Interdisciplinary Mathematics Institute

2010:08

Tree Approximation of the Long Wave Radiation Parameterization in the NCAR CAM Global Climate Model

A. Belochitski, P. Binev, R. DeVore, M. Fox-Rabinovitz, V. Krasnopolsky, and P. Lamby

IMI

College of Arts and Sciences

University of South Carolina

Tree Approximation of the Long Wave Radiation Parameterization in the NCAR CAM Global Climate Model

Alexei Belochitski, Peter Binev, Ronald DeVore,
Michael Fox-Rabinovitz, Vladimir Krasnopolsky, and Philipp Lamby

November 8, 2010

Abstract

The computation of Global Climate Models (GCMs) presents significant numerical challenges. This paper presents new algorithms based on sparse occupancy trees for learning and emulating the long wave radiation parameterization in the in the NCAR CAM climate model. This emulation occupies by far the most significant portion of the computational time in the implementation of the model. From the mathematical point of view this parameterization can be considered as a mapping $\mathbb{R}^{220} \to \mathbb{R}^{33}$ which is to be learned from scattered data samples (x^i, y^i), $i = 1, \ldots, N$. Hence, the problem represents a typical application of high-dimensional statistical learning. The goal is to develop learning schemes that are not only accurate and reliable but also computationally efficient and capable of adapting to time-varying environmental states. The algorithms developed in this paper are compared with other approaches such as neural networks, nearest neighbor methods, and regression trees as to how these various goals are met.

1   Introduction

The main computational burden in general circulation models (GCMs) for high-quality weather prediction and climate simulation is caused by the complexity of the physical processes that include, in particular, atmospheric radiation, turbulence, convection, clouds, large scale precipitation, constituency transport and chemical reactions. These processes are modeled by *parameterizations* which are complex 1D subgrid-scale schemes formulated using relevant first principles and observational data. The evaluation of these parameterizations typically consumes 70% -90% of the overall CPU-time in the GCM we consider in this work - the National Center of Atmospheric Research (NCAR) Community Atmospheric Model (CAM).

To overcome this computational "bottleneck", it has been proposed to employ statistical learning methods like neural networks in order to accelerate the evaluation of the parameterizations. This idea originates from [CFCC98], where a battery of neural

networks has been used to approximate certain partial fluxes within long wave radiation (LWR) parameterization of the European Center for Medium-Range Weather Forecasts, explicitly exploiting the structure of that particular model.

Another more direct and more general approach is pursued in [KRC05] where a single neural network *emulation* is used to entirely replace the LWR parameterization in the NCAR CAM *as a whole*. This parameterization takes a surface characteristic and the discretizations of ten vertical profiles of local physical properties and gas concentrations as input and returns a vertical profile of heating rates and seven heat fluxes as output. Mathematically this parameterization can be considered as a mapping $f : \mathbb{R}^{220} \to \mathbb{R}^{33}$, see Section 2.1. The basic idea of an emulation is, for any given input, to get an approximation of the output variables by evaluating a *learned* approximation (a fast process) instead of evaluating the original parameterization (which is a rather complex numerical simulation in its own right). The approximating function is learned from a set of training data points (x^i, y^i) for which the outputs $y^i = f(x^i)$ have been computed by solving the original physical parameterization. The approximation should be both accurate and easy to evaluate in order to provide a significant speed-up of the GCM without significantly changing the prediction of a long-term climate simulation.

While artificial neural networks can be considered as the current state-of-the-art black box methodology for a wide range of high-dimensional approximation problems, and justifiably so, they may not necessarily be the best solution for this particular application. The accuracy of neural network emulations depends on the number of layers and hidden neurons employed. While it is known that neural networks are *universal approximators*, *i.e.*, they can approximate any continuous functions to any predetermined accuracy (see for instance [Hor89] and [DOP97]), this is achieved only by allowing the number of neurons to increase arbitrarily. However, the learning of the network parameters (weights) requires the solution of a large, non-linear optimization problem, which is very time-consuming, prone to deliver sub-optimal solutions and, thus, severely limits the complexity of the network that one can afford to train.

This becomes a practical issue considering the following task: the approximation is trained by a data set that consists of evaluations of the original parameterization gained during a reference run of the climate model. The inputs of this training data set, therefore, cover the physical states observed during a certain time period of climate history. However, the domain in which the parameterization is to be evaluated may change with time as in the case of climate change. In such situations the approximation may be forced to extrapolate beyond its generalization ability, which may lead to large errors. In this case it could become necessary to re-train the emulation in order to adapt it to the new environment.

It would therefore be advantageous to have an alternative to neural networks that would offer an easier training process and that would perhaps even be capable of incremental learning. Additionally, if one could estimate the error of the approximation for a certain input, one could use the original parameterization as a fall-back option during a run of the GCM and immediately incorporate the new data into the emulation. Actually, [KFRTB08] addresses the problem of error estimation for a neural network emulation, but the question of how to dynamically adapt the approximation is left open.

In the present paper we search for an alternative to neural networks within the class of *non-parametric* approximation methods. We cannot offer a full realization of the

program outlined above, but we restrict ourselves to basic design decisions and testing whether such a program has any chance to be successfully implemented. In particular, we discuss the features of two common statistical learning paradigms, (approximate) nearest neighbors and regression trees, and present a new algorithm based on what we call *sparse occupancy trees*, which aims to provide a very efficient nearest-neighbor type algorithm capable of incremental learning. The development of the latter concept was originally motivated by the present application and is comprehensively described in [BDL10].

In order to motivate why we were actually trying to design new algorithms instead of just using an off-the-shelf algorithm, we briefly describe the aforementioned techniques. More information can be found in Section 3 and in standard textbooks like [HTF09]. Non-parametric learning methods typically try to partition the input space and then use simple local models like piecewise constants to approximate the data. In the case of nearest neighbor methods, the input space is implicitly partitioned by the way the training data is distributed: the approximation is constant for query points that have the same set of nearest neighbors. Unfortunately in high dimensions there are no fast algorithms which could answer the question "what are the nearest neighbors to a given query point x?". Therefore one must be content with approximate answers to this question that can be realized using so-called *kd*- or *bd*-trees. Here, assuming that all the training data is available beforehand, the input domain is *recursively* partitioned depending on the distribution of the input points.

Regression trees follow a more adaptive approach and also use the y-values in order to define the domain partition. Here, starting with the entire input domain, the cells in the partition are *recursively* subdivided such that the residual of the resulting approximation is minimized in each step. Obviously, due to their recursive definition, none of these techniques is available for incremental learning without modification: a new data point could theoretically change the decision how to perform the first split in the tree, which would require relearning the tree from the very beginning. Sparse occupancy trees, on the other hand, encode the data in a format that is independent of the distribution of the incoming data.

It must be noted here that no partitioning scheme can be expected to be successful for arbitrary high-dimensional data. (The same could be said about neural networks, although for other reasons.) For instance, if the data points were uniformly distributed in a very high-dimensional space, the attempt to generate local approximations like those described above would be doomed, because the average distance between a query point and the best fitting data point might become large even for huge training data sets. This is often referred to as the *curse of dimensionality*. One usually makes the assumption that the data is actually distributed over some lower-dimensional submanifold or is concentrated in a subset of small measure within the whole input space. In our special case this assumption is justified because the input data is group-wise strongly correlated. One purpose of this work is to quantify this effect, and in Section 4.3 we give an estimate of the intrinsic dimensions of the data set which shows that non-parametric approximation of the long wave radiation data should indeed be feasible.

