Online Nonlinear System Identification in High Dimensional Environments

Duncan Potts and Claude Sammut
The University of New South Wales, Australia
{duncanp,claude}@cse.unsw.edu.au

Abstract

This paper compares a range of techniques from both the system identification and machine learning literature that can construct nonlinear models online. Strong parallels exist between the various representations giving insights into which methods can scale up to a high number of dimensions. It is found that tree-based algorithms scale well, are easy to implement, and require minimal prior knowledge. The algorithms are empirically tested on a number of tasks ranging from the modelling of a simple illustrative test function up to the identification of complex high-dimensional aircraft dynamics.

1 Introduction

A standard method for identifying systems is to compile a set of example data containing samples of the input and output signals. A batch process can then use this data set to optimise various model parameters. Sometimes it is not possible to obtain this data set in advance, or there is too much data for the batch process, in which case some form of online algorithm is required that can update the system model as more data arrives. For example an autonomous system may have an approximate initial model of its environment, but would benefit from updating this model in an online fashion as new information is received from its sensors.

In this paper it is assumed that a discrete time dynamic environment can be fully described by the time-invariant state-space model

$$\mathbf{z}_{k+1} = f(\mathbf{z}_k, \mathbf{u}_k) \tag{1}$$

where $\mathbf{z}_k$ is the n-dimensional state vector and $\mathbf{u}_k$ is the m-dimensional action vector at time k (this is a simple extension of continuous time representations such as in [Slotine and Li, 1991] where the input is constant between sampling times). Furthermore it is assumed that the entire state and action vectors can be sensed (with observation noise) at each sampling instant k. The system identification task is to find a sufficiently accurate approximation to the multiple-input multiple-output (MIMO) function f given any available prior knowledge and a sequence of observations of the system's behaviour.

Common classical methods of system identification include analysing either the frequency response of the system to sine waves of different frequencies or the time-domain response to impulse or step inputs. These techniques only apply to linear systems and are not suited to the MIMO domains in which we are interested. If the underlying system is linear and the loss function being minimised is chosen to be a sum of the squared errors, the problem is a linear optimisation and many well known batch and online solutions exist. However most interesting real world domains exhibit some degree of non-linearity and system identification becomes significantly harder.

The identification of nonlinear MIMO models usually involves a transformation of the inputs such that the model f is linear in the new feature space, and standard online learning techniques can be applied [Ljung, 1987]. This transformation, however, requires detailed prior knowledge of the types of non-linearity present. When there is minimal prior knowledge a number of standard techniques from the literature can still be applied in an online manner, although each suffers from particular drawbacks.

This paper compares several methods from the system identification and machine learning literature that can construct nonlinear models online. With the exception of neural networks, strong parallels between the different representations are observed, whereas the learning algorithms themselves are very different. These insights indicate which techniques can scale to higher dimensional problems, and the algorithms are empirically tested on a simple 2D test function, a pendulum on a cart, and on the identification of complex aircraft dynamics.

It is often advantageous to decompose the large MIMO

problem (1) into n multiple-input single-output (MISO) problems where the outputs are the components of $\mathbf{z}_{k+1}$ [Nelles, 2001]. Each of these tasks can be formulated as a regression problem

$$y = f(\mathbf{x}) + \epsilon$$

where $f(\mathbf{x})$ is the corresponding component of $\mathbf{z}_{k+1}$ and $\mathbf{x} = [\mathbf{z}_k^T \ \mathbf{u}_k^T \ 1]^T$ is a $d = n+m+1$ dimensional column vector of regressors (the constant regressor is added to simplify the notation). The observed values y are corrupted by zero-mean noise ϵ with unknown variance σ^2. The online system identification task is to construct an approximation $\hat{f}(\mathbf{x})$ to $f(\mathbf{x})$ that attempts to minimise some cost function (e.g. sum of squared errors) over a sequence of training examples.

All results in this paper show the mean of 20 experiments, with error bars and errors in tables indicating one unbiased estimate of the standard deviation over these experiments. Approximation errors are given as the normalised root mean square error (nRMSE) over an independent set of test data drawn from the same distribution as the training data. In the experiments this gives very similar results to measuring the average difference between predicted and actual states over time.

2 Online Nonlinear System Identification

In this section six techniques from both the system identification and machine learning literature are examined:

1. Neural networks
2. Radial basis functions (RBFs)
3. Locally weighted learning (LWL)
4. Receptive field-weighted regression (RFWR)
5. The online locally linear model tree (LOLIMOT) algorithm
6. Incremental model tree induction (IMTI)

The behaviour of each algorithm is illustrated by learning the simple nonlinear function

$$y = \max\left\{\mathrm{e}^{-10x_1^2}, \mathrm{e}^{-50x_2^2}, 1.25\mathrm{e}^{-5(x_1^2+x_2^2)}\right\} + \epsilon \qquad (2)$$

where ϵ is zero-mean Gaussian noise with variance $\sigma^2 = 0.1$ (see Figure 1).

