A Design Methodology for Energy Aware Neural Networks

Mehiar Dabbagh, Hazem Hajj, Ali Chehab, Wassim El-Hajj, Ayman Kayssi, Mohammad Mansour

American University of Beirut
Department of Electrical and Computer Engineering
E-mails: {mmd43, hh63, chehab, we07, ayman, mm14}@aub.edu.lb

Abstract*-The increasing demand for mobile devices and high performance computing has made energy consumption a main issue in computer technology. Mobile devices require extended battery life, but the available technology still puts limits on the need for recharging the devices. High performance computing has a high price tag on energy for compute-intensive applications such as data mining. As a result, optimizations at various layers of the computer platform are becoming necessary to minimize energy usage or extend the time before a battery needs to be recharged. This paper focuses on back-propagation neural network algorithm, one of the popular compute-intensive data mining algorithms. The goal is to present a design methodology for developing an energy aware algorithm. The key idea revolves around identifying operations called kernels, which are frequently used in the algorithm, and that can be implemented in hardware. Optimizing these kernels for performance or energy would then lead to a major impact in these areas. These kernels are analyzed for their impact on the overall application energy using energy-based asymptotic analysis. The methodology then considers additional optimizations not related to kernels, but are specific to the back-propagation algorithm. Suggestions are provided to improve the performance and reduce energy consumption. Experiments show that there are significant potentials in energy reduction through the use of alternative lower energy kernels or through custom optimizations with tradeoffs in the accuracy of the results.

***Keywords*- Data Mining, Neural Networks, Back-Propagation Algorithm, Energy Aware.**

I. INTRODUCTION

Energy consumption is a major issue from different aspects. It is an important environmental problem as we are experiencing climatic changes due to the high toxic emissions. In fact, it was noted in [1] that Information and Communication Technology (ICT) is responsible for 2% of the global emissions, equivalent to aviation. Energy consumption is also financially very important as the energy use of ICT is expected to double by 2020 and triple by 2030.

In technology, energy consumption is an important topic and especially after Moore's law started hitting against power density barriers, where the heat density on a device could reach heat levels close to what would be found on the surface of the sun. Other reasons for the importance of energy explorations include the cost of power for high performance computing centers and the widespread use of different mobile devices that are battery dependent.

Reduction of energy can be achieved by optimizing components at the platform level, or examining different computer layers and their interactions, including hardware, architecture, compiler, operating system, and application. This paper explores energy awareness and potential optimizations starting from the application level, but with the goal of integrating with lower layers of the system through low-energy kernels provided from the compiler, operation system, architecture or hardware. We apply our suggested methodology to neural networks back-propagation (BP) [2, 3] as it is a good example of a compute-intensive application and because of its importance in data mining. The first step in the analysis is similar to algorithm asymptotic analysis for the different kernels [4], but with focus on energy instead of performance. The analysis gives an indication for the overall energy cost of the algorithm and the energy contribution for each of the kernels to the overall energy. This energy-based asymptotic analysis helps in determining the priority kernels for energy reduction. Beyond the kernel-based energy optimizations, we propose additional energy improvements specific to BP, which include mathematical approximations and data pre-processing.

The paper is organized as follows: In section II, we review the work done on analyzing data mining (DM) algorithms and energy optimization techniques. In section III, we introduce the different steps of our proposed methodology for analyzing BP. In section IV, we discuss our experimental and simulation results which show that a significant amount of energy reduction can be achieved by applying approximation and preprocessing techniques. In section V, we conclude the paper and present our future work.

II. RELATED WORK

Energy reduction can be achieved by performing optimizations at various abstract layers including hardware, architecture, compiler, operating system, and application. In our work, we consider the application level and suggest a methodology to customize data mining algorithms and in particular BP for the purpose of reducing energy consumption.

Energy reduction at the application level has been addressed in mainly two ways: (1) using energy optimization techniques on various applications to reduce energy consumption [5-12] and (2) optimizing the runtime complexity of some DM algorithms and consequently reducing energy consumption [4, 13]. The first approach is applied to general applications and not to DM algorithms. The second approach is applied to DM algorithms, but it reduces energy indirectly by optimizing runtime. To the best of our knowledge, reducing energy consumption of DM algorithms has not been addressed directly before.

