of state neuron S^T_O to second order around the noise-free weight set yields

$$\begin{aligned} S^T_O \quad \longrightarrow \quad & S^T_O + \sum_{t=0}^{T-1}\sum_{ijk} W_{t,ijk}\Delta_{t,ijk}\left(\frac{\partial S^T_O}{\partial W_{t,ijk}}\right) + \\ & \frac{1}{2}\sum_{t,u=0}^{T-1}\sum_{ijk,lmn} W_{t,ijk}\Delta_{t,ijk}W_{u,lmn}\Delta_{u,lmn}\left(\frac{\partial^2 S^T_O}{\partial W_{t,ijk}\partial W_{u,lmn}}\right) + O(>3) \end{aligned} \tag{12}$$

The error E_p for pattern p on a noiseless network is defined as

$$E_p = \frac{1}{2}\epsilon_p^2 \tag{13}$$

where ϵ_p is

$$\epsilon_p = S^T_O - d_p. \tag{14}$$

However, the injection of noise represented by equation (12) augments E_p as follows

$$\begin{aligned} E_p \quad = \quad & \frac{1}{2}[S^T_O + \sum_{t=0}^{T-1}\sum_{ijk} W_{t,ijk}\Delta_{t,ijk}\left(\frac{\partial S^T_O}{\partial W_{t,ijk}}\right) + \\ & \frac{1}{2}\sum_{t,u=0}^{T-1}\sum_{ijk,lmn} W_{t,ijk}\Delta_{t,ijk}W_{u,lmn}\Delta_{u,lmn}\left(\frac{\partial^2 S^T_O}{\partial W_{t,ijk}\partial W_{u,lmn}}\right) + \\ & O(>3) - d_p]^2. \end{aligned} \tag{15}$$

Dropping Δ^3 and higher order terms,

$$\begin{aligned} E_p \quad = \quad & \frac{1}{2}\Bigg[\epsilon_p^2 + \sum_{t,u=0}^{T-1}\sum_{ijk,lmn} W_{t,ijk}\Delta_{t,ijk}W_{u,lmn}\Delta_{u,lmn}\left(\frac{\partial S^T_O}{\partial W_{t,ijk}}\right)\left(\frac{\partial S^T_O}{\partial W_{u,lmn}}\right) + \\ & 2\epsilon_p^2\sum_{t=0}^{T-1}\sum_{ijk} W_{t,ijk}\Delta_{t,ijk}\left(\frac{\partial S^T_O}{\partial W_{t,ijk}}\right) + \\ & \epsilon_p\sum_{t,u=0}^{T-1}\sum_{ijk,lmn} W_{t,ijk}\Delta_{t,ijk}W_{u,lmn}\Delta_{u,lmn}\left(\frac{\partial^2 S^T_O}{\partial W_{t,ijk}\partial W_{u,lmn}}\right)\Bigg]. \end{aligned} \tag{16}$$

Since the noise is zero-mean white and uncorrelated, taking the time average over the training phase for the per time step case yields

$$
\begin{aligned}
< \Delta_{t,ijk} > &= 0 & (17)\\
< \Delta_{t,ijk}\Delta_{u,lmn} > &= \sigma^2 \delta_{tijk,ulmn} & (18)
\end{aligned}
$$

where $\delta_{tijk,ulmn}$ is the Kronecker delta. Using equations (17)(18) simplifies (16) to

$$
< E_p^{noisy} > = < E_p > + \frac{1}{2} \sum_{t=0}^{T-1} \sum_{ijk} (W_{t,ijk}\sigma)^2 \left[\left(\frac{\partial S_O^T}{\partial W_{t,ijk}} \right)^2 + \epsilon_p \left(\frac{\partial^2 S_O^T}{\partial (W_{t,ijk})^2} \right) \right]. \tag{19}
$$

