Evolutionary Neural Networks

Midhath Ahmed & S1700908.

1 INTRODUCTION

In Computer Science, methods of finding optimisations has been vital part of finding solutions to problems. In General, these Optimisation methods allow generic problems to solved easily. This document outlines how I have attempted to find a general optimisation method, involving classifying three separate data sets into two categories using a neural network and training (adjusting weights) using evolutionary algorithm approach, namely a genetic algorithm instead of using the venerable feed-forward and backpropagation approach. In addition, I have compared the two approaches and have tested more data sets to validate the integrity and flexibly of the approach I have come up with.

2 BACKGROUND RESEARCH

Evolutionary Computation is a subset of artificial intelligence, that merges the trial and error optimisation methods with the famous theory of evolution from Charles Darwin. All these methods take the concepts found in nature in finding a good enough solution to a problem in reasonable and cost-effective time frame using evolutionary theory of survival of the fittest: i.e.: Selecting the current best individuals for reproduction and produce new individuals with crossover and mutation, and Finally replace the least fit individuals with the new offspring's.

Ant Colony Optimisation

Ant Colony Optimisation is a probabilistic method of finding solutions to problems that can be solved by finding good paths in traversing a graph(Yang, Shi, Marchese and Liang ,2008). As the name suggests this method inspired by the ants' foraging behaviour in finding the best path of travelling to food back and forth and searching for it.

It is interesting to see that the relatively trivial path optimisation problems such as Traveling Salesmen and robotic pathfinding(Zhangqi, Xiaoguang and Qingyao, 2011) can be solved with great results, this method can also find solutions to some unconventional problems, as a group of researchers managed to find a method to maximize social network profit by identifying the most influential people in a mobile network and theoretically maximise the effectiveness of viral marketing(Salavati and Abdollahpouri, 2019).

Particle Swarm Optimisation

As discussed, previously path finding and navigation is a relatively trivial problem, how ever what if there are constraints such as time and other particles. I.e.: in the case of road traffic, such problems are highly dynamic, adaptive versions of Particle Swarm Optimisation can give solution to these dynamic and everchanging problems. PSO is inspired by the behaviour of swarm animals such as bees and birds, this method is used successfully to not only find paths for UAVs (Shao, Peng, He and Du, 2019) formations, but also to manage and regulate traffic as stated by Marinakis *et al.* (2019). In addition, PSO is also a very viable method of finding optimised architectures for deep convolutional neural networks used in cases like image classification (Fernandes Jr and Yen, 2019), significantly reducing the cost of training such models and improves feasibility of such approaches greatly.

Genetic Algorithm

Genetic Algorithm is another Evolutionary algorithm approach that treats solutions having genetics that can undergo mutation and crossover, paralleling the natural genetic operations in biological sexual reproduction. This approach seems highly compatible with many problems including optimisations and even classifications, the requirement being the problem domain being able to represent a solution as consisting of many atomic parts called genes.

Researchers have been able to use this approach in creating various models()and even a fraud detection model (Yan, Li, Liu and Qi, 2019). In addition, some researchers (Dutta *et al*. 2020)) also managed to get a highly performant classifier when they used a hybrid approach of GA with Artificial Neural Network.

All in all, from my research, the hybrid GA + ANN approach seems to be the most flexible and efficient approaches to my classification problem.

3 EXPERIMENTATION

In order to implement the hybrid ANN + GA, it is essential to understand how an ANN work. The atomic part of an ANN is a neurone, each neurone has many inputs and many outputs. At its core a neurone is summation operator that produces its output by adding all the values and passing that through it an activation to get the output().

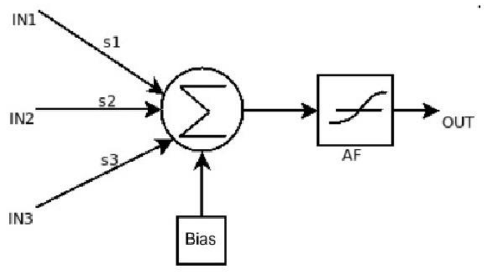


Figure 1-An Artificial Neuron (Perceptron)

Each input has a weight is the sum function is a weighed sum, and bias is a fixed input that have a value of 1.

