New Optimal Solutions for Real-Time Scheduling of Operating System Tasks Based on Neural Networks

Ghofrane Rehaiem
Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST), Tunis El Manar University, TUNISIA
LISI-INSAT Laboratory (INSAT), Carthage University, TUNISIA
E-mail: rh.ghofrane@gmail.com

Hamza Gharsellaoui
National Engineering School of Carthage (ENIC), Carthage University, Tunisia
LISI-INSAT Laboratory (INSAT), Carthage University, TUNISIA
E-mail: gharsellaoui.hamza@gmail.com

Samir Ben Ahmed
Faculty of Mathematical, Physical and Natural Sciences of Tunis (FST), Tunis El Manar University, TUNISIA
LISI-INSAT Laboratory (INSAT), Carthage University, TUNISIA
E-mail: samir.benahmed@fst.rnu.tn

[1]

***Abstract*—This paper work deals with the implementation of a neural networks based approach for real-time scheduling of embedded systems composed by Operating Systems (OS) tasks in order to handle real-time constraints in execution scenarios. In our approach, many techniques have been proposed for both the planning of tasks and reducing energy consumption. In fact, a combination of Dynamic Voltage Scaling (DVS) and time feedback can be used to scale the frequency dynamically adjusting the operating voltage. In this study, Artificial Neural Networks (ANNs) were used for modeling the parameters that allow the real-time scheduling of embedded systems under resources constraints designed for real-time applications running. Indeed, we present in this paper a new hybrid contribution that handles the real-time scheduling of embedded systems, low power consumption depending on the combination of DVS and Neural Feedback Scheduling (NFS) with the energy Priority Earlier Deadline First (PEDF) algorithm. Experimental results illustrate the efficiency of our original proposed approach.**

***Index Terms*—Optimization, Neural Networks, Real-Time Scheduling, Low-Power Consumption.**

I. Introduction

The main use of neural networks is to classify possible results, from a list of informations. More generally, a neural network allows the approximation of a function. Let's take the example of a processor, it can adjust its speed in a continuous range to ensure that a task is submitted to, at most, a fault occurrence, and the cost of restoration of state is zero. In this paper, we want to discuss these issues to propose a heuristic approach based on neural networks (NN) which can handle real-time constraints allowing the real-time scheduling with minimization of power consumption of real-time embedded systems. A neuron is then the elementary processing unit of a neural network. It is connected to sources of information as input and returns output information. All these are organized through the learning method. In new technology, the use of fixed parameter method has allowed the development of most embedded detective devices, what is known as the mapping relationship between test input data and test conclusion. In this work, we must develop and implement an optimal approach that considers the following issues in the design which are the choice of the:

- network model,
- the number of input and output nodes,
- the number of hidden layer nodes,
- the choice of transfer function,
- the choice of initial weight,
- the choice of the learning rules,
- the preprocessed of input and output data in training sample in order to have a good solution balancing between the real-time scheduling and the low power consumption optimization of the embedded systems at the same time.

In this paper, Section 2 gives an overview of the neural network platform design and the neural feedback scheduling. Also, we describe the real-time dynamic voltage loop scheduling DVS. In Section 3, we present our PEDF heuristic and in Section 4 we shows our experimentation results. Finally, Section 5 concludes the paper work.

II. Background and Related Works

Neural network which can adapt the sample data by training has good fault-tolerance and can be used in the field of intelligence widely. In the embedded systems, restricted to the resources and the capacity of processor, the neural network application has some problems such as losing timeliness, also the system could be collapsed easily.

A. Neural Network Platform Design:

In practical applications, the input conditions of detection and the testing standards may need to be modified, which can damage the existing detection system. The neural network owns the ability of self-adaptation and self-learning. Therefore, by studying on the sample data, it can determine the mapping

[1]DOI reference number: 10.18293/SEKE2017-025

relationship between the input and the output and if the neural network is applied to the detection system, it may adjust the mapping relationship automatically by training samples, which can make the detection system more flexible and adaptable [(Bernard, 2008)]. The following paragraph is about the description of issues needed to be considered in the design.

