Speeding Up the Generalized Adaptive Neural Filters

Henry Hanek and Nirwan Ansari, Senior Member, IEEE

Abstract—Recently, a new class of adaptive filters called generalized adaptive neural filters (GANF's) emerged. They share many things in common with stack filters and include all stack filters as a subset. The GANF's allow a very efficient hardware implementation once they are trained. However, the training process can be slow. This paper discusses structural modifications to allow for faster training. In addition, these modifications can lead to an increase in the filter’s robustness, given a limited amount of training data. This paper does not attempt to justify use of a GANF; it only presents an alternative implementation of the filter.

I. BACKGROUND

Ansari et al. [1] and Yin et al. [2] have recently proposed to employ the gradient descent algorithm to configure stack filters and their variants. These filters belong to a class of nonlinear filters known as generalized adaptive neural filters (GANF’s) [3]. GANF’s are based on stack filters [4], and consequently, they share some similarities with them. The GANF’s, though, have a more generalized structure that allows the implementation of a broader range of filters. All stack filters and generalized stack filters form a subset of the class of GANF’s [5]. A diagram of a simple GANF is shown in Fig. 1. Here, we window an integer input sequence and form a vector

$$r_B(n) = [r_1(n) \ r_2(n) \ \cdots \ r_B(n)]$$

(1)

where $B$ is the window size and $r_k(n) \in \{0, 1, \ldots, M-1\}$ are the elements in the filter’s window at time $n$. We then threshold-decompose this vector on $(M-1)$ levels $i$ according to

$$x_i^B(n) = T^i[r_B(n)]$$

$$= [T^i[r_1(n)] \ T^i[r_2(n)] \ \cdots \ T^i[r_B(n)]]$$

(2)

with

$$T^i[x] \triangleq \begin{cases} 1, & \text{if } x \geq i \\ 0, & \text{otherwise.} \end{cases}$$

(3)

This is an ordered threshold decomposition operation that uniquely creates $(M-1)$ binary vectors from the integer window vector. These binary vectors are then used as inputs to independent neural operators $N_i[\cdot]$ on each level of the filter. In general, a neural operator can be any function that maps a binary input vector to a continuous output in the range $[0, 1]$. Examples of neural operator types and the resulting filter properties can be found in [6]. In the GANF, each neural operator receives as input the vector on its own level, along with adjacent vectors on levels above and below. In other words, $N_i[\cdot]$ will process a binary input matrix defined by

$$X_{i,B} = \begin{bmatrix} T^{i+1}[r_B(n)] \\ \vdots \\ T^i[r_B(n)] \\ \vdots \\ T^i-1[r_B(n)] \end{bmatrix}$$

(4)

The dimension of this matrix is $(2I+1) \times B$, where $I$ equals the number of adjacent levels fed in. Based on these input matrices, the neural operators generate outputs that are added to produce the total filter output at time $n$. Thus, the GANF output can be described by

$$y(n) = F_{i,B}[r_B(n)] = \sum_{i=1}^{M-1} N_i[X_{i,B}(n)].$$

(5)

For simplification, we will restrict our attention to neural operators that produce binary outputs $\in \{0, 1\}$ only. With this restriction, we can more readily relate the GANF to other similar filters. If all of the neural operators $N_i[\cdot]$ possess the stacking property for all possible input matrices produced by the threshold decomposition, the GANF can implement any generalized stack filter [7]. The GANF will further reduce to a regular stack filter if both of the following conditions hold: 1) No adjacent-level inputs are used, and 2) the neural functions are all identical positive Boolean functions. It has been shown [3] that the optimal GANF can perform better, or at least as good as, an optimal stack filter. The GANF is optimized by training the neural operators to minimize the classification error on their respective levels. This procedure, which has the effect of minimizing the filter’s MAE [5], can be accomplished by using a neural net training algorithm.

