Biologically Inspired Computation Coursework (F21BC)

(Coursework Report)

Karan Singh
Heriot-Watt Universiy
Email: k107@hw.ac.uk

Lucía Parga
Heriot-Watt Universiy
Email: lp52@hw.ac.uk

***Abstract*—Artificial Neural Networks are algorithms with inspiration from the human neurons that humans use as tools to solve any problem. In this case, ANNs will be used for a problem of function approximation. For optimizing the parameters of the ANNs, the Particle Swarm Optimization algorithm will be used. This algorithm is inspired by the swarm behaviour and in this case, each particle will be one ANNs who will modify its parameters (position) simulating the movement of the particles in the swarm and hence, moving towards an ideal value influenced by the global best position of the swarm, its the particle best position, the particle position, adn the informants of the swarm best position. To evaluate the ability of the PSO at optimizing the parameters of the ANNs, different experiments modifying the hyperparameters of the ANNs and the PSo will be carried out.**

***Index Terms*—ANN, PSO, MSE, networks, neuron, optimisation, particle, informants, velocity, position.**

I. Introduction

In this assessment, we are asked to implement and build from scratch two techniques covered in the course. Firstly, build a feed-forward Artificial Neural Network (ANN) architecture using the activation functions provided. Secondly, the implementation of the Particle Swarm Algorithm (PSO) to optimise the ANN's hyperparameters. The aim of the ANN is to solve a function approximation problem.

We then are asked to investigate how the hyperparameters affect the ability of the PSO in the optimization of ANNs. We will run experiments modifying both the hyperparameters of the ANNs (number of layers, choice of activation functions) and the PSO (populaton size, number of informants per particle, acceleration of coefficients) to obtain results on which set of hyperparameters works better for the optimisation of the ANNs given the mathematical function.

A. Language

For this coursework, Python language has been used for the implementation of the code.

II. Program development rationale

A. Neural Network

Neural networks are algorithms that take inspiration from the structure of a human brain and were first modeled using electric circuits in 1943 by McCulloch & Pitts[12]. Neural networks can be used for classification, clustering and predictive analysis[1].

1) ANN architecture: According to Nielson[10], neural networks are able to compute any function or a close approximation, even if the function has multiple inputs and outputs. He adds that neural networks possess $universality$ or the ability to compute any function and by increasing the number of hidden neurons we can improve the approximation. He however, states that this is only true for $continuous$ functions and even simple network architectures can prove to be powerful. Keeping this in mind, we have chosen a simple neural network architecture which consists of 3 hidden neurons.

Hence, the architecture of our ANNs consist in an input layer, a hidden layer (with 3 neurons) and an output layer (1 neuron). (See

Figure 1 below) [1]. To simulate the behaviour of a biological neuron [6] we have also included biases in our architecture: b1, b2 which will help control the value at which activation function will be triggered. Bias is used by the activation functions to better fit the data and determines when neurons fire[3].

We followed the naming convention used by tutorials and documentation sources [14] [9] to describe the neural network's structure in our code e.g. $z2$, $yHat$ etc. and as it can be reflected in the Fig 1.

(a) ANN with 1 input layers (b) ANN with 2 input layers

Fig. 1: ANN's architecture

Our neural network class holds hyperparameters (number of neurons in each layer, number of layers) and parameters (weights, biases and activation functions) of the Neural Network architecture. This class also contains the feedforward (returning predicted output) and Mean Square Error(MSE) which returns a fitness value that can be called directly for each network. To ease calculations with the PSO, both the predicted values and fitness of the NN are saved as a parameter of the network.

Since we are using PSO for optimising the ANN parameters, each NN is treated as a particle and hence, the ANN also holds parameters for the particle of the swarm: position, particle best position, particle best value (best fitness or min MSE calculated), velocity of particle, informants (as an array) and the best value and position of all informants. The parameters that are going to be optimised are the weights and the biases, so for the case of 1 input functions, the Particle position and velocity will have *8 dimensions* to move (6 weights and 2 biases) and in the case of 2 input functions, *11 dimensions* (9 weights dand 2 biases) 1

[1]To ease our evaluation, we have created two ANNs for the two different types of input data for the functions given: 1 or 2 inputs (See Figure 1b)

2) Activation functions: From some examples of literature [11] we've found the function ReLu is very popular as activation function for NN (specially for its use in Deep Neural Networks). We have decided to include it in our experiments and evaluate how well it behaves with our function approximation problem.

