Design Philosophy for Optimizing Genetic Algorithms Through Embedded Intelligence

Lorick Jain, Akash Basabhat, Srikanth HR
PES University
lorick.jain@gmail.com, akash.basabhat@gmail.com, srikanthhr@pes.edu

***Abstract*—Traditionally Genetic algorithms are thought of as brute force approaches, aimed to arrive at solutions to problems which do not have a specific answer. In problems where the data is not structured for the general implementation of a specific idea, genetic algorithms are most useful. This paper proposes to mitigate the above problem of brute force approaches through elucidation of procedures ranging from exploratory analysis, followed by pattern analysis and classification. This novel conceptualization of the scheme and design will help in arriving at solutions through reduced iterations. Research conducted involves dropping of poorly performing hypotheses, controlled mutation, thereby adding a dimension of intelligence to evolutionary algorithms. The following paper describes the methodology used to solve the problem of addition of numbers using evolutionary algorithms of Neural Networks, whilst building intelligence into the system. The specific problem of addition has been dealt with in the following paper, however the same design philosophy can be extended for a paraphernalia of problems. The end goal is to obtain a generation of adroit and capable hypotheses to solve the problem in reduced number of iterations. The solution provided is generic and can be reused, it has been applied to a specific problem in the following paper.**

I. Introduction

Genetic algorithms are considered to be brute force algorithms that involve the application of concepts originating from biology(crossover and mutation)[1] to solve problems and arrive at good solutions. These algorithms are strongly influenced by the process of natural selection depending on processes such as selection, crossover, and mutation. They were developed by John Holland at the University of Michigan. GAs are more suitable for data where analytics is not easy and modeling is difficult. GAs are also suitable for parallelization and when the search space is large.

A lot of parallels can be drawn between genetic algorithms and reproduction[2](the origin of the idea of evolutionary algorithms). There are two ways in which reproduction takes place, mitosis and meiosis. In the mitosis process of reproduction, the cell multiplies into two, that is, a cell copies itself into duplicates. In meiosis the cell generates four different cells through crossover mechanisms, the idea being adapted into GAs. During copying, there may be small changes to the chromosome. This is called gene mutation. In each passing generation, the chromosomes are selected based on their fitness values and undergo mutation and crossover to produce two chromosomes for the new generation.

In GA terminology the generation or population is a collection of chromosomes which are basically hypotheses or models that are candidate solutions. The constraint is that each hypothesis must be of the same type, that is, a population can consist only of neural networks or SVMs or any other model for that matter. The hypotheses or chromosomes consist of genes, which are analogous to the hyperparameters of the model. All the possible values that these hyperparameters can take are analogous to alleles of the genes. The fitness function in the following implementation is a manual observation of accuracy scores of the candidate solutions. The following process of mimicking natural selection gives rise to good as well as bad hypotheses. These bad hypotheses lead to an increase in the number of iterations required to solve a problem. This paper aims to find these bad hypotheses and eliminate them, their progeny to prevent increased iterations. The same idea can be extrapolated to various problems using a different set of models/hypotheses that constitute the population of the GAs. The neural networks or hypotheses are trying to predict the 4^{th} column in Fig1 taking the first 3 columns as input to the model. The file is divided into train, validation and test and fed to each of the hypotheses, their accuracy scores are monitored whilst collecting their representations. The hyperparameters shown in Fig2 constitute the representation of Neural Networks(hypotheses) that belong to the population of the GA in that particular iteration along with their accuracy scores. These are fed as data points to classifiers to distinguish between good and bad hypotheses keeping the final accuracy column as the target based on the threshold which is explained in further sections.

A	B	C	D
5	4	3	12
6	2	1	9
1	1	6	8
5	3	9	17
9	3	5	17
7	5	1	13
4	9	6	19
9	1	8	18
7	2	9	18
1	5	3	9
2	8	5	15
7	8	5	20
5	4	6	15
3	3	8	14
8	1	1	10

Fig. 1. Dataset for hypotheses(NNs) trying to predict 4th column. The first three columns(A, B and C) are the inputs to the neural network whose sum is equal to the last column(D)

actual_layers	layers	neurons	optimizers	learning rate	accuracy
4	2	121	Adadelta	1	0.64516129
5	3	150	RMSprop	0.001	0.75576037
3	1	72	RMSprop	0.001	0.29032258
5	3	135	Adadelta	1	0.55299539
6	4	425	RMSprop	0.001	0.19354839
12	10	403	adam	0.001	0.20276498
7	5	163	Adagrad	0.01	0.91705069
12	10	1013	adam	0.001	0.37788018
4	2	209	adam	0.001	0.51152074

Fig. 2. Hyperparameters of hypotheses that represent NN representations

Genetic algorithms take an iterative approach to arrive at a solution as shown in Fig 3. The first step is to initialize the population, the set of chromosomes, randomly or using heuristics. In the next step, fitness values are calculated for each chromosome. Based on these fitness values, chromosomes are pushed to the future generations or they are dropped off. These chromosomes later undergo crossover to produce new offspring chromosomes. Hence, giving rise to a new population of the same size as the initial population. The process is repeated until a termination condition is reached.