Fig. 1. GANF with eight levels and window size of 3.
In order to achieve good performance, the filter must be trained on a large number of samples. The exact number depends on the data, the VCdim [8] of the network used, and the training scheme employed. In general, increasing the size of the training set will produce better results. However, as the length of the training set is increased, the training time increases proportionately. Excessive training times can prevent the filter’s use in practical, real-world problems. Therefore, in order to train the GANF on enough samples and minimize the training time, the training time per sample must be minimized.

It became clear through simulations that filter training times were very long. Considering the complexity of the filter, this is understandable. With an input vector of 8-bit precision, there are 255 levels of neural operators in the GANF. If these levels are independently trained on a data set of length 16 384, there would be a total of 4.2 million training operations. Even at a 1 ms/level update, it would take 1.16 hr to train the filter. In addition, there are massive memory requirements necessary for implementing many of the networks. Once the networks are trained, however, VLSI implementation would allow very fast operation. In fact, while other filter types such as [9] avoid the threshold decomposition architecture, it is this structure that allows an efficient VLSI construction of the filter [4]. As a result, most of the need for a speed increase is focused on the training. Of course, the improved training structure could also be applied to the filtering if microprocessor implementation (an algorithm) is chosen over VLSI.

II. SIMPLIFIED STACK FILTERS AND THE ATD ARCHITECTURE

Before we discuss how to speed up the GANF, we will first mention another class of filters that achieves a similar goal. This class (the adaptive threshold decomposition (ATD) filters) was created by Lin et al. [10] to decrease the time required for stack filter [4] training. Since GANF S are based on stack filters, the framework of ATD filters can be applied to GANF’s. Note that in a stack filter, GANF, or any filter that threshold decomposes its input vector, there are a total of $M-1$ binary vectors produced by the threshold decomposition. However, there are at most only $(B+1)$ different binary vectors, where $B$ is the window size. Since each Boolean function in a stack filter is the same, there are at most $(B+1)$ unique outputs. Making use of these ideas, the ATD filters can be described as follows: First, consider an integer input vector in the form of (1). As previously shown, the threshold decomposition operation produces $(M-1)$ binary vectors of length $B$

$$x_B(n) = T^{i} r_B(n).$$

Now, let us define $R_{ik}(n)$, $k \in \{1, 2, \ldots, B\}$ to be the $k$th smallest sample in the window at time $n$. Then, let

$$u_k(n) = x_B^{R_{ik}(n)}(n)$$

where $x_B^{R_{ik}(n)}(n)$ is the binary vector resulting from the threshold decomposition on level $R_{ik}(n)$. Next, let us define $\Delta_k$ as the difference between samples of rank $k$ and $k-1$

$$\Delta_k(n) \triangleq R_{ik}(n) - R_{ik-1}(n).$$

Note here that the window is as defined in (1), and $R_{ik}(n)$ is always assigned the value of zero. We can then represent the ATD filter by

$$y(n) = S(r_B(n)) = \sum_{k=1}^{B} f_k[u_k(n)] \Delta_k(n)$$

where $f_k(\cdot)$ can be a Boolean function. These ATD filters are very general, and by using neural operators to implement the functions $f_k(\cdot)$, we can implement a modified GANF in this form. It is important to note, however, that this type of ATD will, in general, not be exactly equivalent to a standard GANF. Here, we use only $(B+1)$ neural functions in place of the $(M-1)$ neural operators in the GANF. The neural operators in the ATD filter are not assigned to specific levels as they are in the GANF. As discussed later, the ATD filter can achieve better generalization performance than the standard GANF, making it more useful for some applications. Therefore, in certain cases, it may perform better. However, to attain the increased classification abilities of the GANF at a speed comparable with the ATD filter, another structure is proposed in the next section.

III. THE FAST-GANF

To speed up a GANF, a two-step procedure can be used to first decrease the number of independent neural operators and then develop a simplified structure based on this. Unlike the ATD filters, this new simplified structure can very nearly approximate the operation of a standard GANF.

