Norwegian University
of Life Sciences

Master's Thesis 2024 30 ECTS
Faculty of Science and Technology

Low-Rank Methods in Julia

Bikesh Shrestha
Master of Science in Data Science

Preface

This thesis is written at the Faculty of Science and Technology at the Norwegian University of Life Sciences (NMBU) in 2024, signifying the conclusion of a two-year master's degree in Data Science.

I would like to express my sincere gratitude to my supervisor, Jonas Kusch, for his continuous supervision and guidance throughout this thesis. Without his involvement, expertise, and encouragement, this thesis would not have been completed. I am also thankful to my co-supervisor, Eirik Valseth for his help and support during the process.

I would like to thank the Faculty of Chemistry, Biotechnology, and Food Science (KBM) for providing the dataset.

Finally, I would like to thank my family members and friends for continuously motivating me while doing this thesis.

Oslo, June 17th 2024
Bikesh Shrestha

Abstract

The huge memory requirement and high computational cost of cutting-edge models limit the advancement in deep neural networks, especially when such models are deployed in resource-constrained environments. Using low-rank methods can be a good solution to this problem as they reduce the memory and computational requirements by restricting the parameter space to the manifold of low-rank matrices. Low-rank methods have been implemented in Python. In this thesis, the low-rank methods are implemented in Julia and evaluated on the MNIST dataset. These low-rank methods are implemented as vanilla low-rank training and dynamical low-rank training where dynamical low-rank training is further implemented as a fixed-rank and rank-adaptive approach. Then, the results are compared with those from the Python implementation. At last, the low-rank methods are applied to the stopped-flow biochemistry dataset to predict the rate constant for the chemical reaction.

This thesis highlights the advantages and costs of adding momentum and gradient clipping to the training algorithm. Further, the study compares the fixed-rank and the rank-adaptive approaches of dynamical low-rank training in terms of accuracy, timing, and memory requirement. The findings from the comparison between Python and Julia in terms of timing and accuracy show that Julia can be used as a tool for optimizing neural networks, however, Julia cannot save much time as it takes more epochs to converge compared to Python. This thesis also demonstrates numerically how low-rank methods are saving memory resources for both standard datasets like the MNIST dataset and real-world datasets like the stopped-flow data.

In summary, this thesis provides detailed results and a comprehensive discussion on the performance of various low-rank methods in Julia for a standard dataset like the MNIST dataset and a real-world dataset like the stoped-flow biochemistry data.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Context

Deep neural networks have been a factor behind the progress in machine learning, however, most of the high-performing networks need prohibitive amounts of computation and memory storage [1]. Such need for a prohibitive amount of computation and memory leads to the increase in the infrastructure cost and the challenges in the deployment of such network in low resource devices [2], [3]. This issue can be addressed in different ways.

A large portion of the parameters can be removed from an existing network by employing a technique called network pruning so that a significantly smaller network with similar accuracy is obtained from a large and accurate network [4]. Binarized neural networks are specifically neural networks with binary(+1 or -1) weights. These binary weights and activation are used for computing parameter gradients during running and it is one of the effective network pruning techniques [5]. However, these solutions are not efficient enough to be applicable because they cannot be pruned during training. Hence, some other methods need to be explored further.

When a neural network's parameter matrix is a large data matrix, it is rarely a full-rank matrix, thus, it is better to constrain the parameter space to the manifold of low-rank matrices [6]. Linear compression techniques can be employed by initiating compression at each convolution layer where an appropriate low-rank approximation is found, followed by fine-tuning the upper layers until the pre-

diction performance is restored to some extent [2]. A simple approach is vanilla training in which the weight matrices are factorized as the product of matrices U, Σ, and $V^\top$ by Singular Value Decomposition (SVD) before applying a training algorithm on U, Σ, and V separately. This vanilla training can be implemented as vanilla warm-up training in which vanilla training is applied to a model for a few epochs and matrix factorization is conducted again on the partially trained model [7]. This warm-up phase is required because the training speed depends on the curvature of the manifold. Whereas, a better solution is to interpret the training problem as a continuous-time gradient flow, for dynamical low-rank approximation, so that low-rank numerical integrator can be used on the matrix ordinary differential equations that modify the training phases to contain only the low-rank parameter matrices [8].

In order to carry out efficient and robust training steps, a Dynamical Low-Rank Training (DLRT) algorithm has been developed in [9] that has three three gradient tapes corresponding to each low-rank component, instead of a single gradient tape of the full weight matrix network.

1.2 Problem Statement

In [9], the DLRT algorithm is implemented in Python for both Tensorflow and Pytorch on common datasets like MNIST, ImageNet1K, and Cifar10. The implementation of the algorithm in faster programming languages is yet to be done. Additionally, the performance of the DLRT algorithm on more complex real-world datasets needs to be observed. To provide direction to this research, the following research questions were formulated:

1. **Q1**: Is the Julia Flux package a viable option to efficiently implement low-rank neural networks? Which optimizers provided in the Flux package can be used to train low-rank neural networks, and do they need to be adapted to achieve a good performance?

2. **Q2**: How does the Flux package differ in computational time and accuracy compared to existing Pytorch implementations of low-rank neural networks? How do different low-rank architectures compare when trained with the Flux package?

3. **Q3**: How do low-rank neural networks perform for a standard benchmark like the MNIST dataset, and how do they perform for real-world data like a

biochemistry dataset?

1.3 Proposed Solution

This thesis is primarily based on the application of the DLRT algorithm from [9] in the Julia programming language. Julia is a high-performance and high-level dynamic programming language for technical and numerical computing and it has syntax familiar to technical computing environment users [10]. Julia has productivity and interactive dynamic behavior similar to Python and in addition, it has the performance of a statically compiled language [11]. The results from Julia's implementation will be compared with those from the Python implementation. At last, the DLRT algorithm from [9] will be implemented in Julia demonstrating its application in a bio-chemistry dataset called stopped-flow data that is obtained from the rapid mixing of enzyme and substrate [12]. Here, the DLRT algorithm will be applied to predict the rate constant for the chemical reaction calculated after fitting the stopped-flow data to an exponential function.

1.4 Paper Structure

This thesis begins with chapter 1 that provides the background behind the thesis and some context to start the thesis. This chapter also provides the problem statement and the proposed solution. This chapter is followed by chapter 2 which provides brief theoretical concepts of neural networks, related mathematics, and the current state-of-the-art. Then in chapter 3, the materials such as the programming language and the packages, algorithm, and a brief description of the dataset are provided. The methodology adopted to carry out the thesis is provided in the chapter 4. Then the results obtained from numerous neural network models are provided in the chapter 5. A comprehensive discussion on various topics in the chapter 6 follows these results. The findings of the research are summarized in the chapter 7. At last, the limitations of the thesis are given along with the possible future works in the chapter 8.

The Julia scripts for this study are present in the GitHub repository: https://github.com/BikeshNMBU/Low-rank-methods-in-Julia

Chapter 2

Theory

2.1 Fundamentals of Neural Networks

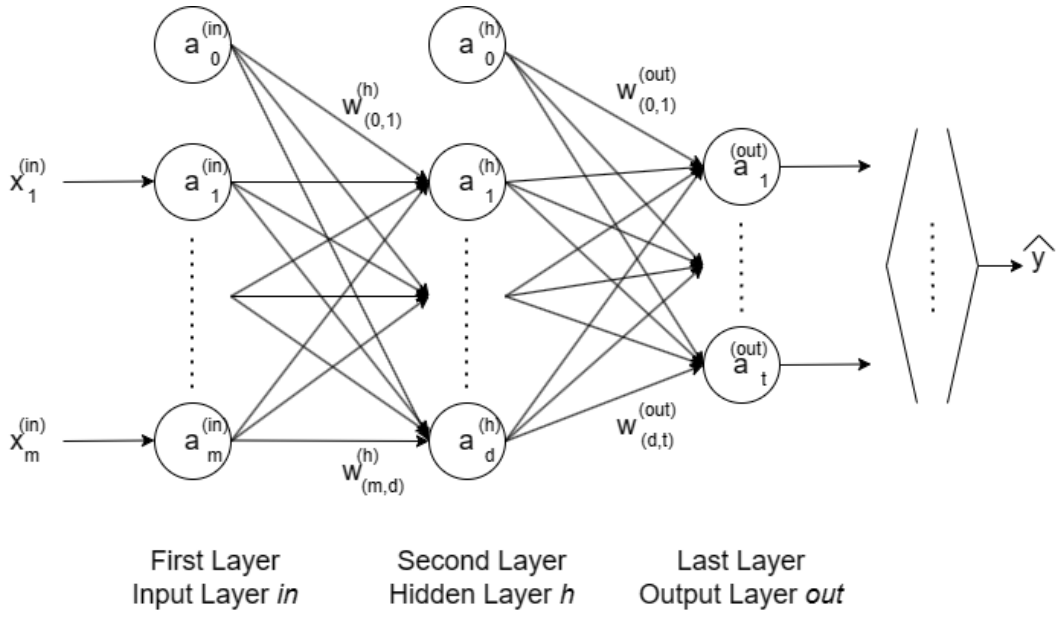


Figure 2.1: An MLP consisting three layers: one input, one hidden, and one output layer. The figure is adapted from the textbook [13].

Some hypotheses and models based on the approaches taken by the human brain to solve complex problems facilitate the building of the basic concepts behind artificial neural networks. Artificial neural networks, often called neural networks, are of two types: feedforward and recurrent neural networks. Feedforward neural networks can be classified as fully connected and convolutional feedforward neural networks [13]. This thesis is based on the fully connected feedforward neural network which is also known as multilayer perceptron(MLP) [13].

An illustration of the concept of an MLP with three layers adapted from [13] is given in Figure 2.1. In this figure, the circles are called units or neurons. The above figure shows that the units in the hidden layer and the output layer are fully connected to the preceding layer. When such a network has more than one hidden layer, it is also called a deep artificial neural network [13].

In the figure 2.1, the superscripts in, h, and out represent the layer type: input, output, and hidden. Similarly, the subscript represents the position of the unit in the layer. Here, m, d, and t represent the number of features, hidden units, and outputs, respectively. In classification problems, the number of outputs is the number of classes. Whereas in regression problems, the number of outputs is the number of variables whose value is to be estimated. The units $a_0^{(in)}$ and $a_0^{(h)}$ are the bias units. The arrows represent the connection from a unit in layer l to one of the units in layer $l+1$. The connections are from a unit in the input layer to a unit in the hidden layer and from a unit in the hidden layer to a unit in the output layer [13].

2.1.1 Overview of feedforward neural networks

Feedforward neural networks are the simplest types of artificial neural networks. In these neural networks, the information moves only in the forward direction, starting from the input layer containing input nodes, going through the hidden layer(s) containing hidden nodes, and finally to the output layer containing output nodes. These neural networks do not contain feedback connections in the form of loops or cycles. Thus, the outputs are not fed back into the network model itself [13], [14]. The straightforward structure of the feedforward neural networks results in the ease of implementation. These neural networks can be used in a variety of classification and regression problems, thus forming the basis of many commercial applications. Moreover, feedforward neural networks form the conceptual basis for the recurrent neural networks [14].

2.1.2 Activation Functions

In a neural network, a function is applied to every layer, known as the activation function. Activation functions introduce non-linear properties that help the neural networks to learn complex patterns in the data. It is usually denoted by $\varphi(.)$. Sigmoid function, hyperbolic tangent (tanh) function, and Rectified Linear Unit (ReLU) function are common activation functions. The sigmoid function produces an output in the range between 0 and 1, whereas the tanh function produces an output in the range between -1 and 1. The ReLU function gives a zero output for a negative input and a linear output for positive inputs. The activation function needs to be chosen properly to achieve good performance during training and evaluation [13], [14]. In the input and hidden layers, the ReLU activation function can be used in most cases as it is faster to compute and it does not saturate for larger input values. Whereas, for the output layers, softmax activation functions can be used for classification tasks, which simply output a probability distribution of possible outcomes [15].

2.1.3 Weights and Biases

Weights are the parameters of a neural network that determine the connection strength between neurons in different layers. The impact of input signals on the output of the neurons is controlled by the weights by scaling the input data during training. Biases are added to the weighted input before the activation function is applied; thus, the neurons can adjust the output independently of their inputs. Mathematically, weights are matrices, and biases are vectors. During the training process, the weights and biases are adjusted by the neural networks to fit the labeled data better through optimization techniques, where the main goal is to minimize the difference between the actual output and the output predicted by the network [13], [14], [15].

An $i \times j$ matrix can be defined as a rectangular array of numbers having i rows and j columns. Here, i and j are positive integers. A vector is an ordered list of numbers. When a matrix has only one column, then it is called a column vector or just a vector [16, Chapter 1]. Mathematically, weights are matrices, and biases are vectors [13], [15]. Referring to figure 2.1, the weight matrix connecting the input to the hidden layer can be denoted as $W^{(h)}$. Similarly, the weight matrix connecting the hidden to the output layer can be denoted as $W^{(out)}$ [13].

2.1.4 Forward Propagation

A feedforward neural network uses a mechanism called forward propagation to process incoming data and produce predictions, which requires a sequence of matrix multiplications in which the network's weight matrices are multiplied by the input matrix one after the other. The output of each multiplication is then passed through an activation function to move on to the next layer. Matrix multiplication is computationally demanding, particularly as input data size and network architecture complexity rise. The computational cost of these operations grows with the number of neurons in each layer and the depth of the network [13], [15].

