Article

Fitness Landscape Analysis of Product Unit Neural Networks

Andries Engelbrecht [1,2,*] and Robert Gouldie [3]

1 Department of Industrial Engigneering and Computer Science Division, Stellenbosch University, Stellenbosch 7600, South Africa
2 Center for Applied Mathematics and Bioinformatics, Gulf University for Science and Technology, Mubarak Al-Abdullah 32093, Kuwait
3 Computer Science Division, Stellenbosch University, Stellenbosch 7600, South Africa; robgouldie@gmail.com
* Correspondence: engel@sun.ac.za

Abstract: A fitness landscape analysis of the loss surfaces produced by product unit neural networks is performed in order to gain a better understanding of the impact of product units on the characteristics of the loss surfaces. The loss surface characteristics of product unit neural networks are then compared to the characteristics of loss surfaces produced by neural networks that make use of summation units. The failure of certain optimization algorithms in training product neural networks is explained through trends observed between loss surface characteristics and optimization algorithm performance. The paper shows that the loss surfaces of product unit neural networks have extremely large gradients with many deep ravines and valleys, which explains why gradient-based optimization algorithms fail at training these neural networks.

Keywords: fitness landscape analysis; higher-order neural networks; product unit neural networks

check for updates

Citation: Engelbrecht, A.; Gouldie, R. Fitness Landscape Analysis of Product Unit Neural Networks. *Algorithms* **2024**, *17*, 241. https://doi.org/10.3390/a17060241

Academic Editor: Stefano Mariani

Received: 15 March 2024
Revised: 6 May 2024
Accepted: 15 May 2024
Published: 4 June 2024

1. Introduction

Usually, neural networks (NNs) are constructed using multiple layers of summation units (SUs) in all non-input layers. The net input signal to each SU is calculated as the weighted sum of the inputs connected to that unit. NNs that use SUs are referred to in this paper as summation unit neural networks (SUNNs). SUNNs with a single hidden layer of SUs can approximate any function to an arbitrary degree of accuracy provided that a sufficient number of SUs are used in that hidden layer and provided that a set of optimal weights and biases can be found [1]. However, this may result in a large number of SUs in order to approximate complex functions of higher orders. Alternatively, higher-order combinations of input signals can be used to compute the net input signal to a unit. There are many types of higher-order NNs [2–5], of which this paper concentrates on product unit neural networks (PUNNs) [2], which are also referred to as pi–sigma NNs. PUNNs calculate the net input signal as the weighted product of inputs connected to that unit. Such units are referred to as product units (PUs). These PUs allow PUNNs to more easily approximate non-linear relationships and to automatically learn higher-order terms [6], using fewer hidden units than SUNNs to achieve the same level of accuracy. Additionally, PUNNs have the advantage of increased accuracy, less training time, and simpler network architectures [7].

Although PUNNs do provide advantages, they also introduce problems. If the weights leading to a PU are too large, input signals are transformed to too high an order, which may result in overfitting. Furthermore, weight updates using gradient-based optimization algorithms are computationally significantly more expensive than when SUs are used. PUs have a severe effect on the loss surface of the NN [2,6,8]. The loss surface is the hyper-surface formed by the objective function values that are calculated across the search space. In the context of NN training, the objective function is the error function, e.g., sum-squared error, and the extent of the search space is defined by the range of values that can be assigned to the NN weights and biases. While analyses of the loss surfaces of feedforward NNs that employ SUs have been done [9–13], the nature and characteristics of higher-order NN loss surfaces are not very well understood [12]. Research has shown that PUs produce convoluted error surfaces, introducing more local minima, deep ravines, valleys, and extreme gradients [7,14]. Saddle points are likely to become more prevalent as the dimensionality of the problem increases [12]. As a result, gradient-based training algorithms become trapped in local minima or become paralyzed (which occurs when the gradient of the error with respect to the current weight is nearly zero) [7]. Additionally, the exponential term in PUs induces large, abrupt changes to the weights, causing good optima to be overshot [15,16].

Furthermore, the high dimensionality of the loss surface makes it very difficult to visualize its characteristics. Recently, Li et al. [17] worked towards approaches to visualize NN loss surfaces and to use such visualizations to understand the aspects that make NNs trainable. Ding et al. [18] considered visualization of the entire search trajectory of deep NNs and projected the high-dimensional loss surfaces to lower-dimensional spaces. However, qualitative mechanisms are still in need in order to quantify the characteristics of the NN loss surface in order to better understand it. Fitness landscape analysis (FLA) is a formal approach to characterize loss surfaces [19,20], with the goal being to estimate and quantify various features of the loss surface and to discover correlations between loss surface features and algorithm performance. FLA can provide insight into the nature of the PUNN loss surfaces in order to better understand the reasons certain optimization algorithms fail or succeed to train PUNNs.

The goal of this paper is to perform FLA of PUNN loss surfaces and to determine how PUNN loss surfaces differ from those of SUNNs. The loss surfaces of oversized PUNNs, the effects of regularization, and the effects of the search bounds of the loss surface are also analyzed. The paper maps the performance of selected optimization algorithms to PUNN loss surface characteristics to determine for which characteristics some algorithms perform poorly or well.

The rest of this paper is structured as follows: Section 2 describes PUNNs. Section 3 discusses FLA, reviews FLA metrics, and describes the random walks used to gather the necessary information about the loss surface. Section 4 provides a review of current FLA studies of NNs. PUNN training algorithms are reviewed in Section 5. The empirical process followed to analyze the loss surface characteristics of PUNNs is described in Section 6. Section 7 discusses the loss surface characteristics, while correlations between the performance of PUNN training algorithms and loss surface characteristics are discussed in Section 8.

2. Product Unit Neural Networks

Higher-order NNs include functional link NNs [4], sigma–pi NNs [3], second-order NNs [5], and PUNNs [2]. PUNNs [2,6,8] calculate the net input signal to hidden units as a weighted product of the input signals, i.e.,

$$net_{y_j,p} = \prod_{i=1}^{\mathcal{I}} z_{i,p}^{v_{ji}} \tag{1}$$

instead of using the traditional SU, where the input signal is calculated as a linear weighted sum of the input signals, i.e.,

$$net_{y_j,p} = \sum_{i=1}^{\mathcal{I}+1} z_{i,p} v_{ji} \tag{2}$$

In the above, $net_{y_j,p}$ is the net input signal to unit y_j for pattern p, $z_{i,p}$ is the activation level of unit z_i, v_{ji} is the weight between units y_j and z_i, and $\mathcal{I}$ is the total number of units in the previous layer [14]. The bias is modeled as the $(\mathcal{I}+1)$-th unit, where $z_{\mathcal{I}+1,p} = -1$ for all patterns, and $v_{j,\mathcal{I}+1}$ represents the bias [14]. A SUNN is implemented with bias units for the hidden and output layer; a PUNN is implemented with a bias unit for the output layer only. There are two types of architectures that incorporate PUs [2]: (1) each layer alternates between PUs and SUs, with the output layer always consisting of SUs; (2) a group of dedicated PUs are connected to each SU while also being connected to the input units. This paper makes use of the former architecture, with one hidden layer consisting of PUs and linear activation functions used in all layers. Using this architecture, the activation of a PU for a pattern p is expressed as

$$net_{y_j,p} = \prod_{i=1}^{\mathcal{I}} z_{i,p}^{v_{ji}} = \prod_{i=1}^{\mathcal{I}} e^{v_{ji} ln(z_{i,p})} = e^{\sum_{i=1}^{\mathcal{I}} v_{ji} ln(z_{i,p})} \tag{3}$$

for $z_{i,p} > 0$. If $z_{i,p} < 0$, then $z_{i,p}$ is written as the complex number $z_{i,p} = i^2|z_{i,p}|$, yielding

$$\begin{aligned} net_{y_j,p} &= e^{\sum_{i=1}^{\mathcal{I}} v_{ji} ln|z_{i,p}|} \\ &\times (\cos(\pi \sum_{i=1}^{\mathcal{I}} v_{ji}\mathcal{I}_i) + i \sin(\pi \sum_{i=1}^{\mathcal{I}} v_{ji}\mathcal{I}_i)) \end{aligned} \tag{4}$$

where

$$\mathcal{I}_i = \begin{cases} 0 & \text{if } z_i > 0 \\ 1 & \text{if } z_i < 0 \end{cases} \tag{5}$$

The above equations illustrate that the computational costs for gradient-based approaches are higher than when SUs are used.

Durbin and Rumelhart discovered that, apart from the added complexity of working in the complex domain, which results in double the number of equations and weight variables, no substantial improvements in results were gained [2,14]. Therefore, the complex part of Equation (4) is omitted. Refer to [14] for the PUNN training rules using stochastic gradient descent (SGD).

Research has shown that the approximation of higher-order functions using PUNNs provides more accurate results, better training time, and simpler network architectures than SUNNs [7]. Training time is less because PUNNs automatically learn the higher-order terms that are required to implement a specific function [6]. PUNNs have increased information capacity compared to SUNNs [2,6]. The information capacity of a single PU is approximately $3N$, compared to $2N$ for a single SU, where N is the number of inputs to the unit. The increased information capacity results in fewer PUs required to learn complex functions, resulting in smaller network architectures.

3. Fitness Landscape Analysis

The concept of FLA comes from the evolutionary context in the study of the landscapes of discrete combinatorial problems [19]. FLA has since been successfully adapted to continuous fitness landscapes [20]. The goal of fitness landscape analysis is to estimate and quantify various features of the error surface and to discover correlations between landscape features and algorithm performance. FLA provides a better understanding as to why certain algorithms succeed or fail as well as providing a deeper understanding of the optimization problem [21]. The features of a fitness landscape are related to four high level properties: namely, modality, structure, separability, and searchability. Modality refers to the number and distribution of optima in a fitness landscape. Structure refers to the amount of variability in the landscape and describes the regions surrounding the optima. Separability refers to the correlations and dependencies among the variables of the loss function. Searchability refers to the ability of the optimization algorithm to improve the quality of a given solution and can further be considered a metric of problem hardness [21].

