



ELSEVIER

Contents lists available at ScienceDirect

## Neural Networks

journal homepage: [www.elsevier.com/locate/neunet](http://www.elsevier.com/locate/neunet)

2015 Special Issue

## Smart sampling and incremental function learning for very large high dimensional data

Diego G. Loyola R\*, Mattia Pedernana, Sebastián Gimeno García

German Aerospace Center (DLR), Oberpfaffenhofen, 82234 Wessling, Germany

## ARTICLE INFO

## Article history:

Available online xxxx

## Keywords:

High dimensional function approximation  
 Sampling discrepancy  
 Design of experiments  
 Probably approximately correct  
 computation  
 Function learning  
 Neural networks

## ABSTRACT

Very large high dimensional data are common nowadays and they impose new challenges to data-driven and data-intensive algorithms. Computational Intelligence techniques have the potential to provide powerful tools for addressing these challenges, but the current literature focuses mainly on handling scalability issues related to data volume in terms of sample size for classification tasks.

This work presents a systematic and comprehensive approach for optimally handling regression tasks with very large high dimensional data. The proposed approach is based on smart sampling techniques for minimizing the number of samples to be generated by using an iterative approach that creates new sample sets until the input and output space of the function to be approximated are optimally covered. Incremental function learning takes place in each sampling iteration, the new samples are used to fine tune the regression results of the function learning algorithm. The accuracy and confidence levels of the resulting approximation function are assessed using the probably approximately correct computation framework.

The smart sampling and incremental function learning techniques can be easily used in practical applications and scale well in the case of extremely large data. The feasibility and good results of the proposed techniques are demonstrated using benchmark functions as well as functions from real-world problems.

© 2015 The Authors. Published by Elsevier Ltd.

This is an open access article under the CC BY license  
 (<http://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Computer-based simulations of tremendously complex mathematical systems describing multifaceted physical, chemical, dynamical and engineering models are usually associated with very expensive costs in terms of processing time and storage. Complex mathematical models are present in a wide variety of scientific areas such as the simulation of atmospheric processes in numerical weather prediction (Han & Pan, 2011; Hsieh & Tang, 1998; Lynch, 2006; Morcrette, 1991), climate modeling (Flato et al., 2013), (Gordon et al., 2000), chemical transport (Grell et al., 2005), (Menut et al., 2013), radiative transfer (Gimeno García, Trautmann, & Venema, 2012) and large eddy simulations (Sagaut, 2006). Other scientific disciplines such as genetics, aerodynamics, or statistical mechanics also make use of highly complex models. The input space of these models can be of high dimensionality with hundreds or more components. The usage of more realistic models usually introduces new dimensions leading to an exponential increase in volume, i.e. "Big Data" (Hilbert & López, 2011; Lynch, 2008).

The speeding-up of such models is a crucial problem in practical applications like weather forecasting, remote sensing, and climate modeling among others. These complex models can be approximated using for example neural networks (Haupt, Pasini, & Marzban, 2009; Loyola, 2006; Qazi & Linshu, 2006), support vector machines (Tripathi, Srinivas, & Nanjundiah, 2006), kernel smoothing models (Cervellera & Macciò, 2013), and the resulting computational intelligence systems are being deployed in operational environments, see for example Krasnopolsky and Chevallier (2003) and Loyola (2006).

Related work found in the literature usually addresses function approximations from complex mathematical models in the general framework of statistical machine learning where it is assumed that the training samples are created independently according to an unknown probability density function (Krasnopolsky & Schiller, 2007). However, having a mathematical model to parameterize, we can freely choose the sample points that better characterize the input and output spaces of the model. The selection of samples to be used in function approximation problems is less explored in the literature, sometimes it is called design of experiments (Sacks, Welch, Mitchell, & Wynn, 1989) or active learning (Enăchescu, 2013).

In this study we develop a general function approximation framework in which the choice of the samples describing the

\* Corresponding author. Tel.: +49 8153 28 1367; fax: +49 8153 28 1446.

E-mail address: [Diego.Loyola@dlr.de](mailto:Diego.Loyola@dlr.de) (D.G. Loyola R).<http://dx.doi.org/10.1016/j.neunet.2015.09.001>0893-6080/© 2015 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

function is an integral part of the learning problem. The goal is to find a regression model that accurately approximates a function  $f : X \rightarrow Y$  with input  $X \subset \mathbb{R}^n$  and output  $Y \subset \mathbb{R}^m$  from a set of training samples  $T = \{x_i, y_i\}$  for  $i = 1$  to  $s$ , subject to the constraint of minimizing the number of calls to the target function  $f$ , i.e. minimizing the number of samples  $s$ . The dimensionality of the data needed to solve this problem is characterized by the number of samples  $s$  and the dimensionality of the mapping represented by  $f$  with  $n$  inputs and  $m$  outputs.

On one side, the complexity of a function does not necessarily increase for a high dimensional input space; moreover the limiting factor for an accurate function approximation is the intrinsic complexity of the target function and not the dimension of the input data space (Kolmogorov, 1957). Accuracy of approximation can be achieved in high dimensional cases for target functions with lower complexity (Gnecco, Kůrková, & Sanguineti, 2011). On the other side, systematically sampling the input space with  $k$  values per dimension will require in total  $s = k^n$  calls to  $f$ , therefore the number of samples  $s$  for mappings with high dimensional input size grows exponentially with the input dimension  $n$ .

These difficulties are usually referred as “the curse of dimensionality” problem (Vapnik, 2006) and it has severe consequences not only for the time needed to create the training dataset but also for the algorithms needed to solve function approximation problems that must deal with very large number of samples of high dimensionality, i.e. “Big Data”.

The data volume component of Big Data is a hot topic in machine learning, see for example Jin and Hammer (2014) and O’Leary (2013) and the references therein. But the main focus on the current literature is the sample size and not the sample dimension problem. Only a few studies address the big dimension problem such as for example theoretical investigations of computational models efficiency in high dimensional context (Kainen, Kůrková, & Sanguineti, 2012) and for classification tasks wherein the explosion of features brings about new challenges to computational intelligence (Yiteng, Yew-Soon, & Tsang, 2014).

In this article we focus on the under-explored topic of big dimensionality for regression tasks. The paper is organized as follows: Section 2 gives an overview of data sampling methods and discrepancy measurements. Section 3 shows a comparison of sampling methods and evaluates their performance for high dimension input problems. Section 4 presents the smart sampling and incremental function learning (SSIFL) algorithm that optimally solves this kind of function approximation problems and Section 5 shows the results of applying SSIFL to a number of benchmark and real-world functions. Finally, the conclusions are given in Section 6.

## 2. Data sampling methods

Generally speaking a good sampling method should create training data that accurately represents the underlying function preserving the statistical characteristics of the complete dataset. This section presents first a survey of sampling methods grouped in four categories that can be applied to generating high dimensional sample data.

### 2.1. Stochastic methods

Pseudo-Random number generators (PR) (Marsaglia & Tsang, 2000; Matsumoto & Nishimura, 1998) are commonly used to create samples in a given range. PR methods are fast and simple to use, but they are not distributed uniformly enough especially for the cases of low number of sampling points and/or large number of dimensions. PR sampling is sometimes called “pure” or “plain” Monte Carlo (Swiler, Slepoy, & Giunta, 2006).

### 2.2. Deterministic methods

Uniform populations can be created using quasi-random or sub-random sequences that cover the input space quickly and evenly, the uniformity and coverage improves continually as more data points are added to the sequence. Deterministic methods can be divided into two subcategories (Kazimipour, Li, & Qin, 2013) described in the next subsections.

#### 2.2.1. Low discrepancy methods

Low Discrepancy methods have the support of theoretical upper-bounds on discrepancy. Halton, Sobol, Niederreiter, Hammersley, and Faure are well known sequences from this category.

In this work we use Halton sequences, shortened as (HA), which are constructed according to a deterministic algorithm that uses prime numbers as bases for each dimension (Halton, 1960). Halton sequences work well in low dimensionality, but they lose uniformity in high dimensions. Workaround solutions such as using big prime numbers, setting leap values, scrambling and shuffling improve the sampling uniformity in such cases. The HA method is computationally efficient even for very high dimensional spaces.

#### 2.2.2. Experimental design methods

Experimental Design methods are commonly used for initializing the population of evolutionary algorithms in order to accelerate convergence speed and improve stability. The most representative methods in this category are:

- Uniform Design: a space-filling method based on prime numbers that generates points uniformly scattered on the input domain (Peng, Wang, Dai, & Cao, 2012).
- Orthogonal Design: based on Latin squares for creating orthogonal arrays (Leung & Wang, 2001).