B. PSO algorithm

Particle Swarm Optimisation (PSO) was developed in 1995 by James Kennedy as social modelling tool but now finds application as an optimisation tool in engineering and technology[4].

We have defined our PSO class as a swarm with all the parameters and methods of the swarm. The class parameters holds an array with all the particles of the swarm, the swarm best value and position but we have also included as a parameter the global best predicted output (as an useful method of checking the algorithm behaviour). The PSO class calls the methods to set the best value of each particle and the best global value (looping trough all the networks in each particle and trough all the particles, respectively). Both of this are included in a *optimise()* method that calls all the methods for optimising the parameters of the ANNs, based on the PSO algorithm.

1) Informants: although the parameters for the informants are held in each NN (particle), it is in the PSO class where the methods to create them are held. We have chosen to use the *adaptive random topology*[2] method to chose informants. In our implementation, for each iteration, k number of informants are selected randomly from the population plus the particle informs itself $(k+1)$.

2) Movement of particles: following the pseudo-code from the Lecture Notes of the PSO algorithm [13] and the book suggested by our teachers [8] we have defined the movement of the particles. The values of the coefficients were firstly randomly chosen. As

a first thought, we aim to follow the guidelines of the pseudo-code (*"a rule of a thumb is weights alpha-lambda sum 4, this discourages the swarm from imploding or exploding"*), but after contrasting this information with other references and examples found in tutorials[14] and papers[1][4], we concluded to initialise our weights with values in the range 0.8-1. In the experiments described in the Methods section, different values of the coefficients are used to evaluate the performance of the PSO within the problem function approximation.

3) Optimisation: the method $optimise()$ includes all the methods needed to describe a pass of the PSO algorithm (and hence, a first optimization of the ANN for the function approximation problem) can be summarized in three parts in the following order:

- A $forward()$ and $mse()$ method that defines an epoch (pass of inputs) for each particle.
- A set of methods from the PSO class (swarm specific methods) that define the optimization of the ANN parameters.
- A third part where the parameters of each neural network are updated.

This way, we only need to call the method optimise() for each iteration.

The iterations are held in the main and it have been included methods to evaluate the time that the iterations and PSO algorithm take and to create graphs for the mean square error and the ANN output and desired output.

III. Methods

A set of different experiments have been run but in the Results section we will only show the most relevant. Since the PSO algorithm is stochastic[13], in all the experiments proposed, for each condition, the experiment has been run 10 times and the average of all this ten times is shown in the Results section and where our conclusions are based on. The data collected will be the best value global (best fitness/MSE) and the time taken after all iterations.

A. ANN hyperparameters

- Activation function. For each function approximation problem, which is the activation function that performs the best? This is evaluated for 100 iterations and 10 particles.
- Number of neurons per each particle (for 100 iterations and 10 particles). This experiment will use the best activation functions resulted in the previous experiment.

B. PSO hyperparameters

- Number of particles per swarm (100 iterations and values of 10, 20, 40, 80, 160 and 320 particles).
- Number of neurons per particle. We evaluate the efect of the number of neurons in the hidden layer is relevant for the PSO and the function aproximation problem. We evaluate this increasing the number of neurons to 6.
- Number of iterations. Using values of 200, 400, 800, 1600, 3200 and 6400 for a given value of Particles in the Swarm (10,1000). From the Lecture Note we know that a typical value of iterations is 10-100 and hence, we wanted to evaluate if using a higher number of iterations will make a difference in the performance of the PSO with the ANNs.
- Number of informants per each particle. Based on the Paper [5] where they claim that 6 informant particles perform better than other neighborhood formulations of PSO, we aim to evaluate this with our function evaluation problem.
- The acceleration coefficients. Based on a research paper about "How to select inertia and acceleration parameters" [**article**] where they claim the PSO algorithm performs best when the inertia weight w=0.4, and the c1=0.2 and c2=0.8 (c1 and c2, the learning factors defined in the $pseudocode$ from [13] as b and d)[2], our experiments will aim to check if this combination works with our problem function approximation.

