Efficient Hierarchical MCMC for Policy Search

Malcolm Strens MJSTRENS@QINETIQ.COM
Room G020, A9 Building, QinetiQ, Ively Road, Farnborough, Hampshire, GU14 0LX, UK.

Abstract

Many inference and optimization tasks in machine learning can be solved by sampling approaches such as Markov Chain Monte Carlo (MCMC) and simulated annealing. These methods can be slow if a single target density query requires many runs of a simulation (or a complete sweep of a training data set). We introduce a hierarchy of MCMC samplers that allow most steps to be taken in the solution space using only a small sample of simulation runs (or training examples). This is shown to accelerate learning in a policy search optimization task.

1. Introduction

Many tasks in machine learning can be converted into a sampling problem with a target density $\pi(w)$ of the form:

$$\pi(w) = \frac{1}{Z}\exp(f(w))$$

Typically w is a real-valued vector, for example the weights of a neural network or parameters in a model. The exponent $f(w)$ takes on various meanings depending on the task; it is variously termed the *error function*, *fitness function*, *objective function*, *expected return*, *energy* or *log likelihood* depending on the setting. Nevertheless, it is often formed from a sum over a set of instances (weighted by some constant K):

$$f(w) = K\sum_{i=1}^{N} f_i(w) \qquad (1)$$

This is the special *additive* structure exploited by our algorithm. For convenience we will also define $p(w) \equiv \exp(f(w))$ for the *unnormalized* target density.

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

Policy search is the process of optimizing a control policy using repeated simulation runs ("trials"). Let the policy be defined by a set of parameters w; for example the gains in a conventional control law, or the weights of a neural network controller. The outcome of each trial depends on both the state in which the simulation is started, and stochasticity in the simulator itself. A single scalar performance measure called the *return* is assumed to be obtained at the end of each trial. For any given policy, many trials may be required to obtain an accurate estimate of the *expected* return (the objective for optimization).

A *scenario* is a particular choice for the initial state and the random number sequence[1] to be used by the simulation (Ng & Jordan, 2000). Given the choice of w and a scenario (indexed by i), a simulation *trial* is deterministic (repeatable) and yields a return $f_i(w)$. A deterministic objective function for policy search can be formed by averaging the empirical returns for a given policy over a large set of scenarios of size N. Hence $f(w)$ may take the form of equation (1) with $K = 1/N$.

Using this estimate for policy evaluation, we can query $p(w) \equiv \exp(f(w))$ at arbitrary policies w, and a sampling method such as Markov Chain Monte Carlo (MCMC) can be applied to explore this density. To achieve optimization rather than sampling, the shape of the target density can be gradually "sharpened" during MCMC sampling. This *simulated annealing* process (Kirkpatrick et al., 1983), subject to strict conditions, causes the sampling chain to become trapped at w^*, a global maximum of $p(w)$. The sharpening is achieved by setting $K = 1/(NT)$ where the temperature T can be gradually reduced to zero during the sampling process.

Unfortunately, the number of scenarios N represents a very difficult trade-off between accurate policy evaluation and the number of MCMC steps that can be achieved in a given amount of computation time.

[1] A random number sequence can be specified as a single "seed" for a pseudo-random number generator.

We will propose a new algorithm that examines only $O(\log N)$ scenarios to achieve a similar policy improvement as would be achieved using the full N scenarios in conventional MCMC. It does this by making use of sequences of steps using inaccurate policy evaluations (few scenarios) to obtain proposed moves (changes to w) that are then accepted or rejected using more accurate policy evaluations (more scenarios).

2. Markov Chain Monte Carlo

Markov Chain Monte Carlo is a sequential form of importance sampling. It provides a simple and practical way to generate samples from a probability density $\pi(w)$ that is not available in analytical form, but can be queried for any policy-state[2] w (up to some unknown normalizing constant). The technique works by noting that we can construct a Markov chain that has $\pi(w)$ as its invariant distribution, and simulating this chain to obtain a sequence of samples. Consecutive samples from the chain are highly correlated, but their distribution converges to $\pi(w)$.

