Function Emulation using Radial Basis Function Networks [1]

SRINIVASA V. CHAKRAVARTHY

Center for Theoretical Neuroscience,

Baylor College of Medicine,

1 Baylor Plaza, Houston, TX 77030.

srini@wallace.neusc.bcm.tmc.edu

JOYDEEP GHOSH

Department of Electrical and Computer Engineering,

The University of Texas, Austin, TX 78712.

ghosh@ece.utexas.edu

Abstract - While learning an unknown input-output task, humans first strive to understand the qualitative structure of the function. Accuracy of performance is then improved with practice. In contrast, existing neural network function approximators do not have an explicit means for abstracting the qualitative structure of a target function. To fill this gap, we introduce the concept of *function emulation*, according to which the central goal of training is to 'emulate' the qualitative structure of the target function. The framework of Catastrophe or Singularity Theory is used to characterize the qualitative structure of a smooth function, which is organized by the critical points of the function. The proposed scheme of function emulation uses the Radial Basis Function Network to realize a modular architecture wherein each module emulates the target function in the neighborhood of a critical point. The network size required to emulate the target in the neighborhood of a critical point is shown to be related to a certain complexity measure of the critical point. For a large class of smooth functions, the present scheme produces a graph-like abstraction of the target, thereby providing a qualitative representation of a quantitative input-output relation.

[1] The work described here was supported in part by the National Science Foundation under grant ECS-9307632 and by AFOSR Contract F49620-93-1-0307.

Keywords -: Function emulation, Catastrophe theory, Radial basis functions, Qualitative learning, Critical points, Function approximation.

1 Introduction

When humans learn a task, it is natural to first get at the qualitative structure of the task, with accuracy improving with practice. For instance, in the initial stages of learning to perform a complex motor task, the amateur is guided by some qualitative rules or 'tips' from the master. Once skill improves by practice and feedback, the amateur-turned-expert need not follow the rules consciously any longer. But it is perhaps impossible for humans to learn most tasks solely by imitating the teacher without the aid of any qualitative rules. For a more direct analogy, consider the case of route description. In most practical situations, one does not need a map correct to the scale to arrive at a destination. It is sufficient to know the major landmarks, decisions to be taken at the cross-roads etc. Once these are fixed other possible variations in path are insignificant.

For concreteness, consider the task of learning to parallel-park a car. In this task, one must control steering angle by taking visual feedback from the surroundings. But guiding oneself solely by error feedback proves to be cumbersome in practice, because it may be difficult to extract precise position information from the available visual input, or to visualize the exact effect of steering movements on one's position. To ameliorate this difficulty certain qualitative information is given to the beginner: at a certain point one must steer to the right, and at a later point to the left. This information gives a priori the *global shape* of the input/output function to be learnt, which makes the task relatively easy.

These general observations can be transferred to the context of applying neural networks to approximate an unknown input-output task from exemplars. Feedforward neural networks typically approximate a target function by minimizing the distance ("error") between the target and the network fit in a suitable metric space of functions. The resulting network solution need not be, in general, *qualitatively* similar to the target function. Consider a simple illustration. A target function (solid lines) and two hypothetical network solutions (dotted lines) are shown in Fig.1a and b. The fit in Fig. 1a has a low mean square error (MSE), and better approximates the target quantitatively. But it is qualitatively dissimilar to the target. The fit of Fig.1b has a larger MSE but is qualitatively similar to the target as both have a single minimum. The latter fit is perhaps more "natural".

How can we formalize the notion of "qualitative" properties of a function? For smooth (differentiable) 1-variable functions one may imagine that the "valleys (x_1^2)" and "hills ($-x_1^2$)" constitute the qualitative properties sought after. But things are already quite complicated in 2-variable functions with "saddles ($x_1^2 - x_2^2$)", "canals (x_1^2)" and a host of other local structural variations. The number of possibilities in higher dimensions is much greater. Also can we be certain that those possibilities are isolated and do not form a continuum? Can one partition a general smooth function ($R^n \to R$) into a small number of qualitatively distinct local structures, and attach distinct labels to each of the parts? Fortunately, it is possible to give a precise form to these issues within the framework of Catastrophe or Singularity Theory (Thom, 1975), (Lu, 1976), (Poston and I.N.Stewart, 1978), (Gilmore, 1981).

Catastrophe Theory (CT) deals with qualitative behavior of dynamic systems governed by smooth "potentials". Naturally, the qualitative behavior of a system is determined by the qualitative structure of the potential, which the theory seeks to study. Further, if the potentials are parameterized by a small number of continuous parameters, CT raises stability issues like: does a small perturbation in parameters change the qualitative structure of the potential? If so, how does it vary? etc. The tools and results of CT can be used to formally characterize qualitative properties of a smooth function.

Based on the concepts of CT, we introduce the notion of *function emulation* using neural networks, wherein the network tries to "emulate" the qualitative properties of the target function. Emphasis is on learning the "shape" of the target; numerical accuracy is given secondary importance. An interesting outcome of such an approach is that the network size directly depends on the qualitative structure of the target. The present approach also make possible a high-level representation of the target in the form of a graph-like abstraction, which is not possible with conventional training of feedforward architectures.

The paper is organized as follows: In Section 2, the relevant concepts and results of CT are sum-

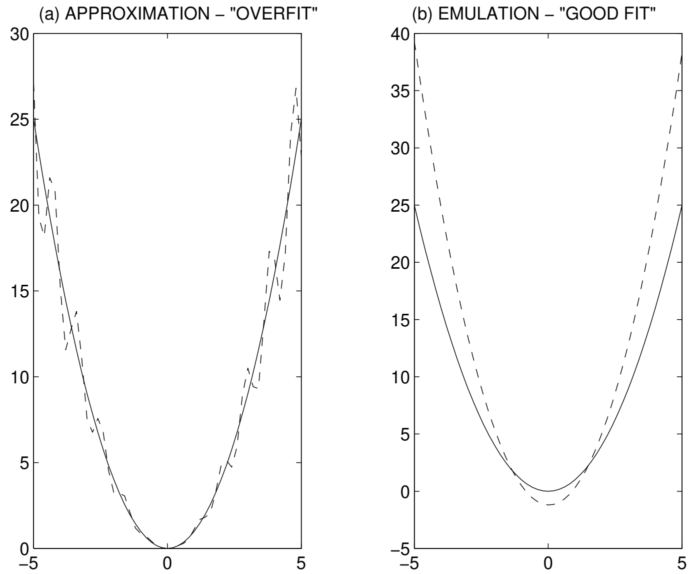

Figure 1: Network solutions obtained for a target function. In Figure (a), network fit has low MSE and is quantitatively accurate, but qualitatively dissimilar. In Figure (b), the network fit is qualitatively similar to the target, - both have a single minimum - but MSE is higher than in (a).

marized. A precise definition of function emulation is given in Section 3. Using CT tools, a training algorithm for target emulation is developed for a specific feedforward network architecture. Empirical studies of the new method are presented in Sections 4 and 5. The present work is discussed, and future directions are indicated in the final section.

2 Elements of Catastrophe Theory

2.1 Critical Points

Catastrophe theory (CT) is mainly concerned with systems whose dynamics can be described by a smooth [2] potential $f(x)$,

$$\frac{d\mathbf{x}}{dt} = -\nabla_{\mathbf{x}} f(\mathbf{x}; \mathbf{c}), \tag{1}$$

where $\mathbf{x} = (x_1, ..., x_n)$ is the vector of *state variables*, $f(x; c)$ is a family of potentials, $\mathbf{c} = (c_1, ..., c_k)$ is the vector of *control parameters*, and '$\nabla_{\mathbf{x}}$' is the gradient operator. Such systems are known as *gradient systems.* Qualitative properties of the system (stable? unstable? how many minima?) can be understood by studying the function in the neighborhood of its *Critical Points.* A smooth function $f(\mathbf{x}) \in R$, $\mathbf{x} \in R^n$, has a **Critical Point (CP)** at $\mathbf{x_0} \in R^n$, if,

$$\nabla f(\mathbf{x})|_{\mathbf{x_0}} = 0. \tag{2}$$

The CP, $\mathbf{x_0}$, is termed to be **non-degenerate** (NDCP) or **Morse** critical point, when, in addition to eqn. (2), the following is also true:

$$det[f_{ij}]|_{\mathbf{x_0}} \neq 0, \tag{3}$$

where, $f_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$ and 'det' denotes matrix determinant. The matrix $[f_{ij}]$ is sometimes called the *stability matrix.* A critical point is **degenerate** (DCP) or **non-Morse** if condition eqn.(3) fails to be true.

[2] By "smooth" we always mean "many times differentiable"

The CPs of a smooth function are important since they organize the global structure of the function. For instance, the overall shape of a smooth univariate function containing two non-degenerate minima, with a maximum in between, can be easily imagined. Further, the degeneracy of a CP tells something about its stability against small perturbations. The NDCP of a function remains an NDCP on a slight perturbation of the function, whereas a degenerate critical point (DCP) may separate into several CPs even on an infinitesimal perturbation.

Example 1: The function $f(x) = x^4$ has a DCP at $x = 0$. A slight perturbation of $f(x)$, $F(x) = f(x) - \epsilon x^2$, where ϵ is a small positive number, has 3 NDCPs at $x = 0, \pm(\epsilon/2)^{1/2}$ respectively.