[2]with swarm size of 10 and 150 iterations

IV. Results

A. Complete Run Through

After running the algorithm 10 times for 100 iterations for each of the activation functions on the data-sets the mean time taken to return a result and the best value or error was recorded.

The following table displays the results for mean time taken to return a result:

	sigmoid	tanh	cosine	gaussian	null	relu
1in_linear	0.08438	0.08438	0.07031	0.07969	0.09375	0.09063
1in_cubic	0.07656	0.08281	0.07656	0.08438	0.07031	0.08125
1in_sine	0.07813	0.08281	0.06406	0.09688	0.07188	0.07813
1in_tanh	0.07188	0.07500	0.07969	0.08125	0.06406	0.08906
2in_xor	0.07969	0.08750	0.07656	0.08594	0.07500	0.07813
2in_complex	0.07500	0.07344	0.07031	0.07969	0.08594	0.09688

Fig. 2: Mean Time taken to run algorithm

Cosine and *null* activation functions take the least time to return a value whereas *gaussian* and *relu* activation functions takes the longest time.

The following table displays the Mean Square Error (MSE) (also called best value in our code) attained:

	sigmoid	tanh	cosine	gaussian	null	relu
1in_linear	0.03328	0.02456	0.11593	0.25889	0.33340	0.18147
1in_cubic	0.05167	0.03268	0.04979	0.25936	0.15154	0.08495
1in_sine	0.54930	0.13891	0.22280	0.41064	0.50675	0.32113
1in_tanh	0.87753	0.15144	0.31056	0.70785	0.90100	0.49360
2in_xor	0.24732	0.21620	0.04029	0.12367	0.50000	0.18822
2in_complex	0.22390	0.12639	0.11015	0.16398	0.15173	0.14406

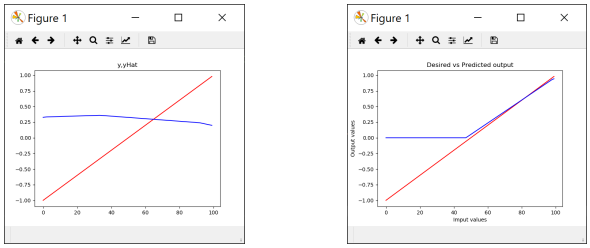
Fig. 3: Mean of best values/MSE

The worst results were achieved with *null* and *gaussian* activation functions and the best results were achieved using the *cosine* and the *tanh* activation functions. It should be noted that the best values/errors remained unchanged through all iterations for *null* activation function which means that no optimisation is possible when using the *null* function both for Hidden and Output Layer.

B. Number of neurons per particle

C. Effects of optimisation

After running the neural net with *sigmoid* activation using the linear data-set we get the result as seen in 15a. Using the PSO with 10 neural nets as particles and 100 iterations we get the result as seen in 15b below.



(a) No optimisation (b) After optimisation

Fig. 4: Optimisation

The graph in 15b clearly shows the PSO optimising the neural network's parameters halfway through the run to converge on the linear function. In graph 5, we can see that by 40 iterations the error rate has converged and does not decrease further. This clearly demonstrates the effectiveness of the PSO algorithm in optimising the neural network's parameters.

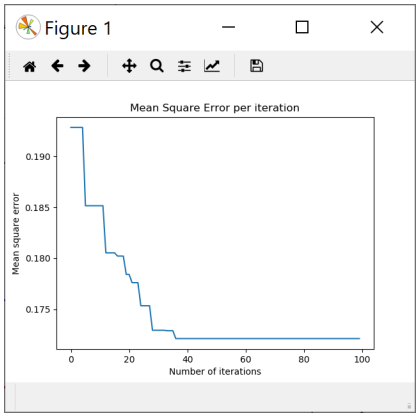


Fig. 5: Error rate in iterations

D. Experiment with number of particles

Using the *cosine* activation function an experiment was conducted to see the influence of increasing the particles on the error in the context of time taken. A cohort of 10, 20, 40, 80, 160 & 320 particles were tested.

As seen from the graph 6, that time complexity increases but there also a reduction in error (graph 7). This would suggest that by increasing the particles in the algorithm the errors can be reduced but this will come at the cost of increased time.