2.1. Metropolis Hastings Rule

Suppose the current policy-state is w_t at time step t, and that the unnormalized target density $p(\cdot)$ can be queried. A *proposal* w' is generated according to some proposal density $P(w'|w_t)$, and $p(w')$ is evaluated. The Metropolis Hastings rule (MHR) (Hastings, 1970) can then be applied to stochastically accept or reject the proposal as the new policy-state of the chain:

$$w_{t+1} = \begin{array}{ll} w' & \{u < \frac{p(w')P(w_t|w')}{p(w_t)P(w'|w_t)}\} \\ w_t & \text{otherwise} \end{array}$$

where u is drawn uniformly in the range $[0, 1]$.

2.2. MCMC Algorithm

Notation is now introduced to work with subsets of the complete set of N scenarios. The objective function for subset j (of size n) is defined as:

$$f_{n,j}(w) \equiv \sum_{i=nj-(n-1)}^{nj} \frac{f_i(w)}{nT}$$

where $j \leq N/n$. For a simple MCMC implementation we are concerned only with the special case $f_{N,1}(w)$ which is identical to $f(w)$; the more general form will be needed later for defining a hierarchy.

[2]Although w is just called a *state* in MCMC terminology, we will refer to it as a *policy-state* to recognize that it will be used for policy search.

Input: w_{in} Current policy-state
Input: n Number of scenarios forming the sum
Input: j Counter
Output: w_{out} Proposed new policy-state

1 $w' \leftarrow w_{in} + N(0, \sigma^2)$
2 $\delta f \leftarrow f_{n,j}(w') - f_{n,j}(w_{in})$
3 **if** accept(δf) **then**
4 $\quad w_{out} \leftarrow w'$
else
5 $\quad w_{out} \leftarrow w_{in}$
end

Algorithm 1: $w_{out} \leftarrow \text{mcmc\_step}(w_{in}, n, j)$

A single MCMC step is defined by algorithm 1. Line 1 perturbs the input policy-state (w_{in}) by Gaussian noise (s.d. σ) to obtain a proposal w'. This specific choice of the proposal density is symmetrical, and so the inequality used in the MHR reduces to the simple Metropolis rule (Metropolis et al., 1953):

$$u < \frac{p(w')}{p(w)}$$

Substituting the quantity of interest, we have:

$$u < \frac{\exp(f_{n,j}(w'))}{\exp(f_{n,j}(w))}$$

Writing $\delta f \equiv f_{n,j}(w') - f_{n,j}(w)$, we implement the Metropolis rule by defining the stochastic function $\text{accept}(\delta f)$ to draw a new value of u uniformly from the range $[0, 1]$, then test the condition $u < \exp(\delta f)$ (returning true or false accordingly).

mcmc_step can be called repeatedly with $n = N$ and $j = 1$ to obtain an unbiased chain of samples from the target density: w_{out} from one call is used as w_{in} for the next. Note that $f_{n,j}(\cdot)$ is assumed to be available from a procedure call. Each step in this chain requires N trials. Our hierarchical algorithm will achieve $O(N)$ such primitive (σ-sized) steps using only $\text{O}(N \log N)$ trials. (The complexity analysis assumes some non-zero lower bound on achieved acceptance rates.)

2.3. Conditions for Unbiased Sampling

The proposal density and acceptance rule together determine the transition function $P_T(w'|w)$ of the Markov chain. The acceptance rule ensures that π is a fixed point of the transition function:

$$\pi(w') = \int_w \pi(w) P_T(w'|w) \mathrm{d}w$$

However this property is not sufficient to ensure that samples from the chain will converge to accurately approximate π. We also require that every policy-state be reachable in a finite number of steps from every other policy-state (ergodicity) and there to be no constraint on the time phase at which a policy-state can be reached (aperiodicity). Many proposal densities, including additive Gaussian noise, are adequate to meet these conditions. Asymmetric proposal densities can also be used if the full MHR is applied. Bounded domains can be accounted for by rejecting proposals that break the hard constraints.

2.4. Hierarchical formulation of MHR

The Metropolis Hastings rule can be implemented by calling $\text{accept}(\delta f - \theta)$ where the additional "Hastings correction" term $\theta \equiv \log P(w'|w) - \log P(w|w')$ accounts for asymmetry in the proposal density. In our hierarchical formulation, a *lower level* sampling chain will be the source of this proposal density. There may have been a series of steps in obtaining w' from w, and so θ becomes a sum of (logged probability ratio) terms over the lower level steps. The detailed balance property of each lower-level application of the MHR ensures that these terms (denoted $\delta\theta$ in algorithm 2) are already known. At the lowest level, the simple Metropolis rule can be applied. Therefore unbiased (and aperiodic) sampling at all levels can be proved by depth-first induction on the hierarchy. Proving ergodicity requires additional (weak) assumptions on the nature of the target density at each level.

