

# **Predicting the success of Gradient Descent for a particular Dataset-Architecture-Initialization (DAI)**

*A Project Report*

*submitted by*

**UMANGI JAIN**

*in partial fulfilment of the requirements  
for the award of the degree of*

**Dual Degree (B.Tech + M.Tech)**



**DEPARTMENT OF ELECTRICAL ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.**

**June 2021**

# THESIS CERTIFICATE

This is to certify that the thesis entitled **Predicting the success of Gradient Descent for a particular Dataset-Architecture-Initialization (DAI)**, submitted by **Umangi Jain (EE16B124)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree**, is a bona fide record of the research work carried out by her under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr. Harish Guruprasad Ramaswamy**  
Research Guide  
Assistant Professor  
Dept. of Computer Science and Engineering  
IIT-Madras, 600 036

Place: Chennai

Date:

## **ACKNOWLEDGEMENTS**

Firstly, I would like to express my sincere gratitude towards my advisor, Prof. Harish Guruprasad Ramaswamy. His teachings motivated me to pursue research in this direction. He gave me the freedom to explore and always helped and guided me in the right direction. This work would not have been possible without his invaluable guidance, time, and support. I would also like to thank Prof. Abhishek Sinha for his valuable insights, feedback, and support.

I am indebted to all the professors in Electrical Engineering, Computer Science, and other departments whose courses I have taken and learned a lot. These concepts were crucial to write this thesis.

I am very grateful to my parents and sister, who have always encouraged and loved me unconditionally.

# ABSTRACT

KEYWORDS: Neural networks, Early give-up, Early success indicators

Despite their massive success, training successful deep neural networks (DNN) still largely relies on experimentally choosing an architecture, hyper-parameters, initialization, and training mechanism. DNNs are also characterized by variability in their performance due to the stochastic nature of deep learning algorithms. Initialization of weights itself can play a critical role in determining the success of gradient descent on a particular DAI (dataset-architecture-initialization) combination, which makes it important to identify a good initialization technique. However, testing the success of initialization can require significant training time and computing power. In this work, several initializations are tested for different architectures and datasets, and an attempt is made to predict which of these initializations will train successfully at an early stage. This can allow for discarding models that are unlikely to train at the final stages of the training process, thus saving training time and compute power.

This work proposes and studies an empirical tool to determine the success of the standard gradient descent method for training deep neural networks on a specified dataset, architecture, and initialization (DAI) combination. It benchmarks several DAI combinations and introduces two methods, noise stability and decay of the distribution of singular values of the output from hidden layers, which aid in predicting the success of a DAI. Through extensive systematic experiments, it is shown that the evolution of singular value decomposition of the matrix obtained from the hidden layers of a DNN can help determine the success of gradient descent technique to train a DAI (even in the absence of validation labels in the supervised learning paradigm). This phenomenon can facilitate early give-up, stopping the

training of neural networks, which are predicted to not generalize well early in the training process.

Experimentation across multiple tasks, datasets, architectures, and initialization schemes reveals that the proposed scores can more accurately predict the success of a DAI than simply relying on the training and validation accuracy at earlier epochs to make a judgment. The idea has been further extended to choose an optimal initialization and architecture from a set of initializations and architectures trained on a dataset using a dynamic run selection algorithm and the proposed early prediction scoring. The current scope of this work is limited to multilayer perceptron (MLP) and convolutional neural networks (CNN) architectures.

# TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS</b>	<b>i</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>ABBREVIATIONS</b>	<b>ix</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Early success indicators . . . . .	3
<b>2 Background Work</b>	<b>5</b>
2.1 Compressibility in trained neural networks . . . . .	5
2.2 Gradient descent aligns the layers of deep linear networks . . . . .	6
2.3 Upper Confidence Bound (UCB) algorithm . . . . .	7
<b>3 Experimental Setup</b>	<b>9</b>
3.1 Dataset . . . . .	9
3.2 Architecture . . . . .	11
3.3 Initialization . . . . .	12
<b>4 Alternative success indicators</b>	<b>14</b>
4.1 Noise Stability . . . . .	14
4.1.1 Experiments on noise stability . . . . .	14
4.2 Automatic dimensionality reduction . . . . .	19
4.2.1 Experiments on automatic dimensionality reduction . . . . .	20
<b>5 Scores for early success indicators</b>	<b>25</b>
5.1 Predicting success of initializations for a dataset and architecture	31
5.2 Improving Scoring . . . . .	34

<b>6</b>	<b>Evaluation of early success indicators</b>	<b>37</b>
6.1	Failure of Noise stability . . . . .	39
6.2	Results on multilayer perceptron . . . . .	39
6.3	Results on convolutional neural networks . . . . .	44
6.4	Performance across architectures . . . . .	45
<b>7</b>	<b>Dynamic run selection algorithm</b>	<b>48</b>
7.1	Introduction . . . . .	48
7.2	Evaluating dynamic run selection algorithm . . . . .	51
7.2.1	Evaluating performance on DAI . . . . .	52
7.2.2	Evaluating performance on DA . . . . .	56
<b>8</b>	<b>Future directions and Conclusion</b>	<b>60</b>
8.1	Future Work . . . . .	60
8.1.1	Subspace for different classes . . . . .	60
8.1.2	Predicting success of a DAI as a classification problem . .	61
8.2	Conclusion . . . . .	63
<b>A</b>	<b>Additional Experiments</b>	<b>67</b>

## LIST OF TABLES

6.1	Correlation of $s_F$ , training accuracy, $s_{Fv}$ , validation accuracy at different checkpoint epochs with the final validation accuracy for DA-1.	42
7.1	Range of validation accuracies and number of acceptable models in each set for DA-1 and DA-2. . . . .	53
7.2	Number of epochs required by each reward scheme to find a good initialization for each set in DA-1 and DA-2. . . . .	54
7.5	Sets comprising of different architectures trained on Shell-1024.	57
7.6	Number of epochs required by each reward scheme to find a good architecture for each set in Shell-1024 dataset. . . . .	57



## LIST OF FIGURES

1.1	Distribution of training accuracy: Variability of outputs - scaled weights in Xavier normal initialization scheme. . . . .	2
3.1	Visualization of the distribution of Shell dataset in $R^2$ . The two classes lie on shells of different radius $r_1$ and $r_2$ respectively. . . .	10
4.1	Noise stability: Shell-1024 dataset, 4-layer MLP, Normal Xavier initialization. Effect of noise injected on higher layer gets attenuated in lower layers as the network learns. . . . .	15
4.2	Noise stability: Shuffled training labels of Shell-1024 dataset, 4-layer MLP, Normal Xavier initialization. Noise stability and validation accuracy reduces with increasing percentage of shuffled training labels. . . . .	16
4.5	Noise stability: CIFAR-10, 16-layer CNN, Normal Xavier initialization. Effect of noise injected on higher convolutional layers gets attenuated as training progresses. . . . .	18
4.6	Comparison of noise stability of CIFAR-10 dataset, 10-layer CNN architecture, initialization from Normal Xavier scheme with correct and shuffled labels. Noise stability reduces for the second model. . . . .	19
4.7	SVD output at various epochs for Shell-1024 dataset, and two different initializations. (a)-(c): initialization from Normal Xavier scheme. (d)-(f) : initialization from Shuffled Trained scheme. . . . .	21
4.8	SVD output at various epochs for Shell-32 dataset. Steep decay of singular values with no effective training. . . . .	22
4.9	SVD output at various epochs and noise stability output on a regression problem. As the validation loss reduces, preferential directions are strengthened. . . . .	23
4.10	SVD output at various epochs for (a)-(c) correct labels, (d)-(f) shuffled labels on Shell-32 dataset. Trend in distribution of singular values follows the trend in the validation accuracy. . . . .	24
5.1	Evolution of SVD and noise stability for Shell-1024 dataset, 2-layer MLP architecture, initialization from Normal Xavier scheme. . . . .	26

5.2	Evolution of SVD and noise stability for Shell-1024 dataset, 2-layer MLP architecture, initialization from Scaled Normal Xavier scheme (Scale 10). . . . .	26
5.3	DAI-1: attains high validation accuracy along with high $s_{nt}, s_{ot}, s_{st}$ . . . . .	30
5.4	DAI-2: attains high validation accuracy along with high $s_{nt}, s_{ot}, s_{st}$ at epoch 30, correctly indicating high chances of success of further training. . . . .	30
5.5	DAI-3: attains low validation accuracy along with low $s_{nt}, s_{ot}, s_{st}$ at epoch 30, correctly indicating low chances of success of further training. . . . .	31
5.6	Shell-1024 dataset trained on a neural network architecture of 8 hidden layers obtains low scores on $s_{st}$ and $s_{ot}$ . . . . .	35
5.7	Normalized singular values (sorted) for all hidden layers at various epochs. Profile of singular value decay can be different for all layers which can reduce the average score for deep neural networks. . . . .	36
6.1	Performance of $s_{nt}$ : $s_{nt}$ at checkpoint epoch shows a negative correlation with the final validation accuracy for DA-1 and DA-2. . . . .	39
6.2	Final training and validation performance of 120 different initialization for DA-1. . . . .	40
6.3	Variation of scores $s_F$ and $s_{Fv}$ with epochs and their corresponding validation accuracy for five initializations of DA-1. . . . .	41
6.4	Breakdown of the prediction of eventual success of all initializations in DA-1 using $s_{Fv}$ and $BV$ at epoch 10. . . . .	41
6.5	$s_F$ , training accuracy, $s_{Fv}$ , validation accuracy at checkpoint epoch 30 plotted against the final validation for all initializations in DA-1. . . . .	42
6.8	Dense layer in well trained CNN show a decay in the singular values upon training on DA-3. . . . .	45
7.1	Dynamic run selection algorithm for 10 initializations trained on Shell-512 dataset, 2-layer MLP run for 100 epochs. The most selected arm attains the highest validation accuracy on full training. . . . .	51
7.2	Distribution of final validation and train accuracy for each set in DA-1. . . . .	53
7.6	Distribution of final validation and train accuracy for each set in Shell-1024 dataset. . . . .	56
7.7	Visualizing the training and validation accuracy of architectures in Set-I trained on Shell-1024 dataset. The good architectures are highlighted in green. . . . .	59

## ABBREVIATIONS

<b>CNN</b>	Convolutional Neural Network
<b>BT</b>	Baseline Training score
<b>BV</b>	Baseline Validation score
<b>DA</b>	Dataset Architecture
<b>DAI</b>	Dataset Architecture Initialization
<b>DNN</b>	Deep Neural Network
<b>DRSA</b>	Dynamic Run Selection Algorithm
<b>ESI</b>	Early Success Indicator
<b>MLP</b>	Multi-layer Perceptron
<b>MSE</b>	Mean Squared Error
<b>SVD</b>	Singular Value Decomposition
<b>UCB</b>	Upper Confidence Bound

# CHAPTER 1

## INTRODUCTION

Neural architectures have achieved state-of-the-art performance on various large-scale supervised tasks in several domains spanning language, vision, speech, recommendations. Modern deep learning algorithms consume enormous energy in terms of compute power ([10], [15]). The abundance of data and improvements on the hardware side has made it possible to train larger models, increasing the energy consumption at a tremendous rate. These deep learning algorithms go through intense mathematical calculation during forward and backward passes for each piece of data to update the large weight matrices. Another crucial factor that significantly adds to the computing power is the immense experimentation and tuning required to choose optimal initialization, architecture, and hyper-parameters. Several variants of a model are generated and trained for a particular task on a particular dataset to make an optimal choice.

While there is active research going on to make deep learning algorithms greener and economical like compressing the model [8], training on a subset of the dataset [2], designing hyper-efficient network [16], there's still no standard mechanism to choose an initialization or architecture design. Though various theories have been proposed for selecting an initialization distribution, architecture design, training mechanism, these nets are still most often manually designed through extensive experimentation. For instance, initialization schemes, such as Xavier [3], He [5], random orthogonal [12] have been proposed in the past. Despite some theoretical backing, these commonly used initialization techniques are simple and heuristic [4]. The more recent neural architecture search (NAS) research has fueled efforts in automatically finding good architectures ([11], [18]). It has been observed that these NAS algorithms are computationally very intensive and time-consuming as well. In addition, changing the dataset or tasks requires starting these algorithms anew.

Furthermore, DNNs are characterized by variability in performance. The performance of the same initialization, training scheme, and architecture on a particular dataset can vary due to the stochastic nature of deep learning algorithms, system architecture, operating systems, or libraries. Even initializing from the same distribution of initializers for a particular dataset and architecture can have significantly different performances. This fickleness can significantly affect the ultimate performance and generalization gap in neural networks. Figure 1.1 demonstrates the variability in the performance of DNNs with MNIST dataset and a fixed choice of architecture, 2-layer MLP (64, 32 hidden layer dimension respectively), with ReLU activation, and RMSprop optimizer. The loss function taken is cross-entropy. The  $x$ -axis is the different initialization distribution, obtained by varying the scale of standard deviation in the Gaussian distribution used in Xavier initialization from 1.0 to 256.0 in factors of 4. Each point along a vertical line has the same distribution. The initializations are sampled from each distribution 5 times. All the 25 networks are trained on the same dataset, architecture, training procedure, and trained until 200 epochs. The corresponding final training accuracy at the final epoch for these models has been plotted along the  $y$ -axis (Validation accuracy also follows a similar trend). It can be seen that even for the same distributions, some neural networks show performance as low as 11.05% to as high as 97.42%. Consequently, experimentation is required to choose an optimal initialization weight matrix even when sampling from a fixed initialization distribution.

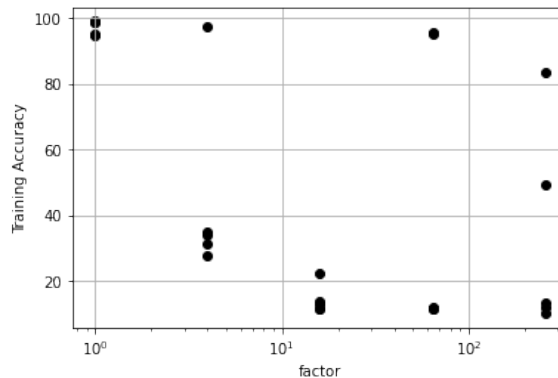


Figure 1.1: Distribution of training accuracy: Variability of outputs - scaled weights in Xavier normal initialization scheme.

Therefore, in practice, choosing good initializations and architectures often re-

duces to experimentally testing several possibilities. The model needs to be trained for several epochs (till either the loss reaches a certain threshold or overfitting begins) to determine the success of initialization on a particular architecture and dataset. This process consumes significant compute power and time and can limit the number of initializers to be tried out. While the need for experimentation can not be eliminated wholly, this work proposes a mechanism to test several weight initializations and architectures in lesser computational power. An early success indicator is presented, which predicts at an early stage of training, the extent to which a DAI (dataset-architecture-initialization combination) that is trained using gradient descent would be successful. It provides insights into the benefits of further training. Such a score would be crucial to test several initializations and architectures with a reduced cost. By discarding early, DAIs that do not have the potential of learning effectively, computing power and run time can be saved.

The following contributions are made in this work:

1. Early success indicators (ESI) are introduced and analyzed to predict the success of an initialization for a particular architecture and dataset through extensive empirical study.
2. The problem of finding a "good" initialization from a set of initializations is formulated in a multi-armed bandit setting.
3. ESI are used to devise a dynamic run selection algorithm to choose a good initialization from a set of initializations for a fixed dataset, architecture, and training procedure. This algorithm is extended to find a good architecture from a set of networks with different architectures trained on the same dataset.

## 1.1 Early success indicators

Early success indicators aim to predict, at an early stage of training, the extent to which a DAI trained using gradient descent would be successful. The extent of success in the experimental set-up of this work is taken as the final validation accuracy (accuracy of validation data in the last epoch  $e_L$ ). The early success indicators give a score, which is calculated at an earlier epoch (called checkpoint

epoch)  $e_C$  ( $e_C < e_L$ ), that is indicative of the success of the DAI. This score enables us to identify and discard models that would not train sufficiently at the end of training and continue training the remaining. Additionally, it can determine the benefit of further training. For instance, if the validation accuracy of the DAI at  $e_L$  is less than that at  $e_C$ , then the score of one of the early success indicators would be negative. Likewise, if the validation accuracy does not change significantly from  $e_C$  to  $e_L$ , the score of one of the early success indicators would be close to zero. A higher positive value of score indicates higher chances of success of the DAI. The prediction regarding the success of an initialization can be helpful in two ways: it can aid in identifying how well the network is trained at the checkpoint epoch. If not trained, it can predict whether the network is capable of learning on further training. In either case, such a prediction will be crucial in determining whether further training of the DAI would be beneficial.

## **Thesis organization:**

This thesis is organized as follows - Section 2 outlines the background work and formalisms required to build the scoring mechanisms for early success indicators. Section 3 explains the experimental set-up and the dataset, architectures, and initialization distributions used throughout the experiments, section 4 describes the basis of the two early success indicators. Section 5 details the proposed scoring mechanism, section 6 shows the experimental results of the performance of these indicators on classification and regression tasks. Section 7 describes and evaluates the dynamic run selection algorithm for a set of initializations/architectures. Section 8 highlights the future directions for the work and establishes the conclusion.

## CHAPTER 2

### Background Work

This section outlines the prior work that has been used in predicting the success of gradient descent on a DAI. Trained neural networks show compressibility property, as explained in section 2.1, which is exploited to design early success indicators. The distribution of singular values from the weight matrices of the hidden layers has revealed intriguing properties for deep linear networks as highlighted in section 2.2. The dynamic run selection algorithm in Section 7 is based on the Upper Confidence Bound or UCB algorithm, which is stated in section 2.3.

#### 2.1 Compressibility in trained neural networks

The early success indicator is motivated by the noise compression properties as outlined in [1]. In their paper, Arora et al. propose an empirical noise-stability framework for DNNs that computes each layer's stability to the noise injected at lower layers. Each layer's output to an injected Gaussian noise at lower layers is stable for a well-trained network. The added noise in a trained neural network (capable of generalization) gets attenuated as it propagates to higher layers. This noise stability allows individual layers to be compressed.

Gaussian noise is injected at one of the hidden layers to test the noise stability of a network, and its propagation throughout the network is studied. It has been observed that as noise propagates to deeper layers, the effect of noise diminishes. Noise sensitivity  $\Psi_N$  of a mapping  $M$  from a real-valued vector  $x$  to another real-valued vector with respect to some noise distribution  $N$  is defined as:

$$\Psi_N(M, x) = E_{\eta \in N} \left[ \frac{\|M(x + \eta\|x\|) - M(x)\|^2}{\|M(x)\|^2} \right]$$



Noise stability from a linear mapping with respect to Gaussian noise has been derived mathematically in [1]. Their results prove that the noise sensitivity of a matrix  $M$  at any vector  $x \neq 0$  with respect to Gaussian distribution  $N(0, I)$  is  $\frac{\|M\|_F^2 \|x\|^2}{\|Mx\|^2}$ . This suggests that if a vector  $x$  is aligned with matrix  $M$ , the noise sensitivity would be low, implying that the matrix  $M$  would be less sensitive when noise is added at  $x$ . Low sensitivity implies that the matrix  $M$  has some large singular values. This gives rise to some preferential directions along which signal  $x$  is carried, while the noise, which is uniform across all directions, gets attenuated. Intuitively, this translates to a non-uniform distribution of singular values from the output of a transformation that generalizes well. On training further, as the noise stability of the neural network strengthens, the noise gets suppressed further, the extent of compression increases, and the preferential direction of propagation gets aligned with the signal (higher singular values of  $M$ ).

Through experimentation, it has been shown that this compression property is observed for non-linear deep neural networks as well. This work shows that the distribution of singular values of the output matrix from hidden layers in DNNs is non-uniform. This non-uniform distribution of singular values and its evolution as training continues is used to build the early success indicators. Its utility in predicting the success of a DAI for feed-forward networks is shown empirically.

## 2.2 Gradient descent aligns the layers of deep linear networks

Ziwei Ji et al. [6] in their work show encouraging results for gradient descent (with decreasing loss function<sup>1</sup>) applied to deep linear networks on linearly separable data. Consider a dataset  $\{(x_i, y_i)\}_{i=1}^n$ , where  $x_i \in R^d$ ,  $\|x_i\| \leq 1$  and  $y_i \in \{-1, +1\}$ , that is assumed to be linearly separable. A linear network of depth  $L$  is parameterized by weight matrices  $W_L, \dots, W_1$ , where  $W_k \in R^{d_k \times d_{k-1}}$ .  $d_0 = d$  is the dimension of input

---

<sup>1</sup>for smooth losses (i.e, with Lipschitz continuous derivatives), gradient descent with decaying step sizes can be designed that never increases the risk

vector and  $d_L = 1$ . Let  $W = (W_L, \dots, W_1)$  denote all parameters of the network.  $w_{prod}$  denotes  $(W_L \dots W_1)^T$ . The (empirical) risk of the network is :

$$R(W) = \frac{1}{n} \sum_{i=1}^n l(y_i W_L \dots W_1 x_i) = \frac{1}{n} \sum_{i=1}^n l(\langle w_{prod}, y_i x_i \rangle)$$

In [6], it is shown that for a loss function  $l$  that is continuously differentiable, unbounded, strictly decreasing to 0,  $\beta$ -smooth and  $G$ -Lipschitz :

- The risk converges to 0.  $\lim_{t \rightarrow \infty} R(W(t)) = 0$
- The normalized weight matrix  $W_i / \|W_i\|_F$  asymptotically equals its rank-1 approximation of  $u_i v_i^T$ , where  $u_i$  is the left singular vector and  $v_i$  is the right singular vector.
- These rank-1 matrices are aligned across layers  $|v_{i+1}^T u_i| \rightarrow 1$  and  $W_{k+1}$  and  $W_k$  have the same singular values (asymptotically). Since  $W_L$  is taken as a row vector, all layers have rank 1.
- For logistic loss, the gradient descent iterations converge in the same direction as the maximum margin solution.

For a well-trained network, gradient descent converges in specific directions. However, most real-world data are not linearly separable, and non-linear deep neural networks can still get the risk to 0 (even for shuffled labels as shown in [17]). Section 4.2 analyzes the distribution of singular values from the output of a hidden layer in real-world datasets and DNNs empirically.

## 2.3 Upper Confidence Bound (UCB) algorithm

The problem of choosing a "good" initialization from a set of initializations, as studied in detail in section 7, has been formulated as a multi-armed bandit problem in this work. Formally, a stochastic bandit (reward for each arm is sampled from an IID distribution specific to that arm) algorithm  $A$  has  $K$  possible arms/actions to choose from in  $T$  rounds. In round  $t$ , the Upper Confidence Bound (UCB) algorithm, as shown in [9], changes its exploration-exploitation balance as it enhances

its knowledge about the environment, to maximize its rewards. UCB algorithm  $A$ , at round  $t$ , chooses an action  $a_t$  given by

$$a_t = \operatorname{argmax}_a \left[ Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

where  $Q_t(a)$  is the estimated average reward of arm  $a$  up to round  $t$ ,  $N_t(a)$  is the number of times action  $a$  has been chosen before round  $t$ ,  $c$  is the hyper-parameter that controls the exploration-exploitation balance. The two terms in the expression can be thought of as exploitation and exploration terms, respectively. The second term is a measure of the variance in the estimation of the reward for arm  $a$ . The UCB algorithm has tight theoretical bounds under three assumptions [14]: algorithm  $A$  observes rewards of only the selected arm (bandit feedback), the reward for each action is IID, i.e., at any round  $t$ , the reward for the chosen arm  $a$  is sampled from its distribution  $D_a$ , and the reward at every round is bounded.

Motivated from the UCB algorithm, a dynamic run selection algorithm has been devised for the problem setting mentioned above.

## CHAPTER 3

### Experimental Setup

This section specifies the notations for the datasets, architectures, and initialization schemes that are used in the experiments to follow.

