Scalable, Probabilistic Simulation in a Distributed Design Environment

by

Wei Mao

Bachelor of Science in Precision Instruments and Mechanology
Tsinghua University, China 1999
Master of Science in Precision Instruments and Mechanology
Tsinghua University, China 2002

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Signature of Author.....
Department of Mechanical Engineering
May 21, 2008

Certified by.....
David R. Wallace
Professor of Mechanical Engineering
Thesis Supervisor

Accepted by.....
Lallit Anand
Chairman, Department Committee on Graduate Students

Scalable, Probabilistic Simulation in a Distributed Design Environment

by
Wei Mao

Submitted to the Department of Mechanical Engineering
on May 21, 2008, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Mechanical Engineering

Abstract

Integrated simulations have been used to predict and analyze the integrated behavior of large, complex product and technology systems throughout their design cycles. During the process of integration, uncertainties arise from many sources, such as material properties, manufacturing variations, inaccuracy of models and so on. Concerns about uncertainty and robustness in large-scale integrated design can be significant, especially under the situations where the system performance is sensitive to the variations. Probabilistic simulation can be an important tool to enable uncertainty analysis, sensitivity analysis, risk assessment and reliability-based design in integrated simulation environments.

Monte Carlo methods have been widely used to resolve probabilistic simulation problems. To achieve desired estimation accuracy, typically a large number of samples are needed. However, large integrated simulation systems are often computationally heavy and time-consuming due to their complexity and large scale, making the conventional Monte Carlo approach computationally prohibitive. This work focuses on developing an efficient and scalable approach for probabilistic simulations in integrated simulation environments.

A predictive machine learning and statistical approach is proposed in this thesis. Using random sampling of the system input distributions and running the integrated simulation for each input state, a random sample of limited size can be attained for each system output. Based on this limited output sample, a multilayer, feed-forward neural network is constructed as an estimator for the underlying cumulative distribution function. A mathematical model for the cumulative probability distribution function is then derived and used to estimate the underlying probability density function using differentiation.

Statistically processing the sample used by the neural network is important so as to provide a good training set to the neural network estimator. Combining the statistical information from the empirical output distribution and the kernel estimation, a training set containing as much information about the underlying distribution as possible is attained. A back-propagation algorithm using adaptive learning rates is implemented to train the neural network estimator. To incorporate a required cumulative probability distribution function monotonicity hint into the learning process, a novel hint-reinforced back-propagation approach is created. The neural network estimator trained by empirical and kernel information (NN-EK estimator) can then finally be attained.

To further improve the estimation, the statistical method of bootstrap aggregating (Bagging) is used. Multiple versions of the estimator are generated using bootstrap resampling and are aggregated to improve the estimator. A prototype implementation of the proposed approach is developed and test results on different models show its advantage over the conventional Monte Carlo approach in reducing the time by tens of times to achieve the same level of estimation accuracy.

Thesis Supervisor: David R. Wallace, Professor of Mechanical Engineering
Committee Members: David C. Gossard, Professor of Mechanical Engineering
Dan Frey, Professor of Mechanical Engineering

Acknowledgments

First of all, I am deeply grateful to my advisor, Professor David Wallace. Without his consistent support, guidance and patience during the past five years, it is impossible for me to have this thesis done. He has been always willing to do whatever he could to help me out. I really appreciate what he has done for me and what I have benefited and learned from him during the years.

I would also like to thank my committee members, Professor David Gossard and Professor Dan Frey, for giving me valuable feedback on my research. They were always there whenever I was in need of a help.

Many thanks to my CADLAB fellows: Qing Cao, Sangmok Han, Sittha Sukkasi, Elaine Yang, Mika, Amy Banzaert, Iason Chatzakis, Barry Kudrowitz, Monica Rush, James Penn and many others. They make CADLAB like a big family.

Thank all my friends at MIT who made my Ph.D. journey easier. Especially I would like to thank Liang Liu and Wei Wang for standing next to me when I was in need of their helps. Thank my basketball buddies in Chinese Basketball Club for making my life at MIT more enjoyable.

Finally, I would like to thank my family, my wife Fang Li, my mother Changxiang Fang, my father Fachun Mao, and my brother Jian Fang. It is their constant support and encouragement that enabled me to go through this long journey.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

1.1.1 Integrated Simulation Environment

Current design of a large product or technology system has evolved to the processes which cross various domains, and require many groups with diverse expertise in different locations. Figure 1-1 is showing such a scenario.

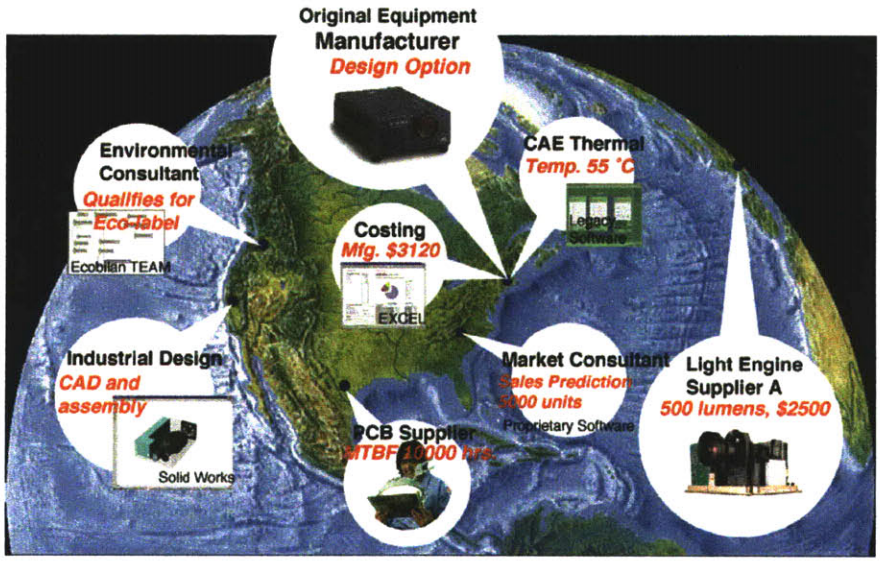


Figure 1-1 Design of a large product or technology system

The benefits of predicting and analyzing the integrated behavior of product and technology systems throughout their definition cycle can't be over-emphasized. However,

it is often found impossible for people to capture the whole structure of the objective system due to the complexity and evolutionary nature of the system, and due to the bounded capability of human intelligence. Also, the analysis is required to be flexible and respond fast to design modifications, error corrections and market changes. However, it is estimated that 30% or more of a design engineer's time is spent in meetings to gather or exchange information about interrelated aspects of a product(Christian 1996), yet researchers at Ford Motor Company estimate they spend hundreds of million dollars per year on integration rework after building complete prototypes(Wallace). In contrast, studies at Ford (Abrahamson 2000) and at Polaroid (Abrahamson 1999) show that integrated assessments requiring weeks or months in a traditional design environment can be understood in seconds using integrated simulations. Integrated simulations finally come into the place to analyze the behavior of large product and technology systems. As such, a number of researchers and companies have attempted to develop integrated simulation environments(Toye 1994; Molina 1995; Bliznakov 1996; Case 1996; Cutkosky 1996; Dabke 1998; Kim 1998), some of which are now used in practice. In order to manage the complexity of building simulations for large, complex systems, an emergent and decentralized model integration approach has also been developed(Pahng 1997; Pahng 1998; Wallace 2000; Senin 2003). This approach helps to mitigate integration difficulties due to scale, complexity, rate-of-change, heterogeneity and proprietary barriers.

1.1.2 Uncertainties in Integrated Design

Any complex product or technology system is not deterministic. Uncertainties arise from many aspects during an integrated design process. Some general sources contribute the uncertainties:

1) Physical dimensions of parts
2) Material properties
3) Error of measurements
4) Manufacturing variations
5) Environmental or operating conditions
6) Market variations
7) Model accuracy and so on

Current integrated simulation environments are working well with deterministic simulations which don't take any uncertainty into consider and don't have the ability to handle any uncertainty. However, when integrating multidisciplinary subsystems to represent large, complex products, concerns about uncertainty and robustness can be very significant. In such an integrated scenario, uncertainties in one discipline will propagate to another discipline through the linked variables. As a result, many variables which are deterministic in their own disciplines may become probabilistic ones, and the final output form the integrated systems will be probabilistic. Figure 1-2 is showing such a scenario.

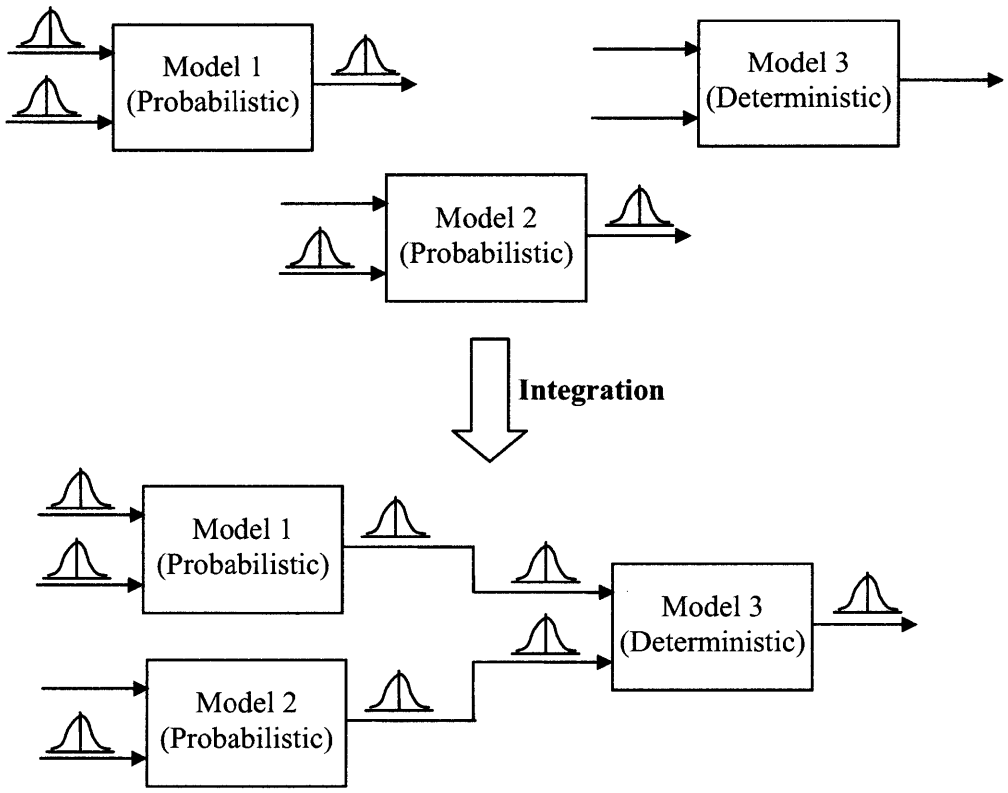


Figure 1-2 Uncertainty propagation in integration

1.2 Research Motivation

To do an integrated design under uncertainties, people have to consider the impact of variations on the performance of the product or technology system to be designed, especially under the situations that the system performance is sensitive to the variations. Probabilistic simulation ability is the premise to enable uncertainty analysis, sensitivity analysis, risk assessment, reliability-based design and robust design in an integrated simulation environment. There is a widely recognized need to incorporate probabilistic modeling and simulation within integrated design processes(Haugen 1980; Siddall 1983; Kowal 1998). My research motivation is to develop a generic probabilistic simulation

approach for integrated simulation environments, including the emergent and decentralized integration environment.

Integrated simulation systems are often complex and large-scale since they are involved with various domains and disciplines. As a result, they are usually computationally heavy and time-consuming. Especially in an integrated simulation environment which is emergent and decentralized, the simulation system can be easily growing whenever a new model is emergent. On the other hand, most probabilistic simulation approaches are sample-based. If a large number of samples are needed for the approach, it will be computationally prohibitive. My research focuses on developing an efficient and scalable approach for probabilistic simulations in integrated simulation environments.

1.3 Thesis Outline

Chapter 1 provides an introduction to the context and the importance of probabilistic simulation ability for integrated design environments.

Chapter 2 introduces the background on probabilistic simulations, including the definition and common steps of a probabilistic simulation, representation of uncertainties in probabilistic simulations, the conventional Monte Carlo method and its limitation in integrated simulations.

Chapter 3 describes a new machine learning approach for probabilistic simulations in an integrated simulation environment. The machine learning based density estimation is discussed, compared with the traditional parametric and nonparametric density estimation methods in the field of statistics.

Chapter 4 is dedicated to discuss how to construct the neural network density estimator, including building an appropriate architecture and choosing a suitable activation function. It also covers creating the mathematical models for the neural network CDF estimator and PDF estimator.

Chapter 5 discusses how to learn from sample for our neural network estimator. The back-propagation learning and its implementation on our neural network estimator are discussed, as well as how to process the random sample by statistical means in order to get a learning set containing as much statistical information as possible. Two versions of neural network estimators, the NN-E estimator and the NN-EK estimator, are described.

Chapter 6 discusses how our neural network estimator can learn from hints, including how the monotonicity hint can be incorporated into the back-propagation learning and how a hint-reinforced training set can be created.

In Chapter 7, the statistical method Bagging is discussed. It covers how to generate multiple versions of our neural network estimator by bootstrap method and aggregate them to gain a further improved estimator.

Chapter 8 gives some case studies to evaluate the proposed approach. Concluding remarks is given in Chapter 9.

Chapter 2

Probabilistic Simulation

2.1 Steps of Probabilistic Simulation

In a deterministic simulation, the input parameters for the simulation system are represented by deterministic values. The simulation is to analyze and predict the behavior of the integrated system by attaining the deterministic output values, i.e., by propagation of deterministic values through the whole system. However, a probabilistic simulation is to predict the system behavior under uncertainties by capturing probabilistic characteristics of final outputs of the system which are caused by uncertainties of the system inputs. In a probabilistic simulation, a lot of efforts are put on analyzing how the system propagates the uncertainties or variations and what is the risk or variation of the predicted system performance. Usually a probabilistic simulation is composed of the following steps:

1) Estimate and quantitatively represent uncertainties in system inputs, i.e. design parameters or variables. This is also called characterization of input uncertainties(Isukapalli 1999).
2) Propagate input uncertainties through the whole integrated simulation system by using an appropriate and feasible method.

3) Quantitatively represent accumulative uncertainties in system outputs which are caused by individual input uncertainties. This also called characterization of output uncertainties.

2.2 Representation of Uncertainties

In simulation-based environment, uncertainties are usually characterized by *probabilities*. Probability can be defined as "frequency of occurrence" of an *event*, or in another word, it can be considered as a numerical measure of the likelihood of occurrence of an event relative to all possible events(Ang 1975).

For a design parameter or variable with uncertainty, its value can be any one in some certain range with randomness. That is to say, the actual outcome is unpredictable to some extent. For example, the dimension of a part from a manufacturing process is always a range of values due to manufacturing variations. All possible values can be looked as a *sample space*. Each individual value is called a *sample point.* An event then is composed of one or more than one sample points within the sample space. In the terminology of set theory, the sample space is a set including all possibilities and an event is a subset of the sample space. Actually the sample space itself is an event called *certain event*. A sample space may be *discrete* or *continuous*. In a discrete sample space, there are countable sample points whose number can be either *finite* or *infinite*. In a continuous sample space, the number of sample points is always infinite. In simulation-based environments, most of sample spaces are continuous.

Probabilities are always associated with specific events. Each event has a probability to happen. Different events may have different probabilities. That means some events may occur more frequently than others.

A system input or output with uncertainty can be represented as a *random variable* since most of the uncertainties are caused by the natural processes or phenomena which are inherently random. For those caused by lack of knowledge about the process or phenomenon, it's also reasonable to treat it as a random variable because the outcome is unpredictable and random under existing knowledge. Random variables make the design parameters or variables with uncertainties to be represented more quantitatively by assigning each possible event a probability. A random variable can be represented either by a probability distribution or by probabilistic characteristics.

2.2.1 Probability Distribution

Different values (or value ranges) of a random variable correspond to different events. Then there is a mapping between the value of a random variable and the probability (or probability measure) of the event since each event has an associated probability (or probability measure). A *probability distribution* is to describe the probability measures over all the values of a random variable. A probability distribution can always be expressed in *cumulative distribution function* (CDF) for a random variable, either discrete or continuous, which is

$$F_X(x) = P(X \leq x) \tag{2.1}$$

Here X and x denote a random variable and its value respectively. For a discrete random variable, its cumulative distribution function is a step function which is nondifferentiable.

In addition to cumulative distribution function, *probability mass function* (PMF) and *probability density function* (PDF) are also used to describe probability distribution. PMF is for a discrete random variable which is simply the probabilities of discrete x, i.e. $P(x)$. For a continuous random variable, PDF is used. The probability of each single value is always zero since the sample space is infinite. As a result, probability density is defined for each value as another probability measure. Mathematically, CDF is the integral of PDF, and PDF is the derivative of CDF, which are,

$$F_X(x) = \int_{-\infty}^{x} f(x)dx \tag{2.2}$$

$$f_X(x) = \frac{dF_X(x)}{dx} \tag{2.3}$$

According to the axioms of probability, distribution functions must satisfied some certain properties.

Theorem 2.1 *The function* $F(x)$ *is a CDF if and only if the following three conditions hold*:

(a) $\lim_{x\to-\infty} F(x) = 0$ *and* $\lim_{x\to\infty} F(x) = 1$.

(b) $F(x)$ *is a nondecreasing function of* x.

(c) $F(x)$ *is right-continuous; that is, for every number* x_0, $\lim_{x\downarrow x_0} F(x) = F(x_0)$

Theorem 2.2 *A function* $f_X(x)$ *is a PDF(or PMF) of a random variable X if and only if*

(a) $f_X(x) \geq 0$ *for all* x.

(b) $\sum_x f_X(x) = 1$ (PMF) or $\int_{-\infty}^{\infty} f_X(x)dx = 1$ *(PDF).*

2.2.2 Probabilistic Characteristics

Once the distribution function of a random variable is known, either cumulative distribution function or probability density function, all of its probabilistic characteristics are known. Of all the probabilistic characteristics, *central value*, *dispersion measure*, and *skewness measure* are most important ones. These characteristics convey very useful statistical information of a random variable, and can be used to describe the random variable in the situation where the distribution function is unknown.

One of the central values is *mean* or the *expected value*. It is the weighted average of a random variable by probabilities, which gives us a typical or expected value of an observation of the random variable. It is usually denoted by $E(x)$, which is

$$E(x) = \sum_{i=1}^{n} x_i P(x_i) \tag{2.4}$$

for a discrete random variable, and

$$E(x) = \int_{-\infty}^{\infty} x f_X(x) dx \tag{2.5}$$

for a continuous random variable. The mean is also denoted by μ_X. The other two measures of the central value are *median* and *mode*. Median is the value of a random variable which makes the probability of the values blow it is equal to the probability of the values above it. Mode is the value of a random variable which has largest probability (discrete) or probability density (continuous).

Variance and *standard deviation* are the measures of dispersion of a random variable. They show how variable a random variable is around its mean. The larger variance means more widely the values are spread. For a discrete random variable, the variance is

$$Var(X) = \sum_{i=1}^{n} (x_i - \mu_X)^2 P(x_i) \tag{2.6}$$

and for a continuous random variable, it is

$$Var(X) = \int_{-\infty}^{\infty} (x - \mu_X)^2 f_X(x) dx \tag{2.7}$$

The standard deviation is simply the square root of the variance; that is

$$\sigma_X = \sqrt{Var(X)} \tag{2.8}$$

Skewness measure is to show if the distribution of a random variable is symmetry or asymmetry, and what is the asymmetry extent. The *third moment* is the most often used skewness measure, which is

$$E(X - \mu_X)^3 = \sum_{i=1}^{n} (x_i - \mu_X)^3 P(x_i) \tag{2.9}$$

for a discrete random variable, and

$$E(X - \mu_X)^3 = \int_{-\infty}^{\infty} (x - \mu_X)^3 f_X(x) dx \tag{2.10}$$

for a continuous random variable.

Table 2-1 is showing some common continuous distributions and their probabilistic characteristics:

Table 2-1 Common continuous distributions

Distribution	**Parameters and Support**	**Probability Density Functions**	**Probabilistic Characteristics**
Uniform	$-\infty < a < b < \infty$ $a \le x \le b$	$\frac{1}{b-a}$ for $a \le x \le b$ 0 for $x < a$ or $x > b$	Mean: $\frac{a+b}{2}$ Median: $\frac{a+b}{2}$ Mode: any x in [a,b]

			Variance: $\frac{(b-a)^2}{12}$ Skewness: 0
Triangle	$-\infty < a < b < \infty$ $a \le c \le b$ $a \le x \le b$	$\frac{2(x-a)}{(b-a)(c-a)}$ for $a \le x \le c$ $\frac{2(b-x)}{(b-a)(b-c)}$ for $c \le x \le b$	Mean: $\frac{a+b+c}{3}$ Median: $a+\frac{\sqrt{(b-a)(c-a)}}{\sqrt{2}}$ for $c \ge \frac{b-a}{2}$ $b-\frac{\sqrt{(b-a)(b-c)}}{\sqrt{2}}$ for $c \le \frac{b-a}{2}$ Mode: c Variance: $\frac{a^2+b^2+c^2-ab-ac-bc}{18}$ Skewness: $\frac{\sqrt{2}(a+b-2c)(2a-b-c)(a-2b+c}{5(a^2+b^2+c^2-ab-ac-bc)^{3/2}}$
Normal	$-\infty < \mu < \infty$ $\sigma > 0$ $-\infty < x < \infty$	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	Mean: μ Median: μ Mode: μ Variance: σ^2 Skewness: 0
Lognormal	$-\infty < \mu < \infty$ $\sigma > 0$ $0 \le x < \infty$	$\frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$	Mean: $e^{\mu+\sigma^2/2}$ Median: e^{μ} Mode: $e^{\mu-\sigma^2}$ Variance: $(e^{\sigma^2}-1)e^{2\mu+\sigma^2}$ Skewness: $(e^{\sigma^2}+2)\sqrt{e^{\sigma^2}-1}$
Exponential	$\lambda > 0$	$\lambda e^{-\lambda x}$	Mean: λ^{-1}

	$0 \le x < \infty$		Median: $(\ln 2)/\lambda$ Mode: 0 Variance: λ^{-2} Skewness: 2
Beta	$\alpha > 0$ $\beta > 0$ $0 \le x \le 1$	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$	Mean: $\frac{\alpha}{\alpha+\beta}$ Median: not exist Mode: $\frac{\alpha-1}{\alpha+\beta-2}$ Variance: $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ Skewness: $\frac{2(\beta-\alpha)\sqrt{\alpha+\beta+1}}{(\alpha+\beta+2)\sqrt{\alpha\beta}}$
Cauchy	$-\infty < \theta < \infty$ $\sigma > 0$ $-\infty < x < \infty$	$\frac{1}{\pi\sigma[1+(\frac{x-\theta}{\sigma})^2]}$	Mean: not exist Median: θ Mode: θ Variance: not exist Skewness: not exist
Gamma	$k > 0$ $\theta > 0$ $0 \le x < \infty$	$\frac{1}{\Gamma(k)} x^{k-1} e^{-x/\theta}$	Mean: $k\theta$ Median: no simple closed form Mode: $(k-1)\theta$ for $k \ge 1$ Variance: $k\theta^2$ Skewness: $\frac{2}{\sqrt{k}}$
Chi-square	$k > 0$ $0 \le x < \infty$	$\frac{1}{\Gamma(k/2)2^{k/2}} x^{k/2-1} e^{-x/2}$	Mean: k Median: approximately $k - 2/3$ Mode: $k-2$ if $k \ge 2$ Variance: $2k$ Skewness: $\sqrt{8/k}$

t-distribution	$v > 0$ $-\infty < x < \infty$	$\frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}(1+\frac{x^2}{v})^{-(\frac{v+1}{2})}$	Mean: 0 for $v > 1$ Median: 0 Mode: 0 Variance: $\frac{v}{v-2}$ for $v > 2$ Skewness: 0 for $v > 3$
Rayleigh	$\sigma > 0$ $0 \le x < \infty$	$\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$	Mean: $\sigma\sqrt{\pi/2}$ Median: $\sigma\sqrt{\ln 4}$ Mode: σ Variance: $\frac{4-\pi}{2}\sigma^2$ Skewness: $\frac{2\sqrt{\pi}(\pi-3)}{(4-\pi)^{3/2}}$
Weibull	$\lambda > 0$ $k > 0$ $0 \le x < \infty$	$(k/\lambda)(x/\lambda)^{k-1} e^{-(x/\lambda)^k}$	Mean: $\lambda\Gamma(1+1/k)$ Median: $\lambda \ln 2^{1/k}$ Mode: $\lambda(\frac{k-1}{k})^{1/k}$ for $k > 1$ Variance: $\lambda^2\Gamma(1+2/k) - \mu^2$ Skewness: $\frac{\Gamma(1+3/k)\lambda^3 - 3\mu\sigma^2 - \mu^3}{\sigma^3}$

2.3 Represented by Probability Density

In probabilistic simulation, probability density is thought as the best representation of a probabilistic output. There are two main reasons for it.

Firstly, it is because probability density is the most complete description of a random variable. Once the density function is attained, all the statistical characteristics of the random variable can be derived. By using the statistical information provided by the

density function, various further analysis and design activities can be carried on, such as risk assessment, sensitivity analysis, reliability-based design and so on.

The second reason is that, probability density can give important indication of the features such as multimodality and skewness. These features are very valuable in a complex and large-scale integrated simulation which can yield a system output with an arbitrary probability distribution. An example is given in Figure 2-1. The curve shown in this figure is constructed from the data generated by the simulation of a rod manufacturing process. It is clear from the figure that this is a multimodal distribution. It has a small proportion of density of a higher mode which is usually undesired. Thus this density curve provides a clue to address the potential problem in the manufacturing process.

