Hyper-parameter optimization with Swarm Intelligence
Study on a traffic classification problem – Phase II

Eliza Czaplicka 2022110771

1. Swarm Optimization

This study is focused on the Swarm Optimization Algorithm. This algorithm is inspired by the natural behaviour of organisms in groups. The base one is called the Particle Swarm Optimization. It works as the movement of fish or birds; each animal is responsible for some space and based on the results of others in their group it travels to the best place. In the animal world, it could be a place with the most food, and in the algorithm, the best coordinates are found by a fitness function.

This algorithm uses many agents to search the search space for the best solution to a given problem. It updates the position of each agent in each iteration based on its performance and the performance of other agents. The benefit of this solution is the greater chance of not falling into a local optimum, because of multiple starting points. What is more, the swarm algorithms can dynamically adjust the learning rates of the process while computing based on different agents' performance.

Swarm Algorithms are usually used for optimization problems but are also especially useful for hyper-parameter tuning of other algorithms, as in this case. The customization of the fitness function allows for a lot of different usages of this algorithm. It can also be used for Neural Network architecture search to find the optimal layers.

A disadvantage of this method is only the computation time needed for the search to find the best solutions. In the case of neural network architecture, each agent needs to train the network to find its fitness and that repeats for each iteration which can be time consuming. Moreover, wrong parameter tuning for the swarm algorithm may lead to a non-optimal solution.

2. Whale Swarm Optimization

Whale Swarm Algorithm is a type of Swarm optimization algorithm based on a hunting pattern of whales. Whales communicate with each other by ultrasounds, letting others in their group know about the amount of food at their location. The movement of whales is then based on the closest whale with better performance than them, which is calculated by distance between whales and the amount of food.

With this algorithm it is possible for agents to discover the global optimum while also looking at the local optima for possible solutions. The movement of each agent is not affected by all the other agents or only the global best, but by best in relation to them.

The Whale Swarm function in SwarmPackagePy is described by parameters of number of agents, fitness function, lower and upper bounds for the search space, dimension of search space and iterations (as other function in this package), as well as ro0 parameter describing the intensity of ultrasound at its origin and eta parameter describing the probability of message distortion over large distances.

The Particle Swarm algorithm has the same base parameters. For its own parameters, it consists of an inertia weight, which is an importance of past velocity with the current one. Next is the cognitive parameter, which controls the impact of current particle's best position, and the social parameter, which represents the importance of the global best position of the agents.

The main difference between this algorithm and the classic Whale Swarm Optimization algorithm is the reliability on agents own position and past positions. While in Particle Swarm the past movement plays a significant role as the velocity factor, in the Whale Swarm algorithm the most important is the best fitness for the agent, current fitness, and fitness of best agent in the neighborhood. Moreover, the Whale algorithm focuses on the best agent in relation to the current agent, enabling it to discover more of the search space, while Particle Swarm focuses on the global best from the start. It is more prone to falling into a local optimum.

3. Applying algorithms to Ackley function

For analysis of different parameters of the Particle Swarm algorithm and the Whale Swarm algorithm tests were concluded with the Ackley function as the accuracy function.

Ackley function is popular for testing optimization algorithms. The plane of the function consists of a nearly flat surface with a large dip in the middle. The closer the tested optimization algorithm gets to a position [0, 0] the better the solution. Of course achieving a perfect 0 point is nearly impossible, but values close to that are highly probable with a right algorithm. The Ackley function for 3 dimensions works similarly. It achieves the best result in the point [0, 0, 0].

In the Particle Swarm algorithm, the inertia variable was tested as 0,3, 0,5, 0,7; the cognitive variable as 1, 1,5, 2, and social variable as 1, 1,5, 2. The tests were computed with dimensions 2 and 3. The best position and fitness results were taken as the best out of 5 runs.

