Chapter 7
Performance Estimation and Probably Approximately Correct Computation

The analysis phase of a problem aims at evaluating, given a computation, its performance. Performance can be intended in several ways depending on the specific target problem as well as the abstraction level where it is carried out. For instance, at the device level we have cost, latency, throughput, power, energy, complexity to name some major performance design indexes. At the algorithm level we have accuracy, confidence, energy, and complexity. Not rarely we constrain such indexes and we saw in Chap. 4 how it is possible to evaluate their satisfaction level.

Performance and design indexes are evaluated through suitable figures of merit applied to architectural and functional elements of the embedded system or the algorithm. Despite the fact that the provided methodological analysis is able to address all figures of merit and architectural aspects, we will mainly focus on accuracy as a case study of a performance/design index without any loss in generality. As a consequence, we look at algorithms to be executed on embedded systems whereas the embedded system being characterized by finite resources introduces physical constraints which, in turn, affect the algorithm itself and its performance. At the same time, as already pointed out in the introduction, we discover that it is too expensive, and most of times not necessary, to provide a worst-case analysis for accuracy. In line with approximate computation and probabilistic computation, we are here interested in an algorithm that, mounted in the embedded system, provides an outcome correct in probability.

The chapter introduces at first methods for assessing the accuracy of a computation, then formalizes the concept of Probably Approximately Correct Computation PACC. Finally, it provides techniques for accuracy assessment in terms of PACC and methods for answering to the following questions:

1. Which are the performances of my algorithm?
2. If I simplify my algorithm, which is the introduced performance loss?
3. I have different algorithms solving my problem. Which one is the best?
4. I have different algorithms solving my problem. Which one is the best on a given embedded system?
5. Shall I use a floating point unit or a cheaper fixed point representation suffices?

7.1 Accuracy Estimation: Figures of Merit

Different figures of merit can be considered to evaluate the discrepancy between two functions which, here, must be intended as the optimal—ideal—solution for a given problem and the approximated solution proposed as the candidate to be implemented in the embedded system. The discrepancy can be intended as a performance loss, difference in accuracy between an identified model and the true one or, simply, a measure of the "distance" between the two. The figure of merit depends on the specific application and the goodness of a solution also depends on the chosen figure of merit in the sense that the same solution can be more or less good depending on the chosen performance evaluation tool.

Since our goal is to provide a function approximating the real one, in the following we consider assessing the discrepancy between two functions according to figure of merit $u \in \mathbb{R}$. The evaluation is carried out by taking into account the punctual discrepancy of two functions $u(x) = u(y(x), \hat{y}(x)) \in \mathbb{R}$,[1] where we assume $x \in X \subset \mathbb{R}^d$. X is a probability space, whose probability measure μ induces pdf f_x over it, and $y(x), \hat{y}(x) \in \mathbb{R}$, which we assume to be measurable functions according to Lebesgue. By applying an aggregation operator to the punctual discrepancy $u(x)$ to $x \in X$ we obtain the discrepancy u. In the sequel, we indicate as y the *reference* function and $\hat{y}$ the *approximating* one. Two interesting cases arise from the applications:

- Functions $y(x)$ and $\hat{y}(x)$ are given. The figure of merit u evaluates the discrepancy over the whole input space X. Again, if f_x is unknown we shall consider a uniform distribution for its worst case properties.
- Function $y(x)$ is not given, but can be queried, i.e., once a sample x_i is drawn from the input space, function $y(x)$ acts as an oracle and provides value $y(x_i)$, possibly affected by uncertainty. The number of samples can be finite or infinite depending on the nature of the application, thus $x_i \in \tilde{X} \subset X, i \in \mathbb{N}$ are sampled according to the pdf f_x induced by the probability measure over X and are subsequently considered for the discrepancy computation. In this situation also function $\hat{y}(x)$ is not known, therefore it must be at first identified, by considering a suitable model family function $\hat{y}(x, \theta)$ parametric in the vector of parameters $\theta \in \Theta \subset \mathbb{R}^l$. Then, with abuse of notation $\hat{y}(x, \hat{\theta}) = \hat{y}(x)$ once $\hat{\theta}$ has been provided.

The first case arises in all those applications for which the theory provides the optimal solution to the problem and we need to approximate it for several reasons, e.g., because our embedded system is not able to host the high accurate solution for its complexity and an approximation needs to be considered instead. As an example, we have the optimal design of filters and representation of a complex numerical algorithm. The second case is what the theory of system identification and learning is about: starting from a sequence of input/output pairs we determine

[1] With an abuse of notation that eases the understanding, we consider the punctual discrepancy as a function of x, since y and $\hat{y}$ are fixed.

the approximating function. This topic, which is fundamental in this book, has been addressed in Sect. 3.4.1.

In the following we will present, without the intent to be exhaustive, three interesting u figures of merit. We recall that it is the application designer who identifies the right figure of merit for a given problem based on a priori knowledge, experience, and application constraints. However, when one does not know which figure of merit to consider, it is a rather common approach to use the mean squared one.

7.1.1 Squared Error

The Squared Error (SE) is a rather common quadratic figure of merit adopted to quantify the difference between two functions. The corresponding discrepancy u_{SE} is a risk function, corresponding to the expected value of the squared punctual discrepancy $u(x) = (y(x) - \hat{y}(x))^2$

$$u_{SE} = E[u(x)] = E\left[(y(x) - \hat{y}(x))^2\right]$$

which represents the second order moment of the error (considering a zero mean error). It is worth observing that the discrepancy between the two functions is weighted by the probability density function f_x over the input space.

Its empirical version evaluated over a finite number of n points drawn according to f_x (i.e., considering a finite space $\tilde{X} = \{x_1, \ldots, x_n\}$ where x_i is a realization of a random variable $x \in X$ with pdf f_x), provides the empirical estimate of the quadratic error or a Mean Squared Error (MSE)

$$u_{MSE} = \frac{1}{n}\sum_{i=1}^{n} u(x_i) = \frac{1}{n}\sum_{i=1}^{n} \left(y(x_i) - \hat{y}(x_i)\right)^2 .$$

We have been using u_{SE} and u_{MSE} widely over the book for their intriguing structure that makes the mathematics amenable. Moreover, despite the fact that the use of a SE loss function has been criticized, e.g., in speech and image applications for its quadratic behavior which amplifies large point-wise discrepancies more than small ones and is not a perception-based figure of merit [41, 42], it is commonly adopted since it is easy to use [40]. Moreover, a quadratic function is a natural way to measure the energy of the discrepancy function $u(x)$ and, thanks to the Parseval theorem, the energy of the signal can be equivalently computed in the signal space or frequency domain. It is clear that, if functions $y(x)$ and $\hat{y}(x)$ are deterministic, then the u_{SE} simply evaluates the approximation risk (model bias), namely the integral of the squared discrepancy. In this case, the smaller the u_{MSE} the better the approximating function. Conversely, if $y(x)$ is affected by noise and $\hat{y}(x)$ is learned as proposed in Sect. 3.4.1, then expectation must be extended to the noise as well. Results presented in Sect. 3.4.1 hold.

7.1.2 Kullback–Leibler

The Kullback–Leibler divergence [43, 44] measures the distance between two probability density functions, which, in the following, we denote as $y(x)$ and $\hat{y}(x)$.

More specifically, if the densities $y(x)$ and $\hat{y}(x)$ exist then the Kullback–Leibler divergence is

$$u_{KL} = \int_X y(x) \log \frac{y(x)}{\hat{y}(x)} \mathrm{d}x \tag{7.1}$$

The figure of merit, also known as information divergence and relative entropy, is not a metric (and this is the reason why it is called divergence), since it does not satisfy the symmetry property, i.e., $u_{KL}\left(y(x), \hat{y}(x)\right) \neq u_{KL}\left(\hat{y}(x), y(x)\right)$. However, $u_{KL}\left(y(x), \hat{y}(x)\right) \geq 0$ and assumes value zero only when $\hat{y}(x) = y(x),\ \forall x \in X$, that is to say when the two distributions are equal.

