Proceedings of the 1998 IEEE
International Conference on Control Applications
Trieste, Italy 1-4 September 1998

TA06

Invertible Piecewise Linear Approximations for Color Reproduction

R. E. Groff D. E. Koditschek P. P. Khargonekar
regroff@eecs.umich.edu kod@eecs.umich.edu pramod@eecs.umich.edu
EECS Department, University of Michigan
1301 Beal Ave., Ann Arbor, MI 48109-2122, USA

Abstract

We consider the use of linear splines with variable knots for the approximation of unknown functions from data, motivated by control and estimation problems arising in color systems management. Unlike most popular nonlinear-in-parameters representations, piecewise linear (PL) functions can be simply inverted in closed form. For the one-dimensional case, we present a study comparing PL and neural network (NN) approximations for several function families. Preliminary results suggest that PL, in addition to their analytical benefits, are at least competitive with NN in terms of sum square error, computational effort, and training time.

1. Introduction

When a computer sends a color document to a laser printer, the color of each pixel is represented as a vector in a standard color space, such as **Lab**, a space based on the psychophysics of the eye. The printer transforms the vector into a device-dependent space, **CMY**, which specifies the quantity of each pigment the printer must lay down in order to reproduce the desired color. This transformation is nonlinear and, approximate first principles models notwithstanding, must be computed in practice from empirical data [1]. When collecting data, each experiment consists of specifying a **CMY** vector to the printer and measuring the output in **Lab** coordinates. Thus, the empirical data is gathered using the inverse of the desired transformation. Color science also indicates that the map will be injective, so it may be inverted, at least over its range. In current industry practice, the color transformation is performed by interpolation on a lookup table with approximately one to two thousand entries. Industry goals are to reduce the number of parameters required to perform the color transformation, thereby reducing the cost of calibration as well as hopefully yielding a functional representation which may be amenable to online updating methods as the function drifts and more calibration data is collected.

Beyond interpolation on a uniform grid, one of the most sophisticated techniques for approximating color space transformations currently in use in the color industry is sequential linear interpolation [2]. This approach applies asymptotic analysis from information theory to find the optimal (nonuniform) grid point placement.

Much attention has been given to various parameterizations of the space of approximations, especially nonlinear parameterizations such as Neural Networks (NN) and Radial Basis Function Networks (RBFN). In higher dimensions, the convergence per parameter rates for such nonlinear families can potentially be better than linear-in-parameter function families [3]. Unfortunately, popular nonlinear families like NN and RBFN generally do not admit the "leveraging" of additional domain knowledge about the function, such as invertibility. In our application setting, NNs and RBFNs require either i.) that a second network be trained in order to construct the function inverse or ii.) that some further numerical procedure be applied to generate the inverse. But in the color space transformation problem, the inverse map is just as important as the forward map. The printer physically realizes the inverse map and the application of control methodology seems most reasonable when working with the function most closely related to the physical system.

A novel approximation method suggested by Atkeson and Schaal [4] uses a population of local "experts." Each "expert" is associated with an affine map and a Gaussian confidence. The "experts" vote on an output computed as the confidence weighted average of their affine components. Since the Gaussian bump has unbounded support, each expert has global influence.

Piecewise polynomial representations (splines) can also be applied to function approximation problems. Qualitatively, these may be thought of as local "experts" which have partitioned the domain. Analytical results are available for the one dimensional case for certain function families [5]. Algorithms exist for one dimension [6] as well as for higher dimensions [7], although analytical results hold only for the one-dimensional case. Stone et al. present a statistical theory for the rate of L_2 convergence [8].

