A Hybrid of Artificial Bee Colony, Genetic Algorithm, and Neural Network for Diabetic Mellitus Diagnosing

Tarik A. Rashid [1,3], Saman M. Abdullah [2,4]

[1]Department of Science and Engineering, University of Kurdistan Hewler, Erbil, Kurdistan Region – F.R. Iraq

[2]Department of Software Engineering, Faculty of Engineering, Koya University, Kurdistan Region – F.R. Iraq

[3]Department of Software and Informatics Engineering, Salahaddin University, Erbil, Kurdistan Region – F.R. Iraq

[4]Department of Computer Engineering, Ishik University, Erbil, Kurdistan Region – F.R. Iraq

***Abstract*–Researchers, widely have introduced the artificial bee colony (ABC) as an optimization algorithm to deal with classification and prediction problems. ABC has been combined with different artificial intelligent techniques to obtain optimum performance indicators. This work introduces a hybrid of ABC, genetic algorithm (GA), and back propagation neural network (BPNN) in the application of classifying and diagnosing diabetes mellitus (DM). The optimized algorithm is combined with a mutation technique of GA to obtain the optimum set of training weights for a BPNN. The idea is to prove that weights' initial index in their initialized set has an impact on the performance rate. Experiments are conducted in three different cases; standard BPNN alone, BPNN trained with ABC, and BPNN trained with the mutation based ABC. The work tests all three cases of optimization on two different datasets (primary dataset and secondary dataset) of DM. The primary dataset is built by this work through collecting 31 features of 501 DM patients in local hospitals. The secondary dataset is the Pima dataset. Results show that the BPNN trained with the mutation based ABC can produce better local solutions than the standard BPNN and BPNN trained in combination with ABC.**

***Index Terms*—Artificial Bee Colony, Artificial Neural Networks, Diabetic Mellitus, Evolutionary Algorithms.**

I. Introduction

Artificial neural network (ANN) is an information processing paradigm that simulates the nervous system of the human brain and its cognitive processes. The key features of an ANN (Priddy and Keller, 2005) are pliability, competence, capacity to simplify, and resolve categorization difficulties, and determining similarity in patterns. ANNs, and their training algorithms have become increasingly important for modeling and optimization in many fields of science and engineering. Among many different ANN models, BP-based trained ANN as a multi-layer structure has been widely utilized rather than other training types due to its great capability in-universe approximation and optimization (Nawi, et al., 2010). Nevertheless, BP-based ANN suffers from a low convergence rate and instability; this would be triggered through falling in local optimum solutions (Karaboga, et al., 2014). For that reasons, numerous procedures for optimizing and enhancing the learning method of the BP-based ANN (Nawi, et al., 2011). In the direction of that end, different global search algorithms have been involved to optimize the weights that initialize a BP trained with Artificial bee colony (ABC) and genetic algorithm (GA) (Nawi, et al., 2014). The idea of searching in ABC, and GA is different. ABC is searching for the best solution among a set of populations through an updating process. The important activities of ABC are to share the indexes of the provided solutions, which are known as food sources. This process in the ABC is called information sharing (Karaboga, et al., 2014; Akay and Karaboga, 2015). However, a typical ABC cannot do this sharing efficiently. Although GA, which also has some populations, can provide efficient index sharing, the process of updating sources has no relation with the learning rate as found in ANN and ABC. GA is just using crossover among the population to get new offspring (Kumar and Verma, 2012). A recent article (Nawi, et al., 2010) surveyed two decades of research and showed that there are numerous ways to search for an optimization of an ANN. It shows that the simplest and best way are to optimize through weight updating. The article also presents different methods for weight updating, because we have found no previous published research on this topic. There are many works utilized different optimization methods in different fields. The most popular methods are practical swarm optimization, ant colony optimization, bacterial foraging optimization, evolutionary algorithm, and GA. In some studies, ABC

ARO-The Scientific Journal of Koya University
Volume VI, No.1(2018), Article ID: ARO.10368, 10 pages
DOI: 10.14500/aro.10368
Received 02 January 2018; Accepted 20 May 2018
Regular research paper: Published 13 June 2018
Corresponding author's, e-mail: tarik.ahmed@ukh.edu.krd

http://dx.doi.org/10.14500/aro.10368

outperforms other methods (Karaboga, et al., 2014; Nawi, et al., 2011).

The objective of this article is to combine the mathematical calculation of ABC and the crossover technique of GA to generate a new initialized weights' set that can increase the efficiency of back propagation neural network (BPNN) training and learning. We then apply the proposed approach to the process of diagnosing diabetes mellitus (DM) cases.

The rest of paper is organized as follows: Section 2, describes clinical and soft classification of DM, next, BPNN is presented, in Section 4, nature-inspired algorithm is introduced, then ABC is defined in Section 6, then, GA is explained, followed by an explanation of ANN training with Mutated ABC, in Section 8, details of experimental results are presented, and finally, the key points are concluded.

II. Clinical and Soft Classification of DM

DM is a chronic disease in which a patient's body is unable to produce or unable to properly use and store glucose. It is a lifelong condition that affects the body's ability to efficiently use the energy found in food. As of 2014, an estimated 387 million people have diabetes worldwide. From 2012 to 2014, diabetes is estimated to have caused 1.5–4.9 million deaths each year. The number of people with diabetes is expected to rise to 592 million by 2035. The global economic cost of diabetes in 2014 was estimated to be $612 billion USD (Federation, 2014).

This disease has many types and forms; however, the most popular types are Type 1 (known as insulin-based DM) and Type 2 (known as noninsulin-based DM) (Wild, et al., 2004). Physician must define the type of DM a patient has so that proper medications can be given and so that patients can be instructed on how to minimize side effects of the disease. Recently, many soft computing models have been built to diagnose and classifying DM cases into either Type 1 or Type 2. Some soft models can predict the rate of glucose in the blood for DM patients based on various predictors. One of the most popular soft computation tools that utilized for DM distinguisher is ANN (Association, 2014).

