MULTI-SCALE MCMC METHODS FOR SAMPLING FROM PRODUCTS OF GAUSSIAN MIXTURES

Daniel Rudoy and Patrick J. Wolfe

Harvard University
Division of Engineering and Applied Sciences
33 Oxford Street Cambridge, MA 02138 USA
{rudoy, patrick}@deas.harvard.edu

ABSTRACT

This paper addresses the important and ubiquitous problem of sampling from a product of Gaussian mixtures. An exact solution is often computationally infeasible, thus motivating the development of efficient sampling schemes. However, naive Markov chain Monte Carlo algorithms perform poorly in cases where the product mixture is highly multi-modal. In this paper we follow the trend of recent work utilizing multi-scale sampling methods, and propose two new multi-scale Markov chain Monte Carlo algorithms based on simulated and parallel tempering. Empirical results indicate that for the same computational budget, this class of methods can improve performance in cases with widely separated modes.

Index Terms— Gaussian Mixtures, MCMC Methods, Parallel Tempering, Gibbs Sampler, Non-Parametric Belief Propagation

1. INTRODUCTION

This paper addresses the problem of efficient sampling from a product of M Gaussian mixtures of d dimensions, each consisting of N components. This calculation arises in a number of statistical inference algorithms which manipulate probability densities that are represented by a Gaussian mixture. An important example involves inference in graphical models, where two related algorithms—Non-Parametric Belief Propagation (NBP) [1] and Sequential Auxiliary Particle Belief Propagation [2]—both require sampling from products of mixtures in order to fuse information from different nodes. Another application is to the Product of Experts framework [3], in which a probabilistic model is constructed from a product of simple distributions. As will be seen later, a key requirement of these algorithms is the generation of iid samples from the product mixture.

It is easily seen that the product density has $O(N^M)$ terms and so naive approaches to sampling are infeasible. In fact, in some applications the number of Gaussians per mixture (N) is on the order of hundreds, with the number of mixtures (M) depending on the density of the graph [1]. Several algorithms have been proposed to address this problem, but current solutions remain unsatisfactory. For instance, importance sampling was considered in [1] and [2], but was shown to perform poorly in cases where the product density contained many separated modes. A method was proposed in [4] to sample efficiently from an approximation to the product density whose quality was controlled by a parameter ϵ; this method was found to perform well for small examples, but is sensitive to dimensionality.

A number of MCMC approaches using Gibbs sampling were introduced in [1] and [4]. The slow mixing of these samplers in cases when the product density is multi-modal led the authors to consider an alternative approach based on simulated annealing [5]. By constructing a multi-scale representation of each Gaussian mixture in the product, a Gibbs sampler was designed to move from a coarse to a fine representation of the product mixture—thereby exploring the state space more widely. Nonetheless, once it moved to the finest level corresponding to the original mixture, the sampler mixed slowly once again, therefore requiring multiple short runs of the overall algorithm in order to obtain iid samples (a desire for which will be explained below). However, while multiple restarts reduce the level of statistical dependency among samples, the small number of iterations may not be sufficient to converge to stationarity, which in turn implies that the sampler may not sample from the correct target distribution.

In this paper, we propose two multi-scale MCMC algorithms based on the ideas of simulated tempering [6] and parallel tempering [7] rather than simulated annealing. The crucial feature of these algorithms is that the scale of the representation is treated as a random quantity; Markov chain moves from fine-to-coarse and coarse-to-fine representations of the product mixture are allowed. This eliminates the need for multiple restarts; instead, one long run of the chain can be used to obtain all the required samples. We show that this class of algorithms is able to outperform previously proposed multi-scale approaches given the same computational budget. In Section 2 we state the precise form of the problem to be considered and discuss the infeasibility of exact computations. We describe existing MCMC algorithms in Section 3 and present the new algorithms in Section 4. Experimental results are given in Section 5.