It is important to realize that although each "layer" has the same set of weights,

$$
W_{0,ijk} = W_{1,ijk} = \cdots = W_{T-1,ijk} \tag{20}
$$

nevertheless their respective partial derivatives are not equal because they are time dependent:

$$
\frac{\partial S_O^T}{\partial W_{0,ijk}} \neq \frac{\partial S_O^T}{\partial W_{1,ijk}} \neq \cdots \neq \frac{\partial S_O^T}{\partial W_{T-1,ijk}}. \tag{21}
$$

Therefore all partial derivatives $\sum_{t=0}^{T-1} \sum_{ijk} [(\frac{\partial S_O^T}{\partial W_{t,ijk}})^2 + (\frac{\partial^2 S_O^T}{\partial (W_{t,ijk})^2})]$ need to be accounted for in equation (19).

A.1.2 Multiplicative Per Sequence

To extend the above derivation to the per sequence case, note that $\Delta_{t,ijk} = \Delta_{ijk}$ for all time steps within each sequence in eq. (11). Therefore,

$$
\begin{aligned}
< \Delta_{t,ijk} > &= 0 \\
< \Delta_{t,ijk}\Delta_{u,lmn} > &= \sigma^2 \delta_{ijk,lmn} & (22)
\end{aligned}
$$

and eq. (19) becomes

$$
< E_p^{noisy} > = < E_p > + \frac{1}{2} \sum_{t,u=0}^{T-1} \sum_{ijk} (W_{t,ijk} W_{u,ijk} \sigma^2) \left[\left(\frac{\partial S_O^T}{\partial W_{t,ijk}} \right) \left(\frac{\partial S_O^T}{\partial W_{u,ijk}} \right) + \epsilon_p \left(\frac{\partial^2 S_O^T}{\partial W_{t,ijk} \partial W_{u,ijk}} \right) \right]. \tag{23}
$$

A.1.3 Additive Per Time Step and Per Sequence

The above per sequence and per time step derivations can be easily extended to the additive noise models by substituting

$$
W_{t,ijk} \longrightarrow W_{ijk}^* + \Delta_{t,ijk}. \tag{24}
$$

in place of eq. (11).

A.2 Cumulative Noise

The cumulative noise models are analyzed below. They are based on and described in the same format as the above section on non-cumulative noise.

A.2.1 Multiplicative Per Time Step

For the multiplicative, per time step case, each noise injection step is such that

$$
W_{t,ijk} = W_{ijk}^* \prod_{\tau=0}^{t} (1 + \Delta_{\tau,ijk}) \tag{25}
$$

To put eq. (25) in a similar format for the error expansion, note that it can be represented by

$$W_{t,ijk} = W^*_{ijk} + W^*_{ijk}\Phi_{t,ijk} \tag{26}$$

where

$$\Phi_{t,ijk} = \prod_{\tau=0}^{t}(1+\Delta_{\tau,ijk}) - 1. \tag{27}$$

The time averages

$$< \Phi_{t,ijk} > \;=\; 0 \tag{28}$$

$$< \Phi_{t,ijk}\Phi_{u,lmn} > \;=\; v\sigma^2\delta_{ijk,lmn} \tag{29}$$

where $v = min(t+1, u+1)$ can be used to simplify the error expression.

A.2.2 Multiplicative Per Sequence

In the per sequence case, the noise injection step $W_{t,ijk} = W^*_{ijk}(1+\Delta_{ijk})^{t+1}$ can be represented by

$$W_{t,ijk} = W^*_{ijk} + W^*_{ijk}\Psi_{t,ijk} \tag{30}$$

where

$$\Psi_{t,ijk} = \sum_{\tau=0}^{t} c_{t,\tau}(\Delta_{ijk})^{\tau+1} \tag{31}$$

and $\{c_{t,\tau}\}$ is a different set of positive constants for each t. The time averages do not simplify the error function.

