PROJECT REPORT

COSC-4F90

BROCK UNIVERSITY

FACULTY OF MATHEMATICS AND SCIENCE

MCPSO and DCPSO with Factorized Node and Layer Decomposition for Large Scale Neural Networks

Author:
Rikveet Singh Hayer

Supervisor:
Dr. Ombuki Berman

September 22, 2022

TABLE OF CONTENTS

TITLE PAGE **1**

TABLE OF CONTENTS **1**

1 Abstract **2**

1.1 Keywords 2

2 Introduction **3**

3 Background **4**

3.1 Neural Networks 4

3.1.1 Background 4

3.1.2 Optimization 4

3.1.3 Mini Batch Training 6

3.2 Particle Swarm Optimization 7

3.3 Cooperative Particle Swarm Optimization 9

3.4 Neural Network Decomposition types for CPSO and PSO 12

3.4.1 Layer Decomposition 12

3.4.2 Node Decomposition 13

3.5 Merge-CPSO 14

3.6 Decompose-CPSO 15

4 Experimental Setup **16**

4.1 Data-sets 18

4.2 Performance Measurements 18

5 Results **19**

5.1 Pso, Mcpso and Dcpso node decomposition variants 19

5.2 Pso, Mcpso and Dcpso factorized node decomposition variants 20

5.3 Pso, Mcpso and Dcpso layer decomposition variants 21

5.4 Pso, Mcpso and Dcpso factorized layer decomposition variants 22

5.5 Mse over iterations 23

5.6 Discussion 24

5.7 Conclusion 25

MCPSO and DCPSO with Factorized Node and Layer Decomposition for Large Scale Neural Networks

1. Abstract

Optimizing neural networks(nn) using particle swarm optimization(pso) or cooperative particle swarm optimization(cpso) has been done in many previous studies. This study introduces two new methods of cpso, merge-cpso and decompose-cpso[1] by replacing the base cpso used for the optimization of decomposed neural networks. Solving high-dimensional problems with cooperative particle swarm optimization can introduce the issue of saturation. It can directly affect the resulting solution of a problem by moving the particles arbitrarily. A previous study[2] used random regrouping and factorization in nn decomposition to observe the effects on the performance of training neural networks with cpso using the decomposition methods discussed earlier. This study will observe the effects of using mcpso and dcpso with factorized, and non-factorized decompositions of the neural network. The experiment performed in this study was done over 5 data sets with dimensions ranging from 35 to 827 to compare the performance of optimization algorithms and decompositions. Using the two new algorithms Mcpso and Dcpso this study has found a slight improvement over the base cpso.

1.1. Keywords

Feed-Forward neural network, Vector, N-Dimensional space, Particle Swarm Optimization, Cooperative Particle Swarm Optimization, Factorization, Variable interdependence, Saturation, Merge-Cpso, Decompose-Cpso.

2. Introduction

A neural network is a mathematical model based on the neurons/nerve cells found in the brain. As nerve cells are connected to each other sharing information about external stimuli. The neural network has a set of nodes connected to each other with the numerical data flowing from the input(stimuli) to the output(response) nodes. This mathematical model makes it capable of learning patterns in a given data set, for example predicting whether someone has cancer or not based on a couple of numerical data points recorded from the patient. This capability of recognizing patterns allows its application to various fields such as finance, healthcare and media bringing them to a new level. Traditionally the neural network is trained using Back-propagation, an algorithm that improves the connections of nodes(neurons) based on the error of the model's prediction for a given input. A number of previous studies have successfully applied particle swarm optimization(pso)/cooperative pso instead of back-propagation [3, 4, 5]. Compared to Backpropagation a supervised learning algorithm, pso is a stochastic population search-based algorithm inspired by nature. An algorithm is known as stochastic if it uses randomness to find an optimal solution to a given problem. Instead of improving the node's connections using the error at the output layer, it uses a population of particles that travel in n-dimensional search space. The best position is the one with the least error in its predicted and expected output for a given input. The particles share this position and move towards it hence optimizing the connections between the nodes. In real-life applications, the size of the data sets are much larger introducing the problem of dimensionality to pso due to which it is not able to optimize the neural network. Cpso solves this problem by breaking down the large neural network into smaller sets which can be optimized by individual pso swarms cooperatively. Another issue discussed in the abstract is known as saturation which causes the particles to move randomly in the search space thus not finding a new best location and stagnating the solution to a local minimum. The previous study successfully solved this issue by using unbounded activation function, weight decay and velocity clamping which have also been used in this study.

3. Background

3.1. Neural Networks

3.1.1. Background

A neural network consists of nodes, these nodes are connected using weights and biases shown in figure 3.1. The first layer of the model is the input layer, the layers between the first and last layer are known as hidden layers. The last layer is known as the output layer. Each node in the hidden layer and the output layer has distinct weights and biases connecting it to the nodes in the previous layer. The nodes in the input layer store the values of the characteristics on which the model is being trained. The input of all nodes in hidden layer and output layer as calculated as follows $(\sum_{i=0}^{n} W_{i,j} * O_i) + b_j$, w is the weight connecting i^{th} node in the previous layer to the j^{th} node in the current layer, O is the output of the i^{th} node in the previous layer and b is the bias for j^{th} node in the current layer. The output of the node is decided by using an activation function $f(x)$, where x is the input of the node. The weight and biases help the model map the input and its expected output for the overall model. In the case of this study, there is only 1 hidden layer [2]. The weights and biases are trained on predefined inputs and outputs this method of training is also known as supervised learning. By learning from known data the model can create a generalized map which helps it to predict unknown data.

