Incremental Learning of Linear Model Trees

Duncan Potts DUNCANP@CSE.UNSW.EDU.AU
School of Computer Science and Engineering, University of New South Wales, Australia

Abstract

A linear model tree is a decision tree with a linear functional model in each leaf. Previous model tree induction algorithms have operated on the entire training set, however there are many situations when an incremental learner is advantageous. In this paper we demonstrate that model trees can be induced incrementally using an algorithm that scales linearly with the number of examples. An incremental node splitting rule is presented, together with incremental methods for stopping the growth of the tree and pruning. Empirical testing in three domains, where the emphasis is on learning a dynamic model of the environment, shows that the algorithm can learn a more accurate approximation from fewer examples than other incremental methods. In addition the induced models are smaller, and the learner requires less prior knowledge about the domain.

1. Introduction

There are many situations when incremental learning is more suitable than a batch processing technique. If the input is a continuous stream of data it may not be tractable to record all of its history and execute a batch algorithm each time an output is required. For example an agent operating in a real-time environment may need to constantly process the latest sensor information to determine the next action. A large processing delay may be unacceptable, and some form of incremental learning algorithm is required that scales linearly with the incoming data.

In this paper we shall focus on the problem of inducing a model of the environment to be used for control, however the methods developed have a potentially much wider applicability. In order to control an agent it is necessary to understand how the world evolves, and how the agent's actions affect this evolution. This knowledge is often obtained by constructing a model of the environment. A non-linear stationary model of a continuous dynamic environment can be formulated in continuous time as

$$\dot{\mathbf{z}} = f(\mathbf{z}, \mathbf{u}) \tag{1}$$

where $\mathbf{z}$ is an n dimensional state vector and $\mathbf{u}$ is an m dimensional input ($\dot{\mathbf{z}}$ is the rate of change of $\mathbf{z}$ with respect to time) (Slotine & Li, 1991). The problem then becomes one of incrementally learning an approximation to the function f using the states experienced by the agent.

Standard parametric methods for learning f incrementally include neural networks, locally weighted learning (LWL) (Atkeson et al., 1997) and radial basis functions (RBFs). For a neural network it is difficult to determine in advance the numbers of layers and units, and training can be slow and prone to local minima. For both LWL and RBFs the range of input values and approximate curvature of the function is required to determine the number, size and shape of the weight or basis functions. In addition the number of parameters scales exponentially with dimensionality.

When there is little prior knowledge it may be beneficial to consider non-parametric alternatives where the number of learning parameters is adjusted by the algorithm. Schaal and Atkeson (1998) developed a LWL-based algorithm that not only dynamically allocates models as required, but also adjusts the shape of each local weighting function. Enhanced dimensionality reduction has also been incorporated (Vijayakumar & Schaal, 2000). These algorithms perform well for data with a low intrinsic dimensionality, even if the input data itself has a high number of dimensions. However there are several parameters that are hard to specify without trial and error, and the range of inputs and a metric over the input space must be defined in advance.

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

A decision tree with a linear model in each leaf (a linear model tree) can also approximate a non-linear function. The induction of such trees in a batch manner has received significant attention in the literature, however the incremental induction of model trees has not been previously addressed.

The contribution of this paper is to develop incremental methods for growing linear model trees, including a node splitting rule, a stopping rule and a method for pruning. The algorithm is empirically tested in three domains and compared with existing incremental, instance-based and batch methods.

Section 2 gives a brief overview of linear model trees and Section 3 surveys related work on both the incremental induction of classification trees and the batch induction of linear model trees. Section 4 describes the algorithm, Section 5 presents the experimental results, and finally we conclude and suggest areas for future work.

2. Linear Model Trees

Learning each component of $\dot{\mathbf{z}}$ in (1) can be formulated as a typical regression problem

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

where $f(\mathbf{x})$ is the component of $\dot{\mathbf{z}}$ and $\mathbf{x} = [\mathbf{z}^T\ \mathbf{u}^T\ 1]^T$ 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 independent zero-mean Gaussian noise ϵ with unknown variance σ^2. The aim of a regression analysis is to find an approximation $\hat{f}(\mathbf{x})$ to $f(\mathbf{x})$ that minimises some cost function (e.g. sum of squared errors) over a set of training examples.

