2021 IEEE Congress on Evolutionary Computation (CEC) | 978-1-7281-8393-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/CEC45853.2021.9504984

Comparative Performances of Neural Networks of Variant Architectures Trained with Backpropagation and Differential Evolution

Zakaria Oussalem
Department of Computer Science
Ecole Centrale de Lille
Villeneuve d'Ascq, France
zakaria.oussalem@centrale.centralelille.fr

Rochan Avlur Venkat
Department of Computer Science
Mahindra University
Hyderabad, India
rochan170543@mechyd.ac.in

Jahnavi Malagavalli
Department of Computer Science
Mahindra University
Hyderabad, India
jahnavi160521@mechyd.ac.in

Arya Kumar Bhattacharya
Department of Computer Science
Mahindra University
Hyderabad, India
arya.bhattacharya@mechyd.ac.in

***Abstract*—Work on Neuro-evolution has of late tended to focus on the design of alternative architectures and hyper-parameters for different classes of Neural Networks to improve performance and computational efficiency, using Evolutionary Algorithms (EA). Predominantly, the underlying mechanism for learning weight parameters remains traditional backpropagation (BP), now considered a paradigm of Deep Learning (DL). Many important facets of DL, like hierarchical construction of features across layers in image recognition, vanishing gradients, etc., are taken for granted without recognizing that these may implicitly be induced by the properties of BP itself. EAs that perform global optimization in contrast to local gradient descent of BP, if used extensively for ANN training, can potentially disrupt these assumed facets of DL – and construct alternative and interesting perspectives. There is a surprising lack of research activity towards one-to-one performance comparison between EA and BP, keeping precisely the same architectures, activation functions and datasets, for complex regression and classification problems. This work partially fills this gap. It is demonstrated that Differential Evolution enhanced with local search, can generate mildly better accuracies on regression problems on noisy industrial datasets compared to standard BP solutions, and comparable accuracies on popular image classification datasets. Consequently, this establishes the baseline for revisiting the above-mentioned assumed facets of DL, and potentially engender new and interesting perspectives.**

Index Terms*—Neural Network architectures, optimizing weight parameters, backpropagation, Differential Evolution, local crossover and mutation, regression and classification, neuro-evolution.

I. Introduction

Machine Learning techniques that come under the ambit of Artificial Neural Networks (ANNs) and its multiple variants have seen remarkable expansion in research and applications over the past decade, enabled at the basics by efficiencies attained by gradient-based algorithms for training of model parameters from data. Indeed, the Backpropagation algorithm [1] introduced back in the 1990s remains the standard training workhorse for ANNs. All major advances, interpretations and explanations in Deep Learning implicitly assume backpropagation (BP) as the baseline mechanism, to the extent that it may be looked upon as a paradigm of this domain.

However, framing the architecture of an ANN variant and selection of related hyper-parameters remains a dominantly manual exercise with scope for heuristic but novel experimentation. Gradient-based optimization has very limited role to play here. Evolutionary optimization Algorithms (EAs) like Genetic Algorithms [2] and Differential Evolution (DE) [3] and similar nature-inspired algorithms like Particle Swarm Optimization (PSO) [4] work on a population of candidate solutions that search the global space concurrently, juxtaposing exploration with exploitation. Importantly, these can operate in optimization scenarios that are mathematically intractable or not easily amenable to mathematical representation, like the architecture of ANN variants, impact of hyperparameters, etc. Hence, EAs have seen significant application in optimization of these aspects of ANNs, and this field is known as Neuro-evolution [5].

Evolutionary Algorithms, apart from their role in the design of ANN architectures and hyper-parameters, can also be applied to the optimization of the model training parameters, namely the weights. However, this role is considered secondary or insignificant compared to the mainstream backpropagation approach, and reference to Neuro-evolution almost implicitly assumes only design of architectures and hyper-parameters using EAs. Yet, there are very significant reasons why optimization of weights by EAs can profoundly impact the fields of ANNs, EAs as well as that of High Performance Computing (HPC).

When applied to ANNs for the optimization of the weight parameters (i.e. training), these methods traverse over the search space without making any assumptions about the underlying architectural landscape. This is in contrast to gradient descent optimization which essentially searches locally; when applied to ANN training this translates into propagating the error backwards from the last (output) layer all the way to the input layer modulating the weights along the way.

In the context of Image Processing applications primarily enabled by Convolutional Neural Networks (CNNs), it is now accepted that the early layers extract the low level features of images and the latter layers start recognizing higher level characteristics and finally the object [6-8]. It is pertinent to try to cognize that to what extent this hierarchical progression of object recognition is induced by the backward propagation of training. Will its replacement by a global optimization technique based on EAs change the hierarchical recognition pattern? Can this lead to more accurate recognition, or with higher efficiency reflected in reduced numbers of layers or in alternative architectures?

Further, it is known [9], [10] that the identification of objects in images are strongly sensitive to values in specific pixels, and variations in such pixels lead to altered recognitions. To what extent is this amplified sensitivity dependent on gradient based training – and will global training methods like EAs result in more consistent identifications? What will be the corresponding impact on Explainability [11]? Also, vanishing and exploding gradients are well known problems related to BP; how will EA-based training impact on the gamut of issues associated with this problem? For example, LSTMs [12] and GRUs [13] have been designed to minimize and obviate this problem in the training of Recurrent Neural Networks by BP.

To investigate all these above aspects using EAs, it is pertinent to first develop trained ANN-variant models using EAs that provide solutions of comparable accuracies to BP trained models, *for precisely the same architectures, activation functions and training data*. This should be investigated for both regression and classification type problems. This becomes crucial because if EAs are proved unable to generate solutions of comparable accuracies as BP for identical ANNs, then the entire *raison d'etre* of the above investigations become somewhat untenable.