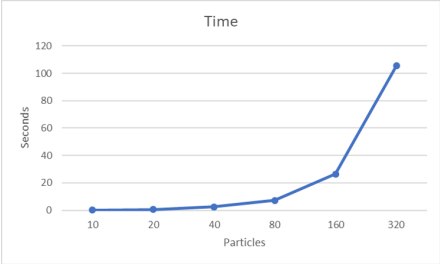


Fig. 6: Time vs particles

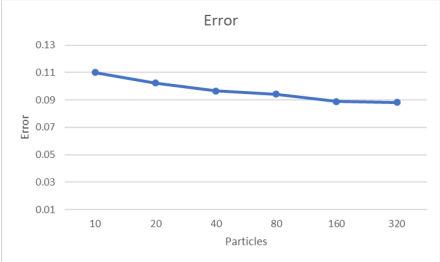


Fig. 7: Error vs particles

E. Experiment with number of iterations

Given that the *cosine* activation function provided the best error rate while running the 2in_complex data-set as shown in Fig 3, it was decided to run an experiment to see the influence of the number of iterations on the error in the context of time it would take. For the experiment, the iterations were from 200, 400, 800, 1600, 3200 & 6400 and the error and time for each iteration cohort was measured.

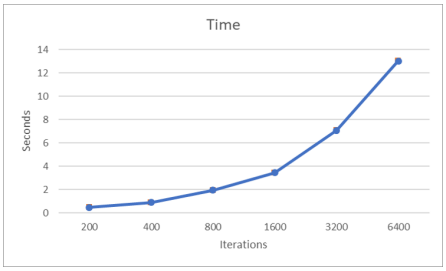


Fig. 8: Time vs iteration

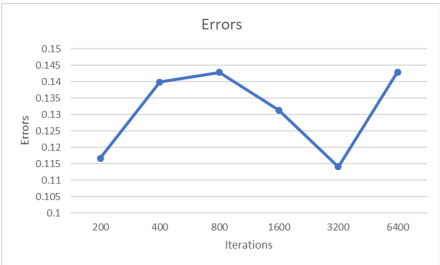


Fig. 9: Error vs iteration

As we could expect, the time taken increases exponentially as the number of iterations increase (see Fig 8) whereas the error rate does not show any clear trend as seen in Fig 9. With these results we can conclude that increasing the number of iterations increases the time complexity but does not contribute to reducing errors.

F. Number of neurons per particle

The table below shows us the results of increasing up to six the number of neurons in the Hidden Layer. We would expected that it would work well with our funciton approximation and specially with XOR (as we have studied in the Lectures f21bcnotes, the problem of XOR was solved by adding one neuron to the hidden layer), but it is not the case: XOR function gives worse result than weh using 3 neurons. For the functions of sinus, tnah and complex, the increase of number of neurons increases the ability of the PSO to optimize the ANNs.

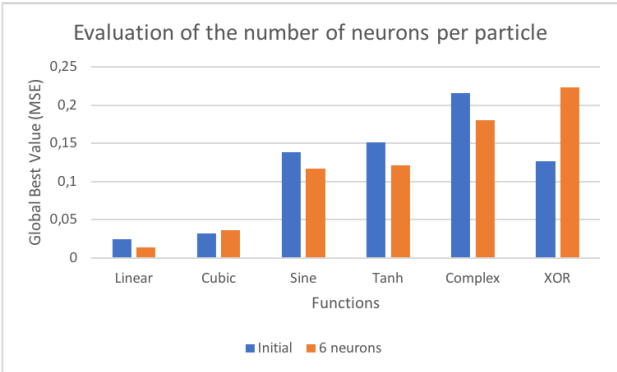


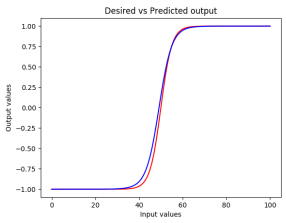
Fig. 10: Evaluation of the effect of the number of neurons per particle

We also show below the accuracy of this optimization with the tanh and sine [3]:

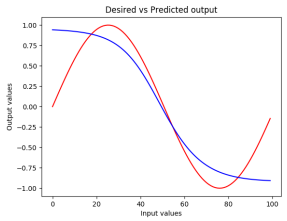
G. Number of informants

We've evaluated the effect of the number of informants over the PSO optimization running experiments over all the functions. In the graph below the results from the *tanh* activation function (used both in Hidden Layer and Output

