– Appeared in NNSP 1999 –

A CONTROL THEORY FORMULATION FOR RANDOM VARIATE GENERATION

Malik Magdon-Ismail
malik@work.caltech.edu
Caltech Learning Systems Group
Department of Electrical Engineering
California Institute of Technology
136-93 Pasadena, CA, 91125

Amir F. Atiya
amir@work.caltech.edu
Caltech Learning Systems Group
Department of Electrical Engineering
California Institute of Technology
136-93 Pasadena, CA, 91125

Abstract. The need to simulate complex systems in a Monte Carlo manner necessitates efficient methods for generating random variates. In this paper we propose a new method for random variate generation. The method is based on a contol-theory formulation. We use a cascade structure consisting of a neural network "controller" and a density estimator ("plant"). The neural network "controller" acts as a density shaper, and is trained until the density of its output (as measured by the density estimator) is as close as possible to the given density. Once training is complete in the design phase, the generation of random numbers can be performed in a very fast manner.

INTRODUCTION

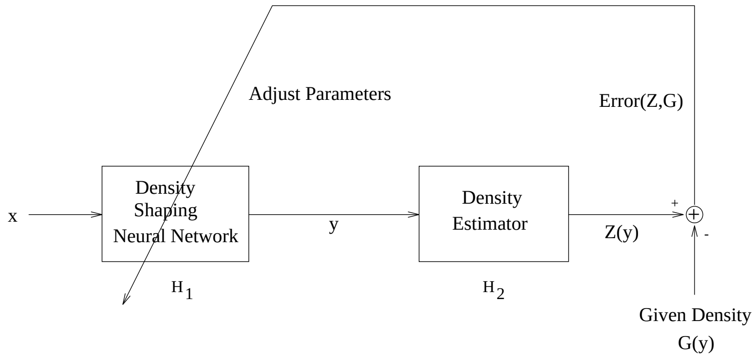
In many scientific problems in areas such as physics, biology, engineering and economics, it is now more the rule than the exception to be confronted with problems where analytic solutions are not possible and simulating the given problem would be the only feasible approach. Problems which possess some random element will necessitate an accurate and fast random variable generation procedure for the Monte Carlo simulation to be efficiently performed. Examples of applications include the simulation of a biological system, such as how nerve cells interact together, the simulation of a communication network where the packets are assumed to arrive according to a particular density function, the simulation of a control system where the plant disturbance is a random variable, and the simulation of the stock market in an attempt to estimate a valuation for some derivative instruments. In the past, the assumption of standard density functions such as Gaussian or exponential densities for these random variables often allowed analytical solutions for these problems. But now, with more accuracy needed, the next step has been to assume more realistic density functions for the random variables (possibly estimated from the data). Unfortunately, there are very few techniques that can generate a random variable possessing an arbitrary density function (see [6] for a review of the approaches). For example the transformation method [2], [3] cannot generate from an arbitrary density (it can generate only those densities where the inverse of the distribution function can be obtained analytically). The other well-known method, the rejection method [6], is more general. The problem, however, is that it can be very slow for large-tailed densities and for densities given by a sample drawn from it rather than an explicit formula.

In this paper we propose a method that formulates the problem using a "control theory" framework. The proposed idea can be implemented using multilayer neural networks. We propose a cascade structure that consists of a density shaping network (acting as the "controller"), and a density estimation network (acting as the "plant"). The controller is trained until the output of the "plant" (i.e. the resulting density) is as close as possible to the given density. For the method that we propose, the bulk of the time is spent in "learning" to generate the random variates. Once this learning is done, the actual generation of the random numbers is fast and can thus be used for Monte Carlo simulations.

CONTROL FORMULATION OF RANDOM VARIATE GENERATION

Suppose that we wish to generate random variates from a specified density $g(x)$. We propose here a method that is based on transforming a known density to the given density. Figure 1 illustrates the basic idea of the method. The structure consists of two cascaded units. A random variable x is generated according to any standard density function, for example a Gaussian or a uniform density. In principle, an unlimited number of x's can be generated. This random variable is the input to the composite structure. The first network acts as a nonlinear transformation that shapes the density of x. Let the distribution function of the transformed variable y be $Z(y)$. The second unit serves to estimate this distribution function. For example, one could use a standard kernel density estimator , [7], for this unit, i.e.

$$z(y) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\Big(\frac{y - y_n}{h}\Big) \tag{1}$$



The first network shapes the density of x. The transformed density is estimated, and then compared with the true density. The parameters of the first network are altered until the transformed density matches the required density.

