Reinforcement Learning Trees

Submitted for oral presentation in the Algorithms and Architectures category.

Tomas Landelius
tc@isy.liu.se

Magnus Borga
magnus@isy.liu.se

Hans Knutsson
knutte@isy.liu.se

Computer Vision Laboratory
Linköping University
S-581 83 Linköping
Sweden

Abstract

Two new reinforcement learning algorithms are presented. Both use a binary tree to store simple local models in the leaf nodes and coarser global models towards the root. It is demonstrated that a meaningful partitioning into local models can only be accomplished in a fused space consisting of both input and output. The first algorithm uses a batch like statistic procedure to estimate the reward functions in the fused space. The second one uses channel coding to represent the output- and input vectors allowing a simple iterative algorithm based on competing subsystems. The behaviors of both algorithms are illustrated in a preliminary experiment.

1 INTRODUCTION

The aim with our research is to develop efficient learning algorithms for autonomous systems (e.g. robots). Such a system is supposed to act in a closed loop with its environment, i.e. the system's output will influence its input. In this case a supervised learning algorithm would not be very useful since it would be impossible for a teacher to foresee all possible situations that can occur in an interaction with a realistic environment. This means that the system must be able to learn from its own experiences and therefore a *reinforcement learning system* is preferable. In reinforcement learning the system is not given the desired responses but only a scalar reward that tells the system how good or bad it has performed. The goal for

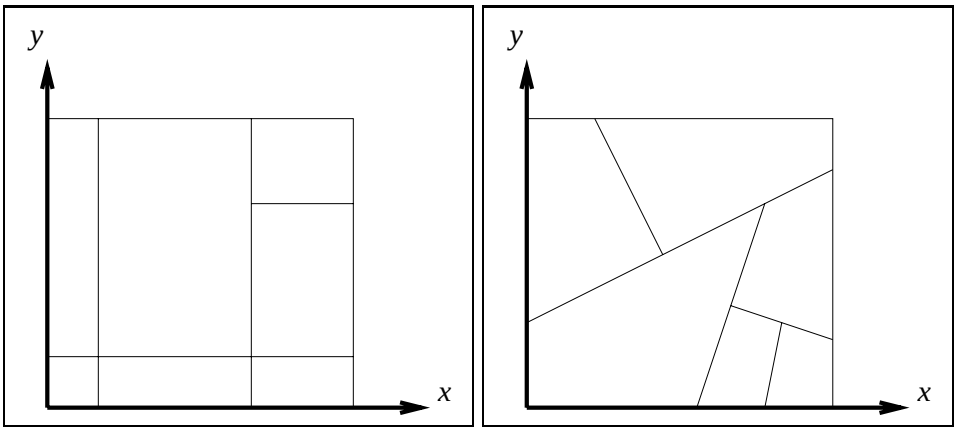

Figure 1: Left: The partitioning of the decision space along the axes made by a k-d tree. Right: The partitioning along hyperplanes.

the system is to maximize the received reward and it does this by searching for the stimuli-response function that gives the highest reward.

An autonomous system must also be able to handle a very large amount of input data in order to to use inputs from different sensors (e.g. visual, tactile, etc.). The dimensionality of the output will probably be smaller but the total dimensionality of the decision space (i.e. input *and* output) will be very large and therefore a structure that can handle problems of this size must be used.

The problems associated with the high dimensionality of the decision space is fought with a divide-and-conquer strategy. The strategy is based on a partitioning of the decision space and the assignment of local models to the emerged partitions. It should be stressed that the space to be partioned is a fusion of *both* the input and output spaces. Note that the input can consist of any information available for the system at the time of the decision, i.e. also old input and output. Partitioning only the input space is meaningless since there is no way to decide whether a given subset of this space should be assigned a model or not. This is however possible in the decision space where input and output which are worth representing are charcterized by a high reward. The fundamental idea with all structures for space partitioning is that signal space properties should be connected with properties of the represented signal.

An incoming stimulus is described in terms of space coordinates which the structure converts to a stored output model. Each node represents not only a subset of the signal space but also the reinforced signals within this subset as a signal model. Since only a small portion of the systems's experience will be relevant to any specific situation, models are preferred which decomposes the experience into components which are directly affected only by a small number of model parameters.