2.2 Canonical Forms and the Splitting Lemmas

To determine what we mean by local shape of a function, we need an equivalence criterion which determines if two functions have the same shape near a point. Such a criterion allows us to put local shapes into certain equivalence classes. This is achieved by the Splitting Lemmas which state that, in the neighborhood of a point, smooth functions can be put into certain canonical forms after a *diffeomorphism* i.e., a differentiable change of variables, $\mathbf{x} \to \mathbf{x}'$, $\mathbf{x}, \mathbf{x}' \in R^n$, whose general form is given below:

$$x_i' = A_i + \sum_j A_{ij} x_j + \sum_{j,k} A_{ijk} x_j x_k + ... \tag{4}$$

where the Jacobian, $[A_{ij}]$, $A_{ij} = \frac{\partial x_i'}{\partial x_j}$, is non-singular. NDCPs and DCPs have different types of canonical forms. Those that correspond to NDCPs are known as the *Morse forms*, and those corresponding to DCPs are *Thom forms.*

(i) **Morse Forms:** In the neighborhood of a NDCP, a smooth $f(\mathbf{x})$ can be expressed, after a diffeomorphic transformation, in the form,

$$f(\mathbf{x}) \doteq \sum_{i=1}^{n} \lambda_i x_i'^2, \tag{5}$$

where λ_i are the eigenvalues of the stability matrix $[f_{ij}]$ evaluated at the NDCP. None of the eigenvalues vanish since f_{ij} is nonsingular. The symbol $\doteq$ means: "equal to, after a smooth change of variables" (Gilmore, 1981).

(ii) **Thom Forms:** In the neighborhood of a DCP, $f(\mathbf{x})$ can be expressed, after the transformation of eqn. (4), in the form,

$$f(\mathbf{x}) \doteq V_{NM}(x_1'(\mathbf{x}), ..., x_l'(\mathbf{x})) + \sum_{j=l+1}^{n} \lambda_j (x_j'(\mathbf{x}))^2. \tag{6}$$

Here, $f(\mathbf{x})$ is divided into a non-Morse part (V_{NM}) and a Morse part (the rest). In this case, since the matrix $[f_{ij}]$ is singular, one or more eigenvalues must vanish. The number l in eqn. (6), known as **nullity**, denotes the number of vanishing eigenvalues. The Morse part involves the variables, $x_{l+1}', ..., x_n'$, which correspond to non-zero eigenvalues, as in the previous case. The non-Morse part contains higher order (≥ 2) terms involving the remaining variables (Gilmore, 1981).

Eqns. (5, 6) simply say that in the neighborhood of a CP the shape of $f(\mathbf{x})$ is the same as that of the expression on the right side. Near a NDCP, $f(\mathbf{x})$ is of the form of a quadratic form with no cross-terms. Near a DCP, terms of degree higher than 2 appear. So far these results are not surprising, but it turns out that, under certain conditions only a finite number of possibilities exist for the forms of eqns. (5, 6). In multi-parameter families of functions, the types of CPs that appear depend on the *number* of parameters that govern the family. Further CT results state that *in generic (without any special symmetries) families of functions parametrized by a small (≤ 5) number of parameters, a member function locally assumes (is diffeomorphic to) only a finite number of different forms.* Let us consider these issues in greater detail.

2.3 Canonical Forms in Multi-parameter Family of Functions

In a multi-parameter family of functions, $f(\mathbf{x}; \mathbf{c})$, which depends on control parameters, $c_1, ..., c_k$, the eigenvalues of f_{ij} also depend on the control parameters. By deliberate choice of the values of control parameters, it may be possible to make one or more eigenvalues vanish at a CP, making it a DCP. In the neighborhood of that DCP, $f(\mathbf{x})$ can be expressed as before in the form,

$$f(\mathbf{x};\mathbf{c}) \doteq V_{NM}(x'_1(\mathbf{x};\mathbf{c}), ..., x'_l(\mathbf{x};\mathbf{c})) + \sum_{j=l+1}^{n} \lambda_j (x'_j(\mathbf{x};\mathbf{c}))^2. \tag{7}$$

The precise form of V_{NM} depends on the *numbers* l and k. Thus by manipulating the control parameters, the CPs of a function can be put into one of several canonical forms.

It would be helpful, at this point, to give an intuitive picture of how a function can be molded into one of several canonical forms by manipulating the control parameters. For detailed derivations the interested reader may consult (Gilmore, 1981), (Lu, 1976).

Example 2:

(a) Consider a member function of a family of functions, $f_1(x;c)$, $x \in R$, $c \in R$. Further, assuming that $f_1(x)$ has a CP at the origin, consider the Taylor series of $f_1(x)$ around the origin, in order to study the local structure of the function:

$$f_1(x;c) = a_2(c)x^2 + a_3(c)x^3 + ...$$

The constant term is left out since it does not affect the qualitative properties of the function, and the linear term $a_1 = 0$ since the origin is a CP. Now assume that the control parameter, c, can be used to set the earliest term, a_2, to zero. We are then left with,

$$f_1(x) = a_3x^3 + a_4x^4 + ... \tag{8}$$

Applying the diffeomorphism,

$$x' = x(a_3 + a_4x + ...)^{1/3},$$

where $a_3 \neq 0$, on $f(x)$, in eqn. (8) we get,

$$f_1 = x'^3.$$

Thus the entire class of functions in eqn. (8), can be put in the canonical form, $f_1 = x'^3$, by a smooth change of variables, provided $a_3 \neq 0$.

(b) Consider another function, $f_2(x_1, x_2)$ whose Taylor series near origin is:

$$f_2(x_1, x_2) = x_1^2 + x_2^3 + \sum_{p+q\geq 4} a_{pq} x_1^p x_2^q.$$

By the diffeomorphism,

$$x_1' = x_1(1 + a_{40}x_1^2 + a_{31}x_1x_2 + a_{22}x_2^2 + ...)^{1/2},$$

and,

$$x_2' = x_2(1 + a_{13}x_1 + a_{04}x_2 + ...)^{1/3},$$

f_2 can be put in the form $f_2 = x_1^{'2} + x_2^{'3}$ (Lu, 1976).

Therefore, the control parameters can be manipulated to set some of the early terms to zero, in the Taylor series of the function. The resulting Taylor series can be put into a canonical form by a diffeomorphism. This canonical form depends on which of the early terms of the Taylor series are absent.

More complicated forms can be derived from multi-variate functions with multiple control parameters. A remarkable finding of CT is that the canonical forms that occur *generically* (i.e. in multiparameter families without any special symmetries) depend on the *number* of control parameters, k.

Theorem 1 *(Thom) In an open neighborhood of* $\mathbf{x_0} \in R^n$*, and at a fixed value of* $\mathbf{c} = \mathbf{c_0} \in R^k$*,* $f(\mathbf{x}; \mathbf{c_0})$ *can be expressed as,*

$$f \doteq CG_l(x_1', ..., x_l') + \sum_{j=l+1}^{n} \lambda_j x_j^{'2} \tag{9}$$

provided, (i) $k \leq 5$, and (ii) the family $f(\mathbf{x}; \mathbf{c})$ is generic (no special symmetries are present).

The function CG_l, known as the **catastrophe germ**, is a special polynomial function (similar to the Legendre or Hermite polynomials) of l variables. Its specific form depends on the numbers l and k. However, there is a subtlety. The *form* of CG_l corresponding to a given catastrophe is fixed, but not its mathematical realization. For instance, the expressions $x_1^2 - x_2^2$ and x_1x_2 represent the same form (both are diffeomorphic), with different mathematical realizations. Each catastrophe germ represents (is diffeomorphic to) a certain type of DCP (Gilmore, 1981).

Name	k	l	Catastrophe Germ
A_2	1	1	x_1^3
$A_{\pm 3}$	2	1	$\pm x_1^4$
A_4	3	1	x_1^5
D_{-4}	3	2	$x_1^2 x_2 - x_2^3$
D_{+4}	3	2	$x_1^2 x_2 + x_2^3$
$A_{\pm 5}$	4	1	$\pm x_1^6$
D_5	4	2	$x_1^2 x_2 + x_2^4$
A_6	5	1	x_1^7
D_{-6}	5	2	$x_1^2 x_2 - x_2^5$
D_6	5	2	$x_1^2 x_2 + x_2^5$
$E_{\pm 6}$	5	2	$x_1^3 \pm x_2^4$

Table 1: The 11 Catastrophes for $k \leq 5$ and $l \leq 2$.

Table 1 gives a short list of catastrophe germs for $k \leq 5$ and $l \leq 2$ (Gilmore, 1981). Note that more than one form can occur for the same k. For instance, there are three of them for $k = 3$. The symbols A_2, $A_{\pm 3}$ etc. in the first column are notations given to these canonical forms. The number k is known as the *codimension* of the catastrophe germ. It denotes the number of "degrees of freedom" necessary to convert a NDCP into a DCP of desired type. Hence codimension of a germ is a measure of the complexity of the DCP which the germ represents. For a formal treatment of the codimension concept see (Lu, 1976).

For $k > 5$, in general it is not always possible to find canonical forms. This is because when $k > 5$, nullity l *may* exceed 2, in which case canonical forms cannot be derived (Gilmore, 1981). For generic functions l does not exceed 2 for $k \leq 5$.

