**HERIOT-WATT UNIVERSITY DUBAI**
**SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES**

**F21BC - Biologically Inspired Computation - 2023-24**

**Coursework**

| Student ID Number | Student Name | Programme |
| --- | --- | --- |
| H00441124 | Ilia | MS Data Science |

| | |
|---|---|
| **Course code and name:** | **F21BC - Biologically Inspired Computation** |
| **Type of assessment:** | **Group** |
| **Coursework Title:** | **F21BC - Biologically Inspired Computation** |
| **Student Name:** | **Ilia Svetlichnyi** |
| **Student ID Number:** | **H00441124** |

**Declaration of authorship. By signing this form:**

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

**Student Signature**:

**Date**: *17.11.2023*

## Introduction

The goal of this coursework is to delve deeper into the fundamentals of Deep Learning and gain practical experience in constructing an Artificial Neural Network (ANN) and Particle Swarm Optimization (PSO) from scratch. The culmination of this project is the integration of these two techniques to tackle a real-world problem: optimizing an ANN for a chosen dataset. By implementing a PSO algorithm equipped with informants, the project explores how different hyperparameters influence the ANN's performance. This practical application aims to provide insights into the effective combination of ANNs and PSO, contributing to an understanding of biologically-inspired computational techniques.

## Project Structure

**Brief Overview of Each Component**

**1. activation.py:** This module implements the activation functions essential for the ANN's operation. It includes functions like the linear, softmax, sigmoid, tanh, and ReLU, each playing a critical role in determining how neurons in the network activate and transmit signals. This set of functions allows the model for both regression and classification problems.

**2. ANNbuilder.py:** This file is responsible for constructing the architecture of the Artificial Neural Network. It allows for the creation of a multi-layered ANN, defining the number of neurons in each layer and linking them together to form a complete network structure.

**3. loss.py:** This module includes the implementation of loss functions. These functions are important in evaluating the performance of the ANN, providing a measure of the error between the network's predictions and the actual data.

**4. layer.py:** Central to the ANN architecture, this file defines the layer structure of the network. Each layer, composed of a set of neurons, is defined here.

**5. network.py:** This file has some main methods of neural network such as adding layers to the network, performing forward propagation to compute the output of the network, setting the weights of the network, and compute the total number of weights in the network.

**6. particle.py:** In this file, the representation of each particle within the PSO framework is detailed. It defines the properties and behaviors of particles as they navigate the solution space in search of optimal ANN parameters. Also, there is a function which updates the best informants position.

**7. pso.py:** This file contains the core implementation function of the Particle Swarm Optimization algorithm. Which initializes a swarm of particles and updates their positions based on the loss function and data, then returns global best position.

**8. swarm.py:** Focuses on the swarm behavior in PSO. It contains methods to initialize the swarm, assign informants to each particle, evaluate the global best position and loss for the swarm, and update the particles in the swarm.

**9. main files:** There are 3 main files each of them is focused on applying a model for a different problem: binary classification, multiclass one or a regression. All of them are produced in two forms: as a python file and as a Jupyter Notebook.These scripts connect all the components together, applying them to the dataset.

In the design of my ANN and PSO implementation, a key focus was on flexibility and adaptability. I adopted a generalized approach, enabling users to specify core parameters such as the architecture and activation functions for the ANN, and important parameters for PSO, like maximum iterations, population size, and loss function. This approach was chosen to ensure that the model could be easily adapted and optimized for a variety of problems, ranging from binary classification to multiclass problems or regression tasks.

## Experimental Investigation

In this section, I delve into the experimental aspect of my project, focusing on how various hyperparameters influence the performance of the ANN optimized by PSO. Experiments are designed to evaluate the impact of changes in PSO settings, ANN architecture, and activation functions. All the tests were applied on a binary classification problem and since my model gets almost 100% accuracy with almost all range of parameters it wasn’t possible to make tables or graphs which may represent statistical summary of performance. Thus, I decided to focus on the process of optimization and show how different parameters influence the speed of finding the global best loss. Here are the key areas which were explored:

**1. PSO Parameters Adjustment:** I experimented with changing the PSO parameters, particularly the population size and the number of iterations. By reducing these parameters, I anticipated a decrease in performance due to the reduced search capability and learning opportunities for the network. This experiment helps understand the trade-off between computational efficiency and the quality of results.