2. Piecewise linear approximation

Piecewise linear approximations (PL), also known as linear splines with variable knots, comprise a function family in which the invertibility condition can be enforced, and the inverse can be calculated directly in closed form as well. In one dimension, the characterization of the piece-

0-7803-4104-X/98/$10.00 ©1998 IEEE

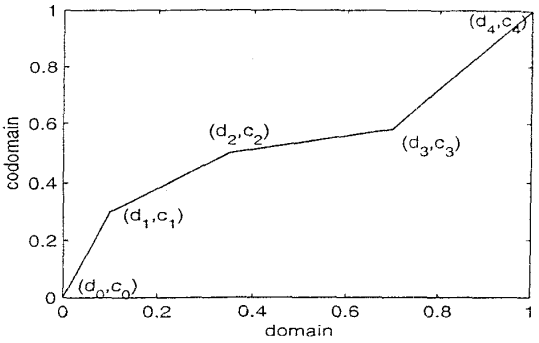

Figure 1: A PL with four line segments ($n=4$)

wise linear approximation is straightforward. Consider a PL with n line segments on the domain $[0,1]$ The PL is characterized by two vectors: $d \in \mathbb{R}^{n+1}$ the vector of domain values, or knots, where $0 = d_0 < d_1 < ... < d_{n-1} < d_n = 1$, and $c \in \mathbb{R}^{n+1}$ the vector of codomain values. This gives PL $2n$ free parameters. Figure 1 shows a PL with four line segments.

Analytical results exist for the one-dimensional function approximation problem. In approximation, a completely known function is given and the objective is to find a PL which minimizes some norm, typically L_2, of the error. Barrow et al. [5] provide some generalized convexity conditions which imply the existence of a unique best L_2 fit from the class of piecewise linear approximations. Gayle and Wolfe [6] provide similar results for approximation using higher order splines. Their proof uses an algorithm to calculate the best approximation over the domain of all knot vectors, for which global, unique convergence is shown via application of the contraction mapping theorem.

In higher dimensions,the domain is partitioned into simplices: triangles in two dimensions, tetrahedra in three dimensions, and so on. Tourigny and Baines [7] present an algorithm for the two-dimensional function approximation problem, which can be generalized to higher dimensions. There are no corresponding analytical results. In order to produce an output for a given domain point in higher dimensions, the partition in which the domain point lies must first be identified. Since the partitions are nonuniform, this step rapidly increases in complexity with dimension. This points out a fundamental tradeoff between the complexity of the approximant (e.g. linear, quadratic, neural) and the complexity of the partition. (e.g. none, uniform, nonuniform)

3. Numerical studies

This section presents a numerical study designed to compare the relative approximation power of PL and NN approximations. The PL and NN were given the same number of free parameters in order to study the relative performance per parameter.

3.1. Choice of function classes studied

Five different function families from the class of homeomorphisms on $[0,1]$ were explored. That is, all functions mapped $[0,1]$ to $[0,1]$ and were continuous and invertible (i.e. monotonic). The five families fall into three groups: sigmoidal, piecewise linear and polynomial, chosen to "favor," respectively, NN, PL or neither. Examples of typical functions from these families are presented in Figure 2.

The sigmoidal group contains superpositions of hyperbolic tangents, of the form

$$f(x) = k_1 \sum_{i=1}^{m} a_i \tanh\left(b_i\left(x - c_i\right)\right) + k_2 \qquad (1)$$

where a_i and c_i are distributed uniformly on $[0,1]$ and b_i is exponentially distributed with mean 30. Then k_1 and k_2 are chosen such that $f(0) = 0$ and $f(1) = 1$. The first family from the sigmoidal group has $m = 5$, so all functions in this family lie within the parameter space of the NN which was trained. Notice that this family is chosen to favor NN, since for $m=5$ there exists a vector of NN parameter values which would give zero error. The second family has $m = 15$, so, while presumably favored, the neural network is underparameterized.

The piecewise linear group contains functions with m line segments, characterized by the points $\{(x_i, y_i)\}_{i=0}^{m}$ with $0 = x_0 < x_1 < ... < x_{n-1} < x_n = 1$ and $0 = y_0 < y_1 < ... < y_{n-1} < y_n = 1$. The points x_i and y_i are chosen uniformly from $[0,1]$ for $i = 1, ..., n-1$. The first family in the piecewise linear group has 10 line segments ($m=10$), so, again all functions in this family lie within the parameter space of the PL approximation. The second family has 30 line segments ($m=30$), so the PL is underparameterized.

