UNIKASSEL
VERSITÄT

Fachbereich Elektrotechnik/Informatik
Fachgebiet Wissensverarbeitung

Volumetric investigation into neural network loss landscapes.

BACHELORARBEIT

im Fachbereich Elektrotechnik/Informatik
der Universität Kassel

Eingereicht von:	Noah Schager
Matrikelnummer:	35370242
E-Mail:	uk061916@student.uni-kassel.de
Vorgelegt im:	Fachgebiet Wissensverarbeitung
Erstprüfer:	Prof. Dr. Gerd Stumme
Zweitprüfer:	Prof. Dr. Bernhard Sick
Betreuer:	Tobias Hille, M.Sc.
Eingereicht am:	June 16, 2025

Contents

1 Introduction

Neural networks and deep learning techniques are at the core of many modern technologies—ranging from image classification and language translation to scientific computing and autonomous systems. As these models keep growing in size and complexity, understanding what actually happens during training becomes increasingly difficult.

One idea to better understand this training behaviour is to take a closer look at the *loss landscape*—the surface defined by the loss function over the network's parameter space. Some previous work (like [LXT+18]) has created impressive visualizations of these landscapes, showing valleys, basins, and smooth paths between local minima. These pictures suggest that loss landscapes might have interesting geometric structure.

In this thesis, we try to dig a bit deeper and ask whether these structures really exist in the full landscape, or if they're just side effects of the way we look at them—through dimensionality reduction or interpolation. Instead of visualizing a slice of the landscape, we focus on exploring it directly, to get a more grounded sense of what's going on.

To do this, we explore the landscape in a discrete way; starting from a trained local minimum, we build a grid and look at neighbouring points by stepping through parameter space in fixed steps. Of course, computing the full landscape this way is too expensive in high dimensions, so we focus our exploration on lower-dimensional subspaces (*flats*) constructed from several local minima.

Our main tool is a heuristic we call $H_c(s)$, which tells us how many points (or voxels) are reachable within a certain loss threshold c, using a step size s. The idea is that the way $H_c(s)$ grows can give us clues about the structure of the landscape—whether it's flat, steep, curved, or uniform.

The goal is to see if this simple heuristic can reveal anything interesting about the local geometry of the loss surface. Ideally, this gives us a small but practical step toward understanding how these models behave in high dimensions.

2 Fundamentals

2.1 Vector Spaces

Vector spaces provide the foundational structure for many concepts used throughout this thesis, such as subspaces, bases, linear transformations, and orthogonality. Before defining a vector space, we first introduce the notion of a field, which defines the allowable scalar operations.

Definition 1 (Field by [Axl24])**.** *A* field $\mathbb{F}$ *is a set equipped with two binary operations: addition* $(+)$ *and multiplication* $(\cdot)$*, such that the following axioms hold:*

- $(\mathbb{F}, +)$ *is an abelian group with additive identity* 0*.*
- $(\mathbb{F} \setminus \{0\}, \cdot)$ *is an abelian group with multiplicative identity* 1*.*
- *Multiplication distributes over addition: for all* $a, b, c \in \mathbb{F}$*,*

$$a \cdot (b + c) = a \cdot b + a \cdot c.$$

Common examples of fields include the real numbers $\mathbb{R}$*, complex numbers* $\mathbb{C}$*, and rational numbers* $\mathbb{Q}$*.*

Definition 2 (Vector Space by [Axl24])**.** *Let* $\mathbb{F}$ *be a field. A* vector space *over* $\mathbb{F}$ *is a set* V *together with two operations:*

- *vector addition:* $+ : V \times V \to V$*,*
- *scalar multiplication:* $\cdot : \mathbb{F} \times V \to V$*,*

satisfying the following axioms for all $u, v, w \in V$ *and all scalars* $a, b \in \mathbb{F}$*:*

1. ***Associativity of addition:*** $u + (v + w) = (u + v) + w$*.*
2. ***Commutativity of addition:*** $u + v = v + u$*.*

3. ***Existence of zero vector:*** *There exists* $0 \in V$ *such that* $v + 0 = v$.

4. ***Existence of additive inverse:*** *For each* $v \in V$*, there exists* $-v \in V$ *such that* $v + (-v) = 0$.

5. ***Compatibility with field multiplication:*** $a \cdot (b \cdot v) = (ab) \cdot v$.

6. ***Identity element of scalar multiplication:*** $1 \cdot v = v$*, where* 1 *is the multiplicative identity in* $\mathbb{F}$.

7. ***Distributivity of scalar multiplication over vector addition:*** $a \cdot (u + v) = a \cdot u + a \cdot v$.

8. ***Distributivity of scalar multiplication over field addition:*** $(a+b) \cdot v = a \cdot v + b \cdot v$.

2.1.1 Subspaces

A *subspace* (or linear subspace) is a subset of a vector space that is itself a vector space under the same operations.

To define subspaces, we first recall the definition of a subset that satisfies the necessary closure properties.

Definition 3 (Subspace [Axl24])**.** *Let* V *be a vector space over a field* $\mathbb{F}$*. A subset* $U \subseteq V$ *is called a* subspace *of* V *if:*

1. *The zero vector is in* U*:* $0 \in U$,

2. U *is closed under vector addition: for all* $u, v \in U$*, we have* $u + v \in U$,

3. U *is closed under scalar multiplication: for all* $u \in U$ *and* $a \in \mathbb{F}$*, we have* $a \cdot u \in U$.

A broader concept than a linear subspace is an *affine subspace*, also known as a *flat*. While a subspace always contains the zero vector and is closed under linear combinations, an affine subspace is a translated version of a subspace and may not contain the origin.

Definition 4 (Flat [Axl24])**.** *Let* V *be a vector space over a field* $\mathbb{F}$*, and let* $U \subseteq V$ *be a linear subspace. An* affine subspace *of* V *is a set of the form*

$$A = v_0 + U = \{v_0 + u \mid u \in U\},$$

where $v_0 \in V$ is a fixed vector. The set A is said to be an affine subspace with direction U and offset v_0.

2.2 Neural Networks

Neural networks are powerful function approximators and serve as the foundation of many modern machine learning systems. They have achieved state-of-the-art performance across a wide range of domains, including image recognition, language modeling, and reinforcement learning. In this work, we focus on a simple but representative architecture: the fully connected feedforward neural network, also known as a multilayer perceptron (MLP) [GBC16].

Definition 5 (Feedforward Neural Network)**.** *Let $D \subseteq \mathbb{R}^d$ denote the input space and Θ the set of all network parameters (weights and biases). A feedforward neural network is a function*

$$L_\theta : D \to \mathbb{R}^k$$

parameterized by $\theta \in \Theta$, and composed of L layers, each applying an affine transformation followed by a nonlinear activation:

$$x^{(l+1)} = \varphi^{(l)}(W^{(l)}x^{(l)} + b^{(l)}), \quad l = 0, \ldots, L-1,$$

where:

- *$x^{(0)} = x \in \mathbb{R}^d$ is the input vector,*
- *$W^{(l)} \in \mathbb{R}^{n_{l+1} \times n_l}$ and $b^{(l)} \in \mathbb{R}^{n_{l+1}}$ are the weight matrix and bias vector for layer l,*
- *$\varphi^{(l)} : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_{l+1}}$ is the element-wise activation function,*
- *$x^{(l)}$ is the output of layer l (also called the hidden activation).*

The final output $x^{(L)}$ is the network's prediction.

Common activation functions include:

- **ReLU (Rectified Linear Unit)**:

  $$\mathrm{ReLU}(z) = \max(0, z),$$

  often used due to its computational simplicity and sparsity-inducing behavior.

- **Sigmoid**:
$$\sigma(z) = \frac{1}{1 + e^{-z}},$$
which maps real-valued inputs to the interval $(0, 1)$ and is often used in binary classification.

The goal of training a neural network is to learn the parameters θ such that the output $L_\theta(x)$ closely matches a desired target t for a given input x. This is achieved by defining a loss function $f : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$, such as mean squared error (for regression) or cross-entropy (for classification), which measures the discrepancy between the prediction and the target.

To minimize the loss over a dataset, we use gradient-based optimization. The gradient of the loss function with respect to a given weight w is computed via backpropagation. Using the chain rule, the derivative is expressed as:

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial o} \cdot \frac{\partial o}{\partial \text{net}} \cdot \frac{\partial \text{net}}{\partial w},$$

where:

- $\text{net} = Wx + b$ is the pre-activation input,
- $o = \varphi(\text{net})$ is the post-activation output,
- f is the scalar loss value for the current output and target.

The gradients $\frac{\partial f}{\partial w}$ are then used by an optimization algorithm such as stochastic gradient descent (SGD) to update the parameters in the direction that reduces the loss.

The goal of training is to find parameters $\theta^* \in \Theta$ that minimize the loss function across the dataset. Formally, we aim to solve the optimization problem:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{(x,t)\sim D} \left[f(t, L_\theta(x)) \right],$$

where D is the data distribution, t is the true target, and $L_\theta(x)$ is the network's prediction for input x under parameters θ.

This structure enables neural networks to learn complex, nonlinear mappings from data and to generalize across unseen inputs. In the following chapters, we explore how the training process shapes the geometry of the loss function over the parameter space, and how this landscape can be studied empirically.

2.3 Loss Landscapes

In the context of neural networks, the **loss landscape** refers to the scalar field induced by the loss function over the network's parameter space. Formally, let the parameter space be denoted by $\Theta \subseteq \mathbb{R}^d$, where d is the total number of trainable parameters in the model. Given a dataset D, the model's empirical loss is defined as follows:

Definition 6 (Loss Function)**.** *Let $\theta \in \Theta \subset \mathbb{R}^d$ denote the parameters of a neural network, and let D be a dataset. We define the loss function as*

$$\mathcal{L}(\theta, D) : \Theta \times D \to \mathbb{R},$$

which maps parameters and data to a scalar error value. For notational convenience, we define the landscape function

$$f(\theta) := \mathcal{L}(\theta, D),$$

which we use throughout to refer to the scalar loss evaluated at a fixed dataset.

The full loss landscape can thus be interpreted as the graph of f, embedded in $\mathbb{R}^{d+1}$. Each point on the landscape corresponds to a tuple $(\theta, f(\theta)) \in \mathbb{R}^{d+1}$, representing the loss value associated with a particular parameter configuration.

In low-dimensional settings (e.g., $d = 2$ or 3), this surface can be visualized directly, revealing valleys, ridges, and flat regions. However, in modern deep networks where $d \gg 10^3$, direct visualization becomes infeasible.

