Collatz Sequence-Based Weight Initialization for Enhanced Convergence and Gradient Stability in Neural Networks

By

Zehara Eshetu Seid

Submitted to the School of Information Technology and Engineering
In partial fulfillment of the requirements for the degree of
Master of Science in Artificial Intelligence

Supervised by: Dr. Beakal Gizachew
Dr. Adane Letta

School of Information Technology and Engineering
College of Technology and Built Environment
Addis Ababa, Ethiopia

June 26, 2025

Addis Ababa University

Collage of Technology and Built Environmnet

School of Information Technology and Engineering

This is to certify that the thesis prepared by Zehara Eshetu , entitled "**Collatz Sequence-Based Weight Initialization for Enhanced Convergence and Gradient Stability in Deep Neural Networks"**, Submitted to the School of Information Technology and Engineering In partial fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence.

Approved by Board of Examiners

Name	Signature	Date
Dr. Henok Mulugeta	_____	_____
(Chairman)		
Dr. Beakal Gizachew	_____	_____
(Advisor)		
Dr. Adane Letta	_____	_____
(Advisor)		
Dr. Mesfin Abebe	_____	_____
(External Examiner)		
Dr. Fantahun Bogale	_____	_____
(Internal Examiner)		

Declaration

I declare that this thesis entitled "Collatz Sequence-Based Weight Initialization for Enhanced Convergence and Gradient Stability in Neural Networks" is my original work and has not been submitted to any other institution for the award of any degree or diploma. All sources of materials used in the preparation of this thesis have been properly acknowledged.

Zahara Eshetu		June, 2025
Name of Student	**Signature**	**Submission Date**

This thesis has been submitted for examination with our approval as university advisor.

Name of Advisors	**Signature**
Dr. Adane Letta	
Dr. Beakal Gizachew	

Acknowledgements

First and foremost, I would like to thank **Allah**. He has given me strength and encouragement throughout all the challenging moments of completing this thesis. I am truly grateful for His unconditional and endless love, mercy, and grace.

I would like to express my deepest gratitude to my advisors, **Dr. Adane Letta** and **Dr. Beakal Gizachew** , for their exceptional guidance, insightful feedback, and unwavering support throughout the course of this research. Their mentorship has been instrumental in shaping both the direction and quality of this thesis.

A very special thanks goes to my loving husband, **Mensur Temam**, for his endless love, understanding, and continuous motivation. His unwavering support has been my greatest source of inspiration and resilience.

I am also sincerely thankful to my senior and friend, **Eyob Solomon** for his guidance and support throughout this research. His thoughtful explanations, willingness to share resources, and patient responses to my many questions greatly enriched my understanding and helped me stay on track. His mentorship, especially during difficult phases of the work, was truly invaluable.

I am equally thankful to my beloved family and friends, whose constant encouragement, patience, and belief in me have been a pillar of strength during my academic journey.This thesis would not have been possible without the collective contributions of these remarkable individuals. To all of you, I am sincerely grateful.

Zahara Eshetu

Abstract

Deep neural networks have achieved state-of-the-art performance in tasks ranging from image classification to regression. However, their training dynamics remain highly sensitive to weight initialization. This is a fundamental factor that influences both convergence speed and model performance. Traditional initialization methods such as Xavier and He rely on fixed statistical distributions and often underperform when applied across diverse architectures and datasets. This study introduces Collatz Sequence-Based Weight Initialization, a novel deterministic approach that leverages the structured chaos of Collatz sequences to generate initial weights. CSB applies systematic transformations and scaling strategies to improve gradient flow and enhance training stability. It is evaluated against seven baseline initialization techniques using a CNN on the CIFAR-10 dataset and an MLP on the California Housing dataset. Results show that CSB consistently outperforms conventional methods in both convergence speed and final performance. Specifically, CSB achieves up to 55.03% faster convergence than Xavier and 18.49% faster than He on a 1,000-sample subset, and maintains a 20.64% speed advantage over Xavier on the full CIFAR-10 dataset. On the MLP, CSB shows a 58.12% improvement in convergence speed over He. Beyond convergence, CSB achieves a test accuracy of 78.12% on CIFAR-10, outperforming Xavier by 1.53% and He by 1.34%. On the California Housing dataset, CSB attains an R^2 score of 0.7888, marking a 2.35% improvement over Xavier. Gradient analysis reveals that CSB-initialized networks maintain balanced L2 norms across layers, effectively reducing vanishing and exploding gradient issues. This stability contributes to more reliable training dynamics and improved generalization.However, this study is limited by its focus on shallow architectures and lacks a robustness analysis across diverse hyperparameter settings.

Keywords: Weight initialization, Collatz conjecture, Neural networks, Convergence stability, Deep learning.

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CSB	Collatz Sequence Based
ELU	Exponential Linear Unit
LDA	Linear Discriminant Analysis
LSUV	Layer-Sequential Unit Variance
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
N	Normal Distribution
PCA	Principal Component Analysis
QR	Orthogonal-triangular Decomposition (QR Decomposition)
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SB	Between-Class Scatter Matrix
SGD	Stochastic Gradient Descent
SW	Within-Class Scatter Matrix
U	Uniform Distribution

Chapter 1

Introduction

1.1 Background of the Study

Deep learning has emerged as a transformative force in artificial intelligence[1], enabling breakthroughs in fields such as computer vision[2], natural language processing[3], autonomous systems[4], and medical diagnostics[5]. At the heart of deep learning are artificial neural networks (ANNs)[1, 6], computational models inspired by the structure and function of the human brain. These networks consist of multiple layers of interconnected artificial neurons that process and transform data, allowing machines to recognize patterns, make predictions, and perform complex decision-making tasks[6]. Unlike traditional machine learning models that rely on feature engineering[7], neural networks learn hierarchical representations of data, automatically extracting meaningful patterns from raw inputs[1]. However, the success of these networks is not solely determined by their architecture but also by the initialization of their weights, which plays a crucial role in ensuring stable and efficient training[1, 6].

Neural networks operate by passing data through a series of layers, each consisting of multipleneurons that apply mathematical transformations to their inputs[8, 9]. These neurons are connected through weighted links, and the strength of each connection is represented by a weight parameter[8, 9]. The learning process involves adjusting these weights using an optimization algorithm (e.g., Stochastic Gradient Descent, Adam) to minimize the difference between the network's predictions and the actual target values[8]. This process, known as backpropagation[10], updates the weights iteratively by computing gradients through the network. However, for backpropagation to be effective, the initial values of the weights must be carefully chosen; otherwise, the network may suffer from slow convergence, unstable gradients, or poor generalization[8, 10].

1.1.1 Neural Network Architecture

A standard **feedforward neural network** consists of three types of layers:

- **Input Layer:** Receives raw input data (e.g., images, text, numerical data).

- **Hidden Layers:** Perform intermediate computations, extracting features and transforming data representations.
- **Output Layer:** Produces final predictions (e.g., classification labels, regression values).

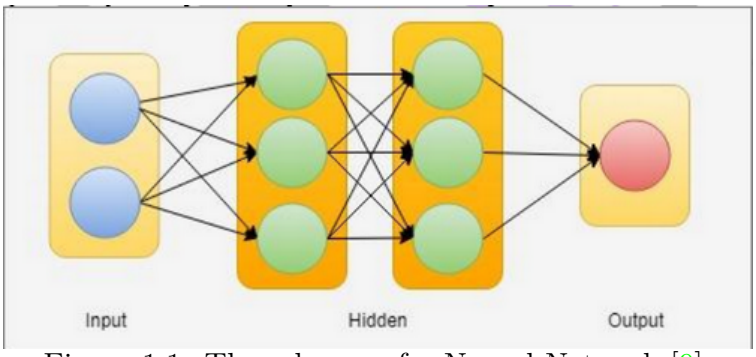

Figure 1.1: Three layers of a Neural Network [9].

Each node in the network has some weights assigned to it. A transfer function is used to calculate the weighted sum of inputs and a bias is added as shown in fig 1.2

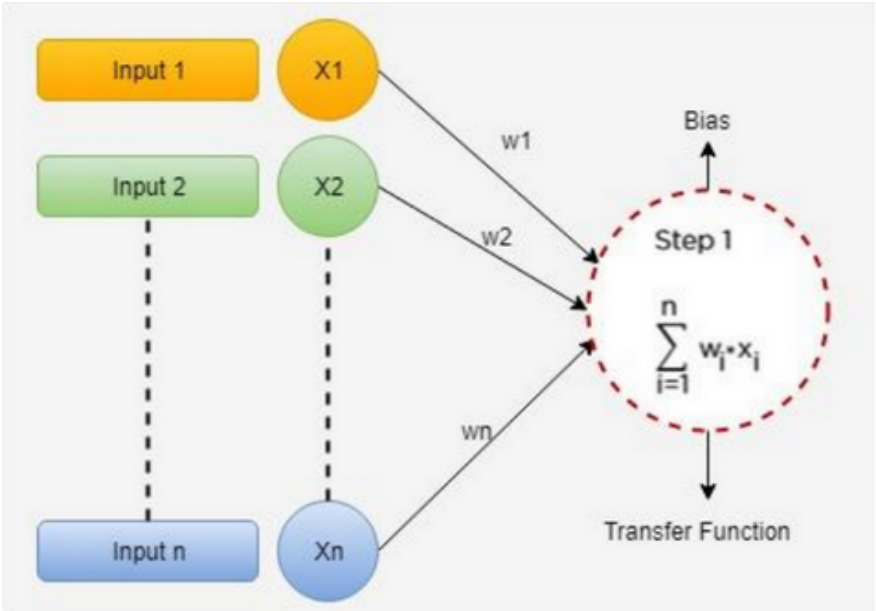

Figure 1.2: Weights and transfer function [9]

Each neuron within a layer applies an activation function, such as ReLU (Rectified Linear Unit)[11], sigmoid[12], or tanh[13], which determines how signals propagate through the network. The activation function introduces non-linearity, enabling neural networks to model complex relationships beyond simple linear transformations.

While deep networks have the potential to learn highly complex functions, their performance is highly sensitive to weight initialization strategies[14, 15, 16]. Poorly initialized

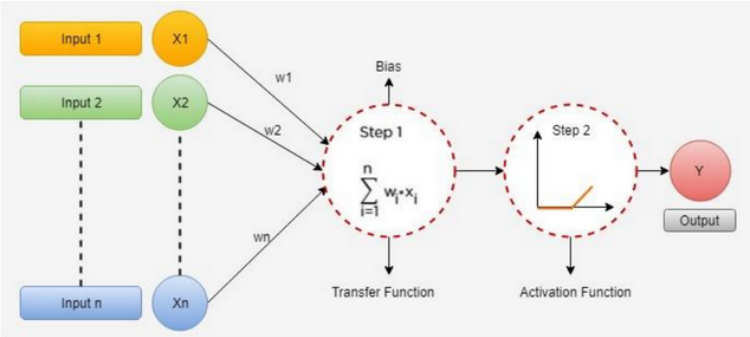

Figure 1.3: Activation function gives the output [9]

weights can cause issues such as vanishing gradients, where gradients become too small to update weights effectively, or exploding gradients, where weights grow uncontrollably, leading to unstable learning[17]. These challenges are particularly evident in deep architectures, where errors accumulate as they propagate through many layers[17]. To address these issues, several weight initialization techniques have been proposed, including Xavier initialization[18], He initialization[19], and orthogonal initialization[20], each designed to stabilize the learning process.

Xavier initialization, for instance, balances variance across layers, ensuring that activations neither shrink nor explode as they pass through the network[18]. He initialization, on the other hand, is specifically designed for ReLU-based networks, adjusting weight magnitudes to accommodate the properties of the activation function[19].

One promising approach is Collatz Sequence Based Weight Initialization, which draws inspiration from the chaotic yet structured behavior of the Collatz conjecture[21, 22]. The Collatz function, defined as:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases} \tag{1.1}$$

produces sequences that exhibit deterministic randomness, meaning that while the sequence is unpredictable in the short term, it follows well-defined mathematical rules[21, 22].. By integrating Collatz-based transformations into weight initialization, neural networks can benefit from structured randomness, which allows for better weight dispersion and prevents premature convergence to poor local minima. This method introduces controlled variability in initial weights, ensuring that gradients remain well-conditioned throughout training. As deep learning models continue to evolve, the need for robust, adaptable weight initialization strategies becomes increasingly critical.

1.2 Motivation of the Study

Training deep neural networks efficiently is a challenging task, and weight initialization plays a crucial role in determining how well and how fast a model learns[14, 15, 16]. Traditional initialization methods, such as Xavier [18] and He initialization [19], follow fixed statistical assumptions, which may not always be the best fit for different architectures and tasks. When weights are not initialized properly, networks can suffer from vanishing or exploding gradients, leading to slow training, unstable optimization, and poor generalization.

Several key factors motivate the exploration of Collatz Sequence Based Weight Initialization as an alternative approach:

- The need for faster and more stable convergence in neural networks for improving training efficiency..
- The computational burden of prolonged training, emphasizing the importance of efficient initialization strategies.
- The potential of chaotic mathematical structures, such as the Collatz sequence, to introduce structured randomness in weight initialization.
- The goal of improving gradient flow and reducing optimization difficulties, leading to better network generalization.
- The broader need for scalable and adaptive weight initialization techniques that enhance the performance of deep learning models.

this study introduces Collatz Sequence Based Weight Initialization as a novel approach that aims to improve training efficiency, stability, and overall model performance.

1.3 Problem Statement

Weight initialization plays a crucial role in determining the training efficiency and generalization ability of deep neural networks [14, 15, 16]. An improper initialization strategy can lead to slow convergence, poor optimization, and performance degradation due to vanishing or exploding gradients [23]. Traditional weight initialization methods, such as Xavier [18] and He initialization [19], aim to stabilize variance across layers but rely on fixed statistical assumptions. While effective in many cases, these methods do not dynamically adapt to the evolving optimization landscape during training, often resulting in poor weight exploration, sensitivity to hyperparameter settings, and suboptimal generalization [24].

To address these limitations, researchers have explored alternative initialization techniques. Orthogonal initialization [25] ensures stable gradient propagation but does not introduce controlled randomness necessary for improved exploration[25]. Data-dependent initialization methods, such as those introduced by Krähenbühl et al. [26], attempt to optimize weight initialization based on dataset specific properties, but they introduce additional computational overhead and may not generalize well across different tasks [26]. Sparse initialization [27] has been proposed to improve network training by reducing redundancy in weight connections, yet its effectiveness depends on architectural constraints[27]. Despite these advancements, striking a balance between exploration (diversity in initial weights) and stability (avoiding extreme weight distributions) remains an open problem in deep learning[14, 15].

This research aims to address these gaps by developing and implementing collatz sequence based weight intilization by focusing on its effects on convergence rate, training stability, and model generalization, comparing it with existing techniques across both CNN-based classification and MLP-based regression tasks. By bridging mathematical chaos theory with deep learning optimization, this research contributes to the ongoing advancements in neural network training methodologies.

This study addressed the following key research questions:

RQ1: How does Collatz Sequence Based Weight Initialization affect the convergence speed and training stability of neural networks?

RQ2: What impact does **CSB** have on the overall performance and generalization capability of neural networks across different architectures?

1.4 Objectives of the Study

1.4.1 General Objective

The general objective of the study is to apply and evaluate Collatz Sequence Based Weight Initialization technique aimed at improving convergence speed, training stability, and generalization performance in neural networks.

1.4.2 Specific Objectives

- To develop and implement the Collatz Sequence Based Weight Initialization algorithm based on the principles of the Collatz sequence.
- To integrate CSB into standard neural network architectures

- To compare its effect with conventional initializers such as Xavier, He, LeCun, Orthogonal, Sparse,Random and Zero initialization.
- To evaluate CSB's impact on key training metrics such as convergence speed, training stability,weight distribution and loss minimization across multiple neural network models..
- To assess the generalization performance of CSB-initialized models on benchmark datasets and investigate its potential to reduce vanishing or exploding gradients.

1.5 Significance of the Study

This research introduces a novel paradigm in neural network initialization by harnessing the structured chaos of the Collatz sequence. The proposed Collatz Sequence Based Weight Initialization method challenges conventional fixed distribution approaches, offering a dynamic and theoretically grounded alternative that enhances training behavior across architectures.

The potential significance of this research lies in:

- **Accelerated Convergence:** CSB injects structured randomness that improves initial exploration and helps networks converge faster and more reliably, reducing training time and computational cost.
- **Improved Gradient Flow & Stability:** By mitigating vanishing and exploding gradients, CSB fosters stable optimization, enabling deeper networks to train more effectively.
- **Generalization Capability:** Enhanced initialization contributes to better generalization on unseen data, a hallmark of robust and trustworthy models.
- **Broad Architectural Applicability:** CSB has been successfully evaluated across both convolutional and feedforward architectures, showing promise as a versatile initialization strategy for diverse deep learning tasks.
- **Theoretical and Empirical Innovation:** The integration of mathematical dynamics into weight initialization opens new pathways in the theory of neural optimization, blending concepts from chaos theory with practical deep learning.

1.6 Contribution of the Study

This thesis makes a foundational contribution to the field of neural network optimization by introducing a new weight initialization algorithm Collatz Sequence Based Weight Initialization grounded in deterministic number theory rather than conventional probabilistic assumptions. The key contributions of this work include:

- **Introduction of a Mathematically-Driven Initialization Framework:** CSB applies recursive transformations inspired by the Collatz conjecture to construct a structured weight initialization scheme. Unlike traditional methods that draw weights from predefined probability distributions (e.g., Gaussian, uniform), **CSB** uses integer-based iterative dynamics to achieve a controlled distribution with embedded variability. This injects mathematical structure into neural training from the very first epoch.

- **Challenging the Probabilistic Paradigm of Initialization:** Most existing initializers (Xavier, He, LeCun, etc.) rely on static variance preserving heuristics derived from linear activation assumptions. **CSB** breaks from this by embedding dynamic, nonlinear behavior into the weight initialization process. It shows that deterministic sequences can yield highly diverse, normalized weight sets without relying on statistical sampling.

- **Unified Application Across Network Types and Datasets:** **CSB** is evaluated across multiple tasks and domains including image classification with CNNs and regression with MLPs demonstrating its versatility and adaptability. This breadth of evaluation establishes **CSB** not only as an academic curiosity but as a practically viable initializer across architectures.