Network applications, multimedia, bioinformatics, and file access are some of the applications that underwent energy optimization attempts. As suggested in [5], three main approaches are generally used to reduce application energy: computational efficiency techniques (e.g. unrolling loops, multithreading), data efficiency techniques (e.g. data buffering, putting data as close as possible to CPU) and context/power-aware behavior (e.g. applications should react to power transitions). The computational efficiency technique (first approach) was used in [8-9] to optimize the energy consumption of general applications. In particular, loop unrolling, function inlining, and loop alignment were used on the application source code. Experimental and statistical analyses were conducted to check the real impact of the suggested optimizations. In [6], all the three approaches presented above were applied to specific applications such as transferring files over a wireless network and reading files from the hard disk. The corresponding energy reductions were calculated and assessed. In [7], the energy consumption of a multimedia application is analyzed. The application consists of video playback for files with different encoding formats. In [10], a wireless sensor network application was considered, where new software techniques were proposed to reduce energy consumption. The optimizations were done at the sensing, communication and computation levels. In [12], the authors proposed a scheduling model for a bioinformatics application that aims to reduce the consumed energy while maintaining an acceptable high performance.

All these mechanisms target general applications and do not address specific applications such as data mining. In fact, very little work has been done to optimize the energy consumption of data mining algorithms. The work in [4] and [13] reduce energy consumption indirectly via reducing runtime. In [4], the authors proposed parallelizing different machine learning algorithms by distributing calculations on different cores in order to improve runtime performance. In [13], the authors also used parallelization to improve performance but their work was focused on Linear time Closed Itemset Miner (LCM), a specific data mining algorithm for mining frequent itemsets.

Our work is different from previous work since it focuses on the energy consumption of DM algorithms, an application that was not addressed directly in previous work and is widely used in many domains (bioinformatics, business, social networks…etc). DM algorithms usually have a segment of code that is repeatedly executed for different training tuples. Therefore optimizing the energy of these segments will be highly reflected on the overall energy consumption. We introduce in this paper a methodology for analyzing any DM algorithm from the energy consumption point of view rather than performance point of view. Then we apply our proposed methodology to BP algorithm. Unlike all the optimization techniques used to reduce energy in general applications (multithreading and parallelism [4-6, 13] computational efficiency [8-9], data efficiency, and power-aware behavior), our approach uses approximation and preprocessing techniques that use lookup tables leading to both performance improvement and energy savings.

III. Proposed Methodology

In this section, we describe a design methodology to achieve energy-aware back-propagation algorithm. Six steps are proposed below, with particular emphasis on kernel-based energy analysis and optimization. The purpose of these steps is to analyze the program details and identify opportunities for reducing energy consumption.

Step 1: High-level program analysis: The program is examined for the major computational structures, including loops, sequential parts, and parallel parts of the algorithm. The objective of this step is to get a preliminary understanding of the execution elements of the code so that future steps can enhance the code details and make it more energy efficiency. Figure 1 is a pseudo code for the general flow of the BP algorithm. The algorithm consists of an initialization step followed by a "for" loop. The body of the loop contains 2 sequential stages: A forward stage and a backward stage.

```
Algorithm:
Backpropagation for classification & prediction.
Input:
■ D, a data set consisting of the training tuples
  and their associated target value.
■ N, number of iterations.
■ Network, a multilayer feed-forward network.
Output: A trained neural network.
Method:
(1) Initialize all weights and biases in Network;
(2) For i=1 to N {
(3)    Pick a training tuple from D;
(4)    // Forward phase;
(5)    Calculate the output of each Neuron
       From first to last layer in Network;
(6)    // Backward phase;
(7)    Calculate the error from last to first layer
       in Network;
(8)    Update the weights and biases of Network
       so that the error between the desired
       output and Network output is minimized;
(9)    }
```

Fig. 1 Back-propagation algorithm.

Step 2: Identification of high frequency kernels: Kernels are operations that can be implemented in hardware, and are executed repeatedly in each iteration. A typical data mining algorithm has some mathematical operations, and as a result may have its own set of kernels. Some basic kernels include:

addition, multiplication, division, subtraction, and count. These operations could be executed on vectors, arrays or single elements depending on the algorithm.

From Figure 1, we can observe that reducing the cost of any step in the body of the loop will be highly reflected on reducing the overall energy since executing the body of the loop is repeated N times (usually $N \geq 5000$). For BP, Table 1 shows the equations that are executed in the loops of the forward and backward phases and the corresponding kernels, where S_k^p. and , y_{jk}^{p-1} are the input and output to the unit (neuron) k in layer p and p-1 respectively, w_{jk} is the weight of the connection from unit j to unit k, b_k is the bias of unit k, $e_j(n)$ is the error in unit j at layer n, $d_j(n)$ is the desired output of unit j at layer n, $y_j(n)$ is the the j-th unit output at layer n, Δw_{jk} is the change in weight of the connection from unit j to unit k, γ is the learning rate and δ_k^p is the error in unit k at layer p and w'_{jk} is the new weight of the connection from unit j to unit k. From Table 1 we can determine the kernels of the BP algorithm which are: multiplication, division, addition, subtraction and exponential operations.