The thrust of the developments reported here is to generate solutions on a series of architectures and classification and regression problems using both EAs and BP, and study how they compare. Both classical and CNN architectures are considered. *Consequently, it is demonstrated that EAs with specific algorithmic features can be synthesized that engender solutions that are as accurate as from BP.*

One may question that considering the high impact value of the above exploratory propositions, what prevented analysts and researchers from delving into them earlier? The answer is twofold. First, the fact that any EA-based solution is expected to take approximately *N x BP-baseline* amount of computation in the training phase, where 'N' is the number of candidates in the population, and *BP-baseline* denotes the computation time of the converged BP solution. Even with BP, deep architectures of advanced ANN variants take large computation times. A reasonable number of candidates like 60-200 will amplify that by two orders of magnitude, which is computationally humongous. The present authors have developed [14] novel hybrid CPU-GPU computational paradigms, exploiting the natural parallelism of EA algorithms, that reduce computing times for EA-based solutions by approximately 400X, which consequently bring the proposed studies into the realm of feasibility. One may just note that when a trained ANN enters into production phase, it doesn't matter how the weights were originally learnt, the computing times will be exactly same.

Second, the trend in Neuro-evolution has always been towards designing architectures and hyperparameters that improve accuracy and time of solution; one-on-one comparison with BP for identical architectures and hyperparameters has hardly been the subject of interest. Perhaps an implicit assumption prevailed that BP will always emerge better? The current study dispels that viewpoint, clearing the way for analysis along the above propositions.

The EA implemented in this work is a specific version of Differential Evolution [15] that is developed by the authors and found to work better than other EAs [16]. An earlier study conducted by Vesterstrom and Thomsen indicated that DEs tend to outperform PSO [17]. The Global DE as above is enhanced with Local Search features on the lines of Zhang and Xu [18] and used in this study for one-on-one comparison with BP for variant ANN architectures and problems.

The rest of the paper is organized as follows. Section II refers to existing work and helps to place the current development in context. Section III outlines the current DE formulation and explains the fusion with Local Search. Section IV presents different architectures and corresponding comparisons between BP and EA solutions. Section V discusses Conclusions and further work.

II. RELATED WORK

The idea of using EAs for optimization of neural networks has been explored in the past and is commonly described as Neuro-evolution [5]. In recent years, EAs have been steadily gaining momentum as computationally feasible methods for: (1) automated optimization of ANN architecture [19-21] and (2) training of ANNs, the latter of which is explored in this paper.

The encouraging results on using EAs for training of ANNs in the 1990s and early 2000's have inspired a handful of researchers in recent times to explore this exciting intersection [22-24]. For simplicity and relevance to this work, we broadly classify the literature on using EAs for training ANNs and its variants into two categories. On one side, we have methods that use Differential Evolution (DE). In the second group, we have all the other variants such as Particle Swarm Optimization (PSO), etc. Algorithms from both groups have been explored for training ANNs. For instance, in [25], PSO is used to successfully train CNNs that achieve 95% accuracy on the MNIST dataset. Others have proposed their own encoding strategy and modified PSO for multi-objective optimization to perform image classification tasks [26]. In [22], a gradient-free method is proposed to evolve the weights of CNNs by using simple Genetic Algorithms (GA), with a population of chromosomes of fixed length for Reinforcement Learning.

Likewise, attempts at using DE for training ANNs is well documented and evaluated on popular benchmarking datasets. In [27], classical DE is used for training ANNs that achieve 88% accuracy on the MNIST dataset. Building on this, the authors of [27] proposed a new self-adaptive version of DE called *MAB-ShaDE* that improved their previous implementation, achieving 89.4 - 90.4% accuracy on MNIST [28]. The above attempts, however, unroll the image into a long array and use a simple linear feedforward ANN architecture. The use of DE for training CNNs has been explored in [29] by using a hybrid approach for the task of image classification. In this work, as discussed in Sec. 1, we have used a modified version of DE that is enhanced with Local Search.

III. ALGORITHMS AND VARIANTS

As discussed above, we are using variants of Differential Evolution (DE) for Global Search and combining it with Local Search. The first sub-section presents the Global DE that is used here. The second presents the Local Search methodology.

A. Baseline Global Differential Evolution

Formally, if the dimensionality of the solution space is denoted as D and the number of candidate solutions (i.e. population size) is N, then the elements of the i^{th} candidate of the solution $X_{i,G}$ at generation G may be denoted as

$$X_{i,G} = (x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, ..., x_{D,i,G}) \tag{1}$$

for all $i = 1, \ldots, N$.

The DE process fundamentally generates new solutions from the current candidate set by adding the weighted difference between two randomly selected candidate solution vectors to a third to generate a “mutant” vector, and then creating a crossover between an existing vector and the “mutant” that is called the “trial” vector. The latter is allowed to replace the existing vector only if it is found to be more “fit” – the complexity of this “fitness determination” exercise depending entirely upon the nature of the problem under consideration.

If $V_{i,G}$ represents the mutant vector, then according to the baseline DE process called DE/rand/1 [3]

$$V_{i,G} = X_{r1,G} + F \times (X_{r2,G} - X_{r3,G}) \tag{2}$$

where r_1, r_2 and r_3 are random integers less than N, different from each other and from i, and F usually lies between 0.5 and 1. There are many variations of this baseline process where two instead of one difference terms are sometimes considered, the best solution in a population is incorporated, etc.; descriptions of alternative schemes may be seen in [3, 30], among others.