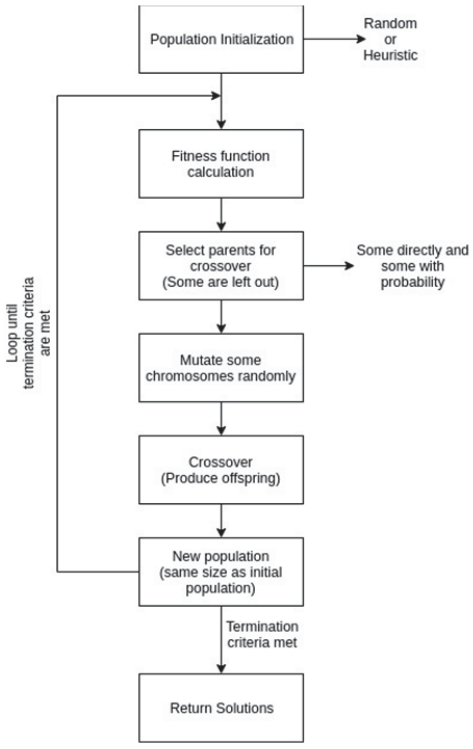

Fig. 3. Iterative process of GAs

Normally, through execution of GAs of Neural Networks to solve the problem of addition of numbers, fitness scores are generated. The good hypotheses(models that have high accuracy for trying to predict the actual sum) are sent to the next generation and the poorly(models with lower accuracy, the threshold is set at 70%) performing hypotheses are dropped off. To regenerate a population of a similar number of chromosomes, new hypotheses are included by crossover. From this pool of models, randomly chosen hypotheses are mutated(hyperparameters are tweaked) and the next generation resumes execution. This process continues until the number of generations is reached or a generation of adroit hypotheses are found(convergence). The novelty of this article is discussed in the implementation phase.

In order to solve the addition of numbers problem, the authors have taken a population of 100 neural networks. In this scenario, each neural network is a chromosome, each hyperparameter of the neural network is a gene and the value of a hyperparameter is an allele. During each iteration, few of the best, models go to the next generation and the bad ones are dropped. The remaining chromosomes are pushed to the next generation using a roulette wheel selector. The chromosomes then go through controlled mutation and regeneration to keep the size of each generation constant. This process is repeated for the population for 10 iterations.

II. Literature Survey

Genetic algorithms have been used to solve or find approximate solutions to problems for a variety of problems such as the Travelling Salesman Problem, Sudoku and other NP-hard problems [6]. The aim of this paper is to reduce the number of iterations taken by GAs to arrive at a solution. The authors of [7] and [8] tried to solve the same problem by introducing an extended migration operator and an immigration operator respectively. These papers try to solve the problem in a specific fashion and a general extendable approach is not used. However, these papers were the initial stepping stones to solve this problem. The solution discussed in this paper extends the above two papers for a broader range of use cases by finding a pattern in the hypotheses generated so that the hypotheses converge faster and generate the solution. The immigration operator, along with other operators is used to see difference between actual GAs and Immigration Gas [17], however there is no concrete robust solution that was obtained. The number of generations also did not show any reduction, however the quality of the hypotheses obtained were noteworthy. [9] describes a method to solve the problem of brain connectivity using GAs by. This problem can be solved better by using the techniques the authors have described in the following paper as model brain connectivity is a critical problem to solve and includes a lot of data. This problem becomes harder to solve using ordinary GAs. Methodology proposed in [10] is a new crossover operator to improve the performance of GAs to solve the Travelling Tournament problem. This paper is aiming to provide a specific solution to optimize GAs only for the travelling tournament problem. However, the approach used in the following paper ushered in the ideation of the novel scheme used in this paper. Alcohol addiction classification is discussed in [11]. The accuracy of the classifier can be increased by identifying a pattern and sifting the good hypotheses from the bad hypotheses. The problem can be solved in a less computationally intensive manner. [12] uses GAs in a way to tune the hyperparameters of the ELM(Extreme learning machine). This poses a problem as it