The polynomial group consists of compositions of quadratic polynomials which satisfy $f(0) = 0$ and $f(1) = 1$ and are monotonically increasing. i.e. $f'(x) \geq 0$ for $x \in [0,1]$. Quadratic polynomials satisfying these constraints can be parameterized as

$$f_\alpha(x) = (1-\alpha)x^2 + \alpha x \qquad (2)$$

for $\alpha \in [0,2]$. Then the polynomial $f = f_{\alpha_1} \circ f_{\alpha_2} \circ ... \circ f_{\alpha_m}$ is indeed a homeomorphism of $[0,1]$, since it is the composition of homeomorphic functions, and it has degree 2^m. The polynomial family presented here used $m = 7$ and the parameters α_i were distributed uniformly over $[0,1]$.

3.2. Training methods

The PL algorithm employed here minimizes square error via gradient descent. Specifically, given a data set $\{(x_i, y_i)\}_{i=1}^{N}$, the algorithm minimizes

$$E = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \hat{f}(x_i)\right)^2 = \frac{1}{2}\sum_{j=0}^{n-1}\sum_{x_i \in I_j}\left(y_i - \hat{f}_j(x_i)\right)^2 \qquad (3)$$

where $I_j = [d_j, d_{j+1}]$ and $\hat{f}$ is the piecewise linear approximation, defined by

$$\hat{f}(x) = \left\{ \hat{f}_i(x) = \frac{c_{i+1} - c_i}{d_{i+1} - d_i}(x - d_i) \text{ for } x \in I_i \right\} \quad (4)$$

The partial derivatives are

$$\frac{\partial E}{\partial c_i} = -R_{i-1} + R_i - S_i \quad (5)$$

$$\frac{\partial E}{\partial d_i} = \Delta_{i-1} R_{i-1} - \Delta_i R_i + \Delta_i S_i \quad (6)$$

where

$$R_i = \sum_{x_j \in I_i} \left(y_j - \hat{f}_i(x_j) \right) \left(\frac{x_j - d_i}{d_{i+1} - d_i} \right) \quad (7)$$

$$S_i = \sum_{x_j \in I_i} \left(y_j - \hat{f}_i(x_j) \right) \quad (8)$$

$$\Delta_i = \frac{c_{i+1} - c_i}{d_{i+1} - d_i} \quad (9)$$

The update law is then

$$d_i^{k+1} = d_i^k - \mu \frac{\partial E}{\partial d_i} \quad (10)$$

$$c_i^{k+1} = c_i^k - \mu \frac{\partial E}{\partial c_i} \quad (11)$$

where the superscript is the iteration number and μ is the step size or "learning rate."

The PL algorithm takes advantage of the fact that it is approximating a function from the class of homeomorphisms on $[0, 1]$ by fixing $c_0 = 0$ and $c_n = 1$, in addition to $d_0 = 0$ and $d_n = 1$. This reduces the number of free parameters for a PL with n line segments from $2n$ to $2(n-1)$. Allowing a neural network to use this information would require significant modification of the backpropagation algorithm. The PL in the following experiments uses 10 line segments, giving a total of 18 free parameters. The PL gradient descent algorithm was implemented in Matlab.

The neural network was also implemented in Matlab using the Neural Network Toolbox. The network has six hyperbolic tangent neurons situated in a single hidden layer, providing a total of 18 free parameters for the NN. The standard backpropagation rule, which minimizes the square error via gradient descent, was used to train the network.

3.3. Design of study

One hundred functions were randomly chosen from each function family discussed above. The algorithms received two data sets generated from each function. The training set contains 136 input/output pairs evenly spaced over the domain, while the validation set has 68 pairs interspersed between the test data, following the heuristic that approximately 2/3 of the data should be used for training and the remaining third should be used for validation.

First the PL was trained on each function until a stopping condition based on the magnitude of the gradient was achieved or the maximum number of iterations (4000) was exceeded. The NN was trained on the same data, given the goal of attaining 1/4 the sum square error of the PL. The NN stopped when this goal was achieved or after the maximum number of iterations, or "epochs," was exceeded. The maximum number of epochs for the NN was set at 12000.