1) The Choice of Network Model: The 2-layer linear perceptron model and the 3-layer Back Propagation (BP) network model are provided and the user can choose one of them in the network parameter settings in accordance with the complexity of the detection system. Based on the [(Hagan, 1996)] works, a 3-layer BP neural network with a hidden layer can approximate to any continuous function of bounded domain with arbitrary precision as long as there are enough hidden layer nodes. Therefore, in terms of the function, 3-layer BP neural network can meet most sophisticated detection systems' requirements, while for some detection systems which are simple mapping, linear perceptron model with a small amount of calculation and fast speed can be chosen so as to maximize the efficiency of the systems.

2) The Choice of the Number of Input and Output Nodes: The number of the input nodes is determined by testing item. The number of the output nodes is generally determined by the number of the testing conclusions. But, if there are a lot of testing conclusions, the number of the output nodes could become great, then, the amount of calculation also increases. So, when there are a lot of testing conclusions, the testing conclusions can be encoded to binary code, then output nodes take the number greater than or equal to $log_2(N)$, where N is the number of conclusions [(Liu, 2011)].

3) The Choice of the Number of Hidden Layer Nodes:: If the network model is Back Propagation (BP) network, the number of the hidden layer nodes needs to be set. Generally speaking, if there are too few hidden layer nodes, the network can not study well, and much more training will be needed and, meantime the training will not be highly precise; while too many nodes will bring issues such as a large amount of calculation, long training time and reduced network fault-tolerance capacity. System uses the default settings as the empirical formula.

$$n_1 = \sqrt{n+m} + 2 \tag{1}$$

In the formula 1, n_1 is the number of the hidden layer nodes; n is the number of the input nodes; *m* is the number of the output nodes. If the training effect of experience value is not good, we can reset the number of the hidden layer nodes in the parameter settings and do many experiments in order to achieve faster convergence rate. In order to know the training efficiency, you should output the training number in training process, changes of the network errors, changes of network weights and the training time, so as to make a report for the network analysis [(Liu, 2011)].

4) The Choice of Transfer Function: In the application of transfer function detection, most of the outputs are the field data and test conclusions. The field data are collected by the acquisition system; detection conclusions are obtained by the aplication of the neural network algorithm. The detection conclusions are indicated with two-value, so the transfer function of output layer generally uses the sigmoid function [(Liu, 2011)] which is presented as follows:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

The transfer function is a function that should return a real close to 1 in the presence of "good" input information and real close to 0 when they are "bad". Generally, we use functions with values in the interval of real [0, 1]. When the real is close to 1, the neuron is called "active" whereas when the real is close to 0, we say that the neuron is inactive. The real in question is called the output of the neuron. If the activation function is linear, the neural network would be reduced to a simple linear function. The sigmoid function has the advantage of being differentiable as well as giving intermediate values (real between 0 and 1). However, this function have a threshold that is 1/2 when x = 0.

5) The Choice of the Learning Rules: Due to the particularity of embedded systems, we should try to select the learning algorithm with simple calculation and low memory consumption. For this reason, and to improve the efficiency, a flexible BP learning algorithm may be adopted. It was only needed to consider the symbol of gradient to the error function, rather than the increasing amplitude of the gradient. The symbol of gradient determines the direction of weight updating, and the change of the weight size is determined by an independent "update value" [(Hagan, 1996)]. The iterative process of weight correction example is shown below:

$$\omega(t+1) = \omega(t) + \alpha\Delta\omega(t) \times sgn\frac{\partial E_t}{\partial \omega_t} \tag{3}$$

In the formula 3 [(Tian, 2009)], $\Delta\omega(t)$ is the previous update value, and the initial value $\Delta\omega(0)$ is set by the actual application. α is the learning parameter, using variable step-size learning.

6) The Preprocessing of Input and Output Data in Training Sample: In the network learning process, because the transfer function of the neuron is a bounded function, while in the detection, as the dimension and unit of input test item data may be different, some values are very large, while others are very small, which has a great influence on the network. To prevent some neurons from reaching saturation state, and meantime make the larger input fall in the region where gradient of neuron activation function is large, input and output vectors need to be normalized before the training.

$$x_i' = \frac{x_i - x_{imin}}{x_{imax} - x_{imin}} \tag{4}$$

In the formula 4, x_{imax} and x_{imin} are the maximum value and the minimum value of each input component of the neuron number *i*; x_i, x_i' are respectively the former and later input component after pretreatment of the neuron number *i*. The input data values after normalization processing range between

0 and 1. If the neural network training is unsuccessful, or if the training time is too long, process of training must be analyzed to train after readjusting the network parameters. If the training results are satisfactory, the testing sample data may be used to test the function of the network. Successful testing demonstrates that the construction of the network is completed, and then the network parameters and the weights of each layer of the network are saved as files, ready for testing procedures.