### 2.3. Geometrical methods

#### 2.3.1. Uniform grid

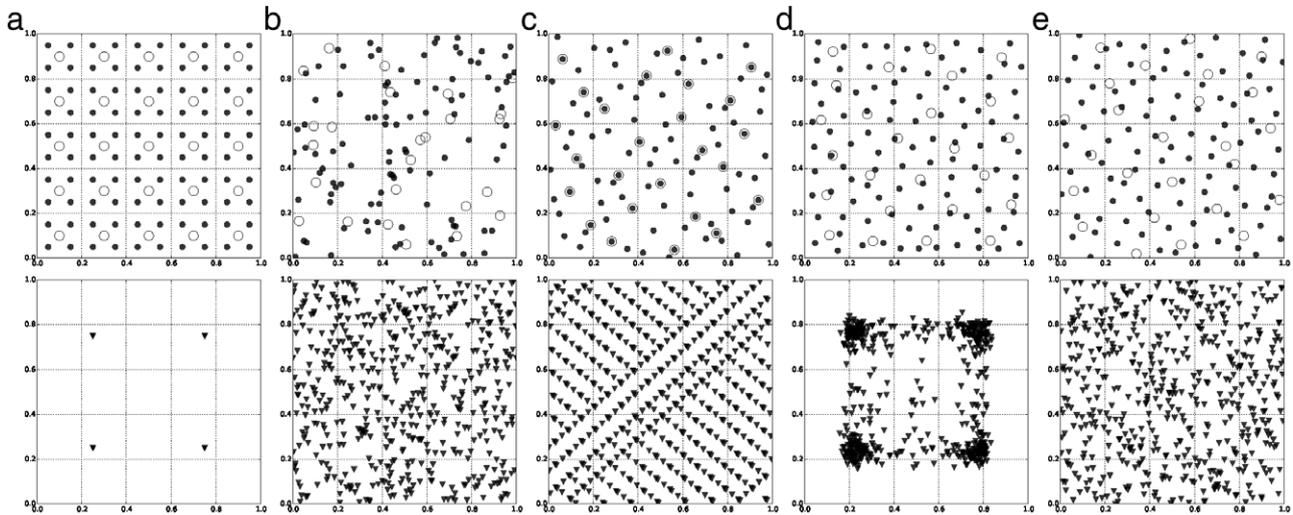
Uniform grid (UG) is the simplest sampling method in which the samples are created using node points at fixed intervals uniformly distributed for every dimension. UG is commonly used for creating look-up tables to accelerate complex model computations (Perkins et al., 2012; Richter, Heege, Kiselev, & Schläpfer, 2014).

Using a uniform grid is convenient for storing the look-up tables in multi-dimensional arrays. As we will show in Sections 3.2 and 3.3, the coverage of the input space using UG is very poor for low number of sampling points and high dimensions.

#### 2.3.2. Latin hypercube

Latin hypercube sampling (McKay, Beckman, & Conover, 1979) partitions the input space into bins of equal probability and distributes the samples in such a way that only one sample is located in each axis-aligned hyperplane. This method and variations like nearly-orthogonal Latin hypercube (Cioppa, 2002) and Distributed Hypercube and Improved Hypercube sampling (Beachkofski & Grandhi, 2002) are very popular in computer model problems. A Particle Swarm Optimization algorithm for solving large-scale Latin hypercube design problems was proposed recently (Aziza & Tayarani-N., 2014).

Latin hypercube sampling is useful when the underlying function has a low order distribution but this method produces clustering of sampling points at high dimensions.



**Fig. 1.** Sampling sequences generated with (a) uniform grid, (b) pseudo-random, (c) Halton, (d) CVT, and (e) LCVT. The top panel shows the sequences corresponding to 2-D input space with a total of 25 (white circle) and 100 (black points) samples. The bottom panel shows plots of the same sampling sequences for a 9-D input space sampled with 2 points per dimension giving a total of  $2^9 = 512$  samples projected to the dimension 7 and 9.

### 2.3.3. Centroidal Voronoi Tessellation

Centroidal Voronoi Tessellation (CVT) generates sample points located at the center of mass of each Voronoi cell covering the input space. The CVT algorithm starts with an initial partition and then iteratively updates the estimate of the centroids of the corresponding Voronoi sub regions. Recent advances in mathematical and computational studies and in practical applications of CVT are presented in [Du, Gunzburger, and Ju \(2010\)](#).

One important advantages of CVT is that sampling points may be easily distributed according to a prescribed density function. One drawback of the CVT method lies in being computationally demanding for high dimensional spaces.

### 2.3.4. Opposite methods

The Opposition Based Learning initialization is a well-known method used to initialize the population in evolutionary algorithms ([Rahnamayan, Tizhoosh, & Salama, 2008](#)). The sequence of opposite points (complimentary point with respect to given bounds) is created from the original sequence, then both sequences are merged and a fitness function is used to select the best samples. The Quasi-opposition Based Learning ([Rahnamayan, Tizhoosh, & Salama, 2007](#)) is an improvement on the previous method that increases the sequence uniformity by creating opposite points using a random range.

The opposite sampling methods are computationally very effective and straightforward to use.

## 2.4. Hybrid methods

Hybrid methods combine different sampling methods with the aim of improving the sampling coverage and uniformity. A sampling method can be used as (a) pre-processing step, e.g. the computation of CVT samples can be initialized with stochastic or deterministic methods and (b) as post-processing, e.g. using geometrical methods to reorganize or create new samples.

In this work we use Latinized CVT (LCVT) ([Romero, Burkardt, Gunzburger, & Peterson, 2006](#)). We create first a Halton sequence as initial population, then the CVT sample is computed and finally the sequence is Latinized.

## 2.5. Sampling in high dimensional space

The sampling methods usually perform very well in low dimensions, the top panels of [Fig. 1](#) contrast the 2-D input space

sequences created using 25 and 100 points drawn from uniform grid, pseudo random, Halton, CVT, and LCVT methods. A visual inspection of the sampling sequences in a 2-D space gives a fast qualitative feedback on the degree of uniformity achieved and how it varies over the input space. In the 2-D case using a uniform grid sampling is very appealing, the points are well spread and the samples with 25 and 100 points are complementary. Note that the Halton sequences with 25 and 100 points overlap whereas the corresponding sequences created with the other sampling methods do not. A workaround solution for creating complementary Halton sequences is to use different prime numbers as basis of different sequences. The random sequence is less uniform than the other ones.

Most of the sample strategies have problems with high dimensional sequences as can be appreciated in the bottom panels of [Fig. 1](#) showing sampling sequences for a 9-D input space with 2 points per dimension given a total of 512 samples. The plots show the sampling points projected into dimensions 7 and 9. All the points with uniform grid sampling concentrate on only 4 places as this sampling algorithm allocate two fixed grid points per dimension. Halton has well known spurious correlation problems ([Robinson & Atcity, 1999](#)) that leave large empty regions not covered, this problem can be at least partially solved using alternative algorithms like scrambled Halton. The points created with the CVT algorithm concentrate mainly in four regions reflecting the good volumetric but bad spatial uniformity of the samples when projected to single dimensions. Only LCVT behaves qualitatively better than pure random.

Developing algorithms for creating good sampling uniformity in high dimensional spaces with low computational cost remains an area of active research.

## 2.6. Sampling non-uniform distributions

A sequence of uniformly distributed points can be mapped into another sequence that reflects a desired non-uniform joint probability distribution function using different techniques like the rejection method or weighted sampling ([Moskowitz & Cafisch, 1996](#)). In this paper we use the more generic procedure described in [Romero et al. \(2006\)](#). The cumulative distribution function  $CDF(x)$  for a random variable  $x$  is

$$CDF(x) = \int_0^{x \leq 1} PDF(x') dx' \quad (1)$$

where  $PDF(x)$  is the probability density function of the random input  $x$ . Since the CDF is defined in the unit hypercube domain it is possible to inverse-map sequences uniformly distributed between 0 and 1 into sequences  $\{x_i\}$  drawn from the density function  $PDF(x)$ .

2.7. Importance sampling

Importance sampling is a commonly used method for variance in Monte Carlo integration (Moskowitz & Caflich, 1996) and system reliability analysis (Patelli, Pradlwarter, & Schuëller, 2011). An importance function  $h$  is introduced which mimics the behavior of the function  $f$  to approximate, sampling points are generated according to the distribution of  $h$  instead of being uniformly distributed. In this work we create importance sampling sequences following the distribution of  $h$  by first creating a uniformly distributed sequence and then computing the inverse-map using (1) and the normalized  $h$  (integral equals 1) as a density function.

The top panel of Fig. 2 depicts the sampling sequences corresponding to the top panel of Fig. 1 transformed using as important sampling function an exponential function. The middle panel of Fig. 2 illustrates the sampling sequences corresponding to the bottom panel of Fig. 1 transformed using as important sampling function a 2-D Gaussian probability distribution with standard deviation of 0.8 and 1.2. Finally the bottom panel of Fig. 2 shows histograms resulting from the sampling sequences of the middle panel. The problems of the sampling methods in this 9-D case are more evident in the histograms. All samples from uniform grid concentrate on 4 positions, similarly the samples from CVT concentrate on 4 regions.

3. Comparison of sampling methods

This section presents a systematic comparison of selected sampling methods from the different categories described in the previous chapter. First the benchmark functions used for the comparison are presented, then we describe how to quantitatively assess the goodness of the sampling methods both in: the input space (uniformity measures) and in the output space (statistical measures). Finally the results of the different sampling methods applied to a number of benchmark function are discussed.

3.1. Benchmark functions

Three benchmark functions commonly used to test the performance of global optimization algorithms are used in this work. Asymmetric benchmark functions containing multiple minima and maxima were selected to better evaluate the effects of the different sampling strategies.