Figure 1: Control formulation for random variate generation.

where K is the kernel, h is the kernel width, d is the dimension of y and y_i are the N samples of y. Alternatively, we can use the neural network density estimator developed in [4] as the second unit (see Appendix for a short description of this density estimation method).

Thus, the output of the second unit is the estimate of the distribution function $Z(y)$ of network 1's output. Let the desired distribution function be $G(y)$. An error signal, $\mathcal{E}(G(y), Z_{w_1}(y))$ can now be generated where it can be seen that the distribution Z is a function of w_1, the weights of the first network. This error signal can be backpropagated to adjust the weights of network 1 so as to minimize $\mathcal{E}(G(y), Z_{w_1}(y))$. Such a training process would change network 1's mapping in such a way that would shape the distribution function of y to get it as close as possible to the desired distribution function. Once training is complete, we can generate numbers according to the distribution function $G(x)$, by generating x according to the standard density, and then passing it through network 1.

Note 1: This algorithm is not restricted to the case of scalar random variable, and so can be used to generate multi-dimensional variables.

Note 2: To generate from a density represented only by a set of data points, we use the same method above but replace $G(y)$ by $\hat{G}(y)$, an estimate of $G(y)$ using the data points. This estimate can be obtained using one of the standard density estimation methods, [4] [7].

Note 3: The entire learning process can be an involved process. However, once the learning has been done, the generation of random variates is fast, since one needs only to generate a uniform or a Gaussian density, and then pass it through a neural network.

The analogy with the control problem is as follows. The second unit represents the "plant," and network 1 is the "controller." The controller is trained, such that the plant produces the desired output[1]. The details of the algorithm are as follows:

I: Initialization:

1. Let the desired density be $g(y)$ and the corresponding distribution function $G(y)$. Generate N samples $x_1, \cdots, x_N \in \mathbf{R}^d$ according to any standard density such as a Gaussian density. The larger N is, the more accurate the final result will be.

2. Let $H_1(x, w)$ and $H_2(x, v)$ be the functions implemented by network 1 and network 2, where w and v represent parameters[2]. Network 1 has d outputs, and network 2 has one output. Initialize the weights of the networks.

3. Calculate network 1's output: $y_n(w) = H_1(x_n, w),\ n = 1, \cdots, N$.

II: Training Iterations:

4. Train network 2 (or construct the kernel estimator) to estimate the distribution of the y_n's. Thus, $H_2(y, v(w))$ is implementing the distribution function of y, where we have explicitly shown the w dependence.

5. Suppose the error function we wish to minimize is $\mathcal{E}(H_2(w), G)$. Possible error functions are an L_1 or L_2 norm, or a cross entropy (Kullback-Leibler distance). For the case of squared error, let $\{z_n\}_{n=1}^{M}$ be any set of points for calculating the error function ($\mathcal{E}$),

$$\mathcal{E} = \sum_{n=1}^{M} \Big[H_2(z_n, v(w)) - G(z_n) \Big]^2 \tag{2}$$

6. One alters the weights (w) of the first network in the direction of the negative gradient:

$$w(t+1) = w(t) - \eta \frac{\partial \mathcal{E}}{\partial w} \tag{3}$$

---

[1]This technique has a similar form to a neural network control structure (see [5]) where, typically, a neural network is trained to identify the plant (estimate the plant output), analogous in our set up to the density estimation unit. Then, a neural network controller is trained to control the plant – i.e., the identification network estimates the plant output and the controller network is altered until this output is as desired.

[2]In the case where network 2 is a kernel density estimator, the parameters v are the outputs of network 1 on the samples $x_1, \ldots, x_2$

For example in the case of squared error we have,

$$\frac{\partial \mathcal{E}}{\partial w} = 2\sum_{n=1}^{M}\Big[H_2(z_n, v(w)) - G(z_n)\Big]\frac{\partial H_2(z_n, v(w))}{\partial w} \tag{4}$$

The weights in the first network can now be updated using a standard standard backpropagation scheme [1, pg 115], where one backpropagates the δ's first through network 2, then through network 1 to adjust the weights of network 1.