The Network consists of layers, usually at least three, one for input, hidden and one for output and are usually fully connected .

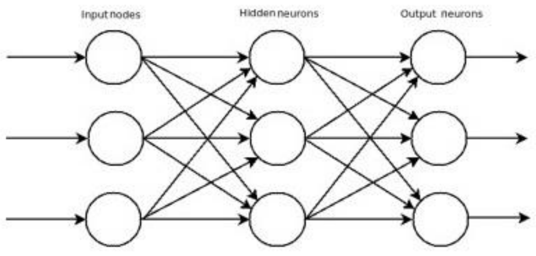
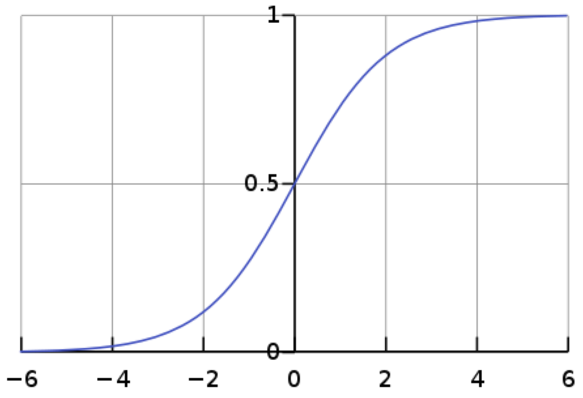


Figure 2- An Artificial Neural Network with 3 inputs and 3 outputs with 1 hidden layer

For Though there are many activation functions, each of them must be differentiable to facilitate back propagation and one of the most used activation functions is sigmoid function.



$$f(x) = \frac{1}{1+e^{-(x)}}$$

Figure 3 - Sigmoid Function

During training the target is to find the set of weights that most accurately maps the inputs to outputs. This is done by feedforward and backpropagation mechanism.

In feedforward for each layer each neurones the formula where activation is a, weight is W and bias is b, is used. After this is applied to the output is generated for the last layer.

$$\boldsymbol{a}^{(l)} = \sigma\left(\boldsymbol{W}\boldsymbol{a}^{l-1} + \boldsymbol{b}\right)$$

Then the Error is calculated by what's called a cost function, and again there are many different types but one of the most common function is the Mean-Square Error, where y_i is the target and $\hat{}y_i$ is the output from the above formula, or the networks guess.

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Next backpropagation can begin and the amount by which each weight needs to be altered is calculated using

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

Finally, weights are updated according to the result of the previous formula:

$$w^{(L)} = w^{(L)} - \text{learning rate} \times \frac{\partial C}{\partial w^{(L)}}$$

In my Approach it is not using the backpropagation that the weights are trained but rather using Genetic Algorithm Approach. This means an individual is a network and its weights are the chromosome, while a separate weight is the gene. As for the fitness function the delta between target and the result of feedforward is used, so the lower this delta the more fit an individual is.

3.1 Data Set 1

This dataset consists of a total 32 data entries, each of those entries having 5 inputs and 1 output. As a reference point I have first used feedforward and back propagation, once with 5 hidden neurones and once with 8, in both cases the learning rate and the number of dataset entries in both trainset and testset is same as **testset_ratio** of 0.1 was used (meaning first 90% of dataset is for trainst, and 10% is for testset) which yielded the following results.

Table 1 - backpropagation results(learning rate = 0.1, hidden = 5, testset ratio = 0.1)

Run#	train set	testset	Network Error
1	96.55%	100.00%	0.063301
2	93.10%	33.33%	0.106554
3	100.00%	66.67%	0.007233
4	93.10%	33.33%	0.097468
Average	95.69%	58.33%	0.068639

Table 2 - backpropagation results(learning rate = 0.1, hidden = 8, testset ratio = 0.1)

Run#	train set	testset	Network Error
1	100%	33.33%	0.009640
2	96.50%	33.33%	0.055112
3	100%	66.66%	0.018226
4	100%	33.33%	0.011529
Average	99.13%	41.66%	0.023627

When the dataset is trained using the genetic algorithm the following is the results yielded.

