NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

An Introduction to Randomizing Deep Polynomial Neural Networks

Spyridon - Ioannis - Chalkias

Supervisor: Yannis Panagakis, Associate Professor

ATHENS

OCTOBER 2023

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μια Εισαγωγή στην Τυχαιοποίηση Βαθέων Πολυωνυμικών Νευρωνικών Δικτύων

Σπυρίδων - Ιωάννης - Χαλκιάς

Επιβλέπων: Ιωάννης Παναγάκης, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2023

BSc THESIS

An Introduction to Randomizing Deep Polynomial Neural Networks

Spyridon - Ioannis - Chalkias
S.N.: 1115201800209

SUPERVISOR: Yannis Panagakis, Associate Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μια Εισαγωγή στην Τυχαιοποίηση Βαθέων Πολυωνυμικών Νευρωνικών Δικτύων

Σπυρίδων - Ιωάννης - Χαλκιάς
Α.Μ: 1115201800209

ΕΠΙΒΛΕΠΩΝ: Ιωάννης Παναγάκης, Αναπληρωτής Καθηγητής

ABSTRACT

Deep Polynomial Neural Networks (DPNNs) represent a novel family of function approximators that have demonstrated promising results in a variety of challenging tasks, including image generation, face verification, and 3D mesh representation learning. Despite their recent introduction to the field of Deep Learning, there are still many aspects of DPNNs that warrant further exploration in order to solidify their standing among other Deep Learning architectures. One such aspect is the investigation of regularization techniques. Regularization plays a crucial role in improving the generalization ability of a model, reducing overfitting, and aiding in the reduction of network redundancy through the introduction of stochasticity. However, the application of regularization techniques to DPNNs remains largely unexplored. Strongly motivated by this gap in the current understanding, this thesis aims to provide an introductory exploration into how regularization can be applied to DPNNs. Two regularization methods, explicit rank reduction and dropout regularization, are proposed and evaluated. Explicit rank reduction involves reducing the rank of parameter tensors in DPNNs, while dropout regularization involves randomly omitting elements of parameter tensors during training. Experimental evaluation on the MNIST and CIFAR10 datasets under varying levels of noise addition and adversarial attacks reveals that DPNNs are impressively resilient to adversarial perturbations. Furthermore, they perform better and overfit less when regularized with explicit rank reduction, while dropout regularization typically leads to a degradation in performance. This endeavor serves as a comprehensive foundation, offering valuable insights and findings that will significantly contribute to and stimulate future research in this domain.

SUBJECT AREA: Computer Vision

KEYWORDS: Deep Learning, Deep Polynomial Neural Networks, Regularization, Explicit Rank Reduction, Dropout, Tensors

ΠΕΡΙΛΗΨΗ

Τα Βαθιά Πολυωνυμικά Νευρωνικά Δίκτυα (ΒΠΝΔ) αντιπροσωπεύουν μια νέα οικογένεια συναρτήσεων που έχουν επιδείξει πολλά υποσχόμενα αποτελέσματα σε μια ποικιλία απαιτητικών εργασιών, όπως η παραγωγή εικόνων, η επαλήθευση προσώπου και η εκμάθηση αναπαράστασης 3D mesh. Παρά την πρόσφατη εισαγωγή τους στον τομέα της Βαθιάς Μηχανικής Μάθησης, εξακολουθούν να υπάρχουν πολλές πτυχές των ΒΠΝΔ που χρήζουν περαιτέρω εξερεύνησης προκειμένου να εδραιώσουν τη θέση τους μεταξύ άλλων αρχιτεκτονικών Βαθιάς Μάθησης. Μια τέτοια πτυχή είναι η διερεύνηση των τεχνικών κανονικοποίησης. Η κανονικοποίηση παίζει κρίσιμο ρόλο στη βελτίωση της ικανότητας γενίκευσης ενός μοντέλου, στη μείωση της υπερπροσαρμογής και στη βοήθεια όσον αφορά στη μείωση του πλεονασμού του δικτύου μέσω της εισαγωγής στοχαστικότητας. Ωστόσο, η εφαρμογή τεχνικών κανονικοποίησης στα ΒΠΝΔ παραμένει σε μεγάλο βαθμό ανεξερεύνητη. Με έντονο κίνητρο από το συγκεκριμένο κενό στην τρέχουσα βιβλιογραφία, αυτή η διατριβή στοχεύει να προσφέρει μια εισαγωγική διερεύνηση του τρόπου με τον οποίο η κανονικοποίηση μπορεί να εφαρμοστεί στα ΒΠΝΔ. Προτείνονται και αξιολογούνται δύο μέθοδοι κανονικοποίησης, η χειροκίνητη μείωση βαθμού και η εφαρμογή του dropout. Η χειροκίνητη μείωση βαθμού περιλαμβάνει τη μείωση του βαθμού των τανυστών παραμέτρων στα ΒΠΝΔ, ενώ η κανονικοποίηση μέσω dropout περιλαμβάνει την τυχαία αδρανοποίηση στοιχείων των τανυστών παραμέτρων κατά τη διάρκεια της εκπαίδευσης του δικτύου. Η πειραματική αξιολόγηση στα σύνολα δεδομένων MNIST και CIFAR10 κάτω από διαφορετικά επίπεδα προσθήκης θορύβου και κακόβουλων επιθέσεων αποκαλύπτει ότι τα ΒΠΝΔ είναι εντυπωσιακά ανθεκτικά σε εκείνα τα είδη επιθέσεων. Επιπλέον, τα μοντέλα αποδίδουν καλύτερα και υπερπροσαρμόζονται λιγότερο όταν κανονικοποιούνται με χειροκίνητη μείωση βαθμού, ενώ η κανονικοποίηση μέσω dropout συνήθως οδηγεί σε υποβάθμιση της απόδοσης. Αυτή η προσπάθεια σκοπεύει να χρησιμεύσει ως ένα ολοκληρωμένο θεμέλιο, προσφέροντας πολύτιμες γνώσεις και ευρήματα που θα συμβάλλουν σημαντικά και θα παρακινήσουν τη μελλοντική έρευνα σε αυτόν τον τομέα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Όραση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Βαθιά Μηχανική Μάθηση, Βαθιά Πολυωνυμικά Νευρωνικά Δίκτυα, Κανονικοποίηση, Χειροκίνητη Μείωση Βαθμού, Dropout, Τανυστές

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor, Professor Yannis Panagakis, for his guidance, support and expertise throughout the completion of this thesis. His influential contributions and commitment to excellence have inspired me to strive for the highest standards in my academic pursuits. I would also like to extend my sincere thanks to Dimitrios Halatsis, since his MSc. thesis, *"Trainability of High Degree Polynomial Networks"*, served as a valuable resource and greatly contributed to my understanding of DPNNs. Last but not least, I would like to thank my friends, family and my girlfriend Helena for greatly supporting me throughout this effort.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Deep Polynomial Neural Networks (DPNNs), also known as $\Pi-$Nets, are a novel class of function approximators based on polynomial expansions. They are polynomial neural networks, meaning the output is a high-order polynomial of the input. Despite being a recent addition to the field of Deep Learning, DPNNs have already demonstrated impressive results. These networks have proven to be highly expressive and efficient in a variety of tasks and signals, including images, graphs, and audio. Remarkably, they have achieved these results without the use of non-linear activation functions. Furthermore, when DPNNs are used in conjunction with activation functions, they have produced state-of-the-art results in challenging tasks such as image generation, face verification, and 3D mesh representation learning. This early success of DPNNs underscores their potential as powerful tools in the Deep Learning toolbox [4].

As a novel class, DPNNs have opened up a new frontier in the field of Deep Learning and their early success is indeed promising. However, it is important to note that these networks are still in their infancy. A comprehensive understanding of their full potential is yet to be achieved and will require extensive investigation and experimentation. Some research directions may include exploring different tensor decompositions to further reduce the number of parameters, investigating the impact of different activation functions on the performance of DPNNs, seeking for ways to mitigate overfitting and network redundancy and further studying the theoretical properties of these networks.

Motivated by the existing research gap in the field of DPNNs, we dedicate this thesis to exploring the potential of introducing randomization to these networks. Our study focuses on two distinct methods of stochastically regularizing the networks. The first method involves explicitly reducing the rank of the parameter tensors within the network, thereby simplifying its structure and potentially improving its generalization capabilities. The second method applies dropout regularization to the network's parameters, a technique that has proven effective in preventing overfitting in other types of neural networks.

In this study, we conduct a comprehensive series of experiments on two widely recognized datasets: MNIST and CIFAR10. Our experimental design incorporates multiple levels of Gaussian Noise addition, providing a robust test of the models' performance under various conditions. Furthermore, we subject the models to three distinct adversarial attacks: Fast Gradient Sign Method (FGSM), Basic Iterative Method (BIM), and Projected Gradient Descent (PGD). This rigorous testing approach aims to extract valuable insights regarding the resilience of DPNNs when faced with noise perturbations and adversarial attacks. Our findings shed light on the impact of different regularization architectures on the models. We observe notable variations in model performance, which leads us to draw several conclusions about the relationship between regularization architectures and model resilience.

The rest of the thesis is organised as follows:

- Chapter 2 provides a comprehensive background on the mathematical foundations, history, and definitions of Deep Polynomial Neural Networks, various regularization schemes, and two influential papers that significantly motivated this thesis.
- Chapter 3 delves into the thesis's research objective and analyses the proposed methods for regularizing DPNNs.
- Chapter 4 details the experiments conducted and their respective results.
- Chapter 5 concludes the thesis.
- The appendix includes charts from all conducted experiments.

2. BACKGROUND AND RELATED WORK

In this chapter, we provide a comprehensive overview of the theoretical and empirical knowledge necessary to understand the research on Deep Polynomial Neural Networks and Regularization in Deep Learning. The chapter is divided into four sections. Firstly, we cover the preliminary notation and knowledge required for understanding tensors and tensor decompositions. Additionally, we provide a quick overview of the essence of low-rank representations of data in machine learning [37]. We explain how low-rankness can be used to reduce computational complexity and we present two of the most popular methods for low-rankness. Secondly, we explore the history and evolution of Deep Polynomial Neural Networks, including a discussion of some key concepts and architectures, such as Deep Convolutional Neural Networks and Vanilla Polynomial Networks. Thirdly, we examine the concept of regularization in deep learning, including its definition and various categories and types of regularizers, such as Implicit, Explicit, Deterministic and Stochastic Regularization. Lastly, we delve into our main thesis motivator, by providing an overview of two major papers, *"Tensor Dropout for Robust Learning"* [26] and *"On the Implicit Bias of Dropout"* [35]. By exploring these key topics, this chapter aims to provide the necessary background knowledge and contextual understanding of the subject matter that underpins the subsequent sections of the thesis.

2.1 Preliminaries

2.1.1 Notation

We symbolize:

- Scalars by lowercase non-bold letters, e.g: x.
- Vectors by lowercase bold letters, e.g: $\boldsymbol{y}$.
- The vector of all ones, for any integer d, as $\mathbf{1}_d \in R^d$
- Matrices by uppercase non-bold letters, e.g: M.
- Tensors by uppercase bold letters, e.g: $\boldsymbol{U}$.
- The outer product of $\boldsymbol{x} \in R^o$ and $\boldsymbol{y} \in R^k$ as $\boldsymbol{x} \otimes \boldsymbol{y} \in R^{o \times k}$.
- The *Hadamard* product of $A \in R^{I \times N}$ and $B \in R^{I \times N}$ as $A * B$ and is equal to $A_{(i,j)} B_{(i,j)}$ for the (i, j) element.
- The *Khatri-Rao* product of matrices $A \in R^{I \times N}$ and $B \in R^{J \times N}$ as $A \odot B$. The product yields a matrix of dimensions $(IJ \times N)$. If we have a set of matrices $\{A_{[m]} \in R^{I_m \times N}\}_{m=1}^{M}$, we denote the product as $\bigodot_{m=1}^{M} A_{[m]}$.
- The mode $-\ m$ vector product of a tensor $\boldsymbol{T}$ with a vector $\boldsymbol{u} \in R^{I_m}$, as $\boldsymbol{T} \times_m \boldsymbol{u} \in R^{I_1 \times I_2 \times \cdots \times I_{m-1} \times I_{m+1} \times \cdots \times I_M}$ that results in a tensor of order $M - 1$. For more than one vectors, we denote $\boldsymbol{T} \times_1 \boldsymbol{u}^{(1)} \times_2 \boldsymbol{u}^{(2)} \times_3 \cdots \times_M \boldsymbol{u}^{(M)}$ as $\boldsymbol{T} \prod_{m=1}^{m} \times_m \boldsymbol{u}^{(m)}$.
- The rank of a real-valued matrix $X \in R^{I \times J}$ as rank(X). It is equivalently defined as:
  - The number of linearly independent columns of X.
  - The number of linearly independent rows of X.

  The above definition immediately implies that if $X \in R^{I \times J}$, then rank$(X) \leq$ min(I, J). If rank$(X) =$ min(I, J), then X is full-rank.
- For a matrix $X \in R^{d_1 \times d_2}$, diag$(X) \in R^{min\{d_1, d_2\}}$ as its diagonal elements. Similarly, for a vector $\boldsymbol{x} \in R^d$, diag$(\boldsymbol{x}) \in R^{d \times d}$ is a diagonal matrix with $\boldsymbol{x}$ on its diagonal.
- $\|\boldsymbol{x}\|$ as the l_2-norm of vector $\boldsymbol{x}$ and $\|X\|$ as the spectral norm of a matrix X.
- The Bernoulli distribution, parametrized by a single parameter p, as $\mathcal{B}er(p)$, where p represents the probability of success in a single trial.
- An unknown distribution, with uppercase, bold and calligraphic letters, e.g: $\boldsymbol{\mathcal{D}}$.

2.1.2 Tensors

2.1.2.1 Definition

Tensors are multidimensional arrays that extend the concept of vectors and matrices to higher dimensions. They serve as a foundational tool in numerous areas, including physics, mathematics, computer science, and machine learning. The versatility and expressive power of tensors make them essential for modeling and analyzing complex phenomena [9] [48].

Formally, a tensor of order n is an entity that possesses n indices and can be described as an $n-$dimensional array of numbers. That said, the order of a tensor is the number of indices (dimensions) needed to address its elements. Each dimension is called a mode. Moreover, an N^{th} order tensor has N indices, with each index addressing a mode of $\boldsymbol{T}$. If we assume that $\boldsymbol{T}$ is real-valued, it is defined over the tensor space $R^{I_1 \times I_2 \times \cdots \times I_N}$, where $I_n \in Z$ for $n = 1, 2, \ldots, N$. An element $(i_1, i_2, \ldots, i_N)$ of tensor $\boldsymbol{T} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ is accessed as $\boldsymbol{T}_{i_1, i_2, \ldots, i_N}$. This corresponds to viewing a tensor as a multi-dimensional array in $R^{I_1 \times I_2 \times \cdots \times I_N}$.

2.1.2.2 Tensors vs. Matrices

Although tensors and matrices share similarities, they have distinct characteristics [6]. Matrices are two-dimensional arrays that represent linear transformations between vector spaces; thus they are a special case of tensors where the order is two. Tensors, on the other hand, can have any order, enabling them to capture higher-dimensional relationships in a more comprehensive manner. In terms of operations, matrices have a well-defined algebraic structure and adhere to specific operations such as matrix addition, subtraction, and multiplication. These operations are performed element-wise or using matrix-specific rules (e.g., matrix multiplication). Tensors, however, have their own set of operations tailored to their multidimensional nature. These operations include tensor addition, tensor contraction, tensor outer product, and additionally allow computations across multiple indices and dimensions, providing a powerful means for manipulating and analyzing complex data structures.

2.1.3 Low-Rankness

Low-rank representation of data has become an important topic in machine learning due to its effectiveness in reducing the dimensionality of data [21] [24]. In many applications, such as image and video processing, large datasets can be computationally expensive and time-consuming to analyze. By reducing the rank of the data matrices, it is possible to compress and represent the data in a more compact form, while retaining important information.

In practice, not all data matrices that represent data sets are exactly low-rank. The observed matrix can deviate from the low-rank structure for several reasons, including noise, outliers, and non-linear structures underlying the data. In these cases, the value of the low-rankness paradigm is, for example, to effectively approximate the observed matrix with a low-rank matrix expressed as a product of two factor matrices.

Low-rank representation has been used in various machine learning tasks such as clustering, classification, and anomaly detection. In clustering, for example, low-rank representation has been used to identify subgroups within a larger dataset. In classification, low-rank representation has been used to reduce the complexity of high-dimensional data and improve the accuracy of classification models. In anomaly detection, low-rank representation has been used to identify outliers in a dataset.

One of the most popular methods for low-rank representation is principal component analysis (PCA) [12]. PCA is a linear technique that decomposes the data matrix into a set of orthogonal eigenvectors, ordered by their corresponding eigenvalues. The first few eigenvectors correspond to the most significant dimensions of the data, while the remaining eigenvectors capture the noise or less significant dimensions. PCA has been widely used in many machine learning applications due to its simplicity and effectiveness.

Another popular method for low-rank representation is singular value decomposition (SVD) [10]. SVD is a matrix factorization technique that decomposes the data matrix into three matrices: a left singular matrix, a diagonal matrix of singular values and a right singular matrix. SVD has been used in various machine learning applications such as recommendation systems, image processing, and natural language processing.

Overall, low-rank matrix decompositions have a multitude of valuable applications in machine learning. With the potential to acquire knowledge from small amounts of data and providing the capacity to fill in missing information, these decompositions hold significant power. Additionally, unlike several other models for machine learning, the rapid computation of low-rank matrix decompositions has been significantly developed. Even though they come with a few limitations, they can always be used as a building block for more advanced machine learning models. This is because they can give an interpretable, low-dimensional representation of very high-dimensional data.

2.1.4 Tensor Decompositions

Tensor decompositions extend the concept of matrix decompositions to higher-order tensors. They are mathematical methods used to analyze tensors by providing a framework for representing and understanding complex relationships among multiple variables. By decomposing tensors into simpler components, tensor decompositions enable the extraction of meaningful patterns and insights from the data [39]. Some of the most known tensor decompositions include *Canonical-Polyadic (CP)* [19], also known as PARAFAC [17], CANDECOMP [3] or CANDECOMP/PARAFAC decomposition and *Tucker* [19] [52] decomposition. In the current thesis, we are interested in both understanding and utilizing the aforementioned CP decomposition.

2.1.4.1 CP Decomposition

Canonical Polyadic (CP) decomposition [19] is a technique used in multi-dimensional data analysis in which a multi-dimensional tensor is decomposed into a sum of rank-one tensors. Each rank-one tensor is composed of the product of a vector from each mode or dimension.

Consider a weight tensor $\boldsymbol{W} \in R^{A \times B \times C}$, where rank$(\boldsymbol{W}) = R$. The CP decomposition of this tensor involves finding three tensors $\boldsymbol{U} \in R^{A \times R}$, $\boldsymbol{V} \in R^{B \times R}$ and $\boldsymbol{K} \in R^{C \times R}$, such that $\boldsymbol{W}$ can be expressed as the sum of the outer products of the r^{th} column vectors of tensors $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$ respectively:

$$\boxed{\boldsymbol{W} = \sum_{r=1}^{R} \boldsymbol{u_r} \otimes \boldsymbol{v_r} \otimes \boldsymbol{w_r}} \tag{2.1}$$

2.2 Deep Polynomial Neural Networks

2.2.1 A glance to the past

Over the past decade, deep learning has reached unprecedented heights of success [30] [34] [8] [43]. When it comes to tasks like image recognition and computer vision, Deep Convolutional Neural Networks (DCNNs) have become extremely popular [47] [42] [50] [18] [11]. However, some DCNN architectures lack interpretability and explainability, while others are not able to capture the long-term dependencies and temporal dynamics in sequential data, such as speech and video [55] [25] [1]. Vanilla Polynomial Neural Networks (PNNs) [22] offer a solution to this problem, but have not been widely adopted yet, due to their relatively shallow architectures.

Researchers sought to exceed this limitation by integrating DCNNs and PNNs, thus establishing Deep Polynomial Neural Networks (DPNNs) [4]. In this architecture, there are

layers designed to extract local features using convolution operations, combined with polynomial activation functions to capture non-linear relationships between the features, similar to PNNs. This combination of DCNNs and PNNs allows the network to learn more complex representations of the data and has shown remarkable performance on a wide range of tasks. In this section, we will further explore the individual components of DCNNs and PNNs.

2.2.1.1 Deep Convolutional Neural Networks (DCNNs)

Deep Convolutional Neural Network (DCNN) [13], is one of the most successful types of neural networks and has achieved state-of-the-art results for both generative and discriminative machine learning tasks.

The key feature of DCNNs is the use of convolutional layers, which are designed to detect local patterns within an image. Each convolutional layer consists of a set of filters, which are convolved with the input image to produce a set of activation maps. The filters are learned during the training process, and their values determine which patterns in the input image will activate the corresponding feature map. This process allows the network to learn increasingly complex and abstract features as it progresses through the layers. In addition to convolutional layers, DCNNs typically include other types of layers such as pooling layers, which downsample the feature maps to reduce the spatial dimensionality [53], and fully connected layers, which map the high-level features to the output classes. Eventually, all the aforementioned components make them well-suited for tasks such as image recognition, where the input data has a complex hierarchical structure.

2.2.1.2 Vanilla Polynomial Networks

Polynomial neural networks are a natural extension of the Group Method of Data Handling (GMDH) [32]. The idea of using polynomial activation functions in neural networks dates back to the 1960s, with the development of the PNN architecture by Prof. Alexey G. Ivakhnenko. The PNN is a type of feedforward neural network where each neuron uses a polynomial activation function. These functions are designed to capture the non-linear relationships between the input features and the output, allowing the network to learn complex representations of the data.

The GMDH algorithm was introduced in the 1970s as a method for automatically selecting and combining polynomial models of different orders [22]. The idea is to use a divide-and-conquer approach to break down a complex problem into smaller sub-problems, and then use polynomial models of increasing order to fit the data at each level of the hierarchy. The resulting model is a combination of the polynomial models at each level, with the coefficients of the polynomials learned through a recursive algorithm.

In the context of neural networks, the GMDH algorithm can be used to automatically select the optimal set of input features and the appropriate order of the polynomial activation functions. The GMDH-based PNN architecture uses the GMDH algorithm to construct a hierarchical model of polynomial neurons, where each neuron is a polynomial function of a subset of the input features. The output of the network is obtained by combining the outputs of the polynomial neurons at each level of the hierarchy [44].

2.2.2 Definition

In the new family of $\Pi-$Nets, the output is a high-order polynomial of the input introduced. The unknown parameters in Π-Nets, which are naturally represented by high-order tensors, are estimated through a collective tensor factorization with factors sharing. Such proposed factorization techniques contributed to the avoidance of the combinatorial explosion in the number of parameters of polynomial activation functions.

More specifically, the authors on "*Deep Polynomial Neural Networks*" [4] wanted to create a function approximator where each element of the output x_j, with $j \in [1, o]$ is expressed as a polynomial of all the input elements z_i, with $i \in [1, d]$. That is, they wanted to learn a function $G : R^d \to R^o$ of order $N \in N$, such that:

$$x_j = G(\boldsymbol{z})_j = b_j + \boldsymbol{w}_j^{[1]^T}\boldsymbol{z} + \boldsymbol{z}^T W_j^{[2]}\boldsymbol{z} + \boldsymbol{W}_j^{[3]} \times_1 \boldsymbol{z} \times_2 \boldsymbol{z} \times_3 \boldsymbol{z} + \cdots + \boldsymbol{W}_j^{[N]} \prod_{n=1}^{N} \times_n \boldsymbol{z} \tag{2.2}$$

where $b_j \in R$ and $\{\boldsymbol{W}_j^{[n]} \in R^{\prod_{m=1}^{n} \times_m d}\}_{n=1}^{N}$ are parameters for approximating the output x_j. Concretely, by vectorizing (2.2) we get:

$$\boxed{\boldsymbol{x} = G(\boldsymbol{z}) = \sum_{n=1}^{N} (\boldsymbol{W}^{[n]} \prod_{j=2}^{n+1} \times_j \boldsymbol{z}) + \boldsymbol{b}} \tag{2.3}$$

where $\boldsymbol{b} \in R^o$ and $\{\boldsymbol{W}^{[n]} \in R^{o \times \prod_{m=1}^{n} \times_m d}\}_{n=1}^{N}$ are the learnable parameters.

There are two main architectures proposed in order to build efficient DPNNs:

1. Expressing the function approximation as a single polynomial by using CP tensor decomposition (see 2.1.4.1) on the parameters.
2. Expressing the function approximation as a product of lower-degree polynomials.