To bulid models the system will have to estimate parameters. Parameter fitting usually results in good generalization but has a fundamental problem in overfitting, i.e. having insufficient data to validate the model at hand as useful for generalization. A solution to the problem of overfitting is to start with a small number of coarse

models, when only a small amount of data is available. Then, as more data arrives, more complex models could be validated. Note that if some performance criteria is available it is possible to stop the model complexity from growing larger than necessary. This makes it possible to vary the model complexity across the signal space. These estimation procedures resembles to the use of regression and neural trees [6, 8].

A major advantage with tree strucures is that it makes it possible to benefit from the signal models being local by using branch and bound techniques. No effort has to be spent on models that are invalid in the current context. Instead the search proceeds down the most promising branch. A well known binary tree structure is the *k-d* tree which partitions the signal space into hyperrectangles along the coordinate axis [1]. The structures used in this paper can however split the space along any direction. This means that the tree stores information both about the direction of the split, and also where along this direction the split is made, figure 1 (right). The directional vector can be seen as the normal vector of the hyperplane that partitions the space at a node. Compared to the *k-d* tree, figure 1 (left), this structure is very flexible and only an extra scalar product between the addressing vector and the normal vector has to be done in the retrieval procedure.

2 AN ADAPTIVE BATCH APPROACH

Experienced decisions, i.e. input-output combinations, provide the raw material from which a tree representation of the distribution of decisions is grown in this approach. Storing large amounts of data should not be a big problem in the near future. Relying only on the experienced data leads to a behavioristic view since experiences are considered to be the *only* source of information available to the system in its struggle for reinforcement. All good enough decisions are summarized in local models which are stored and addressed using a binary tree structure. Good decisions need to be highly reinforced and considered to contribute to the knowledge acquired so far. When the number of new experiences are considered to be large enough to motivate a new estimation of the behavior distribution, the decision distribution is re-estimated in order to improve system behavior, which is why this approach is said to be batch-like. The procedure is also adaptive since the tree is re-grown when a certain percentage of the stored decisions have been exchanged with better ones. Such a policy leads to the tree being re-grown more often at the beginning of the learning phase, when the system is more or less without any knowledge of what to do, than later on when the system has more experiences to base its output generations on. The ideas behind this approach are presented in more detail elsewhere [4].

Given the tree and new input data, what is the system to do next? The tree is the system's guide to output generation and must be asked for an output likely to receive a high reinforcement. In order to answer the question not only should the probability function $p(\mathbf{d})$, for the decisions $\mathbf{d} = (\mathbf{x}, \mathbf{y})$, be represented but also, at least indirectly, the conditioned distribution $p(\mathbf{y} \mid \mathbf{x})$. This distribution is proportional to the distribution of decisions where the input is fix, $\mathbf{x} = \mathbf{x}_0$:

$$p(\mathbf{y} \mid \mathbf{x}_0) = \frac{p(\mathbf{x}_0, \mathbf{y})}{p(\mathbf{x}_0)} \propto p(\mathbf{x}_0, \mathbf{y}). \tag{1}$$

Hence representing $p(\mathbf{x}, \mathbf{y})$, will allow for an ouput to be generated to the current

input $\mathbf{x}_0$, by selecting an output at random from the distribution $p(\mathbf{x}_0, \mathbf{y})$. Each of the nodes in the tree contains a normal distribution estimating the behavior distribution in its part of the decision space. The global behavior distribution, $p(\mathbf{d})$, can be found by expanding the sum of local normal distributions top-down from the root down to the leaves:

$$p(\mathbf{x}, \mathbf{y}) = \alpha_l \; p_l(\mathbf{x}, \mathbf{y}) + \alpha_r \; p_r(\mathbf{x}, \mathbf{y}) = \ldots = \sum \alpha_i \; p_i(\mathbf{x}, \mathbf{y}). \tag{2}$$

Here the indices l and r referres to the distributions in the left and right child of the root node. These two distributions are themselves sums of the distributions in their childs and so on. The recursion stops with the distribution as a sum of the leaf distributions as seen in equation 2. It can be shown that the projection of a normal distribution onto the hyperplane $\mathbf{x} = \mathbf{x}_0$, specified by the current input, is a new normal distribution, $p(\mathbf{y})$, times a constant:

$$p(\mathbf{y} \mid \mathbf{x}_0) \propto p(\mathbf{x}_0, \mathbf{y}) = c(\mathbf{x}_0, \mathbf{m}, \mathbf{C}) \; p(\mathbf{y}), \tag{3}$$

where $\mathbf{m}$ and $\mathbf{C}$ are the mean vector and covariance matrix respectively. The projection property of the normal distribution allows the global conditioned distribution $p(\mathbf{y} \mid \mathbf{x}_0)$ to be calculated as a linear combination of the conditioned leaf distributions, yielding the distribution of possible output as a new sum of normal distributions:

$$p(\mathbf{y} \mid \mathbf{x}_0) \propto \alpha_l \; p_l(\mathbf{x}_0, \mathbf{y}) + \alpha_r \; p_r(\mathbf{x}_0, \mathbf{y}) = \alpha_l \; c_l \; p_l(\mathbf{y}) + \alpha_r \; c_r \; p_r(\mathbf{y}). \tag{4}$$