## Pop_size adjusting

### Pop_size = 25, Max_iterations = 300

```
Iteration 25/300, Current Best Loss: 0.01738071165466526
Iteration 50/300, Current Best Loss: 0.010939869289252599
Iteration 75/300, Current Best Loss: 0.009708817708079066
Iteration 100/300, Current Best Loss: 0.007464579811506142
Iteration 125/300, Current Best Loss: 0.006954394041006546
Iteration 150/300, Current Best Loss: 0.004595505478656214
Iteration 175/300, Current Best Loss: 0.0038563881931853302
Iteration 200/300, Current Best Loss: 0.003718773140565432
Iteration 225/300, Current Best Loss: 0.0037036123477027006
Iteration 250/300, Current Best Loss: 0.0028655077188973078
Iteration 275/300, Current Best Loss: 0.002839518964591415
Iteration 300/300, Current Best Loss: 0.0028354802142147354
Final Best Loss: 0.0028354802142147354 on Iteration: 296
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       148
           1       0.99      0.98      0.99       127

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

### Pop_size = 50, Max_iterations = 300

```
Iteration 25/300, Current Best Loss: 0.03006511062361988
Iteration 50/300, Current Best Loss: 0.004654219048031615
Iteration 75/300, Current Best Loss: 0.002589184775738911
Iteration 100/300, Current Best Loss: 7.150483924442828e-06
Iteration 125/300, Current Best Loss: 1.6912859061244844e-10
Iteration 150/300, Current Best Loss: 1.6912859061244844e-10
Iteration 175/300, Current Best Loss: 1.6912859061244844e-10
Iteration 200/300, Current Best Loss: 1.6518846255136701e-12
Iteration 225/300, Current Best Loss: 8.381927088513221e-13
Iteration 250/300, Current Best Loss: 2.7643623415401346e-13
Iteration 275/300, Current Best Loss: 2.2674668266009894e-13
Iteration 300/300, Current Best Loss: 9.35565370874332e-14
Final Best Loss: 9.35565370874332e-14 on Iteration: 298
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

### Pop_size = 100, Max_iterations = 300

```
Iteration 25/300, Current Best Loss: 0.032652044801709135
Iteration 50/300, Current Best Loss: 0.0158679457195031
Iteration 75/300, Current Best Loss: 0.008584640229388554
Iteration 100/300, Current Best Loss: 0.007536272647951948
Iteration 125/300, Current Best Loss: 0.001563490532607177
Iteration 150/300, Current Best Loss: 6.0264777868587215e-05
Iteration 175/300, Current Best Loss: 1.978401885782642e-06
Iteration 200/300, Current Best Loss: 1.6442137933969458e-07
Iteration 225/300, Current Best Loss: 1.438396464360521e-07
Iteration 250/300, Current Best Loss: 4.230094225440823e-08
Iteration 275/300, Current Best Loss: 2.98089392543898e-08
Iteration 300/300, Current Best Loss: 2.5941189679825935e-08
Final Best Loss: 2.5941189679825935e-08 on Iteration: 279
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

### Pop_size = 150, Max_iterations = 300

```
Iteration 25/300, Current Best Loss: 0.021975423095082592
Iteration 50/300, Current Best Loss: 0.0008457686172791785
Iteration 75/300, Current Best Loss: 1.3905468312668775e-05
Iteration 100/300, Current Best Loss: 1.2581233468312939e-05
Iteration 125/300, Current Best Loss: 9.99240695749345e-07
Iteration 150/300, Current Best Loss: 8.456649640206991e-07
Iteration 175/300, Current Best Loss: 7.757370433125447e-07
Iteration 200/300, Current Best Loss: 4.81126516741108e-07
Iteration 225/300, Current Best Loss: 2.841157237140188e-07
Iteration 250/300, Current Best Loss: 1.571221343689693e-07
Iteration 275/300, Current Best Loss: 6.005841286402352e-08
Iteration 300/300, Current Best Loss: 3.389481060794035e-08
Final Best Loss: 3.389481060794035e-08 on Iteration: 295
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

By changing the population size of a swarm in PSO we can see that it affects the finding of global best loss. For example, size 25 gave the lowest best loss among others, however, it was still enough to get almost 100% accuracy. 50 particles in the swarm is the optimal number which I got during testing and increasing population size further didn’t give performance gains.

