Monte-Carlo methods for Computer Vision

Alfredo Kalaitzis
School of Informatics
University of Edinburgh

May 22, 2009

1 Introduction

There are several ways that Monte Carlo Methods that have been applied to computer vision problems. We give some examples from:

- sampling
- function integration
- rejection sampling
- particle filtering
- Markov Chain Monte Carlo

2 The *sampling* idea

In figure 1, suppose we are throwing darts to each of the Cartesian systems. The probability of a dart falling anywhere in the red area is easy to compute when we are dealing with analytical shapes such as a rectangle or a circle; but how do we compute the probability of success in the third case?

One approach is the Riemann integration method by separating the whole system into a thin grid and computing the ratio $\frac{RedSquares}{AllSquares}$, from which we get an approximation of the red area.

A problem with this approach is that the **computational cost increases exponentially with the dimensionality of the area**.

3 The Monte-Carlo principle

Estimating the probability of getting a dart in any red area is mathematically equivalent to estimating the integral of that area. The Monte-Carlo *way* of doing the above integration would be to throw an arbitrary number of darts (samples) in the Cartesian system, and while assuming that we make no effort of aiming into the red area[1], we compute the ratio $\frac{DartsInRedArea}{AllDarts}$.

[1] i.e. the samples are **Independently and Identically Distributed** from a 2D uniform distribution

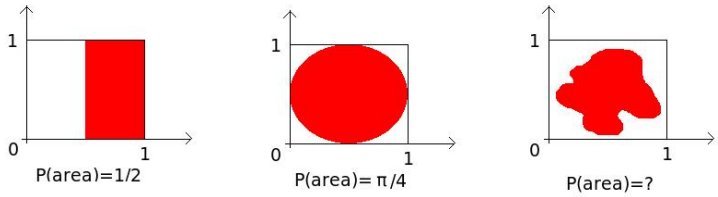

Figure 1: (left) rectangle red area, (mid) red area as an ellipse, (right) a non-analytic red area.

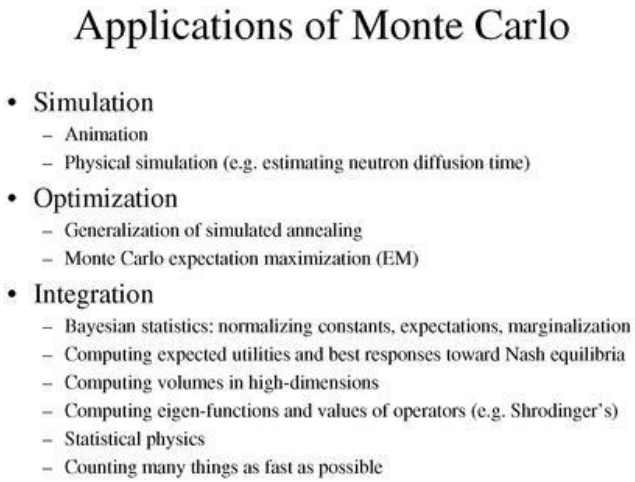

Figure 2: source. $http://www.cs.ubc.ca/\sim nando/papers/mlintro.pdf$

4 The role of *sampling* in Computer Vision

We will be concerned with the case of integration. Many advanced computer vision methods perform *probabilistic inference*, which is basically the iterative application of the two fundamental rules of probability theory, the *sum-rule* and the *product-rule*.

$$sumrule : P(A) = \sum_{B} P(A, B) \tag{1}$$

$$productrule : P(A, B) = P(A|B)P(B) = P(B|A)P(A) \tag{2}$$

In typical inference problems these rules are combined with *assumptions of conditional independence* between *random variables*, to *infer* the most probable class of an object in an image, or the most probable position of an object when tracking its position thought time. In typical real-world applications, we dont get to pick one out of a discrete set of probable states but instead,

out of an infinite amount of probable state in a continuous state-space; hence **the estimation of integrals is one of the biggest issues in Bayesian inference**.

