International Journal of Speech Technology (2019) 22:1039–1049
https://doi.org/10.1007/s10772-019-09640-7

Evaluation of PNN pattern-layer activation function approximations in different training setups

Nikolay T. Dukov[1] · Todor D. Ganchev[1] · Michael N. Vrahatis[2]

Received: 27 May 2019 / Accepted: 23 September 2019 / Published online: 9 October 2019

Abstract
The processing of inputs in the first two layers of the probabilistic neural network (PNN) is highly parallel which makes it quite appropriate for hardware implementations with FPGA. One of the main inconveniences however remains the implementation of the nonlinear activation function of the pattern layer neurons. In the present study, we investigate the applicability of three approximations of the exponential activation function with look-up tables of different precision and the effect this has on the training process and the classification accuracy. Furthermore, seeking for a highly-parallel hardware-friendly algorithm for the automated adjustment of the spread factor σ_i, we investigated the performance of fifteen PNN training setups, which are based on the differential evolution (DE) or unified particle swarm optimization (UPSO) methods. The experimental evaluation was performed following a common experimental protocol, which makes use of the Parkinson Speech Dataset, as this research aims to support the development of portable medical devices that are capable to detect episodes with exacerbation in patients with Parkinson's disease. The performance of the most successful setups is discussed in terms of error rates and from the perspective of the resources required for an FPGA-based implementation.

Keywords Probabilistic neural network · Differential evolution · Particle swarm optimization · Parkinson speech dataset · Hardware-friendly algorithm

Abbreviations

CPU	Central processing unit
DE	Differential evolution
EA	Evolutionary algorithm
FPGA	Field-programmable gate array
GA	Genetic algorithm
GPU	Graphics processing unit
LUT	Look-up table
PDF	Probability density function
PNN	Probabilistic neural network
PSO	Particle swarm optimization
UPSO	Unified particle swarm optimization

✉ Todor D. Ganchev
tganchev@tu-varna.bg

[1] Applied Signal Processing Laboratory (ASPL), Faculty of Computing and Automation, Technical University of Varna, 9010 Varna, Bulgaria

[2] Computational Intelligence Laboratory (CI Lab), Department of Mathematics, University of Patras, 26110 Patras, Greece

1 Introduction

Parkinson's disease, which causes motor system problems, reportedly is the second most common degeneration of the central nervous system in elder ages after the Alzheimer's disease (Sakar et al. 2013). Although at present there is no cure, it is known that diet and rehabilitation help suppressing motor symptoms for certain periods of time; however the disease development needs to be monitored on a daily basis as common long-term consequences include dementia, depression, anxiety, sensory, and behavioral problems. Non-invasive automated detectors of episodes with exacerbation in patients with Parkinson's disease would be especially important for people who live on their own. Such medical monitors and other portable devices can benefit from the automated detection of events with diagnostic significance (Parisi et al. 2018; Ullah et al. 2018; Palo et al. 2018; Zhang et al. 2016; Lavanyadevi et al. 2017; Mathew et al. 2018). A straightforward approach to implement such a functionality is to deploy a software application on a readily available hardware platform with some popular operating system, for instance on a smart-phone (Brandenburg et al. 2013; Boulos et al. 2014; Alshurafa et al. 2014; Moser and

Melliar-Smith 2015). Among the major weaknesses of such an approach are the complexity to combine multiple sensors measuring physiological signals (in addition to speech), the low reliability and low power efficiency. The low reliability stems from the multi-functionality of smart-phones and the risks other software applications to compromise the data collection and monitoring, or simply drain the battery. Even when avoiding collision with other applications is manageable and hardware resources are abundant, it is difficult to provide prolonged autonomy of operation due to the limited capacity of smart-phones' battery.

Hardware implementations of machine learning algorithms bring certain benefits among which are a lower energy consumption and improved reliability. These advantages are especially prominent when CPU-, GPU-, and FPGA-based implementations providing equivalent computational throughput are compared (Hussain et al. 2011; Van Essen et al. 2012). The lower energy consumption in FPGA-based implementations is due to the opportunity to organize the required computations in a highly parallel manner, which translates to a multi-fold reduction of the operational clock frequency and therefore to a significant reduction of the overall energy consumption (Birk et al. 2012; Mittal and Vetter 2014). However, hardware implementations of automated recognizers also bring an inconvenience due to the limited flexibility of model retraining or adaptation. For instance, when the patient condition improves due to treatment or worsens due to progress of disease, model adaptation might be required. Here we investigate the opportunities to implement the required modeling and classification functionality through a probabilistic neural network (PNN) (Specht 1990) as it is easy to retrain and is not sensitive to the finite-word-size effects, associated with the representation of numbers with limited number of bits. Furthermore, PNN is among the classifiers shown as particularly appropriate for various hardware implementations (Zhou et al. 2010; Wang et al. 2011; Akhmetov and James 2019; Krestinskaya and James 2018; Chen et al. 2019; Vipin et al. 2018).

In brief, PNN (*cf.* Section 2.1) implements a robust statistical classification method, which deals very well with outliers and offers a good discriminative capability as soon as some (representative) training samples are available for each class. The original PNN introduced by Specht (Specht 1990) has one adjustable parameter—the spread factor σ_i, which controls the smoothing of the probability density function surface defined by the underlying training data. Although in many applications the value of σ_i is not crucial for the reliable operation of the classifier, the overall recognition accuracy could be significantly improved when σ_i is adjusted in a sensible data-driven manner. Most often the parameter σ_i had been adjusted heuristically, by conjugate gradient and approximate Newton methods, conjugate gradient-based approach proposed by Specht, fusion of orthogonal and genetic algorithm, reinforced learning, or by means of some evolutionary algorithms (EA)-based method (Chtioui et al. 1998; Kusy and Zajdel 2015; Mao et al. 2000). For instance, the applicability of a particle swarm optimization (PSO)-based method for adjustment of σ_i was studied in (Georgiou et al. 2006, 2008; Hsieh and Chen 2009; Huang and Du 2008; Ganchev 2009; Zhou et al. 2017). Other studies with self-adaptive PNNs investigated automated adjustment of σ_i based on representative observational datasets. Successful results were reported on various applications, such as driving assistance systems (Li and Ma 2008), economics (Ciarelli et al. 2009), and fault detection (Samanta et al. 2006). Nearly all studies supported that the performance of the PNN could be improved significantly when the parameter σ_i is set in an adequate way. Although recent studies focused on PSO as the preferable evolutionary algorithm for the optimization of σ_i, there is evidence that Differential evolution (DE)- and genetic algorithm (GA)-based methods may offer favorable results (Behera et al. 2011; Acharya et al. 2013; Ford et al. 2013). For comprehensiveness of exposition in Section 2.3.2 we summarize the fundamentals of DE and in Section 2.3.3 the UPSO-based optimization.