III. BPNN

BPNN is one of the most effective ANN supervised learning algorithms. It causes an ANN to learn through the process of minimizing errors at the output layer's neurons. Errors in the hidden layer of any BPNN can also be minimized based on the rate of errors in the output layer. This computation is the core fundamental of the learning process in any BP-based ANN structure. Through this process, a BPNN calculates and adjusts the weights utilizing gradient descent method (Atakulreka and Sutivong, 2007; Dai and Liu, 2012; Ojha, et al., 2016; Ojha, et al., 2017). Through this weights adjustment, BPNN can minimize the error rate at the output layer. Errors at the output layer correspond to the sum of the squares of the errors recorded between the actual and desired outputs, as indicated in Eq. (1).

$$E_P = \sum_{i=1}^{j} (d_i - y_i)^2 \tag{1}$$

In (1), d is the desired output and y is the actual output. E represents the total sum of errors that can be obtained for the P pattern, whereas i is the i^{th} neuron and j is the number of the output neuron. BPNN uses the Gradient Descent method as indicated in Eq. (2), to minimize the rate of E_P.

$$W_{ki} = -\mu \frac{\partial E_P}{\partial W_{ki}} \tag{2}$$

The variable in Eq. (2), is the weight located between the i^{th} neuron of the n-1 layer, and k^{th} neuron in the n layer. The output layer errors ∂_l, and the hidden layer errors ∂i can be obtained using Eq. (3) and (4), respectively.

$$\partial_l = \mu(d_i - y_i) f'(y_i) \tag{3}$$

$$\partial_i = \mu \sum_i \partial_l W_{lj} f'(y_i) \tag{4}$$

Based on the error rates that obtain in both hidden and output layers, weights can be adjusted to calculate new weights, using Eq. (5) at the hidden layer, Eq. (6) at the output layer, and Eq. (7) for updating the bias values.

$$W_{ij}(K+1) = W_{ij}(K) + \mu \partial_i y_i \tag{5}$$

$$W_{lj}(K+1) = W_{lj}(K) + \mu \partial_l y_i \tag{6}$$

$$b_i(K+1) = b_i(K) + \mu \partial_I \tag{7}$$

In the past three equations, K is the number of epochs and μ is the learning rate. These equations are the core of learning for any BP based NN structure. The learning is the process of updating the weight values that connecting layers through existing neurons (Atakulreka and Sutivong, 2007; Dai and Liu, 2012; Ojha, et al., 2016; Ojha, et al., 2017). The new value of any weight depends on the learning rate and the rate of errors computed in the neurons of the output layer. The value of the learning rate is simply a fixed number between (0, 1), and the value that initializes each weight laid between neurons of two layers is chosen randomly between (−1, +1). With such initialization of the learning rate and the weight values, PBNN falls into local minima.

IV. Nature Inspired Algorithm

These are swarm intelligence and bio-inspired algorithms that would form a hot topic in the expansions of new algorithms inspired by nature (Karaboga, et al., 2014; Nawi, et al., 2014; Akay and Karaboga, 2015; Kumar and Verma, 2012). These are regarded as nature-inspired metaheuristic algorithms that are based on swarm intelligence, biological, physical, and chemical organizations. As a result, these algorithms can be called swarm-intelligence-based, bio-inspired based, physics-based, and chemistry-based, reliant on the foundations of stimulation. It can be said that not all of these different algorithms are effectual and successful (Karaboga, et al., 2014; Nawi, et al., 2014; Akay and Karaboga, 2015; Kumar and Verma, 2012). Some of these algorithms have established to be extremely good, consequently, they have become frequently implemented and modified for solving real-world problems. Although

the research in this area of interest is very dynamic and self-motivated purely for the reason that problems with which researchers and scientists are typically conscious are becoming progressively complex because of size and other aspects. Moreover, recent problems are gathering up constantly on which existing approaches are not dynamic. This gives us the awareness that the Natured Inspired Algorithms and Swarm Intelligence techniques have been there for researchers and scientists in various fields and finalized it for them. That is why now and in the foreseeable future, we need to look as if to attain loads of inspiration from these nature-inspired algorithms. Current Nature-Inspired Algorithms would cover the ABC Algorithm, the Bat Algorithm, and many others (Karaboga, et al., 2014; Nawi, et al., 2014; Akay and Karaboga, 2015; Javadi, et al., 2010; Gen, and Cheng, 1997). The above algorithms have been very fruitful in terms of solving various applications, if they are gaged against initial Nature-Inspired Algorithms such as the GA and others (Gen, and Cheng, 1997). Besides, some of them, for instance, the GA, would have very few parameters that can be randomly set (Kumar and Verma, 2012). We also need to be watchful, since the number of new nature-inspired algorithms is receiving greater. This will make it very difficult for researchers, scientists, and users to select appropriate algorithms for solving their applications. Thus, it is imperative to step back and see the differences among these algorithms, and how they are related to each other. It can be seen that some of these algorithms are nothing more than a reuse of prior optimization ideas. It is worth saying that this problem will bring us to an action plan to be taken by researchers and scientists in general and the action plan is to make a list and group these algorithms, or maybe list out these algorithms in terms of how these algorithms are constructed, their performance, precision, computation complexity, memory, and power usage, etc.

V. ABC

ABC is a population-based optimization algorithm that tries to achieve global minimum (Karaboga, et al., 2014; Akay and Karaboga, 2015). It is a stochastic optimization algorithm that mimics the foraging behavior of the honeybee. Solutions in this algorithm have multi-dimensional search space, which is represented as a food source. Only three types of bees (employed, onlooker, and scout) can maintain the solutions that are given by ABC algorithm. Scout bees will locate the best food source. They will then return to the hive, where other bees onlookers will be recruited to the food source for collection. At this point, the bees are now "employed" in the collection of the food source. By analogy, the fundamental idea of the ABC algorithm is for an agent to look for the best solution. It starts by selecting a random solution among existing space of possible solutions; after that, it continues attempting to find better solutions, while also abandoning the unpromising ones. The iteration of the ABC stops when it reaches either the maximally optimal solution or if or no better suboptimal solution can be found.

The ABC starts with initializing the solutions randomly, as indicated in Eq. (8). Then, the new food source location is updated and obtained by Eq. (9).

$$x_{i,j}=x_{min,j}+rand[0,1]\times(x_{max,j}-x_{min,j}) \quad (8)$$

Where:

$i = 1, 2,..., N$ and $j = 1, 2,..., D$.