2.1 Neural Networks

A neural network is built from multiple neurons, each with an activation function $g(.)$. The basic structure contains a hidden layer of H neurons, and a single neuron at the output (Figure 2). The final estimate is therefore

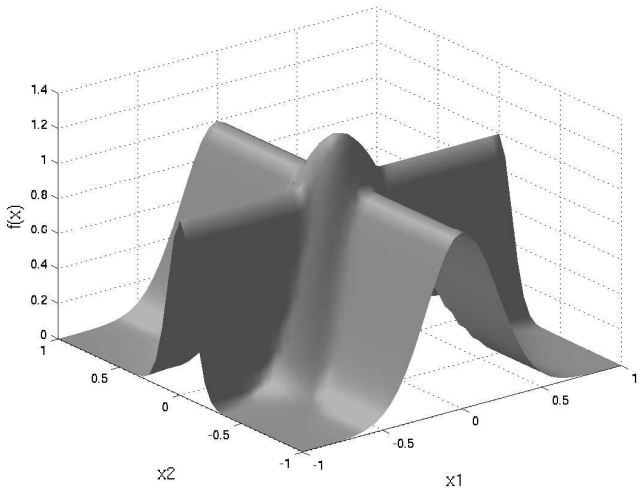


Figure 1: 2D test function.

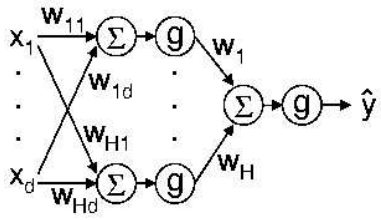


Figure 2: Neural network structure.

$$\hat{f}(\mathbf{x}) = g\left(\sum_{i=1}^{H} w_i g\left(\sum_{j=1}^{d} w_{ij} x_j\right)\right)$$

Although sophisticated training algorithms exist, a basic gradient descent learning rule is used as an illustration

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \alpha \frac{\partial J_k}{\partial \boldsymbol{\theta}_{k-1}}$$

where

$$\boldsymbol{\theta} = [w_1 \ w_1 \ldots w_H \ w_{10} \ w_{11} \ldots w_{1d} \ w_{20} \ w_{21} \ldots w_{Hd}]^T$$

is a vector containing all parameters and

$$J_k = e_k^2 = \left(y_k - \hat{f}(\mathbf{x}_k)\right)^2$$

is the squared prediction error at time k.

Figure 3 shows the decrease in approximation error of such a network with sigmoidal activation functions, 10 hidden units and a learning rate $\alpha = 0.1$. More hidden units or a different learning rate do not improve the results. Convergence is seen to be extremely slow,

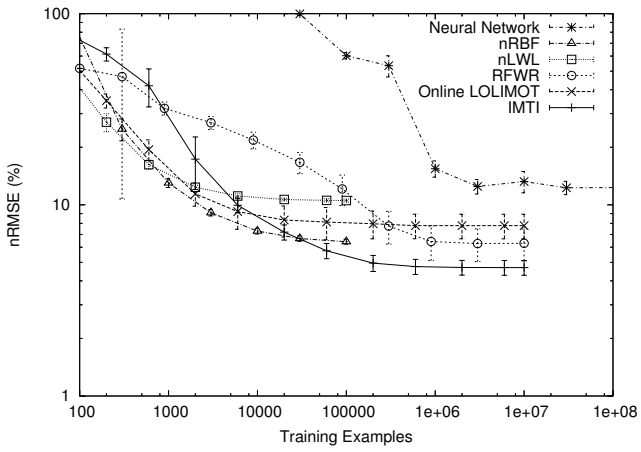


Figure 3: Approximation error on the 2D test function.

Table 1: Model sizes and training times per example for the 2D test function.

Algorithm	Number of local models	Training time (ms)
Neural Network	-	0.02 ± 0.00
nRBF	64 ± 0	362.90 ± 5.21
nLWL	64 ± 0	1.20 ± 0.00
RFWR	92 ± 3	8.71 ± 0.28
Online LOLIMOT	64 ± 0	2.08 ± 0.03
IMTI	64 ± 4	0.82 ± 0.01

highlighting a well known problem with neural networks, and the final asymptotic error is high. In practice neural networks are difficult to implement, offer no guarantee of convergence, and often become stuck in local minima. Although it can be seen from Table 1 that this basic algorithm is extremely fast, the issues associated with neural networks are difficult to overcome.

2.2 Radial Basis Functions

A radial basis function (RBF) approximation is formed by a weighted sum of M local functions $\hat{f}_i(\mathbf{x})$

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{M} \hat{f}_i(\mathbf{x}) \mathbf{\Phi}_i(\mathbf{x}) \tag{3}$$

where the weights $\mathbf{\Phi}_i(\mathbf{x})$ depend on the distance of $\mathbf{x}$ from the centre $\mathbf{c}_i$ of the basis function. The most common RBF is a multi-dimensional Gaussian with shape matrix $\mathbf{D}_i$

$$\mathbf{\Phi}_i(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{c}_i)^T \mathbf{D}_i (\mathbf{x}-\mathbf{c}_i)\right)$$