3.1.1. Shifted Ackley function

A number of unimodal and multimodal basic functions like Sphere, Rosenbrock, Rastrigin, Weierstrass, Griewank, and Ackley are commonly used for testing optimization problems. We selected the shifted Ackley's function defined in a multi-dimensional input space as

$$F_{SA}(x) = -20 \exp \left( -0.2 \sqrt{1/n \sum_{i=1}^n z_i^2} \right) - \exp \left( 1/n \sum_{i=1}^n \cos(2\pi z_i) \right) + 20 + e + f\_bias_i \quad (2)$$

with  $z = (x - o)$  the shifted global optimum. This function is multimodal, shifted, separable and scalable, see Tang et al. (2008).

3.1.2. Composition function

More complex functions can be created by combining basic functions

$$F_{CO}(x) = \sum_{i=1}^n \{w_i * [f_i(x_i - o_i) / (\lambda_i * M_i) + bias_i] + f\_bias_i\}. \quad (3)$$

The various parameters used in this equation define the way the basic functions are combined, see CF5 in Liang, Suganthan, and Deb (2005) for details.

3.1.3. Fast Fractal Double Dip function

This multimodal function is defined as

$$F_{FD}(x) = \sum_{i=1}^n fractal(x_i + twist(x_{(i \bmod n)+1})). \quad (4)$$

Specifics about this function including the definition of  $twist()$  and  $fractal()$  can be found in Tang et al. (2008).

Fig. 3 shows 3-D plots of the functions Shifted Ackley, Composition, and Fast Fractal Double Dip for the case of 2-D input space.

3.2. Discrepancy as measure of uniformity in the input space

Discrepancy is a quantity that measures the uniformity of a set of points in a hyper dimensional cube and it is usually employed in the integral approximation field. Given a dataset of  $s$  points  $P = \{x_1, \dots, x_s\}$  with  $x \in \mathbb{R}^n$  the  $n$ -dimensional unit cube  $I^n = [0, 1]^n$ ,  $n \geq 1$ , the infinite star discrepancy  $D_s^*(P)$  is defined as

$$D_s^*(P) = \sup_{J \in I^*} \left| \frac{A(J, P)}{s} - V(J) \right| \quad (5)$$

where  $I^*$  is the family of all subintervals of  $I^n = [0, 1]^n$  of the form  $\prod_{i=1}^n [0, J_i)$ ,  $V(J)$  is the volume of a subinterval  $J$ , and  $A(J, P)$  is the number of points of  $P$  belonging to  $J$  (Niederreiter, 1992). The star discrepancy describes the relationship between the number of points in a subinterval and its volume.  $D_s^*(P)$  varies between 0 and 1; low discrepancy values indicate that the points are well scattered over the input space. The optimal sampling methods are the ones that rapidly converge to a star discrepancy of 0 as the number of sample points increases.

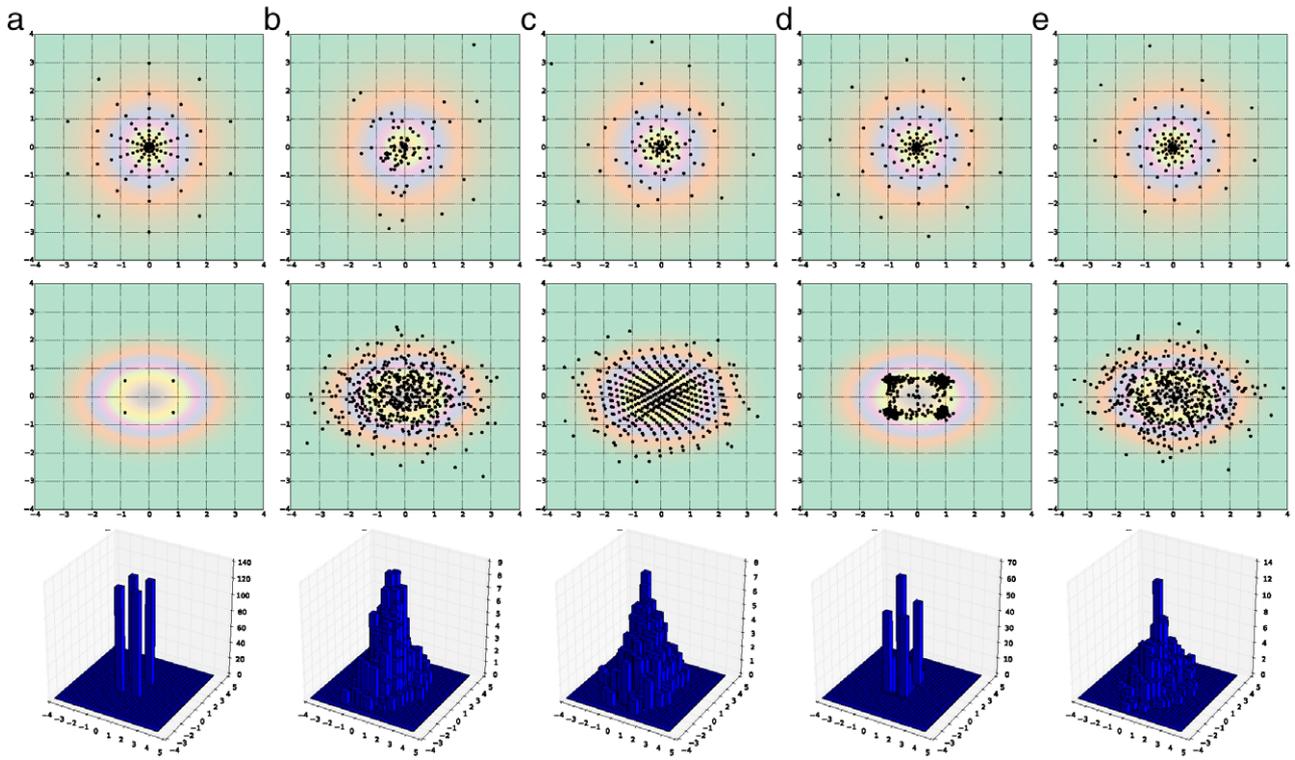
The computation of  $D_s^*(P)$  for  $n \geq 2$  is a time consuming and not an easy task (Thiérmard, 2001). Alternatives more suitable for high dimensional data are the  $L_2$ -star discrepancy (Braaten & Weller, 1979)

$$D_s^{L_2}(P) = \sqrt{\frac{1}{s} \sum_{k=1}^s \left( \frac{A(J_{(k)}, P)}{s} - V(J_{(k)}) \right)^2} \quad (6)$$

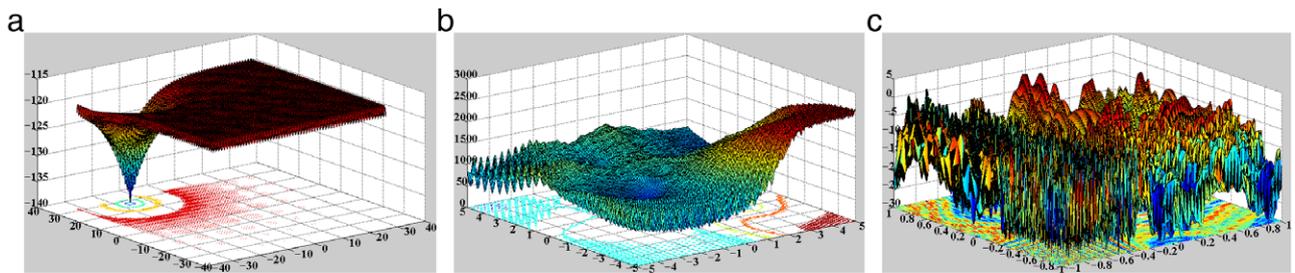
and the modified  $L_2$ -star discrepancy (Qazi & Linshu, 2006) defined as

$$D_s^{ML_2}(P) = \left( \frac{4}{3} \right)^n - \frac{2^{1-n}}{s} \sum_{d=1}^s \prod_{i=1}^n (3 - x_{di}^2) + \frac{1}{s^2} \sum_{d=1}^s \sum_{j=1}^s \prod_{i=1}^n (2 - \max(x_{di}, x_{ji})). \quad (7)$$

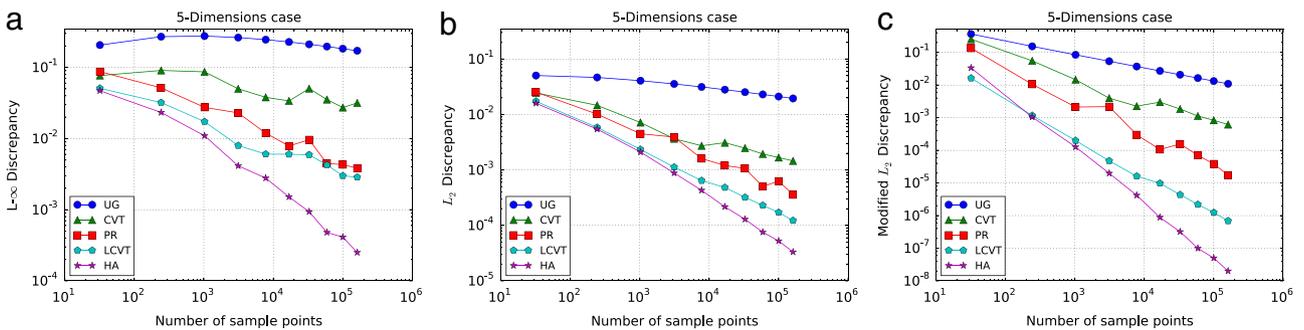
The discrepancy of 5-D input space sequences as function of the number of samples created using different sampling methods is depicted in Fig. 4. The three plots correspond to discrepancy computed with (5), (6), and (7). The best sampling methods are HA and LCVT with similar convergence rates of discrepancy; both



