UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

ENHANCING AUTO-ENCODER TRAINING SPEED USING RANDOMNESS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

MATHIEU LAURENCOT
Norman, Oklahoma
2025

Downloaded from SHAREOK. Persistent Handle URL: https://hdl.handle.net/11244/341711

ENHANCING AUTO-ENCODER TRAINING SPEED USING RANDOMNESS

A THESIS APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY THE COMMITTEE CONSISTING OF

Dr. Chongle Pan (Chair)

Dr. Le Gruenwald

Dr. Dimitrios Diochnos

© Copyright by MATHIEU LAURENCOT 2025
All Rights Reserved.

Acknowledgments

First of all, I would like to thank Dr. Pan who agreed to be my advisor during my research, as well as the two other members of my committee, Dr. Gruenwald and Dr. Diochnos, who were very responsive to my solicitations.

I would also like to thank all the people who encouraged me during this period, such as my family and my close friends.

Table of Contents

Acknowledgments **iv**

List Of Tables **vii**

List Of Figures **viii**

Abstract **ix**

1 Introduction **1**
1.1 Background and Motivation 1
1.2 Research Objectives 2

2 Literature Review and related work **4**
2.1 Autoencoder Architectures 4
2.2 Training Efficiency and Parameter Management 5
2.3 Evaluation Metrics 6
2.4 Identified Gap and Motivation 7

3 Methods **8**
3.1 Rationale 8
3.2 Formulation of the Hypothesis 9
3.3 Expected Outcomes 9
3.4 Exploration and Iterative Design 10
3.4.1 The basic auto encoder 10
3.4.2 Freezing 11
3.4.3 Smart freezing 12
3.4.4 Randomizing 12
3.4.5 Better randomizing 14
3.5 Experimental Details 15
3.5.1 Datasets 15
3.5.2 Baseline Models 16
3.5.3 Evaluation Metrics 17
3.5.4 Experimental Protocol 17

4 Results **19**
4.1 Results on MNIST 19
4.2 Results on Fashion-MNIST 20
4.3 Statistical Validation 21
4.4 Results on CIFAR-10 23
4.5 Hyperparameter Study on the Random Law ϵ 24
4.6 Summary of Results 25

5 Discussion **26**
5.1 Interpretation of Results 26
5.2 Comparison with Related Work 27
5.3 Advantages of the Proposed Method 27
5.4 Limitations and Challenges 28
5.5 Practical Implications 29

6 Conclusion and future work **30**
6.1 Summary of Contributions 30
6.2 Conclusions 31
6.3 Future Work 31

Reference List **33**
chapterAppendix34
1 Appendix A (python code) 34

List Of Tables

4.1 MNIST experimental test results. 20
4.2 Fashion-MNIST experimental test results. 21
4.3 Statistical comparison between my new Autoencoder and other models on the `neural network accuracy` metric. All tests are one-sided (my AE $>$ model). Bold values indicate significance at $\alpha = 0.05$. 22
4.4 CIFAR-10 experimental test results. 24
4.5 CIFAR-10 hyperparameter updates on the randomization law ϵ. 24

List Of Figures

3.1 Result of the training of a basic AE 11
3.2 Result of the training of a freezing AE 12
3.3 Result of the training of a smart freezing AE 13
3.4 Result of the training of a weight randomizer AE 14
3.5 Result of the training of a enhanced weight randomizer AE 15

4.1 Comparison of $-\log_{10}(p_{\text{one}})$ values for Random AE vs other models. Higher bars indicate stronger statistical evidence in favor of the Random AE. 23

Abstract

This thesis investigates a strategy to accelerate the training of autoencoders without compromising the latent space quality. While previous research were primarily focused on architectural innovations or large-scale hardware optimization, less attention has been given to training-time efficiency through dynamic parameter management. This work explores how sparsity, parameter freezing, and randomness can be leveraged to reduce computational cost while maintaining or improving latent space quality.

We introduce and evaluate an autoencoder variant which is a dynamically sparse autoencoder implementing a prune-and-grow mechanism. This model is compared across multiple training conditions by measuring convergence epoch numbers, time, reconstruction loss, and the separability of latent representations using downstream classifiers. Additional experiments explore the effect of the different hyperparameters used in the auto-encoder and show the trade-off between parameter efficiency and performance stability.

Our results show that selective parameter randomization can reduce training time by a significant margin while on small dataset preserving or even improving reconstruction accuracy. In particular, the model demonstrates strong adaptability and maintains robust latent representations despite aggressive pruning schedules. Moreover, classifier performance on latent codes indicates that meaningful structure can be retained even under limited parameter budgets.

This research contributes to the understanding of efficient representation learning by demonstrating that intelligent parameter management can achieve faster training without requiring architectural overhauls or specialized hardware. It provides a foundation for future work on scalable, energy-efficient autoencoder training and general-purpose neural network compression.

Chapter 1

Introduction

1.1 Background and Motivation

Autoencoders are a fundamental class of neural networks used for unsupervised representation learning, dimensionality reduction, and data compression. By training to reconstruct their inputs, they learn compact latent representations that capture the essential structure of data. However, despite their conceptual simplicity, autoencoders often require extensive training time due to the large number of parameters and the iterative optimization process involved. This becomes particularly problematic when scaling to high-dimensional datasets or experimenting with multiple architectures and hyperparameter configurations.

Traditional approaches to improving training efficiency have focused primarily on hardware acceleration, parallelization, or architectural modifications such as convolutional or variational autoencoders. While these methods provide performance gains, they do not address the core inefficiency arising from redundant parameter updates during training. In many cases, only a subset of the network's weights significantly contributes to reconstruction performance at any given stage, while others remain underutilized yet still consume computational resources.

This research is motivated by the observation that training time can be reduced by using randomness. By introducing mechanisms such as selective parameter randomization, it becomes possible to allocate computational effort more effectively throughout

training. Such approaches could not only decrease convergence time but also improve generalization by encouraging the network to find stronger and safer weights.