will be suitable to solve the problem pertaining to the ELM generalizing the same, will be hard as this requires a lot of manual intervention. Optimization is an important task as it saves cost and time. The research in [13] talks about the need for optimization and how it can be done with regard to GA. This paper stresses the need for pattern analysis in order to achieve better optimization. Route optimization is solved as a brute force using regular GAs, the same can be improved by adding intelligence to the existing system. [14] proposes a portfolio optimization algorithm based on genetic algorithms and affinity propagation. Pattern analysis can help in solving the problem described in [14] in a better fashion. The techniques described in this paper can mitigate the problems that may arise in [14]. The specifics of [18] deal with another interesting problem encountered with GAs and that is of premature convergence, interesting results were obtained regarding the same, which laid the foundation of the research carried out in the following paper. The immigration operator though deals with premature convergence, it randomly introduces new members to the GA's successive population, this is a random heuristic solution, while the methodology posed in this paper aims to learn patterns and not rely on randomization to solve the problem. This may not eliminate premature convergence as randomization is not intelligence in the truest form, it is simply relying on the similar brute force strategy to solve the problem at hand [17]. The ideas discussed to use mechanisms to learn global and local patterns also ushered to further research. However the movement of hypotheses to subsequent populations used random selection using the selection operator [19].

III. Implementation

The myriad phases explained in this section highlights the implementation methodology for solving the problem of addition of numbers. The solution is depicted as a pool of neural networks(hypotheses), (to be precise, one thousand neural networks, in batches of one hundred neural networks for each generation) used to solve the problem using the concept of evolutionary algorithms. It involves pushing the capable hypotheses to the next generation/iteration and discarding the ones that are poor performers. Prior to performing the above-mentioned function, an exploratory analysis was conducted to test if the following research could actually be feasible by noticing possible patterns in the good and bad hypotheses respectively. The good hypotheses are ones that perform well and have a good accuracy score(setting the threshold at 70%) on the dataset of addition of numbers(checking if the sum of numbers in the columns equals the target column). The bad hypotheses are ones that perform poorly on the same dataset(in the same iteration) and have an accuracy score lower than 70%. After the exploratory analysis step, pattern analysis is done, and finally classification is done to distinguish between good and bad hypotheses, therefore help in dropping the bad hypotheses. To account for the dropped off hypotheses, controlled mutation is done and new offsprings are generated to maintain the length of each generation at 100 neural networks in the next iteration.

Overview of Techniques Used:

t-SNE: t-distributed stochastic neighbor embedding is a machine learning algorithm developed by Laurens van der Maaten and Geoffrey Hinton for the purpose of visualization[3]. The algorithm is a nonlinear dimensionality reduction technique that works well for embedding high dimensional data into a low dimensional space of two or three dimensions. It models each high dimensional data point in such a way that similar data points are modeled by nearby data points and dissimilar data points are modeled by distant data points with high probability. In this paper, the authors have used t-SNE to visualize the hyperparameters of the neural networks used in each generation to verify the clusters formed by them. The clusters are visualized in a two-dimensional space. Apart from this, PCA(Principal Component Analysis) was also used to visualize the clusters. t-SNE and PCA help in pursuing exploratory analysis that is required to understand the patterns and visualize the clusters formed by the hypotheses. The mathematics behind t-SNE is as explained below:

t-SNE computes probabilities p_{ij} for the various data points in a high dimensional space. These probabilities are in proportion to data points x_i and x_j. 'N' is the number of data points in the high dimensional space. The probabilities are calculated as follows:

$$p_{j|i} = \frac{\exp(-\frac{|x_i - x_j|^2}{2\sigma_i^2})}{\sum_{k \neq i} \exp(-\frac{|x_i - x_k|^2}{2\sigma_i^2})} \quad (1)$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \quad (2)$$

Classifiers to find good and bad representations of hyperparameters(Classify hypotheses):

1) Support vector machines(SVM) [4]: SVM is a supervised learning model that analyzes data used for classification and regression analysis. An SVM model is a representation of data points in space mapped in such a way that the data points of separate categories are divided by a gap that is as wide as possible. New data points are predicted to belong to a category based on the side of the gap they fall under.

2) Deep neural networks(DNN)[5]: A DNN as in [5] is an artificial neural network(ANN) with multiple hidden layers. DNNs can model complex non-linear relationships. The extra layers enable composition of features from lower layers, which makes modeling complex data with fewer units probable than a similarly performing shallow network.