3.1.2. Optimization

In terms of cpso, the goal is to reduce the difference between the model's prediction and the expected output for a given input. To do so the weights and biases must be optimized. Hence the weights and biases of a neural network are treated as the optimization problem for cpso with the overall dimension as the total number of weights and biases [2] present in the neural network. For a 3-layered network weights and biases would be (I + 1) * H + (H + 1) * O. Where I is the number of nodes in the input layer, H is the number of nodes

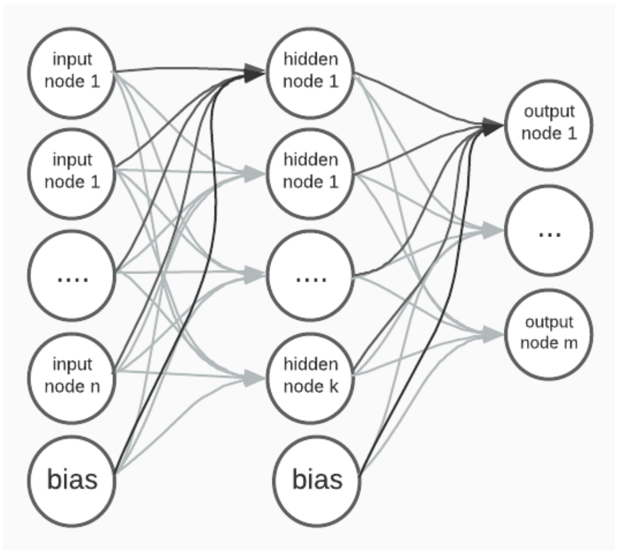


Figure 3.1.1.1: Visualization of a CPSO

in the hidden layer and O is the number of nodes in the output layer.

- Objective function: The Mean Squared Error of the neural network can be used as the fitness value for the CPSO.

$$MSE = \frac{\sum_{p=1}^{P}\sum_{k=1}^{K}(t_{kp} - o_{kp})^2}{P * K} \tag{3.1.2.1}$$

  It represents the mean difference between the model's prediction and the training data set's expected output. P is the total number of training examples, K is the total number of nodes in the output layer, t is the model's output and o is the expected output for a given node.

- As discussed in the introduction cpso and pso suffer from saturation caused by bounded activation functions [2, 6, 7]. It happens in the first couple of iterations causing the weight adjustments to stagnate [6]. A bounded activation function is a function in which for a given input the output is in a defined range. For example the sigmoid function's range is $(0,1)$. This causes the gradient to become shallow reducing the influence of the input of a node on its output [2]. Which further causes the position vector's magnitude to change randomly and the cpso/pso gets stuck at local minima because no new best position is found. To avoid this a couple of solutions are implemented [2].

- Non-bounded Activation Function: The Rectified Linear function(Relu) is used as the activation function [2]. $output = max(0, output)$. This removes the +ve bound reducing the saturation [2].
- Using Relu introduces a new problem, for all inputs in range $(-\infty, 0]$ the output will be 0. It causes the weights to become large increasing saturation. To make sure the magnitude of weights stays in the optimal range Weight decay is added to the output of the objective function.

$$WD = \lambda * \Sigma_{i=1}^{W} w_i^2 \tag{3.1.2.2}$$

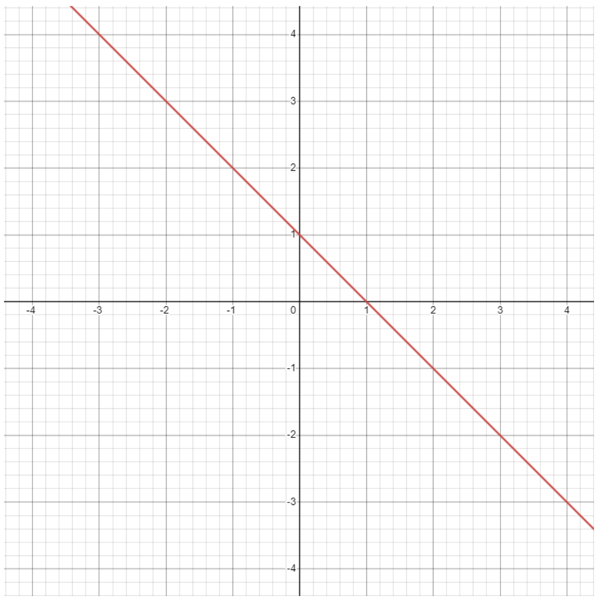
  [2] It is the sum of all the squared weights and biases. To regulate the decay a constant value λ is used [2].
- Finally, due to the unbounded search space, the magnitude of the velocity can become large causing the particles to go out of optimal search space. the velocity is clamped to avoid this problem. This issue is further discussed in the section Particle roaming3.3.0.1

3.1.3. Mini Batch Training

As mse is the objective function for cpso and pso, this function is prone to be called many times during training due to which iterating over larger data sets can become slower. Hence, mini-batch training is implemented. A mini-batch is a set of randomly selected training examples from the overall set. In this methodology, each time a position is evaluated a new mini-batch is created [8]. Pso/Cpso stores a particle's personal best position and fitness to calculate the velocity for the next iteration. With the current method of mini-batch training, a particle's best position will be evaluated on a different batch compared to the current batch. Which makes the comparison of previous fitness and current fitness invalid as they are calculated on different batches. To solve this, the update method to the particle's best position has been modified in the following way. To update the particle's personal best fitness, the current best position is re-evaluated on the current mini-batch and the particle's current position is evaluated on the current mini-batch if the fitness is less than the best position's fitness, the particle's best position is updated. Global best positions are evaluated on the whole training data set and if the fitness using the current position is improved, the context vector is updated. This change was used in a previous paper [2] to balance the performance gained by additional functional evaluations with a decrease in performance due to fewer iterations.

3.2. Particle Swarm Optimization

Particle swarm optimization is a population search-based algorithm. It is derived from the swarm methodology[9] found in flocks of animals such as birds. In the case of a flock of birds the problem can be the search for food. When one bird finds a source of food the rest of the flock tends to follow. As multiple birds try to solve this problem at the same time. The required time to solve it is reduced by a large amount. Similarly, this algorithm optimizes a problem by generating multiple particles as a swarm in n-dimensional space. These particles search the n-dimensional space to find the best solution. The particles are initialized at a random position in a predetermined range. A position of a particle in the pso is a vector of size n and the range of the dimensions of the search space belongs to Real numbers $(-\infty,\infty)$. For example in the problem x+y=1 the dimension would be of size 2 (x_0,x_1).