In the present work, we study a PNN-based classifier for the identification of Parkinson's disease from speech. Heading towards an FPGA-based implementation of the PNN, we aim to select an appropriate approximation of the exponential activation function in the pattern-layer neurons (*cf.* Sect. 2.2), which requires a significant portion of the FPGA resources. For that purpose, here we investigate the applicability of three approximations with look-up tables of different precision, and the effect this has on the training process and the classification accuracy. Furthermore, we investigate fifteen training setups (*cf.* Sect. 2.3) for the adjustment of σ_i, which are compared in a common experimental protocol based on the Parkinson Speech Dataset. Among these are DE and UPSO-based setups, which are known to be intrinsically parallel and were reported to provide a good trade-off between computational demands and classification accuracy (Parsopoulos and Vrahatis 2002; Plagianakos and Vrahatis 2002). To the authors' best knowledge a study on the applicability of DE and UPSO algorithms for the adjustment of σ_i, when the pattern-layer activation function is approximated with a look-up table, has not been carried out, yet. Discussion on the experimental results and a comparative assessment of complexity for the top-4 setups, which provide the highest classification accuracy, is offered in Sect. 3. Based on the experimental results, we draw conclusions (Sect. 4) about the appropriateness of various approximations of the exponential activation function and for the training setups for adjusting σ_i. Eventually, we discuss the most advantageous solutions in terms of recognition accuracy and complexity of potential implementation with an FPGA chip.

2 Materials and methods

2.1 Probabilistic neural network

The PNN (Specht 1990) is a supervised, nonlinear, non-parametric pattern recognition algorithm that estimates a probability density function (PDF) for each and every class and then uses it in Bayesian optimal classification scheme. Most often, the PDF is estimated by employing a sum of spherical Gaussian functions that are centered at each training vector. Consequently, the PNN makes the classification decision in accordance with the Bayes' strategy for decision rules. Besides the decision, PNN also provides probability and reliability measures of each classification. The first layer of the PNN, designated as an input layer, accepts the input vectors to be classified. The nodes of the second layer, which is designated as a pattern layer, are grouped depending on the class they belong to. These nodes, also referred to as pattern units or kernels, are connected to all inputs of the first layer. Here we consider each and every pattern unit defined as having the Gaussian activation function:

$$f_{ij}(\mathbf{x};\mathbf{c}_{ij},\sigma_i) = \frac{1}{(2\pi)^{d/2}\sigma_i^d}\exp\left(-\frac{1}{2\sigma_i^2}(\mathbf{x}-\mathbf{c}_{ij})^T(\mathbf{x}-\mathbf{c}_{ij})\right), \quad (1)$$

where $i = 1, \dots, K, j = 1, \dots, M_i$. Here, σ is the standard deviation, and is also known as *spread* or smoothing factor σ_i, which regulates the receptive field of the kernel. The input vector $\mathbf{x}$ and the centers $\mathbf{c}_{ij} \in R^d$ of the kernel are of dimensionality d, and M_i is the number of pattern units in a given class k_i. Finally, exp stands for the exponential function, and the superscript T denotes the transpose of the vector. Obviously, the total number of the second-layer nodes is given as the sum of the pattern units for all classes.

Next, the weighted outputs of the pattern units in the second layer are connected to the summation unit in the third layer (*aka* summation layer) corresponding to that specific class. The weights for the member functions of class k_i are positive numbers that sum to unity and are determined based on the decision cost and *a priori* class distribution. Each node of the summation layer estimates the class-conditional PDF:

$$\begin{aligned}\mathbf{U}_i^{(k+1)} &= u\mathbf{G}_i^{(k+1)} + (1-u)\mathbf{L}_i^{(k+1)}\\ \mathbf{w}_i^{k+1} &= \mathbf{w}_i^k + \mathbf{U}_i^{(k+1)}\end{aligned} \quad (2)$$

where $i = 1, \dots, K$. In the output layer (*aka* competitive layer), the PNN decides the winning class k_i to which the input vector $\mathbf{x}$ belongs based on the Bayesian decision rule (3). This makes use of the estimation of the class-conditional PDF (2) for all classes computed from the training data and the *a priori* class probability $P(k_i)$, which in many applications is known in advance, or otherwise is usually assumed uniform.

$$D(\mathbf{x}) = \text{argmax}_i\{P(k_i)p_i(\mathbf{x} \mid k_i)\}, i = 1, \dots, K. \quad (3)$$

In such a way, once the training vectors are stored in the pattern layer and the parameter σ_i is computed, the PNN classifier is ready for operation.

2.2 Approximation of the exponential function

Although, different activation functions exist, traditionally for the PNN the exponential function is used, which is proven as an advantageous in variety of tasks. In FPGA designs, the exponential function exp in (2) can be implemented by means of a look-up table (LUT) with precomputed values or computed on-the-fly by means of some power series (for instance the Taylor series), Euler's continued fraction formula, or another approximation. Aiming at the highest speed of computation during training and operation of PNN, here we assume LUT-based approximation of the exponential function, where the LUT values are precomputed as $\Lambda = \exp(-V)$. The values of $\mathbf{V}$ correspond to the argument of the exponential function in (2) and therefore depend on the computed Euclidean distance and the spread factor σ_i. Due to this dependence on the Euclidean distance, the range of $\mathbf{V} = [V_1, \dots, V_i, \dots, V_N]$ depends on the dynamic range of feature vectors $\mathbf{x}$, i.e. whether the feature vectors are raw, normalized, scaled, etc. Here, the range of values is specified with the lowest boundary V_1, below which the approximated exponential function Λ_{idx} will not change value, and the upper boundary V_N, above which the approximated exponential function Λ_{idx} will not change value. During the LUT creation, any V_i is obtained as the increased value of V_1 with $i-1$ steps of size q. The number of steps $N = (V_N - V_1)/q$ of such an approximation, i.e. the size of the LUT table, depends on the desired resolution. For a precomputed LUT, we can calculate the index idx (4) for any given argument V and retrieve the value Λ_{idx}:

$$idx = \text{round}\left(\frac{V}{q}\right), \text{ for } V = -\frac{1}{2\sigma_i^2}(\mathbf{x}-\mathbf{c}_{ij})^T(\mathbf{x}-\mathbf{c}_{ij}), \quad (4)$$

where round(.) rounds to the closest integer number. Certainly the useful range of V is restricted to $V_1 \le V \le V_N$, and values outside this range will be attributed to the lowest or the upper LUT value, i.e. $1 \le idx \le N$.

2.3 Experimental setup

The experimental protocol and the Parkinson Dataset are outlined in Section 2.3.1, and Sections 2.3.2 and 2.3.3 provide a concise outline of the DE and UPSO methods used in the optimization of σ_i. The fifteen setups considered here for the adjustment of the class-specific σ_i values are outlined in Sect. 2.3.4.