**Fig. 2.** Sampling sequences (black points) created from the sequences from Fig. 1 resampled using important sampling functions represented as color surface plot on the background. (top panel) 2-D input space with 100 samples and exponential important sampling function. (middle panel) A 9-D input space with 512 samples projected to the dimension 7 and 9 and resampled with a 2-D Gaussian probability distribution important sampling function. (bottom panel) Histogram showing the distribution of the samples shown in the middle panel. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)



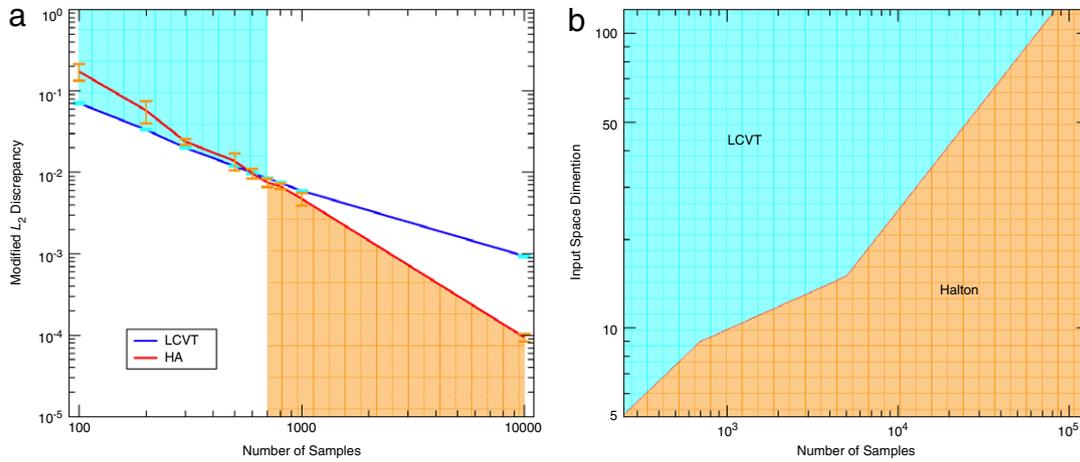
**Fig. 3.** Plots of the benchmark functions for a 2-D input space: (a) Shifted Ackley, (b) Composition, and (c) Fast Fractal Double Dip.



**Fig. 4.** Discrepancy measurements in (a) infinite, (b)  $L_2$  and (c) modified  $L_2$  star norms for 5-D input space sequences as function of the number of samples. The five curves per plot correspond to different sampling methods: uniform grid (UG), CVT, pseudo-random (PR), LCVT and Halton (HA).

outperform pseudo random sequences by a factor of 10 regarding the number of samples needed to reach a given discrepancy. The worst sampling method is by far the uniform grid; it needs  $10^5$  samples to reach similar discrepancy levels achieved with only  $10^2$  samples created with HA or LCVT.

It is important to note that discrepancy is calculated using the 5-D samples and not their 2-D projections depicted in the plots. Spurious correlation problems on samples projected into single dimensions, like the ones from HA shown in the bottom of Fig. 1, are not reflected in such discrepancy measurements.



**Fig. 5.** (a) Modified L<sub>2</sub>-star discrepancy as function of the number of samples for 9-D input space for LCVT (cyan) and HA (orange) sampling sequences, LCVT performs better than Halton for sequences with less than 700 samples. (b) Sampling method with lower discrepancy as function of the input space dimension and the number of samples, LCVT outperforms Halton in case of low dimensions and relative few samples. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The left plot of Fig. 5 shows the modified L<sub>2</sub>-star discrepancy as function of the number of samples for a 9-D input space for LCVT and HA sampling sequences, the error bars correspond to the worst discrepancy computed in the 2-D projections. As it can be appreciated, LCVT outperforms HA when relative few samples are available but the situations change around 700 samples where HA has a lower discrepancy as LCVT. The right plot of Fig. 5 shows the inflection point between LCVT and HA modified L<sub>2</sub>-star discrepancy as function of the input space dimension and the number of samples.

3.3. Statistical measures of the response function in the output space

The goodness of a sampling method with respect to the coverage in the output space of a response function  $f$  can be assessed by the multivariate function integration expressed as

$$I(f) = \int_{C^n} f(x)dx \tag{8}$$

with  $C^n = [0, 1]^n$ . The sample mean gives an approximation to  $I(f)$  by

$$\hat{I}(f, P) = \frac{1}{s} \sum_{i=1}^s f(x_i) \tag{9}$$

where  $P = \{x_i\}$  for  $i = 1$  to  $s$  is a set of points in  $C^n$ . If the sampling points  $x_i$  are i.i.d. uniformly distributed on  $C^n$  then it is known that the sample mean is unbiased. The Koksma–Hlawka inequality gives an upper bound for the approximation error

$$\left| I(f) - \hat{I}(f, P) \right| \leq D(P)v(f) \tag{10}$$

where  $D(P)$  is the discrepancy of  $P$  as defined in Section 3.2, and  $v(f)$  is a measure of the variation of  $f$  as defined in Fang, Ma, and Winker (2002). If the sampling method used to create  $P$  converges to a discrepancy of 0 as the number of sample points increases, then this inequality ensures that the sample mean converges to the multivariate function integral. Therefore the convergence rate of the sample mean from the output space of a response function depending on the number of sample points can be used to measure the goodness of a sampling method.

The efficacy of sampling methods is commonly compared using estimates of response function mean and standard deviation

(Romero et al., 2006). In this work we propose to use not only the mean and standard deviation, but also the higher order statistics skewness (measure of the asymmetry) and the Kurtosis (measure of the peakedness) as they provide complementary information of  $f$  computed from the samples in  $P$ .

Fig. 6 depicts the statistical moments of the Fast Fractal Double Dip function for a 5-D input space and the Composition function for a 10-D input space as function of the number of samples. The uniform grid sampling for the case of 10 dimensions does not converge even with a large number of samples, the performance of CVT is slightly better, while PR, LCVT and HA nicely converge with increasing samples.

The convergence of the statistical moments on the response function can be used to take an informed decision on the adequate number of samples required to properly cover the output space.

4. Smart sampling and incremental function learning algorithm

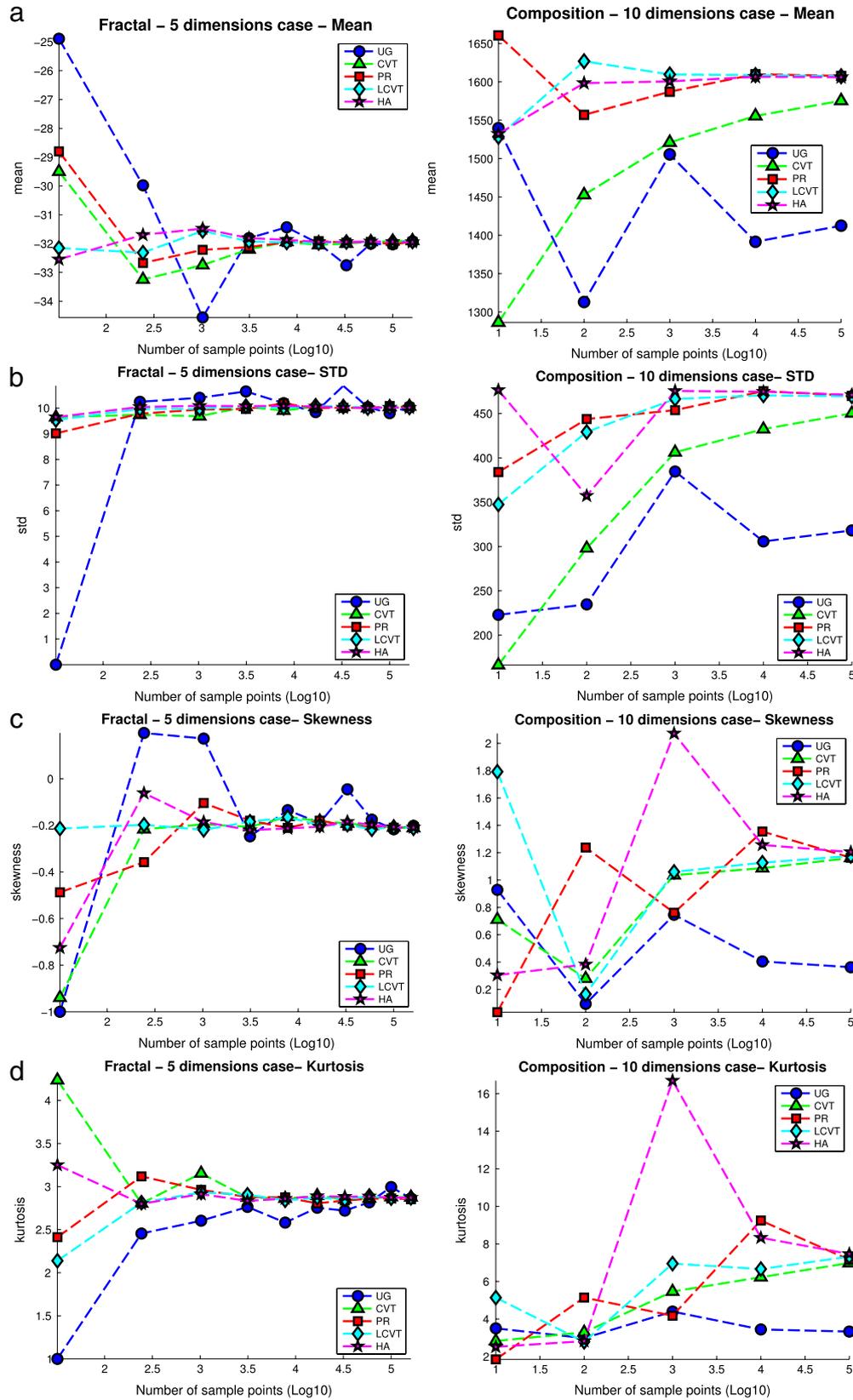
In this section we present the Smart Sampling and Incremental Function Learning Algorithm (SSIFL) based on the results from the previous section. First the problem is formalized, then the algorithm SSIFL stages are listed and then finally the single steps are described in detail in the following subsections.