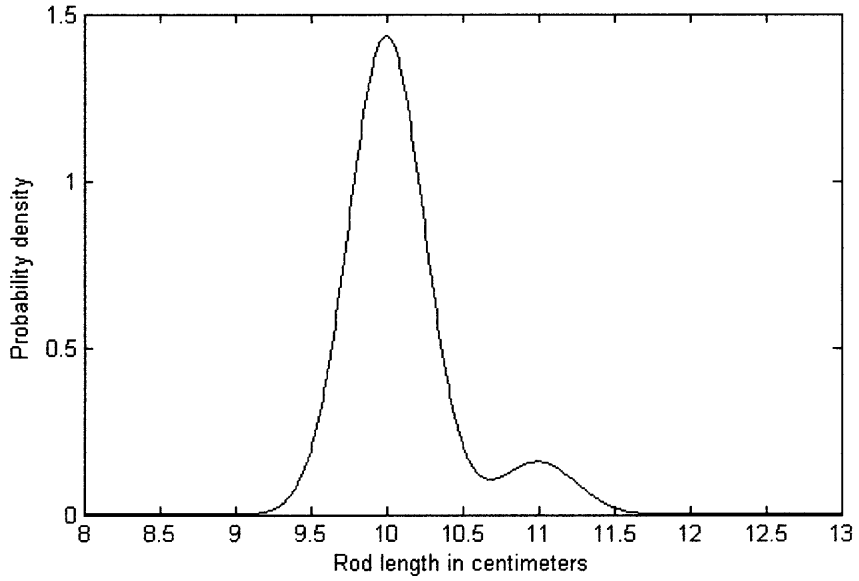


Figure 2-1 Probability density constructed from the rod manufacturing simulation

Another example is shown in Figure 2-2 (Silverman 1998). It is constructed from the data collected in an engineering experiment described by Bowyer (Bowyer 1980). The

height of a steel surface was measured at about 15000 points. Then the density curve was constructed based on these observations. The density curve is clearly showing that the height has a skew distribution with a long lower tail. This is an important indication because the lower tail represents hollows where fatigue cracks can start and also where lubricant might gather. From the curve, it is clear that the Gaussian models are not appropriate to model these surfaces.

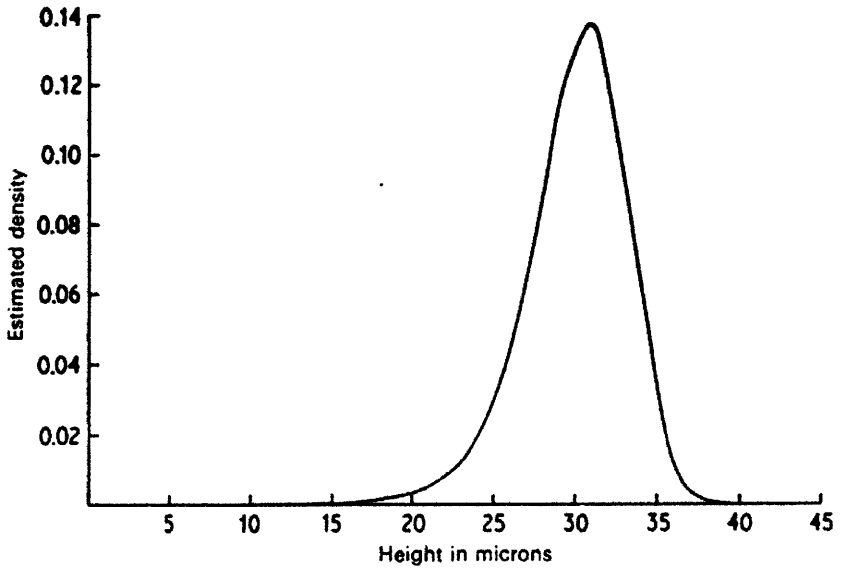


Figure 2-2 Density constructed from observations of the height of a steel surface

2.4 Monte Carlo Methods

A Monte Carlo method is a very general concept and the way it is used varies from field to field. Monte Carlo methods can be loosely defined as simulation methods which solve the problems by using random sampling and statistically computation. They are usually used under such situations where an analytic result can't be achieved or it is impossible to get the result by deterministic simulations. The applications of Monte Carlo methods are found in diverse fields, including complex simulations in aerospace engineering,

statistical mechanics, computational physics and chemistry, operation research, finance, biology statistics and so on (Rubinstein 1981; Kalos 1986; Fishman 1996; Gentle 2003; Robert 2004).

Monte Carlo methods are also widely used to resolve probabilistic simulation problems. Usually, they have the following steps:

1) Random sampling from the probability distributions of the simulation system inputs.
2) Perform the deterministic simulation on each set of input sample points.
3) Statistically aggregate the results of individual simulations to get the final probabilistic results.

In Monte Carlo based probabilistic simulations, the final probabilistic results representing the system outputs are histograms, which are one kind of estimations for probability density functions. For example, Figure 2-3 shows the histogram of 200 sample points resulting from the sum of two independent normal distributions which are N(1,1) and N(2,1) respectively.

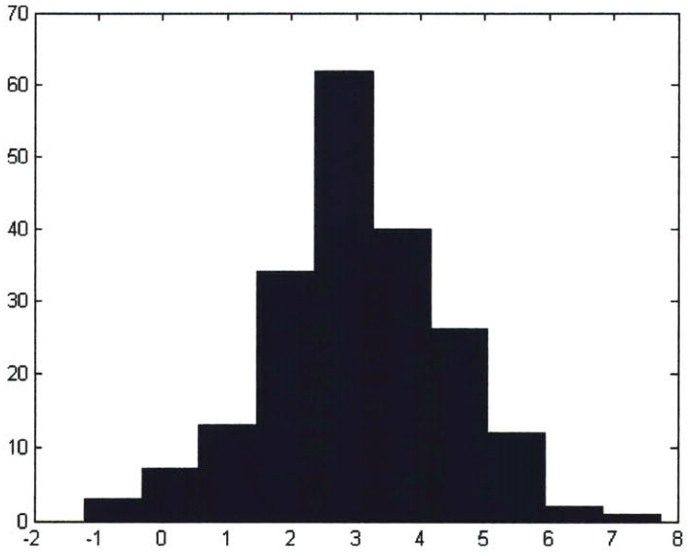


Figure 2-3 Histogram of sum of N(1,1) and N(2,1) by Monte Carlo

To get desired estimation accuracy, hundreds of thousands of iterations are usually needed in the conventional Monte Carlo approach. Figure 2-4 is showing the approach.

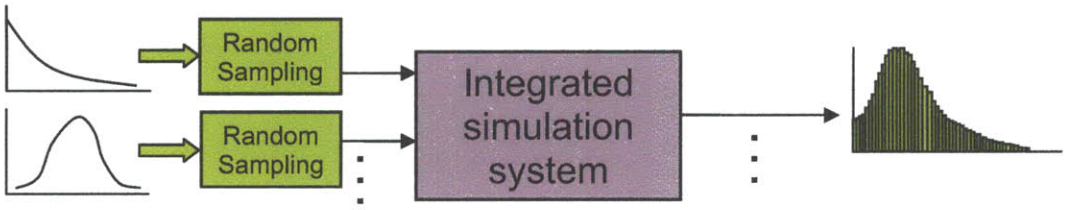


Figure 2-4 Conventional Monte Carlo approach

However, the integrated simulation is usually time-consuming due to its complexity and large scale. As a result, computation time makes it prohibitive to directly apply conventional Monte Carlo methods to large-scale integrated simulations since the traditional Monte Carlo method needs to run the integrated simulation a large number of times to get desired accurate results, and the total time then becomes unaffordable. On the other hand, conventional Monte Carlo methods put no effort into exploring more information on the underlying distribution from the data generated by the probabilistic simulation except for representing the result by a histogram. And we will see in section 3.3.1 that, there are various drawbacks to represent probability density by a histogram.

In this context, the significant challenge is how to make Monte Carlo-like probabilistic simulations more feasible in large-scale integrated simulations. The computational expense needs to be reduced greatly so that the emergent integrated simulation system using Monte Carlo methods can be scalable and practical in probabilistic design fields.

2.5 Advanced Monte Carlo Methods

There are some advanced Monte Carlo methods which have been developed with different goals. Simply speaking, these advanced Monte Carlo methods can be divided into two categories.

For those in the first category, the efforts are aimed at improving the quality of sampling. For example, stratified sampling can yield a sample more consistent to the underlying distribution than that from the traditional Monte Carlo sampling, by partitioning the sample space into some strata with even probabilities (Fishman 1996). Tong developed some refinement strategies for stratified sampling methods in (Tong 2006). Blasone carried on sampling using an adaptive Markov Chain Monte Carlo scheme (the Shuffled Complex Evolution Metropolis (SCEM-UA) algorithm) to improve the computational efficiency in (Blasone 2008). Tari's refined descriptive sampling (Tari 2006) provides a good approach to reduce the sampling bias and eliminate the problem of descriptive sampling related to the sample size. There is no doubt that all these methods can improve sampling of the system inputs in probabilistic simulations. However, just like the conventional Monte Carlo methods, they do little work on exploring the underlying distributions from the simulation results for the system outputs.

The second category is composed of the advanced Monte Carlo methods which are usually developed for the specific applications and to solve the specific problems. And most of them have to use the information inside the model as much as possible to achieve the improvement. Pradlwarter developed an advanced Monte Carlo simulation approach to analyze the stochastic structural dynamics (Pradlwarter 1997). This approach was exclusively developed to lead the generated samples towards the low probability range of

the system response which was the focus of the study. A suitable criterion was created for indicating the desired realizations by using the mathematical relationships inside the model. Thunnissen's method named Subset Simulation was developed for quantifying uncertainties in conceptual-level design (Thunnissen 2007). This advanced method was focused on estimating the extreme tail values like 99.99 percentile by using Markov Chain Monte Carlo simulation. Obviously these advanced methods were not developed for generic probabilistic simulations. Also, in integrated simulations, most models are black-boxes from user's aspect of view. The information inside the models can not be used easily for probabilistic simulations in a generic way.

2.6 Predictive Machine Learning Approach

For a computationally intensive integrated simulation system, only a limited number of simulations can be performed to meet the requirement of design cycle. A random sample of limited size can be attained for each probabilistic system input by (advanced) Monte Carlo sampling. Running the integrated simulation for each input state, a random sample of limited size can be attained for a system output to be studied. This sample can be looked as a random sample from an underlying probability distribution which is exactly what we are investigating. Instead of running more simulations to get more sample points to represent the underlying distribution, an effort is put on discovering the statistical information hidden in the sample data we already have, to estimate the underlying probability distribution.

Machine learning is to design and develop some algorithms and techniques so that a computer can learn from samples, data or experiences. It is widely applied to extract

pattern, logic and knowledge from data, by using different computational and statistical methods. In the process of learning, the computer can keep improving its inductive or deductive performance by studying the samples and data repeatedly. Its applications range from handwriting and speech recognition to a vehicle driving learning(Mitchell 1997).

In my thesis, a predictive machine learning approach is created to explore the underlying probability distribution, which is simply described in Figure 2-5. Its purpose is to learn the output probability distribution of an integrated simulation system from the sample of limited size which hides the statistical information inside. In this approach, the Monte Carlo methods are only used in sampling, like a pre-processing. Most work is focused on predictive machine learning, more like a post-processing. Details of this approach will be discussed in the next chapter.

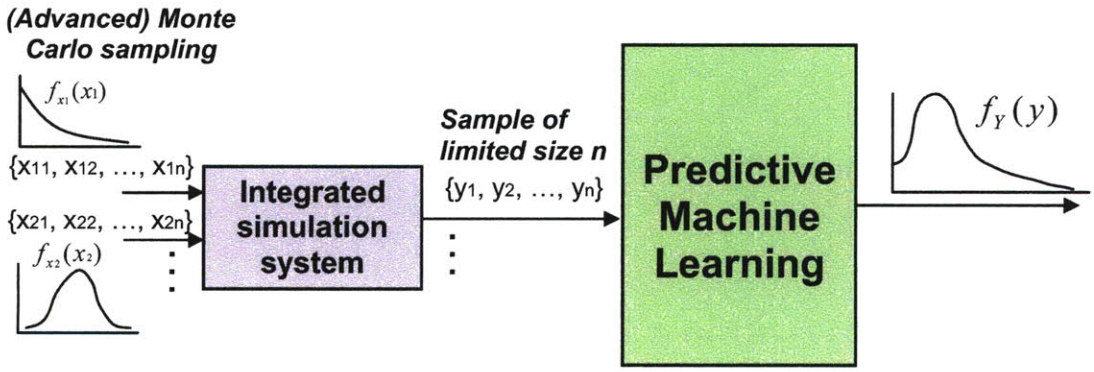


Figure 2-5 Predictive Machine Learning approach

Chapter 3

Machine Learning based Density Estimation

3.1 Density Estimation

In probabilistic simulations, what people are most interested in are the probability distributions of the system outputs. A probability distribution is a complete description for a random variable. Once a distribution function is known, all the statistical characteristics of the random variable can be derived. By using the statistical information provided by the distribution functions, further analysis and design activities can be carried on, such as risk assessment, sensitivity analysis, reliability-based design and so on.

In an integrated simulation system, the distribution of a system output is determined by the distributions of system inputs and the mathematical relationships inside the simulation system. In most cases, the individual simulation models inside the integrated simulation system are black boxes to a designer of system level. Their internal structures, mechanisms and mathematical relationships are unknown by a system designer since those individual simulation models are usually developed by individual model designers from various domains and disciplines. As a result, it is impossible to apply analytical methods to derive the distribution of a system output from the distributions of system inputs. Actually, even if every mathematical relationship inside an integrated system is

known, it's usually intractable to derive the output distributions due to the complexity and large scale of the whole simulation system.

By random sampling on the probabilistic system inputs for an affordable size and running the integrated simulation on each one, a random sample of a limited size can be achieved for each system output. Random sampling from inputs is to encode the distribution information into the samples. Doing integrated simulation for individual iterations is to propagate the probabilistic information. The random sample finally got for a system output encodes the underlying distribution information.

Suppose that we have got a set of observations or a random sample $\{x_1, x_2, ..., x_i, ..., x_N\}$ on a system output X by steps above. The next work is to deduce or construct an estimate for the probability density function based on the observations. In statistics, this problem is referred to *density estimation*. Traditional density estimation methods are divided into two categories, which are *parametric density estimation* and *nonparametric density estimation*.

3.2 Parametric Density Estimation

In parametric density estimation, it is assumed that the form of the underlying probability density function is known, for example, a Gaussian distribution. Then what is needed to do is to optimally estimate the parameters of the density function by fitting the assumed distribution model to the observed random sample. For a Gaussian distribution, it is to estimate μ and σ.

Suppose the density function form of a system output X is already known as $f_X(x \mid \theta)$ which depends on a vector of parameters $\theta = \{\theta_1, ..., \theta_m\}$. The random sample on X has been got as $\mathrm{x} = \{x_1, x_2, ..., x_i, ..., x_n\}$ which are independently coming from the same distribution $f_X(x \mid \theta)$. According to the probability theory, the joint probability density of the random sample is

$$p(\mathrm{x} \mid \theta) = \prod_{i=1}^{N} f_X(x_i \mid \theta) \tag{3.1}$$

A likelihood function is then defined by (Casella 2002)

$$L(\theta \mid \mathrm{x}) = p(\mathrm{x} \mid \theta) = \prod_{i=1}^{n} f_X(x_i \mid \theta) \tag{3.2}$$

Now the estimators for the parameters $\theta = \{\theta_1, ..., \theta_m\}$ are needed to be derived. *Maximum Likelihood* estimator is the most popular one in all existing methods. It is achieved by making $L(\theta \mid \mathrm{x})$ attain a maximum as a function of $\theta = \{\theta_1, ..., \theta_m\}$ while $\mathrm{x} = \{x_1, x_2, ..., x_i, ..., x_n\}$ is fixed. To get a maximum, a common way is to let the gradient of the likelihood function equal to zero:

$$\frac{\partial}{\partial \theta_i} L(\theta \mid \mathrm{x}) = 0, \qquad i = 1, ..., m \tag{3.3}$$

By solving the equations above, the estimates for the parameters can be derived as $\hat{\theta}(\mathrm{x}) = \{\hat{\theta}_1(\mathrm{x}), ..., \hat{\theta}_m(\mathrm{x})\}$. However, the first derivative is equal to 0 is not sufficient for a maximum. It can also happen for a minimum. To verify if it is a maximum, the second derivative is to be checked to see if

$$\frac{\partial^2}{\partial \theta_i^2} L(\theta \mid \mathrm{x}) \Big|_{\theta = \hat{\theta}(\mathrm{x})} < 0, \qquad i = 1, ..., m \tag{3.4}$$

Once MLE (Maximum Likelihood Estimate) is derived for each parameter, the parametric density estimation can be easily got by inserting the parameter estimates into the assumed density function, which is,

$$\hat{f}_X(x \mid \theta) = f_X(x \mid \hat{\theta}(\mathrm{x})) = f_X(x \mid \hat{\theta}_1(\mathrm{x}),\ldots,\hat{\theta}_m(\mathrm{x})) \tag{3.5}$$

Parametric density estimation can get a very good result in the situations where the forms of the underlying density functions are known based on some prior knowledge. However, in practice, especially in an integrated simulation environment, it is usually impossible to know the density function form of a system output due to the complexity and large scale of the integrated system. It can be an arbitrary distribution form which depends on the distributions of system inputs and the complicated unknown mathematical relationships inside the simulation system. In addition, most of the commonly used parametric densities are unimodal, while many practical densities are multimodal and can be arbitrary forms which have been never seen before. As a result, those parametric forms are not always good choices for the densities actually encountered in practice. A very bad result will be got if the assumed density function form doesn't match the true one.

3.3 Nonparametric Density Estimation

Nonparametric density estimation is to construct a density estimator only based on the random sample $\mathrm{x} = \{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ without any assumption about the functional form of the underlying density distribution. The simplest nonparametric approach is the histogram and the most popular method is kernel density estimation. They will be discussed in the following sections respectively.

3.3.1 Histogram Estimation

A histogram can be constructed by the following steps (Härdle 2004):

1) Decide an origin point and create bins with a width h along the real line:

$$b_j = [x_0 + (j-1)h, x_0 + jh), \qquad j \in \mathbb{Z} \tag{3.6}$$

2) For each bin, count the outcomes in it. Denote the number of observations that are located in bin j by n_j.

3) For each bin, convert the frequency count into the relative frequency, the sample analog of probability, by dividing the count by the sample size n, and by the bin width h in order to make the total area under the histogram equal to one:

$$f_j = \frac{n_j}{nh} \tag{3.7}$$

4) The histogram estimation can be expressed as

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} \sum_{j} I(x_i \in b_j) I(x \in b_j) \tag{3.8}$$

where

$$I(x_i \in b_j) = \begin{cases} 1 & x_i \in b_j \\ 0 & otherwise \end{cases}$$

Then an estimate of f for all x is given by formula (3.8) as well as its corresponding graph, the histogram. Denoting by m_j the center of the bin b_j, formula (3.8) is telling us that the histogram gives the same estimate for f, namely $\hat{f}_h(m_j)$, for any x in $b_j = [m_j - \frac{h}{2}, m_j + \frac{h}{2})$.

The total area of a histogram can prove to be equal to one without doubt, which is a desired property for any reasonable estimator of a probability density function.

By looking at formula (3.8), it can be found that the histogram estimation depends on the choice of width h. The following two plots are showing different choices for width for a random sample from N (0,1):

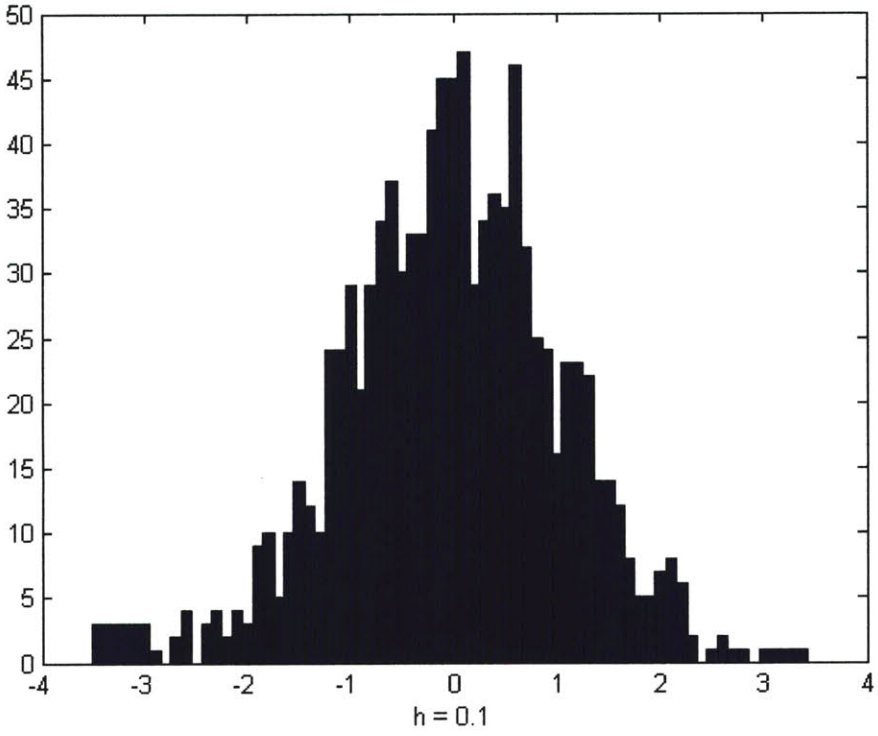


Figure 3-1 Histogram with bin width h = 0.1

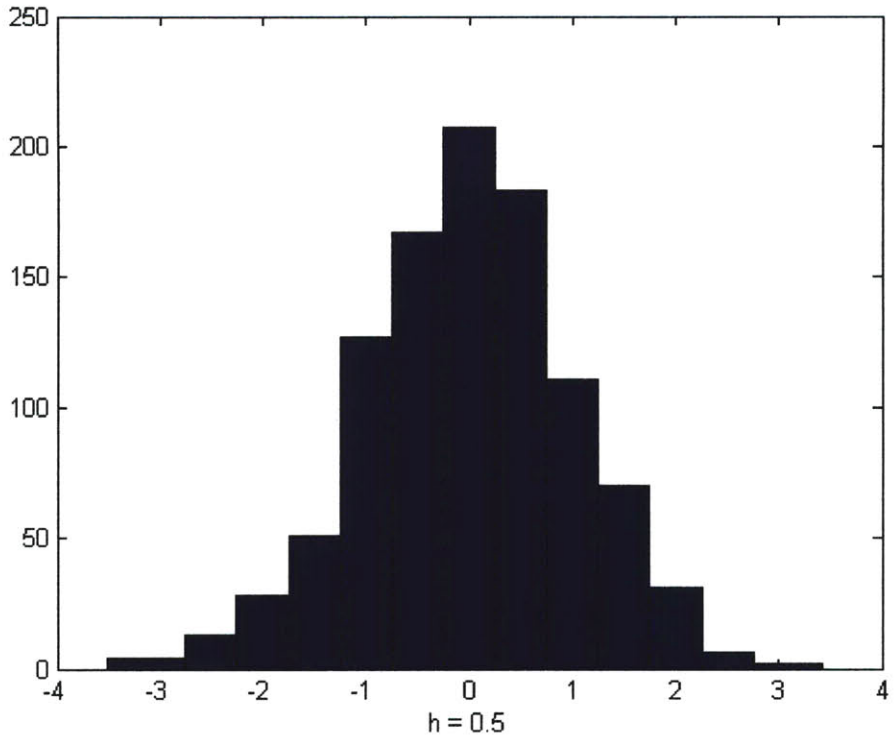


Figure 3-2 Histogram with bin width h = 0.5

Clearly, if we increase h, the histogram appears to be smoother, but some reasonable criterion is needed to say which bin width provides the "optimal" degree of smoothness.

By minimizing *mean integrated squared error* (MISE), which is defined as

$$MISE(\hat{f}_h) = E[\int_{-\infty}^{\infty} \{\hat{f}_h(x) - f(x)\}^2 dx] \tag{3.9}$$

we can get an optimal bin width, which is

$$h_0 = \left(\frac{6}{n \| f' \|_2^2} \right)^{1/3} \sim n^{-1/3} \tag{3.10}$$

But we can find that there is a problem to calculate $\| f' \|_2^2$ since f is unknown. A solution will be discussed in the next section. For now, a rule-of-thumb optimal bin width is given as(Scott 1992)

$$h_0 \approx 3.5n^{-1/3} \tag{3.11}$$

The histogram is a very simple and straightforward estimation, but it has various drawbacks:

- The final shape of the density estimate depends on the starting point of the bins and the bin width. Optimal bin width is not easy to get.
- The histogram estimation is not continuous. Its discontinuities are located at the boundaries of the bins, where the histogram function is not differentiable. However, it has zero derivatives elsewhere. It is obviously undesirable if we want to estimate a smooth, continuous PDF.
- The discontinuities of the estimate are not due to the underlying density, they are only an artifact of the chosen bin locations. These discontinuities make it very difficult, without experience, to grasp the structure of the data.

3.3.2 Kernel Density Estimation

Kernel density estimation is the most widely used nonparametric method. The basic intuition behind kernel density estimation is that each sample point x_i in the random sample $\mathrm{x} = \{x_1, x_2, ..., x_i, ..., x_n\}$ provides evidence for non-zero probability density at that point. A simple way to harness this intuition is to place an "atom" of mass at that point, just like what is shown in Figure 3-3. Moreover, making the assumption that the underlying probability density is smooth, we let the atoms have a non-zero "width". Superimposing n such atoms, one per sample point, we obtain a density estimate.

