AN INTRODUCTION TO MULTISCALE DEFINED SYSTEMS: SELF-ORGANISING IFS FRACTAL NETWORKS

Graham C. Freeland & Tariq S. Durrani

Signal Processing Division
Dept. of Electronic And Electrical Engineering
University of Strathclyde
Glasgow G1 1XQ, Scotland, UK
{g.freeland, t.durrani}@eee.strath.ac.uk
http://www.spd.eee.strath.ac.uk/~gcf/icassp98/

ABSTRACT

Deterministic multiscale defined representational forms have found a significant role in the theory and application of signal processing over the last decade. With little argument the most widely important for signal and system modelling is likely to be multiscale defined wavelets. Another class of multiscale representation which has attracted consistent interest over the same period is the group of signal models defined in terms of Iterated Function Systems (IFS). This paper is concerned with widening the IFS application to include system modelling, particularly of neural network-like structures. We introduce an interpolating IFS model as a form of self-organising map with global fractal constraints. Symbolic addressing is employed to discretize the attractor into pseudo network nodes. We present in detail an online gradient based algorithm for training. This particular model is intended for efficient pattern recognition in complex environments, for example, with multifractal sources such as those seen in network traffic and general turbulence.

1. MOTIVATION AND BACKGROUND

1.1. Research Aims and Paper Outline

The core research presented in this paper is on the training of Iterated Function System (IFS) constrained networks for use in pattern recognition-like problems. More generally it is part of a larger attempt to introduce and develop the use of IFS models in a wider range of application, particularly system representation. The initial motivation stemmed from experiments carried out by the authors into the behaviour of simple IFS defined nonlinear dynamics [1]. From this synthesis orientated work the question arose as to whether IFS could be useful in a system analysis or modelling role; much of their previous application has been lossy image coding and compression.

Given the novelty of the programme, the first section of the paper is concerned with justifying in detail our research work by answering the following two questions,

- **Can IFS be used to represent systems?**
- **Why use IFS to represent systems?**

This work was supported by EPSRC under Contract #GR/L16170

The overall representational possibilities of IFS models have to be established and we begin by generalising the argument. We interpret them as a particular form of Multiscale Difference Equation (MSDE) model and restate the connections with wavelet based representations. Focus returns to the IFS specific case with a discussion of how the benefits brought to other applications may be exploited in system modelling and we introduce the example of (neural) network models constrained by IFS. The subsequent section describes at length the training algorithm for a two-dimensional self-organising map (SOM) based on a one-dimensional fractal interpolation function. We conclude with brief comment on further research directions.

1.2. MSDE Representations

Multiscale Difference Equations (MSDE) While linear difference equations have always been a central representional tool for most things, signals and systems, the past decade has seen the rise in importance of a number of generalisations to the basic model. One such extension is the *dilation equation* or *multiscale (or multirate) difference equation* (MSDE). Based around dilation and translation, the generic system 'input' and 'output' now operate across different scales:

$$y(t) = \sum c_i x(a_i t - b_i) \ . \tag{1}$$

This mathematical structure plays an implicit and defining role in many aspects of modern theory and application.

Wavelets and Iterated Function Systems (IFS) Perhaps the most significant single MSDE discovery, not least by the broad range of interests which have embraced it, is the class of time-frequency localised orthonormal basis functions given by multiscale defined wavelets.

Another class of MSDE defined representation which has attracted consistent interest over the same period is the set of extensions to the basic Iterated Function Systems (IFS) first introduced by Barnsley. For a number of reasons these models have experienced less penetration with their main impact being restricted to lossy image coding and compression. The original motivation for these particular image applications stemmed from the impressive coding efficiency of IFS for 'natural' image patterns.

While quite different in their approaches to and abilities for representation, there exist potentially valuable connections between the above two techniques [2]. The exact

form and usage of the MSDE decide the associated models' structure, properties and applicabilities.

Can IFS be used1; to represent systems? The success of this wavelet representation, for both signal and system modelling, typically arises from the (orthonormal) integral transform structure. A single MSDE specifies the 'mother' wavelet or *basis function* from which, via further dilation and translation, the entire basis set of an orthonormal transform is expanded. As such, the transform is capable of optimally (non-redundantly) representing any squared integrable function.

A typical IFS approach to representation does not result in this classical linear algebraic solution. Again a single MSDE is usually employed but, in strong contrast to the above, this equation is defined explicitly in terms of (subsets) of the entire 'domain' set. IFS based models generally operate by approximation and do not offer the possibility of lossless representation.