Despite these challenges, the geometric and topological properties of the loss landscape are crucial to understanding training dynamics and generalization. Key structural elements include:

- **Sharp minima:** Regions where the loss increases rapidly in most directions, often associated with poor generalization.
- **Flat minima:** Wide valleys of low loss, typically correlated with robustness and better generalization.
- **Saddle points:** Critical points that behave like minima in some directions and maxima in others; common in high-dimensional landscapes.
- **Basins and connectivity:** Regions of connected low loss and potential low-loss paths between minima, which may indicate redundancies or symmetries in parameterization.

Although mathematics provides formal tools to describe such geometric structures—such as the concept of *manifolds*—we intentionally refrain from employing their full formalism. This is because our methodology does not rely on specific manifold properties, nor do we assume that invoking them yields additional insight for our analysis.

2.4 Linear Interpolation

Linear interpolation is a simple and widely used method for estimating unknown values between two known data points. It is based on the assumption that the function varies linearly between the given values. This technique is often employed in numerical analysis, curve fitting, and scientific computing [Dav75].

Definition 7 (Linear Interpolation)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a function and let $x_0, x_1 \in \mathbb{R}$ with $x_0 < x_1$ be two known points. The* ***linear interpolation*** *polynomial $p(x)$ between x_0 and x_1 is defined as:*

$$p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} \cdot (x - x_0),$$

which is the unique degree-1 polynomial that satisfies $p(x_0) = f(x_0)$ and $p(x_1) = f(x_1)$.

The expression arises from solving the proportion:

$$\frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

which represents the slope of the secant line between the two known points.

The error introduced by linear interpolation is given by the remainder term:

$$R(x) = f(x) - p(x),$$

which quantifies the difference between the true function value and the interpolated estimate.

In high-dimensional spaces, linear interpolation can be generalized to operate between vectors in $\mathbb{R}^d$, often used to interpolate between parameter configurations $\theta_0, \theta_1 \in \mathbb{R}^d$ in neural network landscapes. In this setting, interpolation is defined as:

$$\theta(t) = (1 - t)\theta_0 + t\theta_1, \quad t \in [0, 1],$$

yielding a linear path in parameter space.

2.5 QR Decomposition

Given a matrix $A \in \mathbb{R}^{d \times m}$ with linearly independent column vectors, we often seek an orthonormal basis for the column space of A. This is particularly useful when constructing subspaces, such as flats in high-dimensional loss landscapes, where a numerically stable basis is crucial. One standard method to obtain such an orthonormal basis is through the *QR decomposition* [GVL13].

The QR decomposition expresses a matrix A as the product of two matrices:

$$A = QR,$$

where:

- $Q \in \mathbb{R}^{d \times m}$ is a matrix with orthonormal columns, i.e., $Q^\top Q = I$,
- $R \in \mathbb{R}^{m \times m}$ is an upper triangular matrix.

A common algorithm for performing this decomposition is the **Gram–Schmidt orthogonalization process**, which incrementally constructs an orthonormal basis for the span of the columns of A.

Let $A = [a_1, a_2, \ldots, a_m]$, where a_k is the k-th column of A. The algorithm proceeds as follows:

- For each column vector a_k, subtract from it the projections onto all previously computed orthogonal vectors u_j:

$$u_k = a_k - \sum_{j=1}^{k-1} \mathrm{proj}_{u_j}(a_k),$$

  where the projection is defined by:

$$\mathrm{proj}_{u_j}(a_k) = \frac{\langle u_j, a_k \rangle}{\langle u_j, u_j \rangle} u_j.$$

- Normalize u_k to obtain the orthonormal vector e_k:

$$e_k = \frac{u_k}{\|u_k\|}.$$

After iterating over all columns, the matrix $Q = [e_1, e_2, \ldots, e_m]$ contains the orthonormal basis, and R is computed by $R = Q^\top A$.

In the context of this thesis, QR decomposition is used to orthogonalize the flat directions in the parameter space that we define based on differences between local minima. This ensures that the steps we take when traversing the loss landscape subspace are linearly independent and well-scaled, reducing numerical instability during exploration.

2.6 Breadth-First Search

Breadth-First Search (BFS) is a classical algorithm in graph theory used for traversing or searching tree and graph data structures [CLRS22].

Definition 8 (Breadth-First Search (BFS))**.** *Breadth-First Search is a graph traversal algorithm that systematically explores a space by visiting all neighbors of a given node before moving on to the next level.*

BFS begins at a specified start node and explores all neighboring nodes at the present depth prior to moving on to nodes at the next level of depth. It maintains a queue data structure to keep track of the nodes to be explored. The procedure operates as follows:

Algorithm 1 Breadth-First Search (BFS)

```
1: procedure BFS(Graph, start)
2:     Create an empty queue Q
3:     Create a set visited to track visited nodes
4:     Enqueue start node into Q
5:     Add start node to visited
6:     while Q is not empty do
7:         current_node ← Dequeue Q
8:         Process current_node (e.g., print it, or record the path)
9:         for each neighbor of current_node in Graph do
10:            if neighbor is not in visited then
11:                Enqueue neighbor into Q
12:                Add neighbor to visited
13:            end if
14:        end for
15:    end while
16: end procedure
```

This process ensures that nodes are explored in increasing order of distance from the start node, hence the term *breadth-first*. The resulting traversal pattern expands uniformly outward from the start, layer by layer.

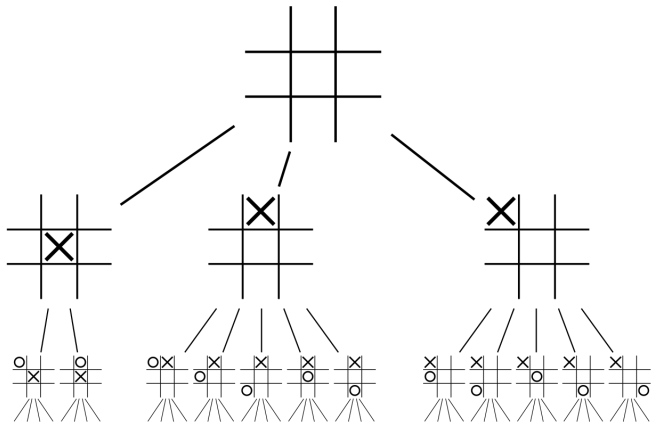


Figure 2.1: Illustration of breadth-first exploration on a graph (from [Wika]).

Compared to its counterpart, Depth-First Search (DFS), which explores as far as possible along a branch before backtracking, BFS is more suitable in scenarios where a comprehensive exploration of a space is required at uniform resolution. In our context, BFS is particularly advantageous for exploring high-dimensional loss landscapes, as it allows us to systematically visit parameter regions within a bounded loss level without prematurely descending into isolated minima.

2.7 Marching Cubes

The Marching Cubes algorithm, originally introduced in the patent [CL87], is a widely used method for extracting polygonal meshes from scalar fields, such as medical imaging data or implicit functions.

Definition 9 (Marching Cubes)**.** *The Marching Cubes algorithm converts a discretized scalar field into a polygonal mesh by mapping local binary corner configurations of voxels to edge intersections, thereby producing an isosurface.*

The algorithm operates by traversing a scalar field that has been discretized into a 3D grid of voxels. Each voxel consists of eight corners (or vertices), and at each corner, the scalar field value is compared to a predefined isosurface threshold. If the value at a corner is greater than or equal to the threshold, it is assigned a binary value of 1; otherwise, it is assigned 0. These eight binary values collectively form an 8-bit number, representing one of $2^8 = 256$ possible corner configurations for a cube.

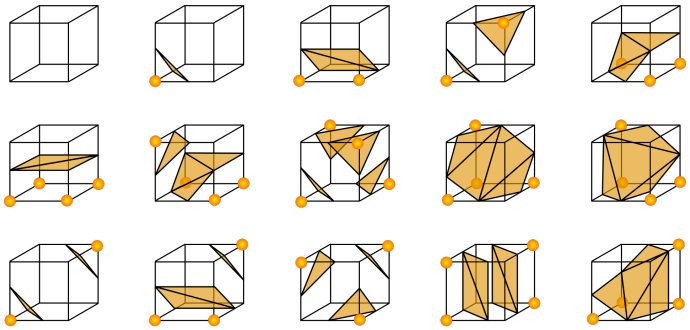


Figure 2.2: Examples of distinct cube configurations in Marching Cubes based on threshold comparisons at the corner (from [Wikb]).

However, due to geometric symmetries (such as rotations and reflections), many of the 256 configurations are redundant. In fact, only 15 unique cases need to be handled explicitly, and the rest can be derived from these via symmetry transformations.

These configurations and their corresponding triangulations are typically stored in a precomputed lookup table, which maps each 8-bit index to a set of edges where the isosurface intersects the cube. The final mesh is constructed by connecting these intersection points with polygons (usually triangles), resulting in a piecewise linear approximation of the isosurface.

The efficiency and simplicity of Marching Cubes make it a standard tool in scientific visualization, though it is not without limitations—particularly in higher dimensions, where its direct extension (such as Marching Hypercubes) becomes increasingly complex. Nevertheless, in three-dimensional settings, it remains an effective way to capture topological and geometric structure in scalar fields.

2.8 Flood Fill

Flood fill is a classic traversal algorithm, originally popularized in image processing and computer graphics [FB85]. It is the foundation of operations such as the “paint bucket” tool in digital editors, where a region connected to a seed pixel is filled with a new color.

The underlying idea is general: starting from a given *seed point*, the algorithm recursively (or iteratively) explores neighboring elements that satisfy a specific condition—such as equality to a target value, staying within a threshold, or being unvisited. Although classically applied in two dimensions, flood fill generalizes naturally to higher-dimensional spaces and grid-based scalar fields.

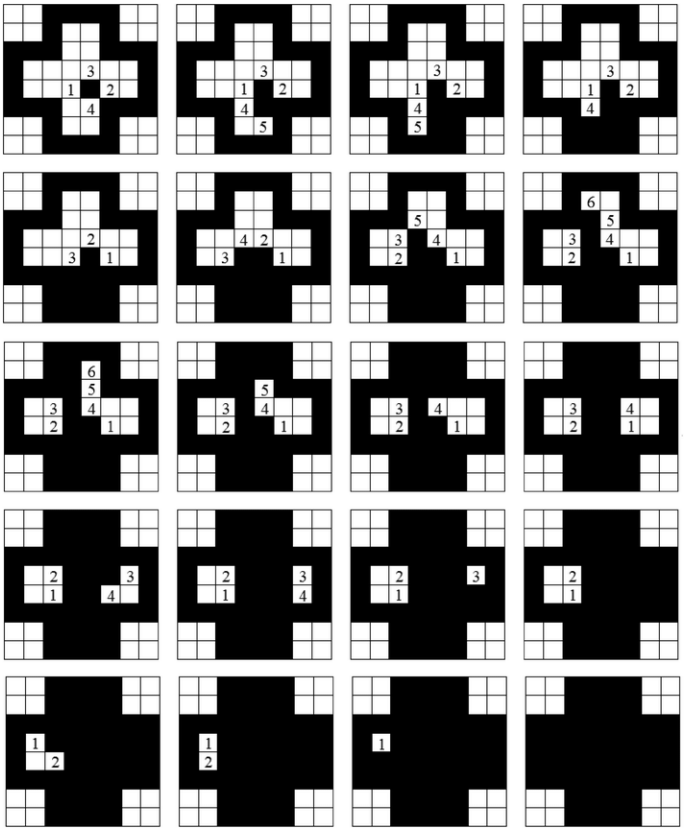


Figure 2.3: Examples of flood fill, where highest pixel is explored first (from [flo]).