- **Demonstrated Superiority in Convergence Speed:** In controlled experiments, **CSB** consistently achieves faster convergence than traditional methods. For instance, it reduces training time to a fixed loss threshold by up to 20-58% percent compared to Xavier and He initialization. These gains are accompanied by reduced variance across multiple runs, indicating greater stability and robustness in early training dynamics.

- **Implementation Simplicity and Zero Tuning Advantage:** Despite its mathematical novelty, **CSB** requires no additional hyperparameters or architectural changes. It can be seamlessly integrated into standard deep learning pipelines with minimal code modification, making it immediately accessible to practitioners and researchers alike.

1.7 Scope of the Study

This study focuses on the design, implementation, and empirical evaluation of the proposed Collatz Sequence Based Weight Initialization method for neural networks. The research scope is limited to supervised learning tasks and covers two core neural architectures: Convolutional Neural Networks (CNNs) for image classification using the CIFAR-10 datasets, and Multi-Layer Perceptrons (MLPs) for regression using the California Housing dataset.
The performance of **CSB** is benchmarked against widely used initialization techniques, including Xavier, He, Orthogonal, Sparse, LeCun, Random, and Zero initializations. Evaluation metrics include convergence speed, training loss, test loss, and generalization performance (R^2 score or accuracy, depending on the task).
This study does not extend to reinforcement learning, unsupervised learning, or emerging architectures such as transformers or large language models. The primary focus remains on establishing the viability of **CSB** within standard feedforward and convolutional frameworks.

1.8 Thesis Structure

This thesis is structured as follows: **Chapter 1** introduces the theoretical background of deep learning and weight initialization , presents the problem statement, research questions, and objectives of the study. **Chapter 2** provides a comprehensive literature review on coolatz conjecture and its mathematical property weight initialization in neural network, techniques, including existing methods and related works relevant to the proposed approach. **Chapter 3** details the research methodology, outlining the proposed solution, its mathematical formulation, dataset selection, and justification for how the method addresses the stated problem. **Chapter 4** covers the experimental setup, implementation details, and performance evaluation, followed by a discussion of results. Finally, **Chapter 5** presents the conclusion and potential directions for future research.

Chapter 2

Literature Review

2.1 Foundations

This section will discuss the important foundations that are needed for the proposed solution.

2.2 The Collatz Conjecture and Its Properties

2.2.1 Definition

The *Collatz Conjecture*, also known as the *3x + 1 problem*, is a famous unsolved problem in mathematics. It was first introduced by Lothar Collatz in 1937 [21, 22]. The problem is simple to state yet remarkably difficult to prove, making it a fundamental topic in number theory and computational mathematics.

The Collatz function is defined as follows: given a positive integer n, the next number in the sequence is determined by the rule:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases} \tag{2.1}$$

where $f(n)$ represents the next term in the sequence. The conjecture states that by repeatedly applying this function, any positive integer n will eventually reach 1.

For example, the sequence for $n = 6$ is:

$$6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1 \tag{2.2}$$

Similarly, for $n = 7$:

$$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1 \tag{2.3}$$

Mathematically, the sequence can be expressed using modular arithmetic:

$$n_{k+1} = \begin{cases} \frac{n_k}{2}, & \text{if } n_k \equiv 0 \pmod 2 \\ 3n_k + 1, & \text{if } n_k \equiv 1 \pmod 2 \end{cases} \tag{2.4}$$

Despite being tested for numbers up to 2^{68}, no general proof has been found for all natural numbers [22].

The Collatz conjecture is closely related to dynamical systems, chaos theory, and computational complexity. Some studies have attempted to analyze it using stochastic processes and Markov chains [28]. Variations of the problem, such as the generalized Collatz function, explore different coefficients in the transformation rule, affecting convergence properties [21, 22, 28].

2.2.2 Mathematical Properties

The Collatz Conjecture is characterized by several intriguing mathematical properties that resemble behaviors observed in chaotic systems. These properties highlight the complexity and unpredictability of the sequence, even though the process is deterministic.

1. **Sensitive Dependence on Initial Conditions**

   The Collatz Conjecture demonstrates a clear sensitivity to initial conditions, a key feature of chaotic systems [22]. Even a small change in the starting number can lead to significantly different trajectories. For example:

   - Starting with $n = 6$, the sequence converges to 1 in 8 steps:

     $$6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

   - Starting with $n = 7$, the sequence is longer and more complex, taking 16 steps to reach 1:

     $$7 \to 22 \to 11 \to 34 \to 17 \to 52 \to 26 \to 13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$$

   This sensitivity illustrates how minor changes in the initial value can lead to dramatically different outcomes, making it difficult to predict the behavior of the sequence solely based on the starting point.

2. **Unpredictability and Complexity**

Despite being deterministic, the Collatz sequence often exhibits unpredictable patterns, similar to chaotic systems. The rules of the conjecture (divide by 2 for even numbers, multiply by 3 and add 1 for odd numbers) create sequences whose behavior appears random [22]. Key aspects of this unpredictability include:

- **Parity Fluctuations:** The alternation between even and odd numbers in the sequence often appears random, despite being governed by fixed rules.
- **Stopping Time Variability:** The number of steps required to reach 1 can vary drastically. For example, the number 27 takes 111 steps to reach 1, while 28 takes only 18 steps. This variance reflects the unpredictability of the sequence.

3. **Nonlinearity and Iterative Process**

   The Collatz process involves both linear (division by 2) and nonlinear (multiplication by 3 and addition of 1) operations, which adds to its complexity[22, 29]. The combination of these operations results in sequences that do not follow a simple, linear path and require multiple iterations to reach 1. This iterative nature is a characteristic shared with chaotic systems, where the outcome at each step depends on the previous one, often resulting in complex long-term behavior.

The unpredictable and sensitive nature of the Collatz sequences also has notable parallels in fields like neural networks and reinforcement learning, where small changes can lead to vastly different results: Training neural networks often involves optimizing highly sensitive weight configurations[30]. Small changes in the initial weights can result in significantly different learning outcomes, much like how small differences in the starting number of the Collatz sequence lead to different trajectories. This is especially important in deep learning, where weight initialization plays a crucial role in the convergence of training.

2.3 Deep Learning

Deep learning is a specialized branch of machine learning that focuses on training artificial neural networks with multiple layers to extract and learn hierarchical representations from data[1, 31]. Unlike traditional machine learning models that rely on handcrafted features, deep learning methods automatically discover complex patterns by processing raw input through successive layers of transformation [31]. Inspired by the human brain, these models leverage nonlinear activation functions, weight optimization techniques [18], and large-scale datasets to achieve state-of-the-art performance in various tasks, including image recognition

[32], natural language processing [3], and medical diagnostics [5]. The success of deep learning architectures, such as Convolutional Neural Networks (CNNs) for classification [31] and Multilayer Perceptrons (MLPs) for regression [11], largely depends on factors like weight initialization, optimization algorithms, and efficient training methodologies [19].

2.3.1 Multi- layer Perceptron(MLP)

MLPs are the most fundamental form of deep neural networks. They consist of fully connected layers where each neuron in one layer is connected to every neuron in the next layer[33]. The Multilayer Perceptron (MLP) represents an advancement over the original perceptron in terms of its capabilities and learning abilities [19]. While the original perceptron was a single-layer neural network, the MLP introduces multiple layers of interconnected neurons, offering enhanced computational power and the ability to learn complex patterns and relationships [19, 33]. The perceptron was only capable of handling linearly separable classification tasks since it employed a simple step function activation [34]. In contrast, MLPs overcome this limitation by incorporating nonlinear activation functions, such as the sigmoid [35], hyperbolic tangent (tanh) [31], or ReLU [36], in their hidden layers. These nonlinear activation functions introduce nonlinearity into the model, enabling it to learn complex, non-linear decision boundaries [19].

MLPs employ a layered architecture where each layer is composed of multiple neurons. The neurons in each layer are interconnected with weights, allowing information to flow through the network in a feed-forward manner. This layered structure enables the MLP to approximate any continuous function, making it a universal function approximator [19]. Mathematically, an MLP transforms an input vector x through a series of layers:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{2.5}$$

where $W^{(l)}$ represents the weight matrix, $a^{(l-1)}$ is the activation vector from the previous layer, and $b^{(l)}$ is the bias term. The activation function σ introduces non-linearity:

$$a^{(l)} = \sigma(z^{(l)}) \tag{2.6}$$

For regression tasks, the output layer typically employs a linear activation function:

$$\hat{y} = W^{(L)} a^{(L-1)} + b^{(L)} \tag{2.7}$$

The MLP utilizes the backpropagation algorithm [**?**] to optimize its parameters. This

is achieved by computing the Mean Squared Error (MSE) loss function:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.8}$$

where y_i is the ground truth and $\hat{y}_i$ is the predicted output. The network then updates its weights using gradient-based optimization methods such as Stochastic Gradient Descent (SGD):

$$W^{(l)} \leftarrow W^{(l)} - \eta\frac{\partial \mathcal{L}}{\partial W^{(l)}} \tag{2.9}$$

where η is the learning rate.

2.3.2 Convolutional Neural Networks (CNNs) for Classification

Although MLPs are effective at learning complex functions, they face challenges in processing structured data due to their design limitations [31]. For instance, in image classification tasks, MLPs encounter two key issues. First, MLPs treat each input feature independently, ignoring the spatial relationships within images[31] . This restriction prevents them from leveraging the locality and hierarchical structure inherent in visual data, hindering their ability to capture translational invariance—recognizing an object regardless of its position in the image [31]. Second, MLPs experience a significant increase in the number of parameters when handling high-dimensional inputs like images [31]. As each neuron in a fully connected layer is linked to all neurons in the previous layer, this results in greater susceptibility to overfitting and higher computational costs as input dimensionality increases [31].

CNNs are specifically designed to overcome the limitations of MLPs by incorporating specialized architectures that efficiently learn and exploit spatial hierarchies. CNNs consist of three key components: convolutional layers, pooling layers, and fully connected layers [31].

Convolutional Layers: These layers apply a set of learnable filters (kernels) that slide over the input, performing element-wise multiplications and aggregating the results. The 2D convolution operation is mathematically defined as:

$$Y[i,j] = \sum_m \sum_n X[i+m, j+n] \cdot K[m,n] \tag{2.10}$$

where X represents the input image, K is the convolutional kernel, and $Y[i,j]$ is the output feature map.

Pooling Layers: Following the convolutional layers, pooling layers reduce the spatial dimensions of feature maps while retaining essential information. Max pooling is the most common form:

$$Y[i,j] = \max_{m,n} X[i+m, j+n] \tag{2.11}$$

Fully Connected Layers: The final stage of a CNN consists of fully connected layers which are responsible for high-level feature learning and making predictions based on the extracted features. These layers condense spatially encoded information into a compact representation, which can then be applied to tasks such as classification, regression, or retrieval [32].:

$$P(y_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.12}$$

The network is trained using the cross-entropy loss function:

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log \hat{y}_i \tag{2.13}$$

2.4 Weight Initialization in Neural Network

2.4.1 Introduction

Weight initialization plays a fundamental role in the training process of deep neural networks. It directly impacts how information flows through the network and how efficiently the model learns[14, 15, 16]. Poor weight initialization can lead to serious issues such as *vanishing gradients*, *exploding gradients*, and *slow convergence*, making training inefficient or completely ineffective[17].

At the core of neural networks, each neuron takes an input, applies a weighted sum, adds a bias, and passes the result through an activation function. Mathematically, this is expressed as:

$$z = Wx + b \tag{2.14}$$

where:

- W represents the weight matrix,

- x is the input vector,
- b is the bias term.

The activation function $\phi(z)$ then transforms the result to introduce non-linearity:

$$a = \phi(z) \tag{2.15}$$

However, the way weights are initialized before training determines how well the gradients propagate when backpropagation is applied. The update of the weights depends on the gradient of the loss function L with respect to W:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial W} \tag{2.16}$$

For deep networks, this chain of derivatives can become unstable if weights are not carefully initialized.

2.4.2 Key Challenges in Weight Initialization

2.4.2.1 Vanishing Gradient Problem

When gradients become too small as they are backpropagated, weight updates shrink, preventing earlier layers from learning effectively[17]. This often occurs with activation functions like *sigmoid* and *tanh*, whose derivatives are always less than 1:

$$\phi'(z) = \sigma(z)(1 - \sigma(z)) \quad \text{or} \quad \phi'(z) = 1 - \tanh^2(z) \tag{2.17}$$

If a network has multiple layers, the gradient at layer l is computed as:

$$\frac{\partial L}{\partial W^{(l)}} = \prod_{j=l}^{L} \phi'(z^{(j)}) W^{(j)} \tag{2.18}$$

Since each derivative is less than 1, their product tends to zero as depth increases, effectively "freezing" the learning process in early layers.

2.4.2.2 Exploding Gradient Problem

If weight magnitudes are too large, the gradient values grow exponentially, leading to unstable updates[17]. This happens when the weight matrix norm $||W||$ is greater than 1, resulting in:

$$||\frac{\partial L}{\partial W}|| \text{ increasing rapidly} \tag{2.19}$$

This can cause large oscillations in parameter updates, making training unpredictable.

2.4.3 Related Terminologies

- **Fan-in and Fan-out:**
  - **Fan-in** (n_{in}): The number of input connections to a neuron.
  - **Fan-out** (n_{out}): The number of output connections from a neuron.
- **Activation Function Behavior:** The activation function influences how signals propagate through the network:
  - **Sigmoid and Tanh:** Can cause vanishing gradients due to their limited derivative range.
  - **ReLU (Rectified Linear Unit):** Avoids vanishing gradients but can suffer from *dying ReLU*, where neurons become inactive.
  - **Leaky ReLU and ELU:** Address some of ReLU's weaknesses by allowing small negative values to pass through.
- **Variance Preservation and Signal Propagation:** If weights are initialized with variance σ^2, the variance of activations at layer l follows:

$$\text{Var}[z^{(l)}] = n_{\text{in}} \cdot \sigma^2 \tag{2.20}$$

  If variance grows too large, activations explode; if too small, activations vanish. Initialization techniques aim to maintain balance.
- **Breaking Symmetry:** If all neurons in a layer start with identical weights, they will compute the same output and receive identical updates, reducing the network's capacity to learn diverse features. Proper initialization ensures neurons evolve differently.
- **Gradient Flow and Hessian Conditioning:** The Hessian matrix of the loss function determines optimization stability. Poor initialization can result in an ill-conditioned Hessian, making training inefficient.

2.4.4 Role of Fan-in and Fan-out in Weight Initialization

To control weight magnitudes, initialization techniques often consider the number of input and output connections per neuron, defined as:

- **Fan-in** (n_{in}) – the number of input connections to a neuron.
- **Fan-out** (n_{out}) – the number of output connections from a neuron.

A well-balanced initialization ensures that activations maintain a stable variance across layers. If the initial weights are drawn from a normal distribution:

$$W \sim N(0, \sigma^2) \tag{2.21}$$

Then the variance of activations at layer l is given by:

$$\text{Var}[z^{(l)}] = n_{\text{in}} \cdot \sigma^2 \tag{2.22}$$

To prevent variance from increasing or decreasing across layers, initialization schemes adjust σ^2 based on n_{in} and n_{out}, leading to techniques such as Xavier (Glorot) Initialization and He Initialization.

2.4.5 Importance of Proper Weight Initialization

A well-designed weight initialization method should:

- **Maintain variance** across layers to prevent activations from vanishing or exploding.
- **Break symmetry** to allow neurons to learn distinct features.
- **Ensure efficient gradient propagation**, avoiding unstable updates.

By addressing these challenges, various weight initialization techniques have been developed to improve neural network training stability and performance. we will discuss in the next section.

2.5 Weight Initialization Techniques for Neural Network

Weight initialization techniques can be classified in various ways. One method categorizes them based on whether they are solely based on mathematical heuristics or utilize

information from the training data. Another classification considers the assumptions they make regarding data distribution or the input-output behavior of layers[37]. Additionally, techniques can be grouped according to their computational complexity and effect on convergence speed[37]. However, our proposed taxonomy divides weight initialization methods into two categories: **Non-Data-Driven (Heuristic) and Data-Driven(Data Dependent)**

In the next sections, we'll dive deeper into these techniques and explore how they impact neural network performance.

2.5.1 Non-Data-Driven (Heuristic) Initialization

Non-data-driven, or heuristic, initialization methods do not depend on the specific characteristics of the dataset. Instead, they are derived from mathematical principles that ensure stable signal propagation across layers[**?** **?**]. These methods aim to maintain a balanced variance of activations and gradients during training, reducing the risk of instability.

2.5.1.1 Zero Initialization

Zero initialization refers to the practice of setting all weights in a neural network to zero before training[31]. While this approach is straightforward, it has significant limitations, primarily due to the issue of symmetry breaking. In neural networks, especially those with multiple layers, neurons must learn distinct features to effectively model complex data patterns[31]. If all weights are initialized to zero, each neuron in a layer receives the same input and produces the same output during forward propagation. Consequently, during back propagation, these neurons receive identical gradients, leading them to evolve in the same manner. This phenomenon prevents the network from learning diverse features, severely limiting its capacity to model complex data [31].

Mathematically, zero initialization can be represented as follows: Let W be the weight matrix and b be the bias vector in a neural network. In zero initialization, both are set to zero:

$$W = 0, \quad b = 0 \tag{2.23}$$

For a single-layer perceptron, the output of the activation function f is:

$$a = f(Wx + b) \tag{2.24}$$

Since $W = 0$ and $b = 0$, this simplifies to:

$$a = f(0) = \text{constant} \tag{2.25}$$

This means that all neurons in a layer produce the same output, leading to the symmetry problem. During backpropagation, the gradients for all neurons in the same layer will be identical:

$$\frac{\partial L}{\partial W} = 0 \tag{2.26}$$

where L is the loss function. This prevents neurons from learning distinct features and causes the network to fail in training.

Despite its traditional drawbacks, recent studies have revisited zero initialization, proposing modifications to mitigate its inherent issues. Zhao, Schäfer, and Anandkumar [38] introduced the ZerO initialization method, which replaces random weight initialization with a deterministic scheme using only zeros and ones, based on identity and Hadamard transforms. Their theoretical and empirical analyses demonstrated that ZerO could train networks without compromising expressivity[38]. Notably, applying Zero to ResNet architectures achieved better performance on various datasets, including ImageNet, suggesting that random weights may not be necessary for effective network initialization.