Generating a random output from the conditioned distribution in equation 4 is quite easy. When a probability density function can be expressed as a linear combination

$$p(x) = \beta_1 p_1(x) + \beta_2 p_2(x) + \ldots + \beta_n p_n(x) \;, \quad \sum \beta_i = 1 \;, \quad \beta_i > 0 \;, \tag{5}$$

where $p_1(x), \ldots, p_n(x)$ are probability density functions, the following two step procedure can be applied to produce a sample from the total distribution $p(x)$ [5]:

1. Generate a random integer, l, being 1 with probability β_1, 2 with probability β_2 and so on.
2. Generate a random variable from the probability density function $p_l(x)$ and let it be the output.

Using this method recursively from the top node and downwards through the tree results in a stochastic traversing of the tree in search for the leaf model to be used for output generation. In each node one of its children will be selected in the first step of the procedure, but since it too will be a sum of two distributions the process will continue until a leaf is reached and the output is generated.

The behavior distribution is estimated by partitioning the decision space $D = X \times Y$ containing the experienced decisions, which are seen as samples from the behavior distribution, and storing local models of the decision distribution in the nodes of a binary tree structure. Within each partition the local decision distribution is modeled with a normal distribution, which parameters are stored in the corresponding node in the tree. The idea behind the approach to employ local normal distributions in the approximation originated from the successful use of tensors as local signal descriptors [3]. When growing the tree a recursive algorithm is first applied to the whole set of stored decisions. A mean decision vector and a covariance matrix are calculated and stored in the root node. The decision space is then partitioned into

two halves across the direction of maximal data variation. This procedure is repeated recursively for each of the two halves of the decision space, forming a binary tree with mean vectors and covariance matrices in its nodes. When the fraction between the second largest and the largest eigenvalue of the covariance matrix is below a given threshold the procedure is halted. The assumption underlying this stop criterion is that the behavior distribution is concentrated along curves in the decision space.

The depth of a tree representing n models will be of order $\mathcal{O}(\log_2 n)$. Growing the tree is of complexity $\mathcal{O}((k+m)^2 n log_2 n)$, where k and m are the dimensionality of the input- and output spaces respectively. The factor $(k+m)^2$ is due to the computation of the eigenvector of the covariance matrix corresponding to the direction of maximal data variation along with the two largest eigenvalues. Output generation needs $\mathcal{O}(\log_2 n)$ steps as mentioned earlier and in each step the coefficients $c(\mathbf{x}_0, \mathbf{m}, \mathbf{C})$ need to be calculated which is of order $\mathcal{O}((k+m)^2)$ since it involves a matrix-vector product. Hence output generation ends up to be of order $\mathcal{O}((k+m)^2 \log_2 n)$.

3 AN ITERATIVE ALGORITHM

In this section an iterative learning algorithm is presented. This means that the tree is grown "on line", i.e. changes are made in the nodes and split- and prune criteria are tested after each input-output pair.

In the previous section the signal vectors $\mathbf{x}$ and $\mathbf{y}$ were directly used as representation of the information fed into and out from the system. Another way to represent information is the so called *channel representation* [7]. In this representation a set of channels is used, where each channel is sensitive to some specific feature value in the signal. The coding is designed so that the channel has its maximum value (e.g. 1) when the feature and the channel are exactly tuned to each other, and decreases to zero in a smooth way as the feature and the channel become less similar. This implies that each channel can be seen as a response from a filter that is tuned to some specific feature value. The signal vectors $\mathbf{x}$ and $\mathbf{y}$ are mapped into the *direction* of the channel vectors $\mathbf{v}$ and $\mathbf{q}$ respectively. In this way the system can disregard the length of the input- and output vectors which allows for a simple learning algorithm.