B. Neural Feedback Scheduling

The basic idea of the neural feedback scheduling (NFS) is to allocate available resources dynamically among multiple real-time tasks based on feedback information about actual resource usage. To tackle the problem associated with the large computational overheads of optimal feedback scheduling algorithms, a NFS scheme must be proposed. The goal is to optimize the overall Quality of Control (QoC) of multitasking control systems through feedback scheduling while minimizing the feedback scheduling overhead [(Xia, 2008)].
On one hand, this scheme has advantages such as low feedback scheduling overhead, wide applicability, and intelligent computation. It is also capable of delivering almost optimal QoC. On the other hand, neural networks are powerful in learning and adapting, and capable of approximating complex nonlinear functions with arbitrary precision [(Hagan, 1996)], [(Zhu, 2006)]. Once well trained using the accurate optimal solutions at design time, neural networks will be able to deliver online almost-optimal feedback scheduling performance.

C. Real-time Dynamic Voltage Loop Scheduling

Low power is extremely important for real-time embedded systems. A real-time loop scheduling techniques were proposed to minimize energy consumption via Dynamic Voltage Scaling (DVS) for applications with loops considering transition overhead. Two algorithms, Iterative Dynamic Voltage Scheduling (IDVS) and Dynamic Voltage Loop Scheduling (DVLS), are designed integrating with dynamic voltage scaling (DVS). IDVS is an algorithm to iteratively optimize the Directed Acyclic Graph (DAG) part of a loop by incorporating transition overhead into optimization scheduling scheme. DVLS is an algorithm to repeatedly regroup a loop based on rotation scheduling and decreases the energy by DVS as much as possible within a timing constraint. DVS is one of the most powerful techniques to reduce energy consumption by adjusting supply voltage at running time.

III. CONTRIBUTION

In this section, we describe our proposed approach and we give its notation to be more clear. So, to do this, we will present how to use processor and a limited memory to perform a neural network algorithm in order to improve adaptability in real-time embedded systems. In the application of the neural network, we need to enter a large number of training data and output of the training process and the results of the training, which requires a higher demand for input and output. Our main goal is about minimizing the number of neurons in order to facilitate the implementation which will be adapted later. In this context, we must ensure a low complexity and fast convergence and as a consequence the number of neurons must be significantly reduced. Therefore, new building regulations have been proposed to design the smallest possible neural network in order to optimize the scheduling of reconfigurable embedded systems in real-time

A. Formalization

Embedded applications are implemented in a complex SOC (System-on-Chip). Other resources, such as cores or dynamically reconfigurable accelerators need to be controlled by the OS. In particular, an instantiation of the task execution resources is performed using the OS scheduling service. As each task can be defined for multiple targets, this service has to decide at run-time, on what resource the task must be instantiated. Neural networks have demonstrated their efficiency in optimization constraint problems with the ability to converge in a reasonable time (i.e., few cycles) if the number of neurons and connections between them can be limited as much as possible. It should be noted that when the network converges to a stable state which does not belong to the set of valid solutions, this network need to be reinitialized.

1) Scheduling Modeling Through Neural Network:: In context of SoC architecture, service implementations for task scheduling are often complex and are not always suitable for real-time systems because they are usually time costly and they do not consider the dynamic behavior of the application. Our solution uses Artificial Neural Networks (ANN) for online real-time scheduling, where we have chosen the Hopfield model [(Hopfied, 1985)] to ensure network convergence to a stable state, while respecting the optimization constraint. This function is defined as follows:

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} T_{i,j} \times x_i \times x_j - \sum_{i=1}^{N} I_i \times x_i \qquad (5)$$

- $T_{i,j}$ is the connection weight between the neurons *i* and *j*.
- x_i is the state of the neuron *i*.
- I_i is the external input of the neuron *i*.

