Artificial Neural Networks and Training via Particle Swarm Optimisation

F21BC - Biologically Inspired Computation Coursework

Vanesa Caballero Alonso, Enrique Espinosa Chavez
Heriot-Watt University, Mathematical and Computer Sciences

Abstract

This document covers the development and results of various experiments involving a creation of different Artificial Neural Networks (ANNs). We focused on the creation of an ANN with the goal to detect possible customers leaving the bank based on certain attributes such as salary, tenure and geolocation. We trained the ANN using different methods: Tensorflow and Keras libraries with adam optimisators and a PSO. PSO training resulted in a long training time compared to the adam optimisator with better accuracy for a smaller dataset. We then developed a Multi-Layer Perceptron (MLP) for function approximation of linear, cubic, sine, hyperbolic tangent, xor and complex functions. We twitched the hyperparameters to try and identify which ones resulted in better accuracy for the training of the ANN.

I. Introduction

This piece of coursework is based on Artificial Neural Networks (ANNs), Multi-Layer Perceptron and Particle Swarm Optimization (PSO). All of them inspired by biological processes. Briefly explained, on one side, ANNs are a model based on the functionality of the human brain. It is composed by sets of nodes known as neurons. These neurons are connected between them and transmit input signals from the input layer to the output nodes in the output layer. Between the input and output layers it is found another set of layers, the hidden layers. As the ANN starts learning, the values of the weights and biases that affect each neuron is updated by propagation. Therefore, the output signal or value will be updated too. MLP is considered a type of neural network. ANNs are one of the most popular techniques for Machine Learning nowadays. On the other side, PSO is an optimization/search technique. It was first described by James Kennedy and Russel Eberhart and it is based on how a swarm of animals (bees, birds) moves when for example they are looking for food or protection. By using PSO in ANNs we introduce a swarm of searching particles in the learning process. These particles will move as a swarm trying to reach the minimum cost and help to optimize the ANN. It is worth to mention that, when using PSO some parameters are considered. For example, the number of particles or velocity, position of the neighbours of a particle, etc. In the next sections, it is explained the implementations made to an ANN and a PSO, the methods that were followed, the obtained results, discussion and finally the conclusions that arose from this assignment.

II. Program Development Rationale

In each of the exercises, different approaches were used to reach the desired results. In exercise 1, we decided to create an ANN for classification using a mock dataset of a bank. In it, we considered attributes of the customers: credit score, geography, gender, age, tenure, balance, number of products, whether they have a credit card or not, active membership and estimated salary. All of these attributes were used to try and predict if a customer is prone to leave the bank. For this, we needed to preprocess the data so we could work only with numbers, so we used a label encoder and a one hot encoder methods to solve the gender and geography attributes. Once we had this, we used Tensorflow and Keras libraries to build an ANN with configurable number of neurons and layers. Since Keras doesn't include activation functions: gaussian, cosine and null, we had to develop these functions manually to implement them in the code. Our first experiment was to train it using an adam optimizer and then compare it with a pyswarms PSO optimizer.

For completion of exercise 2, it has been used a basic PSO algorithm based on Engelbrecht's in order to check how a swarm of particles behaves (Miranda, 2017). For this purpose it was used the 'backend' module available in pyswarms. And a 'Star' topology was chosen. Thus, one particle is connected with the other particles. At the same time, the other particles are independent of each other (Ni and Deng, 2013).The basic PSO requires three steps. First: generation of the particles' velocities and positions, second: update the velocity and third: update the position. These basic three steps correspond in the code presented as 'Initialize the loop' present in LocalBestPSO.ipynb. In this exercise, it was demonstrated the capabilities for optimizing a function (sphere) included in pyswarms library. As per exercise requirements it was chosen a Local-Best PSO optimizer (Miranda, 2017). This optimizer enables each particle to compare to each other and not just the population best. For checking the effect of the swarm on the algorithm, it was plotted the cost history. This is how the 'Cost' is being reduced throughout the iterations or how well the swarm reaches the optimal solution. For demonstrating the

effect of the Star topology and The Local-Best PSO it has been included some animations. By using the 'Mesher' class, particles can be recognized with respect the sphere function. Moreover, by using the position history they can be placed in the 2D space. For the 3D space, the complexity increments as it needs to create a matrix containing the position in a 2D space and the fitness of the position.

In exercise 3, we used again the Keras library to build a different ANN for function approximation. The code has a condition to indicate which function and dataset will be loaded and approximated in the same code. We also included a plot of the actual results of the function and the approximation obtained by the ANN. This approach enabled us to twitch the hyperparameters in exercise 4 and come up with the best ones to minimize the Mean Squared Error (MSE) in each of the functions.

