Optimizing ANNs Using PSO

Omar Basem

This is a discussion of the results of my tests and investigations of the hyperparameters of the PSO and ANN. The parameters that will be covered in detail are: PSO population size, PSO informants count, ANN neurons per layer and ANN hidden layers count. For the rest of the hyperparameters I will show the results of my tests briefly. Also, some graphs for the results will be shown along the description.

For each test one parameter will be investigated and the rest will be fixed. I carried out some tests before deciding on the default parameters which are: 100 swarm size, 8 informants, 0.75 inertia, 0.75 jump size, 1.0 noise coefficients, 2 ANN layers, 3 neurons per layer (except for complex, default is 6), and tanh as an activation function. Each test will go through 500 epochs and will be the average of 10 runs. The results will be reported in the following format showing 3 measures of quality:

MSE / epoch at which MSE reaches below 0.0025 (2.5e-3) / execution time (s)

For example: (1.32e-3 / 62i / 2.7s) means 0.00132 MSE, and MSE reached below 0.0025 at the 62nd iteration or epoch, and the execution time was 2.7 seconds. The test which gives least number of iterations to reach below 0.0025 MSE will be picked and the "best" result and bolded in the tables below. If no test reached below 0.0025, the test which gives the least MSE will be picked. I give my conclusions at the end of each tests section. The following table shows the mathematical functions that will be approximated using the ANN:

Function name	Equation
linear	$y = x$
cubic	$y = x^3$
sine	$y = \sin(x)$
tanh	$\tanh(x)$
xor	$y = x_1 \oplus x_2$
complex	$y = 1.9\{1.35 + e^{x_1 - x_2}\sin[13(x_1 - 0.6)^2]\sin[7x_2]\}$

1.1. Population size

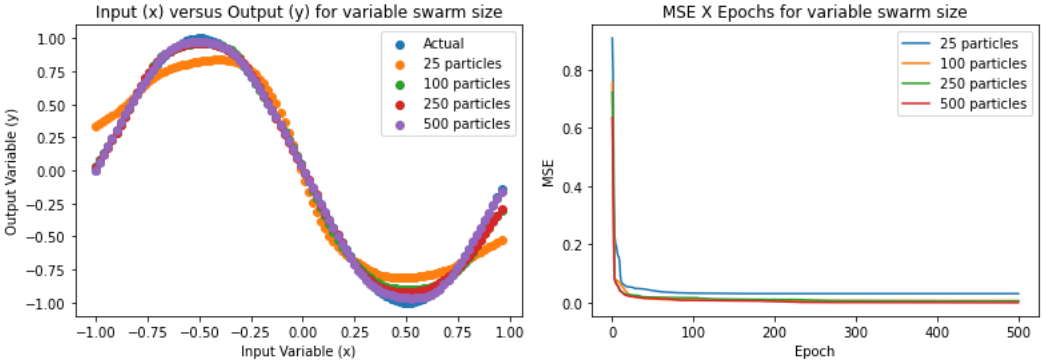
Population size is crucial for PSO. A too small population size can cause the algorithm to fall into a local minimum, while a too large population size can cause slow convergence. So, an appropriate population size is needed to maintain the effectiveness of the algorithm. Piotrowski et al., (2020) claims that the classical choice of 20-50 particles since PSO's introduction in 1995 is too small, and they recommend a swarm size of 70-500 particles for best performance. Based on that I will test these 4 different swarm sizes: 25, 100, 250 and 500. The following table summarizes the results.