Based on this model and by using an optimization function of the constraints, a design rule can be defined in order to facilitate construction of the neural network. The rule k-out-of-N is a major result in ANN for optimization. This rule allows the construction of *N* neurons for which the evolution leads to a stable state with "exactly *k* active neurons among *N*". The energy function is defined as follows:

$$E = (k - \sum_{i=1}^{N} x_i)^2 \qquad (6)$$

This function is minimal when the active neuron sum is equal to k, and is positive otherwise. The results of this scheduling solution in real-time demonstrate the interesting

convergence speed which makes ANN suitable for online utilization. However, this technique has two major weaknesses. The first is the large number of slack neurons needed to model the problem, which depends on the cycle, so that when the schedule time increases, the number of slack neurons also increases. The second problem is the presence of several local minima when many rules are applied to the same set of neurons. These local minima are particular points of the energy function representing invalid solutions. Our work is to considerably reduce the number of hollow neurons for any period. The principal consequence is the simplification of network control. It is clear at this point that our proposition is guided by a main objective which consists of reducing the number of neurons while ensuring the convergence of the network.

∗ **Monoprocessor architecture:**

In the case of monoprocessor architecture, the scheduling problem is modeled through ANN by the following representation:

Neurons n_{ij} are organized in a matrix form, with the size N_T x N_C, where line *i* represents the task τ_i and the column *j* corresponds to schedule time unit *j*. The number of time units N_C is the least common multiple of all the task periods and N_T is the number of tasks.

- A neuron n_{ij} is considered active when the task τ_i is being executed, during the corresponding time unit *j*.
- One line of neurons is added to model the possible inactivity of the processor during the schedule times. These neurons are called slack neurons.

∗ **Multiprocessor architecture:**

In the case of homogeneous multiprocessor architecture, several matrices arranged in layers are required to model the different execution resources. New slack neurons are then needed to manage the execution of each task on resources. For each couple (task τ_i, resource *j*), $C_{i,j}$ new slack neurons should be added. So, the total number of slack neurons is equal to:

$$\sum_{i=1}^{N_T} \sum_{j=1}^{p} C_{i,j} + p \times N_C \tag{7}$$

Running example of network with *p* resource layers is presented in Figure 1. Grey circles represent slack neurons.

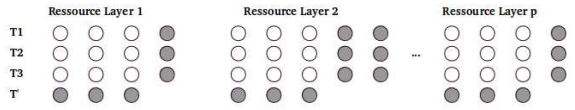

Fig. 1. Classical Structure to Model the Scheduling Problem with ANN.

To reduce the required number of neurons, we must modify neural network structure. The changes correspond to an adaptation of the Hopfield model. We have adopted the idea of creating a mutual exclusion between the possible task instantiations on execution resources [(Chillet, 2007)]. This mutual exclusion is provided by the presence of an inhibitory neuron $IN_{i,j}$ for the task τ_i and the execution of resource *j*. In Figure 2, an example of the scheduling problem is presented with one task τ_i and *R* possible resources.

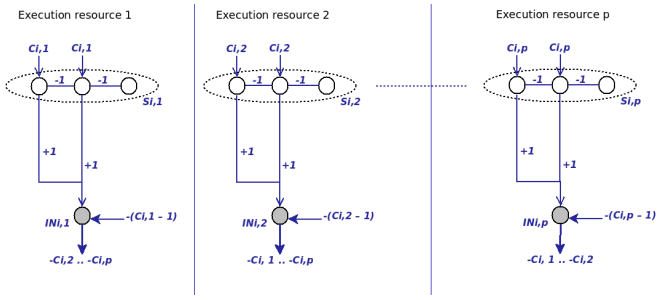

Fig. 2. Scheduling Problem Modeled with Inhibitor Neurons.

A set of N_C neurons is called $S_{i,j}$ (N_C = 3 in our use case example) and represents the possible scheduling cycles of the task τ_i on the resource *j*. For each resource j, the Worst Case Execution Time (WCET) of the task τ_i is defined as $C_{i,j}$. The set of neurons S_j is configured (definition of inputs and weights) to converge towards $C_{i,j}$ active neurons among N_C. The main characteristic of this neuron network is its capacity to converge to a stable state from any initial state. One or more lines of slack neurons can be added to ensure the network convergence during the application of k-out-of-N rule on each vertical line of neurons. As shown in Figure 1, the number of added lines in each layer is equal to the number of identical processors in this layer (In our example, one resource can execute the tasks, so one line of slack neurons are added, line *T'*). In this case, the convergence can not be always obtained.

To delete these lines, we choose the application of a k1-out-of-N1 classical rule on the horizontal sets of neurons and a at-most-k2-among-N2 rule on the vertical sets of neurons. If N_T tasks must be scheduled on *p* identical resources (*p* processors in the same layer) during the N_C cycles, N_C at-most-*p*-among-N_T rules must be applied on each vertical set of N_T neurons.