2.2.2.1 Single Polynomial

CCP (Coupled CP decomposition)

In order to avoid factorizing each parameter tensor $\boldsymbol{W}^{[n]}$ individually, the authors proposed a method to jointly factorize all the parameter tensors, via a coupled CP decomposition with a specific pattern of factor sharing. Let's say that we have a third order approximation $(N = 3)$ and that the parameter tensors admit the following coupled CP decomposition with the factors corresponding to lower order levels of approximation being shared across all parameter tensors:

- First level of approximation: $W^{[1]} = CU_{[1]}^T$.
- Second level of approximation will result in $\boldsymbol{W}^{[2]}$ by combining two weight tensors: $\boldsymbol{W}_{1:2}^{[2]} + \boldsymbol{W}_{1:3}^{[2]}$. $\boldsymbol{W}_{1:2}^{[2]}$ represents the parameters associated with the second-order interactions across the first and second levels of approximation. It captures the interactions between neurons in the first and second layer of the network. Similarly, $\boldsymbol{W}_{1:3}^{[2]}$ represents the parameters associated with the second-order interactions across the first and third levels of approximation. It captures the interactions between neurons in the first and the third layer of the network. By enforcing the CP decomposition of the above tensors to share the factor with tensors corresponding to lower-order of approximation we obtain in matrix form:

$$\boldsymbol{W}^{[2]} = C(U_{[3]} \odot U_{[1]})^T + C(U_{[2]} \odot U_{[1]})^T$$

- Third level of approximation: $\boldsymbol{W}^{[3]} = C(U_{[3]} \odot U_{[2]} \odot U_{[1]})^T$

Note that the parameters are $C \in R^{o \times k}$ and $U_{[m]} \in R^{d \times k}$ for $m = 1, 2, 3$. By combining all of the above levels of approximation with the equation (2.3), we finally end up with:

$$\begin{aligned} G(\boldsymbol{z}) = \boldsymbol{b} + CU_{[1]}^T\boldsymbol{z} + C(U_{[3]} \odot U_{[1]})^T(\boldsymbol{z} \odot \boldsymbol{z}) + \\ C(U_{[2]} \odot U_{[1]})^T(\boldsymbol{z} \odot \boldsymbol{z}) + C(U_{[3]} \odot U_{[2]} \odot U_{[1]})^T(\boldsymbol{z} \odot \boldsymbol{z} \odot \boldsymbol{z}) \end{aligned} \tag{2.4}$$

If we have a set of N matrices $\{A_{[\nu]}R^{I_\nu \times K}\}_{\nu=1}^N$ and $\{B_{[\nu]}R^{I_\nu \times L}\}_{\nu=1}^N$, the following equality holds:

$$(\bigodot_{\nu=1}^{N} A_{[\nu]})^T \cdot (\bigodot_{\nu=1}^{N} B_{[\nu]})^T = (A_{[1]}^T \cdot B_{[1]}) * \cdots * (A_{[N]}^T \cdot B_{[N]}) \tag{2.5}$$

Therefore, If we apply (2.5) to (2.4), we obtain:

$$\begin{aligned} G(\boldsymbol{z}) = \boldsymbol{b} + C\{(U_{[3]}^T\boldsymbol{z}) * [(U_{[2]}^T\boldsymbol{z}) * (U_{[1]}^T\boldsymbol{z}) + \\ U_{[1]}^T\boldsymbol{z}] + (U_{[2]}^T\boldsymbol{z}) * (U_{[1]}^T\boldsymbol{z}) + U_{[1]}^T\boldsymbol{z}\} \end{aligned} \tag{2.6}$$

and we can finally construct a corresponding neural network that represents it. The network's structure is illustrated in figure 2.1. The recursive relationship considering the N^{th} order approximation is:

$$\boxed{\boldsymbol{x}_n = (U_{[n]}\boldsymbol{z}) * \boldsymbol{x}_{n-1} + \boldsymbol{x}_{n-1}} \tag{2.7}$$

for $n \in N^+, n \leq N$ where :

- $\boldsymbol{z} \in R^d$ is the input signal.
- $\boldsymbol{x}_n \in R^o$ is the output after n recursive iterations.
- $U_{[n]} \in R^{o \times d}$ is the n^{th} parameter matrix.
- $\boldsymbol{x}_1 = U_{[1]}\boldsymbol{z}$

The resulting equation will be:

$$\boxed{\boldsymbol{x} = G(\boldsymbol{z}) = C\boldsymbol{x}_n + \boldsymbol{b}} \tag{2.8}$$

where :

- $G(\boldsymbol{z}) : R^d \rightarrow R^o$ is the polynomial output of the input $\boldsymbol{z}$.
- $C \in R^{k \times o}$ and $\boldsymbol{b} \in R^o$.

Note that U_n, C and $\boldsymbol{b}$ are learnable parameters.

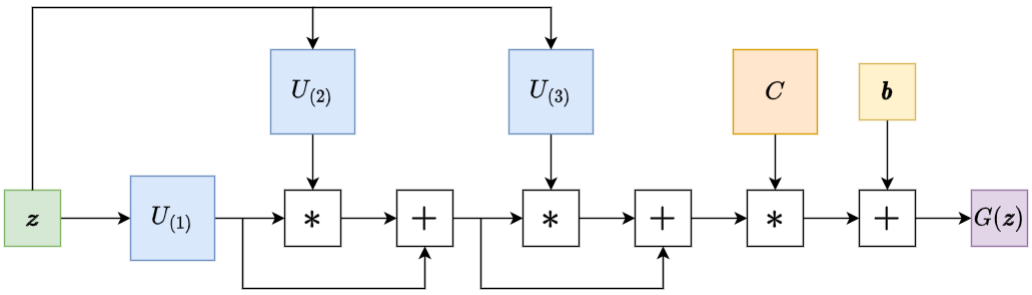

Figure 2.1: Schematic illustration of CCP for third order approximation.

NCP (Nested coupled CP decomposition)

While CCP decomposition involves decomposing two or more tensors simultaneously using a shared factor matrix, NCP decomposition on the other hand, is an extension of coupled CP decomposition that allows for a hierarchical factorization of the data. That means that each tensor can be decomposed into multiple layers of shared and unique factors. In other words, the shared factors that are identified in the first level of the decomposition are further decomposed into their own shared and unique factors in the second level, and so on. The authors introduced some learnable hyperparameters $\{\boldsymbol{b}_{[n]} \in R^{\omega}\}_{n=1}^{N}$, which act as scaling factors for each parameter tensor. That said, the equation (2.3) is transformed to:

$$\boxed{G(\boldsymbol{z}) = \sum_{n=1}^{N}(\boldsymbol{W}^{[n]} \times_2 \boldsymbol{b}_{[N+1-n]} \prod_{j=3}^{n+2} \times_j \boldsymbol{z}) + \boldsymbol{\beta}} \tag{2.9}$$

with $\{\boldsymbol{W}^{[n]} \in R^{o \times \omega \times \prod_{m=1}^{n} \times_m d}\}_{n=1}^{N}$.

Similarly to CCP, the resulting model can be formulated in a recursive relation as follows:

$$\boxed{\boldsymbol{x}_n = (U_{[n]}\boldsymbol{z}) * (S_{[n]}\boldsymbol{x}_{n-1} + B_{[n]}\boldsymbol{b}_{[n]})} \tag{2.10}$$

for $n \in N^+, n \leq N$ where :

- $\boldsymbol{z} \in R^d$ is the input signal.
- $\boldsymbol{x}_n \in R^o$ is the output after n recursive iterations.
- $U_{[n]} \in R^{o \times d}$ is the n^{th} parameter matrix.
- $S_{[n]} \in R^{o \times o}$, $B_{[n]} \in R^{o \times \omega}$, $\boldsymbol{b}_{[n]} \in R^{\omega}$
- $\boldsymbol{x}_1 = (U_{[1]}\boldsymbol{z}) * (B_{[1]}\boldsymbol{b}_{[1]})$

And of course, the final outcome after N iterations will still be $G(\boldsymbol{z})$ as mentioned above. Note that $U_{[n]}$, $S_{[n]}$, $B_{[n]}$ and $\boldsymbol{b}_{[n]}$ are learnable parameters.

Variations of CCP and NCP decompositions

There is a wide array of architectures for both CCP and NCP that hold significant potential for theoretical analysis and experimental investigation. In the realm of CCP, two notable variations emerge: the CCP-no-skip and CCP-bias architectures.

The CCP-no-skip model bears a strong resemblance to the original CCP architecture, with the primary distinction being the absence of the additional skip-connection term:

$$\boxed{\boldsymbol{x}_n = (U_{[n]}\boldsymbol{z}) * \boldsymbol{x}_{n-1}} \tag{2.11}$$

In the CCP-bias formulation, there exists a *bias term* that enables the short circuiting of the CP terms to the output:

$$\boxed{\boldsymbol{x}_n = (U_{[n]}\boldsymbol{z}) * (\boldsymbol{x}_{n-1} + \boldsymbol{b}_{[n]})} \tag{2.12}$$

Regarding NCP decomposition, inspired by the skip-connections inherent in the standard CCP architecture, scholars have introduced the NCP-skip model. This model incorporates the traditional NCP formulation and augments it with a skip-connection, thereby enhancing its expressivity:

$$\boxed{\boldsymbol{x}_n = (U_{[n]}\boldsymbol{z}) * (S_{[n]}\boldsymbol{x}_{n-1} + B_{[n]}\boldsymbol{b}_{[n]}) + V_{[n]}\boldsymbol{x}_{n-1}} \tag{2.13}$$

where $V_{[n]} \in R^{o \times o}$.

2.2.2.2 Product of Polynomials

Instead of expressing the function approximation as a single polynomial, the product of polynomials formulation gathers successive blocks of polynomials and connects them by passing the output of the i^{th} polynomial as the input to the $(i+1)^{th}$ polynomial. Note that the single polynomial model is increasing the degree of the polynomial by applying hadamard product to a linear combination of the initial input, while the product of polynomials stacks N polynomials of, let's say, degree D and results to a total order of D^N. It is worth mentioning that the above example was trivial, since this formulation does not require the polynomial-blocks to have the same order. Moreover, researchers have proven that not only it requires much fewer parameters for achieving the same order of approximation, but it also allows using different decompositions in contrast to the single polynomial architecture.

2.3 Regularization in Deep Learning

2.3.1 What is regularization?

Deep Learning has become a dominant approach for tackling complex challenges across various domains, such as computer vision, natural language processing (NLP) and speech recognition. It is worth noting, that deep neural networks are capable of learning millions, billions or even trillions of parameters while building a model and can be highly flexible. That leads us to situations where they can easily fit to the training data, often leading to overfitting, a phenomenon where a model learns the training data too well, including noise and random fluctuations, and is not able to generalize well to new, unseen data. Regularization is a deep learning technique that aims to mitigate overfitting by encouraging the model to produce simpler or smoother solutions that tend to generalize better to new data [15].

2.3.2 Categories & types of regularizers

While there are several ways to categorize regularization procedures, the two most prevalent classifications are implicit versus explicit regularization and deterministic versus stochastic regularization. The distinction between implicit and explicit regularization hinges on whether the regularization is directly incorporated into the objective function. On the other hand, the classification of deterministic versus stochastic regularization is determined by whether the regularization employs deterministic or stochastic operations [36].

2.3.2.1 Implicit Regularization

Implicit regularization occurs naturally as a consequence of the optimization algorithm used to train the model. This regularization is typically attributed to the optimization process ”pushing” the model towards solutions that are less likely to overfit the training data. This hypothesis asserts that the dynamics of the training procedure implicitly introduces a regularization effect and therefore the explicit imposition of a regularizer is unnecessary. Some of the most famous implicit regularization techniques are Early Stopping [54] [40] and Stochastic Gradient Descent (SGD) [2].

In the concept of Early Stopping, during training, the model’s performance on a validation set is monitored. If the performance on the validation set stops improving or starts to deteriorate, the training is stopped early. This prevents the model from becoming too specialized to the training data and allows it to generalize better to unseen examples. Early stopping requires a separate validation set and the determination of an optimal point to stop training, which is often based on a predefined number of iterations or epochs without improvement. By finding the right balance, early stopping helps improve model performance and generalization.

In Stochastic Gradient Descent (SGD), instead of computing gradients using the entire dataset, the algorithm randomly selects a subset of data (a mini-batch) to compute the gradient estimate. This reduces computational requirements and speeds up training. SGD iteratively updates the model's parameters by taking steps in the direction of the negative gradient with a predefined learning rate. This process continues until a stopping criterion is met, such as reaching a maximum number of iterations or achieving satisfactory performance.

2.3.2.2 Explicit Regularization

Explicit regularization works by explicitly adding a penalty term to the loss function that the model optimizes during training. The penalty term is typically a function of the model parameters, and the regularization strength controls the trade-off between fitting the training data well and producing a simpler or smoother model.

The simplest formula that describes the explicit regularization on an optimization problem is as follows:

$$\min_{f} \sum_{i=1}^{N} L(f(x_i), y_i) + \lambda R(f) \tag{2.14}$$

where:

- There are N labels we want to train on.
- f is our model function.
- L is a loss function that describes the cost of predicting $f(x)$ when the label is y.
- $R(f)$ is a regularization term (or regularizer) which is added to a loss function.
- λ is a parameter which controls the importance of the regularization term.

Therefore, we wish to minimize the loss of our model function for all target labels with the addition of the regularization term, in order to impose a penalty on the complexity of f. A few of the most famous explicit regularization techniques are L1 [51], L2 [20] and Elastic Net Regularization [57].

L1 (or Lasso) Regularization encourages the model to simultaneously minimize the loss and the sum of the absolute values of the coefficients. This promotes sparsity in the model by driving some coefficients to exactly zero, effectively performing feature selection.

L2 Regularization (or Ridge Regression) encourages the model to simultaneously minimize the loss and the sum of the squared values of the coefficients. It effectively controls

the magnitude of the coefficients, shrinking them towards zero. L2 regularization is particularly useful for preventing overfitting and improving generalization.

Elastic Net Regularization combines L1 and L2 regularization by adding both penalty terms to the loss function. It provides a balance between the sparsity-inducing property of L1 regularization and the coefficient magnitude control of L2 regularization. Elastic Net is useful in scenarios where there are multiple correlated features and can effectively handle feature selection and collinearity.

2.3.2.3 Deterministic vs Stochastic Regularization

Deterministic regularization involves adding a fixed penalty term to the objective function to encourage simpler or more generalizable models. Common deterministic regularization techniques include $L1$ and $L2$ regularization, which add an L1 or L2 norm of the weights to the objective function.

Stochastic regularization involves introducing randomness or noise into the training process, in order to prevent the model from overfitting to the training data. Dropout [49] and data augmentation [46] are common stochastic regularization techniques.

Data augmentation is a technique used in machine learning to increase the diversity of a training dataset. It involves applying transformations and modifications to existing data samples to create synthetic samples. By introducing variations representative of real-world scenarios, data augmentation improves the model's ability to generalize and perform well on unseen examples. It helps prevent overfitting by exposing the model to a larger and more diverse set of examples, reducing the risk of memorizing specific instances. Data augmentation is particularly useful when labeled data is limited or when dealing with imbalanced datasets. It enhances model performance by providing a larger and more representative training dataset.

While this thesis covers a range of topics, particular attention is given to dropout.
Dropout is a regularization technique commonly used in neural networks to prevent overfitting. The term "dropout" refers to dropping out the nodes in a neural network. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

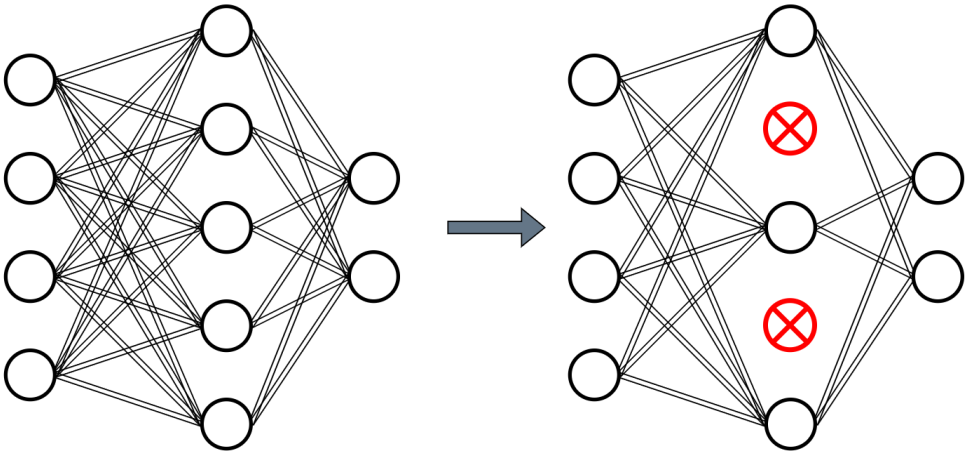

Figure 2.2: Neural network before and after dropout.

2.4 Thesis Motivator

In this section, we will focus on the essence of stochastic regularization for deep neural networks, by introducing two papers that were a powerful motivator for the development of this thesis. By presenting these papers, we hope to provide a thorough understanding of the background that supports the contributions made in this thesis.

2.4.1 Tensor Dropout for Robust Learning

This paper introduces a novel approach to improving the robustness of deep neural networks to both adversarial and random perturbations [26]. The approach involves using tensor regression layers instead of flattening and fully-connected layers, which allows for the preservation of the structure of multidimensional data. A higher-order randomized factorization method is applied to these layers, leading to the stochastic reduction of the rank during both training and inference. More specifically, the proposed method is called tensor dropout. The authors demonstrate the effectiveness of tensor dropout in improving the inductive bias of CNNs, resulting in superior performance compared to regular deep neural network architectures with fully-connected layers and networks with tensor regression layers that do not incorporate their proposed randomized decomposition. They establish a new state-of-the-art for large scale regression from MRI data and show that their model is significantly more robust to noise in the input, as well as adversarial noise, without adversarial training. They also demonstrate that their proposed method implicitly regularizes the tensor decomposition and establish theoretically and empirically the link between tensor dropout and deterministic low-rank tensor regression.

One of the key takeaways from this paper that we will shift our focus to, is that the proposed tensor dropout method applies dropout to the latent factors of the tensor decomposition, effectively randomizing the rank of the weight tensor and providing a regularization effect

that eventually improves the overall generalization performance of the model.

2.4.2 On the Implicit Bias of Dropout

The main focus of the paper is to understand the implicit bias in deep learning systems through the mathematical analysis of the dropout technique [35]. While dropout has had a tremendous success in training deep neural networks, the theoretical understanding of how dropout provides regularization in deep learning remained limited.

More specifically, the paper investigates the bias induced:

- in single hidden-layer linear neural networks,
- in the special case of an autoencoder with tied weights,

and also provides a complete characterization of the optimization landscape induced by dropout.

The authors propose a natural learning algorithm which consists of back-propagation with dropout and they show that the objective function for dropout can be written as the sum of the expected loss and a regularizer.

In mathematical terms, the authors consider the following learning problem. Let $\boldsymbol{x} \in R^{d_2}$ represent an input feature vector with some unknown distribution $\boldsymbol{\mathcal{D}}$ such that $E_{\boldsymbol{x}\sim\boldsymbol{\mathcal{D}}}[\boldsymbol{x}\boldsymbol{x}^T] = I$. The output label vector $\boldsymbol{y} \in R^{d_1}$ is given as $\boldsymbol{y} = M\boldsymbol{x}$ for some $M \in R^{d_1 \times d_2}$. They consider the hypothesis class represented by a single hidden-layer linear network parametrized as $h_{U,V}(\boldsymbol{x}) = UV^T\boldsymbol{x}$, where $V \in R^{d_2 \times r}$ and $U \in R^{d_1 \times r}$ are the weight matrices in the first and the second layers, respectively. The goal of learning is to find weight matrices U, V that minimize the expected loss:

$$l(U,V) := E_{\boldsymbol{x}\sim\boldsymbol{\mathcal{D}}}[\|\boldsymbol{y} - h_{U,V}(\boldsymbol{x})\|^2] = E_{\boldsymbol{x}\sim\boldsymbol{\mathcal{D}}}[\|\boldsymbol{y} - UV^T\boldsymbol{x}\|^2] \tag{2.15}$$

The aforementioned natural learning algorithm is the following:

$$f(U,V) := E_{b_i\sim\text{Ber}(\theta),\boldsymbol{x}\sim\boldsymbol{\mathcal{D}}}[\|\boldsymbol{y} - \frac{1}{\theta}U\text{diag}(\boldsymbol{b})V^T\boldsymbol{x}\|^2] \tag{2.16}$$

and they prove that the expression can also be written as:

$$f(U,V) = l(U,V) + \lambda\sum_{i=1}^{r}\|\boldsymbol{u}_i\|^2\|\boldsymbol{v}_i\|^2 \tag{2.17}$$

where $\lambda = \frac{1-\theta}{\theta}$ is the regularization parameter and $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ represent the i^{th} columns of U and V, respectively. By setting $R(U,V) := \lambda\sum_{i=1}^{r}\|\boldsymbol{u}_i\|^2\|\boldsymbol{v}_i\|^2$, they prove that R (the regularizer) is closely related to path regularization. The authors formally prove that this

regularizer serves as an explicit instantiation of the implicit bias of dropout.

The paper contributes to the understanding of implicit regularization in deep learning systems by analyzing the behavior of dropout in a simple model. The findings of the paper suggest that dropout induces an implicit bias that helps deep learning systems generalize even in over-parametrized settings.

3. METHOD

3.1 Research Objective

3.1.1 The expressiveness of DPNNs

As shown extensively in 2, Deep Polynomial Neural Networks are widely known for their high expressive power, which is attributed to their use of polynomial activation functions in their hidden layers. These functions allow the network to represent complex, nonlinear relationships between the input features and the target variable by expanding the input features into higher-order polynomial terms. This allows the network to capture higher-order interactions between the features and learn more intricate relationships within the data. By using polynomial expansions, DPNNs can approximate any continuous function to an arbitrary degree of accuracy, making them a powerful tool for solving complex problems.

3.1.2 The danger of overfitting

Despite their high expressive power, one of the main challenges associated with the use of deep polynomial networks is overfitting. The flexibility provided by the polynomial activation functions makes these networks highly susceptible to overfitting, which occurs when the model becomes too complex and captures noise in the training data instead of generalizing to new data. In particular, high-order polynomials are significantly prone to overfitting, especially when the data do not have sufficient variability or the hypothesis space is large. This occurs because high-order polynomials have a large number of coefficients, which can be used to fit even the smallest variations in the training data. However, when presented with new data, the network may perform poorly due to its inability to generalize beyond the training data. Therefore, it is essential to carefully balance the model's complexity with its generalization performance to avoid overfitting.

3.2 Methodology

In our research, we experiment with the classic DPNN implementation using the plain CCP decomposition, as shown in recursive formula (2.7). Furthermore, our codebase is based on a minimal DPNN implementation, using only linear transformation layers. However, for those interested in tackling more substantial datasets and achieving state-of-the-art results, exploration of more intricate architectures, such as these ones, is encouraged.

Focusing on the methodology, in order to mitigate the risk of overfitting and improve the generalization ability of DPNNs on new, unseen data, we aim to apply two levels of regularization:

1. Explicitly control the rank of the weight tensors, in order to create low rank representations of the parameters.
2. Add dropout to the model's weights as an additional regularizer.

Each one of the regularization levels will be thoroughly discussed in this chapter.

3.2.1 Explicit rank control

Consider a third order approximation of an input vector $\boldsymbol{z}$, done with plain CCP decomposition. Based on (2.7), the recursive formula will result in:

$$G(\boldsymbol{z}) = C\boldsymbol{x_3} + b$$

where,

$$\boldsymbol{x}_3 = (U_{[3]}\boldsymbol{z}) * \boldsymbol{x}_2 + \boldsymbol{x}_2$$
$$\boldsymbol{x}_2 = (U_{[2]}\boldsymbol{z}) * \boldsymbol{x}_1 + \boldsymbol{x}_1$$

and finally,

$$\boldsymbol{x}_1 = (U_{[1]}\boldsymbol{z})$$

The schematic illustration is depicted in figure 2.1. Note that $U_{[i]},\ 1 \leq i \leq 3$ are the weight matrices of the network. For example, if we want to build a $t-$order DPNN, we should add t consecutive layers.

Let's say that $U_{[i]} \in R^{N \times M}$, $1 \leq i \leq 3$. It is important to consider that the rank of each one of the weight matrices is at most $\min(N, M)$, since the rank of a matrix is the number of its linearly independent rows or columns. We can try to explicitly reduce the rank of $U_{[i]}$ by decomposing it to two smaller matrices, such that each $U_{[i]}$ is decomposed to the product $B_{[i]}A_{[i]}^T$, where $A_{[i]} \in R^{M \times k}$ and $B_{[i]} \in R^{N \times k}$, $k \ll N, M$. That way, the rank of the matrix product $B_{[i]}A_{[i]}^T$ is at most equal to the minimum of the rank of matrix $A_{[i]}$ and the rank of matrix $B_{[i]}$; thus the total rank cannot be more than $k \ll \text{rank}(U_{[i]})$. By applying this de-

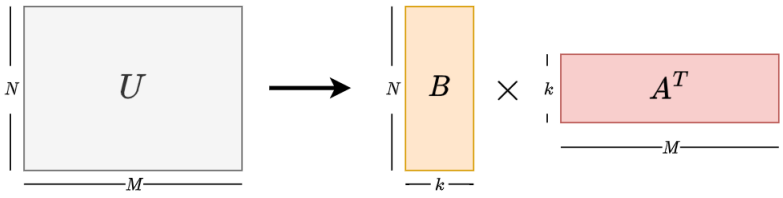

Figure 3.1: Low rank representation of U matrix.

composition, we manage to add a regularization layer to the current DPNN architecture, by explicitly representing the model's parameter matrices with lower rank structures. The modified version of CCP is illustrated in schema 3.2.