Referring to the figure 2.1, the activation unit of the hidden layer $a_1^{(h)}$ ca be calculated as below [13]:

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^{(h)} + a_1^{(in)} w_{1,1}^{(h)} + \cdots + a_m^{(in)} w_{m,1}^{(h)} \tag{2.1}$$

$$a_1^{(h)} = \varphi\left(z_1^{(h)}\right) \tag{2.2}$$

Where $z_1^{(h)}$ is the net input. These expressions can be written in a more compact format as below [13]:

$$\boldsymbol{z}^{(h)} = \boldsymbol{a}^{(in)} \boldsymbol{W}^{(h)} \tag{2.3}$$

$$\boldsymbol{a}^{(h)} = \varphi\left(\boldsymbol{z}^{(h)}\right) \tag{2.4}$$

Here, $\boldsymbol{a}^{(in)}$ is a $1 \times m$ dimensional feature vector made up of a sample $\boldsymbol{x}^{(in)}$ and a bias unit. $\boldsymbol{W}^{(h)}$ is an $m \times d$ dimensional weight matrix. $\boldsymbol{z}^{(h)}$ the $1 \times d$ dimensional net input vector to calculate the activation $\boldsymbol{a}^{(h)}$ (where $\boldsymbol{a}^{(h)} \in \mathbb{R}^{1 \times d}$) [13].

Similarly, the expressions for the output layer can be obtained as below:

$$\boldsymbol{z}^{(out)} = \boldsymbol{a}^{(h)} \boldsymbol{W}^{(out)} \tag{2.5}$$

$$\boldsymbol{a}^{(out)} = \varphi\left(\boldsymbol{z}^{(out)}\right) \tag{2.6}$$

Here, $\boldsymbol{W}^{(out)}$ is an $d \times t$ dimensional weight matrix. $\boldsymbol{z}^{(out)}$ is the $1 \times t$ dimensional net input vector to calculate the activation $\boldsymbol{a}^{(out)}$ (where $\boldsymbol{a}^{(out)} \in \mathbb{R}^{1 \times t}$) [13].

The above equations can be generalized to all n samples in the train set by replacing $\boldsymbol{a}$ with $\boldsymbol{A}$ as below [13]:

$$\boldsymbol{Z}^{(h)} = \boldsymbol{A}^{(in)} \boldsymbol{W}^{(h)} \quad \text{(hidden layer net input)} \tag{2.7}$$

$$\boldsymbol{A}^{(h)} = \varphi(\boldsymbol{Z}^{(h)}) \quad \text{(hidden layer activation)} \tag{2.8}$$

$$\boldsymbol{Z}^{(out)} = \boldsymbol{A}^{(h)} \boldsymbol{W}^{(out)} \quad \text{(output layer net input)} \tag{2.9}$$

$$\boldsymbol{A}^{(out)} = \varphi(\boldsymbol{Z}^{(out)}) \quad \text{(output layer activation)} \tag{2.10}$$

2.1.5 Loss Function

The loss function computes the difference between the predicted output of the neural network and the actual target values. By giving the quantity of this difference, the loss function helps the neural network improve its performance through an optimization process. It is also called the objective function or cost function. There are many loss functions available that need to be used according to the nature of the task. For example, Cross-Entropy Loss is used for classification problems, whereas Mean Squared Error is used for regression problems. The training process aims to minimize the loss function by adjusting the weights and biases of the network. The network predictions are more accurate when the loss function is minimized [13], [15].

2.1.6 Mathematical definition

A feed-forward fully-connected neural network denoted by $\mathcal{N}(x) = a_M$ has $a_0 = x \in \mathbb{R}^{n_0}$ as the input to the network and $a_k = \varphi_k(W_k a_{k-1} + b_k) \in \mathbb{R}^{n_k}$, $k = 1, \ldots, M$ as the output of the k-th layer of the network where φ_k, W_k and b_k are the activation function, weights and biases of that k-th layer.It can be trained based on the optimization of a loss function $\mathcal{L}(W_1, \ldots, W_M; \mathcal{N}(x), y)$ [9].

2.1.7 Backpropagation

Referring to figure 2.1, the backpropagation propagates the error from right to left [13]. At first, the backpropagation algorithm makes a prediction for each training instance. Then, it calculates the error as the difference between the predicted and the actual label. Then, the algorithm travels through each layer in reverse and measures the error contribution from each layer. Finally, the layer weights are updated to reduce the error [15].

With reference to the figure 2.1, the backpropagation can be summarized with the help of the following equations. These equations are adapted from the textbook [13]. First, the error term of the output layer is defined with the help of the actual labels y.

$$\boldsymbol{\delta^{(out)}} = \boldsymbol{a}^{(out)} - \boldsymbol{y} \tag{2.11}$$

Then, the loss gradient is computed.

$$\frac{\partial}{\partial w_{i,j}^{(out)}} \mathcal{L}(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)} \tag{2.12}$$

Further, the error term of the hidden layer is defined.

$$\boldsymbol{\delta^{(h)}} = \boldsymbol{\delta^{(out)}} \left(\boldsymbol{W}^{(out)}\right)^\top \odot \frac{\partial \phi\left(\boldsymbol{a}^{(h)}\right)}{\partial \boldsymbol{a}^{(h)}} \tag{2.13}$$

The loss gradient is computed again.

$$\frac{\partial}{\partial w_{i,j}^{(out)}} \mathcal{L}(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)} \tag{2.14}$$

The gradient is the extension of the concept of the derivative to functions with multiple variables. It depicts the rate of change of the function in different directions from a point. This allows us to estimate the change in function when taking a small step from that point in any direction [17]. The derivative describes the rate of change of a real-valued function at a given point. For instance, $f'(x)$ or $\frac{df(x)}{dx}$ is the derivative of a function $f(x)$ with respect to the variable x. It describes the rate at which the function $f(x)$ alters at the particular point x. Derivatives are valuable in optimization as they provide information on changing a particular point to decrease the objective function [18]. When dealing with a multivariate function, the partial derivative of the function with respect to one of the variables can be calculated as the derivative of the function with respect to that variable while holding all other variables as constant. For instance, $\frac{\partial}{\partial y} f(x, y)$ is the partial derivative of a function $f(x, y)$ with respect to the variable y and the variable$x$ is held constant. Gradient is a vector of derivatives where each component is a partial derivative [17], [18].

2.2 Low-Rank Matrix Approximation

2.2.1 Rank

A basic idea in linear algebra is the rank of a matrix, which is the maximum number of linearly independent column vectors or row vectors in the matrix. A higher number of linearly independent vectors is implied by a high-rank matrix, which

frequently denotes more complicated and diverse data. A low-rank matrix, on the other hand, contains fewer linearly independent vectors, indicating correlations or redundancies in the data [19]. These redundancies and correlations are crucial in dimensionality reduction and data compression methods [20].

2.2.2 SVD

SVD is a technique in which a matrix A of size $m \times n$ and rank r is decomposed into three matrices U, Σ, and $V^\top$ as shown in equation (1) below.

$$A = U\Sigma V^\top \tag{2.15}$$

Here, U is an orthogonal matrix of size $m \times m$, containing the left singular vectors of A, and V is another orthogonal matrix of size $n \times n$ containing the right singular vectors of A whose transpose $V^\top$ is being used in the equation. Σ is called a singular matrix of size $m \times n$ for which the singular values of A are in the diagonal entries.

$$\Sigma = \left[\begin{array}{c|c} D & 0 \\ \hline 0 & 0 \end{array}\right] \tag{2.16}$$

D in equation (2) is an $r \times r$ diagonal matrix containing the r singular values of A. Apart from D, others are blocks of zeros that help to make the size of Σ as $m \times n$.

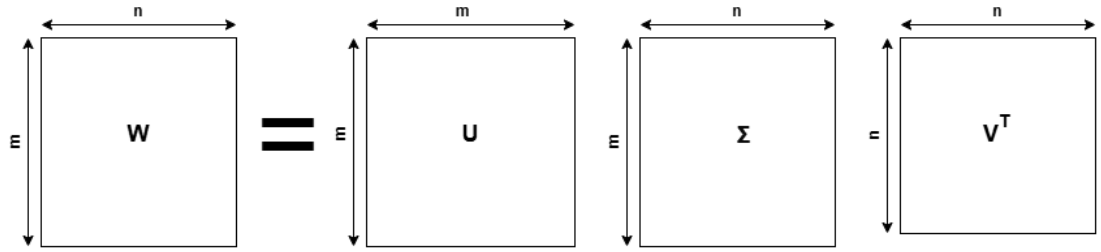


Figure 2.2: SVD of a matrix W of size $m \times n$. The Σ matrix has size $m \times n$. The figure is inspired by the knowledge obtained from the textbook [16].

There are essential variations of Σ known as compact Σ and truncated Σ. Compact Σ is simply the diagonal matrix D of size $r \times r$ such that the sizes of U and $V^\top$ become $m \times r$ and $r \times n$. Whereas, truncated Σ is the diagonal matrix of size $k \times k$ containing k most significant singular values of A in the diagonal entries where $k < r$, such that the sizes of U and $V^\top$ becomes $m \times k$ and $k \times n$ [16, Chapter 7].

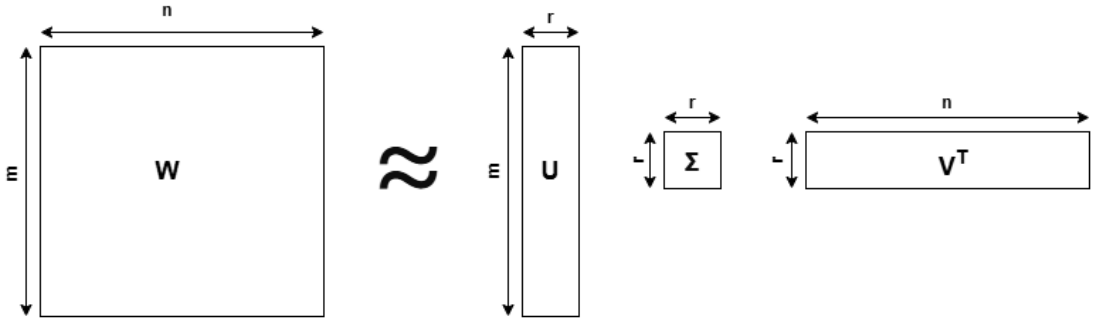


Figure 2.3: Truncated SVD of a matrix W of size $m \times n$ with truncated Σ which is a diagonal matrix of size $r \times r$ containing r largest singular values. The actual rank of matrix W is much larger than r. The figure is inspired by the knowledge obtained from the textbook [16].

From the above two figures, it can be clearly observed that a large matrix W can be approximated by two thin and tall matrices and a small diagonal matrix without losing much information.

2.3 Optimization Techniques for Training Neural Networks

2.3.1 Gradient Descent

The gradient is the extension of the concept of the derivative to functions with multiple variables. It depicts the rate of change of the function in different directions from a point. This allows us to estimate the change in function when taking a small step from that point in any direction [17].

Gradient descent, a popular optimization technique in machine learning, is defined as the process of updating the parameters in the opposite direction of the gradient of the objective function with respect to the parameters. The model performance is enhanced by iterating this process. The goal is to minimize an objective function that is parameterized by a model's parameters. The learning rate determines the size of the steps to be taken while descending downhill along the surface's slope created by the objective function until a valley is reached. Thus, the learning rate helps in determining the number of steps required to obtain a (local) minimum

[15], [21].

Employing gradient descent, the weights of $\mathcal{N}$ at iteration $t \in \mathbb{N}$ are updated for a learning rate λ with the help of an equation given below [9].

$$W_k^{t+1} = W_k^t - \lambda \nabla_{W_k} \mathcal{L}(W_1, \ldots, W_M; \mathcal{N}(x), y), \quad \forall k = 1, \ldots, M \tag{2.17}$$

Here, $\nabla_{W_k} \mathcal{L}()$ represents the gradients of loss function $\mathcal{L}$ with respect to the weights W.

2.3.2 Variants of Gradient Descent

Three frequently employed variants of gradient descent are Batch Gradient Descent, Stochastic Gradient Descent (SGD), and Mini-Batch Gradient Descent. They differ in the quantity of data needed to compute the gradient of the objective function. Batch gradient descent uses the complete dataset to calculate the objective function's gradient. With a learning rate controlling the update's magnitude, it modifies the parameters in the direction of the gradients. SGD, on the other hand, updates a parameter for every label and training sample. SGD prevents redundant computations done by batch gradient descent that occur while computing the gradients of similar training samples. Thus, SGD is much faster. Finally, mini-batch gradient descent uses a mini-batch, or a small subset of the data, to compute the gradient instead of the full dataset, achieving a compromise between batch gradient descent and SGD [15], [21].