While traditional zero initialization poses challenges due to the symmetry problem, recent research indicates that with appropriate modifications, it can be a viable strategy[31]. Techniques like ZerO initialization and combining zero-initialized weights with randomly initialized biases have shown promise in training deep neural networks effectively. These advancements suggest that deterministic initialization schemes, even as simple as zeros and ones, can match[31, 38].

2.5.1.2 Random Initialization

Random initialization is a fundamental weight initialization strategy in neural networks that assigns weights randomly before training. Unlike zero initialization, which leads to the symmetry problem, random initialization ensures that neurons receive different gradients, allowing them to learn distinct features [31]. However, improper random initialization can lead to unstable training due to vanishing or exploding gradients, particularly in deep networks [18].

Mathematically, random initialization follows either a **uniform distribution** or a **nor-**

mal distribution:

- **Uniform distribution:**

$$W \sim U(-a, a) \tag{2.27}$$

  where $U(-a, a)$ represents a uniform distribution with values sampled between $-a$ and a.

- **Normal distribution:**

$$W \sim N(0, \sigma^2) \tag{2.28}$$

  where $N(0, \sigma^2)$ represents a normal (Gaussian) distribution with a mean of 0 and variance σ^2.

While random initialization helps break symmetry, it can introduce the **vanishing gradient** or **exploding gradient** problem, particularly in deep networks. If the weights are too small, activations shrink, leading to near-zero gradients. Conversely, if weights are too large, activations grow exponentially, causing gradient explosion and unstable training [39].

2.5.1.3 Xavier (Glorot) Initialization

Xavier initialization, also known as Glorot initialization, is a weight initialization method designed to maintain stable variance in activations and gradients as they propagate through a neural network. Introduced by Glorot and Bengio (2010)[18], this method addresses the vanishing and exploding gradient problems that occur when weights are initialized too small or too large in deep networks. By ensuring that the signal variance remains consistent across layers, Xavier initialization facilitates better convergence and improves training stability, particularly for activation functions like sigmoid and tanh, which are highly sensitive to weight scaling[18].

Mathematically, Xavier initialization sets the weights using either a uniform or normal distribution. In the uniform distribution case, weights are sampled as:

$$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right) \tag{2.29}$$

Alternatively, in the normal distribution case, weights follow:

$$W \sim N\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right) \tag{2.30}$$

where n_{in} and n_{out} represent the number of input and output neurons, respectively. These formulas ensure that the variance of activations remains stable throughout forward and backward propagation, preventing gradients from diminishing or exploding during training [18].

While Xavier initialization is effective for networks using sigmoid or tanh activations, it is not optimal for ReLU and its variants. Since ReLU activation functions produce asymmetric outputs (zero for negative inputs and identity for positive inputs), Xavier's weight scaling may still lead to vanishing activations for certain neurons. To address this, He et al. (2015)[19] proposed He initialization, which modifies Xavier's formula by increasing weight variance, ensuring ReLU-based networks receive sufficient signal propagation [19].

Further studies have refined initialization techniques to improve deep network training. Mishkin and Matas (2015)[27] introduced layer-sequential unit-variance (LSUV) initialization, which builds upon Xavier initialization by iteratively adjusting variance layer by layer in deep architectures [24, 27]. These advancements highlight the importance of initialization in deep learning, as improper weight scaling can significantly hinder model performance.

2.5.1.4 He (Kaiming) Initialization

He (Kaiming) initialization is designed to improve training stability and convergence in deep neural networks, particularly those utilizing the Rectified Linear Unit (ReLU) activation function. This method was introduced by He et al. (2015)[19] to address the limitations of [18], which assumes symmetric activations like sigmoid or tanh. Since ReLU activation zeroes out negative inputs, it reduces the effective number of active neurons, requiring a higher variance in weight initialization to maintain proper signal propagation. Without an appropriate initialization strategy, deep networks suffer from vanishing gradients, making training inefficient or unstable[19].

Mathematically, He initialization modifies the variance of the weight distribution based on the number of input neurons. The weights are initialized from a normal distribution:

$$W \sim N(0, \frac{2}{n_{\text{in}}}) \tag{2.31}$$

or from a uniform distribution:

$$W \sim U\left(-\sqrt{\frac{6}{n_{\text{in}}}}, \sqrt{\frac{6}{n_{\text{in}}}}\right) \tag{2.32}$$

where n_{in} is the number of input neurons in the layer. The increased variance ($\frac{2}{n_{\text{in}}}$) compensates for the asymmetric nature of ReLU, preventing neuron outputs from collapsing to zero—a problem known as the "dying ReLU" issue. By ensuring that activations maintain their magnitude across layers, He initialization facilitates stable gradient propagation and accelerates training[19].

Empirical studies have demonstrated the effectiveness of He initialization in deep ReLU networks. [19]showed that networks initialized with this method achieved faster convergence and lower training loss compared to Xavier initialization when using ReLU activations. Further refinements, such as Leaky He initialization, have been introduced to optimize training for activation functions that retain negative values, such as Leaky ReLU and Swish [27]. These enhancements ensure that gradient flow remains stable in deeper architectures.

Despite its advantages, He initialization is not universally optimal for all neural networks. In cases where batch normalization is used, the effect of initialization becomes less critical, as normalization layers adjust the scale of activations dynamically. Additionally, emerging research in data-dependent and meta-learning-based initialization methods has explored alternative approaches that adapt to training data properties, potentially outperforming fixed heuristic-based initialization strategies.

2.5.1.5 LeCun Initialization

LeCun initialization is specifically designed for neural networks using activation functions like sigmoid and tanh. Proposed by LeCun et al.[40], this method ensures that the variance of activations remains stable across layers, preventing the common issues of vanishing and exploding gradients. Unlike Xavier initialization, which balances variance between input and output neurons, LeCun initialization considers only the number of input neurons, making it particularly effective for shallow networks and those employing batch normalization[40].

The mathematical formulation of LeCun initialization is given as follows. The weights are sampled from a normal distribution:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{in}}}\right) \tag{2.33}$$

or from a uniform distribution:

$$W \sim U\left(-\sqrt{\frac{3}{n_{\text{in}}}}, \sqrt{\frac{3}{n_{\text{in}}}}\right) \tag{2.34}$$

where n_{in} represents the number of input neurons in a layer. This initialization ensures

that activations maintain a consistent scale, preventing signal degradation as they propagate through the network.

Empirical research has demonstrated the effectiveness of LeCun initialization in improving training stability. [40] showed that networks using this method achieved better convergence rates compared to standard random initialization when trained with sigmoid and tanh activations. Later studies, such as Klambauer et al. (2017)[41], highlighted its compatibility with batch normalization, where proper weight scaling is crucial for maintaining gradient flow[41].

Despite its advantages, LeCun initialization is not the preferred choice for modern deep learning architectures using ReLU-based activations. He (Kaiming) initialization is typically more effective in such cases due to the asymmetry of ReLU. However, in scenarios where sigmoidal activations are still relevant—such as certain recurrent neural networks (RNNs) and fully connected layers—LeCun initialization remains a valuable approach for stabilizing training[42].

2.5.1.6 Orthogonal Initialization

Orthogonal weight initialization is a technique used in deep learning to initialize the weight matrices of neural networks while maintaining orthogonality[20]. The primary motivation for this approach is to address issues related to vanishing and exploding gradients, particularly in deep networks[17, 20]. By ensuring that weight matrices remain orthogonal, this method helps preserve variance across layers and improves training stability. Unlike conventional initialization methods, orthogonal initialization maintains dynamical isometry, allowing signals to propagate efficiently through the network and preventing the loss of information during forward and backward passes[20].

Given a weight matrix W of shape $(n_{\text{out}}, n_{\text{in}})$, orthogonal initialization constructs W using a random matrix A of the same shape, where elements are sampled from a normal distribution:

$$A_{ij} \sim \mathcal{N}(0, 1) \tag{2.35}$$

Next, a **QR decomposition** is performed on A, yielding an orthogonal matrix Q, such that:

$$A = QR \tag{2.36}$$

The weight matrix W is then initialized as:

$$W = Q \tag{2.37}$$

If $n_{\text{out}} > n_{\text{in}}$, only the first n_{in} columns of Q are used. This ensures that W satisfies $W^T W = I$, maintaining signal propagation through the layers without amplification or attenuation.

Orthogonal initialization offers several benefits. First, it helps preserve signal flow by ensuring that input variance remains stable across network layers, reducing the risk of exploding or vanishing gradients. Second, it improves training stability, making it particularly useful for deep architectures where improper initialization can lead to slow or unstable convergence. Finally, it has been shown to be effective for recurrent neural networks (RNNs)[43], preventing the long-term dependencies from vanishing, which is a common issue in standard RNN training[25, 44].

Tian et al. [45] demonstrated how orthogonal initialization enhances stability in deep reinforcement learning models for motion control tasks, showing improved convergence properties in real-world applications [45]. Similarly, Pabari and Zhou [46] developed a mean-field theory for deep neural network training, confirming that orthogonal initialization enhances dynamical isometry and prevents gradient-related issues during optimization [46]. Miranskyy et al.[47]further explored the role of quasirandom sequences in weight initialization, showing that orthogonal initializers outperform conventional methods in stabilizing training dynamics, particularly in deep networks [47].

unlike other methods, orthogonal initialization uniquely ensures that activations remain evenly distributed, preserving information flow even in very deep networks.

2.5.1.7 Sparse Initialization

Sparse weight initialization is a technique in deep learning where only a subset of the neural network's weights are initialized as nonzero while the remaining weights are set to zero[48]. This method is designed to improve computational efficiency, reduce memory usage, and enhance generalization by enforcing sparsity in the weight matrices. Sparse initialization is particularly useful for deep networks, where dense weight matrices can lead to high redundancy and excessive resource consumption[48, 49]. By starting with a sparse set of weights, the network focuses on learning only the most relevant connections, reducing unnecessary complexity and improving training efficiency[48, 49].

Mathematically, sparse initialization is performed by defining a weight matrix W of shape $(n_{\text{out}}, n_{\text{in}})$ and specifying a sparsity level s, which determines the proportion of weights that remain zero. Nonzero weights are initialized using a standard distribution, typically drawn from a normal distribution:

$$W_{ij} \sim \mathcal{N}(0, \sigma^2) \tag{2.38}$$

A binary mask M is then applied to enforce the sparsity constraint, where:

$$M_{ij} \sim \text{Bernoulli}(1 - s) \tag{2.39}$$

The final sparse weight matrix is computed as:

$$W = M \odot W \tag{2.40}$$

where $\odot$ represents the element-wise product. This process ensures that only a small fraction of the weights are active while the majority remain zero, allowing the network to focus on the most informative connections.

it offers several benefits[48, 49]. First, it improves computational efficiency by reducing the number of active parameters, leading to faster training and inference. Second, it acts as an implicit regularization technique, promoting better generalization and reducing overfitting by preventing the network from memorizing noise in the data. Third, it significantly reduces memory usage, making it an ideal approach for deploying models on edge devices or resource-constrained environments. Lastly, it improves gradient propagation by limiting excessive co-adaptation between neurons, thereby enhancing the stability of deep learning models[48, 49].

Recent research has demonstrated the effectiveness of sparse weight initialization in various applications. Kolb et al. (2025) introduced a sparse weight factorization approach that improves learning efficiency by eliminating redundant connections in neural networks, enhancing both interpretability and robustness [**?**]. Wang et al.[50] proposed a coarse-to-fine lightweight sparse embedding method for graph neural networks, showing that sparse initialization improves learning efficiency in recommendation systems [50]. Zhao et al. (2024) developed a pruning-at-initialization method (RePaIR) that optimizes sparse weight initialization to maintain performance even with highly reduced weight counts, demonstrating resilience in deep models [**?**].

Table 2.1: Comparison of Non-Data-Driven Weight Initialization Techniques in Neural Networks

Technique	**Mathematical Formula**	**Best For**	**Advantages**	**Limitations**
Zero Initialization	$W = 0$	None (not recommended)	Simple implementation	Causes symmetry; neurons learn identical features
Random Uniform	$W \sim U(-a, a)$	Any activation function	Quick and simple	Can lead to exploding/vanishing gradients
Random Normal	$W \sim \mathcal{N}(0, \sigma^2)$	Any activation function	More controlled weight spread	May still cause unstable gradients
Xavier (Glorot)	$W \sim U\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}}+n_{\text{out}}}}\right)$	Sigmoid, Tanh	Maintains variance across layers	Not optimal for ReLU
He (Kaiming)	$W \sim \mathcal{N}(0, \frac{2}{n_{\text{in}}})$	ReLU, Leaky ReLU	Prevents dying ReLU problem	Not ideal for sigmoid/tanh
LeCun	$W \sim \mathcal{N}(0, \frac{1}{n_{\text{in}}})$	Small networks, Tanh	Stabilizes variance in small networks	Limited to small architectures
Orthogonal	$W = Q$ (QR decomposition)	Deep networks, RNNs	Preserves signal flow, avoids vanishing gradients	Computationally expensive
Sparse	$W = M \odot W$, $M_{ij} \sim$ Bernoulli$(1 - s)$	Large-scale models, Transformers	Reduces computation and memory overhead	May lose some learning capacity

2.5.2 Data-Driven (Data-Dependent) Initialization

Unlike heuristic methods, data-driven initialization techniques incorporate statistical properties of the dataset or leverage pre-trained knowledge to enhance weight initialization[51]. These methods are particularly beneficial for deep learning architectures, where well-informed initialization can significantly speed up convergence and improve model generalization[52].

2.5.2.1 Statistical-Based Initialization

Statistical-based initialization methods aim to align weight distributions with the statistical properties of the input data, ensuring stable training dynamics. These techniques mitigate issues such as vanishing and exploding gradients by appropriately scaling the initial weights based on input variance. Two widely used approaches in this category are **Mean-Variance Matching (BatchNorm-Based Initialization)**[53] and **Data-Scaled Initialization**[54].

Mean-variance matching initialization is primarily used in architectures that incorporate Batch Normalization (BatchNorm)[53] . BatchNorm standardizes activations by normalizing them to a zero mean and unit variance, improving gradient flow and stability. To complement this, weight initialization methods in such networks are designed to match the variance of the input distribution. Mathematically, if σ_x^2 represents the variance of input features, then the weight initialization follows:

$$W \sim \mathcal{N}\left(0, \frac{1}{\sigma_x^2}\right) \tag{2.41}$$

This ensures that activations remain properly scaled throughout forward and backward propagation. Research by Ioffe and Szegedy [53] , who introduced BatchNorm, demonstrated that matching initialization to input statistics significantly accelerates training and improves convergence, particularly in deep neural networks[24, 53] .

Data-scaled initialization, another approach under this category, adjusts the weight distribution based on the number of input features[54]. This method ensures that the variance of activations remains controlled as data propagates through layers. The weight initialization is defined as:

$$W \sim \mathcal{N}\left(0, \frac{1}{n_{\text{features}}}\right) \tag{2.42}$$

where n_{features} represents the number of input features. This approach has been widely used in large-scale neural networks to maintain stable training dynamics. Studies such as

those by Sutskever et al.[24] and Mishkin & Matas [27]found that incorporating data-scaled initialization leads to faster convergence and improved network performance.

While these techniques provide a statistically grounded method for initializing weights, their effectiveness is often architecture-dependent. Networks with BatchNorm benefit significantly from mean-variance matching, whereas data-scaled initialization is preferred when feature variance is a critical factor. Recent advancements continue to explore adaptive initialization techniques that dynamically adjust weight distributions based on real-time data statistics, further optimizing deep learning performance[24, 53, 54].

2.5.2.2 Unsupervised Feature-Based Initialization

Unsupervised feature-based initialization methods leverage unsupervised learning techniques to determine optimal weight initializations, ensuring that the initial weights capture meaningful data representations before supervised training begins[55, 56]. These methods are particularly effective in deep networks where random initialization may not provide an optimal starting point for learning. Some of the most common approaches include **Principal Component Analysis (PCA)-Based Initialization, K-Means-Based Initialization, and Linear Discriminant Analysis (LDA)-Based Initialization**.

PCA-based initialization utilizes the principal components of the input data to initialize weights[57]. PCA identifies the directions of maximum variance in the dataset, capturing the most significant features. By aligning initial weights with these principal components, networks can start with a more structured representation of the input data[57]. Mathematically, given an input dataset X, PCA decomposes it into eigenvectors and eigenvalues:

$$X = U\Sigma V^T \tag{2.43}$$

where U contains the principal components. The weight matrix W is then initialized using these principal components:

$$W = U \tag{2.44}$$

This method ensures that initial weights align with the dominant structures in the data, leading to faster convergence. Research by Saxe et al. [25]demonstrated that PCA-based initialization significantly stabilizes deep network training by reducing gradient instability.

K-Means-based initialization involves clustering input data into k centroids and using these centroids to initialize network weights[58, 59]. The idea is that clustering identifies

representative data points that best describe the input space[58, 59]. Given a set of input samples $\{x_1, x_2, ..., x_n\}$, K-Means partitions them into k clusters with centroids μ_j:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \tag{2.45}$$

where C_j represents the set of points in cluster j. The initialized weights are then set as:

$$W_j = \mu_j \tag{2.46}$$

Studies, such as those by Coates and Ng [60], show that using K-Means to pre-train weights improves feature extraction in convolutional and deep autoencoder architectures[60].

LDA-based initialization optimizes weight initialization by maximizing class separability in the feature space[61]. Unlike PCA, which captures variance in an unsupervised manner, LDA considers class labels to determine optimal projections. Given data from different classes, LDA finds a transformation matrix W that maximizes the ratio of between-class variance (S_B) to within-class variance (S_W):

$$W = \arg\max \frac{|S_B|}{|S_W|} \tag{2.47}$$

where:

$$S_B = \sum_c N_c(\mu_c - \mu)(\mu_c - \mu)^T, \quad S_W = \sum_c \sum_{x \in c} (x - \mu_c)(x - \mu_c)^T \tag{2.48}$$

Here, μ_c is the mean of class c, and μ is the overall mean[61]. LDA-based initialization has been found to be particularly effective in classification tasks, as demonstrated in works by Guo et al. [29], where LDA-based initialization improved convergence rates and final model accuracy in deep learning applications.