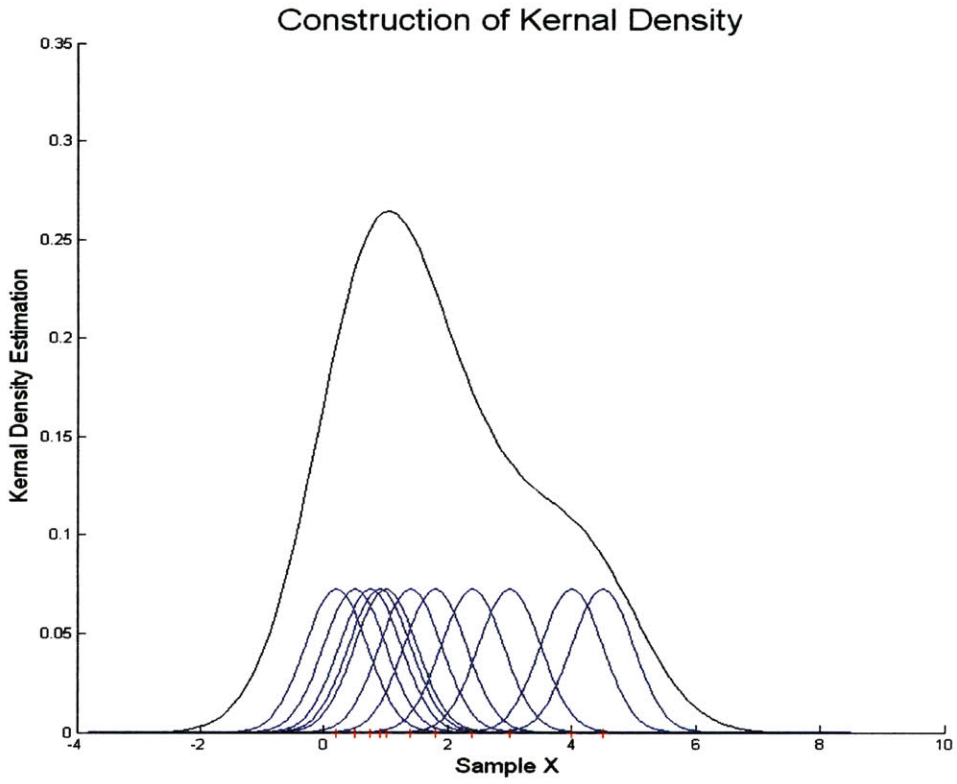


Figure 3-3 An example of kernel density estimation

More formally, let $K(u)$ be a *kernel function* – a nonnegative function integrating to one. Then $K_h(\frac{x-x_i}{h})$ will be the atom resulted from applying the kernel function to each sample point. The argument x_i determines the location of the kernel function; kernels are generally symmetric about x_i. The parameter h is a general "smoothing" parameter that determines the width of the kernel functions and thus the smoothness of the resulting density estimate. Superimposing n such kernel functions, and dividing by n, we obtain a probability density:

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(\frac{x-x_i}{h}) \tag{3.12}$$

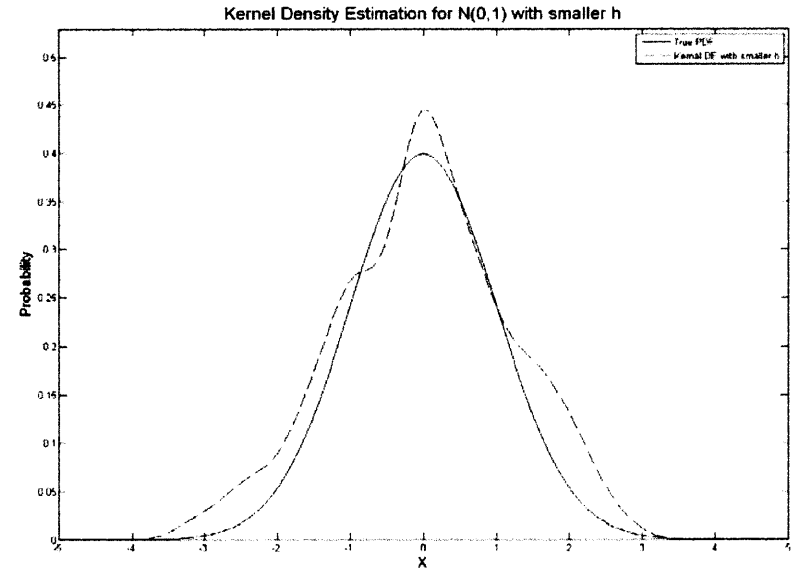
The commonly used kernel functions in practice are listed in Table 3-1(Härdle 2004).

Table 3-1 Common kernel functions

Name	Function form
Uniform	$\frac{1}{2}I(\lvert u \rvert \le 1)$
Triangle	$(1-\lvert u \rvert)I(\lvert u \rvert \le 1)$
Epanechnikov	$\frac{3}{4}(1-u^2)I(\lvert u \rvert \le 1)$
Quartic (Biweight)	$\frac{15}{16}(1-u^2)^2 I(\lvert u \rvert \le 1)$
Triweight	$\frac{35}{32}(1-u^2)^3 I(\lvert u \rvert \le 1)$
Gaussian	$\frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}u^2)$
Cosine	$\frac{\pi}{4}\cos(\frac{\pi}{2}u)I(\lvert u \rvert \le 1)$

It has been known for some time that although the Epanechnikov kernel minimizes the optimal asymptotic MISE with to K, MISE is quite insensitive to the shape of the kernel. As a result, for practical purposes the choice of the kernel function is almost irrelevant for the efficiency of the estimate (Härdle 2004). For simplicity, a Gaussian kernel is usually used.

Similar to the histogram, h controls the smoothness of the estimate and the choice of h is a crucial problem. The Figure 3-4 and Figure 3-5 are showing kernel density estimation based on a random sample from N(0,1) with different choices of the width h. The Gaussian kernel function is used in the estimation.



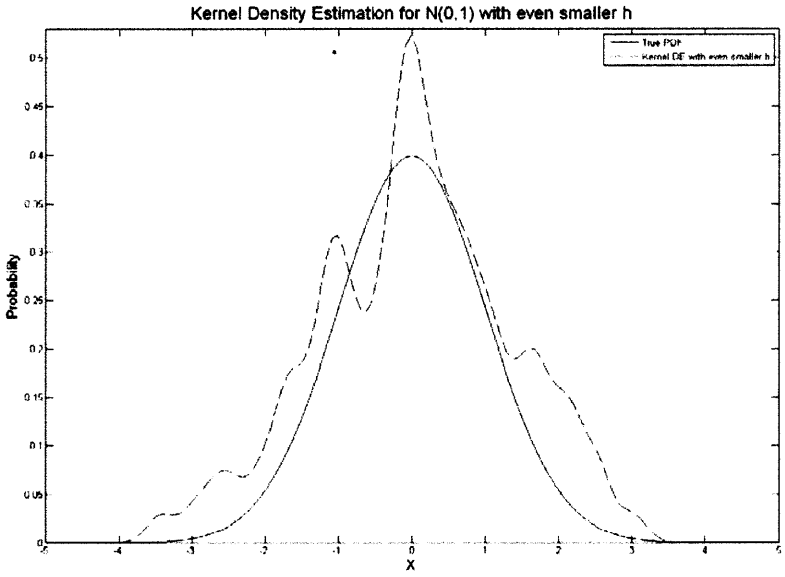


Figure 3-4 Kernel density estimates with small width

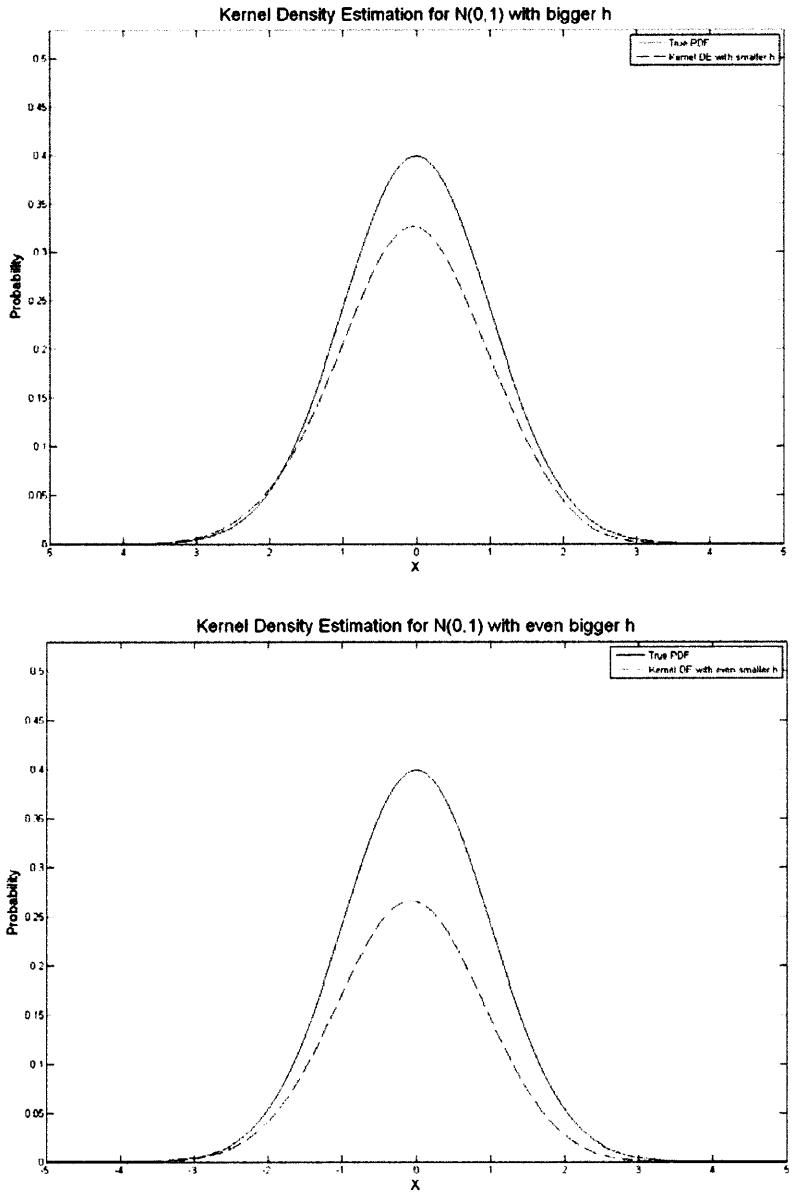


Figure 3-5 Kernel density estimates with large width

From the plots, we can see that kernel density estimation is very sensitive to the choice of the width h. How to determine an optimal h has been the main research focus for a long time. The criterion for an optimal h is usually set as minimizing *mean integrated squared error* (MISE), which is defined in formula (3.9). In kernel density estimation, this will become,

$$MISE(\hat{f}_h) = \int MSE\{\hat{f}_h(x)\}dx$$

$$= \frac{1}{nh} \| K \|_2^2 \int f(x)dx + \frac{h^4}{4}\{\mu_2(K)\}^2 \int \{f^{"}(x)\}^2 dx + o(\frac{1}{nh}) + o(h^4)$$

$$= \frac{1}{nh} \| K \|_2^2 + \frac{h^4}{4}\{\mu_2(K)\}^2 \| f^{"} \|_2^2 + o(\frac{1}{nh}) + o(h^4) \tag{3.13}$$

Here we denote $\mu_2(K) = \int s^2 K(s)ds$ and $\| K \|_2^2 = \int K^2(s)ds$. Ignoring higher order terms an approximate formula for the MISE, called AMISE, can be given as

$$AMISE(\hat{f}_h) = \frac{1}{nh} \| K \|_2^2 + \frac{h^4}{4}\{\mu_2(K)\}^2 \| f^{"} \|_2^2 \tag{3.14}$$

Differentiating the AMISE with respect to h and solving the first-order condition for h yields the AMISE optimal width

$$h_{opt} = \left(\frac{\| K \|_2^2}{\| f^{"} \|_2^2 \{\mu_2(K)\}^2 n} \right)^{1/5} \sim n^{-1/5} \tag{3.15}$$

But, here we find the problem of solving unknown term, which is $\| f^{"} \|_2^2$. This is the same problem we met in histogram estimation. Two most frequently used methods to solve this problem are the plug-in method and the method of cross-validation.

Generally speaking, plug-in methods derive their name from their underlying principle: if you have an expression involving an unknown parameter, replace the unknown parameter with an estimate. For the cross-validation method, the basic algorithm involves removing a single value, say x_i, from the sample, computing the appropriate density estimate at that x_i from the remaining $n-1$ sample points, denoted by $\hat{f}_{h,i}(x_i)$, and then optimizing some given criterion involving all $\hat{f}_{h,i}(x_i)$ to get h. For

details about the plug-in method and the cross-validation method, refer to (Härdle 1991; Park 1992)

Kernel density estimation is very sensitive to the choice of the width h. Choosing a smaller width makes the density curve bumpier, and choosing a larger width yields smoother curve but may lose density details and precisions. Although we have the plug-in method and the cross-validation method to get an "optimal" width, there is no one best method existing.

3.4 Machine Learning based Estimator

The core of the predictive machine learning approach in this thesis is to create a machine learning based estimator for the probability density function underlying the random sample coming out from an integrated simulation. In the field of machine learning, the neural network technique is the best candidate for function approximations since the neural networks with an appropriate structure can map any complex function. In our approach, this technique is used to construct the density estimator.

There have been some methods to use neural networks for density estimation. However, most of the approaches inevitably have the limitations of the parametric approaches discussed earlier since they're based on parametric models. For example, a mixture model of Gaussian distributions is assumed in the approaches of (Bishop 1995; Husmeier D. and Taylor 1998), where a multilayer network is used to estimate the mean and variance. Williams's approach(Williams 1996) is similar with the above except that it is for the multidimensional estimation. The mixture of Gaussian density estimations is used in Traven's approach(Traven May 1991) and Cwik and Koronacki's method(Cwik

1996). Roth and Baram(Roth 1996), and Miller and Horn(Miller 1998) trained the neural networks to maximize the entropy of the outputs. Van Hulle(Hulle 1996) developed a self-organizing approach by which the density of the weight vectors is an estimate of the unknown density with the algorithm converging to a solution. Schioler and Kulczyki's method(Schioler 1997) is based on the kernel estimation method. Weigend and Srivastava's approach(Weigend 1995) uses the method of fractional binning for time series prediction. It first partitions the input space into different bins, and then creates an output neuron for each bin, and finally gets the network trained on the fraction of data in each different bin. Zeevi and Meir(Zeevi 1997) proposed an approach to use convex combinations of various density estimators. Smyth and Wolpert(Smyth 1998) also brought out a method to combine the estimates of different density estimators such as different kernel estimators. Neural networks have also been developed for estimating discrete distributions in the approaches of Thathachar and Arvind(Thathachar 1999), and Adali et al.(Adali 1997).

The approach proposed in this thesis can be looked as a nonparametric approach since it is not assuming a certain functional form for the distribution to be estimated. An arbitrary function can be represented by the weights of the networks which are attained by learning from the sample. Unlike the kernel density estimation, this approach does not trap you into how to determine an optimal width. Also, some research(Hornik 1990; Hornik 1994) has been done to show that the ability of neural networks to approximate a function does not decay with the dimensions increasing, that is a problem for the kernel methods. The approach developed here is to construct a multilayer neural network and train it by sample data to get a neural network estimator for the cumulative distribution

function, and then the probability density estimator can be derived simply by differentiation. There are two main reasons to estimate the cumulative distribution function first. The first reason is that the cumulative distribution function is very important in practice to give a quantile or do the reliability assessment. Secondly, estimation of the cumulative distribution function is less sensitive than that of the probability density function to statistical fluctuations since the integral does the work of regularization(Magdon-Ismail 2002). Figure 3-6 shows the histograms of 4 random samples drawn from the same random variable which reflect the statistical fluctuations.

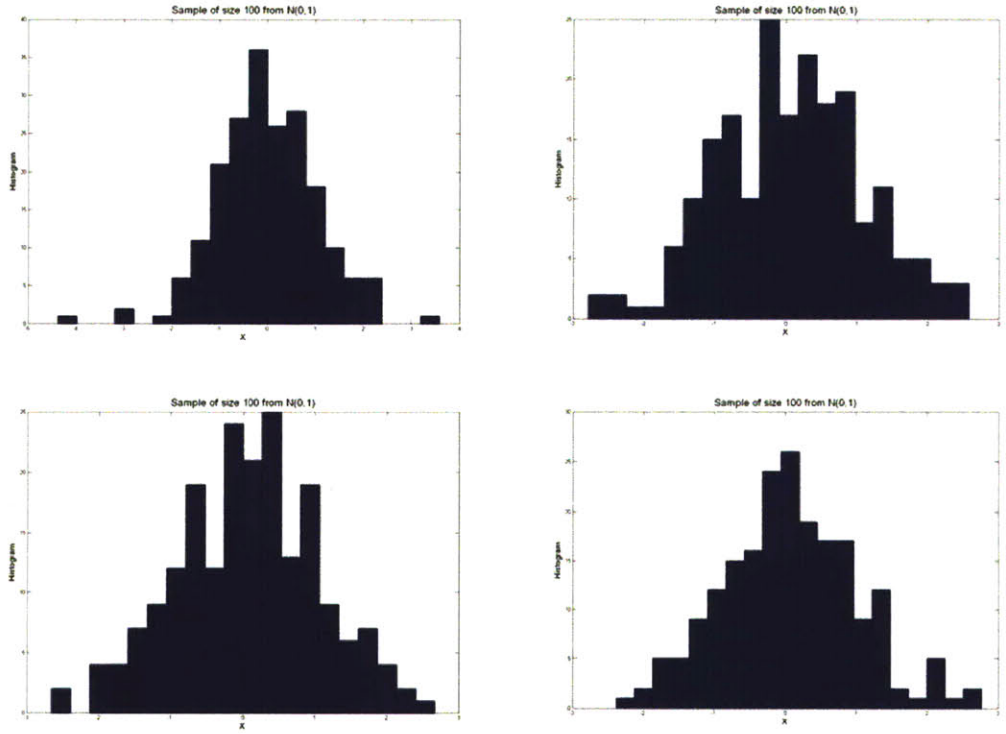


Figure 3-6 4 random samples from the same random variable

The approach to create a neural network estimator is described in the Figure 3-7:

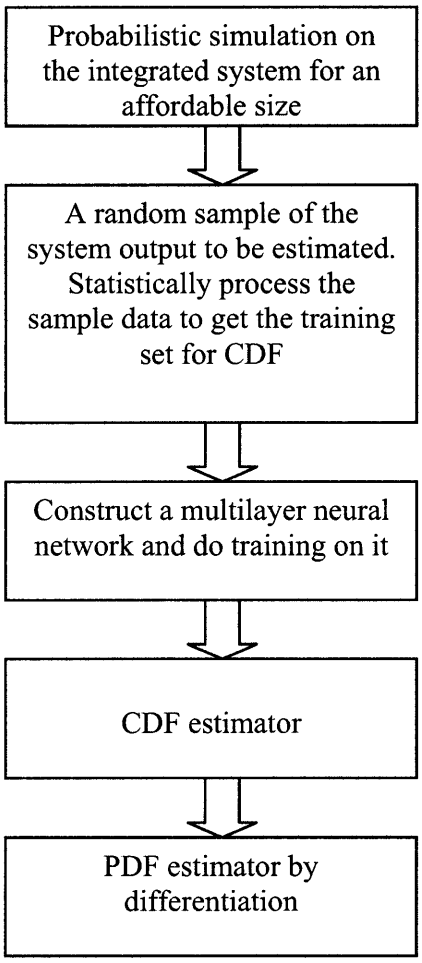


Figure 3-7 Approach to create a neural network estimator

Chapter 4

Construction of Neural Network Estimator

4.1 Neural Network Construction for CDF Estimator

4.1.1 Neural Network Architecture

A neural network is designed to simulate the behavior of nervous systems of humans to learn from experiences and make generalizations(Haykin 1999). Just like the human brain which is a nonlinear and parallel information-processing system, a neural network is composed of many neurons linked together by weighted connections called synapses. Each neuron is a computing or information-processing unit. A neuron model is described in Figure 4-1.

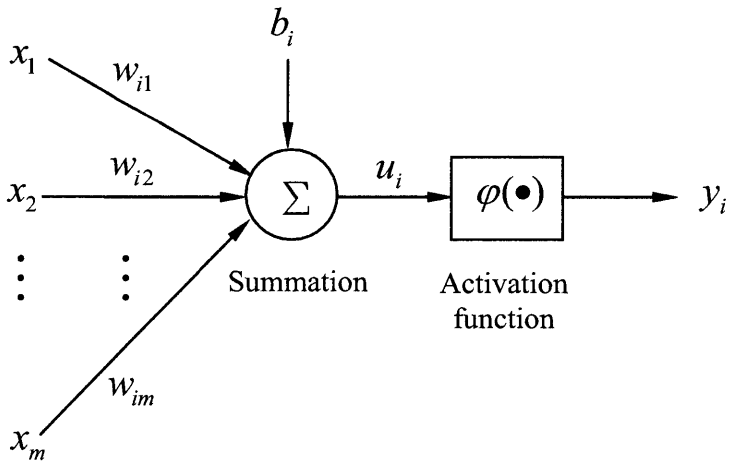


Figure 4-1 A neuron model

In this neuron model, the subscript i means it is the ith neuron in the neural network. $\{x_1,...,x_m\}$ denotes the input signals external to the network or the responses from other neurons which are connected to the ith neuron. The sum of the input signals weighted by w_{ij} of the synapses is calculated first. The bias b_i of the neuron is also added in. It is used to adjust the net input of the activation function. We can write the following equation for the summation:

$$u_i = \sum_{j=0}^{m} w_{ij} x_j \tag{4.1}$$

where $w_{i0} = b_i$ and $x_0 = 1$. This can be equivalently looked as there is a new input signal with a weight $w_{i0} = b_i$ whose value is fixed to 1.

The neuron produces a response by applying an activation function to the sum of its input signals. The activation function, also called the transfer function, is usually nonlinear. The response can be expressed as:

$$y_i = \varphi(u_i) \tag{4.2}$$

Neurons are the fundamental blocks of a neural network. According to the way they are connected by the weighted synapses, architectures of neural networks can be grouped into three categories: single-layer feedforward architecture, multilayer feedforward architecture and recurrent architecture(Haykin 1999).

Single-Layer Feedforward Architecture

If a neural network organizes its neurons layer by layer, it is called a layered neural network. When we count the layers of a neural network, the input layer is not included since it does not do any computation, but only pass the input signals to the next layer. As

a result, a single-layer network is composed of the input layer and the output layer. All computing and information-processing work is done by the output layer. The word *feedforward* means the signals are processed and passed layer by layer, starting from the input layer and ending at the output layer. There is no feedback loop existing in the network, and there is no connection between any two neurons in the same layer. A single-layer feedforward network has the simplest architecture and its application is limited.

Multilayer Feedforward Architecture

A multilayer feedforward network organizes its neurons in the same way with a single-layer one. The difference lies in that it has one or more *hidden* layers. The internal layers between the input layer and the output layer are called hidden layers. The hidden layers endow more power to a neural network for high-order computation. The input signals are passed to the first hidden layer, then the computed response from the first hidden layer are continuously fed to the second hidden layer, and so on until the response from the output layer are achieved.

Recurrent Architecture

If a neural network has at least one *feedback* loop or closed path, it is called a recurrent neural network. A feedback loop means that, the response from a neuron can be passed back directly or indirectly to the input of that neuron. A direct feedback is called *self-feedback*. A recurrent neural network can organize its neurons in any way by means

of feedback loops. Figure 4-2 is showing a recurrent neural network with self-feedback and hidden neurons.

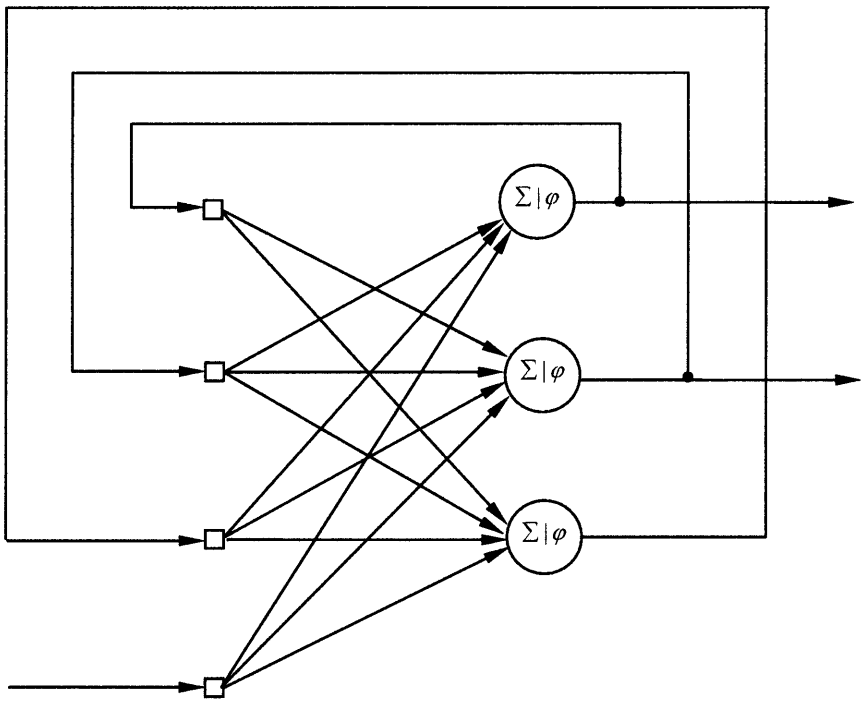


Figure 4-2 A recurrent neural network

Construct the CDF Estimator as a Multilayer Feedforward Neural Network

The neural network estimator for CDF is to estimate the cumulative distribution function:

$$y = F_X(x)$$

The input to the estimator is any value x from a random variable, and the output is the estimated cumulative probability y corresponding to x. The cumulative probability should never have an influence on the sample value x. That is to say, there should be no

feedback loops in the network. As a result, feedforward architecture is to be used. In this thesis, the focus is on the individual distributions of the outputs of an integrated system, not the joint distributions between the outputs. Then it is actually a univariate estimation problem. In nature, this is a one-dimension function approximation problem. The network is composed of the input layer with one neuron, the output layer with one neuron, and some hidden layers.