Crossover is performed between the ‘mutant’ vector $V_{i,G}$ and the target vector $X_{i,G}$ to generate a ‘trial’ vector $Z_{i,G}$ according to

$$z_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if rand}_j(0,1) \le Cr \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{3}$$

where $z_{j,i,G}$ is the element j of the trial vector $Z_{i,G}$, rand$_j$ (0, 1) denotes a random number between 0 & 1 applied to the element j, Cr is the crossover threshold usually set between 0.4 and 1. At the final selection step the choice for candidate i in the next generation is made between $Z_{i,G}$ and $X_{i,G}$ on the basis of higher fitness by direct one-to-one comparison.

In the present work the mutant vector is generated according to the alternate scheme (proposed in [3] and also used by current authors in [31, 32] where it is found to work better than other DE variants)

$$V_{i,G} = X_{r1,G} + R \times (X_{best,G} - X_{r1,G}) + F \times (X_{r2,G} - X_{r3,G}) \tag{4}$$

where R is set at 0.5 and F varies randomly between -2 and +2 across generations (and are same for all i within a generation). The crossover probability Cr in eq. (3) is set at 0.9.

B. Modifications for Local Search

The Local Search variant proposed by Zhang and Xi [18] is modified, particularly aspects related to mutation, and then combined with our Global Search, eqs. (3-4), to generate the working optimization algorithm used extensively in this work.

The local-search procedure at any generation G works sequentially on Local Crossover and Local Mutation. The Crossover step sorts all the candidates X_i in descending order (for a minimization problem) based on their individual objective values $f(X_i)$. Thus, for two solutions X_i and X_j, where $i, j = 1, \ldots, N$, X_i is placed ahead of X_j if

$$f(X_i) \ge f(X_j) \tag{5}$$

Next, this ordered sequence of candidates is traversed and an intermediate temporary candidate is generated between the current candidate and its next-in-sequence, by simply interpolating, in each dimension, between the two. This is represented as

$$X_{i,temp} = X_i + r(i) \times (X_{i+1} - X_i) \tag{6}$$

where $r(i)$ is a random number between 0 and 1.

If the intermediate candidate has higher fitness than the better one in the original pair, then it replaces X_i. Thus

$$iff:\ f(X_{i,temp}) < f(X_{i+1}),\ f(X_i) \leftarrow f(X_{i,temp}) \tag{7}$$

After completion of the traversal over all pairs, the candidates are again arranged in descending order for the mutation step.

For mutation, first the lower and upper bounds of each of the j dimensions among all candidates X_i are identified and denoted as L_j and U_j. A candidate selected for mutation is shifted from its position in the D-dimensioned space to a certain extent either towards the U or L points in that space. The probability of mutation is varied across candidates and generations based on two considerations. First, the “worse-fitness” candidates, i.e. those with small values of i after sorting, are mutated more than the better fitness ones, because one would like to “explore” with bad-fitness candidates more than the good ones. This is reflected in the value of pm shown in the algorithm in fig. 1.

Second, the probability of mutation is varied cyclically across generations across a certain user-defined hyper-parameter GC shown in fig. 1, for the calculation of variable q (defined in the fig.). This probability is high at the beginning of the cycle and falls to near zero towards the end. The shifted candidate’s fitness value is compared with its pre-shift value and the candidate is placed at the new position if its fitness is found to be better. This is expressed algorithmically below:

Algorithm for Local Mutation for Minimization Problem, at any generation G

```
$L_j \leftarrow \min[X_{j,1}, X_{j,2}, ..., X_{j,N}], \quad \forall j \in \{1,...,D\}$
$U_j \leftarrow \max[X_{j,1}, X_{j,2}, ..., X_{j,N}], \quad \forall j \in \{1,...,D\}$
GC ← Range for cyclical variation of mutation parameters
for $i$ = 1, ..., N do
    $pm = (N-i+1)/N$    # probability of mutation
    $q = \exp(-2\times G/GC)\times sig((GC/2)-G)$
    # sig(.) denotes the sigmoid function
    for $j$ = 1, ..., D do
        $r, b \leftarrow$ rand(0, 1), randint(0, 1)
        if $b == 0$ then
            $X_{i,j}^{temp} = X_{i,j} + (U_j - X_{i,j}).r.q.pm$
            # '.' denotes scalar multiplication
        else
            $X_{i,j}^{temp} = X_{i,j} + (X_{i,j} - L_j).r.q.pm$
    end for
    if $f(X_i^{temp}) \le f(X_i)$ then
        $X_i \leftarrow X_i^{temp}$
end for
```

Fig. 1. Algorithm for Mutation under Local Search, at a generation G. The range for cyclical variation of mutation parameters considered in this work is 50.

IV. ACCURACY COMPARISONS OF BP AND DE SOLUTIONS

This section presents comparisons between solutions obtained using Back-Propagation (BP) and Differential Evolution enhanced with Local Search (DE), performed in accordance with discussions in Sec. 1. This implies comparisons across identical architectures, activation functions and training data, but learning mechanisms either BP or DE. First we present results for two regression datasets, and then for the MNIST classification dataset.

A. *Regression Case 1 – Prediction from Noisy Industrial Data*

The first regression data set relates to prediction of the Yield Strength of steel strips rolled through a Skin Pass Mill (SPM) [33] in a steel manufacturing facility. The SPM is the final stage of the steel manufacturing process chain where a small elongation is imparted to a thin steel strip in the rolling process that changes its yield strength. Each data sample (i.e. one strip) consists of 13 input variables and 1 output. The inputs consist of various parameters accrued over the entire chain with emphasis towards those in the final cold rolling stages, the output is of course the Yield Strength. One of the inputs is the percentage elongation imparted in the SPM. A total of 8000 such samples are accumulated over a period of five years of operation. All data, except that of the percentage elongation which is a control set point (this is the 8th input variable), are accumulated from sensors and needless to say, there is significant measurement error manifesting as noise.