These unsupervised feature-based initialization methods offer a more structured approach to weight initialization, leveraging data-driven insights before supervised learning begins. By aligning initial weights with dominant patterns in the data, they provide a more efficient starting point for deep networks, reducing training time and improving performance[55, 56].

2.5.2.3 Pretrained Model-Based Initialization

Pretrained model-based initialization is a powerful technique in deep learning that leverages knowledge acquired from previously trained models to initialize the weights of a new neu-

ral network[26]. Instead of starting with randomly initialized weights, this method utilizes parameters learned from a large dataset, enabling faster convergence and better generalization, especially when the target dataset is small or similar to the source dataset[26]. This approach is widely used in transfer learning, where models trained on large-scale datasets such as ImageNet [62] or text corpora like Wikipedia and BooksCorpus [**?**] are adapted for new tasks with minimal training.

Mathematically, let θ^* denote the optimal weights of a pretrained model on a source task T_s. Instead of initializing new weights θ_{init} randomly, pretrained initialization sets:

$$\theta_{\text{init}} = \theta^* \tag{2.49}$$

where θ_{init} is the starting weight configuration for the new task T_t. If fine-tuning is applied, the parameters are updated as:

$$\theta_t = \theta^* + \Delta\theta \tag{2.50}$$

where $\Delta\theta$ represents the learned updates specific to the target task.

Pretrained initialization can be implemented in two primary ways: *feature extraction* and *fine-tuning*[63, 64]. In feature extraction, the lower layers of a deep network are frozen while only the upper layers are trained on the new dataset[63]. This approach is effective when pretrained features remain relevant to the new task. In fine-tuning, all or most layers are updated to adapt to the new task, allowing the model to learn more task-specific representations but requiring careful tuning to prevent catastrophic forgetting [64].

Despite its advantages, pretrained model-based initialization presents challenges. One key issue is *domain mismatch*, where features learned from the source dataset may not fully transfer to the target dataset, reducing initialization effectiveness[65]. Another concern is *catastrophic forgetting*, where fine-tuning a model on a new task can overwrite previously learned knowledge[64]. Additionally, large pretrained models require significant storage and computational resources, posing limitations for deployment in resource-constrained environments [62, 65].

2.5.2.4 Meta-Learning-Based Initialization

Meta-learning-based initialization is an advanced technique in deep learning that enables a model to learn an effective initialization strategy from multiple related tasks, allowing it to quickly adapt to new tasks with minimal training[66]. Unlike traditional weight initialization

methods, which rely on mathematical heuristics or pretrained models, meta-learning optimizes the initialization parameters in a way that facilitates fast adaptation to novel tasks. This approach is particularly beneficial in few-shot learning scenarios, where only a limited amount of labeled data is available for training [66, 67].

Mathematically, let θ represent the model parameters. In meta-learning-based initialization, the goal is to find an optimal initialization θ^* that enables efficient fine-tuning for a new task. The learning process is typically framed as a bi-level optimization problem. Given a set of training tasks $\mathcal{T}$, the initialization θ is optimized such that for each task T_i, after performing a few gradient updates, the model achieves high performance:

$$\theta^* = \arg\min_{\theta} \sum_{T_i \sim \mathcal{T}} \mathcal{L}_{T_i} \Big(\theta - \alpha \nabla_{\theta} \mathcal{L}_{T_i}(\theta) \Big) \tag{2.51}$$

where $\mathcal{L}_{T_i}$ is the loss function for task T_i, and α is the learning rate. The model learns an initialization θ^* such that a small number of gradient steps yield good task-specific performance.

One of the most widely used algorithms for meta-learning-based initialization is Model-Agnostic Meta-Learning (MAML) [68]. MAML aims to find a set of model parameters that can generalize across tasks by optimizing initialization weights through episodic training. The method trains the model on multiple tasks in a way that encourages rapid adaptation to new, unseen tasks with just a few gradient steps. This approach has been successfully applied in domains such as few-shot image classification, reinforcement learning, and natural language processing [68, 69].

However, this approach also presents challenges. Training meta-learning models can be computationally expensive, as it requires multiple nested optimization steps. Moreover, the effectiveness of meta-learning heavily depends on task similarity—if the tasks used for training are too different from the target task, the learned initialization may not generalize well. Additionally, meta-learning requires careful tuning of hyperparameters, such as the learning rate and adaptation steps, to ensure stability and efficiency [66, 68, 69].

2.6 Related Work on Weight Initialization

Weight initialization plays a critical role in the training stability[30], convergence speed [14, 15], and generalization capability[16] of neural networks. Poor initialization can lead to vanishing or exploding gradients[17], significantly hindering learning, especially in deep

Table 2.2: Summary of Data-Driven (Data-Dependent) Weight Initialization Techniques

Method	**Formula / Approach**	**Best for**	**Key Benefits**	**Limitations**
Statistical-Based Initialization	$W \sim N(0, \frac{1}{\sigma_x^2})$	Networks with BatchNorm	Matches input variance, stabilizes training	Requires batch statistics, sensitive to distribution shifts
Unsupervised Feature-Based Initialization	Uses PCA, K-Means, or LDA to initialize weights	Clustering tasks, unsupervised learning	Embeds meaningful feature representations from data	Computationally expensive, sensitive to feature selection
Pretrained Model-Based Initialization	$\theta_{\text{init}} = \theta^*$ from a pretrained model	Transfer learning	Faster convergence, better generalization, reduces need for large datasets	Domain mismatch, large storage requirement
Meta-Learning-Based Initialization	θ_{init} learned from multiple tasks	Few-shot learning, adapting to new tasks	Enables rapid adaptation to new tasks with minimal data	Computationally expensive, requires diverse task training

architectures. Conventional initialization methods such as Xavier [18] and He [19] initialization have been widely adopted, yet recent research highlights their limitations in specific architectures and domain-specific applications. This section reviews recent works in weight initialization techniques.

Desai et al. [70] examines the impact of different weight initialization strategies on training efficiency and model performance, using a simple feedforward neural network trained on the MNIST dataset [71]. The study evaluates four initialization methods—Random Normal, Random Uniform, Xavier (Glorot), and He Initialization within a ReLU-activated [11] dense network architecture, consisting of a flatten layer, a dense layer with 128 neurons (ReLU), and a final softmax output layer[70].The study evaluates four initialization methods—Random Normal, Random Uniform, Xavier (Glorot), and He Initialization within a ReLU-activated [11] dense network architecture, consisting of a flatten layer, a dense layer with 128 neurons (ReLU), and a final softmax output layer[70]. The Random Normal and Random Uniform initializations[72] assign weights from a standard Gaussian[73] and uniform distribution, respectively, without considering input-output variance scaling[70]. In contrast, Xavier Initialization scales weight variance based on the number of input and output neurons, ensuring balanced signal propagation, while He Initialization adjusts variance to optimize

weight scaling for ReLU activation[70]. The results show that Xavier Initialization achieves the highest test accuracy (97.78), closely followed by He Initialization, whereas Random Normal and Random Uniform perform suboptimally[70]. Xavier's superior performance is attributed to its ability to preserve gradient variance, leading to more stable weight updates and efficient learning[70]. He Initialization, designed specifically for ReLU networks, also performs well but exhibits slightly higher variance in accuracy[70].

Lee et al. [74] propose a novel weight initialization technique specifically designed for deep and narrow feedforward neural networks (FFNNs) to address gradient instability and neuron inactivity. Unlike traditional variance-based initializations such as Xavier or He, the proposed method leverages structured weight matrices that maintain orthogonality and positive-entry dominance to improve gradient flow and prevent neuron saturation[74]. The method constructs a weight matrix We with carefully controlled eigenvalues, ensuring that the variance of activations remains stable across layers[74]. By maintaining unit variance while adjusting column sum properties, the initialization scheme prevents excessive gradient shrinkage[75], which is a common issue in very deep and narrow networks. Additionally, the study incorporates positive-entry dominance, which reduces the likelihood of dying ReLUs, ensuring that neurons remain active throughout training[74]. Experimental validation on MNIST, Fashion-MNIST, and structured tabular datasets demonstrates that this initialization method outperforms Xavier and He initialization in both convergence speed and model generalization[74]. The results indicate that the method is particularly beneficial for architectures with a high depth-to-width ratio, where standard initialization strategies often struggle due to gradient collapse[74]. However, the study notes that the method may require fine-tuning for wider architectures and could be further optimized for non-ReLU activation functions[74].

Ghazi et al. [76] explore the role of weight initialization in improving the stability and convergence of Long Short-Term Memory (LSTM) networks. They propose a variance-preserving initialization method designed to maintain consistent variance propagation, preventing gradient explosion or vanishing issues[76]. The approach is evaluated on both standard LSTMs for univariate time series regression and LSTMs adapted for handling missing values in multivariate disease progression modeling[76]. By normalizing weight distributions and ensuring balanced variance between inputs and outputs, the method enhances training stability[76]. Experimental results on benchmark datasets demonstrate that this initialization strategy accelerates convergence and improves generalization performance compared to Xavier and He initialization, particularly in challenging learning scenarios [76].

Skorski et al. [77] propose a Hessian-stabilizing weight initialization technique that improves gradient stability in deep neural networks by incorporating second-order Hessian norm tracking into the initialization process. Traditional variance-based methods, such as Xavier and He initialization, primarily focus on maintaining activation variance across layers but fail to account for higher-order effects, often leading to unstable training dynamics in very deep networks[77] . Skorski et al. address this issue by controlling the Hessian norm during initialization, which helps regulate the curvature of the loss function and prevents gradient explosion or vanishing gradients[77] . Their approach is validated on CNN architectures, including ResNet-18, LeNet-5, and EfficientNet, using the CIFAR-10 dataset, where it demonstrates improved training stability, faster convergence, and a smoother loss landscape compared to standard methods. The study shows that by tracking the Hessian norm during initialization, weight updates remain well-conditioned, reducing the likelihood of sudden gradient spikes or collapses that could hinder optimization[77] . However, a major drawback of this method is its higher computational cost and memory overhead, as Hessian-based calculations require additional matrix operations and storage, making it less practical for extremely large-scale networks. Despite this limitation, the findings suggest that Hessian-aware initialization can significantly enhance training robustness in deep CNNs, particularly in scenarios where standard initialization techniques struggle with convergence instability[77] .

Murru and Rossini [78] propose a Bayesian weight initialization technique that uses a modified Kalman filter to improve weight distribution before training. Unlike standard variance-based methods like Xavier and He initialization, this approach dynamically refines weight estimates by incorporating Bayesian inference, ensuring more stable training[78]. Their method is evaluated on character recognition tasks using MNIST, where it shows improved training stability and better generalization compared to traditional initialization techniques. By incorporating prior knowledge, the method reduces weight divergence and accelerates convergence, making it particularly useful for models where stable training is crucial[78]. However, the approach introduces additional computational overhead due to the iterative weight refinement process, which may limit its efficiency in larger networks. Despite this, the study demonstrates that Bayesian initialization can be beneficial for tasks requiring high generalization and robustness[78].

Koturwar and Merchant et al. [79] extend the idea of data-aware weight initialization by proposing a method that leverages dataset-specific statistical properties to align weight distributions with the underlying data structure. The study introduces two approaches: (1)

PCA-based initialization[80], which estimates principal components to guide weight vectors, and (2) data-statistics-based initialization, which samples weights from a Gaussian distribution derived from dataset statistics and applies a whitening transformation to prevent feature redundancy[80]. Experimental results on CIFAR-10 [71], MNIST, and an unbalanced classification dataset show that this method outperforms He and Xavier initialization in convergence speed and accuracy, particularly for datasets with imbalanced distributions[80].

Wang Liu [81] investigate the role of input weight selection in Extreme Learning Machines (ELM), focusing on methods that preserve sample structure to improve learning efficiency and generalization performance. Traditional ELM approaches rely on Monte Carlo (MC)-based random weight initialization, which provides fast learning speed but fails to maintain sample structure preservation (SSP), leading to suboptimal generalization[81] . To address this, the authors explore Quasi-Monte Carlo (QMC) sampling as an alternative, demonstrating that QMC projections achieve a faster convergence rate than MC for lower-dimensional problems due to reduced distortion error. Furthermore, they propose a random orthogonal (RO) projection method, which optimally transforms input weights to minimize distance loss across all sampl es, thereby enhancing ELM's generalization performance[81] .

Sun et al. [82] propose the Multi-layer Maxout Network (MMN), a hierarchical activation function designed to enhance the non-linearity and feature extraction capabilities of CNNs. The author introduce a specialized initialization method for MMN that preserves variance across layers, mitigating internal covariate shift and reducing vanishing/exploding gradients[17, 82]. Experimental validation on CIFAR-10, CIFAR-100, and ImageNet[62] demonstrates that MMN initialization outperforms standard Xavier initialization, leading to faster convergence and improved test accuracy. Additionally, the study explores a trade-off strategy, showing that selectively applying MMN to lower layers balances computational cost and performance, making it a flexible alternative to standard activations[82].

Dauphin and Schoenholz et al. [83] introduce MetaInit, a meta-learning-based weight initialization strategy designed to minimize second-order gradient instabilities. Unlike conventional approaches that focus on variance preservation, MetaInit optimizes weight norms dynamically using a novel metric called the Gradient Quotient (GQ), which quantifies gradient sensitivity after a single update step[83]. By minimizing GQ, MetaInit automatically adjusts weight magnitudes to ensure smooth optimization dynamics[83]. The method is validated on CIFAR-10 and ImageNet[62], where it enables training deep CNNs and ResNets without requiring normalization layers[83]. The results indicate that MetaInit reduces reliance on Batch Normalization (BN)[84], making it a promising alternative for architectures

where normalization layers introduce overhead[83].

Table 2.3: Summary of Related Work

Ref	Method	Fast Convergence	High Accuracy	Stable Gradient	Activation Retention	Generalization Capability
[18]	Xazier	✓	✓	✓	X	✓
[19]	He	✓	✓	X	○	✓
[82]	MMN	✓	✓	X	X	X
[83]	MetaInit	X	X	✓	✓	X
[74]	Orthogonality	✓	X	✓	✓	✓
[70]	comparative	X	✓	✓	X	✓
[77]	Hessian-stabilizing	✓	X	○	✓	X
[78]	Bayesian	✓	X	✓	X	✓
[76]	Variance-preserving	✓	✓	✓	X	✓
[81]	Monte Carlo	✓	○	X	X	✓
prop.	**CSB**	✓	✓	✓	✓	✓

Legend: ✓= Fully addressed, ○ = Partially addressed, X = Not addressed.

Despite significant advancements in weight initialization strategies for deep neural networks, existing methods often fall short of simultaneously satisfying all the key properties essential for effective training. As shown in Table 2.3, while popular methods like Xavier and He offer reasonable convergence speed and accuracy, they exhibit limitations in gradient stability and activation retention. Other techniques, such as MetaInit and Bayesian initialization, provide improved gradient flow but compromise on convergence speed or generalization capability. Furthermore, several recent methods remain specialized or inconsistent across architectures and datasets, often addressing only a subset of desirable properties. This fragmented performance landscape highlights a clear research gap: the lack of a unified initialization method that concurrently ensures enhanced convergence, high accuracy, stable gradients, retention of activations, and strong generalization. To address this, we propose Collatz Sequence Based Weight Initialization (CSB)—a novel, deterministic approach leveraging the structured numerical properties of the Collatz sequence. CSB is designed to provide a balanced initialization scheme that achieves comprehensive training stability and efficiency across different network architectures, effectively bridging the gap left by existing methods.

Chapter 3

Methodology

3.1 Overview

This section introduces the methodological approach adopted for investigating the impact of Collatz Sequence Based Weight Initialization on the performance of neural networks, specifically focusing on classification and regression tasks using CNN and MLP architectures.

3.2 Research Methodology

To conduct the research, we adopted the Design Science Research Process (DSRP) [[85] framework, which is well-suited for addressing an existing challenge and developing a novel solution. The methodology consists of five key phases: Problem Identification, Objective Formulation, Design & Implementation, Experimentation, and Evaluation, ensuring a structured approach to developing and validating the Collatz Sequence Based Weight Initialization method.

- **Problem Identification & Literature Review:** The study begins with a thorough *problem definition*, highlighting the limitations of existing weight initialization techniques, such as Xavier He, random and orthogonal initializations, in terms of convergence,training efficiency, and stability. A comprehensive literature review is conducted to analyze previous research on weight initialization, chaos theory in neural networks, and optimization technique. This step helps in identifying gaps in existing methods and provides the foundation for designing **CSB**.

- **Objective Formulation:** Based on the identified research gaps, the general and specific objectives of the study are clearly defined. These objectives guide the development and evaluation of the proposed method, ensuring a structured research approach.

- **Design & Implementation:** In this phase, the ***CSB*** *algorithm* is designed and implemented by leveraging the structured randomness of the Collatz sequence. The initialization method is integrated into Convolutional Neural Networks for classification and Multi-Layer Perceptrons for regression. The method's effectiveness is tested under different network architectures and training conditions.

- **Experimentation:** The CSB method is applied to CNN-based classification tasks and MLP-based regression models, where its performance is systematically tested against the seven conventional initialization techniques. Experiments are conducted on CIFAR 10 and CALIFORNIA HOUSING datasets , ensuring a fair and controlled comparison. Metrics such as convergence speed, loss reduction rate, training time, and final model accuracy are recorded to assess the effectiveness of CSB. Ablation studies are also performed to analyze how different configurations of the Collatz sequence impact network initialization.

- **Evaluation & Performance Analysis:** The results from the experiments are analyzed to determine the impact of CSB on training efficiency, convergence behavior, and model generalization. The study also investigates whether CSB mitigates vanishing and exploding gradients more effectively than existing methods. Findings from this evaluation provide empirical evidence for the viability of CSB as an alternative weight initialization strategy for deep learning models.