The goal of the SSIFL algorithm is to find a Probably Approximately Correct Computation (PACC) regression model  $\hat{f}$  that accurately approximates a Lebesgue measurable function  $f : X \rightarrow Y$  with input  $X \subset \mathbb{R}^n$  and output  $Y \subset \mathbb{R}^m$  from a set of finite training samples  $T = \{x_i, y_i\}$  for  $i = 1$  to  $s$ , subject to the constraint of minimizing the number of calls to the function  $f$  needed to create the training samples  $s$ .

Following the PACC framework from Alippi (2014) and using as figure of merit  $u(x) = \left( |f(x) - \hat{f}(x)| \right)$ , we have that  $\hat{f}$  is a PACC of function  $f$  at accuracy  $\tau$  and confidence  $\eta$  when

$$\Pr \left( |f(x) - \hat{f}(x)| \leq \tau \right) \geq \eta, \quad \forall x \in X. \tag{11}$$

In other terms, the function  $\hat{f}$  is approximately correct in the sense that it approximates  $f$  at level  $\tau$  with probability  $\eta$ . If  $\tau$  is small, then  $\hat{f}$  provides a value which approximates the true  $f$  with high probability.



**Fig. 6.** Statistical moments (a) mean, (b) standard deviation, (c) skewness, and (d) kurtosis of the response function in the output space as function of the number of sample patterns for two of the benchmark functions depicted in Fig. 3. The left panels correspond to the Fast Fractal Double Dip function for a 5-D input space and the right panels correspond to the Composition function for a 10-D input space.

The PACC formalism allows a probabilistic performance estimation of  $\tau$  and  $\eta$  from a finite dataset as described in

Section 4.6. The minimum number of samples  $s_{PACC}$  required for assessing inequality (11) can be calculated using the Chernoff bound

(Chernoff, 1952) as

$$s_{PACC} \geq \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta} \quad (12)$$

where  $\varepsilon \in (0, 1)$  are the expected accuracy and confidence  $1 - \delta$  levels,  $\delta \in (0, 1)$ . According to this inequality, the accuracy is more demanding on the number of samples (quadratic bound) than the confidence (linear bound). It is important to note that the Chernoff bound does not depend on the input dimension of  $f$ , that means that the PACC framework allow us to escape the curse of dimensionality problem for assessing  $\tau$  and  $\eta$  (Alippi, 2014).

The SSIFL algorithm is described in the following table. The algorithm numbered steps are described in more detail in the following subsections, the step numbers are the same as the corresponding subchapter numbers.

**Algorithm 1:** Smart Sampling and Incremental Function Learning Algorithm

<b>Input:</b>	Lebesgue measurable target function $f: X \rightarrow Y$ with $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$
<b>Output:</b>	Approximator function $\hat{f}$ with accuracy $\tau$ and confidence $\eta$
<b>Begin</b>	
1.	<i>// Initialization phase:</i> Select accuracy $\varepsilon$ and confidence $\delta$ Compute $s_{PACC}$ using (12) Set $s_{val}$ such that $s(n, s_{val} - 1) \leq s_{PACC} \leq s(n, s_{val})$ $iteration = 0$ Create function approximator $\hat{f}(\theta_0)$ using initial model parameters $\theta_0$
2.	<i>// Initialize test data set and create train dataset using smart sampling</i> $Test = \{\}$ $Train =$ Create input/output dataset using $s(n, s_{val})$ sampling points
	<b>Repeat</b>
3.	<i>// Use previous train as test dataset and create new train dataset</i> $Test = Train$ $Train =$ Create input/output dataset using $s(n, s_{val} + iteration)$ points
4.	<i>// Incremental function learning</i> $\hat{f}(\theta_{iteration-1}, Train, Val) \mapsto \hat{f}(\theta_{iteration})$ $iteration = iteration + 1$
5.	<b>Until</b> Convergence reached
6.	<i>// Determine approximator accuracy and confidence levels</i> $Val =$ Create input/output dataset using $s(n, s_{val})$ sampling points Compute $\tau$ and $\eta$ of $\hat{f}$ using $Val$ dataset
<b>End</b>	

4.1. Initialization phase

The number of samples to create the approximation function with input space dimension  $n$  is selected using the following sequence

$$s(n, i) = \begin{cases} i^d & \text{if } n \leq 9 \\ 10^i & \text{if } n \geq 10. \end{cases} \quad (13)$$

This equation creates a sequence of the form  $\{2^d, 3^d, 4^d, \dots\}$  for 9 input dimensions or less, for larger dimensions the sequence is  $\{10^2, 10^3, 10^4, \dots\}$ . For given dimension and index, the sequence  $s(d, i)$  determines the number of samples following a good compromise on the number of samples to generate per iteration. Alternative formulations can be used as well.

The accuracy  $\varepsilon$  and confidence  $\delta$  levels are selected, then  $s_{PACC}$  is computed using (12) and  $s_{val}$  is selected from the sequences defined by (13) such that  $s(n, s_{val} - 1) \leq s_{PACC} \leq s(n, s_{val})$ . In other terms, the minimum number of samples used to create the training, testing and validation datasets is selected such that it satisfies the Chernoff bound for the given  $\varepsilon$  and  $\delta$ .

The number of *iteration* is set to 0 and the function approximator  $\hat{f}(\theta_0)$  is created using the initial model parameters  $\theta_0$ . In case that a neural network is used, then the model parameters are the network weights and biases.

4.2. Smart sampling during initialization

The sampling method LCVT or HA is selected based on the right plot of Fig. 5 and using the current input space dimension  $n$  and the number of samples  $s$  to generate. Once the sampling method is selected then the set of inputs  $X = \{x_1, \dots, x_s\}$  with  $X \subset \mathbb{R}^n$  is created.

In case that a probability density function of  $f()$  is available then the input sequence is resampled using the important sampling technique as described in Section 2.7.

In the next step, the output  $Y = \{f(x_i)\}$  for  $i = 1$  to  $s$  with  $Y \subset \mathbb{R}^m$  is created. This is usually the most time consuming part of the sampling generation process, the processing time needed for computing  $f(x)$  for only a few samples is usually larger than the time needed for creating the complete input sequence.

Finally the dataset  $Train = \{x_i, y_i\}$  for  $i = 1$  to  $s$  samples is created and the *Test* dataset is initialized with an empty set.

It is important to note that by using LCVT or HA the input space is uniformly covered independently of the number of samples generated.

4.3. Smart sampling during iteration

First the *Train* dataset from the previous iteration is used as the new *Test* dataset and then a new *Train* dataset is created using the generic procedure described in the previous subsection. The number of samples of the *Train* dataset is increased in each iteration according to the sequences defined by (13) and starting with  $s(n, s_{val})$ . In other words, the first training dataset contains the same amount of samples as the testing dataset (satisfying the Chernoff bound) and then new samples are created in each iteration.

It is worth noticing that the test and training datasets generated in each smart sampling iteration are complementary (no duplicated inputs) and they cover the full input/output space.