Table 1: Mean Square Error and Iteration Results

Function Family		MSE (Validation Data) min/ave/max ($\log_{10}$)	Iter. ave
tanh	PL	-5.388 / -4.670 / -3.354	3026
m=5	NN	-4.538 / -3.236 / -2.423	12000
tanh	PLH	-5.562 / -4.639 / -3.758	3216
m=15	NN	-4.427 / -3.556 / -2.834	12000
pl	PL	-7.419 / -5.304 / -3.218	2783
m=10	NN	-4.213 / -3.430 / -2.685	12000
pl	PL	-4.749 / -4.257 / -3.721	3566
m=30	NN	-4.024 / -3.571 / -2.869	12000
polynom.	PL	-5.985 / -5.405 / -4.410	2243
m=7	NN	-5.517 / -4.276 / -3.526	12000

3.4. Results

The mean square error (MSE) on a data set $\{(x_i, y_i)\}_{i=1}^N$ is defined as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \hat{f}(x_i) \right)^2 \quad (12)$$

where $\hat{f}(x)$ is the approximation given by NN or PL. Because MSE takes on a wide range of values, $\log_{10} MSE$ is presented here. Table 1 shows the minimum, average, and maximum of $\log_{10} MSE$ for PL and NN on each family, as well as the average number of iterations on that family. For simplicity, $10^{ave(\log_{10} MSE)}$ will be referred to hereafter as the mean MSE. Figure 3 shows the MSE for PL and NN on every function for each of the families. The solid line shows the ratio MSE_{PL}/MSE_{NN}.

Notice that PL regularly achieves a smaller MSE than NN, with only a handful of exceptions, on all function families. This is true even for the two families which "favor" NN, superpositions of hyperbolic tangents. For the sigmoidal family with $m = 5$, it would be possible for NN to represent the functions exactly, but still the mean is $10^{-3.236}$, which is an order of magnitude worse than the PL. This is well illustrated by the ratios, MSE_{PL}/MSE_{NN}. The results for the sigmoidal family

with $m = 15$ are similar. Surprisingly, the mean MSE is not significantly different from that of the family with $m=5$. In fact, NN actually does better on average, even though it is "underparameterized."

PL is capable of exactly representing the piecewise linear function family with $m=10$. On some of the functions, PL performs approximately the same as it does on the other function families, but in many cases PL comes very close to the actual parameterization of the target function, with MSE_{PL} as small as $10^{-7.419}$. There are a total of 26 functions with MSE_{PL} less that 10^{-6}, and hence these points do not fall in the range of the plot. The minimum and maximum for MSE_{PL} are separated by over 4 orders of magnitude, giving MSE_{PL} a high variance over the family. Comparatively, NN performed consistently around the mean of $10^{-3.430}$, with its minimum and maximum being less than 2 orders of magnitude apart. The performances of PL and NN are most similar on the family of piecewise linear functions with $m=30$. For both PL and NN, the MSEs cluster closely around the algorithms' family means, and, unlike the other families, the mean of MSE_{NN} is within an order of magnitude of mean MSE_{PL}.

The neutral family, the polynomials, shows similar results. Once again the mean for MSE_{PL} is more than an order of magnitude below the mean MSE_{NN}, and fits for PL and NN cluster tightly around their respective means, similar to the piecewise linear family with $m=30$. In this family, however, there is typically a greater space between the two clusters. Note that this is the only family on which NN outperforms PL on a function, as seen by the spot where the ratio MSE_{PL}/MSE_{NN} goes above 1. The average of MSE_{PL} and MSE_{NN} is lower on this family than on the others, indicating that these polynomials are in some way easier to approximate than the other families.

Notice in the table that PL also had a lower number of iterations than NN. In general, NN timed out while trying to achieve the sum square error goal based on the PL performance. The lower number of iterations is also significant, since NN also uses more flops per iteration than PL. Thus PL could potentially have significantly shorter training times, at least over families such as these.

4. Conclusion

The color space transformation problem requires a function to be fit to data. The problem also presents the engineer with additional information about the underlying function. It is invertible. Piecewise linear algorithms afford the ability to check and enforce this invertibility.