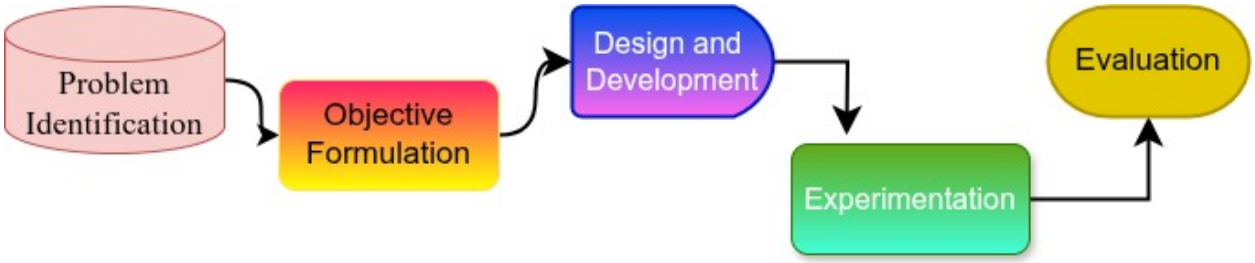

Figure 3.1: Design Science Research Process for this thesis work.

3.3 Proposed Method

Weight initialization plays a pivotal role in the training dynamics of neural networks, influencing convergence speed, training stability, and final model accuracy. While conventional methods such as Xavier, He, and LeCun initialization have demonstrated effectiveness by preserving variance across layers, they often rely on fixed statistical assumptions that may not generalize well across tasks and architectures.

In this research, we propose a novel deterministic weight initialization method Collatz Inspired Weight Initialization motivated by the structured yet chaotic behavior of the Collatz sequence.

The key idea behind CSB is to introduce a controlled form of randomness and diversity into the initial weights through iterative transformations derived from the Collatz conjecture.

This approach aims to improve the exploration of the weight space early in training while preserving global variance control.

Figure 3.2 presents the overall framework used to evaluate the proposed Collatz Sequence Based Weight Initialization method against baseline initializers. It summarizes the entire pipeline from dataset acquisition to evaluation, highlighting the roles of different components used in both CNN and MLP.

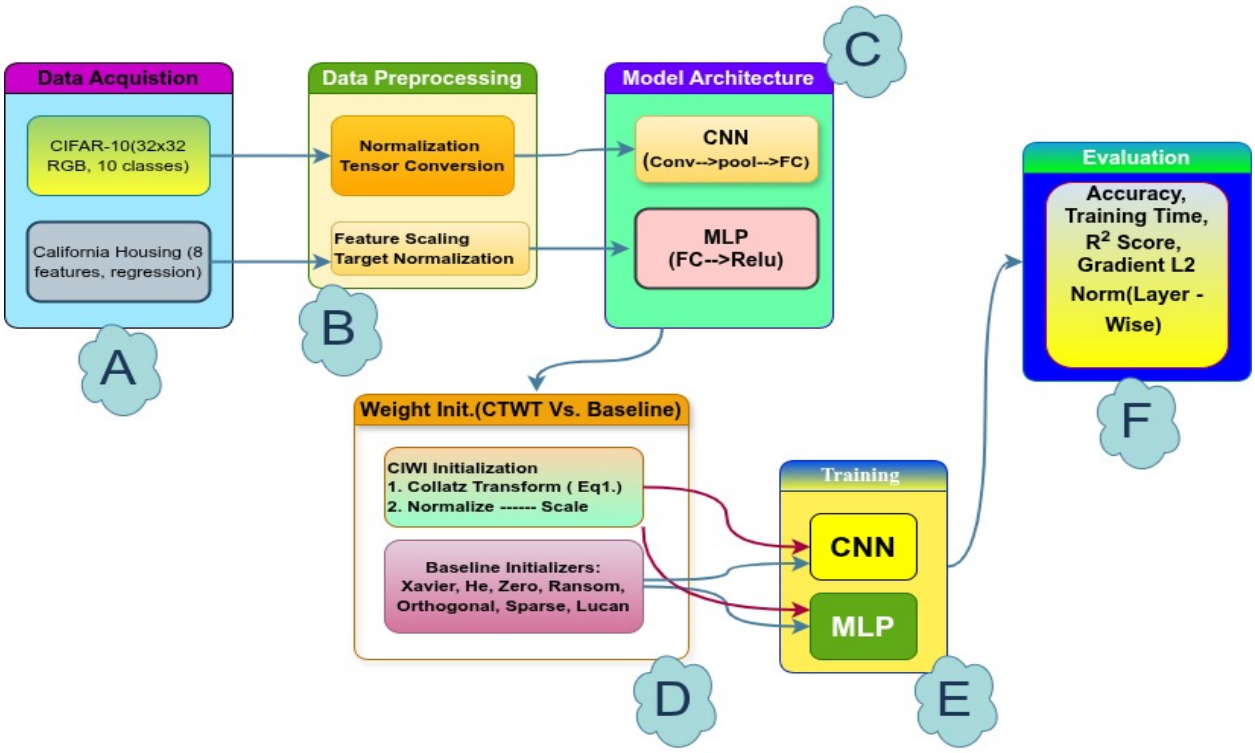

Figure 3.2: Framework for Evaluating Collatz Sequence Based Weight Initialization

Below is a brief explanation of each labeled block in the framework:

- **Block (A)** shows the data acquisition process. It involves two datasets: CIFAR-10 for image classification using CNNs, and California Housing for regression using MLPs
- **Block (B)** shows the preprocessing steps applied to each dataset. For CIFAR-10, normalization and tensor conversion are performed, while California Housing undergoes feature scaling and target normalization. The flow of operations is specific to the nature of each dataset.
- **Block (C)** shows the model architectures used in the experiments. Two architectures are defined: CNN, which consists of convolution, pooling, and fully connected layers; and MLP, which is composed of fully connected layers with ReLU activations. Each model is selected based on dataset type.

- **Block (D)** shows the weight initialization strategies. The proposed CSB method applies a Collatz transformation followed by normalization and scaling. It is compared against seven baseline initializers including Xavier, He, Zero, Random, Orthogonal, Sparse, and LeCun. This block determines how the model weights are initialized before training begins.

- **Block (E)** shows the training process of both CNN and MLP models using the respective weight initializers. This stage evaluates how different initializations affect learning efficiency, convergence behavior, and gradient dynamics.

- **Block (F)** captures the evaluation step. Performance metrics such as accuracy, training time, R^2 score, and gradient stability are used to compare CSB against baseline initializers.

3.3.1 Intuition Behind CSB

The CSB approach is motivated by the inherent balance between order and chaos in the Collatz sequence [86, 87]. In traditional weight initialization, randomness is often injected purely through Gaussian or uniform distributions [18, 19], which may not introduce sufficient diversity or structure into the early weight space. The Collatz transformation despite its simplicity produces sequences that demonstrate highly dynamic, yet bounded behavior [87]. By applying this transformation iteratively to initial weights, CSB introduces non linear, structured perturbations that encourage heterogeneity among neurons. This breaks symmetry more effectively than purely random initializations [32] and stimulates a more expressive feature learning process during early training stages. Moreover, the deterministic nature of the transformation offers reproducibility, while the normalization and scaling steps ensure numerical stability [86]. The goal is not just to randomize, but to seed the model with weights that are both diverse and inherently dynamic, potentially improving gradient flow, convergence speed, and generalization [39, 88, 89].

3.4 Datasets

This study utilizes two benchmark datasets to evaluate the effectiveness of Collatz Inspired Weight Initialization in classification and regression tasks. CIFAR-10[90] is used for classification, while California Housing[91] is used for regression. The details of these datasets are discussed below.

3.4.1 CIFAR-10

CIFAR-10 [90]is a widely used dataset in computer vision and machine learning, originally released by the Canadian Institute for Advanced Research (CIFAR). It consists of 60,000 color images, each of size 32×32 pixels, categorized into 10 distinct classes, with 6,000 images per class. The dataset includes a diverse range of objects, such as **airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks**. Each training batch contains 5,000 images, ensuring a balanced distribution across categories[90].

CIFAR-10 serves as a benchmark dataset for evaluating image classification models, offering a more challenging task than MNIST due to its smaller image size and color variations. This complexity requires models to learn more advanced visual representations. Due to its widespread adoption, CIFAR-10 remains a crucial resource for researchers, educators, and practitioners, helping to develop and compare machine learning techniques for real-world image classification problems.

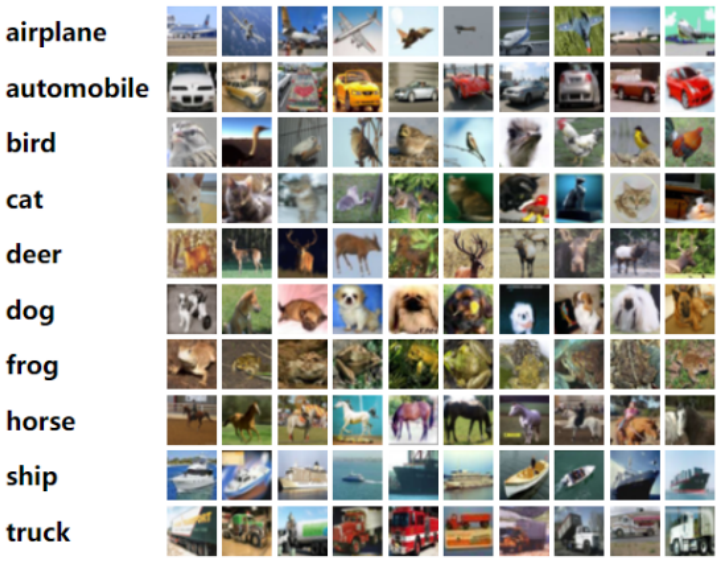

Figure 3.3: CIFAR-10 Dataset Sample

3.4.2 California Housing

The California Housing dataset is a widely used benchmark for regression tasks, consisting of 20,640 instances with 8 numerical features derived from the 1990 U.S. Census data. These features include metrics such as median income, average number of rooms per household, population, and housing age. The target variable is the median house value in a given California district, making it suitable for evaluating the performance of machine learning models in predicting continuous values.it provides a larger and more modern sample, of-

fering greater variability and regional representation. It is particularly useful for assessing regression models under realistic socioeconomic and geographic conditions, and has become a standard choice for evaluating the generalization capability of algorithms in housing price prediction and related applications.

Table 3.1: Summary of Datasets Used

Name	Size (Train/Test)	Type	Classes
CIFAR-10	50,000 / 10,000	Image	10
California Housing	16,512 / 4,128	Tabular Data	–

3.5 Data Preprocessing

Appropriate preprocessing steps are essential to ensure efficient model training and numerical stability[88].

- **CIFAR-10 for CNN**:
  - **Normalization**: All pixel values were scaled from [0, 255] to [0, 1] using torchvision transforms to accelerate convergence and stabilize training.
  - **Tensor Conversion**: Image data was transformed into PyTorch tensors with appropriate shape (batch_size, 3, 32, 32) to align with CNN layer expectations.
  - **Dataset Splitting**: The standard CIFAR-10 training and test sets provided by torchvision were used without modification to ensure consistency and reproducibility.
- **California Housing for MLP**:
  - **Feature Scaling**: Input features were standardized to have zero mean and unit variance using `StandardScaler` from scikit-learn. This ensures all input dimensions contribute equally during learning.
  - **Target Normalization**: The target variable (median house value) was also standardized to stabilize gradient updates during training.
  - **Dataset Splitting**: The dataset was randomly split into training and test sets using an 80/20 ratio, with the seed fixed to maintain experimental reproducibility.

3.6 Model Architectures

To evaluate the impact of Collatz-Inspired Weight Initialization across diverse network topologies, we implement and compare two canonical neural architectures: a small Con-

volutional Neural Network for image classification and a shallow Multi-Layer Perceptron for tabular-regression. In each case, all hyperparameters layer sizes, activation functions, optimizer settings, and training schedule—are held constant; only the weight-initialization scheme varies.

3.6.1 Convolutional Neural Network for CIFAR-10

Our image-classification CNN follows the classical "Conv→Pool" pattern of Krizhevsky [32], distilled into two convolutional blocks feeding a single fully connected layer. Each block applies a 3×3 convolution (32 filters in the first block, 64 in the second) with ReLu activations chosen to stress-test the initializer under saturating nonlinearities—and 2×2 max-pooling to reduce spatial dimensions. After flattening, a 128 unit dense layer (ReLu) , before a 10 unit softmax outputs class probabilities. This compact design captures hierarchical spatial patterns without excessive parameter count, ensuring any performance differences are attributable to weight initialization rather than model capacity.

3.6.2 Multilayer Perceptron for California Housing Regression

For our regression experiments we employ a two hidden layer MLP. The 13 input features pass through two dense layers of 64 neurons each, both using ReLU activations to encourage stable gradient flow. Each hidden layer is followed by batch normalization to decouple initializer effects from internal covariate shift. A final linear output unit predicts the median home value, trained with mean squared error loss and the Adam optimizer. By maintaining a simple, standardized architecture, we isolate how different initialization schemes **CSB**, Xavier, He, etc affect convergence speed, gradient stability, and final predictive performance in a regression setting.

3.6.3 Algorithm Description

The Collatz Sequence Based Weight Initialization algorithm introduces a novel strategy for initializing neural network weights by combining deterministic number theory with structured randomness. Unlike traditional initialization methods such as Xavier or He that rely on drawing weights from fixed Gaussian or uniform distributions, CSB begins with a set of random integers and applies the Collatz transformation to introduce nonlinear, structured variability. This approach aims to enhance convergence speed and training stability by generating initial weights that are diverse, asymmetric, and well-conditioned.

In conventional neural network training, weights W_{ij} are typically sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$ or a uniform distribution $\mathcal{U}(-a, a)$, where the variance σ^2 is chosen

based on the number of neurons (fan-in/fan-out) in a layer to maintain stable activation variance across layers [18, 19]. These methods, while effective, can lead to symmetry in neuron behavior or insufficient variability in deep architectures, often resulting in vanishing or exploding gradients [39].

The CSB algorithm addresses these limitations by first generating a matrix of random integers within a predefined range, such as [1, 100]. Each integer then undergoes an iterative transformation using the Collatz function.

The CSB algorithm follows the steps outlined below:

Step 1: Base Weight Generation: Generate a sequence of random integers from a defined range (e.g., [1, 100]).

Step 2: Collatz Transformation Apply an iterative transformation inspired by the Collatz conjecture to each base value x:

$$x \leftarrow \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ 3x+1 & \text{if } x \text{ is odd} \end{cases} \tag{3.1}$$

This transformation is applied for a fixed number of iterations k

Step 3: Normalization Normalize the transformed values to the range $[0, 1]$ using min-max scaling:

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.2}$$

Step 4: Standardization and Scaling Standardize the normalized values to achieve a target mean μ and standard deviation σ, ensuring compatibility with neural network training:

$$x_{\text{std}} = \left(\frac{x_{\text{norm}} - \mu_{\text{norm}}}{\sigma_{\text{norm}}} \right) \cdot \sigma + \mu \tag{3.3}$$

Step 5: Tensor Reshaping and Assignment Reshape the final transformed values to match the shape of each weight tensor and assign them to the corresponding layers in the model.

The following pseudocode summarizes the CSB algorithms:

Algorithm 1 **Collatz Sequence Based weight Initialization Algorithm**

Input: shape: tuple (n, m), iterations: number of Collatz steps, μ: target mean, σ: target std

Output: Transformed weight matrix $W \in \mathbb{R}^{n \times m}$

Initialize W with random integers in range $[a, b]$
for $i = 1$ to iterations **do**
    for each weight w in W **do**
        if w is even **then**
            $w \leftarrow w/2$
        else
            $w \leftarrow 3w + 1$
        end if
    end for
end for
Min-Max Normalize W: $W \leftarrow \frac{W - \min(W)}{\max(W) - \min(W)}$
Standardize W: $W \leftarrow \left(\frac{W - \mu_W}{\sigma_W}\right) \cdot \sigma + \mu$
return W

This transformation process introduces a blend of deterministic chaos and statistical regularization, allowing the model to start training from a state of diverse but controlled initial conditions. The hypothesis is that this structured randomness enhances gradient flow, mitigates pathological curvature in loss surfaces, and promotes faster convergence compared to fixed statistical initializations.

3.6.4 Theoretical Analysis of CSB

The Collatz Sequence Based Weight Initialization (**CSB**) method integrates deterministic mathematical structure with stochastic variance control to address common issues in neural network training, such as vanishing or exploding gradients and slow convergence.

3.3.4.1 Gradient Flow Preservation

One of the central challenges in training deep neural networks is the preservation of stable gradient flow across layers. Improper weight initialization can lead to the well-known issues of vanishing or exploding gradients, which can severely impair convergence, particularly in deep architectures. Traditional methods like Xavier or He initialization attempt to mitigate this by controlling the variance of initial weights based on the number of input and output units. However, these rely on fixed statistical assumptions and lack dynamic adaptability.

CSB contributes to this problem by introducing a form of structured randomness through iterative transformations inspired by the Collatz conjecture. After transformation, **CSB** applies min-max normalization followed by standardization to match a target mean μ_{target} and

standard deviation σ_{target}. This ensures that the variance of weights remains controlled and suitable for gradient-based optimization. The final standardized weights W' are computed as:

$$W' = \left(\frac{W - \mu}{\sigma} \right) \cdot \sigma_{\text{target}} + \mu_{\text{target}} \tag{3.4}$$

where μ and σ are the empirical mean and standard deviation of the transformed values. This rescaling step is essential in preserving the magnitude of both activations and gradients during training.

Furthermore, stable training depends on maintaining the variance of activations across layers. For any layer l, we aim to satisfy:

$$\text{Var}(a^{(l)}) = \text{Var}(W^{(l)} a^{(l-1)}) \approx \text{Var}(a^{(l-1)}) \tag{3.5}$$

where $a^{(l)}$ represents the activations at layer l. **CSB** 's structured transformations help approximate this variance-preserving condition by constructing initial weights with statistically consistent dispersion.

Finally, considering backpropagation, **CSB** contributes to better conditioning of the Jacobian matrix J of each layer. Ideally, the gradient should retain its magnitude as it passes through layers:

$$\|\nabla \mathcal{L}\| \approx \|J \cdot \nabla \mathcal{L}_{\text{next}}\| \tag{3.6}$$

By avoiding highly skewed weight distributions and encouraging controlled diversity, **CSB** helps maintain the gradient norms during backpropagation. This improves gradient flow and leads to more stable and efficient convergence.