There are however highly *redundant* exceptions which allow *lossless* signal representation [2]. This general applicability is reinforced by the relationship between IFS coding and wavelet representations (see ref. in [2]). To the extent that 'signals' are temporal functions, IFS should be equally capable therefore of modelling system transfer functions.

Why use IFS to represent systems? Of course in using IFS for system modelling, we wish to retain and exploit their beneficial features. These include,

- Simple, recursive specification
- Compressive description
- Discrete or symbolic interpretation

We can contrast this with neural network-like approaches to system representation, whether they be for pattern recognition or nonlinear transfer function modelling, where wavelets have played a role too.

In addition to providing 'optimal' performance, either in terms of classification or prediction error reduction, it is a typical requirement for the network to be of minimal size. This system 'compression' is achieved by methods ranging from parameter quantization to *a posteriori* pruning.

In this paper we represent networks in terms of IFS attractors. Instead of looking for redundancy in the model after training is complete, the IFS notation allows for an arbitrary number of 'nodes' but imposes a global recursive structure on the parameter set. Encoding is therefore implicit to this model. An significant issue in training is how to turn the 'pointwise' local updates of online training into that which will update the entire set of maps. Central to the network representation and subsequent training is use of symbolic dynamics to discretize the attractor into a partition, the subsets of which can be interpreted as pseudo 'nodes'.

2. SELF-ORGANISING IFS NETWORKS

2.1. Self-Organising (Neural) Maps

The self-organising map (SOM) is a form of neural network which acts to 'compress' high-dimensional data by extracting and projecting data features and their topological relations onto a low dimensional network structure. They have been found useful in a wide range of tasks, from computer vision to communications [3]. Training typically proceeds in a manner similar to that for vector quantization. In the case of online training, network nodes often have a simple update rule of the form,

$$m_k(t+1) = m_k(t) + \xi_k(t)[X(t) - m_k(t)] \qquad (2)$$

where $X(t)$ is the current input vector and m_k is the k-th node value. The exact form of the weighting function $\xi_k(t)$ decides the training properties. The FIF model considering here has a similar outward structure but the important difference is that global constraints are placed on the node values and it is these constraints whose parameters are to be trained. This implicitly allows for interesting new possibilities such as network coding and interpolation and extrapolation.

2.2. Fractal Interpolation Functions (FIF) as Networks

Basic FIF Mathematical Model The defining components of the simplist IFS comprise a set of contractive maps $\{w_n\}$ and an associated set of probabilities $\{p_n\}$. The representional object defined by the maps is the set of points, known as the attractor R, that is attracting and invariant under the collective action of the maps. This set displays scale redundancy and, by virue of the contractive maps, can be seen to be formed from smaller copies $\{w_n(R)\}$ of the whole, R. In all that follows, the probabilities are assumed such that there is a uniform distribution across the attractor.

The attractor of the particular IFS variant considered here, an example of an affine Fractal Interpolation Function (FIF), defines the graph of a mapping, F, from the unit interval, I to a continuous curve C, embedded in the plane, i.e. $F([0,1]) = C$. As might be expected with interpolation functions, they are defined by a set of points $P = \{(x_n, (y_n, z_n)) : n = 0, 1, \ldots, N\}$ (the set of interval domain points $\{x_n\}$ is strictly increasing) through which the set $R = I \times C$ will then pass.

To formalize the associated model, we define and parameterise a finite set of N affine maps $\{w_n : n = 1, 2, \ldots, N\}$ of the form

$$w_n \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_n & 0 & 0 \\ 0 & d_n & h_n \\ 0 & l_n & h_n \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_n \\ p_n \\ q_n \end{bmatrix}$$
$$= \begin{bmatrix} a_n x & + & e_n \\ A_n \begin{bmatrix} y \\ z \end{bmatrix} & + & B_n \end{bmatrix} = \begin{bmatrix} L_n(x) \\ M_n(y,z) \end{bmatrix} \qquad (3)$$

such that they satisfy the requirement that, for all n,

$$w_n(0 \;\; y_0 \;\; z_0) = (x_{n-1} \;\; y_{n-1} \;\; z_{n-1}) \qquad (4)$$
$$w_n(1 \;\; y_N \;\; z_N) = (x_{n-1} \;\; y_{n-1} \;\; z_{n-1}) \qquad (5)$$

The free parameters, which we will take to be the set of pairs $S = \{d_n, m_n\}$, are known as scaling parameters and control the fractal dimension or irregularity of the curve. To ensure contractivity their absolute values are taken to be less than unity (when all zero, a piece-wise linear interpolation function results).