Normalised RBFs (nRBFs) often give better results in higher dimensional problems and have better extrapolation behaviour [Nelles, 2001]. The nRBF output is calculated as

$$\hat{f}(\mathbf{x}) = \frac{\sum_{i=1}^{M} \hat{f}_i(\mathbf{x}) \mathbf{\Phi}_i(\mathbf{x})}{\sum_{i=1}^{M} \mathbf{\Phi}_i(\mathbf{x})} \tag{4}$$

Constant functions $\hat{f}_i = \theta_i$ are common, although in this paper linear models $\hat{f}_i(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\theta}_i$ are found to give a smoother estimate. In either case it can be seen from (3) or (4) that the estimate is linear in the parameters. If the basis functions are fixed in advance and the loss function is the sum of the squared errors over all N training examples

$$J = \sum_{k=1}^{N} e_k^2 = \sum_{k=1}^{N} (y_k - \hat{f}(\mathbf{x}_k))^2$$

then standard linear optimisation techniques can be applied. When each $\hat{f}_i(\mathbf{x})$ is a linear function containing d parameters, the global optimisation is over all Md parameters.

Figure 3 shows the nRBF approximation error for the test function (2) with linear local models. The online updates are made with the well known recursive least squares (RLS) algorithm. The basis functions are centred on an 8×8 grid and the shape matrix $\mathbf{D}_i$ is calculated so that the standard deviation of the basis functions is equal to the grid spacing multiplied by a constant k_σ, with $k_\sigma = 0.5$. The nRBF approximation is seen to reach a high degree of accuracy with very few training examples. However Table 1 shows that the training time per example is huge, reflecting the high complexity of the global optimisation process. Each RLS update takes $O(M^2 d^2)$. Worse still the number of models M increases exponentially with the number of dimensions if the basis functions are fixed on a grid. In addition the spacing and sizes of the basis functions must be defined in advance, or found using an even more expensive nonlinear optimisation process. Clearly alternative techniques must be found for higher dimensional problems.

It is interesting to note that the nRBF representation with Gaussian basis functions lying on a grid and a diagonal shape matrix is equivalent to either a singleton neuro-fuzzy system (when $\hat{f}_i$ is a constant) or a Takagi-Sugeno neuro-fuzzy system (when $\hat{f}_i(\mathbf{x})$ is a linear model) with the product operator as the t-norm [Hunt *et al.*, 1996].

2.3 Locally Weighted Learning

Locally weighted learning [Atkeson *et al.*, 1997] uses the same representation as RBFs, i.e. (3) or (4). However in an attempt to avoid the expensive global estimation, each local model optimises its own loss function over the N training examples observed so far

$$J_i = \sum_{k=1}^{N} \mathbf{\Phi}_i(\mathbf{x}_k) e_{i,k}^2 = \sum_{k=1}^{N} \mathbf{\Phi}_i(\mathbf{x}_k)(y_k - \hat{f}_i(\mathbf{x}_k))^2 \quad (5)$$

This is efficiently achieved using a weighted recursive least squares (wRLS) update with each training example weighted by the value of the basis function $\mathbf{\Phi}_i(\mathbf{x}_k)$. Only d parameters are updated in each of the M models, taking a total time per training example of $O(Md^2)$. This can be reduced further by only updating those models where the weight function is greater than a certain minimum threshold.

Figure 3 shows the nLWL approximation error for the test function (2) with basis functions centred on the same 8×8 grid and a shape matrix calculated using $k_\sigma = 0.3$ (see Section 2.2). The graph shows that normalised LWL (nLWL) gives a lower quality estimate than nRBF. [Murray-Smith, 1994] has shown that LWL reduces the effective number of model parameters, therefore increasing the bias error and hence the asymptotic error in Figure 3. However the nLWL training times in Table 1 are much faster due to the decreased complexity of an update. Unfortunately, along with nRBF, nLWL suffers from an exponential increase in complexity with dimensionality if the basis functions are fixed on a grid, and the spacing and sizes of the basis functions must also be defined in advance, or found using an expensive nonlinear optimisation process.

2.4 Receptive Field Weighted Regression

The natural answer to this "curse of dimensionality" [Bellman, 1961] is to turn to non-parametric learning algorithms where the complexity of the representation can be varied dynamically. More parameters can be allocated to modelling more intricate parts of the function, and less parameters to simpler regions.

Receptive field weighted regression (RFWR) [Schaal and Atkeson, 1998] dynamically constructs the same normalised RBF representation as (4) with linear models as the local functions. New basis functions are added when the system reaches areas of the input space without sufficient coverage. In addition the size and shape of the basis functions (the matrices $\mathbf{D}_i$) are optimised by online second-order gradient descent. The result is a set of basis functions that are longer in directions of less curvature and shorter in directions of high curvature, thereby improving the prediction accuracy. Each local linear model

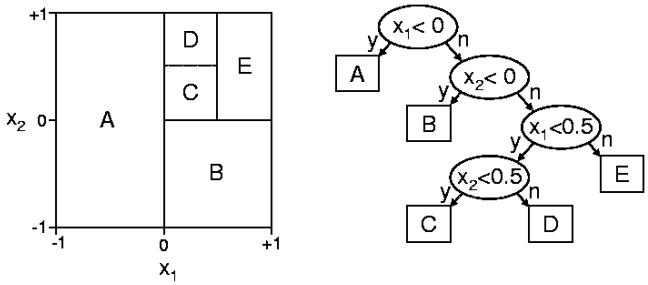


Figure 4: Partitioning of an input space by a decision tree.

is trained using a variation of the locally weighted learning update with exponential forgetting because of the changing shape of the basis functions.