TABLE 1
BP equations and the corresponding kernels.

Forward phase:	
Equation	**Kernels**
$S_k^p = \sum_j w_{jk} y_{jk}^{p-1} + b_k$	Multiplication Addition
$y^p = \frac{1}{1+e^{-S^p}}$	Division Addition Exponential function
Backward Phase:	
Equation	**Kernels**
$e_j(n) = d_j(n) - y_j(n)$	Subtraction.
$\Delta w_{jk} = \gamma\, \delta_k^p y_j^p$	Multiplication.
$w'_{jk} = w_{jk} + \Delta w_{jk}$	Addition

Step 3: Algorithm asymptotic analysis for kernel energy and computation growth rates: The purpose of this step is to examine the effect of the different kernels and computations on the energy and performance of the algorithm as the data size grows. This step is similar to the standard algorithm asymptotic analysis conducted for performance, but applied here for energy. The analysis consists of:

a) Counting the number of times each kernel is executed while accounting for size of arrays or vectors in the operations.
b) Calculating the energy cost of each kernel or choosing a set of base kernels for which energy is computed and giving relative cost of other kernels relative to the base set.
c) Calculating the overall energy of the algorithm.
d) Determining the energy contribution of each kernel to the overall energy.

For the NN algorithm, the following assumptions and notations are made:

- IN represents the number of input neurons in the input layer (the number of input neurons is equal to the number of attributes in each input tuple).
- HN represents the number of hidden neurons in the hidden layer.
- ON represents the number of output neurons in the output layer.
- The non-linear activation function for the hidden layer, the sigmoid function, is given by equation (1), where S^p and y^p are the input and output of the function.

$$y^p = \frac{1}{1+e^{-S^p}} \quad (1)$$

- The linear activation function is used for the output layer. It is given by:

$$y^p = S^p \quad (2)$$

- The training phase is repeated for N iterations. The number of iterations in the main loop is usually proportional to the data size. So the number N is also representative of data size.
- The variables that are used in the high-level language code of the algorithm are defined in Table 2. Columns two and three in Table 2 represent the number of rows and the number of columns respectively for array variables.

To conduct the asymptotic analysis, we count the number of times each kernel is executed in every line of the high-level language code for one iteration of the algorithm (Table 3). For example, the first row in Table 3 breaks the first line of

TABLE 2
Name and definition of BP code variables.

Name	Num. of rows	Num. of col.	Description
X	1	*IN*	Input neurons
w1	*IN*	*HN*	Weights from the input to the three hidden neurons.
b1	1	*HN*	Bias for the hidden neurons.
w2	*HN*	*ON*	Weights from the hidden neurons to the output neurons.
b2	1	*ON*	Bias for the output neurons.
s1	1	*HN*	Input to the hidden neurons.
y1	1	*HN*	Output of the hidden neurons.
So	1	*ON*	Input of the output neurons.
Yo	1	*ON*	Final output of the NN.
D	1	*ON*	Desired output.
E	1	*ON*	Error used to change the weights.
delta_o	1	*ON*	Delta for output layer.
delta_h	*HN*	1	Delta for hidden layer.
gama	*1*	1	Learning rate.
d_w1	*IN*	*HN*	Delta for the weights of the hidden neurons.
d_b1	1	*HN*	Delta for the bias of the hidden neurons.
d_w2	*HN*	*ON*	Delta for the weights of the output neurons.
d_b2	1	*ON*	Delta for the bias of the output neurons.

TABLE 3
Number of executed kernels for each instruction in one iteration.

Line of high-level code	Matrix Multiplication		×	÷	+	-	exp.
	×	+					
s1=x*w1+b1	I N× HN	(IN-1)×HN	0	0	HN	0	0
y1=1./(1+exp(-s1))	0	0	HN	HN	HN	0	HN
so=y1*w2+b2;	HN×ON	(HN-1) × ON	0	0	ON	0	0
e=d-yo	0	0	0	0	0	ON	0
delta_h=w2*delta_o'.*(y1'.*(1-y1'))	ON×HN	(ON-1) ×HN	2×HN	0	0	HN	0
d_w1=gama*x'*delta_h'	IN×HN	0	IN	0	0	0	0
d_b1=gama*delta_h	0	0	HN	0	0	0	0
d_w2=gama*y1'*delta_o	ON×HN	0	HN	0	0	0	0
d_b2=gama*delta_o	0	0	ON	0	0	0	0
w1=w1+d_w1	0	0	0	0	IN×HN	0	0
w2=w2+d_w2	0	0	0	0	HN×ON	0	0
b1=b1+d_b1'	0	0	0	0	HN	0	0
b2=b2+d_b2	0	0	0	0	ON	0	0