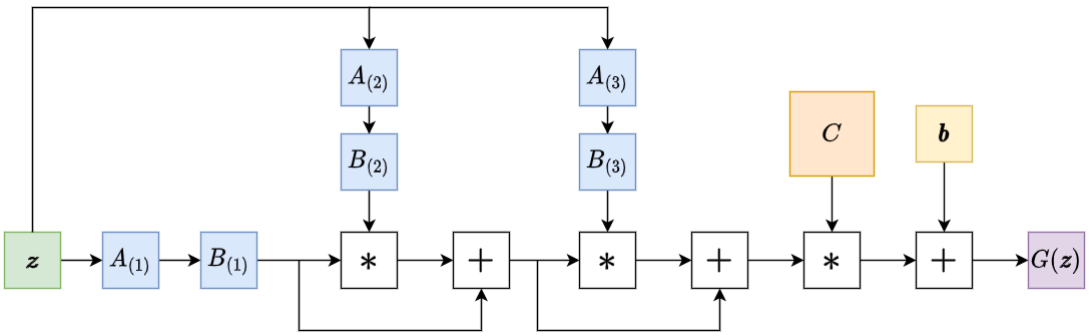

Figure 3.2: Schematic illustration of modified CCP for third order approximation.

3.2.2 Dropout Regularizer

As seen in subsection 2.4.2, the authors of paper "*On the Implicit Bias of Dropout*" mathematically prove that the dropout technique serves as a regularizer in two architectures:

- In single hidden-layer linear neural networks.
- In the special case of an autoencoder with tied weights.

We will try to extend this claim, by adding the family of DPNNs. Let's say we have a n^{th} degree DPNN with linear layers, $A \in R^{M\times k}$, $B \in R^{N\times k}$ and input vector $\boldsymbol{z} \in R^{M}$. Following (2.7), the recursive formula will be as follows:

$$\boldsymbol{x_0} = B_{[0]}A_{[0]}^T\boldsymbol{z}$$

$$\boldsymbol{x_1} = (B_{[1]}A_{[1]}^T\boldsymbol{z}) * \boldsymbol{x_0} + \boldsymbol{x_0} = (B_{[1]}A_{[1]}^T\boldsymbol{z} + \mathbf{1}_N) * \boldsymbol{x_0} = (B_{[1]}A_{[1]}^T\boldsymbol{z} + \mathbf{1}_N) * B_{[0]}A_{[0]}^T\boldsymbol{z}$$

$$\boldsymbol{x_2} = (B_{[2]}A_{[2]}^T\boldsymbol{z}) * \boldsymbol{x_1} + \boldsymbol{x_1} = (B_{[2]}A_{[2]}^T\boldsymbol{z} + \mathbf{1}_N) * (B_{[1]}A_{[1]}^T\boldsymbol{z} + \mathbf{1}_N) * B_{[0]}A_{[0]}^T\boldsymbol{z}$$

$$\vdots$$

$$\boldsymbol{x_n} = (B_{[n]}A_{[n]}^T\boldsymbol{z} + \mathbf{1}_N) * (B_{[n-1]}A_{[n-1]}^T\boldsymbol{z} + \mathbf{1}_N) * \ldots * (B_{[1]}A_{[1]}^T\boldsymbol{z} + \mathbf{1}_N) * B_{[0]}A_{[0]}^T\boldsymbol{z} \Leftrightarrow$$

$$\boxed{\boldsymbol{x_n} = [\Pi_{i=0}^{n-1}(B_{[n-i]}A_{[n-i]}^T\boldsymbol{z} + \mathbf{1}_N)] * B_{[0]}A_{[0]}^T\boldsymbol{z}} \tag{3.1}$$

By plugging DPNN's recursive formula (3.1) in equations (2.15) and (2.16), we end up with a modified expected loss $l(A, B)$ and objective function $f(A, B)$, as seen below. Note that for simplicity, we set diag($\boldsymbol{b}$) to D, for every diagonal matrix that has its items in the diagonal sampled from the Bernoulli distribution. In the same style as above, for a n^{th} degree DPNN with the same A, B layers and dropout regularization, we have:

$$\boldsymbol{x_0} = \frac{1}{\theta}B_{[0]}D_{[0]}A_{[0]}^T\boldsymbol{z}$$

$$\boldsymbol{x_1} = \frac{1}{\theta} B_{[1]} D_{[1]} A_{[1]}^T \boldsymbol{z} * \boldsymbol{x_0} + \boldsymbol{x_0} = (\frac{1}{\theta} B_{[1]} D_{[1]} A_{[1]}^T \boldsymbol{z} + \mathbf{1}_N) * \boldsymbol{x_0}$$

$$\vdots$$

$$\boldsymbol{x_n} = (\frac{1}{\theta} B_{[n]} D_{[n]} A_{[n]}^T \boldsymbol{z} + \mathbf{1}_N) * (\frac{1}{\theta} B_{[n-1]} D_{[n-1]} A_{[n-1]}^T \boldsymbol{z} + \mathbf{1}_N) * \dots$$

$$\dots * (\frac{1}{\theta} B_{[1]} D_{[1]} A_{[1]}^T \boldsymbol{z} + \mathbf{1}_N) * \frac{1}{\theta} B_{[0]} D_{[0]} A_{[0]}^T \boldsymbol{z} \Leftrightarrow$$

$$\boldsymbol{x_n} = \frac{1}{\theta} (B_{[n]} D_{[n]} A_{[n]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * \frac{1}{\theta} (B_{[n-1]} D_{[n-1]} A_{[n-1]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * \dots$$

$$\dots * \frac{1}{\theta} (B_{[1]} D_{[1]} A_{[1]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * \frac{1}{\theta} B_{[0]} D_{[0]} A_{[0]}^T \boldsymbol{z} \Leftrightarrow$$

$$\boldsymbol{x_n} = \frac{1}{\theta^{n+1}} (B_{[n]} D_{[n]} A_{[n]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * (B_{[n-1]} D_{[n-1]} A_{[n-1]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * \dots$$

$$\dots * (B_{[1]} D_{[1]} A_{[1]}^T \boldsymbol{z} + \theta \mathbf{1}_N) * B_{[0]} D_{[0]} A_{[0]}^T \boldsymbol{z} \Leftrightarrow$$

$$\boxed{\boldsymbol{x_n} = \frac{1}{\theta^{n+1}} [\Pi_{i=0}^{n-1} (B_{[n-i]} D_{[n-i]} A_{[n-i]}^T \boldsymbol{z} + \theta \mathbf{1}_N)] * B_{[0]} D_{[0]} A_{[0]}^T \boldsymbol{z}} \tag{3.2}$$

The neural network created by the equation (3.2) is illustrated in figure 3.3. All in all, by

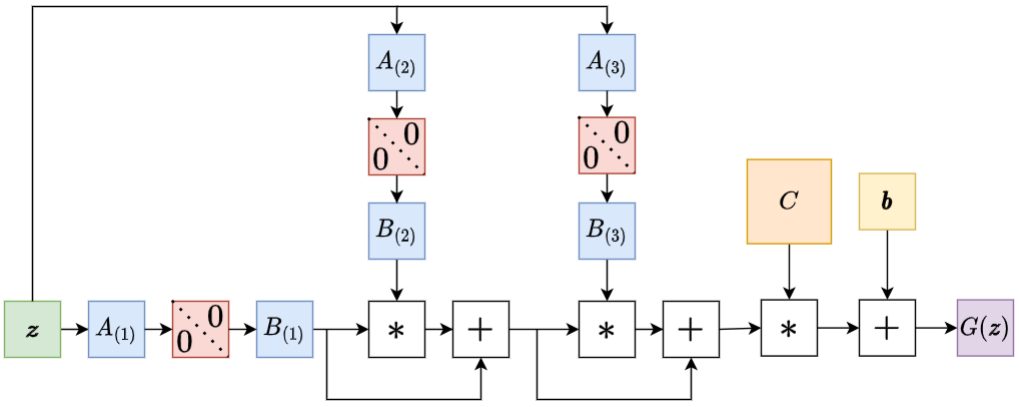

Figure 3.3: Schematic illustration of modified CCP with dropout regularization, for third order approximation.

plugging (3.2) to the corresponding loss $l(A, B)$ and objective $f(A, B)$, we will end up with:

$$l(A, B) := E_{\boldsymbol{x} \sim \boldsymbol{\mathcal{D}}} [\|\boldsymbol{y} - [\Pi_{i=0}^{n-1} (B_{[n-i]} A_{[n-i]}^T \boldsymbol{z} + \mathbf{1}_N)] * B_{[0]} A_{[0]}^T \boldsymbol{z}\|^2]$$

$$f(A, B) := E_{b_i \sim \text{Ber}(\theta), \boldsymbol{x} \sim \boldsymbol{\mathcal{D}}} [\|\boldsymbol{y} - \frac{1}{\theta^{n+1}} [\Pi_{i=0}^{n-1} (B_{[n-i]} D_{[n-i]} A_{[n-i]}^T \boldsymbol{z} + \theta \mathbf{1}_N)] * B_{[0]} D_{[0]} A_{[0]}^T \boldsymbol{z}\|^2]$$

Likewise, as shown in 2.17, we expect $l(A, B)$ and $f(A, B)$ to be connected by a similar formula and provide a legitimate regularization effect related to the one proven on the paper *"On the Implicit Bias of Dropout"*. The mathematical proof of the aforementioned relationship will not be further developed in this thesis, as it is left as future work.

4. EXPERIMENTS

4.1 Datasets

For our experiments, we utilized the DPNN architecture for entirely image classification tasks. In terms of datasets, we used two of the most famous classification datasets: MNIST [31] & CIFAR10 [27].

4.1.1 MNIST

MNIST (*Modified National Institute of Standards and Technology*), is a dataset of handwritten digits that consists of:

- a training set of 60.000 images,
- a test set of 10.000 images.

All images are grayscale and have dimensions of 28x28 pixels. Each image in the dataset represents a handwritten digit from 0 to 9 (10 classes), and the goal is to classify each image correctly.

4.1.2 CIFAR10

CIFAR10 is a dataset of small color images. It consists of:

- a training set of 50.000 images,
- a test set of 10.000 images.

Each image is a color image represented in RGB format and has dimensions of 32x32 pixels and belongs to one of 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). The goal is again to classify each image correctly into one of these 10 classes.

4.2 Implementation Details

4.2.1 General

We used Google Colab & Kaggle as the environments on which the experiments ran. The code is written in Python and is built with PyTorch framework [38]. Before training,

the images are normalized to having mean=0.0 and std=1.0. The chosen optimization method is Stochastic Gradient Descent (SGD) with a learning rate of 0.001. Regarding the loss function, we used both Cross Entropy Loss (CEL) [45] and Mean Squared Error (MSE) [7] [14] [23] [29]. The weights of the network are being initialized using the Xavier normal method [16] and for the flowing gradients, gradient clipping is being used, with a clipping threshold of 10 (maximum 2-norm value) [56]. Various configurations have been tested, for an epoch range of 10 - 20. The degrees of the polynomial neural networks we experimented with are 8 and 16.

4.2.2 Data Perturbation

In order to test both the robustness of our network and its capabilities of generalizing when trained and tested on real-world data, we applied two categories of data perturbation techniques:

- The first category involves adding Gaussian Noise to our data.
- The second category focuses on three gradient-based adversarial attacks:
    1. Fast Gradient Sign Method.
    2. Basic Iterative Method.
    3. Projected Gradient Descent.

4.2.2.1 Addition of Gaussian Noise

Gaussian Noise is a statistical noise with a Gaussian (normal) distribution. It means that the noise values are distributed in a normal Gaussian way. Of course, the Gaussian noise is added to both training and test data, as it better simulates real-world data, where noise and other forms of data distortion affect them [5]. The probability density function p of a Gaussian random variable z is calculated by the following formula:

$$\boxed{p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}}}$$

where:

- where z represents the grey level,
- μ the mean grey value,
- σ the standard deviation.

The addition of Gaussian noise to the input data acts as a form of regularization by introducing small perturbations that force the model to learn more robust and generalizable features. The noise effectively adds a level of uncertainty to the training data, forcing the model to learn more robust representations of the data that are less dependent on the specific details of the training samples. As a result, the model becomes less likely to overfit to the training data and can better generalize to new, unseen data.

Additionally, adding Gaussian noise to the input data during training can also help to reduce the impact of outliers or anomalies in the data. Outliers or anomalies are data that are significantly different from the rest of the data and can have a disproportionate influence on the model's training. By adding noise to the input data, the impact of outliers can be reduced, as the added noise can help to smooth out the effects of these data points.

However, it's worth noting that the amount of noise added to the input data should be carefully chosen to balance the benefits of regularization with the potential for degradation in performance due to excessive noise. Adding too much noise can negatively impact the model's performance, as it may make it more difficult for the model to learn the underlying patterns in the data. Conversely, adding too little noise may not provide enough regularization to prevent overfitting.

All in all, adding Gaussian noise to both the training and test data can act as a form of regularization that can help prevent overfitting and improve the generalization performance of machine learning models. By introducing small perturbations to the input data, the model is forced to learn more robust and generalizable features that are less dependent on the specific details of the training data.

We tested the robustness of our model to Gaussian noise added to both train and test data, using the Signal-To-Noise (SNR) ratio, where σ is the variance and is defined as follows:

$$\boxed{\text{SNR} = \frac{\sigma^2_{\text{signal}}}{\sigma^2_{\text{noise}}}}$$

We gradually increase the added noise in the train set, while maintaining the noise in the test set steady (SNR=50), in order to best simulate real-world data. For a given digit, the gradual addition of noise is depicted in figure 4.1, via different SNR values. As we can observe, for high values of SNR, the signal's strength is much larger than the strength of the added noise, resulting in minor distortion of the images. That way, anyone can recognize the number depicted in the images without difficulty. However, as the SNR value approaches 1, the signal's strength is pretty comparable to the strength of the added noise, resulting in very distorted images, where it is difficult not only to recognize the illustrated numbers, but also to even distinguish the digits from the noisy images.

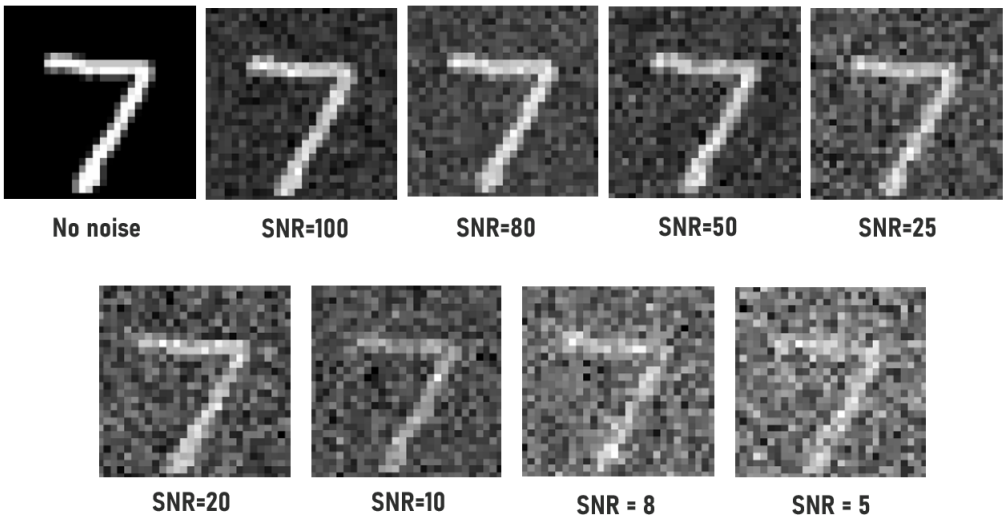

Figure 4.1: Gradual addition of noise for a random digit from MNIST dataset.

4.2.2.2 Adversarial Attacks

Adversarial attacks are a well-established technique for assessing the robustness of deep learning models. Such attacks involve perturbing input data in a manner that is not readily perceivable by humans but leads to the misclassification of the input by the targeted model. While this may appear counterintuitive, there are several reasons why adversarial attacks are vital for evaluating deep learning models:

1. Adversarial attacks can identify security vulnerabilities in deep learning models that malicious actors could exploit. This is especially important in safety-critical applications such as autonomous vehicles and medical diagnosis, where an incorrect prediction or decision could result in dire consequences. By conducting adversarial attacks, researchers can identify weaknesses in deep learning models and develop appropriate countermeasures.

2. Adversarial attacks can serve as a debugging tool for identifying weaknesses in the model's architecture or training process. Through intentionally perturbing input data and observing the model's response, researchers can gain insights into how the model makes decisions and where improvements are needed.

3. Adversarial attacks can be used to enhance the robustness of deep learning models by training them to be more resilient to attacks. By exposing models to a diverse range of adversarial examples during training, researchers can improve the model's ability to generalize and make accurate predictions on new and unseen data.

In our experiments, we have used three adversarial attacks to assess the robustness of our deep learning model:

- The Fast Gradient Sign Method (FGSM) is a popular method that involves perturbing the input data using the sign of the gradient of the model's loss function with respect to the input data. This approach generates adversarial examples that are designed to cause the model to misclassify the input data in a specific manner [28]. The noise is calculated by multiplying the sign of the gradient with respect to the image we want to perturb by a small constant epsilon. As epsilon increases, the model is more likely to be fooled, but the perturbations become easier to identify as well:

$$\text{adv}_{\boldsymbol{x}} = \boldsymbol{x} + \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}}, J(\Theta, \boldsymbol{x}, \boldsymbol{y}))$$

  where $\boldsymbol{x}$ is the original image, ϵ is a very small number, $\nabla_{\boldsymbol{x}}$ is the gradient function, J is the loss function, Θ is the model's weights and $\boldsymbol{y}$ is the true label.

- The Basic Iterative Method (BIM) is an iterative extension of FGSM that involves applying multiple small perturbations to the input data over several iterations [33]. This approach often leads to more effective adversarial examples than FGSM.

- Projected Gradient Descent (PGD) is a more sophisticated attack method that employs multiple iterations of BIM while constraining the perturbations within a specified epsilon range. PGD is among the most powerful adversarial attacks and can create highly effective adversarial examples that can fool deep learning models with high confidence [33].

In order to perform the adversarial attacks, we used foolbox library [41]. The sign of the optimization gradient multiplied by the perturbation magnitude is added to the image in a single iteration. The perturbations used are of magnitudes $\lambda \times 10^{-1}\lambda \in \{2, 8, 16\}$. For the iterative methods BIM and PGD, we set the step size to 1 and the number of iterations to $\min(\lambda + 4, 1.25\lambda)$. The model was trained without any adversarial training on the training set; the adversarial noise was added to the test samples.

4.3 Experimental Setup

In the current section, we provide a detailed description of the experimental agenda used in this thesis. This information allows readers to evaluate the reliability and validity of the study and to replicate the experiments themselves. The experiments palette involves:

- Experimenting with and without the aforementioned regularization layers in the plain CCP, DPNN architecture.
- Testing both Mean Squared Error (MSE) and Cross Entropy Loss (CEL) as loss functions.
- Running the experiments with $\{8, 16\}-$degree deep polynomial networks in order to demonstrate their generalization ability.
- Testing all of the above with and without perturbations (added noise and adversarial attacks).

4.4 Results

4.4.1 Explicit Rank Reduction

We commence our experiments by observing the rank of the weight matrices of two DPNN architectures:

1. The classic CCP architecture, as analyzed in (2.8).
2. The proposed modified CCP architecture with reduced rank in the weight matrices, as shown in subsection 3.2.1.

We have trained a $16-$degree DPNN on MNIST and an $8-$degree DPNN on CIFAR10 with and without explicit rank regularization, using the Cross Entropy Loss metric. As mentioned in 4.1.1, each image on the MNIST dataset is a 28×28 grayscale image, therefore each linear layer on our network has a width of $28 \times 28 \times 1 = 784$. Similarly, as mentioned in 4.1.2, each image is a 32×32 RGB image (3 channels), so each linear layer has a width of $32 \times 32 \times 3 = 3072$. That said, we want to inspect if the weight matrices of the network are full-rank. One simple way to investigate this, is to calculate the mean of the singular values for the set of weight matrices using Singular Value Decomposition (SVD). Then, we could display a stemplot where the vertical bars (stems) represent the magnitude of the mean singular values. If all the mean singular values are non-zero and relatively large, it suggests that the matrices are full-rank or they are close to being full-rank. On the other hand, if some mean singular values are close to zero or significantly smaller than others, it may indicate that the matrices have a lower effective rank or are close to being rank-deficient.

In figures 1 and 2, we can observe that the mean of the singular values shows that each of the weight matrices is close to being full-rank. In figures 3, 4, 5, 6, 7 and 8 we have trained the network with exactly the same parameters, but its architecture follows the guidelines described in 3.2.1. We displayed the same stemplots for the layers $A_{[i]}$, $B_{[i]}$ and their product $B_{[i]}A_{[i]}^T$. The corresponding A and B layers that are used to explicitly reduce the rank of the network's weight matrices, are tested by having a wide range of ranks. More specifically:

- For the MNIST dataset, given that 784 is the maximum rank that a linear layer's weights can have, we tested our network with the ranks:

  $$\{78, 98, 112, 157, 196, 261, 392, 784\}$$

  that correspond to the rank being $\frac{1}{10}, \frac{1}{8}, \frac{1}{7}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}$ and $\frac{1}{1}$ of the full rank.
- For the CIFAR10 dataset, given that 3072 is the maximum rank that a linear layer's weights can have, we tested our network with the ranks:

  $$\{307, 384, 439, 614, 768, 1024, 1536, 3072\}$$

that again correspond to the rank being $\frac{1}{10}, \frac{1}{8}, \frac{1}{7}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}$ and $\frac{1}{1}$ of the full rank.

The same pattern is observed for all the ranks tested. Both implementations (regularized and non-regularized) showed that each individual weight matrix is close to being full-rank. Moving on, the test results regarding whether the regularization through explicit rank control actually benefits the model are of vital importance, as they will prove whether this kind of regularization is applicable and works for DPNNs.

As depicted in Figures 9 and 10, we present the training and testing performance of the model on the MNIST dataset, respectively. It is evident that even when the rank of the model's weight matrices is approximately $\frac{1}{3}$ of the full rank, the model maintains efficiency comparable to the original, non-regularized model. Interestingly, for larger ranks, the model exhibits enhanced performance relative to the non-regularized version. Conversely, for smaller ranks, while the model's efficiency does not match that of the initial model, the discrepancy in scores between it and the non-regularized model remains minimal.

In figure 11, we see the maximum scores (accuracy) our model achieved as the weight matrices' ranks change. We can obviously notice the upwards trend of both the training and testing lines, as the rank increases. However, not only the scores increase, but also the gap between the training and testing lines increases, giving a sign that we cannot ignore, as it indicates that the model is prone to overfitting more easily. The chart 12 depicts this overfitting over ranks trend separately for each configuration, in order to showcase the relation between each regularization configuration and overfitting. The figures 13, 14, 15 and 16 illustrate the same experiments for the CIFAR10 dataset. All in all, from the aforementioned diagrams, we can confidently conclude to the fact that the model performs better and overfits less when regularized with explicit rank reduction on its weight matrices. It is noteworthy that the same conclusions were reached when the model was tested using the Mean Squared Error metric.

4.4.2 Dropout Regularization

We continue our experiments by applying dropout regularization as seen in [35] in the classic CCP architecture, as analyzed in (2.8). We have trained a $16-$degree DPNN on MNIST and on CIFAR10, using the Mean Squared Error metric with the following configurations:

1. Without explicit rank reduction (parameter tensors are full-rank), but with dropout regularization.
2. With explicit rank reduction, but no-dropout regularization.

This study aims to evaluate the impact of dropout regularization on Deep Polynomial Neural Networks (DPNNs), specifically for the MNIST and CIFAR10 datasets. The goal

is to determine whether dropout regularization enhances or hinders the performance of DPNNs. Consider for example a DPNN with a standard CCP architecture that does not employ explicit rank reduction on its parameter tensors. If we introduce dropout between its layers with a 90% Bernoulli success probability, we can then compare this configuration with another DPNN. This second DPNN also has a standard CCP architecture but does not use dropout regularization. Instead, it implements a 10% explicit rank reduction, meaning the output retains 90% of its full size. In essence, both configurations result in roughly the same level of output sparsity. However, the source of this sparsity differs: in the first setup, it arises from explicit rank reduction, while in the second, it is due to the application of dropout to the network's layers.

That said, the experiments agenda is as follows:

1. We employ explicit rank reduction in the parameter tensors of the DPNNs. This approach allows us to systematically decrease the complexity of the network by reducing the full rank of the parameter tensors by a factor ranging from 10% to 40%.

2. We incorporate dropout regularization into the DPNNs. We vary the success probability of the dropout from 90% to 60%.

3. We monitor the performance of DPNNs that have undergone training with dropout regularization and no explicit rank reduction.

4. We compare these dropout regularized architectures with those where the rank has been explicitly reduced but no dropout regularization has been applied, offering a comparative analysis of their performances.

Performance Monitoring Figures 17 - 22 illustrate the training process of DPNNs under various conditions on the MNIST dataset. The training data encompasses a range of Signal-to-Noise Ratio (SNR) values, providing a comprehensive overview of the network's performance under different noise levels. Furthermore, the figures also explore the impact of varying configurations of dropout regularization on the DPNN's training process. Similar to the previous, the figures 23 - 28 illustrate the respective experimental results for the CIFAR10 dataset.

Several observations can be drawn from the diagrams. Primarily, in relation to the MNIST dataset, it is evident that as the Bernoulli probability of success escalates, neither overfitting nor underfitting is discernible. However, this increase is accompanied by a deterioration in the network's performance. Furthermore, a decrease in the Signal-to-Noise Ratio (SNR) value results in the training and testing curves converging towards one another. This convergence signifies a more stable training process and reduced overfitting. Despite these improvements, a slight degradation in performance is observed as well. Lastly, with respect to SNR, it is noteworthy that for values less than 50, the curves begin to diverge, indicating increased instability in the training process. Furthermore, when examining the CIFAR10 dataset, we can largely infer the same conclusions. The only minor

deviation observed from the MNIST dataset is associated with the Bernoulli probability of success. As this probability escalates, the rate at which the network's performance deteriorates also exhibits a corresponding increase.