In this paper, we mainly consider CPs with nullity equal to 1, i.e., catastrophes of the type A_{k+1} in Table 1. The number of RBF nodes (N) required to emulate smooth functions around CPs of type A_{k+1}

will be shown to be related to codimension (k) of A_{k+1}. Preliminary work that suggests existence of a similar result for catastrophes of the type D_{k+1} is also presented. Possible extension to general higher order catastrophes will be discussed in the final section. The immediate task is to apply the knowledge of these canonical forms to construct a qualitative fit to a smooth target.

3 Function Emulation using a Feedforward Neural Network

We may now formulate the concept of **function emulation** whereby the network aims to learn the qualitative features of the target function. We say that a network $f(\mathbf{x})$ emulates a target function $t(\mathbf{x})$ if all of the following conditions are satisfied.

(i) $f(\mathbf{x})$ and $t(\mathbf{x})$ have the same number of CPs.

(ii) Let $\xi_{\mathbf{1}}, ..., \xi_{\mathbf{m}}$ be CPs of the target $t(\mathbf{x})$. There exists an ordering $\zeta_{\mathbf{1}}, ..., \zeta_{\mathbf{m}}$ of the CPs of $f(\mathbf{x})$, such that: $t(\mathbf{x})|_{\mathbf{x}=\xi_{\mathbf{i}}} \doteq f(\mathbf{x})|_{\mathbf{x}=\zeta_{\mathbf{i}}}$ for $i = 1, .., m$.

(iii) Construct two Voronoi tessellations of the input space R^n with ξ_is and ζ_is as centers respectively [3]. If the compartments containing ξ_i and ξ_j are adjacent in the first tessellation, then those containing ζ_i and ζ_j are also adjacent in the second tessellation.

Condition (i) and (ii) ensure that the target and the network fit have the corresponding sets of CPs of same type. Condition (iii) assures that the ordering of the CPs of the target and the network fit in the input space is identical. But where is the guarantee that the CPs of the target function are isolated? There is a known result that for a generic smooth function, the CPs are indeed isolated (see (Gilmore, 1981):607).

The immediate question is: how to produce CPs of the right type in the network function, $f(\mathbf{x})$? To answer this, let us recall from Section 2 that CPs of various types can be produced by manipulating the control parameters in a multi-parameter family of functions. A feedforward network can be viewed as a multi-parameter family, with the learnable parameters, the "weights", as the control parameters.

[3] That is, the partition to which ξ_i is the 'center' is the set of all the points closest to xi_i in the input space.

However, for a general multi-layer feedforward network, the problem of calculating weights such that canonical forms of appropriate type appear in the output, is nearly hopeless. Therefore we consider a specific feedforward architecture, the Radial Basis Function Network (RBFN) (Broomhead and Lowe, 1988), (Moody and Darken, 1989). RBFN architecture is more manageable because the second stage is linear.

The RBFN is a single hidden layer feedforward network with an input-, hidden/kernel- and an output-layer. The output of the ith output node is given as:

$$f_i(\mathbf{x}) = \sum_j w_{ij} g_j(\mathbf{x}), \tag{10}$$

$$g_j(\mathbf{x}) = g\left(\frac{\|\mathbf{x} - \overline{\mathbf{x}}_{\mathbf{j}}\|}{\sigma_j}\right), \tag{11}$$

where $g(\mathbf{x})$ is a radially symmetric function, usually a Gaussian. We call $\overline{x}_j$ the jth centroid, σ_j its width and, w_{ij} the weight connecting the j th kernel/hidden node to the ith output node. For Gaussian RBFs, the width σ_j is the standard deviation.

Our approach to function emulation is now described. The RBFN is regarded as a multi-parameter family of functions with some of the learnable parameters as the control parameters. Aided by the CT results of Section 2, we describe a way of producing CPs of desired types in RBFN output by manipulating the control parameters. The number of parameters (nodes) required to produce a CP of a given type will be determined. However, since the canonical forms of Section 2 are valid only in the neighborhood of a point, it is more straightforward to use a separate network for each CP in the target. Thus we fit the target in a piecewise manner, with a separate network or a *module* for each CP. Note that the goal of function emulation is categorical; there is no such thing as a better or a worse fit, there are only right or wrong fits.

To determine what type of CPs appear in the output of a RBFN, we must ask first (i) how many learnable parameters are present in the network? This is because the number of control parameters determine the canonical forms that occur generically. The second question is: (ii) how to train them?

An N hidden node network with n inputs and a single output has N weights, Nn centroid parameters and N widths - $N(n+2)$ learnable parameters in all. The output of the network is expressed as:

$$f(\mathbf{x}) = \sum_i w_i g(\mathbf{x}; \overline{\mathbf{x}}_\mathbf{i}; \sigma_i), \tag{12}$$

which can be expanded around the origin [4] as,

$$f(\mathbf{x}) = b_0 + \sum_j^n b_j x_j + \sum_j^n \sum_k^n b_{jk} x_j x_k + ... \tag{13}$$

where $\mathbf{x} \in R^n$. The coefficients b_0, b_j etc. are functions of network parameters: $w_i, \overline{\mathbf{x}}_\mathbf{i}$, and σ_i. From eqns.(12) and (13), it is clear that b_0 etc. *linearly* depend on w_i. This can be verified by expanding $g(\mathbf{x}; \overline{\mathbf{x}}_\mathbf{i}; \sigma_i)$s in eqn. (12) as Taylor series and grouping like powers of x. This linear dependence holds good even when $f(\mathbf{x})$ is expanded at any point other than origin.

In Section 2.3, we mentioned that the nature of a CP depends on which of the early terms of Taylor series of the function at the CP vanish. To produce a CP of given type, near $\mathbf{x_0}$, in network output, the network must be trained such that in the Taylor series expansion around $\mathbf{x_0}$ appropriate early terms vanish. Since the Taylor series coefficients of $f(\mathbf{x})$ depend linearly on w_is, it should be possible, in principle, to set early terms in the Taylor series (13) appropriately, by manipulating w_is. But at present it is not obvious how to train w_is since we have no direct access to the Taylor series coefficients of the target, $t(\mathbf{x})$. Conventional training procedures are not obliged to yield a network solution with the desired properties. In the following section we will formulate training as a Least Squares problem and develop an appropriate training procedure.

3.1 Designing a Training Algorithm

In the present approach, the target function is emulated in a modular fashion in the neighborhood of its CPs. To construct such a fit, the training data is partitioned, into several non-overlapping sets,

[4] Taking the analysis to the origin is only to simplify expressions, it does not restrict the final outcome.

Q_s, each covering the neighborhood of a CP. In this section only training of individual modules is considered. Data partitioning, and integration of modules will be considered in a later section. Once the partitioned data are obtained, each subset is used for training a separate module. Since the data in each partition Q_s is taken from the neighborhood of a single CP, we assume that the data in Q_s can be approximated by the Taylor series of the target $t(\mathbf{x})$, expanded around that CP upto a finite order. The truncated Taylor series is usually a low-order polynomial, which has a CP of the same type as the target function in the same neighborhood. Hence each RBFN module is required to approximate a low-order polynomial. But it might seem that a large number of locally-tuned Gaussian RBFs are required to approximate a polynomial. However, since σ controls local sensitivity of a RBF, we will see how it can be manipulated to construct good polynomial approximations.

Consider a univariate target function. Width (σ) is assumed to be the same for each of the N RBFs. Now express the radial functions g_i as $g_i(x) = \sum_{j=1}^{M} a_{ij} x^j$, using Taylor series expansion, such that M is large enough for the expansion to be sufficiently accurate in the neighborhood Q_s, and for a permissible range of σ. Construct a $(M \times N)$ matrix A of the coefficients of g_is such that $A_{ij} = a_{ji}$, where N is the number of RBF nodes. Let $Q_s = \{x_1, x_2, ..., x_d\}$, where d is the number of data points in Q_s, $z(x) = [1, x, x^2, ..., x^M]^T$, and, $Z = [z(x_1), ..., z(x_d)]^T$. Let the target function $t(x)$ be locally represented by a pth degree polynomial $t(x) = z(x)^T b$, where $b = [b_0, b_1, ..., b_p, 0, 0...]^T$, $b \in R^M$.

Training determines 3 sets of parameters: centroids, widths and weights. Centroids are determined using a standard clustering algorithm. Keeping σ fixed, weights (w) are determined by solving the following Least Squares problem,

$$min\|ZAw - Zb\|. \tag{14}$$

From the basic properties of matrix norm, we know that

$$0 \leq \|ZAw - Zb\| \leq \|Z\|\|Aw - b\|.$$

The above inequality becomes tighter and tighter as $\|Aw - b\| \to 0$. We will show that this can be nearly achieved if A satisfies certain conditions. For the moment, we consider the following Least

Squares problem:

$$min\|Aw - b\|. \tag{15}$$

Depending on what A is, we have three cases.

Case (i): A is a square non-singular matrix.

In this case, the solution to the above problem is,

$$w_{sol} = A^{-1}b.$$

Network output $f(x)$ is then expressed as,

$$f(x) = z(x)^T Aw_{sol} = z(x)^T(A^{-1}A)b = z(x)^T b = t(x).$$

We obtain an exact polynomial fit to the data, but a large number of nodes N $(= M)$ are required since A is a square matrix. Therefore, it is desirable to look for a good solution when A is rectangular.

Case (ii): A is a rectangular matrix and A^TA is nonsingular.