In the machine learning field, the Kullback–Leibler divergence plays a leading role. For instance, in Bayesian machine learning it is used to approximate an intractable density model [48]. In other application scenarios, the divergence is used for parameter estimation [46], text classification [45] and, again multimedia applications [47] just to name the few.

7.1.3 L^p Norms and Other Figures of Merit

Several other figures of merit can be designed to asses the discrepancy between two functions based on the L^p norms. For instance, if we consider as punctual discrepancy $u(x) = y(x) - \hat{y}(x)$, we can use its L^p norm as figure of merit

$$u_{L^p} = \|u(x)\|_p = \|y(x) - \hat{y}(x)\|_p = \left(\int_X |y(x) - \hat{y}(x)|^p f_x(x)\mathrm{d}x\right)^{\frac{1}{p}}. \tag{7.2}$$

where $f_x(x)\mathrm{d}x = \mathrm{d}\mu(x)$ is the differential of the probability measure over X. Of particular interest are the L^1, L^2 (equivalent to the SE approach, since $u_{SE} = u_{L^2}$) and L^∞ norms.

In some cases, the punctual discrepancy is weighted by a given function $w(x) \geq 0$, $\forall x \in X$ and (7.2) becomes discrepancy induced by the weighted L^p norm

$$u_{L^p,w} = \left(\int_X w(x)|y(x) - \hat{y}(x)|^p f_x(x)\mathrm{d}x\right)^{\frac{1}{p}}$$

Other figures of merits may be derived by taking into account the mutual information [49], cross entropy [49], or maximum likelihood. However, whatever the chosen figure of merit is, the designer has solely to provide a Lebesgue measurable function with domain in a probability space, which is the unique requirement we ask for in subsequent analyses.

7.2 Probably Approximately Correct Computation

Consider given algorithm A associated with function $y(x), x \in X$ measurable according to Lebesgue and X a probability space. Denote as $\hat{y}(x), x \in X$ a given function approximating the $y(x)$ implementing algorithm $\hat{A}$. Let f_x be the probability density function associated with the measure over X. As previously mentioned we can relax the assumption by assuming that function $y(x)$ is not known but operates as an oracle providing value $y(x_i)$ once queried on x_i.

Definition *We say that function $\hat{y}(x)$ is a Probably Approximately Correct Computation (PACC) of function $y(x)$ at accuracy τ and confidence η when, given a Lebesgue measurable discrepancy function $u\left(y(x), \hat{y}(x)\right) \in \mathbb{R}$, we have that*

$$\Pr\left(u\left(y(x), \hat{y}(x)\right) \leq \tau\right) \geq \eta, \quad \forall x \in X. \tag{7.3}$$

In other terms, we are requesting that the two functions are close enough according to function $u(x)$; closeness must be intended in probabilistic terms within accuracy τ on the discrepancy satisfied with probability $\eta, \forall x \in X$. The computation provided by function $\hat{y}(x)$ is approximately correct in the sense that it approximates $y(x)$ according to $u(\cdot)$ at level τ; such a statement holds at least with probability η.

From the definition, we derive several interesting cases. Consider at first $u(x) = |y(x) - \hat{y}(x)|$. Given this loss function (7.3) can be expressed as

$$\Pr\left(|y(x) - \hat{y}(x)| \leq \tau\right) \geq \eta, \quad \forall x \in X \tag{7.4}$$

Since

$$\Pr\left(-\tau \leq y(x) - \hat{y}(x) \leq \tau\right) \geq \eta, \quad \forall x \in X$$

if we assume that τ is small then (7.4) can be cast in a more immediate and intuitive, yet less formal, form

$$\Pr\left(y(x) \simeq \hat{y}(x)\right) \geq \eta, \quad \forall x \in X.$$

The computation provided by our algorithm is approximately correct, i.e., it provides a value which nicely approximates the true one with high probability.

Example: Scalar product

As a first example of a PACC consider the error free set-up where

$$y(x) = x^T \theta^{\mathrm{o}} \tag{7.5}$$

with $X = [-1, 1]^d$, f_x is uniform and x and $\theta^{\mathrm{o}} \subset \mathbb{R}^d$ represent the column vectors of inputs and coefficients, respectively. The scalar product computed in (7.5) can be a linear filter of coefficients θ^o, as those used in the waivelets, e.g., see [27, 30, 31], in a Cordic computer [28, 29] or a filter bank [32]. We request function $y(x)$ to be either implemented in a digital hardware or executed on a microcontroller. Assume for simplicity that no overflow/underflow occurs in the computation and that the truncation operator has been envisaged to reduce the number of bits needed to represent the coefficients. Finite precision representation can be modeled as the perturbation vector $\delta\theta$ affecting coefficients θ^o and leading to approximated computation

$$\hat{y}(x) = x^T(\theta^o - \delta\theta).$$

Assume that $d >> 1$ and that inputs are mutually independent. If we consider the

$$u(x) = u(y(x), \hat{y}(x)) = y(x) - \hat{y}(x)$$

loss function, then the point-wise discrepancy is the variable

$$u(x) = x^T\delta\theta$$

which, from the central limit theorem, is asymptotically Gaussian with mean $E_x[u(x)] = 0$ and variance $E_x[u^2(x)] = \sigma^2 = \frac{\delta\theta^T\delta\theta}{3}$. Finally,

$$\Pr\left(|u(x) - E_x[u(x)]| \le \lambda\frac{\sigma}{\sqrt{d}}\right) = \operatorname{erf}\left(\frac{\lambda}{\sqrt{2}}\right)$$

i.e.,

$$\Pr\left(|y(x) - \hat{y}(x)| \le \lambda\frac{\sigma}{\sqrt{d}}\right) = \operatorname{erf}\left(\frac{\lambda}{\sqrt{2}}\right).$$

The PACC computation is characterized $\tau = \lambda\frac{\sigma}{\sqrt{d}}$ and $\eta = erf\left(\frac{\lambda}{\sqrt{2}}\right)$.

Example: linear regression

Consider function $y(x) = x^T\theta^o + \zeta$ where x and θ^o are the d-dimensional column vectors of inputs and given—but unknown—parameters, respectively. ζ is a zero mean white noise of variance σ_ζ^2 ruled by Gaussian pdf f_ζ. Inputs are zero centered and independent and identically distributed, extracted according to probability density function f_x of diagonal covariance $\sigma_x^2 I_d$. Consider the n samples training set $Z_n = \{(x_1, y(x_1)), \cdots (x_1, y(x_n))\}$.

Define $\chi = [x_1, \ldots, x_n]$ to be the (n, d) dimensional matrix containing the input vectors and $Y = [y(x_1), \ldots, y(x_n)]$ the $(n, 1)$ vector of associated outputs.

If we define

$$u(y(x), \hat{y}(x)) = \hat{y}(x) - y(x)$$

the least mean squared error estimate

$$\hat{y}(x) = x^T \hat{\theta}$$

can be obtained by minimizing the

$$u_{MSE} = \hat{E}_n(u(x)) = \frac{1}{n} \sum_{i=1}^{n} \left(y(x_i) - \hat{y}(x_i) \right)^2$$

and provides the parameter estimate

$$\hat{\theta} = \left(\chi^T \chi \right)^{-1} \chi^T Y.$$

We know from Sect. 3.4.4 that the distribution of $\hat{\theta}$ is centered in θ^{o}. $\hat{\theta}$ can then be seen as a perturbed value of θ^{o} so that $\hat{\theta} + \delta\theta = \theta^{o}$. We can repeat the derivation carried out in the previous experiment and the point-wise error

$$u(x) = x^T \delta\theta + \zeta$$

converges to a Gaussian distribution of zero mean and variance $\sigma^2 = \sigma_x^2 \delta\theta^T \delta\theta + \sigma_\zeta^2$ provided that d is large enough.