#### 3.1 Dataset

Success of a DAI is tested on two tasks, classification and regression. Following datasets are used for experimentation in this work:

##### Classification Datasets

1. **Dataset1** - MNIST : Digit classification dataset consisting of 60,000 training examples and 10,000 examples for validation. Number of classes is 10. Image size is  $28 \times 28$ . For MLP, it is flattened to 784 dimensional vector.
2. **Dataset2** - Shell-d : Shell-d dataset (norm of vector is a constant) consists of 20,000 training and 4,000 validation samples in all experiments. Input vector can be of different dimensions  $d$ , where ( $d \in \{32, 64, 128, 256, 512, 1024\}$ ). The default vector size is 1024. Number of classes is 2.
3. **Dataset3** - CIFAR-10 : Colour images consisting of 50,000 training examples and 10,000 examples for validation. Number of classes is 10. Image size is  $32 \times 32 \times 3$ .

**Shell-d Dataset:** Shell-d dataset is a synthetic dataset comprising of 2 classes. Each class lies in a shell in  $\mathbb{R}^d$ . In the experiments, the two classes form a concentric shell of radius 1 and 1.1 respectively. In 2-dimensional space ( $d = 2$ ), Shell-2 data can be visualized on the 2 curves as shown in Figure 3.1. Sharp shells are used to generate the data (no noise added) for all the experiments.

**Regression Datasets :** The following two synthetic datasets are generated for regression task.

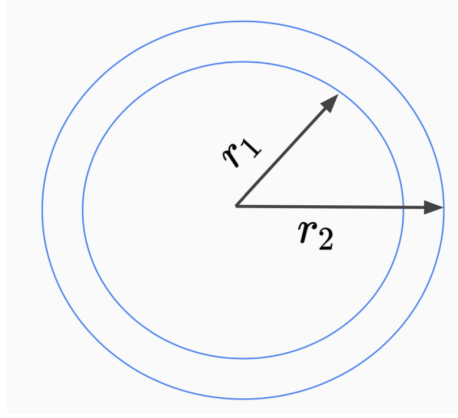


Figure 3.1: Visualization of the distribution of Shell dataset in  $R^2$ . The two classes lie on shells of different radius  $r_1$  and  $r_2$  respectively.

1. **Dataset4** - Cosine of Sum : Data points are generated from the below function

$$y = \cos\left(\sum_{i=b}^{b+f-1} x_i\right) + \sin\left(\sum_{i=b+f}^{b+2f-1} x_i\right) + \cos\left(\sum_{i=b+2f}^{b+3f-1} x_i\right) + \sin\left(\sum_{i=b+3f}^{b+4f-1} x_i\right)$$

- $x$  is a 100 dimensional vector. ( $x^T = [x_1, x_2, \dots, x_{100}]$ )
- $b$  is an arbitrary index. (Taken as 1 in all experiments)
- $4f$  is the total number of indices considered for calculating  $y$ .
- Only some units of the input vector contribute to the output  $[b, b + 4f)$ .
- The range of  $y$  is  $[-4, 4]$ .

Getting a uniform distribution over  $y$  by sampling  $x_i$  appropriately is non-trivial. (Because sum of two or more independent random variables is the convolution of their distributions). For instance, even if  $x_i$  is sampled such that  $\cos(\sum_{i=b}^{b+f-1} x_i)$  and  $\sin(\sum_{i=b+f}^{b+2f-1} x_i)$ , are both uniform over  $[-1, 1]$ ;  $z = \cos(\sum_{i=b}^{b+f-1} x_i) + \sin(\sum_{i=b+f}^{b+2f-1} x_i)$  will not have a uniform distribution as shown in Figure 3.2. Therefore, to get a uniform distribution over  $y$ , a different approach is taken. The range of  $y$  is divided into smaller bins, and equal numbers of samples are extracted from each bin. The samples are shuffled before training. The bin size for this dataset is taken to be 15.

2. **Dataset5** - Sum of Cosines : Data points are sampled from the below function

$$y = \frac{\sum_{i=b}^{b+f-1} \cos(x_i) + \sum_{i=b+f}^{b+2f-1} \sin(x_i)}{2f}$$

- $x$  is a 100 dimensional vector. ( $x^T = [x_1, x_2, \dots, x_{100}]$ )
- $b$  is an arbitrary index. (Taken as 1 in all experiments)
- $2f$  is the total number of indices considered for calculating  $y$ .

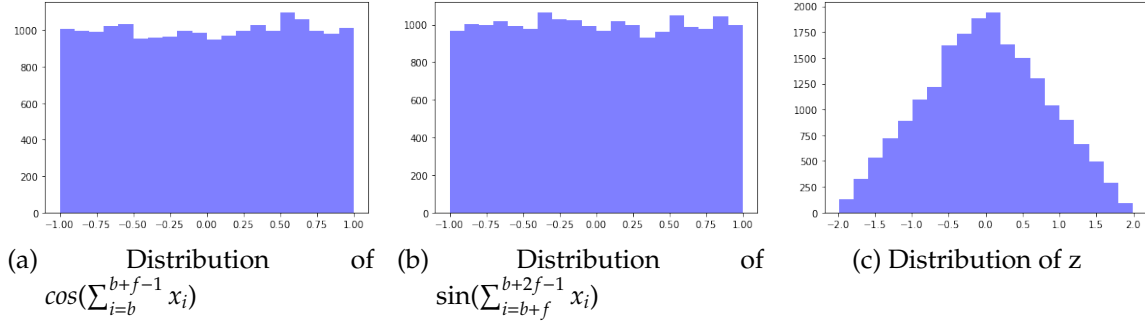


Figure 3.2: Samples from Cosine of Sum dataset for  $f = 4, b = 1$ . The distribution of  $z$  approximates the convolution of the distribution of the two independent variables.

- Only some units of the input vector contribute to the output  $[b, b + 2f)$ .
- The possible ranges of  $y$  is  $[-1, 1]$ .

The distribution of  $y$  is obtained in the same way as for Cosine of Sum dataset.

## 3.2 Architecture

Experiments are done on multilayer perceptron (MLP) and convolutional neural network (CNN). Following architectures are chosen for experiments:

### Multilayer Perceptron Architectures

1. **Architecture1** - 4-layer MLP: Four hidden layers of dimensions  $\{512, 256, 256, 128\}$  is taken. No dropout or batch-normalization layer included.
2. **Architecture2** - 2-layer MLP: Two hidden layers of dimensions  $\{64, 32\}$  is taken. No dropout or batch-normalization layer included.
3. **Architecture3** - 2-layer MLP: Two hidden layers of dimensions  $\{256, 128\}$  is taken. No dropout or batch-normalization layer included.

### Convolutional Neural Network Architectures

1. **Architecture4** - 16-layer CNN: VGG-16 [13] architecture is used. Another variant with dropout and batch-normalization layer is also considered in some experiments.

2. **Architecture5 - 10-layer CNN** : 10 layered convolutional neural architecture is used. 10 hidden layers are as follows: (C16 – A – C16 – A – M – C32 – A – D – C32 – A – M – C64 – A – C64 – A – C128 – A – M – F256 – A – F128 – A – F10 – softmax), where C is Conv2D layer, A is activation (ReLU), M is MaxPooling, F is Fully connected Layer, D is dropout.
3. **Architecture6 - 8-layer CNN** : 7 layered convolutional neural architecture is used. 8 hidden layers are as follows: (C32 – A – BN – C32 – A – M – D – C64 – A – C64 – M – D – C128 – A – C128 – A – M – Flatten – F512 – A – F10 – softmax), where BN is batch normalization.

Filter size is  $3 \times 3$  for all CNN architectures.

### 3.3 Initialization

Three different initialization schemes are used to obtain a varied set of generalization gaps and training accuracies in neural networks.

1. **Scheme1 - Normal Xavier** : Initialization using Normal Xavier method [3]. It draws samples from a truncated normal distribution with mean 0 with standard deviation  $= \sqrt{\frac{2}{fan\_in + fan\_out}}$ , where  $fan\_in$  is the number of input units in the weight tensor and  $fan\_out$  is the number of output units in the weight tensor.
2. **Scheme2 - Shuffled Trained** :  $y\%$  of labels in the training data (where  $y \in \{20, 40, 60\}$ ) is shuffled and the network is trained for  $x$  epochs. The weights of this partially trained network on shuffled data then initializes the model on a correctly labeled dataset. In all experiments,  $x \in \{40\}$ .
3. **Scheme3 - Scaled Normal Xavier** : Scales the standard deviation of the normal distribution in Xavier initialization.

#### Please Note:

- In all the experiments, rectified linear activation function (ReLU) activation is used.
- Optimizer is chosen from SGD, RMSProp, Adam.
- Loss function is cross-entropy for classification and MSE (mean squared error) for regression.
- For some experiments,  $y\%$  labels in the training data are shuffled, where  $y \in \{20, 40, 60\}$  to design networks with varied generalization gap.

## **Dataset-Architecture (DA) combination**

Evaluation of initialization for performance of early success indicators and dynamic run selection algorithm is done on the following dataset-architecture combinations.

1. **Dataset-Architecture-1 - DA1** : Shell-1024 dataset and Architecture1 (4-layer MLP).
2. **Dataset-Architecture-2 - DA2** : Shell-512 dataset and Architecture3 (2-layer MLP).
3. **Dataset-Architecture-3 - DA3** : CIFAR-10 dataset and Architecture6 (8-layer CNN).



## CHAPTER 4

### Alternative success indicators

The two main metrics generally used for evaluating a trained neural net are training loss and validation accuracy. While validation accuracy is still the gold standard, it is of vital importance to build other surrogate metrics that are predictive of validation accuracy and the state of training. In this section, two such metrics are introduced: noise stability and automatic dimensionality reduction.

#### 4.1 Noise Stability

This section studies generalization gaps in DNNs via a simple compression-based framework introduced in Arora et al. [1]. The compression effect is investigated on classification and regression problems. For the classification task, experimentation has been done for MLP and CNN.

As introduced in section 2.1, noise-stability for deep nets is characterized by the stability of each layer's computation to Gaussian noise injected at lower layers. It is hypothesized that this added error gets attenuated for a trained neural network capable of generalizing as it propagates to higher layers. In all the experiments, noise with  $l_2$  norm to be 10% of the norm of the original input is added. As its effect decreases on higher layers, this property is called compressibility.

##### 4.1.1 Experiments on noise stability

In this section, several experiments on different datasets, architectures, and initializations are done to test the robustness of the compressibility property. The plots for noise stability begin at the layer at which noise is added. The  $x$ -axis denotes the

index of layers and  $y$ -axis denotes the relative error due to added Gaussian noise. It is calculated as  $\frac{\|x^i - \hat{x}^i\|_2}{\|x^i\|_2}$ , where  $x^i$  is the original input to layer  $i$  and  $\hat{x}^i$  is the input to layer  $i$  after adding noise.

**Compression after training:** Consider Shell-1024 dataset, 4-layer MLP architecture, and initialization from Normal Xavier scheme, which is trained using RMSprop optimizer with learning rate  $10^{-3}$  till 500 epochs. Figure 4.1 shows the noise stability of the DAI before and after training. It can be seen that Figure 4.1b shows significant attenuation of noise in all the layers. This demonstrates that the signal is propagating in specific preferential directions, unlike noise.

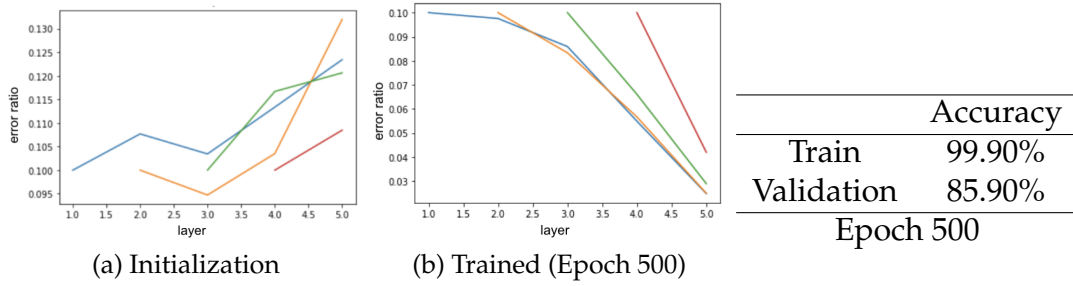


Figure 4.1: Noise stability: Shell-1024 dataset, 4-layer MLP, Normal Xavier initialization. Effect of noise injected on higher layer gets attenuated in lower layers as the network learns.

## Results on multilayer perceptron

In this section, the attenuation of noise in well-trained MLP networks is tested on Shell-1024 and MNIST dataset with varying generalization gaps. This is achieved by shuffling the training data labels and scaling the weights of the network at initialization.

**Compression after training on shuffled labels:** Zhang et al. in [17], experimentally showed that the standard architectures using SGD and regularization could reach low training error even when trained on randomly labeled examples (which will clearly not generalize). In this experiment,  $y\%$  labels of Shell-1024 dataset

(in the training data) are shuffled to test the robustness of compressibility on networks that reach almost zero training error but have a huge generalization gap. This provides networks with widely different validation accuracy. Consider the noise stability profile for same dataset, architecture, and initialization scheme with  $y \in \{20, 40, 60\}$  in Figure 4.2. All models are trained till 500 epochs with RMSProp optimizer and learning rate  $10^{-3}$ .

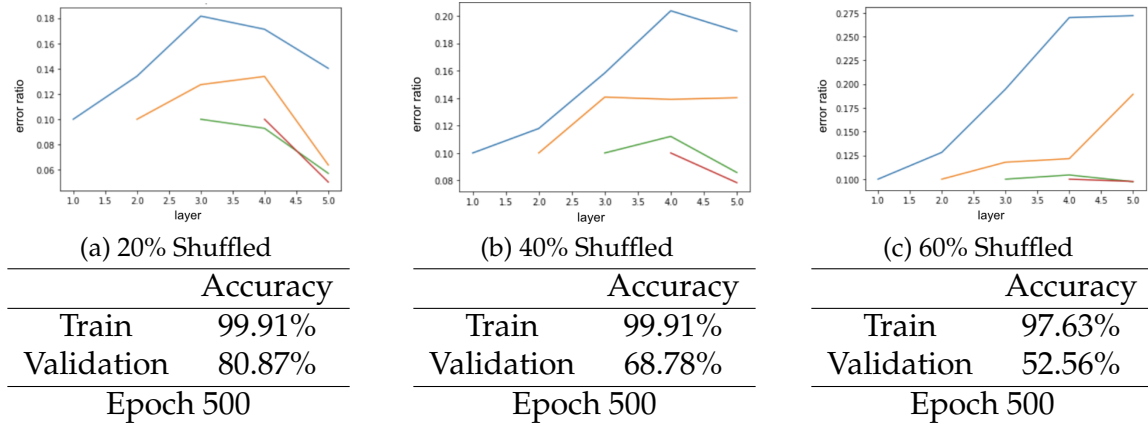


Figure 4.2: Noise stability: Shuffled training labels of Shell-1024 dataset, 4-layer MLP, Normal Xavier initialization. Noise stability and validation accuracy reduces with increasing percentage of shuffled training labels.

Clearly, the network’s noise stability reduces when trained on shuffled data, although the training accuracy remains almost the same. The extent of compression also correlates with the validation accuracy for these DAIs.

**Compression after training on initialization from Shuffled Trained:** To further test the robustness of noise stability, it is tested on models that have been initialized from Shuffled Trained initialization scheme. Consider the compression profile in Figure 4.3 for MNIST dataset, 4-layer MLP architecture. The model is trained on 40% shuffled labels till 40 epochs to get this initialization. It is further trained using RMSprop optimizer with a learning rate of  $10^{-3}$  till 200 epochs on correctly labeled data. It can be seen that irrespective of the initialization scheme, when validation accuracy is high, the noise compresses significantly.

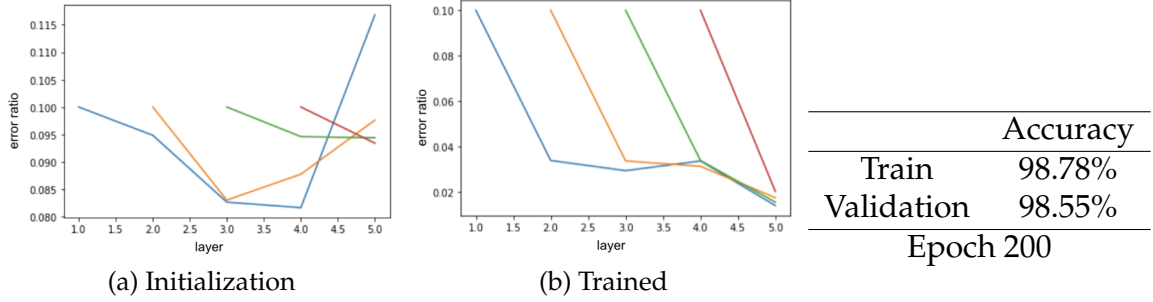


Figure 4.3: Noise stability: MNIST dataset, 4-layer MLP, Shuffled Trained initialization. Noise stability is observed for well trained network with Shuffled Trained initialization scheme.

**Compression with initialization from Scaled Normal Xavier scheme:** Scaling the weights of Xavier distribution can affect the final validation accuracy of the models. If the weights of Xavier’s normal distribution are scaled to very high values, it can diverge further on training, leading to no effective training. Conversely, if the weights are scaled down to very low values, training can be prolonged. Three scaling factors : (1) $f = 0.1$ , (2) $f = 20$  and (3) $f = 100$  are considered for Shell-1024 dataset, 4-layer MLP architecture. The noise stability profiles are as shown in Figure 4.4. The models are trained using SGD, Adam, and RMSprop optimizer, respectively, with a learning rate of  $10^{-3}$  till 200 epochs. It can be seen that at  $f = 0.1$  (although slow), the model still trains and displays stability against noise. For  $f = 20$ , although the training accuracy is very high, there’s a huge generalization gap. The validation accuracy is low, and the noise stability plot displays poor compression in this case. For  $f = 100$ , no effective training occurs. The noise stability plot clearly shows lower compression than in the first case.

## Results on convolutional neural networks

To observe noise stability in CNNs, CIFAR-10 dataset is trained on 16-layer CNN architecture (with dropout and batch-normalization layers) with an initialization from Normal Xavier scheme. It is trained for 100 epochs using RMSprop optimizer and a learning rate of  $10^{-3}$ . It can be seen from Figure 4.5 that the noise is compressed in this setup as well. The stability profile at 50 epochs also shows

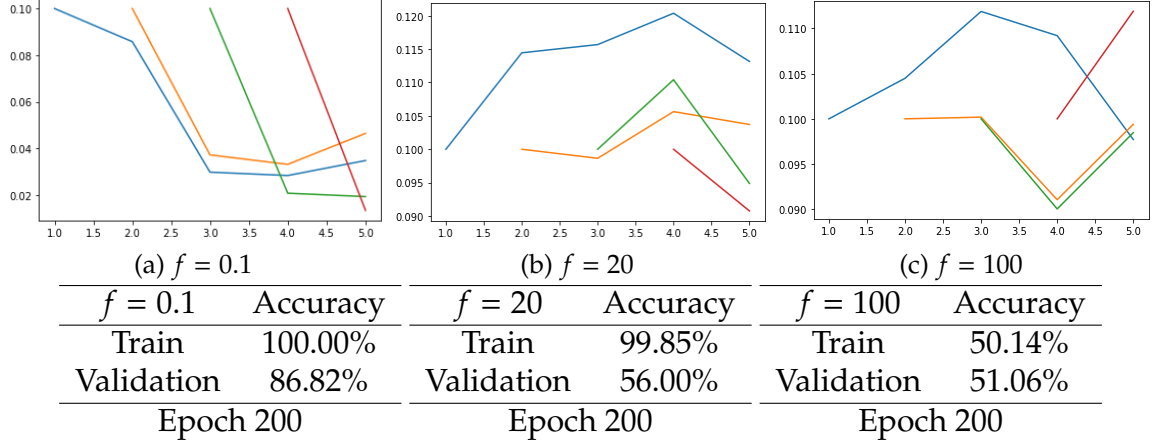


Figure 4.4: Noise stability: Shell-1024 dataset, 4-layer MLP, Scaled Normal Xavier initialization. Varying the scale of Normal Xavier initialization from  $10^{-1}$  to  $10^2$ .

compression (although lower than when trained further). This shows that significant attenuation occurs as training continues.

Refer Figure A.2 in Appendix for noise stability in other set-ups of CNN.

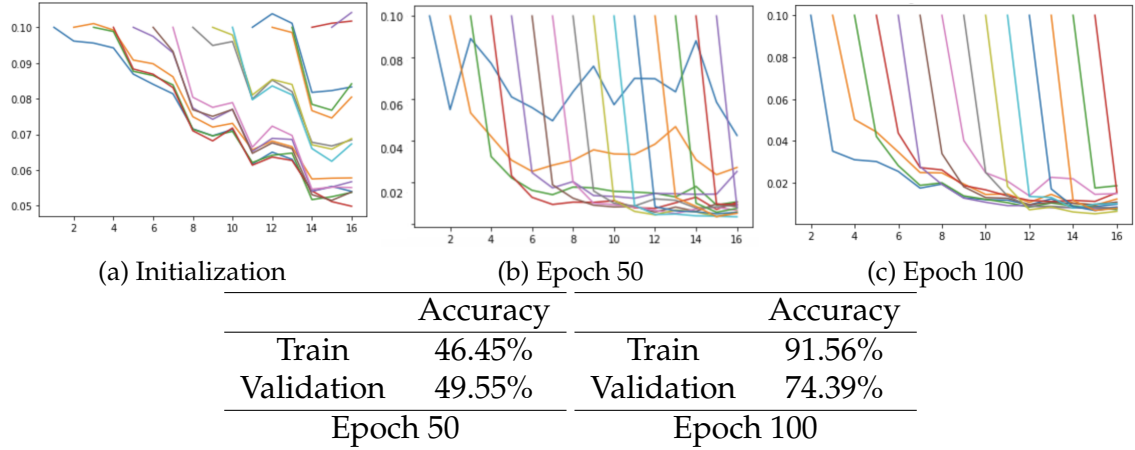


Figure 4.5: Noise stability: CIFAR-10, 16-layer CNN, Normal Xavier initialization. Effect of noise injected on higher convolutional layers gets attenuated as training progresses.

**Compression for Convolutional Neural Networks (Shuffled Labels):** Due to high training time of CNN on shuffled labels, a subset of CIFAR-10 dataset of 10,000 training examples is considered with 10-layer CNN architecture and initialization from Normal Xavier scheme. 60% of the labels are shuffled, and the model is

trained for 200 epochs using SGD optimizer and a learning rate of  $10^{-3}$ . Figure 4.6 compares the compression profile of two models trained on a subset of CIFAR-10 dataset with all correct labels and with 60% shuffled labels, respectively. There is a clear decline in the noise compression for models trained on shuffled data in the case of CNNs as well.

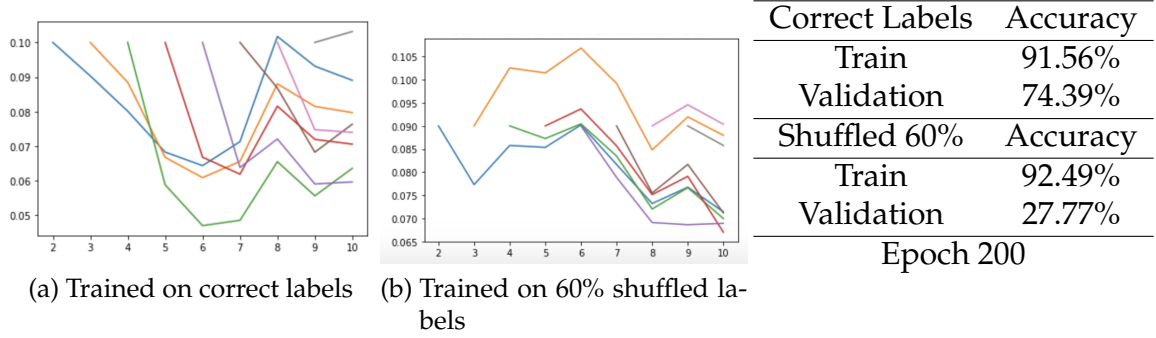


Figure 4.6: Comparison of noise stability of CIFAR-10 dataset, 10-layer CNN architecture, initialization from Normal Xavier scheme with correct and shuffled labels. Noise stability reduces for the second model.