The graph is a visual representation of 2d search space. The red line in the graph represents the possible solution positions in the 2-dimensional search space. The search space can be constricted to a smaller range to ensure convergence. To test if the particle's current position is the best possible position, an objective function $fitness = f(x_0,\ x_1,\ ...,x_n)$ is used to find the particle's fitness where $\{x_0,\ x_1,\ ...,x_n\}$ are it's position coordinates and domain of the function belongs to **R**. In the case, of our example, the particle's fitness would be the $1-(x+y)$, where the lower the fitness the better the position.

Algorithm 1 PSO algorithm

```
for t = 0; t < Total Iterations; i++ do
    for Particle P in Swarm do
        fitness = $f(x_1,x_2,...,x_n)$
        if fitness < P's Best Fitness then
            P's best position vector = current position vector
            P's best fitness = fitness
        end if
        if fitness < Global Best Fitness then
            Global best position vector = current position vector
            Global best fitness = fitness
        end if
    end for
    for Particle P in Swarm do
        $v_{x_i,\ t+1} = (\omega * v_{x_i,\ t}) + (c_1 * r_1 * (\hat{p}_{x_i,\ t} - p_{x_i,\ t})) + (c_2 * r_2 * (\hat{g}_{x_i,\ t} - p_{x_i,\ t}))$ - (1)
        $p_{x_i,\ t+1} = p_{x_i,\ t} + v_{x_i,\ t+1}$ - (2)
    end for
end for
```

To search for the optimal position particle has a velocity. This velocity is updated every iteration using equation (1) in Algorithm 1.

- ω: Inertia factor. It is used to control the influence of a particle's previous velocity on the particle's next position.
- $c_1,\ c_2$: Cognitive and Social factors. These factors are used to control the influence of the particle's personal best position and global best position on the particle's next position.

- r_1, r_2: n-dimensional vectors who's each value is a random value between [0, 1], used to introduce randomness to a particle's next position.
- t, x_i, v, $\hat{p}$, $\hat{g}$, p: t is the current iteration, x_i is the particle's i^{th} coordinate value in the n-dimensional vector, v is the current velocity vector, $\hat{p}$ is the particle's personal best position vector, $\hat{g}$ is the global best position vector and p is the particle's current position vector.

Finally, the particle's position vector is updated using equation (2) in Algorithm 1. The new position is the sum of the previous position and current velocity. Over iterations, the best global position is updated when the fitness of the particle's current position vector is lower than the global vector. The update is the same for the particle's best position vector. These updates can take place in two ways synchronous and asynchronous. In sync, the global position is updated after all the personal best positions are updated whereas in async the global position is updated as soon as the local position is updated. As discussed in the introduction pso suffers from the problem known as two steps forward, and one step back. With an increase in the size of the problem the search space dimension also increases. As the particle and global best position's fitness depends on the current position vector the increase in the size of the positional vector can have some dimensions improve while others degrade[10]. Even though the position's fitness improves for the current iteration, the overall solution can degrade and get stuck on local minima or maxima.

3.3. Cooperative Particle Swarm Optimization

Cooperative PSO solves the problem of two steps forward, and one step back. It solves this by reducing the size of the problem solved by a particle. Instead of a single swarm with the particle's position as a vector of size n. Cpso splits the problem into n sub-swarms with the particle's position vector of size 1 [10]. Each sub-swarm has the same amount of particles as a pso swarm would have. The overall solution is produced by combining the sub-swarms into a single vector also known as the context vector. The size of the context vector would be n, in an n-dimensional problem. Each sub-swarm updates the decomposed problem and the global best position in pso is the sub-swarm's

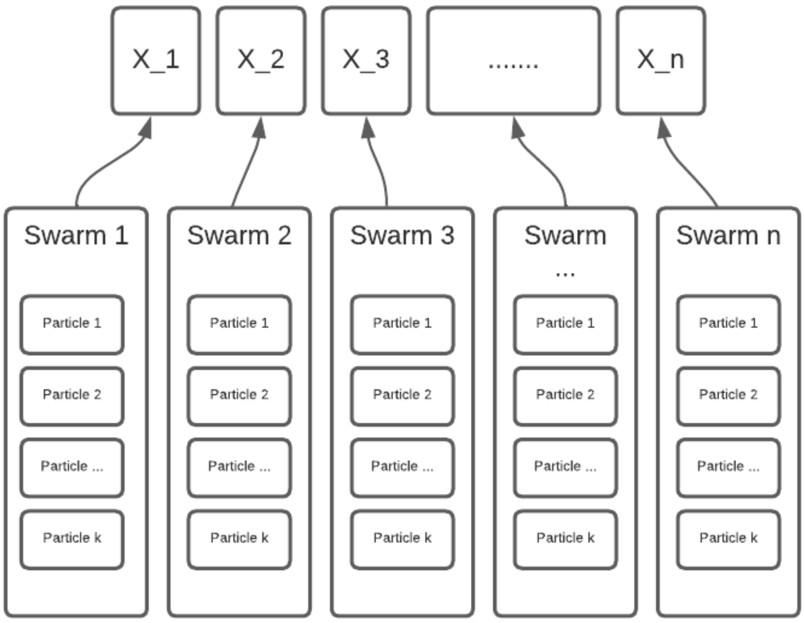


Figure 3.3.0.1: Visualization of a Neural Network

best position in Cpso. The context vector is updated once the fitness of the context vector improves by replacing the current coordinate value with the updated swarm's best position. Even after decomposing the problem into smaller sub-swarms the curse of dimensionality persists for larger dimensions. The two main contributors to this are.

- Particle roaming [2, 11]: With the increase in dimensions, the particles tend to go out of optimal search space in the first couple of iterations. This is due to an increase in velocity because of the increased dimensions. Velocity clamping can be used to prevent this rapid growth[2, 12], it is the max amount by which a particle can step. Velocity for sub-swarm s, particle k and dimension i can be found by the following formula.

$$velocity = \begin{cases} v_{s,\,k,\,i} & if\ -max\_velocity < v_{s,\,k,\,i} < max\_velocity \\ max\_velocity & if\ max\_velocity < v_{s,\,k,\,i} \\ -max\_velocity & if\ v_{s,\,k,\,i} < -max\_velocity \end{cases} \tag{3.3.0.1}$$

[11]

- Variable interdependence: When the problem has multiple dependent dimensions the context vector may only improve when all the dependent dimensions improve simultaneously. If not it can stop the swarms from reaching the optimal solution as individual vectors will depend on others to improve and vice-versa [2, 13]. Factorized decomposition optimizes the same dependent variables in multiple sub-swarms. Optimizing the same variable causes multiple sub-swarms to overlap improving the interdependent variables together. For example in problem P with the variables A, B and C. Where A and B overlap, B and C overlap. Both sub-swarms optimizing A and C will also optimize B [2]. In terms of a neural network, a node's output dependents on its input weights and bias.