Architecture Comparison Figures 29 - 36 illustrate a comparative study of the two aforementioned DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. Each chart depicts the accuracy that these DPNN models achieved at various SNR values after 15 epochs of training for a specific Bernoulli probability of success, ranging from 90% to 60%. The networks are trained using the MNIST dataset. Moreover, figures 37 - 44 extend this study with the same model comparisons on the CIFAR10 dataset.

Upon analysis of both the MNIST and CIFAR10 datasets, it is observed that the training and testing curves of both models exhibit a similar trend. This trend is particularly noticeable as the Signal-to-Noise Ratio (SNR) values and the Bernoulli probability of success (θ) decrease. Interestingly, the model incorporating dropout regularization consistently underperforms in comparison to the model with explicit rank reduction. This performance gap widens as θ decreases. The primary reason for this divergence in performance can be attributed to the precipitous decline in the efficacy of the dropout regularized model. In contrast, the model with explicit rank reduction demonstrates only minor performance degradation.

What if we applied dropout regularization to a subset of the model's parameter tensors? This query stems from the observation that, at least for the MNIST and CIFAR10 datasets, dropout regularization does not enhance the performance of DPNNs as explicit rank reduction does. To investigate this further, we conducted an experiment on the MNIST dataset. The results of this experiment are presented in charts 45, 46 and 47.

As anticipated, a discernible pattern emerges in the performance of the models. The model employing explicit rank reduction, without the use of dropout regularization, exhibits superior performance. Following closely is the model that applies dropout regularization to only the latter half of the parameter tensors. The model that applies dropout regularization to all its parameter tensors trails behind. This trend suggests a correlation between the extent of dropout application and model performance. As dropout is applied to fewer parameter tensors, the model's performance appears to converge towards that of the model without any dropout regularization.

4.4.3 Effect of Adversarial Attacks

As analysed thoroughly in 4.2.2, Gaussian noise perturbations and adversarial attacks serve as a litmus test for the robustness of deep learning models, revealing their susceptibility to minute, carefully crafted perturbations in the input data, and thereby driving the development of more resilient models and contributing to the overall advancement of the field.

Our investigation has unveiled the remarkable resilience of Deep Polynomial Neural Networks (DPNNs) to noise attacks. As illustrated in the preceding sections, the performance of the models under test improved as the Signal-to-Noise Ratio (SNR) values ranged from 100 to 50. This trend is clearly visible in most of the presented charts. However, beyond an SNR value of 50, there was a noticeable decline in performance. Despite this drop, it is important to highlight that DPNNs successfully covered a broad spectrum of SNR values that mimic real-world phenomena. Intriguingly, not only did they maintain performance across this range, but they also enhanced it.

Additionally, DPNNs demonstrated respectful resilience against adversarial attacks of the FGSM, BIM, and PGD varieties. As the value of lambda (λ) increased, we observed a small, consistent decrement in performance across all types of attacks, with the variance remaining minimal. To provide a comprehensive understanding of these observations, we present four representative charts:

1. Table 4.1 provides an evaluation of the DPNN architecture utilizing the CP decomposition, devoid of any explicit rank reduction or dropout regularization. The model was initially subjected to testing on the MNIST dataset using Cross Entropy Loss, without any adversarial attacks, as depicted in the top right box. Subsequently, the model was exposed to FGSM, BIM, and PGD adversarial attacks with varying lambda values of 2, 8, and 16.

2. Table 4.2 provides an evaluation of the DPNN architecture with the same configurations as above, but with Mean Squared Error as a loss function.

3. Table 4.3 offers a comparative analysis between two variants of the Deep Polynomial Neural Network (DPNN) architecture: one that incorporates explicit rank reduction and another that utilizes dropout regularization. Both models were evaluated on the MNIST dataset.

4. Table 4.4 illustrates the same comparison, but the models are tested on the CIFAR10 dataset.

Attack	FGSM	BIM	PGD
λ = 2	95.17	95.17	95.17
λ = 8	93.13	93.10	93.10
λ = 16	89.6	89.32	89.32

Clean (no attack)
95.82

Table 4.1: Real-valued network performance on MNIST for FGSM, BIM and PGD attacks with $\lambda \in \{2, 8, 16\}$. We report classification accuracy on test set in all cases. The model was trained using Cross Entropy Loss metric on MNIST dataset.

Attack	FGSM	BIM	PGD
λ = 2	77.81	77.82	77.82
λ = 8	71.97	71.83	71.83
λ = 16	64.27	63.54	63.54

Clean (no attack)
79.3

Table 4.2: Real-valued network performance on MNIST for FGSM, BIM and PGD attacks with $\lambda \in \{2, 8, 16\}$. We report classification accuracy on test set in all cases. The model was trained using Mean Squared Error metric on MNIST dataset.

Attack	FGSM	FGSM	BIM	BIM	PGD	PGD
λ = 2	80.32	65.11	80.32	65.22	80.32	65.01
λ = 8	75.81	60.97	75.77	61.32	75.77	60.85
λ = 16	68.26	55.39	67.88	54.25	67.89	54.70

Clean (no attack)
81.86

Clean (no attack)
66.74

Table 4.3: Real-valued network performance on MNIST for FGSM, BIM and PGD attacks with $\lambda \in \{2, 8, 16\}$. We report classification accuracy on test set in all cases. The models were trained for 15 epochs on MNIST dataset using Mean Squared Error metric and a SNR value of 80%. The results in black represent the performance of the model incorporating a 20% explicit rank reduction, while the data in red depict the performance of the model employing an 80% dropout regularization on its parameter tensors.

Attack	FGSM	FGSM	BIM	BIM	PGD	PGD
λ = 2	34.03	25.95	34.02	26.04	34.02	25.71
λ = 8	28.71	21.17	28.68	21.82	28.69	21.01
λ = 16	21.25	16.39	20.18	15.15	20.65	15.74

Clean (no attack)
34.72

Clean (no attack)
27.25

Table 4.4: Real-valued network performance on MNIST for FGSM, BIM and PGD attacks with $\lambda \in \{2, 8, 16\}$. We report classification accuracy on test set in all cases. The models were trained for 15 epochs on CIFAR10 dataset using Mean Squared Error metric and a SNR value of 100%. The results in black represent the performance of the model incorporating a 10% explicit rank reduction, while the data in red depict the performance of the model employing an 90% dropout regularization on its parameter tensors.

5. CONCLUSION AND FUTURE WORK

In this thesis, we investigate the uncharted territory of regularizing Deep Polynomial Neural Networks (DPNNs). We propose two methods to regularize DPNNs: explicit rank reduction and dropout regularization. Explicit rank reduction reduces the rank of the parameter tensors in DPNNs, while dropout regularization randomly drops out elements of the parameter tensors during training. We conduct experiments on MNIST and CIFAR10 datasets to compare the performance of the two methods under different noise levels and adversarial attacks.

The main findings of the thesis are:

1. DPNNs perform better and overfit less when regularized with explicit rank reduction on their parameter tensors.
2. Dropout regularization does not enhance the performance of DPNNs as explicit rank reduction does; on the contrary, it degrades it.
3. Explicit rank reduction outperforms dropout regularization in all cases, especially when the noise level is high or the dropout regularization variable θ (Bernoulli probability of success) is low.
4. Applying dropout regularization to a subset of the parameter tensors may improve the performance of DPNNs compared to applying it to all parameter tensors.
5. The addition of noise to both the training and test data surprisingly results in enhanced performance. This observation suggests that DPNNs exhibit a high degree of resilience to noise attacks. Furthermore, it indicates their ability to generalize effectively to real-world data, which is often imperfect and noisy.
6. DPNNs demonstrate robustness against three well-known adversarial attacks, specifically FGSM, BIM and PGD. This resilience against multiple adversarial attack strategies further underscores the potential of DPNNs as a reliable tool in the field of deep learning.

This thesis contributes to the literature on DPNNs by providing a comprehensive analysis of two regularization methods and their effects on the robustness and generalization of DPNNs. However, it is important to note that our observations should not be generalized. The complexity and diversity of DPNN configurations, as well as the vast array of datasets (particularly those of higher complexity), necessitate further testing to draw definitive conclusions. Therefore, we suggest some directions for future work, such as exploring other types of regularization methods, investigating the implicit bias of dropout regularization paired with the architecture of DPNNs, and applying DPNNs to other domains and more complex tasks.

ABBREVIATIONS - ACRONYMS

BIM	Basic Iterative Method
CANDECOMP	Canonical Decomposition
CCP	Coupled Canonical-Polyadic
CEL	Cross Entropy Loss
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CP	Canonical-Polyadic
DCNN	Deep Convolutional Neural Network
DPNN	Deep Polynomial Neural Network
FGSM	Fast Gradient Sign Method
GMDH	Group Method of Data Handling
MNIST	Modified National Institute of Standards and Technology database
MSE	Mean Squared Error
NCP	Nested Coupled Canonical-Polyadic
NLP	Natural Language Processing
PARAFAC	Parallel Factor Analysis
PCA	Principal Component Analysis
PGD	Projected Gradient Descent
PNN	Polynomial Neural Network
RGB	Red, Green, Blue
SGD	Stochastic Gradient Descent
SNR	Signal-To-Noise Ratio
SVD	Singular Value Decomposition
ΒΠΝΔ	Βαθιά Πολυωνυμικά Νευρωνικά Δίκτυα

ANNEX

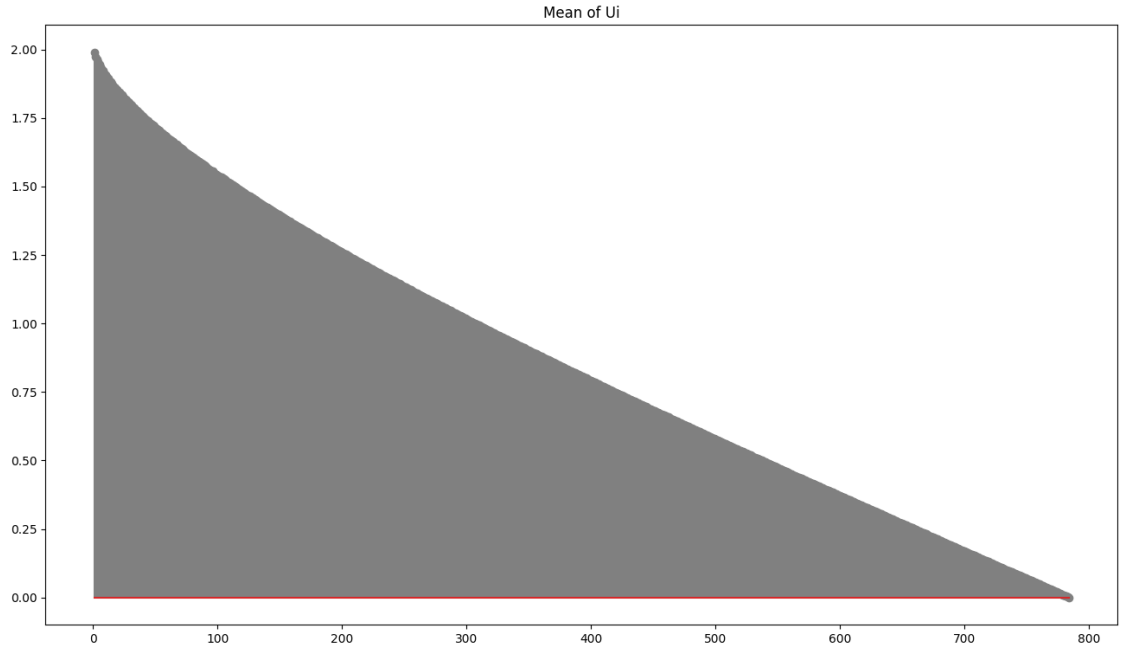

Figure 1: Stemplot illustrating the mean of the singular values for the set of weight matrices in a 16-degree DPNN without regularization, using the Singular Value Decomposition (SVD). The network is trained on the MNIST dataset.

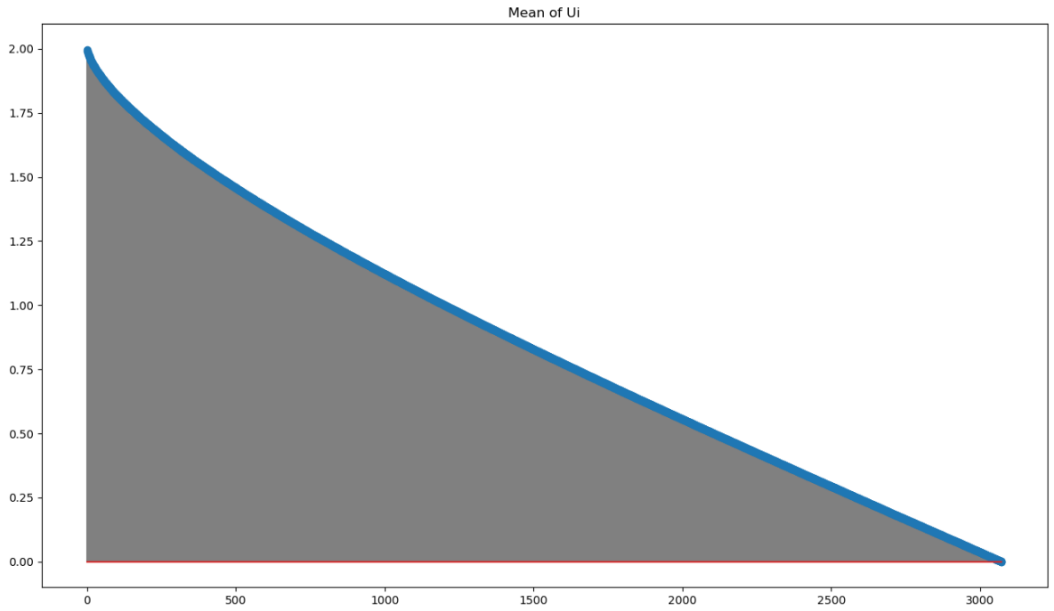

Figure 2: Stemplot illustrating the mean of the singular values for the set of weight matrices in a 8-degree DPNN without regularization, using the Singular Value Decomposition (SVD). The network is trained on the CIFAR10 dataset.

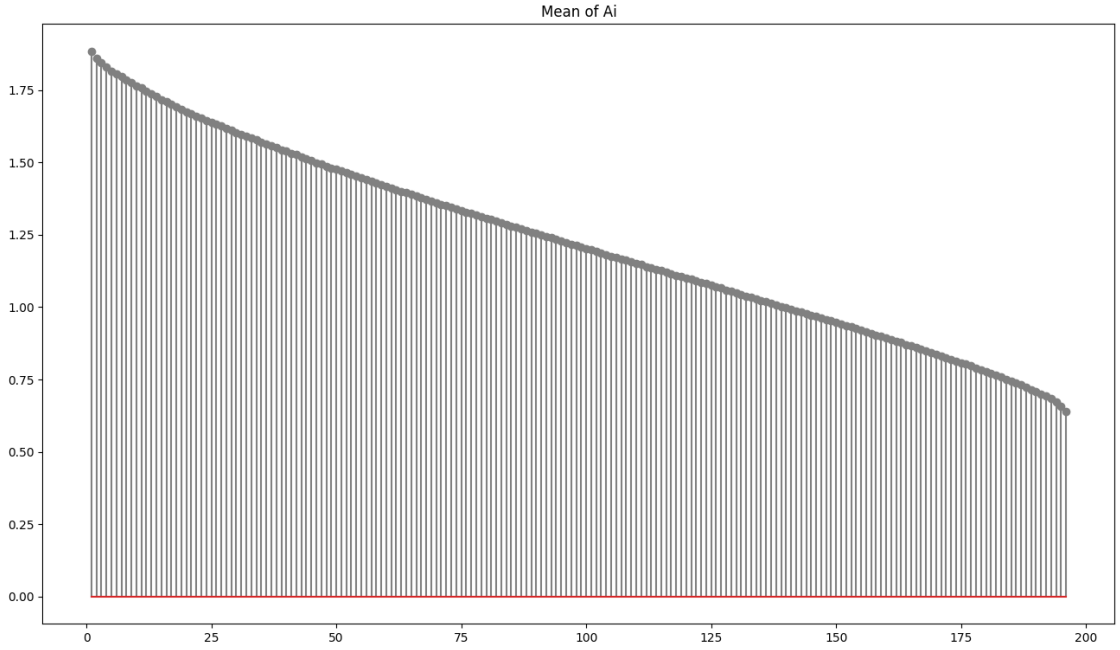

Figure 3: Stemplot illustrating the mean of the singular values for the first set of weight matrices ($A_{[i]}$) in a 16-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the MNIST dataset.

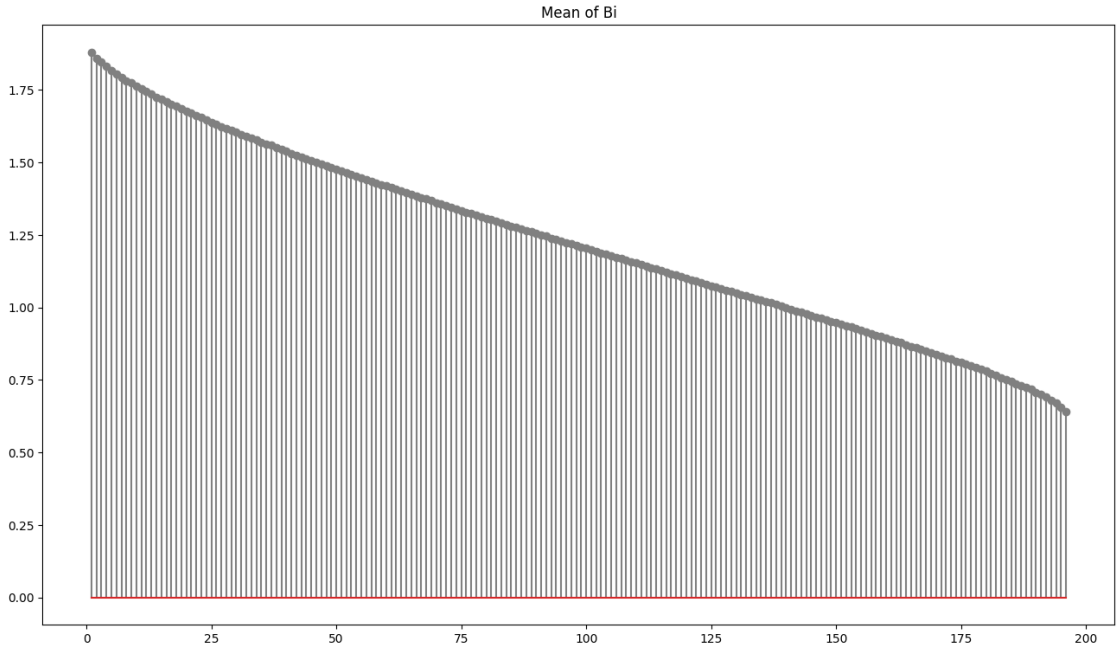

Figure 4: Stemplot illustrating the mean of the singular values for the second set of weight matrices ($B_{[i]}$) in a 16-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the MNIST dataset.

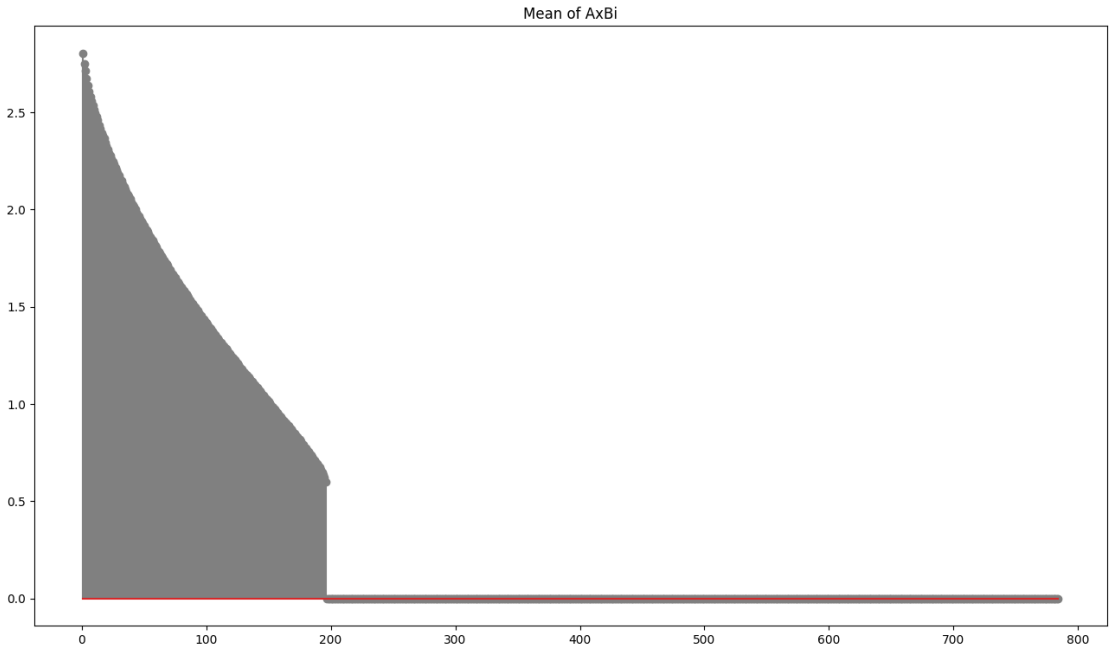

Figure 5: Stemplot illustrating the mean of the singular values for the entire set of weight matrices ($B_{[i]}A_{[i]}^T$) in a 16-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the MNIST dataset.

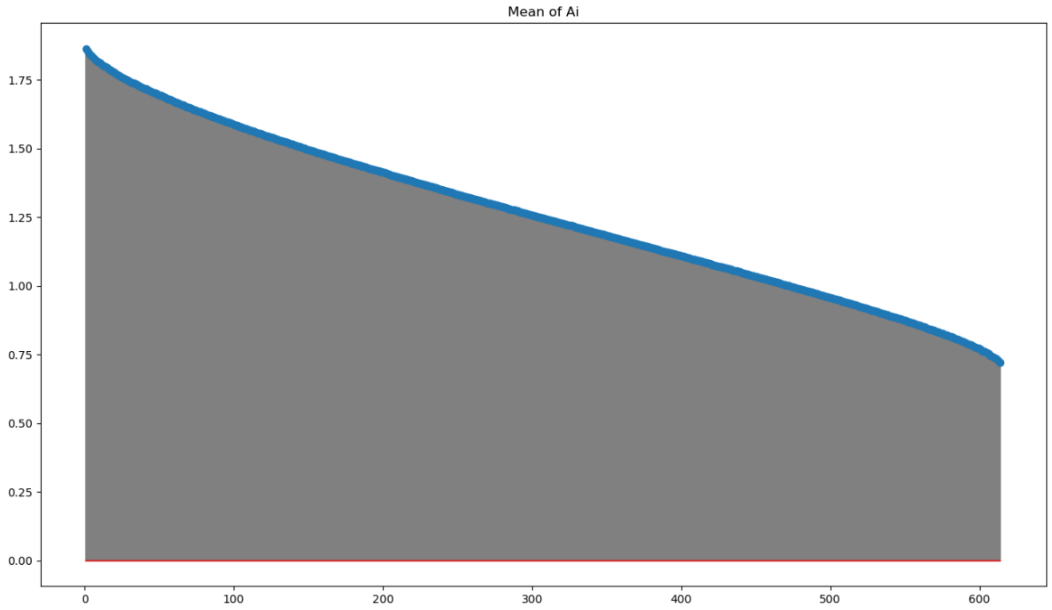

Figure 6: Stemplot illustrating the mean of the singular values for the first set of weight matrices ($A_{[i]}$) in a 8-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the CIFAR10 dataset.

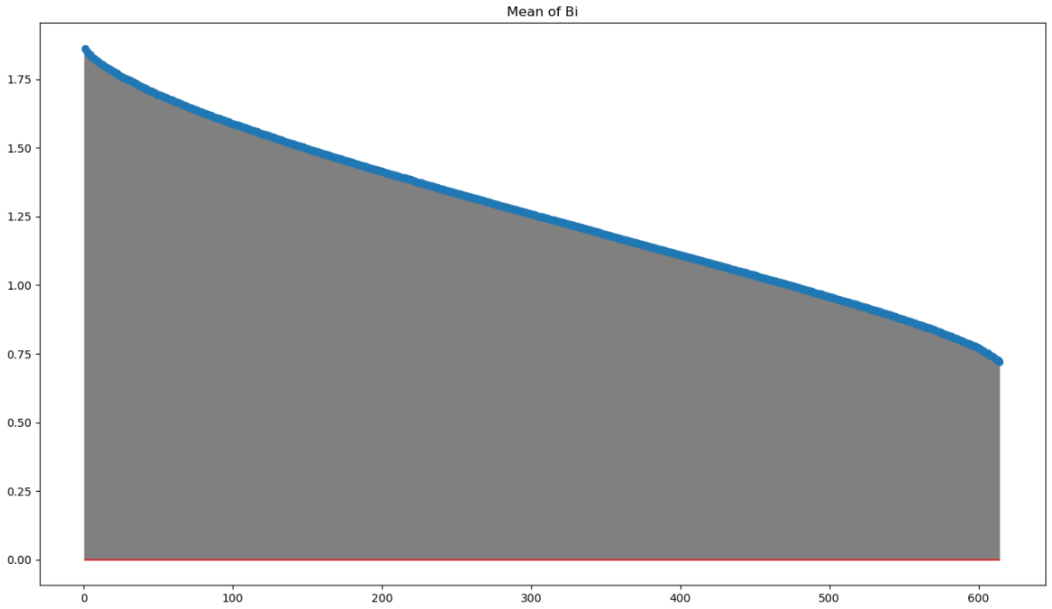

Figure 7: Stemplot illustrating the mean of the singular values for the second set of weight matrices ($B_{[i]}$) in a 8-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the CIFAR10 dataset.