$x_{i,j}$ is the parameter to be optimized for ith employ bee, N is the employ bee number.

D is the dimensional size of solution. j is the associated solution in D space with the i^{th} employ bee.

$x_{max,j}$ *and* $x_{min,j}$ are the upper and lower bound for the .

$$v_{i,j}=x_{i,j}+\varphi(x_{i,j}-x_{k,j}) \quad (9)$$

In Eq. (9), $x_{i,j}$ is the i^{th} employed bee, and $v_{i,j}$ is the new solution for the $x_{i,j}$. $x_{k,j}$ is a neighbor bee for the $x_{i,j}$. φ is selected randomly [−1,1]. $j \in \{1, 2,..., D\}$ and $k \in \{1, 2,..., N\}$ selected randomly.

The new food position (solution) will be memorized in an onlooker for one of the n employers, based on the result of the fitness function. The detailed pseudo-code for the ABC algorithm is given below:

Step-1: Initialize the population of solutions $x_{i,j}$, $i = 1. N$, *and* $j = 1 .D$

Step-2: Evaluate the population

Step-3: iteration=1

Step-4: ***repeat***

Step-5: Produce new solutions $v_{i,j}$ *for the employed bees using (9) and evaluate them*

Step-6: Calculate the fitness values for the solutions $x_{i,j}$

Step-7: Produce the novel solutions $v_{i,j}$ *for the onlookers from the solutions* $x_{i,j}$ *selected depending on the evaluation or fitness function*

Step-9: if occurs, control the scout's uncontrolled solution and randomly exchange that with a novel $x_{i,j}$ *solution.*

Step-10: Keep the optimal solution attained until now.

Step-11: iteration=iteration+1

Step-12: while waiting for iteration that is equal to maximum iteration number (MIN).

VI. GA: Mutation or Swapping Process

GA is a method for moving from one population to a new population using a kind of "natural selection" or inspired operators of selection, crossover, and mutation. These three rules are involved in each iteration for producing new individuals for future generations. After selecting an individual, GA uses two rules for creating the next generation. The first rule is a crossover, where new generation comes from a combination of two parents. The second rule is a mutation in which a parent will be reordered to get new generation. The last rule creates random changes on a parent to generate a new child. In most cases, it swaps an element or elements in the parent set. Sometimes, the mutation is only a reordering already existing element.

The number of elements involved in the swapping process affects the time complexity of the process. Involving many elements decreases the complexity measure. However, involving many elements in a swap, on the other hand,

decreases the possibility rate of the swapping impact on finding the global minima for a problem. This article proposes a window that slides bidirectionally from both sides of the weight's vector to the center. The size of this window will be checked in the coming section so that the perfect number of elements that should be involved in the swapping process can be obtained. Figure 1 shows an example of the process of sliding and swapping elements.

In this example, the window size has been set up on two elements, which means that at each slide two elements will be involved in the mutation or swapping process.

The example above shows a vector with 14 elements. The figure clearly shows the direction of sliding, which starts at both ends of the vector forwarding to the center. Step1-A shows the element at their original positions in the vector. Step 1-B shows that the swapping has occurred. Accordingly, two elements have swapped their positions. At Step 2, the windows have shifted just over one element. Step 3 shows the shifting and sliding process. This step also shows the changes that occurred with the position of elements.

Two crucial factors should be noted. The first is the size of the window, and the second is the window's sliding size. Tests have been done to identify these two factors. The aim of this swapping is to get different sequences of elements within the same vector, which means the change has occurred at the position or the index of each element. This requires placing an element in as many various positions in the same vector as possible. Through different tests, the suitable window's size selected as two, and sliding size is one. The other alternatives, such as two or more, showed two main shortages. The first is duplicating and removing many elements. The second is reducing the possibility of getting a new sequence of elements.

VII. ANN Training with Mutated ABC

The core fundamental of training ANN is presented in Eq. (8). Through this equation, a new set of weights is generated from the old set based on learning and error rates. ANN with Back-Propagation learning stacks more often with the problem of local minimum through using such a way of weight updating. The reason for facing such a problem is going back to the structure of the ANN or the way that the ANN is trained (Nawi, et al., 2014). To solve this problem, some solutions have been proposed. The most common is using optimum algorithms, which is somewhat time-consuming. Another way is to focus on initialization methods of the weights. An additional method is to train the neural network more than once; each time the neural network will be initialized randomly on different weights. All these methods can help us find the global optimum of the training (Akay and Karaboga, 2015).

The only difference between neural network and other optimization methods is in the means by which of weights are updated. Equations (6) and (9) show two ways of weight updating that are used in ANN and ABC, respectively. The question is how the accuracy of an ANN will be if the network is trained through Eq. (9) instead of Eq. (6)? Another alternative to repeatedly initialize an ANN on different weights is swapping the weights and changing their locations. Thus, within these two processes, the problem of trapping the ANN in the local minima can be solved. Figure-2 shows the process of the training an ANN through these weights updating and relocations in two simple loops. The outer loop is to initialize and update the weights for an ANN using the Eq. (9). Then, if the global minima are not obtained, the mutation or swapping weights will be tested as the inner loop. The size of the weightings set will be defined with the structure definition of an ANN.

The steps below show the pseudo-code of the mutation based ABC optimization for training ANN:

Step A: Through the predefined structure of the ANN (by which the structure should fit the problem that the ANN is designed for) the size of the initial set (weights) is dynamically defined.

Step B: Initializing the ABC algorithm through the steps that are shown in the section of ARTIFICIAL BEE COLONY. Get the first source of weights.