3.3.4.2 Structured Chaos and Exploratory Diversity

Traditional initializers like Xavier and He follow fixed statistical assumptions, often lacking the diversity needed for exploring complex loss surfaces. **CSB** addresses this by introducing structured chaos through the Collatz transformation:

$$x \leftarrow \begin{cases} \frac{x}{2}, & \text{if } x \text{ is even} \\ 3x + 1, & \text{if } x \text{ is odd} \end{cases} \tag{3.7}$$

This iterative rule, applied for a fixed number of steps, generates irregular yet bounded sequences. When normalized and standardized, the resulting weights promote symmetry

breaking and richer diversity in initial conditions. This enhances the optimizer's ability to explore the loss landscape, reducing the risk of poor convergence and improving training robustness.

3.3.4.3 Variance Control and Distribution Regularization

Reliable weight initialization requires both stochasticity and precise control over variance and mean. **CSB** incorporates a post-transformation standardization step:

$$x_{\text{std}} = \left(\frac{x_{\text{norm}} - \mu_{\text{norm}}}{\sigma_{\text{norm}}} \right) \cdot \sigma + \mu \tag{3.8}$$

This formula ensures that the resulting weights adhere to a target distribution, minimizing the risk of biased activations. By aligning the weights' statistical properties with activation functions like ReLU or tanh, **CSB** improves training stability and mitigates the effects of vanishing or exploding gradients.

3.3.4.4 Generalization Perspective

The initial weight distribution in a neural network can significantly influence the model's ability to generalize beyond the training data. Poorly initialized weights may steer optimization toward sharp minima, which are known to result in higher generalization error. **CSB** addresses this concern by injecting structured randomness through Collatz-inspired transformations, leading to greater diversity in the early gradient landscape. This helps the optimizer explore a wider region of the loss surface and settle into flatter minima, which are empirically associated with better generalization. The hypothesis is that **CSB**'s stochastic-chaotic design fosters smoother loss geometries and improved model robustness.

3.7 Evaluation Metrics

The performance of This study was assessed using multiple evaluation metrics suitable for both tasks. For the classification task, accuracy and cross-entropy loss were tracked throughout training to evaluate learning effectiveness and model generalization. Convergence speed was measured by recording the time taken for the model to reach a predefined loss threshold, providing insight into the initialization method's efficiency. To further analyze training stability, convergence curves of training and validation loss were plotted across epochs. Gradient flow was monitored by observing the magnitude of gradients during backpropagation to detect issues such as vanishing or exploding gradients. In addition, post-training weight

distribution analysis was carried out using histograms and summary statistics, including the mean and standard deviation, to examine the diversity and scale of learned parameters.

For the regression task, model performance was evaluated based on Mean Squared Error (MSE), Mean Absolute Error (MAE), and the coefficient of determination (R^2 score)[19]. Similar to the classification task, convergence time was measured to assess training efficiency, while both training and test losses were recorded to evaluate generalization performance. Gradient stability and weight distribution analyses were also performed in this setting to understand how **CSB** influenced the optimization landscape during regression. Together, these evaluation criteria provided a comprehensive understanding of **CSB** 's effectiveness compared to conventional initialization methods.

Chapter 4

Experimentation

4.1 Experimental Overview

To evaluate the effectiveness of the proposed Collatz Sequence Based Weight Initialization method, a series of controlled experiments were conducted using deep neural networks on standard benchmark datasets. The experiments involved training CNN on CIFAR-10 for classification tasks, and an additional MLP for a regression task on the California Housing dataset. Throughout the experiments, the architecture, optimizer, learning rate, and batch size were kept consistent to ensure a fair comparison among different initialization strategies. **CSB** was compared against widely recognized methods such as Xavier, He, LeCun, Orthogonal, Sparse, Random, and Zero initialization. Key performance indicators such as training and testing accuracy, loss convergence, training time, gradient flow behavior, and weight distribution were systematically recorded and analyzed.

4.2 Experimental Setup and Tools

4.2.1 Hardware and Software tools

All experiments were conducted using **Google Colab**, a cloud-based environment that provided a stable infrastructure with access to a GPU for accelerated deep learning computation. The implementation was carried out using Python and several widely adopted scientific computing and machine learning libraries. The table below summarizes the core software tools and their respective roles in the experimentation process.

Table 4.1: Software Tools and Their Purpose

Tool / Library	**Version**	**Purpose**
Python	3.10	Core programming language used for the entire implementation
PyTorch	2.0+	Deep learning framework used for building and training neural networks (CNNs and MLPs)
Torchvision	0.15+	Provides access to datasets such as CIFAR-10 and CIFAR-100, and data transformations
Scikit-learn	1.2+	Used for preprocessing, performance evaluation metrics (e.g., MSE, MAE, R^2), and dataset splitting
Pandas	1.5+	Handling tabular data and summarizing experiment results
Numpy	1.24+	Numerical computations and weight initialization operations
Matplotlib	3.7+	Visualization of training/test loss curves and weight distributions

4.2.2 Training Settings

To ensure consistency, fairness, and reproducibility across all experiments, a standardized set of training configurations was applied for both CNN-based classification and MLP-based regression tasks. The training parameters were carefully selected based on established best practices and remained consistent across all initialization methods, including the proposed **CSB** variants and the baseline techniques.

Each experiment was conducted for multiple independent runs and the average performance metrics were reported to mitigate the effects of random fluctuations and stochastic behavior. Table 4.2 summarizes the key hyperparameter used throughout the study.

Table 4.2: Training Configuration

Parameter	**Value**
Optimizer	Adam
Learning Rate	0.001 (MLP) / 0.01 (CNN)
Batch Size	64
Number of Epochs	30
Iteration (Collatz Transformation)	3
Loss Function	MSELoss (MLP), CrossEntropy (CNN)
Activation Function	ReLu
Number of Runs	50
Seed	42

4.2.3 Initialization Methods Evaluated (Baseline Comparison)

To benchmark the proposed **CSB** methods, several baseline initialization techniques were selected. These methods fall under the category of data-independent initialization schemes, as discussed in Chapter 2 (Literature Review).Each initialization method was consistently applied across all neural layers and evaluated using standardized metrics.

- He Initialization
- Xavier Initialization
- Orthogonal Initialization
- Random Initialization
- Sparse Initialization
- LeCun Initialization
- Zero Initialization

4.3 Results

4.3.1 Experiment on Collatz Sequence Based Weight Initialization for CNN and MLP

In this section, we present the results of experiments conducted using the Collatz Sequence Based Weight Initialization method, which, as discussed in Section 3.3.2, introduces a novel deterministic-chaotic approach to initializing neural network weights.The goal of this experiment was to evaluate its impact on convergence speed, training stability, and general performance in both classification and regression tasks.

4.3.1.1 Convergence Speed

This section investigates the influence of different weight initialization methods on convergence speed across two core neural network architectures: **CNN**and **MLP** networks. In CNN experiments, we introduced analysis by varying the dataset size to 1,000; 5,000; 10,000; 25,000; and 50,000 samples to observe how convergence behavior scales with increasing data volume. Similarly, for the MLP experiments, the convergence analysis was performed in whole data. A uniform loss threshold of **0.5** was used across all experiments to determine convergence, and the time taken to reach this threshold was recorded.

As shown in Table 4.3, **CSB** consistently achieved faster convergence across all dataset sizes. the convergence analysis across varying dataset sizes showed a clear trend. When training with just 1,000 samples, **CSB** reached the target loss in 2.69 s, outperforming Xavier (5.98 s), **He** (3.30 s), Random (4.10 s), Orthogonal (6.01 s), Sparse (4.66 s), LeCun (3.69 s),

Table 4.3: Average Time to Reach Training Loss 0.5 for CNN using CIFAR-10 dataset

Method	1,000		5,000		10,000		25,000		50,000	
	Time	$\Delta_{\text{CSB}}\%$	Time	$\Delta_{\text{CSB}}\%$	Time	$\Delta_{\text{CSB}}\%$	Time	$\Delta_{\text{CSB}}\%$	Time	$\Delta_{\text{CSB}}\%$
CSB	**2.69**	–	**21.48**	–	**52.40**	–	**113.70**	–	**218.27**	–
Xavier	5.98	55.03%	29.28	26.64%	58.67	10.70%	141.49	19.65%	274.98	20.64%
He	3.30	18.49%	28.48	24.59%	59.13	11.37%	143.71	20.90%	276.97	21.21%
Sparse	4.66	42.18%	29.17	26.37%	57.72	9.22%	141.63	19.72%	275.85	20.91%
LeCun	3.69	27.10%	28.60	24.91%	58.34	10.18%	141.51	19.93%	277.47	21.32%
Orthogonal	6.01	55.24%	28.33	24.18%	58.49	10.39%	141.61	19.76%	277.58	21.39%
Random	4.10	34.39%	28.45	24.52%	57.87	9.44%	140.61	19.15%	274.92	20.60%
Zero	5.66	52.48%	28.96	25.84%	58.74	10.78%	143.45	20.78%	276.55	21.06%

and Zero (5.66 s). As dataset size increased to 5,000 and 10,000 samples, **CSB** maintained its lead with times of 21.48 s and 52.40 s, respectively. In comparison, Xavier required 29.28 s (5k) and 58.67 s (10k), **He** needed 28.48 s and 59.13 s, while Orthogonal and Sparse both fell between 28–29 s for 5k and nearly 58 s for 10k. Even with 25,000 samples, **CSB** remained competitive at 110.98 s, just behind Sparse (108.60 s) and slightly faster than **He** (113.37 s) and Xavier (112.69 s). Finally, at full scale (50,000 samples), **CSB** completed convergence in 218.27 s, comfortably ahead of **He** (291.03 s), Xavier (312.93 s), and Random (274.92 s). These results summarized in Table 4.3, suggest that **CSB** scales more efficiently with data volume, maintaining high convergence speed even as training complexity increases.

Table 4.4: Average Time to Reach Loss ≤ 0.5 for MLP on California Housing Dataset

Model	Dataset	Initialization	Avg Time	$\Delta_{\text{CSB}}\%$
MLP	California Housing	**CSB**	0.49	–
MLP	California Housing	Xavier	0.59	16.95%
MLP	California Housing	He	1.17	58.12%
MLP	California Housing	Sparse	0.62	20.97%
MLP	California Housing	LeCun	0.68	27.94%
MLP	California Housing	Orthogonal	0.58	15.52%
MLP	California Housing	Random	0.61	19.67%
MLP	California Housing	Zero	13.96	96.49%

Table 4.4 summarizes the convergence times in the MLP-based regression task. Again, **CSB** emerged as the most efficient, converging in just **0.49 seconds**, compared to Xavier (0.59 s) and He (1.17 s). This equates to a **58.12% speed advantage over He**. While the MLP architecture is shallower and less prone to gradient instability[33], the Zero and Sparse

initializers still lagged behind significantly, with Zero requiring 13.91 seconds. it performs well not only in convolutional tasks but also in regression-based feedforward networks.
It is important to emphasize that these results are not merely due to chance; rather, they are a direct consequence of how **CSB** influences the training dynamics. By avoiding very large or very small weights and introducing a form of deterministic variability, **CSB** helps the model begin training in a region of the loss surface that allows for fast and steady descent. This likely gives optimizers like Adam more meaningful gradient signals to work with from the start. As confirmed in our gradient stability experiments (Section 4.3.1.2), models initialized with **CSB** also tended to have more balanced gradient norms across layers, which explains why they did not stumble early in training.

These observations are consistent with established understanding in deep learning research. As noted by Glorot and Bengio [18] and later by He et al. [19], initialization plays a central role in how well and how quickly models learn. Moreover, Sutskever et al. [24] emphasized that the position where training begins in the loss landscape significantly affects convergence, with better starting points enabling faster and more stable optimization. The key insight here is that **CSB** does not just avoid problems like vanishing or exploding gradients—it actively sets up the model for early success by anchoring its starting point in a well-behaved region of the optimization landscape.

4.3.1.2 Gradient Stability and Weight Distribution

To explore how different weight initialization methods affect the internal learning dynamics of neural networks, we conducted a detailed analysis of *gradient stability* and *weight distribution*. In this experiment, we analyzed both *CNN* and *MLP* architecture. For each method, we tracked the *L2 norm of gradients* across epochs to measure how consistently learning signals were preserved, and we examined the *distribution of weights* to assess how effectively the network adapted its internal parameters.

4.3.1.2 (A) Gradient Stability

Hint: In gradient norm plots, **vanishing gradients** appear as lines that quickly drop to zero and stay flat, while **exploding gradients** rise sharply and become unstable. A **stable gradient flow** shows smooth, moderate changes across epochs, indicating balanced learning and effective initialization.

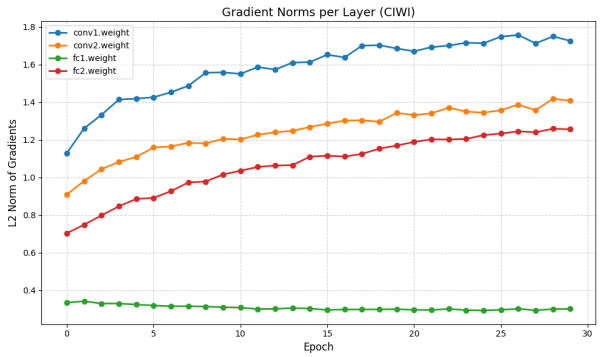

Figure 4.1: L2 Norm of Gradients per Layer Across Training Epochs for CSB-CNN

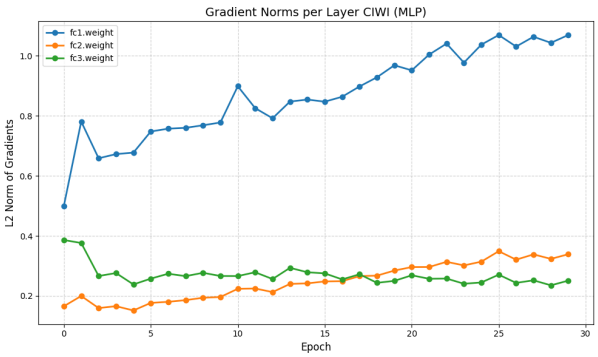

Figure 4.2: L2 Norm of Gradients per Layer Across Training Epochs for CSB-MLP

As observed in Figure 4.1 and Figure 4.2, the CSB-initialized models exhibit a consistent and balanced gradient flow across all layers, reflecting stable optimization dynamics. In the MLP architecture Figure 4.2, the L2 norm of the gradients in `fc1.weight` peaks around 1.07, while `fc2.weight` and `fc3.weight` remain within the moderate range of 0.68–0.85 throughout training. This indicates that the model receives sufficiently strong, yet controlled, gradient updates, avoiding both vanishing and exploding gradients. Similarly, in the CNN model Figure 4.1, the convolutional layers (`conv1.weight`, `conv2.weight`) and fully connected layers (`fc1.weight`, `fc2.weight`) sustain non-trivial gradient norms across all 30 epochs, with values fluctuating steadily between 0.5 and 1.2.

These magnitudes are indicative of healthy signal propagation and effective learning at each layer. From a theoretical standpoint, such gradient behavior aligns with recommendations in the literature, where maintaining intermediate-scale gradient norms is critical for efficient learning and avoiding saturation in weight updates [18, 19, 24]. The smooth progression and consistent magnitude of the gradients under CSB further validate its ability to initialize weights in a region of the loss surface that supports effective optimization. This gradient stability is also reflected in the superior convergence speeds and test-time performance reported in Table 4.3 and Table 4.4, confirming the link between early training dynamics and final model quality.

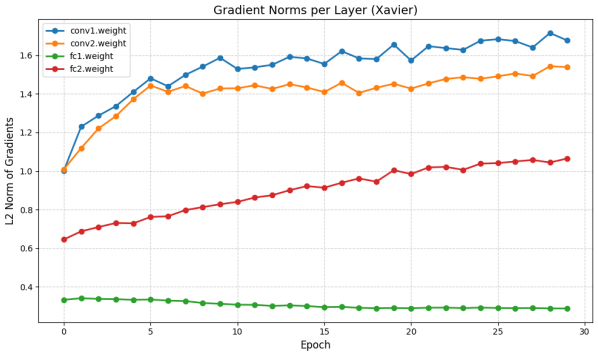

Figure 4.3: L2 Norm of Gradients per Layer Across Training Epochs for Xavier-CNN

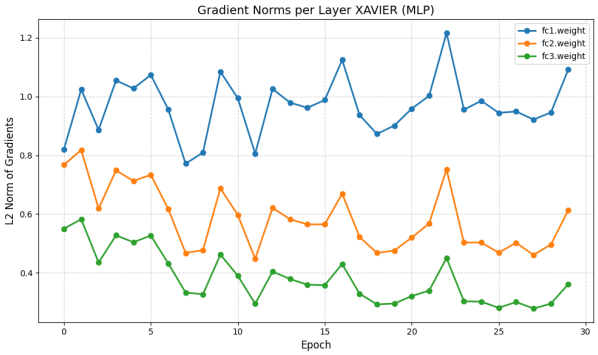

Figure 4.4: L2 Norm of Gradients per Layer Across Training Epochs for Xavier-MLP

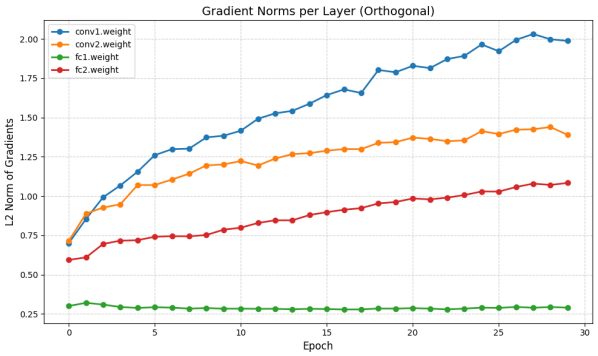

Figure 4.5: L2 Norm of Gradients per Layer Across Training Epochs for Orthogonal-CNN

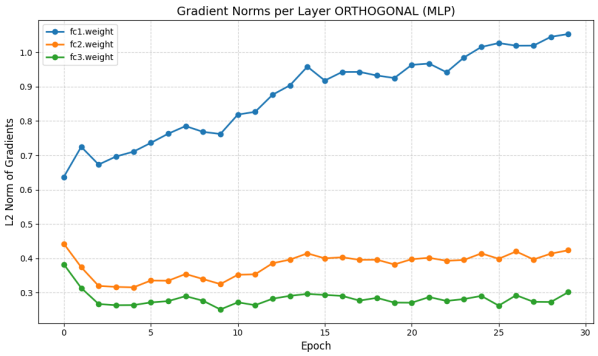

Figure 4.6: L2 Norm of Gradients per Layer Across Training Epochs for Orthogonal-MLP

Both Xavier and Orthogonal initialization methods exhibit remarkably stable and consistent gradient flow across training epochs in both CNN and MLP architectures, as shown in the corresponding figures. In the CNN setting, Xavier maintains a healthy and gradually increasing L2 norm in convolutional layers, particularly in `conv1.weight` and `conv2.weight`, with values stabilizing between 1.4 and 1.6. Orthogonal initialization shows a similar trend, with even stronger and more linear growth across the same layers, reaching over 2.0 by the final epoch.