The main obstacle for the application of tree-based approximation schemes seems to be implementing it in a highly parallel computer system, which is unavoidable in the simulation of huge, complex systems like global climate. Non-parametric ap-

proximation methods are *memory based*, *i.e.*, they need to store all the training data permanently. This limits their practical use to shared memory systems, because on distributed memory systems each core or processor can address only a limited amount of data. The current implementation of the NCAR-CAM system however, seems to be optimized for such distributed memory architectures and requires the emulation to be stored on each individual processor. Nevertheless, as proof of concept, we performed a 10-year climate simulation where the original parameterization was replaced by a regression tree. We report on this numerical experiment in Section 5. The tree was chosen as a compromise between accuracy and storage considerations and trained with a rather limited amount of data. That means that one of the main advantages of non-parametric learning, namely its ability to cope with large amounts of data, was not really exploited. Despite that, we were able to achieve a rather promising result, which indicates the potential usefulness of this approach in future projects.

2   General Considerations

In this section we provide the mathematical formulation of the approximation problem and introduce some basic notation used throughout the paper. Furthermore, we make some very general remarks concerning the design of approximation schemes for heterogeneous vector-valued functions.

2.1   Structure of the LWR Parameterization

The input vectors for the LWR parameterization include one surface characteristic and ten profiles: atmospheric temperature; humidity; the concentrations of ozone, CO_2, N_2O, and CH_4; two chlorofluorocarbon mixing ratios; pressure; and cloudiness. The profiles are discretized and presented by the values in 26 vertical layers of the atmosphere. Some of the quantities are constant in certain layers and have therefore been filtered out of the input vectors, effectively leaving 220 input parameters. The output vectors consist of a profile of heating rates and seven radiation fluxes, altogether 33 values. Hence, the parameterization can be considered as a function

$$f : \left\{ \begin{array}{ccc} \mathbb{R}^{220} & \longrightarrow & \mathbb{R}^{33} \\ x = (x_j)_{j=1,\ldots,220} & \longmapsto & y = (y_l)_{l=1,\ldots,33} \end{array} \right\}.$$

In what follows, we denote the components of a vector with subscripts, while use superscripts to indicate different data points in the set.

2.2   Error Statistics

The goal is to find an approximation that closely matches the original parameterization when used in the GCM. Of course, one cannot afford to test this in the development stage of an approximation, but one can test the quality of the approximation using statistical means. The procedure is usually as follows: we generate two data sets. The training data set $X = \{(x^i, y^i), i = 1, \ldots, N\}$ is used to train the parameters of the approximation, for instance the weights of the neural net or the underlying partition of the tree algorithm. Then the test data set is used to measure the bias and residual mean square errors to achieve a statistical evaluation of the approximation accuracy.

We denote the test data set with $\hat{X} = \{(\hat{x}^i, \hat{y}^i), i = 1, \ldots M\}$. Furthermore we indicate an emulation of the original parameterization with $\tilde{f}$ and its output $\tilde{f}(\hat{x}^i)$ by $\tilde{y}^i$. Using these conventions, the bias or systematic error of the approximation for the l-th layer is defined by

$$B_l = \frac{1}{M} \sum_{i=1}^{M} \hat{y}_l^i - \tilde{y}_l^i. \tag{1}$$

and the total bias is

$$B = \frac{1}{D} \sum_{l=1}^{D} B_l \tag{2}$$

where D is the number of output components. Since the heating rates and the heat fluxes in the output vector are measured in different units, it hardly makes sense to compute error measures for all 33 output components simultaneously; instead we restrict ourselves to the heating rates, *i.e.*, the first 26 components. In this case D can be considered as the number of layers. The bias is actually not a measurement of accuracy (it could be zero even for an arbitrarily inaccurate approximation), but a good approximation with a large bias would be unacceptable, because one has to fear that systematic deviations could accumulate in the GCM.

We measure accuracy using the root mean square error (RMSE). The RMSE of the l-th output component is defined as

$$\mathrm{RMSE}_l = \sqrt{\frac{1}{M} \sum_{i=1}^{M} (\hat{y}_l^i)^2 - (\tilde{y}_l^i)^2} \tag{3}$$

and the total RMSE is

$$\mathrm{RMSE} = \sqrt{\frac{1}{D} \sum_{l=1}^{D} \mathrm{RMSE}_l^2}. \tag{4}$$

2.3   Neural Network Approximations

A natural benchmark for comparison with the present is the neural network emulation described in [KRC05]. This is a standard feed-forward neural net with one hidden layer of neurons of the form

$$\tilde{f}(x) = \alpha_0 + \sum_{i=1}^{m} \alpha_i \, \sigma(\beta_i \cdot x + \gamma_i) \tag{5}$$

where σ is the sigmoidal function $\sigma(x) = \tanh(x)$, and the directions $\beta_i \in \mathbb{R}^d$, weights $\alpha_i \in \mathbb{R}^{d'}$ and intercepts $\gamma_i \in \mathbb{R}$ are learned from the training data by least squares fitting. In particular, we use as reference a network with $m = 80$ hidden neurons which has been trained with 196,608 selected evaluations of the original parameterization from a reference run of the NCAR-CAM simulating the climate for the years 1961–1962.
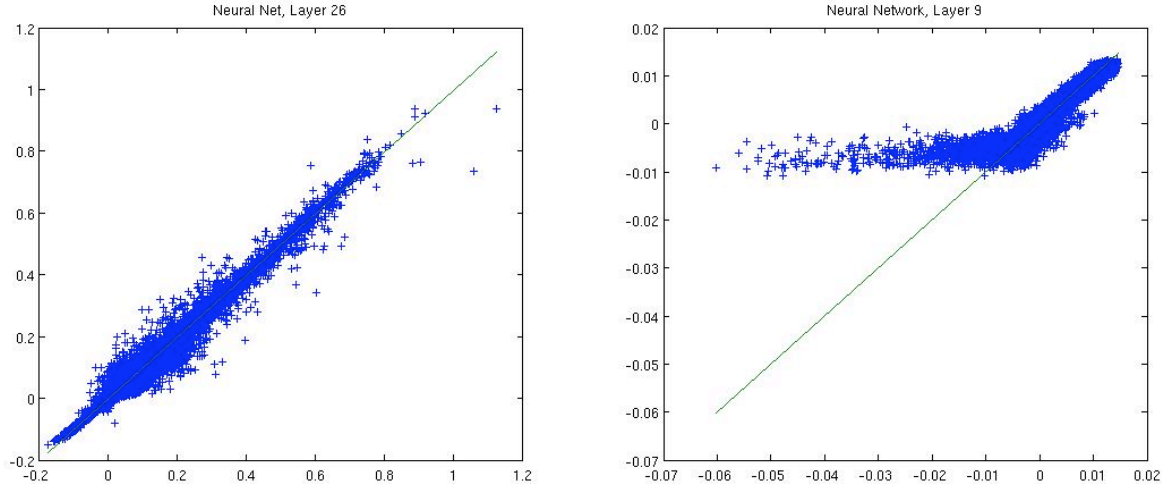
Figure 1: Layer Bias. Scatterplots of the neural network approximation for the layers $l = 26$ and $l = 9$. In each diagram the values of the original parameterization f_l are plotted along the horizontal axis, the approximated values $\tilde{f}_l$ along the vertical axis. The green line shows the diagonal $f_l = \tilde{f}_l$.