2. PROBLEM STATEMENT

We begin by fixing notation and presenting a number of formulae that will be used repeatedly. Let $\boldsymbol{x} \in \mathbb{R}^d$ and $\{p_1(\boldsymbol{x}), \ldots, p_M(\boldsymbol{x})\}$ denote an associated set of M Gaussian mixtures, with $\mathcal{N}(\boldsymbol{x}; \mu, \Sigma)$ denoting a Gaussian density in $\boldsymbol{x}$ having mean μ and covariance Σ. We wish to draw a sample from the product mixture $p(\boldsymbol{x})$:

$$p(\boldsymbol{x}) \propto \prod_{m=1}^{M} p_m(\boldsymbol{x});\ p_m(\boldsymbol{x}) = \sum_{n=1}^{N} w_{mn} \mathcal{N}(\boldsymbol{x}; \mu_{mn}, \Sigma_{mn}).$$

To explicitly index each component in the product mixture $p(\boldsymbol{x})$, we employ an M-dimensional vector $\boldsymbol{L} = (l_1, \ldots, l_M)$. Note that each coordinate $l_m \in \mathbb{Z}_{N-1}$, implying that $\boldsymbol{L}$ can take on N^M distinct values. Each Gaussian in this product mixture arises as a cross term in the product over all mixtures $p_m(\boldsymbol{x})$, and $\boldsymbol{L}$ indexes Gaussian components that appear in the cross term. Specifically, if $l_m = n$

then the n^{th} component of the m^{th} mixture is in the Gaussian cross-term indexed by $\boldsymbol{L}$. In addition to restricting each mixture to have an equal number of components N, we also require that N be a power of two. These assumptions are only for clarity of exposition; they are readily relaxed and the algorithms we present generalize naturally.

Given a specific label vector $\boldsymbol{L}$, the formulae needed to compute the weight, mean and covariance of the Gaussian in the associated product mixture indexed by $\boldsymbol{L}$ are as follows:

$$\Sigma_{\boldsymbol{L}}^{-1} = \sum_{m=1}^{M} \Sigma_{l_m}^{-1}; \qquad \Sigma_{\boldsymbol{L}}^{-1}\mu_{\boldsymbol{L}} = \sum_{m=1}^{M} \Sigma_{l_m}^{-1}\mu_{l_m} \tag{1}$$

$$w_{\boldsymbol{L}}\mathcal{N}(\boldsymbol{x};\mu_{\boldsymbol{L}},\Sigma_{\boldsymbol{L}}) = \prod_{m=1}^{M} w_{l_m}\mathcal{N}(\boldsymbol{x};\mu_{l_m},\Sigma_{l_m}). \tag{2}$$

These expressions allow for exact computation of all parameters in the product mixture. However, sampling directly from this mixture requires calculation of the normalizing constant $Z = \sum_L w_{\boldsymbol{L}}$, which in turn has complexity $O(N^M)$. This makes exact sampling from the product mixture a prohibitively expensive procedure, and motivates the pursuit of approximate methods. To begin, consider a joint distribution on the mixture labels l_i defined according to (2):

$$p(\boldsymbol{L}) = p(l_1,\ldots,l_M) = \frac{w_{\boldsymbol{L}}}{Z} \propto \frac{\prod_{m=1}^{M} w_{l_m}\mathcal{N}(\boldsymbol{x};\mu_{l_m},\Sigma_{l_m})}{\mathcal{N}(\boldsymbol{x};\mu_{\boldsymbol{L}},\Sigma_{\boldsymbol{L}})}\ .$$

To draw a sample from the product mixture $p(\boldsymbol{x})$, we first draw a sample from $p(\boldsymbol{L})$ and then draw a sample from the Gaussian density indexed by it. Thus the problem at hand is to draw samples from $p(\boldsymbol{L})$ without computing the normalizing constant Z. While past approaches such as ϵ-exact sampling [4] focused on estimating Z directly, MCMC methods require only the ability to evaluate $p(\boldsymbol{L})$ pointwise up to normalizing constants. As the basis for our work, we briefly review two recently derived MCMC algorithms.

3. EXISTING MCMC ALGORITHMS

3.1. Sequential Gibbs Sampler

Sampling one label at a time conditioned upon the others defines a Gibbs sampler with respect to $p(\boldsymbol{L})$. After the state of the sampler is initialized to $\boldsymbol{L}^0 = \left(l_1^0,\ldots,l_M^0\right)$, a single iteration of the Gibbs sampler consists of M steps, each of which updates a label l_m while leaving the other M-1 labels fixed as follows:

Algorithm 1 Sequential Gibbs Sampler

1. Determine $\bar{\mu}$ and $\bar{\Sigma}$ of the product distribution of $M-1$ Gaussians using (1) and (2):

$$\mathcal{N}(\boldsymbol{x};\bar{\mu},\bar{\Sigma}) \propto \prod_{i\neq m} \mathcal{N}(\boldsymbol{x};\mu_{l_i},\Sigma_{l_i}).$$