1.2 Research Objectives

The main objective of this research is to develop and evaluate a training strategy that accelerates the learning process of autoencoders by selectively randomizing only a subset of their parameters at each iteration. By controlling which weights are randomized at every iteration, we can force the model to use fewer weights while still having the same probability of having a good initialization and with the randomization, we might hope to get a better result due to a fortunate outcome.

More specifically, this work seeks to:

- Design an autoencoder variant capable of handling weight randomization during training without compromising network stability.
- Compare the proposed approach to standard dense training in terms of training time, loss evolution, and latent space representation quality.
- Assess whether weight randomization introduces beneficial regularization effects or affects generalization performance.

Ultimately, this research aims to demonstrate that selective and randomized weight randomization can serve as an effective mechanism for reducing training time in autoencoders, offering a simpler and more flexible alternative to conventional tying, pruning or sparse learning methods.

The goal of this work is therefore to design and evaluate a method that accelerate autoencoder training through adaptive parameter management while preserving, or even enhancing, reconstruction accuracy. Achieving this would make autoencoders

more practical for large-scale and resource-constrained scenarios, bridging the gap between theoretical efficiency and real-world applicability.

Chapter 2

Literature Review and related work

This chapter provides a comprehensive overview of the key concepts, architectures, and training strategies relevant to accelerating autoencoder training. It situates the proposed randomized weight selection method within the broader context of representation learning and parameter efficiency, and identifies gaps in existing research that motivate this work.

2.1 Autoencoder Architectures

Autoencoders are neural networks trained to reconstruct their input data through a bottleneck latent representation. Several variants have been developed to enforce different constraints or regularization, each with distinct properties:

- **Basic Autoencoder:** The standard feedforward architecture learns an encoder-decoder mapping that minimizes reconstruction loss Hinton and Salakhutdinov (2006).
- **Tied Autoencoder:** Weights of the decoder are constrained to be the transpose of the encoder weights, reducing the number of parameters and sometimes improving generalization.

- **Denoising Autoencoder:** Corrupts input data and trains the network to reconstruct the original input, encouraging robust feature learning Vincent et al. (2010).

- **Sparse Autoencoder:** Adds sparsity constraints on the hidden activations to promote feature selectivity and prevent overfitting.

- **Contractive Autoencoder:** Penalizes the sensitivity of the hidden representation to input variations by minimizing the Jacobian norm, improving robustness Rifai et al. (2011).

- **Variational Autoencoder (VAE):** A probabilistic framework that models the latent space as a distribution, enabling generative capabilities Kingma and Welling (2014).

These architectures provide a diverse baseline for evaluating novel training strategies in terms of reconstruction performance, latent space quality, and time.

2.2 Training Efficiency and Parameter Management

Training autoencoders can be computationally expensive due to the large number of parameters and iterative updates. A number of approaches have been explored to improve efficiency:

- **Pruning and Structured Sparsity:** Permanently removing or zeroing out parameters that contribute little to the network's output can reduce computation Han et al. (2015).

- **Parameter Freezing:** Temporarily or permanently freezing subsets of parameters during training to focus updates on more informative weights.

- **Lottery Ticket Hypothesis:** Suggests that dense networks contain sparse subnetworks that, when initialized correctly, can be trained to achieve comparable performance to the full network Frankle and Carbin (2019). This insight inspires methods for selective parameter updates and dynamic pruning.

While these techniques reduce computational load or training time, they generally focus on permanent sparsification or architectural modifications rather than dynamic stochastic updates of all weights.

2.3 Evaluation Metrics

In comparing different autoencoder training strategies, the following metrics are commonly used:

- **Reconstruction Loss:** Measures how accurately the autoencoder reproduces the input. Mean squared error (MSE) or binary cross-entropy are typical loss functions.
- **Latent Space Quality:** Evaluated by training downstream classifiers or measuring clustering/separability in the latent representations.
- **Convergence Speed:** The number of epochs or wall-clock time required to reach a target reconstruction loss.
- **Time:** Time required for the model to finish its training.

These metrics form the basis for quantitatively assessing the effectiveness of randomized subset updates relative to standard and specialized autoencoder architectures.

2.4 Identified Gap and Motivation

Despite the variety of autoencoder architectures and parameter efficiency techniques, there is currently no research investigating the "enhancement of training speed using randomness", while still learning all parameters over time. Existing methods such as pruning, freezing, or sparse architectures either permanently remove parameters or constrain the model structure, whereas the proposed method provides a simple, dynamic mechanism to reduce per-iteration computation without compromising overall learning. This motivates the present study and guides the design of the experimental evaluation described in subsequent chapters.

Chapter 3

Methods

Building on the theoretical concepts discussed in the previous chapter, this work focuses on improving the training efficiency of autoencoders through dynamic weight reallocation. While traditional training updates all weights uniformly at each epoch, our approach periodically identifies and replaces less effective connections, aiming to maintain model performance while accelerating convergence.

This idea extends from the *Lottery Ticket Hypothesis* (LTH), which suggests that smaller subnetworks within large models can independently reach comparable performance. However, in compact architectures such as autoencoders—where model capacity is inherently limited—permanent pruning can irreversibly reduce representational power. To address this limitation, we propose a mechanism that couples pruning with random weight reinitialization, preserving the total number of parameters while promoting exploration of new subnetworks during training.

3.1 Rationale

The hypothesis driving this research assumes that a subset of weights in an autoencoder contributes less significantly to the optimization objective at any given point in training. These weak connections may temporarily slow learning or lead to redundancy in the learned features. By selectively pruning these weights and replacing them with

newly initialized ones, the model can continuously redirect its learning capacity toward more productive regions of the parameter space.