An ANN is trained to predict the Yield Strength after splitting the data 80 : 20 for training : validation. The ANN architecture is shown in fig. 2. The total number of parameters is 3301 including bias. This ANN is trained independently by BP and by DE. The same data samples are apportioned into training & validation subsets in both cases.

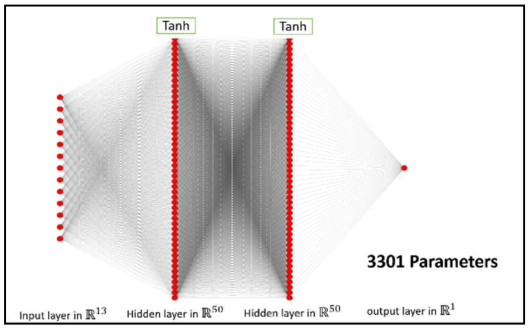

Tanh
Tanh
3301 Parameters
Input layer in $\mathbb{R}^{13}$
Hidden layer in $\mathbb{R}^{50}$
Hidden layer in $\mathbb{R}^{50}$
output layer in $\mathbb{R}^{1}$

Fig. 2. Architecture of ANN for Regression problems 1 and 2.

For the BP training process, we used a minibatch size of 32, Adam optimization with default training parameters used in the *Pytorch* library, namely learning rate parameter of 0.001, beta1 and beta2 of 0.9 and 0.999, and epsilon 1.0e-7.

For the DE training process, we used a population size of 200, crossover probability of 0.9, R-value in eq. (4) as 0.5, F-value in the same equation varying randomly in the range [-2, 2] across all candidates in every generation. Moreover, we introduced upper and lower weight bounds as [-30, 30], i.e. outlying values of the training parameters were clamped to their respective bounds. The cyclicity value *GC* shown in fig. 1 is set at 50.

Fig. 3 compares the R-squared value, as it evolves over epochs (for BP) or generations (for DE). This value is obtained by comparing the predicted value against actual value for the validation samples, so that if all the predicted sample values sit exactly on the regression line (here linear) extracted from the actual values, an R-squared of 1 will be attained, and as the scatter of predicted values increase from the regression line, R-squared will proportionately decrease. It is seen that the best R-squared obtained is about 0.84 from the DE solution, and it is 0.007 better than the best obtained from the BP solution, i.e. *DE provides slightly higher accuracy for this regression problem*.

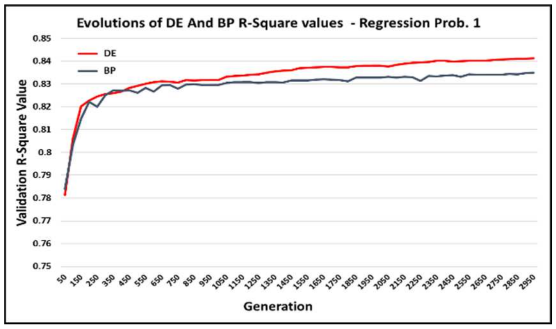


Fig. 3. Increase in accuracy expressed as R^2 values, across generations, for DE and BP trainings on Regression problem 1.

Figs. 4 & 5 compare the evolution of R-square values from BP and DE respectively, between validation and training datasets. It is seen that validation data has consistently higher accuracy than training, which is counterintuitive and is a characteristic of noise in data spoiling the overall accuracy. Fig. 6 compares the predicted Yield strength with the actual measured values, from BP (with DE very similar). This illustrates and explains the R-squared values discussed; actual values are on the x-axis and predicted values on the y-axis.

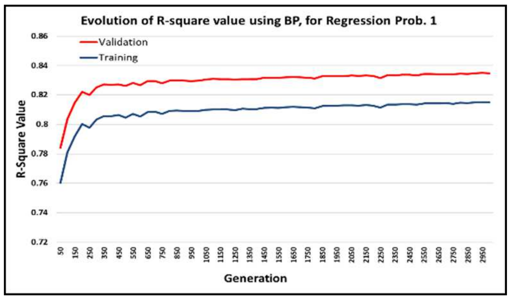


Fig. 4. R^2 evolution of BP-trained Regression Problem 1, training and validation datasets.

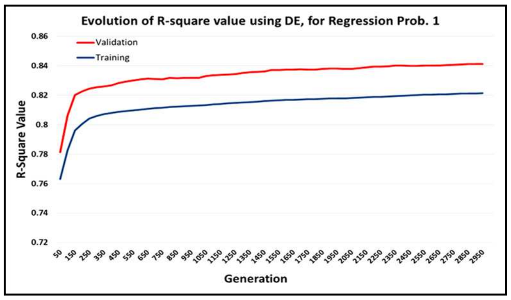


Fig. 5. R^2 evolution of DE-trained Regression Problem 1, training and validation datasets.

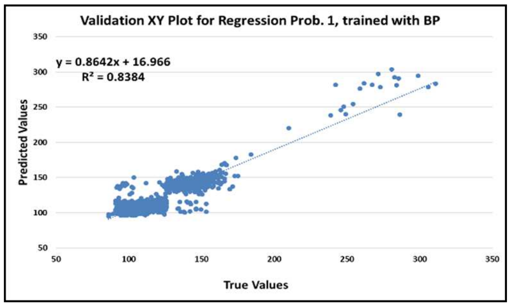


Fig. 6. Final Predicted vs. True values of output for Regression Problem 1 trained with BP, reflecting final R^2 value. Very similar plot for DE (not shown), with R^2 value at 0.8426.