If A^TA is nonsingular, the solution is,

$$w_{sol} = (A^TA)^{-1}A^Tb.$$

where $(A^TA)^{-1}A^T$ is termed the *pseudo-inverse* of A (Golub and Van Loan, 1989). Network output $f(x)$ is,

$$f(x) = z(x)^T Aw_{sol} = z(x)^T(A(A^TA)^{-1}A^T)b \equiv z(x)^T Cb.$$

Obviously $f(x) = z(x)^T Cb \neq z(x)^T b = t(x)$ for a general A. The equality is given for any b only when $C = I$, which is unlikely.

Case (iii): A^TA is singular.

This case is more interesting and holds the promise of a good solution. If A^TA is singular, there are an infinite number of solutions. Of these the solution vector with the least 2-norm is selected. Problem (15) is then replaced by,

$$min\|Aw - b\| \;\; and \;\; min\|w\|. \tag{16}$$

the solution to which is given as,

$$w = A^+ b. \tag{17}$$

where A^+ is the pseudo-inverse of A and can be obtained using Singular Value Decomposition (Golub and Van Loan, 1989). Network output $f(x)$ is,

$$f(x) = z(x)^T A w_{sol} = z(x)^T (AA^+) b. \tag{18}$$

Now let us take a closer look at the matrix A. The jth column of A has Taylor series coefficients of $g_j(x)$. If $g_j(x)$s are Gaussians, higher order coefficients decrease as inverse powers of the width, σ. For odd m, A_{mj} is proportional to $O(\frac{1}{\sigma^{m+1}})$; for even m, A_{mj} varies as $O(\frac{1}{\sigma^m})$. Therefore, A has smaller and smaller entries in its lower rows. For large values of σ, this tends to even make A rank deficient, with rank usually decreasing in steps as σ is increased. For such an A, it turns out that AA^+ has the following approximate form:

$$AA^+ \approx \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}, \tag{19}$$

where $rank(A) = r$. The proof of this claim is given in Appendix A.

When AA^+ is of the above form, and rank $r = p+1$, where p is the degree of the target polynomial, $t(x)$, the network output is,

$$t(x) = z(x)^T (AA^+) b \approx \sum_{i=1}^{r} b_i x^i, \tag{20}$$

provided the number of RBF nodes, $N \geq p+1$. Then, $\|Aw - b\| \to 0$, and therefore $\|ZAw - Zb\| \to 0$. Therefore, training aims to determine the value of σ for which $r = p+1$, and error is nearly zero.

Based on the above analysis, a multi-step training procedure can be formulated for a given partitioned data set, Q_s.

- Step 1: Train the centroids using a standard clustering procedure e.g. the k-means algorithm.
- Step 2: Keep the widths of the RBFs the same ($\sigma_i = \sigma$), or scale them using a single parameter, κ, as: $\sigma_i = \kappa d_i$, where d_i is the distance between the ith centroid and the centroid nearest to, and different from, the ith centroid.

- Step 3: Compute weights, w_i, using Singular Value Decomposition.
- Step 4: Calculate squared output error, E.
- Step 5: Update, σ (or κ) as: $\sigma = \beta\sigma$, where $\beta > 1$. If $\sigma > \sigma_{max}$ TERMINATE, else go to Step 3.

The desired network solution is obtained at the value of σ (or κ) that globally minimizes $E(\sigma)$.

4 Numerical Studies

In this section, we apply the training method of the previous section to emulate functions with a single CP. The number of nodes required for emulating NDCPs and DCPs of type A_{k+1} is determined. For these cases, the network size is shown to be related to codimension. Preliminary experiments suggesting a similar relation for higher catastrophes (D_{k+1}, E_6 etc.) are also conducted. A method for finding the desired network size when the type of target CP is unknown is also described.

To emulate a smooth function $t(\mathbf{x})$, near a NDCP, it is sufficient to approximate its Taylor series upto the quadratic terms. If $\mathbf{x} \in R^n$, then 1 (constant) $+n$ (linear) $+n(n+1)/2$ (quadratic) $= (n+1)(n+2)/2$ terms must be approximated, which requires $N = (n+1)(n+2)/2$ RBF nodes. Since DCPs are more interesting these will be studied now with examples.

4.1 Critical Points of A_{k+1} type

CPs of the type A_{k+1} occur in multivariate functions when nullity equals 1. In univariate functions, all CPs are of the type A_{k+1}, since l cannot exceed 1. For CPs of type A_{k+1} it is possible to relate the required network size with the codimension (k).

If the network output $f(x)$, $x \in R$, has a CP of type A_{k+1} at x_0, the Taylor series of $f(x_0+u)$ at x_0 is of the form $p_{k+2}u^{k+2} + p_{k+3}u^{k+3} + ...$, where $p_{k+2} \neq 0$. To put $f(x)$ in this form, $k+2$ terms ($p_0, ..., p_{k+1}$) must be set to 0, and p_{k+2} must be approximated at least correct to the sign. The codimension of A_{k+1} is k. Therefore, for univariate functions the network size (N) required to emulate a DCP of type A_{k+1}

is:

$$N = k + 3 = codim + 3. \tag{21}$$

A similar formula can be derived for multi-variate functions with type A_{k+1} DCPs. We count the number of terms to be set in the Taylor series of $f(\mathbf{x})$, $\mathbf{x} \in R^n$: 1 constant term, n linear terms (to be set to 0), $n(n+1)/2$ quadratic terms of the symmetric stability matrix $[f_{ij}]$, and *codim* number of higher order terms along one of the transformed coordinates. This totals to: $1+n+n(n+1)/2+codim$, yielding:

$$N = (n+1)(n+2)/2 + codim. \tag{22}$$

Verify that, for $n = 1$, the formula reduces to eqn. (21).

Numerical studies consider two univariate functions and a bivariate function.

(i)The first target function is the Logistic Map given by,

$$x(t+1) = 4rx(t)(1 - x(t)).$$

The map produces a complicated time series and exhibits period-doubling and chaos as r is increased. We fix r at 1 for simulations. Being a univariate quadratic function, the map has a single Morse critical point. Hence the codimension of the CP is 0. Since there is only one CP the data set need not be partitioned. The entire data set can be used to train a network. Invoking eqn. (21), the number of RBF nodes required to fit the map equals $3+0 = 3$. Network fit for 3 different cases will be considered.

Case (a): (# RBFs = 3): A RBFN with 3 nodes in the hidden layer is trained as described in Section 3.1 for an entire range of values of σ between 0.1 and 10^4. The mean squared output error (E) is calculated over a grid of 60 points uniformly spaced over the logspace of σ in the interval $[10^{-1}, 10^5]$. The fit obtained at $\sigma = 0.379$ is shown in Fig. 2a typifies a solution obtainable using a 3-node RBFN. However, the fit obtained at $\sigma = 10^3$, a value that minimizes E at 0.000348 is shown in Fig. 2b. The fit is almost indistinguishable from the original map. This is interesting since the prescribed network size assures a mere 'emulation' of the smooth target function. In contrast, the best fit obtained with 2 hidden nodes reveals a sharp deterioration in performance (figure not shown). It may be noted that,

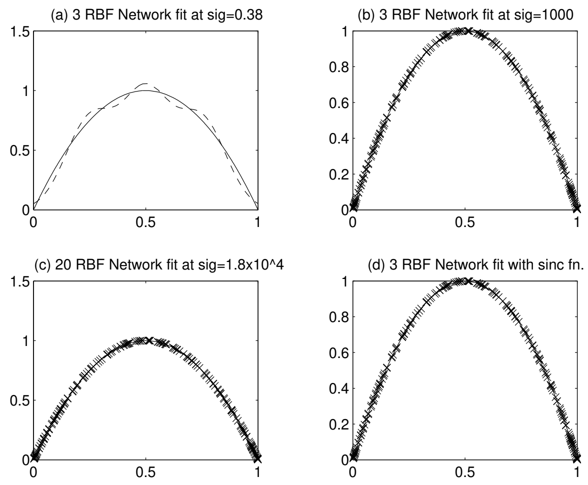

Figure 2: Network solutions obtained for the Logistic Map. In Figure (a), the network fit (dashed line) obtained, at $\sigma = 0.38$, and 3 RBF nodes, is superimposed on the original map (solid line). In Figure (b), the network solution ('+'s) obtained, at $\sigma = 10^3$, and 3 RBF nodes is shown. In figure (c), the network fit ('+'s) obtained, at $\sigma = 1.83 \times 10^4$, and 20 RBF nodes is shown. Figure (d) depicts the network solution obtained at $\sigma = 10^3$, using 3 Sinc RBFs.

for the same task using RBFN, Broomhead and Lowe ((Broomhead and Lowe, 1988)) used used $O(100)$ nodes, and Kadirkamanathan *et al* ((Kadirkamanathan et al., 1991)) used a network of 8 hidden nodes.

Case (b) - (#RBFs = 20): A RBFN with 20 hidden nodes is trained as before. The minimum value of E is obtained at $\sigma = 1.83 \times 10^4$, at which the network fit is a near-perfect match to the target map (Fig. 2c). It is remarkable that even with such a large difference in network size, the fits obtained in Cases (a) and (b) are nearly identical.

Case (c) - (Other RBFs): Note that our requirement of $n^2 + n + 1 + codim$ nodes did not assume Gaussian RBFs. In theory, the basis function need not be gaussians, not even radially symmetric. With the *sinc* function,

$$R_j(x) = sinc(\frac{x - x_j}{\sigma}),$$

as the RBF, the results of the previous cases are reproduced using a network of 3 hidden nodes (Fig.