Algorithm	inertia	cognitive	social	Best Position	Best Fitness
PSO	0.3	1.0	1.0	[0.39202388 0.97991452]	4.787299070786659
PSO	0.3	1.0	1.5	[0.3202193 0.15107241]	2.65829484714228
PSO	0.3	1.0	2.0	[0.6627839 0.42263767]	4.577299796393149
PSO	0.3	1.5	1.0	[0.50835509 0.90559884]	4.823975866385155
PSO	0.3	1.5	1.5	[-0.06988541 0.06606693]	-3.0919130869951803
PSO	0.3	1.5	2.0	[0.68307203 0.17420064]	4.048979297177425
PSO	0.3	2.0	1.0	[0.63649584 -0.15478945]	3.7828995900137667
PSO	0.3	2.0	1.5	[0.50207682 0.72133991]	4.98229087848118
PSO	0.3	2.0	2.0	[0.37483036 0.40022217]	4.3454099562290835
PSO	0.5	1.0	1.0	[0.38913841 0.6178715]	4.971902569312821
PSO	0.5	1.0	1.5	[0.4161173 -0.02054605]	3.142376169778857
PSO	0.5	1.0	2.0	[0.14603288 0.05024679]	-1.489988311093128
PSO	0.5	1.5	1.0	[0.25869549 -0.25396965]	2.8740647785510807
PSO	0.5	1.5	1.5	[0.7183727 0.8077931]	4.741199642339492
PSO	0.5	1.5	2.0	[0.53470124 0.09321007]	3.40820704081137
PSO	0.5	2.0	1.0	[0.32014291 0.65521566]	4.586001631448596
PSO	0.5	2.0	1.5	[0.11786097 0.82054654]	2.2439480445941276
PSO	0.5	2.0	2.0	[0.68879214 0.15306072]	3.94124819031892
PSO	0.7	1.0	1.0	[0.25263528 0.12160557]	1.6893986238885899
PSO	0.7	1.0	1.5	[0.47251397 0.553861]	4.204725703435148
PSO	0.7	1.0	2.0	[0.05042122 -0.01844199]	-4.05800852967279
PSO	0.7	1.5	1.0	[0.86711908 0.40487908]	4.884976944379336
PSO	0.7	1.5	1.5	[0.9779206 0.79468021]	3.6151296962298933
PSO	0.7	1.5	2.0	[-0.29204511 0.04778213]	1.4076823817284887
PSO	0.7	2.0	1.0	[-0.01342327 0.66741717]	3.261934498348254
PSO	0.7	2.0	1.5	[0.63863528 -0.27106971]	4.5840012042507645
PSO	0.7	2.0	2.0	[0.70309022 0.71967487]	4.787916503067947

Figure 1. PSO algorithm performance with dimension 2.

Algorithm	inertia	cognitive	social	Best Position	Best Fitness
PSO	0.3	1.0	1.0	[0.63550739 0.15892478 0.96668819]	5.459904027324146
PSO	0.3	1.0	1.5	[-0.01939568 0.50060302 0.87395607]	4.34785360125851
PSO	0.3	1.0	2.0	[0.43843034 0.59362901 -0.34192873]	5.284725379686471
PSO	0.3	1.5	1.0	[-0.24620572 0.5830374 0.20894082]	4.018629251992152
PSO	0.3	1.5	1.5	[0.36875007 0.78870052 0.24482594]	5.022026075655155
PSO	0.3	1.5	2.0	[-0.13633765 0.850423 0.0676423]	-2.9191658342649762
PSO	0.3	2.0	1.0	[0.82872569 0.62234735 0.30484667]	5.443841945230169
PSO	0.3	2.0	1.5	[0.9039116 0.57613677 0.42815293]	5.139720284133109
PSO	0.3	2.0	2.0	[0.79486632 -0.02064674 -0.05374579]	-4.296082058095639
PSO	0.5	1.0	1.0	[0.84600493 0.09562019 0.0607929]	-4.852985063410198
PSO	0.5	1.0	1.5	[0.21227227 0.8964871 0.06481505]	-1.1085310891442997
PSO	0.5	1.0	2.0	[0.01978672 0.36426418 -0.03792257]	0.26467252817894904
PSO	0.5	1.5	1.0	[0.86457553 0.34104201 -0.07334219]	3.2123531262971388
PSO	0.5	1.5	1.5	[0.54151348 0.97526243 0.80608782]	5.439931390481448
PSO	0.5	1.5	2.0	[0.87268773 -0.18856052 0.21759591]	2.0594152988899066
PSO	0.5	2.0	1.0	[0.18716041 0.84619067 -0.0089311]	-2.07423352295531
PSO	0.5	2.0	1.5	[0.51996984 0.85733948 0.17942696]	4.863356099165781
PSO	0.5	2.0	2.0	[0.50031495 0.47261589 0.9695784]	5.299351772040946
PSO	0.7	1.0	1.0	[0.65170174 0.52030289 0.98121829]	5.219612818344849
PSO	0.7	1.0	1.5	[-0.55244467 0.90048694 -0.65008826]	4.7966917383729495
PSO	0.7	1.0	2.0	[0.97638311 0.42822724 0.91929376]	5.240981073618851
PSO	0.7	1.5	1.0	[-0.20640216 0.54742132 0.26729459]	4.1997662181776505
PSO	0.7	1.5	1.5	[0.38161873 0.93008258 0.94875428]	5.017998348635697
PSO	0.7	1.5	2.0	[0.43593521 0.44784954 -0.09885577]	4.782950757253488
PSO	0.7	2.0	1.0	[0.93546161 0.07249467 0.15204445]	-4.99831024343808
PSO	0.7	2.0	1.5	[0.84495595 0.4305277 -0.34844145]	4.940086522691725
PSO	0.7	2.0	2.0	[0.53277886 0.81100582 0.64754564]	5.685913053188482