RFWR has since been adapted to improve its dimensionality reduction capability [Vijayakumar and Schaal, 2000], however the test domains do not contain redundant dimensions and the original algorithm is more competitive.

Figure 3 shows the RFWR approximation error for the test function (2) with parameters as published in Schaal and Atkeson's paper and learning rates set to 250. Learning is seen to converge slowly, although the final asymptotic prediction accuracy is good. Unfortunately the algorithm is sensitive to various parameters, including the initial shape matrix assigned to the basis functions $\mathbf{D}_0$ and a penalty term γ. The published values result in a larger number of local models than the alternative techniques reviewed in this paper (Table 1). The advantages of RFWR are that it potentially scales sub-exponentially with the number of dimensions.

2.5 Online LOLIMOT Algorithm

The locally linear model tree (LOLIMOT) algorithm also uses the normalised RBF representation (4) with linear models as local functions [Nelles, 1999]. The basis functions are fitted to a rectangular partitioning of the input space performed by a decision tree with axis-orthogonal splits at the internal nodes (see Figure 4 for an example). For each rectangle, the corresponding basis function has a standard deviation in each dimension that is equal to the width of the rectangle in that dimension multiplied by a constant k_σ. In the experiments $k_\sigma = 0.25$. Locally weighted learning is used to train each local linear model with wRLS updates (see Section 2.3).

The difficult part is deciding when to grow or prune the tree. For each local model the online LOLIMOT algorithm maintains two background sub-models on either side of each potential split down the middle of each dimension. Therefore in each region in Figure 4 LOLIMOT maintains four models, one on each side of either a horizontal or vertical split through the centre of the region.

At each time instant k the most active linear model a is considered for splitting:

$$a = \arg\max_i (\mathbf{\Phi}_i(\mathbf{x}_k))$$

Its local loss function J_a (defined in (5)) is compared with the sum of the local loss functions J_{aj}^- and J_{aj}^+ on each side of the split in dimension j where

$$j = \arg\min_i (J_{ai}^- + J_{ai}^+)$$

is the most promising split. A split is made if $J_a > (J_{aj}^- + J_{aj}^+)k_{\text{improve}}$.

An internal node i is considered for pruning if both children l and r are leaves. The leaves are removed if $J_i < (J_l + J_r)$. The implementation must ensure that the loss functions being compared are all calculated from the same number of examples.

Unfortunately it is very difficult to determine the optimal k_{improve} without extensive trial and error, and the value is sensitive to the level of noise in the data. In the experiments it also proved necessary to stop splitting when the input space had fragmented into a predetermined number of regions.

Figure 3 shows the online LOLIMOT approximation error for the test function (2) with $k_{\text{improve}} = 1.5$ and a maximum of 64 models. Although initial learning is rapid the asymptotic error is higher than RFWR. It is observed that a fast partitioning of the input space occurs, certainly before enough data has been collected to determine a more effective set of regions. Unfortunately increasing k_{improve} leads to an extremely unbalanced partitioning and worse performance. The problem is also exacerbated by increasing noise levels. Figure 5 shows the final approximation errors at different noise levels, keeping the algorithm parameters constant. Online LOLIMOT is seen to deteriorate rapidly with increased noise.

The next section introduces another tree-based technique. However the learning method is firmly rooted in statistics and therefore proves to be inherently more robust and less sensitive to noise levels.

2.6 Incremental Model Tree Induction

In the machine learning community recent work has also concentrated on the online learning of model trees [Potts, 2004b]. The incremental model tree induction (IMTI)[1] algorithm takes a different approach to the system identification methods described in the previous sections. The model tree is grown under the assumption that there is no interaction between the local linear models in the leaves. This assumption enables the likelihood that a split is beneficial to be calculated no matter what level of noise is present. Moreover each training example is only passed down the tree to a single leaf, resulting in fast training times.

[1]Incremental induction in the machine learning literature refers to online learning.

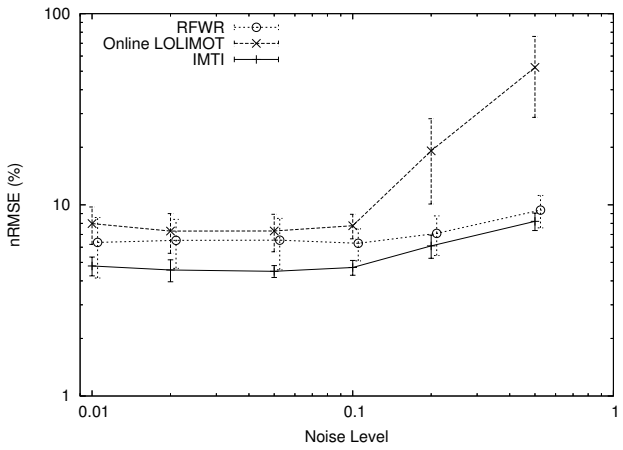


Figure 5: Sensitivity to noise on the 2D test function.