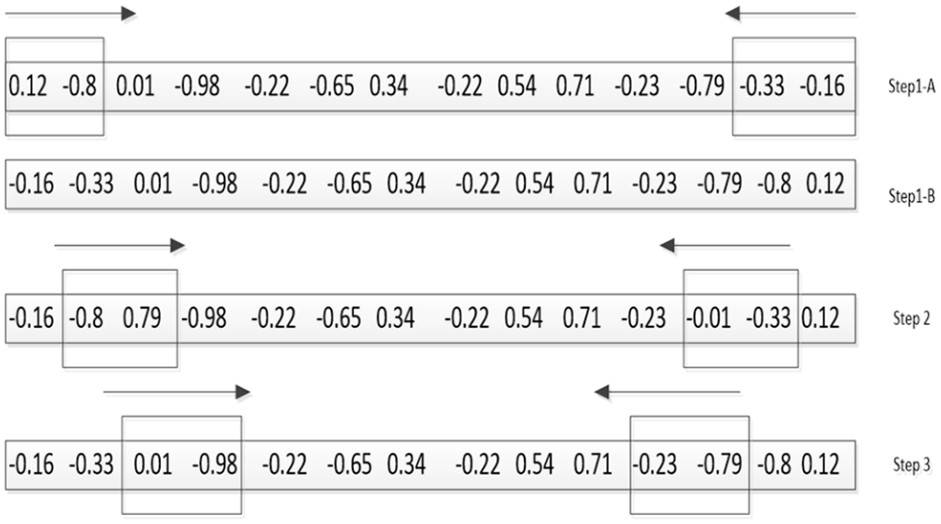

Fig. 1. Sliding and swapping elements.

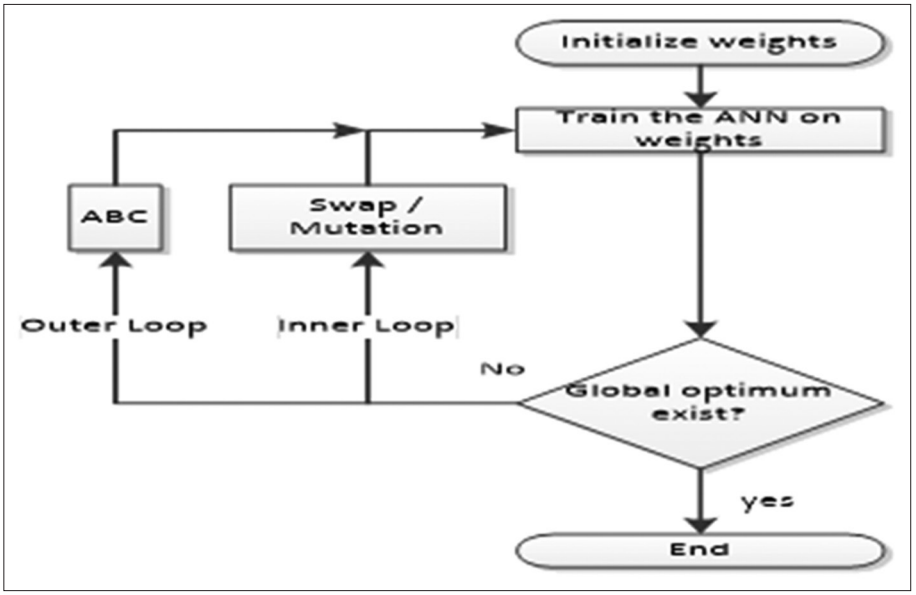

Fig. 2. The process of mutation artificial bee colony training of artificial neural network.

Step C: Do the mutation between the elements in the initial set of weights (get from Step B).

Step D: Initialize the ANN training, validation, and testing.

Step E: Compare the obtained performance with the previous one. Select the best.

Step F: Repeat Step C until no new offspring will be obtained. Repeat the B until no more sources are available.

VIII. Simulation and Experimental Works

Various experimental tests are carried out, and details of these can be described as follows:

A. BPNNs

The BPNN has three layers, where they are connected trimly through the neurons that lay at each layer, and the weights that lay between two neurons of two different layers. The first layer is input layer, which the number of neurons there should be equal to the number of the features that are selected as significant for a problem. The subsequent layer is the hidden layer, which enables the BPNN to achieve better performance. The last layer is the output, where the actual output is expected. Fig. 3 shows a typical structure of a BPNN (Ojha, et al., 2017). The structure of the BPNN for both datasets is not the same. For the local dataset, the number of features is 13 (Rashid, et al., 2016); however, for the Pima dataset, the input features are 8 (UCI, Pima India Dataset, 2015).

This means that the input layer for both cases has 13 and 8 neurons, respectively. For both cases, the neurons at the hidden layer are ,10 and at the output layer is one. The only difference in both structures is the number of neurons at the input layer. Therefore, the set of initial weights and biases will be different too (151 for local dataset and 101 for Pima dataset). The training function that used by this work is "*trainlm"* function.

The activate function proposed for neurons in the hidden layer structure is sigmoid function. The neuron at output layer has a linear activate function.

The process of ANN's training has been tested in three different situations. The first is training ANN normally, which means updating the weights of the ANN per Eq. (6). The second situation is forcing the proposed ANN to update their weights and bias values per Eq. (9), which is the equation that used by ABC algorithm. The last scenario is the same as the second scenario; however, the positions of the initialized weights in the second scenario will be changed (swapped) based on the steps of GA. For all scenarios, the performance indicator measured is calculating the accuracy rate based on the true and false classification rates.

B. Normal Training of BPNN

"Trainlm" is the most common training function used in BPNN. It is also known as the Levenberg-Marquardt optimization algorithm (Priddy and Keller, 2005). The main target of this learning function is to train the network until a goal is achieved through the adjustment of the weights' values that initialized the network. Through this learning process, the BPNN minimizes the errors that obtained at output layer. The training process of the BPNN terminates immediately when the goal is obtained. However, the obtained goal is not guaranteed to be optimal, as the BPNN depends on gradient descent method. Another factor that terminates the training is the number of failure validation checks. The BP algorithm stops training when the validation check reaches the predefined maximum number. This situation of BP is an indicator of trapping into a local problem, where no performance improvements can be achieved as the training directs the network to the worst situation of learning.