The experiments demonstrated in this section show that a neural network with higher validation accuracy generally shows higher noise stability. Several datasets, architectures, and initialization schemes were considered for testing the robustness of noise stability. It has also been observed that noise stability for lower dimensional datasets shows a different profile than that for higher-dimensional data. Another interesting observation was observed from the initialization of the CNN architecture, which shows certain compression property during the initialization itself. This property is, however, not observed for the MLP architecture. Please refer Section A.1.1 in Appendix for further details.

## 4.2 Automatic dimensionality reduction

This section experimentally demonstrates that the singular value decomposition of the outputs of the hidden layers is characterized by a steep curve for trained neural networks, which can be helpful while predicting the success of a DAI.

The preferential directions in which signal is carried in well-trained networks

are utilized to build the early success indicators. Two metrics are introduced: (a) to capture the singular value distribution of the output from hidden layers of DNNs at a training step and (b) its evolution as training progress are introduced. Both of these metrics are combined to predict the success of a DAI early, enabling us to stop training of networks that are likely not to benefit from training further.

#### 4.2.1 Experiments on automatic dimensionality reduction

Consider a deep neural network of depth  $L$  with hidden layers  $(W_1, W_2, \dots, W_L)$  of dimension  $d_1, \dots, d_L$ . Let  $n$  test data points be passed through the hidden layers of the trained network and matrices obtained from the output of all hidden layers be of dimensions  $(n \times d_1), \dots, (n \times d_k)$ . The singular values of the matrices are obtained and normalized by the largest singular value and therefore lies in the range  $[0, 1]$ . The evolution of singular values over the epochs is studied for classification and regression on all the datasets mentioned in Section 3.

### Classification

The experiments below are done on classification of Shell-1024 dataset using MLP architecture.

**Validation accuracy and SVD outputs:** Consider two different networks (Shell-1024 dataset, initialization from Normal Xavier scheme) and (Shell-1024 dataset, initialization from Scaled Normal Xavier scheme - scaled by a factor of 20) with architecture consisting of only one hidden layer of size 128. The two networks show very different generalization gap, and their SVD profile is shown in Figure 4.7. Both are trained till 60 epochs; the optimizer used is SGD.

Figure 4.7 (a-c) are for initialization from Normal Xavier scheme. It obtains a validation accuracy of 81.04%. With each epoch, the distribution of SVD value compresses. In contrast, in Figure 4.7 (d-f) for initialization from Scaled Normal

Xavier scheme, the validation accuracy achieved is 51.92%, and the distribution of SVD values enlarges with subsequent epochs. Even the noise compression for the first network is much more significant. This illustrates that compression of SVD values over subsequent epochs can positively correlate with the neural network’s learning. Refer section A.2 in Appendix for SVD output for a deeper neural network which also demonstrates steep decay in each layer.

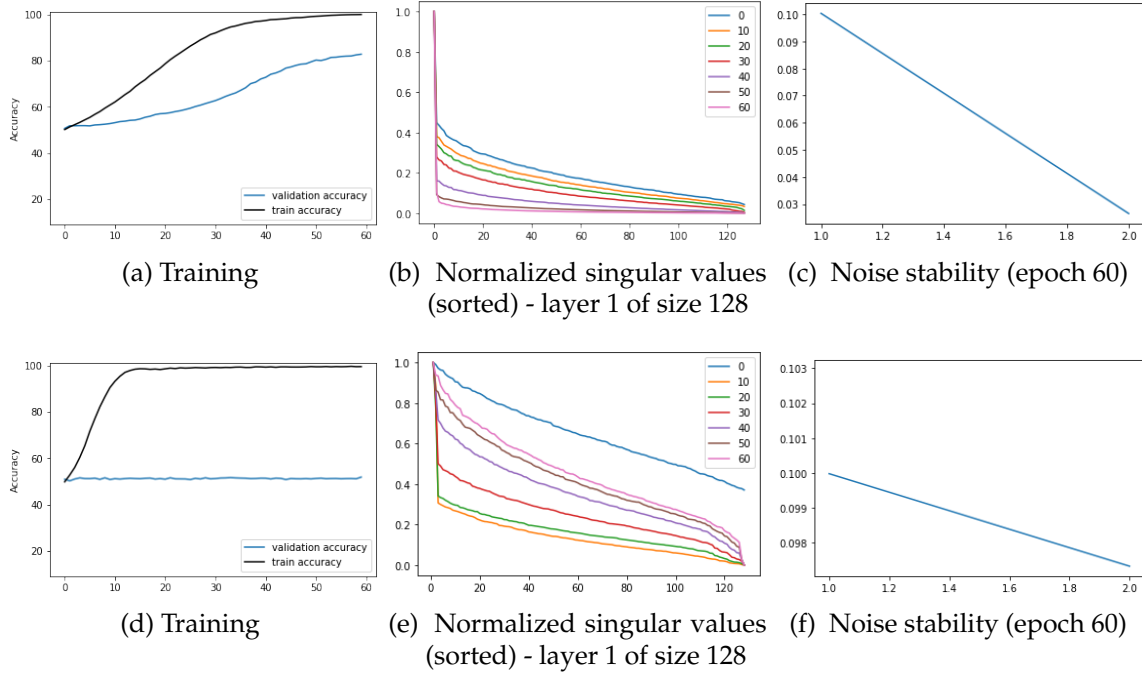


Figure 4.7: SVD output at various epochs for Shell-1024 dataset, and two different initializations. (a)-(c): initialization from Normal Xavier scheme. (d)-(f): initialization from Shuffled Trained scheme.

However, it is worth noting that compressed singular values do not necessarily imply a subspace in which the signal propagates effectively. There can be initializations of neural networks with extremely compressed singular values that do not optimize or generalize further on training. Consider Shell-32 dataset, and an architecture consisting of two hidden layers of sizes [128, 128]. No explicit regularization is added. The optimizer used is SGD, trained till 100 epochs, and the learning rate is  $(10^{-5})$ . Its SVD profile is shown in Figure 4.8. There is no effective training in this network, but still, SVD output is very compressed.



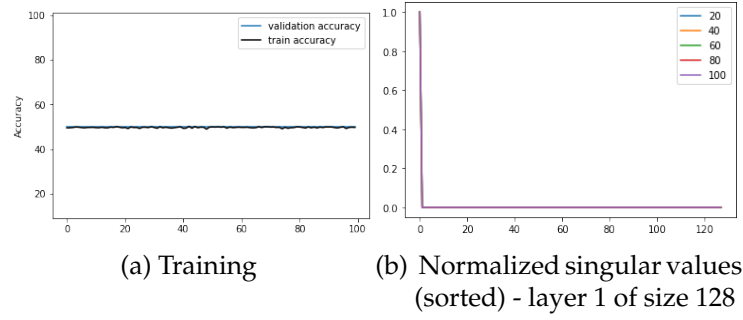


Figure 4.8: SVD output at various epochs for Shell-32 dataset. Steep decay of singular values with no effective training.

## Regression

This section studies the effect of decay of singular values on regression. Consider the function below:

$$y = \cos(x_1 + x_2) \times \sin(x_3 + x_4)$$

- $x$  is a 100 dimensional vector. ( $x^T = [x_1, x_2, \dots, x_{100}]$ ). Only the first four directions contribute to the signal.
- In this experiment,  $x$  is sampled from a unit sphere of 100 dimension. Due to this, the values of  $y$  are not uniformly distributed.

It is trained on an MLP architecture comprising of 4 hidden layers [256, 128, 128, 64] with no batch normalization, dropout layer. It is trained till 500 epochs, a learning rate of  $10^{-3}$ , and optimizer RMSProp. The singular values and noise stability output for this regression problem is shown in Figure 4.9 (singular value pattern of layer 3 and layer 4 is similar to layer 2). It can be seen that the validation loss reduces continuously, and the distribution of singular values in subsequent epochs shrinks for all layers in the regression problem as well. The effect, in this case, is more prominent in the first hidden layer.

Refer section A.2.1 in Appendix for SVD profile and noise stability in regression for Cosine of Sum dataset and Sum of Cosines dataset.

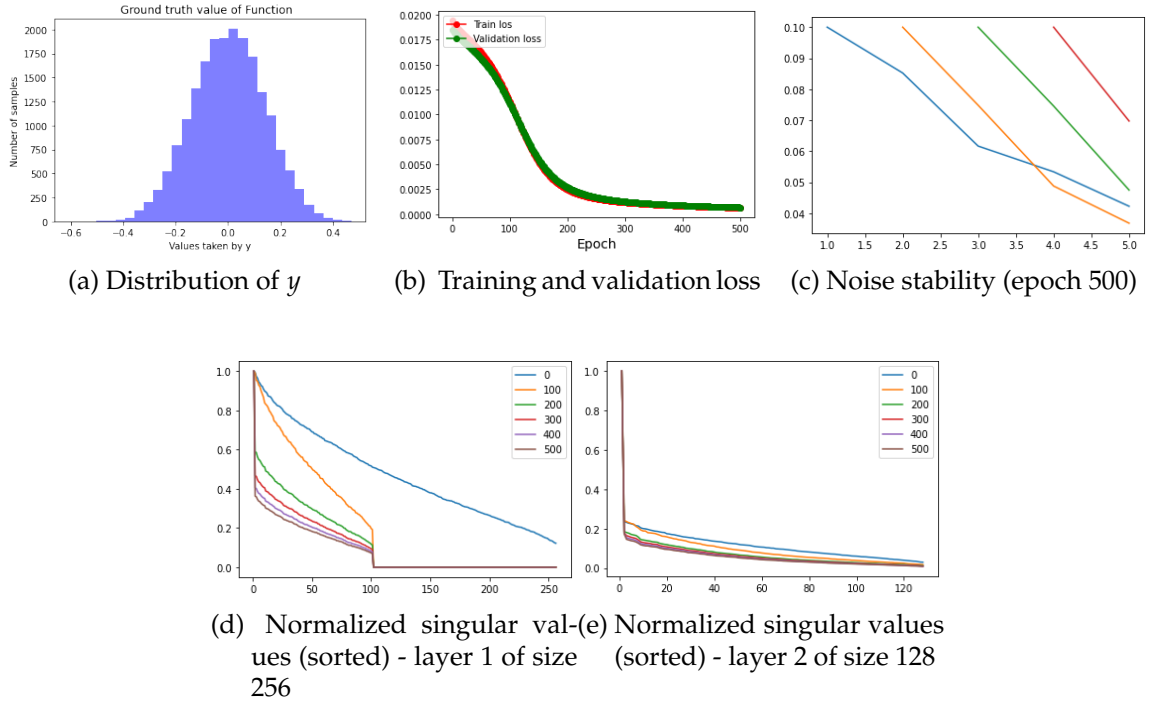


Figure 4.9: SVD output at various epochs and noise stability output on a regression problem. As the validation loss reduces, preferential directions are strengthened.

**Validation accuracy and singular values:** When trained on shuffled data as in section 4.1.1, it is observed that the validation accuracy rises and then begins to fall with further training. This section studies how the output from SVD accounts for such variability in the validation accuracy. In this experiment, 40% of labels are shuffled for Shell-32 dataset. Architecture consists of two hidden layers of dimension  $\{128, 128\}$  and initialization from Normal Xavier scheme. The optimizer used is SGD with a learning rate  $10^{-3}$ . The results are presented in Figure 4.10.

When the labels are shuffled, the validation accuracy is higher than the training accuracy for some time but eventually goes below training accuracy with further training. Interestingly, the SVD value plot for each layer also displays such variation. The values were initially decaying, but with further training, it expands again. The SVD decomposition starts reverting its effect as validation accuracy dips further. This phenomenon can be crucial in detecting overfitting in models in the absence of validation labels.

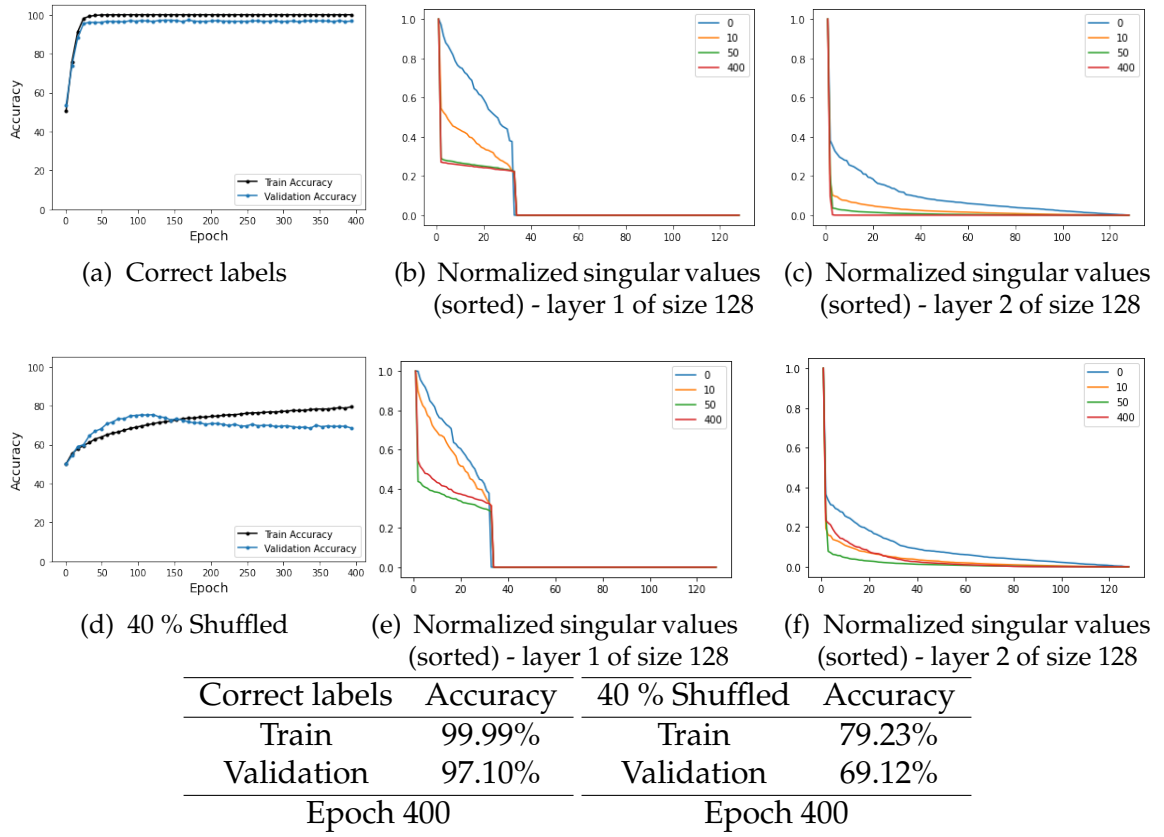


Figure 4.10: SVD output at various epochs for (a)-(c) correct labels, (d)-(f) shuffled labels on Shell-32 dataset. Trend in distribution of singular values follows the trend in the validation accuracy.

## CHAPTER 5

### Scores for early success indicators

This section aims to determine the success of a DAI during the initial stages of training using the tools developed in Section 4. In the last section, it is shown that well-trained networks often show a high degree of noise stability. It is also characterized by a steep decay of SVD value as training progresses. Consider the evolution of two DAIs in Figure 5.1 and Figure 5.2 respectively (SVD output for layer 3, layer 4 is similar to layer 2). Eventually, the first network achieves good validation accuracy, while the second fails to learn. The output from SVD values of hidden layers and noise stability value gives insights about the final validation accuracy at an earlier stage.

It can be seen that the first network demonstrates steep decay of SVD values, even at the 50th epoch. Furthermore, the generalization gap of the second network is much higher than the first network, which does not show stability against noise at epoch 50. The SVD curve and noise stability are more prominent in the first network, which has better validation accuracy. As training progresses, the SVD values from the first network decay further, in contrast to the second network. Noise stability also follows a similar pattern.

In this section, a quantifying mechanism for these characteristics is devised. Three scores are introduced to quantify the performance of a network at a particular epoch.

**Notations:** The following notations are used for the task of classification with  $k$  classes: Consider a dataset  $\{(x_i, y_i)\}_{i=1}^n$ , where  $x_i \in \mathbb{R}^d$ , label  $y_i$  is an integer between 1 to  $k$ . A multi-class classifier  $f$  transforms an input  $x$  in  $\mathbb{R}^d$  to  $\mathbb{R}^k$ . Let  $f$  be a feed-forward neural network of depth  $L$  parameterized by weight matrices  $W_L, \dots, W_1$ , where dimension of each hidden layer  $i$  be  $d_i$ , such that  $W_k \in \mathbb{R}^{d_k \times d_{k-1}}$ .  $(X, Y)$  be

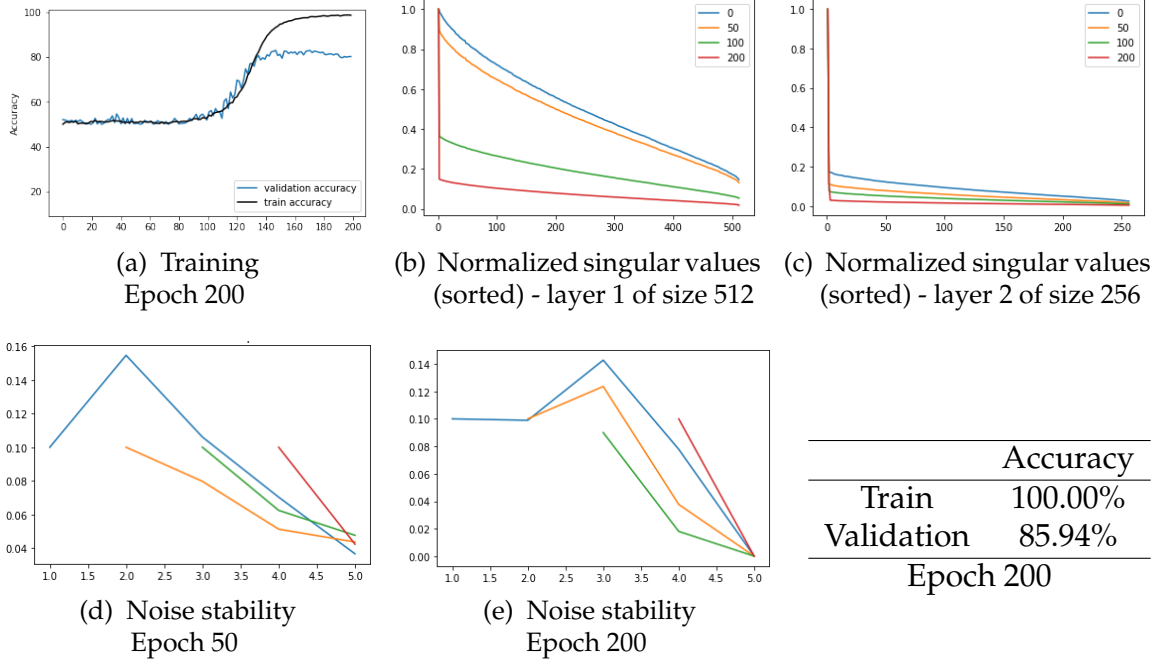


Figure 5.1: Evolution of SVD and noise stability for Shell1-1024 dataset, 2-layer MLP architecture, initialization from Normal Xavier scheme.

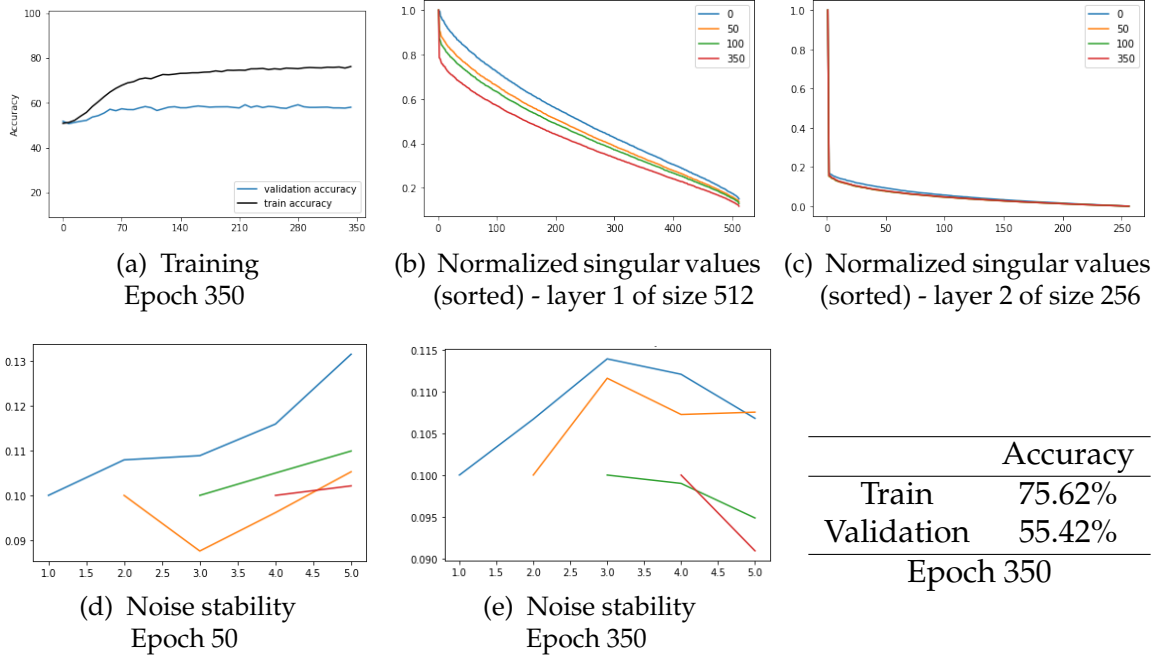


Figure 5.2: Evolution of SVD and noise stability for Shell1-1024 dataset, 2-layer MLP architecture, initialization from Scaled Normal Xavier scheme (Scale 10).

the test dataset with  $m$  samples  $\{(x_{ti}, y_{ti})\}_{i=1}^m$ .  $A_i$  be the matrix obtained on passing the test dataset through the  $i^{th}$  hidden layer of the trained network.  $\{\sigma_{tij}\}_{j=1}^{d_i}$  is the singular values obtained in descending order from the matrix  $A_i$  at epoch  $t$ . For a hidden layer  $l$  at epoch  $t$ , three metrics are defined  $s_{nlt}$ ,  $s_{olt}$ , and  $s_{slt}$ . These scores are computed over a window of the last  $t_0$  epochs as described in the sections below. The same notations are used for the task of regression as well.

### Capturing noise stability - $s_{nt}$

Score  $s_{nt}$  quantifies the degree of noise stability for MLP/CNN architecture. Standard experimental setting of adding noise with  $l_2$  norm as 10% of the norm of input is maintained for calculating the score.

$$s_{nt} = \frac{2}{(L)(L+1)} \times \frac{1}{\alpha_n \times \sum_{i=1}^L \sum_{j=i+1}^{L+1} y_{ij}}$$

- $\alpha_n$  is a hyperparameter for scaling the values of noise compression. (1 for all experiments below).
- $y_{ij}$  is the norm of the normalized difference of the output of layer  $j$  when noise is added to layer  $i$  ( $i < j$ ).
- The higher the value of  $s_{nt}$ , the better the noise stability as  $y_{ij}$  is relatively lower for models showing better noise stability.