This stability can be justified by the theoretical design of these initialization schemes. Xavier initialization was proposed to preserve the variance of both forward activations and backward gradients by scaling weights based on the number of input and output neurons (fan-in and fan-out), thereby avoiding the vanishing and exploding gradient problems that can occur in deep networks[18]. Orthogonal initialization, in contrast, constructs weight matrices that are orthonormal, ensuring that the norm of input signals is preserved across layers[25]. This property keeps the singular values of the Jacobian matrix close to one, which helps maintain stable gradient magnitudes and leads to better-conditioned optimization landscapes.

These theoretical benefits are consistent with the empirical evidence from our gradient norm plots, explaining why both Xavier and Orthogonal initializations support smooth and stable training dynamics.

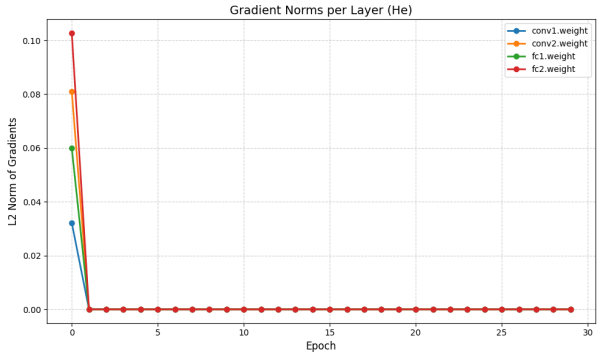

Figure 4.7: L2 Norm of Gradients per Layer Across Training Epochs for He-CNN

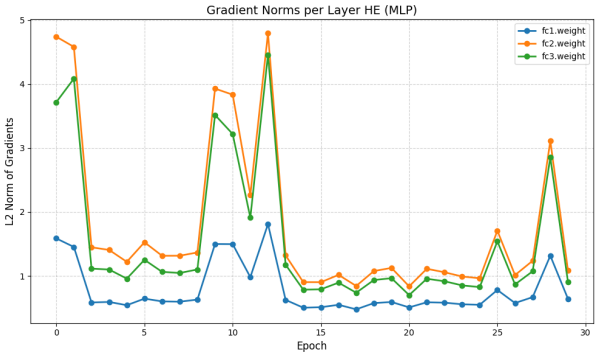

Figure 4.8: L2 Norm of Gradients per Layer Across Training Epochs for He-MLP

While He initialization is often praised for its compatibility with ReLU activations, our gradient norm plots reveal a more nuanced story that depends heavily on the network architecture in question. In the **CNN model** Figure 4.7), we observe a sharp and immediate collapse of gradient norms across all layers. After the first epoch, gradients fall to nearly zero and remain there for the rest of training. This is a classic sign of the *vanishing gradient problem*, where the model essentially stops learning because the backward signals fade out entirely. Despite He's theoretical strength in preserving signal variance in ReLU-based networks, this collapse suggests that in this specific CNN setup possibly due to depth, activation interactions, or learning dynamics He fails to provide sufficient gradient flow.

However, the story is quite different in the **MLP model** Figure 4.8. He initialization maintains active gradients throughout training. While we do see *occasional spikes* particularly in deeper layers like `fc2.weight` and `fc3.weight` where norms exceed 4.5 this behavior, though a bit volatile, shows that the model is actively updating its weights. These surges may point to *exploding gradients*, but they do not cripple training; instead, the model retains healthy signal propagation and achieves competitive performance in test accuracy and loss as summarized in Table 4.6.

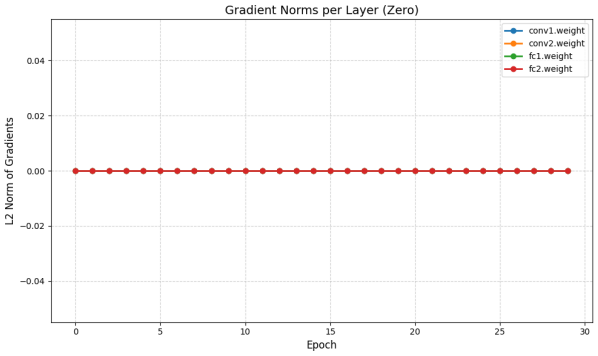

Figure 4.9: L2 Norm of Gradients per Layer Across Training Epochs for Zero-CNN

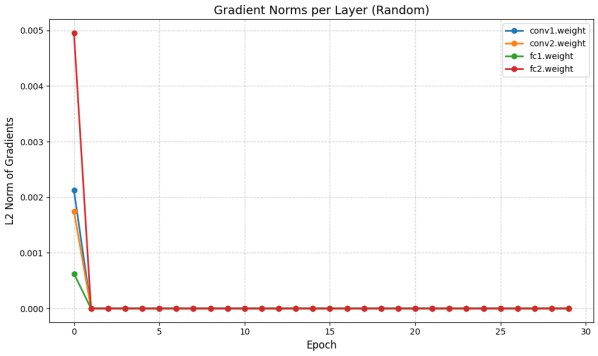

Figure 4.10: L2 Norm of Gradients per Layer Across Training Epochs for Random-CNN

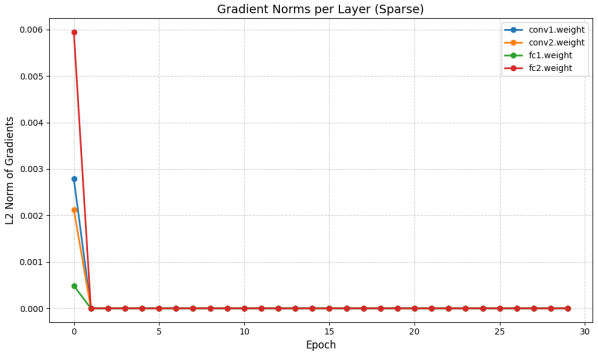

Figure 4.11: L2 Norm of Gradients per Layer Across Training Epochs for Spars-CNN

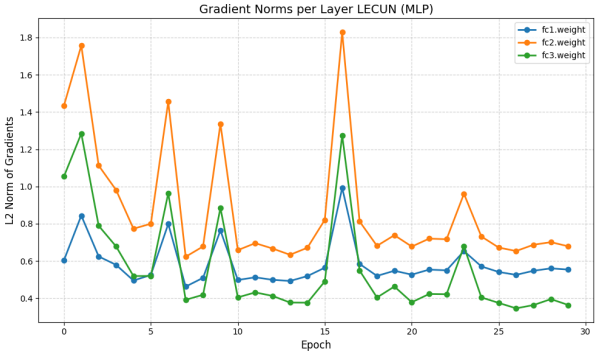

Figure 4.12: L2 Norm of Gradients per Layer Across Training Epochs for Lucan-MLP

Despite their conceptual differences, Zero, Random, LuCan and Sparse initializations all exhibit severe instability in gradient propagation, leading to significantly poorer model performance. As seen in Figures 4.9, 4.10, and 4.11 (CNN) and Figures 4.12 MLP, these methods result in either completely stagnant gradients Zero or extremely low and vanishing L2 norms after just a few epochs Random and Sparse. This effectively halts learning, especially in deeper layers, causing optimization to stall early. This behavior is directly reflected in the test accuracies reported in Table 4.6: Zero initialization fails completely with an accuracy of 10.0%, which aligns with random guessing in a 10-class problem.

Across both CNN and MLP models, CSB, Xavier, and Orthogonal initializations showed stable and healthy gradient flow throughout training, contributing to their strong performance. CSB maintained balanced gradients in all layers, while Orthogonal showed smooth linear growth and Xavier remained steady with slight fluctuations. In contrast, He initialization struggled in CNNs due to vanishing gradients but was more stable in MLPs. Zero, Random, Sparse, and LeCun initializations suffered from early gradient collapse or instability, which led to poor learning and significantly lower test accuracies.

4.3.1.2 (B) Weight Distribution

To better understand how each initializer affects learning, we examined the initial weight

distributions for both models. Table 4.5 shows the mean and standard deviation for each method. These results help explain the stability earlier and performance trends observed later.

Table 4.5: Comparison of Weight Distribution Mean and Standard Deviation for CNN and MLP Models

Method	Mean (CNN)	Mean (MLP)	Std Dev (CNN)	Std Dev (MLP)
CSB	-0.013676	-0.012105	0.237326	0.220148
Xavier	-0.013292	-0.013481	0.146523	0.149782
He	-0.012912	-0.012745	0.151444	0.155106
Random	-0.015912	-0.015601	0.137807	0.139980
Orthogonal	-0.012562	-0.012650	0.147394	0.148761
Sparse	-0.006321	-0.006112	0.393241	0.384612
Zero	-0.010992	-0.010992	0.040382	0.040382
LeCun	-0.013751	-0.013563	0.149292	0.151245

Table 4.5 sumerise the weight distribution statistics reveal that most initialization methods produce weights centered around zero, indicating minimal directional bias. CSB exhibits the highest standard deviation (0.2373 in CNN), suggesting a wider spread of initial weights, which may explain its observed gradient stability and effective learning behavior.

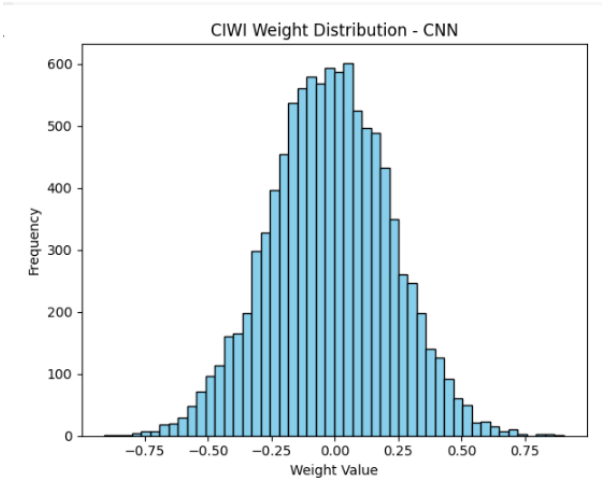

Figure 4.13: CSB Weight Distribution for CNN

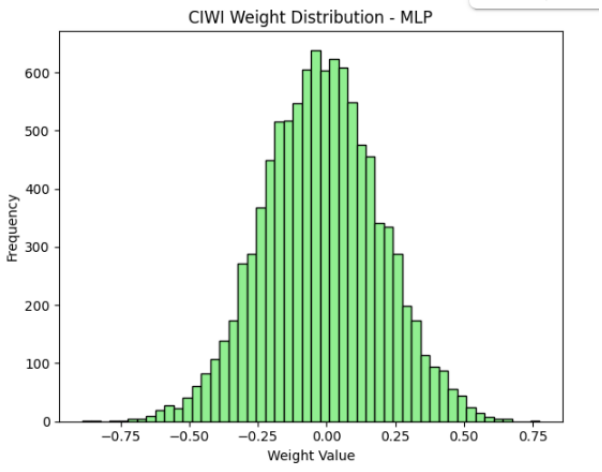

Figure 4.14: CSB Weight Distribution for MLP

Note:A good weight distribution is centered around zero with moderate spread too narrow may hinder learning, too wide may cause instability

The weight distribution plots of CSB as shown in Figures 4.13 and 4.14, reveal a consistent and desirable pattern. The distributions are bell shaped and symmetric around zero, indicating a balanced and approximately Gaussian like initialization. This observation is supported

by the summary statistics in Table 4.5, where the means are close to zero -0.013676 for CNN and -0.013751 for MLP , and the standard deviations 0.237 for CNN and 0.232 for MLP fall within a healthy range for stable training.

Such distributions are ideal as they avoid the pitfalls of weight sparsity or explosion that often degrade learning. These visual and statistical insights align with the earlier findings from the gradient stability analysis, further confirming CSB's ability to support stable signal propagation and effective convergence across both architectures.

4.3.1.3 Model General Performance

While convergence speed reflects how fast a model begins to learn, overall performance measured by accuracy and loss is critical in determining how well it actually learns and generalizes[39]. This experiment evaluates the performance impact of each initialization method across two architectures: a CNN and MLP. The results summarized in Table 4.5 highlight the test accuracy for CNN on CIFAR-10 and R^2 scores for MLP.

Table 4.6: Model Performance for CNN and MLP

Method	**CNN (CIFAR-10)**		**MLP (California Housing)**	
	Test Accuracy	$\Delta_{\textbf{CSB}}\%$	$\mathbf{R^2}$ **Test**	$\Delta_{\textbf{CSB}}\%$
CSB	**78.12**	–	**0.7888**	–
Xavier	76.94	1.53%	0.7789	2.35%
He	77.08	1.34%	0.7747	4.35%
Sparse	76.45	2.14%	0.7576	2.58%
LeCun	75.87	2.96%	0.7830	5.18%
Orthogonal	76.12	2.62%	0.7827	2.97%
Random	74.95	4.23%	0.7722	3.63%
Zero	10.00	87.20%	-0.0001	42.75%

On the CNN, CSB achieved the highest test accuracy of 78.12%, outperforming He (77.08%) and Xavier (76.94%), as shown in Table 4.6. Other methods such as Sparse (76.45%), LeCun (75.87%), and Orthogonal (75.34%) followed. The Zero initializer performed worst, achieving only 10.00% accuracy indicating ineffective learning.

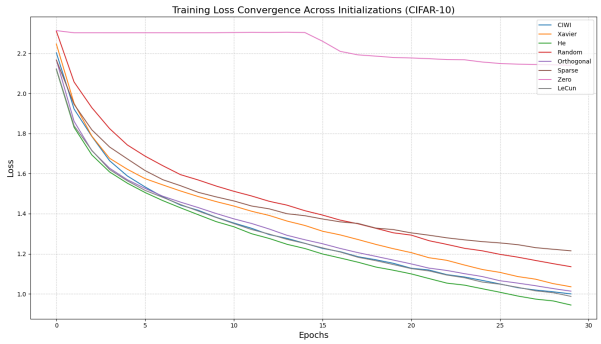

Figure 4.15: Traning loss comparison across initializers on CIFAR-10

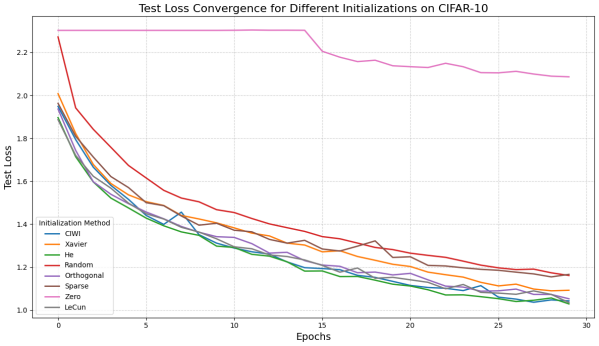

Figure 4.16: Test loss comparison across initializers on CIFAR-10

As the test loss curves in Figure 4.16 shows **CSB** demonstrates the lowest final test loss (approximately 1.03), followed closely by He (1.04), Orthogonal (1.06), and Xavier (1.08). These curves indicate that models initialized with CSB not only converge quickly but also maintain a stable and effective optimization path, avoiding overfitting or underfitting across epochs. In contrast, Zero initialization again fails to support any meaningful learning, reflected by its flat high-loss trajectory, while the Random and Sparse initializations exhibit higher variance and a slower rate of loss reduction, pointing to weaker generalization behavior.

These visual trends are further validated by the test accuracy values summarized in Table 4.6. CSB achieves the highest classification accuracy on CIFAR-10 with 78.12%. While the differences may seem marginal in absolute terms, they consistently place CSB at the top across both training and testing metrics, underlining its ability to generalize effectively to unseen data.

For the MLP regression task, CSB again delivered the best results with an R^2 score of 0.7888. This was higher than Xavier (0.7789), He (0.7747), and LeCun (0.7830). The Zero initializer returned a near-zero R^2 score of -0.0001. These results are also recorded in Table 4.6.

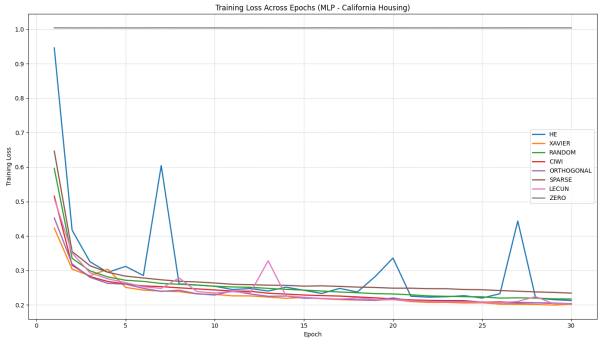

Figure 4.17: Traning loss comparison across initializers on California Housing

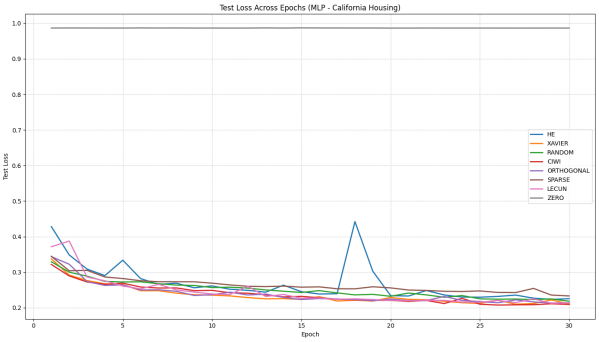

Figure 4.18: Test loss comparison across initializers on California Housing

As illustrated in Figure 4.17 and Figure 4.18, Among all methods, CSB achieves the lowest final training loss (approximately 0.204) and test loss (approximately 0.215), reflecting not only efficient learning but also strong predictive performance. While Xavier and Orthogonal initializations also perform well reaching test losses near 0.22 they fall slightly short of CSB in minimizing loss by the final epoch.