	25 swarm size	***100 swarm size***	***250 swarm size***	***500 swarm size***
linear	1.32e-3 / 62i / 2.7s	2.03e-5 / 24i / 12.5s	3.14e-5 / 19i / 39.8s	**1.77e-6 / 13i / 110.4s**
cubic	9.45e-3 / 63i / 2.6s	3.68e-4 / 66i / 12.0s	**2.79e-5 / 23i / 39.6s**	4.10e-5 / 24i / 111.5s
tanh	1.23e-6 / 9i / 2.7s	1.36e-7 / 5i / 12.4s	**7.93e-10 / 2i / 40.4s**	5.92e-10 / 3i / 112.6s

sine	3.13e-2 / null / 2.7s	**2.19e-3 / 79i / 12.7s**	5.93e-3 / 182i / 40.4s	2.81e-4 / 166i / 113.5s
XOR	1.15e-1 / null / 2.7s	5.81e-11 / 27i / 12.5s	8.47e-12 / 20i / 39.8s	**3.10e-12 / 12i / 112.1s**
complex	5.81e-2 / null / 2.8s	2.41e-2 / null / 12.7s	9.23e-3 / null / 40.5s	**2.23e-03 / 378i / 113s**

My conclusions: a small swarm size of 25 may not be enough in many cases to reach the global optimum. Sine, XOR and complex functions could not reach below 0.0025 MSE with a swarm size of 25. The complex function was able to reach MSE below 0.0025 using at least 500 swarm size, however such big swarm size increases the execution time about 3 times compared to a swarm size of 250. A swarm size of 100-250 seems to be the most reasonable.

Different swarm sizes graphs for sine function and the MSE against Epochs is shown below. It is clear that swarm size of 25 had the worse resutls:

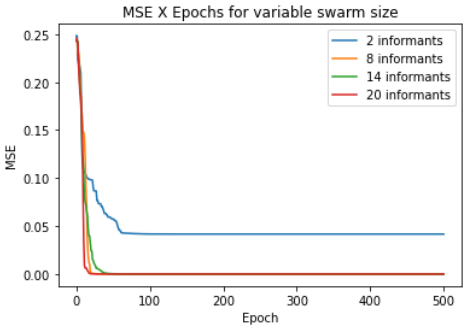


1.2. Number of informants

Garcia-Nieto et al., (2012) suggests that a number of 4-8 informants to be optimal. Based on that I will test the following different number of informants: 2, 8, 14, 20

	2 informants	***8 informants***	***14 informants***	***20 informants***
linear	9.67e-5/ 34i / 10.6s	3.90e-5/ 23i / 13.1s	8.85e-5 / 31i / 17.5s	**9.49e-5/ 20i / 22.8s**
cubic	1.35e-4 / 50i / 10.8s	1.26e-4/ 33i / 13.5s	2.57e-4 / 36i / 17.7s	**3.49e-4 / 29i / 23.1s**
tanh	**8.55e-9 / 3i / 10.5s**	3.58e-8 / 4i / 13.1s	1.21e-8 / 5i / 17.3s	7.63e-8 / 4i / 22.8s
sine	**1.71e-3 / 108i / 9.8s**	6.76e-3 / null / 12.8s	1.21e-2 / null / 16.4s	3.89e-03 / null / 21.3s
XOR	4.17e-2 / null / 10.2s	9.22e-09 / 18i / 12.9s	1.01e-11 / 27i / 16.9s	**1.48e-11 / 15i / 22.0s**
complex	2.21e-2 / null / 10.8s	**2.06e-2/ null / 13.6s**	2.09e-2 / null / 17.8s	3.59e-2 / null / 22.4s

My conclusions: contrary to Garcia-Nieto et al., (2012) claim, 8 informants did not give the best results for any of the functions apart from the complex function. So, I suspect his claim was based on complex functions. For tanh and sine lesser informants showed better results, while for linear, cubic and XOR more informants showed better results. The following figure shows MSE against epochs number of the XOR function for variable number of informants:

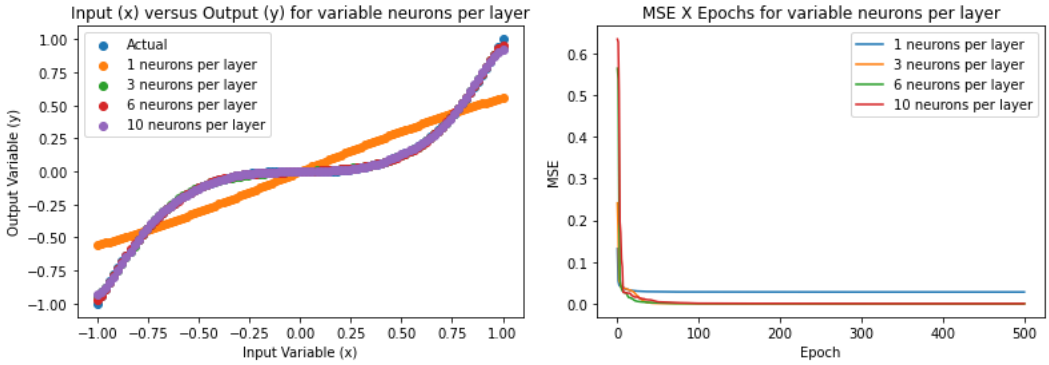


1.3. Number of neurons per layer

In this test I will investigate the effect of the number of neurons per layer. I test the following 4 different number of neurons per layer: 1, 3, 6 and 10. The following table summarizes the results.

	1 neuron/layer	***3 neurons/layer***	***6 neurons/layer***	***10 neurons/layer***
linear	3.34e-3 / null / 12s	**2.21e-5 / 17i / 13.5s**	9.35e-5 / 29i / 13.8s	7.20e-5 / 31i / 14.5s
cubic	2.80e-2 / null / 11.9s	**1.98e-4 / 35i / 12.6s**	8.40e-5/ 36i / 13.0s	2.51e-4 / 71i / 13.3s
tanh	4.07e-9 / 4i / 11.8s	**3.27e-8 / 3i / 12.4s**	4.73e-9 / 5i / 12.6s	1.13e-7 / 7i / 13.2s
sine	5.87e-2 / null / 14.1s	1.16e-3 / 207i / 13.9s	**7.82e-3 / 70i / 14.1s**	1.39e-2 / 65i / 15.5s
XOR	1.67e-1/ null / 12.2s	**7.54e-16 / 15i / 12.9s**	8.84e-13 / 25i / 13.5s	2.34e-9 / 20i / 13.8s
complex	7.81e-2 / null / 12.1s	3.22e-2 / null / 13.0s	2.01e-2 / null / 13.5s	**1.95e-2 / null / 14.0s**

My conclusions: more neurons allow to approximate functions more accurately. It is clear that 1 neuron/layer had the worst performance, and this shown well in the figure below of the cubic function. However, more neurons will not always give better results. The perfect number of neurons depends on the complexity of the function. As the complexity of the function increases, the number of neurons needed to approximate it more accurately increases. The complex function got the best result with 10 neurons/layer.

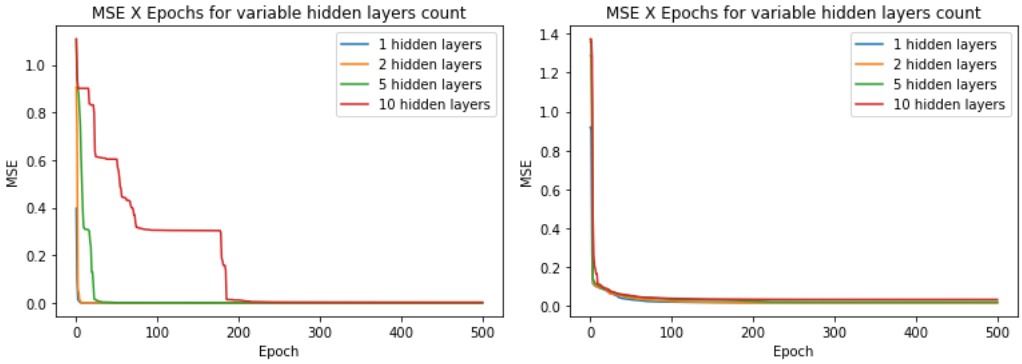


1.4. Number of layers

In this test I will investigate the effect of the number of hidden layers. I will test the following 4 different number of hidden layers: 1, 2, 5 and 10. The following table summarizes the results.