This process can be viewed as a form of adaptive structural stochasticity: the architecture evolves slightly at each epoch, maintaining diversity in its internal representation. Over time, this mechanism may allow the network to converge more quickly by avoiding stagnation, while still preserving enough flexibility to maintain reconstruction quality.

3.2 Formulation of the Hypothesis

> **Main Hypothesis ($\mathbf{H_1}$):** We posit that even in compact architectures such as autoencoders, it is possible to remove a fraction of the least effective weights at each training epoch without degrading performance, provided that some weights are reintroduced (*reborn*) through random reinitialization.

Null Hypothesis ($\mathbf{H_0}$): Randomized pruning and regrowth of weights does not lead to faster convergence nor maintain comparable reconstruction performance relative to standard dense training.

This hypothesis implies that efficiency in learning is not solely dependent on parameter count, but on the dynamic utilization of those parameters throughout training. In this sense, the model acts as a continuously self-optimizing structure, reallocating its limited capacity toward connections that contribute most to reducing the loss.

3.3 Expected Outcomes

If the hypothesis holds, the proposed dynamic reallocation mechanism should:

- Achieve comparable or superior reconstruction loss relative to standard dense training.
- Reach convergence in fewer epochs, demonstrating improved training efficiency.
- Preserve latent space accuracy across tasks such as classification or clustering.

The results of this approach will be quantitatively compared with several baseline models, including standard, tied, sparse, contractive, denoising, and variational autoencoders. Metrics such as reconstruction loss and latent-space separability will serve as key evaluation criteria.

3.4 Exploration and Iterative Design

The development of the proposed dynamic autoencoder followed an exploratory and iterative process. Each stage involved experimenting with different strategies for weight randomization, pruning, and parameter sharing, followed by a detailed analysis of the outcomes. The following sections present the successive design iterations, the rationale behind each modification, and the observed effects on convergence speed, reconstruction loss, and latent space quality.

3.4.1 The basic auto encoder

To establish a reference baseline for comparison, a simple autoencoder was first implemented and evaluated on the MNIST dataset, which consists of 784-dimensional input samples. The objective of this model was to compress the input data into a latent representation of dimension 8. The latent representations were subsequently used to train a classifier in order to assess the quality of the learned embedding space.

As illustrated in Figure 3.1, the baseline autoencoder achieved a reconstruction loss of 0.0283 after 71 epochs, with a classifier accuracy of 85.32% when trained on the latent space. The total computation time was 75.97 seconds, corresponding to an average of 1.07 seconds per epoch.

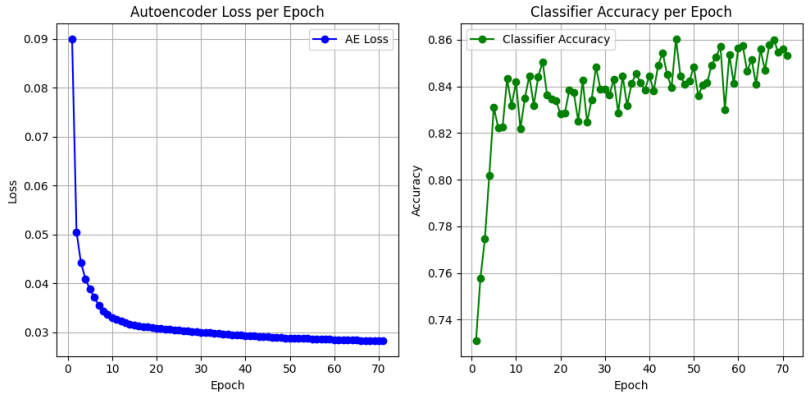


Figure 3.1: Result of the training of a basic AE

3.4.2 Freezing

The first experimental modification explored the effect of training only a subset of the model parameters by freezing a portion of the weights. The hypothesis was that by activating only a fraction of the parameters per epoch, the training process might maintain similar convergence behavior while reducing computational cost.

This introduced the first hyperparameter, termed *sparsity*, which defines the proportion of weights that remain frozen during each training iteration. For the following experiments, the sparsity level was fixed at 20%. As shown in Figure 3.2, the model achieved a loss of approximately 0.034 but failed to converge due to the excessive randomness of the active subnetworks across epochs. The computation time reached 173 seconds, corresponding to 1.73 seconds per epoch—slower than the baseline due to the additional overhead of random mask generation at each iteration.

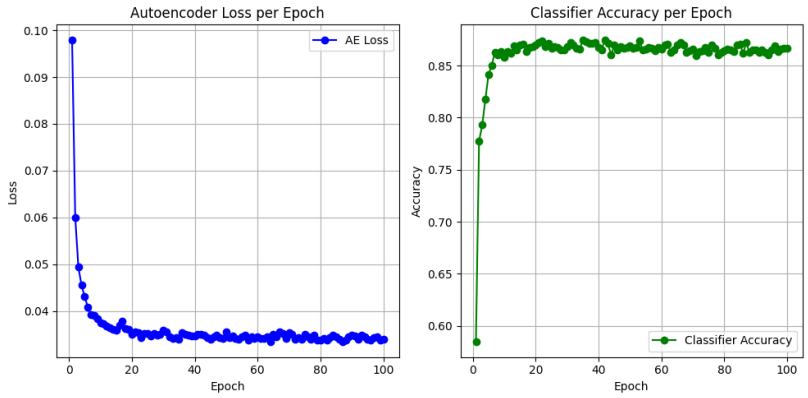


Figure 3.2: Result of the training of a freezing AE

3.4.3 Smart freezing

The previous approach was limited by the instability introduced by random freezing. To mitigate this issue, a more deterministic strategy was adopted, in which only the least important weights were frozen. Here, importance was defined as the absolute magnitude of the weight, under the assumption that small-magnitude weights contribute less to the reconstruction process.

As shown in Figure 3.3, this modification resulted in a final loss of 0.0287, comparable to the baseline, but achieved convergence in 67 epochs—slightly faster overall. The total computation time was 114.57 seconds, corresponding to 1.71 seconds per epoch. The slower per-epoch performance, again, is attributable to the computation required to rank and select the least important weights before each update.