B. Exploratory Analysis

The aim of the research involves a way to find a pattern among the poor performing hypotheses so they can be dropped off if they are regenerated by noticing the pattern that occurs in either of the hypotheses(good or bad). The method used to test this ideology involved visualizing the hypotheses in a two-dimensional space and noticing if the hypotheses form distinct clusters for the good and bad chromosomes. As explained in earlier sections t-SNE aided in the visualization of these

clusters by squashing multiple features into a two-dimensional space, and as noticed the good and bad hypotheses formed different clusters. The other methods to visualize the same would be PCA or SVD, but t-SNE is preferred as it involves lossless compression, as the original data can be re-obtained also it performs a better job than PCA as seen in the figures below. The figures show the formation of clusters in the range (0-30%, 30-50%,50-70% and above 70%). The figures also show the formation of clusters for 2500 and 5000+ iterations in different steps. T-SNE uses the concept of distance based on the probability of the data points being similar to each other in higher dimensions and mapping this similarity as a probability in lower dimensions. This t-SNE visualization was implemented using TensorBoard[15]. The hyperparameters for the TensorBoard implementation involved, a learning rate of 0.01, a perplexity of 5, and the number of iterations that were required for the algorithm to converge was 5000. The common hyperparameters were- activation unit(ReLU), and mean squared error loss. The hyperparameters generated by 300(more neural networks can be used) neural networks(through normal execution of GAs), consisting of various features like- number of actual layers of the neural network, number of hidden layers of the neural network, number of neurons of the neural network, the learning rate of the neural network, and the optimizer used for the neural network, were squashed by t-SNE to a two and three dimensional space and visualized, the clusters obtained are as shown in the figures below, form different clusters which are proof that they must follow different patterns. If it is possible to learn such a pattern, there will be no regeneration of the poor performers, by discarding the poor hypotheses, learning the pattern specific to their cluster. The clusters show how far away the hyperparameters of good and bad hypotheses are in space giving an idea that the patterns that these two follow might vary to a considerable extent. Below are figures showing the implementations of PCA and t-SNE performing the squashing in two dimensions and their respective clusters achieved.

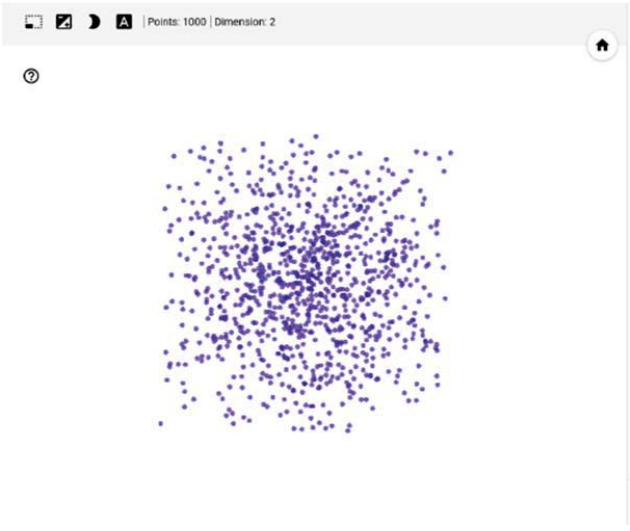

Fig. 4. PCA Visualizing the clusters formed by accuracy scores. The Total Variance described is 100% and PCA is approximate

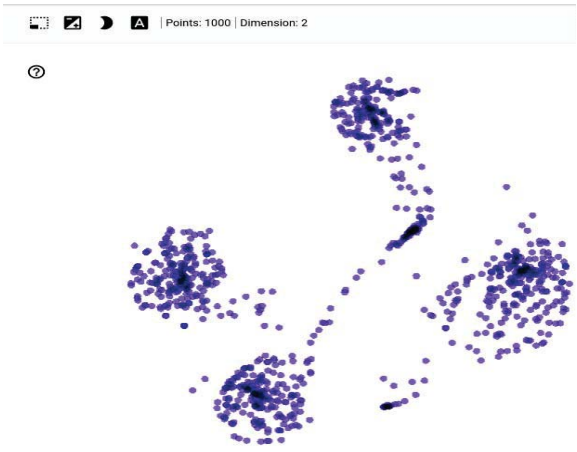

Fig. 5. t-SNE for 2500 iterations in 2D space forming clusters. The parameters used for computing t-SNE are: Perplexity was 5, Learning rate was 0.01 and Supervise was 0. t-SNE was paused at iteration number 2507

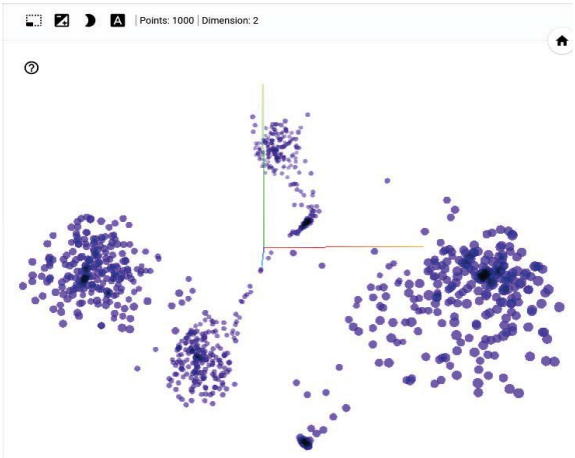

Fig. 6. t-SNE for 2500+ iterations in 3D space forming clusters. The parameters used for computing t-SNE are: Perplexity was 5, Learning rate was 0.01 and Supervise was 0. t-SNE was paused at iteration number 2507