2.5. Parallel Tempering

Evolutionary MCMC methods run multiple chains in parallel, and allow them to interact to obtain better samples (Liang & Wong, 2001). Parallel tempering (Liu, 2001) is a special case in which the chains are operated at several different temperatures; the lowest temperature chain corresponds to the target density of interest while the high temperature chains aid exploration. Our hierarchical sampler will also exploit this benefit by allowing temperature to differ between levels.

3. Overview of the New Algorithm

We introduce an algorithm called *hierarchical importance with nested training samples* (HINTS) that exploits the additive form of the objective function. It is applicable whenever the individual terms $f_i(w)$ are faster to compute than the complete average $f(w)$ and will be most useful for a large number N of terms, for example $N > 16$.

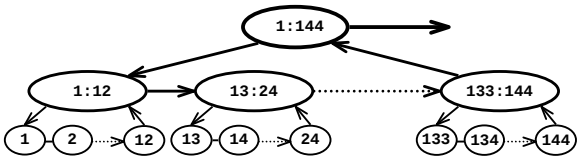

Figure 1. HINTS sampling architecture $(1 : 12 : 12)$.

3.1. Defining the Hierarchy

A tree or hierarchy defines how the complete set of scenarios (size N) is broken down into smaller subsets. Levels in the hierarchy are indexed by l with $l = 0$ for leaves and $l = L$ for the root. Let m_0 be the size of the smallest subsets (at the leaves). Let the branching factors be m_l $(l > 1)$. The number of scenarios n_l considered at level l is therefore:

$$n_l = \prod_{l'=0}^{l} m_{l'}$$

The branching factors must be chosen so that $n_L = N$; *i.e.* the complete set is considered at the root node. For example, figure 1 shows a HINTS tree with $L = 2$, $m_0 = 1$ and $m_1 = m_2 = 12$. Therefore $N = n_2 = 144$. Nodes in the figure are labelled with their associated scenario subsets.

Associated with each node in the hierarchy is the corresponding approximation for the objective function. For node j at level l this is given by $f_{n_l,j}(w)$ using the MCMC notation, or $f_j^l(w)$ for compactness. Thus the complete objective function $f(w) \equiv f_{N,1}(w) \equiv f_1^L(w)$ is associated with the root node, whereas averages of only m_0 terms are found at each leaf node. Also each non-leaf node is the average of its own children[3]:

$$f_j^l(w) = \frac{1}{m_l} \sum_{k=1}^{m_l} f^l_{\text{child}(l,j,k)}(w)$$

where $\text{child}(l, j, k) \equiv m_l(j - 1) + k$ is the index of the k^{th} child.

3.2. Illustration of HINTS Operation

HINTS uses the observation that average returns for subsets of scenarios can provide useful information about the full objective $f(w)$ (which depends on all N scenarios). In particular, a sequence of moves is taken at some level in the hierarchy to propose a move at a higher level. This allows many small steps to be combined before a costly policy evaluation with N scenarios takes place at the root. Before formalizing this process we give an illustrative example.

[3]This is true only when there is no temperature difference between level l and level $(l - 1)$.