One way to deal with this is through sampling. In figure 1, as the number of samples reaches infinity, our estimation converges to the true value of the red area. Note that the computational cost of this approach does not depend on the dimensionality of the area. This simplified version of MC integration is also the most common application of the MC principle.

5 Monte Carlo integration

Suppose we wish to estimate the expectation (mean) of a function $f(x)$ (e.g $f(x) = x$, $f(x) = x - \mu$ (variance)),

$$\hat{\mu_f} = \int f(x)P(x|D) \tag{3}$$

and that we already know the posterior distribution $P(x|D)$ (fig 3). We

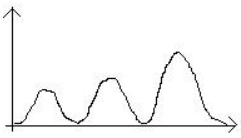

Figure 3: Example of a posterior probability distribution

generate many samples $x^i|_1^N$ from $P(x|D)$ and we bin them to infinitesimal intervals dx. So instead of $P(x|D)$, we get to estimate its approximation,

$$P(dx|D) = \frac{1}{N}\sum_{i=1}^{N}\delta_{x^i}dx \tag{4}$$

By substitution of (4) into (3) we have,

$$\mu_f = \int f(x)\sum_{i=1}^{N}\delta_{x^i}dx = \frac{1}{N}\sum_{i=1}^{N}f(x^i) \tag{5}$$

which is intuitive too, as it says that we approximate the expectation of $f(x)$ at N randomly generated x^i, according to $P(x|D)$, as in figure 4. In fact, the problem is that **we dont know how to sample from** $P(x|D)$[2].

6 Rejection Samping [2]

In terms of application of all the methods we are going to describe from now on, suppose that the distribution we want to approximate, represents the x positions

[2]This doesn't mean that we can't measure $P(x|D)$ at a specific x^i. A practical consequence of the problem is that we don't know how to make a random number generator that generates samples x^i which represent the distribution $P(x|D)$.

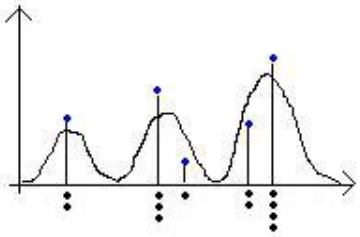

Figure 4: Function value (blue dots) is proportional to the number of samples (black dots) at that interval.

of multiple objects in a particular frame of a video. We wish to sample from an unknown distribution $P(x)$.

One approach is to sample x from a known distribution, $x \sim Q(x)$ which is normally-distributed (Gaussian), and choose a scalar M. Also, we sample $u \sim U_{(0,1)}$ from a uniform distribution 0 to 1. If

$$uMQ(x^i) > P(x^i) \tag{6}$$

then we reject x^i as an invalid sample from $P(x)$ (fig 5).

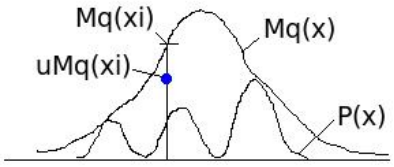

Figure 5: The proposal $Q(x)$ covers appropriately the true distribution $P(x)$.

The problem is now shifted to choosing an appropriate $MQ(x)$. If it is too large, we will waste too many samples. If it is too small, we will gather inappropriate samples. High dimensional distributions are tricky too cover because most of their probability mass is concentrated on low density areas[3]. Also, outliers create heavy-tailed distributions, which are difficult to sample from. As a result, we would have to raise M significantly, but that would cause to waste many samples. Hence, in high dimensions and with lots of outliers, this method is ineffective. *Importance sampling* is an alternative to rejection sampling, and is used more frequently when dealing with a few extra dimensions, but is also ineffective in high dimensions. MCMC methods (Metropolis, Hamiltonian) are more suitable for large dimensionalities.

7 Particle Filtering [3][4][5][6]

Particle filtering is used to track the *movement* of distributions, which - as we mentioned earlier - represent tracked positions. We start with samples from an initial distribution (see fig 6, first layer of blue dots):

[3]i.e. they form a shell of low density around a small area of high density.