Definition 10 (Flood Fill)**.** *Flood Fill is a traversal algorithm that expands from a given seed point to all adjacent elements that satisfy a specified condition, such as value similarity or threshold inclusion. It proceeds in all directions according to a chosen neighborhood structure until no further expansion is possible.*

Flood fill can be implemented using either a Depth-First Search (DFS) or Breadth-First Search (BFS) strategy. The BFS variant is particularly useful for bounded or sparse regions as it avoids stack overflow and offers more uniform coverage.

Below is a simplified recursive implementation for two-dimensional grids:

Algorithm 2 Recursive Flood Fill

```
1: procedure FloodFill(grid, x, y, target_value, replacement_value)
2:     if x < 0 or x ≥ number of rows in grid or y < 0 or y ≥ number of columns in grid then
3:         return
4:     end if
5:     if grid[x][y] ≠ target_value then
6:         return
7:     end if
8:     grid[x][y] ← replacement_value
9:     FloodFill(grid, x + 1, y, target_value, replacement_value)
10:    FloodFill(grid, x − 1, y, target_value, replacement_value)
11:    FloodFill(grid, x, y + 1, target_value, replacement_value)
12:    FloodFill(grid, x, y − 1, target_value, replacement_value)
13: end procedure
```

3 Related Work

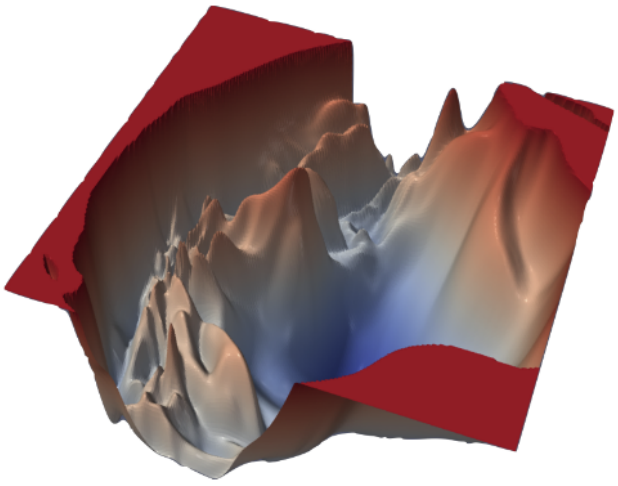
In this section, we review influential works that investigate the structure of loss landscapes in deep neural networks, focusing on approaches relevant to understanding curvature and connectivity properties.

Visualizing Loss Landscapes. Li et al. [LXT+18] introduced a method for visualizing two-dimensional slices of the high-dimensional loss surface around trained solutions. Given random directions $\delta_1, \delta_2 \in \mathbb{R}^d$, they analyzed the loss in the plane

$$f(\alpha, \beta) = \mathcal{L}(\theta + \alpha\delta_1 + \beta\delta_2),$$

where θ is a trained parameter vector (similar to our use of $\theta^\star$ for a local minimum). Their visualizations demonstrated how architecture and optimization choices influence curvature and flatness of minima. For instance, residual networks tended to converge to broader, flatter regions than comparable architectures without skip connections.

They further analyzed the Hessian $\nabla^2\mathcal{L}(\theta)$, which is the matrix of second-order derivatives of the loss with respect to the parameters. The eigenvalues of this matrix provide insight into local curvature: a concentration of small eigenvalues typically indicates a flat region. The symbol ∇ (nabla) denotes the gradient operator, which computes first derivatives.



(a) ResNet-110, no skip connections

Figure 3.1: Visualization of ResNet (from [LXT+18])

Deep Ensembles and Functional Diversity. Fort et al. [FHL19] examined deep ensembles using loss landscape geometry, proposing that ensemble generalization gains stem from sampling distinct modes in the loss surface. They analyzed linear paths:

$$\theta(t) = \theta_0 + tv, \quad v \in \mathbb{R}^d,$$

where θ_0 is a reference minimum (as in $\theta^\star$), and v is a direction vector. They found that models sampled along these paths were functionally similar, while independently trained models occupied diverse basins—regions of parameter space that yield different functional outputs.

Symmetries and Overparameterization. Şimşek et al. [SGJ+21] explored how symmetries and overparameterization affect the topology of the loss landscape. In underparameterized networks, global minima tend to be isolated. However, with additional neurons, many of these isolated solutions merge into continuous manifolds of functionally equivalent networks, arising from neuron permutations and the introduction of zero-sum neurons (whose outputs cancel each other).

They express this with

$$f(x \mid \theta_m) = f(x \mid \theta_r),$$

where θ_r and θ_m are parameter vectors corresponding to networks with r and m hidden units respectively, and $r < m$. The notation $f(x \mid \theta)$ denotes the output of the network for input x under parameters θ, emphasizing that distinct architectures (in terms of neuron count) can realize the same function. This suggests that in overparameterized settings, the loss surface contains large, connected regions of equivalent solutions, rather than isolated optima.

Wedge Geometry of Loss Landscapes. Fort and Jastrzębski [FJ19] proposed a geometric model in which loss landscapes are composed of intersecting "wedges": regions that are flat along n directions and steep along the remaining $D - n$. In such settings, successful optimization only requires moving in directions that intersect these wide regions.

They consider a parameterized subspace $\mathcal{H}$ spanned by direction vectors $\{v_i\}$, leading to trajectories of the form:

$$\theta(\alpha) = \theta_0 + \sum_i \alpha_i v_i.$$

When $\dim(\mathcal{H}) > D - n$, optimization remains effective, as the subspace intersects enough flat directions. They also introduce "m-connectors": low-loss interpolation paths that connect $m + 1$ optima. These connectors lie within the convex hull of trained models, and provide a mechanism for exploring landscape connectivity.

Loss Landscape Volume and Generalization. Wu et al. [WZ+17] examined how generalization correlates with the volume of attraction basins in the loss landscape. A *basin of attraction* $\mathcal{B}_i$ is the region of parameter space from which gradient descent will converge to a particular minimum θ_i. That is, for gradient flow defined as $\dot{\theta} = -\nabla_\theta \mathcal{L}(\theta)$ (where $\nabla_\theta \mathcal{L}(\theta)$ denotes the gradient of the loss with respect to the parameters θ), the basin $\mathcal{B}_i$ contains all points that lead to θ_i under this flow.

They proposed that wider basins correspond to better generalizing solutions, because they are easier to reach from random initialization. To estimate curvature and volume, they use the Hessian spectrum and define a proxy:

$$V(k) = \sum_{i=1}^{k} \log \lambda_i,$$

where λ_i are the top k eigenvalues of the Hessian. Larger values of $V(k)$ indicate sharper, narrower minima; smaller values correspond to flatter regions with potentially larger basins. This supports the idea that flatness and volume are key indicators of generalization—an idea echoed in our own heuristic $H_c(s)$, which quantifies the number of reachable voxels within a loss threshold c.

Together, these works provide conceptual and empirical tools that guide our research. Rather than relying on interpolation or curvature estimation, we focus on a grid-based flood fill heuristic to explore how much of the parameter space around a minimum is reachable under a loss constraint.

4 Methodology

In this chapter, we present our computational approach for discretely exploring neural network loss landscapes. We introduce a voxel-based grid traversal technique and the heuristic $H_c(s)$ to quantitatively characterize the local geometry surrounding trained solutions.

Our approach centers on discretizing affine subspaces of the loss landscape and systematically traversing them using a grid-based procedure. This enables us to probe the landscape in a structured and resolution-aware manner, avoiding reliance on gradients or continuous interpolation.

While prior work often relied on interpolation between parameter vectors or random projections, our method instead performs an exhaustive and deterministic exploration of local neighborhoods. This allows for systematic and reproducible sampling, free from the artifacts introduced by interpolation or low-dimensional projection.

4.1 Grid Abstraction

A central difficulty in exploring the geometry of neural network loss landscapes lies in the high dimensionality of the parameter space and the continuous nature of real-valued functions. Directly analyzing the loss landscape $f : \mathbb{R}^d \to \mathbb{R}$ requires expensive evaluations at continuous coordinates. To simplify this, we adopt a discrete grid-based abstraction over the parameter space, which replaces the continuous domain with a structured, finite set of integer-valued coordinates.

This discrete formulation allows us to treat the landscape as a combinatorial object—a voxel grid—where each node corresponds to a well-defined position determined by integer steps from a starting point. The main advantage of this approach is conceptual and computational clarity: rather than navigating through continuous $\mathbb{R}^d$, we move through $\mathbb{Z}^d$, where each step corresponds to an exact multiple of a predefined resolution $s > 0$. This makes counting, caching, and traversal both precise and efficient.

Additionally, this abstraction helps avoid the pitfalls of floating-point arithmetic. For example, when probing a point like $\theta^\star + \varepsilon \cdot e_i$, where e_i is a unit basis vector and $\varepsilon > 0$, repeated arithmetic operations may produce slightly different results due to rounding

errors. This can cause logically identical points to be treated as distinct in memory, leading to redundant evaluations or missed connections. By working in a discrete coordinate system indexed over $\mathbb{Z}^d$, such ambiguities are eliminated entirely.

To address this, we introduce a *grid abstraction* that discretizes the traversal domain into a structured grid over $\mathbb{Z}^d$. Instead of operating directly in continuous space, we define exploration steps as movements on this discrete integer lattice. Each point $a \in \mathbb{Z}^d$ represents a relative offset from the origin (i.e., the minimum $\theta^\star$), and is uniquely mapped back to parameter space via:

$$\theta(a) = \theta^\star + s \cdot Ba,$$

where:

- $s \in \mathbb{R}^+$ is a fixed step width controlling traversal resolution,
- $B \in \mathbb{R}^{d \times m}$ is a (possibly orthogonal) basis matrix that spans the subspace of exploration,
- $a \in \mathbb{Z}^d$ is an abstract coordinate encoding grid position.