2.4 Monolithic Schemes vs. Batteries

Note, that the above representation can be described as a monolithic, vector-valued approach. Another possible network design could have been

$$\tilde{y}_j = \alpha_{0,j} + \sum_{i=1}^{m} \alpha_{i,j}\, \sigma(\beta_{i,j} \cdot x + \gamma_{i,j}) \tag{6}$$

with $\beta_{i,j} \in \mathbb{R}^d$ and $\gamma_{i,j} \in \mathbb{R}$ as above but $\alpha_{i,j} \in \mathbb{R}$. That is, one could have used a battery of neural networks approximating each output component individually. This would, of course, increase the training and evaluation costs, but would also be more accurate. Actually, the vector-valued design potentially can become the source of bias. In the training of the neural network, the total RMSE is used as the optimization criterion. However, the variance of the output data is higher for the layers nearer to the surface of the earth, hence the error is dominated by these layers and the neural net is adapted mainly to them. In other layers, the errors might be relatively larger. This is demonstrated in Figure 1 which shows scatterplots of the reference neural net for the layers 9 and 26. One can clearly see that the approximation in layer 9 is biased. One has to note however, that the scales in the two diagrams differ by almost a factor of 30, *i.e.*, the absolute error introduced by this bias is still very small. A possible remedy for this is to normalize the outputs in the learning process according to their variances. Then, all output components will have the same relative accuracy, possibly at the cost of the total accuracy. This technique could also be applied to the tree-based methods considered in the current paper.

Since we are interested in examining the potential of possible alternatives to the current emulation design, we consider mainly, but not exclusively, component-wise approximationin the following. The disadvantages of component-wise approximations

6

are that they might become computationally more expensive and can potentially return unrealistic profiles, because they may not properly represent the correlations between components of the profile vector. These correlations are naturally represented by vector-wise approximation.

3 Description of Algorithms

In order to keep this paper self-contained we give a concise description of the non-parametric algorithms that we will consider in the following numerical experiments. Thereby, we discuss the nearest neighbor and regression trees only very briefly, because they are well established and comprehensively discussed in the literature. We give a more comprehensive account of the sparse occupancy trees, because, as explained in the introduction, they are new and have been developed specifically for this application.

3.1 (Approximate) Nearest Neighbors

3.1.1 Basic Concepts

The k-nearest neighbor method works as follows: one defines a metric $\|\cdot\|$ on the input space and given a query point x finds a permutation $i \to i_n$ of the training data such that

$$\|x^{i_1} - x\| \leq \|x^{i_2} - x\| \leq \ldots \leq \|x^{i_k} - x\| \leq \|x^{i_p} - x\|$$

for all $p > k$. Then, one averages the function values corresponding to these nearest neighbors

$$\tilde{f}(x) = \sum_{n=1}^{k} y^{i_n}.$$

to define an approximation of $f(x)$. Unfortunately, it is well known that in very high dimensions it is not possible to design fast algorithms that provide the permutation sought. Instead one relaxes the search and is content with an algorithm that returns points $\bar{x}^{i_n}$ such that

$$\|\bar{x}^{i_n} - x\| < (1 + \varepsilon)\|x^{i_n} - x\|.$$

There are algorithms based on kd-trees or bd-trees that provide a fast answer to this relaxed problem, if ε is chosen large enough. An introduction to this topics can be found in [Wen05].

3.1.2 Data Scaling

The central point in the above description is, of course, how to define the metric $\|\cdot\|$ on the input space. This is a non-trivial task because the input vectors include several physical quantities measured in different units and are varying over several orders of magnitude. A trivial method to equilibrate the various input quantities is to compute the maximum and minimum of each parameter in the training data set and to then scale this each component of the input vector individually to the interval $[0, 1]$. Then, one uses the standard Euclidian norm on $[0, 1]^d$ to measure the distances. Another self-evident idea is to scale the variables belonging to the same profile with the same factors. Numerical experiments showed that the second type of scaling yields better

results. Therefore, we use this scaling in the following experiments. Adaptive nearest neighbor methods try to learn a problem dependent metric from the data, but we have not pursued this approach any further, because the data seems to be too sparse to define local metrics reliably for this application.

3.2 Regression Trees

3.2.1 CART

The most basic algorithm for the generation of regression trees is the Classification and Regression Tree (CART) algorithm intensively analyzed in [BFOS84]. It can be summarized as follows: we initialize the partition $\mathcal{P} = \{\Omega\}$ where $\Omega = \prod_{i=1}^{d}[a_i, b_i]$ is a hyper-rectangle that contains all the training points and d is the dimension of the input space. Then, each hyper-rectangle in the partition that contains more than a given number m of data points is recursively subdivided along a hyperplane $x_i = c$, where $i \in \{1, \ldots, d\}$ and $c \in [a_i, b_i]$ is chosen such that the RMSE of the best piecewise constant approximation on the refined partition is minimized. That is, the regression function assumes the average value of all the points in a given hyper-rectangle of the partition. This is a reasonably efficient algorithm that can be used for a wide range of classification and regression problems. It also has the advantage that it is independent of the scaling of the data.

3.2.2 Random Forests

The Random Forests algorithm by Breiman [Bre01] averages the response of a given number T of CART trees. Hereby, before each subdivision step in the generation of the CART trees, the algorithm chooses a random subset of size P of the input parameters (typically about one third of all input parameters) along which the cell is allowed to be subdivided. This ensures that each single CART trees generates different partitions of the domain. Usually the single CART trees in a Random Forest are not pruned, *i.e.*, one takes $m = 1$ in the CART-algorithm and deliberately overfits the data. Nevertheless, Random Forest approximations are relatively smooth due to the averaging process. Random Forests are generally considered to be one of the best available black-box non-parametric regression methods.

3.3 Sparse Occupancy Trees

Sparse Occupancy Trees have been developed as an alternative for approximate nearest neighbor methods (see [BDL10]) . They are designed to scale well in high dimensions and to be readily available for online-learning.

3.3.1 Setting

The sparse occupancy algorithms try to exploit basic ideas from multiresolution analysis for high spatial dimension. The main underlying data structure of the multilevel construction is the subdivision tree which is linked to a hierarchy of nested partitions. That is, we assume that for each level $l \geq 0$ the sets $\mathcal{P}_l = \{\Omega_{l,k}, k \in \mathcal{I}_l\}$ are partitions of the input space Ω and each cell $\Omega_{l,k} \in P_l$ is the disjoint union of cells on the next

finer level $l + 1$:

$$\Omega_{l,k} = \bigcup_{r \in \mathcal{I}_{l,k}} \Omega_{l+1,r} .$$

The hierarchy of partitions induces an infinite *master tree* $\mathcal{T}^*$, whose root is Ω and whose other nodes are the cells $\Omega_{l,k}$. Each node $\Omega_{l,k}$ of this tree is connected by an edge to its children $\Omega_{l+1,r}$ where $r \in \mathcal{I}_{l,k}$.

Now let us assume that a training data set $X = \{(x^i, y^i), i = 1, \ldots, N\}$ is given. An occupancy tree $\mathcal{T}(X)$ is a finite subtree of $\mathcal{T}^*$ that contains only the nodes that are occupied by at least one sample x^i from the data set X. In high dimensions occupancy trees are typically sparse, because there are many more cells than data points. Furthermore, the points can be stored in a compact way. In the following two subsections, we present two algorithms that make use of this data structure.

3.3.2 Piecewise Constant Approximation on Cube Subdivisions

A piecewise constant approximation based on an occupancy tree can be defined as follows: given a query point x we average the values of points in the finest cell of the occupancy tree containing the query point. This general idea, which works for any subdivision geometry, is most easily realized for dyadic cube subdivision, because it can be very easily encoded. Assuming that all data are scaled such that they fit into the unit hypercube $[0, 1]^d$ the partitions are given by

$$\mathcal{P}_l = \left\{ \prod_{i=1}^{d} [k_i 2^{-l}, (k_i + 1)2^{-l}], \quad k_i \in \{0, \ldots, 2^l - 1\} \right\} .$$

For any point $x = (x_1, \ldots, x_d) \in [0, 1]^d$ we can write its components in binary representation as

$$x_i = \sum_{k=1}^{\infty} b_{ik} 2^{-k} .$$

Note that the finite sequence $b_{i1}, b_{i2}, \ldots, b_{il}$ is a binary representation of the integer k_i of the level-l cell to which the data point belongs. Assuming that a maximum refinement level L is chosen we compute the bitstream

$$(b_{11}, b_{21}, \ldots, b_{d1}, b_{12}, b_{22}, \ldots, b_{d2}, \ldots, b_{1L}, b_{2L}, \ldots, b_{dL}).$$

for each training and query point. The subsequences of d bits $b_{1l}, \ldots, b_{dl}$ are called characters. The bitstreams of the training points are sorted lexicographically. Then, in order to determine which points one has to average to answer a query, one finds the points whose bitstreams have the most number of leading characters in common with the bitstream corresponding to the query point. This is essentially a binary search.