7. Go to step 4 to perform another iteration of training, and continue doing so until the error becomes small enough.

8. After training is complete, the random numbers can be generated by generating x according to the standard density that was innitially used (e.g. Gaussian), and computing $y = H(x, w^*)$ (where w^* represents the final weights after the entire learning process). Now, y should be the distributed according to $g(y)$.

APPENDIX: DESCRIPTION OF A DENSITY ESTIMATION NETWORK

We mentioned in Section 2 that the second unit in the proposed structure can be implemented using the neural network density estimation procedure proposed in [4]. In this appendix we will briefly describe this method.

Let $x_n \in \mathbf{R}$, $n = 1, ..., N$ be the data points. Let the underlying density be $g(x)$ and its distribution function $G(x) = \int_{-\infty}^{x} g(t)dt$. Let the neural network output be $H(x, w)$, where w represents the set of weights of the network. Ideally, after training the neural network, we would like to have $H(x, w) = G(x)$. It can easily be shown that the density of the random variable $G(x)$ (x being generated according to $g(x)$) is uniform in $[0, 1]$. Thus, if $H(x, w)$ is to be as close as possible to $G(x)$, then the network output should have a density that is close to uniform in $[0, 1]$. This is what our goal will be. We will attempt to train the network such that its output density is uniform, then the network mapping should represent the distribution function $G(x)$. In [4] we have developed two algorithms to achieve that:

SLC (Stochastic Learning of the Cumulative)

In this method we use the N data points as inputs to the network. For every training cycle, we generate a different set of N network targets randomly from a uniform distribution in $[0, 1]$, and adjust the weights to map the data points (sorted in ascending order) to these generated targets (also sorted in ascending order). Thus we are training the network to map the data to a uniform distribution. This method applies only to the one-dimensional case.

It can be proved that this stochastic algorithm converges to a minimum of the sum of square error between the given distribution function $G(x)$ and the network output $H(x, w)$.

SIC (Smooth Interpolation of the Cumulative)

This method applies to the multi-dimensional case. Again, we have a multilayer network, to which we input the point $\mathbf{x}$, and the network outputs the estimate of the distribution function. Let $g(\mathbf{x})$ be the true density function, and let $G(\mathbf{x})$ be the corresponding distribution function. Let $\mathbf{x} = (x^1, ..., x^d)^T$. The distribution function is given by

$$G(\mathbf{x}) = \int_{-\infty}^{x^1} \cdots \int_{-\infty}^{x^d} g(\mathbf{x}) dx^1 \cdots x^d, \tag{5}$$

a straightforward estimate of $G(\mathbf{x})$ could be the fraction of data points falling in the area of integration:

$$\hat{G}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \Theta(\mathbf{x} - \mathbf{x}_n), \tag{6}$$

where Θ is defined as

$$\Theta(\mathbf{x}) = \begin{cases} 1 & \text{if } x^i \geq 0 \text{ for all } i = 1, \ldots, d, \\ 0 & \text{otherwise.} \end{cases}$$

The method we propose uses such an estimate for the target outputs of the neural network. The estimate given by (6) is discontinuous. The neural network method developed here provides a smooth, and hence more realistic estimate of the distribution function. The density can be obtained by differentiating the output of the network with respect to its inputs.

ACKNOWLEDGMENTS

We would like to acknowledge Yaser Abu-Mostafa and the Caltech Learning Systems Group for their useful input. We would like to acknowledge the support of NSF's Engineering Research Center at Caltech.

REFERENCES

[1] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[2] G. Johnson. Construction of particular random processes. In *Proceedings IEEE*, volume 82(2), pages 270–281, February 1994.

[3] D. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, 1981.

[4] M. Magdon-Ismail and A. Atiya. Neural networks for density estimation. *Advances in Neural Information Processing Systems (NIPS 98), to appear*, 1998.

[5] K. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[6] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes*. Cambridge University Press, Cambridge, UK, 1988.

[7] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, UK, 1993.