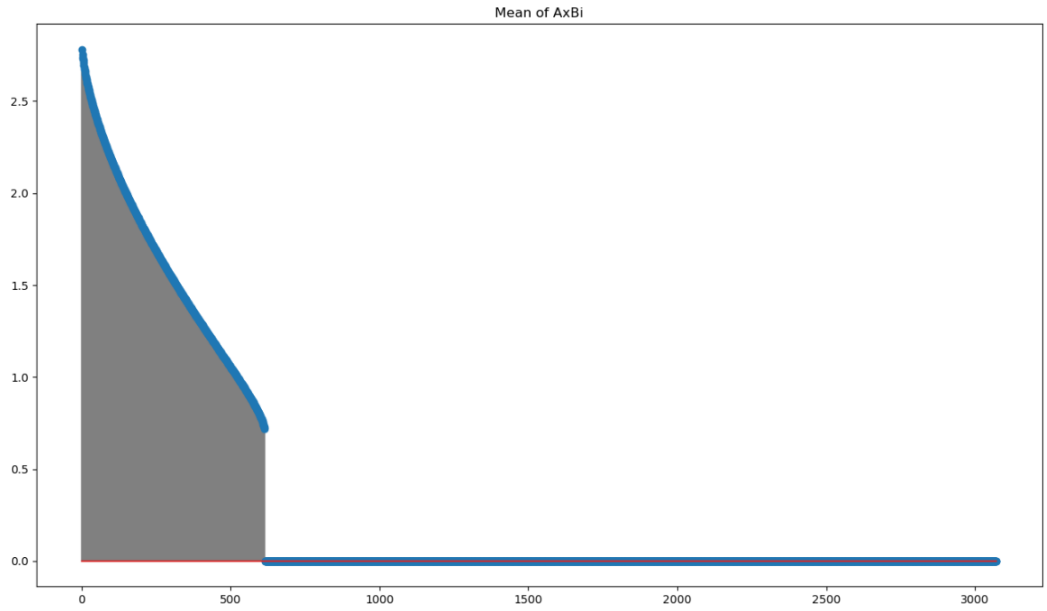

Figure 8: Stemplot illustrating the mean of the singular values for the entire set of weight matrices ($B_{[i]}A_{[i]}^T$) in a 8-degree DPNN with explicit rank regularization, using the Singular Value Decomposition (SVD). The network is trained on the CIFAR10 dataset.

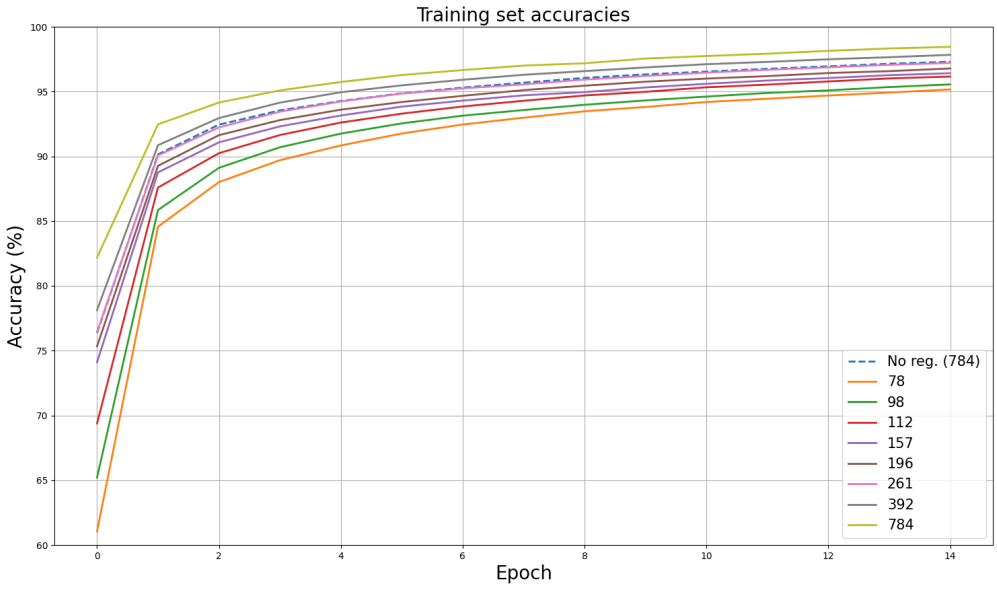

Figure 9: Accuracies on the training set for 15 epochs and different numbers of ranks of the weight matrices. A $16-$degree DPNN model is trained on MNIST dataset with and without regularization using Cross Entropy Loss. The dashed line corresponds to the trial without regularization. The continuous colored lines correspond to the trials with regularization, which is applied via explicit rank reduction.

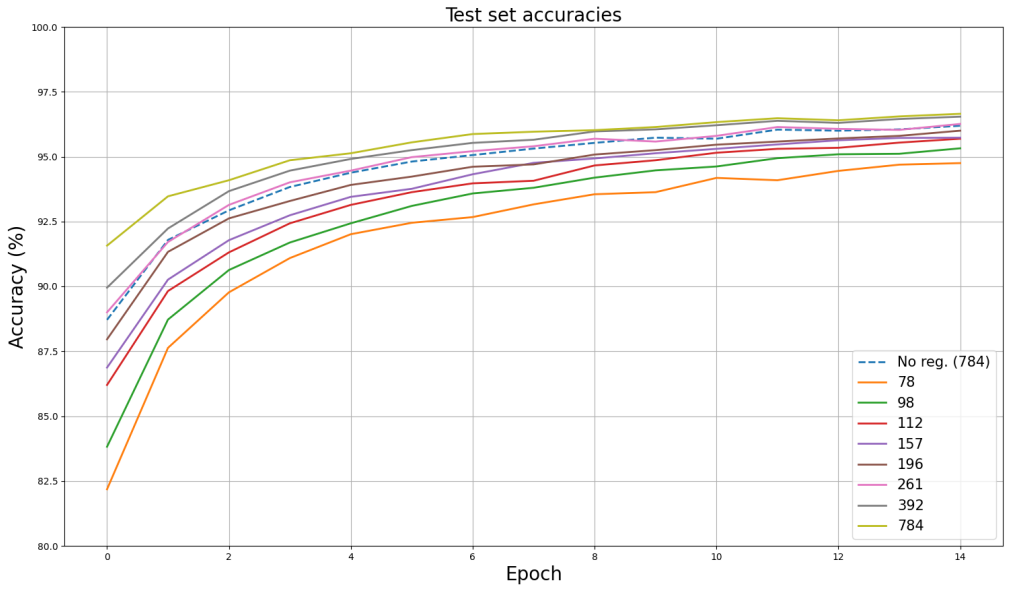

Figure 10: Accuracies on the testing set for 15 epochs and different numbers of ranks of the weight matrices. A $16-$degree DPNN model is trained on MNIST dataset with and without regularization using Cross Entropy Loss. The dashed line corresponds to the trial without regularization. The continuous colored lines correspond to the trials with regularization, which is applied via explicit rank reduction.

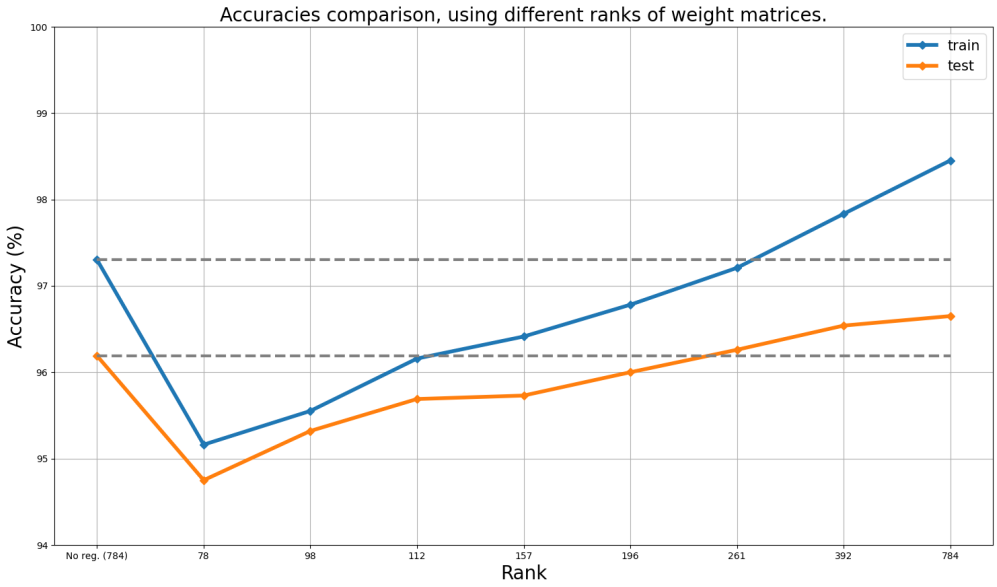

Figure 11: Maximum training and testing accuracies for different numbers of rank of the weight matrices. A $16-$degree DPNN model is trained on MNIST dataset with and without regularization using Cross Entropy Loss. The dashed lines correspond to the model's performance on the trial without regularization. The regularization is applied via explicit rank reduction.

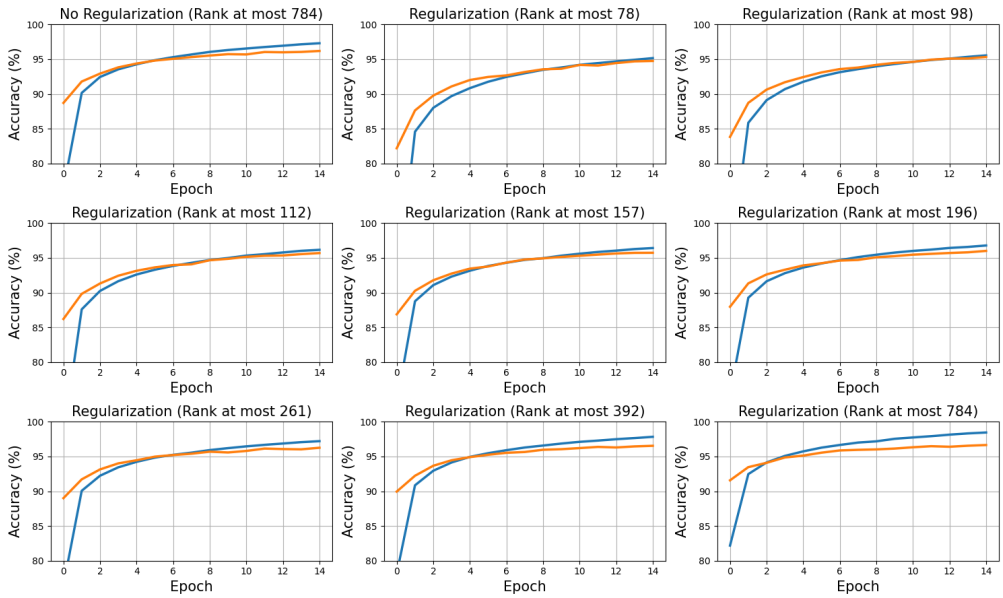

Figure 12: Comparison between the performance of the model with and without regularization on MNIST. The regularization is applied via explicit rank reduction. The gap between the training and the testing lines is analogous to the possibility of overfitting.

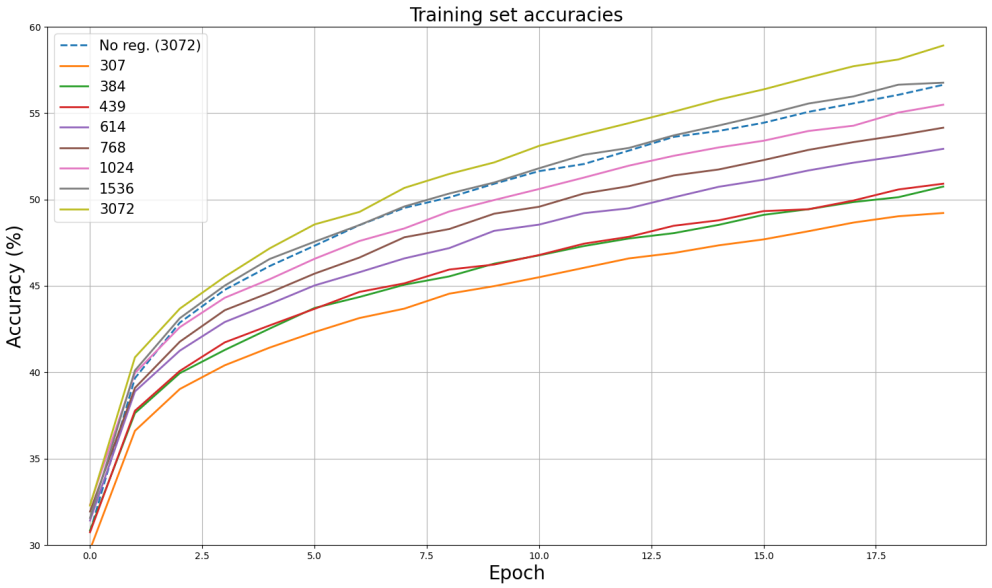

Figure 13: Accuracies on the training set for 20 epochs and different numbers of ranks of the weight matrices. A $8-$degree DPNN model is trained on CIFAR10 dataset with and without regularization using Cross Entropy Loss. The dashed line corresponds to the trial without regularization. The continuous colored lines correspond to the trials with regularization, which is applied via explicit rank reduction.

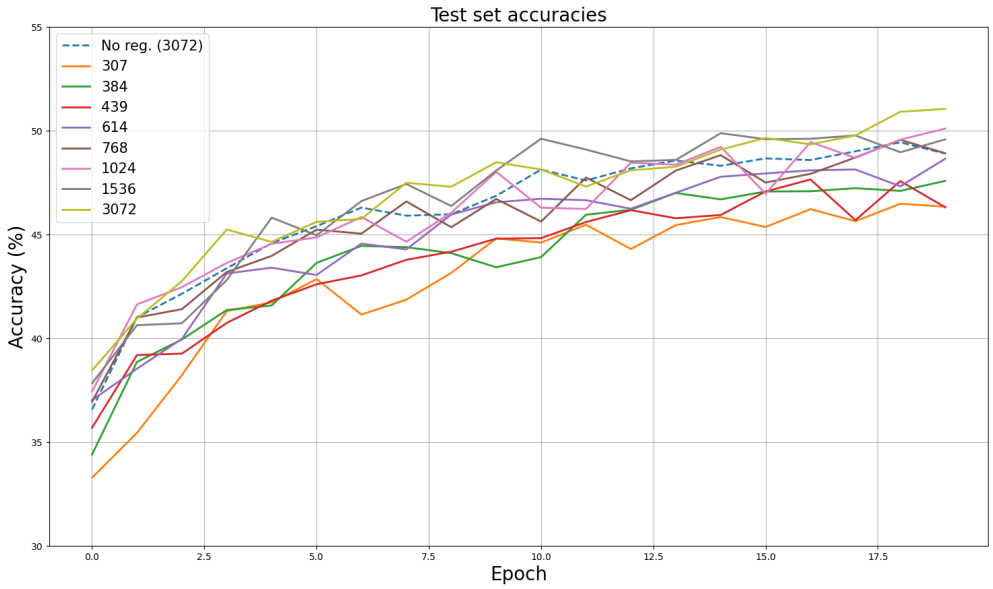

Figure 14: Accuracies on the testing set for 20 epochs and different numbers of ranks of the weight matrices. A $8-$degree DPNN model is trained on CIFAR10 dataset with and without regularization using Cross Entropy Loss. The dashed line corresponds to the trial without regularization. The continuous colored lines correspond to the trials with regularization, which is applied via explicit rank reduction.

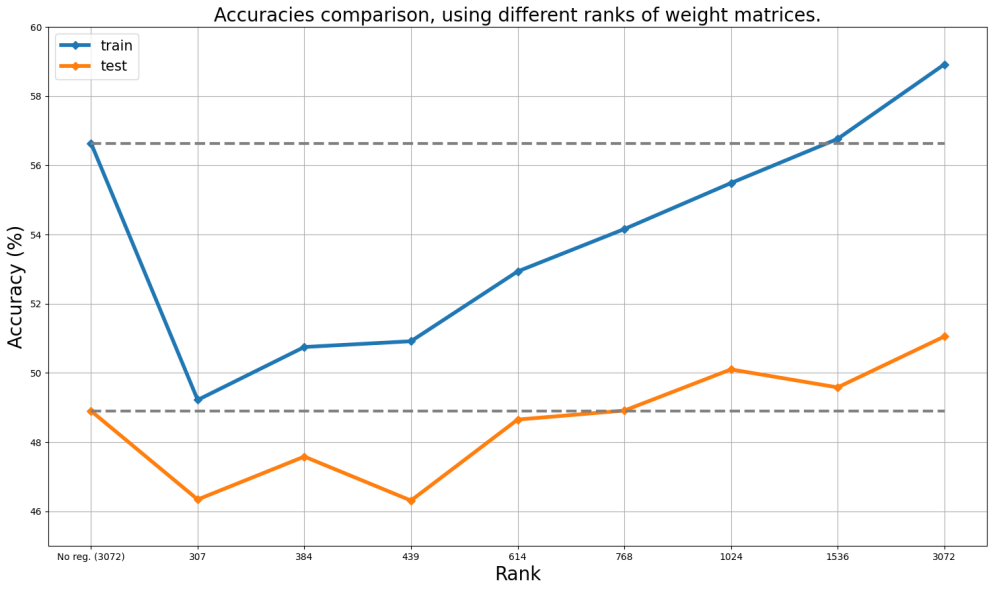

Figure 15: Maximum training and testing accuracies for different numbers of rank of the weight matrices. A $8-$degree DPNN model is trained on CIFAR10 dataset with and without regularization using Cross Entropy Loss. The dashed lines correspond to the model's performance on the trial without regularization. The regularization is applied via explicit rank reduction.

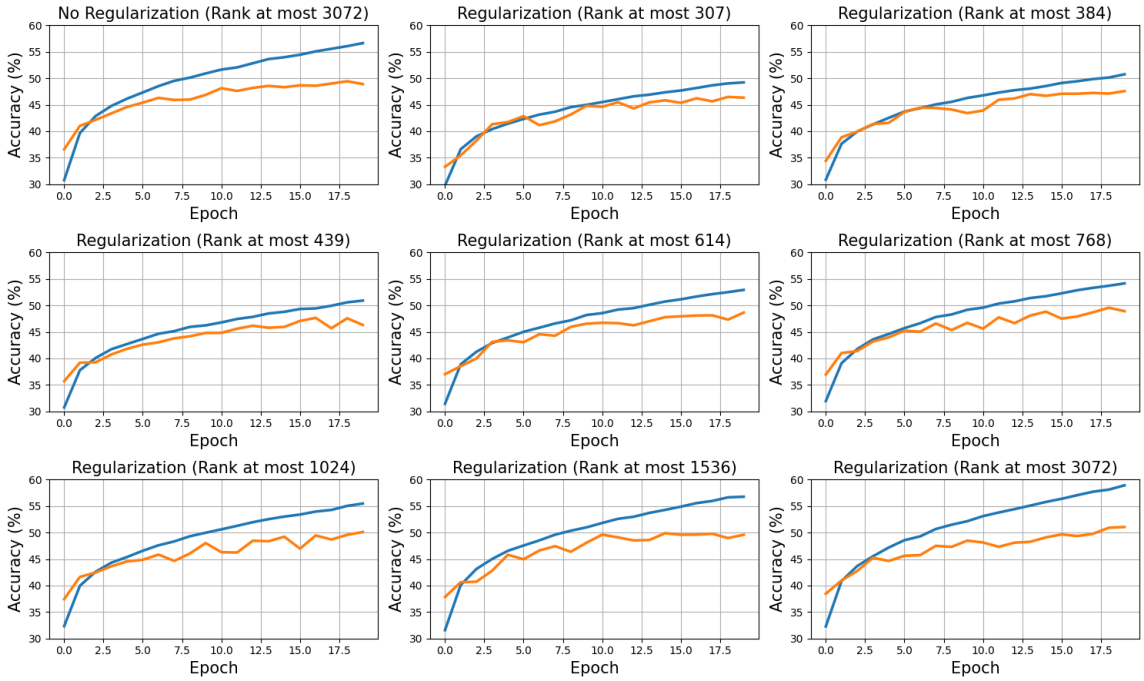

Figure 16: Comparison between the performance of the model with and without regularization on CIFAR10. The regularization is applied via explicit rank reduction. The gap between the training and the testing lines is analogous to the possibility of overfitting.

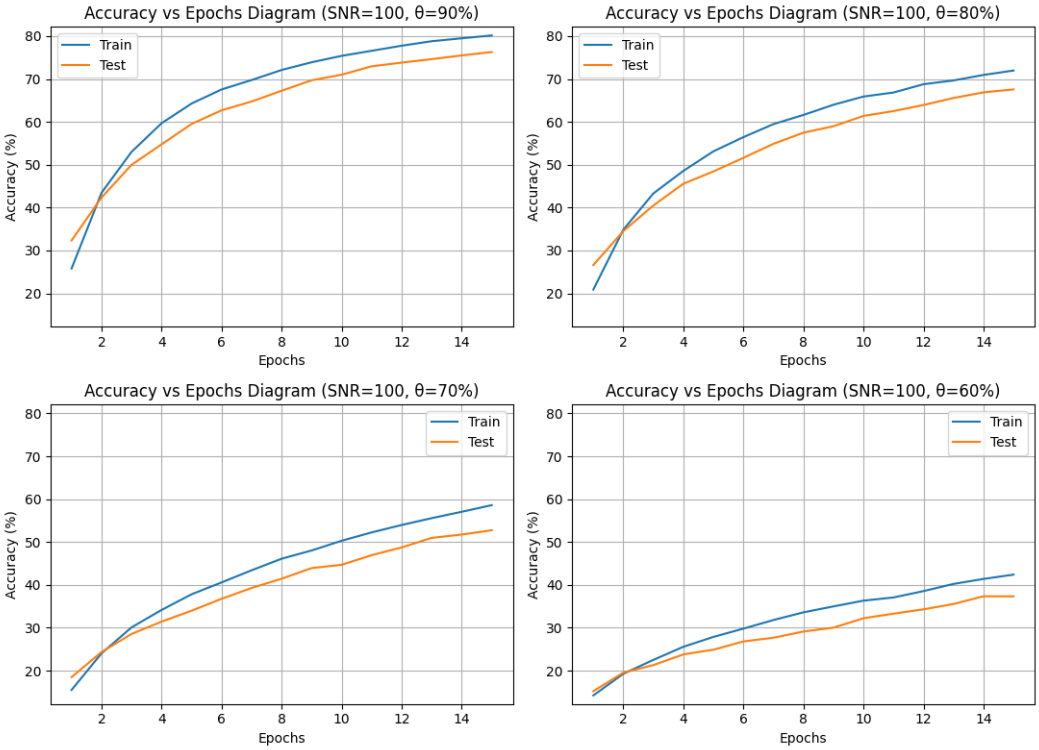

Figure 17: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 100 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

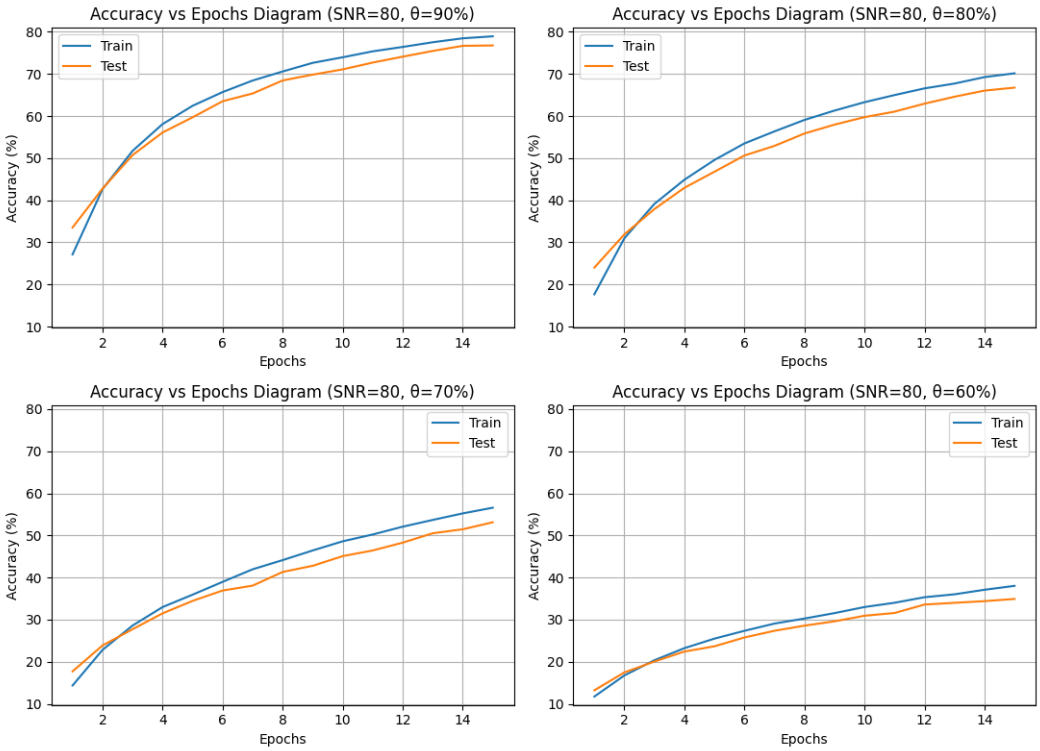

Figure 18: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 80 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