B. *Regression Case 2 – Control of Noisy Industrial Process*

Here the data is exactly the same as that used in Case 1, with the difference that the Yield Strength prediction problem is transformed into the SPM elongation set point control problem. This is created by simply interchanging the 8th input variable (8th column in the dataset) with the output variable, so that instead of predicting the Yield Strength for a given elongation set point (all other variables remaining exactly same for every sample), we are trying to compute what the elongation set point should be if a certain Yield Strength is to be achieved, i.e. controlling the elongation (as ANN output) to attain a specific Yield Strength for a strip, (entered as an ANN input) when all other parameters (i.e. inputs) are known.

An important point to note here is that we have used exactly the same architecture as in Case 1, and exactly the same hyper-parameters for both BP and DE optimizations. Fig. 7 compares the evolution of R^2 from both BP and DE validation datasets; it can be seen that, the accuracy (R^2) attained is 0.97 for DE and about 0.004 less for BP *implying DE generates higher accuracy in this case too*.

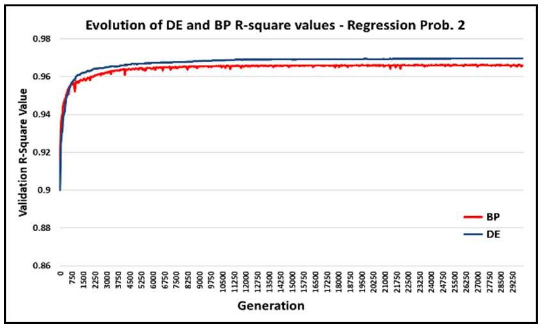


Fig. 7. Increase in accuracy expressed as R^2 values, across generations, for DE and BP trainings on Regression problem 2.

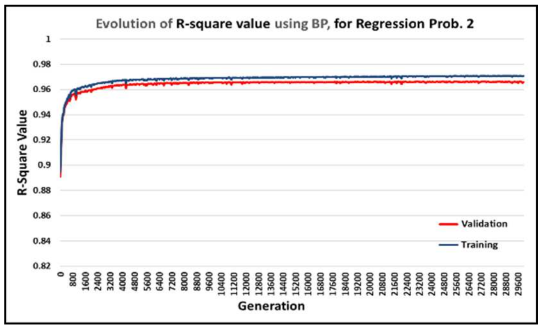


Fig. 8. R^2 evolution of BP-trained Regression Problem 2, training and validation datasets.

One may question why a much higher R-squared is obtained in this Case as compared to the previous, this is addressed shortly below. Figs. 8 and 9 compare the evolution of accuracies from training and validation datasets for both BP and DE trainings, here it is seen to follow the intuitive rule that training should be more accurate. Fig. 10 illustrates the actual R-squared by plotting computed versus actual values. An important observation is that the output actual values are at specific points on the x-axis, this is because this - the elongation percentage - is a value set according to certain rules (of the domain) at specific points and is not varying arbitrarily across the entire range, and hence unlike the previous case where the output was the Yield Strength and measured with a noisy error, here the error or noise is zero. Because the ANN output is noise-free, it follows that the overall accuracy is much higher than in Case 1.

Interestingly, though the DE-trained cases have shown mildly superior performance for these single-output regression cases, when the number of outputs increase to 2 and above, DE performance deteriorates. This is because the global DE process tries to modulate all weights together for a single cost function composed of 2 or more error components, while the local BP process modulates weights to satisfy each output node's error locally. There are two potential responses to this observation, one, that when dealing with regression problems with say m outputs, one can create in parallel m number of

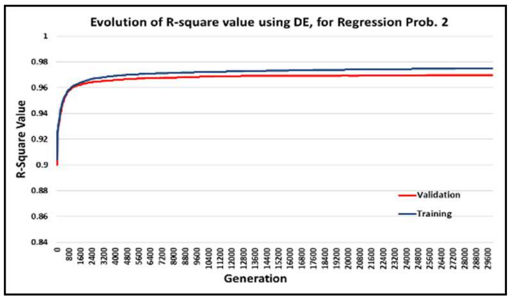


Fig. 9. R^2 evolution of DE-trained Regression Problem 2, training and validation datasets.

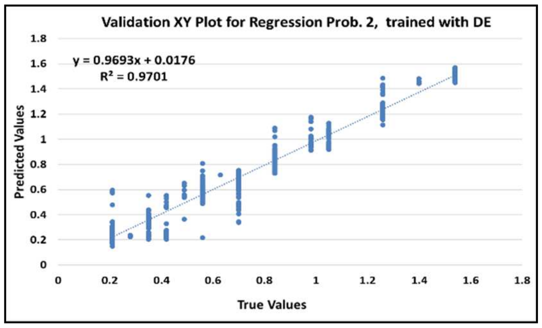


Fig. 10. Final Predicted vs. True values of output for Regression Problem 2 trained with DE, reflecting final R^2 value. Very similar plot for BP (not shown), with R^2 value at 0.9662.

ANNs each with one output node, trained concurrently and independently. Second, one may investigate multi-objective DE with as many objectives as number of output nodes (m needs to be very small), and then trade off at a point on the converged Pareto surface which is equidistant from all corners of the surface.

It may be noted that in the regression cases we have used only a classical fully-connected ANN; in the classification problem that follows we have used both classical ANN and CNN.