the code ***s1=x*w1+b1*** into its basic kernels. The first part ***x*w1*** is a matrix multiplication in which we multiply x whose size according to Table 2 is (1×IN) by w1 which is a matrix of size (IN×HN). Multiplying x by w1 consists of IN×HN multiplication operation and (IN-1)×HN addition operation and the resulting matrix has a size of (1×HN). The resulting matrix is then added to b1 whose size is also (1×HN) and this step involves HN addition operation. As a result s1=x*w1+b1 consists of IN×HN multiplication operation and (IN-1)×HN addition operation for matrix multiplication and HN addition operation for summing the resulting matrix to b1 as shown in Table 3. The same procedure is applied to the rest of the code.

From Table 3, it is possible to calculate energy consumption for NN with IN input neurons, HN hidden neurons, ON output neurons and N iterations. The number of times each kernel is executed per one iteration of the main loop can be determined by summing the number of times each kernel in Table 3 (multiplication, addition, division, subtraction and exponential) is executed for all the lines of the code for one iteration. The results are shown in the following equations:

$$N_{\times} = 2 \times IN \times HN + 3 \times ON \times HN + 5 \times HN \times ON \quad (3)$$
$$N_{+} = 2 \times HN \times IN + HN + 3 \times HN \times ON + ON \quad (4)$$
$$N_{\div} = HN \quad (5)$$
$$N_{exp} = HN \quad (6)$$
$$N_{-} = ON + HN \quad (7)$$

where: $N_{\times}$, N_{+}, $N_{\div}$, N_{exp} and N_{-} are the total numbers of multiplications, additions, divisions, exponentials and subtractions per one iteration respectively.

For N iterations, the total numbers in equations (3)-(7) are magnified N times, and therefore we can determine the total order of computation for each kernel as shown in Table 4. Once the asymptotic analysis is complete, we can assess the relative energy consumption for the different kernels, and determine which kernels have the highest impact on energy. In section IV, we show simulation experiments indicating the relative energy impact of using alternative lower-energy kernels.

TABLE 4
Order of BP kernels

Kernel	Order
×	$\theta(N \times [2 \times \text{IN} \times \text{HN} + 3 \times \text{ON} \times \text{HN} + 5 \times \text{HN} + \text{ON}])$
+	$\theta(N \times [2 \times \text{HN} \times \text{IN} + \text{HN} + 3 \times \text{HN} \times \text{ON} + \text{ON}])$
÷	$\theta(N \times \text{HN})$
-	$\theta(N \times [\text{ON} + \text{HN}])$
Exp	$\theta(N \times \text{HN})$

Step 4: Prioritize kernels in terms of energy impact: This step helps in knowing which kernels should be targeted for maximizing the reduction of energy consumption of the overall algorithm.

In order to prioritize kernels, we study how much saving in the overall cost we can achieve if we reduce each kernel's cost by the same amount. The one that gives the highest overall energy reduction is the best target for energy reduction. In section IV, we use simulation to get an insight into the priority energy kernels.

Step 5: Use alternative lower energy kernels: At this stage, we use the results of step 4 to determine various ways to reduce energy cost for priority kernels. Potential approaches include:

a) Using approximations to the kernels with less computations and lower energy with a tradeoff of lower accuracy for better performance and energy. In this scenario, it is important to assess the error resulting from the approximation to make sure that the algorithm still yields acceptable results.
b) Using alternative hardware implementation of instruction set with lower energy and without

compromising performance. For example a totally new optimized instruction set such as those used in DSP processors can be utilized.

c) Parallelizing the code to improve the performance of the algorithm. In this scenario, it is important to ensure that the performance improvements lead to energy improvements.

In section IV we propose an approximation technique for exponential kernel using lookup tables.

Step 6: Examine other opportunities of energy reduction specific to the algorithm under consideration: This could be preprocessing the data to simplify algorithm computations. In section IV we show that normalizing the training set results in notable energy savings.

IV. Experiments and Results

In this section, we conduct three experiments to evaluate the proposed methodology for energy analysis. In the first experiment, we perform simulation analysis to assess the energy impacts of the kernels derived in step 3 of section III. This experiment also provides an indication of the kernels with highest energy impacts. In the second experiment, we consider alternatives to the kernels by way of approximation, and study the impact of the approximation on the accuracy of BP. In the third experiment, we consider data manipulation as another possibility for energy reduction with BP neural network.

For the experiments below, we assume the NN has two input neurons, three hidden neurons, two output neurons and 10,000 iterations. We can calculate the number of times each kernel is executed for the NN under consideration by simply replacing IN, HN, ON, N in Table 4 by 2, 3, 2, and 10000 respectively.