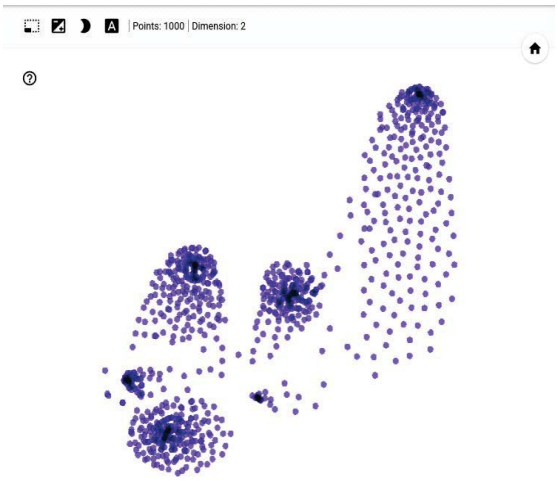

Fig. 7. t-SNE for 5000 iterations in 2D space forming clusters. The parameters used for computing t-SNE are: Perplexity was 5, Learning rate was 0.01 and Supervise was 0. t-SNE was paused at iteration number 5007

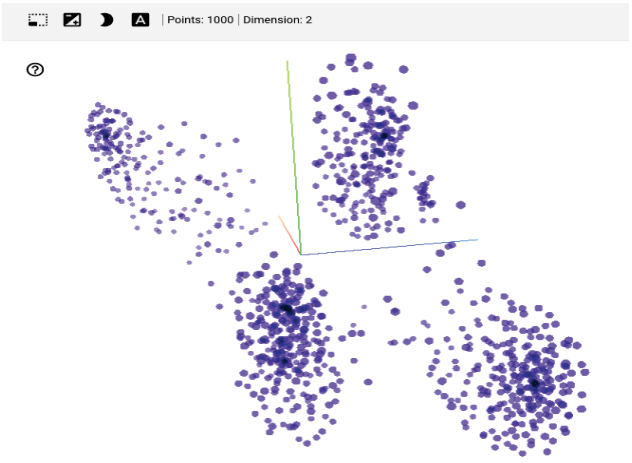

Fig. 8. t-SNE for 5000 iterations in 3D space, a clear formation of 4 different clusters in ranges(0-30%,30-50%,50-70% and above 70%). The parameters used for computing t-SNE are: Perplexity was 5, Learning rate was 0.01 and Supervise was 0. t-SNE was paused at iteration number 5007

C. Pattern Analysis and Classification

The crux of the problem at hand that needs to be solved requires this phase as the most important step of the entire process. The population length, that is, the number of chromosomes per generation is 100 neural networks, and the bar was set manually for 10 generations, implying that the algorithm would stop once ten generations were generated irrespective of the scores of the constituent chromosomes. The data can be collected for about 200-300 neural networks to learn a pattern, however, more data from a pool of neural networks can be collected as well. This step involved the collection of hyperparameters of 300 neural networks to learn a pattern. The hyperparameters that were collected will give an idea about the patterns that the good and bad hypotheses will follow as changing the hyperparameters will change the scores(performance) of these neural network hypotheses as explained [16]. The two algorithms that were used to learn the pattern across the hyperparameters(that were collected and stored in a .csv file) were a Deep Neural Network(Deep Learning) and an SVM(Support Vector Machine). After the patterns were learned(training of the above-mentioned models), classification was performed to see if the models could actually classify good and bad hypotheses, therefore these classifiers could help in identifying the good and bad hypotheses. This would help in dropping the bad performers, therefore ensuring that the next generation does not contain bad hypotheses. This process was followed by controlled mutation of the hypotheses, that involves mutating the hyperparameters of the individual chromosomes in a range that is representative of the hyperparameters of the good hypotheses. This implies that the chromosomes that are generated to maintain the length of each generation at 100 neural networks, will at least converge to perform well and generate a good score.

The hyperparameters of the deep neural network classifier involved 2 hidden layers and a total of 4 layers including the input and the output layers, 102 neurons, activation unit as ReLU(Rectified Linear Unit), sparse categorical cross entropy as the loss function, gradient descent optimizer adam. The SVM classifier that was also used for learning the pattern was compared to the deep neural network. The comparisons are highlighted in the results section. The SVM kernel that was employed for classification was a linear kernel.