There remains the problem concerning the application of two rules on both sets of neurons with a common neuron. Figure 3 shows an example of this case, where the first set of neurons is composed of three horizontal neurons n_1, n_2, n_3, and the second is composed of three vertical neurons n_1, n_4, n_5 (Neuron n_1 is the common neuron of the two sets). The classical additive for various rules mentioned that if a $k1$-out-of-3 rule is applied on the first set and at-most-$k2$-among-of-3 rule is applied on the second set, then the inputs and weights are defined as follows:

-Inputs are equal to:

$I_1 = (2 \times k_1 - 1) + (2 \times k_2 - 1)$

$I_2 = I_3 = (2 \times k_1 - 1)$

$I_4 = I_5 = (2 \times k_2 - 1)$

- Weights are equal to $w_{i,j} = -2$; $\forall i, j = 1..5$

$k2$ slack neurons can be added with a specific weight

connection with other neurons in order to ensure the at-most-$k2$-among-of-3 rule. In the figure 3, the slack neurons (S_{n1}, S_{n2}) are represented.
To remove the slack neurons while ensuring convergence, we need to simplify the k-out-of-N rule by adopting the simple redefinition of input and weight values. The energy function given in the equation 6, energy was rewrited as $E = (\frac{1}{\sqrt{2}}k - \frac{1}{\sqrt{2}}\sum_{i=1}^{N} x_i)^2$.
With these new input values and weight, we can suggest a simple additive between $k1$-out-of-$N1$ and at-most-$k2$-among-$N2$ rules. The main idea is to apply the rule $k1$-out-of-$N1$ rule at first on horizontal lines and $k2$-out-of-$N2$ rule on the vertical lines and secondly the change of weight of horizontal lines.
Thus, the common neuron has its input set at the value $k1 + k2$ as shown in Figure 3. The change of the weight values of the horizontal rule (here the horizontal ruler $k1$-out-of-3) compensates the increase of input values on the common neuron. This compensation is done by decreasing the weight value between the horizontal neurons and the common neuron between the two rules. An example of additive of the two rules is given in Figure 3. We present

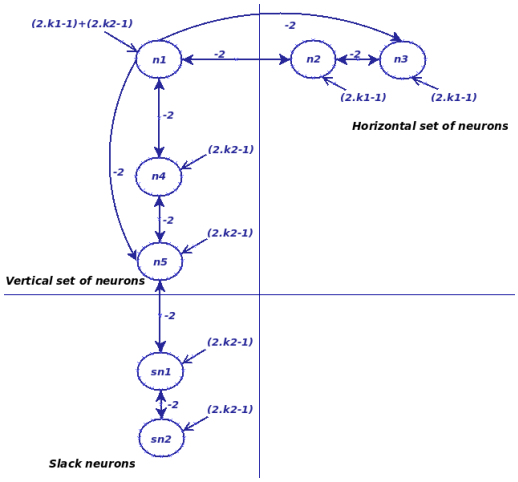

Fig. 3. Additive of k_1-out-of-N_1, and at-most-k_2-among-N_2 Rules with Slack Neurons.

an online scheduling algorithm for real-time systems that attempts to minimize the energy consumed by a periodic task set. This is based on the well-known earliest-deadline-first (EDF) algorithm [(Liu, 1973)]. We attempt to find the voltage at which each task must be executed such that the energy consumed by the entire set of periodic tasks is minimized and generate a schedule for the task set such that the release time requirements are satisfied and the deadline for each task is met. Although the scheduling methods cited above are very efficient, most of them make the assumption that the Central Processor Unit (CPU) can operate at several different voltage levels (and hence different clock frequencies) which can be varied continuously. In addition, a number of these methods are aimed at the synthesis of low-power designs and they do not address energy minimization during field operation.

The optimization problem that we address in this work, is to minimize the total energy consumed by the set of *n* tasks by optimally determining their start times, their voltages and corresponding execution speeds at the real-time scheduling. The following constraints need to be modeled:
(i) CPU speeds are limited to one of three values S_1, S_2 and speed-medium $S_3 = S_m = (S_1 + S_2)/2$,
(ii) The deadline of each task must be met,
(iii) Tasks are non preemptable,
(iv) A task may start only after it has been released.
The objective function is to minimize $\sum l_i v_i^2$. The modeling of the constraints and their subsequent linearization are omitted here due to space limitations.