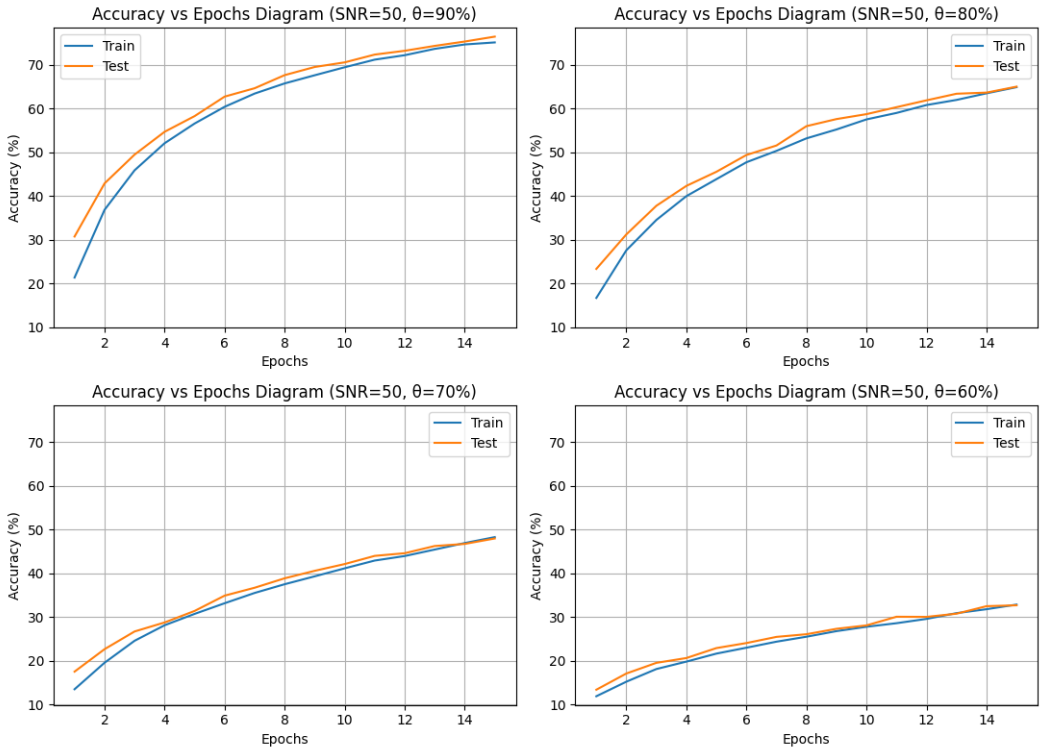

Figure 19: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 50 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

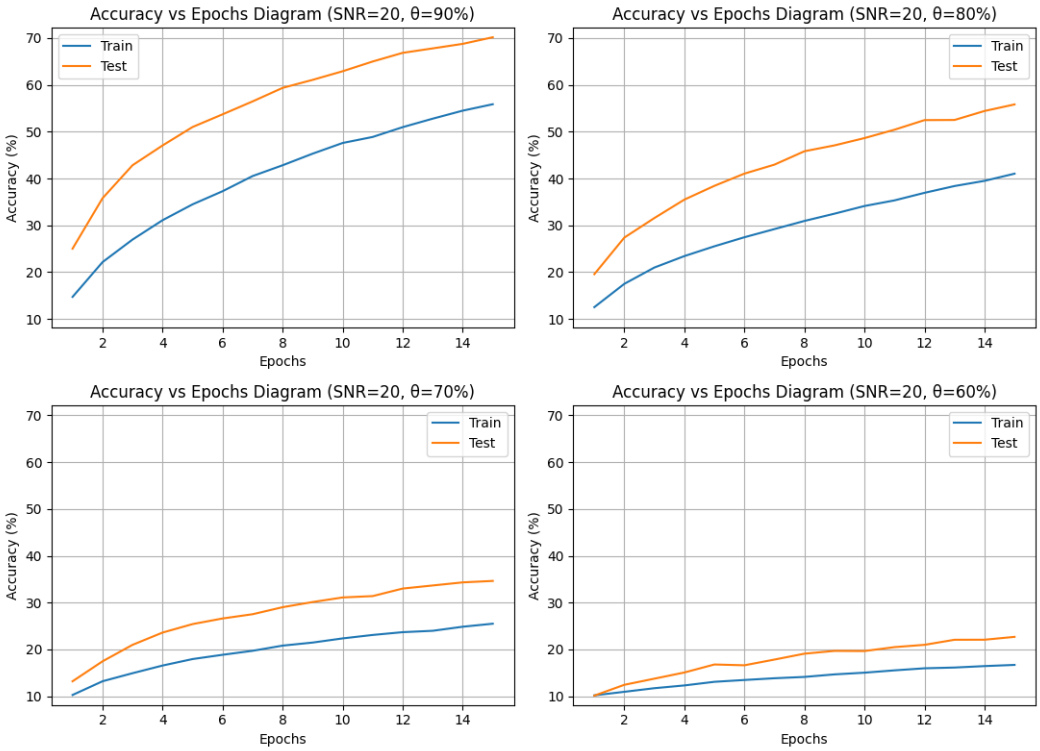

Figure 20: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 20 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

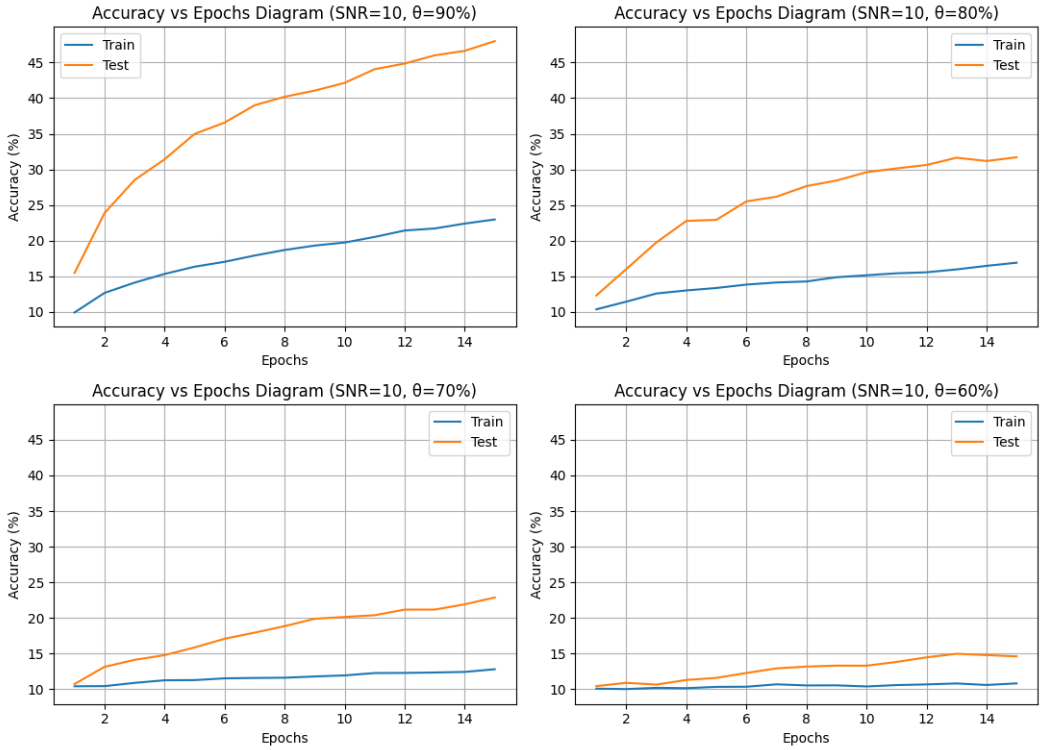

Figure 21: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 10 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

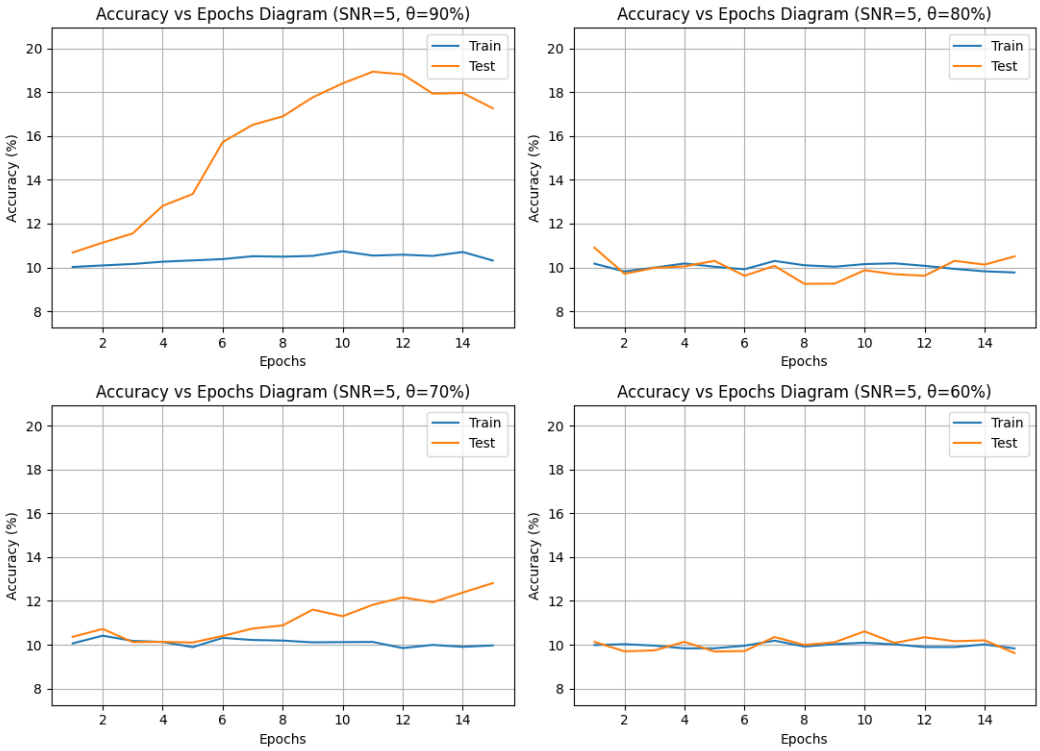

Figure 22: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 5 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the MNIST dataset with dropout regularization, without explicit rank reduction.

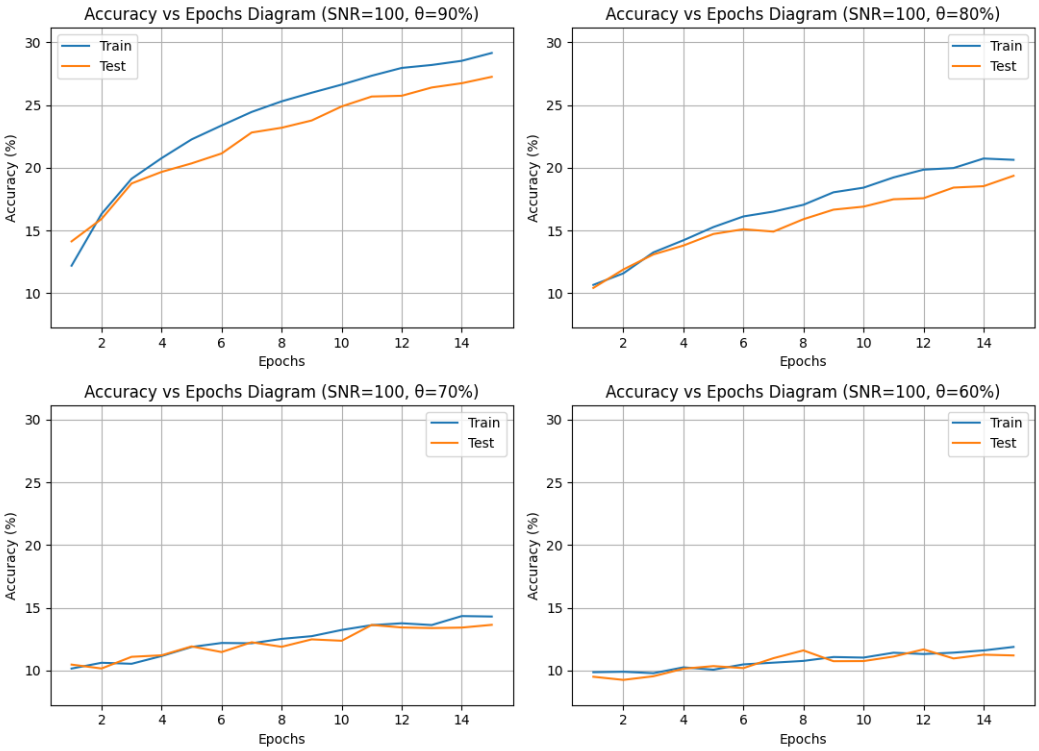

Figure 23: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 100 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

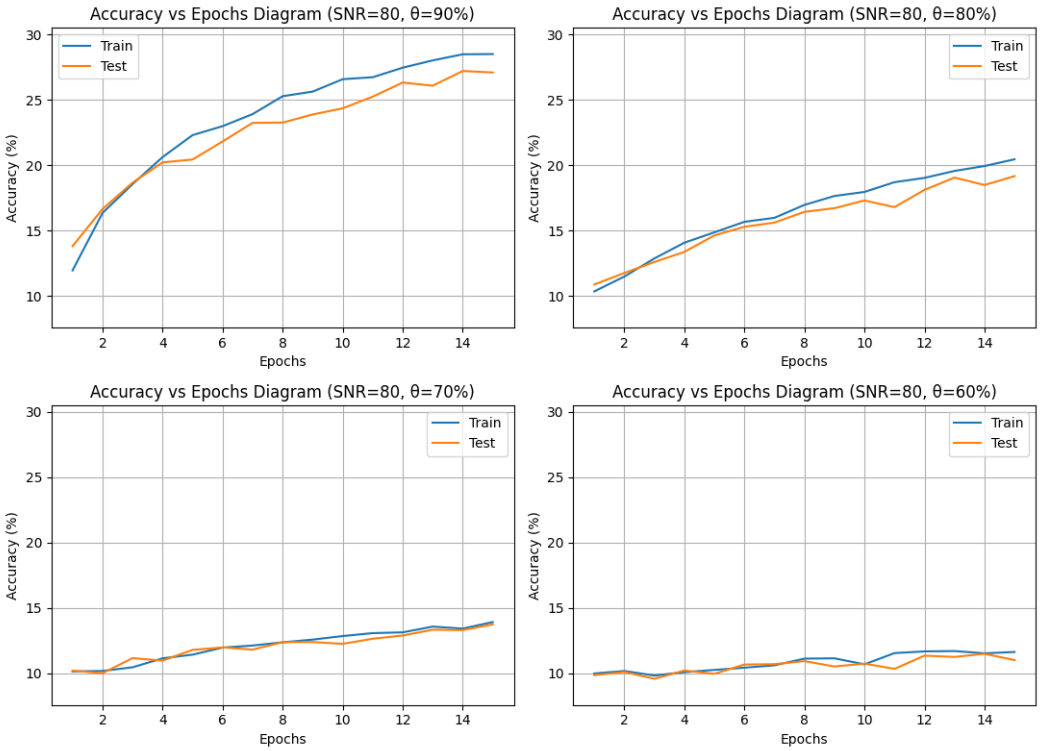

Figure 24: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 80 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

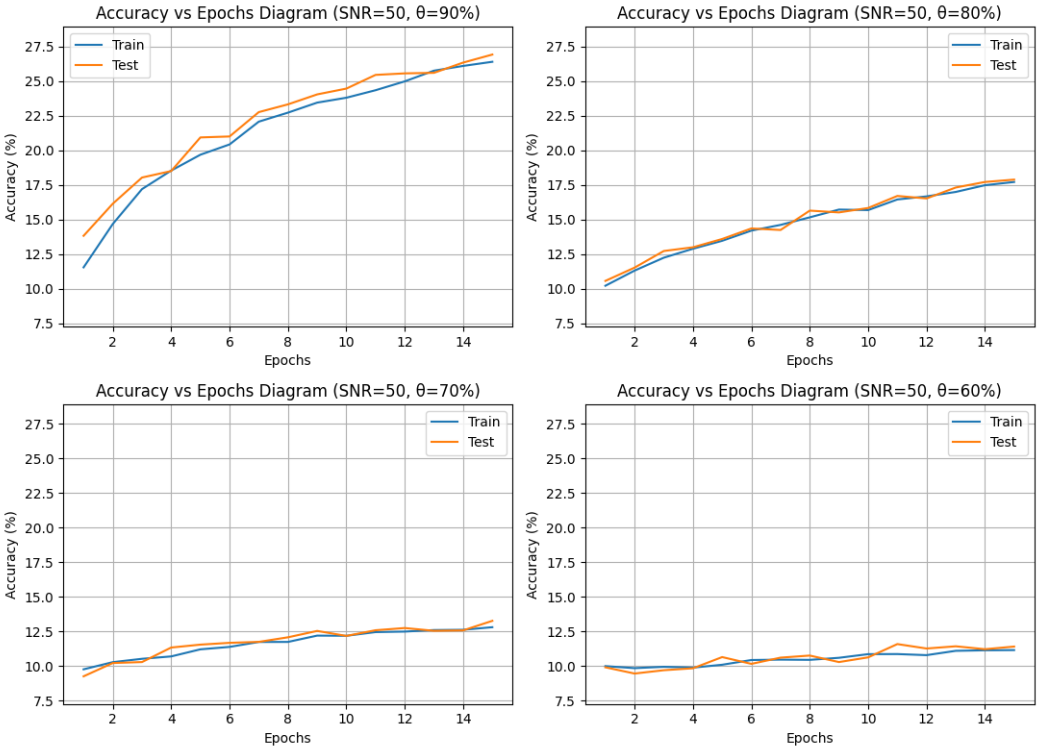

Figure 25: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 50 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

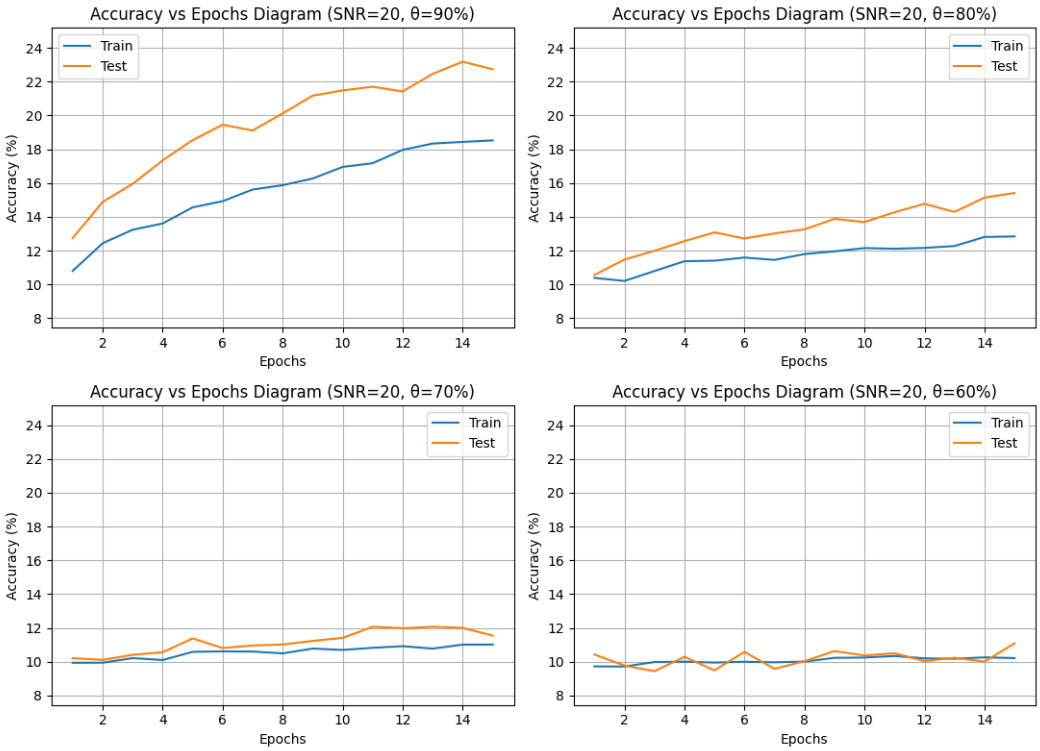

Figure 26: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 20 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

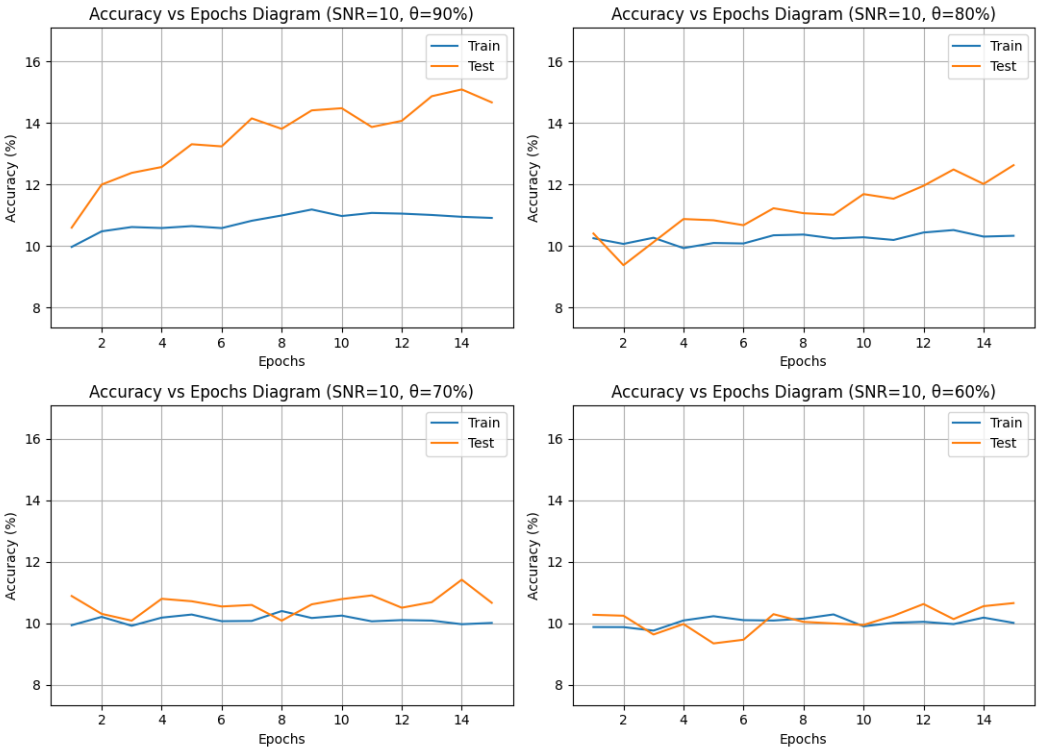

Figure 27: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 10 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

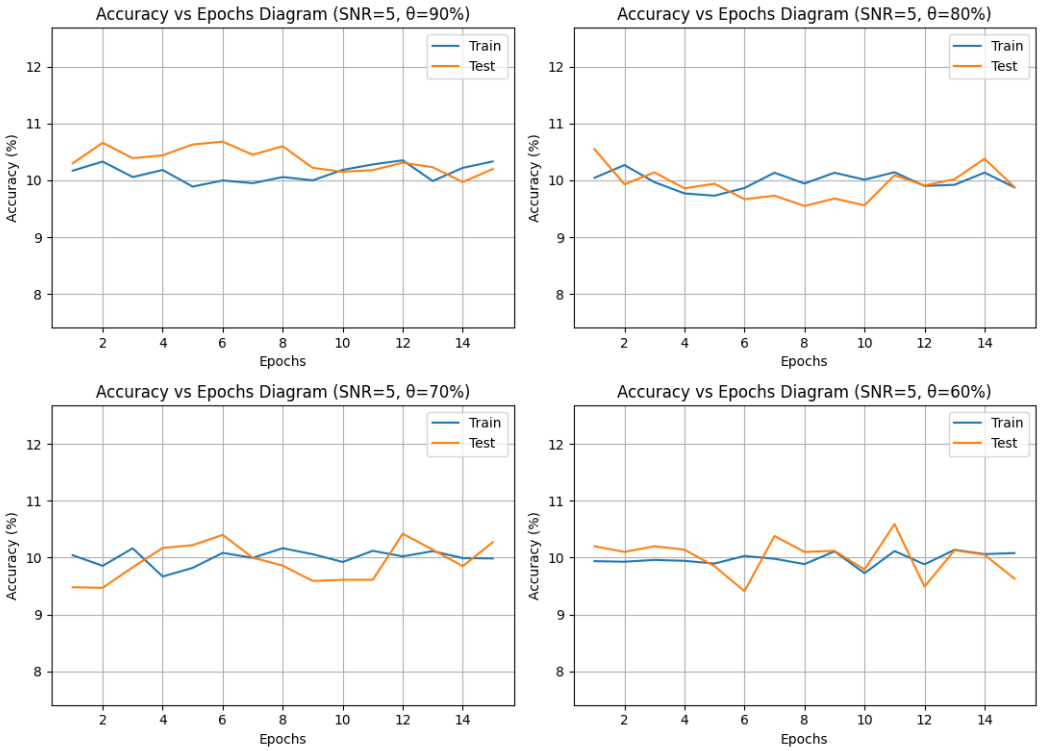

Figure 28: Accuracy with respect to epochs diagram. The SNR value for the training data is equal to 5 and the Bernoulli probability of success varies from 90% to 60%. The network is trained on the CIFAR10 dataset with dropout regularization, without explicit rank reduction.