FLA is performed by randomly sampling points from the landscape, calculating the fitness value for each sampled point, and then analyzing the relationship between the spatial and qualitative characteristics of the sampled points. Therefore, it is important to consider the manner in which points are sampled for FLA. The samples need to be large enough to sufficiently describe and represent the search space in order to accurately estimate the characteristics of the search space. However, samples need to be obtained without a complete enumeration of every point in the search space because the search space is infinite. A balance needs to be obtained between comprehensive sampling of the search space and the computational efficiency in doing so. It is important to note that FLA has to be done in a computationally affordable manner to make it a viable option compared to selecting the optimization algorithm and hyper-parameters through a trial-and-error approach. However, Malan argues that this is not completely true, as FLA still provides a deeper understanding of the problem, providing clarification of the "black-box" nature of NNs [20]. The computational effort in FLA is largely dependent on the sampling techniques.

The sampling techniques considered in this paper are uniform and random-walk-based sampling. Uniform sampling simply takes uniform samples from the whole landscape within set bounds. No bias is given to any points in the landscape, thus providing a more objective view of the entire landscape. However, many points are required in order for it to be effective [20]. Alternatively, random walk sampling refers to "walking" through the landscape by taking random steps in all dimensions. Random walk methods have the advantage of gathering fitness information of neighboring points, which is required for certain fitness measures. However, simple random walks do not provide enough coverage of the search space [20]. Instead, progressive random walks (PRWs) are used [22]. PRWs provide better coverage by starting on the edge of the search space and then randomly moving through all dimensions, with a bias towards the opposite side of the search space. Finally, the Manhattan random walk (MRW) [22] is similar to the PRW, but each step moves in only one dimension. MRWs allow gradient information of the landscape to be estimated. Refer to [22] for a more detailed discussion and a visualization of the coverage of the sampling techniques.

The magnitude of change in fitness throughout the landscape is quantified using gradient measures [23]. The average estimated gradient G_{avg} and the standard deviation of the gradient G_{dev} are both obtained by sampling with MRWs. A low value for G_{dev} is indicative that G_{avg} is a good estimator of the gradient. Larger values of G_{dev} indicate that the gradients of certain walks deviate a lot from G_{avg}. This is an indication of "cliffs" or sudden "peaks" or "valleys" present in the landscape [23].

The variability of the fitness values or ruggedness of the landscape is estimated with the first entropic measure (FEM) [22]. Malan and Engelbrecht [22] proposed two measures based on the FEM: namely, micro ruggedness ($FEM_{0.01}$), where the step sizes of the PRWs are 1% of the search space, and macro ruggedness ($FEM_{0.1}$), where the step sizes of the PRWs are 10% of the search space. The FEM measures provide a value in $[0, 1]$, where 0 indicates a flat landscape, and larger values indicate a more rugged landscape. For a detailed description of the FEM measures and pseudocode, see [23].

The fitness–distance correlation (FDC) was introduced by Jones [24] as a measure of global problem hardness. The FDC measure is based on the premise that for a landscape to be easily searched, error should decrease as distance to the optimum decreases in the case of minimization problems. The FDC measures the covariance between the fitness of a solution and its distance to the nearest optimum. Fitness should therefore correlate well with the distance to the optimum if the optimum is easy to locate. However, the FDC requires knowledge of the global optima, which is often unknown for optimization problems. Therefore, this measure was extended by Malan [20] by making use of the fittest points in the sample instead of the global optima (FDC_s). Instead of estimating how well the landscape guides the search towards the optimum, the FDC_s quantifies how well the problem guides the search towards areas of better fitness. Therefore, FDC_s changes the focus from a measure of problem hardness to searchability. The FDC_s measure gives a value in $[-1, 1]$, where 1 indicates a highly searchable landscape, -1 indicates a deceptive landscape, and 0 indicates a lack of information in the landscape to guide the search.

The dispersion metric (DM) [25] is calculated by comparing the overall dispersion of uniformly sampled points to a subset of the fittest points. The DM describes the underlying structure of the landscape by estimating the presence of funnels. A funnel in a landscape is a global basin shape that consists of clustered local minima [20]. A single-funnel landscape has an underlying unimodal "basin"-like structure, whereas a multi-funnel landscape has an underlying multimodal-modal structure. Multi-funnel landscapes can present problems for optimization algorithms because they may become trapped in sub-optimal funnels [20]. A positive value for DM indicates the presence of multiple funnels.

Neutrality of the landscape can be characterized by the M_1 and M_2 measures [26]. M_1 calculates the proportion of neutral structures in a PRW in order to estimate the overall neutrality of the landscape. M_2 estimates the relative size of the largest neutral region. The M_1 and M_2 measures both produce values in $[0, 1]$, where 1 indicates a completely neutral landscape, and 0 indicates that the landscape has no neutral regions.

4. Neural Network Fitness Landscape Analysis

Though NNs have been studied extensively and have been widely applied, the landscape properties of the loss function are still poorly understood [12]. A review of early analyses of NN error landscapes can be found in [21].

Recent FLA of feedforward NNs have provided valuable insights into the characteristics of the loss surfaces produced when SUs are used in the hidden and output layers. Gallagher [27] applied principal component analysis to simplify the error landscape representation to visualize NN error landscapes. It was found that NN error landscapes have many flat areas with sudden cliffs and ravines: a finding recently supported by Rakitianskaia et al. [28]. Using formal random matrix theory, proofs have been provided to show that NN error landscapes contain more saddle points than local minima, and the number of local minima reduces as the dimensionality of the loss surfaces increases [12]. This finding was also recently supported by Rakitianskaia et al. [28] and Bosman et al. [29].

Bosman et al. [30] analyzed fitness landscape properties under different space boundaries. The study showed that larger bounds result in highly rugged error surfaces with extremely steep gradients and provide little information to guide the training algorithm. Rakitianskaia et al. [28] and Bosman et al. [29] showed that more hidden units per hidden layer reduce the number of local minima and simplify the shape of the global attractor, while more hidden layers sharpen the global attractor, making it more exploitable. In addition, the dimensionality of loss surfaces increases, which results in more rugged, flatter landscapes with more treacherous cliffs and ravines. Bosman et al. [10] investigated landscape changes induced by the weight elimination penalty function under various penalty coefficient values. It was shown that weight elimination alters the search space and does not necessarily make the landscape easier to search. The error landscape becomes smoother, while more local minima are introduced. The impact of the quadratic loss and entropic loss on the error landscape indicate that entropic loss results in stronger gradients and fewer stationary points than the quadratic loss function. The entropic loss function results in a more searchable landscape.

In order to cover as much as possible insightful areas of the loss surfaces of NNs, Bosman et al. [11] proposed a progressive gradient walk to specifically characterize basins of attraction. Van Aardt et al. [26] developed measures of neutrality specifically for NN error landscapes.

Dennis et al. [13] evaluated the impact of changes in the set of training samples to NN error surfaces by considering different active learning approaches and mini-batch sizes. It was shown that aspects of structure (specifically gradients), modality, and searchability are highly sensitive to changes in the training examples used to adjust the NN weights. It was also found that different subsets of training examples produce minima at different locations in the loss surface.

Very recently, Bosman et al. [31] analyzed the impact of activation functions on loss surfaces. It was shown that the rectified linear activation function yields the most convex loss surfaces, while the exponential linear activation function yields the flattest loss surface.

Yang et al. [32] analyzed the local and global properties of NN loss surfaces. Changes to the loss surface characteristics, such as variation of control parameter values, were analyzed, as well as the impact of different training phases on the loss surfaces. Sun et al. [33] provided a recent review of research on the global structure of NN loss surfaces, with specific focus on deep linear networks. Approaches to perturb the loss function to eliminate bad local minima were analyzed, as well as the impact of initialization and batch normalization. Recent loss surface analyses focused on gaining a better understanding of the loss surfaces of deep NNs [34–37].

Despite the advances made in gaining a better understanding of NN loss surfaces, no FLA studies exist to analyze the characteristics of loss surfaces produced when PUs are used. Therefore, a need exists for such an analysis, which is the focus of this paper.

5. Training Algorithms for Product Unit Neural Networks

Various optimization algorithms have been applied to train PUNNs, including SGD and meta-heuristics such as particle swarm optimization (PSO) and differential evolution (DE). This section reviews these optimization algorithms for training PUNNs. The general training process is discussed in Section 5.1, while SGD, PSO, and DE are respectively discussed in Sections 5.2–5.4.

5.1. Training Procedure

Training is the process of finding a set of weights and biases such that the NN approximates the mapping of inputs to outputs well. Training of a network is therefore an optimization problem. The purpose of training is to obtain the best combination of weight and bias values such that the error function is minimized for a particular set of examples. PSO and DE are both population-based optimization algorithms, with each algorithm using a population of individuals. Each individual represents a candidate solution, i.e., a unique combination of weights and biases: thus, one NN. The objective function used to determine the quality of an individual is the training error of all training patterns passed through the NN which the individual represents. The optimization algorithm provides the best individual, which is the NN with the optimal combination of weights and biases that minimizes the training error.

5.2. Stochastic Gradient Descent

Gradient descent (GD) [38] is possibly the most popular approach used to train NNs. GD requires an error function to measure the NN’s error at approximating the target. GD calculates the gradient of the error function with respect to the NN weights to determine the direction the algorithm must move towards in the weight space in order to locate a local optimum. The “stochastic” component in SGD is introduced by adjusting the weights after a single pattern that is randomly selected from the training set. Random selection of training examples also prevents any bias that may occur due to the order in which patterns occur in the training set [14]. SGD has the disadvantage of fluctuating changes in the sign of the error derivatives as a result of weight adjustment after each pattern. This causes the NN to occasionally unlearn what the previous steps have learned. Therefore, SGD makes use of momentum to average the weight changes in order to ensure that the search path continues in the average downhill direction. Refer to [14] for the PUNN training rule using SGD, and find the pseudocode in Algorithm 1. In this algorithm, α is the momentum, η is the learning rate, t is the number of epochs, $\mathcal{E}_T$ is the training error, $t_{k,p}$ and $o_{k,p}$ are the target and actual output values for the k’th output unit, respectively, for pattern p, and K is the number of units in the output layer. Refer to [14] for the derivations of $\Delta w_{kj}(t)$ and $\Delta v_{kj}(t)$ in the context of PUNNs.