3.4. Neural Network Decomposition types for CPSO and PSO

3.4.1. Layer Decomposition

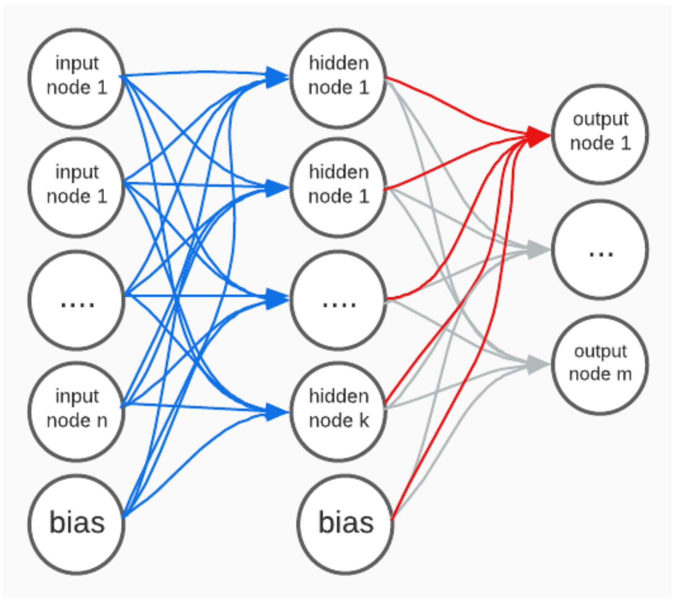


Figure 3.4.1.1: Layer Decomposition

The layer-based decomposition decomposes the neural network into L-1 sub-swarms where L is the number of layers. It optimizes all layers except the input layer. In the case of this study, layer decomposition will create two sub-swarms. All the inputs and biases for a layer will be optimised by a single swarm as shown in the figure 3.4.1.1with the weights and biases marked in blue. The size of the hidden layer's sub-swarm will be (I+1)*H, and the size of the output layer's sub-swarm will be (H+1)*O [2, 5].

A factorized version of layer decomposition will create O sub-swarms where O is the number of output nodes. For each output node, all the weights and biases used to calculate the input of the node are optimized in a single sub-swarm. As shown in the figure 3.4.1.1with all the weights and biases marked in red and blue. The dimensions of all the sub-swarms will be (I+1)*H + (H+1). This creates an overlap as all the weights and biases before the last hidden layer are optimized by all the sub-swarms.

3.4.2. Node Decomposition

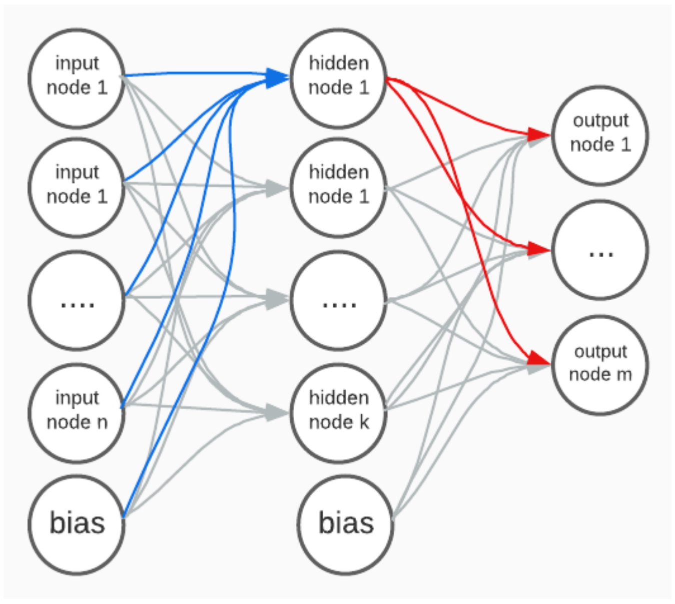


Figure 3.4.2.1: Node Decomposition

The node-based decomposition decomposes the neural network into H+O sub-swarms where H and O are the numbers of nodes in the hidden layer and output layer respectively. It optimizes all the input weights and biases for a given node as highlighted in blue in the figure 3.4.2.1. All nodes in the hidden layer dimension would be (I+1)*H and all the nodes in the output layer would be (H+1)*O [2, 5].

The factorized version of node decomposition will still create the same number of sub-swarms but increase the dimensions. For a node in the hidden layer all the inputs weights, biases and output weights are optimized by a single sub-swarm as highlighted in red and blue in the figure 3.4.2.1. In this case, a node in the hidden layer would also optimize the input of output nodes or the next hidden layer node creating an overlap between sub-swarms. In comparison to Layer decomposition with an increase in the scale of a neural network, the number of sub-swarms and dimensions of a sub-swarm increase at a lower rate.