This algorithm is very fast and storage efficient, because it replaces the input data with bitstreams. Furthermore, it is inherently suited to incremental learning: if one gets a new data point, one computes its bitstream, sorts it into the list of already-existing bitstreams, and then the next query can immediately be accepted. Conceptually this code is related to nearest neighbor approximation because it only considers proximity of the points in the input space, using the similarity of the bitstreams as metric. Unfortunately, this recovery scheme is by far not as accurate as the nearest neighbor scheme because of the tree structure. That is, for any two points $x, x' \in X$ the

9

tree distance $\text{dist}_{\mathcal{T}}(x, x')$ is the shortest path in the tree $\mathcal{T}(X)$ connecting the nodes $\Omega_{L,k}(x) \ni x$ and $\Omega_{L,k'} \ni x'$. Of course, whereas $||x - x'||$ may be arbitrarily small for any fixed norm $|| \cdot ||$ on $\mathbb{R}^d$, the tree distance $\text{dist}_{\mathcal{T}}(x, x')$ could be $2L$. The above recovery scheme takes local averages of function values whose tree distance is small, possibly omitting values for arguments that are geometrically very close. There are several possible remedies. Since the recovery scheme is very fast, perhaps the simplest one is to perform several different recoveries with respect to different shifts of the coordinate system (chosen randomly) and then take the average of the outputs. For example, in our implementation we scale the data to the interval $[0.3, 0.7]$, and then shift the data with random vectors in $[-0.3, 0.3]^d$. Let $\tilde{f}_\rho(x)$ denote the result of a query at x with the data shifted by the vector ρ and $X_\rho(x)$ be the corresponding set of training points in the leaf of the sparse occupancy tree containing x. Furthermore, let $R(x)$ be the set of shifts ρ for which the level of the evaluation is maximal. Then

$$\tilde{f}(x) = \frac{1}{\#(R(x))} \sum_{\rho \in R(x)} \tilde{f}_\rho(x) \ . \tag{7}$$

With a moderate number of random shifts, one can usually achieve an accuracy similar to that of the nearest neighbor method.

3.3.3   Sparse Occupancy Trees Using Simplices

Overcoming the tree-distance problem motivates another approach that constructs piecewise linear approximations on simplex subdivisions. In the following description we leave out the technical details, which can be found in [BDL10]. We base the approximation on a hierarchy of partitions generated by dyadic simplex subdivision. That is, each simplex S in the partition $\mathcal{P}_l$ on level l is subdivided into 2^d simplices (its children) on level $l + 1$.

For this purpose we use a scheme described in [Tra97] which is based on recursive binary subdivision. In each binary subdivision step one edge of the current simplex is chosen and subdivided at its midpoint. To be more precise let us denote with $(x_0, x_1, \ldots, x_d)_g$ a simplex that arises from a simplex in the dyadic tree by g binary subdivisions. Then this simplex is divided into the two subsimplices

$$\left(x_0, \frac{x_0 + x_d}{2}, x_1, \ldots, x_g, x_{g+1}, \ldots, x_{d-1}\right)_{g+1}$$

and

$$\left(x_d, \frac{x_0 + x_d}{2}, x_1, \ldots, x_g, x_{d-1}, \ldots, x_{g+1}\right)_{g+1}$$

where the sequences $(x_{g+1}, \ldots, x_{d-1})$ and $(x_1, \ldots, x_g)$ should be read as void for $g = d - 1$ and $g = 0$, respectively, and the sequence $x_{d-1}, \ldots, x_{g+1}$ features decreasing indices.

Given x, we denote by $S_l(x)$ the simplex at level l which contains x and given any simplex S, we denote its set of vertices by $\mathcal{V}(S)$. If v is a vertex of a level-l simplex in the master tree, then $\mathcal{S}_l(v)$ denotes the set all level-l simplices that share v as a corner point. We also define $\mathcal{V}_l$ to be the set of all vertices at level l of occupied cells.

In the training stage we compute the values

$$y_l(v) = \text{Average}\{y^i : x^i \in \mathcal{S}_l(v)\}$$

for each vertex of a simplex on level l in the occupancy tree. In the evaluation stage, if we are given an input x, we first determine the maximum level l such that $\mathcal{V}(S_l(x)) \cap \mathcal{V}_l \neq \emptyset$ and then we compute

$$\tilde{f}(x) = \frac{\sum_{v \in \mathcal{V}(S_l(x)) \cap \mathcal{V}_l} \tau(S_l(x), v, x) y_l(v)}{\sum_{v \in \mathcal{V}(S_l(x)) \cap \mathcal{V}_l} \tau(S_l(x), v, x)}. \tag{8}$$

where $\tau(S, v, x)$ is the barycentric weight of the point x with respect to the vertex v of the simplex S.

To summarize: the value of the query point is found by interpolating vertex values of the simplex containing it. Hence, the query response becomes an average of all training points in the neighborhood of the query coordinates, even including the points in simplices that are far away in tree distance. Again, note that this algorithm is suited for incremental learning: if one gets a new sample, one just computes its place in the occupancy tree and adds its value to all adjacent vertices. Any query after that can immediately use the new information.

4   Numerical Experiments

In this section we present the results of three groups of numerical experiments. In the first two subsection, the sets of training data and test data each contain 196,608 data samples collected during a reference climate simulation for the years 1961–1962 using the original parameterization. In the first subsection we compare the regression trees with the benchmark neural network approximation. Note that this comparison tends to overestimate the advantages of the neural network. First of all, it does not reflect the training time, which is about a week for the neural network, but only between a few seconds and less than a few hours for the tree-based methods. Second, whereas the neural network would profit only slightly from taking more training data (its accuracy is basically limited by the number of neurons), the non-parametric methods benefits significantly from allowing more data, and limiting the training data size is artificial and unnecessary. Nevertheless, we perform the comparison in this form, because it's the same training data we will use for the experiment in Section 5 where we have to comply with memory limitations. As it turns out, nearest neighbor methods and sparse occupancy trees do not deliver competitive accuracy, if applied naïvely, but their performance can be enhanced by dimension reduction. We demonstrate this in some experiments presented in subsection 4.2. Finally, the last experiment presented in section 4.3 gives some insight into the internal structure of the data. Here, we try to obtain an estimate of the intrinsic dimension of the input data.

4.1   Comparison of Approximation Errors

In Figure 2 we see the RMSE profiles (left) and the bias profiles (right) for the following methods:

1. The benchmark neural network emulation (see Section 2.3, blue line).

2. The approximate nearest neighbor approximation (ANN, with $k = 5$, $\varepsilon = 1$, red line), where we used the profile-wise input scaling.

3. A single vector-valued regression tree (CARTV, $m = 5$, cyan line).

4. One regression tree for each output component individually (CARTC, $m = 5$, magenta line).

5. a vector-valued Random Forest approximation (RFV, $T = 20$, $P = 80$, green line), and

6. an approximation where we compute a Random Forest (RFC, $T = 20$, $P = 80$, black line) for each component individually.