Algorithm 1: Stochastic gradient descent learning algorithm.

```
Initialize weights ($\boldsymbol{w}$, $\boldsymbol{v}$), $\eta$, $\alpha$, and the number of epochs while stopping condition(s) not true do
    Let $\mathcal{E}_T = 0$;
    for each training pattern $p$ do
        Do the feedforward phase to calculate $y_{j,p} (\forall j = 1, ..., J)$ and $o_{k,p} (\forall k = 1, ..., K)$;
        Compute output error signals $\delta_{o_{k,p}}$ and hidden layer error signals $\delta_{y_{j,p}}$;
        Adjust weights $w_{kj}$ and $v_{ji}$ (backpropagation of errors);
        $w_{kj}(t) += \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1)$;
        $v_{kj}(t) += \Delta v_{kj}(t) + \alpha \Delta v_{kj}(t-1)$;
        $\mathcal{E}_T += \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2$;
    end
    t = t + 1
end
```

5.3. Particle Swarm Optimization

PSO is a population-based stochastic search algorithm inspired by the flocking behavior of birds [39]. A swarm of particles is maintained, where the position of each particle is adjusted according to its own experience and that of its neighbors while trying to maintain the previous search direction [14]. The positions of the particles are adjusted by adding a velocity, $\boldsymbol{v}_i(t)$, to the current position, $\boldsymbol{v}_i(t)$, as follows:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1) \tag{6}$$

The optimization process is driven by the velocity vector, which reflects the experiential knowledge of the particle and socially exchanged information from the particle's neighborhood. The experiential knowledge of a particle is generally referred to as the cognitive component and the socially exchanged information is referred to as the social component of the velocity equation. The inertia global best (*gbest*) PSO [40] is considered for the purposes of this study, for which the social component of the particle velocity update reflects information obtained from all the particles in the swarm. The social information is the best position found by the swarm, and is referred to as $\hat{\boldsymbol{y}}(t)$ [14]. The velocity of particle i is calculated as

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) + c_2 r_{2j}(t)(\hat{y}_j(t) - x_{ij}(t)) \tag{7}$$

where $v_{ij}(t)$ and $x_{ij}(t)$ are the velocity and position, respectively, in dimension j at time t. The inertia weight is given by ω, while c_1 and c_2 are the acceleration constants. The stochastic element of the PSO is incorporated with the random variables $r_{1ij}(t)$, $r_{2ij}(t)$, which are sampled from a uniform distribution over $[0,1]$. Finally, $y_{ij}(t)$ and $\hat{y}_j(t)$ denote the personal and global best positions, respectively, in dimension j for particle i.

The performance of the PSO algorithm has been shown to be sensitive to the choice of control parameter values [41]. The control parameter values must be chosen such that a good balance of exploration and exploitation is obtained. Eberhart and Shi [42] found empirically that $\omega = 0.7298$ and $c_1 = c_2 = 1.496$ lead to convergent trajectories. This control parameter value combination satisfies theoretically derived stability conditions and exhibits strong performance characteristics [43].

In the context of NN training, each particle represents the weights and biases for one NN. Pseudocode for the *gbest* PSO is provided in Algorithm 2.

Algorithm 2: Pseudocode for the inertia *gbest* PSO.

```
Create and initialize an $n_x$-dimensional swarm;
while stopping condition(s) not true do
    for each particle $i = 1,...,n_s$ do
        //set the personal best position;
        if $f(\boldsymbol{x}_i) < f(\boldsymbol{y}_i)$ then
            $\boldsymbol{y}_i = \boldsymbol{x}_i$;
        end
        //set the global best position;
        if $f(\boldsymbol{y}_i) < f(\hat{\boldsymbol{y}})$ then
            $\hat{\boldsymbol{y}} = \boldsymbol{y}_i$;
        end
    end
    for each particle $i = 1,...,n_s$ do
        update the velocity using equation (17);
        update the position using equation (16);
    end
end
```

5.4. Differential Evolution

DE [44] is a population-based stochastic search algorithm for solving optimization problems over continuous spaces. DE is a variant of the family of evolutionary algorithms (EAs). An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. DE differs from other EAs in the order that the operators are applied and the way that the mutation operator is implemented. Mutation occurs through the creation of trial vectors, which are recombined with the parent individuals to produce offspring. The trial vectors are calculated as

$$\boldsymbol{u}_i(t) = \boldsymbol{x}_{i1}(t) + \beta(\boldsymbol{x}_{i2}(t) - \boldsymbol{x}_{i3}(t))$$

where $\boldsymbol{x}_i(t)$ is a selected individual that is perturbed with the difference of two randomly selected individuals, $\boldsymbol{x}_{i2}(t)$ and $\boldsymbol{x}_{i3}(t)$, multiplied by a scalar $\beta \in (0, \infty)$. Offspring $\boldsymbol{x}'_i(t)$ are then produced through discrete recombination of the trial vectors $\boldsymbol{u}_i(t)$ and the parent vectors $\boldsymbol{x}_i(t)$ through the following crossover operator:

$$x'_{ij}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in \mathcal{J} \\ x_{ij}(t) & \text{otherwise} \end{cases}$$

where $\mathcal{J}$ is the set of indices that will undergo perturbation according to a recombination probability p_r. The next generation is created by replacing a parent with its offspring only if its offspring has better fitness than the parent.

In the context of NN training, each individual in the DE population represents the weights and biases of a single NN. Refer to Algorithm 3 for pseudocode for the DE algorithm.

Algorithm 3: Pseudocode for a general DE algorithm.

```
Let $t$ = 0 be the generation counter;
Initialize the control parameters $\beta$ and $p_r$;
Create and initialize the population $C(0)$ to consist of $n_s$ individuals;
while stopping condition(s) not true do
    for each individual $\boldsymbol{x}_i(t) \in C(t)$ do
        Evaluate the fitness $f(\boldsymbol{x}_i(t))$;
        Create the trial vector $\boldsymbol{u}_i(t)$ by applying the mutation operator;
        Create an offspring $\boldsymbol{x}'_i(t)$ by applying the crossover operator;
        if $f(\boldsymbol{x}'_i(t))$ is better than $f(\boldsymbol{x}_i(t))$ then
            Add $\boldsymbol{x}'_i(t)$ to $C(t+1)$;
        else
            Add $\boldsymbol{x}_i(t)$ to $C(t+1)$;
        end
    end
end
Return the individual with the best fitness as the solution;
```

6. Empirical Procedure

The empirical procedure followed in this study is presented. Section 6.1 outlines the datasets used, and the corresponding network architectures are detailed in Section 6.2. Section 6.3 describes the sampling and FLA metric parameters. Section 6.4 describes the performance metrics and parameters of the NN training algorithms used.

6.1. Datasets

Four well-known benchmark classification problems of varying dimensionality and five regression problems are used:

1. The **quadratic** function (f_1): $f(x) = x^2$, with $x \sim U(-1,1)$. The training and test sets consisted of 50 randomly generated patterns.
2. The **cubic** function (f_2): $f(x) = x^3 - 0.04x$, with $x \sim U(-1,1)$. The training and test sets consisted of 50 randomly generated patterns.
3. The **Hénon time series** (f_3): $x_t = 1 + 0.3x_{t-2} - 1.4x^2_{t-1}$, with $x_1, x_2 \sim U(-1,1)$. The training and test sets consisted of 200 randomly generated patterns.
4. The **surface** function (f_4): $f(x,y) = y^7x^3 - 0.5x^6$, with x, $y \sim U(-1,1)$. The training and test sets consisted of 300 randomly generated patterns.
5. The **sum of powers** function (f_5): $f(x) = x^2 + x^5$, with $x \sim U(-1,1)$. The training and test sets consisted of 100 randomly generated patterns.

The classification datasets used are listed in Table 1.

6.2. Network Architecture

Three different types of network architectures, with different numbers of hidden units, are used, as listed in Table 1:

- **Optimal architectures:** For fair comparisons between the loss surfaces of PUNNs and SUNNs, optimal architectures were used. For the regression problems, the architectures were taken from [16], and the classification problems were determined by training on an increasing number of hidden units until overfitting was observed. The optimal architectures result in models that do not overfit nor underfit the training data.
- **Oversized architectures:** Oversized architectures were used to investigate the effects of overfitting on the PUNN loss surfaces. Oversized PUNNs used the same number of hidden units as the SUNNs for all problems except for f_3, where seven hidden units were used.

- **Regularized architectures:** Weight decay with the L_2 penalty was used on oversized architectures to study the effects of regularization on the loss surfaces. The loss function becomes $\mathcal{E} = \mathcal{E}_T + \lambda \sum_{i=1}^{N} w_i^2$, where $\mathcal{E}_T$ is the training error, λ is the penalty coefficient, N is the number of weights, and w_i is the i-th weight. The effects of regularization were only investigated for the classification problems. The optimal value for the penalty coefficient, λ, was obtained using a grid search over values for λ in $\{10, 1, 0.1, 1e-2, 1e-3, 1e-4, 1e-5\}$. The optimal λ value was 0.0001 for all problems.

Table 1. Datasets and optimal network architecture.

Dataset	Inputs	PUNN Hidden	SUNN Hidden	Outputs	PUNN Dimensionality	SUNN Dimensionality
XOR	2	1	2	1	4	9
Iris	4	2	4	3	17	35
Wine	13	2	10	3	35	173
Diabetes	8	1	8	1	10	81
$f1$	1	1	2	1	3	7
$f2$	1	2	3	1	5	10
$f3$	2	5	5	1	16	21
$f4$	2	3	8	1	10	33
$f5$	1	2	3	1	5	10

6.3. Fitness Landscape Measures and Sampling Parameters

Since the weights of an NN can take on any real number, the boundaries of the fitness landscape are infinite. However, bounds have to be provided within which the FLA metrics can be applied. This study used two sets of bounds for all problems. The first set of bounds, i.e., $[-1, 1]$, focuses on areas where optimization algorithms are most likely to explore [21]. Additionally, $[-1,1]$ bounds limit the hyper-volume of the landscape, which facilitates better coverage when sampling and, subsequently, more accurate estimates of the fitness landscape measures. Larger bounds were also used. It is important to note that the bounds of the landscape represent the order of the exponential terms in the PU. With larger bounds, the PUNN performs significantly higher-order transformations of the input signals. To investigate the non-linear nature of the PUNN, $[-3, 3]$ was used for all classification problems. For the regression problems, $[-3, 3]$ was used for functions f_1, f_2 and f_3, $[-7, 7]$ was used for f_4, and $[-5, 5]$ was used for f_5.