D. Controlled Mutation

The classifier predicts which hypotheses are good and which are bad, the bad ones are dropped and not pushed further to the future generations, to accommodate for the dropped off hypotheses, controlled mutation is performed and new offsprings are generated. Controlled mutation involves pseudorandom mutation of the hyperparameters of the constituent chromosomes, in the range of the hyperparameters of the good hypotheses, obtained vaguely through manual observation. Therefore, the length of each generation is restored to its original length of 100 neural networks and pushed for the next generation.

IV. Threats to Validity

Evolutionary algorithms that involve the mutation of hyperparameters may sometime result in varied results for different configurations of hyperparameters. This implies that sometimes deep networks may perform good, and sometimes slight variation in these deep networks may cause a sudden drop in performance. Similar results can be seen in smaller networks that perform good and bad, hence it becomes a problem to learn a specific pattern, and to accurately classify the hypotheses as good and bad. To combat the above problem more data will be needed to feed into the classifier(the first overhead), and to make the classifier a very robust algorithm(the second overhead), which is both complex and tricky. The overhead of this classifier may cause some drop in time improvement. This will reduce the gain that the process aimed to achieve at the very beginning, or provide minimal improvement over the existing functionality provided by genetic algorithms.

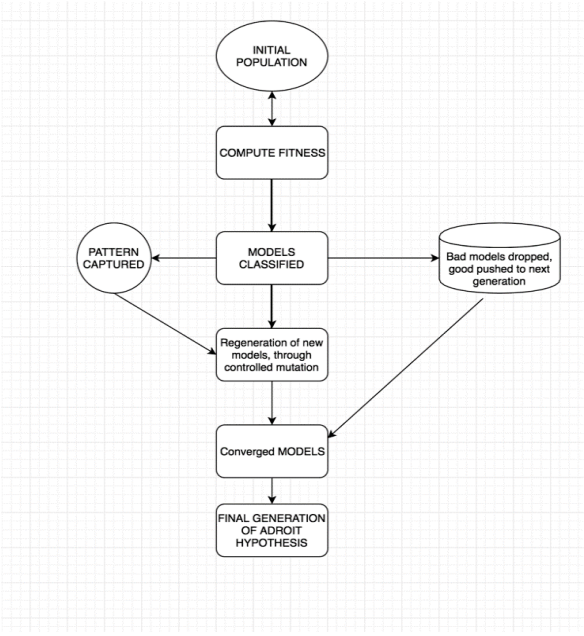

Fig. 9. GAs flow using our approach

V. RESULTS

The first implementation phase(t-SNE exploratory analysis) yielded results showing the formation of different clusters by the good and bad hypotheses, this showed that the patterns followed by the clusters would vary, as the configuration of the data points will vary. These figures would be helpful in learning patterns. The accuracy obtained by the pattern analysis models- Deep Neural Network, was 91.6% accurate(averaged for 10 runs when running for 50 epochs) and the highest accuracy obtained by the DNN classifier was 92.3%. The second classifier model, the Support Vector Machine obtained an accuracy of 81% for classifying the hyperparameters of good and bad hypotheses. Evolutionary algorithms to generate a population of 100 good neural networks when executed for 10 iterations, ran in 7 hours 26 minutes. When the same was run for to generate 100 good neural networks without controlled mutation, default execution of evolutionary algorithms(10 in each generation) for 10 iterations, the time it took was 55 minutes with the distribution as shown in the graph below. The NNs show promise in solving the problem using this approach and speed up the process using the design implemented above, including the classifier's overhead. As seen in the distribution the hypotheses in the last generation(last 10 hypotheses scores), are not very promising, as compared to the controlled mutation execution with scores for 30 neural networks(3 iterations) which converges in 3 iterations. The controlled mutation almost shows convergence in the 3rd iteration compared to default genetic algorithms that provide not very promising results in the 10th iteration as well. The figures below show the various graphs holding the above results.

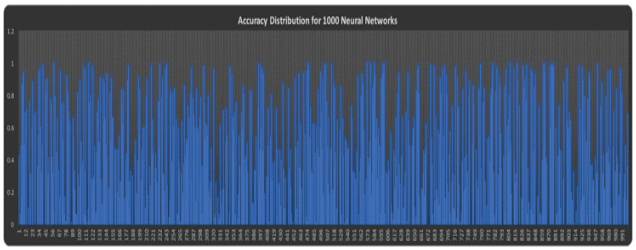

Fig. 10. Accuracy Distribution of 1000 Neural Networks

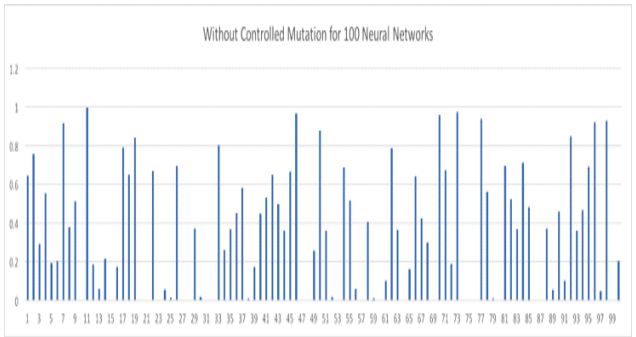

Fig. 11. Without Tweaking Genetic Algorithms accuracy distribution.(100 Neural Networks)