4.4. Incremental function learning

The goal of learning is to build a PACC approximator of the target function  $f()$

$$\hat{f}(\theta) \approx f() \quad (14)$$

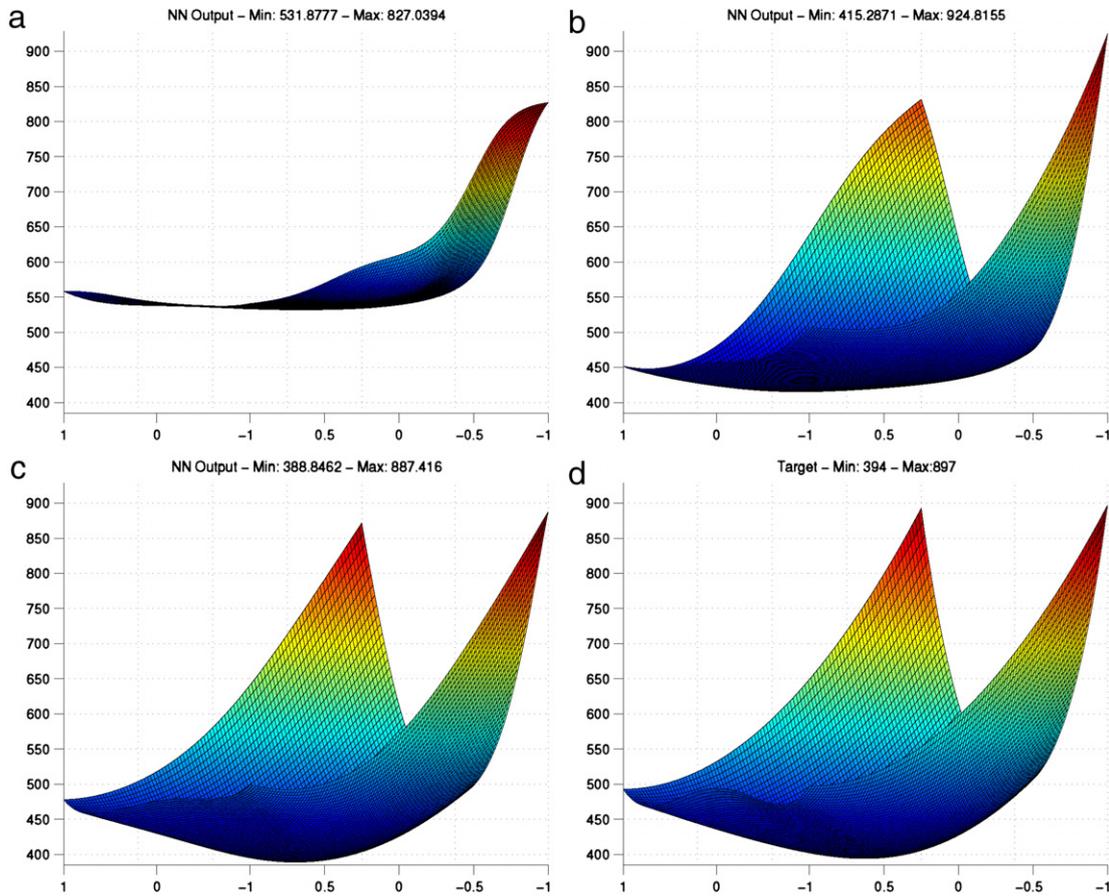
with  $\theta$  the parameter vector describing the family of models. In case that artificial neural networks are used as approximators, then  $\theta$  represents the weights and biases of the network.

In each iteration, the parameter vector  $\theta$  is updated based on the information present in training dataset *Train*. Overfitting is avoided using the early-stopping technique based on the average approximation error over the testing dataset *Test*.

The training algorithm is selected according to the dimensions of the network input/output and the number of training samples. For small- and medium-sized dimensionality the Levenberg-Marquardt algorithm can be used, but with large- and high-size dimensions a backpropagation algorithm working in parallel over a multi-core architecture (Schuessler & Loyola, 2011) is preferred.

The smart sampling procedure described in the previous subsections ensures that the size of the training dataset is continuously increased in every iteration. That means that the target function is learned from the training data in an incremental way during each iteration.

The proposed incremental function learning optimizes the function approximation problem by learning first the ground structure of the target function with relative few sampling points that cover the complete input/output space and then fine-tuning the model parameters to better approximate the target function using more and more sampling points in every iteration. This process is visualized in Fig. 7 showing the incremental learning



**Fig. 7.** Incremental learning of the Rosenbrock function in 5-D input space. The plots show the approximation results from (a)  $4^5$  samples, (b)  $6^5$  and (c)  $8^5$  samples. After a few iterations the learned function accurately approximates the target function (d).

of the Rosenbrock function in 5-D, the learned function accurately approximates the target function after a few iterations.

#### 4.5. Check convergence

The iteration over smart sampling for creating training datasets and incremental function learning is repeated until pre-defined convergence criteria are reached.

The first criteria are the convergence of the statistical moments of the target function output from iteration to iteration as described in Section 3.3. The difference between the statistical moments of  $Y$  from the current iteration  $Y_i = f(X_i)$  and the previous iterations  $Y_{i-1} = f(X_{i-1})$  should be smaller than a given delta to ensure that the target function is well represented by the generated sampling points.

Interestingly, using the minimum number of samples  $s_{PACC}$  for calculating the expectation  $E[f(X)]$  of the function  $f(X)$  by estimating the empirical mean  $\hat{E}_{s_m}[f(X)] = 1/n \sum_i^{s_m} f(x_i)$  ensures that the following inequality

$$P\left(\left|\hat{E}_{s_m}[f(X)] - E[f(X)]\right| \leq \varepsilon\right) \geq 1 - \delta \quad (15)$$

holds for any accuracy level  $\varepsilon \in (0, 1)$  and confidence  $\delta \in (0, 1)$  (Alippi, 2014). That means that the convergence check on the first statistical moment is actually a convergence check on the expectation of the target function as function of the number of samples.

The second criteria are the convergence of the histograms of the network parameters  $\theta$  from iteration to iteration. Following the Bayesian formalism applied to neural networks (MacKay, 1995),

learning means changing our belief about the network parameters  $\theta$  from the prior  $P(\theta)$  to the posterior  $P(\theta|T)$  as a consequence of seeing the training data  $Train$

$$P(\theta|Train) = \frac{P(Train|\theta)P(\theta)}{P(Train)} \quad (16)$$

$P(Train|\theta)$  is the likelihood and  $P(Train)$  is called the evidence for the neural network. In other terms, the posterior distribution incorporates the prior knowledge and the information conveyed by the training dataset.

The Occam factor (MacKay, 1995) that penalizes a neural network for having the parameters  $\theta$  defined as

$$Occam\ factor = \frac{\sigma_{\theta|Train}}{\sigma_{\theta}} \quad (17)$$

where  $\sigma_{\theta|Train}$  is the width of the evidence and the prior is uniform on a large interval  $\sigma_{\theta}$ . The second convergence criteria are reached when the ratio of the Occam factors from iteration to iteration is close to one. This ratio is difficult to compute directly but it can be approximated by the ratio of the FWHM from the histogram of the network parameters  $\theta_i$  and  $\theta_{i-1}$  from the previous and current iteration.

The third and last stop criterion is the check that a maximum allowed number of iterations are reached.

#### 4.6. Determine approximator accuracy and confidence levels

The PACC formalism allows a probabilistic performance estimation of the accuracy  $\tau$  and confidence  $\eta$  of  $\hat{f}(x)$  from Eq. (11) by defining a  $p(\gamma)$  such that

$$p(\gamma) = \Pr\left(\left|f(x) - \hat{f}(x)\right| \leq \gamma\right) \quad (18)$$

Following the method from Alippi (2014), this probability can be approximated with  $\hat{p}(\gamma)$  defined as

$$\hat{p}(\gamma) = \frac{1}{s} \sum_{i=1}^s L\left(\left|f(x_i) - \hat{f}(x_i)\right| \leq \gamma\right), \quad (19)$$

$$L(u(x_i) \leq \gamma) = \begin{cases} 1 & \text{if } u(x_i) \leq \gamma \\ 0 & \text{if } u(x_i) > \gamma \end{cases}$$

and evaluated over the finite dataset  $x_i \in Val$ . The *Val* dataset is created using the generic procedure described in Section 4.2 and using  $s(n, s_{val})$  sampling points, that means that the *Val* dataset has at least  $s_{PACC}$  element and therefore the Chernoff bound holds.

The value  $\bar{\gamma}$  is defined as the smallest value from an arbitrary incremental set of points  $\{\gamma_1, \dots, \gamma_k\}$  with  $\gamma_i < \gamma_j, \forall i < j$  from which  $\hat{p}(\gamma_i) \geq 1 - \varepsilon, \forall \gamma_i \geq \bar{\gamma}$ . Selecting  $k \geq s_{PACC}$  points, the Chernoff bound discrepancy  $|p(\bar{\gamma}) - \hat{p}(\bar{\gamma})| \leq \varepsilon$  holds with probability  $1 - \delta$  and therefore  $p(\bar{\gamma}) \geq 1 - \varepsilon$ .

The PACC accuracy of  $\hat{f}$  is then  $\tau = \bar{\gamma}$  with confidence  $\eta = 1 - \varepsilon$  (Alippi, 2014).

## 5. Results

This section shows the results of using the SSIFL algorithm for two benchmark functions and one function from a real-world problem. Note that the selected asymmetric benchmark functions with multiple minima and maxima are very complex not only from the sampling but also from the function approximation point of view.

### 5.1. Regression of 5-D input space benchmark function

The Shifted Ackley function  $F_{SA}$  defined in (2) with an input space dimension of 5 is approximated using the SSIFL algorithm. This function is characterized by a nearly flat outer region and a deep hole at the shifted center, see left plot in Fig. 3. Therefore we use a Gaussian probability distribution important sampling function center at the hole.