2d). Similar results (figure not shown) are obtained using the *inverse quadratic* function,

$$R_j(x) = 1/(1 + \frac{(x - x_j)^2}{\sigma^2}).$$

(ii) The second function is a higher degree polynomial: $y_2(x) = x^4 - 0.1x^5 + 0.02x^6$ over the interval $[-3, 5]$.

The function $y_2(x)$ has a single real CP of the type A_3 at the origin, since the leading term is quartic. Codimension of the germ A_3 is 2. Therefore, from eqn. (21) it requires $3 + 2 = 5$ RBF nodes for emulating $y_2(x)$. However, information about the CP type may not be always available to predetermine the network size (N). In this example we assume no prior knowledge except that it has only one CP in the interval of interest. Our task is to construct a network that emulates $y_2(x)$.

We train networks of various sizes starting from a small number.

$N = 3$: The steps of training procedure of Section 4.1 are executed. The error $E(\sigma)$ is plotted in Fig. 3. The flat region in $E(\sigma)$ for $\sigma > 10^3$ corresponds to $rank(A) = 2$, where A is as defined in Section 3. A minimum error of = 68.56 is obtained at $\sigma = 10^3$, at which $rank(A) = 3$. $Rank(A)$ changes suddenly at the point of minimum error. At $\sigma = 10^3$, when $rank(A)$ drops to 2 the network switches to a linear fit; hence the sharp change in error. As the minimum error is still too high, we consider a larger network.

$N = 5$: The error plot for this case is shown in Fig. 4. The minimum error of 8.47 is obtained at $\sigma = 45.2$, beyond which there are 3 nearly flat regions corresponding to $rank(A) = 4, 3$ and 2 respectively. $Rank(A) = 5$ at the minimum point. Note that the minimum error is significantly smaller than the previous case.

$N = 7$: The error plot is shown in Fig. 5. The minimum error is 3.0, obtained at $\sigma = 17.43$, where $rank(A) = 7$. At this point the network generates a 6th degree fit to the target, and naturally the error is low. There is another local error minimum (= 7.9) at $\sigma = 92.37$, where $rank(A) = 5$. There are many sharp bends in the error plot, all of which correspond to change of $rank(A)$.

Finding desired network size with no prior knowledge of CP type: Minimum error on the

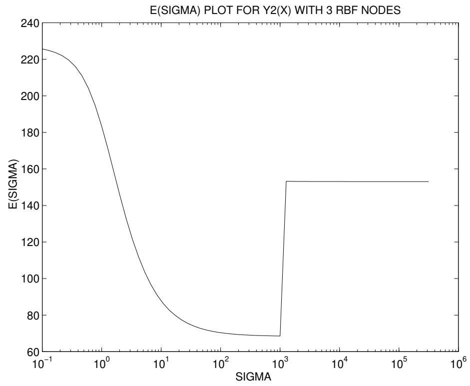

Figure 3: Error vs. σ plot for the target function $y_2(x) = x^4 - 0.1x^5 + 0.02x^6$ trained on a network of 3 RBFs. Error is minimum at $\sigma = 10^3$.

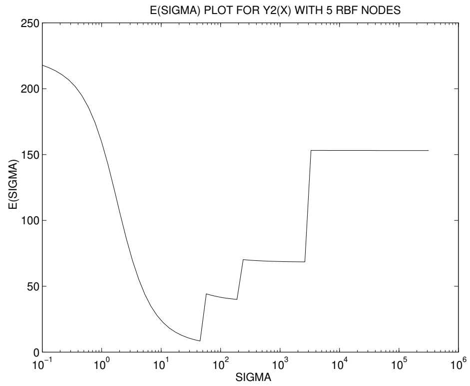

Figure 4: Error vs. σ plot for the target function $y_2(x) = x^4 - 0.1x^5 + 0.02x^6$ trained on a network of 5 RBFs. Error is minimum at $\sigma = 45.2$.

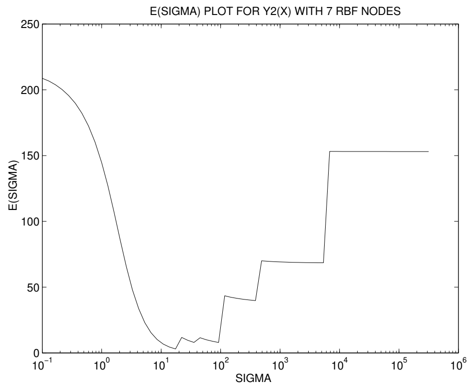

Figure 5: Error vs. σ plot for the target function $y_2(x) = x^4 - 0.1x^5 + 0.02x^6$ trained on a network of 7 RBFs. Error is minimum at $\sigma = 17.43$.

training data would certainly decrease further if a few more RBF nodes are used. But our primary interest is to generate a fit that emulates the target, and not one that minimizes quadratic error. We must make use of error information to determine the network size *sufficient* for emulation. To this end, we have drawn a plot of the minimum error vs. the number of nodes (N), generated by varying σ for each N as above (Fig. 6). Minimum error decreases monotonically with increasing N, but note that the change is steep up to the point $N = 5$; rate of change is much lower beyond that point. There is a sharp transition at $N = 5$. This is the network size sufficient for target emulation. Using several examples of target functions with known qualitative properties, we will verify that *this rule gives the network size sufficient for target emulation.*

(iii) $f(x_1, x_2) = x_1^2 x_2 + x_2^3/3 + x_2^2/2$.

This function is chosen as a non-trivial example that illustrates some of the main strengths of CT. The function $f(x_1, x_2)$ has a complicated root structure (the set of x_1 and x_2 for which $f(x_1, x_2) = 0$). Expressing $f(x_1, x_2)$ in a convenient form, it is easy to see that the solution of,

$$f(x_1, x_2) = x_2(x_1^2 + (x_2 + 3/4)^2/3 - 3/16) = 0,$$

consists of a circle $(x_1^2 + (x_2 - 3/4)^2/3 - 3/16 = 0)$ touching a line $(x_2 = 0)$. A network that emulates

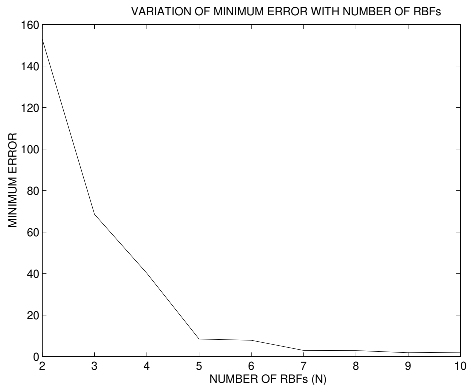

Figure 6: Minimum Error vs. N for the target function $y_2(x) = x^4 - 0.1x^5 + 0.02x^6$. Note that minimum error does not decrease appreciably beyond $N = 5$.

$f(x_1, x_2)$ near origin must also possess an identical root structure near origin whatever be its behavior away from it.

The function $f(x_1, x_2)$ can be put in the form $f \doteq f'(x_1, x_2) = -x_1^4 + x_2^2$, by a smooth change of variables (see Chapter 23 in (Gilmore, 1981) for proof). f' has a Morse part (x_2^2) and a non-Morse part ($-x_1^4$). Since the non-Morse part involves only 1 variable, $f(x_1, x_2)$ has a CP of the type A_{k+1}. However, this is not obvious from the mathematical form of the original function, f. The non-Morse part ($-x_1^4$) is of type A_{-3}, whose codimension is 2. Since codimension does not change by a diffeomorphism, codimension of f is 2.

To find the number of nodes required for emulating $f(x_1, x_2)$, we invoke the formula (22). For $n = 2$, $N = 6 + 2 = 8$ nodes are required. With 8 nodes, the best fit (Fig. 7a and b) is obtained at an error of 0.553, at $\sigma_1 = 84.83$ and $\sigma_2 = 227.58$. Minimum error is much higher (> 5.0) when 7 nodes are used.

4.2 Critical Points of D_{k+1} Type

CPs of D_{k+1} type may occur in n-variable smooth functions when $n \geq 2$. For CPs of type D_{k+1}, the relation between codimension and the desired network size is not clear. Therefore, using 2-variable germ functions of type D_{k+1} (Table 1) with known codimension, we determine the minimum network size

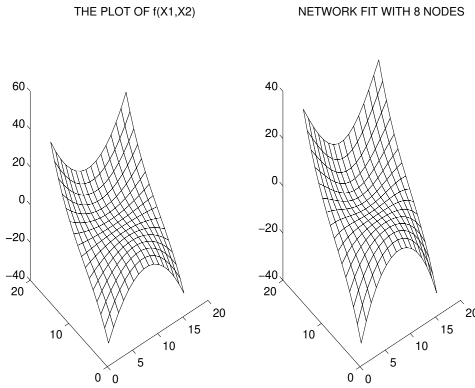

Figure 7: In Figure (a), the original function $f(x_1, x_2) = x_1^2 x_2 + x_2^3/3 + x_2^2/2$ is depicted, and in figure (b) the network fit obtained at $\sigma_1 = 84.83$ and $\sigma_2 = 227.58$, using 8 RBF nodes.

(N) as we have done above with 1-variable example. The objective is to see if there is a simple relation of the form $N = codim+$ a constant integer.