In Table 1 we also give the total RMSEs and bias for all these methods.
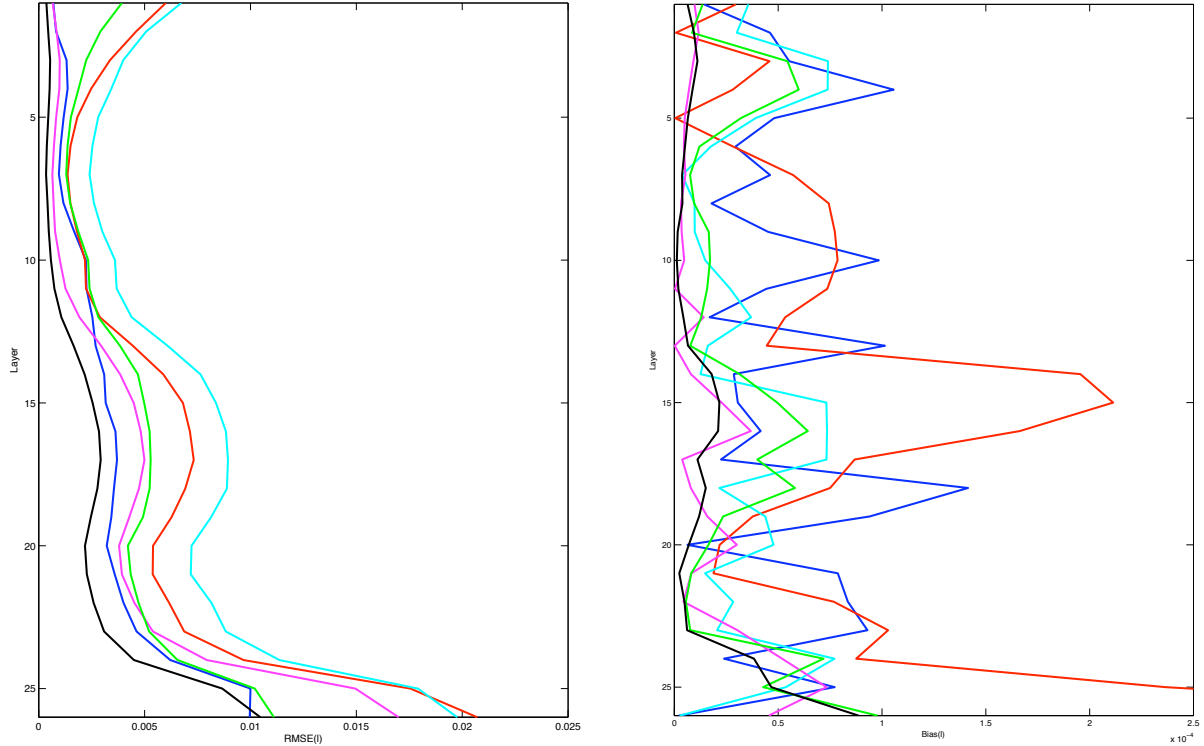

Figure 2: Comparison of neural network, approximate nearest neighbor, and several regression tree emulations. Left: layer-wise root mean square errors. Right: layer-wise absolute values of Bias.

Some major observations to be taken from this Figure and Table can be summarized as follows:

1. Nearest Neighbors do not deliver competitive accuracy, if applied directly to the 220-dimensional input data. It is however surprising that the vector-valued CART does not yield a better result, even though it generates an adaptive partition of the input domain. One needs to use ensembles of regression trees to achieve good approximation accuracy.

2. It is possible to improve on the neural network emulation with moderate computational effort. The generation of regression trees is cheap, so even the generation of the 520 trees for RFC(20,80) takes only a few hours (7h on a single 2.2 Ghz

12

Method	RMSE (J/kg/s)	Bias (J/ks/s)
Neural Net	$3.94836 \cdot 10^{-3}$	$3.11643 \cdot 10^{-6}$
ANN	$7.26535 \cdot 10^{-3}$	$2.70421 \cdot 10^{-5}$
CARTV	$8.17753 \cdot 10^{-3}$	$1.27022 \cdot 10^{-5}$
CARTC	$5.54573 \cdot 10^{-3}$	$1.26559 \cdot 10^{-6}$
RFV(20,80)	$4.75692 \cdot 10^{-3}$	$6.08371 \cdot 10^{-6}$
RFC(20,80)	$3.27711 \cdot 10^{-3}$	$3.99269 \cdot 10^{-7}$

Table 1: Total RMSE and absolute value of total bias for emulation with neural network, approximate nearest neighbors, CART and Random Forests applied to the whole vector or componentwise. Training and test data each consist of 196,608 evaluations of the original parameterization.

AMD-Opteron processor) on a standard PC. However, due to its storage requirements ($26 \cdot 20 = 520$ trees have to be computed) this result is not of great practical interest. The two practical competitors are the CARTC and the RFV emulations, which use 26 trees (one for each component) or 20 trees, respectively. RFV seems to be a little bit more accurate, but CARTC has a lower bias, for the reasons we already exposed in Section 2.4. To demonstrate the latter point we show in Figure 3 scatterplots for both emulations. The component-wise CART approximation clearly has a higher variance in layer 26, but delivers good, unbiased approximation in layer 9.

3. Notice that except for the somewhat inaccurate nearest neighbor approximation, the neural network approximation exhibits the most biased approximation.

Finally, in Figure 4 we show the emulated heating rates for three representative profiles in order to compare the vector-valued random forest and the componentwise CART approximation. In general CART very accurately follows the profile of the original parameterization. However, in extraordinary cases it can overshoot, which is most noticeable in the third graph. The random forest approximation has the tendency to flatten out the original profiles but does not produce extreme outliers.

4.2 Performance of the Sparse Occupancy Trees

In the previous section we have shown that the nearest neighbor method is not as accurate as the benchmark neural network. This result is inherited by the sparse occupancy schemes which, as explained in Section 3, are conceptually similar and do not improve on the nearest neighbor approximation, but rather try to mimic it with data structures that allow faster processing of large, incrementally growing data sets. This is confirmed by the numbers given in Table 2, which, in particular, shows how increasing the number of random shifts converges towards the quality of the original nearest neighbor approximation.