For a linear model tree $\hat{f}(\mathbf{x})$ is a decision tree with a linear model in each leaf. The decision tree partitions the input space and within each leaf

$$\hat{f}(\mathbf{x}) = \mathbf{x}^T\hat{\theta} \tag{2}$$

where $\hat{\theta}$ is a column vector of d parameters. The linear least squares estimate of the function $f(\mathbf{x})$ is the value of $\hat{\theta}$ that minimises

$$J = \sum_{i=1}^{N}(y_i - \mathbf{x}_i^T\hat{\theta})^2$$

over the N training examples $\langle \mathbf{x}_i, y_i \rangle$ in the leaf. Defining the $N \times 1$ vector $\mathbf{y}$ and the $N \times d$ matrix $\mathbf{X}$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$$

the linear least squares estimate is

$$\hat{\theta}_{LS} = [\mathbf{X}^T\mathbf{X}]^{-1}\mathbf{X}^T\mathbf{y} \tag{3}$$

The residual for each example is the difference between the value y_i and the prediction $\hat{f}(\mathbf{x}_i)$, and the residual sum of squares (RSS) is the minimum value of J (occurring when $\hat{\theta} = \hat{\theta}_{LS}$). Both $\hat{\theta}_{LS}$ and RSS can be calculated incrementally using the recursive least squares (RLS) algorithm. The difficulty lies in defining the tree structure itself.

3. Related Work

3.1. Incremental Tree Induction

Utgoff et al. (1997) consider the incremental induction of classification trees where each leaf determines an element from a finite set of classifications. One of their aims is that the incremental algorithm generates an identical tree to one built in a batch manner, and therefore that the resultant tree is invariant to the order in which the examples are processed. To achieve this all examples must be stored, and although the average incremental update is fast, in some cases it can take longer than re-building the entire tree. Our goal, on the other hand, is a fully incremental algorithm that is guaranteed to be fast. In addition their learning technique does not extend to model trees because the best split can no longer be determined from a small number of statistics kept in the node.

3.2. Batch Induction of Model Trees

The general approach to building trees from a training set is to start at the root and perform top down induction. At each node the training set is recursively partitioned using a splitting rule until the tree is sufficiently large. A number of alternative rules have been proposed for the induction of linear model trees. Denote the N examples at a particular node as $\langle \mathbf{x}_i, y_i \rangle$. A potential split divides these examples into two subsets. Denote the N_1 examples in the first subset $\langle \mathbf{x}_{i1}, y_{i1} \rangle$, and the N_2 in the second subset $\langle \mathbf{x}_{i2}, y_{i2} \rangle$.

M5 (Quinlan, 1993; Frank et al., 1998) chooses the split that minimises a measure of the standard deviation of the y values, and HTL (Torgo, 1997) minimises the mean square error of the y values, measuring error from the average y value. These measures are closely related to those originally used by Breiman et al. (1984) to grow regression trees with a constant value in each leaf. However when linear models are fitted to each leaf, distance to the mean y value is an inappropriate measurement of error and distance to the linear regression plane should be used instead. RETIS (Karalic, 1992) minimises the total RSS over the two subsets

$$\sum_{i=1}^{N_1}\Big(y_{i1}-\hat{f}_1(\mathbf{x}_{i1})\Big)^2+\sum_{i=1}^{N_2}\Big(y_{i2}-\hat{f}_2(\mathbf{x}_{i2})\Big)^2 \qquad (4)$$

where $\hat{f}_k(\mathbf{x})$ is the linear least squares estimate for subset k (see Figure 1).

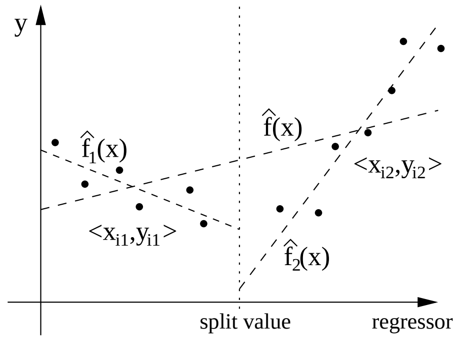

Figure 1. Split selection with linear models in the leaves.

The number of potential split values increases with the number of examples (assuming the regressors are drawn from a continuous set), and $\hat{\theta}_{LS}$ must be calculated using (3) for the two subsets of examples on each side of every split to minimise (4). It therefore quickly becomes intractable to test all potential split values with this method.