Splitting Rule

In each leaf IMTI maintains a linear model, and κ candidate splits along each dimension (therefore generalising the single candidate split in each dimension maintained by the LOLIMOT algorithm). The assumption of no interaction between the models in the leaves means that the loss function is no longer weighted, and is only calculated over the N_i examples observed in each leaf i

$$J_i = \sum_{k=1}^{N_i} e_{i,k}^2 = \sum_{k=1}^{N_i} (y_k - \hat{f}_i(\mathbf{x}_k))^2$$

The loss functions, and the linear model parameters in each leaf, are calculated with the recursive least squares algorithm. The linear models on each side of every candidate split are also maintained in a similar manner. Denote the loss function on one side of the kth split along the jth dimension as J_{jk}^-, and the loss function on the other side as J_{jk}^+. Assuming Gaussian noise with unknown variance the Chow test for homogeneity amongst sub-samples [Chow, 1960] can determine the probability that the value of the statistic

$$F = \frac{(J_i - J_{jk}^- - J_{jk}^+) \times (N_i - 2d)}{(J_{jk}^- + J_{jk}^+) \times d}$$

occurs under the null hypothesis that the data comes from a single linear model. Under this null hypothesis F is distributed according to Fisher's $\mathcal{F}$ distribution with d and $N_i - 2d$ degrees of freedom, and the probability of obtaining F is the associated p-value (probability in the tail of the distribution). The best split

is found by minimising this probability, and therefore maximising F, over every split k along each dimension j. Clearly F is maximised when $(J_{jk}^- + J_{jk}^+)$ is minimised, and this strategy corresponds to the split selection in the LOLIMOT algorithm. The advantage of this method is that the minimum probability α gives the likelihood that the split is incorrect, and therefore a split is only made when $\alpha < \alpha_{\text{split}}$. A small enough value of α_{split} is suitable for any level of noise. Pruning is performed when the probability α at an internal node rises above the threshold α_{prune} [Potts, 2004b]. In the experiments $\alpha_{\text{split}} = 0.01\%$, $\alpha_{\text{prune}} = 0.1\%$ and $\kappa = 5$.

Stopping Rule

When approximating a smooth surface, IMTI will fragment the input space into smaller and smaller regions to give a more and more accurate approximation. However it is often desirable to limit the growth of the model tree and make do with a certain approximation error. The stopping parameter in [Potts, 2004b] is scaled by the estimated overall variance of the output s_y^2 to ensure a better distribution of models when approximating a multiple-output function with a separate model tree for each output:

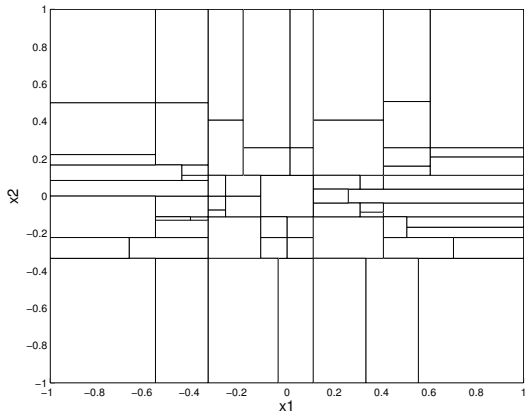
$$\delta = \frac{1}{s_y^2}\left(\frac{\mathrm{J_i}}{N_i - d} - \frac{\mathrm{J_{jk}^- + J_{jk}^+}}{N_i - 2d}\right) \tag{6}$$

As the model tree grows and forms a more accurate approximation, δ decreases. Splitting is halted if δ falls below a certain threshold δ_0 that controls the trade-off between overall model complexity and approximation error.

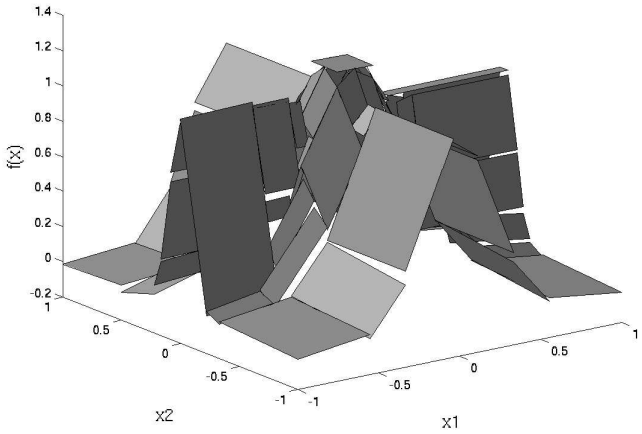
Smoothing

Although the tree is grown under the assumption of no interaction between local models, better predictions are obtained by applying a smoothing technique that results in the same representation (4) as nRBF, nLWL and LOLIMOT. Gaussian basis functions are fitted to the rectangular leaf regions in the same manner as for LOLIMOT, with the standard deviation in each dimension equal to the width of the rectangle in that dimension multiplied by the constant k_σ. In the experiments $k_\sigma = 0.25$. The local functions $\hat{f}_i(\mathbf{x})$ are the linear models in the leaves trained by the unweighted RLS algorithm.