The reason for the unsatisfactory results of both the piecewise constant simplex and the piecewise linear vertex algorithm is revealed in Table 3, which shows the level of resolution at which the test queries are evaluated. In the case of the simplex scheme almost all the evaluations take place on level 0, which indeed means that most of
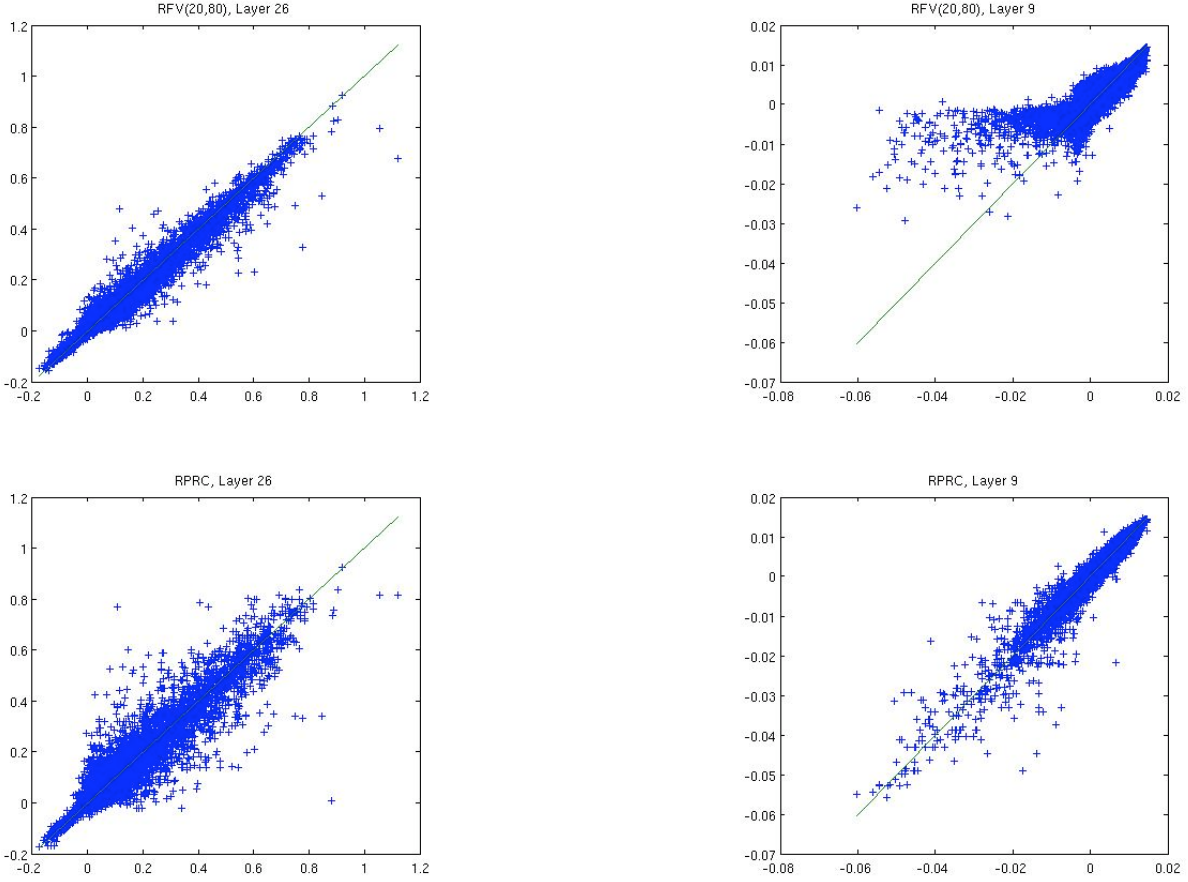
Figure 3: Scatterplots for the approximation of the heating rates in the 26th and 9th vertical layer with componentwise CART (RPRC) and the vector-valued Random Forest (RFV) approximation.

Single Trees			
	Dyadic cubes	Binary cubes	Dyadic simplices	Vertex method
RMSE	0.0147266	0.0132963	0.0200573	0.014531
Random Shifts with Dyadic Cubes			
Shifts	1	10	100	1000
RMSE	0.0147266	0.00937559	0.00835964	0.00806051

Table 2: RMSE of Sparse Occupancy Methods

the evaluations just return the global average. The reason for this is, that simplex subdivision does not exploit the properties of the input data. The input variables are highly correlated. Hence, if within one of the cube-subdivion schemes a split along one variable does not separate two data points, with high probability the subsequent split along the next input variable will also not separate the points. However, in a simplex grid the split lines are oblique to the coordinate lines and therefore every split will
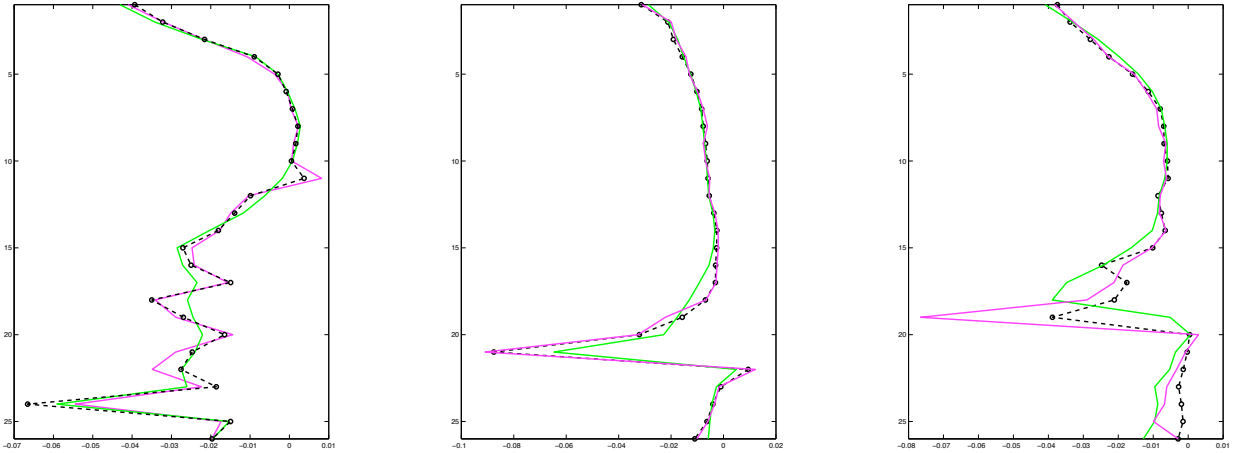
Figure 4: Approximation of three representative heating rate profiles. Black line with markers: original parameterization. Magenta: Component-wise emulation. Green: vector-valued Random-Forest emulation. Heating rates units are J/kg/s.

separate points with high probability. It is an open question whether this issue can be resolved by transforming the data in a suitable way before starting the subdivision process.

Level	Simplices	Cubes-1	Cubes-10	Cubes-100	Cubes-1000
0	97.998	44.0552	2.13725	0.208537	0.00203451
1	1.68864	33.4686	35.9996	25.6978	18.8019
2	0.217692	14.7217	22.8271	25.5249	26.9145
3	0.0701904	5.57658	17.9555	18.5211	19.5536
4	0.0203451	1.5350	12.4695	14.7339	15.3971
5	0.00406901	0.455729	5.87667	9.75138	10.0141
6	0.00101725	0.142415	1.88293	3.90828	5.88277
7		0.0386556	0.581868	1.10779	1.74052
8		0.00508626	0.18514	0.37028	0.519816
9		0.00101725	0.0640869	0.127157	0.18514
10			0.0203451	0.0457764	0.0742594
11				0.00305176	0.142415

Table 3: Relative frequencies (in percent) of the evaluation at a given dyadic level for the sparse occupancy trees and the random shift method.

However, the situation is not as bleak as it might look like from this result. As has become clear in the previous subsection, even regression trees do not yield very good results singly, but rather one needs ensembles of them to achieve high accuracy. We need a different mechanism for the generation of nearest-neighbor ensembles, though.

In this case we employ dimension reduction. The basic idea is that not all input parameters are equally relevant for all output parameters, and "superfluous" parameters have a negative effect on the quality of the nearest neighbor approximation. A possible remedy could be as follows: by some statistical analysis one tries to determine which input variables are relevant for the computation of a given output component. Then one applies nearest neighbor methods or sparse occupancy trees for each output component using only the assigned "relevant" variables. With this intention, during the runs of the CART algorithms documented in the previous section we collected some statistics about the variables on which the splits were performed and how much the splits along these variables improved the results. Then, we chose only the 15 variables for which the splits had been most effective as input parameters for each output component. This is only an ad-hoc solution, but it proved quite effective, as shown in Table 4. It should be added that the approximate nearest neighbors method yields a RMSE of 0.00521976 using this approach.

Single Trees				
	Dyadic Cubes	Binary cubes	Dyadic simplices	Vertex method
RMSE	0.00736002	0.00737307	0.0127928	0.00708333
Random Shifts of Dyadic Cubes				
Shifts	1	10	100	1000
RMSE	0.00736002	0.00580438	0.00550121	

Table 4: RMSE of Sparse Occupancy Methods

Unfortunately, even in this setting the methods based on simplex subdivisions do not perform as well as one could have expected considering the promising results presented in [BDL10] . Nevertheless, sparse occupancy trees together with dimension reduction methods seem to be another viable way of constructing emulation for the LWR parameterization.