BPNN starts to find solutions from an initial point and continues until the steepest point (a point down toward the

 http://dx.doi.org/10.14500/aro.10368

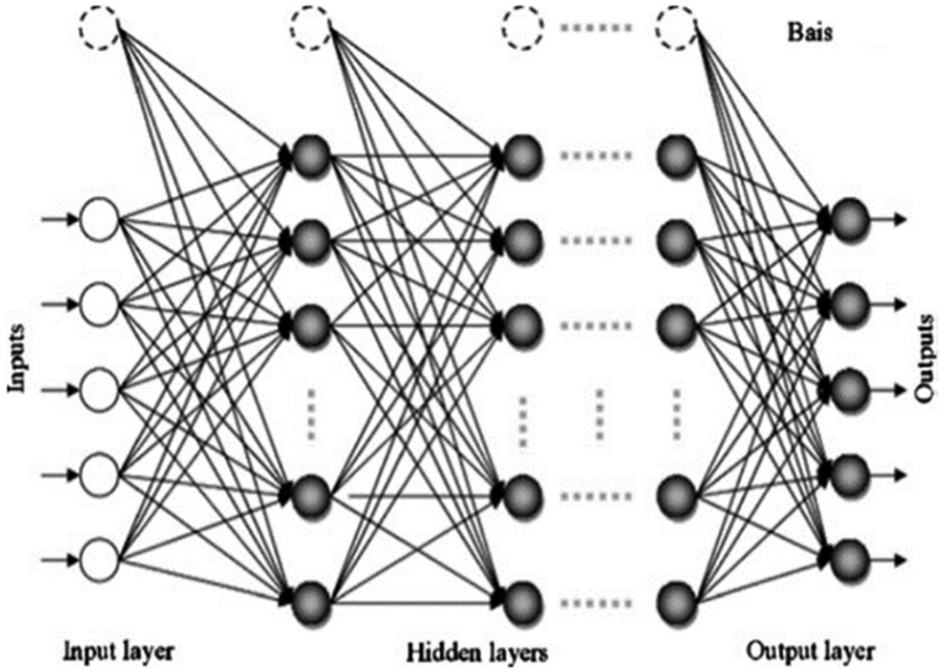

Fig. 3. Typical structure for backpropagation neural network (Priddy and Keller, 2005).

optimum minimization). Two parameters affect the training of the BPNN. The first is called learning rate, in which the value is ranged between zero and one. A small value increases the iteration number, and a large value avoids the BPNN to find the optimum solution. To address that, this work tests the training of the BPNN with two datasets. For each dataset, the network has been initialized with 300 initial sets. Each set has *w* number of weights and *b* number of bias values. This work needs to initialize the proposed BPNN 300 times, and each time the initial weights have a different number.

The performance of the BPNN for both dataset and for each set of initial weights will be recorded. Then, the minimum and maximum performance for each dataset will be selected and compared, as shown in Table I. Table 1 also shows the CPU time, which measured by iterations, for both datasets. This performance indicates that the network has found a solution. However, it is not clear whether the solution is optimum. Nevertheless, Fig. 4 makes clear that the no further improvement can be obtained from that solution, as the validation check shows that training is going in the wrong direction. Even more, the gradient degree at the same epoch (iteration = 8) is going up, which confirms trapping the network into a solution where no better solution could be reached.

Fig. 5 shows the performance of a BPNN that normally trained with 300 different sets of the initial weights. The network starts each time training with an initial set of weight and bias values (1 × 151), which obtained randomly.

Normally, when the BPNN enters the training phase, the values of the weights and the biases will be randomly initialized in the range of −1, +1. The figure shows numerous solutions, which obtains when the initial weight has been changed. The figure also explains the impact of the initial weight on the process and the direction of the training phase of the BPNN. This means that the BPNN can achieve better

TABLE I
BPNN PERFORMANCE WITH NORMAL TRAINING

Datasets	Performance (MSE)		CPU time
	Best	Worst	
Pima	0.1324	0.1856	23.2
Local data	0.1298	0.1926	95.4

BPNN: Back propagation neural network

learning if the weights and biases are initialized with proper values. However, the BPNN has no ability to change the initial values of weights and biases after the training phase begins. Therefore, the BPNN cannot change the direction of learning when the network has been initialized with improper values of weights and biases.

To overcome this problem, BPNN should be able to find a proper set of weight and bias values among available alternatives.

Practically, researchers follow the *K-fold* method (Kohavi, 1995) to determine the best solution or the optimal average of the obtained solutions. They run the training phase of a BPNN several times. With each run, the accuracy of the network will be preserved. Then, after *K-fold*, either the best accuracy will be selected or (in most cases) the average of available solutions will be calculated. With such a process, it is not guaranteed that the best solution is the optimum, because there might exist a set of weights and biases that makes BPNN yield higher accuracy than that obtained through the *K-fold* method. Therefore, with such a process there is no guarantee that the best solution found is the optimum solution.

C. ABC based BPNN training

The core of the training in BPNN is updating the weights to minimize errors at the output stage. To achieve this, the set

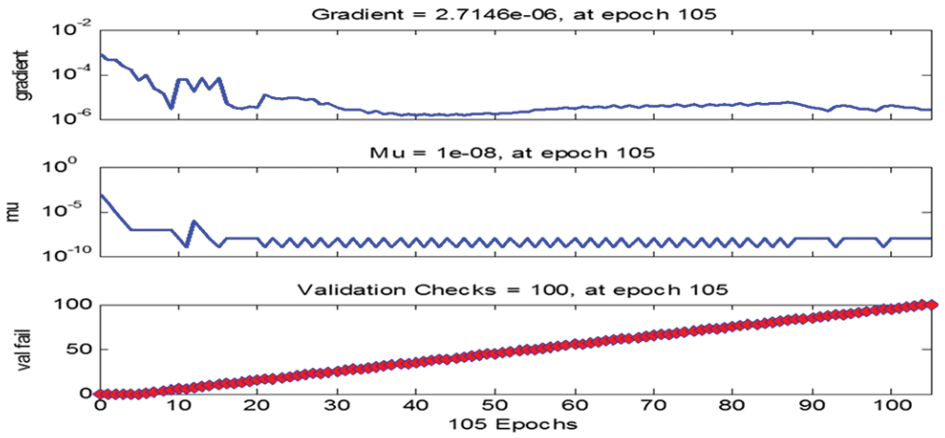

Fig. 4. Failed validation for better solution obtaining.