2. Fixing $\boldsymbol{x}$, determine the probability of each l_m via:

$$\bar{w}_{l_m} \propto \frac{w_{l_m}\mathcal{N}(\boldsymbol{x};\mu_{l_m},\Sigma_{l_m})\mathcal{N}(\boldsymbol{x};\bar{\mu},\bar{\Sigma})}{\mathcal{N}(\boldsymbol{x};\mu_{\boldsymbol{L}},\Sigma_{\boldsymbol{L}})}.$$

3. Sample a new label l_m according to $p(l_m = l) \propto \bar{w}_l$.

In practice, samples from $p(\boldsymbol{L})$ are taken as the output of multiple short runs of the Gibbs sampler each of length T, and samples from $p(\boldsymbol{x})$ are then drawn from the Gaussians indexed by the sampled labels. The computational complexity of drawing a *single* sample is $O(TMN)$, a significant improvement from $O(N^M)$. However, in cases when $p(\boldsymbol{L})$ is multi-modal, the Gibbs sampler described mixes slowly—thus requiring T to be very large in order to produce samples in this manner.

3.2. Multi-Scale Gibbs Sampler

To speed up the mixing rate of the Gibbs sampler, a multi-scale approach based on KD-Trees (Appendix A) that builds up a sequence of coarser approximate models has been found to be helpful [4]. The coarser the representation, the easier it is for an MCMC sampler to move among modes. Here we detail the multi-scale sampler of [4] built upon Algorithm 1 that is associated with this representation.

Corresponding to each mixture $p_m(\boldsymbol{x})$ in the product $p(\boldsymbol{x})$, we construct a KD-Tree $\mathcal{T}_m$ having $J = \log_2(N) + 1$ levels. This allows us to construct coarser approximations to the overall product mixture by successively substituting corresponding approximations to each of the product terms in turn. Namely, let $j \in \{1,\ldots,J\}$ index resolution, and let $p_m^j(\boldsymbol{x})$ be the m^{th} Gaussian mixture represented by the j^{th} level of its KD-Tree, where j=1 corresponds to the top level (with one node) and j=J corresponds to the bottom level (with N nodes). We now consider a total of J approximations to the product mixture, indexed according to resolution as:

$$p^j(\boldsymbol{x}) \propto \prod_{m=1}^{M} p_m^j(\boldsymbol{x}). \tag{3}$$

Corresponding to each product mixture $p^j(\boldsymbol{x})$ is a distribution on the labels $p^j(\boldsymbol{L})$ indexing the Gaussians in it. Notice that setting j=J yields $p^J(\boldsymbol{x}) \triangleq p(\boldsymbol{x})$ and $p^J(\boldsymbol{L}) \triangleq p(\boldsymbol{L})$, thereby recovering the original problem. Also, the cardinality of the label space increases by a factor of 2 with each increase in resolution; since the distributions $p^j(\boldsymbol{L})$ are then defined on different state spaces, we introduce a superscript $\boldsymbol{L}^j$ in the sequel, indicating $l_m^j \in \mathbb{Z}_{2^{j-1}-1}$.

Algorithm 1 is run for n_j iterations at each resolution j, beginning with level 1 and ascending to level J. To move from level j to j+1, a sample $\boldsymbol{x}$ is drawn conditioned on the current label, and then the labels at the next level are drawn conditioned on $\boldsymbol{x}$ according to the following equations:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x};\mu_{\boldsymbol{L}},\Sigma_{\boldsymbol{L}});\ p(l_m{=}n|\boldsymbol{x}) \propto w_{mn}\mathcal{N}(\boldsymbol{x};\mu_{mn},\Sigma_{mn}). \tag{4}$$

Continuing in this way, the sampler progresses to the finest resolution (j=J) and draws samples from the desired product mixture. When the number of iterations per level is fixed at T, the complexity of drawing one sample is $O(TMN\log N)$.

4. NEW MULTI-SCALE MCMC METHODS

The small increase in complexity associated with the multi-scale sampler described above results in dramatically improved performance; however, once the sampler reaches the bottom level, it once again mixes slowly. Consequently, consecutive samples produced by one long run of the algorithm are highly correlated. However, since the samples are typically used to construct a kernel density estimate of the product mixture [1], it is important to obtain iid samples because dependency among them introduces bias into the bandwidth selection rule. Restarting the sampler alleviates this problem, but introduces another; if we were to keep the number of iterations the

same, random restarts would turn one long run into multiple short runs. On the other hand, the algorithm may require many iterations to converge to stationarity, so long runs are necessary to sample from the correct distribution.