We will first look at combining neural functions in the GANF in terms of increasing the filtering speed. Later on, we will discuss how this can also lead to an increase in classifier generalization. The idea here is to use the same neural operator to process information on a number of adjacent levels. Recall that the previously discussed GANF had the capability of implementing different Boolean operations on each of the levels. We will refer to this as a nonhomogeneous GANF. With level combinations, only certain groups of levels are processed with independent neural operators. In other words, we will reuse the same Boolean function for a specific range of levels. The limiting case for this is the use of the same neural operator for all of the levels, which is called a homogeneous GANF.

In general, we wish to combine levels, but at the same time maintain the best possible filtering performance. As a result, levels cannot be combined randomly. Note, however, that if two neural operators produce the same output for all given inputs, then they are functionally identical. Based on this idea, we can develop a measure for neural operator similarity. It would be a type of correlation between neural responses for a given input set. This idea is summarized as Proposition 1.

**Proposition 1:** Suppose we are given two neural operators on different levels in the GANF. If the same inputs applied to the neural operators produce the same output, the neural functions are consistent with each other. If this is the case for all inputs in the input set, then we can consider the two neural functions to be identical. If not, we can use the likelihood of identical neural responses to measure their similarity. If the
two functions are judged to be similar enough, we can replace them with one function.

This proposition seems appropriate since the neural operators we consider can be represented as functions that map elements of an input set to elements in an output set. For example, a binary neural operator is defined by its responses to the binary input vectors comprising its domain. The same applies to a nonbinary neural operator in general, but the likelihood measure would have to be changed to reflect some type of mean distance measure between neural responses.

For now, we will restrict our attention to the case where no adjacent-level inputs are used (I = 0). We can then define this measure of similarity between neural functions on levels l and j as

\[
g(l, j) = P\{\text{same outputs on levels } l \text{ and } j \mid \text{same inputs on levels } l \text{ and } j\} \tag{10}
\]

where \( l \) is a level number \( \in \{0, 1, \ldots, M - 1\} \), \( j \) is a different level number \( \in \{0, 1, \ldots, M - 1\} \), and the binary input vectors on the two levels are the same. To compute the best estimate of \( g(l, j) \), we would have to keep track of all of the times that \( x_B^i(n) \) equals \( x_B^j(n) \), even if \( n \neq m \). This would require a lot of effort. In addition, we do not want the simplification method to make things more complicated than if it was not used at all. Therefore, we will estimate \( g(l, j) \) by considering function similarities at the same time instants. Next, we can assume, without loss of generality, that \( j > l \). Then, if \( s(n) \) is the desired output signal

\[
g(l, j) = P\{s(n) < l \mid x_B^i(n) = x_B^j(n)\} + P\{s(n) \geq j \mid x_B^i(n) = x_B^j(n)\} \tag{11}
\]

or

\[
g(l, j) = \frac{P\{s(n) < l, x_B^i(n) = x_B^j(n)\}}{P\{x_B^i(n) = x_B^j(n)\}} + \frac{P\{s(n) \geq j, x_B^i(n) = x_B^j(n)\}}{P\{x_B^i(n) = x_B^j(n)\}}. \tag{12}
\]

Since we are considering only single instants of time for the computation of \( g(l, j) \), we note the following.

**Observation 1:** The two binary input vectors on levels \( l \) and \( j \) are equal at time \( n \) if there exists no integer element \( r_k(n) \) in the filter window such that \( l \leq r_k(n) < j \).