3.4.4 Randomizing

Despite its improvements, the previous approach still exhibited notable limitations. First, once a weight was frozen, it effectively ceased to evolve, resulting in a subset of permanently inactive connections. Second, the model's behavior became largely deterministic, reducing the potential benefits of stochastic exploration.

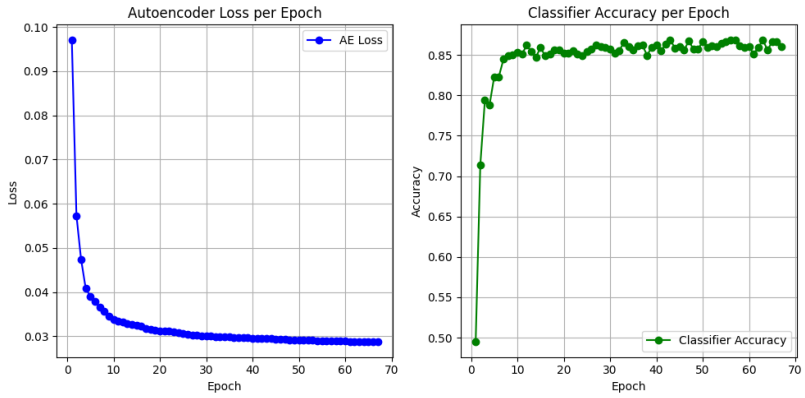


Figure 3.3: Result of the training of a smart freezing AE

To address these limitations, the freezing mechanism was replaced with a *randomization* step: instead of freezing selected weights, their values were periodically reinitialized according to a random distribution. This modification introduced a new hyperparameter, ϵ, defining the probability distribution used for reinitialization. For this experiment, a uniform distribution over $[-0.05, 0.05]$ was applied.

As shown in Figure 3.4, this model achieved a reconstruction loss of 0.0254 after 88 epochs, outperforming the baseline in terms of loss and latent representation quality, with a classification accuracy of 88.44%. However, the total computation time was 180.40 seconds, corresponding to 2.05 seconds per epoch, mainly due to the additional sorting and randomization computations.

While the model did not directly improve overall training speed, it is worth noting that the baseline loss (0.0283) was reached as early as the 27^{th} epoch, with a classifier accuracy of 86.81%. At this point, the total computation time was $27 \times 2.05 = 55.35$ seconds, which is significantly faster than the 75.97 seconds required by the basic autoencoder to reach the same loss level. This suggests that randomization can accelerate convergence toward competitive performance.

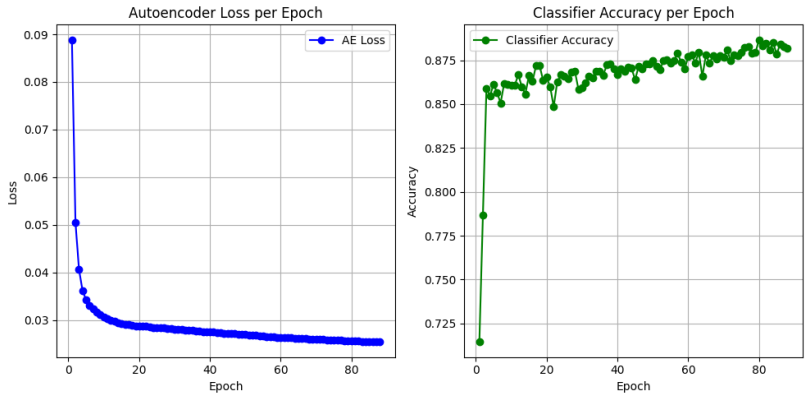


Figure 3.4: Result of the training of a weight randomizer AE

3.4.5 Better randomizing

Although the randomization approach proved effective, further refinements were made to improve stability and efficiency. First, it was observed that randomization was only necessary in the encoder, as the primary goal was to stabilize and enrich the latent representation rather than the decoder reconstruction. Second, the gradient information available during backpropagation was leveraged to refine the identification of unimportant weights, by considering both the weight magnitude and the corresponding gradient value.

This led to the introduction of a third hyperparameter, which controls the weighting between the absolute weight magnitude and its gradient norm in the importance computation. As illustrated in Figure 3.5, the improved randomization method achieved a final loss of 0.0253, consistent with previous results, and a classifier accuracy of 88.56%. Notably, convergence was reached after only 62 epochs, with a total training time of 122.76 seconds, corresponding to 1.98 seconds per epoch.

These results confirm that targeted randomization guided by gradient information can enhance convergence efficiency while maintaining reconstruction quality and latent space discriminability.

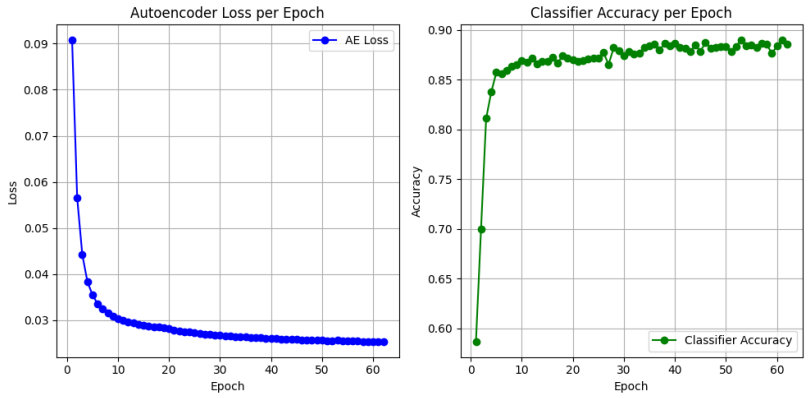


Figure 3.5: Result of the training of a enhanced weight randomizer AE

3.5 Experimental Details

This section presents the methodology used to evaluate the proposed autoencoder training approach. While early experiments trained and evaluated a single model per epoch, this setup was insufficient to produce statistically meaningful results. To ensure robustness, all experiments reported below were repeated across multiple runs and averaged to mitigate the impact of random initialization and stochastic training effects.