As before, we can then write

$$\Pr\left(|\hat{y}(x) - y(x)| \leq \lambda \frac{\sigma}{\sqrt{n}} \right) = erf\left(\frac{\lambda}{\sqrt{2}} \right) \tag{7.6}$$

With the choice $\tau = \lambda \frac{\sigma}{\sqrt{n}}$ and $\eta = erf\left(\frac{\lambda}{\sqrt{2}} \right)$, $\hat{y}(x)$ represents a PACC computation of $y(x)$ at level τ and probability η.

Example: Maximum value estimate

Another interesting case emerging from applications is the one where we aggregate punctual discrepancies with the maximum operator. Define

$$u_{\max} = \max_{x \in X} u(x)$$

with $\hat{u}_{\max}$ being the estimate of such a maximum as provided by a suitable algorithm. We have a good estimate $\hat{u}_{\max}$ when

$$\Pr\left(u_{\max} - \hat{u}_{\max} \leq \tau \right) \geq \eta \tag{7.7}$$

In other terms, the (7.7) states that estimate $\hat{u}_{\max}$ is a good estimate when the probability of having the true value within distance τ is high. In a way, (7.7) is closely related to the weak law of empirical maximum.

While the (7.7) is a well-posed formulation from the mathematical point of view it is of scarce use in the practice apart from very simple cases. In fact, for a generic $u(x)$ loss function, we cannot guarantee, even in probability, that given $\hat{u}_{\max}$ belongs to a neighborhood of $u_{\max}$ within distance below τ.

As we have seen in Chap. 4, this is a well-known problem whose solution within a PACC framework requires introduction of an additional level of probability.

Comments

The probabilistic theory behind the PACC well describes the operational modality of those applications for which an exact computation suffice. If an embedded system is considered, and given the comments raised in Chap. 3, we discover that very few applications require a high accuracy in the computation, e.g., those involving financial data, all the others being affected by uncertainty that affects the correctness of the computation output.

However, the use of the PACC theory appears to be of limited use in practical cases for generic $y(x)$ and $\hat{y}(x)$ functions due to the difficulty in providing (or determining) the η and τ values requested by (7.4) or identifying a $\hat{u}_{\max}$ that satisfies the (7.7).

Fortunately, thanks to the procedures based on randomized algorithms we introduce in the sequel, we will be able to provide estimates for both η and τ and $\hat{u}_{\max}$, hence making effective and operational the PACC framework. The main outcome is that any computation, under the very weak hypothesis of Lebesgue measurability, can be effectively cast into the PACC framework that makes available a set of algorithms able to identify η, τ, and $\hat{u}_{\max}$.

7.3 The Performance Verification Problem

The performance verification problem aims at verifying to which degree a performance level is attained, computing the probability that an inequality on performance is satisfied or estimating the maximum value associated with a performance discrepancy loss function.

In the following, the key actors will be the given functions $y(x)$ and $\hat{y}(x)$ and the Lebesgue measurable figure of merit $u\left(y(x), \hat{y}(x)\right)$. It is recalled that a probability density function f_x is induced by the measure of probability space X. We already pointed out in Chap. 4 that if f_x is unknown the user should consider a uniform distribution which, under mild hypotheses about the regularity of functions $y(x)$ and $\hat{y}(x)$, acts as a worst case scenario (e.g., derived bounds tend to be overestimated).

The computationally difficult problem of evaluating the different performance associated with a given figure of merit over the whole input space X can be tamed by resorting to probability and accepting a PACC computation.

We will address in the sequel the following problems

- *The performance satisfaction problem.* Given a tolerated performance loss τ for an application, compute the probability that the discrepancy $u\left(y(x), \hat{y}(x)\right) \leq \tau$, $\forall x \in X$;

- *Figure of merit expectation problem.* Provide an estimate of the expected value $E\left[u\left(y(x), \hat{y}(x)\right)\right]$ at arbitrary accuracy and confidence levels;
- *The maximum performance problem.* The goal is to provide an estimate $\hat{u}_{\max}$ for the maximum value $u_{\max} = \max_{x \in X} u\left(y(x), \hat{y}(x)\right)$;
- *The PACC problem.* Compute those η and τ characterizing the PACC degree for the given application;
- *The minimum(maximum)-perturbed expectation problem.* Estimate the minimum/maximum value performance function $u(x)$ assumes when perturbations affect it.

Clearly, $u\left(y(x), \hat{y}(x)\right)$ can degenerate to $\hat{y}(x)$. When this happens, the considered problems must be intended as applied to function $\hat{y}(x)$.

7.3.1 The Performance Satisfaction Problem

The performance satisfaction problem aims at assessing the level of performance satisfaction of a given $\hat{y}(x)$ function in approximating $y(x)$ according to figure of merit $u\left(y(x), \hat{y}(x)\right)$ and a tolerated performance loss τ.

Examples of applications

- We designed application $\hat{y}(x)$ solving my problem. Is that satisfying the accuracy constraint set at level τ requested by the application?
- We designed solution $y(x)$ which is working well in a high performance machine where it satisfies the requested real time performance. We port it on our embedded system where it becomes solution $\hat{y}(x)$. Is porting granting a loss in execution time between the two platforms below τ?
- We designed solution to our problem $y(x)$ within a high resolution platform (e.g., Matlab, Mathematica, Simulink). Then we need to port the solution to an embedded system characterized by a given finite precision representation (limited word-length for data and inner variables, truncation mechanisms and look up tables). Is the performance loss in accuracy we are introducing tolerable if τ is what we are willingly to loose?
- Our application solution $\hat{y}(x)$ has an accuracy scalable with complexity in the sense that the solution performance (accuracy, execution performances, power consumption) scales with the solution complexity (the higher the complexity the better the performance). We would like to minimize complexity provided that performance loss τ is attained.

The above problems can be formalized as follows: Given a tolerated performance loss τ we wish to estimate the satisfaction level

$$u\left(y(x), \hat{y}(x)\right) \leq \tau, \quad \forall x \in X$$

that is to say, determine the percentage of points of X satisfying the inequality. The problem can be immediately related to the algorithm verification problem presented in Sect. 4.4.1 by simply substituting x to each occurrence of ψ. We recall the main operational steps.

The percentage of points $x \in X$ satisfying $u(x) \le \tau$ is simply the ratio

$$n_{u(x)\le\tau} = \frac{\int_X I(x)\mathrm{d}x}{\int_X \mathrm{d}x}$$

where

$$I(x) = \begin{cases} 1 \text{ if } & u(x) \le \tau \\ 0 \text{ otherwise} \end{cases}$$

Since determination of $n_{u(x)\le\tau}$ is a computationally hard problem for a generic function we move to a probabilistic problem.

In order to do that, consider the probability density function f_x defined over X and request to evaluate the probability

$$p_\tau = \Pr\left(u(x) \le \tau\right) = \frac{\int_X I(x) f_x \mathrm{d}x}{\int_X f_x \mathrm{d}x} = \int_X I(x) f_x \mathrm{d}x$$

since $\int_X f_x \mathrm{d}x = 1$. We have seen in Chap. 4 that p_τ can be evaluated through randomization and that, given τ, we can compute the estimate $\hat{p}_n$ of p_τ by drawing n samples $x_1, \ldots x_n$ according to f_x and evaluate the indicator function

$$I\left(u(x) \le \tau\right) = \begin{cases} 1 \text{ if } u(x) \le \tau \\ 0 \text{ if } u(x) > \tau \end{cases}$$

and the estimate

$$\hat{p}_n = \frac{1}{n} \sum_{i=1}^{n} I\left(u(x_i) \le \tau\right).$$

By selecting a number of samples according to the Chernoff bound

$$n \ge \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}$$

we have that

$$\Pr\left(|\hat{p}_n - p_\tau| \le \varepsilon\right) \ge 1 - \delta$$

holds for any accuracy level $\varepsilon \in (0, 1)$ and confidence $\delta \in (0, 1)$. Value $\hat{p}_n$ is the probabilistic outcome of the performance satisfaction problem provided that small ε and confidence δ values are set. The randomized algorithm solving the problem is Algorithm 6.