3.5. Merge-CPSO

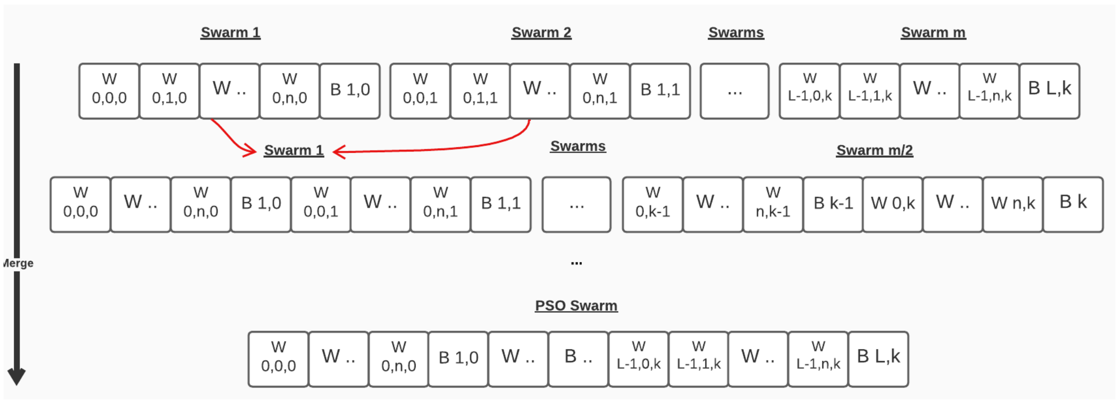


Figure 3.5.0.1: MCPSO

Merge cooperative particle swarm optimization utilizes the exploitation and exploration of a Cpso merging into a Vanilla Pso [1]. The algorithm starts with a Cpso and over iterations, the swarms are merged till only one swarm is left which is a Pso. The figure 3.5.0.1 is an example of node-based decomposition for Neural Networks where $W_{l,i,j}$, $B_{l,j}$ with l as the layer number, i as the node in the previous layer and j is the node in the current layer which is being optimized by the Swarm. When the merging condition is satisfied a pair of swarms are merged and the particle's position and velocity are reset, only the swarm's best values are carried to the new swarm. While merging an overlap for best values can occur. For example, in Factorized node decomposition a swarm for a node in the hidden layer will also have inputs for nodes in the output layer. In the case of overlap, the values are compared and the best value is carried to the merged swarm. Finally, during the merge, the decision variables are shuffled to ensure a variety of the variable interaction.[1].

In the previous research [1], the merging condition becomes valid after a certain number of iterations found by the following formula.

$$n_f = \frac{n_t}{1 + \frac{\log(n)}{\log(n_r)}} \tag{3.5.0.1}$$

In the equation 3.5.0.1

- n_f is the number of iterations in-between merge.
- n_t is the total number of iterations.
- n is the total number of dimensions. For a neural network, it will be the total number of weights and biases in the whole network (I + 1) * H + (H + 1) * O.
- n_r is the number of sub-swarms merged at each join. For this study, this number is 2.

[1]

3.6. Decompose-CPSO

Decomposition cooperative particle swarm optimization utilizes the exploitation and exploration of a Vanilla Pso decomposing into a Cpso [1]. This algorithm is modified for Node and Layer decomposition for the Neural Network optimization. The decomposition condition is the same as the merging condition. Instead of decomposing the Pso over a certain number of iterations, it is decomposed when the condition is valid for the first time. The Best global values in the Pso are carried over to the respective decomposed swarms. In terms of Factorized decomposition, the same best value can be used for overlapping weights. Furthermore, n_r used in the equation 3.5.0.1 is the total number of swarms created after the decomposition.

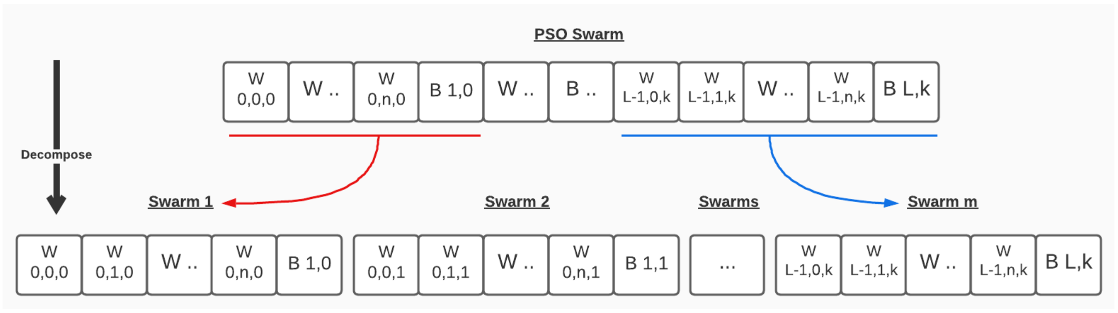


Figure 3.6.0.1: MCPSO

Figure 3.6.0.1 is an example of Pso to Cpso with Node decomposition. In this case, n_r would be the total number of nodes in the hidden layer and the output layer.

4. Experimental Setup

All parameters used have been proven convergent for neural network optimization in the Factorized decomposition study [2] and previous study [14]. In the factorized decomposition study the algorithm was run till 5,000,000 evaluations. Every time the Mse is calculated it is known to be an evaluation. As Merge and Decomposition equation 3.5.0.1 requires, iterations instead of function evaluations. For this study, the algorithm is run for 1000 iterations.

Variants tested

- P: A simple Pso decomposition.
- P_n: DCpso variant in which a Pso is decomposed into node decomposition.
- P_{nf}: DCpso variant in which a Pso is decomposed into Factorized node decomposition.
- P_l: DCpso variant in which a Pso is decomposed into layer decomposition.
- P_{lf}: DCpso variant in which a Pso is decomposed into Factorized layer decomposition.
- N: Cpso variant with node decomposition 3.4.2.1.
- N_m: MCpso variant in which a Cpso with node decomposition is merged into vanilla Pso.
- NF: Cpso variant with Factorized node decomposition.

- NF_m: MCpso variant in which a Cpso with factorized node decomposition is merged into vanilla Pso.
- L: Cpso variant with a Layer decomposition 3.4.1.1.
- L_m: MCpso variant in which a Cpso with layer decomposition is merged into vanilla Pso.
- LF: Cpso variant with Factorized layer decomposition.
- LF_m: MCpso variant in which a Cpso with factorized layer decomposition is merged into vanilla Pso.

Parameters

- Batch Size: a mini-batch should reflect the distribution of the data-set for this study a size of 100 was used.
- V_{max}: as discussed before in section 3.3 to avoid the particle roaming a range for the velocity must be used to avoid the particles leaving the search space making the solution converge on the local minima. The following value has proven to be 0.14286 optimum [12, 15] for dimensions in range from $[10, 10000]$.
- Weight Decay: The weight decay is normalized to a range of $(-\lambda, \lambda)$. It is can be calculated by the following formula $WD = \frac{WD}{WD+1}$ [2].
- λ: 0.001 [2].
- C1, C2: 1.49618 [2].
- ω: 0.729844 [2].