Table 3 - Genetic Algorithm results (learning rate = 0.1, hidden = 8, testset ratio = 0.1, pop=40, mutation = 0.1)

Run#	train set	testset	Best Fitness
1	96.55%	66.67%	1.959731
2	93.10%	66.67%	2.940096
3	93.10%	66.67%	2.492537
4	93.10%	66.67%	2.316147
Average	93.96%	66.67%	2.427128

Table 4 – Genetic Algorithm results (learning rate = 0.1, hidden = 8, testset ratio = 0.1, pop=100, mutation = 0.1)

Run#	train set	testset	Best Fitness
1	100.00%	66.67%	1.703108
2	100.00%	66.67%	1.145207
3	93.10%	66.67%	2.682849
4	96.55%	66.67%	2.092922
Average	97.41%	66.67%	1.906022

As for the results, the accuracy on both backpropagation and genetic algorithm is good but it could be better, this is most likely due to the small dataset size of the dataset1, also increasing the number of hidden neurons yielded better accuracy in both cases, in addition it seems population size 40 vs 100 does not make much of a difference and all in all the genetic algorithm had better accuracy on the testset in all cases.

3.2 Data Set 2

This dataset consists of total 64 data entries, each of those entries having 6 inputs and 1 output. Again, as a reference I have first done backpropagation, in addition I have decided that for this dataset the hidden layer neurones will be fixed to 12 and population size fixed to 100 as a control, to compare mutation rate.

Training a model using backpropagation on data set yielded the following results.

Table 5 - backpropagation results(learning rate = 0.1, hidden = 12, testset ratio = 0.1)

Run#	train set	testset		Network Error
1	100.00%	50.00%		0.004101
2	100.00%	66.67%		0.003740
3	100.00%	50.00%		0.004222
4	100.00%	33.33%		0.003857
Average	100.00%	50.00%		0.003980

As predicted with more data the network has improve the accuracy both on train and test sets, though there is once case where actually it was worse for the test set.

Table 6 - Genetic Algorithm results (learning rate = 0.1, hidden = 12, testset ratio = 0.2, pop=100, mutation = 0.1)

Run#	train set	testset	Best Fitness
1	100.00%	33.33%	0.422202
2	100.00%	25.00%	0.305705
3	100.00%	33.33%	0.498152
4	100.00%	25.00%	0.399729
Average	100.00%	29.17%	0.406447

Table 7- Genetic Algorithm results (learning rate = 0.1, hidden = 12, testset ratio = 0.2, pop=100, mutation = 0.2)

Run#	train set	testset	Best Fitness
1	92.31%	33.33%	5.317925
2	90.38%	25.00%	5.231562
3	88.46 %	25.00%	7.000213
4	96.15%	33.33%	4.251578
Average	91.83%	29.17%	5.450320

Table 8 - Genetic Algorithm results (learning rate = 0.1, hidden = 12, testset ratio = 0.2, pop=100, mutation = 0.3)

Run#	train set	testset	Best Fitness
1	78.85%	25.00%	9.413559
2	82.69%	33.33%	9.327710
3	82.69%	33.33%	8.809472
4	76.92%	50.00%	9.765058
Average	80.29%	35.42%	9.328950

Table 7 - Genetic Algorithm results (learning rate = 0.1, hidden = 12, testset ratio = 0.2, pop=100, mutation = 0.4)

Run#	train set	testset	Best Fitness
1	73.08%	33.33%	10.848238
2	80.77%	33.33%	10.267149
3	78.85%	41.67%	10.280285
4	82.69%	33.33%	9.765058
Average	78.85%	35.42%	10.290183

When mutation is increased from 10% to 40%, the accuracy drops even for trainset and accuracy becomes much more random form run to run.