Alexander and Grimshaw (1996) reduce the complexity by only considering simple linear models (with a single regressor) in each leaf, but this limits the representation to surfaces with axis-orthogonal slopes. Malerba et al. (2001) build up a multivariate linear model using a sequence of simple linear regressions which also simplifies the split selection, however splits near the root are therefore only selected on the basis of a few regression components and comprehensibility is lost due to the transformation of regressors. Dobra and Gehrke (2002) determine the split by separating two Gaussian clusters in the regressor attributes enabling oblique splits to be found. Li et al. (2000) also find oblique splits using principal Hessian directions, however the procedure is complex and requires interaction from the user and an iterative process to find the split value.

These methods are effective in a batch setting where typically an overly large tree is grown initially, and a pruning process is later applied to try and optimise the prediction capability on unseen examples. For an incremental algorithm, however, it is desirable to limit the growth of the tree in the first place and avoid any complex pruning procedure. The algorithm must therefore not only determine *where* to make a split but also *when*, and the splitting rules considered so far do not help. There is clearly no need to make any split if the examples in a node can all be explained by a single linear model, however the node should be split if the examples suggest that two separate linear models would give significantly better predictions.

Fortunately this problem has also received attention in the statistics community. Both the SUPPORT (Chaudhuri et al., 1994) and GUIDE (Loh, 2002) compute the residuals from a linear model and compare the distributions of the regressor values from the two sub-samples associated with the positive and negative residuals. Using statistical tests it is possible to determine the likelihood of the examples occurring under the hypothesis that they were generated from a single linear model, and therefore a confidence level can be assigned to the split. An incremental algorithm has been developed with the statistical tests used by SUPPORT, however predictions are more accurate using an alternative statistical test based on the residual sums of squares. The details of this test are explained in the next section.

4. Incremental Induction Algorithm

Section 4.1 presents the splitting rule used in the incremental model tree induction (IMTI) algorithm, and Sections 4.2 and 4.3 describe incremental methods for stopping the growth of the tree and pruning. Section 4.4 explains how smoothing is used to obtain more accurate gradient estimates and Section 4.5 analyses the complexity of IMTI.

4.1. Splitting Rule

The question of whether the two linear models on each side of a potential split give a better estimation of the underlying function $f(\mathbf{x})$ than a single linear model can be tested as a hypothesis. The null hypothesis is that the underlying function is linear over the entire node ($H_0 : f(\mathbf{x}) = \mathbf{x}^T\theta$) while the alternative hypothesis is that it is not. Three linear models are fitted to the examples in the node as in Figure 1; $\hat{f}(\mathbf{x})$ using all N examples, $\hat{f}_1(\mathbf{x})$ using the N_1 examples lying on one side of the split, and $\hat{f}_2(\mathbf{x})$ using the N_2 examples on the other side. The residual sums of squares are also calculated for each linear model, and denoted $\mathrm{RSS}_0, \mathrm{RSS}_1$ and RSS_2 respectively. The two smaller linear models will always fit the data at least as well, and $\mathrm{RSS}_1 + \mathrm{RSS}_2 \leq \mathrm{RSS}_0$. However if the alternative hypothesis is true $\mathrm{RSS}_1 + \mathrm{RSS}_2$ will be significantly less than RSS_0, and this can be tested using the Chow test, a standard statistical test for homogeneity amongst sub-samples (Chow, 1960). Under the null hypothesis it can be shown that the statistic

$$F = \frac{(\mathrm{RSS}_0 - \mathrm{RSS}_1 - \mathrm{RSS}_2) \times (N - 2d)}{(\mathrm{RSS}_1 + \mathrm{RSS}_2) \times d} \tag{5}$$

is distributed according to Fisher's $\mathcal{F}$ distribution with d and $N-2d$ degrees of freedom (d is the dimensionality as defined in Section 2). The candidate split least likely to occur under H_0 should make the best choice, and this corresponds to the F statistic with the smallest associated p-value (probability in the tail of the distribution). This method was used by Sicilano and Mola (1994) to grow linear model trees in a batch setting. Denote the smallest p-value as α. The key advantage of such a statistical test in an incremental implementation is that if the probability α is not small enough to discount H_0 with the desired degree of confidence, no split should be made until further evidence is accumulated.