Figure 2. PSO algorithm performance with dimension 3.

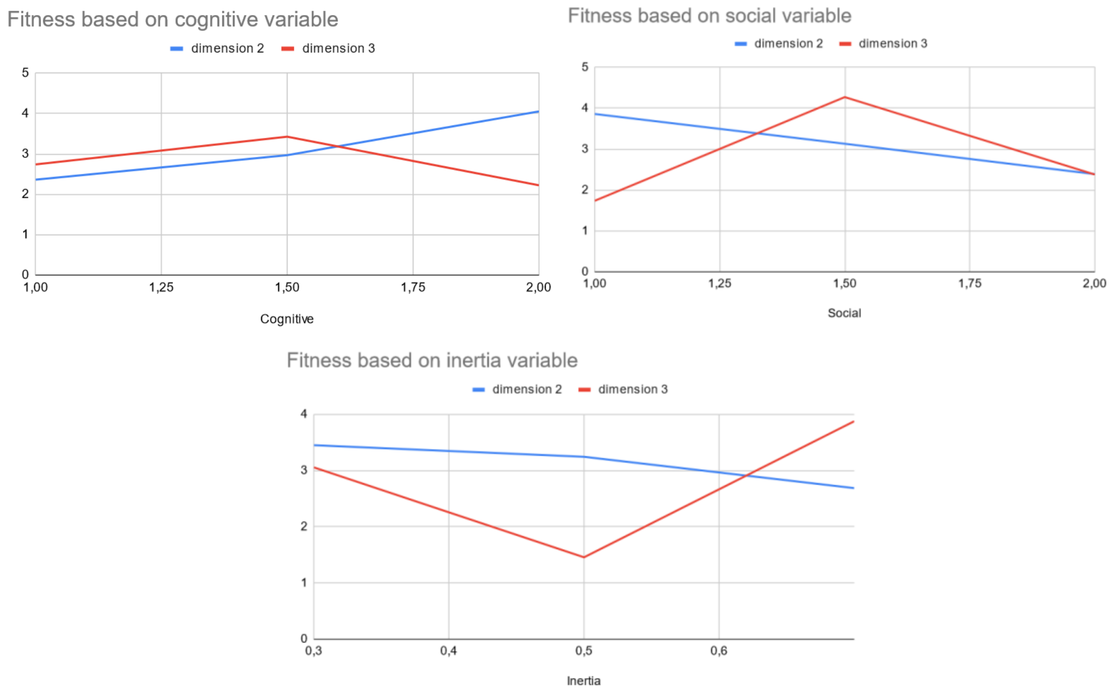

Figure 3. Performance of PSO algorithm in relation to its parameters.

In the Whale Swarm algorithm, the accuracy for ro0 of values 1, 2, 3 and the eta values of 0,005, 0,01, 0,05, 0,1, 0,5 was tested. It was computed for dimensions of 2 and 3. The best position and fitness was taken as the best out of 5 runs.