For a learning rate λ, the batch gradient descent updates the parameters ω using the following equation [21]:

$$\omega = \omega - \lambda \cdot \nabla_\omega \mathcal{L}(\omega) \tag{2.18}$$

Similarly, SGD updates the parameters for every label $y^{(i)}$ and training sample $x^{(i)}$ using the following equation [21]:

$$\omega = \omega - \lambda \cdot \nabla_\omega \mathcal{L}(\omega; x^{(i)}; y^{(i)}) \tag{2.19}$$

At last, mini-batch gradient descent updates the parameters for every mini-batch of n training samples and labels using the following equation [21]:

$$\omega = \omega - \lambda \cdot \nabla_\omega \mathcal{L}(\omega; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.20}$$

2.3.3 Gradient descent optimization algorithms

A few challenges may be encountered during training while using the gradient descent variants as the optimizer. It can be difficult to choose a proper learning rate that is neither too small nor too large. Also, the learning rate is the same for all the parameters, but the features may need different magnitudes of updates. Moreover, there is a risk of getting trapped in suboptimal local minima in some cases. To address those challenges, there are some gradient descent optimization algorithms frequently employed in the Deep Learning community, such as Momentum, AdaGrad, RMSprop, and Adam [15], [21]. Among these algorithms, momentum is implemented in this thesis.

SGD has trouble navigating deep narrow valleys around suboptimal local minima. Mathematically, these are the regions of ill-conditioned Hessian matrices [22]. This problem is mitigated by momentum, which enables the optimizer to increase velocity in gradient-consistent directions and reduce oscillations in high-variance directions. Applying momentum is analogous to pushing a ball down a hill. The ball starts slowly, but it quickly picks up the momentum and reaches the terminal velocity. Momentum optimization considers previous gradients as well and uses them to reach the minimum faster. At each iteration, local gradient multiplied by learning rate is added to the momentum vector v and the resulting momentum vector is subtracted from the parameter which updates that parameter. A fraction γ of the past time momentum vector is added to the current one. This fraction hyperparameter γ prevents the momentum from growing too large and its value needs to be set between 0 and 1.[15], [21].

$$\begin{aligned} v_t &= \gamma v_{t-1} + \lambda \nabla_\omega \mathcal{L}(\omega) \\ \omega &= \omega - v_t \end{aligned} \tag{2.21}$$

2.4 Low Rank Methods

In deep neural networks, low-rank factorization is a model compression and acceleration technique. By approximating weight matrices with lower-rank matrices, low-rank factorization techniques seek to minimize the computational complexity and memory footprint of neural network models. Faster computations can be accomplished in the training and inference phases because of this approximation [23]. By substituting the weight matrix with three smaller matrices, the approach drastically decreases the number of parameters in the network, resulting in quicker training periods without significant loss of accuracy [24].

Low-rank methods are primarily based on the SVD where the singular matrix is truncated Σ such that the matrices U, Σ, and $V^\top$ are of much smaller sizes. Thus, the large-sized weight matrices would be approximated by much smaller U, Σ, and $V^\top$ matrices. In these methods the training algorithms are applied on the small-sized U, Σ, and V matrices instead of large-sized weight matrices [9], [25].

2.4.1 Low-rank training through gradient flow

In the case of the traditional descent algorithms such as (2.3), the low rank structure $U_k\Sigma_k V_k^\top$ are not necessarily preserved during training, and the knowledge of the whole W_k is required rather than the factors U_k, Σ_k and V_k. In such traditional systems, the weight updates through loss minimization are discrete. However, the loss minimization leading to weight updates can be reinterpreted as as a continuous time gradient flow giving a new training method which overcomes the previously mentioned limitations [9].

Diminishing the loss function with respect to W_k as shown in (2.3) is similar to assessing the long time behaviour of the matrix Ordinary Differential Equation (ODE) given below which enables us to interpret the training phase as a continuous process [9].

$$\dot{W}_k(t) = -\nabla_{W_k}\mathcal{L}(W_1, \ldots, W_M; \mathcal{N}(x), y), \tag{2.22}$$

Here, the 'dot' denotes the time-derivative. If $\mathcal{M}_{r_k}$ represents a matrix manifold containing matrices with rank r_k and at a time t_0, the weights lie in the matrix manifold, i.e., $W_k(t_0) \in \mathcal{M}_{r_k}$, then we can use this continuous time interpretation to obtain a strategy to develop the weights by referring to the dynamics in (2.4) so as $W_k(t) \in \mathcal{M}_{r_k}$ for all $t \geq t_0$ [9].

When the dynamical system of a single weight matrix W_k is considered, the remaining weight matrices are fixed in time and are regarded as the gradient's parameters. These parameters can be left out in further derivations to keep it concise and focused. Supposing $W_k(t) \in \mathcal{M}_{r_k}$, we can express (2.4) as

$$\min\left\{\left\|\dot{W}_k(t) + \nabla_{W_k}\mathcal{L}(W_k(t))\right\|_F : \dot{W}_k(t) \in \mathcal{T}_{W_k(t)}\mathcal{M}_{r_k}\right\} \tag{2.23}$$

where $\mathcal{T}_{W_k(t)}\mathcal{M}_{r_k}$ is the tangent space of $\mathcal{M}_{r_k}$ at position $W_k(t)$ [9].

For the individual factors of W_k, which are U_k, Σ_k and V_k, a system of matrix

ODEs can be written as below [9]:

$$\begin{cases} \dot{\Sigma}_k = -U_k^\top \nabla_{W_k} \mathcal{L}(W_k(t)) V_k, \\ \dot{U}_k = -(I - U_k U_k^\top) \nabla_{W_k} \mathcal{L}(W_k(t)) V_k \Sigma_k^{-1}, \\ \dot{V}_k = -(I - V_k V_k^\top) \nabla_{W_k} \mathcal{L}(W_k(t))^\top U_k \Sigma_k^{-\top}. \end{cases} \tag{2.24}$$

2.5 Review of Related Works

Many network pruning techniques have been proposed in recent years and it is not possible to accurately compare them because of the missing baseline and inconsistent experimental settings [4]. In recent years, significant research effort has been devoted to developing low-rank factorization strategies for deep neural networks, building on extensive studies of low-rank factorization using the SVD and other matrix decomposition techniques and training with either fixed low rank or variable low rank in the scientific computing and machine learning communities [9].

In [26], the fixed rank approach is being used to construct a low-rank basis of filters that are rank-1 in the spatial domain. In [25] it is shown that low-rank methods based on fixed rank, specifically factorized neural layers with spectral initialization and Frobenius decay, can outperform structured sparsity methods and even some uncompressed models in terms of accuracy and memory efficiency and these methods can provide acceleration via low-rank convolutions. Similarly, in [27] a scheme for representing convolutional filters as linear combinations of basis filters with different spatial dimensions is proposed such that composite layers comprise several sets of filters where the filters in each set have different spatial dimensions allowing for the creation of layers as linear combinations of layers of different ranks. Fixed rank methods require the rank of the matrix decomposition to be fine-tuned, which is computationally expensive and difficult to optimize, whereas rank-adaptive methods automatically determine and adapt the low-rank structure after training, eliminating the need for hyperparameter grid search [9].

In [28], the initial low-rank approximation is followed by a fine-tuning phase for a certain number of epochs where the network's weights are adjusted to improve the accuracy of the approximated network and the hyperparameters except learning rate and momentum used are reused from the training phase. Similarly, the algorithm described in [29] uses a variable rank approach to determine the optimal rank of each layer in a neural net by interleaving stochastic gradient descent

steps that train the uncompressed net with SVD steps that determine the currently optimal rank and weight matrices. Rank-adaptive approaches like these still need the full weight matrix information during the training and thus require more computations than standard training [9].

To solve the previous issues, a new training algorithm based on a variable rank approach for efficient low-rank neural networks that determines and adapts low-rank subnetworks during training, reducing time and memory resources is proposed in [9], which is based on a gradient flow of the network weight matrices and is formulated as a matrix Ordinary Differential Equation (ODE), allowing for the use of recent advances in dynamic low-rank approximation methods for matrix ODEs and finally evolve the solution of the differential equation on a low-rank manifold. This algorithm is going to be implemented in Julia in this work and the results will be compared with its previous implementation in Python.

New users were easily able to read, modify, imitate, and extend the standard library code after Julia was released in February 2012 as an open-source project leading to the growth of a significant community in a short time [11]. The just-in-time compilation feature makes Julia faster than Pytorch and its design makes it more suitable for large-scale machine-learning tasks [10]. In [11], the speed of Julia has been compared with that of six other languages: C++, Python, MATLAB, Octave, R, and JavaScript and timings for five scalar micro-benchmarks and two simple array benchmarks have been tabulated, where it can be seen that Julia is taking less time than python in all of the tests and Julia is a lot faster than python in six of the tests.

Chapter 3

Materials

3.1 Julia Flux

3.1.1 Julia

Julia is a high-level, flexible, dynamic programming language that performs well in scientific and numerical computing. The syntax is familiar to the users of technical computing environments [10], [30]. Julia's productivity and interactive dynamic behavior are similar to those of Python. However, the performance of Julia is identical to that of a statically compiled language like C++ [11]. Julia is designed to be easy and fast. Julia can efficiently execute numerical and computational tasks, providing high performance through the just-in-time compilation using Low-Level Virtual Machine (LLVM) [31]. Furthermore, the functionality and syntax of Julia are designed to be expressive and concise so that the lines of codes required for the complicated tasks are reduced, and the codes' readability and maintainability are enhanced [32].

Optional typing, multiple dispatch, and good performance are the features provided by Julia type inference and just-in-time compilation (using LLVM). Julia is a multi-paradigm language because it combines the features of imperative, functional, and object-oriented programming. The expressiveness and ease of Julia language are similar to dynamic languages such as Python, R, and MATLAB for high-level numerical computing. Besides, it also supports general programming. Though Julia is built upon the ancestry of mathematical programming languages, it also adopts many well-known dynamic programming languages like Lisp, Lua, Perl,

Python, and Ruby [30]. Detailed information about Julia can be obtained from the documentation [30].

3.1.2 Flux.jl

Flux.jl is a machine learning library based on the Julia programming language. It enables us to create models employing the simple mathematical syntax of Julia and train them using automatic differentiation. To analyze and optimize code, Flux exploits the characteristics of the Julia language, such as support for GPU compilation. Further, it enables the just-in-time creation of unique GPU kernels for model layers [33]. Flux has many functional built-in tools. However, it also allows one to utilize Julia's full power when required. It provides only a few explicit APIs and encourages the users to write down the mathematical form directly. Flux can give good performance and high flexibility and can be extended.

Moreover, it performs well with unrelated Julia libraries, from images to differential equation solvers. Flux re-exports gradient from Zygote and uses this function to differentiate the model during training [34]. In the world of Julia, Zygote.jl is a ground-breaking tool for automatic differentiation that supports various programming structures by exploiting reverse-mode algorithmic differentiation based on source code modification. Zygote is closely integrated with the compiler and is compatible with the Flux machine learning stack. Thus, it is an essential tool for machine learning capabilities and provides high-performing and efficient code [35]. Flux supports all the necessary built-in layers, activation functions, weight initialization functions, training APIs, optimization rules, and other essential items required for machine learning [34]. Detailed information about Flux can be obtained from the documentation [34].

3.2 KLS-based Algorithm

The system of ODEs (2.6) can be numerically integrated according to the unconventional KLS integrator and its rank-adaptive version such that robust and efficient training steps can be performed [36], [8], [9].

In this KLS-based algorithm, the product $W_k = U_k \Sigma_k V_k^\top$ is alternately represented as $W_k = K_k V_k^\top$ and $W_k = U_k L_k^\top$ and the corresponding coupled ODEs from (2.6) are considered to perform the following steps [9]:

1,2. K&L-steps (in parallel)
The current K_k and L_k are updated by integrating the differential equations

$$\begin{cases} \dot{K}_k(t) = -\nabla_{W_k}\mathcal{L}(K_k(t)V_k^\top)V_k, & K_k(0) = U_k\Sigma_k, \\ \dot{L}_k(t) = -\nabla_{W_k}\mathcal{L}(U_kL_k(t)^\top)^\top U_k, & L_k(0) = V_k\Sigma_k^\top, \end{cases} \tag{3.1}$$

from $t = 0$ to $t = \eta$; then new orthonormal basis matrices $\tilde{U}_k$ and $\tilde{V}_k$ are formed which span the range of the computed $K_k(\eta)$ and $L_k(\eta)$ using a QR factorization.

3. S-step
The current Σ_k is updated by integrating the differential equation

$$\dot{\Sigma}_k(t) = -\tilde{U}_k^\top\nabla_{W_k}\mathcal{L}(\tilde{U}_k\Sigma_k(t)\tilde{V}_k^\top)\tilde{V}_k \tag{3.2}$$

from $t = 0$ to $t = \eta$, where the initial value condition is $\Sigma_k(0) = \tilde{U}_k^\top U_k\Sigma_k V_k^\top\tilde{V}_k$.

This algorithm can be easily extended to rank adaptivity, allowing us to dynamically evolve the rank of Σ_k; hence, the rank of W_k is also evolved simultaneously [8]. In the K and L-steps, the dimension of the calculated basis matrices U_k and V_k are doubled by calculating orthonormal bases that span $[K_k(\eta) \mid U_k]$ and $[L_k(\eta) \mid V_k]$, respectively, i.e., the current basis is augmented with the basis obtained from the previous time step. Then, a truncation step is performed on the new Σ_k matrix calculated by the S-step. This truncation step removes all the singular values from the newly computed S_k matrix under a certain threshold θ [9].