7.3.2 The Figure of Merit Expectation Problem

The performance satisfaction problem returns the percentage of points satisfying a given bound on the tolerated performance associated with my solution. The expectation problem aims at evaluating the expected value of the loss function.

The problem solves the cases where

- We are interested in quantifying the expected performance loss having moved from solution $y(x)$ to $\hat{y}(x)$.
- We are interested in quantifying the expected performance of our application $\hat{y}(x)$ for a given problem, e.g., in execution on our embedded system. In such a case $u\left(y(x), \hat{y}(x)\right) = u\left(\hat{y}(x)\right)$.

In the following to ease the derivation assume that $u(x)$ is defined in the $[0, 1]$ interval. This normalization operation is simply done with a rescaling of function $u(x)$.

Define $E[u(x)]$ to be the expectation of function $u(x)$ according to f_x and follow the derivation given in Sect. 4.4.3 for the evaluation of the expectation problem with randomization. Briefly, set accuracy level $\varepsilon \in (0, 1)$, confidence $\delta \in (0, 1)$ and draw

$$n \geq \frac{1}{2\varepsilon^2} \ln \frac{2}{\delta}$$

i.i.d. samples $x_1, \ldots, x_n$ from random variable x defined over X according to f_x and generate the estimate

$$\hat{E}_n(u(x)) = \frac{1}{n} \sum_{i=1}^{n} u(x_i)$$

then,

$$\Pr\left(|\hat{E}_n(u(x)) - E[u(x)]| \leq \varepsilon\right) \geq 1 - \delta.$$

Value $\hat{E}_n(u(x))$ is the probabilistic outcome of the algorithm. By selecting small ε and δ, we obtain a good approximation. The randomized algorithm solving the problem is given in Algorithm 9.

The interesting reader should refer to Sect. 4.4.3 for the interesting relationships between the needed samples requested by the Chernoff bound and those needed from the central limit theorem.

7.3.3 The Maximum Performance Problem

The maximum performance problem aims at estimating the maximum value function $u(x)$ can assume. Its relevance in many application cases is immediate. As we have seen in Sect. 4.4.2 (were all details are given) the problem requires evaluation of

$$u_{\max} = \max_{x \in X} u(x)$$

and its analytical determination is impossible for most of functions. We saw in Sect. 4.4.2 that a manageable probabilistic version of it requires two levels of probability. From the operational point of view, once fixed accuracy ε and confidence δ, we have to draw

$$n \geq \frac{\ln \delta}{\ln(1-\varepsilon)}$$

i.i.d. samples $x_1, \ldots, x_n$ and generate the estimate $\hat{u}_{\max}$

$$\hat{u}_{\max} = \max_{i=1,\ldots,n} u(x_i)$$

then,

$$\Pr\left(\Pr\left(u(x) \geq \hat{u}_{\max}\right) \leq \varepsilon\right) \geq 1-\delta.$$

The estimate of the maximum performance level is $\hat{u}_{\max}$. All comments raised in Sect. 4.4.2 hold. The reference randomized algorithm solving the maximum performance problem is given in Algorithm 8.

7.3.4 The PACC Problem

We have seen that the PACC problem requires the evaluation of τ and η so that

$$\Pr\left(|y(x) - \hat{y}(x)| \leq \tau\right) \geq \eta \tag{7.8}$$

holds. Then, if τ is small $y(x) \simeq \hat{y}(x)$ with probability η. The problem can be solved by considering the figure of merit $u(x) = |y(x) - \hat{y}(x)|$, although a more general discrepancy function could be considered.

Solution of the PACC problem for a given function requires the evaluation of τ for which (7.8) holds with high probability. This problem can be addressed by using the method proposed in Sect. 4.4.1 and here reproposed in its main steps. Define $p(\gamma)$ as

$$p(\gamma) = \Pr\left(u(x) \leq \gamma\right) = \Pr\left(|y(x) - \hat{y}(x)| \leq \gamma\right)$$

for an arbitrary but given γ. Estimate $p(\gamma)$ according to method given in Algorithm 6 which returns $\hat{p}_n(\gamma)$. Explore the γs by selecting arbitrary incremental points and generate set $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ s.t. $\gamma_i < \gamma_j \; \forall i < j$. For all $\gamma \in \Gamma$ evaluate $\hat{p}_n(\gamma)$ so as to build

$$\hat{p}_\Gamma = \{\hat{p}_n(\gamma_1), \dots, \hat{p}_n(\gamma_k)\}$$

as done in Algorithm 7.

Define $\bar{\gamma}$ to be the smallest value for the finite sequence $\{\gamma_1, \dots, \gamma_k\}$ for which $\hat{p}_n(\gamma_i) = 1, \forall \gamma_i \geq \bar{\gamma}, \gamma_i \in \Gamma$. Having selected k according to the Chernoff bound discrepancy

$$|p(\gamma) - \hat{p}_n(\gamma)| \leq \varepsilon$$

holds with probability $1 - \delta$ and is satisfied for all γ. As such, it also holds for $\hat{p}(\bar{\gamma})$. We have that

$$\hat{p}_n(\bar{\gamma}) - \varepsilon \leq p(\bar{\gamma}) \leq \hat{p}_n(\bar{\gamma}) + \varepsilon,$$

i.e.,

$$1 - \varepsilon \leq p(\bar{\gamma}) \leq 1$$

from which $p(\bar{\gamma}) \geq 1 - \varepsilon$. The PACC problem is solved and provides $\eta = 1 - \varepsilon$ and $\tau = \bar{\gamma}$.

7.3.5 The Minimum(Maximum)-Perturbed Expectation Problem

The minimum (maximum) expectation problem aims at estimating the minimum (maximum) value of the expectation of $u(x)$ when perturbations Δ affects $u(x)$, thus providing a perturbed version of the performance function $u(x, \Delta)$. Examples of applications are the following

- Part of my embedded application has been designed with an analog technology which is subject to electronic noise. I would like to know the minimum(maximum) performance of my device subject to the fact perturbations affect it. I am happy to know the minimum(maximum) performance by taking expectation of the perturbation space.
- There is a source of uncertainty affecting my embedded system and I would like to obtain an estimate of the performance loss by taking the average over the perturbation space.

The formalization of the problem is as follows:

Consider $u(x, \Delta) \in [0, 1]$, $x \in X \subset \mathbb{R}^d$ and $\Delta \in D \subset \mathbb{R}^k$ (being X and Δ probability spaces) and focus, e.g., on the minimum problem

$$u_{\min} = \min_{x \in X} E_{\Delta}[u(x, \Delta)]$$

which is equivalent to

$$\begin{cases} \phi(x) = E_{\Delta}[u(x, \Delta)] \\ u_{\min} = \min_{x \in X} \phi(x) \end{cases}$$

solution to this computationally demanding problem is given in Sect. 4.4.4 and addressed by randomized algorithm given in Algorithm 10.