Figure 3 shows the IMTI approximation error on the test function (2) with $\delta_0 = 0.004$. Although initial learning is a little slower, the final approximation is very accurate. Table 1 shows that IMTI uses no more models than the previous techniques, and training times are fast. The final partitioning of the input space and the unsmoothed approximation are shown in Figure 6, illustrating the intelligent manner in which IMTI decomposes the problem with a denser partitioning where the curvature is higher.



a. Input space partitioning.



b. Piecewise linear approximation.

Figure 6: Model tree built by IMTI.

2.7 Complexity Comparison

Table 2 summarises the complexity of each algorithm. The online construction of RFWR and the model tree algorithms ensures that the number of local models M is less than a uniform distribution of c^d local models where c is a constant. The height of the model trees is assumed to be $O(\log M)$.

3 Cart and Pendulum

The problem of swinging up a pendulum on a cart demonstrates the behaviour of the algorithms in a more complex domain (see Figure 7). The system is highly nonlinear when the pendulum is allowed to rotate through 360°. The simplified dynamic model (not taking into account frictional effects) is

Table 2: Algorithm complexity with respect to the number of input dimensions d.

Algorithm	Complexity
Neural Network	$O(Hd)$
nRBF	$O(M^2d^2)$, where $M=c^d$
nLWL	$O(Md^2)$, where $M=c^d$
RFWR	$O(Md^2)$, where $M<c^d$
Online LOLIMOT	$O(M \log Md^3)$, where $M<c^d$
IMTI	$O(\log M\kappa d^3)$, where $M<c^d$

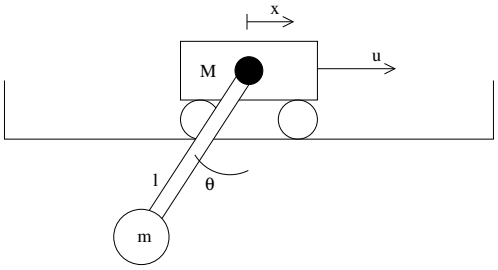


Figure 7: Cart and pendulum.

$$\begin{aligned} 0 &= \ddot{x}\cos\theta - l\ddot{\theta} - g\sin\theta \\ u &= (m+M)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta \end{aligned}$$

where x is the position of the cart (limited to ± 2), θ is the angle of the pendulum, $l = 1$ is the length of the pendulum, g is gravity, $M = m = 1$ are the masses of the cart and pendulum respectively, and u is the lateral force applied to the cart (limited to ± 7). The state at time k is $\mathbf{z}_k = [x_k\ \dot{x}_k\ \theta_k\ \dot{\theta}_k]^T$. The next state of the system is calculated using 5 successive Euler integrations with a time step of 0.01 seconds to give an overall sampling rate of 20 times per second. The system is initialised at rest with the pendulum hanging vertically downward. A simple hand-coded control strategy repeatedly swings up the pendulum and balances it for a short period using the observed state vector $\mathbf{y}_k = \mathbf{z}_{k+1} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is a vector of independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. The system identification task is to learn the dynamics in (1) given examples of $\langle \mathbf{x}_k, \mathbf{y}_k \rangle$. The sequence of states generated is given directly to the learner without changing the order, and is therefore very highly correlated. The algorithms are tested using 10,000 examples randomly drawn from a similar sequence, but without noise.

The RFWR initial distance metric $\mathbf{D}_0 = 10\mathbf{I}$, the penalty $\gamma = 10^{-7}$ and the learning rates are 1000. Online LOLIMOT uses $k_{\text{improve}} = 1.3$ and the number of models is limited to 100. IMTI uses $\delta_0 = 0.001$. Figure 8 shows how the approximation error evolves over time

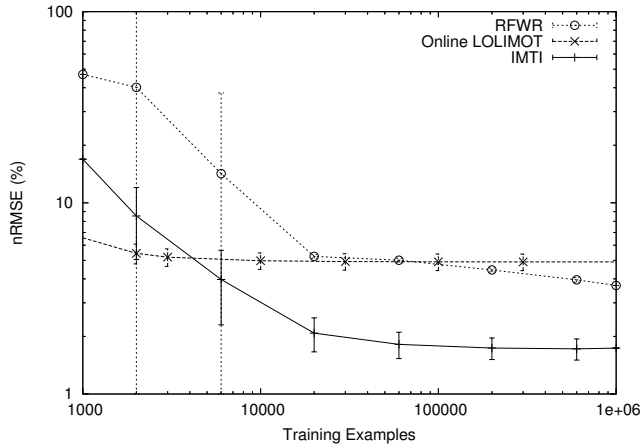


Figure 8: Approximation error on the cart and pendulum.