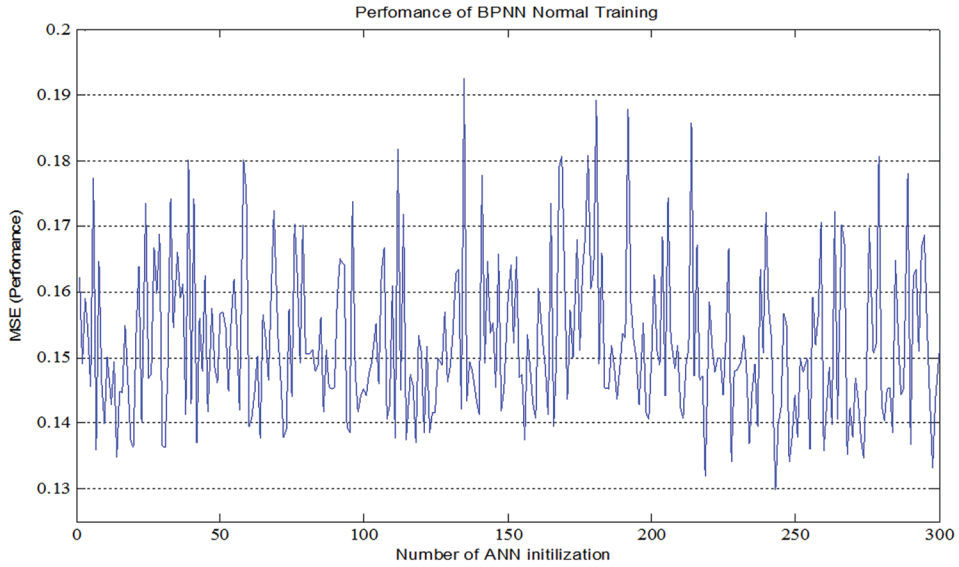

Fig. 5. Performances of normal trained Backpropagation neural (local dataset).

of initialized weights of the network will be updated based on (6). The network uses only one set of initial weights and based on the error rate at the output stage, and the learning rate the new set of weights will be calculated for the next round of training. This means that only one set of weights is utilized in the training process. The idea of involving the ABC algorithm in this work is for utilizing various sources (sets) of weights as initialized weight, where the BPNN can start with. It is a kind of collaboration between these two techniques to find out the perfect set of weights that directs BPNN to find the perfect (optimal) solution. The main changes on the process of the weight updating have been presented in this work, which, are initializing the weight approximately300 times. As a result, each time the process of updating the weight is going on through the Eq. (9) instead of Eq. (6). Fig. 6 shows the training performance, which becomes more stable than that found in Fig. 5. The local minima solution that obtained through this weight initialization and updating was not found in the obtained performance when the network trained normally.

ABC can serve BPNN with finding the proper set of weights and biases value among the n numbers of the population. However, there is a possibility of getting a better solution than the combination of the BPNN with ABC would do.

All elements in the sets have the same index during training a BPNN with ABC. The question is about the possibility of improving the accuracy of a BPNN when an element with an index of (i,j) can be swapped with another element that has an index $(i\pm n,j\pm n)$ when $n\neq 0$.

To test the impact of mutation the weight's indexes on the performance of the BPNN, this article checks the performance of the network with a single set of weights (1 × 81) for the local dataset. As mentioned in Section 5, the sliding window is shifting by one element each time with doing the mutation on two elements. Accordingly, each single set of weights that have (1 × 81) dimension can be used to generate 80 different sets that are having the same elements of the mother set. However, in each iteration, the elements will be in different indexes. Fig. 7 shows that a

http://dx.doi.org/10.14500/aro.10368

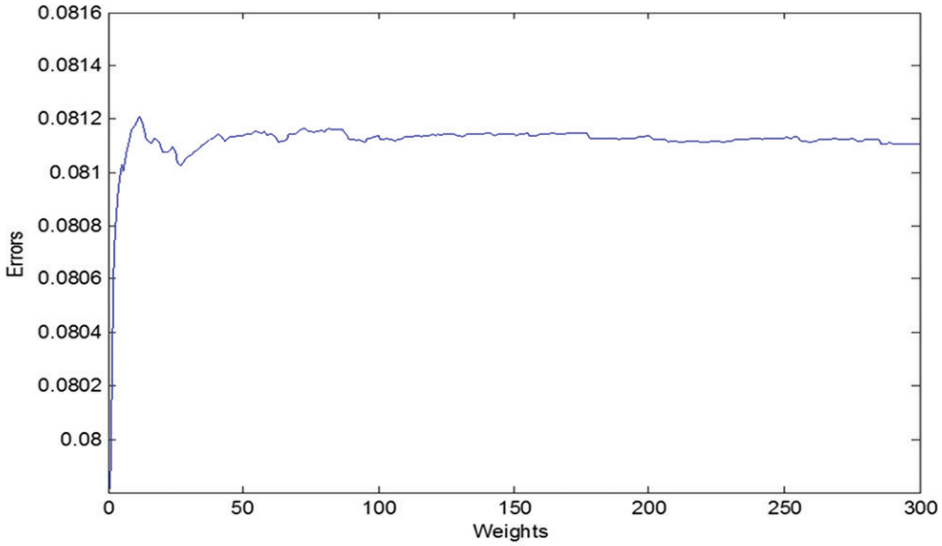

Fig. 6. Backpropagation neural network training based on artificial bee colony source (weights) initializing and updating.

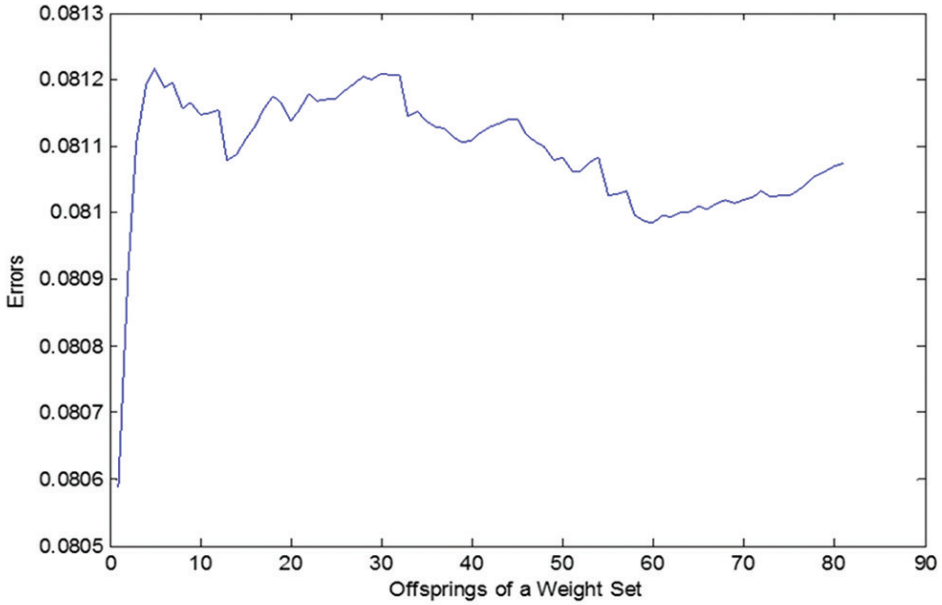

Fig. 7. The 81 offspring of a weight set with different performance.