The intractability of maintaining linear models on each side of every split value has already been noted, therefore a constant number κ of candidate split values is defined in advance for each regressor. In each leaf a linear model is maintained on both sides of the $\kappa(d-1)$ candidate split values, and α is calculated from these models. Table 1 defines the steps needed to decide whether to split a leaf node given a training example. In our implementation $\alpha_{split} = 0.01\%$ and $\kappa = 10$. Therefore a split is only made when there is less than a 0.01% chance that the data in the node came from a single linear model. Such a low level reduces the number of incorrect splits when testing many times. Reducing κ appears to have little effect on the results with the advantage of less computation.

Table 1. Leaf training function.

function Train(leaf t, example $\langle \mathbf{x}, y \rangle$)
  update leaf model m with $\langle \mathbf{x}, y \rangle$ (using RLS)
  for each regressor i
    for each potential split j
      if $x_i <$ split value s_{ij}
        update leaf model m_{ij1} with $\langle \mathbf{x}, y \rangle$
      else
        update leaf model m_{ij2} with $\langle \mathbf{x}, y \rangle$
      end if
      calculate F using (5) and the p-value
    end for
  end for
  calculate δ using (6) for split with min p-value
  if min p-value $< \alpha_{split}$ and $\delta > \delta_0$
    split the leaf
  end if
end Train

It is, however, often desirable to limit the growth of the tree. The parameter

$$\delta = \frac{\mathrm{RSS}_0}{N - d} - \frac{\mathrm{RSS}_1 + \mathrm{RSS}_2}{N_1 + N_2 - 2d} \tag{6}$$

is the difference between the variance estimate using a single linear model and the pooled variance estimate using separate linear models on each side of a candidate split. 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 (see Table 1).

An advantage of the tree representation in an incremental setting is that δ_0 can be lowered 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.

4.2. Stopping Rule

To illustrate the need for a stopping rule, assume a linear model tree is being grown to approximate a smooth convex function. As more examples are observed, the statistical splitting rule will repeatedly split the leaf nodes of the tree. The tree will grow without bound, fracturing the input space into a large number of linear models that together result in a very good approximation. Setting the confidence limit on the statistical test higher will only slow this process.

4.3. Pruning

A fully incremental algorithm that does not store any training examples cannot perform cross-validation, and therefore many standard pruning techniques are not possible. However if a bad split has been made, it is desirable to identify and prune branches of the tree that are not contributing to the overall accuracy.

In Section 4.1 the p-value associated with the F statistic was seen to give the probability that the examples in the node could have come from a single linear model. Hence a high p-value at an internal node indicates that it may be beneficial to prune the node's children. The F statistic (5) is therefore maintained in each internal node, and pruning takes place if the corresponding p-value rises above a certain threshold α_{prune}. In the experiments $\alpha_{prune} = 10\%$.

4.4. Smoothing

Although we have focussed on learning an approximation to the function f, what we are really interested in for control purposes is the derivative of f with respect to the state $\mathbf{z}$ and input $\mathbf{u}$. A popular method of non-linear control is to linearise the system about a certain operating point ($\mathbf{z}=\mathbf{z}_0, \mathbf{u}=\mathbf{u}_0$) using the Taylor series expansion

$$\dot{\mathbf{z}} = f(\mathbf{z}_0, \mathbf{u}_0) + \frac{\partial f}{\partial \mathbf{z}}(\mathbf{z}-\mathbf{z}_0) + \frac{\partial f}{\partial \mathbf{u}}(\mathbf{u}-\mathbf{u}_0) + \ldots \text{(h.o.t)}$$

where the derivatives are evaluated at the operating point. Ignoring the higher order terms (h.o.t.) we obtain the standard equations for a linear system

$$\dot{\mathbf{z}} = \mathbf{F}\mathbf{z} + \mathbf{G}\mathbf{u} + \mathbf{c} \tag{7}$$

which have been intensively studied in control theory. From these equations we can determine the local stability of the system, and control policies to move the state along all possible local trajectories.