## Max_iterations adjusting

### Pop_size = 50, Max_iterations = 50

```
Iteration 25/50, Current Best Loss: 0.03006511062361988
Iteration 50/50, Current Best Loss: 0.004654219048031615
Final Best Loss: 0.004654219048031615 on Iteration: 44
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       148
           1       0.98      1.00      0.99       127

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

### Pop_size = 50, Max_iterations = 100

```
Iteration 25/100, Current Best Loss: 0.03006511062361988
Iteration 50/100, Current Best Loss: 0.004654219048031615
Iteration 75/100, Current Best Loss: 0.002589184775738911
Iteration 100/100, Current Best Loss: 7.150483924442828e-06
Final Best Loss: 7.150483924442828e-06 on Iteration: 92
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

Moreover, the adjustment of iterations shed light on its role not just as a stopping criterion but also in fine-tuning the precision of the search. The influence of it is easily feasible that a low number of iterations might be not enough for finding best loss, but, on the other hand increasing the number of iterations more than 1000 is too much for simple datasets, thus emphasizing the need for a balanced approach in PSO parameterization.

**2. Influence of the coefficients in the velocity equation of PSO:** The aim was to assess how each coefficient influences the performance and learning dynamics of the PSO algorithm. My experiment involved tweaking the coefficients C1 (cognitive), C2 (social), and C3 (informant influence) in the PSO's velocity equation. These coefficients determine the extent to which particles in the swarm are influenced by their own best position, the swarm's global best position, and the best position of their informants, respectively.

- *The cognitive component (C1)* represents the particle's individual learning experience, influencing its movement towards its personal best position.
- *The social component (C2)* reflects the collective swarm intelligence, guiding particles towards the globally recognized best solution.
- *The informant influence (C3)* is a unique aspect of our implementation, where particles are influenced by a subset of peers, adding an additional layer of social learning.

**Best set of coefficients**
**w=0.5, c1=1, c2=2, c3=1**

```
Iteration 25/150, Current Best Loss: 0.03006511062361988
Iteration 50/150, Current Best Loss: 0.004654219048031615
Iteration 75/150, Current Best Loss: 0.002589184775738911
Iteration 100/150, Current Best Loss: 7.150483924442828e-06
Iteration 125/150, Current Best Loss: 1.6912859061244844e-10
Iteration 150/150, Current Best Loss: 1.6912859061244844e-10
Final Best Loss: 1.6912859061244844e-10 on Iteration: 118
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

**Changing cognitive component (C1)**
**w=0.5, c1=2, c2=2, c3=1**

```
Iteration 25/150, Current Best Loss: 0.033397188571094714
Iteration 50/150, Current Best Loss: 0.033397188571094714
Iteration 75/150, Current Best Loss: 0.033397188571094714
Iteration 100/150, Current Best Loss: 0.033397188571094714
Iteration 125/150, Current Best Loss: 0.033397188571094714
Iteration 150/150, Current Best Loss: 0.033397188571094714
Final Best Loss: 0.033397188571094714 on Iteration: 9
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       148
           1       0.96      0.98      0.97       127

    accuracy                           0.97       275
   macro avg       0.97      0.98      0.97       275
weighted avg       0.97      0.97      0.97       275
```

**Changing social component (C2)**
**w=0.5, c1=1, c3=3, c3=1**