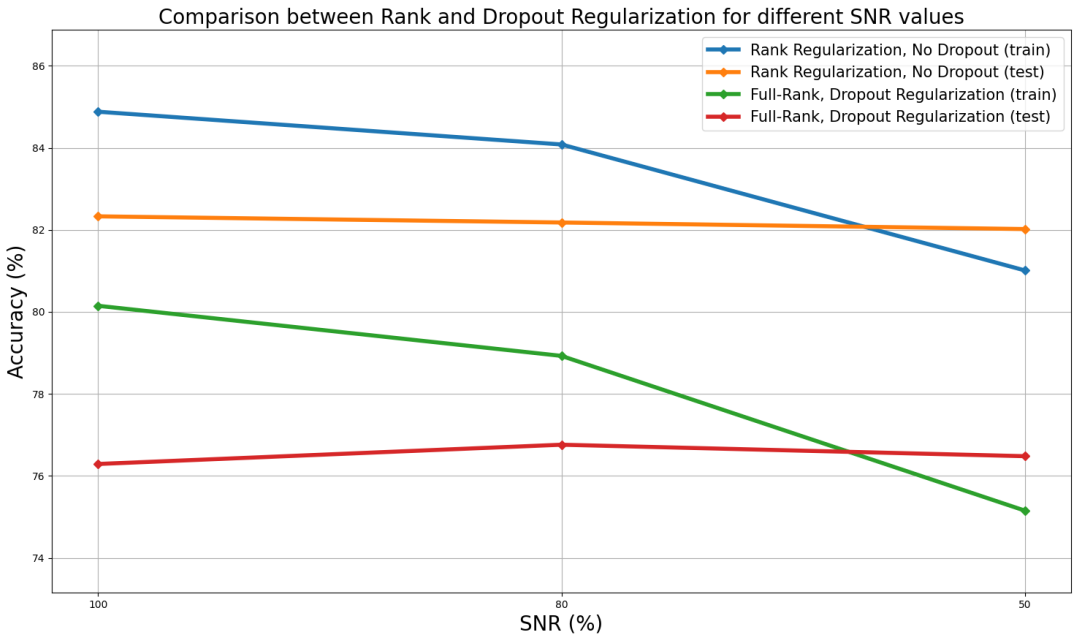

Figure 29: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 90% is utilized.

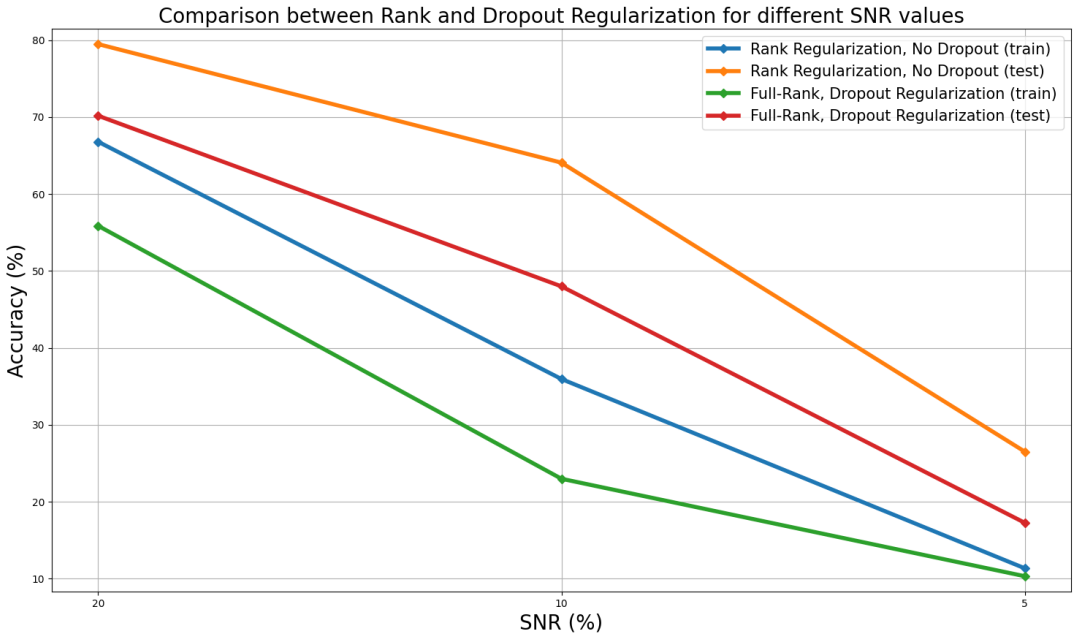

Figure 30: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 90% is utilized.

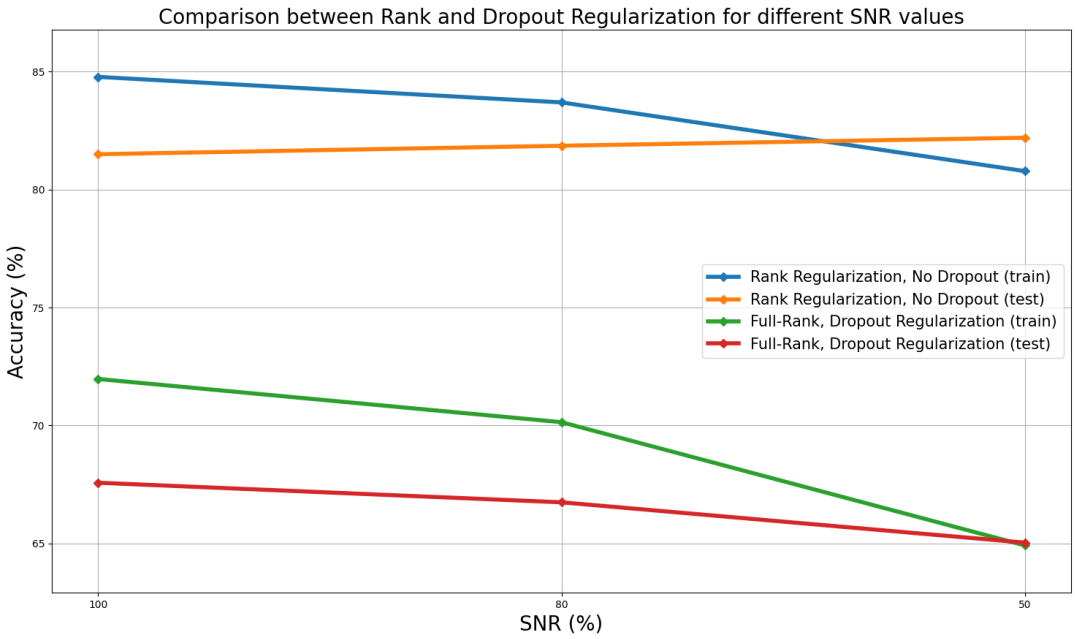

Figure 31: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 80% is utilized.

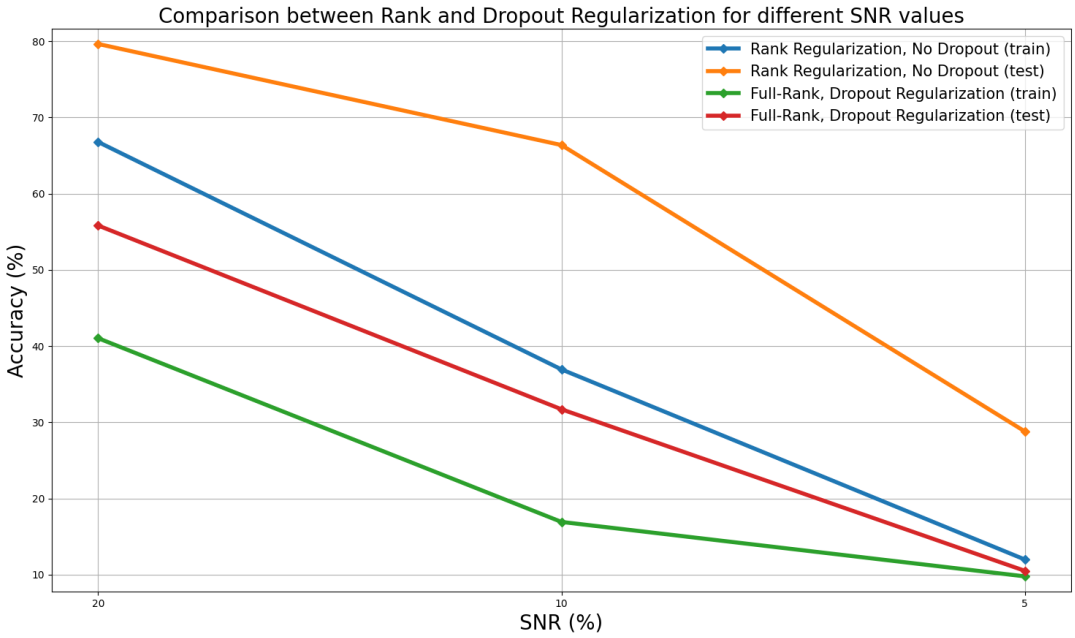

Figure 32: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 80% is utilized.

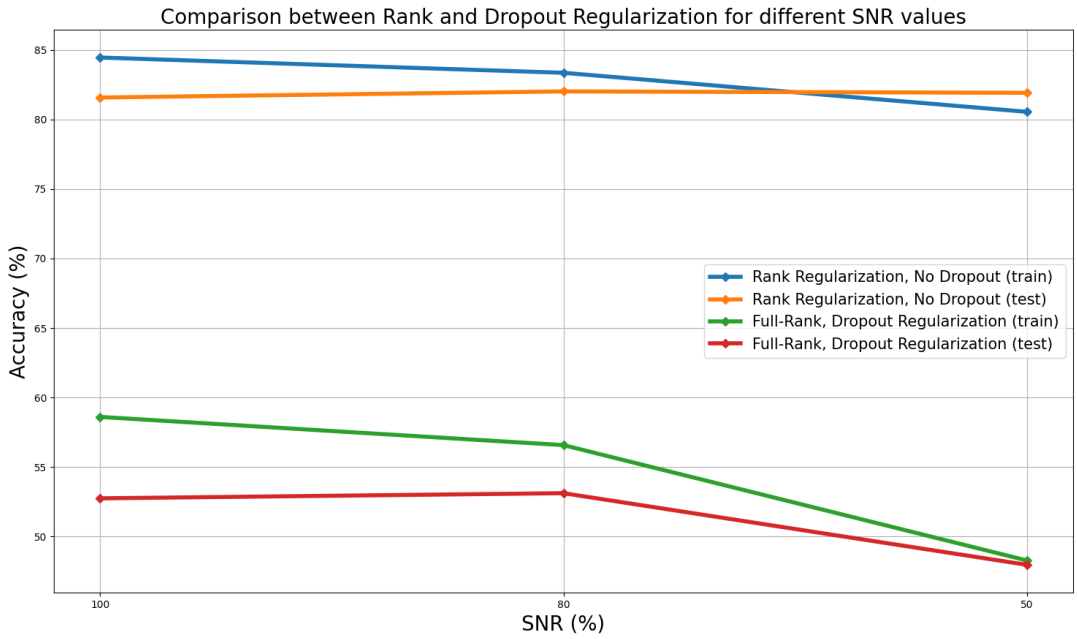

Figure 33: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 70% is utilized.

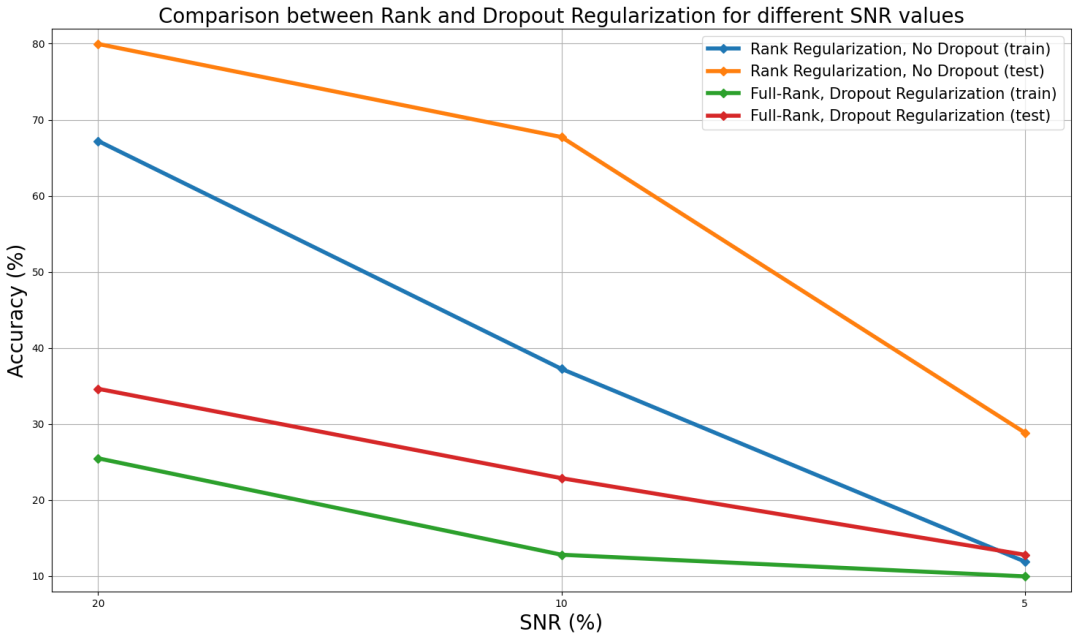

Figure 34: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 70% is utilized.

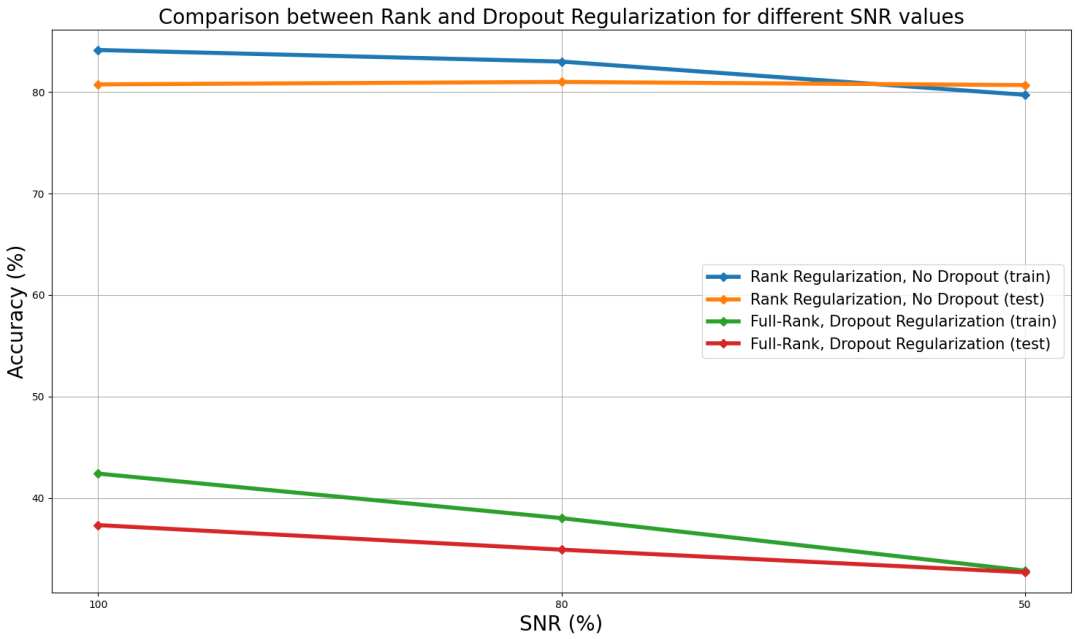

Figure 35: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 60% is utilized.

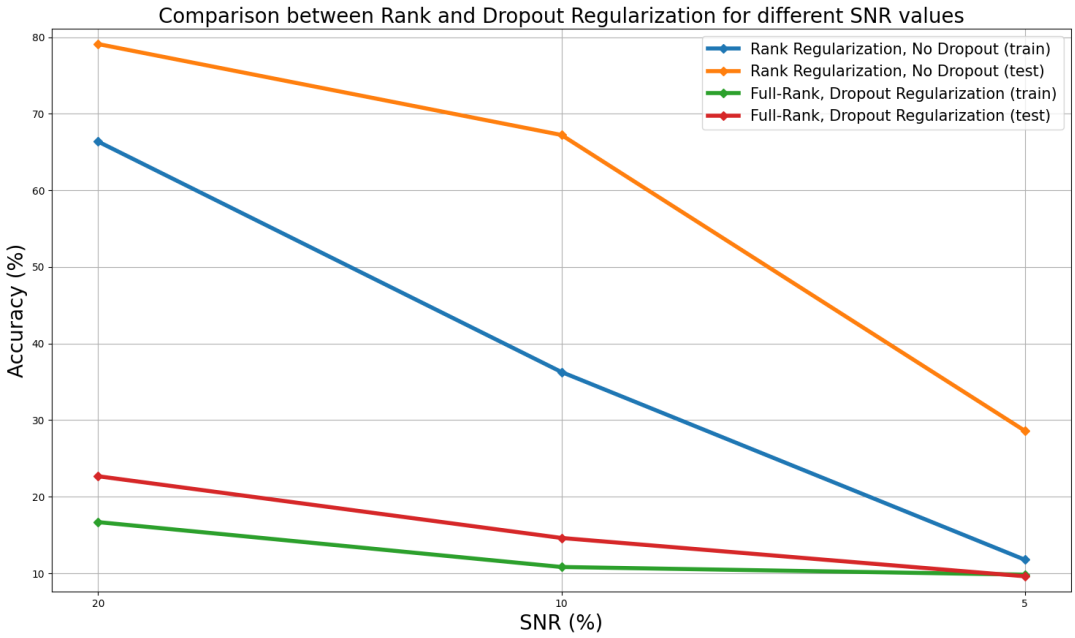

Figure 36: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the MNIST dataset. For the dropout regularized model, a Bernoulli probability of success set at 60% is utilized.

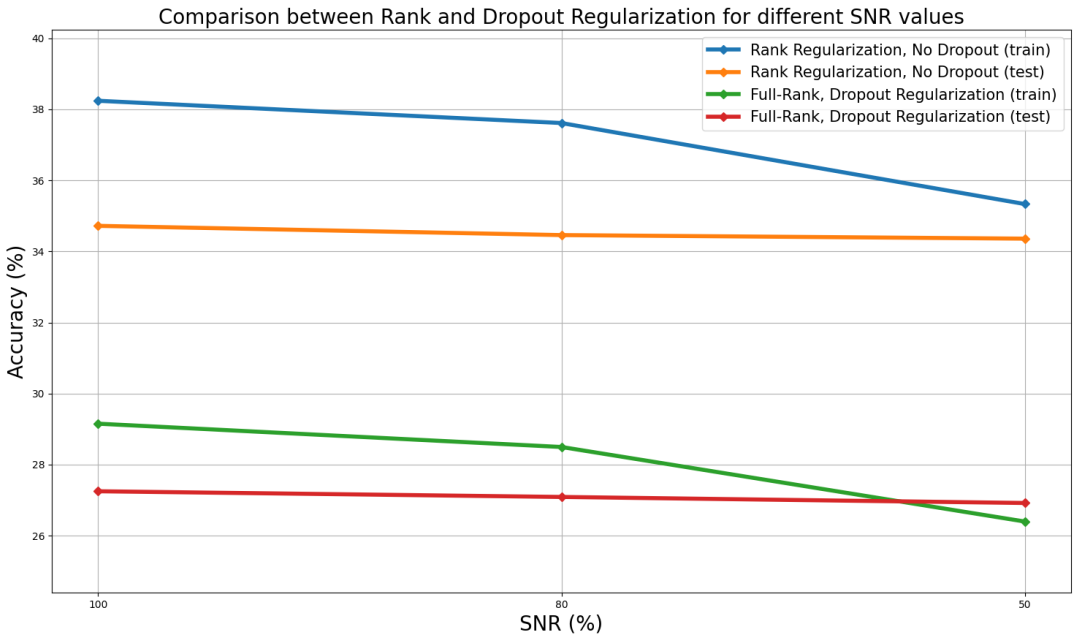

Figure 37: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 90% is utilized.

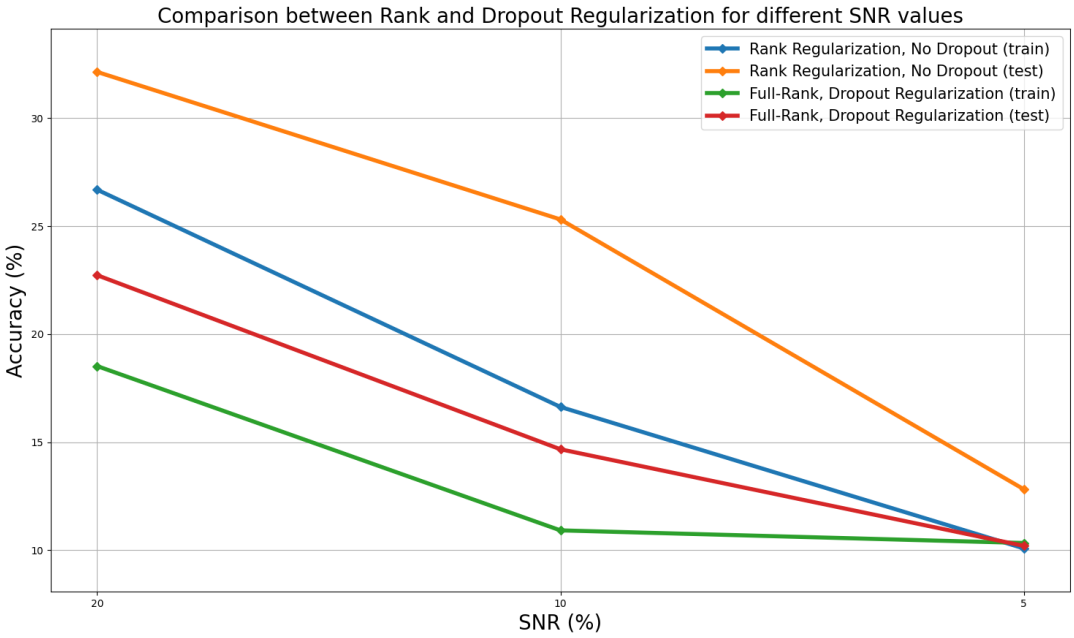

Figure 38: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 90% is utilized.

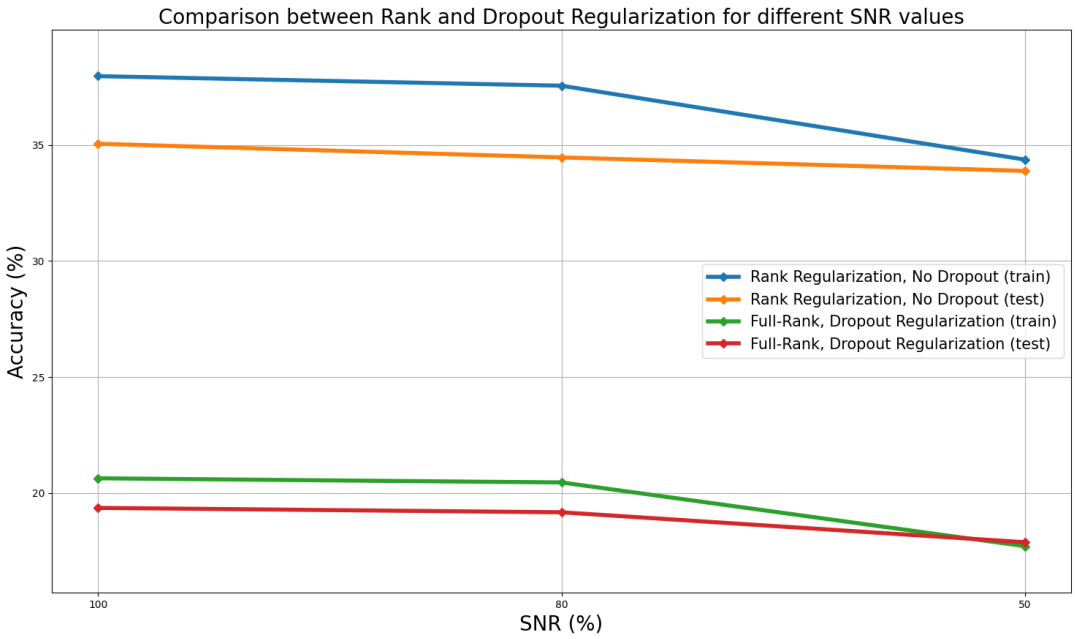

Figure 39: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 80% is utilized.