2.3.1 Experimental protocol

The experimental evaluation was carried out on the Parkinson Speech Data Set with Multiple Types of Sound Recordings from the UCI machine learning repository (Sakar et al. 2013). This dataset contains speech feature vectors with dimensionality 26, computed from recordings of 20 healty individuals and 20 patients diagnosed with Parkinson's disease. The original dataset is split to 1040 train and 168 test feature vectors which are provided in two separate ASCII files with comma separated values. For this specific split of data, our heteroscedastic PNN detector obtained 100 % recognition accuracy, regardless of the manner in which the class-specific σ_i values were adjusted. As this is not convenient for the purpose of the intended comparative evaluation, likewise (Sakar et al. 2013), we implemented an experimental protocol which does not make use of the provided split to train and test data. Specifically, we merged all train and test data to a single matrix with cardinality 1208, and in the following considered *ten times cross-validation* scheme with 50 runs per data split. The main goal was to evaluate the PNN training and operational performance for different approximations of the pattern-layer exponential activation function. Here we consider a comparison of the three approximations (*cf.* Table 1) vs. the exponential function computed with 64-bit double-precision according to the IEEE Standard for Floating-Point Arithmetic (IEEE 754). In the last column of Table 1, we show the number of LUT values used in each approximation.

2.3.2 Differential evolution

The differential evolution (DE) (Storn and Price 1997) is a well-known iterative optimization method which systematically searches the space of potential solutions via a population of particles. At each iteration, called generation g, three steps, called *mutation*, *recombination*, and *selection*, are performed (Storn and Price 1997). According to the DE method, initially all weight vectors are randomly initialized. Then at the mutation step, new mutant weight vectors v^i_{g+1} are generated by combining weight vectors, randomly chosen from the population. For that purpose some variation operator is used, such as:

$$\mathbf{v}^i_{g+1} = \mathbf{w}^{best}_g + \mu(\mathbf{w}^{r1}_g - \mathbf{w}^{r2}_g), \tag{5}$$

$$\mathbf{v}^i_{g+1} = \mathbf{w}^{r1}_g + \mu(\mathbf{w}^{r2}_g - \mathbf{w}^{r3}_g), \tag{6}$$

$$\mathbf{v}^i_{g+1} = \mathbf{w}^{i}_g + \mu(\mathbf{w}^{best}_g - \mathbf{w}^{t}_g) + \mu(\mathbf{w}^{r1}_g - \mathbf{w}^{r2}_g), \tag{7}$$

$$\mathbf{v}^i_{g+1} = \mathbf{w}^{best}_g + \mu(\mathbf{w}^{r1}_g - \mathbf{w}^{r2}_g) + \mu(\mathbf{w}^{r3}_g - \mathbf{w}^{r4}_g), \tag{8}$$

$$\mathbf{v}^i_{g+1} = \mathbf{w}^{r1}_g + \mu(\mathbf{w}^{r2}_g - \mathbf{w}^{r3}_g) + \mu(\mathbf{w}^{r4}_g - \mathbf{w}^{r5}_g), \tag{9}$$

where $\mathbf{w}^{r1}_g$, $\mathbf{w}^{r2}_g$, $\mathbf{w}^{r3}_g$, $\mathbf{w}^{r4}_g$, and $\mathbf{w}^{r5}_g$ are randomly selected vectors, different from $\mathbf{w}^{i}_g$, $\mathbf{w}^{best}_g$ is the best member of the current generation, and the positive mutation constant μ controls the magnification of the difference between two weight vectors. At the recombination step, each component $j = 1, \ldots, d$ of these new weight vectors is subjected to a further modification. A random number $r \in [0, 1]$ is generated, and if r is smaller than a predefined crossover constant C_{pr}, the jth component of the mutant vector $\mathbf{v}^i_{g+1}$ becomes the jth component of the trial vector. Otherwise, the jth component is obtained from the target vector. Each mutation strategy could be combined with either the *exponential* type crossover or the *binomial* type crossover, resulting in ten different DE strategies. Finally, at the selection step, the trial weight vectors obtained at the crossover step are accepted for the next generation only if they yield a reduction of the value of the error function; otherwise, the previous weights are retained. The training process ends when the target error margin is reached, when the error does not decrease after certain number of iterations, or after completing the predefined maximum number of iterations.

The variation operator sets the strategy for modification of the particles on which depends the speed of convergence and the objective function guides the algorithm towards the global optimum. In some aspects the DE is similar to the GA, however, the population does not consist of binary strings but of real vectors, as well as the mutation step is dynamic which makes DE less vulnerable to genetic drift than GA.

Table 1 Created LUTs for approximating the exponential function

LUTname	V_1	q	V_N	*Size*
LUTe1	0	0.1	4.9	50
LUTe2	0	0.1	13.9	140
LUTe3	0	0.01	13.82	1383

2.3.3 Particle swarm optimization

Particle swarm optimization (PSO) is a population based stochastic optimization method (Kennedy and Eberhart 1995; Clerc and Kennedy 2002; Trelea 2003) which has many different variants. According to (Parsopoulos and Vrahatis 2007):

$$\begin{cases} \mathbf{v}_i^{k+1} = \chi\left(\mathbf{v}_i^k + a_c\mathbf{r}_1(\mathbf{p}_i^k - \mathbf{w}_i^k) + a_s\mathbf{r}_2(\mathbf{p}_{g_i}^k - \mathbf{w}_i^k)\right), \\ \mathbf{w}_i^{k+1} = \mathbf{w}_i^k + \mathbf{v}_i^{k+1}, \end{cases} \tag{10}$$

where the velocity $\mathbf{v}_i^{k+1}$ denotes the amount of change in the position, $\mathbf{w}_i^k$, of a particle (in the multidimensional space) with respect to the difference with its *personal best position* $\mathbf{p}_i$ ever visited, and the difference between the *neighborhoods' best position* $\mathbf{p}_{g_i}$ and its current position; g_i is the index of the particle that attained the best previous position among all the particles in the neighborhood of $\mathbf{w}_i^k$; $\mathbf{r}_1$ and $\boldsymbol{r}_2$ are vectors of random numbers, whose elements $r_{1,i}$ and $r_{2,i}$ receive values within the range $r_{1,i}, r_{2,i} \in [0, 1]$, i=1,2,…,d, where d is the dimensionality; a_c and a_s are the *cognitive* and *social* acceleration factors; and k is the time index, which serves as the iterations' counter. All vector operations are performed element-wise.

Here we consider the algorithm proposed in (Parsopoulos and Vrahatis 2010), as it combines the advantages of the global and local exploration and exploitation strategies in a Unified PSO (UPSO).

$$\begin{aligned} \mathbf{G}_i^{(k+1)} &= \chi\left(\mathbf{v}_i^k + a_c\mathbf{r}_1(\mathbf{p}_i^k - \mathbf{w}_i^k) + a_s\mathbf{r}_2(\mathbf{p}_g^k - \mathbf{w}_i^k)\right), \\ \mathbf{L}_i^{(k+1)} &= \chi\left(\mathbf{v}_i^k + a_c\mathbf{r}_1'(\mathbf{p}_i^k - \mathbf{w}_i^k) + a_s\mathbf{r}_2'(\mathbf{p}_{g_i}^k - \mathbf{w}_i^k)\right) \end{aligned} \tag{11}$$

$$\begin{aligned} \mathbf{U}_i^{(k+1)} &= u\mathbf{G}_i^{(k+1)} + (1-u)\mathbf{L}_i^{(k+1)} \\ \mathbf{w}_i^{k+1} &= \mathbf{w}_i^k + \mathbf{U}_i^{(k+1)} \end{aligned} \tag{12}$$

where $u \in [0,1]$ is called the *unification factor*, $\mathbf{G}_i^{(k+1)}$ is the velocity update of ith particle of the swarm for the global PSO variant, $\mathbf{L}_i^{(k+1)}$ is the velocity update of ith particle of the swarm for the local PSO variant, g is the index of the best particle in the whole swarm, and k is the iteration counter. The typical form of local or global search is obtained by setting the parameter u to 0 or 1, respectively. However when u is selected with value $0 < u < 1$, the PSO algorithm is adjusted to a certain combination of exploration and exploitation behaviors depending on the actual value of u (Parsopoulos and Vrahatis 2007).