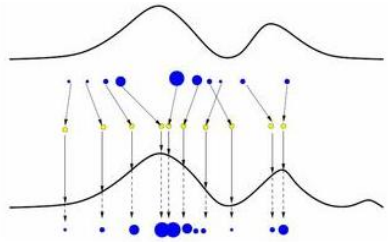

Figure 6: source. http://www.cs.ubc.ca/∼nando/papers/mlintro.pdf

We weight the samples (visualised by size of dot) with importance sampling. Then we propagate them to the next state of the distribution and re-weight them. As the distribution changes over time, we are able to track the distribution by moving the particles through the dynamics of the distribution change, re-weighting with importance sampling, and then selecting the fittest by *re-sampling*. Re-sampling is the process of taking the *cumulative* of the approximated distribution so far and projecting onto it a uniformly random number from 0 to 1. Since the approximated cumulative distribution has the shape of a *rising staircase*, the random number is more likely to *hit* a specific *step* corresponding to the most probable sample from P(x). Hence, only the fittest samples survive.

In the case where the object moves very fast, so does the distribution; and we still want to maintain the mode of the distribution at the object's true location. To avoid ending up with a lot of samples with very small weights (as in fig 7), we keep only the fittest and re-weight them. As a result we get a new set of weights and we are now able to track the distribution/position of the fast object.

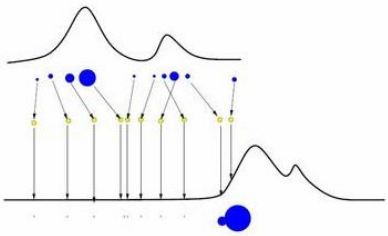

Figure 7: source. http://www.cs.ubc.ca/∼nando/papers/mlintro.pdf

8 Markov Chain Monte Carlo [1][7]

We start with the inverse problem of MCMC: Suppose we have a Markov chain of 3 discrete states x1,x2,x3 (see fig 8):

the transition matrix (or stochastic matrix) of this Markov chain is

$$T = P(x^t|x^{t-1}) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0.1 & 0.9 \\ 0.6 & 0.4 & 0 \end{pmatrix} \tag{7}$$

A property of a Markov chain is that the probability of having a specific state depends only on the previous state and not on the whole series of states in

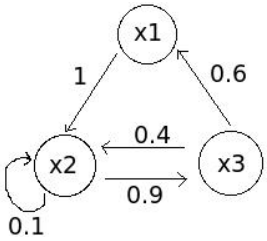

Figure 8: Example of a Markov chain as a Finite State Automaton with 3 discrete states. Labels on edges are transition probabilities.

the past. Our goal is to learn the prior of being in each node, or the *equilibrium/stationary/invariant matrix*. If the graph is aperiodic and irreducible then for any vector of probabilities v:

$$v^T T^t \rightarrow \pi^T, \ as \ t \rightarrow \infty \tag{8}$$

In other words, if we multiply T to v^T t times, we will always end up with π^T as t goes to infinity. π^T is called the *invariant* or *stationary* distribution of the Markov chain and it contains the prior probabilities of being in each state. In the chain is reducible, then we might never be able to visit some nodes. If it is periodic, it will have oscillations and we might never be able to converge towards the invariant distribution. For information of what a irreducible and aperiodical Markov chain is, see $http://en.wikipedia.org/wiki/Markov\_chain$.

MCMC works by doing the inverse of what we have done so far: Now π^T is the distribution from which we want to generate samples, and we construct a transition matrix T such that when multiplied to a vector, it gives samples from π^T. In the case of discrete states, (8) is be written as

$$\sum_{i=1}^{N} \pi_i T_{ij} = \pi_i \tag{9}$$

and in the case of continuous state-spaces, we have

$$\int \pi(x) P(y|x) dx = \pi(x) \tag{10}$$

where $P(y|x)$ is the *Markov chain kernel*.