However the linear model tree lacks continuity, resulting in poor gradient estimates. Therefore the estimates are smoothed using a Gaussian kernel. From the linear model (2) it is clear that the gradient estimate in each leaf k is simply the parameter vector $\hat{\theta}_k$, hence the smoothed gradient estimate over all L leaves is

$$\frac{\partial \hat{f}(\mathbf{x})}{\partial \mathbf{x}} = \nabla \hat{f}(\mathbf{x}) = \frac{1}{W}\sum_{k=0}^{L} w_k(\mathbf{x})\hat{\theta}_k \ , \ W = \sum_{k=0}^{L} w_k(\mathbf{x})$$

where $w_k(\mathbf{x})$ is the Gaussian kernel

$$w_k(\mathbf{x}) = \exp\left(-\frac{\rho}{2}(\mathbf{x}-\mathbf{c}_k)^T\mathbf{D}_k(\mathbf{x}-\mathbf{c}_k)\right)$$

The centre of the leaf in regressor space is $\mathbf{c}_k$, and the elements of the diagonal matrix $\mathbf{D}_k$ are the inverse squares of each leaf width component. The same smoothing process is also applied to the actual function estimates $\hat{f}(\mathbf{x})$ although with less improvement. The value $\rho = 16$ is used for the smoothing parameter.

4.5. Training Complexity

Assuming that the stopping rule has limited the growth of the tree, the time taken to pass each training example down to a single leaf in the tree according to the tests in the intermediate nodes is bounded by the depth of the tree. In the leaf $O(\kappa d)$ models are updated, and each RLS update takes $O(d^2)$. Hence the overall training complexity is $O(N\kappa d^3)$ where N is the total number of examples. The algorithm therefore fulfills our goal of scaling linearly with the number of examples. Also pleasing is the polynomial increase with dimensionality, and the fact that a strict bound can be placed on the worst case processing time for a training example (if the tree has stopped growing).

5. Experimental Results

The new incremental model tree induction (IMTI) algorithm is empirically tested in three domains, and results are compared with four alternative algorithms:

1. A single linear model updated using RLS.
2. The k-nearest neighbour instance-based approximator, with $k=10$ (10-NN). This algorithm stores all training examples, and returns the average of the 10 nearest neighbours when making predictions. Comparisons can therefore be made with a representation that is not piecewise linear.
3. The incremental adaptive locally weighted learning algorithm RFWR (Schaal & Atkeson, 1998).
4. The batch model tree induction algorithm SUPPORT (Chaudhuri et al., 1994) using linear models and the smoothing technique identified in Section 4.4.

The batch algorithm from which the IMTI splitting rule was derived (Sicilano & Mola, 1994) does not scale to the numbers of examples considered here, and therefore we compare with SUPPORT which performs better than both CART (Breiman et al., 1984) and M5′ (Frank et al., 1998) on these problems.

RFWR has since been adapted to improve its dimensionality reduction capability (Vijayakumar & Schaal, 2000), however the test domains do not contain redundant dimensions and the original algorithm, without any ridge regression parameters, is more competitive.

IMTI, RFWR and SUPPORT all construct a piecewise linear approximation. Parameters in each algorithm limit the number of linear models, resulting in a certain degree of asymptotic approximation error. These parameters are optimised in each domain to give the best performance while keeping the number of models to within a comparable range.

The RFWR initial distance metric $\mathbf{D}_0$ is optimised over the set $\{2.5\mathbf{I}, 5\mathbf{I}, 10\mathbf{I}, 25\mathbf{I}\}$ ($\mathbf{I}$ is the identity matrix), the penalty γ is optimised over the set $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}\}$ and the learning rates are optimised over the set $\{2.5, 5, 10, 25, 50, 100, 250\}$. The SUPPORT parameters are MINDAT=30, v=10 and both f and η are optimised over the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$.

Both 10-NN and RFWR require a metric to be defined over the input space. For these algorithms each regressor is scaled to the range $(-1, +1)$ and the metric is the standard Euclidean distance. The observed y values are not scaled.

Training is performed using a single stream of non-repeating examples, such as would be obtained by an agent in the real world. The non-incremental SUPPORT algorithm is re-run for each point on the following graphs.

In all domains the measurement noise ϵ is (a vector of) independent zero-mean Gaussian noise with variance σ^2 and $\sigma = 0.1$. Error bars and numerical errors reported in tables indicate one unbiased estimate of the standard deviation over 20 trials.

5.1. 2D Test Function

Initial tests are performed using the same function as Schaal and Atkeson (1998)

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

Training examples are drawn uniformly from the square $-1 \le x_1 \le +1, -1 \le x_2 \le +1$. The algorithms are tested using 2000 examples drawn in a similar manner, but without noise.