III. Methods

As mentioned at the beginning, this assessment is based on the techniques briefly described above and was developed in Python 3.8.0 code. First, an ANN was implemented. For this, it was used a dataset called Churn_Modelling.csv. The data were pre-processed, standardize and split into training set and testing set prior to run the ANN. For building the ANN, it was used modules from numpy, pandas, keras and tensorflow. Following exercise 1 requirements, the number of neurons in the input layer can be programmable and different activation functions (linear, sigmoid, gaussian, cosine, null and tanh) were introduced in the hidden layers. Then, the network was compiled using Adam optimizer and binary cross entropy as the loss function. Next, the ANN was trained for 100 epochs and accuracy was measure after each epoch. In the last steps, the model was tested and it obtained and accuracy over 86%.

Second, for the implementations of the PSO it was used mainly four libraries: numpy, matplotlib, IPython and pyswarms. First it was build the swarm with 50 particles. The PSO algorithm that was implemented was based on Engelbrecht's pseudocode. By following instructions of exercise 2, for making each particle to have a group of informants it was chosen Local Best PSO as the optimizer. Local Best PSO algorithm enables each particle to compare itself only to its nearest-neighbours. For helping to understand the effect of these algorithms the Cost History thoughtout the iterations was plotted. Also, in the code folder it is included some animations where the behaviour and tendency of the swarm can be visualized.

IV. Results

From the requirements of the CW, we made a list of our results in order of the instructions.

A. *Exercise 1*

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[1515   80]
 [ 194  211]]