Algorithm	ro0	eta	Best Position	Best Fitness
WSO	1.0	0.005	[-0.0014480679365827204, 0.014566297705784444]	0.04709855061939949
WSO	1.0	0.01	[-0.8933196715714681, 0.04214445389943053]	2.696940143642881
WSO	1.0	0.05	[-7.024296988438002, 8.02313471428308]	15.603171466346252
WSO	1.0	0.1	[14.055283049007471, 6.066510063194564]	17.896003601627104
WSO	1.0	0.5	[-3.0345142113127395, 2.6654205045917934]	10.156403093628745
WSO	2.0	0.005	[1.3562462814129727e-09, 1.0237253395621077e-08]	2.9208326868257473e-08
WSO	2.0	0.01	[-3.3302168202361667e-07, -1.872388548946403e-06]	5.379123966253729e-06
WSO	2.0	0.05	[-2.9386367269543623, 2.926466760759795]	9.106507418251303
WSO	2.0	0.1	[-0.0783929887746142, 0.9930282570365115]	2.7860825696407905
WSO	2.0	0.5	[-11.997206111502509, -6.0033019941022]	17.000808269610033
WSO	3.0	0.005	[4.393180233397953e-13, -7.548015882107231e-15]	1.2438938767900254e-12
WSO	3.0	0.01	[2.081237447903163e-12, -2.308741770490534e-13]	5.922817791770285e-12
WSO	3.0	0.05	[-1.0717506255392357e-09, -1.6551920021064135e-10]	3.0673068529551983e-09
WSO	3.0	0.1	[0.008201061131035952, 0.9525093783421996]	2.581781619626011
WSO	3.0	0.5	[-3.985108185362421, -0.8499771541748089]	9.271281318633775

Figure 4. WSO algorithm performance with dimension 2.

Algorithm	ro0	eta	Best Position	Best Fitness
WSO	1.0	0.005	[-2.2244676488933384, -2.2656944097385465, 1.9977108453619596]	8.324861414319919
WSO	1.0	0.01	[9.792267582983126, 2.9429977920326693, -2.0024367421379217]	14.631828135142545
WSO	1.0	0.05	[10.970501378835129, -15.005671432076664, -3.980281348104259]	17.79585792832651
WSO	1.0	0.1	[4.947144692720261, -4.217379209067148, 14.032157826531387]	17.331778411210546
WSO	1.0	0.5	[-15.15718430682766, -5.759728585942298, -1.9738993225399923]	17.98145666671776
WSO	2.0	0.005	[-1.8887163239959635, 0.0020254559027878343, 0.023417460256264195]	4.132833035516153
WSO	2.0	0.01	[2.854487582192143, -0.1669247907950938, -1.035846048788218]	6.638137067416075
WSO	2.0	0.05	[-3.9005487841451183, -4.912219819391347, 4.851728362782749]	12.589721672989816
WSO	2.0	0.1	[-21.043753477694537, -24.90150022121827, 7.964847248566813]	19.800158240706683
WSO	2.0	0.5	[2.950273137951868, -8.996545344332588, -2.0648580776958454]	13.583368930184553
WSO	3.0	0.005	[5.837002284558006e-08, -4.779089050006365e-08, -1.0118172122136202e-09]	1.7423446907471885e-07
WSO	3.0	0.01	[3.5969212829547834e-07, -1.2411448342787544e-07, -4.531323773476653e-08]	8.849466621718705e-07
WSO	3.0	0.05	[6.948649164303426, -4.020443355677674, -1.0357021185228508]	12.22152186063497
WSO	3.0	0.1	[8.17140192580639, 2.9560238180661536, -0.028562719909044035]	13.14521297196195
WSO	3.0	0.5	[-13.946734535847966, -66.9384993717089, 31.962660931155398]	20.134985544490217

Figure 5. WSO algorithm performance with dimension 3.

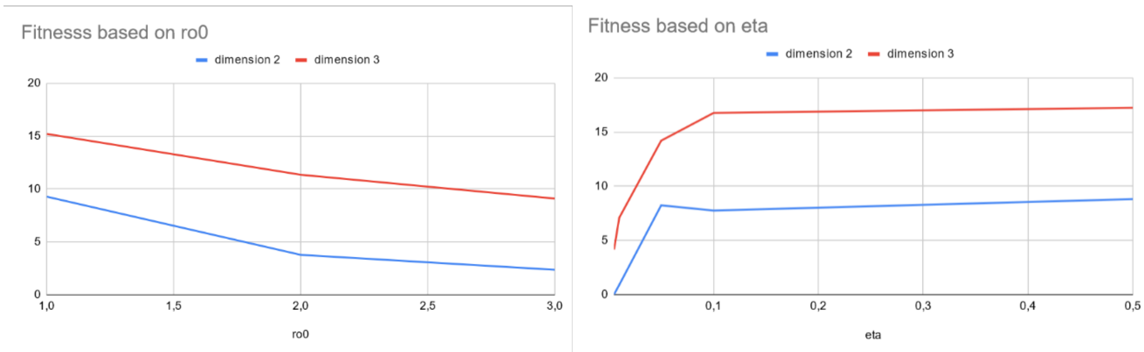

Figure 6. Average fitness of the WSO algorithm with given parameters.