Then, to compute a relative frequency estimate of \( g(l, j) \), we need three counter variables, as shown in the following:

\[
g(l, j) \approx \frac{\sum_{n=0}^{N-1} I_1(l, j, n) + \sum_{n=0}^{N-1} I_2(l, j, n)}{\sum_{n=0}^{N-1} I_3(l, j, n)} \tag{13}
\]

where

\[
I_1(l, j, n) = \begin{cases} 1, & \text{if } s(n) < l \text{ and } \beta l \leq r_k(n) < j, \\ 0, & \text{otherwise} \end{cases} \tag{14}
\]

\[
I_2(l, j, n) = \begin{cases} 1, & \text{if } s(n) \geq j \text{ and } \beta l \leq r_k(n) < j, \\ 0, & \text{otherwise} \end{cases} \tag{15}
\]

and

\[
I_3(l, j, n) = \begin{cases} 1, & \text{if } \beta l \leq r_k(n) < j, \\ 0, & \text{otherwise}. \end{cases} \tag{16}
\]

In this equation, \( N \) is the length of the training set, and \( r_k(n) \) is an element of the window input vector. Note that to compute this measure, we really only need two counter variables, as \( I_1(l, j, n) \) and \( I_2(l, j, n) \) can be combined into one. Once \( g(l, j) \) is computed, we can set a threshold \( \beta \), where \( 0 \leq \beta < 1 \). If \( g(l, j) \geq \beta \) and the denominator of (13) is above a certain threshold, then we use the same neural operator for levels \( l \) and \( j \). At present, there has been no investigation into the setting of \( \beta \); it is purely heuristic. In addition, placing a threshold on the denominator of (13) will assure that enough similar inputs were considered to make the estimate (13) meaningful. A large \( \beta \) should provide the best performance for given signal and noise distributions, although it could result in a complicated filter. A small \( \beta \) will provide a simpler filter but will usually decrease the filter’s performance.

We can now make use of the decreased number of neural operators and create a FAST structure for the GANF. This FAST-GANF will be nearly identical in operation to the non-FAST setup, which may be desirable for many signal types. Unlike the ATD filter, the standard GANF allows different classification functions on different levels, as they are required by the signal statistics. The FAST-GANF tries to maintain this advantage by retaining independent functions on the levels for which they are needed. (Recall that the level functions in an ATD filter process inputs based on relative amplitudes only.)

As discussed previously, the number of neural operators in a FAST-GANF is determined by the parameter \( \beta \). Consider for a moment the standard GANF structure that happens to have identical neural functions on certain groups of its levels. In this case, each different neural operator \( N_i[\cdot] \) can be assigned a range of operation (in terms of levels \( a_i \) and \( b_i \)). In other words, \( N_i[\cdot] \) will process binary input vectors on all levels between \( a_i \) and \( b_i \), inclusively. The FAST-GANF implements the same operation but eliminates redundancy in the binary vectors. The integer input vector is threshold-decomposed only on levels that have meaning. This both increases the efficiency of the threshold decomposition operation and saves on neural operations that are not needed. In general, there will be at most \( B + K \) decompositions and neural outputs, where \( B \) is the window size, and \( K \) is the number of independent neural operators. The filter can be described in detail as follows:

We are given a GANF with \( M - 1 \) levels, a window size \( B \), no adjacent levels fed in, and \( K \) neural operators \( N_i[\cdot] \). These neural functions provide independent outputs for each group of input vectors \( x_B^i(n) \), where \( a_i \leq i \leq b_i \). In other words, we use the same neural function \( N_i[\cdot] \) to provide outputs for the input vectors on levels \( a_i \) through \( b_i \). The values of \( a_i \) and \( b_i \) are determined by the method previously discussed (using (13)). To make a FAST structure out of this, we form a set

\[ X(n) = \{ r_1(n) \cup \cdots \cup r_B(n) \cup b_1 \cup \cdots \cup b_K \} \tag{17} \]

where \( r_i(n) \) are the integer window inputs, and \( b_i \) are the greatest levels processed by the respective neural operator. Basically, this set contains the filter levels where there is a
transition in binary vectors or neural operators. Since \( b_i \neq b_j \) when \( i \neq j \), there will be anywhere from \( K \) elements to \( K + B \) elements in set \( X(n) \).