In this algorithm the output channel vector $\mathbf{q}$ is a linear function of the input channel vector $\mathbf{v}$. This means for the whole system that a matrix $\mathbf{W}$ is trained so that a correct output vector is generated as

$$\mathbf{q} = \mathbf{W}\mathbf{v}. \tag{6}$$

In reinforcement learning the correct output is not known and the only feedback is a scalar r that is a measure of the performance of the system. But the reward signal is a function of the input vector $\mathbf{v}$ and the output vector $\mathbf{q}$. If this function can be represented in the system, then the best output for each input can be found. Assume that a prediction p of the reward r can be approximated with a linear function of the outer product between $\hat{\mathbf{q}}$ and $\hat{\mathbf{v}}$, that is

$$p = \langle \mathbf{W} \mid \hat{\mathbf{q}}\hat{\mathbf{v}}^T \rangle, \tag{7}$$

where $\langle \cdot \mid \cdot \rangle$ denotes the scalar product. Then $\mathbf{W}$ can be trained in the same manner as in supervised learning but now with the aim at minimizing the error function

$$\epsilon = |p - r|^2. \tag{8}$$

Consider a system that has learned this reward function. How should a proper response be chosen by the system? The prediction p in equation (7) can be rewritten as

$$p = \langle \mathbf{W}\hat{\mathbf{v}} \mid \hat{\mathbf{q}} \rangle. \tag{9}$$

The choice of $\hat{\mathbf{q}}$ that gives the highest predicted reward is obviously the $\hat{\mathbf{q}}$ that lies in the same direction as $\mathbf{Wv}$. Now, if p is a good prediction of the reward r for a certain input vector $\mathbf{v}$ that choice of $\hat{\mathbf{q}}$ would be the one that gives the best reward. This yields the following equation for the systems response:

$$\hat{\mathbf{q}} = \frac{\mathbf{Wv}}{|\mathbf{Wv}|}. \tag{10}$$

This equation together with equation (9) gives the prediction as

$$p = |\mathbf{W}\hat{\mathbf{v}}|. \tag{11}$$

Now we have a machinery that both calculates a proper output for a certain input *and* a prediction of the reward for that input-output pair.

The linear functions learned by this system will obviously not solve any complicated problems. One solution is to use several linear models in a tree structure. The main idea is to have a sufficiently large set of linear functions to be able to solve the problem by, for each input vector, selecting the linear model that is likely to give the best response. Consider the simple example where two models are available. Since each model gives a prediction of the reward based upon the input vector, the obvious choice would be the model that gives the highest predicted reward. This example is in fact quite general since this is the decision that has to be made for each node in a binary tree.

Since the size of the tree in general can not bee known in advance we will have to start with a single node and grow the tree until a satisfying solution is obtained. Hence, we start the learning by trying to solve the problem with a single linear function. This node will now try to approximate the optimal function as good as possible. If this solution is not good enough (i.e. a low average reward is obtained) the node will be split. To find out when the node has converged to a solution (optimal or not optimal) a measure of consistency of the changes of $\mathbf{W}$ is calculated as

$$c = \left| \frac{\sum \Delta \mathbf{W}}{\sum |\Delta \mathbf{W}|} \right|. \tag{12}$$

These sums are accumulated continuously during the learning process. Note that the consistency measure is normalized so that it does not depend upon the step length $\Delta\mathbf{W}$ in the learning algorithm. Now, if a node is split then two new nodes will be created, each with a new weight matrix. The nodes will be selected with a probability that depends on the predicted reward and the weight matrices that are used will be the sum of all the matrices from the root node down to the leaf node, i.e. $\mathbf{W} = \mathbf{W}_0 + \mathbf{W}_{0k_1} + ... + \mathbf{W}_{0k_1..k_n}$, where $k_j \in \{1, 2\}$ and n is the depth of the tree at that leaf node. When a node's matrix is updated the father will be updated so that it remains in between its two children in the decision space. This means that if a node is split accidently and that node's children converge to the same solution the node itself will also contain this solution. If this is the case the two redundant children could be pruned away.