Hence, the design of this multi-scale algorithm presents us with a trade-off between two desired goals: producing iid samples, and sampling from the correct distribution. This trade-off can be avoided, however, if the MCMC sampler can move up and down in the multi-scale representation without being restarted. In this way consecutive samples are less correlated, and the sampler can be run long enough to more closely approach the stationary distribution.

Two algorithms that take advantage of these ideas are simulated and parallel tempering. The key idea behind them is to induce a multi-resolution representation of some target density $\pi(\boldsymbol{x})$ defined on state space Ω in which the resolution parameter is treated as a random variable. Both algorithms work by constructing a multi-scale representation of $\pi(\boldsymbol{x})$ as a sequence $\pi^1(\boldsymbol{x}), \ldots, \pi^{J-1}(\boldsymbol{x})$ of distributions that are successively finer approximations to $\pi^J(\boldsymbol{x}) \triangleq \pi(\boldsymbol{x})$.

4.1. Simulated Tempering Sampler

The simulated tempering algorithm [6] defines an MCMC sampler through a Markov chain on the state space $\Omega \times \{1 \ldots J\}$. The chain can make two moves from state $(\boldsymbol{x}, j)$: One possibility is to leave j fixed and mutate $(\boldsymbol{x}, j)$ to $(\boldsymbol{y}, j)$ using the original MCMC sampler. Alternatively, a (swap) move from $(\boldsymbol{x}, j)$ to $(\boldsymbol{x}, j')$ is attempted whereby $j' = j \pm 1$ with equal probability. The acceptance probability of the move $(\boldsymbol{x}, j)$ to $(\boldsymbol{x}, j')$ is given by

$$\min\left(1, \frac{\pi^{j'}(\boldsymbol{x}) Z_j}{\pi^j(\boldsymbol{x}) Z_{j'}}\right), \tag{5}$$

where Z_j is the normalizing constant of $\pi^j(\boldsymbol{x})$.

To extend simulated tempering to the problem at hand we use KD-Trees rather than the temperature-based scaling that is common in the statistics literature. Let $0 < \tau < 1$ be the probability of a swap move, and suppose the sampler has been run for k iterations and is in the state $(\boldsymbol{L}_k^j, j)$. The next iteration proceeds as follows:

Algorithm 2 Simulated Tempering Sampler

1. With probability τ, perform a mutation move at scale j using Algorithm 1, thereby sampling $(\boldsymbol{L}_{k+1}^j, j)$ from $(\boldsymbol{L}_k^j, j)$.
2. Otherwise perform a swap move:
   (a) Set j'=$j \pm 1$ with probability 1/2. Draw $\boldsymbol{x}$ from the Gaussian indexed by the label $\boldsymbol{L}_k^j$.
   (b) Draw $\boldsymbol{L}_{k+1}^{j'}$ conditioned on $\boldsymbol{x}$ using (4) and move to $(\boldsymbol{L}_{k+1}^{j'}, j')$ with probability given by (5):

$$\min\left(1, \frac{p^{j'}(\boldsymbol{L}_{k+1}^{j'}) q(\boldsymbol{L}_k^j | \boldsymbol{L}_{k+1}^{j'}) Z_j}{p^j(\boldsymbol{L}_k^j) q(\boldsymbol{L}_{k+1}^{j'} | \boldsymbol{L}_k^j) Z_{j'}}\right).$$

Here Z_j is the normalizing constant of $p_j(\boldsymbol{L})$ and τ controls the expected number of iterations at each level. The Hastings correction $q(\boldsymbol{L}_k^j | \boldsymbol{L}_{k+1}^{j'}) / q(\boldsymbol{L}_{k+1}^{j'} | \boldsymbol{L}_k^j)$ is required to preserve detailed balance; however, a computationally expensive integration is required to yield this ratio in closed form. Moreover, the normalizing constants (also unknown) are needed in order to evaluate the acceptance ratio. We overcome both of these difficulties in the next algorithm.