The key component to this FIF is that *the 2-D curve segments of C defined between the interpolation points will*

be affinely contracted copies of the whole, and the structure is independent of the additional dimension x. As we will see below, the purpose of the additional unit interval dimension is to provide a method of parameterisation for the curve based around the natural discrete partition. This parameterisation is very useful in defining the error equations and in forcing the curve to behave in a 'network'-like fashion.

Invariance Relations and Symbolic Discretization We have not yet formally described the way in which the 2-D function $F = (f_{(1)}, f_{(2)})$ is invariant under the collective action of the maps. We proceed by defining the function operator $T : F \to G$,

$$\begin{aligned} G(x) &= (TF)(x) \\ &= A_n F(L_n^{-1}(x)) + B_n \qquad (6) \\ &\text{for } x \in I_n = [x_{n-1}, x_n] \text{ and } n \in \{1, 2, \ldots, N\} . \end{aligned}$$

where $L_n : (I = [0,1]) \to (I_n = [x_{n-1}, x_n])$. If the absolute values of vertical and horizontal scaling factors $\{d_n, m_n\}$ are all less than unity then the operator is contractive with an attracting fixed point function, F, i.e.

$$F(x) = TF(x) \implies \int F(x)\,dx = \int (TF)(x)\,dx . \qquad (7)$$

Being attractive, for any initial function H, the sequence $T^{\circ i} H$, i.e. the repeated application of T on H , will converge towards F as $i \to \infty$.

It is obvious that the set $\{I_n\}$ form a symbolic partition of the interval I. We denote the function F restricted to I_n as F_{I_n} or $F_{(n)}$ (similarly the restricted integral is denoted by $(IF)_{(n)} = (\int_{I_n} f^1\,dx, \int_{I_n} f^2\,dx)$). This represents the contracted affine copy of the entire curve C which 'sits between' interpolation points (y_{n-1}, z_{n-1}) and (y_n, z_n) i.e. $M_n(C)$. We can extend this domain restriction to finer partition levels, writing,

$$\begin{aligned} F_{(n_1 n_2 \cdots n_M)}(x) &= \{F(x) : x \in I_{n_1 n_2 \cdots n_M}\} \qquad (8) \\ &\text{where } I_{n_1 n_2 \cdots n_M} = L_{n_1}(L_{n_2}(\cdots L_{n_M}(I))) . \end{aligned}$$

The sequence $n_1 n_2 \cdots n_M$ is known as the symbolic address of $I_{n_1 n_2 \cdots n_M}$ and $F_(n_1 n_2 \cdots n_M)$.

It is via this partition that we generate the pseudo network or SOM. For a given level of partition, M, N^M 'subfunctions' can be identified, $Fs = \{F_{(n_1 n_2 \ldots n_M)}\}$. The endpoints of these function sets, Ps, are the offspring of the interpolation points, i.e. the result of applying all (finite) combinations of FIF maps to P,

$$\begin{aligned} Ps &= \{F(L_{n_1}(L_{n_2}(\cdots L_{n_N}(\{0,1\})))\}_{n_1 n_2 \ldots n_M} \qquad (9) \\ &= \{M_{n_1}(M_{n_2}(\cdots M_{n_N}(\{P\})))\}_{n_1 n_2 \ldots n_M} . \qquad (10) \end{aligned}$$

In defining 'network nodes' associated with this model, we can proceed by two routes. The most direct is to use the extended interpolation point set Ps. This result comes close to what might be regarded as a 'traditional' constrained network.

The alternative employed here is the use of the function sets between these points as nodes, i.e. Fs. Simply, each curve 'unit' $F_{(n_1 n_2 \cdots n_M)}$ is taken to represent a node. Instead of operating directly on the 'node' set, any training or manipulation will be via their averages. This approach leads to more generally applicable theory.

2.3. FIF Training

Algorithm Structure In undertaking the training of a FIF network model, we proceed along the similar lines to that for self-organising maps (after VQ). An input data sequence X(t) acts as input to the *online* training procedure which, for the purposes of this paper, follows a simple two stage procedure involving,

- **Nearest Neighbour Identification**
- **Global Parameter Update**.