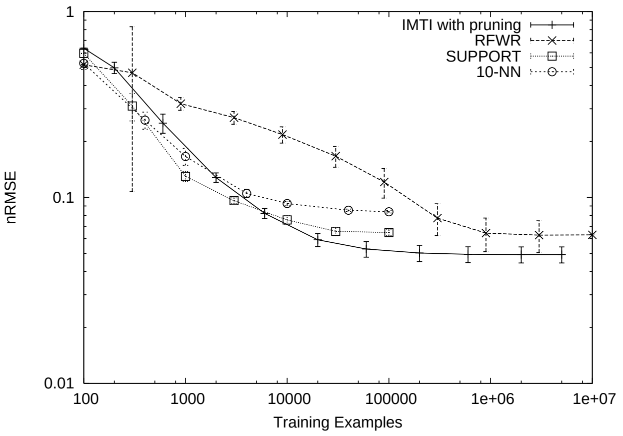

Figure 2. Prediction errors for the 2D test function.

Figure 2 plots the normalised root mean square error (nRMSE) against the number of examples. IMTI uses the stopping parameter $\delta_0 = 10^{-3}$. RFWR uses the parameters specified in Schaal and Atkeson (1998) and learning rates set to 250. SUPPORT uses $f = 0.1$ and $\eta = 0.4$. The single linear model can only achieve nRMSE = 1.0. The graph shows that IMTI accurately learns the function with fewer training examples than RFWR, and surprisingly its performance is even comparable to the batch SUPPORT algorithm.

Table 2. Number of models built for the 2D test function.

Algorithm	# Models	# Parameters
IMTI with pruning	68±6	3
RFWR	92±3	3
SUPPORT	61±3	3

Table 2 reports the final number of local models built by each algorithm and the number of parameters within each local model. IMTI also uses less models than RFWR.

5.2. Pendulum

A pendulum rotating 360° around a pivot is a simple non-linear dynamic environment with a closed form for the gradient, allowing gradient errors to be examined. The dynamic model of a pendulum can be written

$$\dot{\mathbf{z}} = \begin{bmatrix} 0 & 1 \\ -\frac{g\sin\theta}{l\theta} & -\frac{\mu}{ml^2} \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \qquad (8)$$

where $\mathbf{z} = [\theta\ \dot{\theta}]^T$, θ is the pendulum angle, g is gravity, $m = l = 1$ are the mass and length of the pendulum, $\mu = 0.1$ is a drag coefficient and u is the torque applied to the pendulum. Define $\mathbf{x} = [\mathbf{z}^T\ u]^T$ and $\mathbf{y} = \dot{\mathbf{z}} + \epsilon$. Training examples of $\langle \mathbf{y}, \mathbf{x} \rangle$ are drawn uniformly from the input domain $-\pi \le \theta \le +\pi$, $-5 \le \dot{\theta} \le +5$ and $-5 \le u \le +5$. The algorithms are tested using 5000 examples drawn in a similar manner, but without noise.

In order to examine gradient predictions, we define the normalised root mean square *gradient* error as

$$\text{nRMSE(Grad)} = \sqrt{\frac{\sum_{i=0}^{N} \left|\nabla\left(f(\mathbf{x}_i) - \hat{f}(\mathbf{x}_i)\right)\right|^2}{\sum_{i=0}^{N} \left|\nabla f(\mathbf{x}_i)\right|^2}}$$

When the gradient estimate is correct everywhere nRMSE(Grad) = 0, and when the gradient estimate is zero everywhere nRMSE(Grad) = 1.

Figure 3 plots the mean (over both $\dot{\mathbf{z}}$ components) nRMSE(Grad) against the number of examples. IMTI uses $\delta_0 = 10^{-3}$. RFWR uses $\mathbf{D}_0 = 5\mathbf{I}$, $\gamma = 10^{-7}$ and learning rates set to 5. Gradients are extracted from RFWR by applying the same weighting kernels to the gradients of each local model that are applied to the