A.2.3 Additive Per Time Step

In the cumulative additive per time step case, each noise injection step is such that

$$W_{t,ijk} = W^*_{ijk} + \sum_{\tau=0}^{t}\Delta_{\tau,ijk}. \tag{32}$$

The time averages

$$< \sum_{\tau=0}^{t}\Delta_{\tau,ijk} > \;=\; 0 \tag{33}$$

$$< \sum_{\tau=0}^{t}\Delta_{\tau,ijk}\sum_{\gamma=0}^{u}\Delta_{\gamma,lmn} > \;=\; v\sigma^2\delta_{ijk,lmn} \tag{34}$$

where $v = min(t+1, u+1)$ can be used to simplify the error expression.

A.2.4 Additive Per Sequence

The noise injection step for the additive per sequence case can be modeled by

$$W_{t,ijk} = W^*_{ijk} + (t+1)\Delta_{ijk} \tag{35}$$

The time averages

$$< (t+1)\Delta_{ijk} > \;=\; 0 \tag{36}$$

$$< (t+1)\Delta_{ijk}(u+1)\Delta_{lmn} > \;=\; (t+1)(u+1)\sigma^2\delta_{ijk,lmn} \tag{37}$$

can be used to simplify the error expression.

For all cases the weight update ΔW_{ijk} descends the noise-enhanced error function $< E_p^{noisy} >$.

References

[1] E.B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[2] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. In S.J Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 244–251, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[3] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–6053. Morgan Kaufmann Publishers, 1990.

[4] A. Dembo and T. Kailath. Model-free distributed learning. *IEEE Transactions on Neural Networks*, 1:58–70, 1990.

[5] Barry Flower and Marwan Jabri. Summed weight neuron perturbation: An O(n) improvement over weight perturbation. In S.J Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 212–219, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[6] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, and Y.C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[7] C.L. Giles and C.W. Omlin. Pruning recurrent neural networks for improved generalization performance. *IEEE Transactions on Neural Networks*, 1994. Accepted for Publication.

[8] Geoffrey E. Hinton, Terrence J. Sejnowski, and David H. Ackley. Boltzmann machines: Constraint satisfaction networks that learn. Technical Report CMU-CS-84-119, Carnegie–Mellon University CS Dept., May 1984.

[9] Marwan Jabri and Barry Flower. Weight perturbation: An optimal architecture and learning technique for analog vlsi feedforward and recurrent multilayer networks. *IEEE Transactions on Neural Networks*, pages 154–157, 1992.

[10] Stephen Judd and Paul W. Munro. Nets with unreliable hidden nodes learn error-correcting codes. In S.J Hanson, J.D. Cowan, and C.L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 89–96, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[11] Danny Kilis and Eugene Ackerman. A stochastic neuron model for pattern recognition. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, Washington, D.C., January 1990.

[12] S. Kirkpatrick, Jr. C.D. Galett, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, May 1983.

[13] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 450–957, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[14] C.B. Miller and C.L. Giles. Experimental comparison of the effect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artifical Intelligence*, 7(4):849–872, 1993. Special Issue on Neural Networks and Pattern Recognition, editors: I. Guyon , P.S.P. Wang.

[15] Jay. I. Minnix. Fault tolerance of the backpropagation neural network trained on noisy inputs. In *Proc. of IJCNN*, volume I, pages I–847–852, 1992.

[16] Alan F. Murray and Peter J. Edwards. Synaptic weight noise during multilayer perceptron training: Fault tolerance and training improvements. *IEEE Trans. on Neural Networks*, 4(4):722–725, 1993.

[17] G.V. Puskorius and L.A. Feldkamp. Neurocontrol of nonlinear dynamical systems with kalman filter-trained recurrent networks. *IEEE Transactions on Neural Networks*, 5(2):279–297, 1994.

[18] Carlo H. Séquin and Reed D. Clay. Fault tolerance in artificial neural networks. In *Proc. of IJCNN*, volume I, pages I–703–708, 1990.

[19] R.L. Watrous and G.M. Kuhn. Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4(3):406, 1992.

[20] Ronald J. Williams and David Zipser. Gradient–based learning algorithms for recurrent connectionist networks. Technical Report NU–CCS–90–9, College of Computer Science, Northeastern University, 1990.

[21] Z. Zeng, R.M. Goodman, and P. Smyth. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5(6):976–990, 1993.