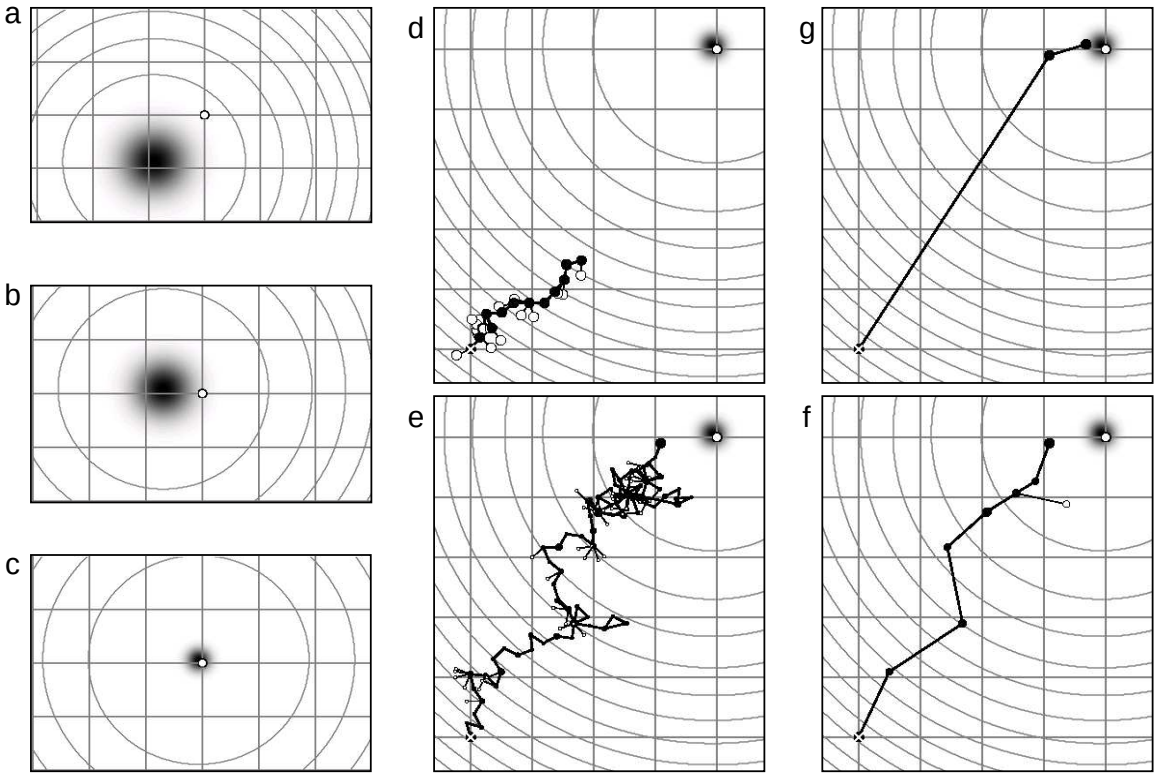

Figure 2. Comparison between MCMC and HINTS: (a-c) probability densities for 1, 8 and 128 scenarios showing decreasing bias and variance; (d) MCMC ($N = 128$) progress after 2048 trials; (e) HINTS progress after 1024 trials; (f) HINTS level 4 policy-states; (f) HINTS root node policy-states (2048 trials).

Consider a simple task that involves aiming (of a bow and arrow, for example) to hit a target. Suppose this aiming "policy" is specified as a 2-element vector w defining a vertical and horizontal displacement from some origin. Let the true optimal policy be w^*, but assume that the intended aim is corrupted by noise (*e.g.* wind turbulence). Specifying a set of scenarios means obtaining a sample set $\{\nu_i\}$ of size N from the noise distribution. The miss distance for policy w in scenario i is therefore $\epsilon_i \equiv ||w + \nu_i - w^*||$. In this illustration one simulation "trial" corresponds to a single miss distance calculation, but in genuine applications each trial involves a costly simulation run. The optimization objective is to minimize the squared miss distance, which can be achieved by defining $f_i(w) = (-\epsilon_i^2)$.

We assume that $N = 128$ scenarios would be adequate to obtain a good approximation for expected return. A sampling hierarchy is defined with $m_0 = 1$ and $m_l = 2$ for $1 \leq l \leq 7$. Therefore we have a binary tree with $N = n_7 = 128$ leaf nodes. The hierarchy is operated at very low temperature to obtain optimization rather than sampling ($T_l = (8 - l)/50$).

Figure 2a marks the optimal policy w^* with a small circle. Gray-levels are used to show the probability density $\exp(f_{1,1}(w))$ corresponding to the first leaf node in the hierarchy. This density is obtained using only a single scenario and therefore exhibits a large bias (its mode is displaced significantly from the target center). Therefore a sample drawn from this density does not usually provide a good estimate for w^*. Figures 2b and 2c show the probability densities $\exp(f_{8,1}(w))$ and $\exp(f_{128,1}(w))$ corresponding to $N = 8$ and $N = 128$ scenarios respectively. These are used at levels $l = 3$ and $l = 7$ (root node) respectively. By increasing N, the bias and variance are both reduced. Any sample at $N = 128$ is a good estimate for w^*.