The next step is to decide how many hidden layers are needed for our estimation. The *universal approximation theorem* is stated as(Haykin 1999):

Theorem 4.1 *Let* $\varphi(\cdot)$ *be a nonconstant, bounded, and monotone-increasing continuous function. Let* I_{m_0} *denote the* m_0 *-dimensional unit hypercube* $[0,1]^{m_0}$. *The space of continuous functions on* I_{m_0} *is denoted by* $C(I_{m_0})$. *Then, given any function* $f \in C(I_{m_0})$ *and* $\varepsilon > 0$, *there exist an integer M and sets of real constants* α_i, b_i *and* w_{ij}, *where* $i = 1,...,m_1$ *and* $j = 1,...,m_0$ *such that we may define*

$$F(x_1,...,x_{m_0}) = \sum_{i=1}^{m_1} \alpha_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right) \tag{4.3}$$

as an approximate realization of the function $f(\cdot)$; that is,

$$| F(x_1,...,x_{m_0}) - f(x_1,...,x_{m_0}) | < \varepsilon$$

for all $x_1, x_2, ..., x_{m_0}$ *that lie in the input space.*

By applying the universal approximation theorem to multilayer feedforward neural networks, we can get a conclusion, that is, by choosing an activation function which is satisfying the conditions in the theorem, a multilayer feedforward neural network with one hidden layer can approximate an arbitrary continuous function $f(x_1,...,x_{m_0})$. As to

our CDF estimator, it is a univariate case of the theorem. We can say that, a multilayer feedforward neural network, composed of one-dimensional input layer, one-dimensional output layer and one hidden layer with m neurons, can approximate an arbitrary cumulative distribution function $F_X(x)$. Finally, the neural network estimator for CDF is constructed like what is shown in Figure 4-3. The superscripts on the weights indicate the linked layers. A weight with '*IH*' means the synapse connects an input neuron to a hidden neuron. A weight with '*HO*' means the synapse connects a hidden neuron to an output neuron.

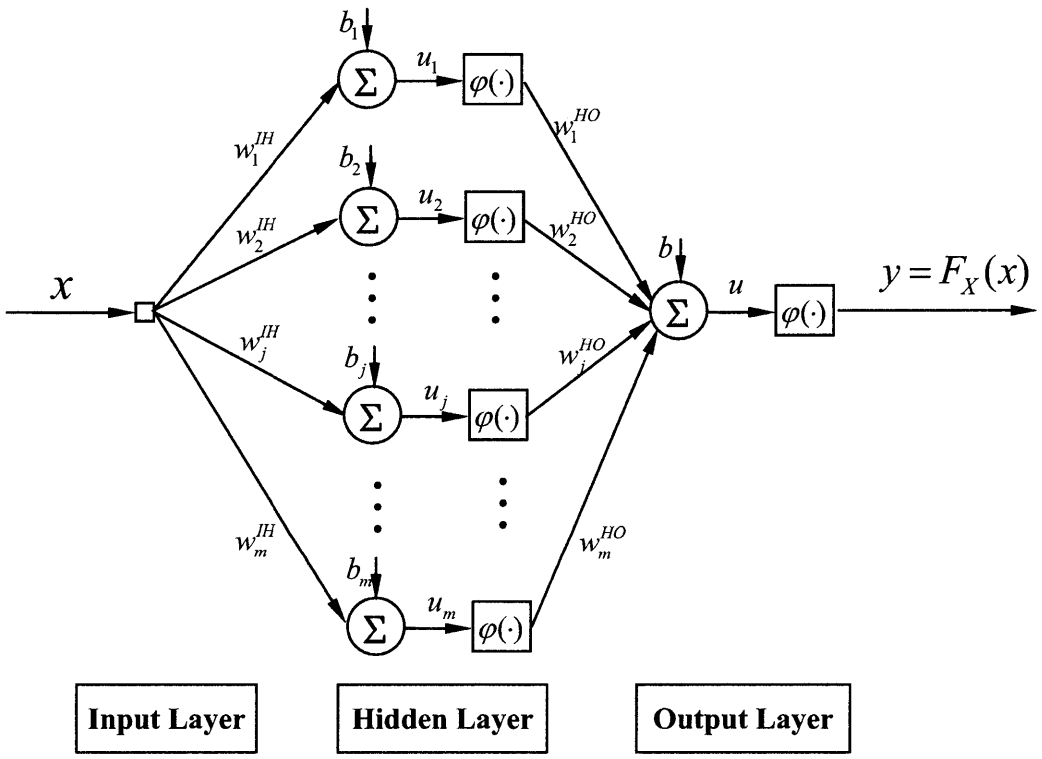


Figure 4-3 Structure of CDF estimator

4.1.2 Choose an Activation Function

A neuron in the network generates its response or output by applying an activation function on its net input. The common activation functions in use with neural networks are listed in the following:

Threshold Function

This function is also called *step function* or *Heaviside function*. The output is 1 if the input is above some threshold value which is 0 here, and 0 if the input is below the threshold value.

$$\varphi(u) = \begin{cases} 1 & if\ u \geq 0 \\ 0 & if\ u < 0 \end{cases} \tag{4.4}$$

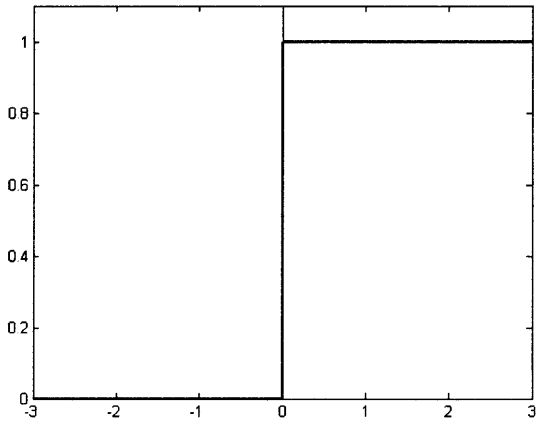


Figure 4-4 Threshold function

Linear Function

Another name for a linear function is *identity function* since the output is always identical to the input.

$$\varphi(u) = u \tag{4.5}$$

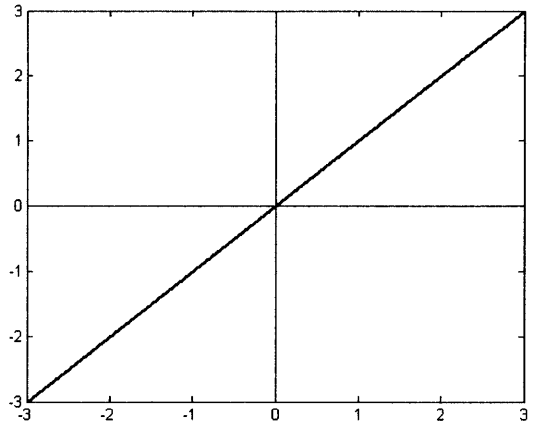


Figure 4-5 Linear function

Ramp Function

The ramp function, also named *Piecewise-Linear function,* is a combination of the threshold function and the linear function. It has two threshold values T_1 and T_2 ($T_1 < T_2$). It fires 0 when the input is below T_1 and 1 when the input is above T_2. It is a linear function for the inputs lying between T_1 and T_2. In Figure 4-6, $T_1 = -0.5$ and $T_2 = 0.5$.

$$\varphi(u) = \begin{cases} 1, & u > 0.5 \\ u + 0.5, & -0.5 \le u \le 0.5 \\ 0, & u < -0.5 \end{cases} \tag{4.6}$$

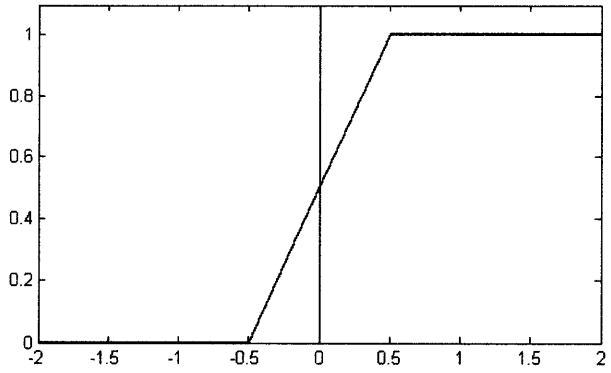


Figure 4-6 Ramp function

Sigmoid Function

The sigmoid function is an s-shaped function. It is monotonically increasing ranging from 0 to 1 continuously. The sigmoid function, showing a good balance between linear and nonlinear behavior, is the most commonly used activation function in construction of a neural network(Haykin 1999). A sigmoid function is defined by

$$\varphi(u) = \frac{1}{1+e^{-cu}} \tag{4.7}$$

where the parameter c is a measure of slope. Figure 4-7 is showing a sigmoid function with $c = 1$.

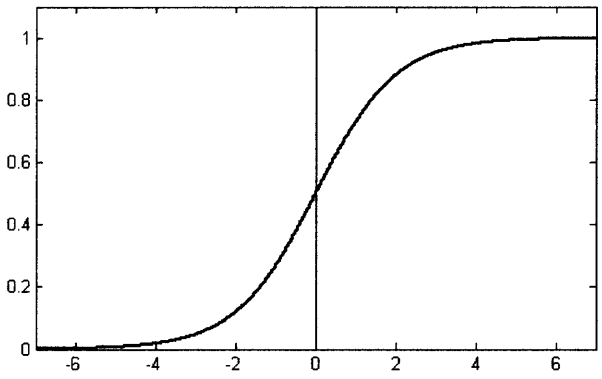


Figure 4-7 Sigmoid function

Hyperbolic Tangent Function

The hyperbolic tangent function has the similar features with the sigmoid function. It is ranging from -1 to 1, which is different from the sigmoid function. The function is defined by

$$\varphi(u) = \tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}} \tag{4.8}$$

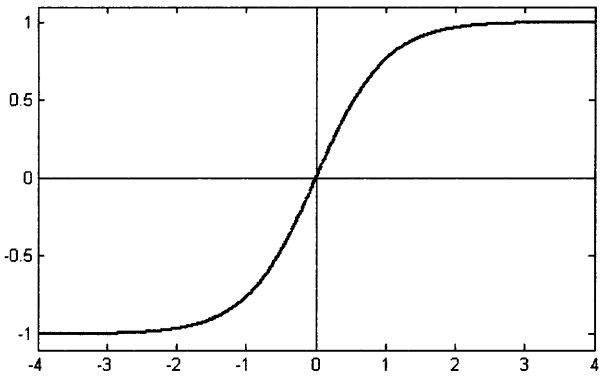


Figure 4-8 Tanh function

Choosing Sigmoid Function

The sigmoid function is chosen as the activation function for our neural network estimator, considering three things in the following.

The first consideration is that the activation function has to be differentiable. There are two reasons for that. One is the way by which the PDF estimator is derived. The CDF estimator is finally represented by the trained neural network (The training set and training algorithm will be discussed in the next chapters). Although a neural network is composed of neurons and synapses, in nature it can be expressed by an analytic mathematical model whose fundamental elements are the activation functions and the weights. The PDF estimator is attained by differentiating the mathematical model standing for the CDF estimator, which requires the activation function be differentiable. The other reason is involved with the training algorithm. The *error back-propagation algorithm*, the most popular training algorithm for multilayer feedforward neural networks, is used in this thesis. The back-propagation algorithm achieved its goal by using differentiation, which needs the activation function to be differentiable. Of the

common activation functions, the linear function, the sigmoid function and the tanh function have good differentiability.

In the universal approximation theorem which is described in theorem 4.1, the activation is required to be a *nonconstant, bounded, and monotone-increasing continuous function* to enable a neural network to approximate any continuous function. This is a must for the CDF estimator since the distribution to be estimated in an integrated simulation environment can be an arbitrary functional form. As a result, the linear function is excluded here. This is also understandable if we think about it in another way. The CDF estimator needs nonlinearity since the cumulative distribution functions are usually nonlinear. However, a linear function of linear functions is again a linear function.

The last reason is that, the range of the sigmoid function is exactly the same with that of a cumulative distribution function, which is always from 0 to 1. By using the sigmoid function, especially in the output layer, no extra scaling work needs to do. This will make less overhead work, especially for the back-propagation algorithm.

4.2 Derive PDF Estimator

Once we have the neural network estimator for CDF, the PDF estimator can be derived from the CDF estimator by differentiation.

Firstly, the mathematical model of the CDF estimator needs to be derived, based on the structure described in Figure 4-3. The net input to the hidden neuron is,

$$u_j = xw_j^{IH} + b_j \tag{4.9}$$

Then the response of the hidden neuron is,

$$v_j = \varphi(u_j) \tag{4.10}$$

where $\varphi(\cdot)$ denotes the activation function. The net input to the output neuron is,

$$u = \sum_{j=1}^{m} v_j w_j^{HO} + b \tag{4.11}$$

where m is the neuron number in the hidden layer. The final output of the CDF estimator is,

$$y = \varphi(u) \tag{4.12}$$

If we put everything in (4.9) – (4.12) together, we can get,

$$y = \varphi\left(\sum_{j=1}^{m} \varphi\left(x w_j^{IH} + b_j \right) w_j^{HO} + b \right) \tag{4.13}$$

where $\varphi(u) = \dfrac{1}{1+e^{-u}}$, the sigmoid function.

Now we have the mathematical model for the CDF estimator. By differentiating it in an inverse order, the PDF estimator can be derived. The differentiation starts from the output layer:

$$y'(x) = \varphi'(u)u'(x) \tag{4.14}$$

where $\varphi'(u) = \dfrac{e^{-u}}{(1+e^{-u})^2}$. Then we have,

$$y'(x) = \frac{e^{-u}}{(1+e^{-u})^2} \cdot u'(x) \tag{4.15}$$

According to the formula (4.11), we get,

$$u'(x) = \sum_{j=1}^{m} w_j^{HO} \cdot v_j'(x) \tag{4.16}$$

Continue to differentiate backward, $v_j^{'}(x)$ can be got from the formula (4.10) as,

$$v_j^{'}(x) = \varphi^{'}(u_j) \cdot u_j^{'}(x) = \frac{e^{-u_j}}{(1+e^{-u_j})^2} \cdot u_j^{'}(x) \tag{4.17}$$

From the formula (4.9), $u_j^{'}(x)$ can be got as,

$$u_j^{'}(x) = w_j^{IH} \tag{4.18}$$

Putting (4.16) – (4.18) together, we have the mathematical model for the PDF estimator, which is,

$$y^{'}(x) = \frac{e^{-u}}{(1+e^{-u})^2} \cdot \sum_{j=1}^{m} w_j^{HO} \cdot \frac{e^{-u_j}}{(1+e^{-u_j})^2} \cdot w_j^{IH} \tag{4.19}$$

where u can be got from (4.11) and u_j can be got from (4.9)

Chapter 5

Learning from Sample

5.1 Learning Algorithm

5.1.1 Supervised Learning

In chapter 4, the neural network estimator for CDF has been created (The PDF estimator is derived from the CDF estimator by differentiation). By learning from the random sample of the integrated system output which is a random variable in nature, the CDF estimator can finally estimate the underlying cumulative distribution function. So the task here is to learn a continuous function, or approximate a continuous function, from the sample. Apparently, this falls into the learning paradigm of *supervised learning*. The other two major types are *unsupervised learning* and *reinforcement learning*. Unsupervised learning is usually used in classification, pattern recognition, and clustering. Reinforcement learning has its main applications in controlling and sequential decision making.

Supervised learning, also called *learning with a teacher*, is to provide a *training set*, or *training sample*, to the neural network which is to be trained. The training set includes a set of input *examples* (or *patterns*) and a set of output examples which are *desired responses* corresponding to the input signals. By using a learning algorithm, the weights of the synapses and the biases in the neural network are adjusted in order to minimize the

difference between the actual responses and the anticipated responses. This process is repeated example by example until some criterion is reached finally.

So far, what we have as the training set is the random sample which is only the training data for input. In order to get the desired responses for the CDF estimator, i.e., the anticipated cumulative probabilities for each sample point, some statistical work needs to be done. The next sections are focused on discussing how to get the training set for the output of the neural network CDF estimator. For now, let's assuming we have attained the output examples, which are denoted by $\{d_1, d_2, ..., d_i, ..., d_n\}$. The training set for the input is denoted by $\{x_1, x_2, ..., x_i, ..., x_n\}$.

5.1.2 Back-propagation Learning

The *back-propagation learning*(Rumelhart 1986) is the most popular algorithm of supervised learning to train the multilayer feedforward neural network. It is also used in this thesis to train the CDF estimator. The back-propagation learning is composed of two processes for each training example(Haykin 1999). The first one is called *forward process*. In this process, a sample value in $\{x_1, x_2, ..., x_i, ..., x_n\}$ is presented to the CDF estimator. Then it is fed forward layer by layer. During this process, all the weights keep unchanged to calculate the responses layer by layer until the output of the CDF estimator is produced, which is the cumulative probability estimate for the current sample point. Of course, the current estimate is not the final one since the CDF estimator is still in learning. The actual output is compared with the desired response in $\{d_1, d_2, ..., d_i, ..., d_n\}$, and an error is calculated by applying some defined *error function*. Then it goes into the second process, *backward process*. In this process, the error is propagated backward layer by

layer, starting from the output layer. The weights are modified according to some rule in order to minimize the error so that the actual response from the network is getting closer to the anticipated one. The whole process of back-propagation learning is described in Figure 5-1.

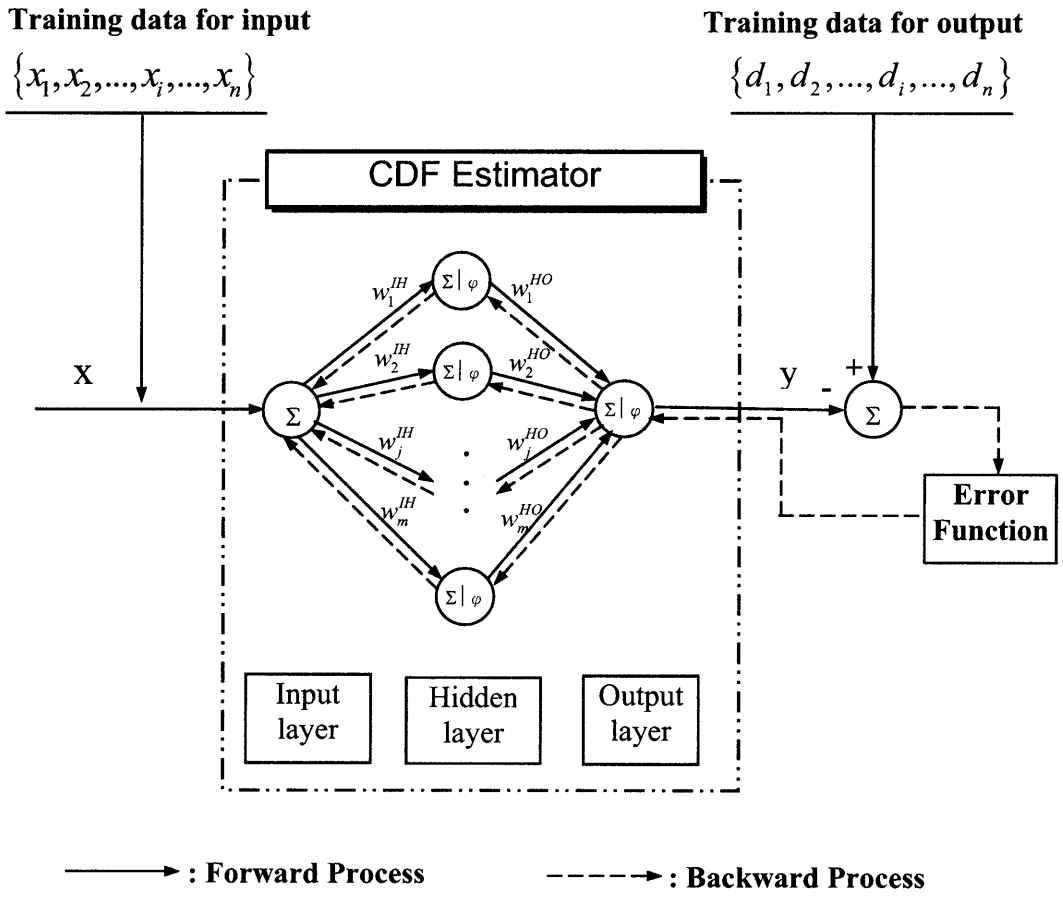


Figure 5-1 Back-propagation learning for CDF estimator

Error Function

The error function is also called the *cost function* or the *objective function*(Reed 1999). It is a measure of the difference between the actual output and the desired output. So it is

also a performance measure of the neural network. The network with a better performance makes the error smaller. A perfectly trained network has a zero error, although it can hardly be achieved in practice.

The most common error functions in training neural networks are *sum of squared error* (SSE) and *mean squared error* (MSE), which are defined respectively as,

$$E_{SSE} = \sum_{i=1}^{n}\sum_{l=1}^{k}(d_{il} - y_{il})^2 \tag{5.1}$$

$$E_{MSE} = \frac{1}{nk}E_{SSE} = \frac{1}{nk}\sum_{i=1}^{n}\sum_{l=1}^{k}(d_{il} - y_{il})^2 \tag{5.2}$$

where k is the neuron number in the output layer and n is the sample size. The sum of squared error is used for our CDF estimator. As there is only one output neuron in the estimator, the error function can be simplified as,

$$E_{SSE} = \sum_{i=1}^{n}E_i = \sum_{i=1}^{n}\frac{1}{2}(d_i - y_i)^2 \tag{5.3}$$

The factor is used for the convenience of differentiation in the error back-propagation process.

Optimize the Weights

Back-propagation learning is a non-linear optimization problem. The purpose is to find a set of weights for the neural network which minimize the error function, by adjusting the weights repeatedly in learning. In the optimization process, the error function is a function of the weights with the training data as the parameters. For our CDF estimator, it can be described as, (The biases can be looked as the special weights)

$$E_{SSE} = E_{SSE}(w_j^{IH}, w_j^{HO} \mid x_i, d_i) \tag{5.4}$$

The back-propagation learning uses the way of *gradient descent* with respect to the weights to find the directions to change the weights so as to decrease the value of the error function. The first step is to calculate the partial derivatives of the error with respect to the weights. For the weights of the synapses connected the output layer in the CDF estimator (See Figure 4-3), we have,

$$\frac{\partial E_{SSE}}{\partial w_j^{HO}} = \sum_{i=1}^{n} \frac{\partial E_i}{\partial w_j^{HO}} \tag{5.5}$$

The overall error is the sum of the independent individual errors for each training pattern so is the derivative. For simplicity, we are focusing on how to get the partial derivates on a single pattern. In the following, the i index for the pattern is omitted. E_i is denoted by E.

$$\frac{\partial E}{\partial w_j^{HO}} = \frac{\partial E}{\partial u} \cdot \frac{\partial u}{\partial w_j^{HO}} = \delta \cdot v_j \tag{5.6}$$

Here δ is the local gradient of the output neuron and v_j is the response of the jth neuron in the hidden layer.

$$\delta = \frac{\partial E}{\partial u} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} = (y - d) \cdot \varphi'(u) \tag{5.7}$$

where $\varphi'(u) = \dfrac{e^{-u}}{(1+e^{-u})^2}$ for the sigmoid activation function.

For the weights connecting the input layer to the hidden layer, we have,

$$\frac{\partial E}{\partial w_j^{IH}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w_j^{IH}} = \delta_j \cdot x \tag{5.8}$$

and,

$$\delta_j = \frac{\partial E}{\partial u_j} = \frac{\partial E}{\partial v_j} \cdot \frac{\partial v_j}{\partial u_j} = w_j^{HO} \delta \cdot \varphi'(u_j) \tag{5.9}$$

where $\varphi'(u_j) = \frac{e^{-u_j}}{(1+e^{-u_j})^2}$ for the sigmoid activation function.

Once we have the derivatives calculated, the weights can be updated. If the derivative is positive, the weight is decreased to make the error decreasing. If the derivative is negative, the weight is increased to keep the error decreasing. By applying this rule, the weight adjustment is given by

$$\Delta w_j^{HO} = -\eta \frac{\partial E}{\partial w_j^{HO}} \quad \text{and} \quad \Delta w_j^{IH} = -\eta \frac{\partial E}{\partial w_j^{IH}} \tag{5.10}$$

where η is called the *learning rate*, which is usually a small positive number. A larger learning rate makes a faster learning, but may cause the network oscillatory. A smaller learning rate generates a more stable network, but makes the learning slower. The typical range for the learning rate is $0.05 < \eta < 0.75$ (Reed 1999). Usually a momentum term is also added in order to increase the stability by considering the previous weight change. Now the formula is updated to,

$$\Delta w_j^{HO}(t) = -\eta \frac{\partial E}{\partial w_j^{HO}} + \alpha \Delta w_j^{HO}(t-1) \quad \text{and} \quad \Delta w_j^{IH}(t) = -\eta \frac{\partial E}{\partial w_j^{IH}} + \alpha \Delta w_j^{IH}(t-1) \tag{5.11}$$

Learning Modes

There are two common learning modes in practice, named the *on-line mode* and the *batch mode*. Before discussing these two modes, some definitions are necessary. An *iteration* is defined as the forward process and the backward process for an individual training pattern. An *epoch* is defined as one complete presentation of the entire training set(Haykin 1999).

The batch mode is to update the weights per epoch. In each iteration, the supposed weight change for each synapse on the current pattern is calculated and saved, but not applied to the weight right away. After an epoch is done, the sum of the weight changes for all patterns is applied to each weight.

The on-line mode, also called *sequential mode* or *pattern mode*, is to adjust the weights during each iteration. Unlike the batch mode, the on-line mode is not an exact implementation of the gradient descent way. However, in practice, the final effect is almost the same due to a large amount of epochs in the learning. In addition, the on-line mode has some advantages over the batch mode. The first one is that the on-line mode does n updates in the time the batch mode makes only one. This makes the on-line learning more stochastic and less likely to be trapped in a local minimum. The second one is that the on-line mode can make better use of the redundant data in the training set by updating the weights for each pattern.

In this thesis, the on-line mode is implemented considering the advantages described above, especially in order to take the second advantage because in Chapter 7 the training set for Bagging may have some redundant data.

Figure 5-2 is describing the back-propagation learning algorithm for our CDF estimator.