set of weights can be used to generate different offspring, and with each, the network gives a specific performance. As a set of weights which has been mutated, based on the process explained in Section 5, 81 sets of weights have been obtained. Then, each set has been used to initialize the BPNN. After that, only one observation is used to test the network performance, with a change of offspring on that original set of weights. Results show that different performances have been obtained due to the change of the indexes of the elements inside the set weight just for a single observation. From those performances, and based on the concept of the ABC algorithm, the best performance can be selected as the global minima for that observation.

For the Pima dataset, the above-mentioned procedure and what has been illustrated in Fig. 7 will be repeated. However, the number of the offspring, in this case, will be 51, as the number of input attributes will be reduced to eight.

Fig. 8 shows the 51 performances that have been obtained for observation with each offspring weight. Considering the locally prepared dataset, the training performance based on 501 local observations can be illustrated as shown in Fig. 9.

This research article assesses the performance of the network with four parameters (CPU time, number of epochs, MSE, and Accuracy) for testing the impact of mutation of the weight's indexes on the performance of the BPNN, the workstation used for the experimentation was equipped with a 2.5 GHz Core-i5 processor and 4-GB of RAM. The simulations are carried out using MATLAB 2013a software. We employed two different data sets to check the impact of the proposed approach of this work on

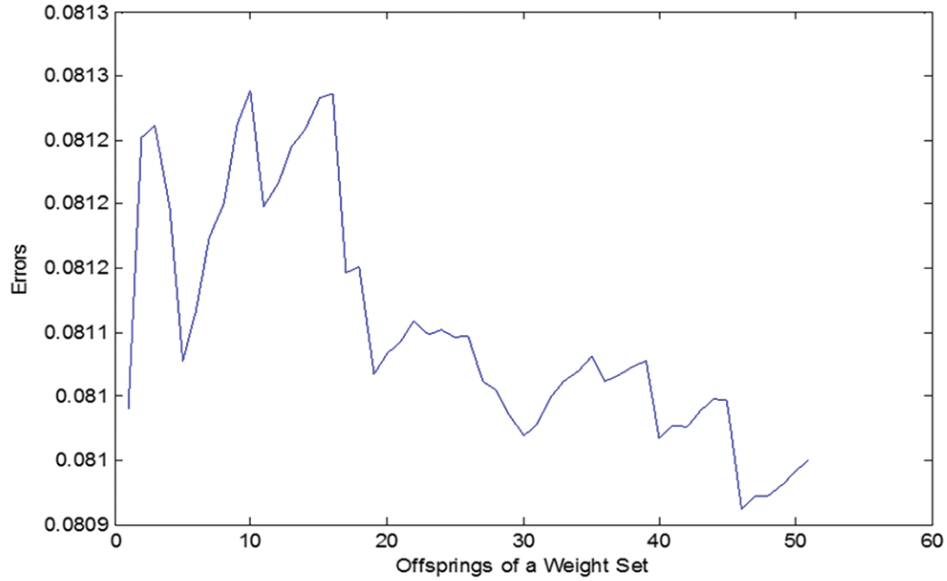

Fig. 8. The 51 offspring of a weight set for Pima dataset with different performances.

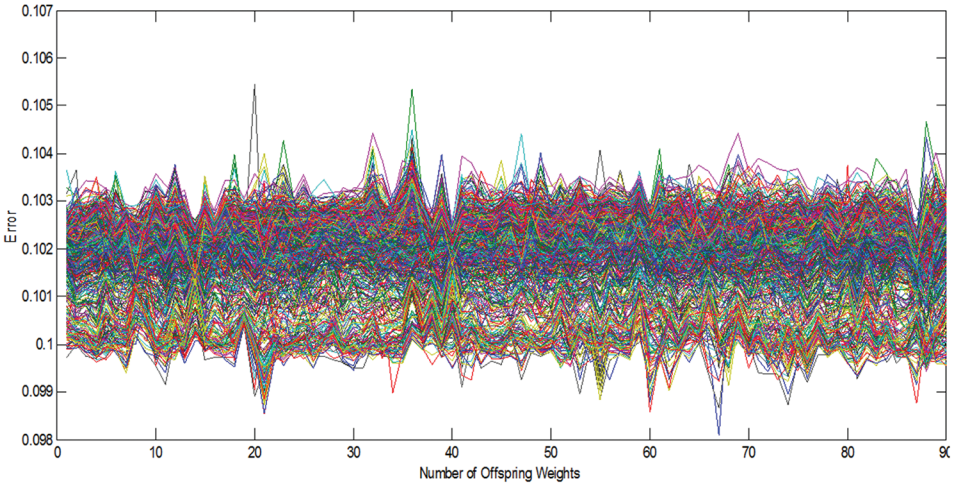

Fig. 9. Network training with 501 observations (each with 91 offspring).

the performance measures. We applied four criteria to three methods: ANN alone, ANN with ABC optimization, and ANN-ABC with weight mutation. Tables II and III show the performance measures of the tests on ANN, ANN-based ABC and ANN-based ABC with weight mutation for both local and Pima datasets, respectively. The tests show that accuracy is improving with the change of weight initialization of the BPNN. The normal initialization cannot improve the accuracy of the trained BPNN within a wide range. The improvement just recorded for a few points of a percent.