Figure 2d shows a MCMC sampling chain for the density shown in 2c. The initial policy-state w is shown by a cross and is at a displacement of (-4,-5) from w^*. Solid and empty markers indicate MCMC proposals that are accepted and rejected, respectively. Each proposal is a displacement of 0.25 units in a random direction, and is accepted or rejected using policy evaluations with all 128 scenarios. Note that the policy-state of the chain makes regular but slow progress towards w^*. The progress shown required 2048 trials. Eventually, the chain would reach the small high-density area

around w^* and optimization would have been achieved.

Figure 2e shows the sampling progress of the HINTS algorithm using a total budget of only 1024 trials. In this period the root node changes its value only once. The size of the circular markers is indicative of the level in the hierarchy (and hence the number of scenarios used for each policy evaluation). Most moves are made with small numbers of scenarios; the full set of scenarios is only considered at the start and finish locations. Although there is an occasional low-level move in the wrong direction, these are always rejected at some level in the hierarchy.

Inspecting the policy-states visited at level 4 in the hierarchy (figure 2f), note that most proposals are in an appropriate direction because they have been generated from sequences of accepted moves at the lower levels. (Only one such proposal is rejected.) Figure 2g shows the progress made at the root node in the hierarchy after two such periods (2048 trials). For output purposes, the policy-state at the root of the tree is the only one of interest; the lower levels are merely providing proposals. Much more progress has been made towards the optimization objective compared with the simple MCMC chain[4], using the same number of trials.

4. Definition of the New Algorithm

Our aim is to generate a sequence of samples from the unnormalized target density $p(w) \equiv \exp(f_1^L(w))$, associated with the root node. The HINTS algorithm has a simple recursive description. To make a *move* at node j of level l, a sequence of m_l lower-level *moves* are first combined to form a proposal. The proposal is then accepted or rejected using the MHR with target density $\exp(f_j^l(w))$. Therefore execution proceeds in a depth-first left-to-right ordering.

For example, in figure 1 the execution order is indicated by arrows. 12 steps are taken at leaf nodes (using one scenario each) to form a proposal. This proposal is accepted/rejected by policy evaluation using 12 scenarios in the middle layer of the tree. This process is repeated 12 times to form a proposal for the root node of the tree, where it is then accepted/rejected using all 144 scenarios. In this process, the policy-state at the root could have moved by the equivalent of up to 144 primitive steps.

Algorithm 2 implements the main recursive procedure $(w_{out}, \delta\theta_{out}) \leftarrow \text{hints\_move}(w_{in}, l, j)$ and can be seen as a direct substitute for $w_{out} \leftarrow \text{mcmc\_step}(w_{in}, n_l, j)$. It returns a new policy-state w_{out} which may be the same as or different to the input policy-state w_{in}, according to whether lower level proposals have been accepted.

[4]The simple MCMC chain is exactly equivalent to HINTS in which the hierarchy is reduced to a single root/leaf node of size 128.

Input: w_{in} Current policy-state
Input: l Level in tree (0 = leaf)
Input: j Counter
Output: w_{out} Proposed new policy-state
Output: $\delta\theta_{out}$ Hastings correction

```
1  if (l = 0) then
2      w_out ← mcmc_step(w_in, m_0, j)
3      δθ_out ← 0
   else
4      w' ← w_in
5      θ ← 0
6      for k = 1 : m_l do
7          (w', δθ) ← hints_move(w', l − 1, child(l, j, k))
8          θ ← θ + δθ
       end
9      δf ← (f_j^l(w') − f_j^l(w_in))
10     if accept(δf − θ) then
11         w_out ← w'
12         δθ_out ← δf
       else
13         w_out ← w_in
14         δθ_out ← 0
       end
   end
```

Algorithm 2: $(w_{out}, \delta\theta_{out}) \leftarrow \text{hints\_move}(w_{in}, l, j)$

If a leaf node is encountered (line 1), a simple MCMC step is taken using m_0 scenarios for policy evaluation. Otherwise, a proposal is obtained using a sequence of m_l moves at a lower level in the hierarchy. This is achieved by a sequence of recursive calls (line 7) that each update w'. If $m_0 = 1$, these recursive calls will terminate at exactly n leaf nodes, and so the proposal will be made up of n primitive σ-sized moves combined. Line 10 accepts or reject the proposal according to the MHR. The "Hastings correction" θ accumulated (in line 8) from lower-level transitions is applied to account for asymmetry in the proposal. See section 2.4 for a discussion of this quantity.