This transformation has several advantages:

1. **Numerical Stability:** All operations in $\mathbb{Z}^d$ are exact, eliminating the risk of floating-point drift or false inequality checks.
2. **Modularity:** The traversal algorithm operates purely in discrete space, while the real-valued evaluations $f(\theta(a))$ are cleanly separated and occur only via explicit mapping.
3. **Dimensional Control:** By selecting B appropriately, we can restrict traversal to low-dimensional flats (e.g., 2D, 3D) even in high-dimensional networks.

In practice, the grid structure enables us to implement traversal algorithms such as breadth-first search (BFS), while avoiding the overhead of tracking floating-point coordinates directly. A queue of relative integer positions is maintained, ensuring every point is visited only once, and all relevant neighbors can be computed using deterministic integer increments.

4.2 Flat Construction

As mentioned previously, modern neural networks operate in extremely high-dimensional parameter spaces, $\mathbb{R}^d$, where d can range from thousands to hundreds of millions. As a result, direct analysis or visualization of the full loss landscape is infeasible. To circumvent this, we restrict our analysis to a low-dimensional affine subspace—commonly referred to as a *flat*—embedded within the full parameter space.

This approach builds on prior work that investigates the loss landscape by examining low-dimensional slices, typically of dimension $d = 1$ or $d = 2$, in order to study properties such as mode connectivity, sharpness, and generalization [LXT+18, GIP+18, **?**]. Rather than relying on randomly chosen directions, our aim is to construct a flat—i.e., a low-dimensional affine subspace—designed to capture meaningful geometric structure in the vicinity of multiple local minima.

4.2.1 Basis Construction via Multiple Minima

To define the flat, we begin by training the same neural network $m+1$ times using different random initializations. This results in $m + 1$ distinct local minima in parameter space:

$$\{\theta_0^\star, \theta_1^\star, \ldots, \theta_m^\star\}, \quad \theta_i^\star \in \mathbb{R}^d.$$

We define the basis for our subspace using the differences between a reference minimum $\theta_0^\star$ and the other obtained minima:

$$b_i = \theta_i^\star - \theta_0^\star, \quad i = 1, \ldots, m.$$

When analyzing lower-dimensional flats, the basis matrix B is an orthonormal matrix of dimension $d \times m$, constructed from the differences between multiple local minima obtained by training the network from distinct initializations.

To ensure numerical stability and interpretability, we apply QR decomposition to orthonormalize the columns of B:

$$Q, R = \mathrm{QR}(B), \quad Q \in \mathbb{R}^{d \times m}, \quad R \in \mathbb{R}^{m \times m}.$$

The matrix Q now provides an orthonormal basis for the flat, which is an m-dimensional subspace of $\mathbb{R}^d$ anchored at $\theta_0^\star$.

4.2.2 Mapping Grid Coordinates to Parameter Space

With this construction, any point θ in the flat can be written as:

$$\theta(a) = \theta_0^\star + Qa \cdot s, \quad a \in \mathbb{Z}^m,$$

where $s > 0$ is a scalar step size that controls the resolution of the grid. This defines a regular voxel grid within the flat, enabling structured exploration of the loss landscape while operating in a numerically stable subspace.

4.2.3 Benefits of Flat-Based Analysis

By restricting exploration to the span of Q, we gain several advantages:

- **Computational feasibility:** Evaluating and storing loss values across a high-resolution grid becomes tractable.
- **Structural relevance:** The directions b_i reflect meaningful differences between converged models, increasing the likelihood that the flat intersects interesting topological features of the loss landscape.
- **Floating-point robustness:** Integer coordinates in $\mathbb{Z}^m$ can be consistently mapped to high-precision floating-point values via the affine transform $\theta(a)$, avoiding cumulative rounding errors.

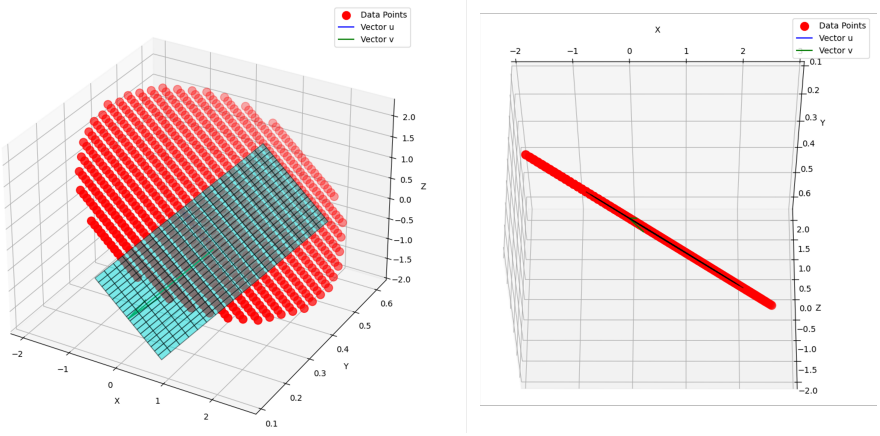


Figure 4.1: Illustration of flat spanned by directions between multiple trained minima (left) and alternative view showing the affine alignment of the flat within parameter space (right)

4.3 Grid Traversal

To investigate the local structure of neural network loss landscapes, we define a discrete traversal algorithm that operates on a voxelized abstraction of the parameter space. Let $\Theta \subseteq \mathbb{R}^d$ denote the parameter space of a neural network trained on dataset D, and let $f : \Theta \to \mathbb{R}$ be the loss function, mapping parameters $\theta \in \Theta$ to a scalar loss value.

Instead of global analysis, we adopt a local exploration approach: beginning from a trained local minimum $\theta^\star \in \Theta$ such that $f(\theta^\star) = \varepsilon$ for some small $\varepsilon > 0$, we discretize the surrounding parameter space into a regular grid and traverse it selectively.

4.3.1 Voxel-Based Landscape Discretization

We define a hypergrid around $\theta^\star$ using a fixed step width $s > 0$, resulting in a lattice of axis-aligned hypercubes (voxels) centered at discrete offsets. Each voxel is identified by a relative integer coordinate $a \in \mathbb{Z}^d$, and mapped back to real-valued parameter space using:

$$\theta(a) = \theta^\star + Ba \cdot s,$$

where $B \in \mathbb{R}^{d \times m}$ is a basis matrix defining a subspace of interest. This allows us to restrict traversal to low-dimensional manifolds (e.g., 2D or 3D planes) while still operating in a high-dimensional ambient space.

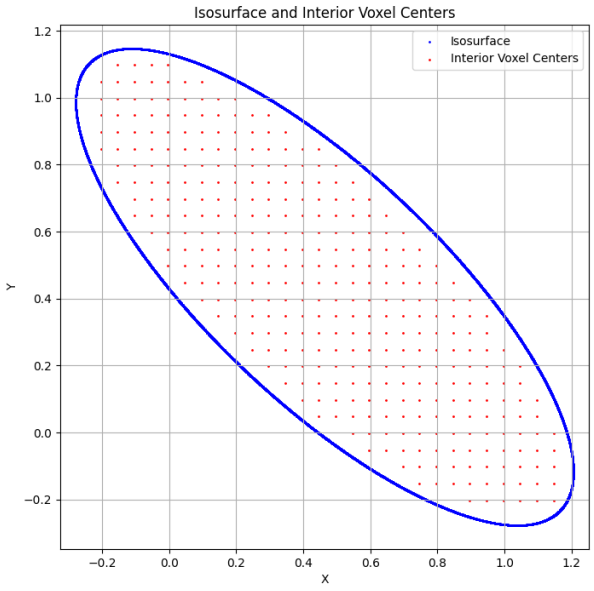


Figure 4.2: Voxel grid centered around the local minimum $\theta^\star$. Each voxel represents a discrete region of parameter space.

This abstraction enables traversal entirely in the discrete space $\mathbb{Z}^d$, avoiding floating-point instability and ensuring consistent comparison of voxel positions.

4.3.2 Exploration Criterion

To control the region of interest, we introduce a threshold value:

$$c = n \cdot \varepsilon,$$

where $n \in \mathbb{N}$ is a user-defined multiplier that sets the exploration radius in terms of acceptable loss increase. Only voxels whose mapped coordinates $\theta(a)$ satisfy $f(\theta(a)) \leq c$ are included in the region of interest.

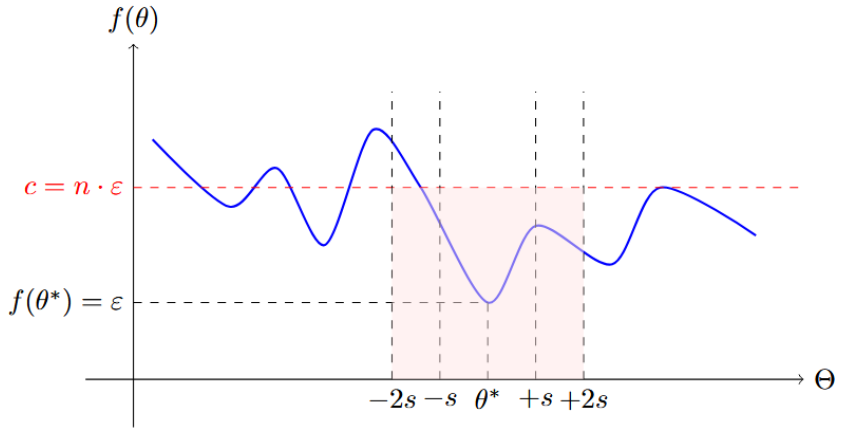


Figure 4.3: Illustration of local minimum ε and corresponding exploration threshold $c = n \cdot \varepsilon$. Voxels within this boundary are considered in the traversal.

4.3.3 Modified BFS Traversal

To explore this constrained region, we employ a modified breadth-first search (BFS) algorithm. Starting at the origin of the grid (corresponding to $\theta^\star$), we enqueue neighboring voxels whose loss lies below the threshold c. This process continues iteratively, expanding the set of reachable voxels. For a fixed threshold c, step size s, and base point $\theta^\star$, the set

$$\mathcal{R}_c(s) := \{a \in \mathbb{Z}^m \mid f(\theta^\star + Ba \cdot s) \leq c\}$$

defines the explored grid region.