4.2. Parallel Tempering Sampler

The parallel tempering algorithm [7] defines an MCMC sampler on the state space $\Omega^{\otimes J}$ by using J Markov chains in parallel. In the mutation move, all J chains move independently of one another so that the j^{th} sampler updates its state according to Algorithm 1 w.r.t. distribution $\pi^j(\boldsymbol{x})$. During the swap move, a pair of neighboring chains is picked uniformly at random and their samples are swapped. The acceptance probability of a move that swaps the coordinates j and j+1 in the state vector $\boldsymbol{y} = \left(y^1, \ldots, y^J\right)$ is given by:

$$\min\left(1, \frac{\pi^{j+1}(y^j)\pi^j(y^{j+1})}{\pi^{j+1}(y^{j+1})\pi^j(y^j)}\right). \tag{6}$$

The extension to our problem follows that of simulated tempering. Let $0 < \tau < 1$ be the probability of a swap move and suppose the sampler has been run for k iterations and is in the state $(\boldsymbol{L}_k^1, \ldots \boldsymbol{L}_k^J)$. Then the next iteration proceeds as follows:

Algorithm 3 Parallel Tempering Sampler

1. Perform a mutation move with probability τ using Algorithm 1 for all J chains w.r.t. the distributions $p^j(\boldsymbol{L})$ defined in (3).
2. Otherwise perform a swap move:
   (a) Pick a pair of neighboring chains (j, j+1) uniformly at random. Sample $\boldsymbol{x}$ from the Gaussian indexed by $\boldsymbol{L}_k^j$.
   (b) Draw $\boldsymbol{L}_{k+1}^{j+1}$ and $\boldsymbol{L}_{k+1}^j$ conditioned on $\boldsymbol{x}$ using (4), and accept the proposed move with probability given by (6):

$$\min\left(1, \frac{p^{j+1}(\boldsymbol{L}_{k+1}^{j+1}) p^j(\boldsymbol{L}_{k+1}^j)}{p^{j+1}(\boldsymbol{L}_k^{j+1}) p^j(\boldsymbol{L}_k^j)}\right).$$

Notice that the form of the acceptance ratio is simplified, as both the Hastings corrections and the normalizing constants cancel.

5. EXPERIMENTAL RESULTS

We compare multiple short runs of the sequential and multi-scale Gibbs samplers of Section 3 to a single long run of the parallel tempering sampler of Section 4.2 using two univariate examples. In the first experiment, we consider a product of three Gaussian mixtures each with four components, and in the second experiment three mixtures each with eight Gaussians. For each algorithm, 100 samples were drawn from the product mixture and a kernel density estimate $\hat{p}(\boldsymbol{x})$ was constructed using the "rule-of-thumb" method [8]. The accuracy of $\hat{p}(\boldsymbol{x})$, as measured by the integrated square error between it and the exact solution $p(\boldsymbol{x})$, reflects the samplers' performance. We report the (Monte Carlo estimate of) integrated square error between $\hat{p}(\boldsymbol{x})$ and $p(\boldsymbol{x})$ as the number of iterations I per generated sample was varied. The implementation of existing algorithms was validated using the Kernel Density Estimation Toolbox [9].

For each value of I, all parameters were set so that each algorithm was alloted the same total number of iterations. For the multiscale algorithms, the number of iterations T at each of J levels was varied, with $I = TJ$ for each trial. Moreover, since 100 individual (short) runs of the multi-scale Gibbs sampler were taken for every trial, a single (long) run of the parallel tempering sampler was downsampled by a factor of 100 to produce the required samples for comparison. Finally, $I = TJ$ iterations of the sequential Gibbs sampler were then run for each trial. In each experiment, T

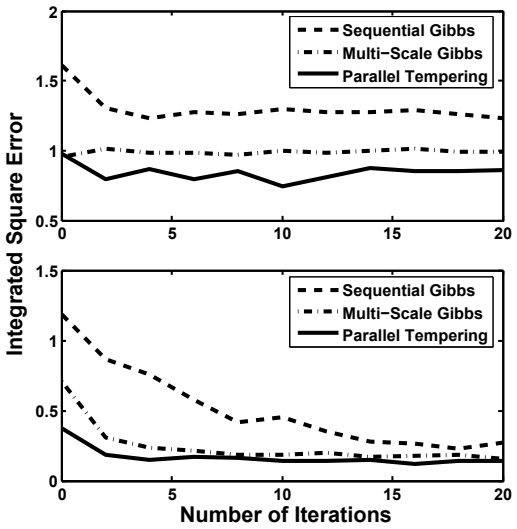

Fig. 1. Performance of three sampling schemes, in which samples are drawn from a product of three mixtures, each having four (Experiment 1, top) or eight (Experiment 2, bottom) components.

was increased successively from one to twenty (further increases of T yielded no improvement in the performance of the multi-scale algorithms), and error estimates were obtained by averaging 50 Monte Carlo runs. The probability of a swap move τ was set to $1/2$, though consistent performance was observed for a broad range of values.