Only the update of the nearest node is considered but, since the FIF constraint is a global one, even the update of a single node will require change across the entire parameter set. Even without the recursive constraint, the continuity of the FIF model ensures that at least neighbouring elements are effected. Our online training algorithm proceeds by stochastic approximation minimisation and has the form,

$$\lambda_k(t+1) = \lambda_k(t) - \xi_k(t) \partial V / \partial \lambda_k \qquad (11)$$

where V is the integral error between the current FIF and a target function incorporating the latest sample of input data $X(t)$. Dropping any dependency on the training time t, the error functional is given by,

$$V = \int_I \sum_{j=1}^{2} (f_j(x, \Lambda) - u_j(x))^2 dx \qquad (12)$$

where $f_j(x, \Lambda)$ and $u_j(x)$ are the pair of dimensions of the current FIF F, and target function U, respectively. The range parameter set (2-D interpolation points and scaling parameters) is denoted by $\Lambda = (S, P)$. This set does not include the domain coefficients $\{a_n, e_n\}$ which are assumed to be constant.

Since, for each update, the target input is a single point $X(t)$, it has to be used to specify a target function defined for every point of the interval parameterised function, $F(x)\,\forall x \in I$. As can be seen from equation (12), the error function is simply the integral of the pointwise error between this target and the FIF. How we define U will decide which 'nodes' are pulled towards the input vector and which are left unchanged *a la* SOM.

Nearest Neighbour Identification and Updating Suppose the depth of the scheme, M has been decided *a priori*. Since the FIF curve is being interpreted discretely through the partition, there is no real need to explicitly calculate the distance between the given input vector X, and every point on the curve. What is required is a measure of the distance between X and each function element $F_{(n_1 n_2 \ldots n_M)}$ that may then be used to define the required target U. Since the form of error functional V, is assumed fixed, the mechanism by which this set of distances is turned into U completely decides the type of error weighting. For this paper we choose a simple nearest neighbour update procedure: only the function element nearest to the input,

$$F_{n^*} \text{ where } n^* = \arg \min_{n_1 n_2 \ldots n_M} d(X, F_{(n_1 n_2 \ldots n_M)}) \qquad (13)$$

is targeted for change with all others being untouched (d is some point to set distance measure). To implement this

update rule the target function is set to

$$\begin{array}{ll} U_{n^*}(x) = F(x) + E & \text{if } x \in I_{n^*} \\ U_{n^*}(x) = F(x) & \text{otherwise} \end{array} \quad (14)$$

where $E = [\epsilon_1, \epsilon_2] = d(X, F_{n^*})$. This has the effect of windowing out the appropriate subinterval for change. Again note that while only a single element identified, it is typically defined by a mixture of the entire parameter set.

There are a number of implementional approaches to the estimation of n^*. We have developed a pointwise targeting algorithm which iteratively estimates the nearest subinterval at the given symbolic resolution. Details will be made available in a followup paper.

Global Parameter Update The final stage of the algorithm requires calculation of the error functional partials with respect to each parameter $\lambda_k \in \Lambda$. A related problem can be found in [4] and we proceed similarly to give,

$$\alpha_k = \frac{\partial V}{\partial \lambda_k} = \int_I \sum_{j=1}^{2} 2(f_j(x,\Lambda) - u_j(x)) \frac{\partial f_j(x,\Lambda)}{\partial \lambda_k} dx \quad . \quad (15)$$

Once the symbolic address n^* of the nearest neighbour subinterval has been found, substitution of U_{n^*} reduces the above to,

$$\alpha_k = 2 \sum_{j=1}^{2} \int_{I_{n^*}} \frac{\partial f_j(x,\Lambda)}{\partial \lambda_k} dx \quad . \quad (16)$$

Now the problem becomes one of calculating the vector integral of $\partial F / \partial \lambda_k$ over a subpartition I_{n^*} of the interval which be denoted by $(Id_k F)_{(n^*)}$. This may be calculated efficiently by taking the partial derivatives of the operator T (eqn (7)) and the invariance relations (7) (differentiation under the invariance integral is possible [4]). The resulting equations are boxed together under Table 1; further implicit definitions employed are $Id_k F$ representing the integral of $\partial F / \partial \lambda_k$ over the entire unit interval and IF denoting the integral of F over the same. Remember that $\{a_n, e_n\}$ are assumed constant.