0.863
```

Fig 1. Confusion matrix and overall score of ANN with linear, sigmoid and tanh activation functions

B. *Exercise 2*

In this section, we implemented a PSO algorithm based on Engelbrecht's pseudocode.

Swarm Size (n_particles)	10	20	50
Best Cost	best_cost=7.6e-8	best_cost=1.03e-7	Best_cost=4.15e-10

Table 1. Best costs reached by the swarms along the iterations

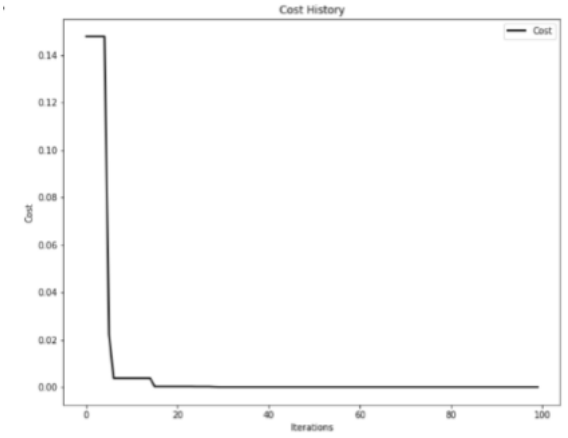

Fig 2. Cost history of 10 particle swarm

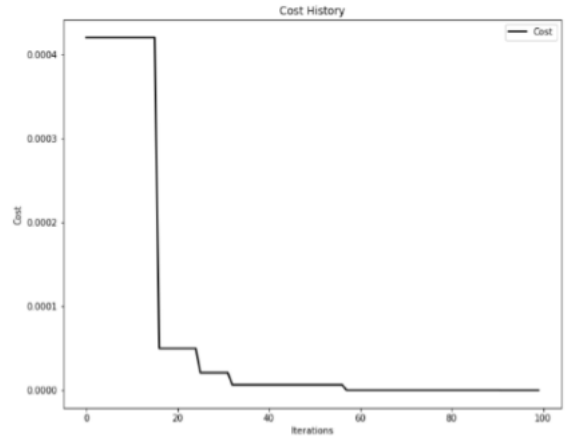

Fig 3. Cost history of 20 particle swarm

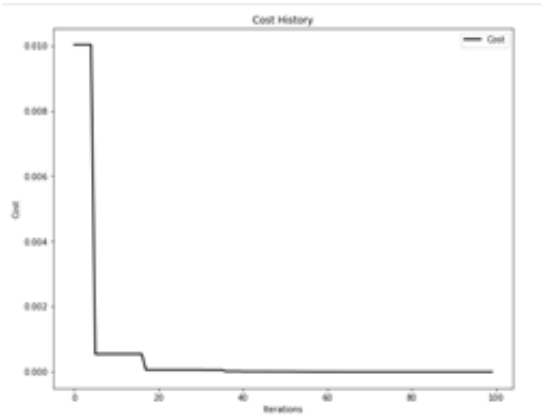

Fig 4. Cost history of 50 particle swarm

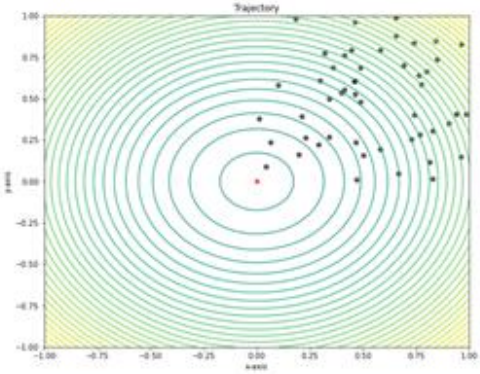

Fig 5. 2D Animation of swarm tendencies

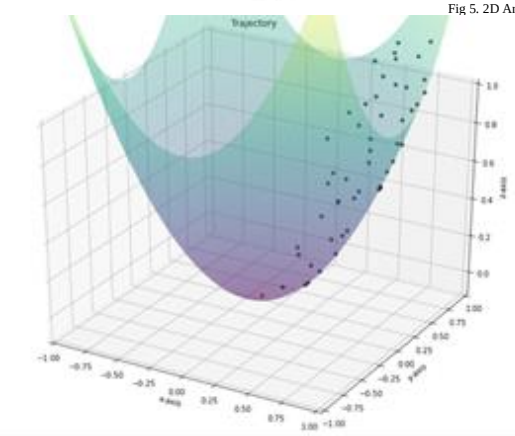

Fig 6. 3D Animation of swarm tendencies

C. *Exercise 3*

The MSE values obtained from a 1 and 2 hidden layer MLP were similar although the predicted values in the 2 hidden layer plot looked closer to the real values. When the number of hidden layers was incremented to 3, the predicted graph completely matched the real values. For the second experiment on hyperparameters it was explored the effect of the activation function used in the last hidden layer on the model's yield. See Table 2, 3 and Figure 7-12.

Number of hidden layers in MLP	1	2	3
MSE (MeanSquareError)	0.059	0.055	0.000

Table 2. Results of Mean Squared Errors with tanh function with different layers

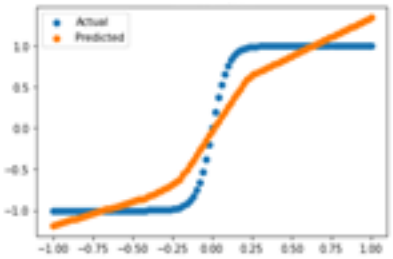

Fig 7. Tanh approximation with 1 hidden layer

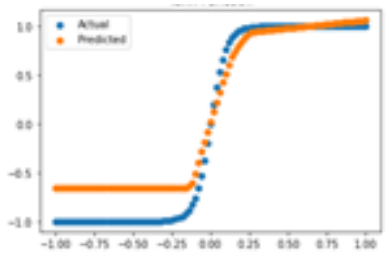

Fig 8. Tanh approximation with 2 hidden layers

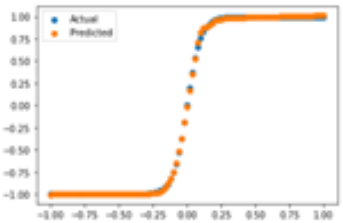

Fig 9. Tanh approximation with 3 hidden layers

Activation function (last hidden layer)	ReLu	Softmax	Sigmoid
MSE (MeanSquareError)	0.059	0.006	0.025

Table 3. MSE using different activation functions on the last layer

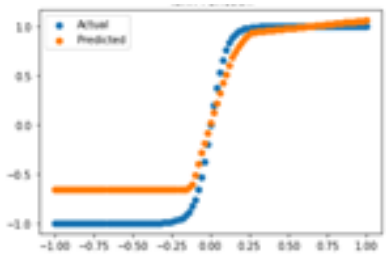

Fig 10. Tanh approximation with Rectified Linear activation function

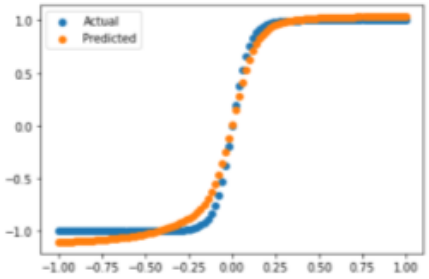

Fig 11. Tanh approximation with Soft-Max activation function

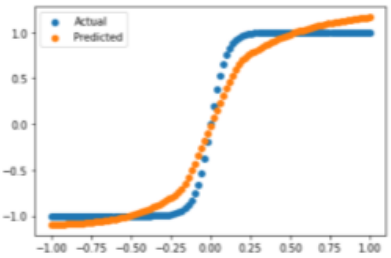

Fig 12. Tanh approximation with Sigmoid activation function

V. Discussion

In the sections above, ANNs and PSO were briefly described. However, these techniques are very complex in real world scenario. Although neural networks and PSO algorithms are very versatile and flexible, it is crucial to correctly tune their hyperparameters. Choosing the right (or wrong) set of hyperparameters may have a great impact on the results and the overall performance of the network.

For our experiments, to satisfy the curiosity of the team and have a taste of the effect of PSO on the ANN we built, we also considered the training of the PSO for the same dataset used in the first exercise with an adam optimizer. It was soon discovered that the number of iterations on the entire dataset was too much for our computers, so we tried to reduce the dataset to be able to run the training. We finally got to the desired result of 79% o accuracy which is the same for our adam optimizer with just one activation function of hyperbolic tangent. With such results, we can conclude that the keras library correctly implements the training modules in a more efficient way but also keeping the same accuracy using PSO training. We can therefore asume that an adam otpimiser will work as expected for a function approximator environment and therefore an available option in our next experiments.