IX. Conclusion

It is a fact that BPNN can give different performances when initialized with different values of weights. This means that changing the values of weights for a BPNN affects the performance of that network. However, if a network initializes with same values of weights, it gives the same performance

TABLE II
Testing the approach with local dataset

Algorithms/performance measure	ANN	ANN-based ABC	ANN-based ABC with weight mutation
CPU	76.2	123.6	149.4
Epoch	500	1000	1000
Error	0.132	0.0047	0.0013
Accuracy%	93.2	96.11	98.38

ANN: Artificial neural network, ABC: Artificial bee colony

TABLE III
Testing the approach with pima data set

Algorithms/performance measure	ANN	ANN-based ABC	ANN-based ABC with weight mutation
CPU	23.2	95.5	113.1
Epoch	1000	1000	1000
Error	0.197	0.09	0.0045
Accuracy%	89.99	94.77	97.32

ANN: Artificial neural network, ABC: Artificial bee colony

if the training process is repeated for unlimited iterations. The target of changing the weight's values for a network is optimizing the performance (avoiding local optima). Many algorithms have been combined with BPNN, such as ABC, for maximizing the performance. With such algorithms, sets of initial weights are randomly generated, and the best set of weights will be selected based on the best performance obtained. We investigated the impact of the indexes of each weight inside the initialized set on the BPNN performance. Through optimized algorithms, weights are initialized statically (no changes in their indexes have been made). When an index of weight inside the initialized set changes, the performance is also changed, as shown in Figs. 7 and 8 with Tables II and III. This means that there is a possibility of getting better performances if the indexes of weights are changed.

The figures and tables show that initializing weights randomly as done by ANN performs less efficiently and accurately than ANN combined with ABC. However, when the combination of ANN and ABC supported by index sliding of GA, the resulting performance is optimal. It is not the impact of changing the sources (as in the ABC algorithm), it is instead the impact of changing the indexes of the weights too. With such combinations, ABC can force the training process of any BPNN to become more optimal in classification.

Another important problem solved in this article is overfitting of ANN training. We tested the approach with two different datasets, which means the proposed approach is generalized for the different dataset. This also means that our approach overcomes the problem of overfitting.

X. ACKNOWLEDGMENT

The authors would like to thank the editorial office of the journal for reviewing of the manuscript. Furthermore, the authors would like to thank both Mr. Edward Bassett from the English Language Centre (a Juris Doctorate from the University of Missouri-Columbia Law School (USA) and a Master's in Fine Arts (Creative Writing) from the University of Southern Maine (USA) and Mr. Shalaw Najat Ghani (MA in TESOL), from Valparaiso University for their continuous effort in editing the manuscript.

References

Akay, B. and Karaboga, D., 2015. A survey on the applications of artificial bee colony in signal, image, and video processing. *Signal, Image and Video Processing,* 9(4), pp. 967-990.

American Diabetes Association., 2014. Diagnosis and classification of diabetes mellitus. *Diabetes Care,* 37(Supplement 1), pp. S81-S90.

Atakulreka, A. and Sutivong, D., 2007. Avoiding local minima in feedforward neural networks by simultaneous learning, in AI 2007, *Advances in Artificial Intelligence*, Springer. Berlin Heidelberg, pp. 100-109.

Dai, Q. and Liu, N., 2012. Alleviating the problem of local minima in Backpropagation through competitive learning, *Neurocomputing*, 94, pp. 152-158.

Gen, M. and Cheng, R., 1997. Genetic Algorithms and Engineering Design, John Wiley & Sons, New York.

International Diabetes Federation., 2014. IDF Diabetes Atlas, IDF. International Diabetes Federation. Available from: http://diabetesatlas.org/resources/2017-atlas.html. [Last retrieved on 2014 Nov 29].

Javadi, M.R., Mazlumi, K. and. Jalilvand, A., 2011. Application of GA, PSO and ABC in optimal design of a stand-alone hybrid system for north-west of Iran, *in Electrical and Electronics Engineering (ELECO)*, 2011 7th International Conference on, pp. I-203-I-210.

Karaboga, D., Gorkemli, B., Ozturk, C. and Karaboga, N., 2014. A comprehensive survey: Artificial bee colony (ABC) algorithms and applications. *Artificial Intelligence Review*, 42(1), pp. 21-57.

Kohavi, R., 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Vol. 2. Morgan Kaufmann, San Mateo, CA, pp. 1137-1143.

Kumar, R. and Verma, R., 2012. Classification rule discovery for diabetes patients by using genetic programming. *International Journal of Soft Computing and Engineering*, 2(4), pp. 183-185.

Nawi, N.M., Ghazali, R. and Salleh, M.N.M., 2010. The development of improved back-propagation neural networks algorithm for predicting patients with heart disease. In: *Information Computing and Applications*, Springer, Berlin. pp. 317-324.

Nawi, N.M., Ghazali, R. and Salleh, M.N.M., 2011. Predicting patients with heart disease by using an improved back-propagation algorithm, *Journal of Computing*, 3(2), pp. 53-58.

Nawi, N.M., Rehman, M.Z. and Khan, A., 2014. A new bat based back-propagation (BAT-BP) algorithm, In: *Advances in Systems Science*, Springer, Berlin. pp. 395-404.

Ojha, V.K., Abraham, A. and Snášel, V. 2017. Metaheuristic design of feedforward neural networks: A review of two decades of research, *Engineering Applications of Artificial Intelligence*, 60, pp. 97-116.

Ojha, V.K., Dutta, P., Chaudhuri, A. and Saha, H. 2016. Convergence Analysis of Backpropagation Algorithm for Designing an Intelligent System for Sensing Manhole Gases. In: *Hybrid Soft Computing Approaches*, Springer, Berlin, pp. 215-236.

Priddy, K.L., and Keller, P.E., 2005. *Artificial Neural Networks: An introduction*. SPIE Press, Bellingham.

Rashid, T.A., Abdullah, S.M., and Abdullah, R.M., 2016. An Intelligent Approach for Diabetes Classification, Prediction and Description, In: Snášel, V., Abraham, A., Krömer, P., Pant, M., Muda, A., editors. *Innovations in Bio-Inspired Computing and Applications. Advances in Intelligent Systems and Computing*, Vol. 424. Springer, Cham.

UCI. 2015. Pima India Dataset. UCI Machine Learning Repository. Available from: https://www.archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

Wild, S., Roglic, G., Green, A., Sicree, R., King, H. 2004. Global prevalence of diabetes estimates for the year 2000 and projections for 2030. *Diabetes Care*, 27(5), pp. 1047-1053.