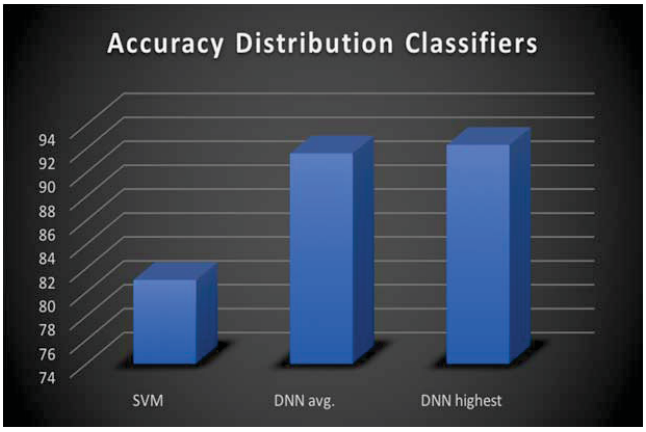

Fig. 11. Accuracy Comparison of DNN classifier and SVM

TABLE I. ACCURACY DISTRIBUTION FOR 30 NEURAL NETWORKS, COMPARING NORMAL AND TWEAKED GENETIC ALGORITHMS RESPECTIVELY

Accuracy	Accuracy
0.64516129	0.56682028
0.75576037	0.0875576
0.29032258	0
0.55299539	0.85253456
0.19354839	0.95852534
0.20276498	0.95852534
0.91705069	0.29493088
0.37788018	0.97235023
0.51152074	0.99539171
0	0.98156682
0.99539171	0
0.1843318	0.42857143
0.05990783	0
0.21658986	0.97695852
0.34232431	0
0.17511521	0.10138249
0.79262673	0.8156682
0.64976959	0.28571429
0.83870968	0.38709678
0.353672	0.47465438
0.368322	0.94356323
0.66820277	0.91751152
0.122622	0.89400922
0.05529954	0.96912442
0.01382489	0.77465438
0.69585254	0.94239631
0.256889	0.98539124
0.56577678	0.99539171
0.67327189	0.90967742
0.01843318	0.98156682