4.1. Data-sets

Dataset	Input	Hidden	Output	Dimensions	Source
Iris	4	4	3	35	[16]
Heart Disease	13	6	4	98	[17]
Wine	13	10	3	173	[18]
Soybean(large)	35	12	19	679	[19]
Breast Cancer Wisconsin	30	25	2	827	[20]

Table 4.1: Datasets and the Neural Network configuration.

All the input examples are normalized to be in the range (0,1) and the output is normalized such that the sum of output nodes is equal to one before calculating the error. For the examples which have missing values the average of that particular example is used as a substitute. Furthermore, the data sets are split into a Training set and a Testing set with a 60:40 split.

4.2. Performance Measurements

- E_T: Mse over Training set.
- E_G: Mse over Testing set.
- P_F: Generalization factor, this factor allows to see if overfitting has occurred. The previous paper [2] used the following formula $\frac{E_G}{E_T}$ to calculate this factor. If $P_F <= 1$, overfitting has not occurred. $P_F > 1$ overfitting has occurred [21].
- $p-value$: Friedman test value. This value is used to compare the train and test values of variants over 30 unique seeds. This value indicates if there is a significant difference between the given classes. If the p-value is < 0.05, the result is significant i.e. there is a statistical difference between the values.

5. Results

5.1. Pso, Mcpso and Dcpso node decomposition variants

Dataset	P	P_n	N	N_m	$p-value$
Iris	0.0222	0.0190	0.0187	0.0186	0.1040
Heart	0.0919	**0.0904**	0.0945	0.0915	0.0075
Wine	0.0413	0.0373	0.0464	0.0418	0.0578
Soybean(large)	0.0046	0.0028	0.0055	**0.0026**	0.0000
Breast Cancer	0.0461	0.0452	0.0407	0.0363	0.1000

Table 5.1: E_T over 30 runs

Dataset	P	P_n	N	N_m	$p-value$
Iris	0.0372	0.0330	0.0325	0.0331	0.7932
Heart	0.1168	0.1175	0.1161	0.1183	0.7244
Wine	0.0690	0.0674	0.0781	0.0753	0.3885
Soybean(large)	0.0122	0.0110	0.0133	**0.0097**	0.0333
Breast Cancer	0.0712	0.0715	0.0557	0.0575	0.6424

Table 5.2: E_G over 30 runs

Dataset	P	P_n	N	N_m
Iris	1.6757	1.7368	1.738	1.7796
Heart	1.2709	1.2998	1.2286	1.2929
Wine	1.6707	1.807	1.6832	1.8014
Soybean(large)	2.6522	3.9286	2.4182	3.7308
Breast Cancer	1.5445	1.5819	1.3686	1.584

Table 5.3: P_F over 30 runs

5.2. Pso, Mcpso and Dcpso factorized node decomposition variants

Dataset	P	P_{nf}	NF	NF_m	$p-value$
Iris	0.0222	0.0188	0.0288	0.0285	0.1753
Heart	0.0919	**0.0905**	0.0947	0.0920	0.0149
Wine	0.0413	0.0374	0.0385	0.0334	0.0503
Soybean(large)	0.0046	**0.0025**	0.0067	0.0039	0.0000
Breast Cancer	0.0461	0.0458	0.0334	**0.0180**	0.0023

Table 5.4: E_T over 30 runs

Dataset	P	P_{nf}	NF	NF_m	$p-value$
Iris	0.0372	0.0328	0.0429	0.0434	0.8578
Heart	0.1168	0.1169	0.1175	0.1195	0.4862
Wine	0.0690	0.0702	0.0651	0.0649	0.7722
Soybean(large)	0.0122	**0.0099**	0.0132	0.0115	0.0196
Breast Cancer	0.0712	0.0715	0.0556	0.0473	0.4370

Table 5.5: E_G over 30 runs

Dataset	P	P_{nf}	NF	NF_m
Iris	1.6757	1.7447	1.4896	1.5228
Heart	1.2709	1.2917	1.2408	1.2989
Wine	1.6707	1.877	1.6909	1.9431
Soybean(large)	2.6522	3.96	1.9701	2.9487
Breast Cancer	1.5445	1.5611	1.6647	2.6278

Table 5.6: P_F over 30 runs

5.3. Pso, Mcpso and Dcpso layer decomposition variants

Dataset	P	P_l	L	L_m	$p-value$
Iris	0.0222	0.0186	0.0122	0.0108	0.7271
Heart	0.0919	0.0904	0.0919	0.0897	0.0696
Wine	0.0413	0.0337	0.0392	0.0306	0.1176
Soybean(large)	0.0046	**0.0023**	0.0049	0.0026	0.0000
Breast Cancer	0.0461	0.0441	0.0181	0.0173	0.3229

Table 5.7: E_T over 30 runs

Dataset	P	P_l	L	L_m	$p-value$
Iris	0.0372	0.0325	0.0258	0.0254	0.7463
Heart	0.1168	0.1170	0.1178	0.1198	0.8937
Wine	0.0690	0.0677	0.0693	0.0601	0.9484
Soybean(large)	0.0122	**0.0091**	0.0120	0.0099	0.0052
Breast Cancer	0.0712	0.0726	0.0452	0.0434	0.6344

Table 5.8: E_G over 30 runs

Dataset	P	P_l	L	L_m
Iris	1.6757	1.7473	2.1148	2.3519
Heart	1.2709	1.2942	1.2818	1.3356
Wine	1.6707	2.0089	1.7679	1.9641
Soybean(large)	2.6522	3.9565	2.449	3.8077
Breast Cancer	1.5445	1.6463	2.4972	2.5087

Table 5.9: P_F over 30 runs

5.4. Pso, Mcpso and Dcpso factorized layer decomposition variants

Dataset	P	P_{lf}	LF	LF_m	$p-value$
Iris	0.0222	0.0184	0.0114	0.0113	0.7817
Heart	0.0919	**0.0894**	0.0983	0.0944	0.0001
Wine	0.0413	0.0345	0.0416	**0.0343**	0.0155
Soybean(large)	0.0046	**0.0020**	0.0078	0.0032	0.0000
Breast Cancer	0.0461	0.0458	0.0279	**0.0243**	0.0000