1. *Energy Impacts of individual kernels on overall algorithm energy*

To simulate the execution frequency of each kernel, a separate counter is incremented in the code of Figure 1, each time a kernel is executed. Based on the 10000 iterations, the number of times each kernel is executed for the NN under consideration is shown in the Figure 2.

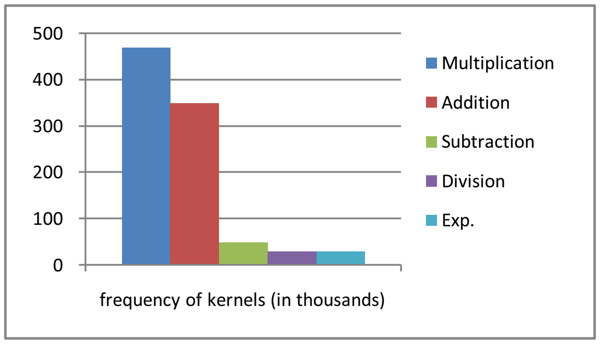

Fig. 2 Number of times each kernel is executed for 10 000 iterations

The simulation result of Figure 2 is consistent with the asymptotic analysis conducted in step 3 of section III, which showed that the highest numbers of kernels executed are multiplications followed by additions. To simulate the energy cost of each kernel, we assign an energy weight for each kernel. To normalize the computation, we use the addition kernel as the base, and give all other kernels relative energy weights. Table 5 shows the relative energy cost of each of the BP kernels.

TABLE 5
Relative energy cost for BP kernels

	×	+	-	÷	exp
Relative Energy cost	10	1	1	10	40

These relative costs can be replaced by actual energy costs that would need to be physically measured, which is part of future work. One possibility to obtain actual energy cost is to use the assessment in [14] that suggests writing the same instruction more than once in a loop and measuring the average current. The overall energy was calculated by multiplying the number of times each kernel is executed by the corresponding cost of each kernel. The overall energy of the algorithm based on Figure 2 and Table 4 was calculated as follows:

$$overall\ energy = 10\times 470 + 1\times 350 + 1\times 50 + 10\times 30 + 40\times 30 = 6600 \text{ (in thousands)} \quad (8)$$

Based on this calculation, we can determine the contribution of each kernel to the overall energy as shown in Figure 3. We can see from Figure 3 that although the exponential function is executed few times (as shown in Figure 2), it has a high contribution to the overall energy due to its high relative energy cost. It is also clear from Figure 3, that the multiplication kernel has the highest impact followed by exponential despite the fact that there are more addition computations than there are exponentials.

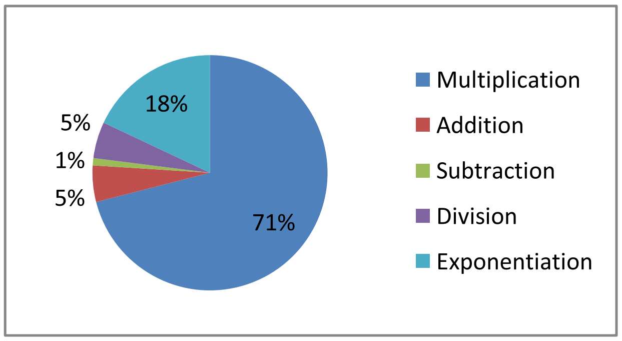

Fig. 3 A pie chart showing the contribution of each kernel to the overall cost

This simulation supports the validity of the proposed methodology to determine kernels with higher energy impact, and which would need to be further considered for energy reduction. To further confirm these conclusions, we study the saving of overall energy that is achieved by reducing each kernel's cost by 25%. This helps in reflecting which kernels should be targeted for maximizing the reduction of energy. Figure 4 shows the overall energy cost of the algorithm as the number of iterations increases after reducing each kernel's cost by 25%. Results indicate that the largest overall energy reduction is obtained by reducing the cost of multiplication followed by reducing the cost of the exponential function. We can also see that reducing the cost of the other kernels (subtraction, division and addition) has an impact on energy reduction, though not as large as multiplication or exponential.

In addition to the high cost of the exponential function, it is also one of the most time-consuming parts of the algorithm. We show next how to improve the performance of the algorithm by using lookup tables instead of calculating the value of the exponential function and show how this improvement also leads to energy saving.