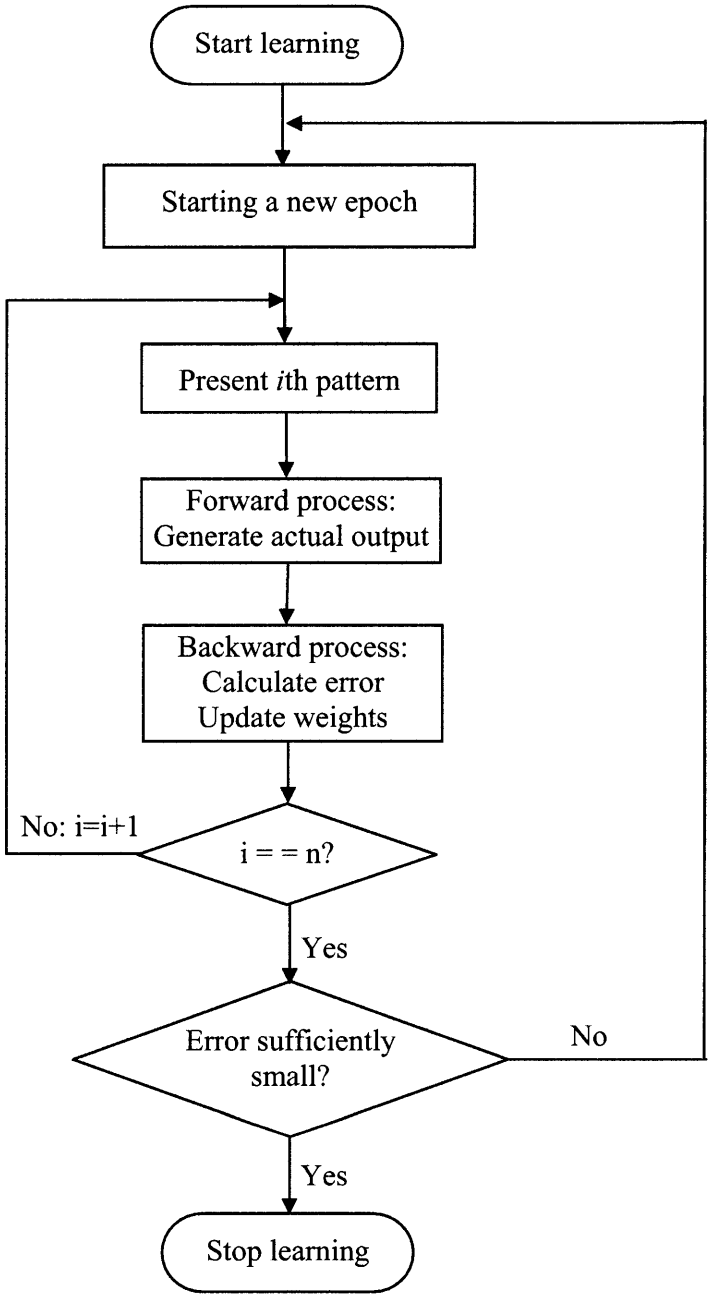


Figure 5-2 Back-propagation learning algorithm

5.2 Empirical CDF Training

So far, the structure of the CDF estimator has been built up and the learning algorithm has been designed. The next important work is to prepare the training data. If we look the CDF estimator as a CDF machine and the learning algorithm as the control program, the training data can be looked as the appropriate material so that the product, the CDF estimate, can be manufactured.

5.2.1 Empirical CDF

Suppose we have a random sample $\{x_1, x_2, ..., x_i, ..., x_n\}$ from the integrated simulation and the sample has been sorted so that it is in ascending order: $x_1 \le x_2 \le ... \le x_i \le ... \le x_n$. Then the *empirical distribution function* is defined as,

$$F_n(x) = \begin{cases} 0, & x < x_1 \\ i/n, & x_i \le x < x_{i+1} \\ 1, & x \ge x_n \end{cases} \tag{5.12}$$

The empirical distribution function is a consistent unbiased estimator of the cumulative distribution function. In another word, $F_n(x)$ converges to the underlying CDF $F_X(x)$ with n increasing. From the definition, it is obvious that an empirical CDF is a staircase function. Figure 5-3 is the empirical CDF derived from a N(0,1) sample of size 100.

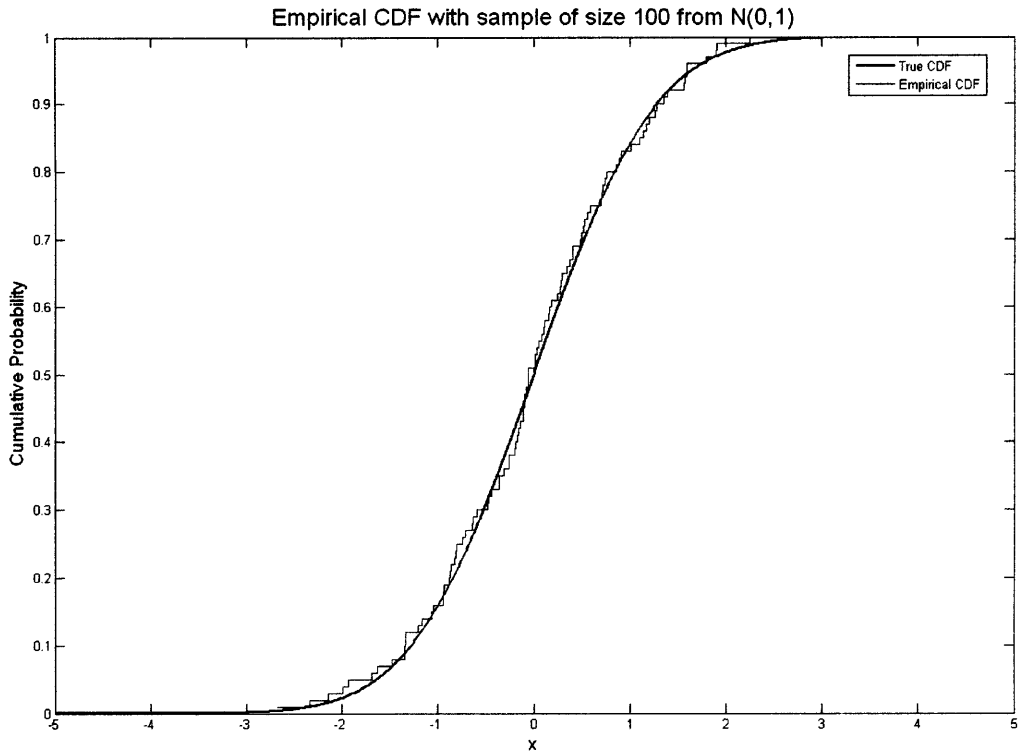


Figure 5-3 The empirical CDF based on a N(0,1) sample

5.2.2 NN-E Estimator

Although the empirical CDF is a consistent estimator of the underlying CDF, a PDF estimator can not be derived from it. This is because the empirical CDF is not differentiable. Also, it is not convincible that all x, between x_i and x_{i+1}, always have the same cumulative probabilities. Despite those problems, the empirical CDF definitely provides very important statistical information on the cumulative probabilities for the sample points. And those cumulative probabilities can be used as the training set for the output of our neural network CDF estimator. Following the definition of the empirical distribution function, we can get the desired outputs for the sample points, as what Figure 5-4 is describing.

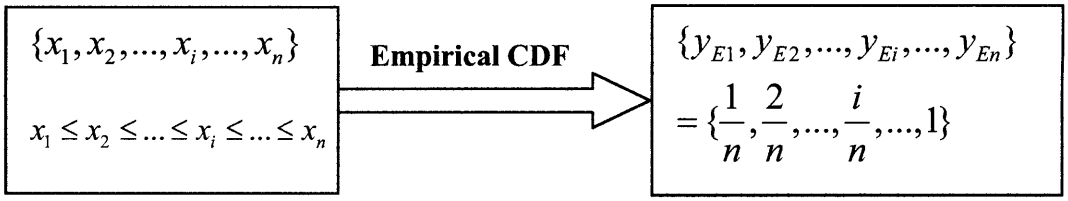


Figure 5-4 The training set derived from the empirical CDF

By back-propagation learning from the training set $\{x_1, x_2, ..., x_i, ..., x_n\}$ and $\{y_{E1}, y_{E2}, ..., y_{Ei}, ..., y_{En}\}$ (see Figure 5-1 and 5-2), the first version of neural network estimator for CDF can be attained. In this thesis, it is called NN-E estimator which means the **N**eural **N**etwork estimator trained by the **E**mpirical cumulative probabilities. The NN-E estimator for PDF can be derived by differentiation which is described in Section 4.2. The trained NN-E estimator for CDF can give the estimate for the underlying CDF by producing the cumulative probability estimate for any given x. It has the mathematical model described in the formula (4.13). It is the same with the NN-E estimator for PDF. It can give the estimate for the underlying PDF by producing the probability density estimate for any given x. It has the mathematical model described in the formula (4.19). Figure 5-5 is showing the CDF estimate given by the NN-E estimator for CDF which is trained by the sample of size 100 from N(0,1). Figure 5-6 is showing the corresponding PDF estimate given by the NN-E estimator for PDF.

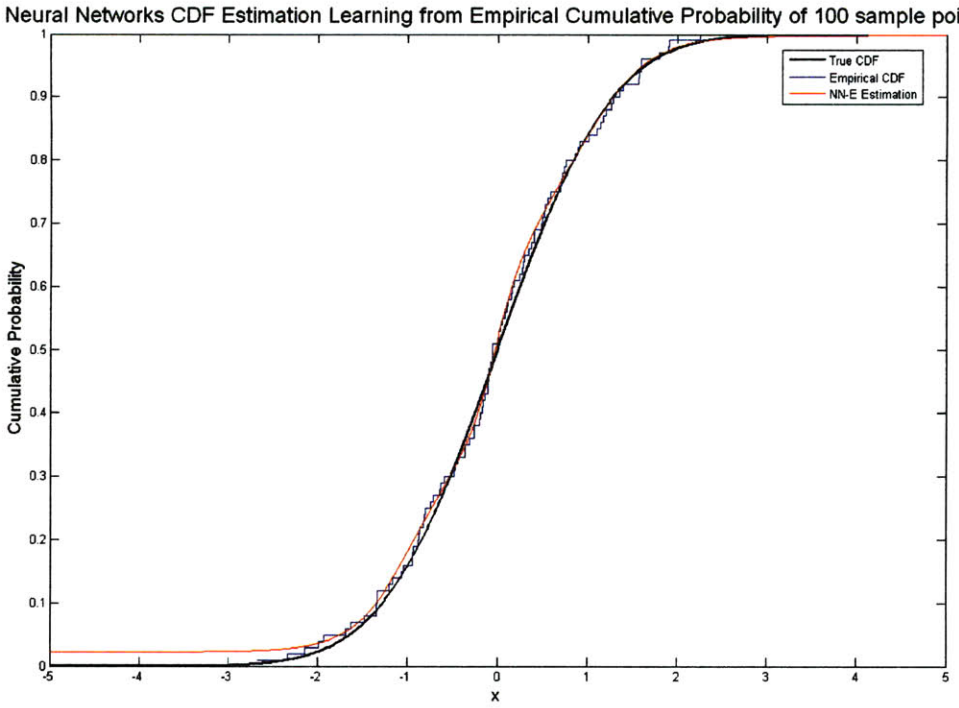


Figure 5-5 The NN-E estimate for CDF based on the sample from N(0,1)

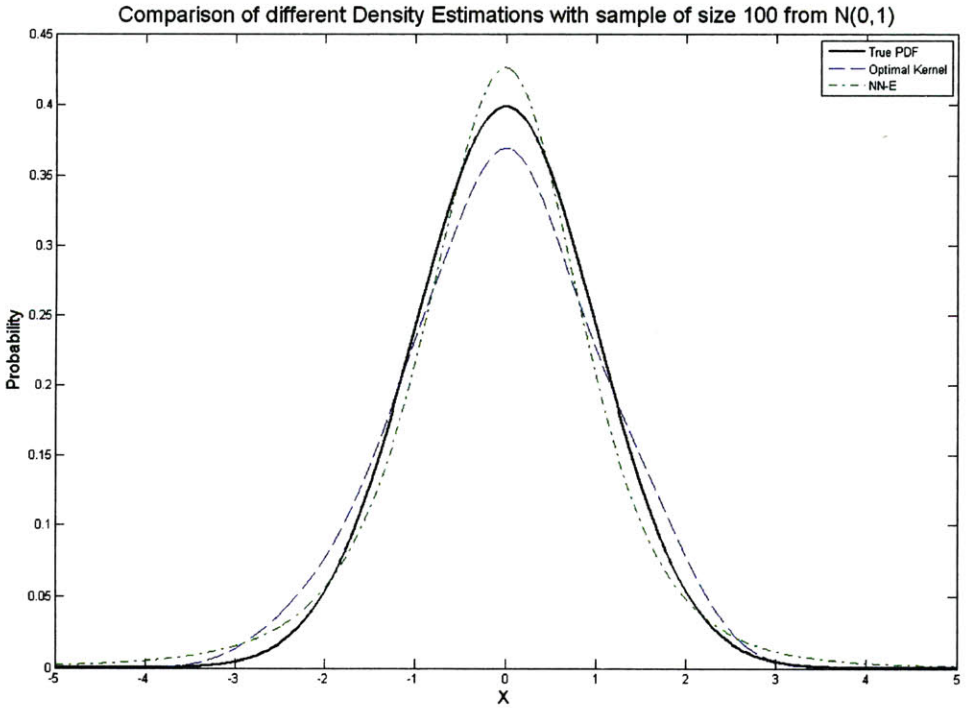


Figure 5-6 The NN-E estimate for PDF based on the sample from N(0,1)

In Figure 5-5, the kernel density estimation based on the same sample is also given for comparison. It can be seen that the NN-E estimate is closer to the true CDF. The estimation error is given in Table 5-1, from which we can see that the NN-E estimation gives smaller error than kernel density estimation.

Table 5-1 PDF estimation errors

	L1 Error	L2 Error
Kernel method	0.011900	2.831513e-004
NN-E estimation	0.010887	2.087665e-004

5.3 Combination of Statistical Information

5.3.1 Limitation of NN-E Estimator

Once the structure of the neural network estimator is created and the learning algorithm is developed, how well the trained neural network estimates the underlying CDF (and PDF) will be determined by how good the training set is. The more statistical information the training set is exposing, the better estimate the trained neural network estimator will finally make. It is found that the training set attained by the empirical CDF has some limitations.

Suppose we have a random sample coming from a probabilistic integrated simulation. It is called a *random* sample because the sample of each time is different, although all of those random samples are coming from the same underlying distribution. Then the situation, described in Figure 5-7, is coming up.

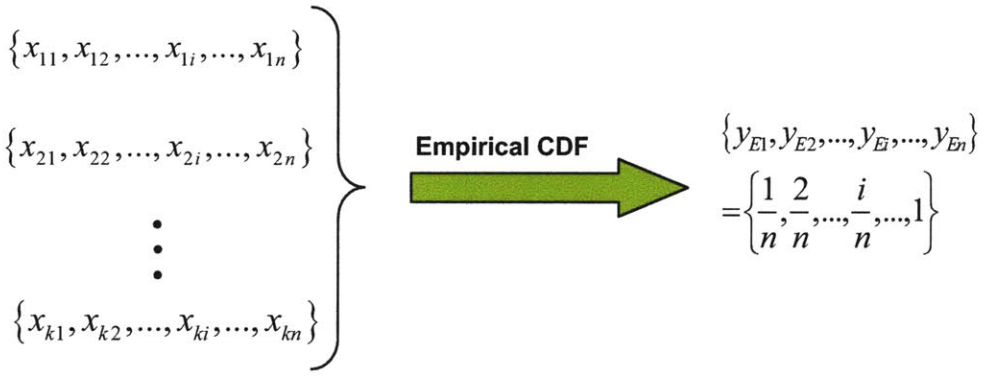


Figure 5-7 Different random samples yield the same set of empirical cumulative probabilities

In Figure 5-7, the different random samples are all sorted and in ascending order.(We assume this for the rest of the thesis.) By the definition of the empirical CDF, they always yield the same training set for the output of the neural network CDF estimator, which is the set of desired cumulative probabilities for the sample points. In another word, all the ith sample points in the different random samples, which are usually different values, always have the same cumulative probabilities. Apparently, this is not reasonable and can be improved.

The other issue with the training set attained by the empirical CDF is that, the maximum point in the sample always has cumulative probability 100%, which means it is also the maximum value of the random variable X, but this is not true obviously.

5.3.2 NN-EK Estimator

The kernel density estimation is a widely used nonparametric method in traditional statistics, despite the difficulty of finding an optimal bandwidth. It discovers the

statistical information, hidden in the random sample, on the underlying distribution in a different way from the empirical CDF. Both the empirical CDF and the kernel density estimation expose the useful statistical information from the different aspects. To some extent, the statistical information discovered by these two different approaches is complementary. By combining the statistical information from the empirical CDF and the kernel density estimation, an improved training set can be attained.

The kernel density estimation based on the sample $\{x_1, x_2, ..., x_i, ..., x_n\}$ is given in the formula (3.12) as,

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(\frac{x - x_i}{h})$$

In this thesis, Gaussian function is used as the kernel function. The optimal bandwidth h is determined by the plug-in method(Wand 1995).

By integral we can calculate the cumulative probability for each sample point:

$$y_{Ki} = \hat{F}(x_i) = \int_{-\infty}^{x_i} \hat{f}_h(x)dx = \int_{-\infty}^{x_i} \frac{1}{n}\sum_{i=1}^{n} K_h(\frac{x - x_i}{h})dx \qquad (5.13)$$

Then we have the training set $\{y_{K1}, y_{K2}, ..., y_{Ki}, ..., y_{Kn}\}$ from the kernel method. We do a combination by averaging the training sets got by the empirical CDF and the kernel density estimation, and get an improved training set:

$$\begin{aligned} &\{y_{EK1}, y_{EK2}, ..., y_{EKi}, ..., y_{EKn}\} \\ &= \left\{\frac{(y_{E1} + y_{K1})}{2}, \frac{(y_{E2} + y_{K2})}{2}, ..., \frac{(y_{Ei} + y_{Ki})}{2}, ..., \frac{(y_{En} + y_{Kn})}{2}\right\} \end{aligned} \qquad (5.14)$$

The new training set $\{y_{EK1}, y_{EK2}, ..., y_{EKi}, ..., y_{EKn}\}$ shows its power immediately by getting rid of the limitations described in previous section. Now for a different random sample, a

different $\{y_{EK1}, y_{EK2}, ..., y_{EKi}, ..., y_{EKn}\}$ can be attained since the kernel estimation has different value for the different sample point. And y_{EKn} is not equal to 1 any more.

By back-propagation learning from the training set $\{x_1, x_2, ..., x_i, ..., x_n\}$ and $\{y_{EK1}, y_{EK2}, ..., y_{EKi}, ..., y_{EKn}\}$ (see Figure 5-1 and 5-2), the neural network estimator for CDF can be attained. In this thesis, it is called NN-EK estimator which means the **N**eural **N**etwork estimator trained by the **E**mpirical and **K**ernel cumulative probabilities. The NN-EK estimator for PDF can be derived by differentiation which is described in Section 4.2. The NN-EK estimators for CDF and PDF have the mathematical models described in the formula (4.13) and (4.19) respectively. Figure 5-8 is showing the CDF estimate given by the NN-EK estimator for CDF which is trained by the sample of size 100 from N(0,1). Figure 5-9 is showing the corresponding PDF estimate given by the NN-EK estimator for PDF.

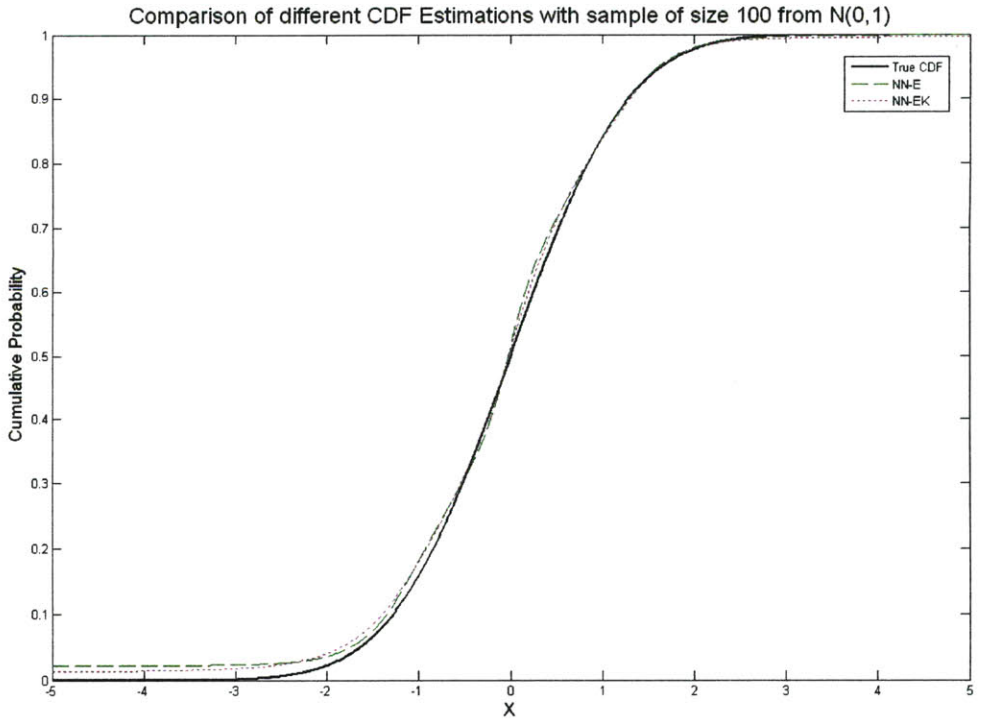


Figure 5-8 The NN-EK estimate for CDF based on the sample from N(0,1)

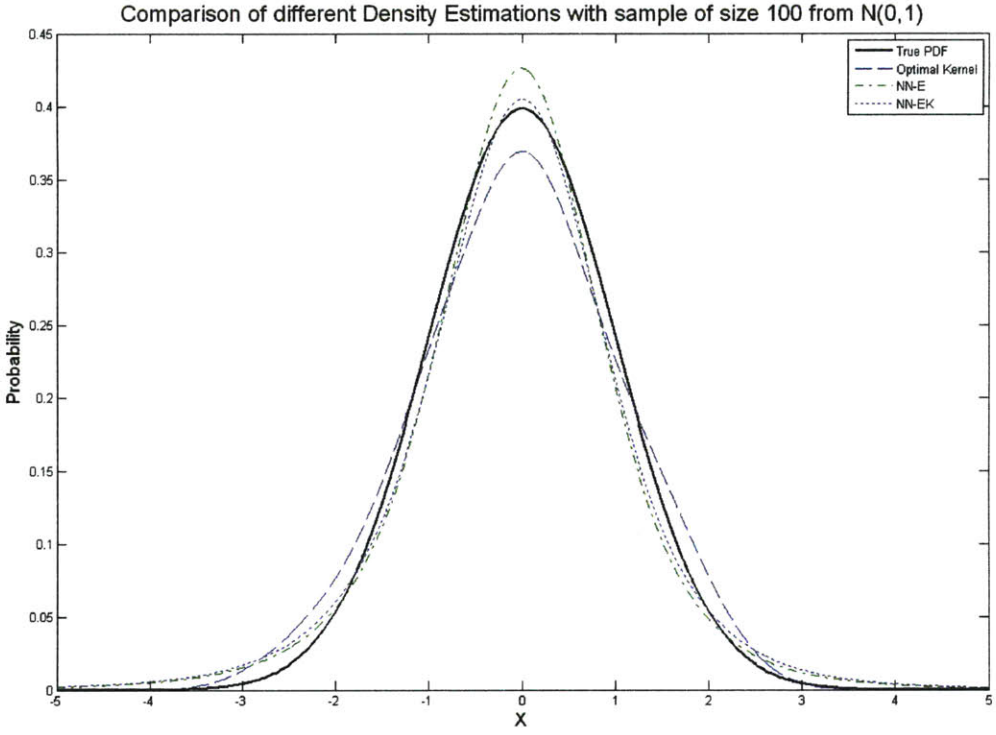


Figure 5-9 The NN-EK estimate for PDF based on the sample from N(0,1)

In Figure 5-5, the kernel density estimation and the NN-E estimation based on the same sample is also given for comparison. It can be seen that the NN-EK estimate is closer to the true CDF than those given by the kernel method with an optimal bandwidth and the NN-E estimator. This is because the training set used by the NN-EK estimator contains more statistical information than that of either the NN-E estimator or the kernel density estimation. The estimation error is given in Table 5-2, from which we can see that the NN-E estimation gives smaller error than kernel density estimation.

Table 5-2 PDF estimation errors

	L1 Error	L2 Error
Kernel method	0.011900	2.831513e-004

NN-E estimation	0.010887	2.087665e-004
NN-EK estimation	0.009938	1.579242e-004

Chapter 6

Learning from Hints

6.1 Hints

In the field of machine learning, a function learning or approximation is usually accomplished by learning from a set of input-output examples. Hints here mean prior knowledge of certain facts about the unknown function, in addition to the set of examples(Abu-Mostafa 1993). Hints are usually known as the properties of the target function which are independent of the training examples(Abu-Mostafa 1995). It is very important to incorporate hints into the learning process because of a few benefits. The first one is to get a more accurate function estimation. The more information used in the learning, the closer to the true function the estimation is. The learning benefits the most from hints in the situation where the training examples are limited due to intensive computation. This is exactly our case. The second benefit to integrate hints into learning is to improve the learning efficiency. With the guideline of hints, the learning can go a shorter way to the goal.

There are different types of hints in leaning a function(Abu-Mostafa 1990; Abu-Mostafa 1993; Abu-Mostafa 1995). For example, for an *odd* function hint, we have:

$$f(-x_i) = -f(x_i)$$

for each sample point. The *invariance hint* asserts

$$f(x_i) = f(x_j)$$

for certain pairs of x_i and x_j. The *approximation hint* asserts for certain points x_i that $f(x_i) \in [a,b]$.

There are different ways to incorporate hints:

(1) Reprocess the training examples in order to contain the hint, for example, add some *virtual examples* to the original training examples.

(2) Customize the learning model so as to reflect the hint, for example, make a neural network of a particular structure

(3) Change the learning algorithm to follow the guideline of the hint.

The implementation depends on the hint and the learning process.