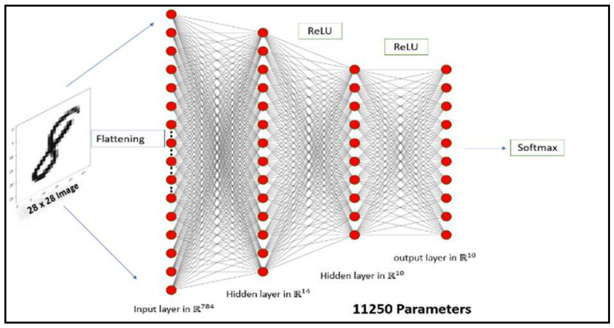
C. Comparing Classification Performances on the MNIST dataset

The standard MNIST dataset is used to train first an ANN, and then a CNN, using both BP and DE approaches, and the final accuracies obtained on the validation subset are compared for the two optimization methods.

The hyperparameters for the BP training on the ANN are same as the two regression cases with the following two differences – minibatch size is 64 and optimization with Adagrad in place of Adam. For Adagrad the initial accumulation value is 0.1, learning rate is 0.001 and epsilon is 1.0e-07. For CNN the minibatch size is changed to 16, all else remains same.

The hyperparameters for DE training of the ANN remain exactly same as in the regression cases. For the CNN, however, better results are obtained upon dispensing with the fitness-dependent probability of mutation factor *pm* (see fig. 1), i.e. *pm* is always taken as 1.

Fig. 11 illustrates the architecture and activation functions used for the ANN (recall this is same for the two optimization approaches). The total number of parmeters is 11250. Fig. 12 shows the variation of accuracy across epochs for the training and validation datasets under BP training. It can be seen that the highest accuracy of 95.4% for the validation set is achieved in less than 1000 epochs. Fig. 13 shows the corresponding evolution of accuracies under DE training. It may be observed that the validation accuracies reach their highest level only at around 15,000 generations, although they have almost plateuaed at about 10,000 generations. The highest value is 94.8%. Thus, for this classification case with fully-connected ANN architecture, not only does BP training produce mildly higher accuracy than DE, importantly, the solution converges much faster for BP learning. Next we move on to CNN architectures.



ReLU
ReLU
Flattening
28 x 28 image
Softmax
output layer in $\mathbb{R}^{10}$
Hidden layer in $\mathbb{R}^{10}$
Hidden layer in $\mathbb{R}^{14}$
Input layer in $\mathbb{R}^{784}$
11250 Parameters

Fig. 11. Architecture of ANN for MNIST Classification problem.

Fig. 14 shows the architecture used for the CNN. It is a rather simple design, avoiding any pooling and zero padding, and most importantly, *completely avoiding the fully-connected pre-final layer*, which is otherwise known to boost accuracy levels but amplify the number of parameters. This helps in keeping the number of parameters low at just 6740. Fig. 15 shows the accuracies obtained with BP training, with a highest validation accuracy of 98.1% obtained in 500 epochs, though the value had plateaued at less that half that number of epochs. Fig. 16 shows corresponding accuracies obtained with DE training. Highest value of validation accuracy of 97.4% was reached after 20,000 generations, though this value was actually obtained at around 10000 generations and plateued thereafter. This again demonstrates that *for classification, BP gives mildly higher accuracies that are reached much faster.*

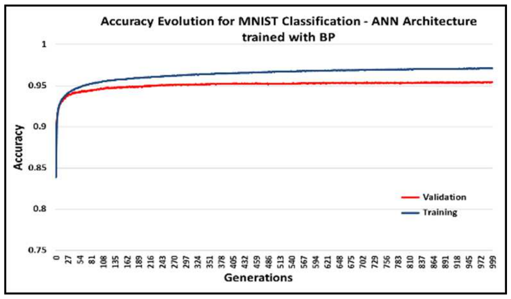


Fig. 12. Accuracy evolution of BP-trained MNIST Classification problem on ANN architecture, training and validation datasets.

A number of observations follow as consequence. First, that

CNNs – even with almost half the number of parameters, generate about 2.5% higher accuracy than ANNs for image classification problems, independent of the approach to training, i.e. BP or DE. This is understandable as local-

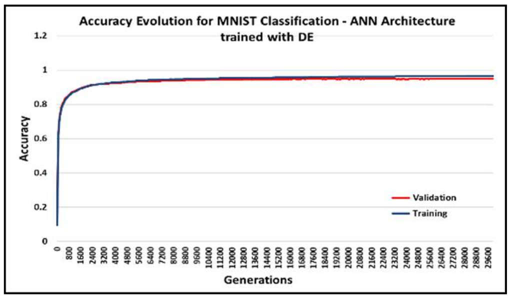


Fig. 13. Accuracy evolution of DE-trained MNIST Classification problem on ANN architecture, training and validation datasets.

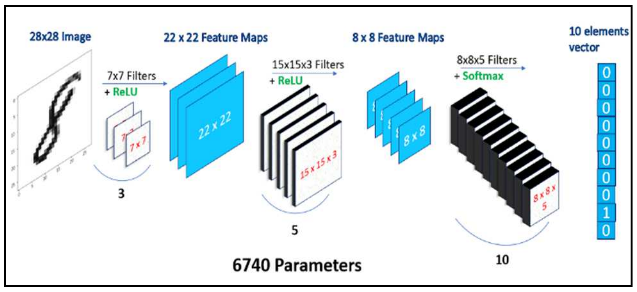


Fig. 14. Architecture of CNN for MNIST Classification problem.