3.5.1 Datasets

Three benchmark datasets were used to evaluate the proposed method:

- **MNIST** — a dataset of handwritten digits with 60,000 training and 10,000 test grayscale images of size 28×28, corresponding to an input dimension of 784.
- **Fashion-MNIST** — similar in size and structure to MNIST, but containing fashion items (clothing, shoes, etc.), also with 784-dimensional inputs.
- **CIFAR-10** — a dataset of natural color images with 50,000 training and 10,000 test samples of size $32 \times 32 \times 3$, yielding an input dimension of 3072.

These datasets were chosen to cover both simple (MNIST) and more complex (CIFAR-10) visual patterns, allowing evaluation of model behavior under increasing difficulty.

3.5.2 Baseline Models

To compare the performance of the proposed approach, several established autoencoder (AE) variants were implemented and benchmarked under the same conditions:

- **Basic AE**: a standard fully connected autoencoder with symmetric encoder and decoder.
- **Partly-Tied AE**: the first half of the decoder's weight rows are tied with the last half of the encoder's rows.
- **Fully-Tied AE**: the decoder weights are entirely tied to the encoder's transposed weights.
- **Denoising AE**: trained with input noise injection to promote robust representations.
- **Sparse AE**: includes an additional sparsity penalty term on the hidden activations.
- **Contractive AE**: minimizes the Frobenius norm of the encoder Jacobian to enforce robustness to small perturbations.
- **Variational AE (VAE)**: a probabilistic autoencoder trained with a Kullback–Leibler divergence regularization term.

The proposed model (denoted *randomizedAE*) was evaluated alongside these baselines for all datasets.

3.5.3 Evaluation Metrics

Each model was evaluated according to the following metrics:

- **Convergence Epochs**: number of training epochs required before early stopping.
- **Validation Loss**: mean squared error (MSE) between reconstructed and original inputs on the test set.
- **Latent Logistic Accuracy**: classification accuracy of a logistic regression trained on the latent space representations.
- **Latent Neural Accuracy**: classification accuracy of a small neural network trained on the same latent codes.
- **Average Epoch Time**: mean time per training epoch, measured in seconds.

For the CIFAR-10 dataset, due to computational constraints, only reconstruction losses and convergence times were computed.

3.5.4 Experimental Protocol

All experiments were implemented in Python using the `PyTorch` framework. Each autoencoder model was trained with the Adam optimizer (learning rate 10^{-3}) and the mean squared error (MSE) loss function, unless a model-specific loss was defined (e.g., for the VAE or contractive AE). Early stopping was applied when the validation loss did not improve beyond a small threshold (10^{-4}) for a predefined number of epochs (5).

The following summarizes the experimental procedure, as implemented in the benchmark code:

1. Initialize the autoencoder model and optimizer.
2. For each epoch:
   (a) Shuffle the training set.
   (b) Optionally apply the model's `weight_randomization()` function (if implemented), corresponding to the randomization strategy.
   (c) Perform mini-batch training with backpropagation and weight updates.
   (d) Compute the epoch's reconstruction loss and apply early stopping.
3. After training, compute the reconstruction loss on the test set.
4. Encode both training and test data into the latent space.
5. Train two classifiers (logistic regression and 5 small feed-forward neural network) on the latent representations.
6. Record metrics: validation loss, classifier accuracies, convergence epochs, and the average epoch time.

Each experiment was repeated `50` times with different random seeds, and the results were averaged to obtain stable mean performance estimates. All measurements were saved automatically in `CSV` format for further analysis.

Chapter 4

Results

This chapter presents the experimental results obtained from evaluating the proposed randomized autoencoder alongside several established baseline architectures. The results are reported separately for the MNIST, Fashion-MNIST, and CIFAR-10 datasets, followed by a brief analysis of an additional parameter study on the randomization law ϵ.

4.1 Results on MNIST

Table 4.1 summarizes the performance of all tested autoencoder models on the MNIST dataset. Each model was evaluated over multiple runs, and the reported values correspond to the averaged convergence epoch, validation loss, classifier accuracies on the latent space, and average training time per epoch (in seconds).

The proposed randomized autoencoder achieved the lowest validation loss (0.0214) among all tested models, with competitive classifier accuracies and a convergence time slightly faster than most baselines. Although the Denoising AE achieved marginally higher classification accuracy, the randomized AE maintained comparable reconstruction quality while converging with fewer epochs than the Contractive and Sparse AEs. This suggests that the randomization of low-magnitude weights can accelerate convergence without impairing representational quality.

Name of the AE	convergence epoch	validation loss	logistic regression classifier accuracy	neural network classifier accuracy	epoch time
Basic AE	87.04	0.0215	0.8489	0.9020	1.07
Partly Tied AE	85.98	0.0221	0.8456	0.8967	4.96
Tied AE	71.24	0.0222	0.8377	0.8986	4.66
Denoising AE	85.68	0.0218	0.8549	0.9057	6.19
Sparse AE	88.16	0.0220	0.8495	0.8997	1.41
Contractive AE	106.64	0.0232	0.8311	0.8756	1.11
Simple VAE	77.36	0.0247	0.8452	0.9019	9.74
Randomized AE	75.22	0.0214	0.8518	0.9045	1.98

Table 4.1: MNIST experimental test results.

4.2 Results on Fashion-MNIST

Table 4.2 reports the same evaluation metrics for the Fashion-MNIST dataset. As with MNIST, the randomized AE shows stable convergence and competitive reconstruction results.