Table 5.10: E_T over 30 runs

Dataset	P	P_{lf}	LF	LF_m	$p-value$
Iris	0.0372	0.0340	0.0291	0.0283	0.9288
Heart	0.1168	0.1177	0.1163	0.1178	0.5404
Wine	0.0690	0.0660	0.0712	0.0648	0.4301
Soybean(large)	0.0122	**0.0096**	0.0134	0.0102	0.0000
Breast Cancer	0.0712	0.0715	0.0434	0.0452	0.9852

Table 5.11: E_G over 30 runs

Dataset	P	P_{lf}	LF	LF_m
Iris	1.6757	1.8478	2.5526	2.5044
Heart	1.2709	1.3166	1.1831	1.2479
Wine	1.6707	1.913	1.7115	1.8892
Soybean(large)	2.6522	4.8	1.7179	3.1875
Breast Cancer	1.5445	1.5611	1.5556	1.8601

Table 5.12: P_F over 30 runs

5.5. Mse over iterations

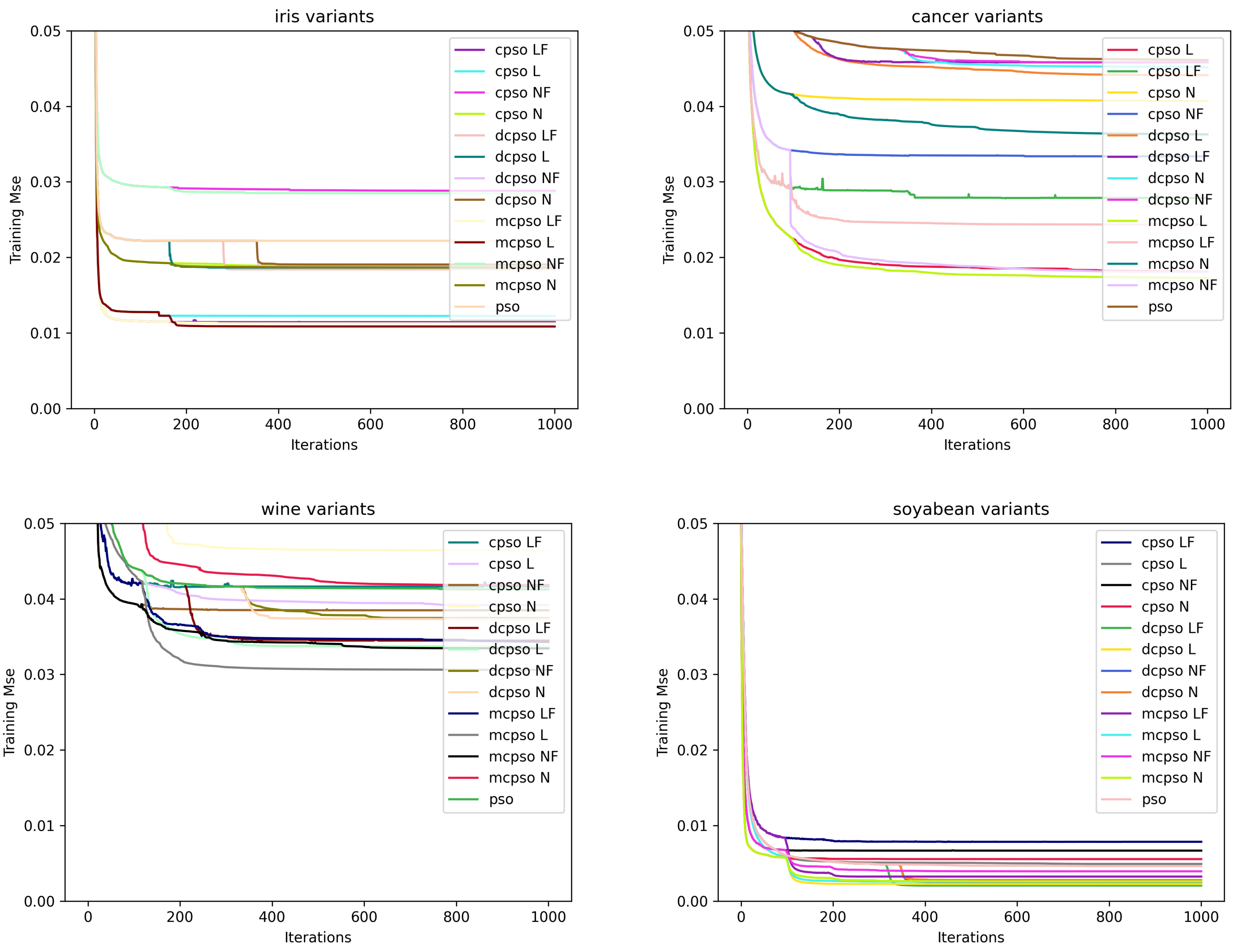


Figure 5.5.0.1: Mse Graphs over 1000 iterations

5.6. Discussion

Section 4.1 to 4.4, represents 4 decompositions known as Node, Node Factorized, Layer, and Layer Factorized discussed before in section 3.4. In each section E_T represents the average of Mse over 30 different seeds. Similarly, E_G is average over the testing data set and P_F represents the generalization factor calculated by the formula in section 4.2. The 5th column of both E_T and E_G tables represents the p-value from Friedman's test of the 4 groups. Each group contains 30 Training and Testing Mse results respectively.

The hypothesis of this experiment is as follows:
H_0: There is no significant difference between the groups.
H_1: There is a significant difference between groups.
H_0 is true if the p-value is greater than 0.05.
H_1 is true if the p-value is less than or equal to 0.05.