For 2-variable functions also the training procedure of Section 3.1 is followed, but with a difference. In place of RBFs we use Elliptic Basis Functions (EBFs) given as,

$$g_j(\mathbf{x}) = e^{-\sum_m \frac{(x_m - \overline{x}_{im})^2}{\sigma_m^2}}.$$

Search for the two width components, σ_1 and σ_2, is conducted over a grid of 15×15 points laid over logspace of $[10^0, 10^3] \times [10^0, 10^3]$.

(i) $f_1(x_1, x_2) = x_1^2 x_2 + x_2^3/3$ over the domain $[-3, 3] \times [-3, 3]$.

This is the germ function, D_4, (see Table 1) of codimension 3. In order to find the smallest network for emulating this germ function, minimum error (with respect to width) is calculated at network sizes varying between 5 and 12 nodes. Fig. 8 shows that minimum error does not vary appreciably for $N \geq 8$. Using a network of 8 nodes, we obtain minimum error of 0.54, at $\sigma_1 = 84.83$, and $\sigma_2 = 227.58$. The corresponding network fit and the target function are juxtaposed in Fig. 9. It remains to be seen if the putative relation $N - codim = 8 - 3 = 5$ holds good for other functions also.

(ii) $f_2(x_1, x_2) = x_1^2 x_2 + x_2^4$ over the domain $[-3, 3] \times [-3, 3]$.

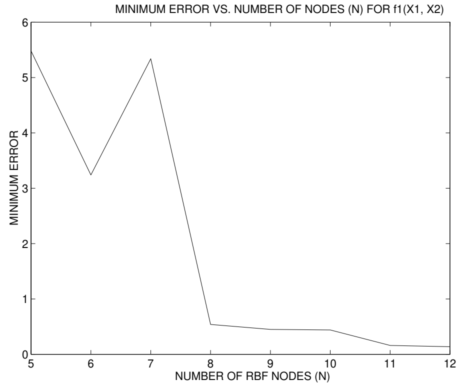

Figure 8: Plot of Minimum Error obtained vs. number of RBF nodes (N) for $f_1(x_1, x_2) = x_1^2 x_2 + x_2^3/3$. Minimum Error is already low for $N = 8$.

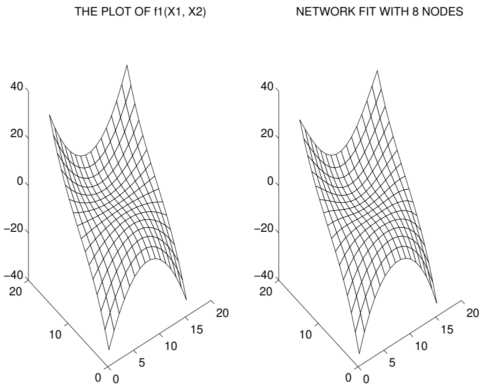

Figure 9: (a) Plot of $f_1(x_1, x_2) = x_1^2 x_2 + x_2^3/3$. $f_1(x_1, x_2)$ has a DCP at origin with codimension 3. (b) Best fit obtained with a network of 8 nodes.

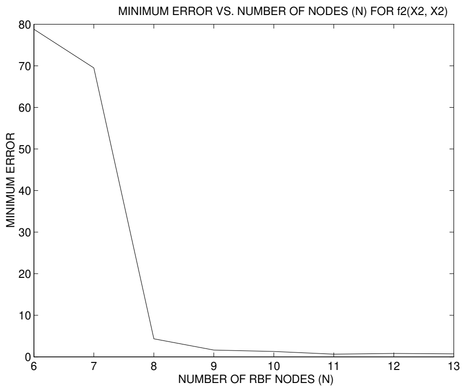

Figure 10: Plot of Minimum Error obtained vs. number of RBF nodes (N) for $f_2(x_1, x_2) = x_1^2 x_2 + x_2^4$. Minimum Error is already low for $N = 9$.

This is the germ function, D_5, (see Table 1) of codimension 4. Again minimum error is calculated at network sizes. Fig. 10shows that minimum error does not vary appreciably for $N \geq 9$, validating the empirical relation $N = codim + 5$ again. With a network of 9 nodes, we obtain minimum error of 1.63 at $\sigma_1 = 84.83$, and $\sigma_2 = 7.19$. The corresponding network fit and the target function are juxtaposed in Fig. 11.

(iii) $f_3(x_1, x_2) = x_1^3 + x_2^4$ over the domain $[-3, 3] \times [-3, 3]$.

The codimension of f_3 is 5 (Gilmore, 1981), and therefore 10 RBFs ought to be sufficient for emulation. However, in this case, variation of minimum error with N shows an anomalous behavior (Fig. 12). Error dropped to a low value for $N = 10$ as expected, increased immediately and dropped again at $N = 13$. Using a network of 10 nodes, we obtain minimum error of 0.677, at $\sigma_1 = 84.83$, and $\sigma_2 = 19.31$. The corresponding network fit and the target function are juxtaposed in Fig. 13.

5 Emulating Targets with multiple Critical Points

In general a smooth target function has multiple CPs. For emulating such targets we prescribe a modular architecture in which each module emulates the target in the neighborhood of a CP. The entire

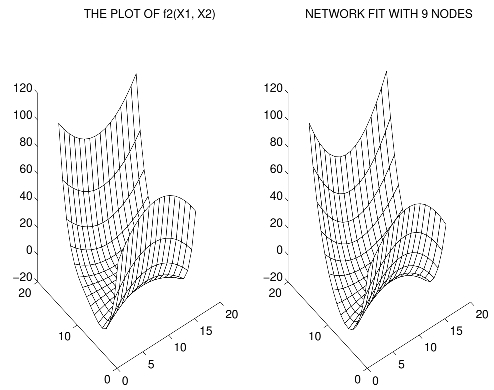

Figure 11: (a) Plot of $f_2(x_1, x_2) = x_1^2 x_2 + x_2^4$. $f_2(x_1, x_2)$ has a DCP at origin with codimension 4. (b) Best fit obtained with a network of $5 + 4 = 9$ nodes.

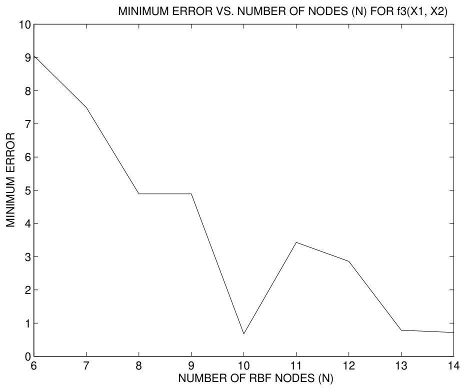

Figure 12: Plot of Minimum Error obtained vs. number of RBF nodes (N) for $f_3(x_1, x_2) = x_1^3 + x_2^4$. Estimated requirement for emulation is $N = codim + 5 = 10$.

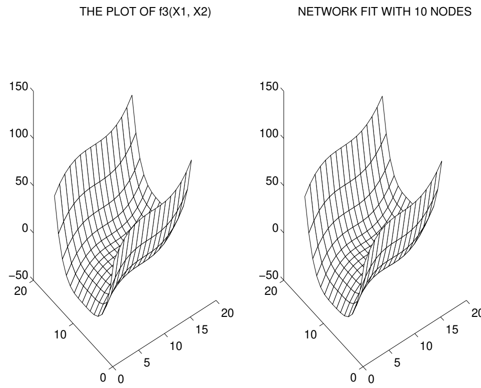

Figure 13: (a) Plot of $f_3(x_1, x_2) = x_1^3 + x_2^4$. $f_3(x_1, x_2)$ has a DCP at origin with codimension 5. (b) Best fit obtained with a network of $5 + 5 = 10$ nodes.

scheme has 4 components:

(i) Find CPs of target function from training data.

(ii) Partition training data Q as $Q = Q_1 \bigcup Q_2 ... \bigcup Q_n$, and $Q_r \bigcap Q_s = \phi$ for $r \neq s$.

(iii) Train module M_s with Q_s.

(iv) Combine modules.

The above subtasks are now described.

(i) *Finding CPs of target function from training data:*

It is perhaps not as easy to find CPs of a function from sampled data, than it is to find it from the function itself. Chances of false alarms are high and usually a large number of samples are required. This, confounded with the "curse of dimensionality", places heavier demands on the sample size requirements. But, given enough sample data, heuristics can always be designed to find CPs from sample data. One such heuristic can be formulated based on a scale-based clustering technique using RBFN developed by us (Chakravarthy and Ghosh, 1994).

Scale-based clustering picks up points of high probability density in the input data set. Exploiting this property of the technique, a suitable procedure for finding CPs can be formulated as follows: Let $y = t(\mathbf{x})$, $y \in R$, $\mathbf{x} \in R^n$, where $t(\mathbf{x})$ is the target function. Form the data set, $S = \{\mathbf{z} : \mathbf{z} = [\mathbf{x_i}, y_i]\}$, where $\mathbf{x_i}$ and y_i are input-output pairs of the training data, and $\mathbf{z} \in R^n \times R$. Cluster centers $\overline{\mathbf{z}}_j$ are initialized by random selection from S. Then the $\overline{\mathbf{x}}_j$ components of $\overline{\mathbf{z}}_j$ are updated as follows:

$$\overline{\mathbf{x}}_j \leftarrow \overline{\mathbf{x}}_j + \sum_{\mathbf{z_p} \in S} \eta(\mathbf{x_p} - \overline{\mathbf{x}}_\mathbf{j}) e^{\frac{-\|\mathbf{z_p} - \overline{\mathbf{z}}_j\|^2}{\delta^2}}. \tag{23}$$

After every step of eqn. (23), $\overline{y}_j$s are updated by a simple assignment:

$$\overline{y}_j \leftarrow y_{j*}, \tag{24}$$

where $\mathbf{z_{j*}} = [x_{j*}, y_{j*}] \in S$, and $\mathbf{x}_{j*}$ is the training input nearest to $\overline{\mathbf{x}}_j$. Eqns. (23, 24) are iterated alternately in every step. The fixed points of eqn. (23) give approximate locations of CPs of the target function.