Firstly, the derivative of defining operator T results in basic multiscale relation for $(Id_k F)_{(\cdot)}$. Using the recursion relation (19) and the symbol sequence n^*, one can iteratively calculate the required integral, $(Id_k F)_{(n^*)}$, starting from $Id_k F$ and IF. These may be explicitly calculated by taking the derivative of the integral relation (7). Solving the resulting pair (20) results in the required values. The final update equations are

$$\lambda_k(t+1) = \lambda_k(t) - \xi_k(t)\, 2\, (Id_k F)_{(n^*)} \begin{bmatrix} 1 & 1 \end{bmatrix}' \quad . \quad (17)$$

Since there is insufficient space available to list the actual form of the derivatives we pass brief comment on their structure. Remembering that the matrices A_n and vectors B_n are designed to satisfy equations (4) & (5), some entries in the parameter set will result in a more complicated update rule than others. All the scaling parameter S updates have a similar structure and so do the interpolation points for $n = 2, 3, \ldots, N-1$. As the endpoints ($n = 0, n = N$) figure in all M_n their updating is more complicated.

When implementing this algorithm the most important issue is to factor into $\{\xi_k\}$ the differing sensitivities between the S and P. Additionally, the scalings must remain contractive. Training can be speeded up by exploiting the implicit hierarchical aspects of the model e.g. allowing the symbolic depth M to vary with t. Full details will appear elsewhere but simulation results will be presented at conference.

3. FUTURE WORK

In this paper we have introduced and motivated a programme of research aimed at employing Iterated Function System as efficient, compressive signal models. The specific application presented was in creating and training a simple IFS constrained self-organising network.

With regard to further work, many opportunities exist. Immediate pattern recognition extensions exist allowing decision regions to be created by completing IFS models. We conjecture that in the case of data being drawn from *multifractal* distributions, the optimal decisions regions will have fractal basin boundaries. In such a senario FIF should be able to outperform their smooth classical counterparts. This approach leads to the interesting idea of an information theoretic version of the IFS collage theorem.

4. REFERENCES

[1] G.C.Freeland and T.S.Durrani, *The Application of Fractal Signals to Communications.* 1995 IEEE Workshop on Nonlinear Signal and Image Processing, pp. 775-778, Halkidiki, Greece, 1995.

[2] D.Saupe, *A New View of Fractal Image Convolution Transform Coding.* IEEE Sig. Proc. Lett., Vol. 3, No. 7, pp. 193-195, 1996.

[3] T. Kohonen et al, *Engineering Applications of the Self-Organising Map.* Proc. IEEE, Vol.84, No. 10, pp. 1358-1383, Oct. 1996.

[4] W.D.Withers, *Newton's Method for Fractal Approximation.* Constr. Approx., No. 5, pp. 151-170, 1989.

$$\begin{aligned} (Id_k F)_n &= \int_{I_n} \frac{\partial F}{\partial \lambda_k} dx = \int_{I_n} \frac{\partial TF}{\partial \lambda_k} dx = a_n A_n \int_I \frac{\partial F}{\partial \lambda_k} dx + a_n \frac{\partial A_n}{\partial \lambda_k} \int_I F dx + a_n \int_I \frac{\partial B_n}{\partial \lambda_k} dx \\ &= a_n A_n Id_k F + a_n \frac{\partial A_n}{\partial \lambda_k} IF + a_n \frac{\partial B_n}{\partial \lambda_k} \quad \forall n \in \{1, 2, \ldots, N\}. \quad (18) \\ (Id_k F)_{(n_1 n_2)} &= a_{n_1} A_{n_1} (Id_k F)_{(n_2)} + a_{n_1} \frac{\partial A_{n_1}}{\partial \lambda_k} IF + a_{n_1} \frac{\partial B_{n_1}}{\partial \lambda_k} \quad (19) \end{aligned}$$

$$\left\{ \begin{array}{rcl} Id_k F &=& \sum_n (Id_k F)_{(n)} = (\sum_n a_n A_n) Id_k F + (\sum_n a_n \frac{\partial A_n}{\partial \lambda_k}) IF + (\sum_n a_n \frac{\partial B_n}{\partial \lambda_k}) \\ IF &=& (\sum_n a_n A_n) IF + (\sum_n a_n B_n) \end{array} \right\} \quad (20)$$

Table 1: Partial Derivatives