As shown in Figure 1, the parallel tempering sampler consistently achieved a lower error rate than the multi-scale Gibbs sampler. (Both algorithms significantly outperformed the sequential Gibbs sampler, consistent with the results of [4].) These results suggest that enabling the sampler to traverse the multi-scale representation in both directions is helpful. Using one long run to draw samples also provides greater confidence that the algorithm is sampling from the correct distribution. Moreover, these results indicate that the sampler does not have to progress all the way to the coarsest level of the representation in order for consecutive samples to sufficiently decorrelate, thereby increasing computational efficiency.

6. DISCUSSION

Here we have presented two new multi-scale MCMC constructions for sampling from products of Gaussian mixtures. By using one long MCMC run that can traverse a multi-scale representation in the coarse-to-fine as well as fine-to-coarse directions, these constructions simultaneously serve to decrease the dependence among consecutive samples, and to better ensure that these samples are drawn from the correct (stationary) product mixture.

In contrast to previously proposed schemes, the resulting multi-scale methods are full MCMC algorithms whose convergence properties are amenable to analysis using available methods. This will be one direction of our future work. Another promising avenue of future research is to incorporate techniques from other multi-level samplers appearing in the literature on population Monte Carlo methods.

Acknowledgements: The first author is supported by the National Defense Science and Engineering Graduate Fellowship. Both authors would like to thank Alexander Ihler and Jose Blanchet for many insightful discussions of the problem.

7. REFERENCES

[1] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," in *Computer Vision and Pattern Recognition*, 2003.

[2] M. Briers, A. Doucet, and S. S. Singh, "Sequential auxiliary particle belief propagation," in *International Conference on Information Fusion*, 2005.

[3] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, pp. 1771–1800, 2003.

[4] A. T. Ihler, E. B. Sudderth, W. T. Freeman, and A. S. Willsky, "Efficient multiscale sampling from products of Gaussian mixtures," in *Neural Information Processing Systems 17*, 2003.

[5] P. J. Van Laarhoven and E. H. L. Arts, *Simulated Annealing: Theory and Applications*, Reidel, 1987.

[6] C. J. Geyer and E. Thompson, "Annealing Markov chain Monte Carlo with applications to ancestral inference," *Journal of the American Statistical Association*, pp. 909–920, 1995.

[7] C. J. Geyer, "Markov chain Monte Carlo maximum likelihood," in *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, 1991, pp. 156–163.

[8] B. W. Silverman, *Density Estimation*, Chapman & Hall, 1986.

[9] A. T. Ihler and M. Mandel, "Kernel Density Estimation Toolbox for Matlab," http://www.ics.uci.edu/∼ihler/code/kde.tar.gz Version of 08 January, 2004.

[10] A. G. Gray and A. W. Moore, "Very fast multivariate kernel density estimation via computational geometry," *Joint Statistical Meetings*, 2003.

A. DYADIC KD-TREES

A KD-Tree (k-dimensional tree) is a data structure that stores a hierarchical representation of a set of points along with their statistics [10]. It is constructed through a two-pass procedure, by first recursively dividing the point set and then recursively computing statistics associated with each set of points.

The construction of the KD-Tree begins by associating all points in the data set with the root node. The points are then partitioned into two sets about the median of the dimension of the highest variance. These sets are then associated to children of the root node, and the procedure is repeated until the resulting child nodes contain only a single point. The statistics at each node are computed as follows. First, each of the leaf nodes s is assigned a weight w_s, a covariance Λ_s, and a mean μ_s. The statistics of a parent node are then computed as a function of the statistics of its children. We denote the left and right children of node s by s_L and s_R, respectively. Its statistics are then updated according to the following equations:

$$w_s = w_{s_L} + w_{s_R}; \qquad w_s\mu_s = w_{s_L}\mu_{s_L} + w_{s_R}\mu_{s_R}$$
$$w_s(\Lambda_s + \mu_s^2) = w_{s_L}\left(\Lambda_{s_L} + \mu_{s_L}^2\right) + w_{s_R}\left(\Lambda_{s_R} + \mu_{s_R}^2\right).$$

KD-Trees provide a natural multi-scale representation of a single Gaussian mixture, with the point set containing the means of the component Gaussians. The leaf nodes then represent each of the Gaussians in the mixture, and assigned covariances and weights are those of the mixture components.