6.2 Hints in Distribution Estimation

The purpose of distribution estimation is to learn the cumulative distribution function and the probability density function from the sample. For the cumulative distribution function, it has the *monotonicity hint* since the function is always monotonically nondecreasing, just like the example in Figure 6-1. And for the probability density function, it has the *nonnegative hint*, like the example showing in Figure 6-2. Accomplishment of either of these two hints always guarantees the other one since the PDF is just the differentiation of the CDF. In this thesis, the monotonicity hint is incorporated into the CDF learning process. Then the PDF estimation derived from the CDF estimation automatically satisfies the nonnegative hint.

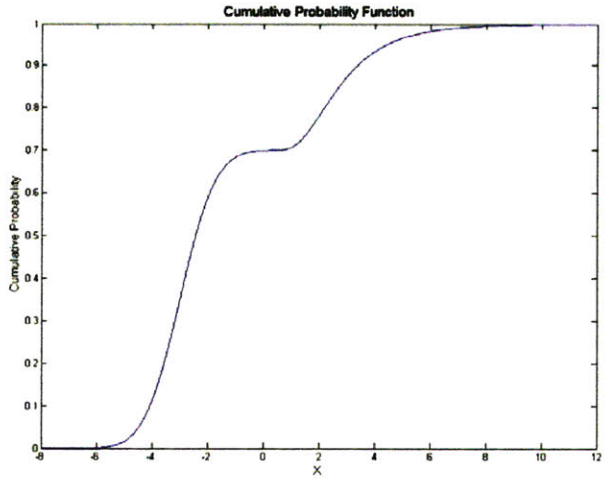


Figure 6-1 Monotonicity hint for CDF

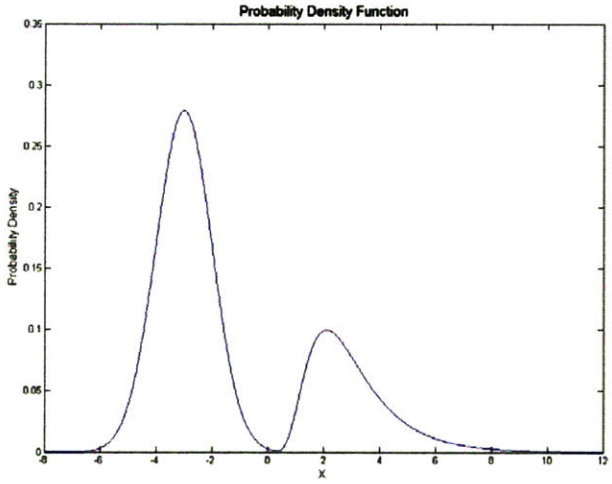


Figure 6-2 Nonnegative hint for PDF

6.3 Incorporation of Monotonicity Hint

6.3.1 Monotonicity Hint Penalty Term

Suppose we have a random sample $\{x_1, x_2, ..., x_i, ..., x_n\}$ which is already sorted and in ascending order. $\{d_1, d_2, ..., d_i, ..., d_n\}$ is the training set for the output of the neural

network CDF estimator. For NN-E estimator, $\{d_1, d_2, ..., d_i, ..., d_n\}$ becomes $\{y_{E1}, y_{E2}, ..., y_{Ei}, ..., y_{En}\}$. And for NN-EK estimator, it is $\{y_{EK1}, y_{EK2}, ..., y_{EKi}, ..., y_{EKn}\}$. Both of these two training sets satisfy $d_1 \le d_2 \le ... \le d_i \le ... \le d_n$ according to the description of the training set generation in Chapter 5. Suppose the actual outputs of the neural network estimator for the sample points are $\{y_1, y_2, ..., y_i, ..., y_n\} = \{y(x_1, w), y(x_2, w), ..., y(x_i, w), ..., y(x_n, w)\}$. If the neural network is perfectly trained, that means the error between $\{d_1, d_2, ..., d_i, ..., d_n\}$ and $\{y_1, y_2, ..., y_i, ..., y_n\}$ is minimized to zero, the monotonicity hint is automatically implemented. However, this is impossible for most cases in practice. Then incorporation of the monotonicity hint basically means that the actual outputs should satisfy $y_1 \le y_2 \le ... \le y_i \le ... \le y_n$.

Recall the error function defined in the formula (5.3) for the back-propagation learning,

$$E_{SSE} = \sum_{i=1}^{n} E_i = \sum_{i=1}^{n} \frac{1}{2}(d_i - y_i)^2$$

To integrate the monotonicity hint into the back-propagation learning, a penalty term is added into the error function, which becomes,

$$E_i = \frac{1}{2}\big(d_i - y(x_i, w)\big)^2 + U\big(y(x_i, w) - y(x_{i+1}, w)\big) \cdot \frac{1}{2}\big(y(x_i, w) - y(x_{i+1}, w)\big)^2 \qquad (6.1)$$

where the unit step function is defined as

$$U\big(y(x_i, w) - y(x_{i+1}, w)\big) = \begin{cases} 1 & y(x_i, w) - y(x_{i+1}, w) > 0 \\ 0 & y(x_i, w) - y(x_{i+1}, w) \le 0 \end{cases} \qquad (6.2)$$

Basically this can be explained as the following. If $y(x_i, w) - y(x_{i+1}, w) > 0$, which is against the monotonicity hint, the penalty term $\frac{1}{2}\left(y(x_i, w) - y(x_{i+1}, w)\right)^2$ shows up in the error function. This term is minimized in back-propagation learning until $y(x_i, w) - y(x_{i+1}, w) \leq 0$. In this way, the monotonicity hint is implemented by achieving $y_1 \leq y_2 \leq \ldots \leq y_i \leq \ldots \leq y_n$.

6.3.2 Hint-reinforced Training Set

Although $y_1 \leq y_2 \leq \ldots \leq y_i \leq \ldots \leq y_n$ only guarantees the monotonicity on the sample points, it usually can attain the monotonicity on the whole domain in practice. However, for some cases, it is not very satisfactory. The Figure 6-3 is the CDF estimate based on the random sample from a distribution which is the mixture of a normal distribution and a log normal distribution, shown in Figure 6-2. It is the NN-EK estimation, with the monotonicity hint incorporated in the way described above. It can be seen that the circled part is not monotonically nondecreasing. As a result, the derived PDF estimate has a negative part, which is shown in Figure 6-4.

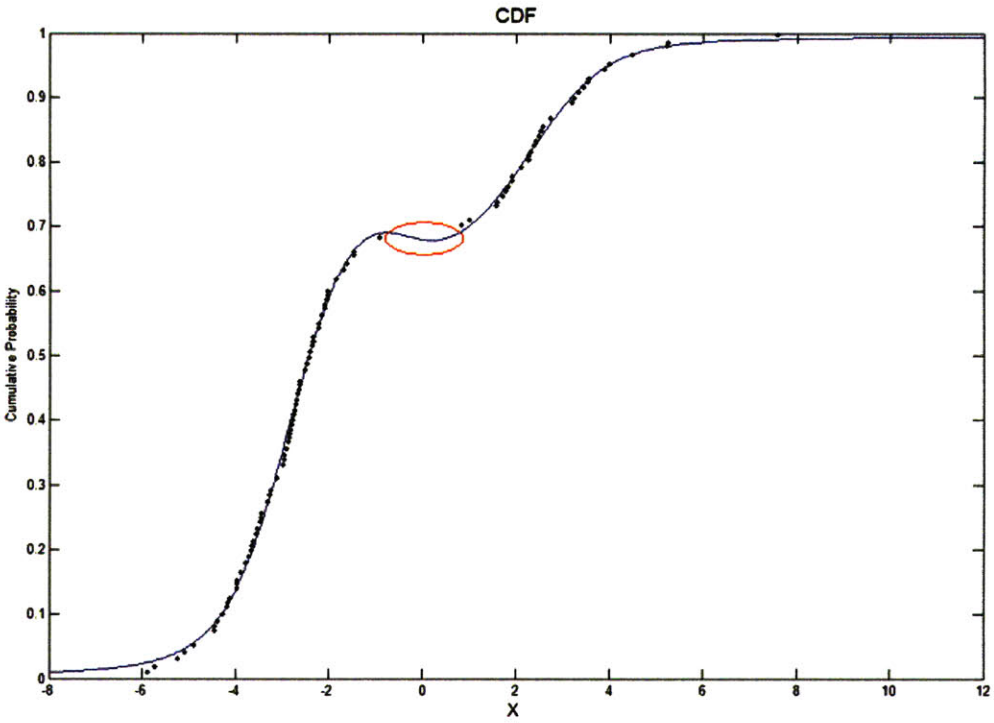


Figure 6-3 CDF estimate with a part not monotonically nondecreasing

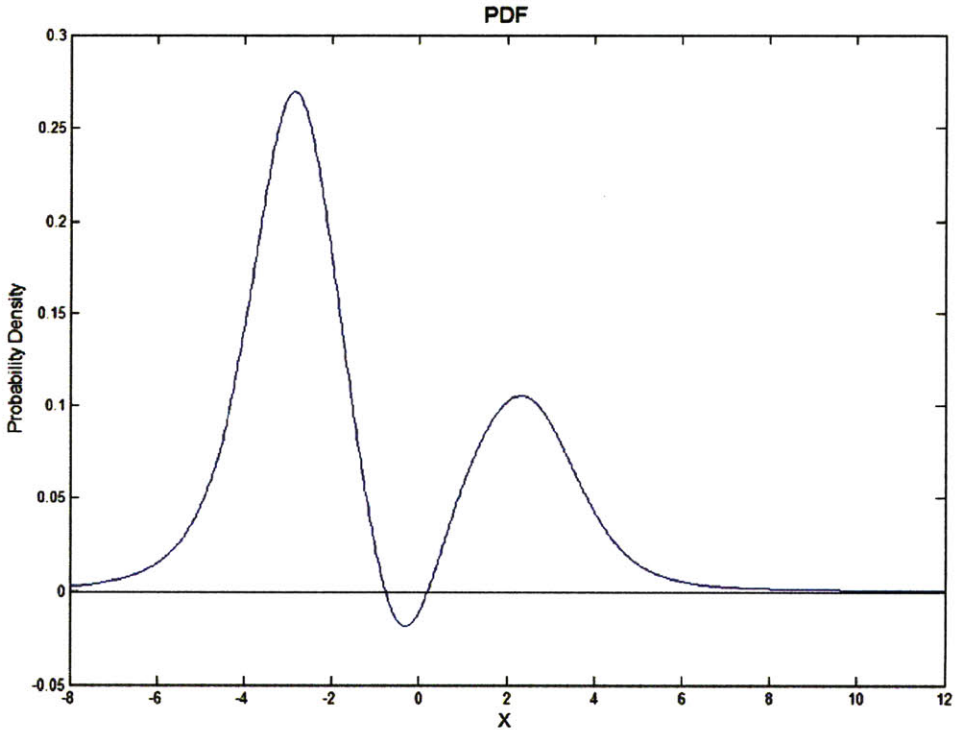


Figure 6-4 PDF estimate with a negative part

The reason is that the penalty term in the error function (6.1) only guarantees the monnotonicity on the sample points. This is not enough for the distribution in Figure 6-2. Actually, this usually happens to a distribution with multiple modes which has a low probability area between the modes. If a sample from the distribution has a limited size, there is usually no sample point from that area, just like the area indicated by a circle in Figure 6-3. Such a part is usually a big gap between the two adjacent sample points. The monotonicity can not be implemented on the whole part only by implementing the monotonicity on the two adjacent sample points.

The way to solve this problem is to regenerate a hint-reinforced training set based on the original training set $\{x_1, x_2, ..., x_i, ..., x_n\}$ and $\{d_1, d_2, ..., d_i, ..., d_n\}$. First we add m hint-only sample points into the original sample. These *virtual sample points* can be picked up evenly from the problem area. Their desired outputs are zeros and are only used to indicate in the learning process that they are hint-only sample points. Then sort the sample again and the new hint-reinforced training set for the input can be got as $\{x_1, x_2, ..., x_i, ..., x_n, ..., x_{n+m}\}$. The corresponding training set for the output is $\{d_1, d_2, ..., d_i, ..., d_n, ..., d_{n+m}\}$ which contains zeros for the hint-only sample points.

In back-propagation learning, a sample point is detected as a virtual sample point if its desired output is found to be zero, and the error function is given as,

$$E_i = U\left(y(x_i, w) - y(x_{i+1}, w)\right) \cdot \frac{1}{2}\left(y(x_i, w) - y(x_{i+1}, w)\right)^2 \tag{6.3}$$

Otherwise, it is an original sample point, and the error function is given as the same with the formula (6.1),

$$E_i = \frac{1}{2}\left(d_i - y(x_i, w)\right)^2 + U\left(y(x_i, w) - y(x_{i+1}, w)\right) \cdot \frac{1}{2}\left(y(x_i, w) - y(x_{i+1}, w)\right)^2$$

The Figure 6-5 is the CDF estimate based on the random sample from a distribution which is the mixture of a normal distribution and a log normal distribution, shown in Figure 6-2. It is the NN-EK estimation, incorporating the monotonicity hint by using the hint-reinforced training set. It can be seen that the problem part in Figure 6-3 is now monotonically nondecreasing. The red dots in Figure 6-5 are those hint-only sample points added to the original sample points. As a result, the derived PDF estimate is nonnegative over the whole domain, which is shown in Figure 6-6.

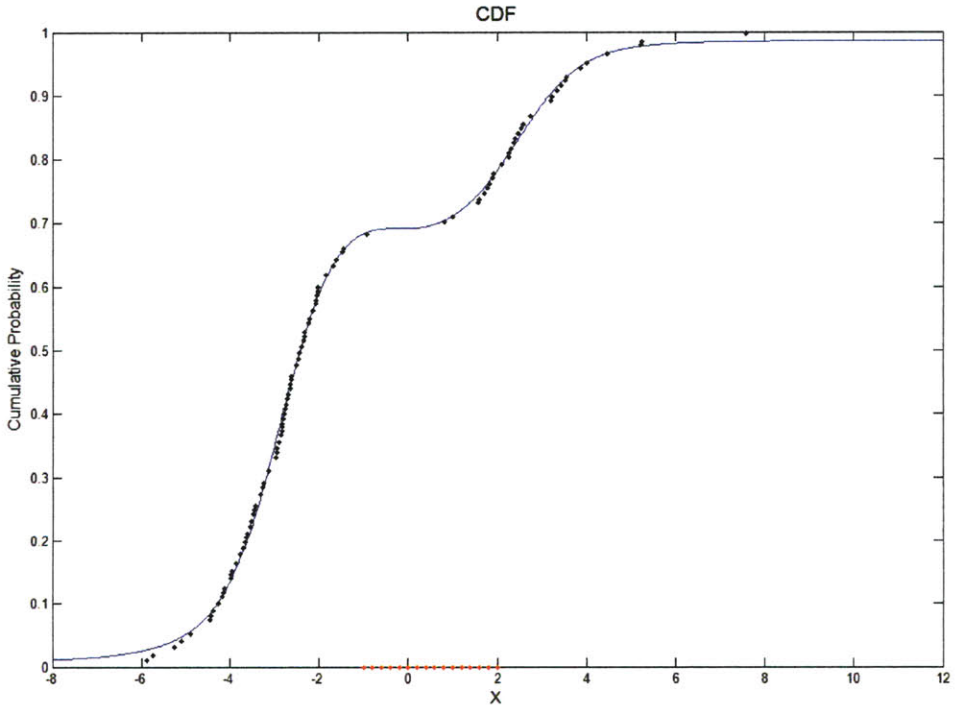


Figure 6-5 CDF estimate by learning from the hint-reinforced training set

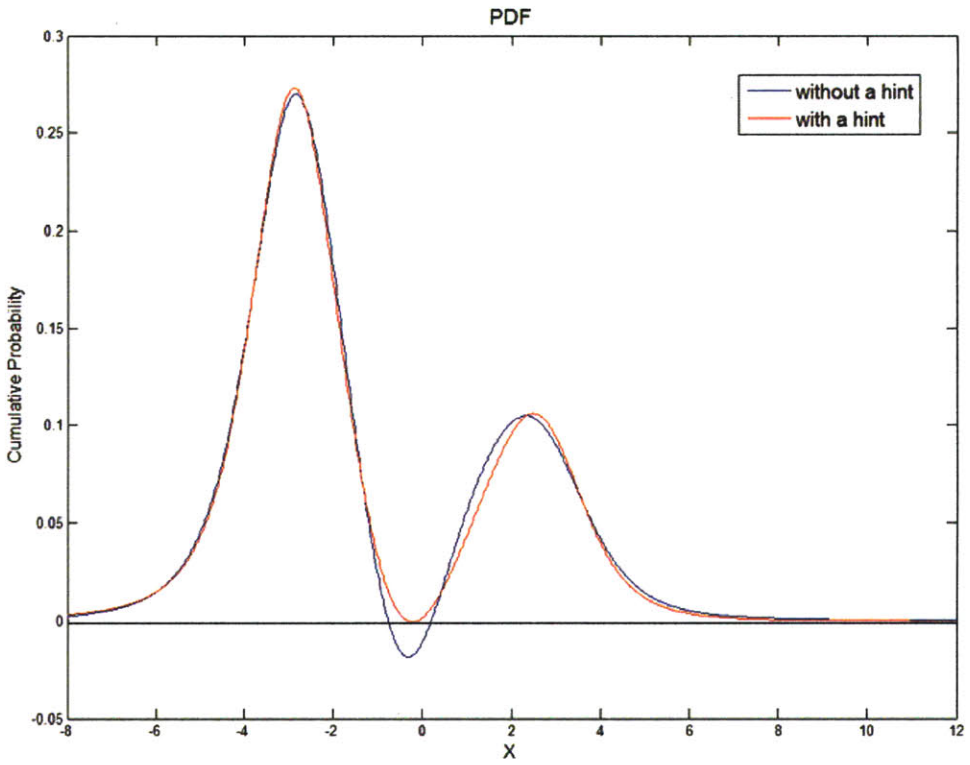


Figure 6-6 PDF estimate by learning from the hint-reinforced training set

Chapter 7

Bagging Estimators

This chapter is discussing how to use the statistical method named Bagging to gain more improvement on our neural network estimator. Bagging, the abbreviation of "bootstrap aggregating", generates multiple versions of an estimator and aggregates them to get a new estimator(Breiman 1996). The multiple versions are attained by using bootstrap resampling technique to generate a bootstrap samples from the original sample for each version and using it as new training set.

7.1 Bootstrap

The bootstrap is a resampling method for statistical inference(Efron 1979; Efron 1993; Davison 1997; Chernick 1999). Suppose we have an original random sample $\{x_1, x_2, ..., x_i, ..., x_n\}$. The bootstrap is resampling of these n sample points x_i. It generates a new sample of size n by random sampling of the original sample. During the sampling, the probability that a sample point is picked is considered as $1/n$. The sample points in the new bootstrap sample are drawn from the original sample at random *with replacement*. As a result, a sample point in the original sample may appear more than one time or not at all in the new bootstrap sample.

The underlying statistical motivation for the bootstrap can be explained in the following way:

1) The underlying distribution function $F_X(x)$ for the sample $\{x_1, x_2, ..., x_i, ..., x_n\}$ is unknown and the empirical CDF $F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I(x \geq x_i)$ is used for the estimate of $F_X(x)$.
2) Use $F_n(x)$ as the original distribution $F_X(x)$.
3) Sampling from $F_n(x)$ is equivalent to sampling with replacement from the original sample $\{x_1, x_2, ..., x_i, ..., x_n\}$.

The bootstrap is often used to get standard error and confidence intervals for the parameter estimates. In bagging, the bootstrap is applied to generate multiple training sets so that multiple versions of the neural network estimator can be attained.

7.2 Bagging NN-EK Estimators

Suppose we have a sample S_0 of size n, i.e. consisting of n sample points. Based on this sample, we have some estimator $\hat{\varphi}(x, S_0)$ by which we estimate y for the input x. Suppose we are given more samples, denoted by a sequence $\{S_k\}$. In this sequence, each sample is from the same distribution with S_0 and has a size n. With the sequence $\{S_k\}$, we want to get a better estimator than $\hat{\varphi}(x, S_0)$. The most straightforward way is to form an estimator $\hat{\varphi}(x, S_j)$ on each sample, and then aggregate these estimators by averaging over k to get a new estimator(Breiman 1996):

$$\hat{\varphi}(x) = \frac{1}{k}\sum_{j=1}^{k}\hat{\varphi}(x, S_j) \tag{7.1}$$

However, usually, it is hard to attain the sample sequence $\{S_k\}$ due to the limitation of sample acquirement. In our probabilistic simulation for the integrated system, usually there is only one sample $S_0 = \{x_1, x_2, ..., x_i, ..., x_n\}$ is available due to the cost of computation time. Under such a situation, the bootstrap can make an imitation of the process above. By the bootstrap resampling of $S_0 = \{x_1, x_2, ..., x_i, ..., x_n\}$, we can get a sequence of bootstrap samples $\{S_k^B\}$. Each bootstrap sample S_j^B in the sequence, i.e. $\{x_1^j, x_2^j, ..., x_i^j, ..., x_n^j\}$, is the input training set for our NN-EK estimator. By the means discussed in Chapter 5, we can derive the output training set for the NN-EK estimator, which is $D_j^B = \{y_{EK1}^j, y_{EK2}^j, ..., y_{EKi}^j, ..., y_{EKn}^j\}$. By back-propagation learning from S_j^B and D_j^B, we can attain the NN-EK CDF estimator $\hat{F}_j(x)$. By differentiation of $\hat{F}_j(x)$, the NN-EK PDF estimator $\hat{f}_j(x)$ can be derived. Finally, we can have the bagging CDF estimator and the bagging PDF estimator by aggregation respectively,

$$\hat{F}_B(x) = \frac{1}{k}\sum_{j=1}^{k}\hat{F}_j(x) \tag{7.2}$$

$$\hat{f}_B(x) = \frac{1}{k}\sum_{j=1}^{k}\hat{f}_j(x) \tag{7.3}$$

For distribution estimation, $k = 30 \sim 50$ is thought reasonable(Breiman 1996). Figure 7-1 is showing the whole process of bagging NN-EK estimators.

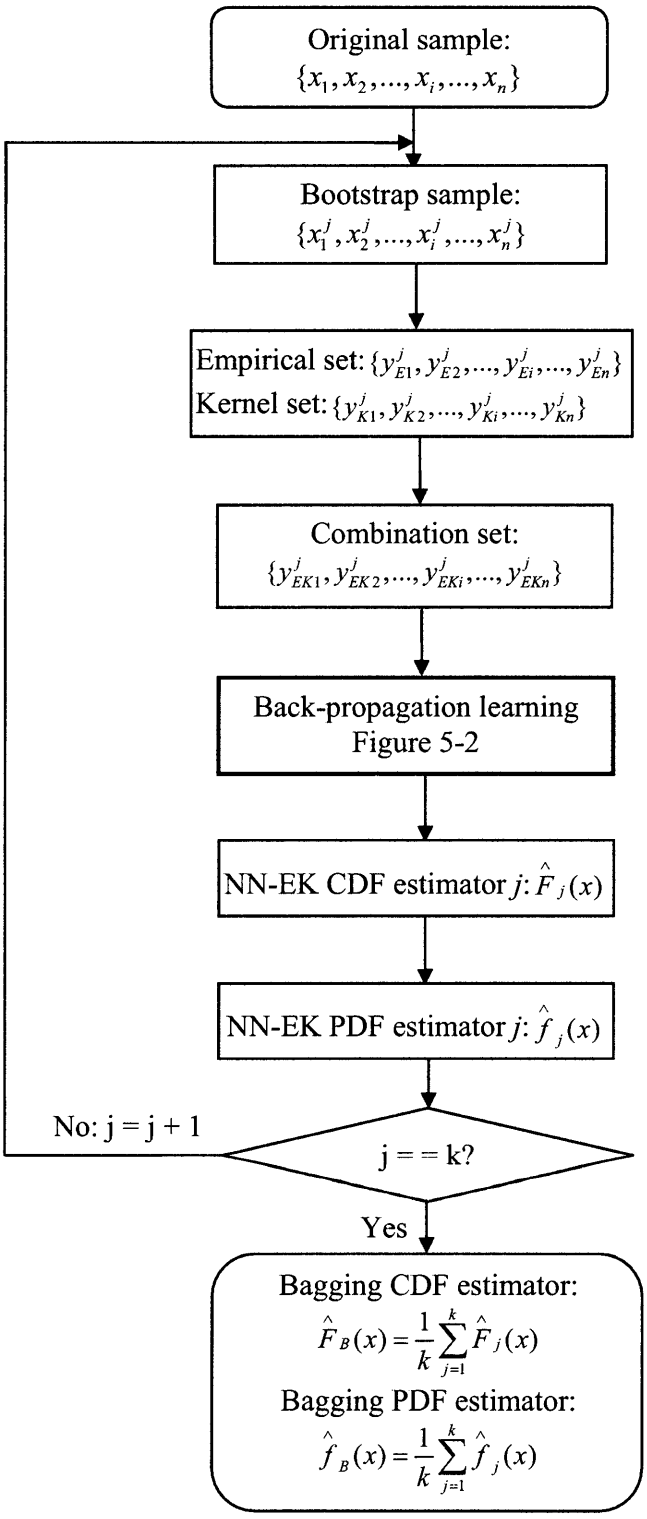


Figure 7-1 Bagging NN-EK estimators

It has been proved by Breiman(Breiman 1996) that bagging method can gain obvious improvement on those procedures involved with neural nets, classification and regression trees. Figure 7-2 is showing the CDF estimate for the distribution N(0,1), given by bagging NN-EK estimators based on the sample of size 100. Compared with the single NN-EK estimator, the bagging estimator can give more accurate estimate. The estimation errors are given in Table 7-1. Figure 7-3 and Table 7-2 are PDF estimation results.