Table 3: Model sizes and training times per example for the cart and pendulum.

Algorithm	Number of local models	Training time (ms)
RFWR	136 ± 4	37 ± 1
Online LOLIMOT	100 ± 0	25 ± 4
IMTI	98 ± 30	18 ± 0

for each algorithm, demonstrating the superior model constructed by IMTI. Table 3 shows that IMTI uses no more local models than RFWR or online LOLIMOT, although the variation in local model numbers over the 20 trials is high.

4 Flight Simulator

Learning to fly an aeroplane is a complex high-dimensional task. These experiments use a flight simulator (see Figure 9) based on a high-fidelity flight model of a Pilatus PC-9 aerobatic aircraft, an extremely fast and manoeuvrable propeller plane used by the Royal Australian Air Force as a 'lead in fighter' for training pilots before they progress to jet fighters. The model was provided by the Australian Defence Science and Technology Organisation and is based on wind tunnel and in-flight performance data. The same simulator has also been used in previous work [Isaac and Sammut, 2003].

The system is sampled 4 times per second, and 9 state variables are recorded (altitude, roll, pitch, yaw rate, roll rate, pitch rate, climb rate, air speed and a Boolean variable indicating whether the plane is on the ground) along with 4 action variables (ailerons, elevator, throt-

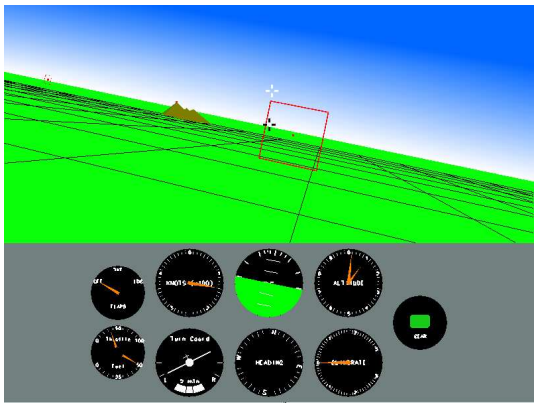


Figure 9: Flight simulator.

tle and the categorical flaps setting which can take the values 'normal', 'take off' and 'landing'). These state variables were selected so that a dynamic model of the plane could be learnt, and therefore the absolute position and heading were disregarded. The combination of states and actions results in a 13 dimensional regressor vector $\mathbf{x}$. The learning task is to predict the 8 continuous values of the next state.

As for the pendulum on a cart the training examples are taken directly from a trace of the aircraft flying so that successive regressors are highly correlated. The trace involves repeatedly taking off, flying a loop and landing back on the same runway. Simulated turbulence is set to a high level resulting in complex noise characteristics that deviate substantially from the independent Gaussian assumption. Additional noise is also added to the inputs to excite the system and provide a richer source of training examples. The algorithms are tested using 10,000 examples randomly drawn from a similar trace.

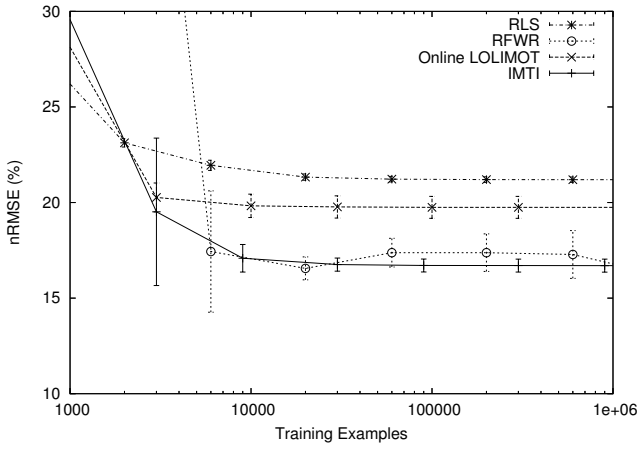


Figure 10: Approximation error on the flight simulator.

Table 4: Model sizes and training times per example for the flight simulator.

Algorithm	Number of local models	Training time (ms)
RFWR	26 ± 5	18 ± 2
Online LOLIMOT	30 ± 0	74 ± 9
IMTI	26 ± 5	108 ± 1

The RFWR initial distance metric $\mathbf{D}_0 = 2.5\mathbf{I}$, the penalty $\gamma = 10^{-5}$ and the learning rates are 1000. Online LOLIMOT uses $k_{\text{improve}} = 2.0$ and the number of models is limited to 30. IMTI uses $\delta_0 = 0.02$. Figure 10 shows how the approximation error evolves over time for each algorithm, and also for a single linear model updated using recursive least squares (RLS). Clearly RFWR and IMTI form good models of the aircraft dynamics. Table 4 shows that each algorithm uses approximately the same number of local models, and that IMTI and online LOLIMOT are computationally expensive on this higher dimensional problem.