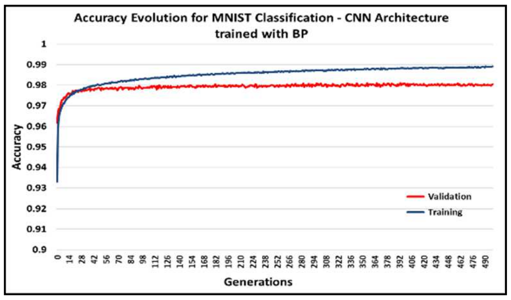


Fig. 15. Accuracy evolution of BP-trained MNIST Classification problem on CNN architecture, training and validation datasets.

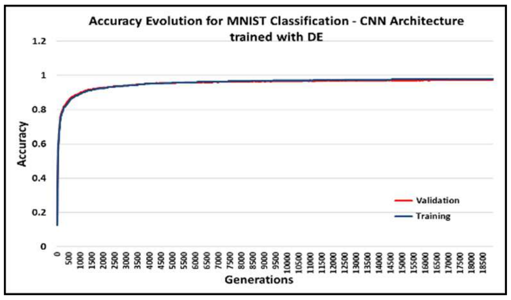


Fig. 16. Accuracy evolution of DE-trained MNIST Classification problem on CNN architecture, training and validation datasets.

neighbourhood-relationships characteristic of images are properly mapped into the philosophy of CNN design. Second, that the 97.4% accuracy attained on our CNN using DE training, *is the best seen by the authors as yet among EA-trained neural networks*, irrespective of architecture.

Investigations are made on alternative CNN architectures which differ essentially in the size of the filters in the last two layers, and hence the total number of parameters. The "light" architecture is the one illustrated in fig. 14, with the important architecture parameters tabulated in Table 1. Tables 2 and 3 provide information on the alternative architectures, the "standard" one with 9180 parameters and the "heavy" with 21210. Input and output sizes are identical for all architectures.

Table 1. Architecture data of "Lighter" CNN with 6740 parameters, this is also illustrated in fig. 14.

Layer Num.	Num. of Filters (channels) = Num. of Feature Maps	Filter size (always square)	Num. of external zero paddings
1	3	7	0
2	5	15	0
3	10	8	0

Table 2. Architecture data of "Standard" CNN with 9180 parameters

Layer Num.	Num. of Filters (channels) = Num. of Feature Maps	Filter size (always square)	Num. of external zero paddings
1	3	7	0
2	5	11	0
3	10	12	0

Table 3. Architecture data of "Heavier" CNN with 21210 parameters

Layer Num.	Num. of Filters (channels) = Num. of Feature Maps	Filter size (always square)	Num. of external zero paddings
1	3	7	0
2	5	11	3
3	10	18	0

Fig. 17 shows accuracy variation for the BP case, it is seen here that the highest validation accuracy of 99% is attained by the "heavy" architecture at just about 20 epochs, and as the number of parameters reduce, the accuracy also falls and is attained with increasing number of epochs.

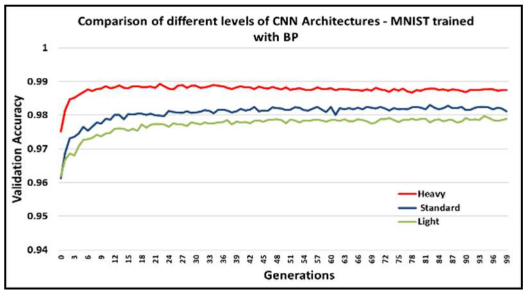


Fig. 17. Accuracy variation across epochs for different architectures of CNN applied to MNIST classification, training with BP.

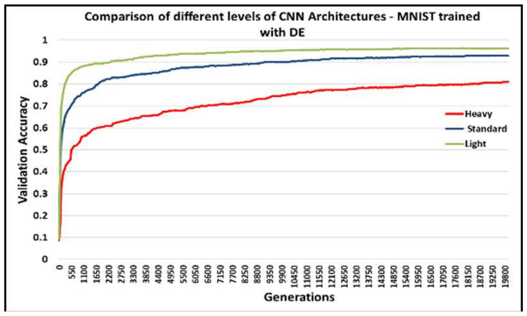


Fig. 18. Accuracy variation across epochs for different architectures of CNN applied to MNIST classification, training with DE.

Remarkably, quite the opposite is observed for DE trained solutions shown in fig. 18. Here the highest accuracy is obtained for the lighter architecture, and as the number of parameters increase, the accuracy actually falls but then convergence is not reached even after 20,000 generations. So it may be argued that if still lighter architectures could have been investigated, even higher accuracies could be obtained at lower number of generations.

V. CONCLUSIONS AND FURTHER WORK

The objective of this work is to compare Evolutionary Algorithms, represented by Differential Evolution enhanced with local search, against Back-propagation, as competing mechanisms for training of weights of different Neural architectures and analyze their performances. The architectures, activation functions, and datasets on which the two mechanisms are applied need to be identical for meaningful comparison, and both regression and classification problems need consideration. The profounder objective is to take EA-training to a stage where many of the facets commonly assumed implicity as characteristics of Deep Learning, but conjectured here to be lateral manifestations of BP-training, can be investigated with alternative perspectives. These objectives have been succesfully realized and it is shown that DE compares favourably for regression and comparably for classification problems.

The obvious extension is to pursue the deeper objective and investigate the different facets discussed in Sec. 1. Another direction of work is to combine with Neuro-Evolution and use Multi-Objective Optimization for concurrently improving both architectures & hyper-parameters, and weights, with the twin objectives of minimizing computing times and maximizing accuracies. Observations discussed on fig. 18 motivate work in that direction.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors”, Nature, vol. 323, no. 6088, 1986, pp. 533–536.