3.3 Data Set 3 (and UCI data)

This dataset consists of a total 2000 data entries, each of those entries having 6 inputs and 1 output.

The following is the results yielded by backpropagation optimisation method.

Run#	train set	Test set	Network Error
1	91.33%	87.50%	0.081405
2	93.17%	84.00%	0.066840
3	91.83%	83.50%	0.080865
4	91.17%	84.00%	0.083705
Average	91.88%	84.75%	0.078204

Finally, the following are the results yielded by the hybrid GA-ANN approach;

Run#	train set	Test set	Best Fitness
1	66.10%	68.00%	528.643533
2	67.00%	64.50%	538.675442
3	65.94%	64.50%	555.212603
4	65.94%	67.00%	545.007194
Average	66.25%	66.00%	541.884693

For the last dataset the proposed method is worse than the conventional backpropagation. This could be fixed by redefining the mutator operators on the genome at cross over and mutation stage.

4 CONCLUSIONS

In Conclusion this project has opened my interest in Evolutionary Algorithms and have had first-hand knowledge about implementing one, rather being a concept in theory but in practice too. There are some improvements that can be made such as better fitness function, and extension that can be added to the neural network implementation such as it being able to from multilayer networks.

REFERENCES

Bagherzadeh, S.A., Sulgani, M.T., Nikkhah, V., Bahrami, M., Karimipour, A. and Jiang, Y. (2019) Minimize pressure drop and maximize heat transfer coefficient by the new proposed multi-objective optimization/statistical model composed of “ANN + Genetic Algorithm” based on empirical data of CuO/paraffin nanofluid in a pipe1. Physica A: Statistical Mechanics and Its Applications [online]. 527 [Accessed 03 January 2020].

Dutta, D., Sil, J. and Dutta, P. (2020) A Bi-phased Multi-objective Genetic Algorithm Based Classifier. Expert Systems with Applications [online]. [Accessed 03 Januray 2020].

Fernandes Junior, F.E and Yen, G.G. (2019) Particle Swarm Optimization of Deep Neural Networks Architectures For Image Classification. Swarm and Evolutionary Computation [online]. 49, pp. 62-74. [Accessed 03 January 2020].

Marinakis, Y., Marinaki, M. and Migdalas, A. (2019) A Multi-adaptive Particle Swarm Optimization For the Vehicle Routing Problem with Time Windows. Information Sciences [online]. 481, pp. 311-329. [Accessed 03 Januray 2020].

Salavati, C. and Abdollahpouri, A. (2019) Identifying Influential Nodes Based on Ant Colony Optimization to Maximize Profit in Social Networks. Swarm and Evolutionary Computation [online]. 51, p. 100614. [Accessed 03 January 2020].

Shao, S., Peng, Y., He, C. and Du, Y. (2019) Efficient Path Planning For Uav Formation Via Comprehensively Improved Particle Swarm Optimization. Isa Transactions [online]. [Accessed 03 Januray 2020].

Yang, J., Shi, X., Marchese, M. and Liang, Y. (2008) An Ant Colony Optimization Method For Generalized Tsp Problem. Progress in Natural Science [online]. 18 (11), pp. 1417-1422. [Accessed 03 January 2020].

Yan, C., Li, M., Liu, W. and Qi, M. (2019) Improved Adaptive Genetic Algorithm For the Vehicle Insurance Fraud Identification Model Based on a Bp Neural Network. Theoretical Computer Science [online]. [Accessed 03 January 2020].

Zhangqi, W., Xiaoguang, Z. and Qingyao, H. (2011) Mobile Robot Path Planning Based on Parameter Optimization Ant Colony Algorithm. Procedia Engineering [online]. 15, pp. 2738-2741. [Accessed 03 January 2020].

CODE BASE

All the code used in implementing the GA-ANN method is accessible from the following GitHub repo, please start with the README file.

https://github.com/mdhthahmd/gaann.git