In summary, the UPSO-based training of σ_i operates as follows: After (random) initialization of the initial values of all particles, $\mathbf{w}_i$, in the d-dimensional space, their new positions are re-estimated iteratively, by computing the velocity and then updating their current position (12). The particles of the swarm perform search of the d-dimensional space, striving to reach the area where the error function has its global minimum. The search process ends when (i) the target error margin is reached, (ii) there is no error reduction for a given number of iterations, or (iii) after completing a predefined maximum number of iterations. The training ends with selecting the particle, which provides the lowest value of the error function.

2.3.4 Optimization of *sigma*

Based on the DE or UPSO algorithms, we designed fifteen optimization setups for the adjustment of σ_i in a heteroscedastic PNN (Georgiou et al. 2008). Thus, finding the most appropriate setup for the given task for increasing the performance in terms of accuracy. Specifically, we made use of ten strategies of DE, provisionally denoted as DE / m / n / c, where m stands for the method of selecting the vectors for the mutation process {best, rand, rand-to-best}, n is the number of pairs of vectors used to compute the difference vector {1 or 2 pairs}, and c the crossover method {binomial or exponential}. In addition, we tested five UPSO-based setups with different unification factors (*cf.* Table 2). For convenience of presentation, we assigned a provisional index to every setup, shown in round brackets in front of the respective designation.

In the comparative evaluation we experimented with two population sizes: 20 and 40 particles for all training setups, where population size 20 is based on the empirical rule (Parsopoulos and Vrahatis 2005) $N_p = 10d$, with N_p standing for the population size and d for the weight vector dimension. In our application scenario $d = 2$ as we consider a two-class problem with independent adjustment of σ_i for each class. In addition, we consider the case with population size 40, in order to investigate whether larger population is beneficial on the current task.

Two strategies for the resolving of ties, i.e. equal values of the probability computed for the two classes, are considered. In the first one the ties are decided in favor of class 1, while in the second the ties are randomly assigned to one of the two classes.

Table 2 Fifteen setups for adjustment of σ_i are compared in a common experimental protocol

Setup	Setup (cont.)
(1) DE/best/1/exp	(9) DE/best/2/bin
(2) DE/rand/1/exp	(10) DE/rand/2/bin
(3) DE/rand-to-best/1/exp	(11) UPSO u=0
(4) DE/best/2/exp	(12) UPSO u=0.25
(5) DE/rand/2/exp	(13) UPSO u=0.5
(6) DE/best/1/bin	(14) UPSO u=0.75
(7) DE/rand/1/bin	(15) UPSO u=1
(8) DE/rand-to-best/1/bin	

3 Experimental results and discussion

3.1 Evaluation of different approximations of the exponential function

We report the average error in percentages computed after fifty runs for each of the four representations of the exponential function as well as for each setup and for both population sizes 20 and 40 particles (Fig. 1). The experimental results show that the LUTe approximations, in most cases, express performance similar to the PNN with the full-precision exponential function and this is true for both ties resolving strategies. Moreover, for the case of LUTe1 the majority of the results for the total error are advantageous, when compared to the PNN with full-precision exponential function. This is well pronounced for the case with 40 particles (Tables 3, 4). However, the number of resulting ties in the probability for the two classes is much higher when LUTe approximations are used (*cf.* Fig. 2). Even when the ties are disregarded, the PNN implementations with LUTe approximations still have advantageous accuracy, and LUTe1 has still the best results. In the cases of LUTe implementations however, it was noticed that ties are more frequent where the expected result is class 1. This is especially true when the strategy of assigning ties to class 1 is considered. This occurrence also seem data dependent, as for some data splits very low number of ties were observed, while for others almost 1/4 of the results were ties. An unpleasant implication of the higher number of occurring ties is the uncertainty they bring to the final decision. The lowest number of ties was observed for the LUTe3 approximation, where their number is close to the case with the full-precision exponent.

Generally speaking, when the DE-based training strategies are considered, the PNN with LUTe approximations of the exponential function show better classification results then the full precision exp. However, the number of ties occurring when DE-based training is employed are considerably higher when compared to the UPSO-based training. Even for data splits where ties are more frequently observed, the UPSO-based training manages to find more solutions for the two σ_i parameters with only few or no ties than for the case of DE-based training. On the other hand, UPSO achieves better classification accuracy with the full-precision exponential function, however the average number of ties is higher than the one with DE-based training (*cf.* Fig. 2). In terms of computational performance, it was observed that DE tends to need more cost function evaluations than UPSO to reach an optimal σ_i values. However, the number of iterations needed by the UPSO-based strategies seemed longer than the one DE needed to converge.

Summarizing the experimental results, we support that the full-precision exponential function and various approximations with lower computational complexity lead to similar classification accuracy of the PNN. This opens opportunities for FPGA designs with different complexity (and demand of resources), which show similar classification accuracy. In order to obtain a more conclusive assessment of these results we carried out statistical significance test.

3.2 Statistical significance tests

In Fig. 3 we show the averaged error for all strategies for a specific LUT approximation of the exponential function and population size vs. the number of resources in terms of LUT size and population size. Aiming at a low-complexity implementation in FPGA chip, we choose to compare the performance obtained for the full-precision exponential function FPexp and the LUTe1 approximation trained with population size 40.

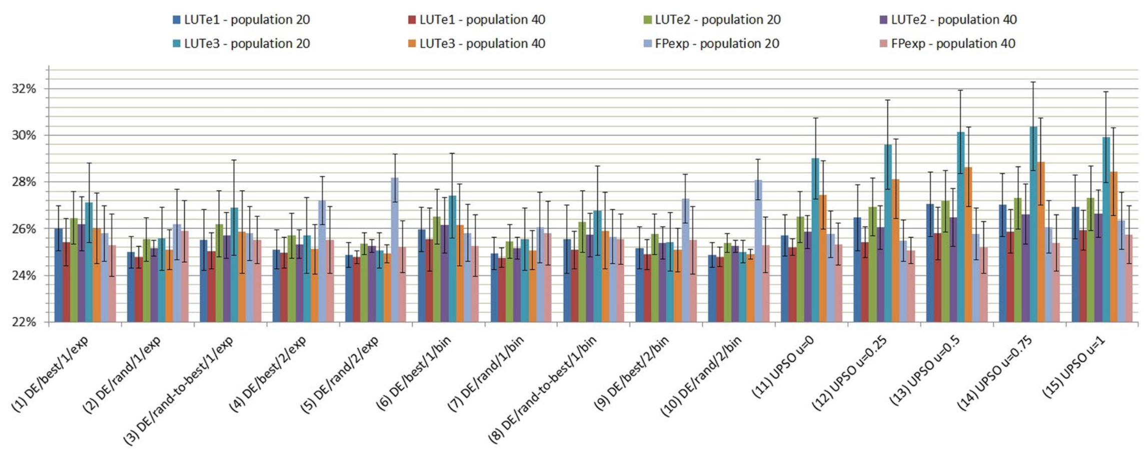


Fig. 1 Average miss-classification error of different approximations of the activation function obtained for fifteen setups for adjustment of σ_i