For adequate coverage of the search space, the number of independent PRWs and MRWs was set to the dimensionality of the loss surface. PRWs and MRWs started each walk at one of the corners of the landscape. To perform a walk from every corner would require performing 2^N walks, which becomes computationally infeasible for high dimensions. Therefore, this study performed a walk at every $(\frac{2^N}{N})^{th}$ corner of the landscape as recommended by Malan [20]. Each PRW performs 1000 steps and each MRW performs 2000 steps at step sizes of 1% of the domain. Uniform sampling makes use of $500 \times N$ independent samples for the FDC_s measure and 2000 samples for the DM. For dispersion, the 10% best solutions were used as recommended by [20]. The values used for the threshold ϵ for neutrality measures M_1 and M_2 are provided in Table 2.

Table 2. Threshold ϵ values used for neutrality measures.

Datasets:	**XOR**	**Iris**	**Wine**	**Diabetes**	f_1	f_2	f_3	f_4	f_5
Small bounds:	0.02	2	2	2	0.02	0.02	0.2	2	0.2
Large bounds:	2	20	2000	20	0.2	2	2	2000	2

6.4. Training Procedure

All weights were randomly initialized in the interval $[-1,1]$. Algorithm performance was evaluated in terms of the mean squared training error $\overline{\mathcal{E}}_T$ and the mean squared generalization error $\overline{\mathcal{E}}_G$. Each simulation was executed for 500 epochs. The results are reported as averages over 30 independent runs to account for the stochasticity in the training algorithms. The standard deviation for each result is also reported. The dataset was split for every training simulation into a training set (75%) and a test set (25%). The control parameters used for PSO were $c_1 = c_2 = 1.496$ and $\omega = 0.7298$ [45], $\beta = 0.7$ and $p_r = 0.3$ for DE [14], and $\alpha = 0.9$ and $\eta = 0.1$ for SGD [14].

7. Empirical Analysis of Loss Surface Characteristics

This section discusses the results of the FLA of PUNN loss surfaces for the different architectures in comparison to the loss surfaces produced by SUNNs. Section 7.1 discusses the results obtained from the optimal network architectures, while Sections 7.2 and 7.3, respectively, consider the oversized and regularized architectures. The results for the regression problems are given in Table 3, and those for the classification problems are given in Table 4. In these tables, oPUNN refers to the optimal PUNN architectures, osPUNN refers to the oversized PUNN architectures, and rPUNN refers to regularized PUNN architectures.

Table 3. Fitness landscape analysis results for the regression problems; values in parentheses are standard deviations.

Function	Bounds	Architecture	G_{avg}	G_{dev}	M_1	M_2	$FEM_{0.01}$	$FEM_{0.1}$	DM	FDC_s
f_1	$[-1,1]$	SUNN	0.726	1.286	0.387	0.07	0.447	0.538	−0.152	0.266
			(0.254)	(0.465)	(0.086)	(0.01)	(0.054)	(0.01)	(0.017)	(0.092)
		oPUNN	8.362	31.416	0.385	0.087	0.495	0.506	−0.251	0.214
			(1.676)	(7.704)	(0.035)	(0.012)	(0.015)	(0.008)	(0.018)	(0.037)
		osPUNN	11.385	45.936	0.259	0.052	0.377	0.495	−0.217	0.261
			(3.737)	(12.606)	(0.121)	(0.039)	(0.068)	(0.016)	(0.016)	(0.064)
	$[-3,3]$	SUNN	12.399	22.713	0.135	0.041	0.411	0.541	−0.143	0.239
			(3.301)	(5.982)	(0.091)	(0.021)	(0.083)	(0.026)	(0.006)	(0.039)
		oPUNN	2.2×10^7	2.03×10^8	0.338	0.106	0.105	0.189	−0.254	0.258
			(1.3×10^7)	(1.12×10^8)	(0.071)	(0.021)	(0.007)	(0.005)	(0.006)	(0.021)
		osPUNN	1.8×10^7	1.62×10^8	0.175	0.052	0.15	0.28	−0.24	0.283
			(5.2×10^6)	(4.27×10^7)	(0.114)	(0.038)	(0.041)	(0.016)	(0.008)	(0.021)
f_2	$[-1,1]$	SUNN	0.672	1.25	0.368	0.066	0.451	0.554	−0.03	0.159
			(0.207)	(0.367)	(0.118)	(0.022)	(0.063)	(0.016)	(0.018)	(0.022)
		oPUNN	13.381	50.094	0.213	0.038	0.393	0.492	−0.024	0.378
			(5.497)	(21.381)	(0.096)	(0.016)	(0.016)	(0.012)	(0.018)	(0.055)
		osPUNN	10.417	37.775	0.178	0.036	0.338	0.493	−0.191	0.345
			(5.75)	(18.663)	(0.075)	(0.014)	(0.046)	(0.011)	(0.021)	(0.055)
	$[-3,3]$	SUNN	12.949	24.253	0.561	0.166	0.404	0.556	−0.101	0.199
			(5.467)	(10.481)	(0.201)	(0.082)	(0.083)	(0.015)	(0.006)	(0.031)
		oPUNN	2.2×10^7	1.95×10^8	0.344	0.144	0.171	0.291	−0.243	0.211
			(1.2×10^7)	(9.7×10^7)	(0.177)	(0.061)	(0.044)	(0.014)	(0.01)	(0.053)
		osPUNN	13,436	93,694	0.372	0.141	0.129	0.17	−0.204	0.053
			(58,348)	(419,741)	(0.091)	(0.034)	(0.032)	(0.003)	(0.003)	(0.0217)
f_3	$[-1,1]$	SUNN	0.848	1.602	0.826	0.328	0.416	0.566	−0.085	0.181
			(0.372)	(0.714)	(0.147)	(0.177)	(0.069)	(0.017)	(0.006)	(0.028)
		oPUNN	4.158	14.343	0.406	0.157	0.326	0.529	−0.184	0.29
			(1.824)	(6.052)	(0.113)	(0.051)	(0.039)	(0.021)	(0.008)	(0.054)
		osPUNN	5.755	17.105	0.3	0.108	0.37	0.574	−0.14	0.234
			(2.356)	(7.527)	(0.061)	(0.024)	(0.04)	(0.018)	(0.012)	(0.045)
	$[-3,3]$	SUNN	19.43	37.182	0.331	0.099	0.41	0.571	−0.101	0.164
			(9.244)	(17.911)	(0.121)	(0.038)	(0.07)	(0.015)	(0.013)	(0.02)
		oPUNN	6.6×10^6	5.84×10^7	0.083	0.037	0.225	0.487	−0.209	0.245
			(4.9×10^6)	(4.12×10^7)	(0.11)	(0.043)	(0.042)	(0.019)	(0.007)	(0.01)
		osPUNN	1.8×10^7	1.18×10^8	0.022	0.008	0.297	0.533	−0.181	0.244
			(2.1×10^7)	(1.10×10^8)	(0.059)	(0.016)	(0.036)	(0.018)	(0.003)	(0.02)
f_4	$[-1,1]$	SUNN	0.403	0.814	0.196	0.039	0.425	0.568	−0.08	0.117
			(0.182)	(0.386)	(0.08)	(0.014)	(0.077)	(0.012)	(0.016)	(0.015)
		oPUNN	182.973	897.002	0.102	0.029	0.296	0.447	−0.266	0.216
			(164.901)	(779.389)	(0.07)	(0.019)	(0.083)	(0.019)	(0.016)	(0.059)
		osPUNN	90.959	517.255	0.02	0.006	0.324	0.525	−0.212	0.208
			(76.819)	(497.354)	(0.022)	(0.005)	(0.041)	(0.014)	(0.017)	(0.031)

Table 3. *Cont.*

Function	Bounds	Architecture	G_{avg}	G_{dev}	M_1	M_2	$FEM_{0.01}$	$FEM_{0.1}$	DM	FDC_s
	$[-7,7]$	SUNN	131.74	263.355	0.969	0.04	0.399	0.578	−0.088	0.128
			(69.262)	(143.143)	(0.093)	(0.094)	(0.078)	(0.021)	(0.012)	(0.005)
		oPUNN	3.54×10^{35}	7.15×10^{36}	0.043	0.018	0.136	0.166	−0.228	0.062
			(5.98×10^{35})	(1.16×10^{37})	(0.12)	(0.045)	(0.009)	(0.011)	(0.017)	(0.018)
		osPUNN	1.64×10^{36}	3.88×10^{37}	0.015	0.01	0.149	0.223	−0.144	0.044
			(2.75×10^{36})	(6.87×10^{37})	(0.068)	(0.044)	(0.016)	(0.028)	(0.009)	(0.004)
f_5	$[-1,1]$	SUNN	0.722	1.361	0.904	0.193	0.433	0.547	−0.059	0.188
			(0.182)	(0.377)	(0.082)	(0.079)	(0.071)	(0.015)	(0.017)	(0.093)
		oPUNN	20.303	91.099	0.611	0.185	0.379	0.48	−0.229	0.229
			(9.02)	(39.774)	(0.038)	(0.064)	(0.052)	(0.003)	(0.019)	(0.047)
		osPUNN	20.6	95.352	0.516	0.156	0.382	0.481	−0.205	0.244
			(6.339)	(29.048)	(0.096)	(0.055)	(0.035)	(0.011)	(0.015)	(0.04)
	$[-5,5]$	SUNN	56.337	105.426	0.602	0.155	0.419	0.547	−0.126	0.178
			(23.83)	(44.429)	(0.195)	(0.05)	(0.072)	(0.019)	(0.017)	(0.019)
		oPUNN	4.5×10^{18}	5.67×10^{19}	0.301	0.129	0.103	0.258	−0.267	0.185
			(3.12×10^{18})	(3.59×10^{19})	(0.167)	(0.063)	(0.024)	(0.012)	(0.01)	(0.018)
		osPUNN	1.45×10^{27}	2.17×10^{28}	0.106	0.061	0.103	0.304	−0.234	0.14
			(4.60×10^{26})	(8.25×10^{27})	(0.076)	(0.049)	(0.012)	(0.014)	(0.014)	(0.022)

Table 4. Fitness landscape analysis results for the classification problems; values in parentheses are standard deviations.