The output of this FAST-GANF can now be described by

\[
y(n) = \sum_{j=1}^{K} \left[ \sum_{i=1}^{K} N_j[u_i(n)] \right] \Delta_i(n) f(i,j,n) \tag{18}
\]

where

\[
f(i,j,n) = \begin{cases} 1, & \text{if } a_j \leq R(i)(n) \leq b_j \\ 0, & \text{otherwise} \end{cases}
\tag{19}
\]

\[
u_i(n) = T_{R(i)(n)}[B(n)]
\tag{20}
\]

\[
\Delta_i(n) = R(i)(n) - R(i-1)(n)
\tag{21}
\]

and \( R(i)(n), i \in \{1, 2, \ldots, |X(n)|\} \) is the \( i \)-th smallest sample in the set \( X(n) \), where \( N_i[] \) is the neural operator \( i \leq j \leq K \).

To more clearly illustrate the operation of the FAST-GANF, consider the example shown in Fig. 2. Here, we assume that levels were combined such that one neuron is active on levels 1 through 4, and another takes care of levels 5 through 7. In a standard GANF, the input would be threshold-decomposed on all seven levels. Then, the lower neural function would generate four outputs for the inputs on levels 1 through 4, and the upper one would generate three outputs. All of these outputs would be added to produce the filter output of 3 levels. The FAST-GANF, on the other hand, would only threshold-decompose the input on levels 1, 3, 4, 5, and 7. Here, the elements of set (17) are \( r_1(n) = 3, r_2(n) = 5, r_3(n) = 1, b_1 = 4, \) and \( b_7 = 7 \). An output of 0 would be generated on level 4, and outputs of 1 would be present for levels 1 and 3. Then, the upper neuron would generate separate outputs on levels 5 and 7. To produce the final output, the neural outputs for levels 1, 3, 4, 5, and 7 would be multiplied by 1, 2, 1, and 2, respectively, and then added. These multiplication values come from (21). Note that although this example used binary neural operators, the FAST-GANF is equally valid in cases where neural outputs take on a continuum of values.

In most cases, the FAST structure will train at a much greater speed than the standard GANF because the multiplication operations are quicker than generating outputs from the neurons. Note that the FAST-GANF is similar to an ATD filter, except for \( f(i,j,n) \) and a different set on which the ranks are based. However, the FAST-GANF approximates operation of the standard GANF much more closely than does the ATD filter.

**Theorem 1:** If the optimal neural functions on certain groups of levels are the same within each group, then the standard GANF is identical to the FAST-GANF for the case with no adjacent levels fed in.

**Proof:** For a standard GANF with no adjacent levels fed in

\[
y(n) = \sum_{i=1}^{M-1} N_i[x^i_B(n)].
\tag{22}
\]

Now, if \( N_i[] \) \( N_i[] \) \( \forall a_i \leq i \leq b_i \), then

\[
y(n) = \sum_{i=m}^{b_i} \sum_{i=m+1}^{b_i} N_i[x^i_B(n)]
\tag{23}
\]

where \( N_i[] \) represents the neural operation \( N_i[] \) \( \forall a_i \leq i \leq b_i \). Let us consider each term of the form

\[
\sum_{i=m}^{b_i} N_i[x^i_B(n)].
\tag{24}
\]

Due to the threshold decomposition property, with \( a_i \leq i < \)

\[
x^i_B(n) = x^i_B(n)
\tag{25}
\]

if and only if \( i \) and \( j \) satisfy

\[
R(i)(n) < i < j \leq R(p+1)(n), \quad 1 \leq p \leq B - 1
\tag{26}
\]

or

\[
i < j \leq R(1)(n)
\tag{27}
\]

or

\[
R(p)(n) < i < j
\tag{28}
\]

where \( R(p)(n) \) is the \( p \)-th smallest sample in the window \( [r_1(n), r_2(n), \ldots, r_B(n)] \). Therefore, all unique vectors \( x^i_B(n) \) seen by neuron \( l \) appear on levels \( R(p)(n) \) if