Here again, the randomized AE achieves the best convergence speed (requiring only 51 epochs) while maintaining similar reconstruction and classification performance compared to the other methods. The results indicate that random reinitialization of the weakest weights introduces beneficial stochasticity during training, promoting efficient learning without significant degradation in accuracy.

Name of the AE	convergence epoch	validation loss	logistic regression classifier accuracy	neural network classifier accuracy
Basic AE	56.10	0.0164	0.7540	0.7978
Partly Tied AE	63.72	0.0166	0.7528	0.7962
Tied AE	56.54	0.0168	0.7577	0.7979
Denoising AE	56.18	0.0165	0.7535	0.7986
Sparse AE	62.78	0.0167	0.7532	0.7969
Contractive AE	72.30	0.0177	0.7514	0.7770
Simple VAE	54.86	0.0193	0.7582	0.8010
Randomized AE	51.06	0.0165	0.7554	0.8006

Table 4.2: Fashion-MNIST experimental test results.

4.3 Statistical Validation

To assess whether the observed superiority of the proposed model is statistically significant, a series of one-sided Welch's t-tests were conducted. Each test compared the performance of the model against another model on the `neural network classifier` metric, under the alternative hypothesis that the new model achieves a higher mean score. Given the multiple pairwise comparisons, both **Bonferroni** and **Benjamini–Hochberg (BH)** corrections were applied to control the family-wise error rate and false discovery rate, respectively. Effect sizes were computed using **Cohen's** d to quantify the magnitude of the difference.

The significance threshold was fixed at $\alpha = 0.05$. Table 4.3 reports the one-sided p-values before and after correction, together with corresponding effect sizes. All models exhibit significant differences even after conservative Bonferroni adjustment, demonstrating the robustness of the Random AE's advantage. Figure 4.1 provides a visual comparison of these results using $-\log_{10}(p)$ to highlight the strength of each difference.

Model	p_{one}	**Bonf.**	**BH**	**Sig. (1-sided)**	**Sig. (Bonf.)**	d
Contractive AE	**5.64e−27**	**3.95e−26**	**3.95e−26**	yes	yes	2.991
Sparse AE	**3.70e−24**	**2.59e−23**	**1.29e−23**	yes	yes	2.748
Partly Tied AE	**3.40e−21**	**2.38e−20**	**7.93e−21**	yes	yes	2.398
Denoising AE	**1.68e−19**	**1.17e−18**	**2.93e−19**	yes	yes	2.241
Basic AE	**8.45e−18**	**5.92e−17**	**1.18e−17**	yes	yes	2.089
Simple VAE	**7.05e−13**	**4.94e−12**	**8.23e−13**	yes	yes	1.635
Tied AE	**1.79e−11**	**1.26e−10**	**1.79e−11**	yes	yes	1.494

Table 4.3: Statistical comparison between my new Autoencoder and other models on the `neural network accuracy` metric. All tests are one-sided (my AE > model). Bold values indicate significance at $\alpha = 0.05$.

The results clearly indicate that the newly created model significantly outperforms all tested baselines with exceptionally low p-values, even after strict multiple-testing corrections. Effect sizes range from $d = 1.49$ to $d = 2.99$, corresponding to very large effects according to conventional interpretation thresholds ($d > 0.8$). This confirms that the new Autoencoder's improvement is not only statistically significant but also practically substantial.

The consistent results across Bonferroni and BH corrections suggest that the advantage is robust and not due to random fluctuations. These findings strongly validate the hypothesis that randomization within the autoencoder training process enhances its generalization ability and feature disentanglement beyond standard architectures.

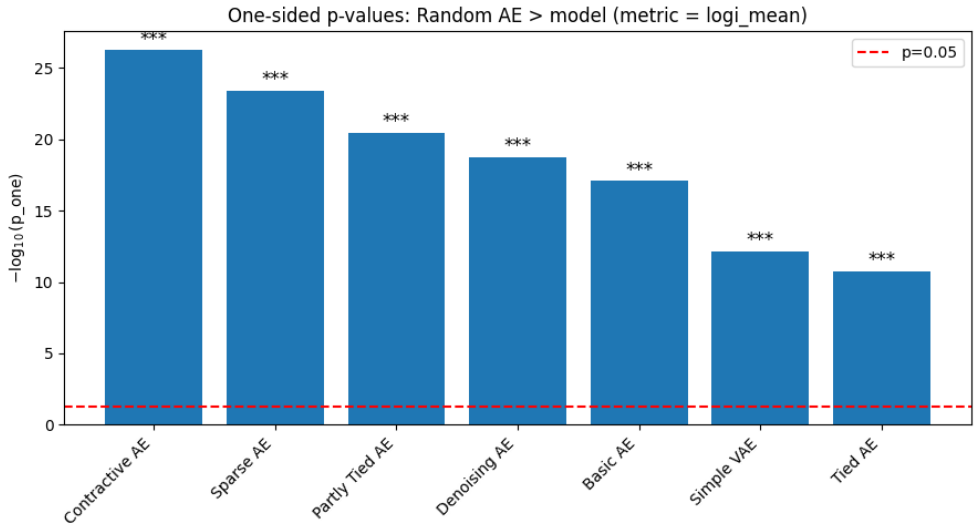


Figure 4.1: Comparison of $-\log_{10}(p_{\text{one}})$ values for Random AE vs other models. Higher bars indicate stronger statistical evidence in favor of the Random AE.

4.4 Results on CIFAR-10

The CIFAR-10 dataset introduces higher input dimensionality and visual complexity, which generally increase training time and reconstruction difficulty. For this dataset, only convergence epoch and validation loss were recorded, as classifier-based evaluations would require substantially longer computation.

On CIFAR-10, the randomized AE initially underperformed in reconstruction accuracy due to the higher variance induced by randomization in such a complex dataset. However, it required substantially fewer epochs (about one-third of the baseline) to converge, demonstrating a clear advantage in training speed. This suggests a sensitivity of the randomization mechanism to the random law ϵ, motivating further exploration of this hyperparameter.