Function	Bounds	Architecture	G_{avg}	G_{dev}	M_1	M_2	$FEM_{0.01}$	$FEM_{0.1}$	DM	FDC_s
XOR	$[-1,1]$	SUNN	0.608	1.139	0.369	0.107	0.461	0.561	−0.163	0.241
			(0.163)	(0.294)	(0.082)	(0.049)	(0.076)	(0.02)	(0.028)	(0.015)
		oPUNN	0.88	1.439	0.383	0.073	0.479	0.583	−0.033	0.374
			(0.059)	(0.096)	(0.047)	(0.027)	(0.08)	(0.019)	(0.013)	(0.011)
		osPUNN	0.873	1.441	0.271	0.051	0.47	0.583	0.012	0.341
			(0.086)	(0.134)	(0.03)	(0.009)	(0.055)	(0.004)	(0.002)	(0.032)
	$[-3,3]$	SUNN	12.166	22.568	0.584	0.208	0.459	0.552	−0.168	0.332
			(3.902)	(7.388)	(0.25)	(0.167)	(0.072)	(0.013)	(0.026)	(0.064)
		oPUNN	3.392	6.095	0.981	0.41	0.475	0.645	−0.153	0.311
			(0.112)	(0.468)	(0.003)	(0.03)	(0.023)	(0.007)	(0.022)	(0.044)
		osPUNN	3.503	6.421	0.938	0.36	0.482	0.672	−0.122	0.280
			(0.381)	(0.713)	(0.028)	(0.147)	(0.036)	(0.01)	(0.009)	(0.033)
		rPUNN	3.82	6.937	0.941	0.391	0.51	0.665	−0.117	0.228
			(0.294)	(0.493)	(0.033)	(0.146)	(0.053)	(0.013)	(0.004)	(0.007)
Iris	$[-1,1]$	SUNN	0.423	0.746	0.997	0.022	0.367	0.579	−0.136	0.216
			(0.15)	(0.287)	(0.014)	(0.09)	(0.074)	(0.017)	(0.011)	(0.008)
		oPUNN	4.88×10^6	3.74×10^6	0.73	0.29	0.21	0.198	−0.255	0.071
			(1.08×10^6)	(8.52×10^6)	(0.091)	(0.035)	(0.077)	(0.034)	(0.003)	(0.006)
		osPUNN	7.23×10^4	6.75×10^5	0.418	0.168	0.212	0.329	−0.165	0.084
			(2.023×10^4)	(1.92×10^6)	(0.093)	(0.045)	(0.067)	(0.045)	(0.01)	(0.01)
	$[-3,3]$	SUNN	8.567	16.237	0.726	0.169	0.169	0.578	−0.143	0.202
			(3.178)	(6.114)	(0.187)	(0.086)	(0.086)	(0.017)	(0.015)	(0.01)
		oPUNN	2.52×10^{26}	2.55×10^{27}	0.194	0.087	0.132	0.157	−0.218	0.028
			(7.51×10^{26})	(7.58×10^{27})	(0.055)	(0.017)	(0.023)	(0.007)	(0.002)	(0.004)
		osPUNN	3.41×10^{26}	4.12×10^{27}	0.022	0.012	0.133	0.169	−0.142	0.013
			(1.85×10^{27})	(2.23×10^{28})	(0.027)	(0.013)	(0.015)	(0.004)	(0.007)	(0.007)
		rPUNN	7.36×10^{26}	1.14×10^{28}	0.024	0.012	0.132	0.171	−0.143	0.022
			(4.03×10^{27})	(6.27×10^{28})	(0.036)	(0.018)	(0.017)	(0.004)	(0.001)	(0.0001)
Wine	$[-1,1]$	SUNN	2.373	5.093	0.317	0.093	0.247	0.567	−0.13	0.141
			(0.763)	(1.609)	(0.036)	(0.025)	(0.023)	(0.02)	(0.003)	(0.005)
		oPUNN	4.45×10^5	2.383×10^6	0.283	0.114	0.217	0.341	−0.208	0.05
			(6.995×10^5)	(3.764×10^6)	(0.089)	(0.036)	(0.062)	(0.05)	(0.01)	(0.002)
		osPUNN	3.197×10^7	2.635×10^8	0.039	0.017	0.166	0.397	−0.103	0.017
			(1.377×10^8)	(1.108×10^9)	(0.045)	(0.017)	(0.036)	(0.073)	(0.001)	(0.005)
	$[-3,3]$	SUNN	60.468	130.321	0.998	0.024	0.244	0.569	−0.132	−0.093
			(19.425)	(40.745)	(0.027)	(0.109)	(0.021)	(0.018)	(0.001)	(0.01)
		oPUNN	6.19×10^{28}	5.11×10^{29}	0.05	0.027	0.149	0.174	−0.206	0.023
			(2.50×10^{29})	(2.04×10^{30})	(0.095)	(0.039)	(0.016)	(0.004)	(0.01)	(0.009)
		osPUNN	3.80×10^{33}	5.15×10^{34}	0.002	0.001	0.161	0.184	−0.093	0.004
			(3.04×10^{34})	(4.06×10^{35})	(0.021)	(0.007)	(0.008)	(0.003)	(0.01)	(0.001)
		rPUNN	8.27×10^{33}	1.04×10^{35}	0.002	0.001	0.161	0.184	−0.067	0.002
			(9.93×10^{34})	(1.23×10^{36})	(0.03)	(0.018)	(0.008)	(0.003)	(0.009)	(0.001)
Diabetes	$[-1,1]$	SUNN	0.644	1.473	0.902	0.352	0.35	0.549	−0.058	0.066
			(0.45)	(1.019)	(0.114)	(0.218)	(0.057)	(0.022)	(0.018)	(0.006)
		oPUNN	2.17×10^5	1.16×10^6	0.606	0.242	0.243	0.203	−0.194	0.067
			(4.26×10^5)	(2.35×10^6)	(0.07)	(0.025)	(0.085)	(0.041)	(0.007)	(0.018)
		osPUNN	3.19×10^5	2.85×10^6	0.145	0.058	0.165	0.362	−0.12	0.031
			(1.01×10^6)	(8.67×10^6)	(0.088)	(0.034)	(0.051)	(0.06)	(0.012)	(0.01)
	$[-3,3]$	SUNN	13.958	33.822	0.41	0.128	0.363	0.553	−0.08	0.068
			(9.367)	(23.323)	(0.077)	(0.04)	(0.051)	(0.02)	(0.007)	(0.006)
		oPUNN	4.74×10^{29}	3.74×10^{30}	0.216	0.103	0.137	0.169	−0.24	0.044
			(1.42×10^{30})	(1.12×10^{31})	(0.138)	(0.052)	(0.023)	(0.005)	(0.008)	(0.007)
		osPUNN	1.68×10^{48}	3.37×10^{49}	0.006	0.002	0.145	0.173	−0.086	0.012
			(1.05×10^{49})	(2.20×10^{50})	(0.051)	(0.019)	(0.006)	(0.003)	(0.008)	(0.001)
		rPUNN	8.41×10^{43}	1.62×10^{45}	0.006	0.002	0.144	0.173	−0.076	0.004
			(2.78×10^{44})	(5.34×10^{45})	(0.046)	(0.016)	(0.006)	(0.006)	(0.003)	(0.003)

7.1. Optimal Architectures

For SUNN and PUNN loss surfaces, the nature of the PUNN loss surface is best captured by the G_{avg} and G_{dev} metrics, which are substantially larger for PUNNs for every scenario except for the XOR problem. Even for loss surfaces with smaller bounds, the PUNN G_{avg} is significantly larger than that of SUNN for the majority of the problems. Larger bounds resulted in loss surfaces with even larger gradients, especially for the diabetes and f_4 problems. The large values for G_{dev} mean that the gradients of certain walks deviate substantially from G_{avg}. This is an indication of sudden cliffs or valleys present in the loss surfaces. The G_{avg} and G_{dev} metrics portray the treacherous nature of the PUNN landscape, i.e., that of extreme gradients and deep ravines and valleys.

The ruggedness of loss surfaces is estimated using entropy using the $FEM_{0.01}$ and $FEM_{0.1}$ metrics. The amount of entropy can be interpreted as the amount of "information" or variability in the loss surface [22]. There exists a prominent trend between the gradient and ruggedness measures. Loss surfaces with smaller gradients are related to very rugged surfaces, where extremely large gradients are related to smoother surfaces. This relationship is observed for all of the problems, where Iris, Wine, Diabetes, and f_4 have large gradients and smaller FEM values. Conversely, XOR, f_1, and f_2 have smaller gradients and larger FEM values. Except for XOR, the PUNN loss surfaces are smoother than the SUNN loss surfaces, which is validated with the smaller values obtained for $FEM_{0.01}$ and $FEM_{0.1}$. SUNN landscapes tend to have more variability or "information", whereas PUNN landscapes tend to be smoother, with more consistent increases or decreases of loss values. Since surfaces with larger bounds have larger gradients, larger bounds tend to produce smoother loss landscapes. The macro-ruggedness values of $FEM_{0.1}$ exceed the corresponding micro-ruggedness values of $FEM_{0.01}$ for all scenarios, indicating that larger step sizes experience more variation in both NN loss surfaces.