```
Iteration 25/150, Current Best Loss: 0.07350397228583834
Iteration 50/150, Current Best Loss: 0.07350397228583834
Iteration 75/150, Current Best Loss: 0.07350397228583834
Iteration 100/150, Current Best Loss: 0.07350397228583834
Iteration 125/150, Current Best Loss: 0.07350397228583834
Iteration 150/150, Current Best Loss: 0.07350397228583834
Final Best Loss: 0.07350397228583834 on Iteration: 3
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       148
           1       0.90      0.86      0.88       127

    accuracy                           0.89       275
   macro avg       0.89      0.89      0.89       275
weighted avg       0.89      0.89      0.89       275
```

**Changing informants component (C3)**
**w=0.5, c1=1, c2=2, c3=2**

```
Iteration 25/150, Current Best Loss: 0.07999671347868585
Iteration 50/150, Current Best Loss: 0.07999671347868585
Iteration 75/150, Current Best Loss: 0.044667274384685506
Iteration 100/150, Current Best Loss: 0.044667274384685506
Iteration 125/150, Current Best Loss: 0.044667274384685506
Iteration 150/150, Current Best Loss: 0.044667274384685506
Final Best Loss: 0.044667274384685506 on Iteration: 70
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       148
           1       0.99      0.89      0.94       127

    accuracy                           0.95       275
   macro avg       0.95      0.94      0.94       275
weighted avg       0.95      0.95      0.95       275
```

My tests showed that using the coefficients *[w=0.5, c1=1, c2=2, c3=1]* worked the best for the dataset. When I increased the cognitive component, C1, to 2, the PSO got stuck early on and couldn't find better solutions, which wasn't ideal. Tweaking the social part, C2, taught me that going higher than 2 makes the PSO get stuck too, but dropping it to 0.5 was still pretty okay, just not as good as 2. Lastly, playing with the informant part, C3, showed that higher or lower than 1 either slows down the PSO or makes it less effective. What this tells us is that we have to be careful with how we set these numbers because leaning too much on any one of them can throw off our PSO's ability to do its job well.

**3. Varying ANN Architectures and Activation Functions:** I tested different architectures and sets of activation functions to measure their effect on the network's performance. For instance, using an architecture like [6, 8, 1] with activation functions such as ['relu', ‘logistic’]. This experiment was crucial to determine how the depth of the network and the type of non-linear transformations applied by activation functions contribute to the learning capability and accuracy of the ANN. At first I tried to just change an architecture while set of activation functions remained constant, after the same was done with activations

**Architectures**

**[4, 8, 1]**

```
Iteration 25/100, Current Best Loss: 0.03006511062361988
Iteration 50/100, Current Best Loss: 0.004654219048031615
Iteration 75/100, Current Best Loss: 0.002589184775738911
Iteration 100/100, Current Best Loss: 7.150483924442828e-06
Final Best Loss: 7.150483924442828e-06 on Iteration: 92
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

**[4, 16, 1]**

```
Iteration 25/100, Current Best Loss: 0.016413605503944446
Iteration 50/100, Current Best Loss: 0.0066310519473096645
Iteration 75/100, Current Best Loss: 0.0066310519473096645
Iteration 100/100, Current Best Loss: 0.002263248705165101
Final Best Loss: 0.002263248705165101 on Iteration: 88
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

**[4, 100, 1]**

```
Iteration 25/100, Current Best Loss: 0.030114713111481048
Iteration 50/100, Current Best Loss: 0.012762078396150504
Iteration 75/100, Current Best Loss: 0.012762078396150504
Iteration 100/100, Current Best Loss: 0.012762078396150504
Final Best Loss: 0.012762078396150504 on Iteration: 33
              precision    recall  f1-score   support

           0       0.99      0.97      0.98       148
           1       0.97      0.99      0.98       127

    accuracy                           0.98       275
   macro avg       0.98      0.98      0.98       275
weighted avg       0.98      0.98      0.98       275
```

**[4, 8, 8, 1]**