Table 3 Average miss-classification error of different approximations of the activation function obtained for fifteen setups for adjustment of σ_i from the ten times cross-validation scheme (Part 1)

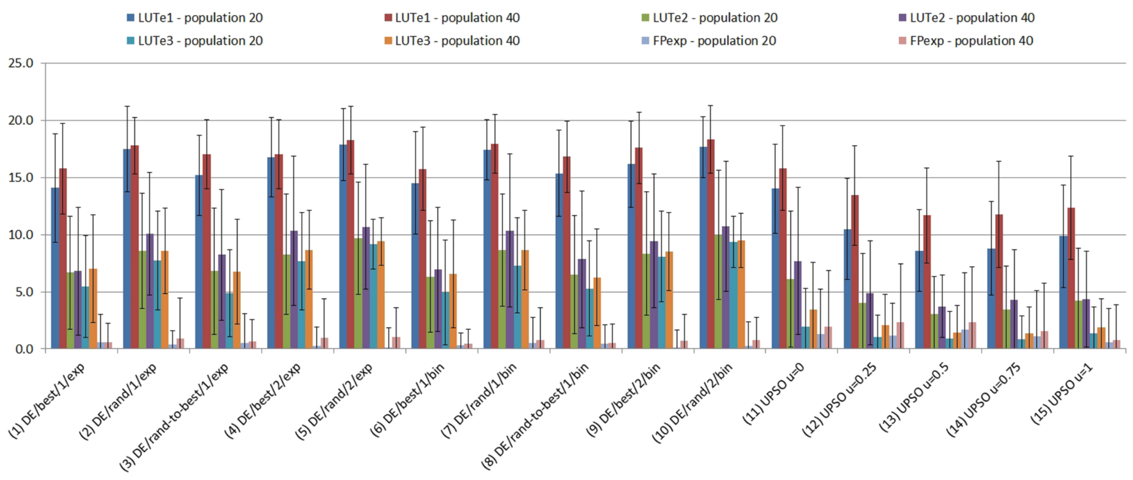
Training setup	LUTe1 population 20 (error in %)	LUTe1 population 40 (error in %)	LUTe2 population 20 (error in %)	LUTe2 population 40 (error in %)
(1) DE/best/1/exp	26.0 ± 4.5	25.4 ± 4.2	26.5 ± 4.2	26.2 ± 4.0
(2) DE/rand/1/exp	25.0 ± 4.0	24.8 ± 3.8	25.5 ± 3.9	25.2 ± 3.7
(3) DE/rand-to-best/1/exp	25.5 ± 4.1	25.0 ± 3.9	26.2 ± 3.9	25.7 ± 3.9
(4) DE/best/2/exp	25.1 ± 4.0	25.0 ± 3.9	25.7 ± 4.0	25.3 ± 3.7
(5) DE/rand/2/exp	24.9 ± 3.8	24.8 ± 3.7	25.4 ± 3.8	25.3 ± 3.7
(6) DE/best/1/bin	26.0 ± 4.5	25.5 ± 4.1	26.5 ± 4.1	26.2 ± 4.0
(7) DE/rand/1/bin	24.9 ± 3.9	24.8 ± 3.8	25.4 ± 3.9	25.2 ± 3.6
(8) DE/rand-to-best/1/bin	25.6 ± 4.1	25.1 ± 4.0	26.3 ± 4.0	25.7 ± 3.9
(9) DE/best/2/bin	25.2 ± 4.1	24.9 ± 3.9	25.8 ± 4.0	25.4 ± 3.8
(10) DE/rand/2/bin	24.9 ± 3.8	24.8 ± 3.8	25.4 ± 3.8	25.3 ± 3.7
(11) UPSO − u = 0	25.7 ± 4.0	25.2 ± 3.9	26.5 ± 3.7	25.9 ± 3.6
(12) UPSO − u = 0.25	26.5 ± 4.0	25.4 ± 4.0	26.9 ± 3.6	26.0 ± 3.7
(13) UPSO − u = 0.5	27.0 ± 4.0	25.8 ± 4.1	27.2 ± 3.5	26.5 ± 3.7
(14) UPSO − u = 0.75	27.0 ± 4.4	25.9 ± 4.3	27.3 ± 3.7	26.6 ± 3.8
(15) UPSO − u = 1	26.9 ± 4.4	25.9 ± 4.3	27.3 ± 3.8	26.6 ± 4.0

Table 4 Average miss-classification error of different approximations of the activation function obtained for fifteen setups for adjustment of σ_i from the ten times cross-validation scheme (Part 2)