7.4 Accuracy Estimation: A Given Dataset Case

In some of previously discussed performance verification problems we were able to assess the quality of an estimator in probabilistic terms. For instance, in the figure of merit expectation problem discussed in Sect. 7.3.2, by extracting n data

$$x_1, \ldots, x_n$$

from random variable x according to f_x over X, we where able to provide an estimate $\hat{E}_n(u(x))$ for the true unknown expectation $E[u(x)]$. At the same time, we provided a quality assessment of the estimator since we wrote that

$$\Pr\left(|\hat{E}_n(u(x)) - E[u(x)]| \leq \varepsilon\right) \geq 1 - \delta. \tag{7.9}$$

The designed framework implicitly permits that we can extract sequences of arbitrary length n from X so that we can meet both accuracy ε and confidence δ according to (7.9).

However, in the real life we generally encounter situations where n is given (e.g., we have n finite samples from an industrial process, n models built on the sensor data stream, etc). Whenever that is the case, (7.9) clearly must hold but n is now given and δ and ε cannot assume any arbitrary values any more. If we assume that δ must be fixed since we want our results to hold at some confidence level, e.g., $\delta = 0.95$, then, accuracy is no more arbitrary and is set as

$$\varepsilon = \sqrt{\frac{1}{2n} \ln \frac{2}{\delta}}$$

by simply inverting the Chernoff bound. Unfortunately, if n is not large enough the accuracy we should expect might be poor.

Two comments need to be made at this point. The first is that results derived by invoking bounds such as the Chernoff one are pdf free and, as such, the bound might be rather conservative. As a consequence, the derived ε might be hardly usable if the number of samples is fixed and not large enough. Second, we should remind that, given a finite number of samples, we cannot pretend to have an arbitrary accuracy since the amount of information carried by the data set is finite and depends on n. As an example, we recall that the estimated standard deviation of mean $\hat{x}$ based on n i.i.d scalar data $x_1, \ldots, x_n$ extracted from a distribution of unknown mean μ and known variance σ^2, is $\sqrt{\frac{\sigma^2}{n}}$. The quality of the estimator $\hat{x}$ scales with $n^{\frac{1}{2}}$ and cannot be improved unless we increase the number of samples n. In following subsections, we wish to assess the quality of an estimate in the case n is given.

7.4.1 Problem Formalization

Consider a data set Z_n obtained by extracting n i.i.d. samples $x_1, \ldots, x_n$ from random variable x defined over X, i.e., $Z_n = \{x_1 \ldots, x_n\}$ and construct the estimator $\Phi_n = \Phi(Z_n)$. We are interested in providing an indication of the quality ζ of Φ_n, e.g., we wish to provide a confidence interval for Φ_n.

The problem is of extreme relevance in many applications and, in particular, in embedded systems where we wish to carry out the performance evaluation of a figure of merit (estimator) having a finite and given data set to be used to compute ζ. The performance verification problem is the same introduced in this chapter with the unique difference that, here, n is fixed, e.g., we have only n i.i.d. data from sensors, n i.i.d. parameter models or extracted features, n time measurements related to the execution of a thread.

Clearly, the ideal framework would recommend to carry out the following procedure

1. Extract m independent data sets of cardinality n from X so as to generate datasets $Z_n^1, \ldots, Z_n^m$;
2. Evaluate, in correspondence of the generic i-th data set Z_n^i the estimator $\Phi_n^i = \Phi(Z_n^i)$. Repeat this procedure for all $i = 1, \ldots, m$.;
3. Estimate the quality $\zeta\left(\Phi_n^1, \ldots, \Phi_n^m\right)$ of the estimator Φ_n based on the m realizations $\Phi_n^i = \Phi(Z_n^i), i = 1, \ldots, m$.

Unfortunately, the above framework is mostly theoretical: if we have m independent datasets Z_n we should use all nm data to provide a better estimate. This means that in practical applications we have only a dataset but, at the same time, we are interested in evaluating the quality ζ of the estimator Φ_n.

The literature introduces interesting approaches for providing an assessment ζ of the quality of an estimator Φ given a limited data set n.

7.4.2 The Bootstrap Method

In the bootstrap method [236] the needed m data sets $Z_n^i,\ i = 1, \ldots, m$ are extracted from Z_n with replacement. It means that, once a sample x_j has been extracted and inserted in the generic Z_n^i set, x_j is also placed back in Z_n that keeps all its original n data. Once all estimates $\Phi_n^i,\ i = 1, \ldots, m$ have been generated they are used to compute $\zeta\left(\Phi_n^1, \ldots, \Phi_n^m\right)$.

The Bootstrap algorithm is given in Algorithm 15. Efron and Tibshirani [236] proves that the distinct number of samples we should expect in Z_n^i is $\approx 0.632n$. This comment allows us for reducing the computational load associated with the execution of the algorithm. In fact, if we are expecting to receive approximatively $0.632n$ independent samples and the estimator requires to compute point wise terms, e.g., $u(x_i)$ for sample x_i given a generic $u(\cdot)$ function, then, there is no need to compute all n values and a weighting approach can be consider.

Algorithm 15: The bootstrap algorithm

```
i = 0;
while $i < m$ do
    Extract $n$ samples with replacement from $Z_n$ and insert them in $Z_n^i$;
    Compute $\Phi_n^i = \Phi(Z_n^i)$;
    i = i+1;
end
Evaluate the assessment $\zeta\left(\Phi_n^1, \ldots, \Phi_n^m\right)$ of the quality of the estimator $\Phi_n$.
```

Example: The Bootstrap Method

Consider, as a straight example, the figure of merit expectation problem discussed at the beginning of the section. Differently from the derivation based on the Chernoff bound, we do not need to require here that the Lebesgue measurable function $u(x)$ is defined in interval [0, 1] but it is sufficient that $u(x)$ is bounded for some value (not to be necessarily known).

Select $\Phi = E[u(x)]$ and $\Phi_n = \hat{E}_n(u(x))$ evaluated on Z_n. The estimate of $E[u(x)]$ we have is $\hat{E}_n(u(x))$. We are now interested in evaluating the quality of the estimate ζ, e.g., by evaluating the variance of the estimates generated with the bootstrap method. This is done by invoking Algorithm 15.

For instance, the quality of the estimator, here considered to be the variance of the estimator evaluated according to the bootstrap method Var_B, can be estimatedas

$$Var_B = \zeta\left(\Phi_n^1, \ldots, \Phi_n^m\right) = \frac{1}{m-1}\sum_{i=1}^{m}\left(\Phi_n^i - \Phi_n\right)^2 .$$

The bootstrap algorithm shows to be accurate for a wide range of estimators but is also characterized by a significant computational load. Variants of the bootstrap method have been suggested in the literature, e.g., the m out of n bootstrap [237] to solve the consistency issue in applications where the bootstrap fails. The computational issue has been addressed, among the others, in the m out of n bootstrap where $Z_{n'}^i$ sets are chosen to have a smaller cardinality than Z_n^i, i.e., $n' < n$, and in [238] where a method aiming at using a small m and extrapolation techniques has been proposed and in the bag of little bootstrap method [239], which will be detailed in the next subsection.

7.4.3 The Bag of Little Bootstraps Method

The Bag of Little Bootstraps (BLB) method is a bootstrap-inspired method proposed in [239] to mitigate the problem posed by Bootstrap and associated with the required computational complexity. BLB shows to be accurate and appears to over-perform all other bootstrap methods in terms of computational complexity, hence becoming a very appealing method for Big Data.