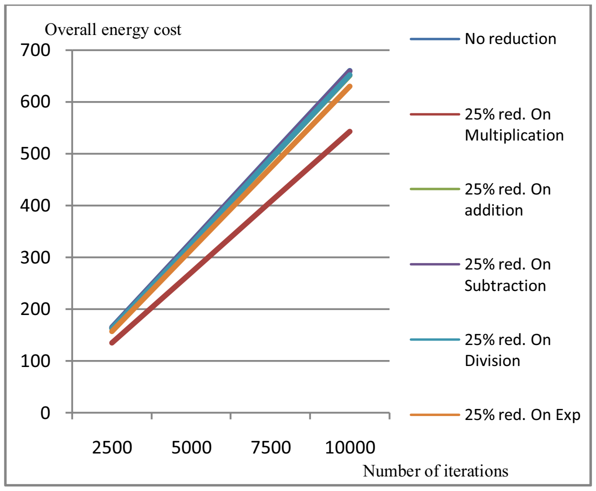

Fig. 4 The impact of reducing different kernels by 25% on the overall algorithm for different iterations for a NN with 2 input points, 3 hidden neurons and 2 output neurons.

2. *Using alternative lower energy kernels*

In this experiment, we examine alternative options to the exponential operation which clearly consumes a lot of energy. We examine the use of approximation techniques [15] instead of exact calculation of the exponential function.

In the BP algorithm we need to calculate the output of the hidden neurons as shown in equation (1). The value of S is usually between -10 and 10. We therefore choose a main lookup table (LUT) of size 20 to store the exponential values of numbers between -10 and 10 with a step of 1.The secondary table saves the exponential values for: -0.25, -0.5, -0.75, 0, 0.25, 0.5, 0.75 and has a size of 7. These two LUTs can help us approximate a real number in the range of -10 to 10. For example, to calculate the value of exp(3.5), 3 is used as an index to the main table for the approximated value of exp(3). Similarly, (0.5) is used as the index to the secondary table and it will return the value of exp(0.5). Since exp(x+y) = exp(x).exp(y), exp(3.5) can be derived from the product of the two values that are fetched from the two lookup tables.

Another way to achieve the same accuracy with only one lookup table, is to use a table of size 80 with a step of 0.25. The two lookup tables achieve high accuracy with smaller size of LUT. The rest of the analysis assumes two LUTs.

Getting a value from LUT requires an approximating operation before getting the value. If we want to calculate the exponential function of 3.6 for example, the index of the main table will bring the value of exp(3), the index of the secondary table will get the value of 0.5 instead of 0.6.

To measure the accuracy of our approximation technique, we calculate the value of exp(s), *s* being the input of the hidden neuron in equation (1), in two different ways: using the normal exponential function (referred to as *expNormal(s)*) and using the suggested approximation technique that utilizes lookup tables (referred to as *expApp(s)*). We then calculate the error which is the difference between these two values i.e. $error(s) = |expNormal(s) - expApp(s)|$ and then divide it by *expNormal(s)* to get a measure of error that is proportional to the normal exponential function i.e.

$$relative\_error(s) = \frac{|expNormal\ (s) - expApp\ (s)|}{|expNormal\ (s)|}$$

Since there exist three inputs (s_1, s_2, and s_3) for the three hidden neurons, we calculate the three corresponding relative errors: *relative_error(s1)*, *relative_error(s2)*, and *relative_error*(s_3) and repeat the calculations for 10000 iteration. The average of all relative errors was calculated to be 0.0627, which shows that $error(s)$ is very small compared to *expNormal(s)* .

Figures 5 and 6 show a comparison between NN output using two lookup tables and using the normal exponential function. In Figure 5, (a) represents the first desired output

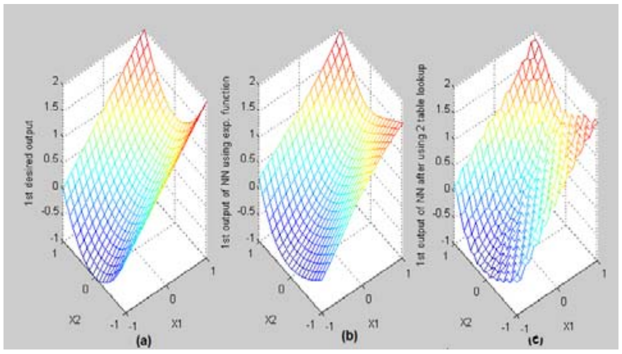

Fig. 5 A comparison (a) 1st desired output (b) 1st NN output using exp. (c) 1st NN output using two lookup tables approximation

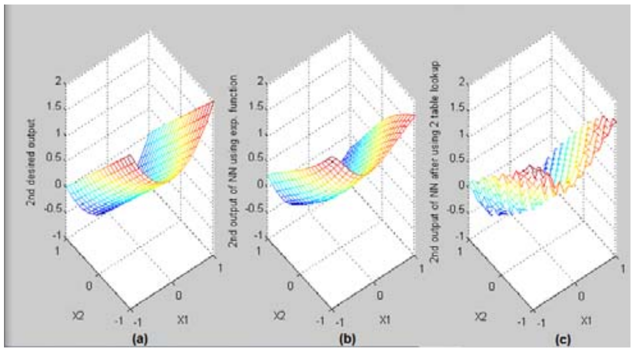

Fig. 6 A comparison: (a) 2nd desired output (b) 2nd NN output using exp. (c) 2nd NN output using two lookup tables approximation

of NN, (b) represents the first output of NN using normal exponentiation function and (c) represents the first output of NN using two lookup tables. The same comparison is shown in Figure 6 for the second output of NN. Visually, we can see that the output of NN with the LUT approximation technique is very close to the output of NN with no approximation.