The expected accuracy and confidence levels are set to  $\varepsilon = 0.03$  and  $\delta = 0.01$ ; then  $s_{PACC}$  is calculated using (12) given 2944 as result and the corresponding  $s_{val}$  is set to  $5^5 = 3125$ .

The initial feedforward neural network with three hidden layers of 25–10–5 neurons is created and the Levenberg–Marquardt algorithm is used to train the NN. Moreover, the activation functions used are tangent sigmoid for the hidden layers and linear for the output layer.

In the first iteration the input sequence for the *Test* and *Train* datasets are created using LCVT with  $5^5$  and  $6^5$  samples respectively, the input sequences are resampled using the Gaussian importance sampling function and the Shifted Ackley function is evaluated in the resampled input sequences to generate the target output. The NN is then trained with the *Test* and *Train* datasets.

In the second iteration the *Test* dataset is replaced with the *Train* dataset from the previous iteration containing a total of  $6^5$  samples. The new *Train* dataset is created by first generating a LCVT input sequence with  $7^5$  new samples, the input sequence is resampled with the Gaussian importance sampling function and the corresponding target outputs are computed using the  $F_{SA}$  function. The NN from the previous iteration is re-trained with the new *Train* and *Test* datasets.

A total of three iterations are needed before the convergence criteria are reached. Table 1 summarizes the smart sampling (target function mean and standard deviation) and incremental function learning (epochs needed, RMS on the training dataset, and Occam factor) results for each iteration.

Finally a validation dataset *Val* with  $s_{val} = 4^5$  samples is created and it is used to compute the accuracy and confidence levels of the

final neural network. The results are 0.9959 and 0.99 respectively, as mean value of the target function is  $-118.454$ , a mean accuracy at the 0.84% level is reached.

### 5.2. Regression of 100-D input space benchmark function

The Rosenbrock function defined as  $F_{RO} = \sum_{i=1}^{n/2} [100(x_{2i-1}^2 - x_{2i-1}^2 + (x_{2i-1} - 1)^2)]$  with an input space dimension of 100 is approximated using the SSIFL algorithm. First the expected accuracy  $\varepsilon = 0.02$  and confidence  $\delta = 0.03$  are defined; then  $s_{PACC} = 5250$  is calculated and finally the corresponding  $s_{val}$  is set to  $10^4$ .

The initial feedforward neural network with one hidden layer of 100 neurons is created; the NN will be trained using the conjugate gradient backpropagation algorithm. Furthermore, the activation functions used are tangent sigmoid for the hidden layers and linear for the output layer.

The NN is trained with *Test* and *Train* datasets created by evaluating the target function  $F_{RO}$  on HA input sequences with  $10^4$  and  $10^5$  samples respectively. In the second iteration the *Test* dataset is replaced with the *Train* dataset from the previous iteration containing a total of  $10^4$  samples and a new *Train* dataset is created evaluating the Rosenbrock function over a new input HA sequence with  $10^6$  samples. The NN from the previous iteration is re-trained using backpropagation and taking as input the new *Train* and *Test* datasets.

The third and last iteration contains  $10^7$  new *Train* samples. Finally the accuracy and confidence levels of the final neural network are computed using a validation dataset *Val* with  $s_{val} = 4^5$  samples. The results for each single iteration are summarized in Table 2.

### 5.3. Regression of 62-D output space function from a real-world problem

The operational processing of atmospheric remote sensing data usually requires the usage of very complex and extremely time consuming radiative transfer (Gimeno García et al., 2012) models. In this subsection we apply the SSIFL algorithm to accurately approximate a radiative transfer model problem characterized by a 5 dimension input space representing the satellite viewing geometry and surface conditions. In contrast to the benchmark functions, the function from this real-world problem has not a single output value but the output space covers 62 dimensions.

The expected accuracy and confidence levels are set to  $\varepsilon = 0.05$  and  $\delta = 0.02$ ; the calculated  $s_{PACC}$  is 921 and the corresponding  $s_{val}$  is set to  $4^5 = 1024$ .

The initial feedforward neural network is created using one hidden layer with 70 neurons. The hidden layers are activated by a tangent sigmoid function and the output layer by a linear function. The weights are initialized using the Nguyen–Widrow algorithm and the training is performed using the conjugate gradient backpropagation.

In the first iteration the input sequence for the *Test* and *Train* datasets are created using HA with  $3^5$  and  $4^5$  samples respectively. The forward model is evaluated using both 5-dimensional input sequences in order to generate the corresponding outputs in the 62-dimension space. The NN is then trained with the *Test* and *Train* datasets.

In the second iteration the *Test* dataset is replaced with the *Train* dataset from the previous iteration containing a total of  $4^5$  samples and a new *Train* dataset is created using  $5^5$  new samples. The NN from the previous iteration is re-trained with the new *Train* and *Test* datasets.

This iteration process is repeated seven times until the convergence criteria are reached. In the final iteration the *Test*

**Table 1**  
SSIFL results for the Shifted Ackley function  $F_{SA}$  in 5 dimensions.

Iteration	Train samples	Test samples	$F_{SA}$ (Shifted Ackley)	$F_{SA}$	NN epochs	NN MSE	Occam ratio
1	$6^5$ (7776)	$5^5$	-118.757	1.0441	34	0.000688	0.9381
2	$7^5$ (16 807)	$6^5$	-118.759	1.0446	17	0.000626	1.0659
3	$8^5$ (32 768)	$7^5$	-118.454	1.0454	17	0.000598	0.9378

**Table 2**  
SSIFL results for the Rosenbrock function  $F_{RO}$  in 100 dimensions.

Iteration	Train samples	Test samples	$F_{RO}$ mean	$F_{RO}$ StdDev	NN epochs	NN MSE	Occam ratio
1	$10^5$	$10^4$	5830.62	789.85	42	459 310	0.9964
2	$10^6$	$10^5$	5809.56	768.17	272	308 080	0.9575

**Table 3**  
SSIFL results for the real-world remote sensing ( $F_{RTM}$ ) function with 5 input and 62 output dimensions.

Iteration	Train samples	Test samples	$F_{RTM}$ Av.mean	$F_{RTM}$ Av.StdDev	NN epochs	NN MSE	Occam ratio
1	$5^5$	$4^5$	0.082274	0.069293	9 700	$2.0e-06$	1.0100
2	$6^5$	$5^5$	0.082301	0.069333	4 315	$1.7e-06$	0.9564
3	$7^5$	$6^5$	0.082296	0.069340	50 662	$8.2e-07$	1.0268
4	$8^5$	$7^5$	0.082333	0.069367	12 918	$6.9e-07$	0.9851
5	$9^5$	$8^5$	0.082319	0.06935	142	$6.9e-07$	0.9823
6	$10^5$	$9^5$	0.082291	0.069338	70 625	$5.7e-07$	1.0141
7	$11^5$	$10^5$	0.082289	0.069343	200 000	$2.8e-07$	1.0081
8	$12^5$	$11^5$	0.082286	0.069342	61 291	$2.5e-07$	0.9797

and *Train* datasets contain a total of  $11^5$  (161 051) and  $12^5$  (248 832) new samples, Table 3 summarizes the smart sampling and incremental function learning results for each iteration.

It is important to note that the generation of the almost half a million sample points is a very CPU demanding task; the time needed for training the NNs is minor compared with the time needed for computing the radiative transfer modeling outputs.

In the final step the validation dataset *Val* with  $s_{val} = 3^5$  samples is created and it is used to compute the accuracy 0.001606684 and confidence levels 0.98 of the final neural network.

#### 5.4. Regression of 280-D input space function from a real-world problem

In the previous subsections we presented the results of applying SSIFL to regression problems where the samples are created as part of the algorithm with the constraint of minimizing the number of calls to the target function. In this subsection we show how to apply SSIFL to problems with pre-existing datasets that follow an unknown probability distribution function. The SSIFL algorithm can be easily adapted for such problems: the smart sampling is not used for creating new patterns, but it is used for resampling patterns from the pre-existing dataset.

The smart sampling technique described in Sections 4.2 and 4.3 is adapted as follows. After creating the set of  $s$  inputs  $X = \{x_1, \dots, x_s\}$  with  $X \subset \mathbb{R}^n$  and  $n$  the input space dimension, the patterns  $\{e_i, y_i\}$  with  $e \subset \mathbb{R}^n$  and  $y \subset \mathbb{R}^m$  from the pre-existing dataset that are closest to  $x_i$  are selected as the *Train* dataset. Depending on the type of the input attributes different distance measurements can be applied to find the  $e_i$  closest to  $x_i$ .

We apply the adapted SSIFL to the “Blog feedback” dataset (Buza, 2014) with 280 input dimensions downloaded from UCI and consisting of 52 397 and 7624 pre-existing training and validation data.