Algorithm 16: The Bag of Little Bootstraps algorithm

```
$n' = n^\gamma$;
i = 0;
while $i < m$ do
    Extract $n'$ samples without replacement from $Z_n$ and insert them in $Z^i_{n'}$;
    j = 1;
    while $j < r$ do
        Extract $n$ samples with replacement from $Z^i_{n'}$ and insert them in $Z^i_{n,j}$;
        Compute $\Phi^i_{n,j} = \Phi(Z^i_{n,j})$;
        j=j+1;
    end
    Evaluate $\zeta_i = \zeta\left(\Phi^i_{n,1}, \cdots, \Phi^i_{n,r}\right)$;
    i = i+1;
end
Evaluate the assessment $\zeta$ for of the quality of the estimator $\Phi_n$ as $\zeta = \frac{\sum_{i=1}^m \zeta_i}{m}$.
```

The BLB algorithm is given in Algorithm 16. The starting point of BLB is to select a smaller data set of cardinality $n' < n$. Authors select $n' = n^\gamma$, e.g., with $\gamma = 0.6$. Then, m subsets $Z^i_{n'}, i = 1, \ldots, n'$ are extracted from Z_n without replacement. For each subset $Z^i_{n'}$ r subsets $Z^i_{n,j}, j = 1, \ldots, r$ are generated with replacement (little bootstraps) and the corresponding estimators $\Phi^i_{n,j} = \Phi(Z^i_{n,j})$ evaluated. Finally, the quality of the estimator is evaluated for each little bootstrap and yields $\zeta_i = \zeta\left(\Phi^i_{n,1}, \ldots, \Phi^i_{n,r}\right)$. At the end of the procedure, we end with m "bags", the i-th one associated with quality assessment ζ_i. The final assessment ζ of the quality of the estimator Φ_n is considered to be the ensemble of the little bootstrap one as $\zeta = \frac{\sum_{i=1}^m \zeta_i}{m}$. r is generally chosen so that ζ_i ceases to fluctuate and, in general, this happens in correspondence with small values of r, e.g., refer to [239].

As mentioned, the BLB shares the consistency properties of the bootstrap and higher order correctness under the same hypotheses assumed by the bootstrap.

As claimed in [239], the BLB shares the fast convergence rate of the bootstrap where the procedure's output scales as $O(\frac{1}{n})$ instead of the $O(\frac{1}{\sqrt{n}})$ rate achieved by asymptotic approximations. This fast convergence rate assumes that $n' = \Omega(\sqrt{n})$, i.e, $\lim_{n\to\infty} sup|\frac{n'}{\sqrt{n}}| > 0$, and m is large enough compared with the variability observed in data samples. Satisfaction of such assumption grants that n' is significantly smaller than n (but larger than $\sqrt{n}$). Moreover, as shown in [239], the BLB is faster than the bootstrap even in a serial execution. We comment that the algorithm can be easily made parallel since each little bootstrap procedure can be parallelized.

7.5 Cognitive Processes and PACC

All mechanisms involving cognitive processing, e.g., those carried out by VM-PFC—ACC are real and, although complex and mostly unknown to us in detail, Lebesgue measurable. The PACC theory can hence be applied to cognitive processing. It is extremely natural to assume that those (sub)systems operate in probability given the highly uncertainty associated with the processing. Our actions are, in fact, mostly correct and the output to an emotion or an action mostly correct, with high probability: exactly what the PACC theory is about.

7.6 Example: Accuracy Assessment in Embedded Systems

The section aims at showing how the accuracy assessment framework can be utilized in embedded systems to assess the quality of a proposed solution and drive the designer toward the identification of the most appropriate one within a given set. The reference application is based on a neural network that, once designed in a high precision framework, needs to be ported onto an embedded system. The relevance of the example is in the diversified computation carried out by the neural computation that requires evaluation of scalar products as well as nonlinear functions.

The porting operation introduces several sources of structural perturbations whose impact on accuracy can be effectively evaluated with the framework presented in the chapter.

In particular, the chosen application refers to a neural network designed to provide a virtual sensor for the chemical process described in the `Mathworks Matlab` neural network toolbox (chemical_dataset).

The virtual sensor is inferred from the readings of other eight sensors, whose values provide the inputs of the network. The chosen network topology is feedforward, with 8 inputs ($x \in X \subset \mathbb{R}^8$), 10 neurons in the hidden layer characterized by a hyperbolic tangent activation function, and a single linear output neuron. Once the neural network was successfully trained, neural function $f(\theta, x)$ was requested to be implemented on the embedded systems.

To illustrate the effect of structural perturbations induced by the embedded system architecture on function $f(\hat{\theta}, x)$, we consider two different digital embedded implementations for the neural network that, once ported, become the approximated $\hat{f}(\theta, x)$. The first implementation is based on a 16 bits word-length solution, the second on a 8 bits one.

In this way, we investigate the performance in accuracy of the two architectural solutions, an operation which also allow us for selecting the final target platform depending on the requested accuracy, the power consumption as well as the requested area (e.g., think of an ASIC or a FPGA implementation).

Results were emulated on a 32 bits ARM Cortex M3 microcontroller. Since the ARM Cortex M3 microcontroller is not equipped with a Floating Point Unit (FPU),

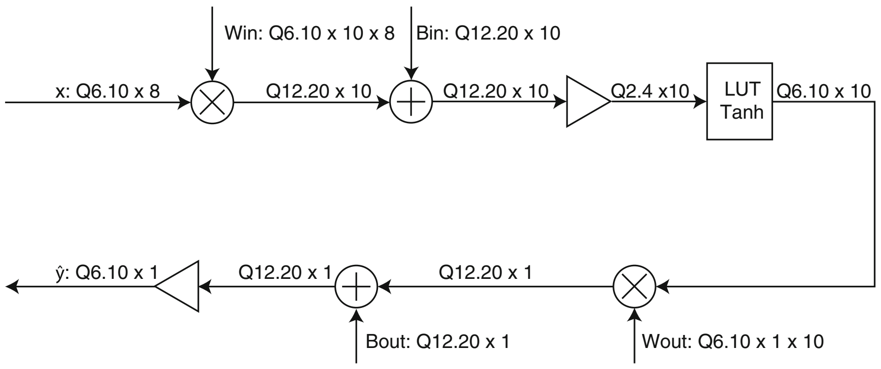

Fig. 7.1 The neural network data flow ported on a 16 bits architecture. The activation value u feeding the hidden units can be obtained with a matrix product between the 10×8 matrix Win containing the weights between the input and the hidden layer and the input column vector x; the bias value Bin is then added so that $u = Win\ x + Bin$. Inputs composing x are represented with a $Q6.10$ notation, weights Win with a $Q6.10$ notation, the outcome of their product on a $Q12.20$ notation. The bias term coded as $Q12.20$ is added and the outcome u on $Q12.20$ is reduced to $Q2.4$ to feed the LUT. The $Th(u)$ value coming from the LUT is defined on $Q6.10$ and multiplied by the weights connecting the hidden layer with the output neuron. The bias is added and the output defined on a $Q12.20$ notation reduced to a 16 bits output in the form $Q6.10$

all computations must be performed with a fixed point 2cp representation as presented in Sect. 3.2.4.

In the following, a fixed point number is represented with the $Qx.y$ notation, which implies a 2cp representation on $x + y$ bits with x bits to the left of the fixed point (integer part, sign bit included) and y bits after the point (fractional part).

We recall that a sum between two numbers defined on $Qx.y$ does not modify the position of the radix point, but overflow might occur. Instead, a multiplication between numbers $Qx.y$ and $Qk.z$ generates a value represented as $Q(x+k).(y+z)$ whose radix point is shifted to the left of y positions compared to z. This effect must be taken into account if the final value needs to be brought back to a $Qx.y$ notation. Usually, multiplications take place with operands characterized by the same Q representation.

The data flows associated with the two 16 bits and 8 bits implementations are shown in Figs. 7.1 and 7.2, respectively. The complete description of the architectural operations is detailed in the caption of Fig. 7.1.