9 MCMC: the Metropolis-Hastings algorithm [8]

How do we construct a good matrix T? The algorithm that builds T goes as follows in pseudo-code:

- for i = 0 to N-1:
- sample $u \sim U_{(0,1)}$ //u is uniformly generated from 0 to 1
- $x^* \sim Q(x^*|x^i)$ // x^* is generated from a proposal distribution $Q(x^*|x^i)$
- if $u < A(x^i, x^*) = min(1, \frac{P(x^*)}{P(x^i)} \frac{Q(x^i|x^*)}{Q(x^*|x^i)})$ //A is the acceptance function which outputs 0 to 1

- $x^{i+1} = x^*$
- else
- $x^{i+1} = x^i$

$\frac{P(x^*)}{P(x^i)}$ shows which x is better, but we have to normalise by the proposals $\frac{Q(x^i|x^*)}{Q(x^*|x^i)}$). This normalisation also ensures that the *detailed balance* condition is satisfied. The *detailed balance* says that

$$\pi(x^t)P(x^{t+1}|x^t) = \pi(x^{t+1})P(x^t|x^{t+1}) \Rightarrow \int \pi(x^t)P(x^{t+1}|x^t) = \pi(x^{t+1}) \quad (11)$$

which looks like (10). The *detailed balance* condition basically shows that the Markov chain is indeed aperiodical so far, hence now the kernel can be constructed in order to get samples from π^T. As in particle filtering, the problem is again shifted to choosing a good proposal distribution Q.

While choosing a Q one should be aware of the following dangers: if the Markov chain starts on one mode of the distribution and Q does not allow jumps to other modes, then the approximation will be that of one mode. If Q is too broad, we will waste a lot of samples.

The kernel is built as follows:

$$K(x, B) = \begin{cases} Q(B|x)A(x, B), \ x \notin B \\ 1 - \int Q(x'|x)A(x, x'), \ x \in B, \ x' \in \{X \setminus B\} \end{cases} \quad (12)$$

where B is a set of states and x' is any other state except the ones in B. It says: if B is accepted, we move from x to B with probability $Q(B|x)$ multiplied by the acceptance of B. If B is rejected, we remain in x (which is in B) with probability 1 minus the integral of all probabilities $Q(x'|x)$ multiplied by the acceptance of x'. In the case of discrete state-spaces, the kernel is replaced by a matrix T and the integral is replaced by a *sum*.

References

[1] F.Dellaert, S.Seitz, S.Thrun, and C.Thorpe. Feature correspondence: A Markov chain Monte Carlo approach. In Advances in Neural Information Processing Systems 13, pages 852-858, 2000.

[2] Robert, C.P. and Casella, G. "Monte Carlo Statistical Methods" (second edition). New York: Springer-Verlag, 2004.

[3] Ristic, B.; Arulampalam, S.; Gordon, N. (2004). Beyond the Kalman Filter: Particle Filters for Tracking Applications. Artech House.

[4] Doucet, A.; Johansen, A.M.; (December 2008). "A tutorial on particle filtering and smoothing: fifteen years later". Technical report, Department of Statistics, University of British Columbia. http://www.cs.ubc.ca/

[5] Doucet, A.; Godsill, S.; Andrieu, C.; (2000). "On Sequential Monte Carlo Methods for Bayesian Filtering". Statistics and Computing 10 (3): 197-208. http://www.springerlink.com/content/q6452k2x37357l3r/.

[6] Arulampalam, M.S.; Maskell, S.; Gordon, N.; Clapp, T.; (2002). "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking". IEEE Transactions on Signal Processing 50 (2): 174188

[7] Christophe Andrieu et al, "An Introduction to MCMC for Machine Learning", 2003

[8] Siddhartha Chib and Edward Greenberg: "Understanding the MetropolisHastings Algorithm". American Statistician, 49(4), 327335, 1995

Online :
http : *//en.wikipedia.org/wiki/Sampling*(*statistics*)
http : *//phys.ubbcluj.ro/ zneda/edu/mc/mcshort.pdf*
http : *//www.cs.ubc.ca/ nando/papers/mlintro.pdf*
http : *//homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SENEGAS/node*2.*html*
http : *//homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/SENEGAS/node*4.*html*

Prepared in LaTeX 2_ε by Alfredo Kalaitzis