### Capturing steep decay of the SVD values - $s_{ot}$

Metric  $s_{ot}$  captures the non-uniform distribution of singular values and the preferential directions for signals which can compress noise in other directions.

$$s_{olt} = \frac{\beta}{d_l \times t_0} \times \frac{1}{\sum_{k=t-t_0}^t \sum_{i=1}^{d_l} \sigma_{kli}}$$

- $s_{olt}$  is the score for output of layer  $l$  at epoch  $t$ . An average score can be calculated across all layers as

$$s_{ot} = \frac{1}{L} \sum_{i=1}^L s_{olt}$$

- $\beta$  is a hyper-parameter that are chosen empirically. ( $10^7$  for all experiments below).
- The higher the value of  $s_{ot}$ , the higher the decay in the distribution of singular values. For a steep decay of singular values,  $\sigma_{kli}$  for any  $i > 1$  would result in small values with increasing  $i$  (as the highest singular value is scaled to 1). This would result in a higher value of  $s_{olt}$  for that layer. Therefore,  $s_{ot}$  can capture the non-uniform distribution of singular values.

A higher  $s_{ot}$  suggests that the distribution of singular values is skewed. Models which generalize well show a high value of  $s_{ot}$  during training.

## Capturing shift of SVD decay - $s_{st}$

$s_{st}$  quantifies the automatic dimensionality reduction phenomenon across epochs. It captures the shift in the non-uniform distribution of singular values as training progresses.

$$s_{slt} = \frac{1}{(t_0 - 1) \times d_l} \times \sum_{k=t-t_0+1}^t \sum_{j=1}^{d_l} \alpha_k (\sigma_{(k-1)lj} - \sigma_{klj})$$

- $s_{slt}$  is the score for output of layer  $l$ . An average score can be calculated as

$$s_{st} = \frac{1}{L} \sum_{i=1}^L s_{slt}$$

- $\alpha_k$  is a hyper-parameter that are chosen empirically. It is an increasing parameter with  $k$ . It varies linearly, giving highest weightage to the latest epoch and least weightage to the last epoch of the window.  $\alpha_k$  lies in the range  $[0, 1]$ .
- The higher the value of  $s_{st}$ , the more the shift towards decay.

As the validation loss of a model reduces, its transformation matrix gets aligned with the input data leading to further skewness and positive value in  $s_{st}$ .

The scorings  $s_{st}$  and  $s_{ot}$  are normalized across different feed-forward architectures by normalizing the score for each hidden layer by the number of units in the layer. Parameter  $\beta$  is used to scale  $s_{ot}$  in the range of  $s_{st}$ , which is crucial when combining the two scores. These two metrics are combined, and a scoring is proposed, independent of the labels of the test dataset, which can be used as an early indicator of predicting the success of a DAI trained using gradient descent. The three examples below demonstrate how the scoring mechanism can be used for predicting the success of a DAI. It is shown that the scoring as outlined above can draw insights into the success or failure of a DAI. Consider three cases- DAI-1, DAI-2, DAI-3 : Shell-1024 dataset, 4-layer MLP architecture, initialized from initialization from Normal Xavier scheme. It was shown earlier that networks with the same initialization distribution could have very different generalization gaps and training accuracy.

Figure 5.3 shows output from DAI-1. It can be seen that the network learns (validation accuracy greater than random guessing) after epoch 25 roughly. The SVD decay becomes steeper, and further noise stability can be observed. It is characterized by high values for  $s_{nt}$ ,  $s_{ot}$ , and  $s_{st}$ .

Figure 5.4 shows output from DAI-2. It can be seen that the network learns (validation accuracy greater than random guessing) after epoch 60 roughly (much later than DAI-1, even though both have the same initialization distribution). However, in this case, the SVD decay and noise compression scores show high positive values (even before epoch 60), indicating that successful training is possible and training should be continued.

Figure 5.5 shows output from DAI-3. The network does not demonstrate effective learning till epoch 100 as the final validation accuracy is 50.00%. This is characterized by stillness in the distribution of singular values and low noise stability. This is accompanied by a small  $s_{nt}$ ,  $s_{ot}$ , and  $s_{st}$  at epoch 30, indicating that this model can be discarded early.



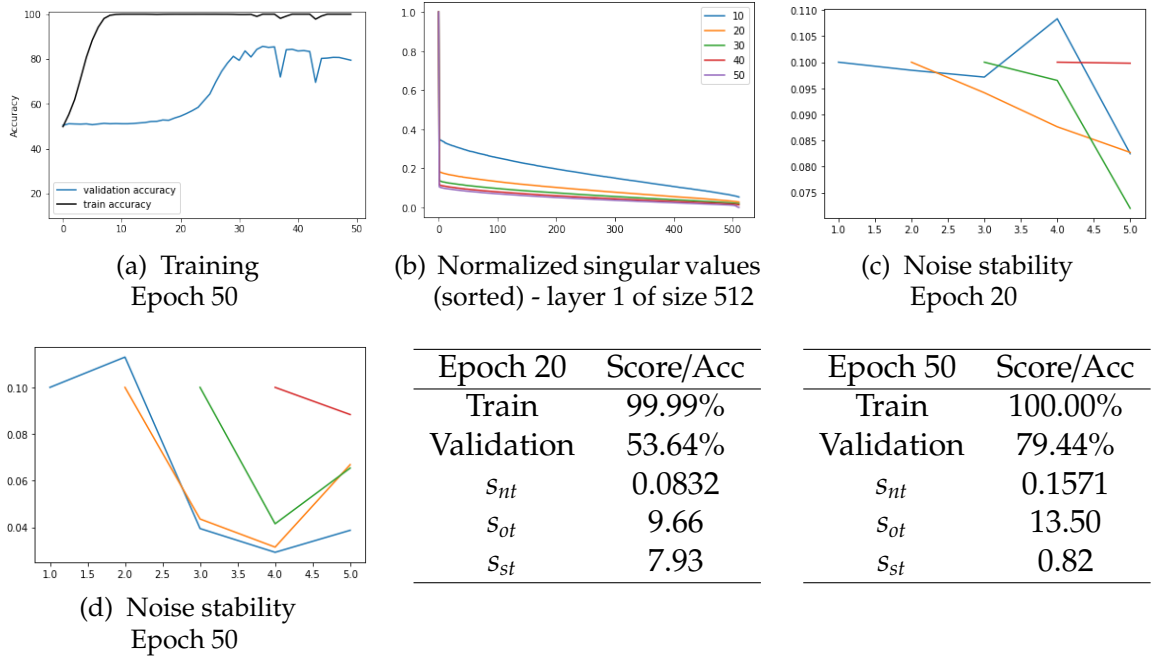


Figure 5.3: DAI-1: attains high validation accuracy along with high  $s_{nt}$ ,  $s_{ot}$ ,  $s_{st}$ .

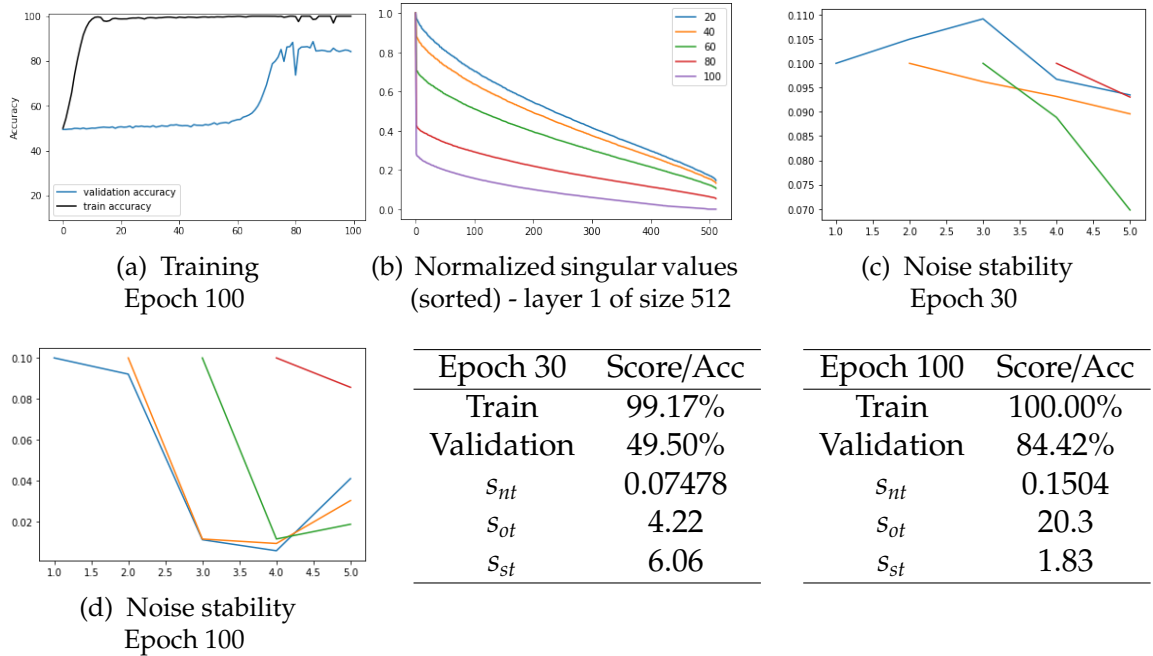


Figure 5.4: DAI-2: attains high validation accuracy along with high  $s_{nt}$ ,  $s_{ot}$ ,  $s_{st}$ , at epoch 30, correctly indicating high chances of success of further training.

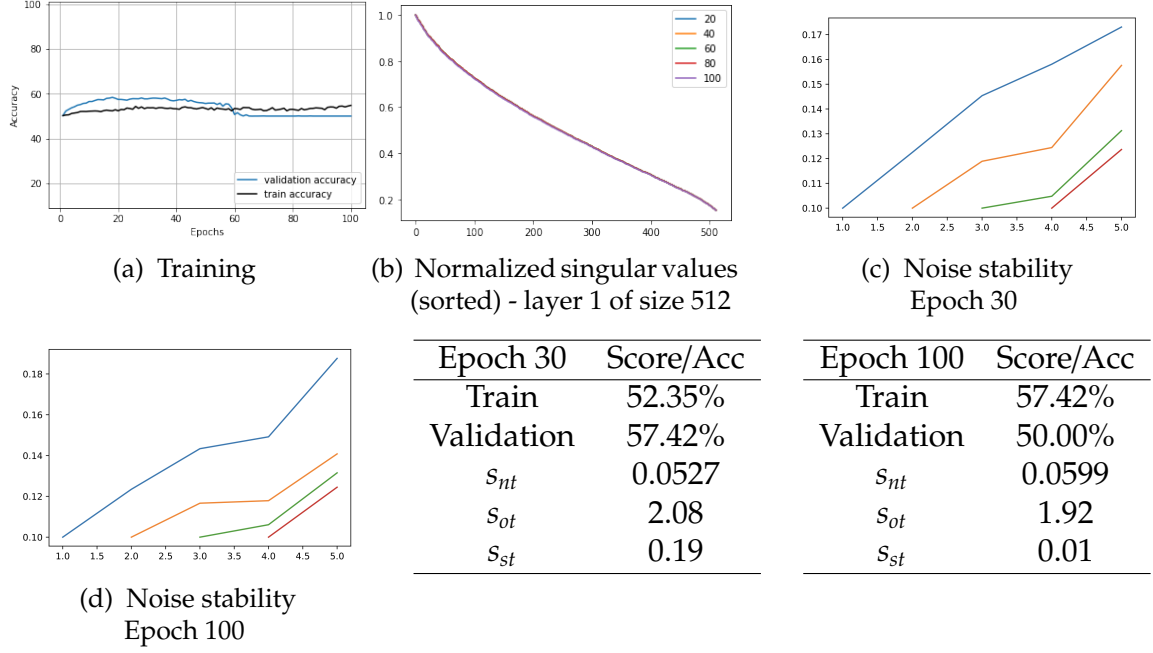


Figure 5.5: DAI-3: attains low validation accuracy along with low  $s_{nt}, s_{ot}, s_{st}$ , at epoch 30, correctly indicating low chances of success of further training.

As shown in the above examples, these scores can provide insights on continuing training for different models. The models which do not score high do not train well generally.

## 5.1 Predicting success of initializations for a dataset and architecture

The metrics described above are combined to propose a scoring, independent of the labels of the test dataset, which can be used as an early indicator of predicting the success of a DAI trained using gradient descent.

For a particular DAI combination, an early success indicator calculates the score by combining the two metrics  $s_{ot}$  and  $s_{st}$  at every epoch  $t$ . This combined score is used to make a binary decision on whether to continue training on that DAI or discard the initialization choice. If the score crosses a certain threshold  $t_1$ , the model is trained further. However, if the score is below  $t_1$ , the model is discarded

before completing the training procedure. This can allow several initializations to be tested for different datasets and tasks with lesser time and computational power. The final score is calculated by combining the two metrics:

$$s_t = \log(s_{ot}) + \eta \times s_{st} \quad (5.1)$$

$\eta$  is a hyper-parameter used to take a weighted average of the two terms in the combined score. It is worth noting that compressed singular values does not necessarily imply a subspace in which the signal propagates effectively (as shown in Figure 4.8). To discard such models which do not train at all, two more thresholds  $t_2$  and  $\delta$  are introduced. These thresholds allow for removing models which have compressed SVD values and still show very small changes in  $s_{st}$ . This phenomenon is usually accompanied by non-decreasing validation loss. Training these models further does not change the distribution of SVD values and therefore the training could be stopped earlier. Considering these factors, the final combined score  $s_F$  is summarized as follows:

$$s_F = \begin{cases} 0 & \text{if } s_{ot} \geq t_2 \text{ and } s_{st} \leq \delta \\ s_t & \text{otherwise} \end{cases} \quad (5.2)$$

While the lower threshold ensures that the models whose SVD values do not attain a minimum level of compression are discarded, the upper threshold is used to discard models which do not train at all. To distinguish models whose SVD values decay but it does not train from the models which train successfully and obtain a steep decay of SVD values, score  $s_{st}$  is used. Since models which do not learn effectively do not change significantly on further training.

The scores in figure 5.3, 5.4, and 5.5 demonstrates how scores based on singular values can provide insight on the training of neural networks. The first DAI shows high values for all three scores at epoch 20. As training saturates at epoch 50, the score  $s_{st}$  goes down, indicating that the benefits of training further are low. The second DAI, obtained from the same distribution, attains only 49.50% validation

accuracy at epoch 30. However, the scores are still indicative of successful training. It can be seen that the DAI trains to a validation accuracy of 84.42%. Through experimentation, this work claims that the insight obtained from this distribution of singular values gives a prescience of the DAI's performance and thereby enables us to only train models with higher chances of generalizing well.

The above scores are calculated on validation data points but do not require the labels of the data points. However, the scores can be improved with the availability of labels for the validation dataset. The validation accuracy at a particular epoch  $a_v$  is incorporated into the scoring mechanism to propose a new score  $s_{Fv}$  as follows:

$$s_{Fv} = \begin{cases} a_v & \text{if } s_{ot} \geq t_2 \text{ and } s_{st} \leq \delta \\ a_v \times (1 + \min(\frac{1-a_v}{a_v}, \gamma s_t)) & \text{otherwise} \end{cases} \quad (5.3)$$

$s_{Fv}$  lies in the interval  $[a_v, 1]$ . It is an indicative measure of the final validation accuracy of the initialization based on the current validation accuracy  $a_v$ .  $\gamma$  is a hyper-parameter to scale the score  $s_t$ , and is constant for all DAIs.

The goal of early success indicators  $s_F$  and  $s_{Fv}$  is to predict which DAIs would be successful. A good initialization is defined as the weight matrices which yield a good validation accuracy upon training for a fixed dataset, architecture, and training procedure. Obtaining a good neural network model is heavily affected by choice of initial parameters. The initial point has an influence on whether or not the learning process converges. In cases where it converges, initialization can affect how fast or slow the convergence happens, the error, and the generalization gap. There is no universal initialization technique as modern initialization strategies are heuristic [4]. The current understanding of how the initial point affects optimization and generalization is still incipient, and therefore, in practice, choosing good initialization is hugely experimental. The model needs to be trained for several epochs (till either the loss reaches a certain threshold or overfitting begins) to determine the success of initialization on a particular architecture and dataset. This process consumes significant compute power and time and can limit the number of initializers to be tried out. Therefore, these scores can, to a reason-

able extent, determine the success of a DAI during the early training process and provide insights on the benefits of further training.

In the experiments detailed in section 6, two cases are considered: with and without the availability of validation data labels at checkpoint epoch (the epoch at which the decision regarding further training is made), denoted by  $s_{Fv}$  and  $s_F$  respectively. To calculate  $s_F$ , training accuracy,  $s_{nt}$ ,  $s_{ot}$ , and  $s_{st}$  are available at the checkpoint epoch. Training accuracy is not considered for calculating  $s_F$  as it alone can not account for the generalization gap. A higher value of  $s_F$  and  $s_{Fv}$  indicates better compression and can potentially indicate better performance of the model. Inclusion of  $s_{nt}$  (for noise stability) gave inferior predictive performance than  $s_F$ , so it is dropped from final prediction score.

**Baseline:** To evaluate the performance of  $s_F$  and  $s_{Fv}$ , the baseline in the experiments is the training accuracy and the validation accuracy at checkpoint epoch, respectively. In the absence of early success indicators, judgment on further training of a DAI relies on the training accuracy alone (or validation accuracy when the labels are available). Therefore, the utility of these indicators is based on performing better than the mentioned baseline.

## 5.2 Improving Scoring

This section briefly highlights some improvement techniques that can be used to enhance the scores  $s_F$  and  $s_{Fv}$ .

1. Decaying weights of units: For calculating  $s_{ot}$ , all the singular values are taken for a hidden layer output. However, the values of singular values go down rapidly, and the majority of the contribution to the score is delivered by the first few singular values. Each singular value's weight (sorted in descending order) is varied linearly from 1 to 0 to enhance this effect.
2. Non-linear difference: The decaying weights of units in the output of hidden layers motivates non-linear difference in calculating  $s_{st}$ , with higher weightage to the difference in the initial singular values. These weights are also varied linearly from 1 to 0.

3. Soft-threshold: Instead of hard-thresholding and completely discarding models with  $s_{ot} \geq t_2$  and  $s_{st} \leq \delta$ , a soft thresholding can be introduced to lower the false-negative predictions. One such threshold used in  $s_F = \frac{t_2}{s_{ot}} \times t_2$  for  $s_{ot} \geq t_2$  and  $s_{st} \leq \delta$ .
4. Weighted average across layer: As shown in the section below, experimentation on deeper networks reveals that not all layers may show the compression properties in the trained DNNs. Instead of taking a linear average across all the layers, different weights can be given to each layer. In this experimental setting, average over only the best  $n$  layers was tested. Since this  $n$  is dependent on the number of hidden layers in the architecture, it has not been used in the results in section 6.

## Problems with deep nets

All layers in a deep neural network may not necessarily show a shift towards higher decay in the singular values as training progresses. For instance, consider the Shell-1024 dataset trained on a 8-layered MLP (hidden units of size [256, 256, 256, 256, 128, 128, 64, 64] respectively). It attains a training accuracy of 98.6% and validation accuracy of 70.2% after training till 60 epochs.

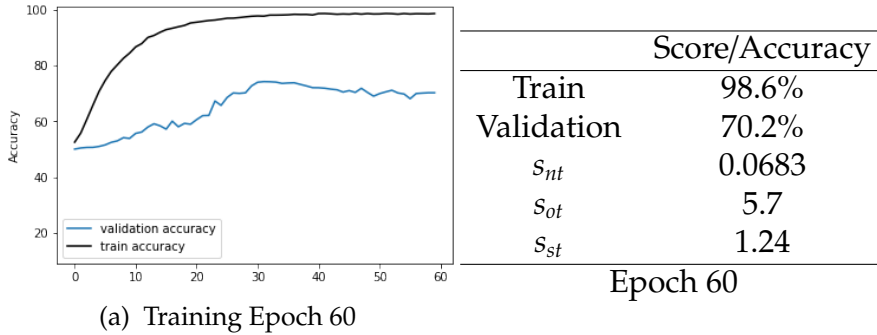


Figure 5.6: Shell-1024 dataset trained on a neural network architecture of 8 hidden layers obtains low scores on  $s_{st}$  and  $s_{ot}$ .

It can be seen that the last 4 hidden layers show very little change in  $s_{slt}$ , while the first 4 layers have a high  $s_{slt}$  score. Linear averaging can bring the score of the DAI down, and this creates a need to set different thresholds for different architectures.

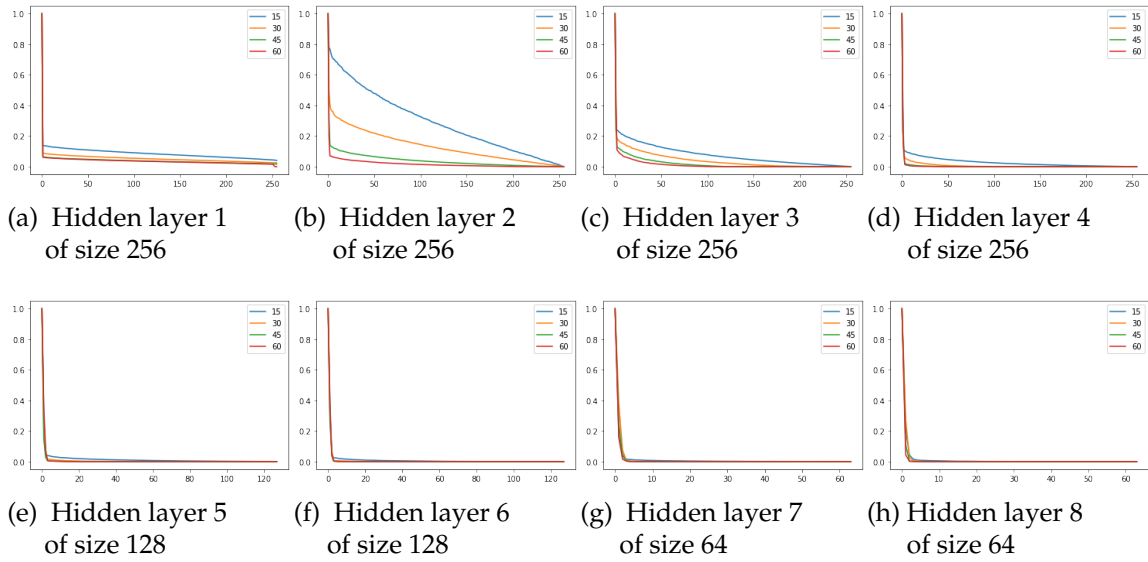


Figure 5.7: Normalized singular values (sorted) for all hidden layers at various epochs. Profile of singular value decay can be different for all layers which can reduce the average score for deep neural networks.

## CHAPTER 6

### Evaluation of early success indicators

In this section, different sets of dataset and architecture combinations are chosen, and various initializations are tested on it. Checkpoints are defined as epochs at which the scores are calculated, and a decision regarding the success of the initialization is made. The decision for a DAI can be one among the three: discarding the models that do not train early, stopping further training for trained models (because training further won't yield further benefits), and continuing training for better results. This decision is taken only based on the values from  $s_F$  (and  $s_{Fv}$  when validation labels are available) at the checkpoint epoch. All initializations are further trained till a pre-decided final epoch, and the prediction at the checkpoint epoch is compared against the validation accuracy at the last epoch. Spearman's correlation is calculated between the score obtained from the early predictor at the checkpoint epoch and validation accuracy at the final epoch.