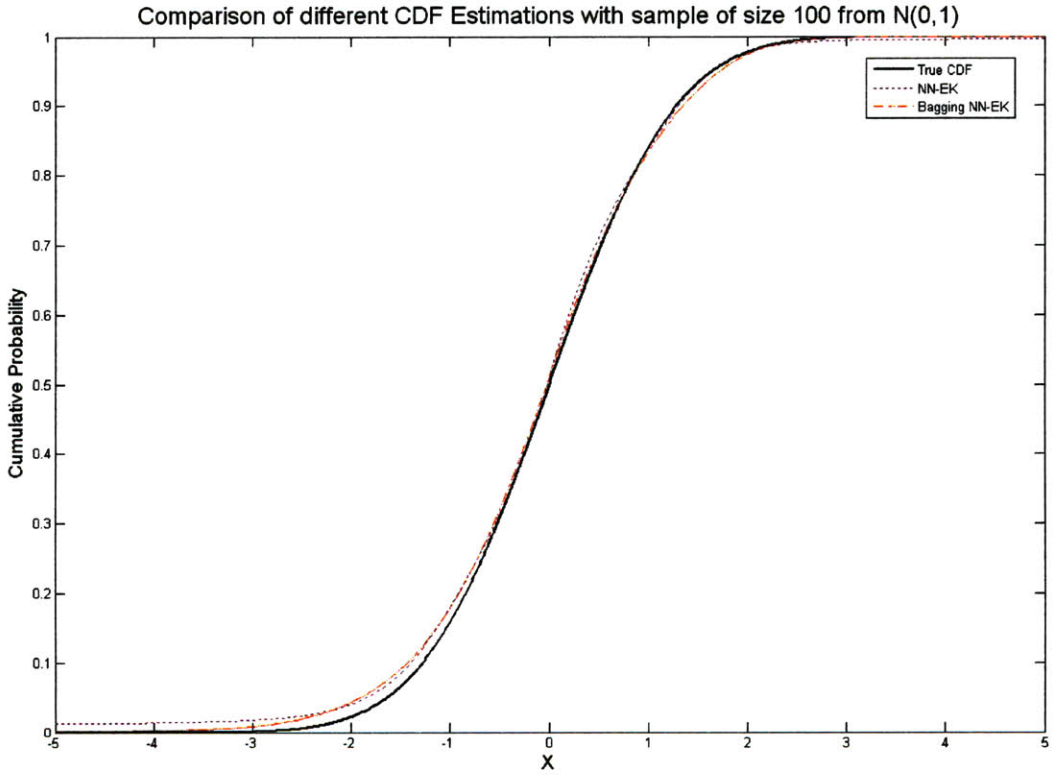


Figure 7-2 The bagging NN-EK CDF estimate based on the sample from N(0,1)

Table 7-1 Estimation errors

	L1 Error	L2 Error
NN-EK estimation	0.010228	1.539430e-004
Bagging NN-EK estimation	0.007525	1.122562e-004

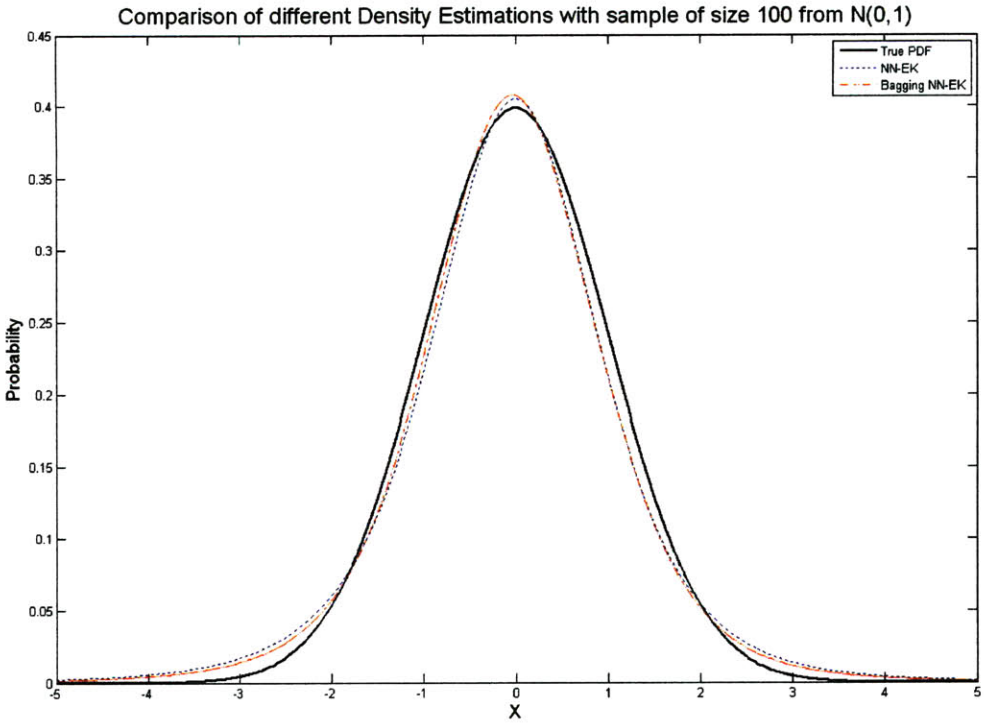


Figure 7-3 The bagging NN-EK PDF estimate based on the sample from N(0,1)

Table 7-2 Estimation errors

	L1 Error	L2 Error
NN-EK estimation	0.009938	1.579242e-004
Bagging NN-EK estimation	0.008312	1.208977e-004

Chapter 8

Case Studies

8.1 Case Study 1: Skew Model

8.1.1 Model Description

Skewness in output distributions can often be found in integrated probabilistic simulations. So firstly we had our approach tested on skew models. The skew model tested here has a logarithmic normal distribution, which is shown in Figure 8-1.

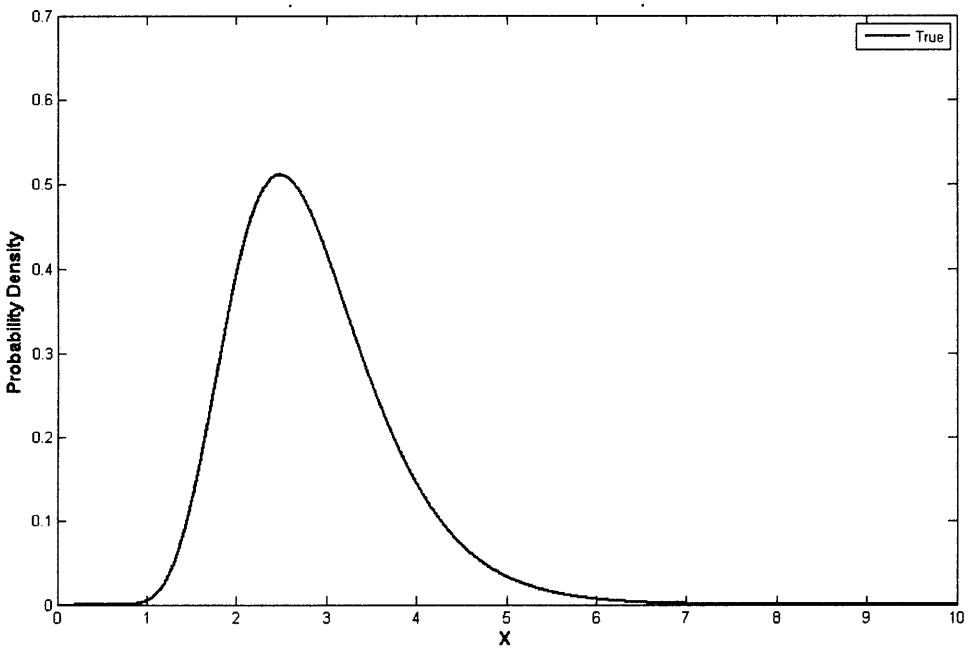


Figure 8-1 Lognormal(1.0, 0.3) model

8.1.2 Estimation by Different Methods

All estimations are based on the random sample of size 100.

1. NN-E Estimation

The NN-E estimate for PDF of lognormal(1.0, 0.3) model is shown in Figure 8-2.

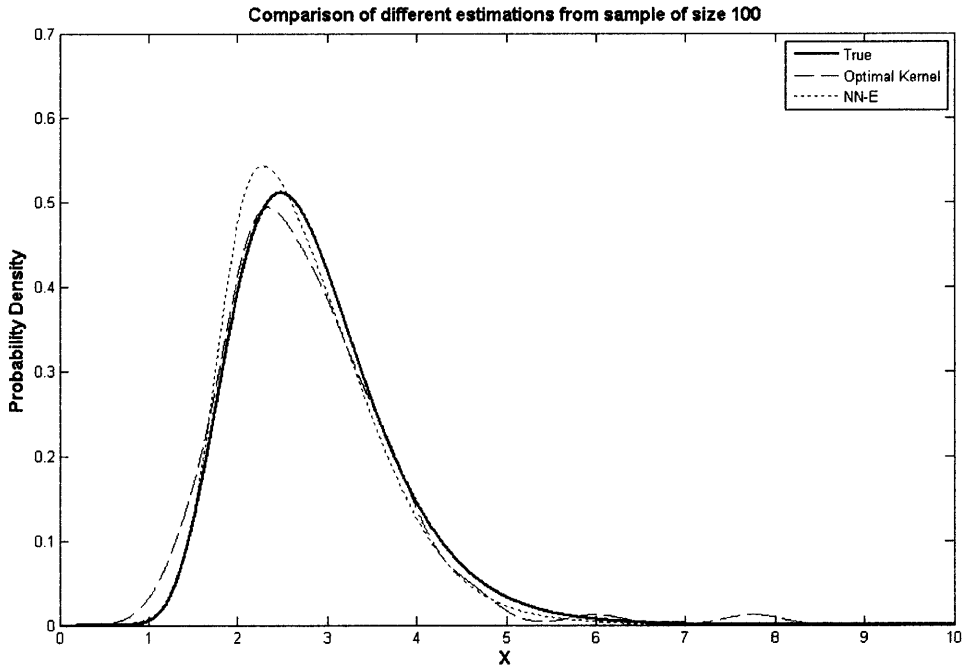


Figure 8-2 NN-E estimate for PDF of lognormal(1.0, 0.3) model

In Figure 8-2, the kernel estimate with an optimal bandwidth is also shown for the purpose of comparison. We can see that the NN-E estimation is obviously better than the optimal kernel estimation. The PDF estimation errors are given in Table 8-1.

Table 8-1 PDF estimation errors for lognormal(1.0, 0.3) model

	L1 Error	RMSE
Kernel estimation	0.011363	0.02014
NN-E estimation	0.010410	0.01729

2. NN-EK Estimation

The NN-EK estimate for PDF of lognormal(1.0, 0.3) model compared with the kernel estimate and NN-E estimate is shown in Figure 8-3.

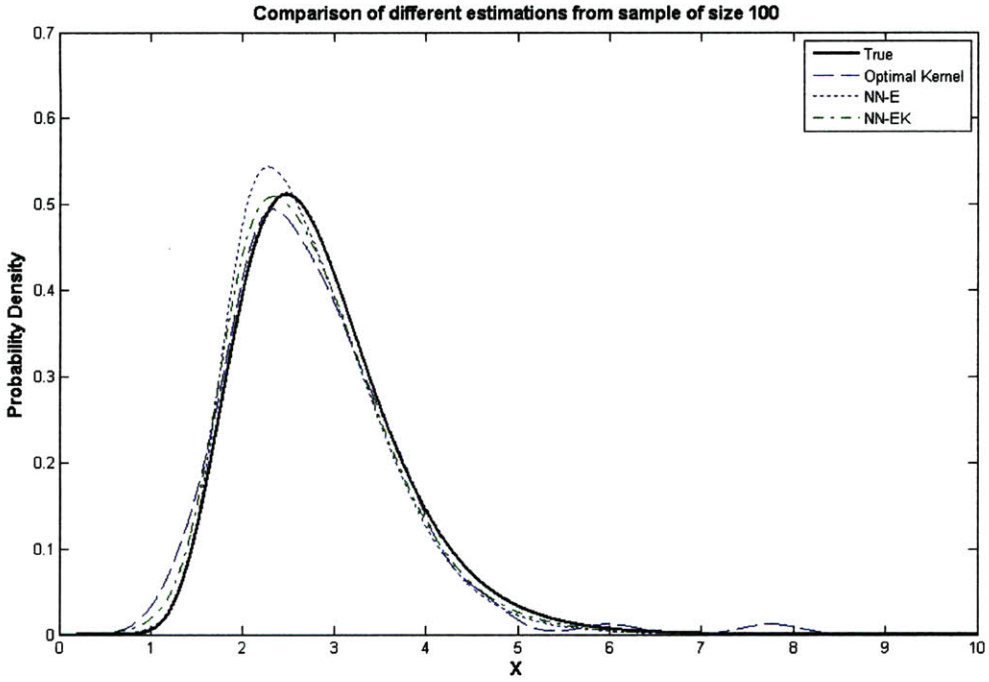


Figure 8-3 NN-EK estimate for PDF of lognormal(1.0, 0.3) model

In Figure 8-3, we can see that the NN-EK estimation is obviously better than the NN-E estimation. The PDF estimation errors are given in Table 8-2.

Table 8-2 estimation errors for lognormal(1.0, 0.3) model

	L1 Error	RMSE
Kernel estimation	0.011363	0.02014
NN-E estimation	0.010410	0.01729
NN-EK estimation	0.009084	0.01502

3. Bagging NN-EK Estimation

In Bagging NN-EK estimation, 30 bootstrap replicates are used. Figure 8-4 is showing the Bagging NN-EK estimate for PDF of lognormal(1.0, 0.3) model. The PDF estimation errors are given in Table 8-3. It can be seen that the Bagging NN-EK estimation gains obvious improvement on the single NN-EK estimation and achieves 40.99%(L1 error) and 31.23%(RMSE) on the optimal kernel estimation.

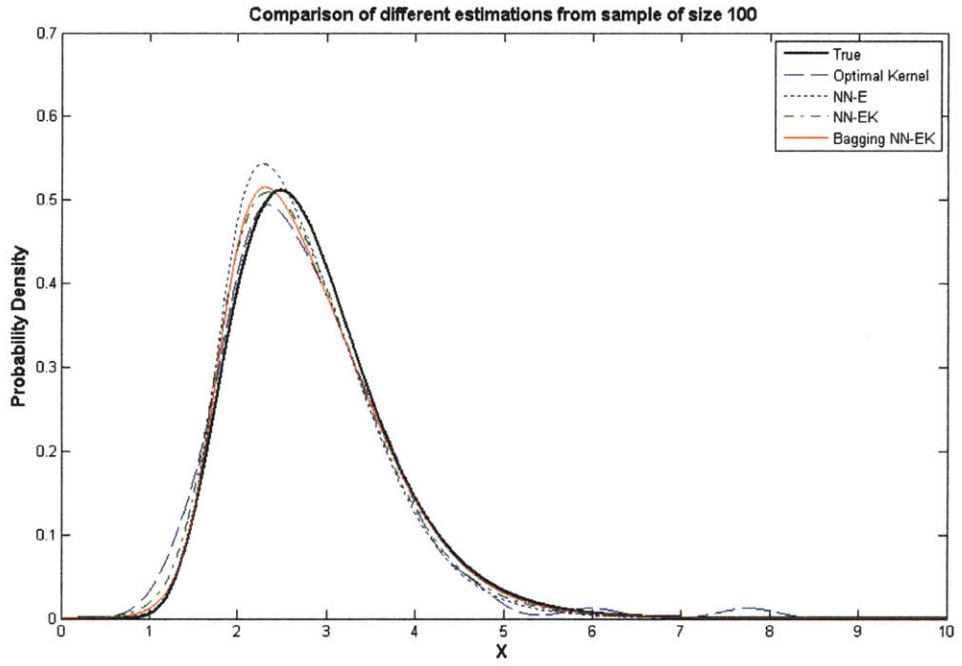


Figure 8-4 Bagging NN-EK estimate for PDF of lognormal(1.0, 0.3) model

Table 8-3 PDF estimation errors for lognormal(1.0, 0.3) model

	L1 Error	RMSE
Kernel estimation	0.011363	0.02014
NN-E estimation	0.010410	0.01729
NN-EK estimation	0.009084	0.01502
Bagging NN-EK estimation	0.006705	0.01385
Improvement	***40.99%***	***31.23%***

8.2 Case Study 2: Multimodal Model

8.2.1 Model Description

Multimodality in output distributions can also be found a lot in integrated probabilistic simulations. So we also had our approach tested on multimodal models. The multimodal model tested here is a mixture model of two normal distributions which are N(5.0, 1.0) and N(8.5, 1.2) with the weights as 0.6 and 0.4 respectively. The true PDF is shown in Figure 8-5.

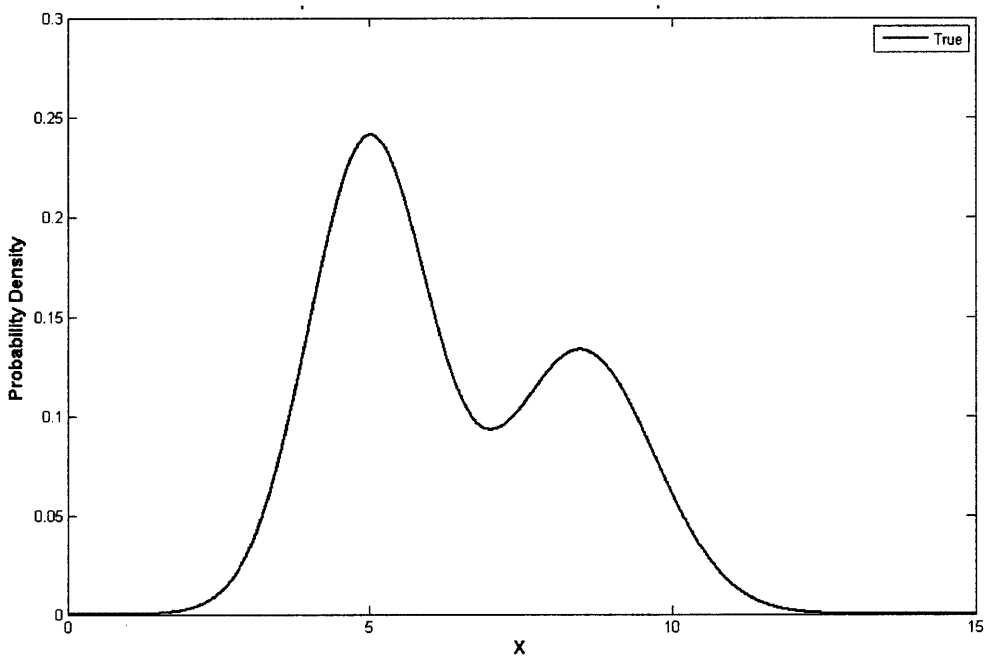


Figure 8-5 The multimodal model

8.2.2 Estimation by Different Methods

All estimations are based on the random sample of size 200.

1. NN-E Estimation

The NN-E estimate for PDF of the multimodal model is shown in Figure 8-6.

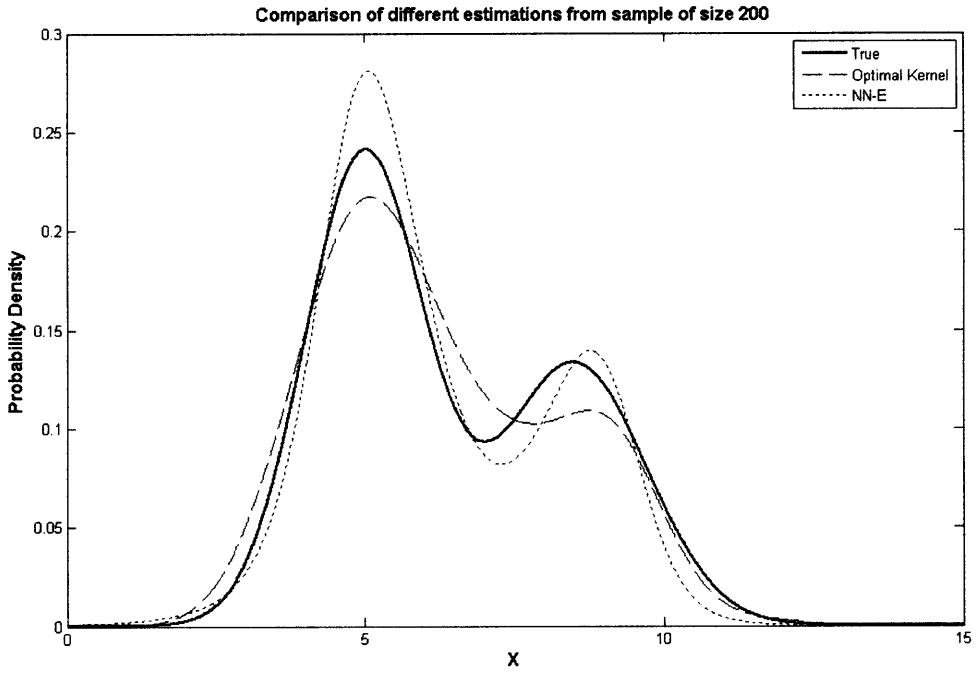


Figure 8-6 NN-E estimate for PDF of the multimodal model

In Figure 8-6, the kernel estimate with an optimal bandwidth is also shown for the purpose of comparison. We can see that the NN-E estimation is obviously better than the optimal kernel estimation. The PDF estimation errors are given in Table 8-4.

Table 8-4 PDF estimation errors for the multimodal model

	L1 Error	RMSE
Kernel estimation	0.009316	0.01371
NN-E estimation	0.008891	0.01310

2. NN-EK Estimation

The NN-EK estimate for PDF of the multimodal model compared with the kernel estimate and NN-E estimate is shown in Figure 8-7.

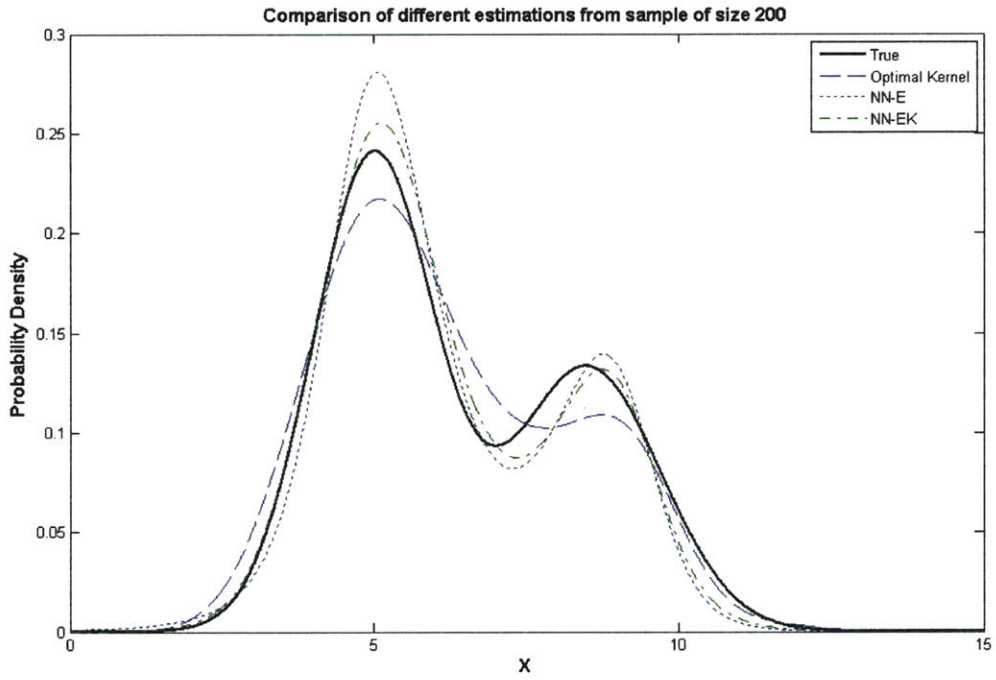


Figure 8-7 NN-EK estimate for PDF of the multimodal model

In Figure 8-7, we can see that the NN-EK estimation is obviously better than the NN-E estimation. The PDF estimation errors are given in Table 8-5.

Table 8-5 PDF estimation errors for the multimodal model

	L1 Error	RMSE
Kernel estimation	0.009316	0.01371
NN-E estimation	0.008891	0.01310
NN-EK estimation	0.006032	0.00945

3. Bagging NN-EK Estimation

In Bagging NN-EK estimation, 30 bootstrap replicates are used. Figure 8-8 is showing the Bagging NN-EK estimate for PDF of the multimodal model. The PDF estimation

errors are given in Table 8-6. It can be seen that the Bagging NN-EK estimation gains obvious improvement on the single NN-EK estimation and achieves 40.95%(L1 error) and 42.67%(RMSE) on the optimal kernel estimation.

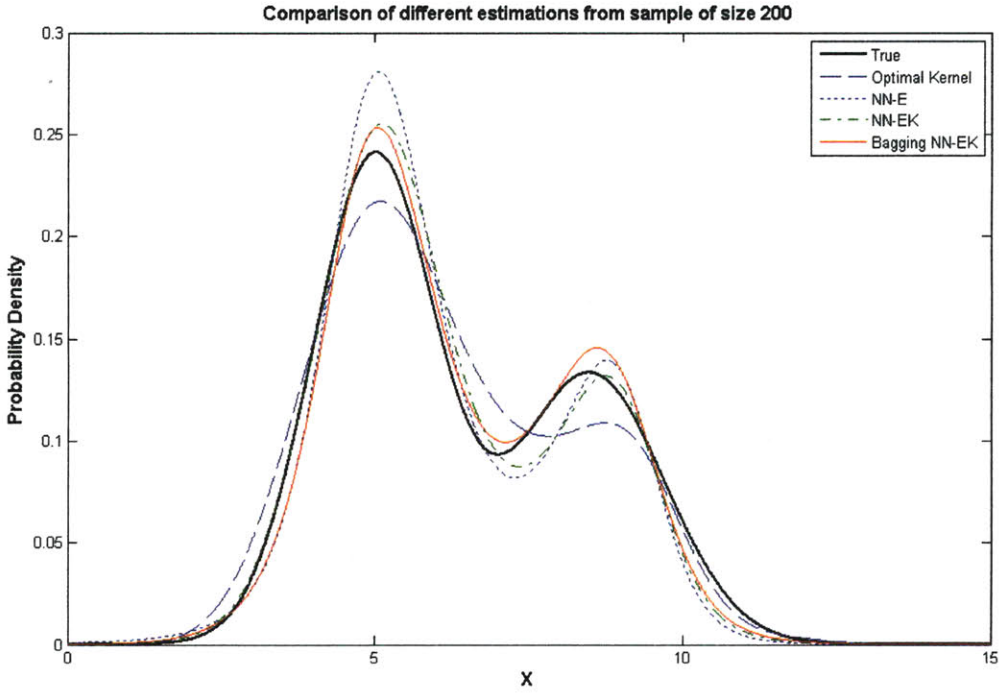


Figure 8-8 Bagging NN-EK estimate for PDF of the multimodal model

Table 8-6 PDF estimation errors for the multimodal model

	L1 Error	RMSE
Kernel estimation	0.009316	0.01371
NN-E estimation	0.008891	0.01310
NN-EK estimation	0.006032	0.00945
Bagging NN-EK estimation	0.005501	0.00786
Improvement	***40.95%***	***42.67%***

8.3 Case Study 3: Piston Model

8.3.1 Model Description

This model, shown in Figure 8-9, is used for the design of a piston assembly. The piston displacement needs to be analyzed and designed to meet customer requirements. Three separate components, the crank length, connecting rod length, and piston height determine the piston displacement. Each of the three parts has a target value. However, the actual lengths of these piston assembly parts are not produced at exactly the target values. Each length has a different degree of uncertainty. Then these lengths are defined as random variables with the appropriate probability distributions (see Table 8-7). α is a constant. In this case, it is 60 degree. As a result, the piston displacement is also a random variable with an unknown distribution, which is what we are going to estimate.