4.1. Efficient Implementation

An efficient implementation re-uses evaluations of $f_i(w)$ when the same values of i and w occur more than once. For example $f_j^l(w_{in})$ on line 9 will normally be known from the output of the preceding call when operating at the top level ($l = L$). Secondly, the

lower level evaluation $f_{(\cdot)}^{l-1}(w)$ made in the recursive call can provide part of the sum required for $f_{(\cdot)}^{l}(w)$. Finally, whenever there is a rejection, there is an opportunity to re-use a previous evaluation. These efficiency savings can be achieved by passing lists of evaluation results into and out of the hints_move procedure, or passing evaluations of $f_{n,j}(w)$ through a caching mechanism. The $O(n \log n)$ computational complexity is unchanged, but the constant of proportionality is improved.

5. Experimental Evaluation

Here, we evaluate HINTS with a "ship landing" policy search problem that is simple to describe and difficult to solve. The problem is described in terms of the altitude (and descent rate) of an air vehicle that is required to land on a ship's deck. The height of the deck changes with time depending on sea turbulence, and is modelled as a damped harmonic oscillator. The vehicle has only one action: selecting between 2 different levels of vertical thrust.

5.1. Dynamics

The vehicle's physical state is $(y, \dot{y})$. At each time step the controller must select a thrust acceleration $a \in \{-0.5, 0.5\}$. The vehicle's state is updated after each time step δt by applying the midpoint method to its dynamics, defined by $\ddot{y} = a$, subject to the limit $\dot{y} < 1$ (maximum climb rate). The deck's state is obtained by applying the midpoint method to its second order dynamics:

$$\ddot{z} = -\alpha z - \beta \dot{z} + \nu$$

where the turbulence acceleration ν is drawn independently at each time step from a zero-mean Normal distribution with standard deviation $0.25/\sqrt{\delta t}$. The constant α determines the resonant frequency and was chosen to be $\pi^2/25$. The constant β determines the amount of damping and was chosen to be $1/10$.

5.2. Performance Measure and Returns

When the vehicle hits the deck ($y \leq z$), the difference $|\dot{y} - \dot{z}|$ between their velocities determines success or failure. If the difference is less than 1, then the trial is deemed a success; otherwise it is a failure. The trial is also a failure if the maximum duration (400 s) is exceeded without the vehicle hitting the deck.

The initial physical state is always $(y, \dot{y}) = (100, -1)$ and $(z, \dot{z}) = (0, 0)$, and so the only source of randomness in trial returns is the turbulence. Therefore each scenario i will be fully defined by a single random number seed, yielding a deterministic return $f_i(w)$ for controller parameters w.

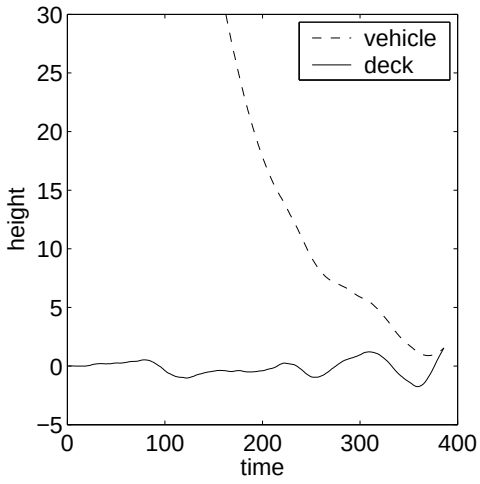

Figure 3. Height of vehicle and deck in a successful trial.

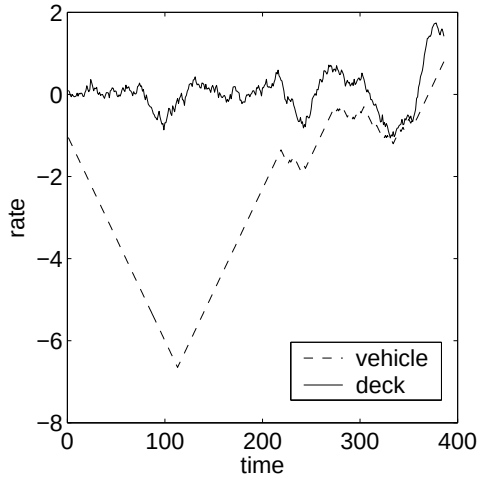

Figure 4. Velocity of vehicle and deck in a successful trial.