**Spearman's Rank Correlation Coefficient:** It evaluates the monotonic relationship between two ordinal or continuous variables. While Pearson's correlation is used to capture linear correlation between two variables, Spearman's rank correlation can be used when the variables tend to change together, but not necessarily at a constant rate. The Spearman correlation coefficient is based on the ranked values for each variable. It is defined as the Pearson Correlation Coefficient between the rank variables. Mathematically, for two variables  $X$  and  $Y$ , it is given as:

$$r_s = \rho_{rg_X, rg_Y} = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X} \sigma_{rg_Y}} \quad (6.1)$$

The notations are as follows:

- $r_s$  is the Spearman's Rank Correlation coefficient
- $rg_X$  and  $rg_Y$  are ranks of  $X$  and  $Y$



- $cov$  is the covariance
- $\sigma_{rgx}$  is the variance of the rank variable of  $X$

Correlation coefficients vary between -1 and +1, with 0 implying no correlation. Correlations of -1 or +1 imply a strong relationship between the variables. Positive correlations suggest that as  $x$  increases, so does  $y$ . Negative correlations indicate that as  $x$  increases,  $y$  decreases. Spearman’s rank correlation between the scores calculated at the checkpoint epoch and the final validation accuracy is calculated to show that the two share a positive relation. A high positive correlation indicates that the proposed scoring can be a good predictor for the final validation accuracy. The performance of  $s_F$  and  $s_{Fv}$  is compared against the baseline training ( $BT$ ) and validation ( $BV$ ) accuracy at the checkpoint, respectively.

**Experiment Setting:** The experiments in section 6.1 and 6.2 on multi-layered perception are done on the following two dataset and architecture pair :

- **DA-1 :** Shell-1024 dataset, 4-layer MLP architecture
- **DA-2 :** Shell-512 dataset, 2-layer MLP architecture

Initializations are taken from all the schemes defined in section 3.3. Additional experiment on real-world MNIST dataset is summarized in the Appendix. The experiments in section 6.3 on CNN are done on the following DA :

- **DA-3 :** CIFAR-10 dataset, 8-layer CNN architecture

Several initializations are trained for each of the two DAs. These initializations are generated by randomly varying the standard deviation of the weights in Xavier’s initialization within a factor of 10. All the models are trained using the same training mechanism.

## 6.1 Failure of Noise stability

Noise stability, captured by  $s_{nt}$ , is characterized by several false positives, i.e., models that have high score  $s_{nt}$  but do not train to be successful. For the two DAIs, figure 6.1 shows that the Spearman’s correlation is negative, and thus,  $s_{nt}$  has not been used further for devising the early success indicators.

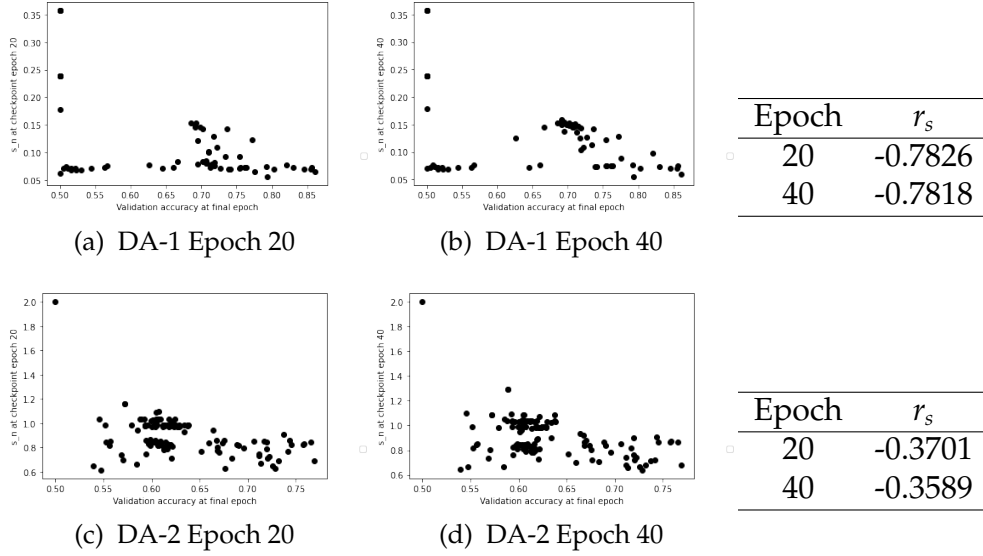


Figure 6.1: Performance of  $s_{nt}$ :  $s_{nt}$  at checkpoint epoch shows a negative correlation with the final validation accuracy for DA-1 and DA-2.

## 6.2 Results on multilayer perceptron

The scores  $s_F$  and  $s_{Fv}$  are calculated at checkpoint epochs, and it is compared with the final validation accuracy at the final epoch. Evaluation is done using two metrics: Spearman’s correlation and the number of correct predictions for the success of a DAI. An arbitrary threshold is set for each DA to mark successful initializations. Prediction of the success of a DAI is formulated as a two-class problem to quantify the performance of the ESI. Initializations that achieve final validation accuracy above a desired threshold belong in class  $y_1$  of successfully trained models. Initializations that do not attain this threshold upon full training belong to class  $y_0$  of not successfully trained models. A prediction is made at the checkpoint epoch

to predict the class in which the model trained of that initialization would belong in the final epoch.

### DA-1 (Shell-1024 dataset, 4-layer MLP architecture)

A total of 120 initializations are trained on this DA. Initializations that achieve validation accuracy above 60% at the final epoch are considered to be in class  $y_1$  and all the other initializations in class  $y_0$ . All initializations are trained to a total of 100 epochs to evaluate the performance of the metrics.

Figure 6.2 shows the distribution of final training accuracies and validation accuracies for DA-1 at epoch 100. The points right to the red line indicates models which qualify to be in class  $y_1$ .

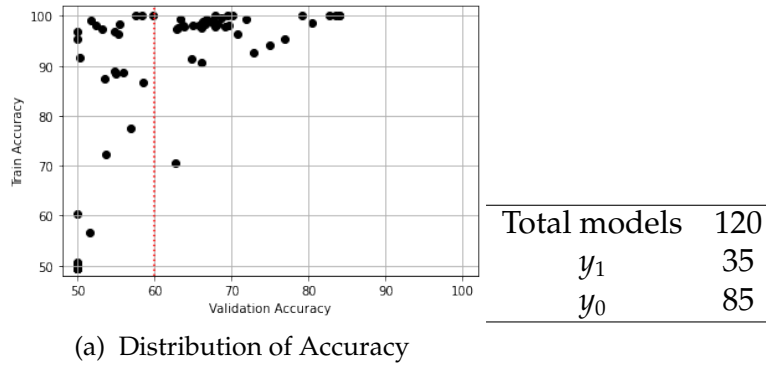


Figure 6.2: Final training and validation performance of 120 different initialization for DA-1.

Figure 6.3 shows the value of validation accuracy,  $s_F$ , and  $s_{Fv}$  at every epoch for five initializations. At checkpoint 30, the points with  $s_F$  greater than 2.25 are predicted to train successfully. The models with a horizontal line in the second curve (initialization 5) represent models whose score does not change much. While  $s_F$  can determine that their accuracies might not change with further training, it can not determine the actual accuracy. This limitation is mitigated when using  $s_{Fv}$ .

A prediction is made whether the model would belong to class  $y_0$  or  $y_1$  at epoch 100 using these scores at checkpoint epoch 10. Table 6.4 compares the performance

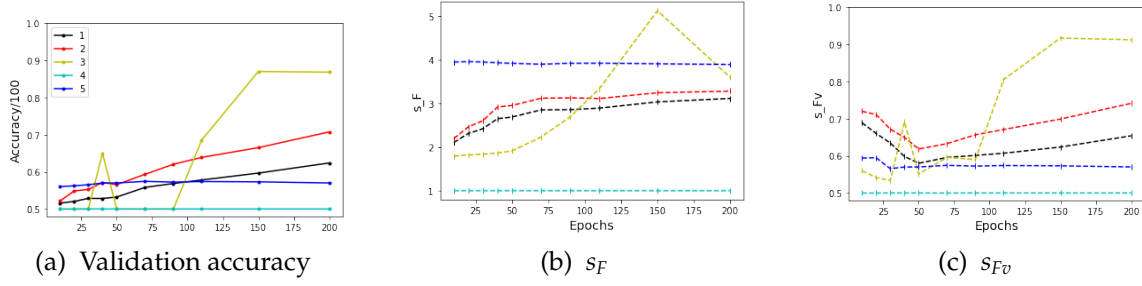


Figure 6.3: Variation of scores  $s_F$  and  $s_{Fv}$  with epochs and their corresponding validation accuracy for five initializations of DA-1.

of  $s_{Fv}$  and validation accuracy at the checkpoint to make a binary prediction of the success of initialization at the checkpoint. It can be seen that  $s_{Fv}$  is better at predicting both trained and not trained networks compared to simply using the validation accuracy.

		Prediction		Total			Prediction		Total
		$y_1$	$y_0$				$y_1$	$y_0$	
True	$y_1$	27	8	35	True	$y_1$	20	15	35
	$y_0$	3	82	85		$y_0$	4	81	85
Total		30	90	120	Total		24	96	120

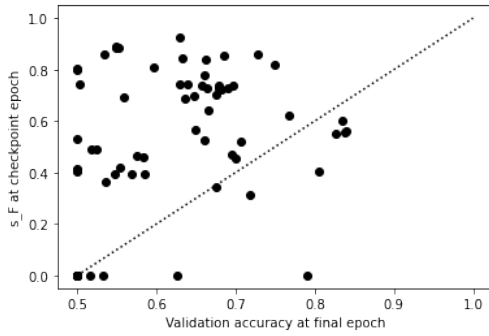
(a) Prediction using  $s_{Fv}$       (b) Prediction using  $BV$

Figure 6.4: Breakdown of the prediction of eventual success of all initializations in DA-1 using  $s_{Fv}$  and  $BV$  at epoch 10.

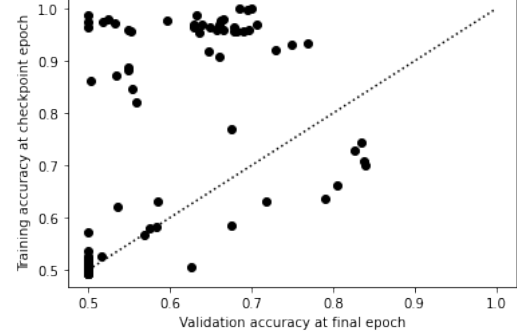
The correlation of the four different measures at checkpoint epoch 30 and final validation accuracy at epoch 100 is shown in figure 6.5. It can be seen that using training accuracy alone to make a judgment at checkpoint epoch can lead to wrong decisions in several cases. Since training accuracy alone can not determine the generalization gap, it can not provide accurate insights about the final validation accuracy. A low training accuracy at the checkpoint epoch, although indicative of low validation accuracy at checkpoint too, can not provide information about the final training and validation accuracy. Score  $s_F$  does not use training accuracy/loss at the checkpoint epoch. Without the labels, the score provides a positive correlation value with the final validation accuracy.

Likewise, using the validation accuracy at the checkpoint epoch to predict the validation accuracy at the final epoch can lead to several misclassifications. A

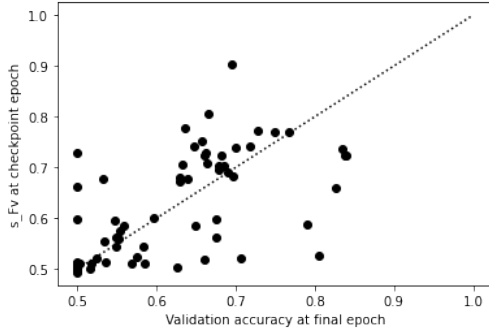
low validation accuracy can either stay at low validation accuracy or get well-trained as well. A high validation accuracy can also either yield a lower final validation accuracy (over-fitting) or get better trained. Score  $s_{Fv}$  supplements this with additional information, boosting the validation accuracy score that has the potential to train further and a low  $s_{st}$  to indicate stopping the training process.



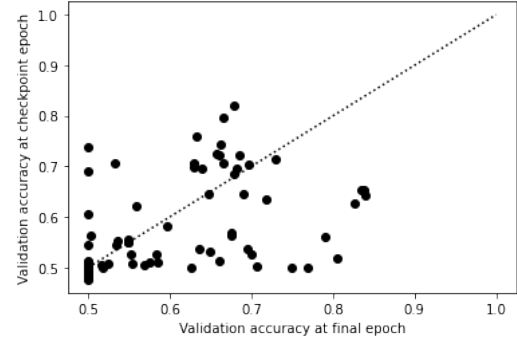
(a)  $s_F$  and final validation accuracy



(b) Train accuracy at epoch 30 and final validation accuracy



(c)  $s_{Fv}$  and validation accuracy



(d) Validation accuracy at epoch 30 and 200

Figure 6.5:  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at checkpoint epoch 30 plotted against the final validation for all initializations in DA-1.

Checkpoint	$s_F$	Training accuracy at checkpoint	$s_{Fv}$	Validation accuracy at checkpoint
10	0.478	0.513	0.551	0.508
20	0.605	0.594	0.673	0.615
30	0.708	0.711	0.746	0.704

Table 6.1: Correlation of  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at different checkpoint epochs with the final validation accuracy for DA-1.

Table 6.1 summarizes the Spearman's correlation values between baseline/scores and final validation accuracy. It can be seen that score  $s_{Fv}$  is better than the baseline

of validation accuracy at the checkpoint at all checkpoints.

### Shell-512 dataset, 2-layer MLP architecture

A total of 137 initializations are trained on this DA. Figure 6.6 shows the distribution of final training accuracies and validation accuracies for DA-2. Initializations that achieve validation accuracy above 65% at the final epoch are considered to be in class  $y_1$  and all the other initializations in class  $y_0$ . The points right to the red line indicates models which qualify in class  $y_1$ .

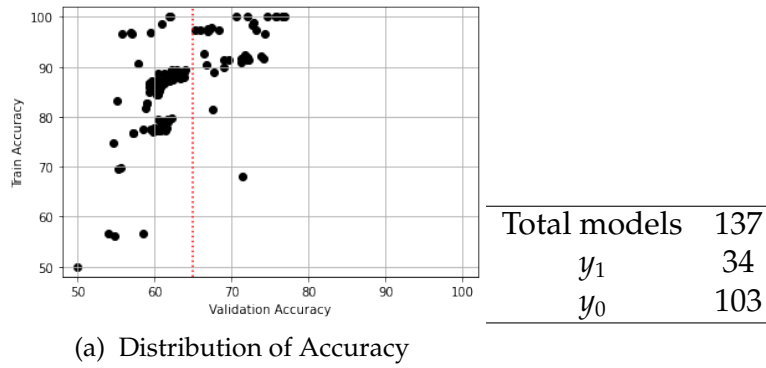


Figure 6.6: Final training and validation performance of 137 different initialization for DA-2.

A prediction is made whether the model would belong in class  $y_1$  or  $y_0$  at epoch 100 using these scores at checkpoint epoch 10. Table 6.7 compares the performance of  $s_{Fv}$  and validation accuracy at the checkpoint to make a binary prediction of the success of initialization at the checkpoint. It can be seen that  $s_{Fv}$  is better at predicting both trained and not trained networks, compared to simply using the validation accuracy for this DA as well.

		Prediction		Total
		$y_1$	$y_0$	
True	$y_1$	29	5	34
	$y_0$	2	101	103
Total		31	106	137

(a) Prediction using  $s_{Fv}$

		Prediction		Total
		$y_1$	$y_0$	
True	$y_1$	26	8	34
	$y_0$	2	101	103
Total		28	109	137

(b) Prediction using  $BV$

Figure 6.7: Breakdown of the prediction of eventual success of all initializations in DA-2 using  $s_{Fv}$  and  $BV$  at epoch 10.

Table 6.2 summarizes the Spearman’s correlation values obtained for all the initializations.

Checkpoint	$s_F$	Training accuracy at checkpoint	$s_{Fv}$	Validation accuracy at checkpoint
10	0.224	0.587	0.766	0.716
20	0.228	0.524	0.767	0.762
30	0.230	0.514	0.818	0.786

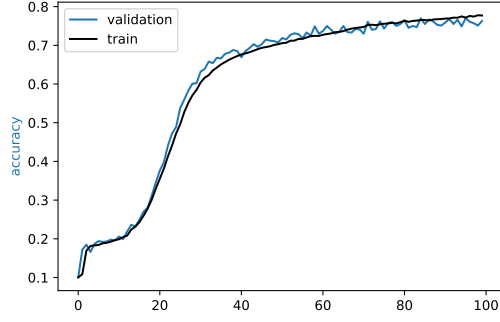
Table 6.2: Correlation of  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at different checkpoint epochs with the final validation accuracy for all initializations in DA-2.

It can be seen that score  $s_F$  performs comparably to the training accuracy at the checkpoint epoch in the absence of labels. Score  $s_{Fv}$  is better than the baseline of validation accuracy at the checkpoint at all checkpoints in this case as well.

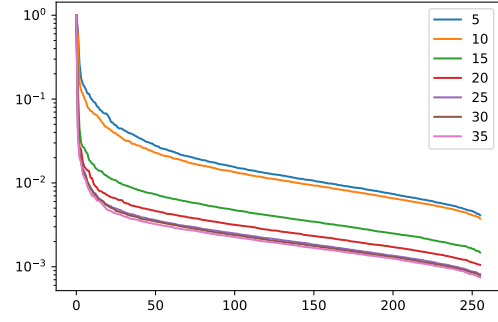
### 6.3 Results on convolutional neural networks

Obtaining singular values of the convolutional layers in CNN requires concatenating the vector obtained from passing the data through all the feature maps in the layer. This method can require large memory and computation. Three variants of SVD calculation are tried for CNN. Although without normalization of score across architectures, a conclusive metric is not yet decided. Refer A.4 in the Appendix for further details.

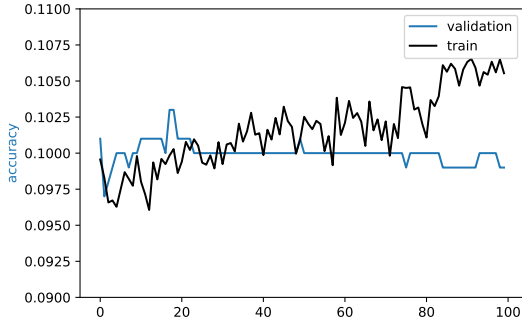
However, the dense layers in CNNs show a compression in SVD values (as observed in MLP). Consider two initializations obtained trained on DA-3 (CIFAR-10, 8-layer CNN) in figure 6.8. It can be seen that the first model trains till a validation accuracy of 76.3% on training till 100 epochs. This is accompanied by the compression of SVD values in the fully connected layer of dimension 256 as training continues. The second initialization does not show this compression and remains untrained till epoch 100.



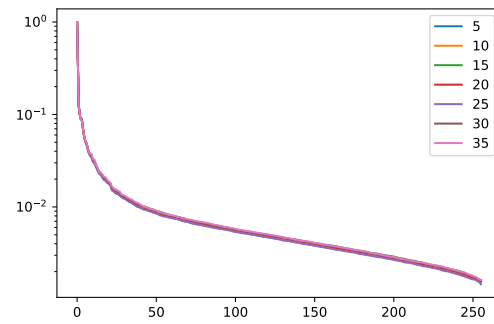
(a) Validation accuracy = 76.3% at epoch 100



(b) Normalized singular values (sorted) - dense layer 1 of size 256 at various epochs



(c) Validation accuracy = 9.89% at epoch 100



(d) Normalized singular values (sorted) - dense layer 1 of size 256 at various epochs

Figure 6.8: Dense layer in well trained CNN show a decay in the singular values upon training on DA-3.

It is observed that the range of  $s_{st}$  and  $s_{ot}$  for CNNs is different from that in MLP. A better normalization technique is required for the scoring to analyze the results obtained from the SVD of fully connected layers in CNNs.

## 6.4 Performance across architectures

The analysis so far has been limited to the same dataset and architecture. However, finding an optimal DNN architecture is also largely experimental. This section expands the utility of early success indicators to test the prediction of both an initialization and architecture for a dataset. For the experiment, Shell-1024 dataset is considered. The various MLP architecture chosen vary from 1 hidden



layer to 11 hidden layers, and the dimension of each hidden layer is chosen arbitrarily in the range of 32 to 512 (in factors of 2). All the models are trained on Adam optimizer with a learning rate of  $10^{-4}$ . A total of 50 such architectures and initializations are trained on this dataset, and their  $s_F$  and  $s_{Fv}$  values are recorded at the checkpoint epoch. All the models are further trained for 100 epochs to find the final validation accuracy. The distribution of final training and validation accuracy for all the models can be observed in figure 6.9

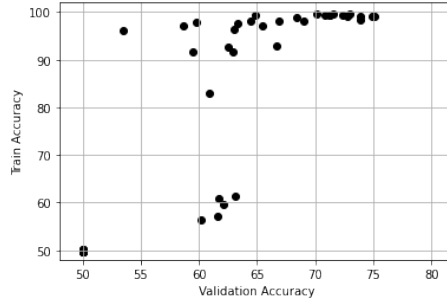


Figure 6.9: Final training and validation performance of 50 different architectures on Shell-1024 dataset.

The correlation values at checkpoint 10, 20, 30 are summarized in table 6.3.  $s_{Fv}$  beats the validation accuracy at checkpoint for all the checkpoints. Figure 6.10 shows the correlation plots for all the four scoring methods.

Checkpoint	$s_F$	Training accuracy at checkpoint	$s_{Fv}$	Validation accuracy at checkpoint
10	0.439	0.678	0.719	0.671
20	0.628	0.761	0.789	0.752
30	0.712	0.781	0.843	0.814

Table 6.3: Correlation of  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at different checkpoint epochs with the final validation accuracy for all 50 architectures trained on Shell-1024 dataset.

**Correlation not a good measure for  $s_F$ :** In the correlation plot, while the  $x$ -axis indicates the final validation accuracy, the  $y$ -axis indicates the chances of success on

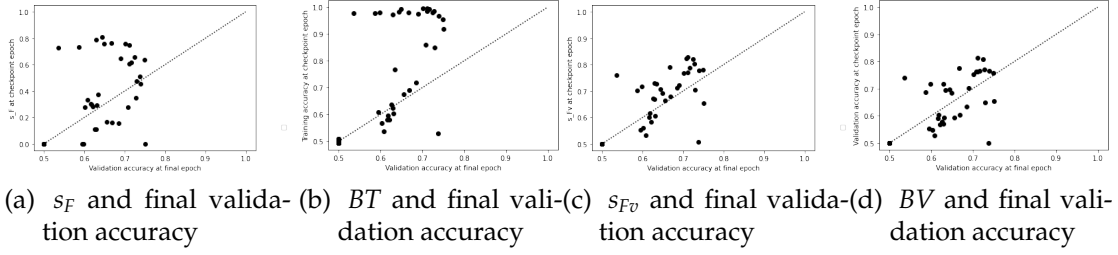


Figure 6.10:  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at checkpoint epoch 30 plotted against the final validation for all 50 architectures.

training further via  $s_F$ . It is observed that as the validation loss stops decreasing, the value of  $s_{st}$  decreases to very small values. As with initialization-5 in figure 6.3 (the blue curve), the value of  $s_F$  becomes a constant. This constant value of  $s_F$  indicates that further training of the model is unlikely to train the model further. This can happen in two scenarios: the model does not train, or the training has saturated. However, the final validation accuracy in both cases can be significantly different. For instance, consider the two initializations below on Shell-1024 dataset and 4-layer MLP. The first initialization obtains a final validation accuracy of 73.2%, and the second remains untrained. In both cases, the validation accuracy does not change significantly since the initialization. These initializations return a similar score  $s_F$  at the checkpoints, indicating that further training might not be beneficial. Therefore, finding the correlation of  $s_F$  with the final validation accuracy is not a good measure to evaluate its performance.

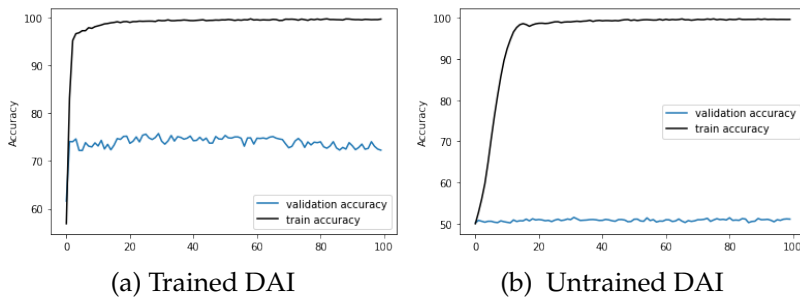


Figure 6.11:  $s_F$  gives same score for two initializations at epoch 30 which do not benefit from further training.