	1 hidden layer	*2 hidden layers*	*5 hidden layers*	*10 hidden layers*
linear	1.09e-4 / 23i / 11.9s	**1.51e-5 / 21i / 12.2s**	2.06e-4 / 70i / 13.9s	9.73e-5 / 90i / 16.6s
cubic	1.88e-4 / 41i / 12.1s	1.50e-4 / 50i / 13.0s	**1.14e-4/ 50i / 14.6s**	9.78e-3 / null / 17.1s
tanh	**1.00e-8 / 5i / 12.2s**	5.39e-9/ 6i / 12.8s	8.00e-6 / 34i / 14.4s	2.40e-3 / 156i / 17.2s
sine	**2.09e-3 / 132i / 12.3s**	6.82e-3/ null / 12.9	7.88e-3 / null / 14.2s	2.31e-2 / null / 16.9s
XOR	**7.32e-13 / 14i / 12.2s**	6.27e-11 / 18i / 12.8s	3.34e-7 / 46i / 14.5s	8.87e-7 / 63i / 16.7s
complex	1.73e-02 / null / 12.5s	**1.65e-2/ null / 13.8s**	2.12e-2 / null / 17.3s	3.36e-2 / null / 22.1s

My conclusions: given that enough number of epochs is being used, the number of hidden layers does not affect much the final MSE. However, it greatly affects the iteration at which it converges for the simpler functions. In general, more hidden layers caused convergence to be reached more lately. This can be noticed clearly on the MSE against epochs graph of the tanh function below (left graph). For the complex function the difference is very minimal (right graph). Based on that I can conclude that the more complex a function is, the less sensitive it is to the number of hidden layers, and vice versa.



4.5 Other hyperparameters

Activation function: my investigations has shown the tanh activation function to be give the best results. This is because tanh is zero centered making it more accurate to model inputs that have both positive and negative values.

Inertia: inertia takes values from 0.0 to 1.0. It affects the velocity in every iteration. Small inertia would result in smaller velocity, bigger inertia would result in bigger velocity. Too big inertia can lead to falling into local minimum trap, while too small inertia may result in not finding the global optimum before the program terminates. I have found an inertia of 0.75 to give the best results. Worth noting that Arasomwan et al., (2013) proposes a paper where they claim Linear Decreasing Inertia Weight PSO (LDIW-PSO) to outperform some other variants of PSO. I tried using their method, but it seemed to actually have a slightly worse

performance than fixed inertia weight, at least for the given functions to be approximated by the ANN.

Jump Size: the jump size takes values from 0.0 to 1.0. It has a similar effect to the inertia. The difference is that unlike the inertia, it does not update the velocity, but it affects how much the particles position will be updated by the velocity. Like the inertia, too big jump size can lead to falling into local minimum trap, while too small jump size may result in not finding the global optimum before the program terminates. My tests concluded that a jump size of 0.75 to give the best results.

Random noise coefficients: random noise coefficients go into the equation of updating the velocity. There is no defined range for them. This added noise allows particles to explore more random space while updating the velocity, thus increases the probability of finding the global optimum. However too much noise may cause the particles to deviate away from the right path. I found that random noise coefficients of 1.0 for each of personal, informants and global coefficients to give the best results.

References

Arasomwan, M.A. and Adewumi, A.O., 2013. On the performance of linear decreasing inertia weight particle swarm optimization for global optimization. *The Scientific World Journal*, *2013*.

Garcia-Nieto, J. and Alba, E., 2012, July. Why six informants is optimal in PSO. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (pp. 25-32).

Piotrowski, A.P., Napiorkowski, J.J. and Piotrowska, A.E., 2020. Population size in Particle Swarm Optimization. *Swarm and Evolutionary Computation*, p.100718.