In order to make the fit amenable to online updating, we desire a parsimonious functional representation. PL representations appear promising in this regard as well. The numerical results show that when the PL takes advantage of the fact that it is approximating a homeomorphism, it is able to achieve on average an order of magnitude lower mean square error on the tested function classes than a neural network with an equal number of parameters.

These results could in part be an artifact of the descent technique, since only simple gradient descent was used. Also, we did not investigate the change in the ratio of MSE_{PL}/MSE_{NN} as the number of parameters given to PL and NN vary. Both the descent technique and the variation of MSE with parameters would be interesting followup work to this study.

Our future work will focus on higher dimensional algorithms for application in the color problem, and the investigation of new descent techniques, including non-gradient methods.

Acknowledgments:
We would like to thank our colleagues at Xerox, Tracy Thieret, L. K. Mestha, and Y. R. Wang, for their encouragement, helpful discussions, and advice concerning this paper.

References

[1] H. R. Kang, *Color Technology for Electronic Imagining Devices*, SPIE Optical Engineering Press, 1997.

[2] J. Z. Chang, J. P. Allebach, and C. A. Bouman, "Sequential linear interpolation of multidimensional functions," *IEEE Transactions on Image Processing*, vol. 92, no. 9, pp. 100, 1997.

[3] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, May 1993.

[4] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, no. 1–5, pp. 11–73, Feb 1997.

[5] D. L. Barrow, C. K. Chui, P. W. Smith, and J. D. Ward, "Unicity of best mean approximation by second order splines with variable knots," *Mathematics of Computation*, vol. 32, no. 144, pp. 1131–1143, October 1978.

[6] R. C. Gayle and J. M. Wolfe, "Unicity in piecewise polynomial L_1-approximation via an algorithm," *Mathematics of Computation*, vol. 65, no. 214, pp. 647–660, April 1996.

[7] Y. Tourigny and M. J. Baines, "Analysis of an algorithm for generating locally optimal meshes for L_2 approximation by discontinuous piecewise polynomials," *Mathematics of Computation*, vol. 66, no. 218, pp. 623–650, April 1997.

[8] C. J. Stone, M. H. Hansen, C. Kooperberg, and Y. K. Truong, "Polynomial splines and their tensor products in extended linear modeling," *The Annals of Statistics*, vol. 25, no. 4, pp. 1371–1470, 1997.