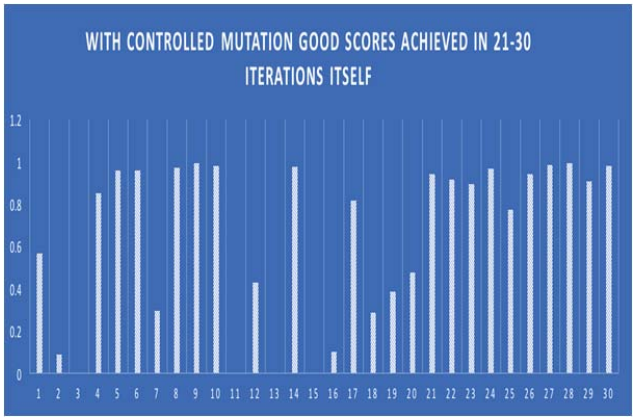

Fig. 12. Controlled and tweaked Genetic Algorithm accuracy distribution

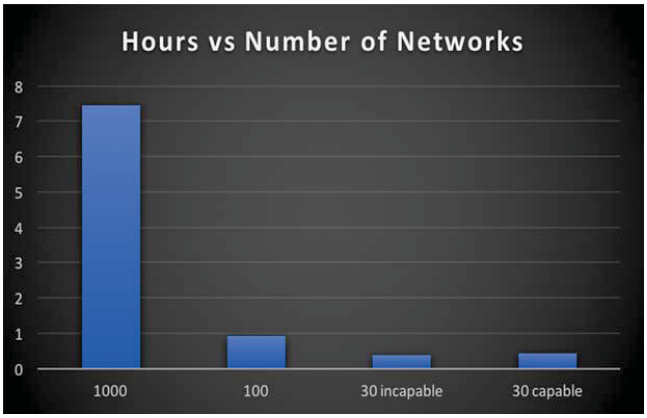

Fig. 13. Time to generate adroit hypotheses

Conclusion and Future Work

The results show that in a reduced number of iterations the algorithm was able to converge in comparison to the vanilla genetic algorithms and this ensured less computation, saving of computation(training time). The classifiers are saved models and hence do not contribute much to the computation time. Therefore, the same problem can be solved in reduced effort and time. However, building intelligence and tweaking the system of evolutionary algorithms is complex and tricky and this may be cumbersome in some cases. This research opens avenues for ways to really democratize AI(Artificial Intelligence), where a layman who has no idea about his/her data can just upload the data, and random algorithms can be run to test performance, and the same can be done in the least computationally heavy manner, giving the user interesting insights into which algorithm works and what configurations are needed for the hyperparameters of the given model that was used to achieve a good performance. The following system which is to be extended as a system for distributed compute service on containers/dockers is still under development, this would allow elastic provisioning of instances and provide a cloud based solution that will ensure automatic scaling and provide a SAAS solution to the user, making AI available at the fingertips.

References

[1] Man, Kim-Fung, Kit-Sang Tang, and Sam Kwong. "Genetic algorithms: concepts and applications [in engineering design]." *IEEE transactions on Industrial Electronics* 43.5 (1996): 519-534.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[2] Mitchell, Melanie. *An introduction to genetic algorithms*. MIT press, 1998.

[3] Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of machine learning research*9.Nov (2008): 2579-2605.

[4] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." *Machine learning* 20.3 (1995): 273-297.

[5] Goodfellow, Ian, et al. *Deep learning*. Vol. 1. Cambridge: MIT press, 2016.

[6] "Applications of Genetic Algorithms" [online]. Available: "https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/tcw2/article1.html"

[7] Moed, Michael C., Charles V. Stewart, and Robert B. Kelly. "Reducing the search time of a steady state genetic algorithm using the immigration operator." *Tools for Artificial Intelligence, 1991. TAI'91., Third International Conference on*. IEEE, 1991.

[8] Gladwin, Dan, Paul Stewart, and Jill Stewart. "A controlled migration genetic algorithm operator for hardware-in-the-loop experimentation." *Engineering Applications of Artificial Intelligence* 24.4 (2011): 586-594.

[9] Nariyoshi, Pedro C., J. R. Deller, and Jinyao Yan. "Modified genetic crossover and mutation operators for sparse regressor selection in NARMAX brain connectivity modeling." *Neural Engineering (NER), 2017 8th International IEEE/EMBS Conference on*. IEEE, 2017.

[10] Khelifa, Meriem, Dalila Boughaci, and Esma Aïmeur. "An enhanced genetic algorithm with a new crossover operator for the traveling tournament problem." *Control, Decision and Information Technologies (CoDIT), 2017 4th International Conference on*. IEEE, 2017.

[11] Saddam, Muhammad, Handayani Tjandrasa, and Dini Adni Navastara. "Classification of alcoholic EEG using wavelet packet decomposition, principal component analysis, and combination of genetic algorithm and neural network." *Information & Communication Technology and System (ICTS), 2017 11th International Conference on*. IEEE, 2017.

[12] Yu, Zhiheng, and Chengli Zhao. "A Combination Forecasting Model of Extreme Learning Machine Based on Genetic Algorithm Optimization." *Computing Intelligence and Information System (CIIS), 2017 International Conference on*. IEEE, 2017.

[13] Tian, Zhiyuan. "Analysis of Route Optimization Based on Genetic Algorithm." Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), 2017 International Conference on. IEEE, 2017.

[14] Liu, Chong, Wenyan Gan, and Yutian Chen. "Research on Portfolio Optimization Based on Affinity Propagation and Genetic Algorithm." *Web Information Systems and Applications Conference (WISA), 2017 14th*. IEEE, 2017.

[15] "Tensorboard Visualizing, Learning| Tensorflow" [online]. Available:https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

[16] Probst, Philipp, Bernd Bischl, and Anne-Laure Boulesteix. "Tunability: Importance of hyperparameters of machine learning algorithms." *arXiv preprint arXiv:1802.09596* (2018).

[17] Ornelas, Francisco, et al. "Genetic algorithm with immigration like strategies of diversification." *Artificial Intelligence (MICAI), 2010 Ninth Mexican International Conference on*. IEEE, 2010.

[18] Potts, J. Craig, Terri D. Giddens, and Surya B. Yadav. "The development and evaluation of an improved genetic algorithm based on migration and artificial selection." *IEEE transactions on systems, man, and cybernetics* 24.1 (1994): 73-86.

[19] Wang, Gai-Ge, Bao Chang, and Zhaojun Zhang. "A multi-swarm bat algorithm for global optimization." *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015.