The 16 bit implementation uses the same $Q6.10$ representation for inputs, output, and weights. With such a choice, we are safe from any possible occurrence of overflows. The sum following the product of the generic input by the corresponding weight is still performed at the full resolution of $Q12.20$ bits. Differently, in the 8 bits implementation, we adopt different resolutions for inputs, outputs, and weights (input and output values are represented using a $Q2.6$ coding, while weights are represented with a $Q5.3$ one). The neuron biases are still represented at the full resolution of $Q7.9$. After the operations shifts are introduced to bring back the obtained

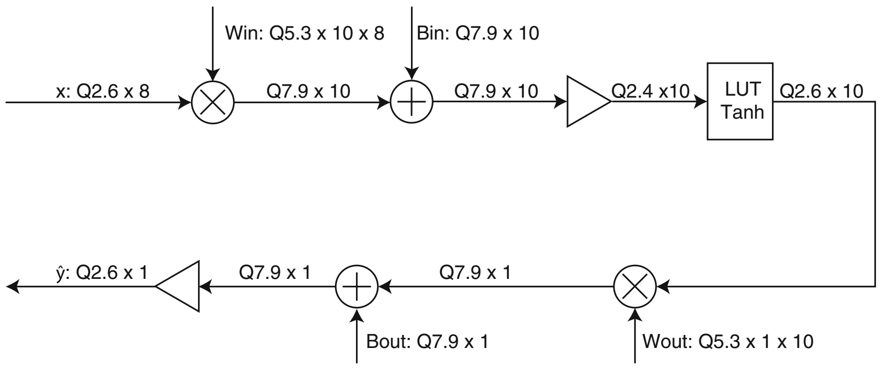

Fig. 7.2 The neural network data flow ported on a 8 bits architecture. The description of the flow is similar to that given in the caption of Fig. 7.1 with the suitable change of word length for the involved entities

numbers to the envisaged word-length. The embedded code for implementing the neural activation value ready to feed the nonlinear activation function of the hidden layer is given in Listing 7.1.

The evaluation of the nonlinear hyperbolic tangent function is particularly costly from the computational point of view. To deal with the issue, the best solution is to rely on a Look Up Table (LUT) memory enumerating the input–output relationship in correspondence of some points. The input of the memory is the neural activation value (i.e., the scalar product between the neuron inputs and the associated weights), the output is the content stored in the memory cells which represents the value the activation function assumes in correspondence of the input value. We aimed at keeping the size of the LUT as small as possible. As such, the input values where coded as unsigned $Q2.4$ values for a total of a 64 cells LUT memory; the output values follow the encoding used for the inputs and, as such, depend on the chosen architecture ($Q6.10$ and $Q2.6$ for the 16 and 8 bits architectures, respectively). We comment that the input of the LUT is unsigned. The reason is that the hyperbolic tangent (Th) is an odd function for which $Th(-u) = -Th(u)$. As such, we can represent the inputs by uniformly subdividing the interval [0, 4] (values above input 4 provide a saturated output at 1): there is no need to represent the full [−4, 4] interval with an immediate memory saving.

The approximation ability of the solution is given in Fig. 7.3 for the $Q6.10$ coding of the output.

The embedded code implementing the LUT is shown in Listing 7.2. As it can be noted in Fig. 7.3, the quantization error does not introduce a bias in the approximated function, since we opted for a rounding of the reduced argument instead of a simple truncation of the value. The cost of an extra shift and the sum needed to implement the rounding operator is well compensated by the disappearing of the bias term in the approximating the hyperbolic tangent function.

Listing 7.1: The C embedded code providing the activation value ”ured” to be inputted to the LUT in the 16 bits architecture. NEURONS=10, INPUTS=8 represent the number of hidden and input units, respectively. ”u” is the full precision activation value. The code returns the number of encountered overflows

```
int sumProductsIn(int16_t x[],int16_t ured[]){

  int i, j, overflow=0;
  int32_t u;

  for (i=0;i<NEURONS;i++){
    u=0;
    for (j=0;j<INPUTS;j++){
        u=u+Win[i][j]*(x[j]);
    }/*Win is a global array
    containing the input-hidden layer weights*/

    u=u+b1[i];
    overflow=overflow+reduxSums(u,&ured[i]);

  }
}

int reduxSums(int32_t arg, int16_t *ured){

/* reduces a Q12.20 value to a Q6.10 value*/

  int16_t temp;
  int32_t redux=(arg>>10);/*realign radix point*/

/*overflow management*/

  if (redux>32767){
        *ured=32767;
        return -1;
  }
  if (redux<-32768){
        *ured=-32768;
        return -1;
  }
  temp=redux&0xFFFF;
  *ured=temp;
  return 0;

}
```

Figures 7.4 and 7.5 compare the output of the virtual sensor in its two implementations with that evaluated on a high precision platform. As expected, the higher the number of bits made available the better the reconstruction performance. However, we comment that the 8 bits architecture provides a good reconstruction ability which lowers in correspondence with high peeks.

To quantify the accuracy level of the porting of the algorithm on the embedded systems, we considered the figure of merit

$$\hat{E}_N(u(x)) = \frac{1}{N}\sum_{i=1}^{N} |f(x_i,\theta) - \hat{f}(x_i,\theta)|.$$

Listing 7.2: Evaluation of the activation function value with a LUT. 16 bits architecture case. At first the activation value is transformed so as to be always positive. The rounding operator is introduced to remove the error bias otherwise present if a truncation operator is considered instead.

```
int16_t tanhQ16(int16_t u){
        int16_t out;
        uint16_t reduced;
        int16_t uc;
        /* leverage the antisymmetry of the
        hyperbolic tangent:
        Th(-u)=-Th(u) */
        if (u<0)
                uc=-u;
        else
                uc=u;
        reduced = ((uc+((uc&0x7F)>>1))>>6)&0xFF;
        /* reduce value from $Q6.10$ to $Q3.5$ with rounding
        to reduce the bias */
        if (reduced>63)
                out=(1<<10); /*saturation value*/
        else
                out=LUT[reduced];
                /* LUT is a 64 locations array
                containing the hyperbolic tangent
                values */
        if (u<0)
                return -out;
        else
                return out;
}
```

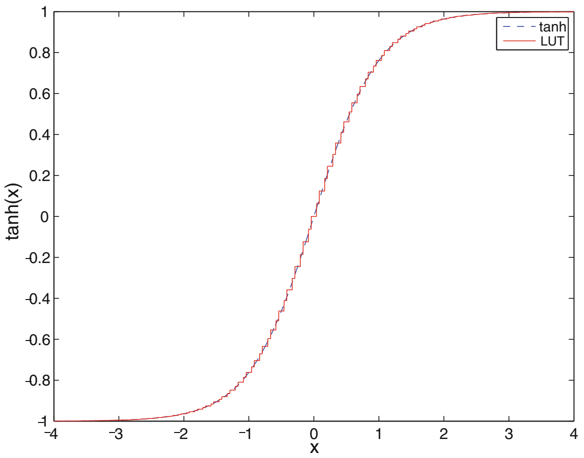

Fig. 7.3 The approximation accuracy of the hyperbolic tangent with a 64 cells LUT in the 16 bits representation for the output value

The accuracy performance assessment was carried out by following the figure of merit expectation problem delineated in Sect. 7.3.2 after rescaling the $u(\cdot)$ function. The input space was explored with a number of samples N drawn according to the Chernoff bound.