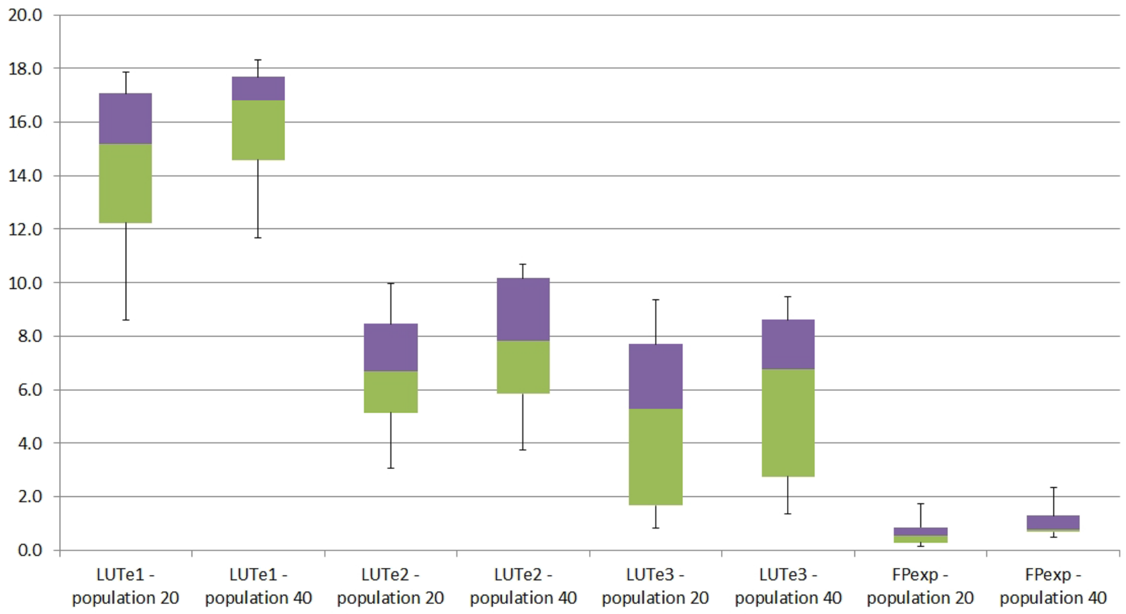
Training setup	LUTe3 population 20 (error in %)	LUTe3 population 40 (error in %)	FPexp population 20 (error in %)	FPexp population 40 (error in %)
(1) DE/best/1/exp	27.1 ± 4.5	26.0 ± 4.1	25.8 ± 2.9	25.3 ± 2.8
(2) DE/rand/1/exp	25.6 ± 3.7	25.1 ± 4.1	26.2 ± 3.3	25.9 ± 2.9
(3) DE/rand-to-best/1/exp	26.9 ± 4.4	25.9 ± 4.0	25.8 ± 2.8	25.5 ± 2.6
(4) DE/best/2/exp	25.7 ± 4.0	25.1 ± 3.4	27.2 ± 3.2	25.5 ± 3.1
(5) DE/rand/2/exp	25.1 ± 3.2	24.9 ± 3.2	28.2 ± 2.9	25.2 ± 2.9
(6) DE/best/1/bin	27.4 ± 4.6	26.2 ± 4.2	25.8 ± 2.9	25.3 ± 2.8
(7) DE/rand/1/bin	25.6 ± 3.6	25.1 ± 3.4	26.1 ± 3.2	25.8 ± 2.9
(8) DE/rand-to-best/1/bin	26.8 ± 4.5	25.9 ± 4.1	25.7 ± 2.7	25.5 ± 2.6
(9) DE/best/2/bin	25.4 ± 3.7	25.1 ± 3.4	27.3 ± 3.3	25.5 ± 3.1
(10) DE/rand/2/bin	25.0 ± 3.2	24.9 ± 3.2	28.1 ± 3.0	25.3 ± 2.9
(11) UPSO – u=0	29.0 ± 3.7	27.4 ± 3.7	25.8 ± 2.8	25.3 ± 2.7
(12) UPSO – u=0.25	29.6 ± 3.7	28.1 ± 4.0	25.5 ± 2.8	25.1 ± 2.7
(13) UPSO – u=0.5	30.1 ± 3.8	28.6 ± 3.9	25.8 ± 3.0	25.2 ± 2.9
(14) UPSO – u=0.75	30.4 ± 4.1	28.9 ± 3.7	26.1 ± 3.1	25.4 ± 3.0
(15) UPSO – u=1	29.9 ± 4.4	28.4 ± 4.1	26.3 ± 3.1	25.7 ± 3.1

Making use of the Kolmogorov-Smirnov test, we rejected the hypothesis that the results are from a standard normal distribution. Therefore, we chose to use nonparametric statistical significance tests—the Friedman test. Given the circumstances, we follow the recommendations in (Derrac et al. 2011) for the comparison of evolutionary and swarm intelligence algorithms.

Specifically, in order to assess the statistical significance of the difference in accuracy, we performed the Friedman statistical test for multiple comparisons ($1 \times N$) with Hochberg post-hoc (Table 5). The comparison was performed between the best performing setup from the LUTe1 approximation in terms of accuracy and all of the full-precision exponential function implementation setups. The results of this test suggested no statistical difference between most of the setups for the full-precision implementation and the LUTe1. Moreover, no statistical difference was found between the best performing LUTe1 setup and the best performing full-precision setup (UPSO u = 0.25). Additional significance tests among different setups and population sizes were carried out as well and the results confirmed these observations. It is worth mentioning that (1) DE/



(a) Average number of ties per training algorithm and population size.



(b) Average number of ties for the different approximations of the exponential function, by training setup and population size.

Fig. 2 Average number of ties for different approximations of the exponential function and population size

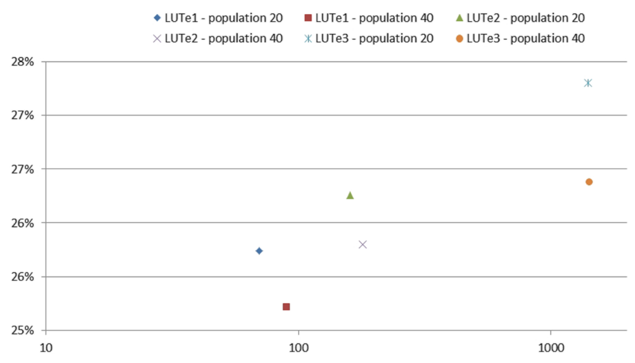


Fig. 3 Resources and error rate for the different exponential approximations (population size is considered as resource as well)

best/1/exp, (6) DE/best/1/bin, (12) UPSO u=0.25 and (13) UPSO u=0.5 are consistent in failing to reject similarity with each other at a significance level $\alpha = 0.05$ and $\alpha = 0.1$. Although, (5) DE/rand/2/exp does not perform well with the full-precision implementation with population size 20, the recognition accuracy obtained after training with population size 40 are comparable to the other best-performing setups. Moreover, as one can expect, significant statistical difference between the two population sizes was found. Overall the Unified PSO (UPSO) showed the best results for unification factor u = 0.25 when full-precision of the exponential function is considered, while DE and more specifically DE/rand/2/exp performs better when reduced precision in terms of LUT approximation is considered. However, between the

Table 5 Adjusted values for the Friedman test between the best LUT approximation – LUTe1 trained with setup (5) – and the best results for the full-precision exponential function, in both cases considering population size 40. Value higher than 0.05 indicates no statistical difference

Comparison	Unadjusted	Hochberg	Comparison	Unadjusted	Hochberg
$(5)_l-(1)_e$	0.6505	0.7714	$(5)_l-(9)_e$	6.3e−6	3.8e−5
$(5)_l-(2)_e$	0.4129	0.6882	$(5)_l-(10)_e$	8.6e−4	0.0032
$(5)_l-(3)_e$	0.4826	0.7240	$(5)_l-(11)_e$	0.1029	0.2204
$(5)_l-(4)_e$	7.7e−6	3.8e−5	$(5)_l-(12)_e$	0.6361	0.7714
$(5)_l-(5)_e$	0.0020	0.0059	$(5)_l-(13)_e$	0.7714	0.7714
$(5)_l-(6)_e$	0.6927	0.7714	$(5)_l-(14)_e$	0.0106	0.0264
$(5)_l-(7)_e$	0.7630	0.7714	$(5)_l-(15)_e$	6.0e−8	9.0e−7
$(5)_l-(8)_e$	0.3534	0.6626			

best performing setups and implementations no statistical difference was found in terms of accuracy.

As for the approximation of the exponential function, the LUTe1 seems the preferred choice. Although, there is a considerable number of ties when LUTe1 is used, its small size, which in turn means less resources needed for implementation and the advantageous classification accuracy are preferred.