Algorithm 3 Modified BFS for Grid Traversal

procedure GRIDTRAVERSAL(f, s, n, B, *queue*, *visited*, *dropout*, *limit*, *sampling_switch_iter*)
Inputs: f: loss function, s: step width, n: threshold multiplier, B: basis matrix or None, *queue*, *visited*: initial sets, *dropout*: skip probability, *limit*: max steps, *sampling_switch_iter*: pop switch step
Outputs: Visited voxels ($f(\theta) \leq c$), starting point $\theta^\star$, *future_queue* for next round
  iterations $\leftarrow 0$
  start_coord $\leftarrow \theta^\star$
  $c \leftarrow n \cdot f(\theta^\star)$
  if $B \neq$ None **then**
    rel_grid_pos $\leftarrow \vec{0} \in \mathbb{Z}^m$
  else
    rel_grid_pos $\leftarrow \vec{0} \in \mathbb{Z}^d$
  end if
  if *queue* = None **and** *visited* = None **then**
    queue $\leftarrow$ [*rel_grid_pos*]
    visited $\leftarrow \{$*rel_grid_pos*$\}$
  end if
  future_queue $\leftarrow$ empty list
  while *queue* not empty **and** *iterations* $<$ *limit* **do**
    if *iterations* $>$ *sampling_switch_iter* **then**
      curr_coord $\leftarrow$ randomly pop from *queue*
    else
      curr_coord $\leftarrow$ pop first from *queue*
    end if
    for all *neighbor* $\in \{$*curr_coord* $\pm\, e_i \mid i = 1, \ldots, m\}$ **do**
      if *neighbor* $\in$ *visited* **then continue**
      else if random() $<$ *dropout* **and** *iterations* $>$ *sampling_switch_iter* **then continue**
      else
        $\theta \leftarrow \theta^\star + B \cdot$ *neighbor* $\cdot\, s$
        if $f(\theta) \leq c$ **then**
          queue.append(*neighbor*)
        else
          future_queue.append(*neighbor*)
        end if
        visited.add(*neighbor*)
      end if
    end for
    iterations $\leftarrow$ *iterations* $+ 1$
  end while
  if *iterations* = *limit* **then return** 0
  end if
  return *visited*, $\theta^\star$, *future_queue*
end procedure

This algorithm is a variant of classical breadth-first search (BFS) adapted for voxel-based traversal in neural network parameter space. The ‘queue‘ tracks points to visit, ‘visited‘

records completed positions, and ‘future-queue‘ stores voxels that exceed the current loss threshold but may be explored in future iterations with larger c.

To reduce bias and cost, a stochastic dropout mechanism skips some voxels (with a fixed probability) after a certain number of iterations (*sampling_switch_iter*). Additionally, we occasionally replace the FIFO queue behavior with randomized dequeueing to avoid directional bias—particularly helpful when not all neighbors are processed.

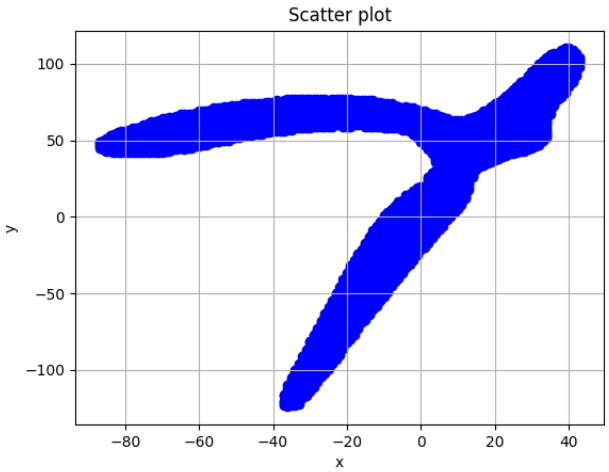


Figure 4.4: Convex hull enclosing all voxels satisfying $f(\theta) \leq c$ after traversal. Used for approximating the local loss basin.

4.3.4 Heuristic Evaluation via $H_c(s)$

For each traversal run, we compute the heuristic:

$$H_c(s) = \sum_{V \in \mathcal{V}} \mathbf{1}_{f(\theta_V) \leq c},$$

where $\mathcal{V}$ is the set of all grid voxels reachable by the BFS procedure. This count approximates the true volume $R_c(s)$. By analyzing the behavior of $H_c(s)$ as c varies—and across different values of s—we can infer structural properties of the loss landscape, such as local curvature, smoothness, or connectivity.

4.4 Wrapper

We utilize the PyTorch Lightning framework [FPLT] (based on PyTorch [Pas19]) to define and train our neural networks. While this framework offers a convenient and modular interface for high-level training procedures, it does not provide straightforward access for

operations that require manual intervention at the parameter level. Specifically, tasks such as extracting the current model parameters, modifying them directly, or evaluating the loss at an arbitrary parameter configuration are not natively supported in a flexible way.

To address these limitations, we implement a custom wrapper around the PyTorch Lightning model. This wrapper exposes essential low-level functionality required for loss landscape exploration. It provides:

- A method to retrieve the current model parameters as a flattened tensor.
- A method to assign a new set of parameters to the model.
- A method to evaluate the loss of the model for a given input and target pair, under arbitrary parameter configurations.

This wrapper enables us to interface with the loss function in a precise and controlled manner, which is critical for voxel-based exploration and perturbation analysis.

4.5 Discussion of Our Approach in Relation to Existing Methods

Our voxel-based heuristic exploration provides a distinct methodological alternative to previous works in the literature..

Unlike Li et al. [LXT+18], who primarily visualize loss landscapes through two-dimensional slices constructed by arbitrary interpolations, our approach systematically explores these landscapes using discrete voxel-based traversal. Rather than relying solely on qualitative visualizations, our heuristic $H_c(s)$ provides a direct quantitative characterization of local geometric structures, explicitly capturing features such as local curvature and connectivity.

Fort et al. [FHL19] focus on functional diversity and continuous interpolation paths between minima, analyzing the landscape structure through predictive disagreement metrics. In contrast, our heuristic operates purely in parameter space, employing discrete voxel coverage to characterize structural properties explicitly, without necessitating the evaluation of functional disagreement or continuous parameter interpolation.

Şimşek et al. [SGJ+21] theoretically analyze the global manifolds of minima arising from parameter-space symmetries and overparameterization. While their method identifies theoretical, symmetry-induced manifold structures explicitly, our heuristic does not depend on

symmetry and instead empirically quantifies local geometric connectivity through discrete voxel traversal, irrespective of explicit symmetry constraints.

Fort and Jastrzębski [FJ19] introduce a geometric model of "wedges," describing the global loss landscape as consisting of high-dimensional structures wide in certain directions and narrow in others. Their investigation, relying on continuous interpolations within random subspaces, contrasts with our method, which discretely explores explicitly defined subspaces using grid-based sampling. Our heuristic provides direct local geometric information, complementing their global wedge geometry insights.

Finally, Wu et al. [WZ+17] utilize spectral analysis of the Hessian eigenvalues to estimate basin volume, correlating these properties with generalization performance. While their method indirectly assesses landscape geometry via spectral properties, our discrete voxel-based approach explicitly quantifies local geometric characteristics without relying on Hessian computations, offering a complementary geometric perspective.

While prior work often quantifies curvature using the Hessian spectrum, our approach estimates the growth of low-loss volume using $H_c(s)$, a computationally simpler and more interpretable proxy for local landscape structure. This discrete, grid-based heuristic enables efficient and reproducible exploration of the loss surface, avoiding both interpolation artifacts and the computational cost of second-order methods. By emphasizing direct measurement of local connectivity and volume growth, our method complements and extends insights from existing continuous and theoretical approaches.

5 Implementation and Experimentation

In this chapter, we present our empirical approach to analyzing neural network loss landscapes using the voxel-based heuristic $H_c(s)$. This heuristic quantifies the number of voxels in parameter space that fall below a specified loss threshold c, based on a discretization of the landscape around a trained local minimum.

We interpret $H_c(s)$ as a resolution-dependent measure that correlates with the effective curvature and local flatness of the loss landscape. Sharp regions with steep walls tend to yield fewer reachable voxels for a fixed c, while flatter, more gradually changing regions yield higher counts. By systematically varying the parameters above, we aim to understand how landscape complexity is reflected in the shape of $H_c(s)$ curves across different architectures and configurations.

5.1 Experiments in Low Dimensions on synthetic data

To build an intuitive understanding of the behavior of our heuristic $H_c(s)$, we begin with a set of controlled experiments on synthetic data using small, non-linear feedforward neural networks. These networks are constructed with low-dimensional parameter spaces, typically ranging from 2 to d dimensions. Training on randomly generated datasets allows us to isolate the core geometric and topological properties of the loss landscape, without interference from the structure or noise of real-world data. We also used synthetic data for its simplicity: it provides a clean and controlled setting to begin our investigation, making it easier to understand and validate the behavior of our method before applying it to more complex, real-world cases.

After training each network to convergence, we evaluate the heuristic $H_c(s)$ for increasing values of the loss threshold $c = n \cdot \varepsilon$, where ε is the final loss at the local minimum $\theta^\star$. Since the parameter dimensionality is low, we directly analyze the full parameter space without projecting onto lower-dimensional subspaces (flats).

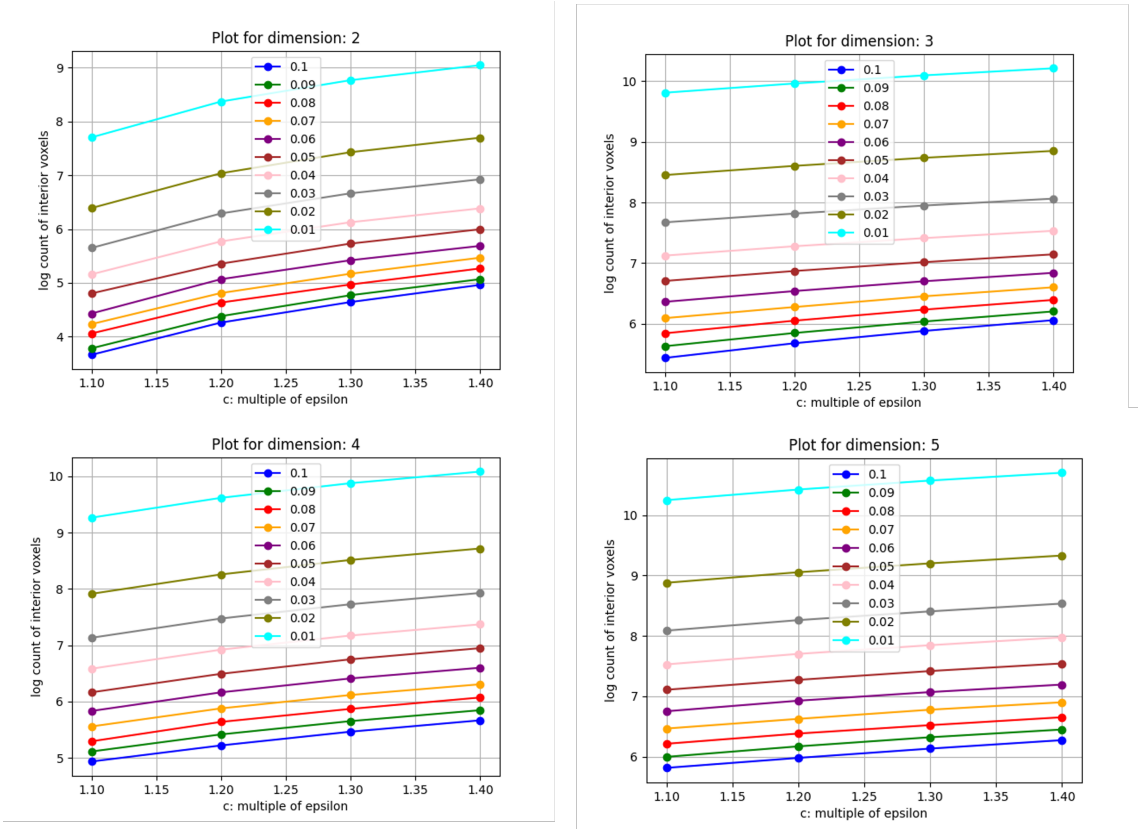


Figure 5.1: Heuristic $H_c(s)$ as a function of $c = n \cdot \varepsilon$ for synthetic networks in dimensions 2 through n. Each curve corresponds to a different step size s.