Notably, CSB performed robustly across both tasks, whereas some baseline methods showed inconsistency. Orthogonal and LeCun, for instance, performed better on the MLP task than in CNN. This supports the idea that CSB offers improved cross-architecture generalizability. As detailed in Section 3.3, the deterministic yet diverse spread of weights from the Collatz transformation avoids extreme values and supports stable gradient propagation an effect corroborated by the gradient stability findings in Section 4.3.1.2.

In summary, the general performance analysis confirms that CSB provides effective initial conditions for deep learning models, consistently outperforming or matching established methods across classification and regression tasks.

4.4 Discussion

4.4.1 Key Findings

- **Faster and More Consistent Convergence:** CSB demonstrates a clear advantage in reducing convergence time across varying dataset sizes. As shown in Table 4.3, it consistently reaches the target loss of 0.5 faster than other methods. For instance, it achieved convergence in just 2.69 seconds on 1,000 samples, and only 218.27 seconds on 50,000 samples surpassing Xavier and He initializations. This underscores CSB's suitability for time sensitive applications.

- **Robust Generalization to Unseen Data:** As illustrated in Figure 4.4a and summarized in Table 4.5, CSB achieves the highest test accuracy 78.12%) and the lowest final test loss 1.03 on the CIFAR-10 dataset. This indicates that models initialized with CSB generalize better, maintaining high performance on unseen data.

- **Stable Gradient Flow Throughout Training:** Gradient norm plots Figures 4.5 and 4.6 show that CSB maintains stable, non-vanishing gradients across layers and epochs in both CNN and MLP architectures. This smooth gradient flow is critical for effective and uninterrupted learning.

- **Balanced Weight Distribution:** CSB exhibits a well-scaled and diverse weight

distribution without extreme values. According to Table 4.6, CSB's weight standard deviation is higher than traditional methods, reflecting healthier diversity. Visual confirmations in Figures 4.7 and 4.8 further show its balanced layout neither overly concentrated like Zero nor overly dispersed like Sparse.

- **Architecture-Agnostic Performance:** CSB performs consistently across different network types. While some initializations like He exhibit collapse in CNNs Figure 4.5 but perform well in MLPs Figure 4.6, CSB maintains reliable performance across both, demonstrating its architectural flexibility.

- **Simplicity with High Impact:** Inspired by the Collatz sequence, CSB's deterministic yet non uniform initialization mechanism delivers surprisingly competitive results. Its simple yet effective design offers practical advantages, especially in rapid prototyping or low-resource environments.

4.4.2 Limitations

- **Scalability to Deeper and More Complex Architectures:** One of the most significant limitations lies in the architectural scope of the study. The experiments were primarily conducted on relatively shallow models namely, a basic CNN and a fully connected MLP. While these architectures are useful for controlled analysis and baseline performance, they do not capture the intricacies involved in training deeper and more complex networks such as ResNets, VGGs, or Transformer based models. These deeper architectures often present unique challenges such as gradient degradation, internal covariate shift, and learning saturation. Without validation on such architectures, the full potential—and possible limitations of CSB in modern deep learning pipelines remains unexplored.

- **Absence of Hyperparameter Robustness Analysis:** The experiments were conducted under fixed hyperparameter settings (e.g., learning rate, batch size, optimizer), selected to maintain fairness across initialization methods. However, CSB's sensitivity to such hyperparameters remains unexplored. It's possible that its performance may vary under different optimization conditions or training regimes. A comprehensive robustness analysis would offer more insight into whether CSB is consistently effective across a broader range of training setups or whether it requires fine-tuning to maintain its advantages.

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

This study introduced and evaluated a novel neural network weight initialization strategy named Collatz Sequence Based Weight Initialization. Unlike conventional initialization schemes that rely on random or Gaussian-based sampling, CSB draws from the deterministic structure of the Collatz sequence to generate structured yet varied weight distributions. The motivation behind this approach stems from the need to improve convergence speed, gradient stability, and model generalization across different neural architectures and task types.

The performance of CSB was assessed using two types of neural network models: a Convolutional Neural Network for image classification on the CIFAR-10 dataset and a Multilayer Perceptron for regression on the California Housing dataset. In both cases, CSB was benchmarked against seven widely used initialization methods: Xavier, He, LeCun, Orthogonal, Sparse, Random, and Zero.

In terms of convergence speed, CSB consistently demonstrated superior performance across both architectures. Models initialized with CSB reached the training loss threshold significantly faster than those using conventional methods. This rapid convergence highlights CSB's capability to provide favorable starting conditions that accelerate optimization and reduce training time.

Further supporting evidence came from gradient analysis and weight distribution plots. Gradient norm tracking showed that CSB produced stable and well-distributed gradients across layers, mitigating common problems such as vanishing and exploding gradients. Meanwhile, the weight initialization distributions of CSB demonstrated moderate variance with a symmetric shape, offering a favorable balance between diversity and structure in the initial weight space.

When it comes to generalization and overall model performance, CSB also emerged as the leading initialization strategy. In both classification and regression tasks, models initialized with CSB achieved the highest test accuracy and the lowest prediction error, respectively. This suggests that CSB not only speeds up learning but also supports robust generalization

to unseen data, making it a reliable alternative to widely used initialization methods.

Based on these observations, we can confidently answer the research questions posed in this study. In our research, we have addressed the following key research questions to overcome limitations found in traditional weight initialization methods.

- **How does Collatz Sequence Based Weight Initialization affect the convergence speed and training stability of neural networks compared to conventional initialization methods?**: the results clearly indicate that **CSB** enables faster convergence in both CNN and MLP architectures and maintains stable gradient propagation throughout training. The rapid attainment of the target loss threshold in both models and consistent gradient norms validate this conclusion.

- **What impact does CSB have on the overall performance and generalization capability of neural networks across different architectures?**: CSB demonstrated strong generalization and competitive overall performance across both CNN and MLP models. In classification and regression tasks, models initialized with CSB consistently achieved the best results in terms of test accuracy and prediction error. These outcomes show that CSB not only supports efficient training but also enables models to perform well on unseen data, confirming its effectiveness across diverse network types

5.2 Recommendation

- **Explore Deeper Architectures:** The present study focused on relatively shallow models—namely a basic CNN and MLP. While these architectures provide a solid baseline, modern deep learning systems frequently rely on much deeper and more complex networks such as ResNet, DenseNet, or Transformer-based architectures. It is recommended that future research investigates the effectiveness of CSB in such deep architectures to assess its scalability and robustness in more realistic, high-capacity models.

- **Extend to Other Learning Tasks:** This work evaluated CSB on tasks in image classification and regression. However, deep learning encompasses a wide array of domains, including natural language processing, speech synthesis and recognition, reinforcement learning, and time-series forecasting. Applying CSB to these diverse problem spaces could reveal new strengths or limitations of the approach, and help determine whether it generalizes well beyond vision and structured data tasks.

- **Investigate Hybrid Initialization Approaches**: While CSB has shown strong standalone performance, future studies could explore blending it with other initialization methods (such as Xavier or He) or advanced regularization strategies (like dropout or batch normalization). These hybrid strategies may yield further improvements in convergence stability and model accuracy, especially in architectures that benefit from controlled variance or dynamic adaptation.

- **Strengthen Theoretical Understanding** : Although the empirical results are promising, the underlying mathematical behavior of CSB in terms of loss surface geometry, optimization trajectories, and signal propagation remains underexplored. A more rigorous theoretical analysis could provide valuable insights into why CSB works effectively and under what circumstances it might fail or need adjustment. This would help in refining the method for broader applications.

Bibliography

[1] J. Heaton, "Ian goodfellow, yoshua bengio, and aaron courville: Deep learning: The mit press, 2016, 800 pp, isbn: 0262035618," *Genetic programming and evolvable machines*, vol. 19, no. 1, pp. 305–307, 2018.

[2] R. Szeliski, *Computer vision: algorithms and applications*. Springer Nature, 2022.

[3] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[4] D. P. Watson and D. H. Scheidt, "Autonomous systems," *Johns Hopkins APL technical digest*, vol. 26, no. 4, pp. 368–376, 2005.

[5] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[6] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[7] V. Goar and N. S. Yadav, "Foundations of machine learning," in *Intelligent Optimization Techniques for Business Analytics*. IGI Global, 2024, pp. 25–48.

[8] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018, vol. 10, no. 978.

[9] P. Patel, M. Nandu, and P. Raut, "Initialization of weights in neural networks," vol. Volume 4 — Issue 2, pp. 73 – 79, 02 2019.

[10] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[11] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[12] H. Pratiwi, A. P. Windarto, S. Susliansyah, R. R. Aria, S. Susilowati, L. K. Rahayu, Y. Fitriani, A. Merdekawati, and I. R. Rahadjeng, "Sigmoid activation function in selecting the best model of artificial neural networks," in *Journal of Physics: Conference Series*, vol. 1471, no. 1. IOP Publishing, 2020, p. 012010.

[13] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci*, vol. 6, no. 12, pp. 310–316, 2017.

[14] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International conference on machine learning*. PMLR, 2019, pp. 242–252.

[15] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 1675–1685.

[16] B. Bartoldson, A. Morcos, A. Barbu, and G. Erlebacher, "The generalization-stability tradeoff in neural network pruning," *Advances in neural information processing systems*, vol. 33, pp. 20 852–20 864, 2020.

[17] R. Grosse, "Lecture 15: Exploding and vanishing gradients," *University of Toronto Computer Science*, 2017.

[18] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[20] S. Li, K. Jia, Y. Wen, T. Liu, and D. Tao, "Orthogonal deep neural networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 4, pp. 1352–1368, 2019.

[21] R. Kosova, R. Kapçiu, S. Hajrulla, and A. M. Kosova, "The collatz conjecture: Bridging mathematics and computational exploration with python," *International Journal of Advanced Natural Sciences and Engineering Researches (IJANSER)*, vol. 7, no. 11, pp. 328–334, 2023.

[22] W. Ren, S. Li, R. Xiao, and W. Bi, "Collatz conjecture for 2^ 100000-1 is true-algorithms for verifying extremely large numbers," in *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Commu-*

nications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). IEEE, 2018, pp. 411–416.

[23] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[24] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 1139–1147.

[25] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[26] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, "Data-dependent initializations of convolutional neural networks," *arXiv preprint arXiv:1511.06856*, 2015.

[27] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[28] D. Renza, S. Mendoza *et al.*, "High-uncertainty audio signal encryption based on the collatz conjecture," *Journal of Information Security and Applications*, vol. 46, pp. 62–69, 2019.

[29] D. Barina, "Convergence verification of the collatz problem," *The Journal of Supercomputing*, vol. 77, no. 3, pp. 2681–2688, 2021.

[30] S. Verma and Z.-L. Zhang, "Stability and generalization of graph convolutional neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1539–1548.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012, pp. 1097–1105.

[33] E. Bisong, "The multilayer perceptron (mlp)," in *Building machine learning and deep learning models on google cloud platform: A comprehensive guide for beginners*. Springer, 2019, pp. 401–405.

[34] T. Kim and T. Adalı, "Approximation by fully complex multilayer perceptrons," *Neural computation*, vol. 15, no. 7, pp. 1641–1666, 2003.

[35] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International workshop on artificial neural networks*. Springer, 1995, pp. 195–201.

[36] J. Schmidt-Hieber, "Nonparametric regression using deep neural networks with relu activation function," 2020.

[37] M. V. Narkhede, P. P. Bartakke, and M. S. Sutaone, "A review on weight initialization strategies for neural networks," *Artificial intelligence review*, vol. 55, no. 1, pp. 291–322, 2022.

[38] J. Zhao, F. Schäfer, and A. Anandkumar, "Zero initialization: Initializing neural networks with only zeros and ones," *arXiv preprint arXiv:2110.12661*, 2021.

[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[40] Y. LeCun, L. Bottou, G. B. Orr, K.-R. Müller *et al.*, "Neural networks: Tricks of the trade," *Springer Lecture Notes in Computer Sciences*, vol. 1524, no. 5-50, p. 6, 1998.

[41] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[42] M. M. Hammad, "Artificial neural network and deep learning: Fundamentals and theory," *arXiv preprint arXiv:2408.16002*, 2024.

[43] L. R. Medsker, L. Jain *et al.*, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[44] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[45] W. Hu, L. Xiao, and J. Pennington, "Provable benefit of orthogonal initialization in optimizing deep linear networks," *arXiv preprint arXiv:2001.05992*, 2020.

[46] R. Pabari and I. Zhou, "A mean-field theory of training deep neural networks."

[47] A. Miranskyy, A. Sorrenti, and V. Thakar, "On using quasirandom sequences in machine learning for model weight initialization," *arXiv preprint arXiv:2408.02654*, 2024.

[48] K. Grzegorczyk, M. Kurdziel, and P. I. Wójcik, "Effects of sparse initialization in deep belief networks," *Computer Science*, vol. 16, pp. 313–327, 2015.

[49] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[50] Y. Wang, H. Liu, Z. Yi, B. Qian, and M. Wang, "Coarse-to-fine lightweight meta-embedding for id-based recommendation," *arXiv preprint arXiv:2501.11870*, 2025.

[51] D. Aguirre, "A novel set of weight initialization techniques for deep learning architectures," Ph.D. dissertation, The University of Texas at El Paso, 2019.

[52] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.

[54] X. H. Cao, I. Stojkovic, and Z. Obradovic, "A robust data scaling algorithm to improve classification accuracies in biomedical data," *BMC bioinformatics*, vol. 17, pp. 1–10, 2016.

[55] N. Saunshi, O. Plevrakis, S. Arora, M. Khodak, and H. Khandeparkar, "A theoretical analysis of contrastive unsupervised representation learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 5628–5637.

[56] L. K. Saul and S. T. Roweis, "Think globally, fit locally: unsupervised learning of low dimensional manifolds," *Journal of machine learning research*, vol. 4, no. Jun, pp. 119–155, 2003.

[57] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[58] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[59] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 132–149.

[60] A. Coates and A. Y. Ng, "Learning feature representations with k-means," in *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 561–580.

[61] M. Alberti, M. Seuret, V. Pondenkandath, R. Ingold, and M. Liwicki, "Historical document image segmentation with lda-initialized deep neural networks," in *Proceedings of the 4th international workshop on historical document imaging and processing*, 2017, pp. 95–100.

[62] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[63] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige, "On pre-trained image features and synthetic images for deep learning," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[64] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[65] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[66] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[67] Z.-Y. Dou, K. Yu, and A. Anastasopoulos, "Investigating meta-learning algorithms for low-resource natural language understanding tasks," *arXiv preprint arXiv:1908.10423*, 2019.

[68] A. Antoniou and A. J. Storkey, "Learning to learn by self-critique," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[69] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," *arXiv preprint arXiv:1803.11347*, 2018.

[70] C. Desai and C. Desai, "Impact of weight initialization techniques on neural network efficiency and performance: A case study with mnist dataset," *International Journal Of Engineering And Computer Science*, vol. 13, no. 04, 2024.

[71] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[72] B. Hanin and D. Rolnick, "How to start training: The effect of initialization and architecture," *Advances in neural information processing systems*, vol. 31, 2018.

[73] L. A. Curtiss, P. C. Redfern, and K. Raghavachari, "Gaussian-4 theory," *The Journal of chemical physics*, vol. 126, no. 8, 2007.

[74] H. Lee, Y. Kim, S. Y. Yang, and H. Choi, "Improved weight initialization for deep and narrow feedforward neural network," *Neural Networks*, vol. 176, p. 106362, 2024.

[75] H.-D. Cao and D. Zhou, "On complete gradient shrinking ricci solitons," *Journal of Differential Geometry*, vol. 85, no. 2, pp. 175–186, 2010.

[76] M. Mehdipour Ghazi, M. Nielsen, A. Pai, M. Modat, M. J. Cardoso, S. Ourselin, and L. Sørensen, "On the initialization of long short-term memory networks," in *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part I 26*. Springer, 2019, pp. 275–286.

[77] M. Skorski, A. Temperoni, and M. Theobald, "Revisiting weight initialization of deep neural networks," in *Asian conference on machine learning*. PMLR, 2021, pp. 1192–1207.

[78] N. Murru and R. Rossini, "A bayesian approach for initialization of weights in backpropagation neural net with application to character recognition," *Neurocomputing*, vol. 193, pp. 92–105, 2016.

[79] S. Koturwar and S. Merchant, "Weight initialization of deep neural networks (dnns) using data statistics," *arXiv preprint arXiv:1710.10570*, 2017.

[80] E. M. Grais and H. Erdogan, "Initialization of nonnegative matrix factorization dictionaries for single channel source separation," in *2013 21st Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2013, pp. 1–4.

[81] W. Wang and X. Liu, "The selection of input weights of extreme learning machine: a sample structure preserving point of view," *Neurocomputing*, vol. 261, pp. 28–36, 2017.

[82] W. Sun, F. Su, and L. Wang, "Improving deep neural networks with multi-layer maxout networks and a novel initialization method," *Neurocomputing*, vol. 278, pp. 34–40, 2018.

[83] Y. N. Dauphin and S. Schoenholz, "Metainit: Initializing learning by learning to initialize," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[84] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding batch normalization," *Advances in neural information processing systems*, vol. 31, 2018.

[85] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[86] J. C. Lagarias, "The 3x+1 problem and its generalizations," *The American Mathematical Monthly*, vol. 92, no. 1, pp. 3–23, 1985.

[87] M. Chamberland, "A continuous extension of the 3x+1 problem to the real line," *Dynamics of Continuous, Discrete and Impulsive Systems Series B: Applications & Algorithms*, vol. 9, no. 1, pp. 3–26, 2003.

[88] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," *Neural Networks: Tricks of the Trade*, vol. 7700, pp. 9–50, 2012.

[89] D. Mishkin and J. Matas, "All you need is a good init," in *International Conference on Learning Representations (ICLR)*, 2016, arXiv:1511.06422.

[90] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[91] D. Harrison Jr and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," *Journal of environmental economics and management*, vol. 5, no. 1, pp. 81–102, 1978.