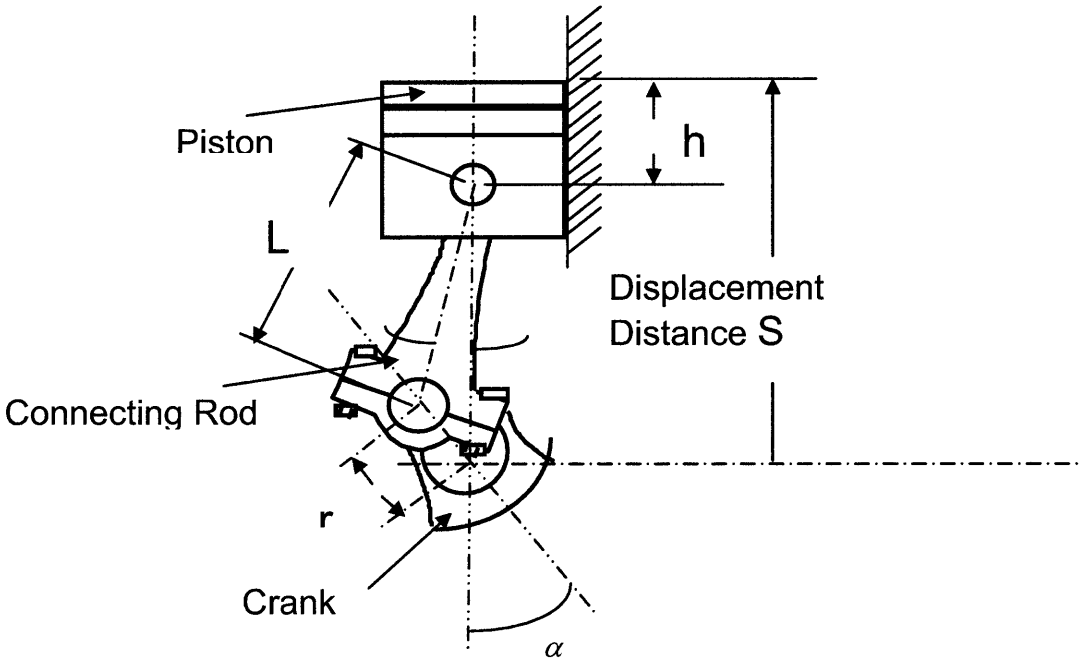


Figure 8-9 Piston model

Table 8-7 Random variables for piston parts (unit: mm)

Random Variable	Name	Distribution

L	connecting rod length	N(7.9, 0.6)
r	crank length	N(2.6, 0.3)
h	piston height	Gamma(2.0, 1.0)

8.3.2 Estimation by Different Methods

All estimations are based on the random sample of size 100, which is attained from the probabilistic simulation of the piston model.

1. NN-E Estimation

The NN-EK estimate for PDF of the piston model compared with the kernel estimate and NN-E estimate is shown in Figure 8-10.

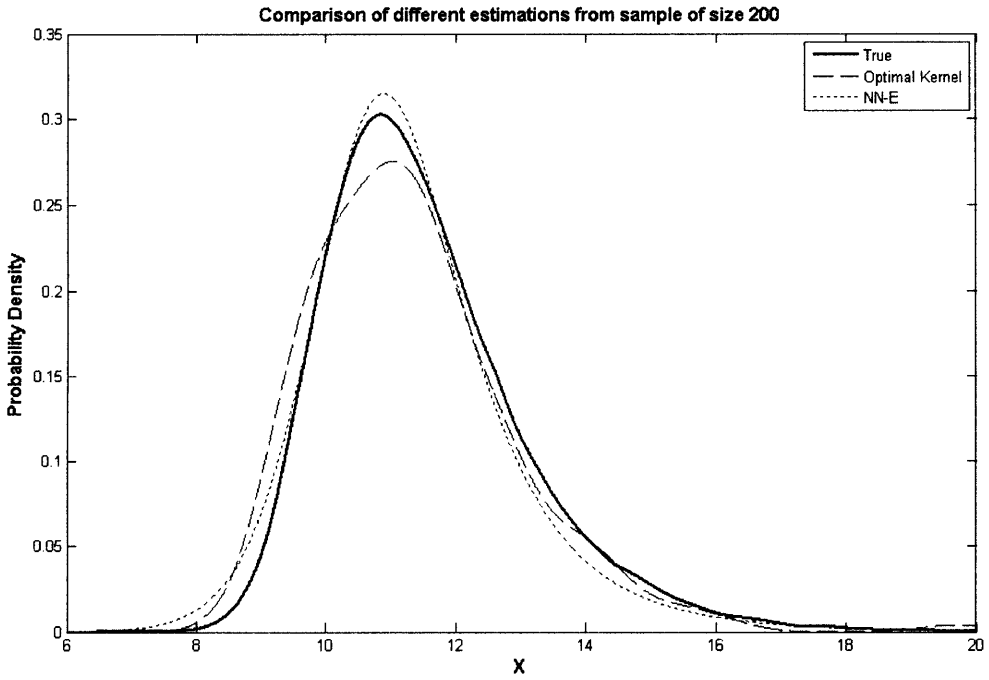


Figure 8-10 The NN-E estimate for PDF of the piston model

In Figure 8-10, the kernel estimate with an optimal bandwidth is also shown for the purpose of comparison. We can see that the NN-E estimation is obviously better than the optimal kernel estimation. The PDF estimation errors are given in Table 8-8.

Table 8-8 PDF estimation errors for the piston model

	L1 Error	RMSE
Kernel estimation	0.009343	0.01558
NN-E estimation	0.007406	0.01045

2. NN-EK Estimation

The NN-EK estimate for PDF of the piston model compared with the kernel estimate and NN-E estimate is shown in Figure 8-11.

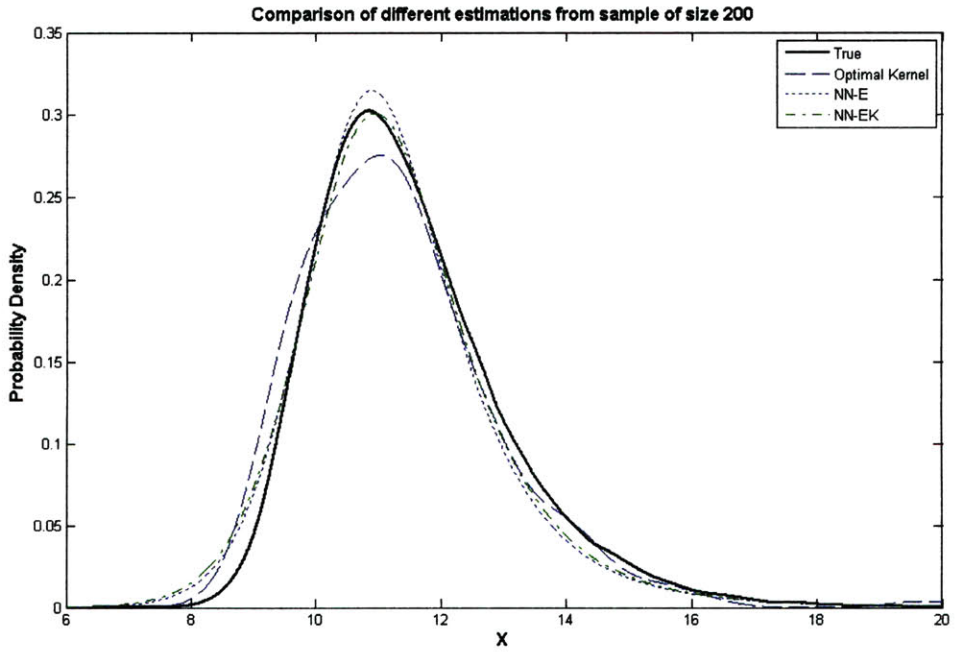


Figure 8-11 NN-EK estimate for PDF of the piston model

In Figure 8-11, we can see that the NN-EK estimation is obviously better than the NN-E estimation. The PDF estimation errors are given in Table 8-9.

Table 8-9 8-9 PDF estimation errors for the piston model

	L1 Error	RMSE
Kernel estimation	0.009316	0.01371
NN-E estimation	0.008891	0.01310
NN-EK estimation	0.006032	0.00945

3. Bagging NN-EK Estimation

In the Bagging NN-EK estimation, 30 bootstrap replicates are used. Figure 8-12 is showing the Bagging NN-EK estimate for PDF of the piston model. The PDF estimation errors are given in Table 8-10. It can be seen that the Bagging NN-EK estimation gains obvious improvement on the single NN-EK estimation and achieves 50.58%(L1 error) and 54.81%(RMSE) on the optimal kernel estimation.

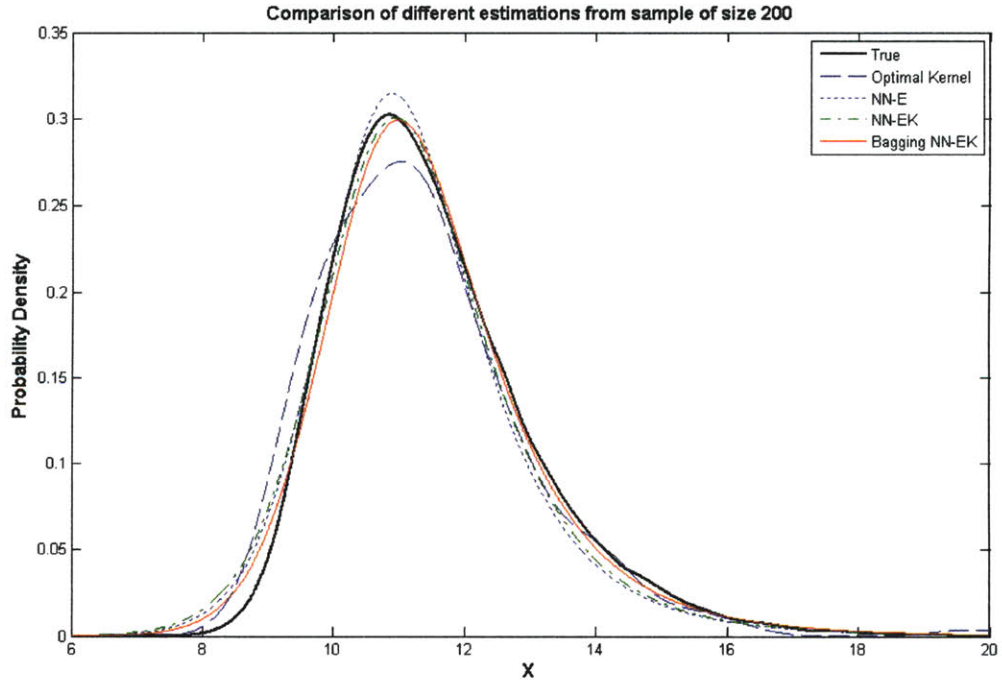


Figure 8-12 Bagging NN-EK estimate for PDF of the piston model

Table 8-10 PDF estimation errors for the piston model

	L1 Error	RMSE
Kernel estimation	0.009343	0.01558
NN-E estimation	0.007406	0.01045
NN-EK estimation	0.006642	0.00981
Bagging NN-EK estimation	0.004617	0.00704
Improvement	***50.58%***	***54.81%***

8.3.3 Comparison with Monte Carlo Method

Mode is a very important feature of a probability density curve. It is the value with the highest probability density around which most outcomes are gathering. Mode is thought as the target value in most engineering applications. Here the mode estimation by Bagging NN-EK approach is compared with that by the conventional Monte Carlo method. The estimation results based on sample size 200 are given in Table 8-11 (The true value is 10.851cm). Obviously, the Bagging NN-EK estimation is much more accurate than the conventional Monte Carlo estimation.

Table 8-11 Mode estimation for the piston model

	Mode(cm)	Error(cm)
Bagging NN-EK estimation	10.973	0.122
Conventional Monte Carlo	11.227	0.376

By increasing sample size, both Bagging NN-EK approach and the conventional Monte Carlo method can attain more accurate estimations. However, Bagging NN-EK approach converges to the true value much faster than the conventional Monte Carlo method. Table 8-12 is showing sample size comparison in order to achieve equivalent accuracy.

Table 8-12 Sample size to achieve equivalent accuracy

Error(cm)	N_1(Bagging NN-EK)	N_2(Monte Carlo)	N_2 / N_1
0.122	200	12800	64
0.061	900	140200	156
0.03	4200	1602400	381
0.02	9800	6086000	621
0.01	43000	60840000	1414

From Table 8-12, we can see that, the conventional Monte Carlo method needs 64 times more sample points than Bagging NN-EK approach so as to achieve the accuracy attained by Bagging NN-EK approach from sample of size 200. To reduce the estimation error from 0.122cm to 0.061cm, Bagging NN-EK approach needs sample size to be 900, and the conventional Monte Carlo method needs 140200 sample points which are 156 times more than what are needed by Bagging NN-EK approach. Apparently, Bagging NN-EK approach converges to the true value with a much higher rate than the conventional Monte Carlo method.

In a summary, compared to the conventional Monte Carlo method, our approach minimally requires an order of magnitude fewer model evaluations to achieve the same level of estimation accuracy.

Chapter 9

Concluding Remarks

In this thesis, a new machine learning based approach is proposed for scalable probabilistic simulation in an integrated design environment since the conventional Monte Carlo method is computationally prohibitive to large-scale integrated simulations due to its need for a large number of sample points. This approach is using machine learning techniques and statistical means to estimate the underlying distribution from the random sample attained by the integrated simulation within the affordable time.

The neural network CDF estimator is first constructed with a feedforward multilayer architecture and a suitable activation function. The mathematical model for the neural network CDF estimator is derived. And the neural network PDF estimator can be derived by differentiation of CDF estimator. The back-propagation learning is implemented on our neural network estimator so that it can learn from the sample and make the estimate. Many efforts are then put on statistically processing the random sample to gain a training set containing as much statistical information as possible. By learning from the training set based on the empirical cumulative probabilities, the NN-E estimator is attained. By learning from the training set combining the information from empirical CDF and kernel estimation, the NN-EK estimator can be attained. The NN-EK estimator can give a better estimate than the NN-E estimator according to our experiments.

How to learn from hints is also studied in order to improve the performance of our neural network estimator. A novel way is created to incorporate the monotonicity hint into the back-propagation learning by creating a hint-reinforced training set.

To further improve the performance of our neural network estimator, the statistical method Bagging is used. Multiple versions of our neural network estimator are generated by bootstrap method and aggregated to a final estimator.

Our experiments show that the proposed approach gets better results compared with those traditional density estimation methods and has a big advantage over the conventional Monte Carlo method in computation cost. This work improves PDF estimation accuracy by 30%-50% compared with kernel estimation. Compared to the conventional Monte Carlo method, this work minimally requires an order of magnitude fewer model evaluations to achieve the same level of estimation accuracy.

Bibliography

Abrahamson, S. D. R. W., Nicola Senin, and Nick Borland (1999). "Integrated Engineering, Geometric, and Customer Modeling: LCD Projector Design Case Study". Proceedings of the ASME DT Conferences, Las Vegas, NV.

Abrahamson, S. D. R. W., Nicola Senin, and Peter Sferro (2000). "Integrated Design in a Service Marketplace." Computer-aided Design **32**: 97-107.

Abu-Mostafa, Y. S. (1990). "Learning from Hints in Neural Networks." Journal of Complexity **6**: 192-198.

Abu-Mostafa, Y. S. (1993). "An Algorithm for Learning from Hints". Proceedings of 1993 International Joint Conference on Neural Networks.

Abu-Mostafa, Y. S. (1993). "A Method for Learning from Hints." Advances in Neural Information Processing Systems **5**: 73 - 80.

Abu-Mostafa, Y. S. (1995). "Hints." Neural Computation **7**: 639 - 671.

Adali, T., Liu, X. and Sönmez, K. (1997). "Conditional Distribution Learning with Neural Networks and its Application to Channel Equalization." IEEE Trans. Signal Processing **45**: 1051–1064.

Ang, A. H.-S., and Tang, Wilson H. (1975). Probability Concepts in Engineering Planning and Design, John Wiley & Sons, Inc.

Bishop, C. M., and Legleye, C. (1995). "Estimating Conditional Probability Densities for Periodic Variables." Advances Neural Inform. Processing Syst. **7**.

Blasone, R. S. (2008). "Generalized Likelihood Uncertainty Estimation (GLUE) Using Adaptive Markov Chain Monte Carlo Sampling." Advances in Water Resources **31**(4): 630 - 648.

Bliznakov, P. I., and Shah, J. J. (1996). "Integration Infrastructure to Support Concurrence and Collaboration in Engineering Design". 1996 ASME Design Engineering Technical Conferences, Irvine, Ca.

Bowyer, A. (1980). Experiments and Computer Modelling in Stick-slip, University of London. **Ph.D. Thesis**.

Breiman, L. (1996). "Bagging Predictors." Machine Learning **24**: 123 - 140.

Breiman, L. (1996). "Heuristics of Instability and Stabilization in Model Selection." Annals of Statistics **24**(6): 2350 - 2383.

Case, M. P., and Lu, S. C.-Y. (1996). "Discourse Model for Collaborative Design." Computer-Aided Design **28**: 333-345.

Casella, G., and Berger, R. L. (2002). Statistical Inference, Thomson Learning Inc.

Chernick, M. R. (1999). Bootstrap Methods: A Practitioner's Guide. New York, Wiley.

Christian, A. D. K. J. G., and Warren P. Seering (1996). "Validation studies of an information-flow model of design". Proceedings of the 1996 ASME Design Engineering Technical Conferences.

Cutkosky, M. R., and Engelmore, R. et al. (1996). "PACT: An Experiment in Integrating Concurrent Engineering Systems." IEEE Computer: 28-37.

Cwik, J., and Koronacki, J. (1996). "Probability Density Estimation Using a Gaussian Clustering Algorithm." Neural Comput. Applicat. **4**(3): 149–160.

Dabke, P., and Cox, A. (1998). "NetBuilder: an environment for integrating tools and people." Computer-Aided Design **30**: 465-472.

Davison, A. C., and Hinkley, D.V. (1997). Bootstrap Methods and Their Application. Cambridge, Cambridge University Press.

Efron, B. (1979). "Bootstrap Methods: Another Look at the Jackknife." Annals of Statistics **7**: 1 - 26.

Efron, B., and Tibshirani, R. J. (1993). An Introduction to the Bootstrap. New York, Chapman & Hall.

Fishman, G. S. (1996). Monte Carlo: concepts, algorithms, and applications, Springer-Verlag.

Gentle, J. E. (2003). Random Number Generation and Monte Carlo Methods, Springer-Verlag.

Härdle, W. (1991). Smoothing Techniques, With Implementations in S New York, Springer.

Härdle, W., Müller, M., Sperlich, S. and Werwatz, A. (2004). Nonparametric and Semiparametric Models, Springer.

Haugen, E. B. (1980). Probabilistic Mechanical Design, John Wiley & Sons, Inc.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation, Prentice Hall.

Hornik, K., Stinchcombe, M. and White, H. (1990). "Universal Approximation of an Unknown Mapping and its Derivatives Using Multilayer Feedforward Networks." Neural Networks **3**: 551–560.

Hornik, K., Stinchcombe, M., White, H. and Auer, P. (1994). "Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives." Neural Comput. **6**: 1262–1275.

Hulle, M. V. (1996). "Topographic Map Formation by Maximizing Unconditional Entropy: A Plausible Strategy for On-line Unsupervised Competitive Learning and Nonparametric Density Estimation." IEEE Trans. Neural Networks **7**: 1299–1305.

Husmeier D. and Taylor, J. (1998). "Predicting Conditional Probability Densities: Improved Training Scheme Combining EM and RVFL." Neural Networks **11**(1): 89–116.

Isukapalli, S. S. (1999). "Uncertainty Analysis of Transport-Transformation Models". Chemical and Biochemical Engineering, Rutgers University.

Kalos, M. H., and Whitlock, P. A. (1986). Monte Carlo Methods : Basics. New York, John Wiley & Sons.

Kim, C., and Kim, Y. (1998). "Internet-based Concurrent Engineering: An Interactive 3D System with Markup". ASME 18th Computers in Engineering Conference.

Kowal, M. T., Dey, Animesh, and Tryon, Robert G. (1998). "Integrated Design Method for Probabilistic Design". Proceedings of the Annual Reliability and Maintainability Symposium.

Magdon-Ismail, M., and Atiya, A. (2002). "Density Estimation and Random Variate Generation Using Multilayer Networks." IEEE TRANSACTIONS ON NEURAL NETWORKS **13**(3): 497 - 520.

Miller, G., and Horn, D. (1998). "Probability Density Estimation Using Entropy Maximization." Neural Comput. **10**(7): 1925–1938.

Mitchell, T. (1997). Machine Learning, McGraw Hill.

Molina, A., and Al-Ashaab, A. H. et al. (1995). "A review of computer aided simultaneous engineering systems." Research in Engineering Design **7**: 38-63.

Pahng, K. F., Senin, N., and Wallace, D. R. (1997). "Modeling and Evaluation of Product Design Problems in a Distributed Design Environment". Proceedings of ASME DETC'97, Sacramento, California.

Pahng, K. F., Senin, N., and Wallace, D. R. (1998). "Distributed Modeling and Evaluation of Product Design Problems." Computer-Aided Design **30**(6): 411–423.

Park, B. U., and Turlach, B. A. (1992). "Practical performance of several data driven bandwidth selectors." Computational Statistics **7**: 251-270.

Pradlwarter, H. J. (1997). "On Advanced Monce Carlo Simulation Procedures in Stochastic Structural Dynamics." International Journal of Non-Linear Mechanics **32**(4): 735 - 744.

Reed, R. D., and Marks, R. J. (1999). "Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks", The MIT Press.

Robert, C. P., and Casella, George (2004). Monte Carlo Statistical Methods, Springer.

Roth, Z., and Baram, Y. (1996). "Multidimensional Density Shaping by Sigmoids." IEEE Trans. Neural Networks **7**: 1291–1298.

Rubinstein, R. Y. (1981). Simulation and the Monte Carlo Method. New York, John Wiley & Sons.

Rumelhart, D. E., Hinton, G. E.and Williams, R. J. (1986). "Learning Internal Representations by Back-propagating Errors." Nature **323**: 533 - 536.

Schioler, H., and Kulczyki, P. (1997). "Neural Network for Estimating Conditional Distributions." IEEE Trans. Neural Networks **8**: 1015–1025.

Scott, D. W. (1992). Multivariate Density Estimation: Theory, Practice, and Visualization. New York, John Wiley & Sons.

Senin, N., Wallace, David R., and Borland, Nicolas (2003). "Distributed Object-based Modeling in Design Simulation Marketplace." Journal of Mechanical Design **125**(1): 2-13.

Siddall, J. N. (1983). Probabilistic Engineering Design, Marcel Dekker, Inc.

Silverman, B. W. (1998). Density Estimation for Statistics and Data Analysis, Chapman & Hall.

Smyth, P., and Wolpert, D. (1998). "Stacked Density Estimation." Advances in Neural Information Processing Systems (NIPS) **8**.

Tari, M. (2006). "Refined Descriptive Sampling: A Better Approach to Monte Carlo Simulation." Simulation Modelling Practice and Theory **14**(2): 143 - 160.

Thathachar, M., and Arvind, M. (1999). "Global Boltzmann Perceptron Network for On-line Learning of Conditional Distributions." IEEE Trans. Neural Networks **10**: 1090–1098.

Thunnissen, D. P. (2007). "Uncertainty Quantification in Conceptual Design via an Advanced Monte Carlo Method." Journal of Aerospace Computing, Information and Communication **4**(7): 902 - 917.

Tong, C. (2006). "Refinement Strategies for Stratified Sampling Methods." Reliability Engineering and System Safety **91**(10-11): 1257 - 1265.

Toye, G., and Cutkosky, M. R. et al. (1994). "SHARE: a methodology and environment for collaborative product development." International Journal of Intelligent & Cooperative Information Systems **3**: 129-153.

Traven, H. (May 1991). "A Neural Network Approach to Statistical Pattern Classification by Semiparametric Estimation of Probability Density Functions." IEEE Trans. Neural Networks **2**: 366–377.

Wallace, D. "The World-wide Simulation Web as an Enabler for Emergent System Definition." MIT CADLAB.

Wallace, D., Abrahamson, S., Senin, N., and Sferro, P. (2000). "Integrated Design in a Service Marketplace." Computer-Aided Design **32**: 97-107.

Wand, M. P., and Jones, M. C. (1995). Kernel Smoothing. London, Chapman and Hall.

Weigend, A., and Srivastava, A. (1995). "Predicting Conditional Probability Distributions: A Connectionist Approach." Int. J. Neural Syst. **6**(2): 109–118.

Williams, P. (1996). "Using Neural Networks to Model Conditional Multivariate Densities." Neural Comput. **8**: 843–854.

Zeevi, A., and Meir, R. (1997). "Density Estimation Through Convex Combinations of Densities: Approximation and Estimation Bounds." Neural Networks **10**(1): 99–110.