In the figure above, the x-axis shows the exploration threshold c, which determines how far we search from $\theta^\star$ in terms of permissible loss values. The y-axis shows the logarithm of the heuristic, $\log H_c(s)$, indicating the number of voxels satisfying the constraint $f(\theta) \leq c$ at a given resolution s.

Each curve in the figure corresponds to a fixed step width s, representing the number of interior voxels (in log scale) that satisfy the loss condition $f(\theta) \leq c$, where $c = n \cdot \varepsilon$ is a multiple of the minimum loss at the starting point $\theta^\star$. As c increases, the BFS traversal explores a broader region of the loss landscape, and the log-count of accepted voxels $\log H_c(s)$ increases accordingly. Notably, there are few sharp transitions or irregularities in $H_c(s)$, implying that at these low dimensions, the landscapes tend to be relatively simple and geometrically consistent.

This experiment serves as a baseline to which more complex, higher-dimensional behavior can later be compared.

5.2 Experiments on Real Data: Iris Dataset

To assess whether the structural behavior observed in synthetic landscapes also appears in real-world scenarios, we conduct a series of experiments using the Iris dataset—a widely studied benchmark for classification tasks. We train moderately sized feedforward neural networks on this dataset—specifically, two-layer architectures with 4 input neurons, one hidden layer of 8 ReLU units, and a single output neuron trained using cross-entropy loss—and explore their loss landscapes through low-dimensional projections.

Due to the higher dimensionality of real network parameter spaces, we restrict our exploration to 2D and 3D affine subspaces (referred to as flats). These flats are spanned by differences between independently trained local minima, ensuring relevance to meaningful regions of the loss landscape.

In each flat, we evaluate the heuristic $H_c(s)$ over a grid, for increasing values of the threshold $c = n \cdot \varepsilon$, where ε is the base loss at the local minimum $\theta^\star$, and s is the discretization step size. The objective is to observe whether the landscape maintains similar growth patterns in accessible volume, as indicated by $H_c(s)$, when compared to the synthetic setting. Each individual traversal may cover tens of millions of voxels, especially in higher-dimensional flats or with finer step sizes, which highlights the computational demands of our exhaustive discrete exploration.

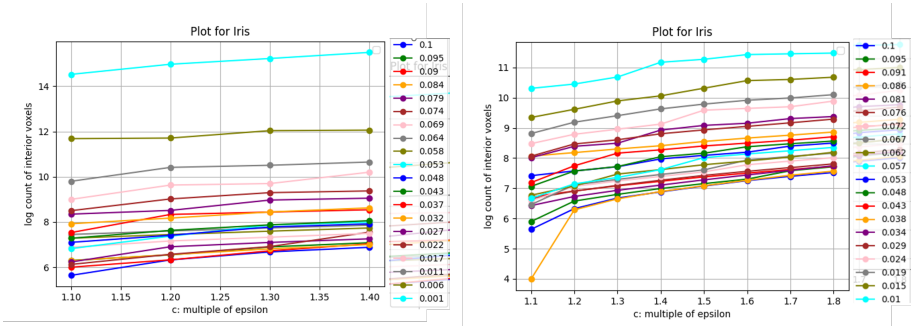


Figure 5.2: Heuristic $H_c(s)$ on the Iris dataset projected onto a 2D flat. Each line represents a different step size s.

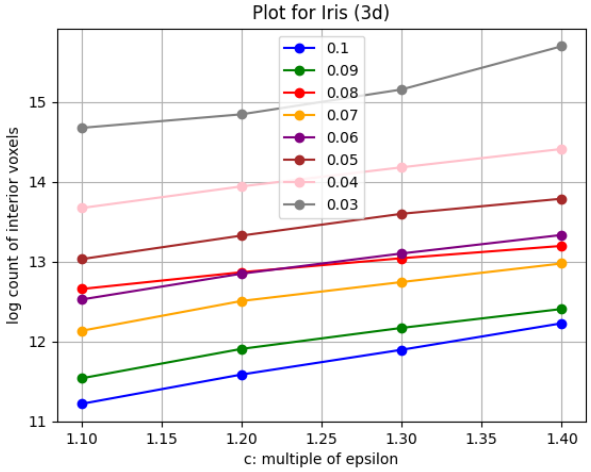


Figure 5.3: Heuristic $H_c(s)$ on the Iris dataset projected onto a 3D flat.

The results show that the behavior of $H_c(s)$ remains largely consistent with that observed in synthetic experiments. Specifically, we again observe a roughly linear trend in log-space across different step sizes s, with no dramatic inflections or anomalies.

5.3 Effect of Different Starting Minima

To further examine the robustness of our heuristic $H_c(s)$, we repeat our exploration on the same loss landscape but initialize the procedure from different local minima. These minima are obtained by training the same network architecture multiple times with different random seeds, resulting in distinct optimization trajectories and final parameter configurations.

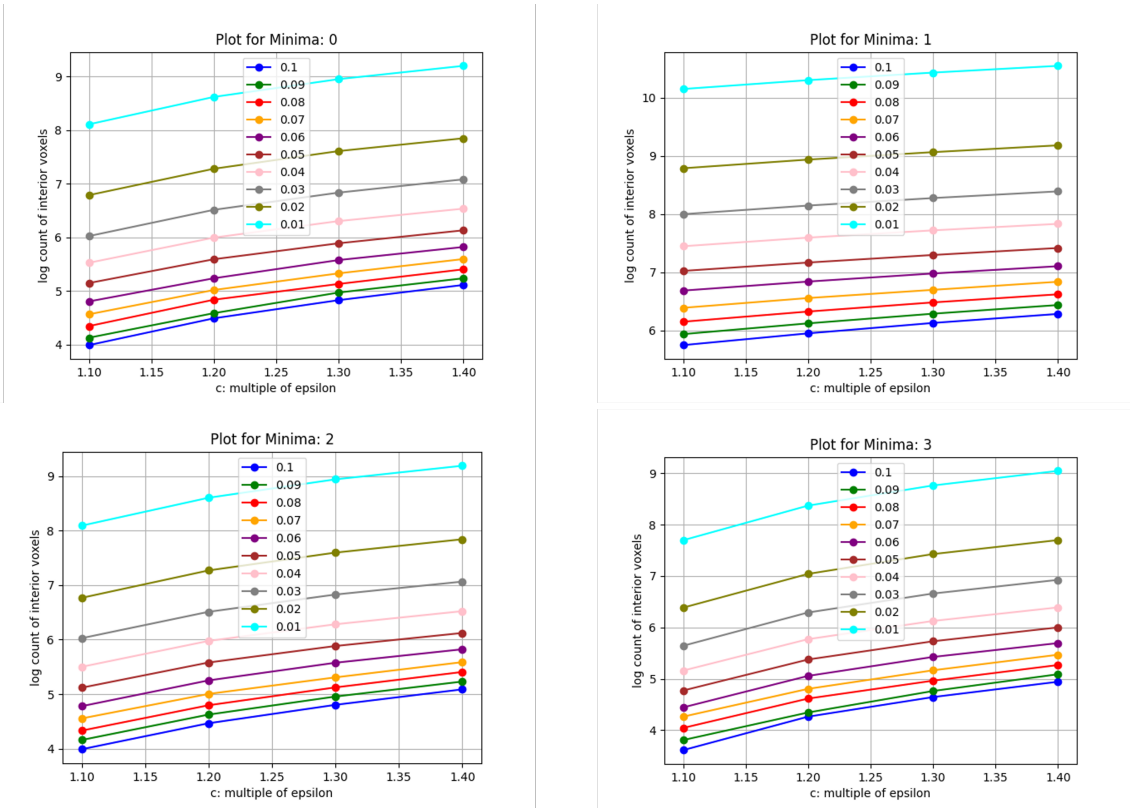


Figure 5.4: Comparison of $H_c(s)$ growth for different starting minima on the same loss landscape.

As shown in the figure, the overall growth behavior of $H_c(s)$ remains largely unchanged across different starting points. The curves exhibit similar slopes and trends, suggesting that the local curvature and volume growth of the loss landscape are relatively consistent within the explored region.

The only notable difference appears in the initial value of $H_c(s)$, which reflects the number of low-loss voxels immediately surrounding the respective starting minimum. This variation is expected, as different minima can vary slightly in shape or local diameter, leading to differences in how much of the surrounding region initially satisfies the loss threshold.

5.4 Effect of Basis Orthogonalization

In this experiment, we examine how the choice of basis orthogonalized vs. non orthogonalized affects the behavior of our heuristic $H_c(s)$. Using the same landscape and starting at the same local minimum, we compare two scenarios: one where the flat is defined using a raw basis from local minima differences, and one where that basis is orthogonalized via QR decomposition.

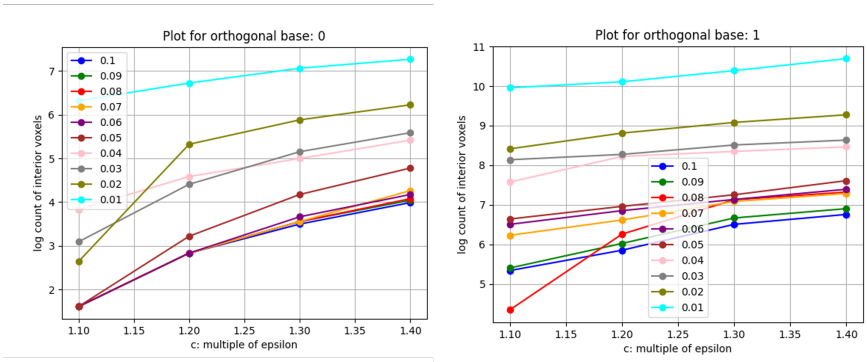


Figure 5.5: Comparison of $H_c(s)$ using non-orthogonal bases (left) vs. orthogonal (right).