Success rate will be the measure of performance, but this does not provide a useful objective function for learning because it does not indicate the degree of failure and hence the direction of policy improvement. Therefore the return used in learning is given at trial end time (t) by:

$$f_i(w) = \frac{1 - t/400}{1 + |\dot{y} - \dot{z}|}$$

unless $t > 400$ in which case $f_i(w) = 0$. This assigns large returns for closely matching the velocity of the vehicle to the velocity of the deck, and for completing the trial as soon as possible. Return is always in the range $[0, 1]$.

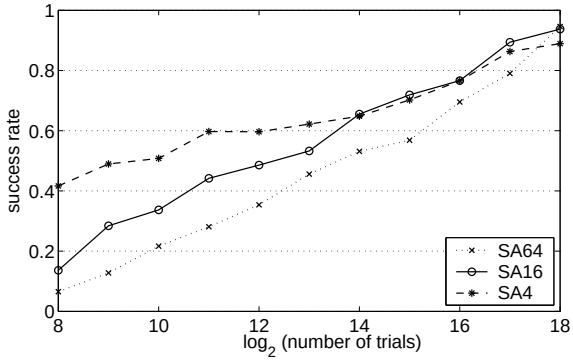

Figure 5. Comparison between simulated annealing solutions.

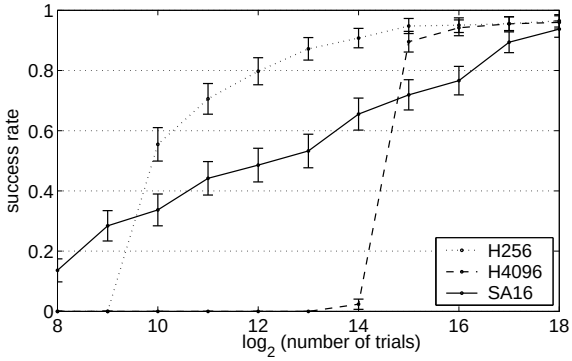

Figure 6. Comparison between HINTS and simulated annealing.

5.3. Controller Parameterization

The optimization problem is expressed as a search through a parameterized family of controllers. The controller is a nonlinear mapping from the state to the control action. A scaled state vector x is formed from $(z,\ y/10,\ \dot{z},\ \dot{y})$. The first layer of our controller applies a linear transformation to obtain the hidden unit activation vector x':

$$x' = Ax + b$$

where A is a 2-by-4 matrix of weights and b is a 2-element bias vector. The action is then determined according to the test:

$$\tanh(x'_1) > c \tanh(x'_2)$$

Together (A, b, c) parameterize this nonlinear function. These can be concatenated into a weight vector w of size 11.

5.4. Example

Figure 3 shows the heights of vehicle and deck during a successful trial (using the optimized controller). Figure 4 shows the rate of height change ($\dot{y}$ and $\dot{z}$) over the same trial. Note that the vehicle initially accelerates to increase its descent rate, then decelerates to a low rate of descent. A period follows when the controller roughly matches the vehicle's velocity to that of the deck. This ensures that the terminal constraint $|\dot{y} - \dot{z}| < 1$ is met even though the deck is rising very rapidly at the end of the trial.

5.5. Simulated Annealing Baseline

A simple approach to this task is simulated annealing (SA) with a fixed N. For small values of N we expect *fast* learning but convergence to a sub-optimal policy, because the policy is "overfitted" to the particular N scenarios that are chosen. Conversely, for large N, learning will be slow but there will be less bias in the final result. Figure 5 shows the success rate for SA with $N \in \{4, 16, 64\}$, evaluated using a test-set of 1000 unseen scenarios, and averaged over 80 runs. The temperature was reduced linearly from 0.05 to 0. (This initial temperature was obtained by trial-and-error.) The $N = 4$ case converges to a sub-optimal solution, whereas $N = 16$ appears to be adequate. Therefore SA with 16 scenarios will form a baseline against which to compare HINTS.