2) The PEDF Heuristic:: The Priority-energy Earliest Deadline First heuristic, or simply PEDF, is an extension of the well-known earliest deadline first (EDF) algorithm. The operation of PEDF is as follows: PEDF maintains a list of all released tasks, called the ready list. When tasks are released, the task with the nearest deadline is chosen to be executed. A check is performed to see if the task deadline can be met by executing it at the lower voltage (speed). If the deadline can be met, PEDF assigns the lower voltage and the execution of the task begins. During the task's execution, other tasks may arrive at any random time. These tasks are assumed to be placed automatically on the ready list. PEDF again selects the task with the nearest deadline to be executed. As long as there are tasks waiting to be executed, PEDF does not keep the processor idle. This process is repeated until all the tasks have been scheduled.

Begin algorithm
Procedure PEDF
Repeat forever
If tasks waiting to be scheduled are in ready list
Sort deadlines in ascending order
Schedule task with earliest deadline
If deadline can be met at lower speed (voltage)
schedule task to execute at lower voltage (speed)
If deadline can not be met at medium speed (voltage)
schedule task to execute at medium voltage (speed)
If deadline can not be met at higher speed (voltage)
schedule task to execute at higher voltage (speed)
Else deadline cannot be met, task cannot be scheduled.
End algorithm

3) Notation:: We present the notation and the underlying assumptions. Let *T* be a set of assumed *n* periodic tasks where *T* = τ_1, τ_2,...,τ_n. Associated with each task $\tau_i \in T$ are:
(i) its arrival time a_i,
(ii) its deadline d_i,
(iii) its length l_i (represented in number of instruction cycles),
(iv) its period p_i.
Each task is released at time t = a_i. Release times are arbitrary where each task may be released at any time before its deadline. We assume that the Central Processor Unit (CPU) can operate at one of the three voltages: V_1, V_2 or V_3.

Depending on the voltage level, the CPU speed may take on three values: S_1, S_2 or S_3. The supply voltage to the CPU is under OS control and the OS may dynamically switch the voltage during run-time with relatively low overhead. CPU speeds are specified in terms of the number of instructions executed per second. Each task τ_i may be executed at a voltage V_i, $Vi \in \{V1, V2, V3\}$.

Task	Arrival Time a_i	Deadline d_i	Length l_i	l_i/v_1	l_i/v_2	l_i/v_3
τ_1	3	7	800	2.66	2	2.28
τ_2	9	21	750	2.5	1.875	2.14
τ_3	0	5	1600	5.33	4	4.57
τ_4	18	25	1000	3.33	2.5	2.85
τ_5	14	16	600	2	1.5	1.71
τ_6	7	10	1200	4	3	3.42
τ_7	20	27	1100	3.66	2.75	3.14
τ_8	14	20	1600	5.33	4	4.57
τ_9	11	14	500	1.66	1.25	1.42

TABLE I
TASK SET COMPOSED OF 9 TASKS.

IV. EXPERIMENTATION RESULTS

We present now our experimental results. First, we show the results of the PEDF for a task set of nine tasks. Our example task set is given in Table I. It consists of tasks τ_1 to τ_9. Each task has a release time a_i, a deadline l_i and a length l_i. We assume that the three processor speeds are 300 million instructions per second (MIPS) at 2.47 V, 350 MIPS at 2.885 V and 400 MIPS at 3.3 V. The energy consumed by the schedule generated through MILP is 75677,4575 units (measured by the sum of the $l_i v_i^2$ values).