4 Conclusion

Analyzing the overall complexity of the studied algorithms for just one-dimensional optimization problem solved by a single particle, it is obvious that the UPSO would require more hardware resources to implement. The combination of a local and a global search schemes will require two multiplications, a summation, and a subtraction. To reach the combination point of the global and local search schemes however, we need to calculate the global and local components. As the algorithm makes use of the constriction factor we have five multiplications, two subtractions, and two summations for each of the two components.

In the same manner, according to Eq. (6), the setup (7) DE/rand/1/bin/ has only one multiplication, subtraction, and summation. In the cross-over we have two more multiplications and a single summation, which means that (7) DE/rand/1/bin will need approximately 4 times less multiplications and ≈3.3 times less summations/subtractions.

Based on these considerations and the results reported in the current study, we can summarize that setup (7) DE/rand/1/bin would be more suitable to implement in a FPGA chip, when compared to the setups UPSO and (5) DE/rand/2/exp. This training strategy achieves one of the best results for the LUTe1 approximation and no statistical difference is found to (5) DE/rand/2/exp – it uses only one vector to compute the difference vector and the crossover method is binomial. This leads to a significantly lower demand of FPGA-resources needed for obtaining good classification accuracy.

Acknowledgements The authors T.D.G. and N.T.D. acknowledge with thanks the support received through the research projects PD5 “Study of biologically substantiated architectures of artificial neural networks for the identification of heart diseases and neurological disorders”, SNP2 “Technological support for improving quality of life of people with the Alzheimer disease”, and the NP4 “Capacity building for object-oriented FPGA design in support of knowledge-based economy” financed by the Technical University of Varna and the Bulgarian National Science Fund. In addition, N.T.D. would like to acknowledge the financial support of the National Science Program “Young Scientists and Postdoctoral Students” financed by the state budget and the Erasmus Mundus Programme of the European Commission. Also, to thank Assoc. Prof. Dimitar Kovachev for the fruitful discussion on certain aspects of FPGA design.

References

Acharya, U. R., Mookiah, M. R. K., Sree, S. V., Yanti, R., Martis, R., Saba, L., et al. (2013). Evolutionary algorithm-based classifier parameter tuning for automatic ovarian cancer tissue characterization and classification. In L. Saba, U. Acharya, S. Guerriero, & J. Suri (Eds.), *Ovarian neoplasm imaging* (pp. 425–440). Boston: Springer. https://doi.org/10.1007/978-1-4614-8633-6_27

Akhmetov, Y., & James, A.P. (2019). Probabilistic neural network with memristive crossbar circuits. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)* (pp. 1–5). https://doi.org/10.1109/ISCAS.2019.8702153

Alshurafa, N., Eastwood, J., Pourhomayoun, M., Liu, J.J., & Sarrafzadeh, M. (2014). Remote health monitoring: Predicting outcome success based on contextual features for cardiovascular disease. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (pp. 1777–1781). https://doi.org/10.1109/EMBC.2014.6943953

Behera, M., Fowler, E. E., Owonikoko, T. K., Land, W. H., Mayfield, W., Chen, Z., et al. (2011). Statistical learning methods as a preprocessing step for survival analysis: Evaluation of concept using lung cancer data. *BioMedical Engineering OnLine*, *10*(1), 97. https://doi.org/10.1186/1475-925X-10-97.

Birk, M., Balzer, M., Ruiter, N., & Becker, J. (2012). Comparison of processing performance and architectural efficiency metrics for FPGAs and GPUs in 3D ultrasound computer tomography. In *2012 International Conference on Reconfigurable Computing and FPGAs* (pp. 1–7). https://doi.org/10.1109/ReConFig.2012.6416735

Boulos, M. N. K., Brewer, A. C., Karimkhani, C., Buller, D. B., & Dellavalle, R. P. (2014). Mobile medical and health apps: State of the art, concerns, regulatory control and certification. *Online Journal of Public Health Informatics*, *5*(3), 229. https://doi.org/10.5210/ojphi.v5i3.4814.

Brandenburg, C., Worrall, L., Rodriguez, A., & Copland, D. (2013). Mobile computing technology and aphasia: An integrated review of accessibility and potential uses. *Aphasiology*, *27*(4), 444–461. https://doi.org/10.1080/02687038.2013.772293.

Chen, C. H., Chang, H. W., & Kuo, C. M. (2019). VLSI implementation of anisotropic probabilistic neural network for real-time image scaling. *Journal of Real-Time Image Processing*, *16*(1), 71–80. https://doi.org/10.1007/s11554-018-0770-3.

Ciarelli, P., Krohling, R., & Oliveira, E. (2009). Particle swarm optimization applied to parameters learning of probabilistic neural networks for classification of economic activities. In *Particle Swarm Optimization, InTech*, chap 19, https://doi.org/10.5772/6756

Clerc, M., & Kennedy, J. (2002). The particle swarm—Explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73. https://doi.org/10.1109/4235.985692.

Derrac, J., Garcia, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18. https://doi.org/10.1016/j.swevo.2011.02.002.

Ford, W., Xiang, K., Land, W., Congdon, R., Li, Y., & Sadik, O. (2013). A multi-class probabilistic neural network for pathogen classification. *Procedia Computer Science*, *20*, 348–353. https://doi.org/10.1016/j.procs.2013.09.284.

Ganchev, T. (2009). Enhanced training for the locally recurrent probabilistic neural networks. *International Journal of Artificial Intelligence Tools*, *18*(6), 853–881. https://doi.org/10.1142/S0218213009000433.

Georgiou, V. L., Pavlidis, N. G., Parsopoulos, K. E., Alevizos, P. D., & Vrahatis, M. N. (2006). New self-adaptive probabilistic neural networks in bioinformatics and medical tasks. *International Journal on Artificial Intelligence Tools*, *15*(3), 371–396. https://doi.org/10.1142/S0218213006002722.

Georgiou, V. L., Alevizos, P. D., & Vrahatis, M. N. (2008). Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. *Neural Processing Letters*, *27*(2), 153–162. https://doi.org/10.1007/s11063-007-9066-5.

Hsieh, S., & Chen, C. (2009). Adaptive image interpolation using probabilistic neural network. *Expert Systems with Applications*, *36*(3), 6025–6029. https://doi.org/10.1016/j.eswa.2008.06.124.

Huang, D., & Du, J. (2008). A constructive hybrid structure optimization methodology for radial basis probabilistic neural networks. *IEEE Transactions on Neural Networks*, *19*(12), 2099–2115. https://doi.org/10.1109/TNN.2008.2004370.