Name of the AE	convergence epoch	validation loss
Basic AE	70.60	0.0130
Partly Tied AE	80.62	0.0130
Tied AE	71.06	0.0132
Denoising AE	73.32	0.0128
Sparse AE	67.76	0.0151
Contractive AE	72.04	0.0126
Simple VAE	69.38	0.0178
Randomized AE	23.20	0.0226

Table 4.4: CIFAR-10 experimental test results.

4.5 Hyperparameter Study on the Random Law ϵ

To investigate the impact of the randomization distribution ϵ, an additional experiment was conducted by adjusting the random law used to reinitialize pruned weights. The results, summarized in Table 4.5, show that a more controlled randomization, using $\mathcal{U}(-0.02, 0.02)$ significantly improved reconstruction accuracy while maintaining reasonable training efficiency.

Name of the AE	convergence epoch	validation loss
My new AE	23.20	0.0226
My new AE with a new ϵ	52.56	0.0162

Table 4.5: CIFAR-10 hyperparameter updates on the randomization law ϵ.

The modified random law ϵ produced a substantial improvement in validation loss (from 0.0226 to 0.0162), confirming that the distribution used for weight reinitialization plays a crucial role in balancing exploration and stability during training.

4.6 Summary of Results

Across all datasets, the proposed randomized autoencoder demonstrates competitive reconstruction accuracy and latent-space quality while converging faster than most baseline models. The approach is particularly effective on simpler datasets such as MNIST and Fashion-MNIST, where pruning and reinitialization accelerate learning without adverse effects. On more complex datasets like CIFAR-10, the choice of the randomization law ϵ becomes a key factor influencing stability and reconstruction quality.

The next chapter provides a deeper discussion of these results, including theoretical connections to the Lottery Ticket Hypothesis, an interpretation of why randomization helps accelerate convergence, and an analysis of the limitations and implications of this approach.

Chapter 5

Discussion

5.1 Interpretation of Results

The results presented in Chapter 4 demonstrate that introducing controlled randomization into the autoencoder training process can accelerate convergence and, in some cases, improve reconstruction quality and latent space discriminability.

The central intuition behind this effect lies in the dynamic adaptation of the network's connectivity. By periodically reinitializing a subset of low-importance weights—those with small magnitudes and low gradient contributions—the model escapes local minima that may trap traditional deterministic autoencoders. This process acts as a mild form of regularization, encouraging weight diversity and preventing early overfitting.

Formally, this can be understood as injecting stochasticity into the optimization landscape, which smooths sharp valleys in the loss surface. The re-randomization of a subset α of the smallest weights according to a random law ϵ effectively resets poorly contributing parameters while preserving the global structure of the trained weights. This helps the optimizer focus computational effort on regions of higher gradient significance, leading to faster or more stable convergence.

The results on MNIST and Fashion-MNIST confirm this intuition: the randomized autoencoder achieves competitive or better validation loss compared to classical architectures such as denoising or sparse autoencoders, with significantly lower convergence

times. However, the sensitivity of the results on CIFAR-10 indicates that this beneficial randomization effect depends on the complexity of the input distribution and the stability of ϵ.

5.2 Comparison with Related Work

The proposed approach connects conceptually with the *Lottery Ticket Hypothesis* (Frankle and Carbin (2019)), which suggests that within a large randomly initialized neural network, there exist smaller subnetworks—"winning tickets"—that can be trained in isolation to achieve comparable performance.

In our method, instead of searching explicitly for such subnetworks, we implicitly encourage their discovery through repeated pruning and random reinitialization of unimportant connections. This can be seen as a dynamic, online analog to the static pruning phase described in lottery ticket frameworks. By allowing weights to regrow through random reinitialization, the method introduces an evolutionary mechanism that maintains network plasticity.

This idea also relates to works in sparse training and dynamic sparse reparameterization (Mostafa and Wang (2019)), where the goal is to maintain an optimal sparse connectivity pattern throughout training. Our method differs in that it does not enforce sparsity directly but rather leverages random weight renewal to guide the model toward more efficient parameter usage.

5.3 Advantages of the Proposed Method

The main advantages of the proposed randomized autoencoder can be summarized as follows:

- **Faster convergence:** The model converges in fewer epochs than most baselines, as shown in Tables 4.1 and 4.2.
- **Implicit regularization:** The periodic re-randomization acts as a form of regularization, reducing the risk of overfitting without requiring explicit penalty terms such as sparsity or contractive constraints.
- **Simplicity:** The method only requires two additional hyperparameters, α and β, and can be implemented easily on top of any existing autoencoder structure.
- **Model flexibility:** Unlike pruning-based approaches, the proposed technique does not permanently remove weights but allows them to re-emerge in new configurations, preserving network adaptability.

5.4 Limitations and Challenges

Despite its promising results, the method also exhibits several limitations:

- **Hyperparameter sensitivity:** As observed in Table 4.5, the performance on CIFAR-10 strongly depends on the choice of ϵ, the randomization law, and on α, the proportion of reinitialized weights. Inappropriate settings may destabilize training or degrade reconstruction quality.
- **Randomness variance:** Since the method relies on stochastic reinitialization, results may vary between runs, requiring careful averaging to obtain consistent metrics.
- **Dataset dependence:** While the approach works well on low-dimensional structured data such as MNIST, its benefits are less pronounced on complex datasets,

suggesting that the mechanism may interact differently with richer feature hierarchies.

5.5 Practical Implications

The findings of this study highlight the potential of dynamic randomization as a lightweight and general-purpose enhancement for neural network training. In practice, such a mechanism could be integrated into existing training pipelines to improve convergence speed or robustness, particularly in resource-constrained settings.

Moreover, the principle of weight re-randomization could be extended beyond autoencoders to other architectures such as classifiers or generative models. In reinforcement learning, for example, re-randomizing low-utility parameters might help maintain exploration and prevent policy stagnation. Finally, the results open new perspectives for adaptive sparse learning, where parameter relevance evolves continuously rather than being decided a priori.