In the first SSIFL iteration the input sequences  $x_i$  using HA are created with  $10^2$  and  $10^3$  samples and 280 dimensions. *Test* and *Train* datasets are then created by resampling from the pre-existing  $\{e_i, y_i\}$  patterns that are nearest to the HA sequence points, the closest patterns are selected using the Euclidean distance between the points  $e_i$  and  $x_i$ . Note that the distance in all dimensions

are equally weighted by normalizing the input attributes of the pre-existing data to  $[0, 1]$ . A feedforward neural network with one hidden layer of 100 neurons is trained using the resampled datasets.

In the second iteration the *Test* dataset is replaced with the *Train* dataset from the previous iteration and a new *Train* dataset is created resampling  $10^4$  patterns from the remaining pre-existing dataset. The third and final iteration uses the remaining 41 397 samples from the pre-existing dataset as the *Train* dataset. The NNs from the previous iterations are re-trained with the new *Train* and *Test* datasets. See Tables 3 and 4 for a summary of the results from applying SSIFL to this problem.

Finally the accuracy of the trained NNs is assessed computing the mean square error (MSE) over the pre-existing validation dataset containing 7624 samples. The NN trained with 10 000 samples is already quite accurate with only a 8.4% worst MSE than the one from the NN trained with the complete dataset. As comparison the MSE of NNs trained with 10 000 randomly subsamples are on average 66.1% worst. In this way we demonstrate that SSIFL optimally sub-samples and accurately approximates the underlying function of pre-existing datasets.

## 6. Conclusions

In this paper we presented a novel approach (SSIFL) for computing a regression model that accurately approximates functions defined over high dimensional input spaces. This has been achieved under the constraint of minimizing the number of calls to the target function for generating the training data. We smartly combine previously published deep theoretical methods and tools into a practical sampling and function learning algorithm and we showed the applicability of the proposed technique to high-dimensional regression problems.

The sampling method used for generating the training dataset plays a key-role in order to reach high performance of the regression model. Different sampling methods have been evaluated and systematically analyzed in this study. Following a comprehensive discrimination analysis, in this paper we showed that the LCVT and HA sampling sequences are superior in handling data in high dimensionality and therefore they have been selected among all

**Table 4**  
SSIFL results for the real-world blog problem with 280 input dimensions.

Iteration	Train samples	Test samples	NN epochs	NN MSE	Occam ratio
1	1000	100	159	0.0003592	4.1621
2	10000	1000	106	8.161e−05	0.8903
3	41397	10000	124	0.000478	1.2640

the other techniques for the proposed smart sampling technique. However, due to high computational effort needed for calculating the sequence, it might be more convenient to use the HA method in very high dimensional spaces, which has been already pointed out in this study to reach good achievements in reasonable computational time.

Smart sampling allows the easy incorporation of *a priori* information about the probability distribution of the target function (importance sampling). The number of samples to be created (i.e., the calls to the target function) is minimized by incrementally creating the new training dataset until it contains an optimal representation of the target function.

Incremental function learning takes place in each iteration: the training datasets from previous iterations are reused as the test datasets for the current iteration. The novel SSIFL technique proposed in this paper confirmed to be very efficient in order to minimize the mean square error in each function learning iteration. Learning takes place in an incremental way, the regression model is improved in each iteration based on the model from the previous iteration and the new train and test datasets.

Finally, the accuracy and confidence levels of the regression function are determined using a probabilistic performance estimation method.

The SSIFL algorithm has been proven to scale well for very high input dimensions dataset as shown with two benchmark functions with 5-D and 100-D. Furthermore the practical applicability and goodness of SSIFL method has been confirmed in two real-world applications with 62-D and 280-D. It is worth noting that SSIFL is (a) a generic approach that may be used for problems where the training patterns are either created or resampled in an optimal way and (b) it can be combined with any regression tool such as neural networks, support vector machines and many others.

## Acknowledgments

We are grateful to the reviewers for their constructive input to the application of SSIFL to real-world problems. This research work was supported by the German Aerospace Center (DLR).

## References

- Alippi, C. (2014). *Intelligence for embedded systems—a methodological approach*. Switzerland: Springer.
- Aziza, M., & Tayarani-N., M.-H. (2014). An adaptive memetic Particle Swarm Optimization algorithm for finding large-scale Latin hypercube designs. *Engineering Applications of Artificial Intelligence*, 36, 222–237.
- Beachkofski, B. K., & Grandhi, R. V. (2002). Improved distributed hypercube sampling. In *43rd structures, structural dynamics, and materials conference* (pp. 22–25).
- Braaten, E., & Weller, G. (1979). An improved low-discrepancy sequence for multidimensional quasi-Monte Carlo integration. *Journal of Computational Physics*, 33(2), 249–258.
- Buza, K. (2014). Feedback prediction for blogs. In *Data analysis, machine learning and knowledge discovery* (pp. 145–152). Springer.
- Cervellera, C., & Macciò, D. (2013). Learning with Kernel smoothing models and low-discrepancy sampling. *IEEE Transactions on Neural Networks and Learning Systems*, 24(3), 504–509.
- Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4), 493–507.
- Cioppa, T. M. (2002). *Efficient nearly orthogonal and space filling experimental designs for high-dimensional complex models*. (Ph.D. dissertation), California: Naval Postgraduate School.

- Du, Q., Gunzburger, M., & Ju, L. (2010). Advances in studies and applications of centroidal Voronoi tessellations. *Numerical Mathematics: Theory, Methods and Applications*, 3(2), 119–142.
- Enăchescu, C. (2013). Supervised learning using an active strategy. In *The 7th international conference interdisciplinarity in engineering* (pp. 220–228).
- Fang, K.-T., Ma, C.-X., & Winker, P. (2002). Centered L2 discrepancy of random sampling and latin hypercube design, and construction of uniform designs. *Mathematics of computation*, 71(237), 275–296.
- Flato, G., et al. (2013). *Evaluation of climate models in Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, (pp. 741–866).
- Gimeno García, S., Trautmann, T., & Venema, V. (2012). Reduction of radiation biases by incorporating the missing cloud variability by means of downscaling techniques: a study using the 3-d mocart model. *Atmospheric Measurement Techniques*, 5, 2261–2276.
- Gnecco, G., Kúrková, V., & Sanguineti, M. (2011). Some comparisons of complexity in dictionary-based and linear computational models. *Neural Networks*, 24(2), 171–182.
- Gordon, C., Cooper, C., Senior, C. A., Banks, H., Gregory, J. M., Johns, T. C., et al. (2000). The simulation of SST, sea ice extents and ocean heat transports in a version of the Hadley Centre coupled model without flux adjustments. *Climate Dynamics*, 16(2–3), 147–168.
- Grell, G. A., et al. (2005). Fully coupled “online” chemistry within the WRF model. *Atmospheric Environment*, 39(37), 6957–6975.
- Halton, J. H. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1), 84–90.
- Han, J., & Pan, H.-L. (2011). Revision of convection and vertical diffusion schemes in the NCEP global forecast system. *Weather and Forecasting*, 26, 520–533.
- Haupt, S. E., Pasini, A., & Marzban, C. (2009). *Artificial intelligence methods in the environmental sciences*. Netherlands: Springer.
- Hilbert, M., & López, P. (2011). The world’s technological capacity to store, communicate, and compute information. *Science*, 332(6025), 60–65.
- Hsieh, W. W., & Tang, B. (1998). Applying neural network models to prediction and data analysis in meteorology and oceanography. *Bulletin of the American Meteorological Society*, 79(9), 1855–1870.
- Jin, Y., & Hammer, B. (2014). Computational intelligence in big data. *IEEE Computational Intelligence Magazine*, (3), 12–13.
- Kainen, P. C., Kúrková, V., & Sanguineti, M. (2012). Dependence of computational models on input dimension: Tractability of approximation and optimization tasks. *IEEE Transactions on Information Theory*, 58(2), 1203–1214.
- Kazimipour, H., Li, X., & Qin, A. K. (2013). Initialization methods for large scale global optimization. In *IEEE congress on evolutionary computation* (pp. 2750–2757).
- Kolmogorov, A. (1957). On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114, 953–956.
- Krasnopolsky, V. M., & Chevallier, F. (2003). Some neural network applications in environmental sciences. Part II: advancing computational efficiency of environmental numerical models. *Neural Networks*, 16, 335–348.
- Krasnopolsky, V. M., & Schiller, H. (2007). Neural network emulations for complex multidimensional geophysical mappings: Applications of neural network techniques to atmospheric and oceanic satellite retrievals and numerical modeling. *Reviews of Geophysics*, 45, RG3009.
- Leung, Y.-W., & Wang, Y. (2001). An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 5(1), 41–53.
- Liang, J. J., Suganthan, P. N., & Deb, K. (2005). Novel composition test functions for numerical global optimization. In *Proceedings of the IEEE swarm intelligence symposium* (pp. 68–75).
- Loyola, D. (2006). Applications of neural network methods to the processing of earth observation satellite data. *Neural Networks*, 19(2), 168–177.
- Lynch, P. (2006). The ENIAC integrations. In *The emergence of numerical weather prediction* (pp. 206–208). Cambridge: University Press.
- Lynch, C. (2008). Big data: How do your data grow? *Nature*, 455(7209), 28–29.
- MacKay, D. J. C. (1995). Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3), 469–505.
- Marsaglia, G., & Tsang, W. W. (2000). The Ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8).
- Matsumoto, M., & Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1), 3–30.
- McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2), 239–245.