\[
a_i \leq R(p)(n) < b_i, \quad 1 \leq p \leq B
\tag{29}
\]

and on level \( b_i \) if

\[
R(p)(n) < b_i \leq R(p+1)(n), \quad 1 \leq p \leq B - 1
\tag{30}
\]

or

\[
b_i \leq R(1)(n)
\tag{31}
\]

or

\[
R(p)(n) < b_i
\tag{32}
\]

are true. Since at least one of (30), (31), or (32) will always be true for a given \( b_i \), it follows that the term (24) can be written as

\[
\sum_{i=m}^{b_i} N_i[x^i_B(n)] + \sum_{i=R(1)(n)}^{R(B)(n)} N_i[x^i_B(n)] + \sum_{i=R(B+1)(n)}^{b_i} N_i[x^i_B(n)]
\tag{33}
\]
where

\[ a_1 \leq R_{(k_0)}(n) \leq b_1 \]  \hspace{1cm} (34)

and

\[ R_{(k_0)}(n) \triangleq \max\{a_1 - 1, R_{(k_0)}(n)\} \]  \hspace{1cm} (35)

with \( R_{(k_0)}(n) \) subject to the restriction of (34). Note that if (34) cannot be satisfied for any element in the window, (33) reduces to the right-most term. We can now replace the summation as follows:

\[
\begin{align*}
N_{1}[x_{B}^{R_{(k_1)}(n)}(n)] & \left[ R_{(k_1)}(n) - a_1 + 1 \right] + N_{1}[x_{B}^{R_{(k_2)}(n)}(n)] \\
\times [R_{(k_2)}(n) - R_{(k_1)}(n) - 1 + 1] + \ldots + \left[ N_{1}[x_{B}^{R_{(k_0)}(n)}(n)] [b_1 - R_{(k_0)}(n) - 1 + 1] \right].
\end{align*}
\]  \hspace{1cm} (36)

However, since \( a_1 = b_{l-1} + 1 \), this becomes

\[
\begin{align*}
N_{1}[x_{B}^{R_{(k_1)}(n)}(n)] [R_{(k_1)}(n) - b_{l-1}] & + N_{1}[x_{B}^{R_{(k_2)}(n)}(n)] [R_{(k_2)}(n) - R_{(k_1)}(n)] + \ldots + \left[ N_{1}[x_{B}^{R_{(k_0)}(n)}(n)] [b_1 - R_{(k_0)}(n)] \right].
\end{align*}
\]  \hspace{1cm} (37)

Note that the terms in the brackets are equivalent to those in (21) if we expand \( R_{(n)}(n) \) to also operate on the \( b_i \) terms. By forming a set \( X(n) \), as shown in (17), on which the redefined \( R_{(n)}(n) \) values are based, it is clear that (37) can be written as the inner summation in (18). With the other neural operators taken into consideration, (18) follows directly. \( \square \)

Depending on the number of neural operators \( K \), the FAST-GANF achieves a compromise between performance and speed. With \( K = (M - 1) \), the FAST-GANF is always identical to a standard GANF and can implement more difficult classifications of the data. This will result in the best filtering performance. With \( K = 1 \), the FAST-GANF is homogeneous and will be the fastest. Its neural net classifier will be maximally generalized, which may increase the filter’s robustness at the expense of classification ability. However, since the combination of neural operators and, hence \( K \), is based on a function of similarity measure, the FAST-GANF can be expected to achieve a reasonable speed/performance compromise. Note also that the FAST-GANF can be directly extended to the case where \( I \neq 0 \) by redefining the set \( X(n) \), the input vectors \( u(n) \), and the variable \( f(i,j,n) \). However, unless the method for combining levels is changed, the FAST structure would not significantly reduce the complexity in this case.