```
Iteration 25/100, Current Best Loss: 0.07748404740361126
Iteration 50/100, Current Best Loss: 0.03555150410209663
Iteration 75/100, Current Best Loss: 0.03190519598906107
Iteration 100/100, Current Best Loss: 0.004557885141294439
Final Best Loss: 0.004557885141294439 on Iteration: 90
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       148
           1       0.98      0.99      0.98       127

    accuracy                           0.99       275
   macro avg       0.98      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

Testing different ANN architectures showed that a basic setup with *[4, 8, 1]* layers performed the best, achieving perfect scores. Increasing the size to [4, 16, 1] kept the high performance, suggesting that the model scales well, yet the global best loss was lower. However, going much larger to [4, 100, 1] didn't add much benefit, it even gave the lowest loss and scores, implying there's a limit to the advantage of more neurons. Adding one more hidden layer didn't give any improvements in comparison with *[4, 8, 1]* structure, which means that for this specific dataset there is no need for more hidden layers than one, however, this might not be the case for more complicated datasets. Overall, the tests showed our ANN is flexible and can manage different architectures, but the best setup depends on the specific problem's complexity.

**Activation Functions**

**[relu, logistic]**

```
Iteration 25/150, Current Best Loss: 0.03006511062361988
Iteration 50/150, Current Best Loss: 0.004654219048031615
Iteration 75/150, Current Best Loss: 0.002589184775738911
Iteration 100/150, Current Best Loss: 7.150483924442828e-06
Iteration 125/150, Current Best Loss: 1.6912859061244844e-10
Iteration 150/150, Current Best Loss: 1.6912859061244844e-10
Final Best Loss: 1.6912859061244844e-10 on Iteration: 118
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       148
           1       1.00      1.00      1.00       127

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275
```

**[tanh, logistic]**

```
Iteration 25/150, Current Best Loss: 0.04316271358733911
Iteration 50/150, Current Best Loss: 0.010483263205379923
Iteration 75/150, Current Best Loss: 0.003479142854626974
Iteration 100/150, Current Best Loss: 0.0008518248361304217
Iteration 125/150, Current Best Loss: 0.0002614318372677715
Iteration 150/150, Current Best Loss: 0.00017709832250341416
Final Best Loss: 0.00017709832250341416 on Iteration: 144
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       148
           1       1.00      0.98      0.99       127

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

In testing activation functions, I compared combinations of 'relu' with 'logistic' and 'tanh' with 'logistic'. The 'relu', 'logistic' combo led to a final best loss that was considerably higher than the 'tanh', 'logistic' pair, which achieved a much lower final best loss.

## Discussion

My exploration of the ANN-PSO system demonstrates the robustness of the ANN model through a variety of architectural choices, consistently achieving high accuracy. The experiments highlighted the sensitivity of the PSO algorithm to its hyperparameters, with particular accent on the trade-off coefficients between personal experience and social sharing of information.

In particular, the impact of the information influence coefficient suggests that placing too much importance on peer information can hinder the optimization process.

The complexity of these PSO parameters represents a complex but important aspect of ANN optimization, requiring careful consideration to avoid early meeting to local minima and ensure searchability to the overall swarm.

## Conclusion

The performed experiments clearly demonstrated that the success of the ANN-PSO system lies in the delicate adjustment of its parameters. A set of optimal PSO coefficients and a well-structured ANN architecture are instrumental in achieving near-optimal performance on the used data set.

While the results are promising, I see limitations such as potential overfitting and degraded performance with networks that are too large or too deep.

Future research may benefit from applying these findings to more diverse and complex datasets, exploring alternative neural network models, and studying adaptive PSO mechanisms that can further improve the balance between exploration and exploitation.

## References


1. Kinsley, H., & Kukieła, D. (2020). Neural Networks from Scratch in Python. Harrison Kinsley.
2. Interactive Particle Swarm Optimisation Dashboard from Scratch in Python [https://www.scottcondron.com/jupyter/optimisation/visualisation/2020/08/02/interactive-particle-swarm-optimisation-from-scratch-in-python.html]
3. Particle Swarm Optimization from Scratch with Python [https://nathanrooy.github.io/posts/2016-08-17/simple-particle-swarm-optimization-with-python/]