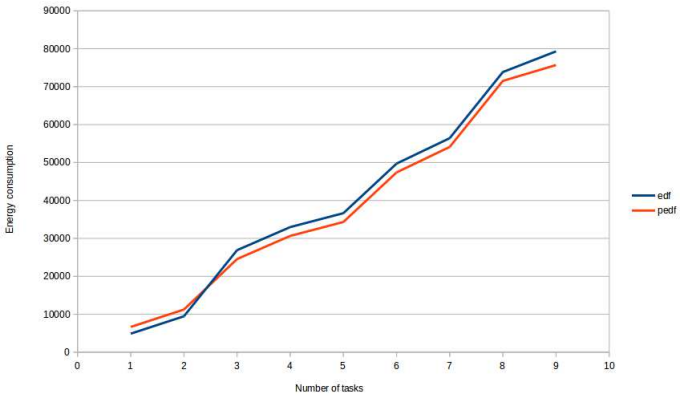

Fig. 4. Comparison of Schedules Generated by EDF and PEDF

In this part, we observe that the increased energy consumption of EDF arises due to the fact that EDF does not possess knowledge of the release times a priori. Our PEDF model, which does not have such a restriction, executes tasks τ_3 and τ_9 at a medium speed (voltage) even though both could have met their respective deadlines by executing at a lower speed (voltage). We can observe that energy consumed by a task is proportional to its length. Since the length of task τ_3 is greater than the lengths of both τ_1 and τ_2. We observe that by executing τ_1 at a medium speed (voltage), and τ_2 at a higher speed (voltage), we can execute τ_3 at a medium speed (voltage) and thus reduces the effect of the length of τ_3 on energy consumption. This, in fact, does result in an optimal schedule. The results, plotted in Figure 4, prove that PEDF produces optimal schedules.

Now we use the first row from the above I, (3, 7, 800) to demonstrate forward propagation: For this example, we take two hidden layers with six neurons. Then, we assign weights to all of the synapses. These weights are selected randomly (based on Gaussian distribution). The initial weights will be between 0 and 1.

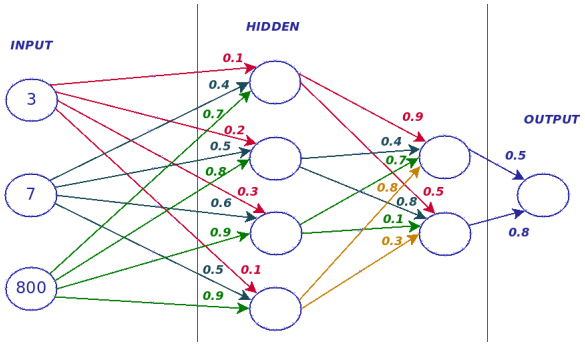

Fig. 5. Determination of Initial Weights

We calculate the sum of the product of the inputs with their corresponding set of weights to arrive at the first values for the hidden layer. Weights may be considered as measures of influence of input nodes on the output.

3 * 0.1 + 7 * 0.4 + 800 * 0.7 = 563.1
3 * 0.2 + 7 * 0.5 + 800 * 0.8 = 644.1
3 * 0.3 + 7 * 0.6 + 800 * 0.9 = 724.2
3 * 0.1 + 7 * 0.5 + 800 * 0.9 = 723.8

We put these sums smaller in the circle, because they are not the final values:

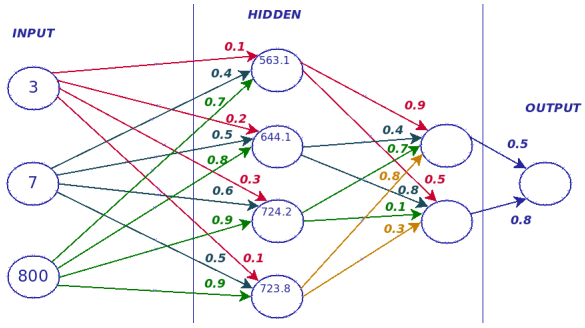

Fig. 6. Calcul of Sums for the First Hidden Layer

We apply the activation function 2 to the hidden layer sums. The purpose of the activation function is to transform the input signal into an output signal and are necessary for neural networks to model complex non-linear patterns. We use the sigmoid function as Transfer Function $f(x) = \frac{1}{1+e^{-x}}$:

f(563.1) = f(644.1) = f(724.2) = f(723.8) = 1
We add the obtained results to our neural network as hidden layer results: Then, we sum the product of the hidden layer

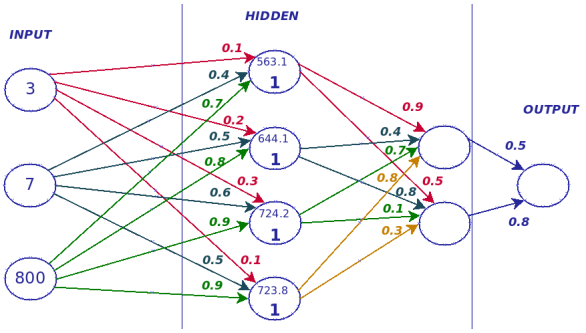

Fig. 7. Application of activation function

results with the second set of weights (also determined randomly the first time around) to determine the output sum.
1 * 0.9 + 1 * 0.4 + 1 * 0.7 + 1 * 0.8 = 2.8
1 * 0.5 + 1 * 0.8 + 1 * 0.1 + 1 * 0.3 = 1.7
We apply now the Transfer Function:
f(2.8) = 0.94
f(1.7) = 0.84

Following the presented method, we had the final output result.