### IV. OTHER ADVANTAGES

The obvious benefit of level combinations is speed. The level combinations allow the use of FAST structures. However, given a limited set of training data, the simplifications may also result in improved performance. Note that with a completely nonhomogeneous GANF, \( (M - 1) \) neural operators must be trained using \( N \) samples of training data. However, at the top and bottom of the stack, there will be many binary inputs consisting of all 0’s and all 1’s, respectively. Therefore, many of the neural operators will be redundantly trained on these trivial inputs. To make this clearer, we will consider a GANF with \( (M - 1) \) levels and a window size \( B \). In this case, there will exist a maximum of \( (B + 1) \) unique binary input vectors after the threshold decomposition operation. If the smallest integer in the window is \( A \) and the largest is \( C \), then levels \( I \) through \( A \) have inputs of \( [1 \ldots 1] \), and levels \( (C + 1) \) through \( (M - 1) \) will see inputs of all zeros \( [0 \ldots 0] \). Here, of course, the vectors \( [1 \ldots 1] \) and \( [0 \ldots 0] \) consist of \( B \) elements. As a result, lower levels and upper levels may not experience a number of unique training samples equal to the size of the training data set.

A level \( I \) will not be trained with anything new at time \( n \) if

\[
r_k(n) < l \quad \forall 1 \leq k \leq B \]  \hspace{1cm} (38)
TABLE V
FILTERING RESULTS FOR LARGE GAUSSIAN NOISE (LENA4 AND ALANE4)

<table>
<thead>
<tr>
<th>Filter description</th>
<th>Image</th>
<th>SNR [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard GANF ($M = 255, I = 0$)</td>
<td>lena4</td>
<td>10.84</td>
</tr>
<tr>
<td>FAST-GANF ($\beta = 0.98, 42$ ml fcts)</td>
<td>lena4</td>
<td>9.64</td>
</tr>
<tr>
<td>FAST-GANF ($\beta = 0.98, 1$ neural fcts)</td>
<td>lena4</td>
<td>6.85</td>
</tr>
<tr>
<td>standard GANF ($M = 255, I = 0$)</td>
<td>alane4</td>
<td>6.35</td>
</tr>
<tr>
<td>FAST-GANF ($42$ neural fcts)</td>
<td>alane4</td>
<td>6.87</td>
</tr>
<tr>
<td>FAST-GANF ($1$ neural fcts)</td>
<td>alane4</td>
<td>4.78</td>
</tr>
</tbody>
</table>

or if

$$r_k(n) \geq l \quad \forall 1 \leq k \leq B$$

(39)

where $r_k(n)$ is as defined in (1). To show this in a more quantitative manner, let us assume that $r_k(n)$ is uniformly distributed and independent of itself at other time instants. Then

$$P[\text{all elements in } r(n) < l] = \left( \frac{l}{M} \right)^B$$

(40)

and

$$P[\text{all elements in } r(n) \geq l] = \left( \frac{M - l}{M} \right)^B$$

(41)

where $M$ equals the number of integers in the domain of the GANF. Suppose, as a worst case, we look at level 1. Then

$$P[\text{all elements in } r(n) \geq l] = \left( \frac{M - 1}{M} \right)^B .$$

(42)
The probability that this level sees an input vector other than
\[ [1 1 \cdots 1] \] is
\[ 1 - \left( \frac{M-1}{M} \right)^B . \tag{43} \]
The expected number of nontrivial training samples given a
training set of length \( N \) is
\[ N \left[ 1 - \left( \frac{M-1}{M} \right)^B \right] . \tag{44} \]
In order for the neural operator on this level to become
generalized, we would like this number to be greater than or
equal to 10 times the VCdim of the network \[ \text{[8]. Therefore} \]
\[ N \left[ 1 - \left( \frac{M-1}{M} \right)^B \right] \geq 10 \text{ (VCdim)} \tag{45} \]
or
\[ N \geq \frac{10 \text{(VCdim)}}{1 - \left( \frac{M-1}{M} \right)^B} . \tag{46} \]