The pseudo-code of the DLRT algorithm, adapted from [9], is provided below.

Algorithm: Dynamic Low Rank Training (DLRT)

Input: Initial low-rank factors S_k^0 of size $r_k^0 \times r_k^0$, U_k^0 of size $n_k \times r_k^0$, V_k^0 of size $n_{k-1} \times r_k^0$ for $k = 1, \ldots, M$;
iter: maximum iterations for each epoch;
adaptable: Boolean flag for deciding rank adaptability;
θ: singular value threshold required to determine new rank while adapting.

for *each epoch* **do**
  for $t = 0$ **to** *iter* **do**
    for *each layer* k **do**
      $K_k^t \leftarrow U_k^t \Sigma_k^t$; // K-step
      $K_k^{t+1} \leftarrow$ one-step-integrate $\left\{\dot{K}(t) = -\nabla_K \mathcal{L}(K(t)(V_k^t)^\top z_{k-1} + b_k^t), K(0) = K_k^t\right\}$
      $L_k^t \leftarrow V_k^t (\Sigma_k^t)^\top$; // L-step
      $L_k^{t+1} \leftarrow$ one-step-integrate $\left\{\dot{L}(t) = -\nabla_L \mathcal{L}(U_k^t L(t) z_{k-1} + b_k^t), L(0) = L_k^t\right\}$
      if *adaptable* **then**
        // Augmentation of basis
        $K_k^{t+1} \leftarrow \left[K_k^{t+1} \middle| U_k^t\right]$
        $L_k^{t+1} \leftarrow \left[L_k^{t+1} \middle| V_k^t\right]$
      $U_k^{t+1} \leftarrow$ orthonormal basis for the range of K_k^{t+1} ; // S-step
      $M_k \leftarrow (U_k^{t+1})^\top U_k^t$
      $V_k^{t+1} \leftarrow$ orthonormal basis for the range of L_k^{t+1}
      $N_k \leftarrow (V_k^{t+1})^\top V_k^t$
      $\tilde{\Sigma}_k^t \leftarrow M_k S_k^t N_k^\top$
      $\Sigma_k^{t+1} \leftarrow$ one-step-integrate $\left\{\dot{\Sigma}(t) = -\nabla_\Sigma \mathcal{L}(U_k^{t+1} \Sigma(t) (V_k^{t+1})^\top z_{k-1} + b_k^t), \Sigma(0) = \tilde{\Sigma}_k^t\right\}$
      if *adaptable* **then**
        // Compression of Rank
        $F, Sig, G \leftarrow \mathrm{SVD}(S_k^{t+1})$
        $\Sigma_k^{t+1} \leftarrow$ truncate Sig using the singular value threshold θ
        $U_k^{t+1} \leftarrow U_k^{t+1} \tilde{P}$ // where $\tilde{F} = [$first r_k^{t+1} columns of $F]$
        $V_k^{t+1} \leftarrow V_k^{t+1} \tilde{Q}$ // where $\tilde{G} = [$first r_k^{t+1} columns of $G]$
      // Bias update step
      $b_k^{t+1} \leftarrow$ one-step-integrate $\left\{\dot{b}(t) = -\nabla_b \mathcal{L}(U_k^{t+1} \Sigma_k^{t+1} (V_k^{t+1})^\top z_{k-1} + b(t)), b(0) = b_k^t\right\}$

3.3 Datasets

3.3.1 MNIST dataset

The Modified National Institute of Standards and Technology (MNIST) dataset is one of the most widely used datasets in deep learning. It contains 70,000 handwritten digit images, 60,000 in the training set, and 10,000 in the test set, where each image is 28x28 pixels in greyscale and represents digits within a range of 0 to 9. In evaluating the performance of various deep learning algorithms for image recognition and classification, this dataset has been a fundamental benchmark [37]. The reasons that make the MNIST dataset suitable for experimenting with neural network optimizations and methodologies where a balance of computational tractability and real-world applicability is required are the relatively simple format and comprehensible scale of the MNIST dataset [38].

3.3.2 Stopped-flow data

Stopped-flow data is the biochemistry dataset created from the rapid mixing of enzyme and substrate [12]. The data is received from one of the research groups under the Faculty of Chemistry, Biotechnology, and Food Science (KBM). This dataset facilitates a fascinating opportunity to apply low-rank methods in biochemistry.

Forty-five experiment folders are provided, each containing 15 to 45 data files and two Excel files containing information about each data file. The data files are tab-delimited .bka (biokine analysis) files that can be parsed as .csv files, each characterized by different substrates, concentrations, and channels. Furthermore, each data file contains 15,001 data points across two unmarked columns for time in milliseconds(ms) and signal strength in volts(V) after the initial 18 lines that provide essential metadata. The initial 15-20 ms of data must be removed since they include mixing artifacts. One of the Excel files contains the actual starting time after artifacts for each .bka file. After that, the cleaned data should fit into a single exponential equation:

$$y = at + b + c + e^{-kt} \tag{3.3}$$

which comprises a linear portion ($at + b$), amplitude shift (c), and an exponential curve fit (e^{-kt}). These parameters a, b, and c are important for assessing

data quality, such as non-flat slopes indicating incomplete reactions and decreasing amplitude on increasing substrate indicating immeasurably fast reactions. If the rate constant (k) is determined for all the data files and plotted against the concentrations, then the relation should be linear[39]. The other Excel file contains the fitted slope, intercept, amplitude, and rate constant values for each .bka file.

Chapter 4

Methods

4.1 Technical Environment

This thesis was conducted on a device built on AMD Ryzen 7 4700U with Radeon Graphics and 8GB DDR4 RAM, where all the code implementations were done. The AMD Ryzen 7 4700 processor has eight cores and eight threads based on the AMD Renoir Zen 2 architecture. This chip is manufactured using a 7 nm process by TMSC with base and boost clocks of 2 GHz and 4.1 GHz, respectively. With the processor, Vega 7 graphics is integrated [40].

The Julia codes for the custom dense layer network training, vanilla training, and the DLRT algorithm were written in Julia version 1.10.4. 1.10.0 is the latest stable version, which gives access to the newest features and optimizations. Moreover, comprehensive package support is also provided, crucial for machine learning and numerical computations [41]. Similarly, the Flux package of version 0.14.15 was used.

4.2 Training a neural network model comprising custom dense layers

A custom layer is a user-defined layer designed by inheriting from the base layer class. Custom layers enable us to create flexible new layers with additional functionalities from the existing layers [42]. Instead of using the built-in dense layer, a

custom dense layer was defined and later used to build a model for neural network training.

In Julia, the custom dense layer was defined by using a structure [30], '**struct MyDense**'. This structure was designed to take the parameters '**in**', '**out**' and '**activation**' for the number of inputs, number of outputs, and the activation function to create an instance of a custom dense layer having parameters: **weights** matrix of size '**out** $\times$ **in**', **bias** vector of size '**out**' and the **activation function**. The variables W, b, and *activation* were defined in the structure to represent the **weights**, **bias**, and **activation function** of the custom dense layer, where only W and b were defined as trainable. Moreover, W and b were initialized with random values taken from a normal distribution. A method was defined as the forward function to handle the computations during the forward propagation that takes a matrix X as the **input**. A forward function is an integral part of a custom layer in which the input data is passed to obtain the output of the layer [42]. This method was defined to compute the product of weights matrix W with input matrix X, add the bias vector b, and apply the activation function to the result. Finally, another method was defined to handle the instances of the structure. This method was used to update the trainable parameters of the custom dense layer through an SGD update mechanism where the weights and biases are adjusted according to the gradients of the loss function. The SGD update mechanism can be explained with the equations given below. Here, λ represents the learning rate and ∇ represents the gradient.

$$W \leftarrow W - \lambda \cdot \nabla W \tag{4.1}$$

$$b \leftarrow b - \lambda \cdot \nabla b \tag{4.2}$$

This custom layer was used to construct a neural network, which was further trained and evaluated on the MNIST dataset. Before training, the dataset was pre-processed by normalizing and reshaping the features and encoding the targets. Then, momentum was added along with the SGD in the update mechanism to improve the accuracy, convergence speed, and stability. This improvement was achieved by adding new variables in the structure to represent the non-trainable momentum parameters and by creating the respective method to update momentum parameters and further adjust the trainable parameters. Also, the structure was redefined as a mutable structure to facilitate the addition of momentum in the update mechanism. Then, a neural network was constructed from the new dense layer to train and evaluate the MNIST dataset. Since the network was still underperforming, gradient clipping was added along with momentum and

SGD in the update mechanism to prevent possible exploding gradients. This final improvement was achieved by creating the respective method in which the gradients of the trainable parameters are clipped just before updating the momentum parameters. A helper function was also defined to facilitate the process of gradient clipping. At last, a neural network was constructed from this final improvement to train and evaluate the MNIST dataset.

The performance of the custom dense layer in Julia was compared with the performance of the custom dense layer in Python.

4.3 Implementing vanilla training

In vanilla training, the weight matrices are represented as the product of matrices U, Σ and $V^\top$ and then a descent algorithm is alternately applied on the variables U, Σ and V [25].

In Julia, the layer for vanilla training was defined by using a structure [30], '**struct VanillaLowRankLayer**'. This structure was designed to take the parameters '**in**', '**out**', '**rank**' and '**activation**' for the number of inputs, number of outputs, rank and the activation function to create an instance of a vanilla low-rank layer having parameters: **U** matrix of size '**out** $\times$ **rank**', **Σ** matrix of size '**rank** $\times$ **rank**', **V** matrix of size '**in** $\times$ **rank**', **bias** vector of size '**out**' and the **activation function**. The variables U, S, V, b, and *activation* were defined in the structure to represent the matrices **U**, **Σ**, and **V**, the **bias** and the **activation function** of the vanilla low-rank layer where only U, S, V, and b were defined to be trainable. Moreover, U, S, V, and b were initialized with random values taken from a normal distribution. Then, QR factorization was applied to the U and the V matrices to ensure that these matrices have orthonormal columns. According to the QR factorization theorem, a matrix of size $m \times n$ can be factored as the product of matrices Q and R where Q is an $m \times n$ matrix with orthonormal columns, and R is a $n \times n$ upper triangular matrix [16, Chapter 6]. The initial U and V matrices were replaced by the respective Q matrices obtained from the QR factorization. Similar to the custom dense layer, a method for forward function was defined, taking a matrix X as the input. However, this method was defined to compute the product of the matrices U, Σ, and $V^\top$ with input matrix X, add the bias vector b, and finally apply the activation function to the result. Finally, a method was defined to handle the instances of the structure. This method was used to update the trainable parameters of the vanilla low-rank layer through an SGD update mechanism where the parameters are adjusted according to the gradients of the

loss function. The SGD update mechanism can be explained with the equations given below. Here, λ represents the learning rate and ∇ represents the gradient.

$$U \leftarrow U - \lambda \cdot \nabla U \tag{4.3}$$

$$S \leftarrow S - \lambda \cdot \nabla S \tag{4.4}$$

$$V \leftarrow V - \lambda \cdot \nabla V \tag{4.5}$$

$$b \leftarrow b - \lambda \cdot \nabla b \tag{4.6}$$

This vanilla training and custom layers were used to construct a neural network. The vanilla low-rank layer was used as the input layer, and the hidden layer/s, while the custom dense layer was used as the output layer in the neural network. This neural network was further trained and evaluated on the MNIST dataset. Before training, the dataset was pre-processed in the same manner as in section 4.2. Then, momentum was added along with the SGD in the update mechanism. Gradient clipping was added along with momentum and SGD to the update mechanism. This improvement was achieved by adding new variables in the structure to represent the non-trainable momentum parameters and by creating the respective method. In that method, the gradients of the loss function with respect to the trainable parameters are clipped. Then, the momentum parameters are updated to adjust the trainable parameters further. A helper function was also defined to facilitate the process of gradient clipping. Also, the structure was redefined as a mutable structure to facilitate the addition of momentum and gradient clipping in the update mechanism. Then, a neural network was constructed from the new vanilla low-rank layer and the custom dense layer to train and evaluate the MNIST dataset.

The neural network comprising vanilla low-rank layers was compared with the neural network from section 4.2 regarding performance. In addition, the performance of the vanilla low-rank layer in Julia was compared with the performance of the vanilla low-rank layer in Python.

4.4 Implementing the DLRT algorithm as the fixed-rank approach

The KLS-based algorithm from section 3.2 was implemented as a fixed-rank approach. In Julia, the layer for this dynamical low-rank training was defined by

using a parametric structure [30], '**struct DynamicLowRankLayer**'. This structure was designed to take the parameters '**in**', '**out**', '**rank**' and '**activation**' for the number of inputs, number of outputs, rank and the activation function to create an instance of a dynamical low-rank layer having parameters: **U** and **U1** matrices of size '**out** $\times$ **rank**', $\mathbf{\Sigma}$ matrix of size '**rank** $\times$ **rank**', **V** and **V1** matrices of size '**in** $\times$ **rank**', **bias** vector of size '**out**' and the **activation function**. Similar to the '**struct VanillaLowRankLayer**' from section 4.3, the variables U, S, V, b, and *activation* were defined in the structure. Besides, variables $U1$ and $V1$ were defined to represent the matrices **U** and **V** after one training step, per the algorithm from section 3.2. However, only U, S, V, and b were defined to be trainable. Moreover, U, S, V, b, $U1$, and $V1$ were initialized with random values taken from a normal distribution. Then, a QR factorization was applied to the U and the V matrices to ensure that these matrices have orthonormal columns. The initial U and V matrices were replaced by the respective Q matrices obtained from the QR factorization in the same way as in section 4.3.