In this assessment it was asked to investigate the effect of hyperparameters on the ability of PSO to optimize an ANN. Nevertheless, this task was unfulfilled, but it was explored the impact of certain hyperparameters, individually on the MLP and PSO models presented in this report. In first place, it was investigated the effect of the number of hidden layers on the performance of the MLP when doing function approximation. Although, it is known that shallow neural networks such as the MLP in this work can reach reasonable results (Géron, 2019), it was obvious the effect on performance when the number of hidden layers was incremented to 3. However, due to the small size of the dataset there was a risk of overfit the model (Witten et al., 2017). Thus, it was determined that 2 hidden layer MLP was going to be the standard for trying different activation functions in the last hidden layer. Witten et al., (2017) stated that ReLu activation function performs well in many different scenarios. For the current one, ReLu is the preference choice. However, the results in the previous experiments pointed out that there was still space for improvement. Specially for the datasets which contained negative values. This was confirmed after applying Softmax and Sigmoid activation functions on the last hidden layer of the MLP.

It was also explored the effect of the number of particles in a swarm. Inspired by the experiments carried out by Dhal et al., (2019). In their study, Dhal et al., concluded that in general a swarm of size 10-20 gave the best outcomes whereas for larger swarms the function deteriorates. However, it was pointed out that the selection of the population size is a complex task that depends on the algorithms, experiments and investigators experience. In exercise 2, the swarm size was initially set to 50. Then, it was worthy to try if the PSO implemented in this assignment was performing best with smaller swarm sizes. As Table 1 and Figures 2-4 show, the initial selection of a particle population of 50 accommodates well in the model and got the best results.

VI. Conclusions

ANN and PSO algorithms are powerful tools for machine learning but knowing the characteristics and nature of the problem where they are being applied is very important to choose the right parameters for obtaining optimal performance of these Machine Learning techniques. But not only knowledge can make the difference, but experimenters experience too. Finding the right hyperparameters for the right scenario is a challenge for ANNs and their training as we demonstrated in this set of experiments. As we say, the same hyperparameters work different for different functions in a function approximation scenario, while a classification scenario depends on the data structure and the number of desired outputs.

References

[1] Dhal, K.G., Das, A., Sahoo, S., Das, R., Das, S.(2019) ‘Measuring the curse of population size over swarm intelligence based algorithms’, Evolving systems. Springer Science and Business Media LLC. doi: 10.1007/s12530-019-09318-0.

[2] Engelbrecht, A.E. (2007) ‘Computational Intelligence: An Introduction’. Wiley

[3] Géron, A. (2019) Hands-on machine learning with Scikit-Learn, Keras and TensorFlow : Second edition. O’Reilly.

[4] Gomila, J.G. (2020) ‘Deep learning a-z' [ONLINE] Available at https://github.com/joanby/deeplearning-az [Accessed 24 October 2020]

[5] Kennedy, J., Eberhart, R. (1995) 'Particle swarm optimization', Neural Networks, 1995. Proceedings., IEEE International Conference

[6] Miranda, L.J.V., (2017). ‘Pyswarms’[ONLINE] Available at https://pyswarms.readthedocs.io/en/latest/index.html [Accessed 15 November 2020]

[7] Ni, Q., Deng, N. (2013) ‘A New Logistic Dynamic Particle Swarm Optimization Algorithm Based on Random Topology’, TheScientificWorld. United States: Hindawi Publishing Corporation, 2013, pp. 409167–8. doi: 10.1155/2013/409167.

[8] Omran, M., Engelbrecht, A., Salman, A.A. (2007) ‘Swarm Intelligence in Data Mining’ DOI: 10.1007/978-3-540-34956-3_6

[9] Sean ,L., 2013, Essentials of Metaheuristics, Lulu, second edition, available at http://cs.gmu.edu/~sean/book/metaheuristics/

[10] Witten, I.H., Frank, E., Hall, M.A., Holmes, G. (2011) Data Mining, San Francisco: Elsevier Science & Technology.