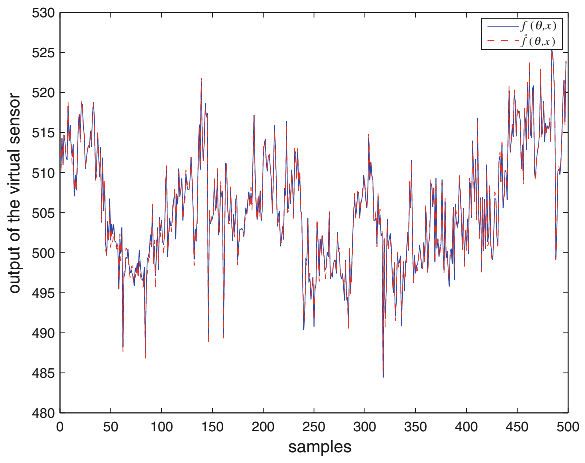

Fig. 7.4 The accuracy of the virtual sensor data stream provided by the 16 bits embedded architecture (function $\hat{f}(\theta, x)$) compared with the reference one (function $f(\theta, x)$)

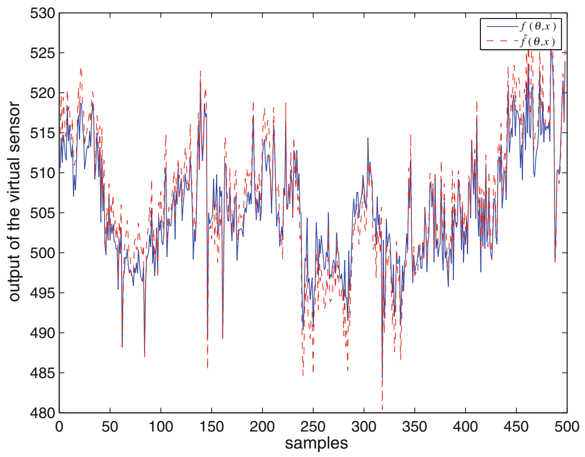

Fig. 7.5 The accuracy of the virtual sensor data stream provided by the 8 bits embedded architecture (function $\hat{f}(\theta, x)$) compared with the reference one (function $f(\theta, x)$)

The figure of merit expectation problem was solved by considering an incremental accuracy ε (the confidence value δ was fixed to 0.05). The numbers of points drawn according to Chernoff are given in Table 7.1. Since the distribution induced on the 8th dimensional input space in unknown we considered an uniform distribution. Each experiment was repeated 40 times. The expected value of the performance loss is given in Fig. 7.6 for the 16 bits architecture. Results show also the confidence interval based on one standard deviation. As expected, the larger the N the smaller the confidence interval.

Similar results are given in Fig. 7.7 for the 8 bits embedded architecture. As expected, the average error is higher for the 8 bits implementation.

Table 7.1 The number of points chosen to address the figure of merit expectation problem.

δ	ε	N
0.05	0.061	500
0.05	0.043	1000
0.05	0.035	1500
0.05	0.030	2000
0.05	0.025	3000
0.05	0.021	4000

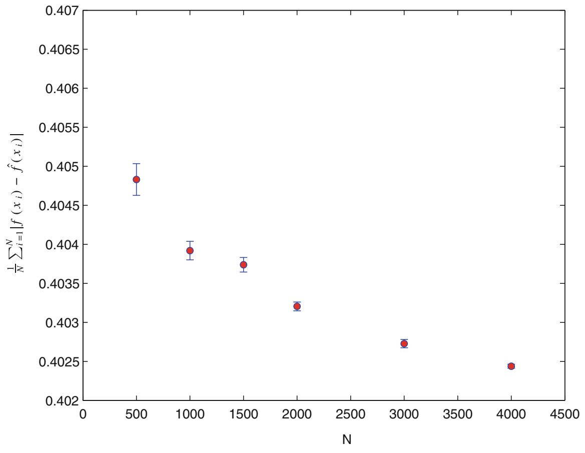

Fig. 7.6 The figure of merit expectation problem. The figure shows the performance loss associated with the porting of the virtual sensor neural network-based code on the 16 bits embedded architecture. Expected values and the confidence intervals are given

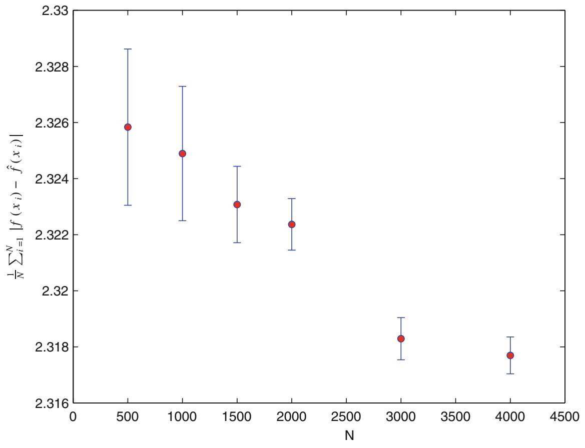

Fig. 7.7 The figure of merit expectation problem. The figure shows the performance loss associated with the porting of the virtual sensor neural network-based code on the 8 bits embedded architecture. Expected values and the confidence intervals are given

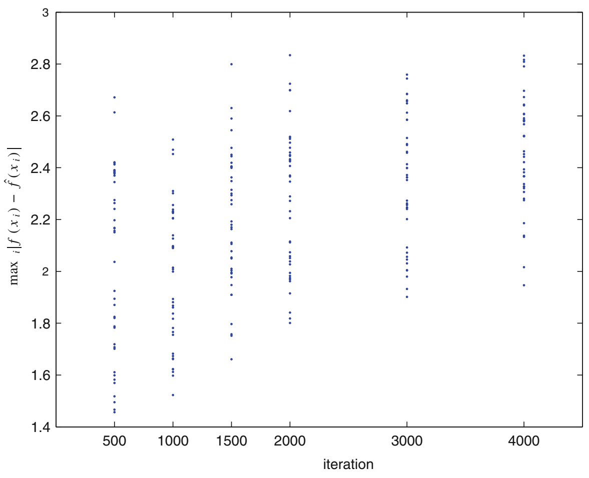

Fig. 7.8 The maximum performance problem for the 16 bits architecture. The experiment is repeated 40 times and each estimate of the maximum coming from the particular realization on N inputs is plotted

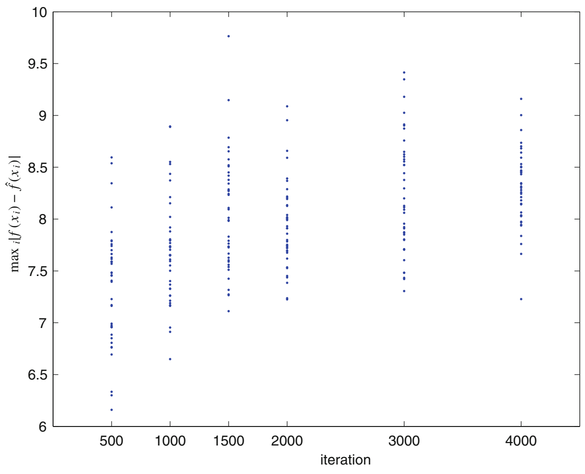

Fig. 7.9 The maximum performance problem for the 8 bits architecture. The experiment is repeated 40 times and each estimate of the maximum coming from the particular realization on N inputs is plotted

Figures 7.8 and 7.9 present the the maximum performance problem presented in Sect. 7.3.3 for the two embedded implementations. The experiment was carried out 40 times in correspondence with each chosen N ($\delta = 0.05$, $N \geq \frac{\ln \delta}{\ln(1-\varepsilon)}$); estimates of the maximum value $\hat{u}_{\max}$, $u(x) = |f(\theta, x) - \hat{f}(\theta, x)|$ are plot (one for each experiment run). No scaling to the [0, 1] interval is requested for $u(x)$. We comment that the variability of the estimates reduce as N increases and that the maximum error introduced by the 8 bits architecture is about four times that introduced by the 16 bits one. Depending on the tolerated accuracy loss and the given constraints on cost and power consumption, the designer might decide which architecture should be considered between the 16 and the 8 bits one.