4.3   A Convergence Study

Finally, we use a database of 5 million points from the period 1960–1963 for a convergence study. For this purpose we use CART and the Sparse Occupancy Tree method, because the other methods like random forest or nearest neighbors are too expensive computationally. We randomly divide the data points into 33 approximately equally sized subsets. The first of these subsets is used a test data set. Then we perform each algorithm using 1,2,4,8,16 or all 32 of the remaining sets as training data. The results of this experiment are listed in table 5. The database contains the evaluations of the original parameterization for every 10th day during a 4-year run (1960–1963) of the global climate model which used 6-hour timesteps and a grid on the globe with 128×64 grid points. Standard estimates for piecewise constant approximation predict that for a smooth function $y = f(x)$ the L_2-error on a non-adaptive partition is proportional to $N^{-1/d}$. Therefore, we use as the measure for the convergence speed the quantity

$$d' = \frac{\log(N_2/N_1)}{\log(RMSE_1/RMSE_2)} \tag{9}$$

16

where N_1, N_2 are the number of training points and $RMSE_1, RMSE_2$ are the calculated root mean square errors of two experiments with the same numerical scheme. In the table we list the values for d' in the case one takes N_1 and N_2 from two consecutive rows in the table. We expect values for d' which are near to the intrinsic dimension of the input data set. Actually numerical experiments with other data sets indicate that the numerical d' computed with Sparse Occupancy Trees slightly underestimates the true dimension. Hence, from this data we conclude that the convergence behavior is similar to what one would expect for data living on submanifold of dimension at most 1 in the input space.

N	OccBin	d'	RPR-C	d'	RPR-V	d'
145911	1.4755		0.6319		0.8961	
292650	1.3648	8.924	0.5737	7.203	0.8459	12.072
586879	1.2635	9.023	0.5313	9.063	0.7978	11.886
1174907	1,1720	9.062	0.4812	7.008	0.7411	9.415
2351709	1.0856	9.062	0.4428	8.344	0.6940	10.586
4702660	1.0055	9.041	0.4044	7.639	0.6467	9.817

Table 5: RMSE for NCAR-CAM Data

5 Results of 10-Year Climate Simulation

Finally, we try to asses the impact of using a tree approximation of the LWR parameterization in a climate simulation. Therefore, we run the NCAR CAM climate model for 10 years using the original parameterization, the neural network emulation and the component-wise CART emulation. As discussed above, this choice of tree design is debatable, since we could have achieved a more stable and more accurate (in terms of the RMSE) approximation using a vector-valued random forest design. However, the CART-design was the best available method at the time the experiment was set up. The relatively small training data set with its 196,608 samples was used because the parallel simulation was performed on a distributed memory systems, where each processor could only address 4GB of memory, the emulation had to be stored on each processor, and on each processor most of the memory had to be reserved for other parts of the simulation. Therefore, we do not give any numbers about the achieved speed-up of the GCM, although even under these imperfect conditions, the speed up was still considerable.

One of the most desirable properties of an emulation is the preservation of the time means of the prognostic and diagnostic fields. In [KRC05] it has been shown, that neural network emulations reliably achieve this aim. The CART emulation in general produces good agreements, too, but it also seems to be more prone to produce local instabilities. As an example we consider the annual zonal means of the LW radiation heating rates (QRL) in Figure 5. Whereas the plots in the left column seem to be in very good agreement, the difference plots in the right column reveal that the CART approximation causes significant differences in the forecast in the lower atmospheric layers near the polar regions.
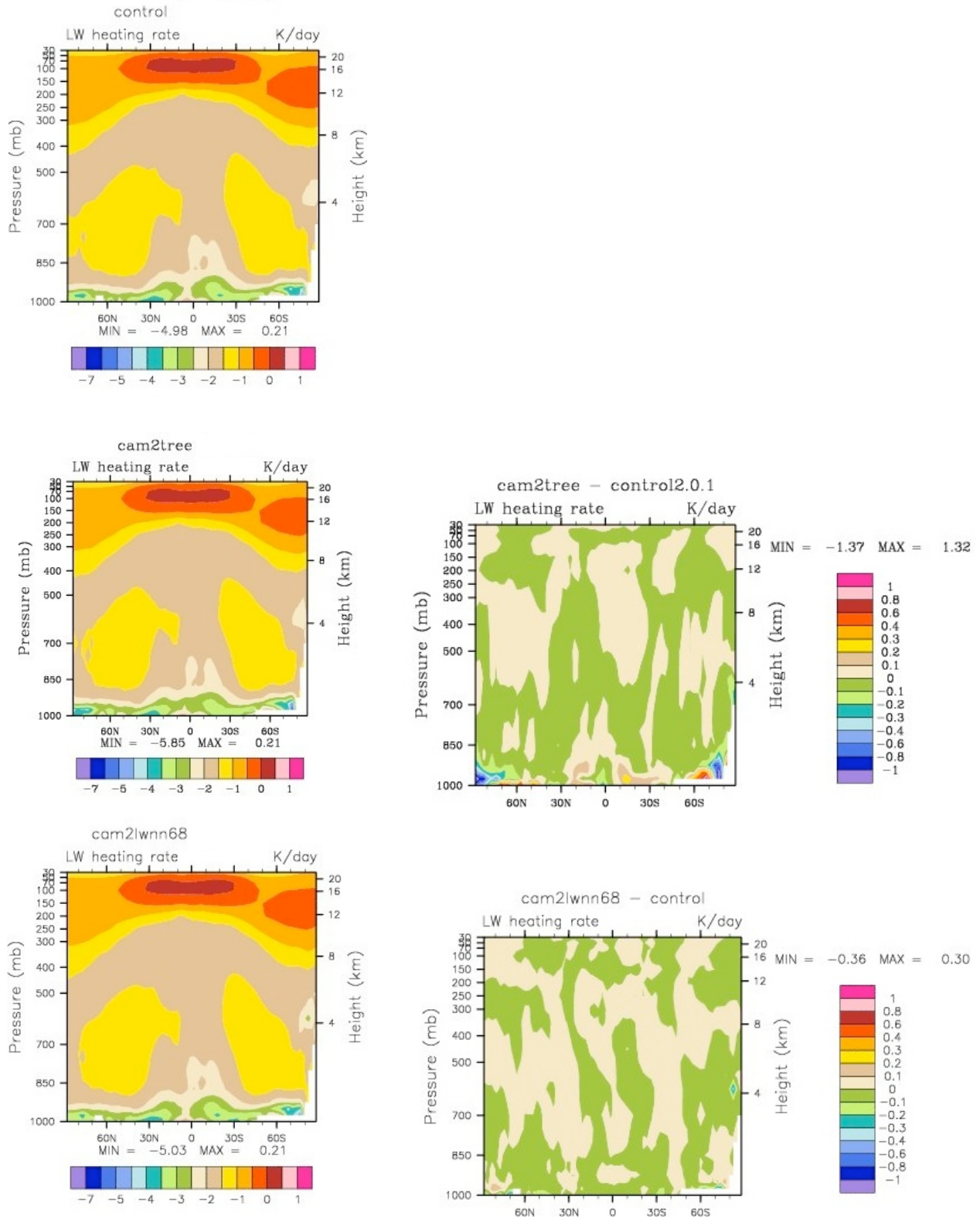
17