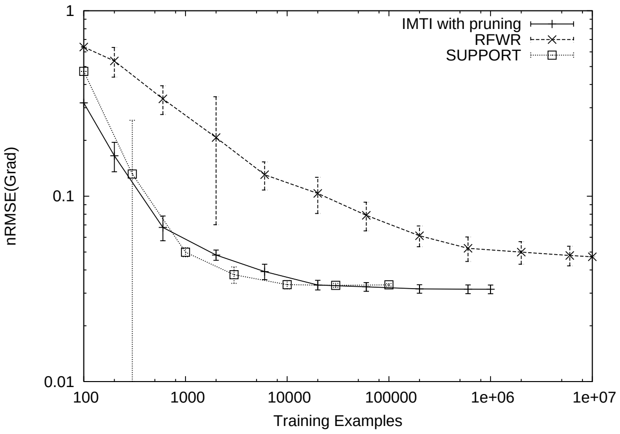

Figure 3. Gradient errors for the pendulum.

predictions themselves. SUPPORT uses $f = 0.1$ and $\eta = 0.5$. It is not possible to obtain gradients from 10-NN. The single linear model achieves nRMSE(Grad) = 0.76. The graph shows that IMTI can learn accurate gradients, and can therefore effectively estimate the linear model given by $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{c}$ in (7) for the pendulum (8).

Table 3. Number of models built for the pendulum.

Algorithm	# Models	# Parameters
IMTI with pruning	30±1	4
RFWR	22±5	8
SUPPORT	25±1	4

Table 3 reports the final number of local models and the number of parameters within each local model. A separate model tree is induced for each component of $\mathbf{y}$ and therefore the local IMTI and SUPPORT models are half the size of the local RFWR models. IMTI uses less overall parameters and forms a more accurate gradient estimate than RFWR.

5.3. Pendulum on a Cart

The problem of swinging up a pendulum on a cart demonstrates the behaviour of the algorithm in a more complex domain. The aim is to balance the pendulum in the vertical position by applying a horizontal force to the cart. The physical parameters of the pendulum are the same as in the previous section (except $\mu = 0$), and the cart has a mass of 1. The system is highly non-linear when the pendulum is allowed to rotate through 360°. So far we have only considered continuous time models, however the analysis applies equally to models formulated in discrete time. For the cart and pendulum the discrete time model is

$$\mathbf{z}_{k+1} = \mathbf{\Phi}\mathbf{z}_k + \mathbf{\Gamma}u_k$$

where $\mathbf{z} = [x\ v\ \theta\ \omega]^T$, x (limited to ± 2) and v are the position and velocity of the cart, θ and ω are the angle and angular velocity of the pendulum, and u (limited to ± 7) is the force applied to the cart. The system is sampled 20 times per second. It is not possible to determine a closed form for $\mathbf{\Phi}$ or $\mathbf{\Gamma}$ and therefore the next state of the system is calculated using successive Euler integrations with a time step of 0.01 seconds.

Instead of sampling randomly across the state space, the system is initialised at rest with the pendulum hanging vertically downward. A simple control strategy was hand-coded to repeatedly swing up the pendulum and balance it for a short period using the observed state vector $\mathbf{y}_k = \mathbf{z}_k + \epsilon$. The regressors for $\mathbf{y}_k$ are the previous state $\mathbf{z}_{k-1}$ and action u_{k-1}. The sequence of states generated is given directly to the learner without changing the order, so that consecutive regressors are very highly correlated. The algorithms are tested using 10,000 examples randomly drawn from a similar sequence, but without noise.

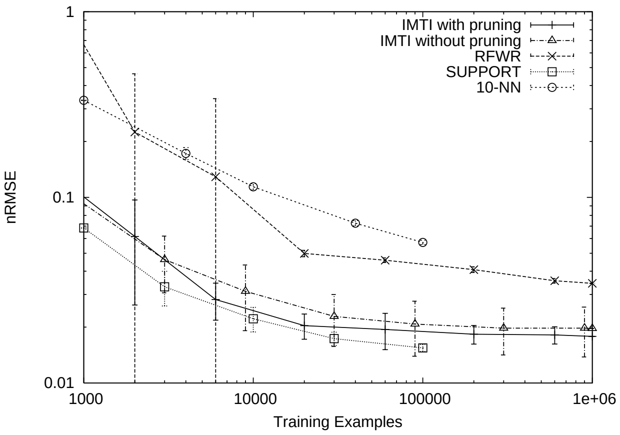

Figure 4. Prediction errors for the cart and pendulum.