[3]for this graphs and the following of same style, red shows the desired output and the blue the predicted final output



(a) Tanh function evaluated with 6 neurons in the Hidden Layer. Global Best Value: 0,0024



(b) Sine function. Global best value:0,13891

Layer) as it was recorded with best results. We shall highlight that the Global Best Values should be minimum, and hence, we can say that, for example, in the case of the XOR function approximation, the use of 6 informants instead of 3 reduces the optimization performance of the PSO (higher value of the Global Best). In the case of the $sine$ and the $Complex$ function, the effect is the opposite and by using a higher number of informants enhances the PSO optimization of the parameters of the ANN.

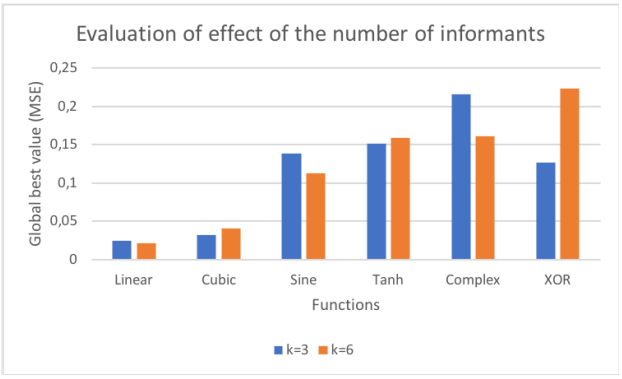
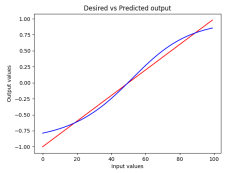


Fig. 12: Evaluation of the effect of the number of informants

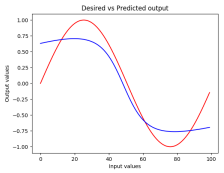
Below, we show how well this works witn the Linear, Sinus and Complex Function approximation:

H. Acceleration coefficients

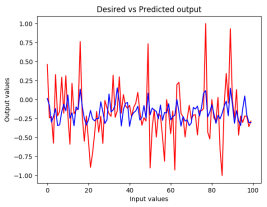
The Fig. 10 shown blow shows the result of evaluating the PSO algorithm within the function approximation problem modifying the acceleration weights. The table only shows the results for the $tanh$ activation function (used both in Hidden Layer and Output Layer). It can be observed that



(a) Linear function. Global best value: 0,0245



(b) Sinus function. Global best value: 0,13891



(c) Complex Function. Global best value:0,2162

Fig. 13: Optimisation

the modification of the weights in the activation function affects negatively the PSO within the funciton approximation problem, highlighting it is worse when applied to the $tanh$, $sine$ and XOR function.The results in the other functions are quite similar to the initial combination. With the observed results we can say that the acceleration coefficients take an important role in the PSO optimization and that combination we've used here based from the research paper [7] mentioned in the Methods Section is not the most suitable for the function approximation problem we are studying.

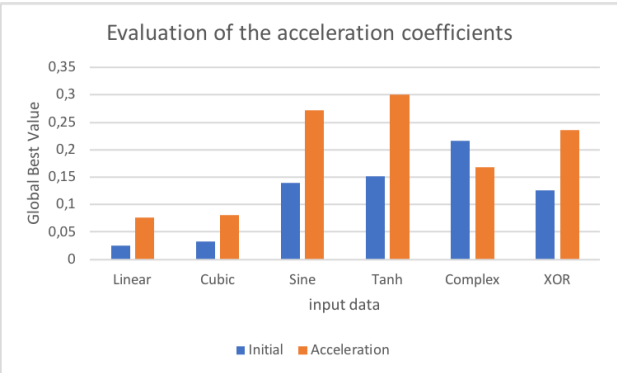
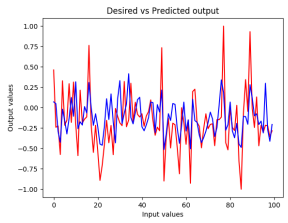
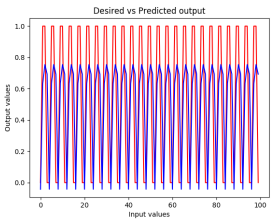