FDC_s estimates how searchable a loss surface is by quantifying how well the surface guides the search towards areas of better quality. The PUNN FDC_s values for the regression problems are all moderately positive, indicating that PUNN landscapes are not deceptive but possess informative landscapes, making them more searchable. PUNN loss surfaces for all regression problems except f_1 are more searchable than those of SUNNs. This does not hold for the classification problems, where PUNN loss surfaces tend to be less searchable. Further, the searchability of both PUNN and SUNN loss surfaces decreases for classification problems. This is a result of the fact that the classification problems are higher dimensional, and thus, the volume of the landscape grows exponentially with the dimension of the landscape. Therefore, the distances between solutions of good quality become very large, producing smaller FDC_s values. This also explains the fact that landscapes with larger bounds are less searchable for all problems.

DM indicates the presence of funnels. Negative values indicate single funnels, while positive values indicate multi-funnels. Negative values were obtained for all loss surfaces, indicating single-funnel landscapes that create basin-like structures for both PUNNs and SUNNs. It is important to note that the DM measure does not estimate modality. Therefore, it is possible and likely to still have multiple local minima residing in the global basin structure. PUNN landscapes tend to produce more negative DM values, which is indicative of a simpler global topology for PUNN surfaces. Landscapes with larger bounds produce more negative DM values, correlating with landscapes of simpler global topology. Single-funneled landscapes are more searchable landscapes [20], which suggests why PUNN landscapes are more searchable with respect to FDC_s than SUNN landscapes for regression problems.

The neutrality metrics M_1 and M_2 show a general trend of smaller neutrality for PUNN landscapes, indicating that the SUNN loss surfaces are more neutral than those of PUNNs. This is in agreement with the observation of larger gradients in the PUNN loss surfaces. Larger bounds create even less neutral loss surfaces for PUNNs, correlating with the observation that larger bounds create larger gradients. The effects that larger bounds have on neutrality is amplified when architectures are higher-dimensional, such as Iris, Wine, Diabetes, and f_4; for lower-dimensional architectures, e.g., f_2 and XOR, larger bounds actually create more neutral PUNN loss surfaces. The higher-dimensional architectures have more weights in the PUs, and thus, solution quality is more susceptible to changes in the weights. Another reason why SUNN loss surfaces are more neutral is because of their tendency to have more saddle points. This is a result of the fact that SUNN architectures tend to be higher-dimensional, for which, according to theoretical findings, saddle points are more prevalent [12]. Furthermore, M_2 tends to differ less drastically and is similar in cases such as f_1, f_4, f_5, and Wine. This indicates that, although PUNN loss surfaces tend not to be as neutral as SUNN loss surfaces in general, the longest neutral areas of both tend to be the same size.

7.2. Oversized Architectures

Recall that oversized architectures are investigated to analyze the effect of overfitting behavior on the PUNN loss surfaces. The loss surfaces produced by PUNNs with oversized hidden layers are referred to as complex PUNN landscapes (CPLs) for the purposes of this section. The landscapes of PUNNs with optimal architectures are referred to as optimal PUNN landscapes (OPLs).

Most of the differences between CPLs and OPLs are a result of the differences in dimensionality: CPLs tend to have larger gradients, as indicated by larger G_{avg} values for most problems. CPLs have larger G_{dev} values, which is indicative of more sudden ravines and valleys in the landscape. Smaller M_1 and M_2 values for CPLs show that OPLs are more neutral than CPLs. This can be attributed to the larger gradients of CPLs. FDC_s values tend to be smaller for CPLs than for OPLs. This a result of the dimensionality differences, as discussed in the previous section, as well as the fact that the oversized architectures have irrelevant weights, introducing extra dimensions to the search space. The extra dimensions do not add any extra information and only divert the search, thus making the landscape less searchable. Larger DM values are obtained from CPLs, indicating that they have multi-funnel landscapes. Therefore, the global underlying structures of CPLs are more complex than OPLs, which is in agreement with the fact that CPLs are less searchable than OPLs, and is the case with multi-funnel landscapes. There is a mixed result with respect to the micro-ruggedness of the CPLs: even though CPLs tend to have larger gradients than OPLs, which is usually an indication of a smoother landscape, CPLs produce larger $FEM_{0.01}$ values than OPLs for XOR, Iris, f_3, and f_4. The macro-ruggedness $FEM_{0.1}$ values of CPLs tend to be larger than OPLs, which suggests that CPLs experience more variation in the landscape with larger step sizes than OPLs. Therefore, CPLs possess higher variability across the landscapes than OPLs.

7.3. Regularized Architectures

For the purposes of this section, the loss surfaces produced by regularized PUNNs are referred to as regularized PUNN landscapes (RPLs). The only noticeable effect that regularization has on the fitness landscapes of a PUNN is changes in the gradient measures. RPLs have larger magnitudes of gradients, as indicated by larger G_{avg} values. Larger gradients are caused by the addition of the penalty term to the objective function, which increases the overall error and causes larger loss values and, thus, larger gradients. Additionally, larger G_{dev} values indicate that regularization creates sudden ravines and valleys in the landscape, possibly introducing more local minima. The regularization coefficient λ has a severe effect on the landscape [14,21]. However, a value of $\lambda < 0.001$ (for SUNNs) is not likely to influence the error landscape significantly [21]. Referring to Table 2, the optimal value obtained from tuning the penalty coefficient was $\lambda = 0.0001$ for all problems. This was most likely due to the fact that a smaller value for λ made the contribution of the penalty term insignificant to the overall error. Therefore, as a result of the small optimal value used for λ, no other significant changes to the fitness landscape were detected by the fitness landscape measures besides G_{avg} and G_{dev}.

8. Performance and Loss Surface Property Correlation

The purpose of this section is to find correlations between good (or bad) performance of the optimization algorithms and the fitness landscape characteristics of the PUNN loss surfaces produced for the different classification and regression problems. The purpose of the section is not to compare the performances of the optimization algorithms. Comparisons of PUNN training algorithms can be found in [7,15,16].

The performance results for the different PUNN training algorithms are summarized in Tables 5 and 6 for the regression and classification problems, respectively. Provided in these tables are the average training error $\mathcal{E}_T$, the best training error achieved over the independent runs, the average generalization error $\mathcal{E}_G$, the best generalization error, and deviation values (given in parentheses).

Table 5. Training results for the regression problems.

Function	Algorithm	Architecture	$\overline{\mathcal{E}}_T$		Best $\mathcal{E}_T$	$\overline{\mathcal{E}}_G$		Best $\mathcal{E}_G$
f_1	*PSO*	oPUNN	0.059	(0.006)	0.05	0.062	(0.008)	0.051
		osPUNN	0.048	(0.013)	0.032	0.055	(0.01)	0.037
		oPUNN	0.01	(0.002)	0.009	0.012	(0.002)	0.009
		osPUNN	0.011	(0.002)	0.008	0.012	(0.003)	0.008
	DE	oPUNN	0.062	(0.01)	0.053	0.059	(0.005)	0.049
		osPUNN	0.051	(0.008)	0.039	0.058	(0.009)	0.051
		oPUNN	0.009	(0.002)	0.008	0.011	(0.002)	0.009
		osPUNN	0.011	(0.001)	0.01	0.011	(0.002)	0.008
f_2	*PSO*	oPUNN	0.029	(0.003)	0.025	0.04	(0.008)	0.028
		osPUNN	0.027	(0.003)	0.024	0.032	(0.006)	0.023
		oPUNN	0.011	(0.005)	0.008	0.015	(0.005)	0.011
		osPUNN	0.016	(0.005)	0.01	0.015	(0.008)	0.01
	DE	oPUNN	0.031	(0.004)	0.027	0.037	(0.01)	0.028
		osPUNN	0.034	(0.003)	0.03	0.038	(0.003)	0.035
		oPUNN	0.013	(0.002)	0.01	0.013	(0.004)	0.01
		osPUNN	0.017	(0.002)	0.014	0.017	(0.003)	0.012
f_3	*PSO*	oPUNN	0.37	(0.016)	0.349	0.578	(0.022)	0.554
		osPUNN	0.409	(0.011)	0.391	0.58	(0.038)	0.538
		oPUNN	0.37	(0.043)	0.324	0.663	(0.045)	0.577
		osPUNN	0.433	(0.071)	0.301	0.646	(0.104)	0.539
	DE	oPUNN	0.386	(0.028)	0.338	0.597	(0.02)	0.563
		osPUNN	0.396	(0.017)	0.376	0.639	(0.125)	0.517
		oPUNN	0.3	(0.028)	0.263	0.862	(0.461)	0.517
		osPUNN	0.343	(0.057)	0.271	0.776	(0.086)	0.696
f_4	*PSO*	oPUNN	0.028	(0.007)	0.019	0.031	(0.005)	0.024
		osPUNN	0.039	(0.006)	0.032	0.051	(0.022)	0.033
		oPUNN	0.038	(0.014)	0.024	0.063	(0.028)	0.035
		osPUNN	0.457	(0.379)	0.121	0.696	(0.646)	0.171
	DE	oPUNN	0.029	(0.007)	0.022	0.033	(0.005)	0.027
		osPUNN	0.032	(0.008)	0.025	0.043	(0.011)	0.027
		oPUNN	0.025	(0.005)	0.02	0.025	(0.005)	0.019
		osPUNN	0.355	(0.167)	0.218	0.37	(0.192)	0.135
f_5	*PSO*	oPUNN	0.081	(0.016)	0.063	0.09	(0.019)	0.061
		osPUNN	0.075	(0.029)	0.043	0.101	(0.037)	0.065
		oPUNN	0.023	(0.014)	0.011	0.027	(0.017)	0.014
		osPUNN	0.029	(0.011)	0.014	0.059	(0.036)	0.021
	DE	oPUNN	0.063	(0.018)	0.041	0.099	(0.031)	0.072
		osPUNN	0.058	(0.015)	0.03	0.331	(0.469)	0.074
		oPUNN	0.018	(0.004)	0.013	0.021	(0.003)	0.018
		osPUNN	0.024	(0.009)	0.014	0.028	(0.006)	0.019

Table 6. Training results for the classification problems.