Similar to the custom dense layer and the vanilla low-rank layer, a method for forward function was defined, which takes a matrix X as the input. This method was defined to compute the product of the matrices U, Σ, and $V^\top$ with input matrix X, add the bias vector b, and apply the activation function to the result. Finally, a method was defined to handle the instances of the structure. This method was used to update the trainable parameters of the dynamical low-rank layer; however, unlike the custom dense layer and vanilla low-rank layer, the mechanism for SGD update requires two individual steps according to the algorithm from section 3.2. This method was designed to handle two steps, 'basis-update' for updating the parameters U, V, and b and 'coefficients-update' for updating the parameter S.

The **K** and **L-steps** from the KLS-based algorithm in section 3.2 were adapted in the 'basis-update'. An intermediate variable K was computed as the product of U and S. Then, K was updated by subtracting the learning rate scaled gradient of the loss function for U multiplied by S. Further, $U1$ with orthonormal columns was obtained from QR factorization of K. Likewise, another intermediate variable, L, was computed as the product of V and $S^\top$. Then, L was updated by subtracting the learning rate scaled gradient of the loss function for V multiplied by $S^\top$. Further, $V1$ with orthonormal columns was obtained from the QR factorization of L. These parameters $U1$ and $V1$, along with parameters U and V, were used to update parameter S through matrix multiplications. Then, these parameters $U1$ and $V1$ were used to update the parameters U and V. Finally, the *bias* parameter was updated using the learning rate and the respective gradient of the loss function. On the other hand, the **S-step** from the KLS-based algorithm in section 3.2 was adapted in the 'coefficients-update'. The S parameter was updated by using the

learning rate and the respective gradient of the loss function.

This dynamical low-rank layer and a custom layer were used to construct a neural network. The dynamical low-rank layer was used as the input layer, and the hidden layer/s while the custom dense layer was used as the output layer in the neural network. This neural network was further trained and evaluated on the MNIST dataset. Before training, the dataset was pre-processed in the same manner as in sections 4.2 and 4.3. However, the forward pass and the backpropagation were defined twice in the training process. The gradients from the initial forward pass and the backpropagation were used to call the update method, selecting the step 'basis-update.' Then, the gradients were recomputed in the second forward pass and the backpropagation. These gradients were used to call the update method, selecting the step 'coefficients-update'.

Then, like section 4.3, momentum and gradient clipping were added along with SGD in the update mechanism. Non-trainable momentum parameters were added, and the respective method was created. In that method, the gradients of the loss function for the trainable parameters U, V, and b are clipped. Then, the corresponding momentum parameters are updated to adjust the trainable parameters for the step 'basis-update'. Also, the gradient of the loss function for the trainable parameter S is clipped. Then, the corresponding momentum parameter is updated to adjust the trainable parameter for the step 'coefficients-update'. Likewise, a gradient clipping helper function was defined, and the structure was redefined as the mutable structure. Then, a neural network was constructed from the new dynamical low-rank layer and the custom dense layer to train and evaluate the MNIST dataset.

The neural network based on dynamical low-rank layers was compared with the neural networks from sections 4.2 and 4.3 regarding performance. In addition, the performance of the dynamical low-rank layer in Julia was compared with the performance of the dynamical low-rank layer in Python.

4.5 Implementing the DLRT algorithm as the rank-adaptive approach

The KLS-based algorithm from section 3.2 was implemented as a rank-adaptive approach. In other words, the dynamical low-rank layer from section 4.4 was upgraded to rank-adaptivity. In Julia, the layer for this rank-adaptive dynamical

low-rank training was defined by using a structure [30], '**struct RADynamicLowRankLayer**'. This structure was designed to take the parameters '**in**', '**out**', '**rank**', '**tol**' and '**activation**' for the number of inputs, number of outputs, rank, tolerance, and the activation function to create an instance of a rank-adaptive dynamical low-rank layer having parameters: **U** and **U1** matrices of size '**out** $\times$ **2** $\cdot$ **rank**', $\mathbf{\Sigma}$ matrix of size '**2** $\cdot$ **rank** $\times$ **2** $\cdot$ **rank**', **V** and **V1** matrices of size '**in** $\times$ **2** $\cdot$ **rank**', **bias** vector of size '**out**', **tolerance** and the **activation function**. Similar to the '**struct DynamicLowRankLayer**' from section 4.4, the variables U, S, V, $U1$, $V1$, b and *activation* were defined in the structure. Besides, variables $rMax$, $rCurrent$, and tol were defined to represent the maximum rank, the current rank, and the truncation tolerance. Initial values were stored in the layer parameters U, S, V, b, $U1$, and $V1$ in the same way as in section 4.4. Also, only U, S, V, and b were defined to be trainable.

Similar to the other three layers, a method for forward function was defined, which takes a matrix X as the input. This function was also defined to compute the product of the matrices U, Σ, and $V^\top$ with input matrix X, add the bias vector b, and apply the activation function to the result. Additionally, the current rank of the layer $rCurrent$ was considered for the matrices U, S, and V, making the forward pass a rank-adaptive process. Similar to the dynamical low-rank layer from section 4.4, a method was defined to update the trainable parameters of the layer with two steps: 'basis-update' and 'coefficients-update.' Before proceeding with either of these two steps, the current rank of the layer was stored as variable r because the rank evolved throughout the training.

At the beginning of the 'basis-update' step, the first r columns were selected from the basis matrices U and V and the coefficients matrix S. In contrast, all the rows were selected from the basis matrices U and V, and the first r rows were selected from the coefficients matrix S. This way, the evolving rank was adjusted in the matrices U, S, and V, and they were stored as $U0$, $S0$, and $V0$. An intermediate variable K was computed as the product of $U0$ and $S0$. Then, K was updated by subtracting the learning rate scaled gradient of the loss function for $U0$ multiplied by $S0$. Then, K was extended by concatenating $U0$ with it. Further, $U1$ with orthonormal columns was obtained from QR factorization of the extended K. Likewise, another intermediate variable, L, was computed as the product of $V0$ and $S0^\top$. Then, L was updated by subtracting the learning rate scaled gradient of the loss function for $V0$ multiplied by $S0^\top$. Then, L was extended by concatenating $V0$ with it. Further, $V1$ with orthonormal columns was obtained from QR factorization of the extended L. These parameters $U1$ and $V1$, along with $U0$ and $V0$, were used to update parameter S through matrix multiplications involving $S0$. Then, these parameters $U1$ and $V1$ were used to

update the parameters U and V. Finally, the *bias* parameter was updated using the learning rate and the respective gradient of the loss function. Additionally, the value of the current rank was doubled.

Like the dynamical low-rank layer from section 4.4, in the step 'coefficients-update', the S parameter was updated using the learning rate and the respective gradient of the loss function. The size of the parameter S was adapted to the current rank. In addition, the new rank was determined and further used to truncate the parameters U, S, and V with the help of a helper function. Inside the definition of the helper function, the current S matrix was truncated up to the current rank, and SVD was performed on it. The Σ matrix obtained from this SVD was used to compute the new rank based on the truncation tolerance.

This rank-adaptive dynamical low-rank layer and a custom layer were used to construct a neural network. The dynamic rank-adaptive low-rank layer was used as the input layer, and the hidden layer/s while the custom dense layer was used as the output layer in the neural network. This neural network was further trained and evaluated on the MNIST dataset for a large number of epochs. Before training, the dataset was pre-processed in the same manner as in sections 4.2, 4.3, and 4.4. The training was carried out the same way as in section 4.4. Then, like section 4.4, momentum and gradient clipping were added along with SGD in the update mechanism. Non-trainable momentum parameters were added, and the respective method was created. Then, a neural network was constructed from the new rank-adaptive dynamic training layer and the custom dense layer to train and evaluate the MNIST dataset for a large number of epochs.

The neural network based on rank-adaptive dynamical low-rank layers was compared with the neural networks from sections 4.2, 4.3, and 4.4 regarding performance. In addition, the performance of the rank-adaptive dynamical low-rank layer in Julia was compared with that of the rank-adaptive dynamical low-rank layer in Python.

4.6 Application in biochemistry data

4.6.1 Data Preparation

There were two Excel files in the experiment folders of the stopped-flow data, one for the actual start times and the other for the fitted parameter values. Both Excel

files were modified to facilitate reading. The file for the actual start times was a Microsoft Excel Comma Separated Values (CSV) file, but the file for the fitted parameter values was a Microsoft Excel Worksheet (XLSX) file. The CSV file was converted into a Microsoft Excel Worksheet (XLSX) file with multiple columns to use the same function to read both files. This conversion was carried out with the help of the '**Text to Columns**' wizard. Since both contain the names of the .bka files within the experiment folder, the index column was removed from each. Forty-two of the forty-five experiment folders were kept in the 'train' directory, and three were kept in the 'test' directory.

4.6.2 Data Pre-processing

For each experiment folder, dictionaries were created from both Excel files. These dictionaries were created so that the start times and the labels for each .bka file could be accessed easily. The columns 'Filename' from the Excel file were read in those dictionaries as the keys. From the Excel file containing the actual start times, the column 'X_start' was read as the values. The column 'Rate' was read as the values from the other Excel file.

While reading the .bka files, the initial 18 lines were removed as they were metadata. The 18^{th} line was checked for containing the string '_DATA' to confirm that the time and voltage pairs have started just after the metadata. Then, the file was read by line while splitting each line into time and voltage values. The starting time value was used to filter out the lines containing time values smaller than the actual starting time. Then, the data was interleaved into a single array of time-voltage pairs. Finally, padding with zeros or truncating ensured a fixed array length for all the .bka files. Thus, a feature vector was created for the .bka files, and the corresponding 'Rate' value from the dictionary was used as the label for the feature vector. Inside an experiment folder, all the feature vectors extracted from the .bka files were concatenated horizontally, forming a feature matrix, while the corresponding labels were stored in a list. For both the 'Train' and the 'Test' directories, the feature matrices obtained from the experiment folders are concatenated horizontally, and the corresponding lists of labels are concatenated accordingly.

The label arrays of both train and test sets were reshaped to have the correct dimensions for training and converted into 2-dimensional arrays with one row. Data loaders were created for both train and test sets. To handle batching and efficient data loading, the 'DataLoader' class was used. The data loader for the

train set also shuffled the data.

4.6.3 Training and Evaluation

After completing the dataset's pre-processing in Julia, different models were created to train and evaluate it. First, a model comprising the built-in dense layers was created. Then, the model was trained for the mean squared error loss function and the ADAM optimizer with varying learning rates. The training was done for 2000 epochs. For every epoch, the train set loss, and test set loss were printed using the mean absolute error loss function. After the last epoch, the actual test and predicted labels were also printed to observe how close the predicted values were to the actual values. Then, another model was created from the custom dense layer defined in section 4.2. All the procedures adopted for the previous model were repeated. Furthermore, a model was created from the vanilla low-rank layer defined in section 4, and all the procedures were repeated. Finally, a model was created from the dynamical low-rank layer defined in section 4.4. All the methods adopted from the earlier models were repeated here as well. The outputs of all these models were compared.

In Python, the dataset's pre-processing was done the same as in Julia. Then, a model comprising the built-in dense layers was created. The model was trained for the mean squared error loss function and the ADAM optimizer with varying learning rates. All the remaining procedures were repeated. Then, the output was compared with the outputs previously obtained from the Julia models.

4.7 Summary

The methodology was based on five major tasks that were carried out sequentially. They were custom dense layer training, implementing vanilla training, implementing the DLRT algorithm first as a fixed-rank approach and later as a rank-adaptive approach, and finally, its application to the biochemistry data. The outcomes from the first four tasks were compared to assess the advantages of the low-rank methods. Also, the outcomes in Julia were compared with the Python implementation to observe the computational efficiency of Julia. As an application, the fixed-rank DLRT algorithm was used to predict the value of rates for stopped-flow data. The performance of the fixed-rank DLRT algorithm in Julia was performed with the performance of the built-in dense layer and custom dense layer in Julia and

the built-in dense layer in Python. The methodology can be summarized in the following block diagram:

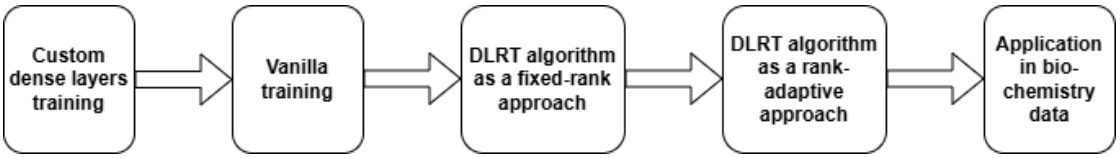


Figure 4.1: Methodology block diagram.