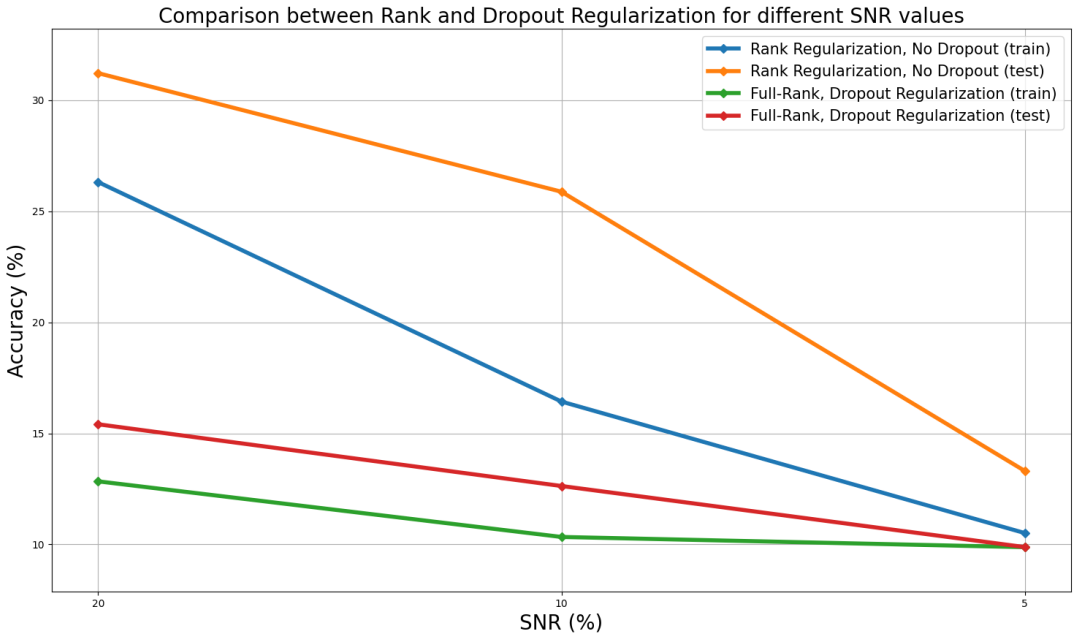

Figure 40: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 80% is utilized.

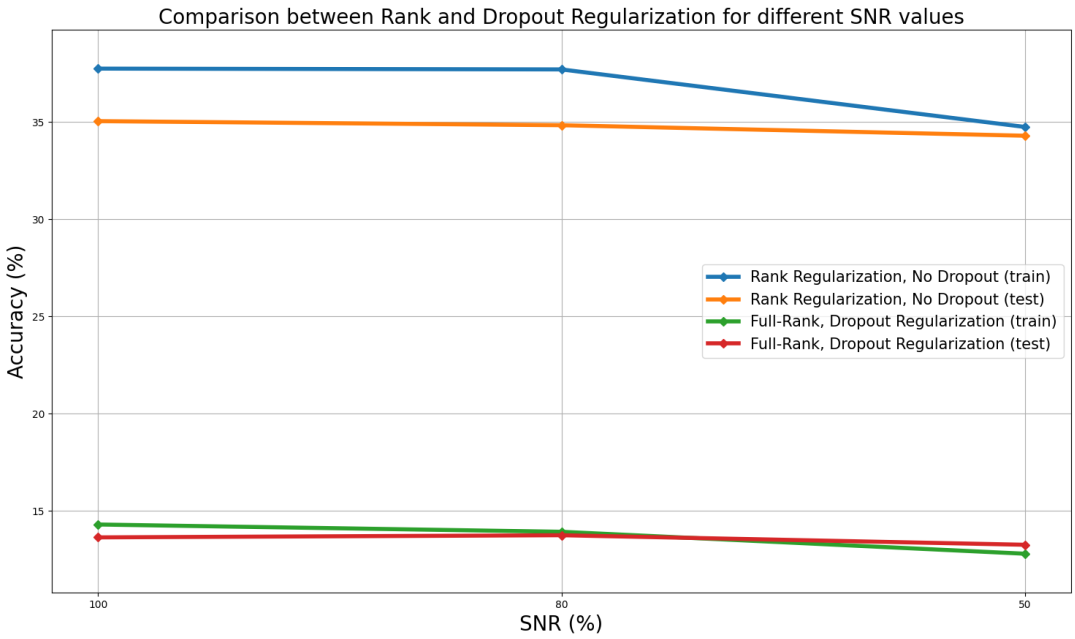

Figure 41: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 70% is utilized.

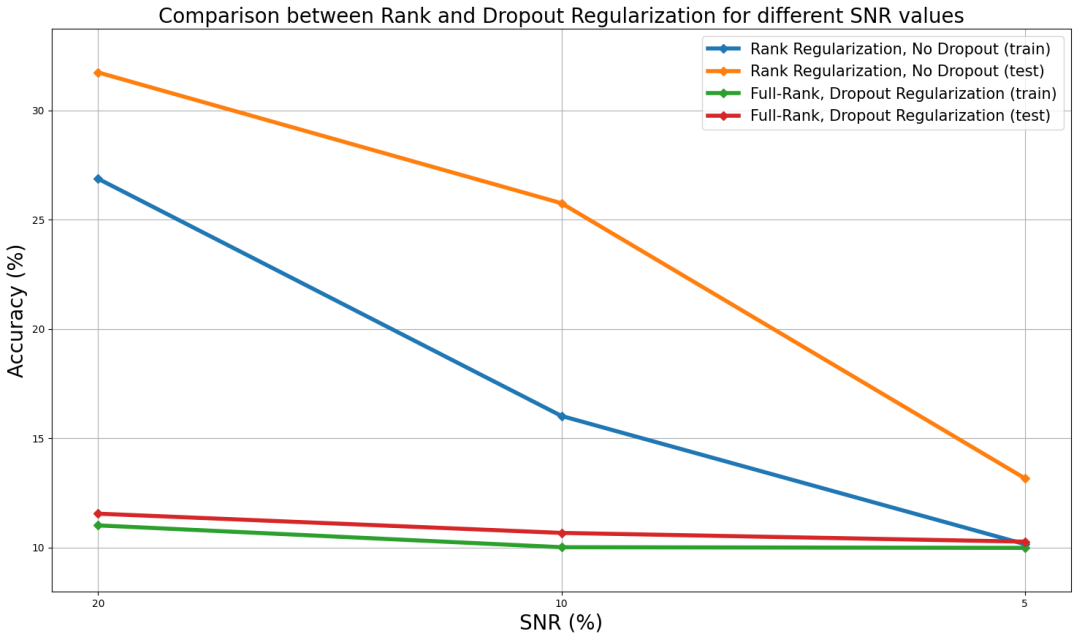

Figure 42: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 70% is utilized.

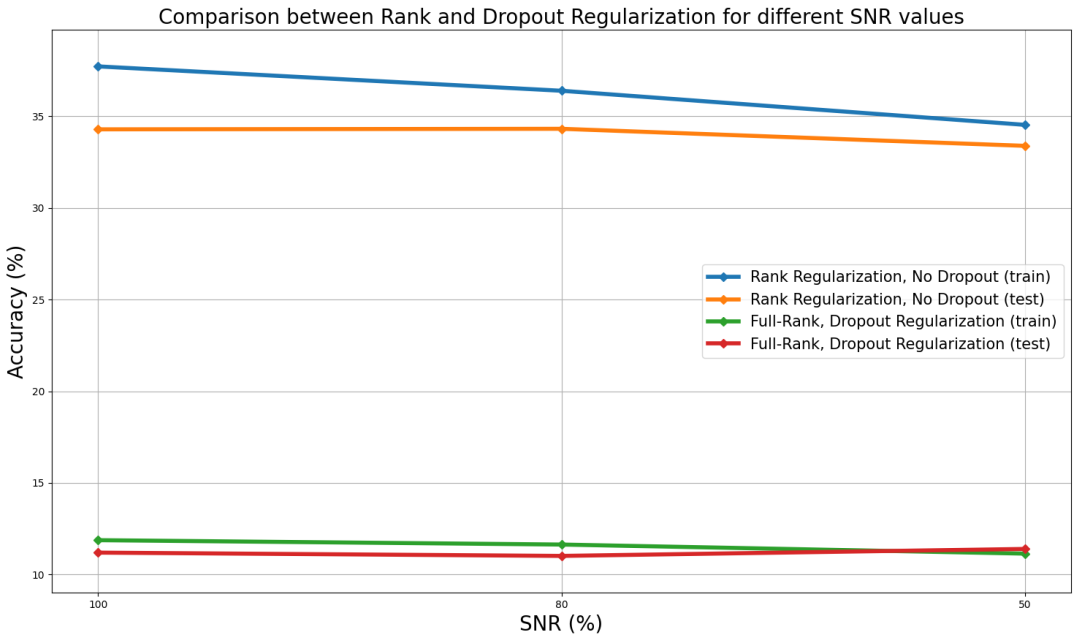

Figure 43: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 60% is utilized.

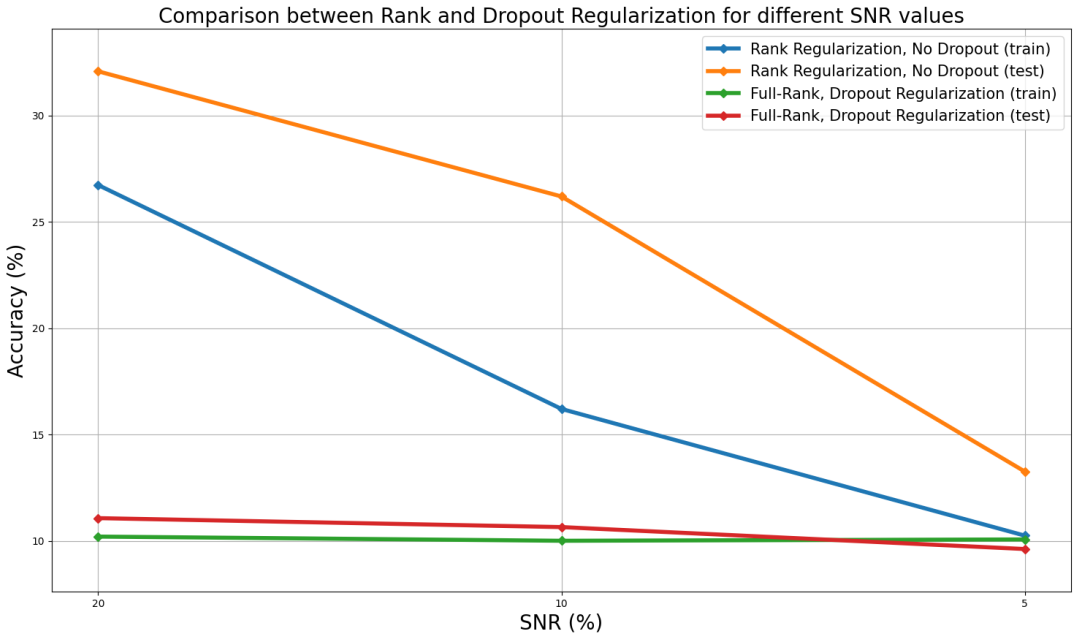

Figure 44: This figure presents the accuracy achieved at various SNR values after 15 epochs of training. It offers a comparative study of two distinct DPNN architectures: one featuring explicit rank reduction in its parameter tensors, and the other employing dropout regularization. The network is trained using the CIFAR10 dataset. For the dropout regularized model, a Bernoulli probability of success set at 60% is utilized.

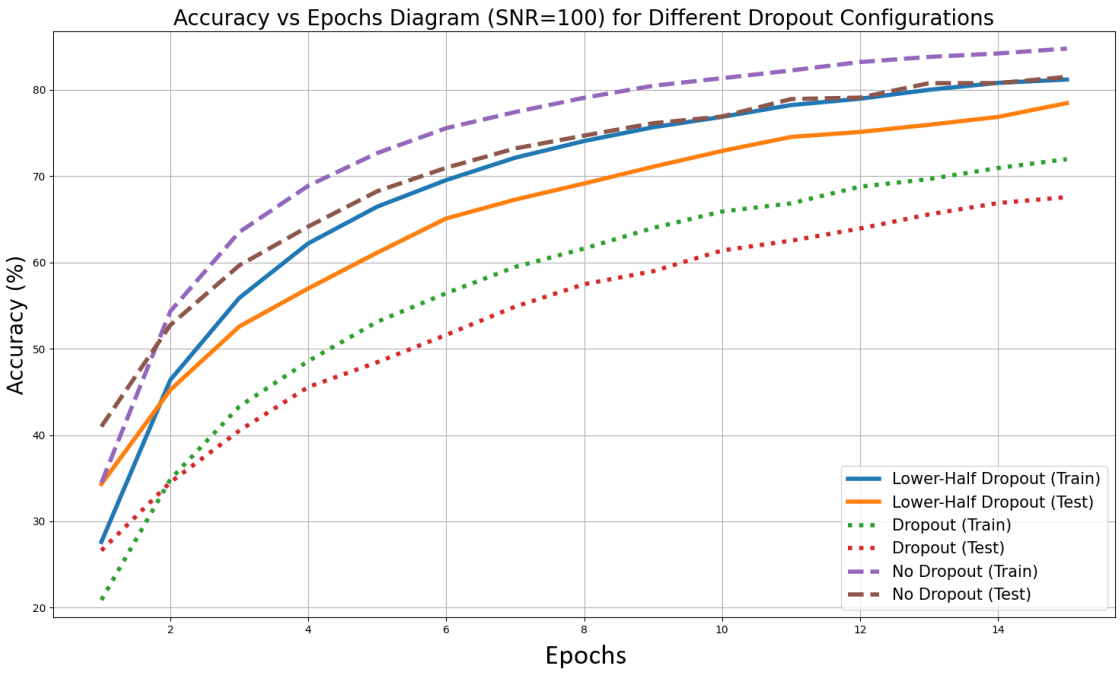

Figure 45: The graph illustrates the accuracy relative to epochs for a constant Signal-to-Noise-Ratio (SNR=100). It compares three distinct configurations of Deep Polynomial Neural Networks (DPNNs): The first configuration employs 20% explicit rank reduction without any dropout regularization. The second configuration does not utilize explicit rank reduction but applies dropout regularization (θ = 80%) to all parameter tensors. The final configuration, similar to the second, does not use explicit rank reduction but applies dropout regularization (θ = 80%) only to the latter half of the parameter tensors.

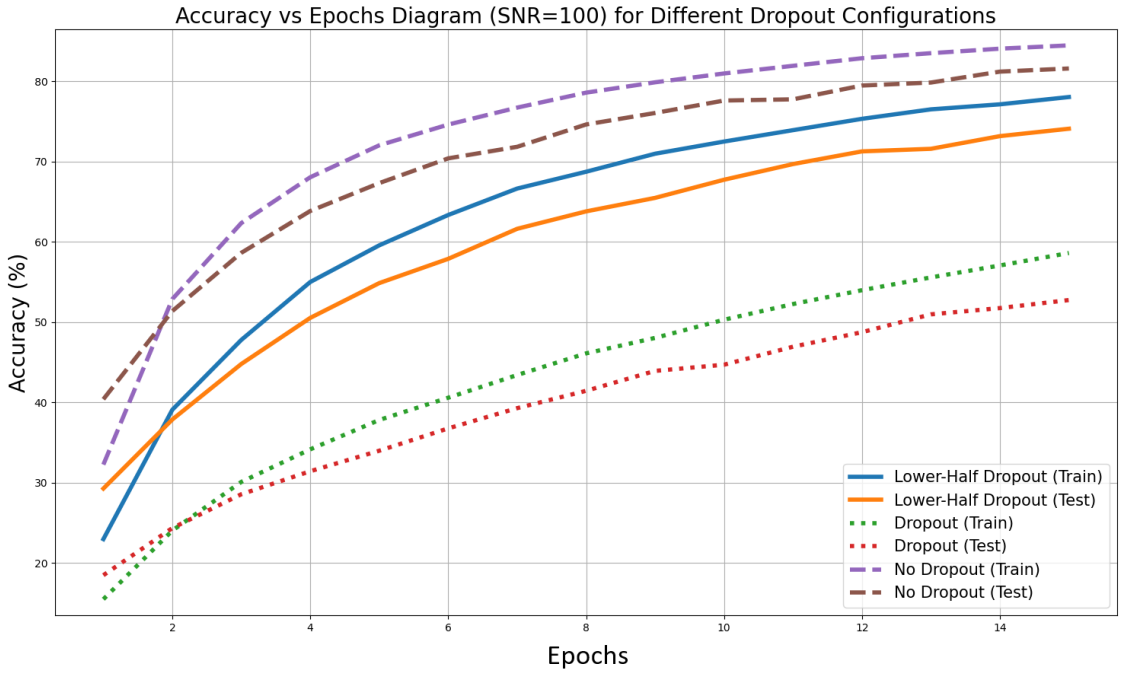

Figure 46: The graph illustrates the accuracy relative to epochs for a constant Signal-to-Noise-Ratio (SNR=100). It compares three distinct configurations of Deep Polynomial Neural Networks (DPNNs): The first configuration employs 30% explicit rank reduction without any dropout regularization. The second configuration does not utilize explicit rank reduction but applies dropout regularization (θ = 70%) to all parameter tensors. The final configuration, similar to the second, does not use explicit rank reduction but applies dropout regularization (θ = 70%) only to the latter half of the parameter tensors.

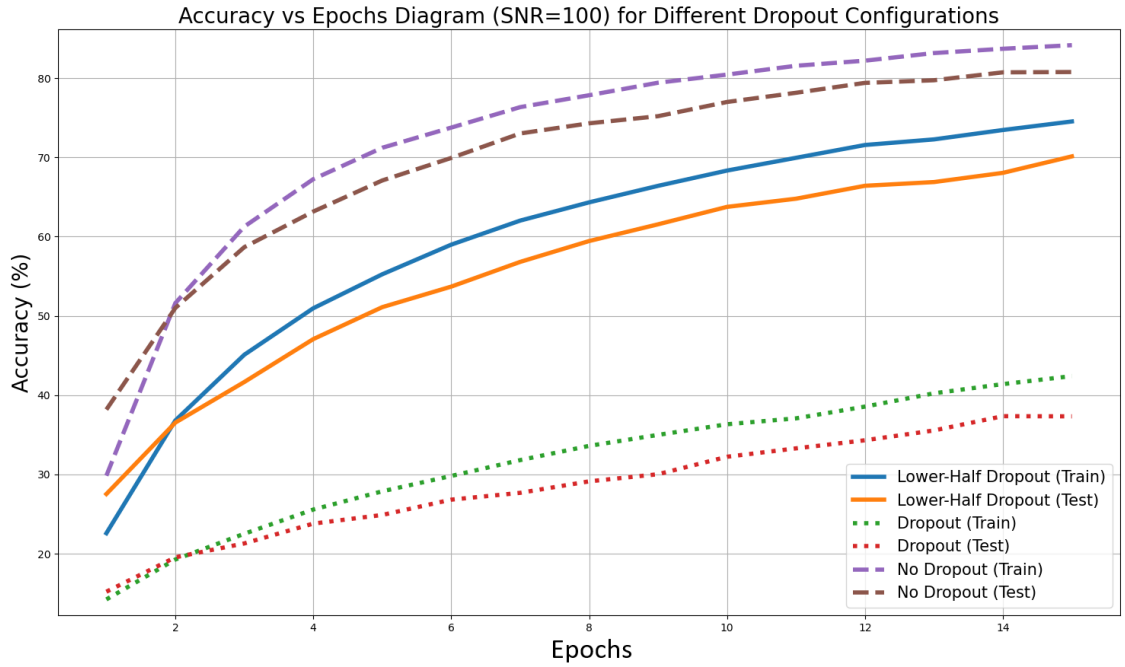

Figure 47: The graph illustrates the accuracy relative to epochs for a constant Signal-to-Noise-Ratio (SNR=100). It compares three distinct configurations of Deep Polynomial Neural Networks (DPNNs): The first configuration employs 40% explicit rank reduction without any dropout regularization. The second configuration does not utilize explicit rank reduction but applies dropout regularization (θ = 60%) to all parameter tensors. The final configuration, similar to the second, does not use explicit rank reduction but applies dropout regularization (θ = 60%) only to the latter half of the parameter tensors.

REFERENCES

[1] L. Alzubaidi, J. Zhang, A. J. Humaidi, *et al.*, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of big Data*, vol. 8, pp. 1–74, 2021.

[2] L. Bottou, *Online Algorithms and Stochastic Approximations*. Cambridge University Press, 1998.

[3] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[4] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. P. Zafeiriou, "Deep polynomial neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[5] J. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of Machine Learning Research*, PMLR, vol. 97, 2019, pp. 1310–1320.

[6] P. Comon, "Tensors versus matrices usefulness and unexpected properties," in *2009 IEEE/SP 15th Workshop on Statistical Signal Processing*, 2009, pp. 781–788.

[7] A. De Moivre, *The Doctrine of Chances*. A. Millar, 1738.

[8] J. Dean, *The deep learning revolution and its implications for computer architecture and chip design*, 2019.

[9] K. Dullemond and K. Peeters, "Introduction to tensor calculus," *Kees Dullemond and Kasper Peeters*, pp. 42–44, 1991.

[10] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[11] A. Elhassouny and F. Smarandache, "Trends in deep convolutional neural networks architectures: A review," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, 2019, pp. 1–8.

[12] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[13] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[14] C. F. Gauss, *Theoria motvs corporvm coelestivm in sectionibvs conicis Solem ambientivm*. Sumtibus Frid. Perthes et IH Besser, 1809.

[15] F. Girosi, M. Jones, and T. Poggio, “Regularization theory and neural networks architectures,” *Neural Computation*, vol. 7, no. 2, pp. 219–269, 1995.

[16] 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*, 2010, pp. 249–256.

[17] R. A. Harshman, “Foundations of the parafac procedure: Models and conditions for an ”explanatory” multi-modal factor analysis,” *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] F. L. Hitchcock, “The expression of a tensor or a polyadic as a sum of products,” *Journal of Mathematics and Physics*, vol. 6, no. 1-4, pp. 164–189, 1927.

[20] A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems,” *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[21] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *Lora: Low-rank adaptation of large language models*, 2021.

[22] A. G. Ivakhnenko, “Polynomial theory of complex systems,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-1, no. 4, pp. 364–378, 1971.

[23] N. L. Johnson, S. Kotz, and N. Balakrishnan, *Continuous Univariate Distributions*. Wiley, 1994.

[24] S. R. Kamalakara, A. Locatelli, B. Venkitesh, J. Ba, Y. Gal, and A. N. Gomez, *Exploring low rank training of deep neural networks*, 2022.

[25] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artificial intelligence review*, vol. 53, pp. 5455–5516, 2020.

[26] A. Kolbeinsson, J. Kossaifi, Y. Panagakis, *et al.*, *Tensor dropout for robust learning*, 2020.

[27] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

[28] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *arXiv preprint arXiv:1607.02533*, 2016.

[29] L. Le Cam and G. Lo Yang, *Asymptotics in Statistics: Some Basic Concepts*. Springer Science & Business Media, 2000.

[30] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687.

[31] Y. LeCun, C. Cortes, and C. J. Burges, “The mnist database of handwritten digits,” *http://yann.lecun.com/exdb/mnist/*, 1998.

[32] H. Madala and A. Ivakhnenko, *Inductive Learning Algorithms for Complex Systems Modeling*. CRC Press, 1994.

[33] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[34] Matiur Rahman Minar and Jibon Naher, *Recent advances in deep learning: An overview*, en, 2018.

[35] P. Mianjy, R. Arora, and R. Vidal, *On the implicit bias of dropout*, 2018.

[36] R. Moradi, R. Berangi, and B. Minaei, "A survey of regularization strategies for deep models," *Artificial Intelligence Review*, vol. 53, pp. 3947–3986, 2020.

[37] Y. Panagakis, J. Kossaifi, G. G. Chrysos, *et al.*, "Tensor methods in computer vision and deep learning," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 863–890, May 2021.

[38] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8024–8035, 2019.

[39] S. Rabanser, O. Shchur, and S. Günnemann, "Introduction to tensor decompositions and their applications in machine learning," *arXiv preprint arXiv:1711.10781*, 2017.

[40] G. Raskutti, M. J. Wainwright, and B. Yu, "Early stopping for non-parametric regression: An optimal data-dependent stopping rule," in *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2011, pp. 1318–1325.

[41] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A python toolbox to benchmark the robustness of machine learning models," *arXiv preprint arXiv:1707.04131*, 2017.

[42] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[43] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *SN Computer Science*, vol. 2, no. 420, 2021.

[44] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks : the official journal of the International Neural Network Society*, vol. 61, pp. 85–117, 2014.

[45] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[46] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[47] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[48] T. Sochi, "Introduction to tensor calculus," *arXiv preprint arXiv:1603.01660*, 2016.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[50] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[51] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[52] L. R. Tucker, “Some mathematical notes on three-mode factor analysis,” *Psychometrika*, vol. 31, no. 3, pp. 279–311, Sep. 1966, ISSN: 1860-0980.

[53] K. Yamaguchi, K. Sakamoto, T. Akabane, and Y. Fujimoto, “A neural network for speaker-independent isolated word recognition,” in *First International Conference on Spoken Language Processing (ICSLP 1990)*, 1990, pp. 1077–1080.

[54] Y. Yao, L. Rosasco, and A. Caponnetto, “On early stopping in gradient descent learning,” *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[55] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European conference on computer vision*, 2014, pp. 818–833.

[56] J. Zhang, T. He, S. Sra, and A. Jadbabaie, “Why gradient clipping accelerates training: A theoretical justification for adaptivity,” *arXiv preprint arXiv:1905.11881*, 2019.

[57] H. Zou and T. Hastie, “Regularization and Variable Selection Via the Elastic Net,” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, Mar. 2005, ISSN: 1369-7412.