[2] J. H. Holland et al., *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press, 1992.

[3] R. Storn and K. Price, “Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces”, Journal of Global Optimization, vol. 11, no. 4, 1997, pp. 341–359.

[4] J. Kennedy and R. Eberhart, “Particle Swarm Optimization”, in Proceedings of ICNN’95-International Conference on Neural Networks, vol. 4. IEEE, 1995, pp. 1942–1948.

[5] Edgar Galván and Peter Mooney, "Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges", *arXiv:2006.05415*, 2020.

[6] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks”, in European Conference on Computer Vision. Springer, 2014, pp. 818–833.

[7] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization”, arXiv preprint arXiv:1506.06579, 2015.

[8] A. Nguyen, J. Yosinski, and J. Clune, “Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks”, arXiv preprint arXiv:1602.03616, 2016.

[9] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps”, arXiv preprint arXiv:1312.6034, 2013.

[10] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Muller, “Evaluating the visualization of what a deep neural network has learned”, IEEE Trans. on Neural Networks and Learning Systems, vol. 28, no. 11, 2016, pp. 2660–2673.

[11] G. Vilone and L. Longo, “Explainable artificial intelligence: a systematic review”, arXiv preprint arXiv:2006.00093, 2020.

[12] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, Neural Computation, vol. 9, no. 8, 1997, pp. 1735–1780.

[13] K. Cho et al, “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, arXiv:1406.1078, 2014.

[14] Rochan Avlur, Zakaria Oussalem and Arya K. Bhattacharya, “Training Convolutional Neural Networks with Differential Evolution using Concurrent Task Apportioning on Hybrid CPU-GPU Architectures”, IEEE Congress on Evolutionary Computation (CEC), 2021.

[15] R. Storn and K. Price, “DE - a simple and efficient adaptive scheme for global optimization over continuous spaces”, Technical Report, vol. 25, no. 6, 1995, pp. 95–102.

[16] A. K. Bhattacharya and D. Sambasivam, “Optimization of oscillation parameters in continuous casting process of steel manufacturing: Genetic Algorithms versus Differential Evolution”. Vienna, Austria: InTech, 2009, vol. 572.

[17] J. Vesterstrom and R. Thomsen, “A comparative study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on numerical benchmark problems”, in Proceedings of the 2004 Congress on Evolutionary Computation, vol. 2. IEEE, 2004, pp. 1980–1987.

[18] J. Zhang and J. Xu, “A new Differential Evolution for discontinuous optimization problems”, in Third International Conference on Natural Computation, ICNC 2007, vol. 3. IEEE, 2007, pp. 483–487.

[19] Thomas Elsken, Jan Hendrik Metzen and Frank Hutter, "Efficient multi-objective Neural Architecture search via Lamarckian evolution", arXiv:1804.09081, 2018.

[20] Liu Hanxiao et al. "Hierarchical Representations for Efficient Architecture Search", International Conference on Learning Representations, 2018.

[21] Risto Miikkulainen et al, Evolving Deep Neural Networks, Ch. 15 in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, Academic Press, 2019. 293-312.

[22] Felipe Petroski Such et al, "Deep Neuroevolution: Genetic Algorithms are a competitive alternative for training Deep Neural Networks for Reinforcement Learning", arXiv:1712.06567, 2017.

[23] K. Pawełczyk, K. Michal and N. Jakub, "Genetically-trained deep neural networks", Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2018.

[24] Chrisantha Fernando et al, "Convolution by Evolution: Differentiable pattern producing networks", Proceedings of the Genetic and Evolutionary Computation Conference, 2016.

[25] Arie Rachmad Syulistyo et al, "Particle Swarm Optimization (PSO) for training optimization on Convolutional Neural Network (CNN)", Jurnal Ilmu Komputer dan Informasi vol. 9, no. 1, 2016, pp. 52-58.

[26] Bin Wang et al, "Evolving deep neural networks by multi-objective Particle Swarm Optimization for image classification", Proceedings of the Genetic and Evolutionary Computation Conference, 2019.

[27] M. Baioletti et al, *Differential evolution for learning large neural networks*. Tech. Rep., available at https://github.com/Gabriele91/DENN-RESULTS-2018, 2018.

[28] M. Baioletti, G. D. Bari, A. Milani and V. Poggioni, “Differential Evolution for Neural Networks optimization”, Mathematics, vol. 8, no. 1, 2020, p. 69, 2020.

[29] Bin Wang et al, "A hybrid Differential Evolution approach to designing deep Convolutional Neural Networks for image classification", Australasian Joint Conference on Artificial Intelligence, Springer, Cham, 2018.

[30] S. Das and P. N. Suganthan, “Differential Evolution: A survey of the State-of-the-Art”, IEEE Transactions on Evolutionary Computation, Vol. 15, No. 1, Feb 2011, pp. 4-31.

[31] A. K. Bhattacharya, D. Aditya and D. Sambasivam, “Estimation of operating Blast Furnace reactor invisible interior surface using Differential Evolution”, *Applied Soft Computing*, Vol. 13, May 2013, pp. 2767-2789.

[32] S.R. Gautam, M. Jahnavi, P. Thangeda and A.K. Bhattacharya, “Synthesis of optimal trajectories in tactical aerial engagements using Multi-Objective Evolutionary Algorithms”, *Advances in Multidisciplinary Analysis and Optimization, Lecture Notes in Mechanical Engineering*, Springer Singapore, 2021 (In Press).

[33] http://www.sms-siemag.com/download/W6_7_304E_Skin-Pass_Mills_References.pdf , last accessed April 18, 2021.