5.6. HINTS Performance

HINTS is applied to this optimization task by setting the temperature of the root node to 0, but operating lower levels of the hierarchy at non-zero temperature to enable exploration. The temperature of level l was chosen to be $(L - l)/(10L)$. Unlike simulated annealing, there is no change in these temperatures during learning. Essentially, the root node is acting as an optimizer whereas the lower levels of the tree are increasingly random to enable exploration.

Figure 6 shows the result of applying HINTS with binary trees having either $N = 256$ or $N = 4096$ leaves ($m_0 = 1$; $m_l = 2$ $(l > 0)$; $n_l = 2^l$). There is no limit on the size of N that could be used (subject to the total experiment duration). Error bars show standard deviations, not standard errors. HINTS with $N = 256$ is significantly better than simulated annealing from 2^{10} trials onward. Furthermore, this has been achieved with a much higher value of N, and so the final result will have less bias.

For $N = 4096$, the root node first changes its state after approximately 2^{14} (16384) trials, but immediately jumps to a good solution. The root node then changes

its value at similar intervals until the budget of 2^{18} trials is exhausted. There is no apparent improvement over $N = 256$, suggesting that no more than 256 scenarios are required to eliminate most bias.

6. Conclusions

HINTS is an algorithm for sampling from a target function where the log probability is a sum or expectation. This additive structure is very common: HINTS is applicable to many noisy optimization and inference tasks in signal processing, data fusion and machine learning. The use of HINTS for Bayesian inference will be addressed in a separate paper. This paper has focussed on policy search, where the high computational costs of simulation trials mean that a sophisticated approach is required.

The method was shown to be several times faster than simulated annealing (and has less terminal bias) in a policy search task. Much greater benefits would come in applications that have larger variation in returns between scenarios and hence require larger values of N. HINTS also has the advantage of being a statistically stationary process that can provide output at any time; there is no need for a cooling schedule. Although the chosen task was fully observable, policy search is also effective in partially observable domains. It is also feasible to use the hierarchical sampling within population-based approaches such as particle filtering for regression (Vermaak et al., 2004) and evolutionary MCMC.

Many simulation models have parameters that control the trade-off between accuracy and computation time. This could be exploited by a "variable fidelity" optimizer that uses higher accuracy simulation at the root of the tree than at the leaves. This would allow larger numbers of scenarios to be used at leaf nodes.

HINTS as described, does not make use of gradient information that may be available. For example, the controller used in our evaluation was differentiable with respect to its weights. This property is exploited by policy gradient ascent methods (Sutton et al., 2000). The *Hamiltonian MCMC* algorithm (Neal, 1993) uses gradient information to determine proposal directions. HINTS could also exploit gradient information (calculated for individual training examples) to obtain its primitive proposals. The resulting algorithm might have the efficiency of policy gradient methods, while offering *global* optimization.

Acknowledgments

This research was funded by the UK Ministry of Defence Corporate Research Programme in Energy, Guidance & Control. I acknowledge numerous helpful comments from Graham Watson, Drew Bagnell, Nick Everett, Simon Maskell and the reviewers.

References

Andrieu, C., de Freitas, N., Doucet, A., & Jordan, M. (2003). An introduction to mcmc for machine learning.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 97–109.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220, 4598*, 671–680.

Liang, F., & Wong, W. H. (2001). Real-parameter evolutionary Monte Carlo with applications to Bayesian mixture models. *Journal of the American Statistical Association, 96*, 653.

Liu, J. S. (2001). *Monte carlo strategies in scientific computing*. Springer Verlag.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics, 21*, 1087–1092.

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Technical Report CRG-TR-93-1). University of Toronto.

Ng, A. Y., & Jordan, M. (2000). PEGASUS:A policy search method for large MDPs and POMDPs. *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)* (pp. 406–415). San Francisco, CA: Morgan Kaufmann Publishers.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information processing Systems 12 (Proceedings of the 1999 Conference)* (pp. 1057–1063). MIT Press.

Vermaak, J., Godsill, S. J., & Doucet, A. (2004). Sequential bayesian kernel regression. *Advances in Neural Information Processing Systems 16 (Proceedings of the 2003 Conference)*. MIT Press.