Fig. 14: Evaluation of the effect of the acceleration coefficientss

In the graphs below only the Complex and XOR function are presented:



(a) Complex function. Global best value: 0,2162



(b) XOR function. Global best value:0,12639

V. Discussion and conclusion

- Using a number of six informants per particle affects negatively the ability of the PSO algorithm in optimizing the parameters of the ANNs in both the different types of function approximation problems. Only when applying it to the $complex$ function, we obtain a significant positive effect.
- The values used to evaluate the effect of the acceleration coefficients [7] are not suitable for our function evaluation problem as worse results are obtained. The acceleration coefficients are tricky to choose but could be a very interesting point for further study since in our experiments we've seen that a little variation results in a big difference in the Global best result
- Increasing the number of neurons affects positively the PSO for our problem. In the case of XOR, this is affecting negatively. We conclude that more experiments in the number of neurons should be run, also adding more Hidden layers to possibly see a different or better behaviour in the PSO optimization.
- The modification in the number of iterations (on a high range) does not affect the ability of the PSO (based on the results obtained): neither positive, neither negative. For our function approximation problem, the Global best value is already achieved within few iterations and so we conclude that the modifications of other hyperparameters should be studied for the PSO performance.
- Increasing the particle size in a PSO algorithm seems to reduce the error rate.
- The choice of activation function is important both in terms of run time and error rate performance.

Overall, we think the perfec combination of hyperparametes is difficult and tedious to achieve but it can be achievable. For this Function Approximation Problem, we obtained the increasing the number of neurons , swarm size, using a low number of informants and a higher swarm size affect positively to the PSO ability of optimizing the ANNs parameters.

VI. References

[1] *A Beginner's Guide to Neural Networks and Deep Learning*. URL: https://skymind.ai/wiki/neural-network.

[2] Maurice Clerc. "Beyond standard particle swarm optimisation". In: *Innovations and Developments of Swarm Intelligence Applications*. IGI Global, 2012, pp. 1–19.

[3] Jaron Collis. *Glossary of Deep Learning: Bias*. 2017. URL: https://medium.com/deeper-learning / glossary - of - deep - learning - bias - cf49d9c895e2.

[4] Juan Luis Fernández-Martínez. "A Brief Historical Review of Particle Swarm Optimization (PSO)". In: *Journal of Bioinformatics and Intelligent Control* 1 (June 2012), pp. 3–16. DOI: 10.1166/jbic.2012.1002.

[5] José Garcia-Nieto and Enrique Alba. "Why Six Informants is Optimal in PSO". In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*. GECCO '12. Philadelphia, Pennsylvania, USA: ACM, 2012, pp. 25–32. ISBN: 978-1-4503-1177-9. DOI: 10.1145/2330163.2330168. URL: http://doi.acm.org/10.1145/2330163.2330168.

[6] Simon Haykin. *Neural networks and Learning Machines*. Prentice Hall PTR, 1994.

[7] Min Jin, Xiangyuan Zhong, and Xudong Zhao. "Whether and How to Select Inertia and Acceleration of Discrete Particle Swarm Optimization Algorithm: A Study on Channel Assignment". In: *Mathematical Problems in Engineering* 2014 (Jan. 2014), pp. 1–6. DOI: 10.1155/2014/758906.

[8] Michael Lones. *Sean Luke: Essentials of Metaheuristics*. 2011.

[9] Lester James V. Miranda. *Training a Neural Network*. URL: https://pyswarms.readthedocs.io / en / development / examples / custom_objective_function.html.

[10] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA: 2015.

[11] Johannes Schmidt-Hieber. "Nonparametric regression using deep neural networks with ReLU activation function". In: *arXiv preprint arXiv:1708.06633* (2017).

[12] Kate Strachnyi. *Brief History of Neural Networks*. URL: https://medium.com/analytics-vidhya/brief-history-of-neural-networks-44c2bf72eec.

[13] Patricia A. Vargas and Michael Lones. *Lecture Notes F21BC*. 2019.

[14] Stephen C Welch. *Neural Networks Demystified*. URL: https://www.youtube.com/watch?v=bxe2T-V8XRs.