As stated in [15], using LUT approximation technique improves runtime performance with an acceptable accuracy for NN as shown in Figures 5 and 6. In general, this improvement in runtime leads to more energy efficiency (calculated using equation (9)).

$$energy\ efficiency = \frac{Energy\ cost\ for\ LUTs}{energy\ cost\ for\ exponential\ function} \quad (9)$$

3. Examining other opportunities of energy reduction specific to the algorithm under consideration

In this experiment, we examine ways that are specific to the algorithm beyond the general examination of kernels. In particular, we look at preprocessing the data to simplify algorithm computations, and examine the energy impact.

As stated in [2], the input values of the NN training phase can be normalized to the range of [0,1] for faster convergence of the NN algorithm. To examine the impact of the normalization on energy efficiency, we examine two scenarios. In the first scenario, the training set values are not normalized, while in the second scenario, the training set is normalized to the range [0,1]. In both cases, the learning phase is continued until the mean squared error (MSE) is less than a predefined threshold, usually between 0.1 and 1. Since the normalized training set requires fewer iterations, it consumes less computation energy. But we should take into account the energy cost of the normalization process when we compare the energy saving between the two scenarios.

Min-max Normalization was used to normalize the training set into [0,1] which is shown below:

$$v' = \frac{v - min_a}{max_a - min_a}(new\_max_a - new\_min_a) + new\_min_a \quad (10)$$

where new_max = 1 and new_min = 0. So the normalization process requires two operations: subtraction (whose relative cost is 1) and division (whose relative cost is 10) for each attribute value in each training tuple, and the energy cost for normalization can be determined as shown below:

$$C_{nor} = 11 \times N_{tuples} \times N_{att} \quad (11)$$

Where C_{nor} is the cost of normalization, N_{tuples} is the number of tuples in the training set, and N_{att} is the number of attributes in each tuple.

Two groups of training sets were tested to demonstrate the savings in energy that are obtained by normalization. Group 1 contained 400 training tuples, each tuple consisted of two attributes with values between 0 and 2.8. Group 2 contained 400 training tuples, each tuple consisted of two attributes but with values between 0 and 3. Our stopping criterion was when MSE reached a value less than 0.8 for both groups.

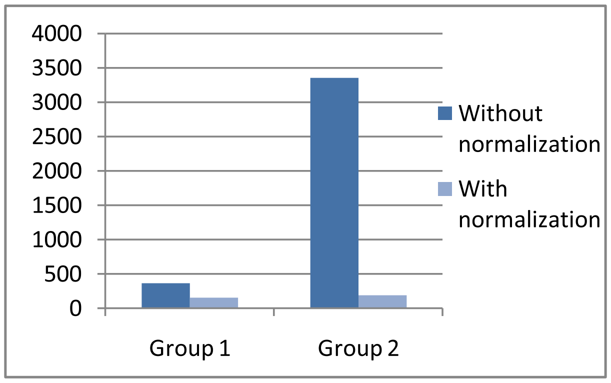

Fig. 7 A comparison in the required of required iterations to reach MSE<0.8

The major difference between the two groups is that the input values of Group 2 are much larger than the input values of Group 1.

Figure 7 shows the number of iterations that was required to reach the stopping criterion with and without normalization for Group 1 and Group 2. It is clear from Figure 7 that the larger the values of the input are, the more iterations were required to reach the MSE threshold. The reason why more iterations were required for Group 2 is because there were larger values in the training set. Hence, the algorithm converged slower and reaching a small value of MSE that is less than 0.8 required too many iterations.

To determine the execution frequency of each kernel, a separate counter is incremented in the code each time a kernel is executed until we reach the MSE threshold. Then we multiply the execution frequency of each kernel by the relative energy cost (from Table 5) and we obtain the training cost of the algorithm by adding the costs of all kernels. The cost of normalizing Group 1 and Group 2 can be calculated from (11) by replacing N_{tuples}, $N_{att.}$ by 400 and 2 respectively. The overall energy for running the algorithm after normalization equals the sum of the normalization cost and the training cost.