In this process the scale parameter (δ) plays an important role. For a small scale, a large number of "CPs" are obtained, and for a sufficiently large scale there is only a single CP. The right set of CPs are defined as those that exist over the longest scale interval. These issues are discussed shortly with the help of an example.

(ii) *Partition training data* Q *as* $Q = Q_1 \bigcup Q_2 ... \bigcup Q_n$, *and* $Q_r \bigcap Q_s = \phi$ *for* $r \neq s$.

The partitions Q_s are essentially Voronoi tessellations of the training input data set around the CPs, $\overline{\mathbf{x}}_j$, computed in the previous step.

(iii) *Train module* M_s *with* Q_s*:*

Training of individual modules is already described in Section 3.1.

(iv) *Combine modules:* A simple selection scheme is adopted for module combination. Suppose a novel input $\mathbf{x}$ is presented to the modular network. The module M_s, one of whose centroids is closest to $\mathbf{x}$, compared to all other centroids of all the modules, is selected. Response of M_s is accepted as the output of the network.

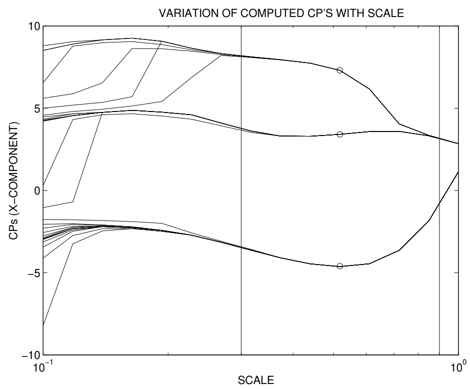

Figure 14: Computing CPs of a target function from sample data. Three branches are stable over the longest interval indicated by the vertical lines. CPs ('o') are taken from the (logarithmic) middle of this interval.

5.1 Numerical Example

A univariate target with 3 CPs is shown in Fig. 15. Circles ('o') indicate locations of CPs. The CP on the left is DCP of type A_2; the central CP is a DCP of the type A_{-3}; on the right there is a simple minimum. The data set D consists of 101 uniformly spaced points on the x-axis. The steps described above are employed to emulate the target.

(i) *Finding CPs of the target:* Eqn. (23) is iterated starting with 30 cluster centers. Scale value (δ) is varied in 15 logarithmic steps between 1 and 10. For each value of δ, variation of the computed CP location is shown in Fig. 14. There are 3 stable branches between $\delta = 0.3$ and 0.9. Final CPs (circles) are computed at a scale value equal to the geometric mean ($= 0.51$) of these two scales.

(ii) *Partitioning Data set Q:* Computed CPs ('x') and actual CPs ('o') are plotted in Fig. 15. The data set Q is partitioned into three intervals Q_1, Q_2 and Q_3, separated by 2 vertical lines. Although the computed CP locations are not very accurate, it will be seen that they are sufficient for good emulation.

(iii) *Training modules:* Following Section 3.1, a separate RBFN is trained on each of the data sets Q_1, Q_2 and Q_3. Since Q_1 contains a DCP of A_2 type, it requires a network of $3 + codim = 3 + 1$ nodes. Similarly Q_2 and Q_3 require 5 and 3 nodes respectively.

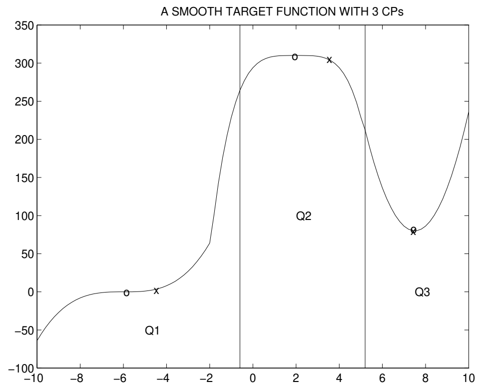

Figure 15: The target function with 3 CPs. Actual CPs ('o') and computed CPs ('x') are depicted. Vertical lines partition the input domain into 3 intervals Q_1, Q_2 and Q_3 each including a CP.

(iv) *Combining Modules:* The trained modules are combined in a piecewise manner (Fig. 16). Over the regions Q_2 and Q_3 the network not only emulates correctly but approximates the target accurately. However, that is not the case in Q_1. Instead of a desired two-fold degenerate CP of the type A_2 the network produced a simple minimum and a simple minimum proximate to each other. Indeed the two could annihilate each other to produce a DCP of the type A_2. The overall fit is also surprisingly smooth near the boundaries. This is only incidental and need not always be the case. However, smoothness can be obtained by use of appropriate interpolation procedures.

6 Discussion

Under certain conditions, Catastrophe Theory provides a classification of Critical Points (CPs) of smooth multivariate functions and hence a basis for function emulation. The present training procedure, designed to achieve this goal using neural networks, is based on the principle that local qualitative structure of a smooth function depends on the early terms of its Taylor series expansion around the CPs. The Taylor series can be truncated at a certain point without disturbing the qualitative structure. The RBFN can be used to generate a sufficiently accurate polynomial fit by manipulating the width (σ) pa-

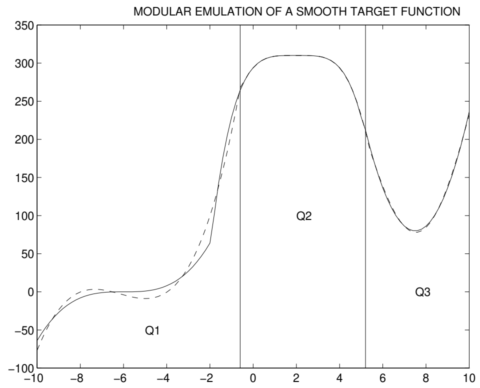

Figure 16: Emulation of the target function with 3 modules.

rameter (Hoskins et al., 1993). Motivated by the receptive field paradigm, some early workers in RBFN architecture have taken the approach of approximating the target piecemeal with locally-tuned RBFs (Moody and Darken, 1989), (Broomhead and Lowe, 1988). Therefore width is usually kept relatively small, requiring a large number of RBF nodes to generate the fit to even simple polynomials over a large input domain. However, Hoskins *et al* demonstrated that accurate, compact fits for polynomials can be obtained if the receptive field approach is dropped and large widths are allowed (Hoskins et al., 1993). Moreover, for a sufficiently large width, and a minimum number of nodes, experiments of Section 4 indicate that the same fit can be obtained irrespective of the number of nodes or the type of the Radial Function. In a sense, the network solution is independent of some of the details of the network, which is a highly desirable network property.

The present emulation scheme is motivated by the observation that humans are adept at extracting qualitative information from quantitative domains. The need to switch between qualitative and quantitative domains often arises in practical applications of neural networks. Two simple mathematical devices allow us to convert the quantitative into the qualitative: (i) thresholding and (ii) order (least, best, median etc). The use of singularities is a third route between the two domains which does not seem to have been well explored in neural network research.

The central aim of function emulation is to generate a *qualitative* representation of the target. The target can be represented as a network of CPs of distinct types. For instance, the multiple CP target function of Section 6 has 3 CPs of the type A_2, A_{-3} and '-' (simple minimum) in that order. Therefore, the function can be represented as the following graph:

$$A_2 - -A_{-3} - --$$

Similar graph representations for general smooth univariate functions consist of chains of nodes where each node represents CP of a distinct type. The graph structures assume more complicated forms in multivariate functions, where higher catastrophes (D_{k+1}, E_6 etc.) may occur.

The graph also tells us what happens to the target, or the network fit, on a smooth perturbation. For instance, a CP of A_{-3} type is three-fold degenerate, which when perturbed might break into a two-fold degenerate CP (A_2) and a non-degenerate CP, or 3 non-degenerate CPs. Thus new nodes might appear in the graph representation on slight perturbation of the network.

The proposed notion of function emulation and the resulting graph-representation of the target, appears to have a cognitive significance. Lieblich and Arbib ((Lieblich and Arbib, 1982)) suggest that in animals cognitive maps of external space takes the form of a directed graph in which each node denotes a recognizable situation or a landmark in the world. Such landmarks can be viewed as CPs of the animal's response to some feature of the world.

In the human context, qualitative and linguistic representations make possible a symbolic-level transfer of knowledge gained about a domain to another agent. The graph structures described above contain symbolic information about the global shape of a function. Note that this information is independent of the architecture or size of the network(s) used to generate it, and therefore can be used to train a second network. This transfer can be done in two ways. (i) For instance assume Network A is trained on Problem-1 and a graph G_1 is generated. Now Network B is also to be trained on Problem-1 but in slightly different circumstances (e.g. data collected at a different temperature, pressure, time of the year etc.), and with fewer training examples. It is plausible that the target functions of the two networks have the same overall shape, but are different in detail. Our approach then enables reliable training of Network B by transfer of symbolic information from Network A. (ii) Although the specific case of RBFN is used in our studies, note that the analysis of Section 3 does not require a Gaussian nonlinearity. In fact, preliminary investigations led us to conclude that the present analysis and calculations can be extended to the Multi-Layered Perceptron (MLP) also. This opens up the possibility of transferring knowledge from a trained RBFN to a MLP and vice versa.