Chapter 5

Results

This chapter presents results obtained after training different neural network models on the MNIST and biochemistry datasets. Each plot has a brief explanation, followed by a detailed explanation in the next chapter. The first four sections revolve around the low-rank and full-rank neural networks in Julia. The fifth section provides the foundations for comparing similar implementations in Python. Finally, the sixth section applies the low-rank neural networks to a biochemistry dataset.

In the first five sections, each figure provided for a neural network model displays two plots. The first plot shows the training and test losses across epochs, and the second plot shows the training and test accuracies. To work on the MNIST dataset, the primary architecture tested was (784, 256), (256, 128), and (128, 10). Additionally, other architectures such as (784, 512), (512, 256), and (256, 10) and (784, 128), (128, 64), and (64, 10) were experimented. In both of the programming languages and for all types of layers, (784, 256), (256, 128), and (128, 10) provided the best performance while maintaining low memory consumption. So, this architecture is selected to work on the MNIST dataset. Further, two values of batch sizes were tested while working on the MNIST dataset, 32 and 64. The batch size of 64 performed better for the majority of the neural network models. So, for consistency, the batch size of 64 is selected to work on the MNIST dataset.

For all the neural networks in Julia based on the layers with momentum and gradient clipping, a momentum coefficient of 0.9 and a gradient clipping threshold of 4 are selected. Other momentum coefficients such as 0.6, 0.7, and 0.8 were also tested. Similarly, gradient clipping threshold values such as 1, 7, and 10 were also tested. For, the majority of the neural network models, the momentum coefficient

of 0.9 and the gradient clipping threshold of 4 gave better performance.

The reasons behind selecting the values of the learning rate are discussed in the chapter 6.

5.1 Performance of custom dense layer on the MNIST dataset

The results are based on the performance of two neural network models consisting of three custom dense layers. In both models, the input layer has 784 input and 256 output nodes with 'relu' activation. Similarly, the hidden layer has 256 input and 128 output nodes with 'relu' activation. Finally, the output layer has 128 input and ten output nodes with 'identity' activation.

The first model consists of the custom dense layers with a regular SGD update in the training algorithm. The model is trained with a learning rate of 0.001. The time taken for training and evaluation is 938.19 seconds for 200 epochs.

Figure 5.1 for the first model shows that the training and the test losses decrease across the epochs and tend to approach 0. The training and test accuracy trend is increasing across the epochs. However, the accuracies are evolving with fluctuations. After 200 epochs, the training and the test accuracies are approximately 85.5 percent and 85 percent, respectively.

The second model consists of the custom dense layers with additional momentum and gradient clipping in the training algorithm. First, the model is trained with a learning rate of 0.001. The corresponding time taken for training and evaluation is 1535 seconds for 200 epochs. Then, the model is trained with a learning rate of 0.01. The corresponding time taken for training and evaluation is 406.7 seconds for 50 epochs.

Figure 5.2 for the second model with a learning rate of 0.001 shows that the accuracies evolve more smoothly compared to the accuracies in Figure 5.1 for the first model. After 200 epochs, the training and test losses are approximately 5 and 8.5, respectively. Similarly, the training and test accuracies are approximately 92.9 and 91.4 percent, respectively.

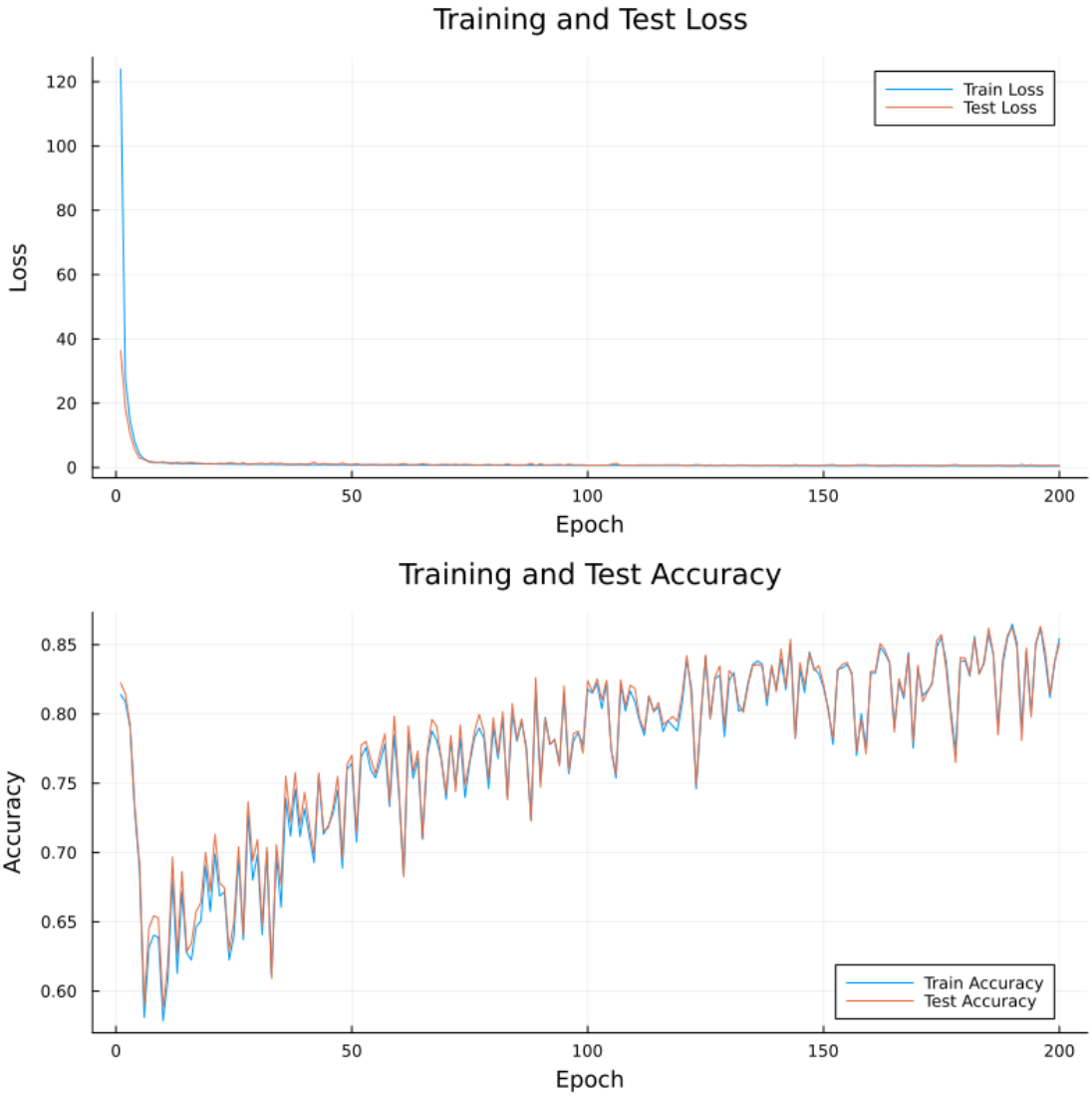


Figure 5.1: Loss and accuracy plots for a model built on custom dense layers with regular update mechanism.

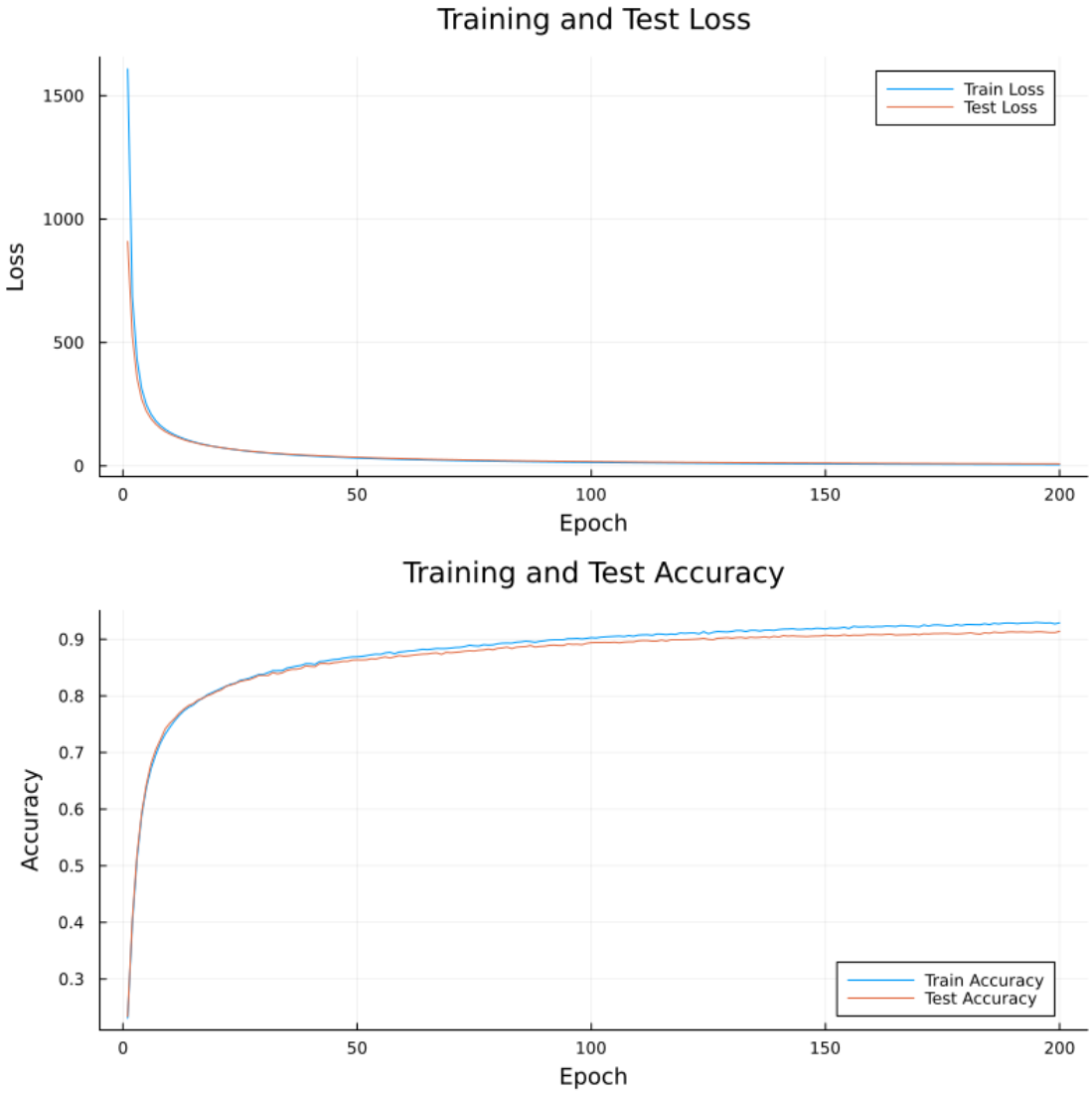


Figure 5.2: Loss and accuracy plots for a model built on custom dense layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.001.

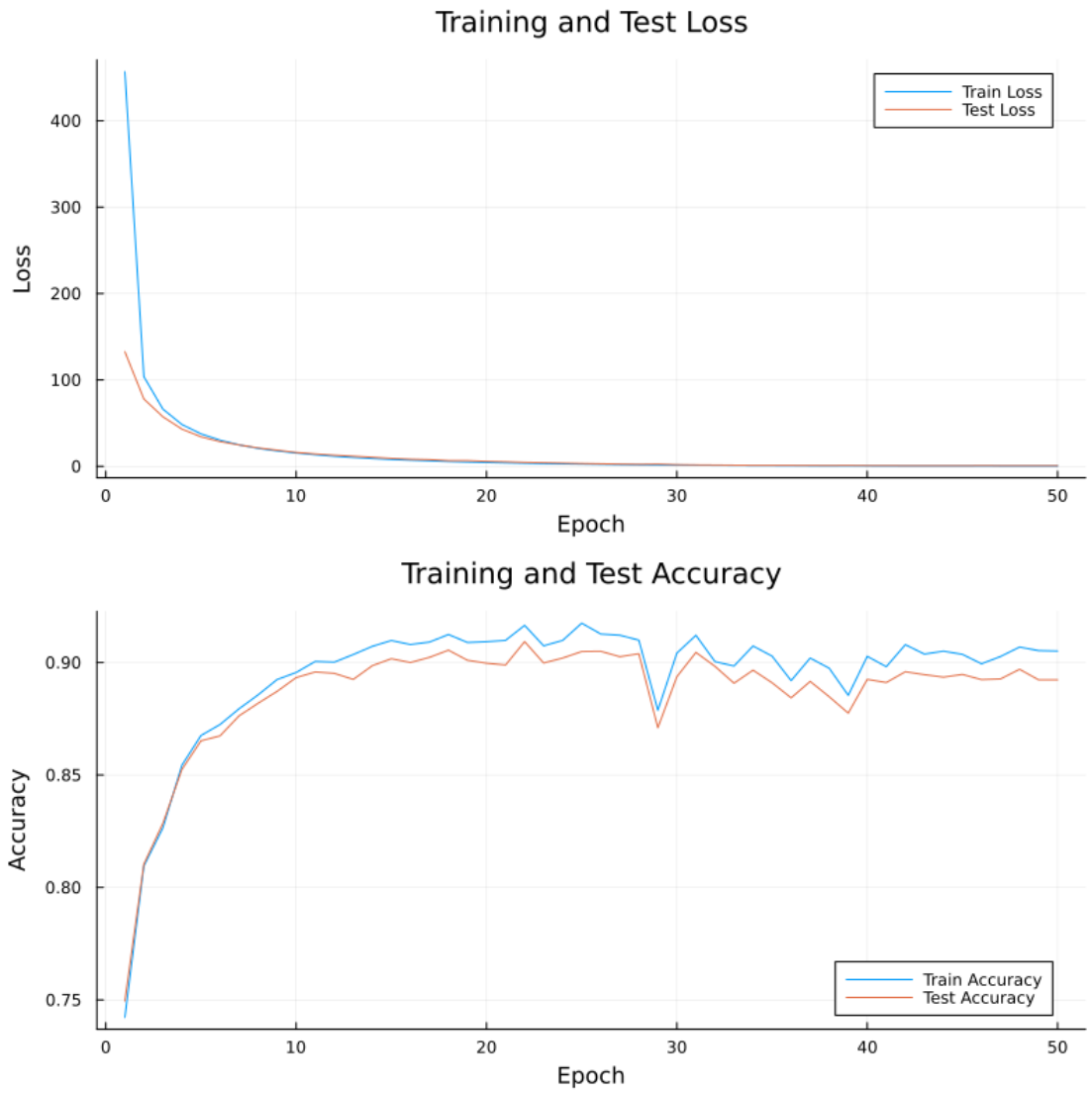


Figure 5.3: Loss and accuracy plots for a model built on custom dense layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.01.