With these results it is clearly visible that the Whale Swarm algorithm is performing a lot better than the basic Particle Swarm. It achieved results as low as 0 with 0's up to the twelfth decimal number. On average it took longer to compute the WSO algorithm but it was compensated in the quality of the solutions. The results supported the choice of WSO over PSO as the optimization algorithm for the problem.

On The graphs (Figure 6.) we can see what could be considered a predictable relation. The algorithm works better with higher importance of "ultrasounds" created by nearby agents. It also performs significantly better for less distortion in the messages from other agents. Based on this, in the final test the values ro0 = 3.0 and eta = 0.005 were used.

4. Application in classification task.

The task the optimization will be used for is the optimization of hyper- parameters in a neural network in order to achieve the best architecture for the traffic classification. The data for the task is taken from a Traffic-Net dataset form GitHub: https://github.com/OlafenwaMoses/Traffic-Net. The goal is to achieve the best classification of test photos into 4 categories: accident, fire, dense traffic and sparse traffic.

The parameters to optimize are the number of layers ([1, 2, 3, 4, 5, 6]) and the number of filters per layer ([4, 8, 16, 32, 64, 128]) to achieve an optimal architecture of the neural network, and the batch size ([32, 64 128]) to optimize the training. To cut down the computation time of all the test, the whole dataset was not used but only a sample of it. Instead of 900 images per category in the training phase, there were 300 picked randomly from each class.

After over 14 hours of running the optimizer fund the best solution to a given problem. It is 1 convolutional layer with 8 filters and a batch size of 64. The best accuracy achieved for the test set was 0.64. This is unfortunately a disappointing result. What is surprising the model worked very well, above 0,95 accuracy, for training and validation instances. What could be the reason for this is bad selection of options for parameters to optimize or inconsistencies in the data set.

```
model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3), padding = 'same'),
                                    tf.keras.layers.BatchNormalization(),
                                    tf.keras.layers.MaxPooling2D(2, 2),
                                    tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(4, activation='softmax')])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 7. Final best model found by WSO.

5. Conclusions

Overall, the Whale swarm algorithm did not achieve results as good as could be hoped for. It works very well in continuous spaces like Ackley function, but transferring this algorithm into finding categorical values brought on some issues. What was also not ideal is the computation time of the algorithm. It took a long time to compute the larger neural networks as well as the smaller batch sizes.

What could be done to improve this solution is dividing the search into smaller segments and not performing it in 3 dimensional space but just 2. It could be also altered in accordance to parameters that are searched. Instead of finding number of layers and filters, and applying the same value to all of them, it could be done as an architecture of 3 layers and the search to be centered around only filter sizes. This would allow for more flexibility among the layers, but also restrict the network to a preset number of layers that could not be optimal.

6. References

- Brodzicki, A.; Piekarski, M.; Jaworek-Korjakowska, J. The Whale Optimization Algorithm Approach for Deep Neural Networks. Sensors 2021,21, 8003. https://doi.org/10.3390/s21238003
- Designing Artificial Neural Network Using Particle Swarm Optimization: A Survey https://www.intechopen.com/chapters/82833
- Dixit, U., Mishra, A., Shukla, A. *et al.* Texture classification using convolutional neural network optimized with whale optimization algorithm. *SN Appl. Sci.* **1**, 655 (2019). https://doi.org/10.1007/s42452-019-0678-y
- Francisco Erivaldo Fernandes Junior, Gary G. Yen, Particle swarm optimization of deep neural networks architectures for image classification, Swarm and Evolutionary Computation, Volume 49, 2019, Pages 62-74, ISSN 2210-6502. https://doi.org/10.1016/j.swevo.2019.05.010
- T. Sinha, A. Haidar and B. Verma, "Particle Swarm Optimization Based Approach for Finding Optimal Values of Convolutional Neural Network Parameters," *2018 IEEE Congress on Evolutionary Computation (CEC)*, Rio de Janeiro, Brazil, 2018, pp. 1-6. https://ieeexplore.ieee.org/document/8477728
- Neural Network Training Using Particle Swarm Optimization, James McCaffery https://visualstudiomagazine.com/Articles/2013/12/01/Neural-Network-Training-Using-Particle-Swarm-Optimization.aspx?Page=1