5 Discussion

For each algorithm various parameters are tuned in each domain to result in effective learning. RFWR has three such parameters; the initial distance metric $\mathbf{D}_0$, the penalty γ and the learning rates. In fact there are two learning rates, one for standard gradient descent and one for meta-learning, but these are taken to be the same. The initial distance metric affects how many linear models are allocated initially, the penalty affects the final size and number of linear models (it plays a similar role to the IMTI stopping parameter δ_0), and the learning rates affect how quickly the size and shape of the linear models change. In practice it proved hard to balance the effects of these parameters and obtain good learning performance. If the learning rates are too high the algorithm becomes unstable, and if too low no perceptible change in the size and shape of the linear models is observed.

Online LOLIMOT has two parameters; the splitting parameter k_{improve} and the maximum number of models. In the experiments the maximum number of models was set to enable a comparison with the other algorithms, although in other applications this limit would be set by trial and error. The algorithm was sensitive to the value of k_{improve}, and poor learning results if it is too large or small.

In contrast IMTI requires only the single stopping parameter δ_0 to be adjusted in each domain. This parameter directly controls the trade-off between the final model

size and prediction accuracy, and in practice δ_0 proved easy to tune. In addition δ_0 ensures a fair distribution of model complexity between each tree in a multiple-output problem. An advantage of the tree representation in an online setting is that δ_0 can be reduced on-line if the approximation is not sufficiently accurate. The tree will grow from its leaves to form a more detailed model, and no re-building or internal restructuring is required.

6 Conclusions and Future Work

This paper compares and contrasts several algorithms that can construct nonlinear models online. It is demonstrated that the classical methods of neural networks, radial basis functions and locally weighted learning suffer from either slow convergence or the curse of dimensionality. In addition the structure of the neural network, or the distribution and sizes of the basis functions, proves difficult to specify without significant trial and error or some form of expensive nonlinear optimisation.

Three recent algorithms are also examined that show potential to scale to a large number of dimensions, and their representations are seen to fit into the basis function paradigm. Receptive field weighted regression dynamically adjusts its basis functions to spread over areas with low curvature. The other two algorithms, incremental model tree induction (IMTI) and the online local linear model tree (LOLIMOT) algorithm use a decision tree to partition the input space. IMTI is seen to have consistently superior learning characteristics, although it starts to become computationally expensive in higher dimensions. A better scaling version of the technique has been developed but with slower convergence [Potts, 2004a].

The tree-based algorithms have fewer parameters and no learning rates to tune, thus avoiding a major cause of instability in many gradient descent systems like RFWR. In addition they require no metric over the input space. IMTI has a single tunable parameter that intuitively controls the trade-off between the number of linear models built and the approximation error, and is shown to be significantly less sensitive to high observation noise than online LOLIMOT.

Having selected IMTI as the most suitable method for the online identification of nonlinear dynamic models, future work will concentrate on control. [Nakanishi *et al.*, 2002] have developed a provably stable adaptive controller based on the representation learnt by RFWR, and given the strong parallels observed in this paper, perhaps a similar approach can be applied to the IMTI model tree representation.

References

[Atkeson *et al.*, 1997] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[Bellman, 1961] R. Bellman. *Adaptive control processes*. Princeton University Press, 1961.

[Chow, 1960] G. Chow. Tests of equality between sets of coefficients in two linear regressions. *Econometrica*, 28(3):591–605, 1960.

[Hunt *et al.*, 1996] K. Hunt, R. Haas, and R. Murray-Smith. Extending the functional equivalence of radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks*, 7(3):776–781, 1996.

[Isaac and Sammut, 2003] A. Isaac and C. Sammut. Goal-directed learning to fly. In *Proceedings of the 20th International Conference of Machine Learning*, pages 258–265, 2003.

[Ljung, 1987] L. Ljung. *System identification: Theory for the user*. Prentice-Hall, 1987.

[Murray-Smith, 1994] R. Murray-Smith. *A local model network approach to nonlinear modelling*. PhD thesis, University of Strathclyde, Strathclyde, UK, 1994.

[Nakanishi *et al.*, 2002] J. Nakanishi, J. Farrell, and S. Schaal. A locally weighted learning composite adaptive controller with structure adaptation. In *IEEE International Conference on Intelligent Robots and Systems*, 2002.

[Nelles, 1999] O. Nelles. *Nonlinear system identification with local linear neuro-fuzzy models*. PhD thesis, TU Darmstadt, 1999.

[Nelles, 2001] O. Nelles. *Nonlinear system identification*. Springer, 2001.

[Potts, 2004a] D. Potts. Fast incremental learning of linear model trees. In *Proceedings of the 8th Pacific Rim International Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence, 3157*, pages 221–230. Springer, 2004.

[Potts, 2004b] D. Potts. Incremental learning of linear model trees. In *Proceedings of the 21st International Conference on Machine Learning*, pages 663–670, 2004.

[Schaal and Atkeson, 1998] S. Schaal and C. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10:2047–2084, 1998.

[Slotine and Li, 1991] J. Slotine and W. Li. *Applied nonlinear control*. Prentice-Hall, 1991.

[Vijayakumar and Schaal, 2000] S. Vijayakumar and S. Schaal. Locally weighted projection regression: Incremental real time learning in high dimensional space. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1079–1086, 2000.