The accuracies in Figure 5.3 for the second model with a learning rate of 0.01 fluctuate more than those in Figure 5.2 for the same model with a learning rate of 0.001, with a significant dip at the 29^{th} epoch. But these fluctuations are still less than those in the evolution of accuracies in Figure 5.1 for the first model. The accuracies do not improve significantly after 20 epochs. Even after 50 epochs, the training and test accuracies are approximately 90 and 89 percent, respectively.

Similarly, the training and test losses are approximately 0.4 and 0.6, respectively.

These results are summarized in table 5.1.

Neural Network based on	**Learning Rate**	**Train and Test Accuracies (%)**	**Epochs**	**Runtime per epoch (s)**	**Total Time Taken (s)**
Custom dense layers	0.001	85.5 and 85	200	4.69	938.19
Custom dense layers with momentum and gradient clipping	0.001	92.9 and 91.4	200	7.68	1535
Custom dense layers with momentum and gradient clipping	0.01	90 and 89	50	8.13	406.7

Table 5.1: Performance of custom dense layers on the MNIST dataset.

5.2 Vanilla low-rank layer on MNIST dataset

The following results are based on the performance of two neural network models consisting of two vanilla low-rank layers and one custom dense layer. All three layers have the same parameter values for input and output nodes, as well as activation, like the models from section 5.1. Both the input and the hidden layers have a rank of 20. Other values of rank such as 30 and 10 were also tested but, the best performance was given by rank 20 for both the vanilla low-rank and the dynamical low-rank layers.

The first model consists of vanilla low-rank layers and a custom dense layer with a regular SGD update in the training algorithm. It is trained with a learning rate of 0.001. The time taken for training and evaluation is 378.16 seconds for 100 epochs.

The second model consists of vanilla low-rank layers and a custom dense layer with additional momentum and gradient clipping in the training algorithm. When the model is trained with a learning rate of 0.001, the time taken for training and

evaluation is 834.63 seconds for 200 epochs. Likewise, when the model is trained with a learning rate of 0.01, the time taken for training and evaluation is 418.58 seconds for 100 epochs.

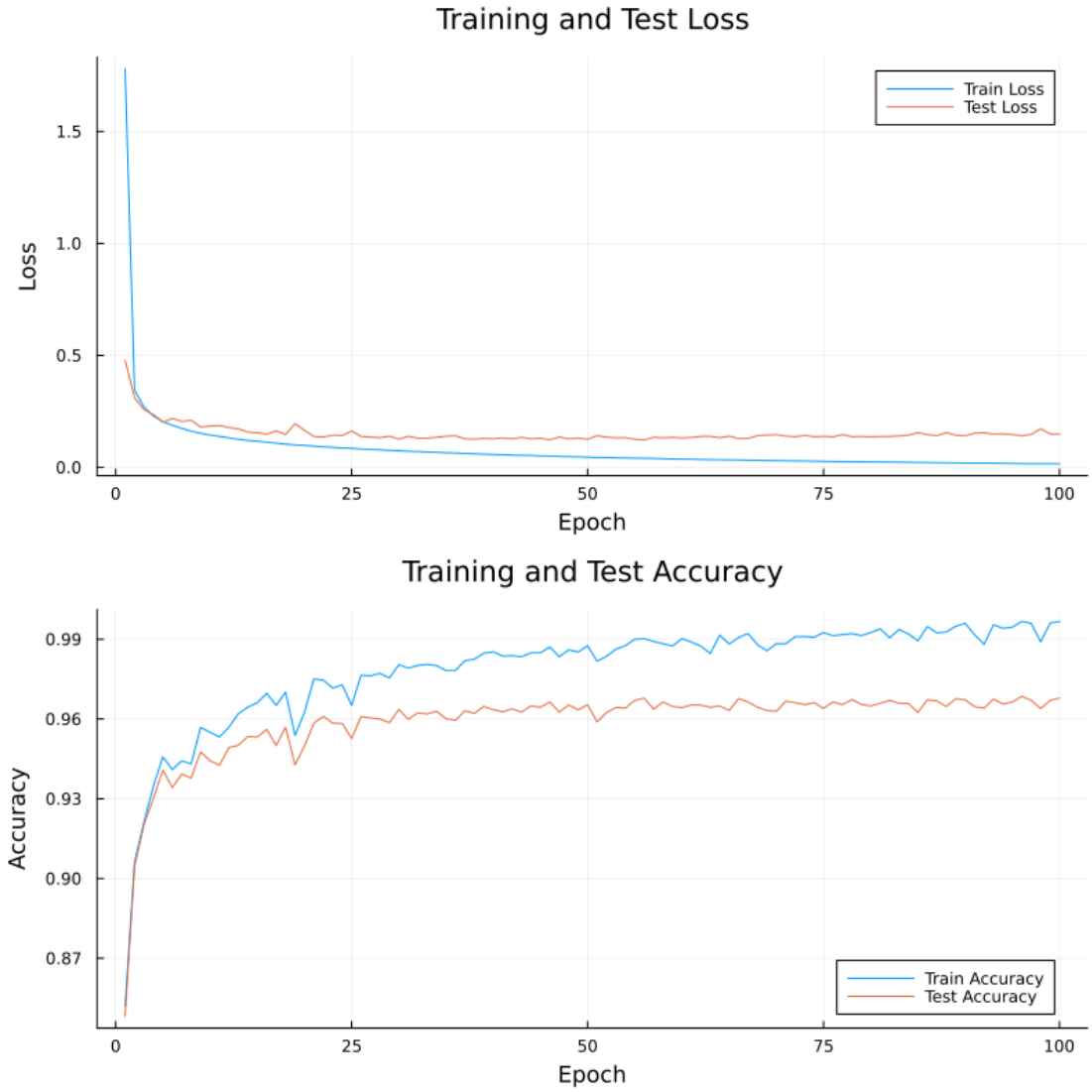


Figure 5.4: Loss and accuracy plots for a model built on vanilla low-rank layers with regular update mechanism.

Figure 5.4 for the first model shows that the evolution of the test loss exhibits some fluctuations, whereas the training loss remains smooth. After 100 epochs, the training and the test losses are approximately 0.02 and 0.15, respectively. While both the training and the test accuracies are evolving with fluctuations,

the trend is increasing. After 100 epochs, the training and the test accuracies are approximately 99.7 and 96.8 percent, respectively.

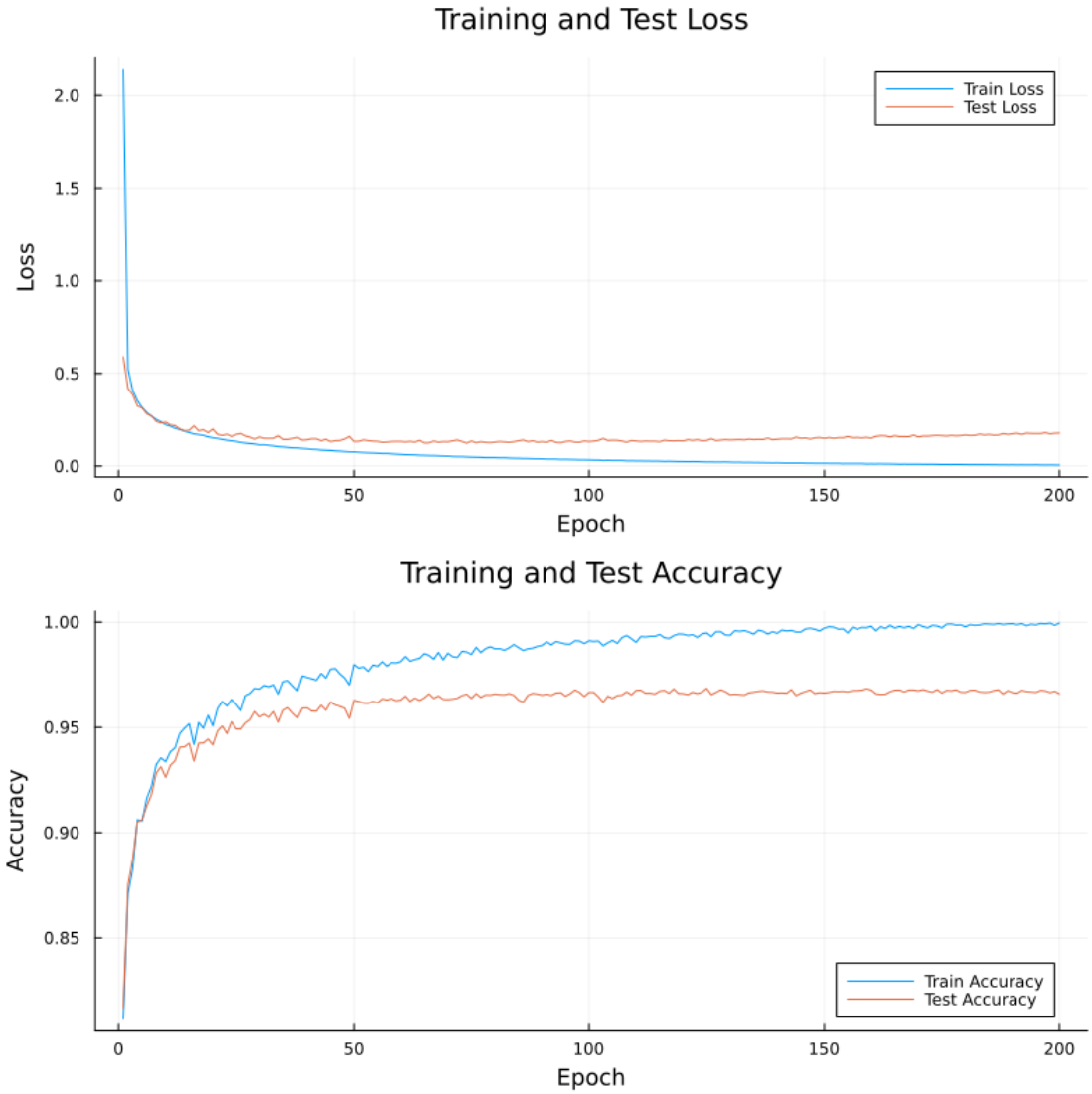


Figure 5.5: Loss and accuracy plots for a model built on vanilla low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.001.

.

Figure 5.5 for the second model with a learning rate of 0.001 exhibits that the evolution of the losses and accuracies is smoother than that in Figure 5.4 for the first model. Additionally, the gap between the plots is smaller. After 200 epochs,

the training and the test losses are approximately 0.01 and 0.17, respectively. Similarly, the training and the test accuracies are approximately 99.95 and 96.60 percent, respectively.