Hussain, H.M., Benkrid, K., Erdogan, A.T., & Seker, H. (2011). Highly parameterized k-means clustering on FPGAs: Comparative results with GPPs and GPUs. In *Proceedings of the 2011 International Conference on Reconfigurable Computing and FPGAs, RECONFIG '11* pp. 475–480. https://doi.org/10.1109/ReConFig.2011.49

Zhang, Jianhai, Chen, Ming, Hu, Sanqing, Cao, Yu, & Kozma, R. (2016). Pnn for eeg-based emotion recognition. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 002319–002323). https://doi.org/10.1109/SMC.2016.7844584

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95—International Conference on Neural Networks* (pp. 1942–1948). https://doi.org/10.1109/ICNN.1995.488968

Krestinskaya, O., & James, A.P. (2018). Approximate probabilistic neural networks with gated threshold logic. In *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)* (pp. 1–4). https://doi.org/10.1109/NANO.2018.8626302

Kusy, M., & Zajdel, R. (2015). Application of reinforcement learning algorithms for the adaptive computation of the smoothing parameter for probabilistic neural network. *IEEE Transactions on Neural Networks and Learning Systems*, *26*(9), 2163–2175. https://doi.org/10.1109/TNNLS.2014.2376703.

Lavanyadevi, R., Machakowsalya, M., Nivethitha, J., & Kumar, A.N. (2017). Brain tumor classification and segmentation in mri images using pnn. In *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)* (pp. 1–6). https://doi.org/10.1109/ICEICE.2017.8191888

Li, L., & Ma, G. (2008). Optimizing the performance of probabilistic neural networks using PSO in the task of traffic sign recognition. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence* (pp. 90–98) https://doi.org/10.1007/978-3-540-85984-0_12

Mao, K. Z., Tan, K., & Ser, W. (2000). Probabilistic neural-network structure determination for pattern classification. *IEEE Transactions on Neural Networks*, *11*(4), 1009–1016. https://doi.org/10.1109/72.857781.

Mathew, N.A., Vivek, R.S., & Anurenjan, P.R. (2018). Early diagnosis of alzheimer's disease from mri images using pnn. In *2018 International CET Conference on Control, Communication, and Computing (IC4)* (pp. 161–164) https://doi.org/10.1109/CETIC4.2018.8530910

Mittal, S., & Vetter, J. S. (2014). A survey of methods for analyzing and improving GPU energy efficiency. *ACM Computing Surveys (CSUR)*, *47*(2), 19:1–19:23. https://doi.org/10.1145/2636342.

Moser, L.E., & Melliar-Smith, P.M. (2015). Personal health monitoring using a smartphone. In *2015 IEEE International Conference on Mobile Services* (pp. 344–351) https://doi.org/10.1109/MobServ.2015.54

Palo, H. K., Chandra, M., & Mohanty, M. N. (2018). *Recognition of human speech emotion using variants of Mel-frequency cepstral coefficients* (pp. 491–498). Singapore: Springer. https://doi.org/10.1007/978-981-10-4762-6_47.

Parisi, L., RaviChandran, N., & Manaog, M. L. (2018). Feature-driven machine learning to improve early diagnosis of Parkinson's disease. *Expert Systems with Applications*, *110*, 182–190. https://doi.org/10.1016/j.eswa.2018.06.003.

Parsopoulos, K., & Vrahatis, M. (2002). Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, *1*(2), 235–306. https://doi.org/10.1023/A:1016568309421.

Parsopoulos, K., & Vrahatis, M. (2005). Unified particle swarm optimization for tackling operations research problems. In *Swarm intelligence symposium, 2005 SIS 2005 Proceedings 2005 IEEE* (pp. 53–59) https://doi.org/10.1109/SIS.2005.1501602

Parsopoulos, K., & Vrahatis, M. (2007). Parameter selection and adaptation in unified particle swarm optimization. *Mathematical and Computer Modelling*, *46*(1–2), 198–213. https://doi.org/10.1016/j.mcm.2006.12.019.

Parsopoulos, K., & Vrahatis, M. (2010). *Particle swarm optimization and intelligence: Advances and applications*. Hershey, PA: Information Science Publishing (IGI Global). https://doi.org/10.4018/978-1-61520-666-7.

Plagianakos, V. P., & Vrahatis, M. N. (2002). Parallel evolutionary training algorithms for "hardware-friendly" neural networks. *Natural Computing*, *1*(2), 307–322. https://doi.org/10.1023/A:1016545907026.

Sakar, B. E., Isenkul, M., Sakar, C., Sertbas, A., Gurgen, F., Delil, S., et al. (2013). Collection and analysis of a Parkinson speech dataset with multiple types of sound recordings. *IEEE Journal of Biomedical and Health Informatics*, *17*(4), 828–834. https://doi.org/10.1109/JBHI.2013.2245674.

Samanta, B., Al-Balushi, K. R., & Al-Araimi, S. A. (2006). Artificial neural networks and genetic algorithm for bearing fault detection. *Soft Computing*, *10*(3), 264–271. https://doi.org/10.1007/s00500-005-0481-0.

Specht, D. (1990). Probabilistic neural networks. *Neural Networks*, *3*, 109–118. https://doi.org/10.1016/0893-6080(90)90049-Q.

Storn, R., & Price, K. (1997). Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*(4), 341–359. https://doi.org/10.1023/A:1008202821328.

Trelea, I. C. (2003). The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information*

Processing Letters, *85*(6), 317–325. https://doi.org/10.1016/S0020-0190(02)00447-7.

Ullah, I., Hussain, M., ul Haq, Qazi E., & Aboalsamh, H. (2018). An automated system for epilepsy detection using EEG brain signals based on deep learning approach. *Expert Systems with Applications*, *107*, 61–71. https://doi.org/10.1016/j.eswa.2018.04.021.

Van Essen, B.C., Macaraeg, C.C., Prenger, R., & Gokhale, M. (2012). Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA? In *International Symposium on Field-Programmable Custom Computing Machines (FCCM'12). IEEE* pp. (232–239). https://doi.org/10.1109/FCCM.2012.47

Vipin, K., Akhmetov, Y., Myrzakhme, S., & James, A.P. (2018). Fapnn: An fpga based approximate probabilistic neural network library. In *2018 International Conference on Computing and Network Communications (CoCoNet)* (pp. 64–68). https://doi.org/10.1109/CoCoNet.2018.8476889

Wang, D., Hao, Y., Zhu, X., Zhao, T., Wang, Y., Chen, Y., Chen, W., & Zheng, X. (2011). FPGA implementation of hardware processing modules as coprocessors in brain-machine interfaces. In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (pp. 4613–4616). https://doi.org/10.1109/IEMBS.2011.6091142

Chtioui, Y., Marsh, R. A., & Panigrahi, S. (1998). Conjugate gradient and approximate Newton methods for an optimal probablilistic neural network for food color classification. *Optical Engineering*, *37*(11), 3015–3024. https://doi.org/10.1117/1.601972.

Zhou, F., Liu, J., Yu, Y., Tian, X., Liu, H., Hao, Y., et al. (2010). Field-programmable gate array implementation of a probabilistic neural network for motor cortical decoding in rats. *Journal of Neuroscience Methods*, *185*(2), 299–306. https://doi.org/10.1016/j.jneumeth.2009.10.001.

Zhou, J., Zhong, T., & He, X. (2017). Auxiliary diagnosis of breast tumor based on pnn classifier optimized by pca and pso algorithm. In *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)* (Vol. 2, pp. 222–227). https://doi.org/10.1109/IHMSC.2017.164