The results show no significant change in the overall growth pattern of $H_c(s)$; however, the initial value of the heuristic is consistently lower when using a non-orthogonal basis.

This can be explained by geometric considerations: the flat spanned by an orthogonal basis tends to enclose a larger area (or volume in higher dimensions) for the same step size s. This is because cubes (or hypercubes) achieve maximal coverage when their axes are aligned at right angles—i.e., when their basis vectors are orthogonal. In contrast, skewed bases may lead to inefficient packing, where fewer axis-aligned voxels fit into the same subspace, reducing the effective resolution of the grid.

5.5 Distance-Based Exploration

In this experiment, we investigate how the heuristic $H_c(s)$ grows as we explore increasingly distant regions of the loss landscape. Given the relatively linear behavior observed in earlier experiments, our aim here is to test whether this trend persists even when exploring farther away from the local minimum.

To do this efficiently, we choose a large threshold c and a relatively large step size s. A larger s reduces the resolution of our exploration but allows us to traverse greater distances at a lower computational cost. We repeat this procedure from several different local minima and average the resulting values of $H_c(s)$ to ensure robustness of the results.

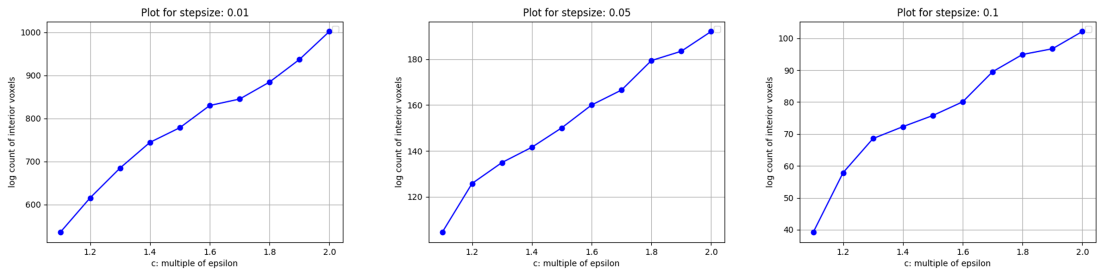


Figure 5.6: Averaged growth of $H_c(s)$ for large thresholds c and coarse step size s.

The results show a consistent pattern: aside from an initial jump—likely due to the immediate expansion into a dense local region—the heuristic continues to grow in a roughly linear fashion, even for large values of c that were previously impractical to explore at finer resolutions.

6 Results and Evaluation

In this chapter, we analyze and interpret the outcomes of our experiments. We focus on understanding what the results suggest about the structure of the loss landscape, and we discuss their broader implications and possible limitations.

6.1 Observed Patterns in $H_c(s)$

Throughout all experiments—ranging from low-dimensional synthetic networks to real-data flats (such as those derived from the Iris dataset), and across different initialization points—we consistently observed the same core phenomenon: when plotting $\log H_c(s)$ against c (with fixed s), the resulting curves are approximately linear. Each curve corresponds to a different fixed value of s, and the experiment is repeated for multiple step sizes at regularly spaced intervals.

This trend suggests that $H_c(s)$, the number of voxels under a fixed loss threshold c, grows exponentially as c increases. Specifically, we assume the form

$$H_c(s) \approx A(s)\, e^{B(s)c},$$

for some constants $A(s)$ and $B(s)$. In log-space, this corresponds to a linear relationship, where the slope $B(s)$ indicates the rate of volumetric expansion in the parameter space at the given resolution s.

To examine this, we applied linear regression to the empirical values of $\log H_c(s)$ over varying thresholds c, aggregating across different s. This yielded the following estimates for two experimental series (both conducted on the Iris dataset in a 2D flat, seen as plots in experimentation):

- Series 1: $A \approx 191.29$, $B \approx 1.989$, with correlation coefficient $R \approx 0.367$
- Series 2: $A \approx 117.19$, $B \approx 2.839$, with correlation coefficient $R \approx 0.023$

6.2 Interpretation and Implications

The appearance of smooth and consistent exponential growth in $H_c(s)$ may at first seem surprising, given the prevailing view of neural network loss landscapes as chaotic, non-convex surfaces with disconnected minima, sharp ridges, and high curvature. Yet the data across our experimental settings consistently supports this regularity—at least within the subspaces and resolutions we explored.

We propose several plausible explanations for this behavior:

- **Local Linearity.** The linearity may reflect a small neighborhood around the minimum $\theta^\star$, where the loss landscape behaves in an approximately convex or smooth manner. This is consistent with the intuition that if one "zooms in" close enough, most surfaces appear linear—analogous to a tangent plane on a manifold.

- **Coarse Grid Resolution.** Our choice of step size s imposes a minimum resolution on the structures we can observe. If s is too large relative to the curvature scale of the loss surface, the voxel-based search will average over finer variations, smoothing out sharp transitions and making the underlying geometry appear more regular than it actually is.

- **Subspace Projection Effects.** Because we perform our traversal within low-dimensional affine subspaces (i.e., flats), much of the global complexity of the full high-dimensional loss landscape may be projected out. In particular, if the subspace is constructed via interpolation between known minima, it may already lie in a region of low curvature or benign structure.

- **Model and Dataset Simplicity.** Many of our experiments involve relatively shallow networks and small datasets (such as the Iris dataset). These simpler scenarios may lack the scale and expressiveness required to induce highly fractured or degenerate regions in the loss landscape, resulting in more regular local geometry.

Taken together, these factors offer a coherent explanation for the surprising linearity of the $\log H_c(s)$ curves. While our analysis is limited to localized and discretized subspace views, the observed behavior suggests that such approximations can still reveal meaningful geometric structure—particularly in understanding the spread and density of low-loss regions.

6.3 Role of Different Initial Minima and Bases

Explorations from different trained minima showed almost identical slope behavior in the $\log H_c(s)$ curves. The only visible differences were vertical shifts, indicating variation in the local "width" or diameter of the valley, but not in the exponential growth rate itself.

Similarly, using orthogonal versus skewed bases to define the flats led to changes in the initial value of $H_c(s)$, but not in the qualitative growth pattern. The larger initial $H_c(s)$ observed with orthogonal bases is likely due to more efficient voxel coverage: cubes fit more evenly between orthogonal axes, and the voxel count scales better when directions are aligned.

This observation was confirmed by computing the angle between the unorthogonalized basis vectors using the formula:

$$\theta = \cos^{-1}\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \, \|\mathbf{v}_2\|}\right),$$

which resulted in an angle of approximately $53.78°$, indicating that the original basis vectors were not orthogonal.

6.4 Theoretical Justification

The exponential growth of $H_c(s)$ can be explained geometrically. Suppose the set of points satisfying $f(\theta) \leq c$ in some local flat region forms a roughly ball-shaped region with radius $r(c)$. Then the number of voxels of side length s needed to cover this region grows proportionally to the volume of the ball, which scales like:

$$H_c(s) \sim \left(\frac{r(c)}{s}\right)^d,$$

where d is the dimensionality of the flat. Since $r(c)$ typically increases smoothly with c, this directly explains why $H_c(s)$ appears to grow exponentially in c, and why $\log H_c(s)$ is linear.

6.5 Caveats and Practical Constraints

While our approach offers a computationally tractable way to analyse neural network loss landscapes, it also comes with several important limitations that must be acknowledged.

First, our method relies entirely on a discretized representation of the loss landscape. This abstraction, while useful for managing numerical stability and simplifying traversal, inevitably omits the fine-grained continuous structure of the underlying surface. As a result, sharp transitions, narrow valleys, or small-scale curvature features may be lost between sampled points, especially at larger step sizes.

Second, due to the high dimensionality of modern neural networks, we are restricted to examining low-dimensional subspaces (flats) of the full parameter space. These flats are constructed based on a fixed number of local minima, and although they are designed to capture representative directions in the landscape, they are only projections. Any behavior or structure orthogonal to the chosen subspace is, by design, inaccessible to our analysis.

Furthermore, the choice of subspace basis — even when orthogonalized — may introduce bias into the regions we explore. Similarly, the resolution of the grid (i.e., the step width s) imposes a trade-off between computational feasibility and landscape fidelity.

7 Conclusion

In this thesis, we proposed and implemented a novel method for traversing and analyzing neural network loss landscapes. Our approach is based on discretized exploration within lower-dimensional subspaces (flats), using a grid-based traversal algorithm. By avoiding continuous optimization and operating in a discrete, countable grid, our method allows for computationally efficient inspection of local geometry around trained minima.

We introduced the heuristic $H_c(s)$, which measures the number of voxels in a subspace for which the loss remains below a threshold c, with step size s. Across all experimental settings—ranging from synthetic landscapes to real datasets like Iris—we observed a consistent and nearly linear relationship between $\log H_c(s)$ and the threshold c. This indicates that the local loss landscape around a minimum expands in a predictable and relatively smooth fashion.

These findings challenge the common intuition derived from prior visualizations and interpolations, which often suggest rugged or fragmented structures. Instead, our results suggest that, at the scales and resolutions probed, the landscape is surprisingly well-behaved. Whether this is an artifact of resolution and subspace choice, or reflective of more general principles in high-dimensional optimization, remains an open and intriguing question.

Our study demonstrates that $H_c(s)$ might serve as a practical and interpretable tool to probe local curvature and structure in neural networks. It offers a complementary perspective to Hessian-based methods and path interpolation techniques seen in prior work.

Several avenues remain open. A natural next step is to apply our method to more complex networks and larger datasets, such as pretrained ResNet architectures or ImageNet-scale problems. Additionally, refining the step size s and threshold range c could allow for finer geometric insight, albeit at greater computational cost. Finally, reproducing existing interpolation-based studies using our method may help unify interpretations of loss landscape structure across different methodologies.

A Unsuccessful Methods

A.1 Marching Hypercubes Approach

An earlier approach in our exploration of the loss landscape was based on an extension of the classic Marching Cubes algorithm to higher dimensions, which we refer to as **Marching Hypercubes**. The motivation behind this method was to generate a hull around regions of interest in the loss landscape, particularly around local minima, without beginning our traversal at the minimum itself.

Instead of starting from a local minimum θ^*, we initialized the traversal at a level set where $f(\theta) = c$, i.e., a constant loss threshold. The idea was to identify the boundary of a sublevel set by marching along the iso-surface defined by this loss value. We reused the same voxel grid discretization as in our main method but inverted the logic of the traversal. For identifying edge intersections and approximate geometry, we employed a lookup table similar to the one proposed in [BWC00], generalized to high dimensions.