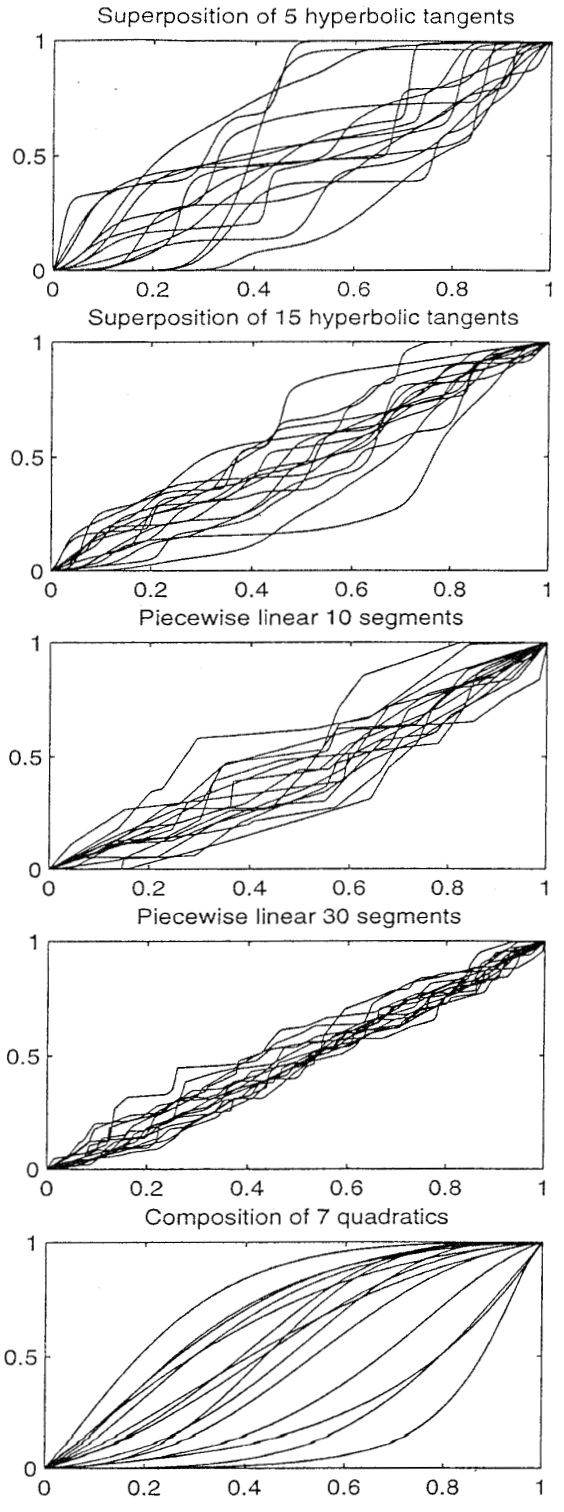

Figure 2: Typical members of the function families

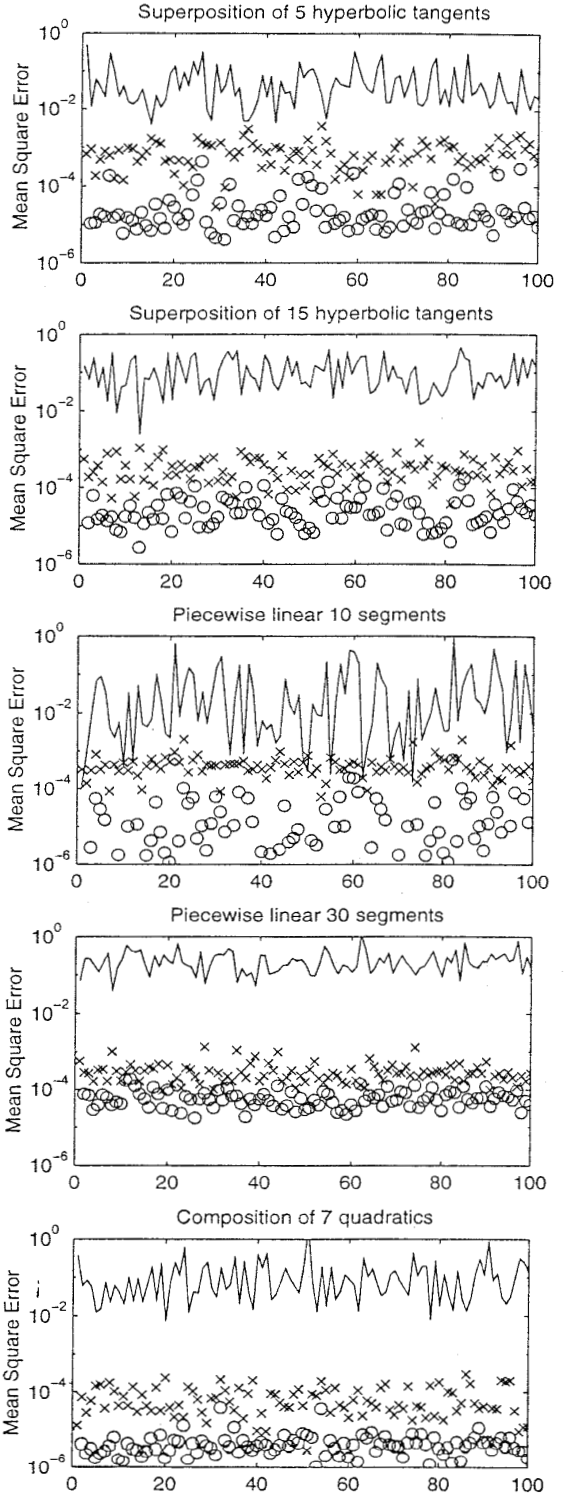

Figure 3: The mean square error attained by NN and PL on each function, as well as the ratio of their MSE, is plotted above. MSE_{PL} o MSE_{NN} x MSE_{PL}/MSE_{NN} —