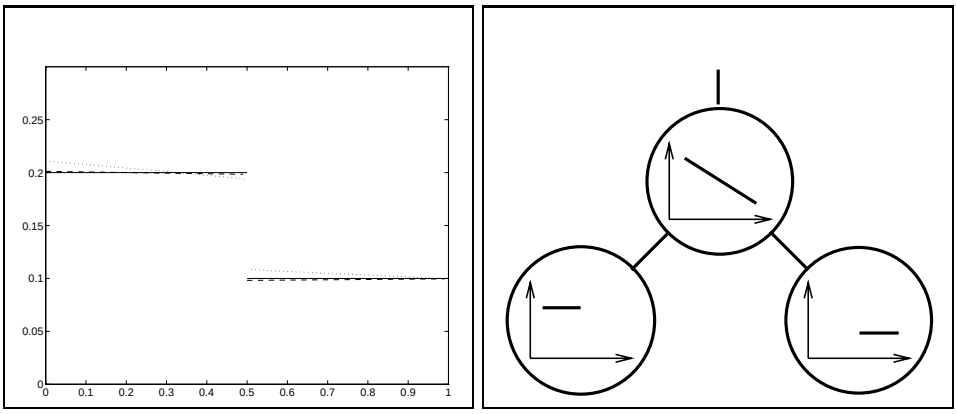

Figure 2: Left: The result with the batched algorithm (dashed) and the iterative (dotted) compared to the optimal solution (line). Right: Illustration of the solutions in the root and the two leaf nodes.

This tree is computationally very simple. The response generation is of the order $\mathcal{O}(\log_2 n)$ multiplications of an $k \times m$ matrix with a k-dimensional vector, where n is the number of leaves, k is the dimension of the input vector and m is the dimension of the output vector. The learning process takes another $\log_2 n$ additions of $k \times m$ matrices. The algorithm is described more in detail in [2].

4 EXPERIMENTS

The reason behind the choice of experiment is not to make an impression but to illustrate the principles of representing local signal models in a tree structure. Our research is still in its infancy but we believe our approach to be applicable to larger and more realistic problem settings. The correct solution for the experiment at hand, illustrated in figure 2, is written:

$$y(x) = \begin{cases} 0.2 & , \quad x \in [0, 0.5] \\ 0.1 & , \quad x \in [0.5, 1] \end{cases} \tag{13}$$

An exponentially decaying function, with its maximum along the correct solution, was used for rewarding purposes. The 500 inputs that were presented to the systems came from a rectangular distribution on the interval $[0, 1]$. Both systems were allowed to use two models for representing the goal function and their outputs, produced after the learning period, are presented together with the correct solution left in figure 2. To the right in the figure the hierarchical decomposition of the tree's solution is outlined. In the root node a global linear approximation of the whole decision space is made. The two leaf nodes then divide the space to make better local approximations.

5 DISCUSSION

In contrast to standard neural networks which maintain a global model, the proposed tree structures models the input-output mapping locally. The necessity to

incorporate the output in the organizational process becomes obvious from a behavioristic point of view. It is unlikely to find relevant structures in the input space alone, since it is the relation between input and output that determines the system's behavior.

In the simple experiment we demonstrated the principles behind the partitioning of the signal space. When studying the differences in performance between the two approaches the following should be noted. The adaptive batch approach rebuilds the tree when it is time to update the structure while the iterative approach modifies the structure every time a new decision is experienced. While the batched approach provides a good estimate it is computationally more expensive than the iterative approach. This difference stems from the fact that the two algorithms uses different signal spaces.

The approaches suggested in this paper enables the system to explore its environment starting with a simple model and then gradually refining its behavior. Neither a complete set of training data nor knowledge about the complexity needed in the model is required from the designer of the system. We believe that a tree structure having nodes based on reinforcement learning could play in important role in the future design of autonomous systems.

References

[1] Jon Louis Bentley. Multidimensional binary trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[2] M. Borga and H. Knutsson. A binary competition tree for reinforcement learning. Report LiTH-ISY-R-1623, Computer Vision Laboratory, S–581 83 Linköping, Sweden, 1994.

[3] H. Knutsson. Representing local structure using tensors. In *The 6th Scandinavian Conference on Image Analysis*, pages 244–251, Oulu, Finland, June 1989. Report LiTH–ISY–I–1019, Computer Vision Laboratory, Linköping University, Sweden, 1989.

[4] T. Landelius. Behavior representation by growing a learning tree, September 1993. Thesis No. 397, ISBN 91–7871–166–5.

[5] I. Mitrani. *Simulation techniques for discrete event systems.* Cambridge University Press, 1982.

[6] J. N. Morgan and J. A. Sonquist. Problems in the analysis of survey data, and a proposal. *J. Amer. Stat. Assoc.*, 58:415–434, 1963.

[7] K. Nordberg, G. Granlund, and H. Knutsson. Representation and learning of invariance. Report LiTH–ISY–I–1552, Computer Vision Laboratory, Linköping University, Sweden, 1994.

[8] Paul E. Utgoff. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, 1:377–391, 1989.