Although the basic edge classification was feasible, triangulation of the hypercube faces in higher dimensions remains nontrivial and is the subject of ongoing research. Therefore, we omitted explicit triangulation and instead collected all surface intersection points into a point cloud P, which served as an approximate representation of the hull.

Algorithm 4 Lookup Table Generation for Marching Hypercubes

Require: Dimension $d \geq 1$
Ensure: Lookup table T mapping case indices to edge intersections

$T \leftarrow \emptyset$ ▷ Initialize empty lookup table
$n \leftarrow 2^d$ ▷ Number of vertices in a d-hypercube
$m \leftarrow 2^n$ ▷ Total possible vertex labelings
for case_index $= 0$ **to** $m - 1$ **do**
    signs $\leftarrow$ [(case_index $\gg i$) & 1 for $i \in 0..n-1$]
    edges $\leftarrow \emptyset$
    for all $(i, j) \in$ Combinations$(0..n-1, 2)$ **do**
        if HammingDistance$(i, j) = 1$ **and** signs$[i] \neq$ signs$[j]$ **then**
            edges.append$((i, j))$ ▷ Store edge with sign change
        end if
    end for
    T[case_index] $\leftarrow$ edges ▷ Map case to intersecting edges
end for
return T

Notations:

- $\gg$: Bitwise right shift, &: Bitwise AND
- HammingDistance(i, j): Count of differing bits in binary representations of i and j
- Combinations(S, k): All k-combinations of set S

A.2 Flood Fill Inside Convex Hull

To estimate $H_c(s)$ based on this method, we attempted to traverse the interior of the approximate hull defined by the point cloud P. This was done using a flood fill algorithm on a second, coarser grid. The idea was to use this coarser resolution to explore the volume inside the hull and count all interior voxels, analogous to our earlier BFS approach.

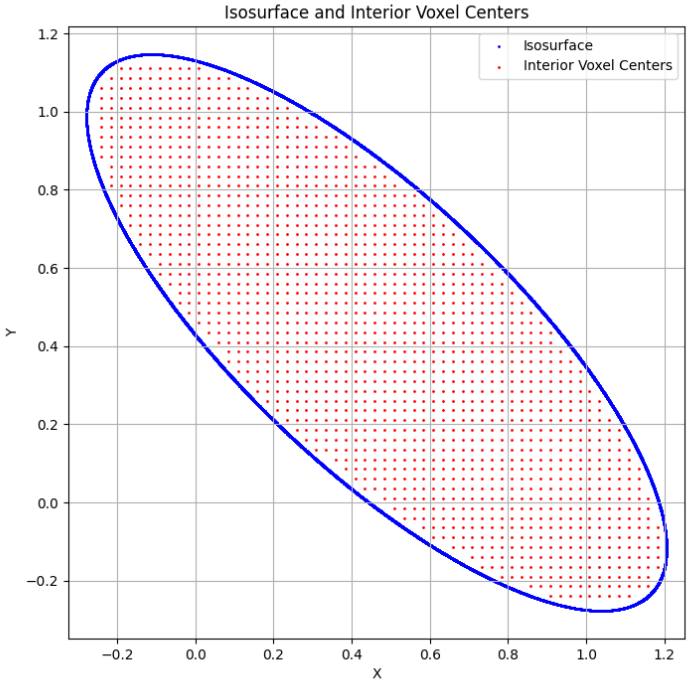


Figure A.1: Flood fill of the convex hull interior on a coarse grid.

However, this approach presented several problems:

- The point cloud P derived from the marching algorithm did not represent a watertight or topologically valid boundary, especially at finer resolutions. This led to leakage during the flood fill step.

- The lack of an explicit surface mesh prevented us from using precise spatial queries to detect whether a voxel was inside or outside the shape.

- To mitigate leakage, we constructed a convex hull from P using the `ConvexHull` implementation from `scipy.spatial`. While this gave us a well-defined closed volume, it represented an over-approximation of the true boundary.

- As a consequence, the final value of H computed from this method was based on an approximation of an approximation: a convex hull fit to an isosurface approximation, introducing significant uncertainty.

Although conceptually appealing, the Marching Hypercubes combined with convex hull flood fill was ultimately deemed too imprecise for our purpose, and we returned to the BFS-based approach outlined in the main methodology.

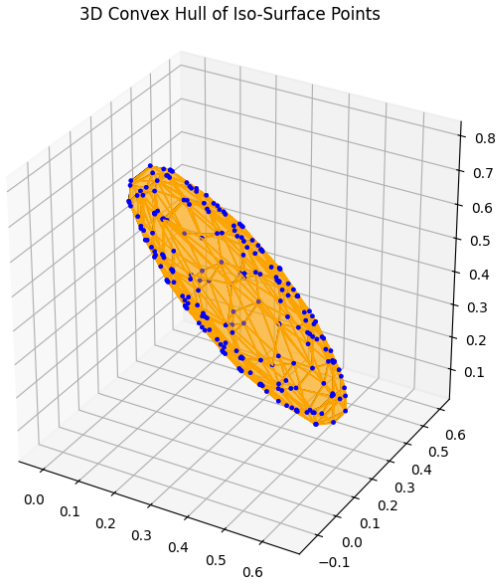


Figure A.2: Comparison of point cloud from marching method and resulting convex hull approximation.

Other major problems included the sparsity and quality of the point cloud P, which was highly sensitive to the choice of hyperparameters used in the marching hypercubes procedure. In cases where these parameters were poorly chosen, P did not contain enough representative surface points to form a meaningful or complete hull.

Furthermore, the lookup table required for marching hypercubes scales exponentially with the dimensionality of the parameter space, with a theoretical complexity of $\mathcal{O}((2^n)^n)$, where n is the number of dimensions. This makes the approach infeasible for networks with even moderate parameter dimensionality.

In contrast, our revised BFS-based approach (described in the main methodology) consistently produces reliable results across various settings and scales significantly better with respect to memory and computational complexity. It avoids the need for explicit surface triangulation and provides a more direct and stable mechanism for estimating loss landscape structure.

Bibliography

[Axl24] Axler, Sheldon: *Linear algebra done right.* Springer Nature, 2024

[BWC00] Bhaniramka, Praveen ; Wenger, Rephael ; Crawfis, Roger: Isosurfacing in higher dimensions. In: *Proceedings visualization 2000. vis 2000 (cat. no. 00ch37145)* IEEE, 2000, S. 267–273

[CL87] Cline, Harvey E. ; Lorensen, William E.: *System and method for the display of surface structures contained within the interior region of a solid body.* Dezember 1 1987. – US Patent 4,710,876

[CLRS22] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to algorithms.* MIT press, 2022

[Dav75] Davis, Philip J.: *Interpolation and approximation.* Courier Corporation, 1975

[FB85] Fishkin, Kenneth P. ; Barsky, Brian A.: An analysis and algorithm for filling propagation. In: *Computer-Generated Images: The State of the Art Proceedings of Graphics Interface'85* Springer, 1985, S. 56–76

[FHL19] Fort, Stanislav ; Hu, Huiyi ; Lakshminarayanan, Balaji: Deep ensembles: A loss landscape perspective. In: *arXiv preprint arXiv:1912.02757* (2019)

[FJ19] Fort, Stanislav ; Jastrzebski, Stanislaw: Large scale structure of neural network loss landscapes. In: *Advances in Neural Information Processing Systems* 32 (2019)

[flo] *Illustration of the flood-fill algorithm.* `https://www.researchgate.net/figure/Illustration-of-the-flood-fill-algorithm_fig3_320906028`

[FPLT] Falcon, William A. ; PyTorch Lightning Team the: *PyTorch Lightning.* `https://github.com/Lightning-AI/pytorch-lightning`

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning.* MIT Press, 2016. – `http://www.deeplearningbook.org`

[GIP+18] Garipov, Timur ; Izmailov, Pavel ; Podoprikhin, Dmitrii ; Vetrov, Dmitry P. ; Wilson, Andrew G.: Loss surfaces, mode connectivity, and fast ensembling of dnns. In: *Advances in neural information processing systems* 31 (2018)

[GVL13] Golub, Gene H. ; Van Loan, Charles F.: *Matrix computations*. JHU press, 2013

[LXT+18] Li, Hao ; Xu, Zheng ; Taylor, Gavin ; Studer, Christoph ; Goldstein, Tom: Visualizing the loss landscape of neural nets. In: *Advances in neural information processing systems* 31 (2018)

[Pas19] Paszke, A: Pytorch: An imperative style, high-performance deep learning library. In: *arXiv preprint arXiv:1912.01703* (2019)

[SGJ+21] Simsek, Berfin ; Ged, François ; Jacot, Arthur ; Spadaro, Francesco ; Hongler, Clément ; Gerstner, Wulfram ; Brea, Johanni: Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In: *International Conference on Machine Learning* PMLR, 2021, S. 9722–9732

[Wika] Wikipedia contributors: *Breadth-first search — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Breadth-first_search`

[Wikb] Wikipedia contributors: *Marching cubes — Wikipedia, The Free Encyclopedia*. `https://en.wikipedia.org/wiki/Marching_cubes`

[WZ+17] Wu, Lei ; Zhu, Zhanxing u. a.: Towards understanding generalization of deep learning: Perspective of loss landscapes. In: *arXiv preprint arXiv:1706.10239* (2017)

List of Figures

Erklärung zur Nutzung von Künstlicher Intelligenz

Im Rahmen der Erstellung dieser wissenschaftlichen Arbeit wurde unterstützend auf Künstliche Intelligenz (KI) in Form eines Sprachmodells zurückgegriffen, wie DeepL, ChatGPT und Perplexity. Die KI wurde ausschließlich zur sprachlichen Überarbeitung, Strukturierung und Ideenfindung verwendet.

Es erfolgten keine automatisierten Generierungen von Inhalten, die ohne eigene inhaltliche Prüfung oder Bearbeitung übernommen wurden. Sämtliche wissenschaftlichen Inhalte, Argumentationen sowie Interpretationen stammen von der Verfasserin/dem Verfasser selbst und entsprechen den Anforderungen an wissenschaftliches Arbeiten.

Die Nutzung der KI diente somit lediglich als unterstützendes Werkzeug im Schreibprozess, vergleichbar mit einem digitalen Korrektur- oder Inspirationshilfsmittel. Eine umfassende Quellenrecherche, Analyse sowie die Bewertung der Literatur wurden eigenständig durchgeführt.

Kassel, June 16, 2025

Noah Schager

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur mit den nach der Prüfungsordnung der Universität Kassel zulässigen Hilfsmitteln angefertigt habe. Die verwendete Literatur ist im Literaturverzeichnis angegeben. Wörtlich oder sinngemäß übernommene Inhalte habe ich als solche kenntlich gemacht.

Kassel, June 16, 2025

Noah Schager