We note that transfer of knowledge and inductive learning is recently attracting strong interest in the machine learning community(Pratt and Thrun, 1997). The transfer mechanisms mentioned above provide a new angle to this problem and are worthy of further pursuit.

The goal of emulating qualitative features of a target is similar, in some respects, to the problem of matching network complexity to target complexity. We have shown that the minimum network size required for emulation is simply related to codimension of the target CP. Codimension is, in a sense, a measure of the "complexity" of a CP. By requiring the network size to depend on codimension, function emulation also satisfies, in a novel way, the goal of matching the network complexity with target complexity. Currently, there are several ways of approaching this problem. A typical approach consists of penalizing larger network sizes by adding a 'regularizing' term to the network performance cost. The regularization term imposes certain smoothness constraints on the fit, and typically favors smaller or less complex networks as they lead to smoother maps (Poggio and Girosi, 1990), (Friedman, 1994). It is difficult to resist comparison of the present scheme with spline models which generate piece-wise, smooth fit to observational data (Wahba, 1990). But our scheme is chiefly distinguished from the above approaches in that it provides a symbolic representation of the target which the other models fail to do.

Our emulation studies at present are largely limited to target functions with CPs of the type A_{k+1}. For multi-variable functions, where higher catastrophes (D_{k+1}, E_6 etc.) may occur, the relation between codimension and network size is not clear. Preliminary experiments involving a few cases, supported by theoretical calculations, indicate that such a relation might exist ($N = codim + 5$) for CPs of type D_{k+1} in 2-variable functions. But it is not certain if the result is valid for all CPs of this type occurring in 2-variable functions. Earlier we stated that the present approach to function emulation is limited to target functions that belong to generic families of functions involving a small number of control parameters ($k \leq 5$). This is because in generic families with $k > 5$, nullity (l) might exceed 2; the DCPs that occur in such families do not have distinct canonical forms describable by a unique symbol. However, even when $k > 5$, if it is guaranteed that $l \leq 2$, two countably infinite sets of canonical forms of type A_{k+1} ($k = 1, 2, ...$), and D_{k+1} ($k = 3, 4, ...$) can be derived ((Gilmore, 1981):32). Remember that the present method places no restriction on the number of state variables. The allowable set of canonical forms are perhaps sufficiently complex to describe a large variety of real-world input-output systems.

At present, determination of CP type of the target rests solely on the relation between codimension and minimum network size. Future work must be aimed at breaking this link for two reasons. Firstly, minimum network size is determined from the point at which there is a sharp drop in minimum error in the plots of minimum error vs. network size. This transition may not always be sufficiently sharp, and small variations in network size might imply a completely different CP type. The second reason is, even if we could determine minimum network size, its relationship to codimension is not yet known for higher catastrophes. And even when codimension is known it is not uniquely related to the type of the catastrophe. However, there seems to be a way out of this problem. There is an algorithm due to Mather (Mather, 1968), (Mather, 1969) (see also Chapter 23, (Gilmore, 1981)), that determines the codimension and type of a function whose Taylor series coefficients are known up to a specified degree. The target CP type can be found from a trained module/network, whose Taylor series coefficients can be automatically calculated from the network parameters. However, the computational feasibility of the approach is yet to be determined.

References

Broomhead, D. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks.

Complex Systems, 2:321–355.

Chakravarthy, S. V. and Ghosh, J. (1994). Scale-based clustering using the Radial Basis Function network. In *IEEE International Conference on Neural Networks*, pages 897–902.

Friedman, J. (1994). An overview of predictive learning and function approximation. In V. Cherkassky, J. F. and Wechsler, H., editors, *From Statistics to neural networks, Proc. NATO/ASI Workshop*, pages 1–61. Springer Verlag.

Gilmore, R. (1981). *Catastrophe Theory for Scientists and Engineers.* Wiley-Interscience, New York.

Golub, G. and Van Loan, C. (1989). *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD.

Hoskins, J. C., Lee, P., and Chakravarthy, S. V. (1993). Polynomial modeling behavior in radial basis function networks. In *Proc. of World Conference on Neural Networks, Portland, OR.*, pages 693–699.

Kadirkamanathan, V., Niranjan, M., and Fallside, F. (1991). Sequential adaptation of Radial Basis Function neural networks. In J.E.Moody, R. and D.S.Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 721–727.

Lieblich, I. and Arbib, M. (1982). Multiple representations of space underlying behavior. *Behavioral Brain Sciences*, 5:627–659.

Lu, Y. (1976). *Singularity Theory and an introduction to catastrophe theory.* Springer-Verlag, New York.

Mather, J. (1968). Stability of C^∞ mappings III. finitely determined map germs. *Publ. Math. IHES*, 35:127–156.

Mather, J. (1969). Stability of C^∞ mappings IV. classification of stable germs by r-algebras. *Publ. Math. IHES*, 37:223–248.

Moody, J. and Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294.

Poggio, T. and Girosi, F. (1990). Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–97.

Poston, T. and I.N.Stewart (1978). *Catastrophe theory and its Applications.* Pittman, London.

Pratt, L. and Thrun, S. (1997) Guest Editors. *Special issue on inductive transfer. Machine Learning.*

Thom, R. (1975). *Structural Stability and Morphogenesis.* Reading:Benjamin.

Wahba, G. (1990). *Spline Models for Observational Data.* Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania.

Mathematical Appendix

We sketch below a proof that AA^+ has the following form:

$$AA^+ \approx \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}$$

where $rank(A) = r$ and A is the coefficient matrix defined in Section 3.

Proof:

To understand the solution obtained, it is obvious that some knowledge of the matrix AA^+ is required.

To compute the pseudo-inverse, A is expressed as a product of 3 matrices as,

$$A = PDQ \tag{25}$$

where P and Q are orthogonal matrices. D has the form,

$$D = \begin{bmatrix} D_1 & 0 \\ 0 & 0 \end{bmatrix}$$

with $D_1 = diag(s_1, s_2, ..., s_r)$. Eqn. (25) is referred to as the *Singular Value Decomposition (SVD)* of A. The quantities s_i are the non-zero *singular values* of A, and $rank(A) = r$. The interested reader may consult original sources for other properties of SVD (Golub and Van Loan, 1989). Given SVD of A, A^+ is calculated as,

$$A^+ = Q^T D^+ P^T \tag{26}$$

We know that elements of A depend on the width, σ with A_{ij} varying as $O(1/\sigma^{i+1})$ for odd i and as $O(1/\sigma^i)$ for even i. That is, A has smaller and smaller entries in its lower rows. Larger values of σ tend to make A rank-deficient, the rank decreasing with inceasing σ. If $rank(A) = r$, then A can be approximated by a matrix A_r of the form,

$$A_r = \begin{array}{cc} \begin{bmatrix} T & S \\ 0 & 0 \end{bmatrix} & \begin{array}{c} r \\ M-r \end{array} \\ \begin{array}{cc} r & N-r \end{array} & \end{array}$$

Computing SVD for A_r as,

$$A_r = U_r S_r V_r^T$$

it can be shown that the orthogonal matrix U will have the form,

$$U_r = \begin{array}{cc} \begin{bmatrix} w_{11} & 0 \\ 0 & 0 \end{bmatrix} & \begin{array}{c} r \\ N-r \end{array} \\ \begin{array}{cc} r & M-r \end{array} & \end{array}$$

Now let $A = A_r + E_a$, where $\|E_a\|$ is a small number. Taking SVD of A, as $A = USV^T$, U can be expressed as $U = U_r + E =$

$$\begin{bmatrix} w_{11} + E_{11} & E_{12} \\ E_{21} & I_{n-r} + E_{22} \end{bmatrix}$$

where $\|E\|$ too is a small quantity. Since U is orthogonal, we have the following conditions on E:

$$\begin{aligned}
(w_{11}+E_{11})(w_{11}+E_{11})^T + E_{12}E_{12}^T &= I_r, \\
E_{21}(w_{11}+E_{11})^T + (I_{n-r}+E_{22})E_{12}^T &= 0, \\
(w_{11}+E_{11})E_{21}^T + E_{12}(I_{n-r}+E_{22})^T &= 0, \\
E_{21}E_{21}^T + (I_{n-r}+E_{22})(I_{n-r}+E_{22})^T &= I_{n-r}.
\end{aligned} \tag{27}$$

Now consider AA^+. From SVD we know that $A^+ = VS^+U^T$, $AA^+ = U(SS^+)U^T$, where,

$$SS^+ = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}$$

Or,

$$AA^+ = (U+E)\begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}(U+E)^T$$

$$= \begin{bmatrix} (w_{11}+E_{11})(w_{11}+E_{11})^T & (w_{11}+E_{11})E_{21}^T \\ E_{21}(w_{11}+E_{11})^T & E_{21}E_{21}^T \end{bmatrix}$$

Substituting eqns. (27),

$$AA^+ = \begin{bmatrix} I_r - E_{12}E_{12}^T & -E_{12} - E_{12}E_{22}^T \\ -E_{12}^T - E_{22}E_{12}^T & E_{21}E_{21}^T \end{bmatrix} \approx \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \tag{28}$$