Figure 4 plots the mean (over all $\mathbf{z}$ components) nRMSE against the number of examples. IMTI uses $\delta_0 = 5 \times 10^{-3}$, and results are shown both with and without pruning. RFWR uses $\mathbf{D}_0 = 10\mathbf{I}$, $\gamma = 10^{-5}$ and learning rates set to 50. SUPPORT uses $f = 0.2$ and $\eta = 0.5$. The single linear model achieves nRMSE = 0.13. Again IMTI learns a more accurate model from fewer examples than RFWR. Pruning slightly improves the prediction accuracy and results in less variation over the different trials.

Table 4 reports the final number of local models and the number of parameters within each model. It shows that pruning significantly reduces the number of linear models but the variation remains high. In this domain

Table 4. Number of models for the cart and pendulum.

ALGORITHM	# MODELS	# PARAMS
IMTI with pruning	111±42	6
IMTI without pruning	187±89	6
RFWR	129±6	24
SUPPORT	116±6	6

RFWR forms its approximation with over four times as many parameters as the model tree algorithms.

6. Conclusions and Future Work

This paper describes and evaluates an algorithm that incrementally induces linear model trees. The algorithm can learn a more accurate approximation of an unknown function from fewer examples than other incremental methods, and in addition the induced model can contain less parameters. The learner is even comparable to a batch algorithm that induces the same type of tree.

In the algorithm the trade-off between the number of linear models built and the approximation error can easily be controlled using the single parameter δ_0, and there are no learning rates to be tuned, thus avoiding a major cause of instability in many gradient descent systems. Other benefits of the algorithm are that no initial knowledge regarding the size of the input domain is required, and no metric need be defined over this space (as opposed to RFWR which requires knowledge of the input domain to set the initial size of the local models, and instance-based methods that require a metric). Pruning, while having little impact in the smaller domains, is seen to reduce the tree size with no detrimental effect on prediction accuracy in a more complex environment.

Having developed a method for rapidly learning nonlinear models of dynamic environments, 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 perhaps a similar approach can be applied to incrementally induced linear model trees.

References

Alexander, W., & Grimshaw, S. (1996). Treed regression. *Journal of Computational and Graphical Statistics*, *5*, 156–175.

Atkeson, C., Moore, A., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, *11*, 11–73.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees.* Wadsworth.

Chaudhuri, P., Huang, M., Loh, W., & Yao, R. (1994). Piecewise-polynomial regression trees. *Statistica Sinica*, *4*, 143–167.

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

Dobra, A., & Gehrke, J. (2002). SECRET: A scalable linear regression tree algorithm. *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Frank, E., Wang, Y., Inglis, S., Holmes, G., & Witten, I. (1998). Using model trees for classification. *Machine Learning*, *32*, 63–76.

Karalic, A. (1992). Employing linear regression in regression tree leaves. *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 440–441).

Li, K., Lue, H., & Chen, C. (2000). Interactive tree-structured regression via principal Hessian directions. *Journal of the American Statistical Association*, *95*, 547–560.

Loh, W. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, *12*, 361–386.

Malerba, D., Appice, A., Bellino, A., Ceci, M., & Pallotta, D. (2001). Stepwise induction of model trees. *AI*IA 2001: Advances in Artificial Intelligence, Lecture Notes in Artificial Intelligence, 2175.* Springer.

Nakanishi, J., Farrell, J., & Schaal, S. (2002). A locally weighted learning composite adaptive controller with structure adaptation. *IEEE International Conference on Intelligent Robots and Systems.*

Quinlan, J. (1993). Combining instance-based and model-based learning. *Proceedings of the 10th International Conference on Machine Learning* (pp. 236–243).

Schaal, S., & Atkeson, C. (1998). Constructive incremental learning from only local information. *Neural Computation*, *10*, 2047–2084.

Sicilano, R., & Mola, F. (1994). Modelling for recursive partitioning and variable selection. *Proceedings of Computational Statistics: COMPSTAT '94* (pp. 172–177).

Slotine, J., & Li, W. (1991). *Applied nonlinear control.* Prentice-Hall.

Torgo, L. (1997). Functional models for regression tree leaves. *Proceedings of the 14th International Conference on Machine Learning* (pp. 385–393).

Utgoff, P., Berkman, N., & Clouse, J. (1997). Decision tree induction based on efficient tree restructuring. *Machine Learning*, *29*, 5–44.

Vijayakumar, S., & Schaal, S. (2000). Locally weighted projection regression: Incremental real time learning in high dimensional space. *Proceedings of the 17th International Conference on Machine Learning* (pp. 1079–1086).