Problem	Algorithm	Architecture	$\overline{\mathcal{E}}_T$		Best	$\overline{\mathcal{E}}_G$		Best
			$\overline{\mathcal{E}}_T$		$\mathcal{E}_T$	$\overline{\mathcal{E}}_G$		$\mathcal{E}_G$
XOR	*PSO*	oPUNN	0.003	(0.002)	0.0	0.003	(0.002)	0.0
		osPUNN	0.005	(0.005)	0.001	0.005	(0.005)	0.001
		oPUNN	0.001	(0.001)	0.0	0.001	(0.001)	0.0
		osPUNN	0.004	(0.003)	0.001	0.004	(0.003)	0.001
		rPUNN	0.003	(0.002)	0.001	0.003	(0.002)	0.001
	DE	oPUNN	0.002	(0.001)	0.001	0.002	(0.001)	0.001
		osPUNN	0.002	(0.001)	0.001	0.002	(0.001)	0.001
		oPUNN	0.001	(0.001)	0.0	0.001	(0.001)	0.0
		osPUNN	0.002	(0.001)	0.0	0.002	(0.001)	0.0
		rPUNN	0.004	(0.004)	0.001	0.004	(0.004)	0.001
Iris	*PSO*	oPUNN	0.137	(0.014)	0.121	0.131	(0.012)	0.119
		osPUNN	0.152	(0.018)	0.125	0.156	(0.015)	0.132
		oPUNN	0.191	(0.036)	0.143	0.188	(0.038)	0.143
		osPUNN	0.602	(0.21)	0.357	0.669	(0.23)	0.354
		rPUNN	0.47	(0.14)	0.286	0.498	(0.141)	0.325
	DE	oPUNN	0.127	(0.01)	0.119	0.127	(0.011)	0.118
		osPUNN	0.161	(0.03)	0.118	0.161	(0.031)	0.121
		oPUNN	0.226	(0.028)	0.185	0.222	(0.018)	0.197
		osPUNN	0.674	(0.32)	0.387	1.059	(0.994)	0.414
		rPUNN	0.477	(0.201)	0.236	0.577	(0.334)	0.256
Wine	*PSO*	oPUNN	0.209	(0.004)	0.202	0.242	(0.033)	0.2
		osPUNN	0.859	(0.175)	0.661	1.231	(0.626)	0.682
		oPUNN	0.271	(0.044)	0.229	37,960.156	(75,919.546)	0.237
		osPUNN	2133.681	(1469.965)	745.362	8.76×10^{12}	(1.75×10^{13})	165,730.721
		rPUNN	1200.283	(1463.764)	201.401	8.35×10^{8}	(1.068×10^{9})	11,838.192
	DE	oPUNN	0.201	(0.006)	0.191	0.219	(0.019)	0.199
		osPUNN	1.266	(0.287)	0.845	1.398	(0.27)	0.938
		oPUNN	0.32	(0.051)	0.224	11738.17	(23,472.638)	0.257
		osPUNN	3225.567	(3009.275)	668.72	9.77×10^{8}	(1.95×10^{9})	458.885
		rPUNN	3387.882	(3452.529)	28.73	4.53×10^{10}	(9.04×10^{10})	12,196.997
Diabetes	*PSO*	oPUNN	0.197	(0.018)	0.166	0.205	(0.016)	0.185
		osPUNN	0.226	(0.013)	0.204	3.36	(6.239)	0.216
		oPUNN	0.22	(0.004)	0.215	0.23	(0.005)	0.224
		osPUNN	0.404	(0.136)	0.256	0.423	(0.161)	0.232
		rPUNN	0.326	(0.081)	0.257	0.435	(0.273)	0.259
	DE	oPUNN	0.192	(0.008)	0.18	0.202	(0.017)	0.176
		osPUNN	0.228	(0.009)	0.211	0.244	(0.009)	0.23
		oPUNN	0.224	(0.002)	0.221	0.225	(0.005)	0.219
		osPUNN	0.512	(0.233)	0.237	0.498	(0.229)	0.239
		rPUNN	0.365	(0.201)	0.231	265.822	(530.72)	0.228

Results for SGD are not provided because it failed to train PUNNs for all problems. SGD only succeeded when the weights were initialized very close to the optimal weights. The reasons behind the failure of SGD can now be understood using FLA: It was observed that the average gradients G_{avg} for PUNN loss surfaces were exceptionally large and were orders of magnitude larger than those of SUNN loss surfaces. The standard deviations G_{dev} for PUNN loss surfaces were also very large—indicative of sudden ravines or valleys in the PUNN loss surfaces. These characteristics trap or paralyze SGD. Larger values of G_{dev} suggest that not all the MRWs sampled such extreme gradients. Taking into consideration that the longest neutral areas of both SUNN and PUNN loss surfaces tend to be the same size, only certain parts of the PUNN loss surface have extreme gradients, whereas some areas are still relatively level. Such loss surfaces are impossible to search using gradient-based algorithms. G_{avg} and G_{dev} are the only measures that differ substantially between PUNN and SUNN loss surfaces. Therefore, the gradient measures are likely to be the most relevant fitness landscape measures that explain why SGD works for SUNNs and fails for PUNNs.

Smaller $\overline{\mathcal{E}}_T$ values were obtained for OPLs compared to CPLs for all classification problems. Note that the dimensionality difference between OPLs and CPLs is the most significant for the classification problems. Loss surfaces with larger bounds—hence, larger landscape volumes—are also correlated with worse training performance. Therefore, the performance of both PSO and DE deteriorates for loss surfaces with higher dimensionality. This agrees with findings in the literature [14] and is referred to as the "curse of dimensionality". The deterioration in training performance for CPLs can be explained by the observed loss surface characteristics. CPLs were found to be less searchable: possessing more complex global structures (multi-funnels) and having increased ruggedness. Therefore, the DM, FEM, and FDC_s measures capture the effects that the "curse of dimensionality" have on the loss surface. A general trend of overfitting and inferior $\overline{\mathcal{E}}_G$ is observed for CPLs for Diabetes, Iris, Wine, and the majority of regression problems. The correlation of DM, FEM, and FDC_s with the training and generalization performance indicates that they are meaningful fitness landscape measures for performance prediction for PUNNs, especially where oversized PUNN architectures are used.

Training of regularized PUNN architectures resulted in lower $\overline{\mathcal{E}}_T$ for nearly all problems compared to oversized PUNN architectures, suggesting that regularization makes the RPLs more searchable. The only effect that regularization had on the PUNN loss surfaces was larger gradient measures. Larger G_{avg} values can be linked with improved training performance on RPLs. For Diabetes and Wine, the PUNNs with larger bounds produced very large $\overline{\mathcal{E}}_G$ and $\overline{\mathcal{E}}_T$ values. However, the best $\mathcal{E}_G$ and $\mathcal{E}_T$ values are still small. This observation along with the fact that $\overline{\mathcal{E}}_G$ and $\overline{\mathcal{E}}_T$ have large deviations suggests that a few simulations became stuck in poor areas. This can be correlated to the fact that large G_{dev} values were observed for RPLs, which suggests sudden valleys and ravines. These landscape features are possibly the reason that the PSO and DE algorithms became stuck, leading to poor performance. Furthermore, DE became stuck in areas of worse quality for more problems, suggesting that large values of G_{dev} are an indication to use PSO instead of DE. Furthermore, $\overline{\mathcal{E}}_G$ decreased for RPLs compared to CPLs; therefore, regularization proved effective at improving the generalization performance of PUNNs.

9. Conclusions

The main purpose of this work was to perform a fitness landscape analysis (FLA) on the loss surfaces produced by product unit neural networks (PUNNs). The loss surface characteristics of PUNNs were analyzed and compared to those of SUNNs to determine in what way PUNN and SUNN loss surfaces differ.

PUNN loss surfaces have extremely large gradients on average, with large amounts of deviation over the landscape suggesting many deep ravines and valleys. Larger bounds and regularized PUNN architectures lead to even larger gradients. Stochastic gradient descent (SGD) failed to train PUNNs due to the treacherous gradients of the PUNN loss surfaces. The gradients of PUNN loss surfaces are significantly larger than those of SUNN loss surfaces, which explains why gradient descent works for SUNNs and not PUNNs. Therefore, optimization algorithms that make use of gradient information should be avoided when training PUNNs. Instead, meta-heuristics such as particle swarm optimization (PSO) and differential evolution (DE) should be used. PSO and DE successfully trained PUNNs of all architectures for all problems.

PUNN loss surfaces are less rugged, more searchable in lower dimensions, and less neutral than SUNN loss surfaces. The smoother PUNN loss surfaces were strongly correlated with the larger gradient measures and were found to possess simpler overall global structures than SUNN loss surfaces, where the latter had more multi-funnel landscapes. Oversized architectures created higher-dimensional landscapes, decreasing searchability, increasing ruggedness, and having an overall more complex multi-funnel global structure. The FEM, DM, and FDC_s metrics correlated well with the poor training performance DE and PSO achieved for oversized PUNN architectures. FEM, DM, and FDC_s captured the effects of the "curse of dimensionality" on PUNN loss surfaces. Regularized PUNN loss surfaces were more searchable than complex PUNN loss surfaces, leading to better training and generalization performance. The G_{avg} metric described the effect regularization had on PUNN loss surfaces and was found to correlate well with the better training performance of DE and PSO for regularized PUNNs. Regularized PUNN loss surfaces had more deep ravines and valleys that trapped PSO and DE. Finally, PSO is suggested for loss surfaces that have large G_{dev} values.