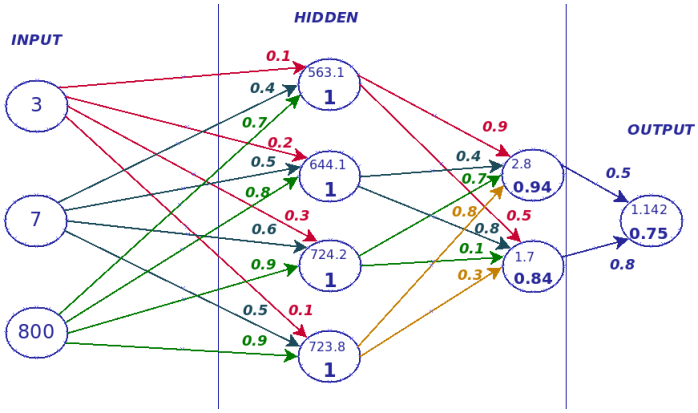

Fig. 8. Full Diagram

V. Conclusion

In this paper we have presented advanced approaches for real-time scheduling of reconfigurable embedded systems to meet real-time constraints in different execution scenarios using neural networks. As a fast and intelligent feedback scheduling scheme, neural feedback scheduling has been proposed in this paper for real-time scheduling of OS tasks. To minimize the total energy consumed, we integrated the PEDF as online scheduling algorithm for real-time systems.
A first perspective for the future works is to extend our original approach based on a neural network approach and especially on the back propagation method to support real-time embedded systems failures.

References

[(Bernard, 2008)] W. Bernard, D.S. Samuel: ”Adaptive Signal Processing”. Machinery Industry Press, Beijing, 2008.

[(Liu, 2011)] D. Li, Y. Liu, and Y. Chen. ”The Application Research of Neural Network in Embedded Intelligent Detection”. (Eds.): CCTA 2010, Part IV, IFIP AICT 347, pp. 376381, 2011.

[(Tian, 2009)] Y. Tian. ”Hybrid Neural Network Technology”. Science Press, Beijing, 2009.

[(Hagan, 1996)] M.T. Hagan, H.B. Demuth and M.H. Beale. ”Neural Network Design”, PWS Publishing, USA, 1996.

[(Xia, 2005)] F. Xia, S.B. Li and Y.X. Sun. ”Neural Network Based Feedback Scheduler for Networked Control System with Flexible Workload”, Int. Conf. on Natural Computation (ICNC), Lecture Notes in Computer Science, vol.3611, pp.237-246, 2005.

[(Weiser, 1994)] M. Weiser, B. Welch, A. Demers and S. Shenker. ”Scheduling for reduced CPU energy”, Proc. Sympo-sium on Operating System Design and Implementation, pp. 1323, 1994.

[(Liu, 1973)] C. L. Liu and J. Layland. ”Scheduling algorithms for multiprogramming in a hard real-time environment”, Journal of the ACM, vol. 20, pp. 4661, 1973.

[(Xia, 2008)] F. Xia, Feng and Tian, Yu-Chu and Sun, Youxian and Dong, Jinxiang, Neural feedback scheduling of real-time control tasks. International Journal of Innovative Computing, Information and Control (IJICIC), 4(11). pp. 2965-2975, 2008.

[(Zhu, 2006)] E. Zhu, Z.B., A simple feasible SQP algorithm for inequality constrained optimization, Applied Mathematics and Computation, vol.182, pp.987-998, 2006.

[(Hopfied, 1985)] J. J. Hopfield and D. W. Tank. Neural computation of decisions in optimization problems. Biological Cybernetics, 52:141-52, 1985.

[(Chillet, 2007)] Daniel Chillet, Sebastien Pillement and Olivier Sentieys. A Neural Network Model for Real-Time Scheduling Heterogeneous SoC Architectures on Heterogeneous SoC Architectures.Proceedings of International Joint Conference on Neural Networks, Orlando, Florida, USA, August 12-17, 2007.