This problem, however, is mitigated when using the validation labels at the checkpoint.

# CHAPTER 7

## Dynamic run selection algorithm

### 7.1 Introduction

In this section, an empirical framework using the early success indicators defined above is proposed to find a "successful" initialization among a set of initializations when trained on the same dataset, architecture, and training procedure. Instead of testing all the initializations in the set to obtain a good initialization, a dynamic run selection algorithm based on the upper confidence bound algorithm is proposed for finding such an initialization. Experiments suggest that such an approach saves significant computational work and training time. For a particular combination of DA (dataset-architecture) and training mechanism, a set of  $n$  initializations are selected to be tested. Let this set of initializations be  $I$ . The proposed algorithm uses early success predictors to evaluate these initializations in a multi-armed bandit setting. The following formalism is used-

1. Each initialization in the set  $I$  is trained for very few epochs  $e_i$  initially before starting the algorithm.
2. Each initialization is an arm/action in the algorithm. Selecting an arm would train the model further, which was initialized on that particular initialization.
3. The algorithm is run for  $T$  rounds. Each round  $t$  comprises of training the selected arm  $a_t$  for a fixed number of epochs ( $e$ ). The total run time of the algorithm can be limited to  $(B - ne_i)/e$  rounds, where  $B$  is the total computational budget allocated for finding a good initialization for a task.  $ne_i$  is the initial computation cost before starting the algorithm.
4. The reward for choosing an arm  $a_t$  in round  $t$  is determined by a scheme  $S$ . Four such schemes are analyzed below.
5. A model trained on initialization  $a_t$  would be trained for a total of  $m_{at}$  epochs before round  $t$ , where  $m_{at} = N_{at} \times e$ , where  $N_{at}$  is the number of times arm  $a_t$  is chosen before round  $t$ .

6. At the end of each round, the validation accuracy of the models trained on  $n$  initializations would be available. This validation accuracy would be calculated for different epochs for different arms, depending on  $N_{at}$ . It is expected that good initializations would be chosen more often and would save the computational time for training the other initializations.

This dynamic run selection algorithm (DRSA) is used to choose the arm to be pulled. Algorithm 1 details the algorithm.  $Q_t(a)$  is the empirical estimate of reward of arm  $a$  before round  $t$ ,  $N_t(a)$  is the number of times arm  $a$  has been selected till round  $t$ ,  $c$  is a hyper-parameter to tune the exploration exploitation trade-off.

Four reward functions are proposed as follows:

1. *Training accuracy (BT)* : Reward of the algorithm is training accuracy at that epoch.
2. *Sf* : Reward of the algorithm is score  $s_F$  based on singular values.
3. *Validation accuracy (BV)* : Reward of the algorithm is validation accuracy at that epoch.
4. *Sfv* : Reward of the algorithm is score based on singular values and validation accuracy,  $s_{Fv}$ .

Although the assumption of IID rewards in the UCB algorithm does not hold true in this experimental set-up, this DRSA has shown to find a good initialization in each set with the goal of using fewer training epochs.

**Baseline:** Baseline algorithm to evaluate the performance of DRSA is defined as follows: Randomly choose an initialization from the set  $I$  and train it till the end (final epoch). The number of epochs required (on average) to find a good initialization for a DA would be the baseline. If  $n$  models are good out of  $N$  models in a set, *Random* baseline is set as  $\frac{N}{n} \times e$ , where  $e$  is the number of epochs for which each model is run totally.

Reward from the *pull()* function depends on the chosen reward function: *BT*, *Sf*, *BV*, *Sfv*. Reward for each function is normalized between  $[0, 1]$ . An initialization is declared good if the final validation accuracy exceeds the threshold, which is decided arbitrarily.

---

Algorithm 1: Dynamic run selection algorithm on DAs

---

```

1: procedure DRSA ▷ Returns a model from the set of models
2:   Train each model (arm) for a fixed number of epochs ( $e_i$ ) and set that reward
   as the initial reward
3:    $trained\_arms = [ ]$ 
4:   for  $t$  in range( $total\_iterations$ ) do
5:      $a$  : list of arms not in  $trained\_arms$ 
6:      $chosen\_arm_t = \operatorname{argmax}_a(Q_t(a) + c \sqrt{\frac{\ln(t)}{N_t(a)}})$ 
7:      $r = \text{pull}(chosen\_arm_t, t);$ 
8:      $N_{t+1}(chosen\_arm_t) \leftarrow N_t(chosen\_arm_t) + 1$ 
9:      $Q_{t+1}(chosen\_arm_t) \leftarrow Q_t(chosen\_arm_t) + \frac{(r - Q_t(chosen\_arm_t))}{N_{t+1}(chosen\_arm_t)}$ 
10:    if  $N_{t+1}(chosen\_arm_t) \geq e$  then
11:      if  $\text{validation\_accuracy}(chosen\_arm_t) > \text{threshold}$  then ▷ break
12:        return ( $chosen\_arm_t$ )
13:      else
14:        add  $chosen\_arm_t$  to  $trained\_arms$ 

```

---

The idea of DRSA is further extended to find a good architecture and initialization for a fixed dataset. Selecting an architecture involves choosing several hyper-parameters. For MLP, the number of hidden layers and number of units in each of the hidden layer has to be determined. Even with the rapidly evolving state-of-the-art performance of deep learning networks in several domains, finding an optimal architecture has remained a challenging task. While there is active on-going research on automatically finding an optimal architecture, these algorithms consume high computational power and time. As stated above, two aspects are crucial for searching optimal architectures, training and evaluating a candidate architecture and a search algorithm to explore different architecture types. Both of these aspects are combined in DRSA, which simultaneously trains and explores architectures from a given set of architectures for a particular dataset and returns a good architecture in less computational time.

The toy example below on DA-1 (Shell-512, 2-layer MLP) illustrates how DRSA on 10 different initializations (arms) when run for 100 total epochs performs. Each arm is run for 4 epochs initially. Then the algorithm chooses the arm with the highest upper confidence bound and updates the parameters.

The final validation accuracy in the plot is obtained by training all the models

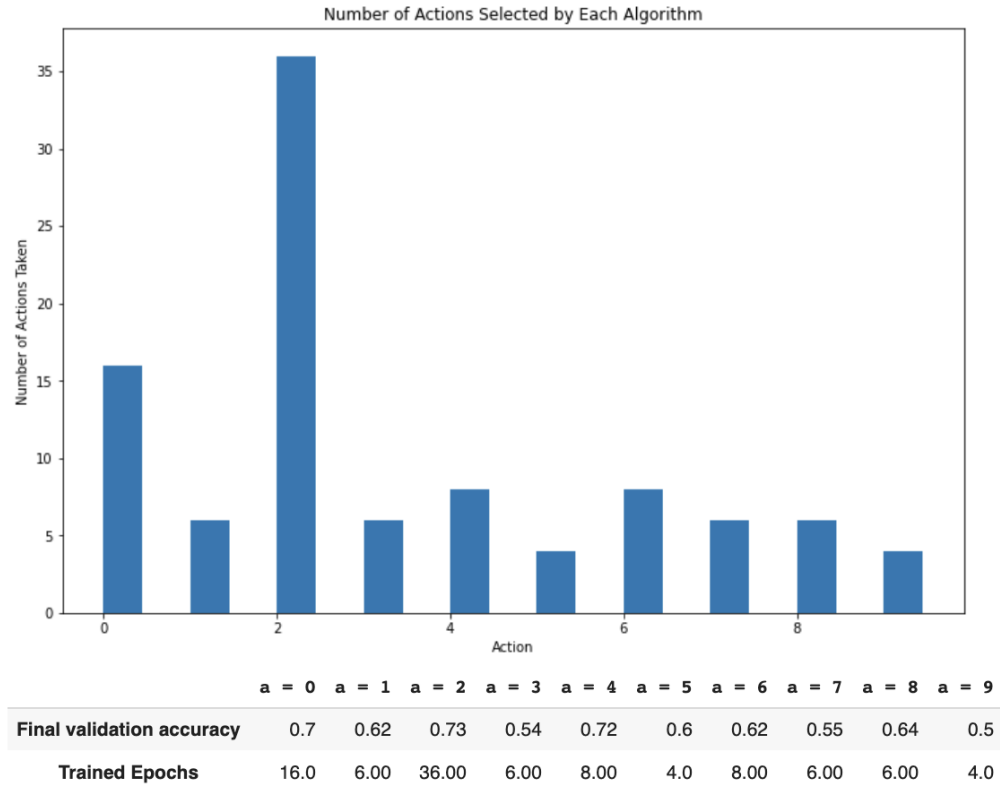


Figure 7.1: Dynamic run selection algorithm for 10 initializations trained on Shell-512 dataset, 2-layer MLP run for 100 epochs. The most selected arm attains the highest validation accuracy on full training.

to a total of 100 epochs individually to evaluate the performance of DRSA. The reward function in this example is  $s_F$ . It can be seen that the arm which eventually gives the highest validation accuracy is the most picked arm. Section 7.2 evaluates the performance of DRSA for larger sets.

## 7.2 Evaluating dynamic run selection algorithm

DRSA is evaluated for two tasks:

1. Finding a good initialization from a set of initializations trained on the same dataset, architecture, and training mechanism.
2. Finding a good architecture and initialization from a set of architectures trained on the same dataset and training mechanism.

To evaluate its performance for finding a good initialization, the models are

trained on two combinations of dataset and architectures (DAs). To further check the reliability of the proposed algorithm, each of these DAs is divided into  $m$  sets. For each set of a DA, the number of training epochs required to find a good initialization is calculated. An initialization is declared "good" if the validation accuracy on full training crosses a certain threshold (this threshold is decided arbitrarily). Each set contains a few such good initializations. Initializations are grouped in sets such that the range of validation accuracies of good models in a set is minimized.

The number of training epochs required to search for a good initialization is used as the metric to evaluate each of the reward functions and the *Random* baseline.

### 7.2.1 Evaluating performance on DAI

In this section, the algorithm is used to find a good initialization from a set of initializations (all trained on same dataset, architecture, and training procedure). Three combination of DA are considered for the evaluation:

- **DA-I**: Shell-1024 dataset, 4-layer MLP architecture
- **DA-II**: Shell-256 dataset, 2-layer MLP architecture
- **DA-III**: CIFAR-10 dataset, 8-layer CNN architecture

For the MLP architectures, all the four reward schemes are analyzed. However, due to the difficulties mentioned in section 6.3, only *BT* and *BV* schemes are used for the CNN DA combination. These initialization for each DA are divided into different sets. For each set, the algorithm is run to find a good initialization.

#### Evaluation on multilayered perceptron

For DA-I and DA-II, a total of 111 and 108 initializations are considered, respectively. Each of this initialization is trained to a maximum of 50 epochs. These are

grouped into 4 and 3 sets, respectively. The statistic for each set is summarized in Table 7.1.

DA	Set	Total Models	Good Models	Validation accuracy range
I	I	27	2	80.2%-81.1%
	II	28	3	80.0%-81.9%
	III	28	3	80.0%-81.5%
	IV	28	3	83.8%-85.3%
II	I	37	3	77.1%-78.0%
	II	36	3	78.3%-79.4%
	III	35	2	80.6%-81.6%

Table 7.1: Range of validation accuracies and number of acceptable models in each set for DA-1 and DA-2.

Each set contains a few good models such that the range of validation accuracy for the good models is minimized. This is done to ensure that finding a good initialization carries equal weightage, and the algorithms can be compared on the basis of the number of epochs required to find the good initializations alone. The distribution of final training and validation accuracies for each of this set can be visualized in figure 7.2 for DA-I and figure 7.3 for DA-II. The threshold for deciding good initialization is set as 80% and 75%, respectively.

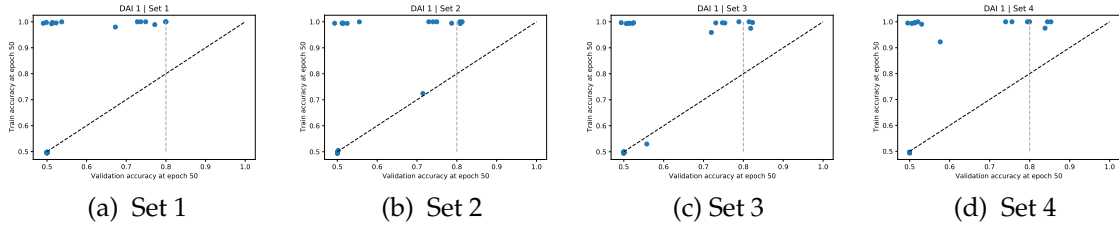


Figure 7.2: Distribution of final validation and train accuracy for each set in DA-1.

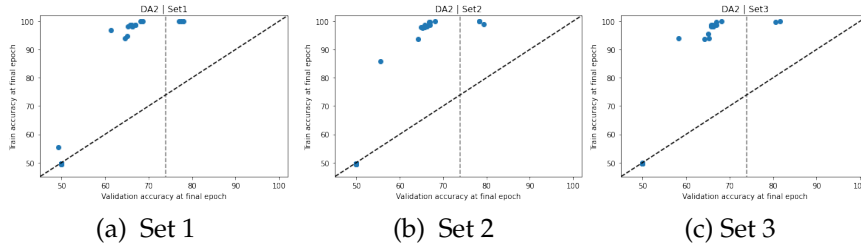


Figure 7.3: Distribution of final validation and train accuracy for each set in DA-2.



Each initialization is run for 2 epochs before starting the algorithm. The algorithm is run for each of the four reward schemes, and the result is compared against the random baseline. For instances in which the ratio of good models to all models is low, the random baseline can be quite high. The algorithm is terminated as soon as a good arm is trained completely (till epoch 50). The results for each set of DA-I and DA-II are summarized in Table 7.2.

DA	Set	Random	<i>Sf</i>	<i>BT</i>	<i>Sfv</i>	<i>BV</i>
I	I	675	254	294	512	560
	II	467	206	296	152	156
	III	467	282	248	596	670
	IV	467	310	342	572	654
II	I	616	198	122	122	122
	II	600	280	120	120	120
	III	875	246	186	194	256

Table 7.2: Number of epochs required by each reward scheme to find a good initialization for each set in DA-1 and DA-2.

It can be seen that the reward scheme *Sfv* beats the random baseline by a huge margin in 6 out of the 8 sets (*Sf* beats the baseline in the remaining two sets). For DA-II, the performance of all the rewards is better than the random baseline. For DA-I, however, the last two sets show inferior performance of *Sfv* and *BV*. Further analysis on these sets reveal the following reasons for this performance:

1. Over-fitting: Several models in set 3 and 4 of DA-1 start over-fitting (decrease in validation accuracy) before epoch 50. Some examples from these sets are shown below:

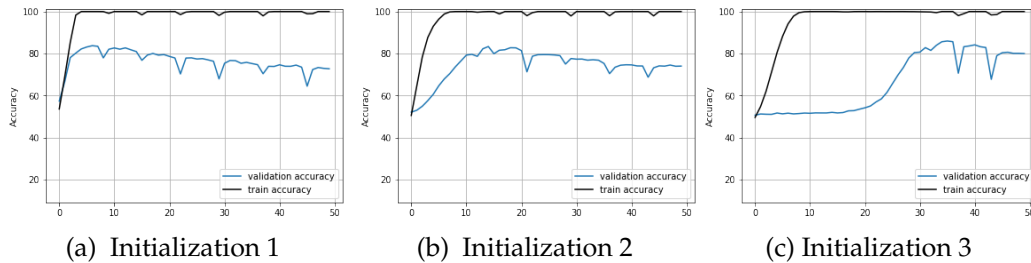


Figure 7.4: Decreasing validation accuracy as training progresses for three initializations from set-III and IV in DA-1.

The validation accuracy of these models is above 80% for a certain range of

epochs and then starts reducing. The sets are built on the final validation accuracy, and these models are not included as good initializations in it. This leads to DRSA exploiting these arms and thus spending computational time on training them as well. One potential method that could overcome this is using a weighted average of rewards instead of a linear running average in step 9 of the algorithm 1. This will ensure that the scores go down rapidly as over-fitting begins.

2. False positives: Some models show decreasing validation loss and unchanging validation accuracy. These models can also be accompanied by high score values, leading to investing computational power in these initializations.

### Evaluation on convolutional neural network

The idea of DRSA for selecting a good initialization can be extended to convolutional neural networks as well. For CNNs, the focus is on random, *BT*, *BV* strategies. For DAI-3, a total of 56 initializations are trained. All models are trained till a maximum of 100 epochs, and 73% is maintained as the threshold for declaring an initialization good with this DA. The distribution of final training and validation accuracy is shown in figure 7.5.

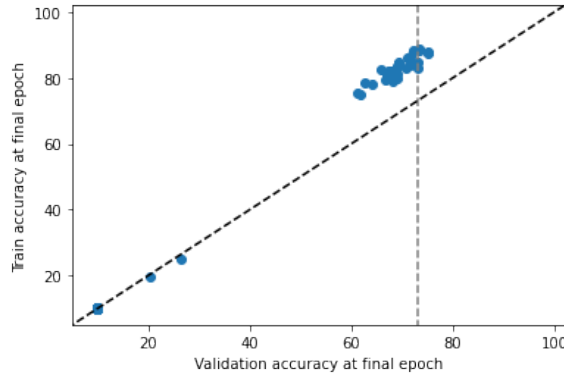


Figure 7.5: Distribution of final validation and train accuracy for each set in DA-3.

These initializations are divided into 2 sets as indicated in Table 7.3. Each set has 28 initializations.

Using *Random*, *BT*, *BV* scoring for heuristic UCB algorithm for CNN on CIFAR-10 dataset gives the results presented in Table 7.4. For both the sets, *BT* and *BV* scoring beats the *Random* baseline by a huge margin.

Set	Total Models	Good Models	Range of validation accuracy
I	28	3	73.1%-73.4%
II	28	2	75.1%-75.1%

Table 7.3: Range of validation accuracies and number of acceptable models in each set for DA-3.

Set	<i>Random</i>	<i>BT</i>	<i>BV</i>
I	933	140	238
II	1400	388	292

Table 7.4: Number of epochs required by *Random*, *BT*, *BV* reward scheme to find a good initialization for each set in DA-3.

## 7.2.2 Evaluating performance on DA

In this section, the result of running DRSA on choosing a good architecture and initialization from a set that is trained on the same dataset and training mechanism is presented. The dataset used for this experiment is *Shell-1024* dimensional. A total of 50 models have been trained to 50 epochs to find the final validation accuracy. The architecture has been chosen arbitrarily by varying the number of hidden layers and the dimension of each hidden layer in an MLP. The number of hidden layers is varied from 1 to 11. These models are then divided into different sets to check the reliability of the UCB algorithm. The summary of these models is as shown in Table7.6.

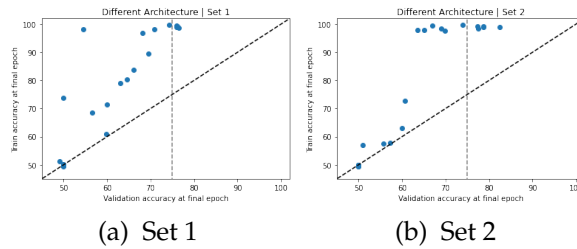


Figure 7.6: Distribution of final validation and train accuracy for each set in *Shell-1024* dataset.

The initialization for these architectures is sampled from *Xavier normal* distribution. Models which attain a validation accuracy higher than 75% are classified as good. The characteristic for each set is summarized in table 7.5

Set	Total Models	Good Models	Range of validation accuracy
I	25	5	75.1%-76.4%
II	25	5	77.3%-82.5%

Table 7.5: Sets comprising of different architectures trained on Shell-1024.

Both of these sets are run on all the reward schemes, and the result is compared against the random baseline. Although the computational cost for training 1 epoch of different architectures is different, the random baseline is calculated, assuming the computation time to be comparable. The results are summarized in table 7.6.

Set	<i>Random</i>	<i>Sf</i>	<i>BT</i>	<i>Sfv</i>	<i>BV</i>
I	250	208	242	110	180
II	250	360	388	196	196

Table 7.6: Number of epochs required by each reward scheme to find a good architecture for each set in Shell-1024 dataset.

It is evident that DRSA with reward scheme  $s_{Fv}$  and  $BV$  performs better than random trials.

Varying the hyper-parameter for exploration-exploitation balance,  $c$  affects the number of training epochs required to find a good initialization. A very small value of  $c$  suggests that the algorithm heavily prefers the empirical estimation of rewards as the confidence bound is shrunk. This often results in yielding initializations that may not give a good performance but performs better than random prediction. A very high value of  $c$  (close to 1) suggests that the algorithm has wide confidence bound and focuses more on exploring. This can result in exploring arms with low empirical rewards finally. Therefore, choosing an optimal  $c$  is necessary. For all the experiments in this section,  $c$  is maintained as 0.1.

**Visualization baseline:** Figure 7.7 shows the training and validation accuracy at each epoch for 24 architectures and initializations used in Set-II in table 7.5. It demonstrates that choosing a model to train at each round  $t$  of the dynamic run selection algorithm is non-trivial. For instance, for several models, the validation

accuracy at epoch 20 is not indicative of the final validation accuracy. The good models indicated in the table are marked in green in the table.

**Problem with DRSA:** The results of the dynamic run selection algorithm are susceptible to the choice of hyper-parameter  $c$  for  $Sf$  and  $Sfv$  schemes. Since the scores are not normalized for different datasets, the same choice of  $c$  can result in high exploration in some DAIs (leading to wastage of computational cost for models that yield low scores and would potentially not train further), and high exploitation in some DAIs (leading to complete training of DAIs which might not give best results in the final epoch). The algorithm is less sensitive to the choice of hyperparameter  $c$  for  $BT$  and  $BV$  scoring, however.

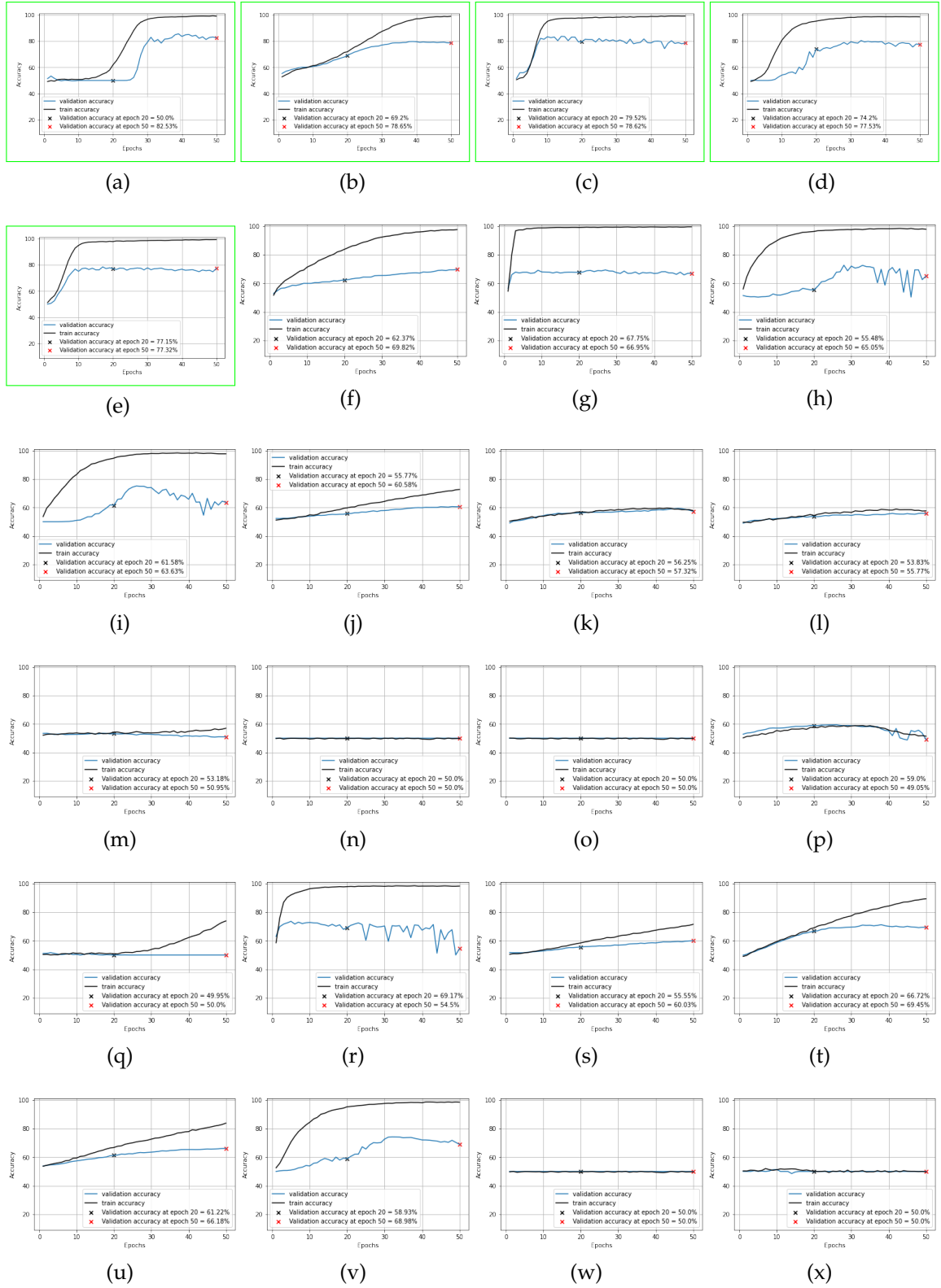


Figure 7.7: Visualizing the training and validation accuracy of architectures in Set-I trained on Shell-1024 dataset. The good architectures are highlighted in green.