For both training and testing tables, the bold values represent the lowest Mse if the p-value is less than 0.05. This value shows the best-performing variant for the data-set name written in the first column. Comparing the Mse values of the significant results, the Decompose Cpso 3.6.0.1 and Merge Cpso 3.5.0.1 show slight improvement over regular Cpso with the 4 decompositions. For all 4 graphs shown above the Cpso algorithm seems to stagnate where as MCpso and DCpso further optimize the solution after merging/decomposing. This behaviour can be seen at the points where the solutions optimize further after stagnating for a couple of iterations. When comparing the node, layer and their factorized decompositions over the same algorithm. The results do not show significant improvement in training and testing, mean square error. This means that the type of decomposition used matters less than the type of algorithm used to optimize the decomposition problem. The base cpso algorithm showed slightly less overfitting when compared to mcpso and dcpso. Finally, slight improvement can be associated with the dimensional range used for this experiment. Pso, cpso, mcpso and dcpso show similar performance for small dimensional data-set and the difference in improvement grows larger as the size increases for example the iris data set with the dimensionality of 35 had no significant improvement across all algorithms and their variants whereas for breast cancer with the dimensionality of 827, the worst performing algorithm was pso with E_T

of 0.0461 and the best was factorized node with mcpso with E_T of 0.0180.

5.7. Conclusion

Overall this paper observed the results of using node and layer decomposition with their factorized versions as problem decompositions for cpso, mcpso and dcpso. It also implemented concepts such as factorization, velocity clamping, mini-batch training, weight decay and data-set normalization to help improve the overall speed and performance of all 3 training algorithms. The experiment was done on the data sets with dimensions ranging from 35 to 827, although the mcpso and dcpso algorithms showed a slight improvement over the base cpso it did not follow the trend of mcpso significantly outperforming dcpso as found in the previous paper[1]. Future studies should test the performance of these algorithms with higher dimensional data sets to observe if the trend follows or if mcpso outperforms dcpso.

References

[1] Mitchell Clark. *Comparative Study On Cooperative Particle Swarm Optimization Decomposition Methods for Large-scale Optimization*. URL: `http://hdl.handle.net/10464/15031`.

[2] Cody Dennis, Beatrice Ombuki-Berman, and Andries Engelbrecht. "Random Regrouping and Factorization in Cooperative Particle Swarm Optimization Based Large-Scale Neural Network Training". In: *Neural Processing Letters* 51 (Feb. 2020). DOI: `10.1007/s11063-019-10112-x`.

[3] Beatriz A Garro, Humberto Sossa, and Roberto Vázquez. "Back-propagation vs particle swarm optimization algorithm: Which algorithm is better to adjust the synaptic weights of a feed-forward ANN?" In: *International Journal of Artificial Intelligence* 7 (Oct. 2011), pp. 208–218.

[4] R. Mendes et al. “Particle swarms for feedforward neural network training”. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN’02 (Cat. No.02CH37290)*. Vol. 2. 2002, 1895–1899 vol.2. DOI: 10.1109/IJCNN.2002.1007808.

[5] Frans Bergh and Andries Engelbrecht. “Cooperative Learning in Neural Networks using Particle Swarm Optimizers”. In: *South African Computer Journal* 26 (Jan. 2000), pp. 84–90.

[6] Albert Volschenk and Andries Engelbrecht. “An Analysis of Competitive Co-evolutionary Particle Swarm Optimizers to Train Neural Network Game Tree Evaluation Functions”. In: June 2016, pp. 369–380. ISBN: 978-3-319-40999-3. DOI: 10.1007/978-3-319-41000-5_37.

[7] Andrich B. van Wyk and Andries P. Engelbrecht. “Overfitting by PSO trained feedforward neural networks”. In: *IEEE Congress on Evolutionary Computation*. 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586333.

[8] A. Baraldi and P. Blonda. “A survey of fuzzy clustering algorithms for pattern recognition. I”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 29.6 (1999), pp. 778–785. DOI: 10.1109/3477.809032.

[9] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95 - International Conference on Neural Networks*. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[10] F. van den Bergh and A.P. Engelbrecht. “A Cooperative approach to particle swarm optimization”. In: *IEEE Transactions on Evolutionary Computation* 8.3 (June 2004), pp. 225–239. ISSN: 1941-0026. DOI: 10.1109/TEVC.2004.826069.

[11] A.P. Engelbrecht. “Roaming Behavior of Unconstrained Particles”. In: *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*. 2013, pp. 104–111. DOI: 10.1109/BRICS-CCI-CBIC.2013.28.

[12] Elre T. Oldewage, Andries P. Engelbrecht, and Christopher W. Cleghorn. “The merits of velocity clamping particle swarm optimisation in high dimensional spaces”. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2017, pp. 1–8. DOI: 10.1109/SSCI.2017.8280887.

[13] Liang Sun et al. “A cooperative particle swarm optimizer with statistical variable interdependence learning”. In: *Information Sciences* 186.1 (2012), pp. 20–39. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2011.09.033. URL: https://www.sciencedirect.com/science/article/pii/S002002551100507X.

[14] Frans Bergh. “An Analysis of Particle Swarm Optimizers”. In: (Jan. 2002). URL: https://www.researchgate.net/publication/33786643_An_Analysis_of_Particle_Swarm_Optimizers.

[15] Elre Talea Oldewage. “The perils of particle swarm optimization in high dimensional problem spaces”. In: (2005). URL: http://hdl.handle.net/2263/66233.

[16] R.A. Fisher. *Iris*. UCI Machine Learning Repository. 1988. URL: https://archive-beta.ics.uci.edu/ml/datasets/iris.

[17] Pfisterer M Detrano R Janosi A Steinbrunn W. *Heart Disease*. UCI Machine Learning Repository. 1988. URL: https://archive-beta.ics.uci.edu/ml/datasets/heart+disease.

[18] *Wine*. UCI Machine Learning Repository. 1991. URL: https://archive-beta.ics.uci.edu/ml/datasets/wine.

[19] *Soybean (Large)*. UCI Machine Learning Repository. 1988. URL: https://archive-beta.ics.uci.edu/ml/datasets/soybean+large.

[20] WIlliam Wolberg. *Breast Cancer Wisconsin (Original)*. UCI Machine Learning Repository. 1992. URL: https://archive-beta.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+original.

[21] Axel Roebel. *The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks*. Research Report. Technical University of Berlin, Institut for Applied Computer Science, 1994. URL: https://hal.archives-ouvertes.fr/hal-02911738.