Figure 5: Comparison of the predicted annual zonal means of the LWR heating rates computed with the original parameterization (top row), a tree based emulation (center row) and a neural network emulation (bottom row). The right column plots the difference between the simulation and the control.
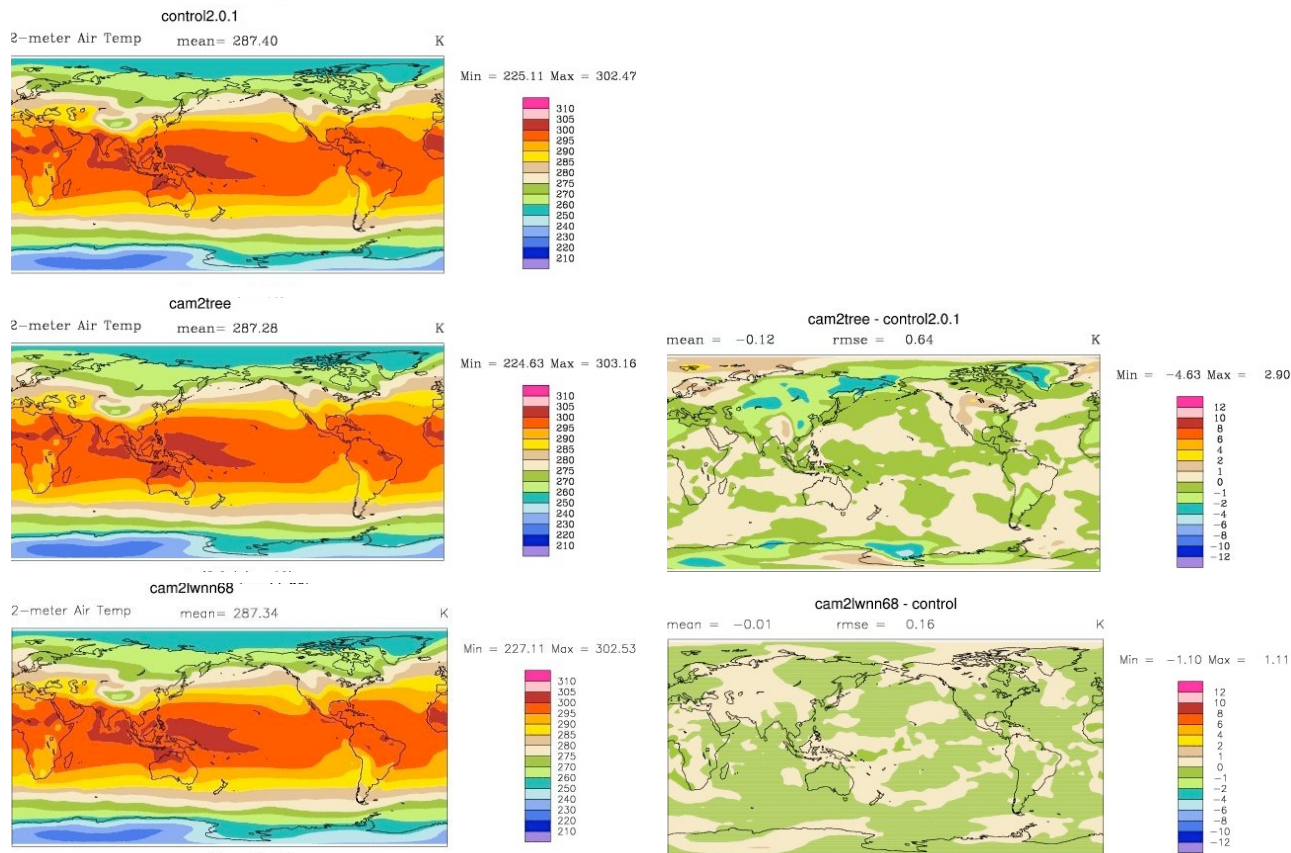
Figure 6: Comparison of the predicted annual means of the two meter air temperatures computed with the original parameterization (top row), a tree-based emulation (center row) and a neural network emulation (bottom row). The right column plots the difference between the simulation and the control.

Basically, the same observation can be made, if we look at the annular means of the two-meter air temperature in Figure 6. The agreement of the control run with the tree-emulation run is satisfactory, but a comparison of the difference plots in the right column reveals that the neural network run is closer to the original parameterization. Again, we see that the largest differences occur in the polar regions.

For this reason we checked the approximation accuracy for all test data samples stemming from these regions separately. It turned out that the RMSE's were much worse for these points than for other regions of the earth, and the CART emulation was biased towards predicting higher heating rates than the original parameterization. The reason for this seems to be that the extreme weather conditions at the poles are represented only by a small fraction of all training samples. As remedy for this problem one can train separate approximation modules for different regions on the earth, or balance the distribution of the training such that the statistical approximation error is equally distributed over the whole globe. The neural network approximation seems to be more reliable with regard to the generalization to rare states. The vector-valued random forest approximation also seems to be stable in this sense.

6 Concluding Remarks

We reported on numerical experiments investigating the possibility of substituting physical parameterizations in global climate models with non-parametric emulations. The results are positive in the sense that they show that both nearest neighbor type methods and regression trees are in principle able to achieve statistical approximation quality on par with neural networks, even if trained with a relatively moderate amount of data. Convergence studies showed that a basic precondition for the nearest neighbor type approximations is fulfilled: namely that the data is concentrated in a small enough region of the whole input space to allow for reasonable convergence in terms of the size of the training data. Special measures have to be taken to ensure error control in underrepresented regions, though. It has been demonstrated that the NCAR CAM with a tree-based LWR emulation gave results in good agreement with the calculation using the original parameterization, except in the polar regions, which could have been expected from the statistical properties of the approximation. Some ideas have been presented for designing new tree-based approximation schemes that offer the opportunity of true incremental learning and automatic adaption to new climate conditions. The main obstacle for the practical use of non-parametric methods is less a mathematical one, but rather one of implementation. Non-parametric approximation methods are memory-based, *i.e.*, they need to store all the training data permanently. This makes its use in a parallel environment more difficult than is the case for the relatively compact neural network representation. Of course, for huge, complex projects like climate simulation software, implementation issues are a major concern. Therefore, the ideas and results presented in the current paper can only be considered as preliminary step towards a new emulation paradigm. As such, in the opinion of the authors, they show very good potential for future developments.

Acknowledgments

References

[BDL10]   Peter Binev, Wolfgang Dahmen, and Philipp Lamby. Fast high-dimensional approximation with sparse occupancy trees. Technical Report 10:03, Interdisciplinary Mathematics Institute, University of South Carolina, Columbia, SC, 2010. Accepted for publication in Journal of Computational and Applied Mathematics.

[BFOS84]   Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[Bre01]   Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[CFCC98]   F. Chevallier, N.A. Scott F. Chéruy, and A. Chedin. A neural network approach for a fast and accurate computation of longwave radiative budget. *J. Appl. Meteor.*, 37:1385–1397, 1998.

[DOP97]   R. DeVore, K. Oskolkov, and P. Petrushev. Approximation by feed-forward neural networks. *Annals of Numerical Mathematics*, 4:261–287, 1997.

[Hor89]   Kurt Hornik. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[HTF09]   Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[KFRTB08] Vladimir M. Krasnopolsky, Michael S. Fox-Rabinovitz, Hendrik L. Tolman, and Alexei A. Belochitski. Neural network approach for robust and fast calculation of physical processes in numerical environmental models: Compound parameterization with a qulaity control of larger errors. *Neural Networks*, 21:535–543, 2008.

[KRC05]   Vladimir M. Krasnopolsky, Michael Fox Rabinovitz, and Dmitry Chalikov. New Approach to Calculation of Atmospheric Model Physics: Accurate and Fast Neural Network Emulation of Longwave Radiation in a Climate Model. *Monthly Weather Review*, 133:1370–1383, 2005.

[Tra97]   C.T. Traxler. An Algorithm for Adaptive Mesh Refinement in n Dimensions. *Computing*, 59:115–137, 1997.

[Wen05]   Holger Wendland. *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005.