## CHAPTER 8

### Future directions and Conclusion

#### 8.1 Future Work

Early success indicator and dynamic run selection algorithm have shown positive results on the task of predicting the success of gradient descent for a particular dataset-architecture-initialization combination. This work could be further extended in the following directions.

##### 8.1.1 Subspace for different classes

The calculation of  $s_{st}$  and  $s_{ot}$  in section 5 is done independent of the labels of the test dataset. It can be extended to automatic dimensionality reduction for individual classes. For a two-class problem with classes  $y_0$  and  $y_1$ , let  $A_0$  and  $A_1$  be the matrix obtained by passing validation dataset of each class through the hidden layer  $l$ . Let  $A_0$  and  $A_1$  be of dimension  $n \times d_l$ , where  $n$  is the number of data points and  $d_l$  is the dimension of output from the hidden layer (assuming  $n \geq d_l$ ).

The direction  $v_1$  along which a matrix  $A$  has the most amplifying power (or impact) is calculated as:

$$v_1 = \operatorname{argmax}_{v \in R^{d_l}} \|Av\|_2, \text{ subject to } \|v\|_2 = 1$$

Similarly, the next most amplifying direction is found by

$$v_2 = \operatorname{argmax}_{v \in R^{d_l}} \|Av\|_2, \text{ subject to } \|v\|_2 = 1, \langle v_1, v \rangle = 0$$

Proceeding along these lines gives us an orthonormal bases  $\{v_1, \dots, v_{d_l}\}$  of  $R^{d_l}$ . The

singular values quantify the amplifying power of each such direction  $\sigma_i = \|Av_i\|_2$ . An orthogonal matrix  $U$  is chosen such that  $Av_i = \sigma_i u_i$  for  $i = 1, \dots, d_l$ .

Intuitively, it is expected that the amplifying power of the matrix  $A_0$  and  $A_1$  would be in different directions. If it were in the exact same direction, distinction between the two classes could not be made out. Therefore, a rudimentary attempt to quantify this intuition is stated as a score  $s$ :

$$s = \frac{\sum_{i=1}^{d_l} \sigma_{i0} \sigma_{i1} v_{i0}^T v_{i1}}{d_l}$$

where  $\sigma_{i0}$  is the  $i^{th}$  singular value of matrix  $A_0$ .

A high score  $s$  can be indicative of separate directions of different classes for the automatic dimensionality reduction.

### 8.1.2 Predicting success of a DAI as a classification problem

Predicting the success can be formulated as a three-class problem. Neural networks at any epoch can have either:

- High training accuracy, high validation accuracy
- High training accuracy, low validation accuracy
- Low training accuracy, low validation accuracy

Please note, the terms "high" and "low" here can be defined quantitatively for a given task and dataset based on simple thresholding. Roughly, the case of low training accuracy and low validation accuracy is when the neural network is close to a random predictor for training and testing data. High training accuracy and low validation accuracy are models with a huge generalization gap. The case of low training accuracy and high validation accuracy is not often observed in real-world problems. (Using dropout can give slightly higher validation accuracy than training accuracy, but the difference is usually not significant).



Let the three classes be called  $y_1, y_2, y_3$  respectively and let a DAI be in class  $s_C$  at checkpoint epoch  $e_C$  and in class  $s_L$  at the final epoch  $e_L$  (where  $s_C, s_L \in \{y_1, y_2, y_3\}$ ). For evaluating the performance of ESI on the classification problem, the classes are decided as follows: high accuracy (validation or train) is defined to be attained if it crosses a threshold of 70%. Table 8.1 shows the possible combination of occurrence at  $e_C$  and  $e_L$  for the Shell-1024 dataset.

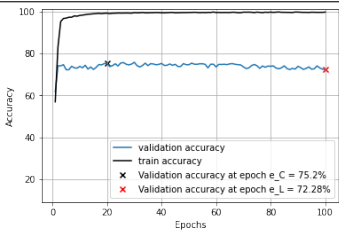
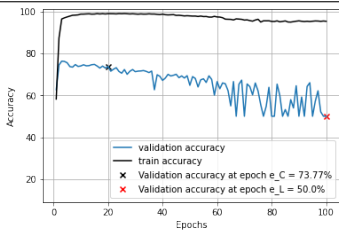
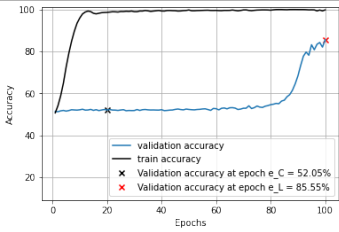
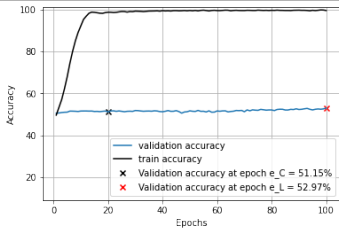
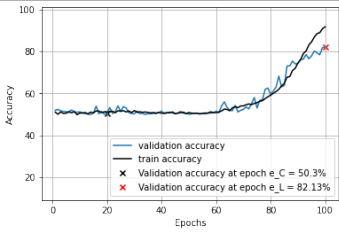
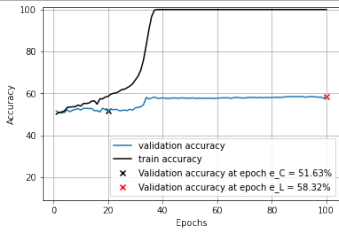
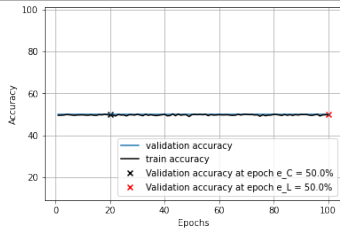
$e_L \backslash e_C$	$y_1$	$y_2$	$y_3$
$y_1$	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 75.2\%</math> x Validation accuracy at epoch <math>e_L = 72.28\%</math></p>	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 73.77\%</math> x Validation accuracy at epoch <math>e_L = 50.0\%</math></p>	
$y_2$	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 52.05\%</math> x Validation accuracy at epoch <math>e_L = 85.55\%</math></p>	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 51.15\%</math> x Validation accuracy at epoch <math>e_L = 52.97\%</math></p>	
$y_3$	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 50.3\%</math> x Validation accuracy at epoch <math>e_L = 82.13\%</math></p>	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 51.63\%</math> x Validation accuracy at epoch <math>e_L = 58.32\%</math></p>	 <p>validation accuracy train accuracy x Validation accuracy at epoch <math>e_C = 50.0\%</math> x Validation accuracy at epoch <math>e_L = 50.0\%</math></p>

Table 8.1: Different possible combination of  $y_1, y_2, y_3$  at  $e_C$  and  $e_L$  for the Shell-1024 dataset.

The use of early success indicators can be extended to predict in which class a DAI would fall after the final epoch. In the absence of validation labels, this prediction could be made at the checkpoint epoch as well. For the Shell-1024 dataset, a total of 144 models are considered, trained on a 4-layered MLP. The true status of this dataset is shown in the table below. With the availability of validation labels at  $e_C$  (In this experiment  $e_C$  is considered at epoch 20), a prediction is made about  $s_L$  using only the score  $s_{st}$ . The results for it are summarized in the table below.

True Status				Prediction using $s_{st}$			
$e_L \backslash e_C$	$y_1$	$y_2$	$y_3$	$e_L \backslash e_C$	$y_1$	$y_2$	$y_3$
$y_1$	8	15	0	$y_1$	10	13	0
$y_2$	25	20	0	$y_2$	17	28	0
$y_3$	5	7	64	$y_3$	4	0	72

The total number of correct predictions is 102 out of 144 (70.83%). All predictions for the  $(y_3, y_2)$  combinations are wrong as  $s_{st}$  can not predict models with low training and validation accuracy at the checkpoint as models with a high generalization gap (it can only predict if the model would be trained successfully or not). This is a preliminary experiment using just  $s_{st}$ . An extensive study can be conducted for each of the seven  $s_C, s_L$  combinations.

## 8.2 Conclusion

In this work, a mechanism has been realized to predict the success of gradient descent for a particular dataset-architecture-initialization combination. Two early success indicators are proposed, analyzed, and tested on several tasks, datasets, architectures, and initializations to facilitate early give-up, i.e., stopping the training of models early, which are predicted to not generalize well upon further training thereby saving computational time and power.

The early success indicators investigate noise stability and automatic dimensionality reduction for the outputs from the hidden layer, indicating that for a well-trained network, signal carries in specific directions (section 4). This phenomenon is used to build quantitative measures to compute scores for making a classification regarding the success of gradient descent on a particular DAI. Evaluation on various DAI combinations demonstrates the success of these early success indicators and the advantage of using them instead of simply relying on the training and validation accuracy at the checkpoint epoch to make a judgment about the final success of the DAI (section 6).

These indicators are used to put forward a dynamic run selection algorithm to find a good initialization weight from a set of such weights in section 7. The algorithm runs for a finite number of rounds and chooses an initialization to train in each round. This combines the training of each initialization and testing of several initializations in one algorithm. The results show that the proposed algorithm is better than the random baseline on a majority of the DAIs. The analysis has been extended to choosing architectures and to CNNs.

In conclusion, using early success indicators can facilitate early give-up by predicting the success of a DAI with prescience at an early stage in training, which can help make an informed decision on the benefits of further training the model.

## REFERENCES

- [1] **Arora, S., R. Ge, B. Neyshabur, and Y. Zhang** (2018). Stronger generalization bounds for deep nets via a compression approach. URL <http://arxiv.org/abs/1802.05296>.
- [2] **Frankle, J. and M. Carbin** (2018). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. URL <http://arxiv.org/abs/1803.03635>.
- [3] **Glorot, X. and Y. Bengio** (2010). Understanding the difficulty of training deep feed-forward neural networks. Technical report. URL <http://www.iro.umontreal>.
- [4] **Goodfellow, I., Y. Bengio, and A. Courville** (2016). Deep Learning. Technical report.
- [5] **He, K., X. Zhang, S. Ren, and J. Sun** (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. URL <http://arxiv.org/abs/1502.01852>.
- [6] **Ji, Z. and M. Telgarsky** (2018). Gradient descent aligns the layers of deep linear networks. URL <http://arxiv.org/abs/1810.02032>.
- [7] **Kalyanakrishnan, S., A. Tewari, P. Auer, and P. Stone** (2010). PAC Subset Selection in Stochastic Multi-armed Bandits. Technical report.
- [8] **Kozlov, A., I. Lazarevich, V. Shamporov, N. Lyalyushkin, and Y. Gorbachev** (2020). Neural Network Compression Framework for fast model inference. URL <http://arxiv.org/abs/2002.08679>.
- [9] **Lai Andherbertrobbins, T. L.** (1985). Asymptotically Efficient Adaptive Allocation Rules\*. Technical report.
- [10] **Li, D., X. Chen, M. Becchi, and Z. Zong**, Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs. In *Proceedings - 2016 IEEE International Conferences on Big Data and Cloud Computing, BDCloud 2016, Social Computing and Networking, SocialCom 2016 and Sustainable Computing and Communications, SustainCom 2016*. Institute of Electrical and Electronics Engineers Inc., 2016. ISBN 9781509039364.
- [11] **Liu, H., K. Simonyan, and Y. Yang** (2018). DARTS: Differentiable Architecture Search. URL <http://arxiv.org/abs/1806.09055>.
- [12] **Saxe, A. M., J. L. McClelland, and S. Ganguli** (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. URL <http://arxiv.org/abs/1312.6120>.
- [13] **Simonyan, K. and A. Zisserman** (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. URL <http://arxiv.org/abs/1409.1556>.
- [14] **Slivkins, A.** (2019). Introduction to Multi-Armed Bandits. URL <http://arxiv.org/abs/1904.07272>.

- [15] **Strubell, E., A. Ganesh, and A. McCallum** (2019). Energy and Policy Considerations for Deep Learning in NLP. URL <http://arxiv.org/abs/1906.02243>.
- [16] **Tang, H., Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han** (2020). Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. URL <http://arxiv.org/abs/2007.16100>.
- [17] **Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals** (2016). Understanding deep learning requires rethinking generalization. URL <http://arxiv.org/abs/1611.03530>.
- [18] **Zoph, B. and Q. V. Le** (2016). Neural Architecture Search with Reinforcement Learning. URL <http://arxiv.org/abs/1611.01578>.

# APPENDIX A

## Additional Experiments

### A.1 Noise stability

This section tests the result of noise stability for other datasets and architectures as well. Figure A.1 demonstrates compressibility of MLP for a different dataset: MNIST dataset, 4-layer MLP architecture, initialization from Normal Xavier scheme. It is trained using RMSprop optimizer with learning rate  $10^{-3}$  till 100 epochs. Significant compression can be seen in this setup as well when the correctly labeled data is passed. This stability reduces when the data is shuffled.

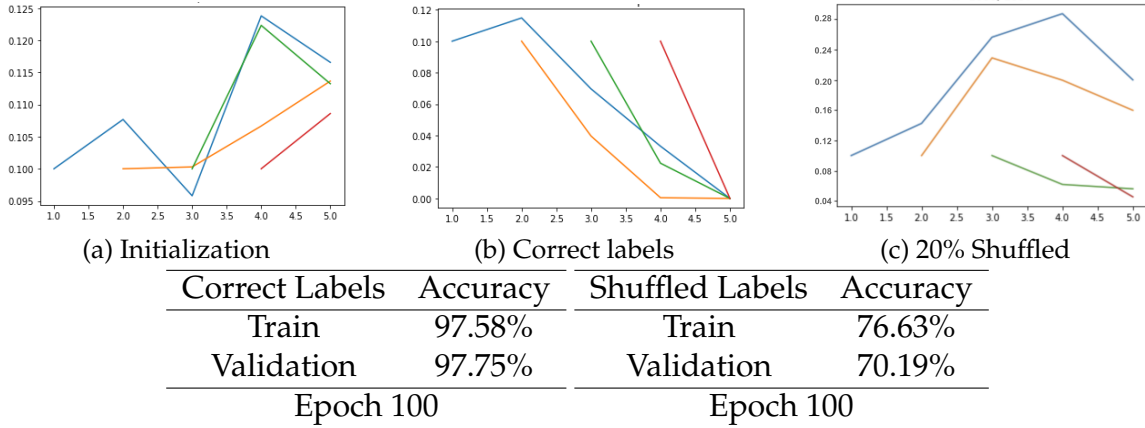


Figure A.1: Noise stability of MNIST dataset, 4-layer MLP architecture, initialization from Normal Xavier. Noise stability and validation accuracy reduce as the labels are shuffled.

Compression of noise on MNIST dataset is also tested with 16-layer CNN architecture and initialization from Normal Xavier scheme as shown in Figure A.2. This DAI also demonstrates significant noise stability and very high validation accuracy.

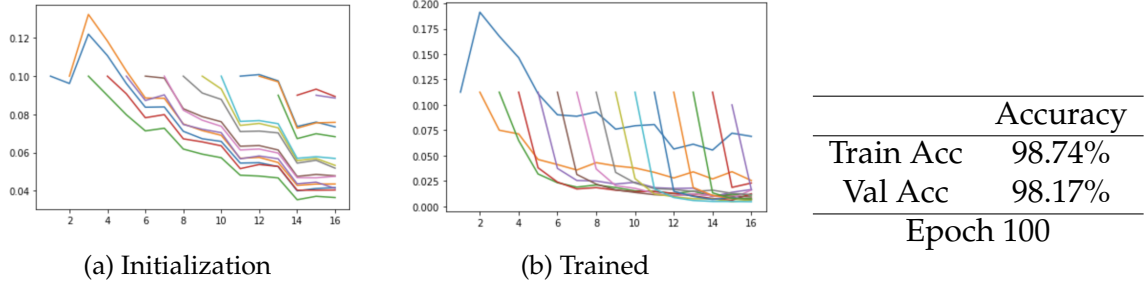


Figure A.2: Noise stability of MNIST dataset, 16-layer CNN architecture, initialization from Normal Xavier scheme.

### A.1.1 Noise stability for smaller dimension dataset

Although, results for MNIST dataset, Shell-1024 dataset, CIFAR-10 dataset provide support to the hypothesis of a positive correlation between learning and noise stability, the results for low dimensional datasets do not always show noise stability. Figure A.3 demonstrate how compression profile changes with increasing dimension.

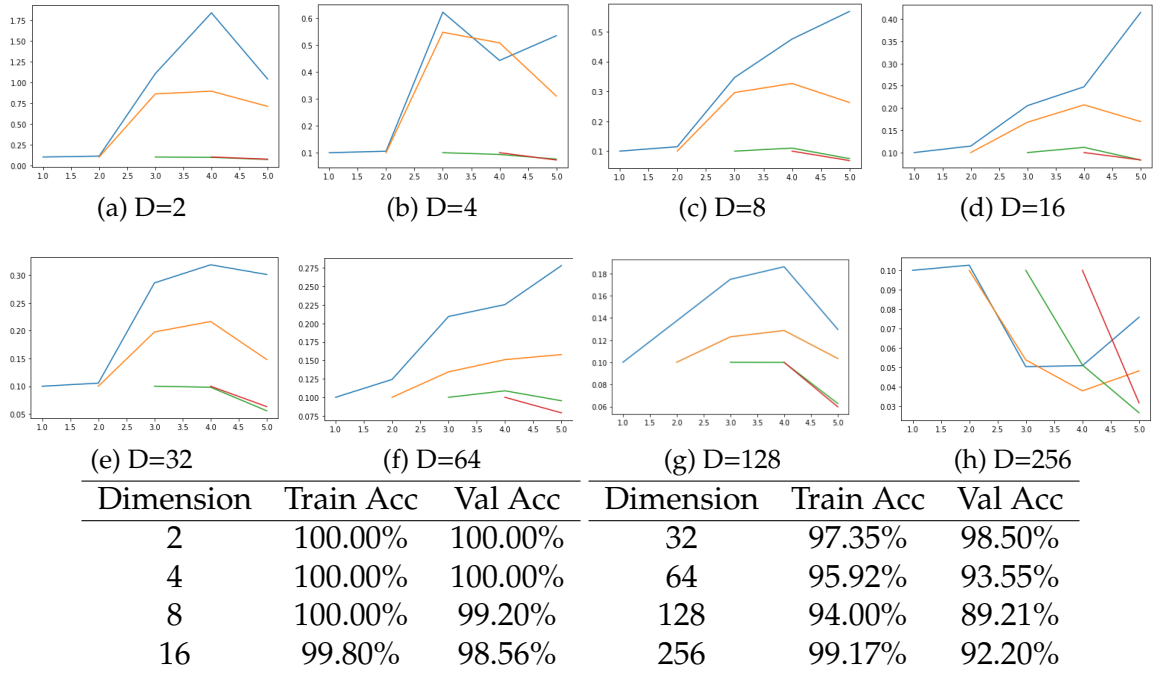
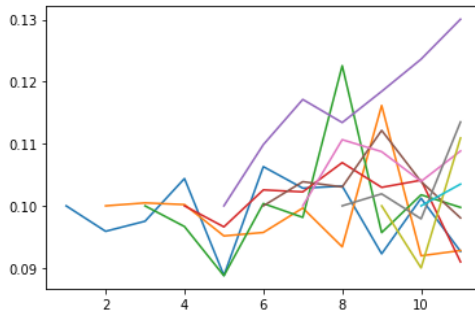


Figure A.3: Noise stability of smaller dimensional Shell dataset, 4-layer MLP architecture, initialization from Normal Xavier scheme. Noise stability increases with increasing dimension of input vectors.

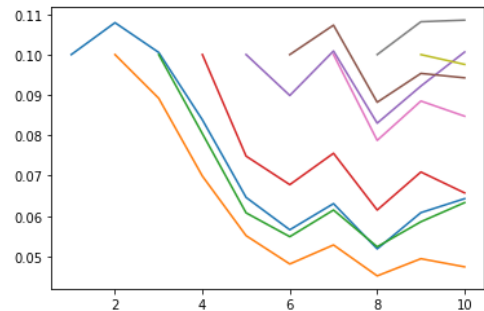
Shell dataset with different dimension ranging from 2 to 256 are used. The stopping criteria for all the above mentioned dataset is early stopping with patience 10. 4-layer MLP architecture and initialization from Normal Xavier scheme is used in all the following networks. Therefore, this shows that generalization does not always imply compression.

### A.1.2 Compression at initialization

It has been observed that MLP and CNN show very different compression profiles at initialization. There is some observable compression in CNNs, unlike MLPs, at initialization. Consider an initialization in Figure A.4 (a) of a 10 layer MLP with 8 hidden layers of 256 units and 2 hidden layers of 128 dimension. Initialization from Normal Xavier scheme is used. It can be seen that there is no significant compression in MLPs at initialization. On the other hand, consider Figure A.4(b) that shows an instance of initialization in CNNs. 10-layer CNN architecture and initialization from Normal Xavier scheme is used. There is significant compression in CNN even without any training.



(a) Initialization of 10-layer MLP



(b) Initialization of 10-layer CNN

Figure A.4: Initialization of MLP and CNN. CNN initialization shows noise stability at initialization.

To test whether ReLU can contribute towards this compression at the beginning, the value of noise transmission is extracted just before the activation function. Consider MNIST dataset, 10-layer CNN architecture, initialization from Normal Xavier scheme. Table A.1 compares the the values for noise stability before and



after the ReLU activation function of untrained (initialization) and trained network. The network is trained for 10 epochs, achieving a validation accuracy of 97.24%. It can be seen that in many layers, the value of noise propagation reduces after passing through ReLU activation function and this could be a possible factor for compression at initialization.

Layer	Before ReLU (init)	After ReLU (init)	Before ReLU (trained)	After ReLU (trained)
1	0.1024	0.1079	0.0928	0.0763
2	0.0861	0.0891	0.0632	0.0631
3	0.0893	0.0804	0.0704	0.0669
4	0.0822	0.0748	0.0444	0.0349
5	0.0985	0.0898	0.0533	0.0615
6	0.1031	0.1072	0.0558	0.0668
7	0.0914	0.0787	0.0506	0.0688
8	0.1055	0.1081	0.0674	0.0643
9	0.0973	0.097	0.0762	0.0655

Table A.1: Effect of activation function on noise stability of MNIST dataset, 16-layer CNN architecture, initialization from Normal Xavier scheme

## A.2 Singular Value Decomposition

Singular value distribution is tested for additional classification and regression tasks in this section.