For a filter of window size \( B = 9 \), operating on an 8-bit input
(\( M = 256 \)), this would be \( N \geq 289 \) times the VCdim, or
in other words, large. As a result, the upper and lower levels
may not always receive enough new training samples to allow
for proper generalization. In addition, while the training may
allow for adequate operation given the statistics of the signal
at hand, the filter may not perform well on different signal
distributions. Lack of unique training samples could result in
improper generalization of the networks, thereby preventing
robust operation. Because of the level combinations, the neural
operators in the FAST-GANF are more likely to be trained
with nontrivial samples at each time instant. As a result, the
simplified structures not only increase the speed but can also
increase the filter’s performance on untrained segments of the
same image or on signals with different statistical distributions.

The extent of these generalization advantages will, however,
be affected by the training set size and the signal and noise
distributions of the image being filtered.

V. RESULTS AND CONCLUSION

The reduction in training time resulting from the use of
a FAST structure is dependent on the window size and the
degree of level combinations. As long as the window size
plus the number of neural functions is less than the range
of input integers \( M \), the FAST-GANF will be more efficient.
In addition, overall performance may be enhanced due to an
increase in generalization.

To verify these results, several simulations were performed.
We corrupted two images (“lena” and “alane”) with varying
amounts of mixture noise and Gaussian noise. Mixture noise
was created by using the probability density function
\[ P(x) = (1 - \epsilon) \phi \left( \frac{x}{\sigma_1} \right) + \epsilon \phi \left( \frac{x}{\sigma_2} \right) \tag{47} \]
where \( \phi(x) \) is the pdf of a Gaussian random variable with zero
mean and unit variance. For both images, noise signals were
generated separately and added to the images. The signals were
then clipped to maintain an 8-bit brightness range. Fig. 3(a)–(c)
show the clean “lena” image and two of the noisy images.
Statistics over the lower right-hand (LRH) three quarters of
the images are provided in Table I.

Three filters with \( 3 \times 3 \) square windows were set up and
separately trained on the upper left-hand (ULH) quarter of the
“lena” images. A standard GANF with 255 levels, a FAST-
GANF with \( \beta = 0.98 \), and a FAST-GANF with \( \beta = 0 \) were
used. Then, with the weights fixed, the 12 trained filters were
used to filter the LRH three quarters of the “lena” and the
“alane” images. Consistent results \[ [6] \] have been obtained
for different images and window sizes, but for illustrative
purposes, three of the output images are shown in Fig. 3(d)–(f).
The numerical results are provided in Tables II through V and were computed over the LRH three quarters of the images.

Basically, the first three lines in each table show how well the filter can perform after training on a signal with similar statistics. The corresponding bottom three lines in each table show what happened when noise statistics remained the same as for training but with the signal changed.

Examining the SNR values, it can be seen that the standard GANF always performed better than the others when operating on distributions for which it was trained. For these cases, the FAST-GANF with $\beta = 0.98$ almost always did better than with $\beta = 0$. Looking at the results for the “alan” images, it can be seen that for small amounts of noise, the standard GANF performed best. However, for large noise variances, the FAST-GANF with $\beta = 0.98$ produced the highest SNR output. This implies the existence of a threshold for the noise, beyond which, better performance in terms of SNR is expected with a FAST structure. This reinforces the results of Section IV since larger noise creates more input values that are clipped to the extremes of 0 and 255. This, in turn, leads to an increase in repeated or trivial inputs, which were shown to affect the standard GANF to a greater extent.

Comparing the two FAST-GANF’s, $\beta = 0.98$ worked better than $\beta = 0$ for large noise powers (Tables III and V) but not for smaller noise (Tables II and IV). This was especially true for the “alan” images. Based on the good overall performance of the standard GANF, one would expect $\beta = 0.98$ to produce better results than $\beta = 0$ for all of the images. This apparent discrepancy remains unresolved at this time but may be due to the varying statistics and correlation exhibited among various levels of the corrupted signals that are not tractable.

REFERENCES