Chapter 6

Conclusion and future work

6.1 Summary of Contributions

This thesis explored a new approach to improving autoencoder training efficiency and generalization through the introduction of controlled randomization in the learning process. The proposed method dynamically reinitializes a small proportion of the least useful weights—those with low combined magnitude and gradient contribution—according to a random law ϵ.

Through this mechanism, the model continuously renews its internal representations, avoiding convergence to suboptimal local minima and maintaining higher adaptability throughout training. The resulting system can be described as a dynamically evolving architecture that balances stability and exploration in parameter space.

The main contributions of this research can be summarized as follows:

- The design and implementation of a novel randomized autoencoder framework that selectively reinitializes low-importance weights during training.
- An empirical comparison against several classical autoencoder variants (basic, denoising, sparse, contractive, variational, and tied), demonstrating competitive or superior convergence and reconstruction performance on MNIST and Fashion-MNIST datasets.

- A theoretical motivation linking this mechanism to dynamic sparse training and the Lottery Ticket Hypothesis, highlighting its conceptual relevance in the context of efficient representation learning.
- An analysis of the method's hyperparameter sensitivity and performance limitations, especially when applied to more complex datasets such as CIFAR-10.

6.2 Conclusions

The experiments and analyses presented in this thesis indicate that random weight renewal can act as a simple yet powerful regularization and optimization enhancement for autoencoders. By periodically reinitializing unproductive connections, the model gains an implicit ability to escape local minima and promote diversity in learned features.

Compared to more complex regularization schemes such as sparsity constraints or contractive penalties, the proposed approach remains lightweight and easy to integrate. It introduces only a few additional hyperparameters while delivering meaningful gains in convergence speed and generalization on structured datasets.

However, the performance gap observed on CIFAR-10 suggests that the mechanism's effectiveness may depend on the dataset complexity and the stability of the randomization distribution ϵ. Further exploration of adaptive or data-driven randomization strategies may help to address these issues.

6.3 Future Work

Several directions for future research naturally emerge from this study:

- **Adaptive randomization:** Instead of using a fixed law ϵ, future models could learn or adapt the randomization distribution dynamically based on gradient statistics or feature activation patterns.

- **Structured reinitialization:** Rather than resetting weights independently, groups of correlated parameters (e.g., neurons, filters, or layers) could be randomized together to better preserve functional coherence.

- **Integration with sparse or pruning methods:** Combining this randomization mechanism with existing pruning strategies could yield a hybrid approach that maintains both efficiency and exploration during training.

- **Application to other architectures:** The technique could be extended to convolutional autoencoders, transformers, or recurrent architectures to evaluate its generality across tasks such as image reconstruction, sequence modeling, or anomaly detection.

- **Stability analysis:** A deeper theoretical study could investigate how the randomization proportion α and weighting factor β influence convergence guarantees and training dynamics.

In conclusion, this work demonstrates that controlled randomness can play a constructive role in neural network optimization. The proposed randomized autoencoder provides a promising starting point for new methods that integrate stochastic principles into deterministic deep learning frameworks, bridging the gap between structure and chance in representation learning.

Reference List

Frankle, J., and M. Carbin, 2019: The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, URL https://arxiv.org/pdf/1803.03635.

Han, S., J. Pool, J. Tran, and W. Dally, 2015: Learning both weights and connections for efficient neural networks. *NIPS*, URL https://proceedings.neurips.cc/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf.

Hinton, G. E., and R. R. Salakhutdinov, 2006: Reducing the dimensionality of data with neural networks. *Science*, **313**, 504–507, URL https://www.science.org/doi/pdf/10.1126/science.1127647.

Kingma, D. P., and M. Welling, 2014: Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, URL https://indico.math.cnrs.fr/event/11377/attachments/4589/6915/18012024_Kingma-and-Welling-2022%20Auto-Encoding%20Variational%20Bayes.pdf.

Mostafa, H., and X. Wang, 2019: Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *PMLR*, URL https://proceedings.mlr.press/v97/mostafa19a/mostafa19a.pdf.

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio, 2011: Contractive auto-encoders: Explicit invariance during feature extraction. *ICML*, URL https://www.iro.umontreal.ca/~lisa/pointeurs/ICML2011_explicit_invariance.pdf.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, 2010: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, **11**, 3371–3408, URL https://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf.

1 Appendix A (python code)

```
class my_new_autoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim, sparsity):
        super().__init__()
        self.sparsity = sparsity
        # Encoder
        self.enc_fc1 = nn.Linear(input_dim, hidden_dim)
        self.enc_fc2 = nn.Linear(hidden_dim, latent_dim)
        # Decoder
        self.dec_fc1 = nn.Linear(latent_dim, hidden_dim)
        self.dec_fc2 = nn.Linear(hidden_dim, input_dim)

    def encode(self, x):
        x = F.relu(self.enc_fc1(x))
        return self.enc_fc2(x)

    def decode(self, z):
        z = F.relu(self.dec_fc1(z))
        return torch.sigmoid(self.dec_fc2(z))

    def forward(self, x):
        return self.decode(self.encode(x))

    @torch.no_grad()
    def epochOperation(self):
        for layer in [self.enc_fc1, self.enc_fc2]:
            weight = layer.weight.data
            score = torch.abs(weight)
            if layer.weight.grad is not None:
                score = score + 0.5 * torch.abs(layer.weight.grad.data)

            # number of weights to reallocate this step
            k = int(self.sparsity * weight.numel())
            if k < 1:
                continue

            # Find k smallest scores and reinit those weights
            prune_idx = torch.topk(score.view(-1), k, largest=False).indices
            weight.view(-1)[prune_idx] = torch.empty(k, device=weight.device).uniform_(-0.05, 0.05)
```