## A.2.1 Regression

In Section 4.2.1, it was showed that compression of SVD values holds for trained regression models as well. The following experiment is done on Cosine of Sum dataset and Sum of Cosines dataset. The SVD profile is compared for different  $f$ .

### SVD outputs for Cosine of Sum dataset

The output is uniformly distributed in  $[-4, 4]$ . The architecture is the same as used in 4.2.1 (4 hidden layers(256,128,128,64). No batch normalization, dropout layer considered). Consider the following cases:

1.  $4f = 40$  :

- **Dataset** : Sampled from  $y = \cos(\sum_{i=1}^{10} x_i) + \sin(\sum_{i=11}^{20} x_i) + \cos(\sum_{i=21}^{30} x_i) + \sin(\sum_{i=31}^{40} x_i)$ .
- **Training** : Trained till 2000 epochs using SGD optimizer, and  $10^{-3}$  learning rate.

Figure A.5 shows the SVD output for each hidden layer and the noise stability (the plot for layer-3 and layer-4 are similar to layer-2). Profile of the first hidden layer is very different from others. Proper decay of SVD values is observed in the first layer. There is some irregular behaviour for the first few singular values in all the other layer. After the 10th singular value, expected behaviour is observed.

2.  $4f = 20$  :

- **Dataset** : Sampled from  $y = \cos(\sum_{i=1}^5 x_i) + \sin(\sum_{i=6}^{10} x_i) + \cos(\sum_{i=11}^{15} x_i) + \sin(\sum_{i=16}^{20} x_i)$
- Since less dimensions of  $x$  are active in this function, a steeper SVD decay was expected. But the results were very similar to the case above.

### SVD outputs for Sum of Cosines dataset

Since Sum of Cosines dataset is a summation of several cosine terms, it gets difficult to obtain a uniform distribution of  $y$  as  $f$  increases. It also becomes more difficult to train the model with increasing value of  $f$ . The architecture is the same as used in the last regression experiment. Consider the following case:

1.  $2f = 4$  :

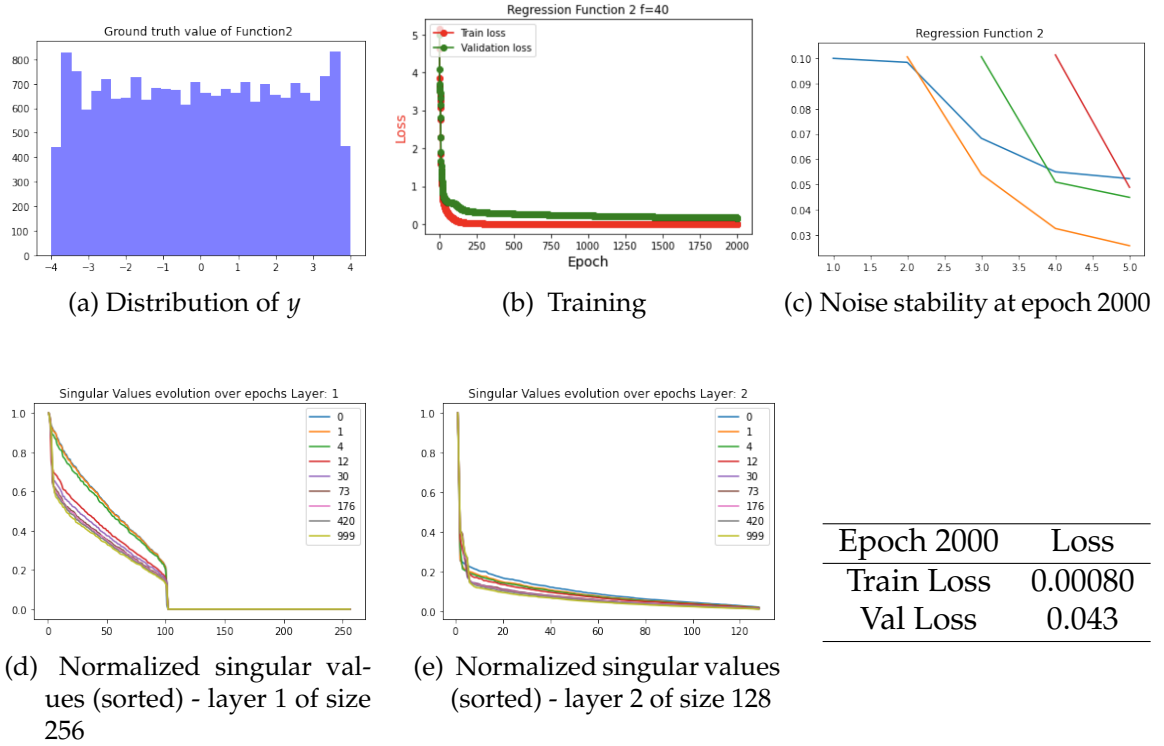


Figure A.5: SVD Output at various epochs for Cosine of Sum dataset at  $4f = 40$ .

- **Dataset :** Sampled from  $y = \frac{\sum_{i=1}^2 \cos(x_i) + \sum_{i=3}^4 \sin(x_i)}{4}$ .
- **Training :** Trained till 2000 epochs using SGD optimizer and  $10^{-3}$  learning rate.

Although there is some decay in the SVD values, the shift is not significant with subsequent epochs. The noise stability plot displays stability to noise.

### A.3 Evaluating early success indicators

Analysis in section 6 was done on synthetic Shell dataset for MLPs. An experiment for MNIST dataset has been shown below.

#### MNIST dataset, 2-layer MLP architecture

Figure A.7 shows the distribution of final training accuracies and validation accuracies. Initializations which achieve validation accuracy above 20% at the final

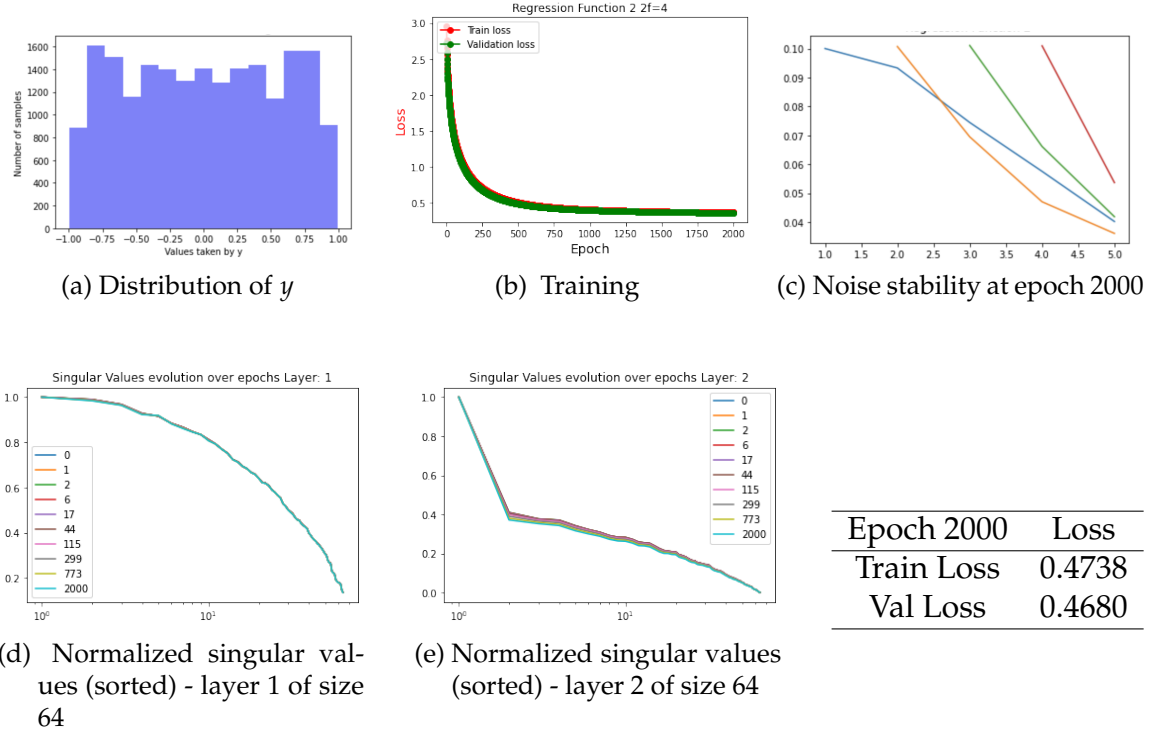


Figure A.6: SVD Output at various epochs (in log scale) for Sum of Cosines dataset at  $2f = 4$ .

epoch are considered to be in class  $y_1$ , and all the other initializations in class  $y_0$ . The points right to the red line indicates models which qualify in class  $y_1$ .

Figure A.8 shows the prediction using  $s_F$  and  $s_{Fv}$ . Table A.2 summarizes the Spearman's correlation values obtained for all the initializations. Since the generalization gap for most of the initializations is very low, training accuracy at checkpoint has a higher correlation with final validation accuracy.

Checkpoint	$s_F$	Training accuracy at checkpoint	$s_{Fv}$	Validation accuracy at checkpoint
30	0.7822	0.0287	0.7677	0.7936
50	0.7822	0.0816	0.7767	0.8027

Table A.2: Correlation of  $s_F$ , training accuracy,  $s_{Fv}$ , validation accuracy at different checkpoint epochs for all initializations.

For this DA, the validation accuracy is very close to the training accuracy for

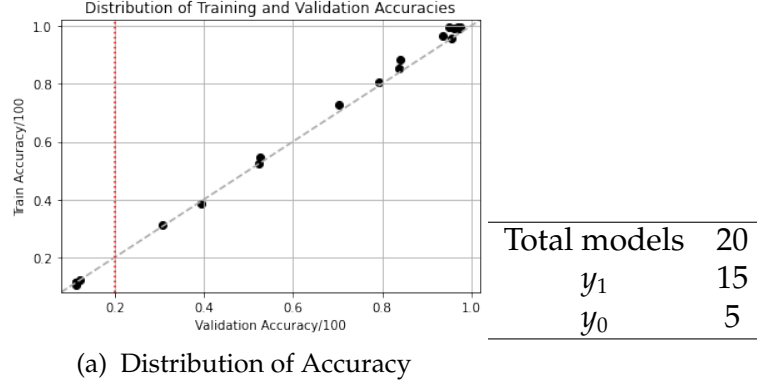


Figure A.7: Final training and validation performance of 20 different initialization for MNIST dataset, 2-layer MLP.

		Prediction		
True		$y_1$	$y_0$	Total
	$y_1$	15	0	15
	$y_0$	0	5	5
Total		15	5	20

(a) Prediction using  $s_{Fv}$

		Prediction		
True		$y_1$	$y_0$	Total
	$y_1$	15	0	15
	$y_0$	1	4	5
Total		16	4	20

(b) Prediction using  $BV$

Figure A.8: Breakdown of the prediction of eventual success of all initializations using  $s_{Fv}$  and  $BV$  at epoch 30.

most of the models. Therefore, the correlation of training accuracy at  $e_c$  and validation accuracy at final epoch.

## A.4 Convolutional Neural Networks

The activation maps of CNNs can be flattened to a 1-D matrix, and the SVD output can be extracted from there. The dimension of the matrix would be  $(n \times (d_x \times d_y \times n_c))$ , where  $n$  is the number of data points,  $d_x$  and  $d_y$  are the dimension of feature map and  $n_c$  is the number of feature maps at that hidden layer. However, this matrix could be huge in size for most of the architectures. Three ways of obtaining SVD output from the CNNs are discussed below. For all the experiments, CIFAR-10 dataset and 10-layer CNN architecture is used.

1. Max/Avg Sampling: Sampling is done on the outputs of hidden layer and before flattening it. Both max sampling and average sampling show similar result. Below is the result from average sampling. Figure A.9 demonstrates the training and SVD output of layer-6.

- Architecture:  $C(16, 16, 32, 32, 64, 64, 128)$ ,  $D(256, 128, 10)$ , where  $C$  is Convolutional Layer and  $D$  is dense layer.
- After sampling, lengths of vectors is as follows (1936, 1936, 1152, 1152, 576, 576, 1152, 256, 128, 10).

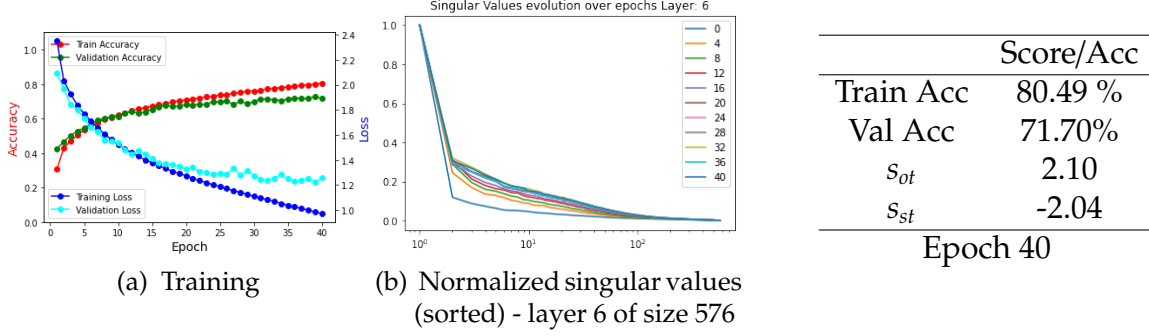


Figure A.9: Training and SVD output at various epochs: CIFAR-10 dataset, 10-layer CNN architecture, average sampling.

2. Individual Activation Map: All the feature maps in a hidden layer are taken individually for calculating the scores. The final score for a hidden layer is averaged by the number of feature maps. The size of SVD matrix in this case becomes  $(n \times (d_x \times d_y))$ . In Figure A.10, one randomly selected SVD output from hidden layer-6 is shown.

- Architecture :  $C(16, 16, 32, 32, 64, 64, 128)$ ,  $D(256, 128, 10)$ , where  $C$  is Convolutional Layer and  $D$  is dense layer.
- After sampling, lengths of vectors is as follows: (1024, 1024, 256, 256, 64, 64, 64, 256, 128, 10).

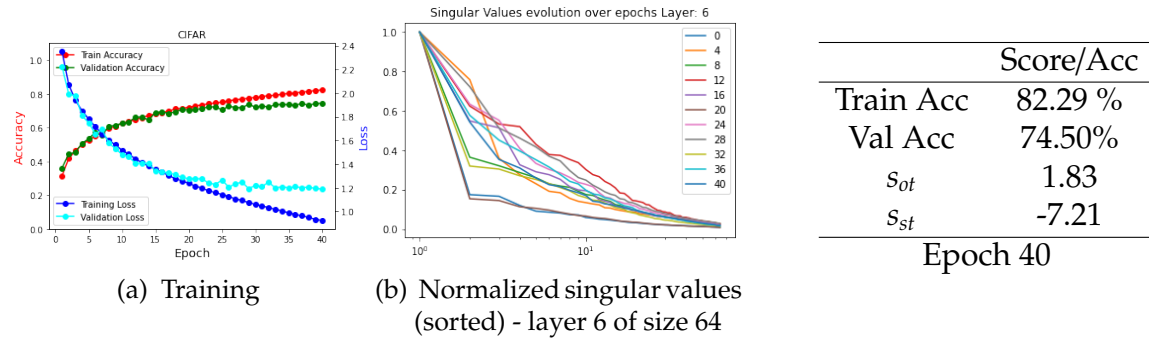


Figure A.10: Training and SVD output at various epochs: CIFAR-10, 10-layer CNN, individual feature map.

3. Entire Hidden Unit: Flatten the entire hidden unit. This method is only feasible for small architectures. Figure A.11 shows the training and the SVD output.

- Architecture :  $C(4, 4, 16, 16, 32, 32, 32), D(256, 128, 10)$ , where  $C$  is Convolutional Layer and  $D$  is dense layer.
- After sampling, lengths of vectors is (4096, 4096, 1600, 1600, 800, 800, 800, 256, 128, 10).

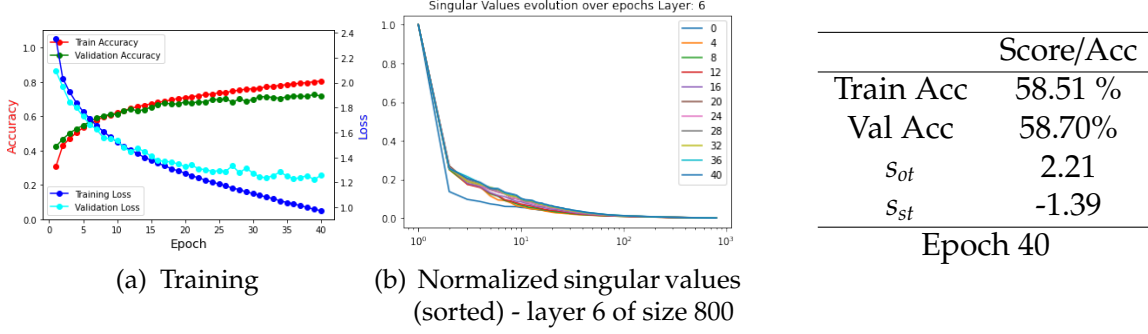


Figure A.11: Training and SVD output at various epochs: CIFAR-10, 10-layer CNN, entire hidden layer.

The SVD profile for CNNs do not indicate a clean decay of the values with subsequent epochs. Other layers of CNN also show similar profile. The comparison for the three methods is not straightforward as the scoring has not been normalized across architectures yet. An extensive experimentation for each method and normalized scoring is further required to predict the success of a DAI based on the two tools.

## A.5 LUCB

LUCB algorithm [7] selects a subset of size  $m$  of those arms with the highest expected rewards from among the arms of a stochastic  $n$ -armed bandit.

### A.5.1 Algorithm

Each sample (or pull) of an arm  $a$  yields a reward of either 0 or 1, generated randomly from a fixed Bernoulli distribution with mean  $p_a$ . An arm  $a$  is  $(\epsilon, m)$  optimal for  $\forall \epsilon \in (0, 1), \forall m \in \{1, 2, 3, \dots, n-1\}$  iff  $p_a \geq p_m - \epsilon$ .

Arms :  $1, 2, \dots, n$

Top:  $1, 2, \dots, m$

Bottom:  $m + 1, m + 2, \dots, n$

There are at least  $m$   $(\epsilon, m)$ -optimal arms. Let *Good* be the set of all  $(\epsilon, m)$ -optimal arms, and let the set *Bad* contain all the remaining arms.

$$m \leq |Good| \leq n \text{ and } 0 \leq |Bad| \leq (n - m)$$

- During round  $t$ ,  $u_a^t$  be the number of times arm  $a$  has been sampled.
- $\hat{p}_a^t$  be the empirical mean of the rewards from arm  $a$ .
- $\beta(u_a^t, t)$  is a positive number interpreted to be a high probability bound on the deviation of the empirical mean of arm  $a$  from its true mean.

$$\beta(u, t) = \sqrt{\frac{1}{2u} \ln\left(\frac{k_1 n t^4}{\delta}\right)}$$

with probability at least  $1 - \delta$  that every arm returned by the algorithm is  $(\epsilon, m)$ -optimal.

- *High* <sup>$t$</sup> : Set of  $m$  arms with the highest empirical average during round  $t$
- *Low* <sup>$t$</sup>  : Set of  $n - m$  arms with the lowest empirical averages during round  $t$
- $h_*^t$ : Among the arms in *High* <sup>$t$</sup>  , it is the arm with the lowest lower confidence bound
- $l_*^t$ : Among the arms in *Low* <sup>$t$</sup>  , it is the arm with the highest upper confidence bound

### A.5.2 Modification

Theoretical bounds for LUCB do not hold for the proposed heuristic LUCB as rewards are not IID and  $u_a^t$  is bounded by  $e$  in this set-up. The changes are explained in the algorithm 2.

The results obtained from LUCB are detailed below. The following two combination of DA are considered for the evaluation:



---

Algorithm 2: Heuristic LUCB on DAs

---

```

1: procedure LUCB ▷ Returns  $m$  models from the set of  $n$  models
2:   Train each model (arm) for a fixed number of epochs (2) and set that reward
   as the initial reward
3:    $trained\_arms = [ ]$ 
4:    $High^t = [m \text{ arms with highest initial reward}]$ 
5:    $Low^t = [n-m \text{ arms with lowest initial reward}]$ 
6:    $h_*^t = \operatorname{argmin}_{h \in High^t} \{\hat{p}_{h_*}^t - \beta(u_{h_*}^t)\}$ 
7:    $l_*^t = \operatorname{argmax}_{h \in Low^t} \{\hat{p}_{l_*}^t + \beta(u_{l_*}^t)\}$ 
8:   for  $t$  in range(total_iterations) do
9:     term =  $(\hat{p}_{l_*}^t + \beta(u_{l_*}^t)) - (\hat{p}_{h_*}^t - \beta(u_{h_*}^t))$  ▷ Check_Stopping_Criterion
10:    if term <  $\epsilon$  then
11:      return  $High^t$ 
12:    Add fully trained arms in  $trained\_arms$ 
13:     $alpha = \text{length}(trained\_arms)$ 
14:    if  $\alpha \geq m$  then
15:      return  $High^t$ 
16:     $High^t = trained\_arms \cup [m - \alpha \text{ arms (not in } trained\_arms) \text{ with highest}$ 
    reward at  $t]$ 
17:     $Low^t = [n - m \text{ arms (not in } trained\_arms) \text{ with lowest reward at } t]$ 
18:    a = list of arms in  $High^t$ , not in  $trained\_arms$ 
19:     $h_*^t = \operatorname{argmin}_{h \in a} \{\hat{p}_{h_*}^t - \beta(u_{h_*}^t)\}$ 
20:     $l_*^t = \operatorname{argmax}_{h \in Low^t} \{\hat{p}_{l_*}^t + \beta(u_{l_*}^t)\}$ 
21:    pull(  $h_*^t, l_*^t$  )
22:    Update rewards and  $u^t$ 

```

---

- **DA-I**: Shell-1024 dataset, 4-layer MLP architecture
- **DA-II**: Shell-256 dataset, 2-layer MLP architecture

For the MLP architectures, all the four reward schemes are analyzed. The distribution of datasets, good models, training procedure is same as detailed for DA-1 and DA-2 in section 7. Epochs represent the number of epochs required for the LUCB algorithm to terminate and Good indicates the number of good initialization retrieved out of the total  $m$  models returned.

### DA-1 (Shell-1024, 4-layer MLP)

Table A.3 summarizes the results. It can be seen that  $Sfv$  performs better than  $BV$  for all the sets and both values of  $m$ .

Set	m	$Sf$		$BT$		$Sfv$		$BV$	
		Epochs	Good	Epochs	Good	Epochs	Good	Epochs	Good
I	1	240	0	304	1	304	1	304	1
	3	456	1	408	2	568	2	572	2
II	1	320	0	260	1	288	0	344	0
	3	552	1	412	1	620	2	692	1
III	1	440	0	304	0	432	0	440	0
	3	512	0	416	1	592	1	584	1

Table A.3: LUCB Results on DA-1.

### DA-2 (Shell-256, 4-layer MLP)

Table A.4 summarizes the results. Scheme  $BV$  performs better for this DA.

Set	m	$Sf$		$BT$		$Sfv$		$BV$	
		Epochs	Good	Epochs	Good	Epochs	Good	Epochs	Good
I	1	236	0	236	0	525	1	416	1
	3	424	0	424	0	656	3	592	3
II	1	232	1	232	1	380	1	324	1
	3	464	2	484	2	556	3	528	3
III	1	232	0	232	0	420	1	348	1
	3	420	1	428	0	564	2	608	2

Table A.4: LUCB Results on DA-2.