Author Contributions: Conceptualization, A.E.; methodology, A.E. and R.G.; software, R.G.; validation, A.E. and R.G.; formal analysis, R.G.; investigation, R.G. and A.E.; resources, R.G.; data curation, R.G.; writing—original draft preparation, A.E. and R.G.; writing—review and editing, A.E.; visualization, R.G.; supervision, A.E.; project administration, A.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated during and/or analyzed during the current study are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CPL	complex product unit neural network landscape
DE	differential evolution
DM	dispersion metric
EA	evolutionary algorithm
FDC	fitness–distance correlation
FEM	first entropic measure
FLA	fitness landscape analysis
gbest	global best
GD	gradient descent
MRW	Manhattan random walk
NN	neural network
OPL	optimal product unit neural network landscape
PRW	progressive random walk
PSO	particle swarm optimization
PU	product unit
PUNN	product unit neural network
RPL	regularized product unit neural network landscape
SGD	stochastic gradient descent
SU	summation unit
SUNN	summation unit neural network

References

1. Funahashi, K.I. On the Approximate Realization of Continous Mappings by Neural Networks. *Neural Netw.* **1989**, *2*, 183–192. [CrossRef]
2. Durbin, R.; Rumelhart, D. Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks. *Neural Comput.* **1989**, *1*, 133–142. [CrossRef]
3. Gurney, K. Training Nets of Hardware Realizable Sigma-Pi Units. *Neural Netw.* **1992**, *5*, 289–303. [CrossRef]
4. Hussain, A.; Soraghan, J.; Durbani, T. A New Neural Network for Nonlinear Time-Series Modelling. *J. Comput. Intell. Financ.* **1997**, *5*, 16–26.
5. Milenkovic, S.; Obradovic, Z.; Litovski, V. Annealing Based Dynamic Learning in Second-Order Neural Networks. In Proceedings of the International Conference on Neural Networks, Washington, DC, USA, 3–6 June 1996; Volume 1, pp. 458–463.
6. Leerink, L.; Giles, C.; Horne, B.; Jabri, M. Learning with Product Units. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 27–30 November 1995; Volume 7, pp. 537–544.
7. Ismail, A.; Engelbrecht, A. Training Product Units in Feedforward Neural Networks using Particle Swarm Optimization. In *Proceedings of the International Conference on Artificial Intelligence, Chicago, IL, USA, 8–10 November 1999*; Bajić, V., Sha, D., Eds.; Development and Practice of Artificial Intelligence Techniques; IEEE: New York, NY, USA, 1999; pp. 36–40.

8. Janson, D.; Frenzel, J. Training Product Unit Neural Networks with Genetic Algorithms. *IEEE Expert* **1993**, *8*, 26–33. [CrossRef]
9. Bosman, A.; Engelbrecht, A.; Helbig, M. Visualising Basins of Attraction for the Cross-Entropy and the Squared Error Neural Network Loss Functions. *Neurocomputing* **2020**, *400*, 113–136. [CrossRef]
10. Bosman, A.; Engelbrecht, A.; Helbig, M. Fitness Landscapes of Weight-Elimination Neural Networks. *Neural Process. Lett.* **2018**, *48*, 353–373. [CrossRef]
11. Bosman, A.; Engelbrecht, A.; Helbig, M. Progressive Gradient Walk for Neural Network Fitness Landscape Analysis. In Proceedings of the Genetic and Evolutionary Computation Conference, Worksop on Fitness Landscape Analysis, Kyoto, Japan, 15–19 July 2018.
12. Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G.; LeCun, Y. The Loss Surfaces of Multilayer Networks. In Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, San Diego, CA, USA, 9–12 May 2015; pp. 192–204.
13. Dennis, C.; Engelbrecht, A.; Ombuki-Berman, B. An Analysis of the Impact of Subsampling on the Neural Network Error Surface. *Neurocomputing* **2021**, *466*, 252–264. [CrossRef]
14. Engelbrecht, A. *Computational Intelligence: An Introduction*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2007.
15. Ismail, A.; Engelbrecht, A. Global Optimization Algorithms for Training Product Unit Neural Networks. In Proceedings of the IEEE International Conference on Neural Networks, Como, Italy, 24–27 July 2000.
16. Ismail, A.; Engelbrecht, A. Pruning Product Unit Neural Networks. In Proceedings of the IEEE International Joint Conference on Neural Network, Honolulu, HI, USA, 12–17 May 2002.
17. Li, H.; Xu, Z.; Taylor, G.; Studer, C.; Goldstein, T. Visualizing the Loss Landscape of Neural Nets. In Proceedings of the Conference on Neural Processing Systems, Red Hook, NY, USA, 3–8 December 2018.
18. Ding, R.; Li, T.; Huang, X. Better Loss Landscape Visualization for Deep Neural Networks with Trajectory Information. In Proceedings of the Machine Learning Research, Seattle, WA, USA, 30 November–1 December 2023.
19. Jones, T. Evolutionary Algorithms, Fitness Landscapes and Search. Ph.D. Thesis, The University of New Mexico, Albuquerque, NM, USA, 1995.
20. Malan, K. Characterising Continuous Optimisation Problems for Particle Swarm Optimisation Performance Prediction. Ph.D. Thesis, University of Pretoria, Pretoria, South Africa, 2014.
21. Bosman, A. Fitness Landscape Analysis of Feed-Forward Neural Networks. Ph.D. Thesis, University of Pretoria, Pretoria, South Africa, 2019.
22. Malan, K.; Engelbrecht, A. A Progressive Random Walk Algorithm for Sampling Continuous Fitness Landscapes. In Proceedings of the IEEE Congress on Evolutionary Computation, Beijing, China, 6–11 July 2014; pp. 2507–2514.
23. Malan, K.; Engelbrecht, A. Ruggedness, Funnels and Gradients in Fitness Landscapes and The Effect on PSO Performance. In Proceedings of the IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 963–970.
24. Jones, T.; Forrest, S. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In Proceedings of the 6th International Conference on Genetic Algorithms, San Francisco, CA, USA, 15–19 July 1995; pp. 184–192.
25. Lunacek, M.; Whitley, D. The Dispersion Metric and The CMA Evolution Strategy. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Washinghton, DC, USA, 8–12 July 2006; pp. 477–484.
26. Van Aardt, W.; Bosman, A.; Malan, K. Characterising Neutrality in Neural Network Error Landscapes. In Proceedings of the IEEE Congress on Evolutionary Computation, San Sebastian, Spain, 5–8 June 2017; pp. 1374–1381.
27. Gallagher, M. Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling. Ph.D. Thesis, University of Queensland, St. Lucia, QLD, Australia, 2000.
28. Rakitianskaia, A.; Bekker, E.; Malan, K.; Engelbrecht, A. Analysis of Error Landscapes in Multi-layerd Neural Nertworks for Classification. In Proceedings of the IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, 24–29 July 2016.
29. Bosman, A.; Engelbrecht, A.; Helbig, M. Loss Surface Modality of Feed-Forward Neural Network Architectures. In Proceedings of the IEEE International Joint Conference on Neural Networks, Glasgow, UK, 19–24 July 2020.
30. Bosman, A.; Engelbrecht, A.; Helbig, M. Search Space Boundaries in Neural Network Error Landscape Analysis. In Proceedings of the IEEE Symposium on Foundations of Computational Intelligence, Athens, Greece, 6–9 December 2016.
31. Bosman, A.; Engelbrecht, A.; Helbig, M. Empirical Loss Landscape Analysis of Neural Network Activation Functions. In Proceedings of the Companion Conference on Genetic and Evolutionary Computation, Lisabon, Portugal, 15–19 July 2023; pp. 2029–2037.
32. Yang, Y.; Hodgkinson, L.; Theisen, R.; Zou, J.; Gonzalez, J.; Ramchandran, K.; Mahoney, M. Taxonomizing Local versus Global Structure in Neural Network Loss Landscapes. In Proceedings of the Conference on Neural Processing Systems, Online, 6–14 December 2021.
33. Sun, R.; Li, D.; Liang, S.; Ding, T.; Srikant, R. The Global Landscape of Neural Networks: An overview. *IEEE Signal Process. Mag.* **2020**, *37*, 95–108. [CrossRef]
34. Baskerville, N.; Keating, J.; Mezzadri, F.; Najnudel, J.; Granziol, D. Universal Characteristics of Deep Neural Network Loss Surfaces from Random Matrix Theory. *J. Phys. A Math. Theor.* **2022**, *55*, 494002. [CrossRef]
35. Liang, R.; Liu, B.; Sun, Y. Empirical Loss Landscape Analysis in Deep Learning: A Survey. *Syst. Eng. Theory Pract.* **2023**, *43*, 813–823.
36. Nakhodnov, M.; Kodryan, M.; Lobacheva, E. Loss Function Dynamics and Landscape for Deep Neural Networks Trained with Quadratic Loss. *Dokl. Math.* **2022**, *106*, S43–S62. [CrossRef]

37. Nguyen, Q.; Hein, M. The Loss Surface of Deep and Wide Neural Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
38. Werbos, P. Beyond Regression: New Tools for Prediction and Analysis in The Behavioural Sciences. Ph.D. Thesis, Harvard University, Cambridge, MA, USA, 1974.
39. Eberhart, R.; Kennedy, J. A New Optimizer using Particle Swarm Theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, 4–66 October 1995; pp. 39–43.
40. Shi, Y.; Eberhart, R. Parameter Selection in Particle Swarm Optimization. In Proceedings of the Seventh Annual Conference on Evolutionary Programming, San Diego, CA, USA, 25–27 March 1998; pp. 591–600.
41. Van den Bergh, F.; Engelbrecht, A. A Study of Particle Swarm Optimization Particle Trajectories. *Inf. Sci.* **2006**, *176*, 937–971. [CrossRef]
42. Eberhart, R.; Shi, Y. Evolving Artificial Neural Networks. In Proceedings of the International Conference on Neural Networks and Brain, Cambridge, MA, USA, 1–3 December 1998; pp. 5–13.
43. Cleghorn, C.; Engelbrecht, A. Particle Swarm Stability A Theoretical Extension using the Non-Stagnate Distribution Assumption. *Swarm Intell.* **2018**, *12*, 1–22. [CrossRef]
44. Storn, R. On the Usage of Differential Evolution for Function Optimization. In Proceedings of the Biennial Conference of the North American Fuzzy Information Processing Society, Berkeley, CA, USA, 19–22 June 1996; pp. 519–523.
45. Clerc, M.; Kennedy, J. The Particle Swarm-Explosion, Stability, and Convergence in A Multidimensional Complex Space. *IEEE Trans. Evol. Comput.* **2002**, *6*, 58–73. [CrossRef]