Table 6 shows a cost comparison for running the algorithm with and without normalization. It can be clearly seen that normalizing the training set has lead to reducing the energy cost for both groups. We conclude that normalizing the training set before the learning phase makes the algorithm converge faster with fewer iterations, and as a result produces savings in energy.

TABLE 6
A cost comparison between running the algorithm with and without normalizing the training set.

	Without normalization	**With normalization**		
	Overall cost	**Cost of normalization**	**Training cost**	**Overall cost**
Group 1	240 240	8800	98 340	107 140
Group 2	2 210 340	8800	124 080	132 880

I. CONCLUSIONS AND FUTURE WORK

This paper has introduced a design methodology for energy analysis of back-propagation algorithm, with particular emphasis on new kernel-based evaluation for energy optimization. Additional innovations include the energy-based asymptotic analysis and new proposals for neural network custom optimizations to achieve further reduction in energy. The methodology can be extended to any data mining algorithm, and can extend kernel optimizations to parallel implementation. The experimental studies demonstrated the effectiveness of the methods and helped determine which kernels contributed most to the overall energy. For the case of back-propagation algorithm, multiplication was a top contender followed by exponential computations. We showed that exponential approximation and data preprocessing can also have significant impacts. For future work, we plan is to pursue further validation by conducting real power measurements with specially instrumented boards for collection of current and voltage measurements.

ACKNOLEDGMENT

This work was funded by Intel's Middle East Energy Efficiency Research (MER) program and the American University of Beirut (AUB) University Research Board (URB).

REFERENCES

[1] C. Pettey, "Gartner Estimates ICT Industry Accounts for 2 Percent of Global CO2 Emission," *Gartner Press Release*, 26 April 2007. URL: http://www.gartner.com/it/page.jsp?id=503867

[2] S. Haykin, "Neural Networks: A Comprehensive Foundation (2nd Edition)," *Prentice Hall*, July 1998.

[3] J. Han, M. Kamber, "Data Mining: Concepts and Techniques," 2nd ed. *Morgan Kaufmann*, January 2006

[4] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, K. Olukotun, "Map-Reduce for Machine Learning on Multicore," *Neural Information Processing Systems*, pp. 281—288, 2006.

[5] P. Larson, "Energy-Efficient Software Guidelines," *Intel Software Solution Group*, 2008.

[6] B. Steigerwald, R. Chabukswar, K. Krishnan, J. D. Vega, "Creating Energy – Efficient Software," *Intel white paper*, 2008.

[7] T. H. Darwish, R. Chabukswar, "Intel Hardware Accelerated High Definition Video Playback Power Analysis," *Intel Software & Service Group*, 2009.

[8] D.A. Ortiz, N.G. Santiago, "Impact of Source Code Optimizations on Power Consumption of Embedded Systems," *2008 Joint 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference*, 2008. *NEWCAS-TAISA 2008*, pp.133-136, 22-25 June 2008.

[9] D.A. Ortiz, N.G. Santiago, "High-level Optimization for Low Power Consumption on Microprocessor-based Systems," *50th Midwest Symposium on Circuits and Systems, 2007. MWSCAS 2007,* pp.1265-1268, 5-8 Aug. 2007.

[10] P. K. Dutta, D. E. Culler, "System Software Techniques for Low-power Operation in Wireless Sensor Networks," *ICCAD '05 Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pp. 925- 932, 6-10 Nov. 2005 doi: 10.1109/ICCAD.2005.1560194

[11] V. Dalal, C.P. Ravikumar, "Software Power Optimizations in an Embedded System," Fourteenth International Conference on VLSI Design, 2001, pp.254-259, 2001

[12] S. Pawaskar, H. H. Ali, "A Dynamic Energy-aware Model for Scheduling Computationally Intensive Bioinformatics Applications," *High Performance Computing and Simulation (HPCS)*, 2010 pp.216-223, June 28 2010-July 2 2010

[13] B. Negrevergne, A. Termier, J. Méhaut, T. Uno, "Discovering Closed Frequent Itemsets on Multicore: Parallelizing Computations and Optimizing Memory Accesses," *High Performance Computing and Simulation (HPCS)*, 2010 pp.521-528, June 28 2010-July 2 2010 doi: 10.1109/HPCS.2010.5547082

[14] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *Very Large Scale Integration (VLSI) Systems IEEE*, vol.2, no.4, pp.437-445, Dec 1994. doi: 10.1109/92.335012

[15] A. Yamamoto, "Computational Efficiencies of Approximated Exponential Functions for Transport Calculations of the Characteristics Method," *Annals of Nuclear Energy*, 31(9):1027- 1037, June 2004.