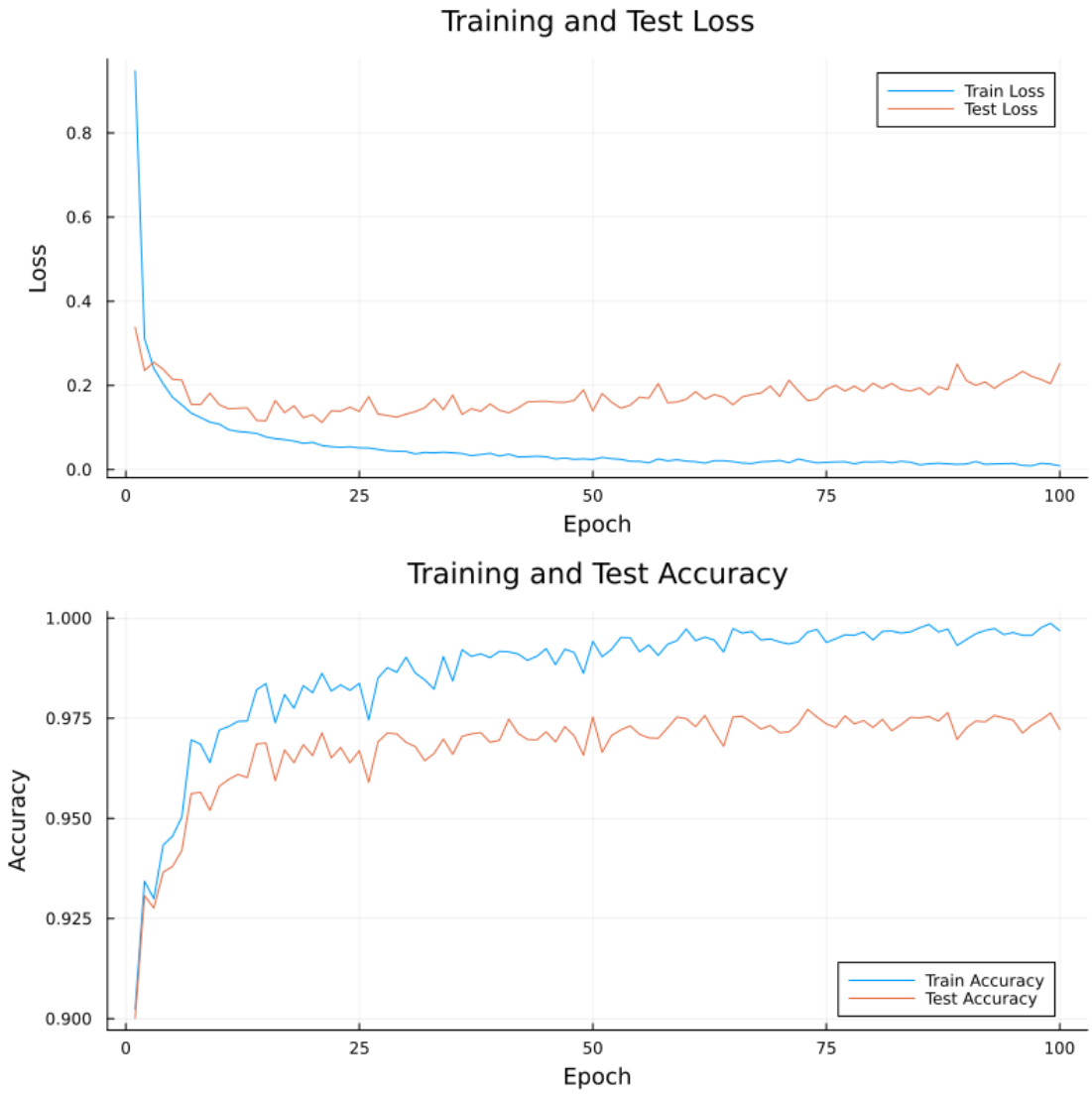


Figure 5.6: Loss and accuracy plots for a model built on vanilla low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.01.

As shown in Figure 5.6 for the second model with a learning rate of 0.01, the accuracies and losses fluctuate more than those in Figure 5.5 for the same model with a learning rate of 0.001. The gaps between the plots are more significant.

After 100 epochs, the training and the test losses are approximately 0.01 and 0.25, respectively. Similarly, the training and the test accuracies are approximately 99.7 and 97.2 percent, respectively.

These results are summarized in table 5.2.

Neural Network based on	**Learning Rate**	**Rank**	**Train and Test Accuracies (%)**	**Epochs**	**Runtime per epoch (s)**	**Total Time Taken (s)**
Vanilla low-rank layers	0.001	20	99.7 and 96.8	100	3.78	378.16
Vanilla low-rank layers with momentum and gradient clipping	0.001	20	99.95 and 96.60	200	4.17	834.63
Vanilla low-rank layers with momentum and gradient clipping	0.01	20	99.7 and 97.2	100	4.19	418.58

Table 5.2: Performance of the vanilla low-rank layers on the MNIST dataset.

5.3 Dynamical low-rank layers on the MNIST dataset

The results are based on the performance of two neural network models consisting of two dynamical low-rank layers and one custom dense layer. All three layers have the same parameter values for input and output nodes, as well as activation, like the models from sections 5.1 and 5.2. Both the input and the hidden layers have a rank of 20.

The first model consists of dynamical low-rank layers and a custom dense layer with a regular SGD update in the training algorithm. It is trained with a learning rate of 0.001. The time taken for training and evaluation is 657.66 seconds for 100 epochs.

The second model consists of dynamical low-rank layers and a custom dense layer with additional momentum and gradient clipping in the training algorithm. When the model is trained with a learning rate of 0.001, the time taken for training and evaluation is 1401.45 seconds for 200 epochs. Likewise, when the model is trained with a learning rate of 0.01, the time taken for training and evaluation is 553.6 seconds for 100 epochs.

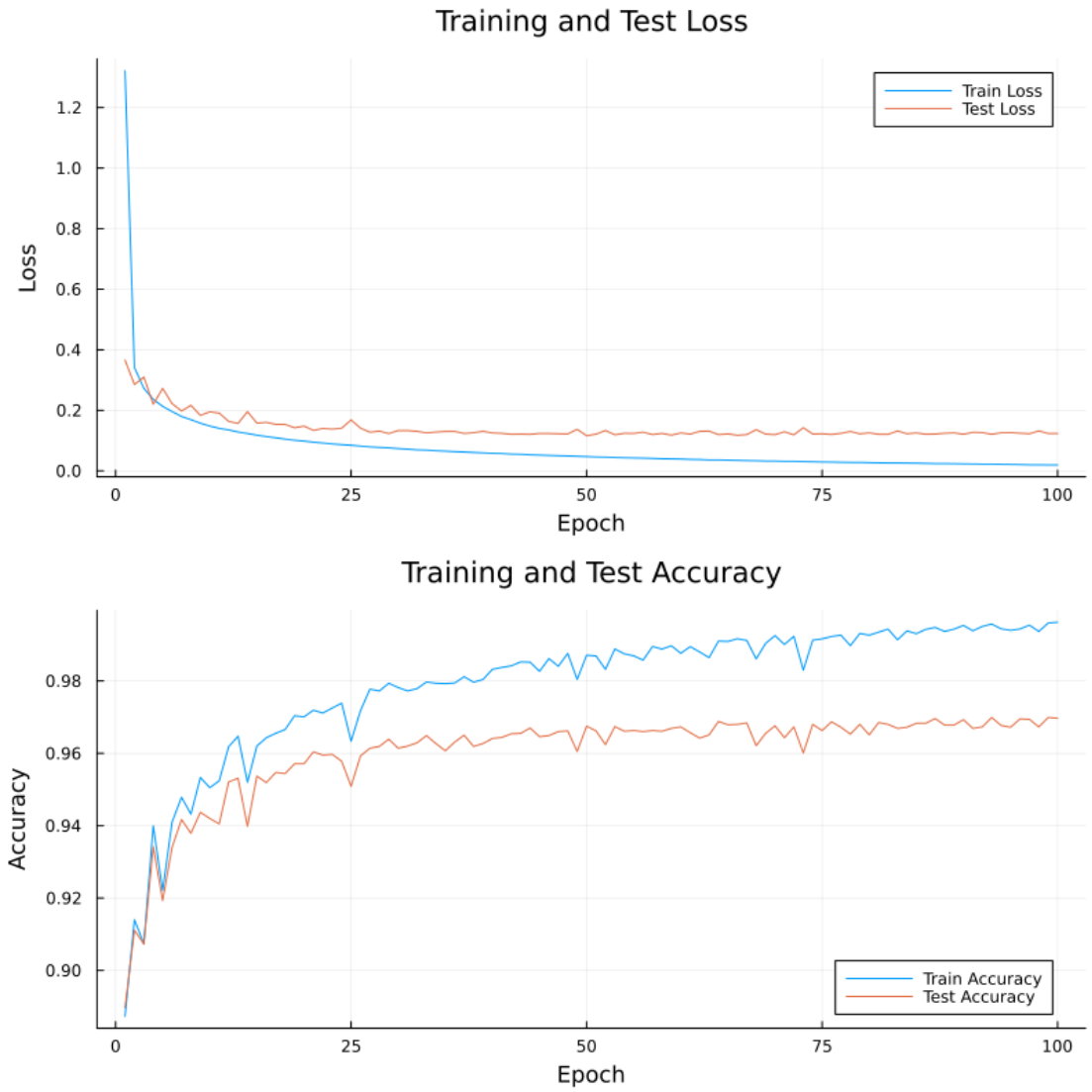


Figure 5.7: Loss and accuracy plots for a model built on dynamical low-rank layers with regular update mechanism.

Figure 5.7 for the first model shows that the test loss evolves with some fluctuations

while the training loss evolves smoothly. After 100 epochs, the training and the test losses are approximately 0.02 and 0.12, respectively. The evolution of both the training and the test accuracies shows fluctuations. After 100 epochs, the training and the test accuracies are approximately 99.6 and 97.0 percent, respectively.

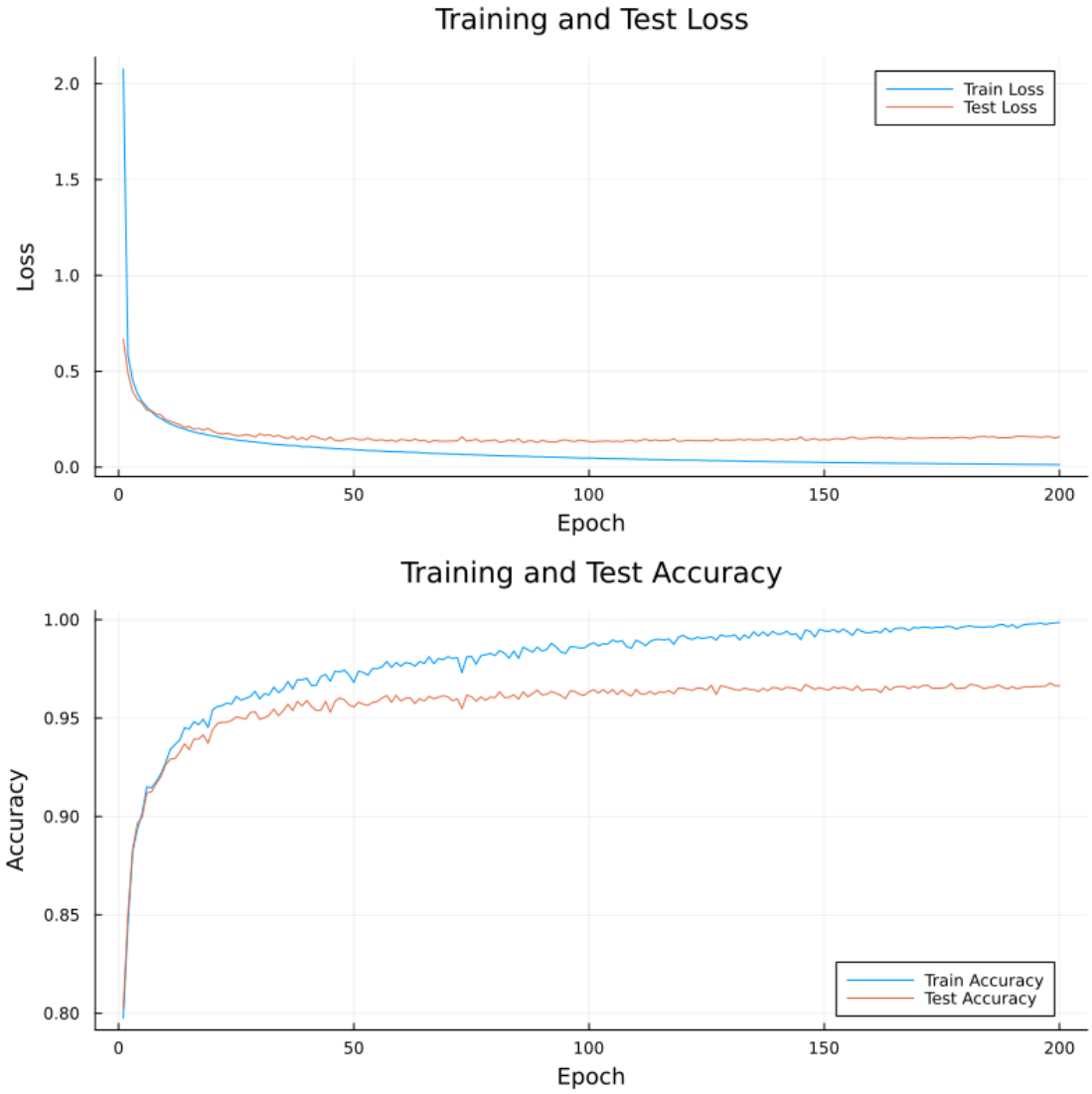


Figure 5.8: Loss and accuracy plots for a model built on dynamical low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.001.

Figure 5.8 for the second model with a learning rate of 0.001 exhibits that the evolution of the losses and accuracies are smoother than that in Figure 5.7 for the

first model. Additionally, the gap between the plots is smaller. After 200 epochs, the training and the test losses are approximately 0.01 and 0.16, respectively. Similarly, the training and the test accuracies are approximately 99.86 and 96.63 percent, respectively.

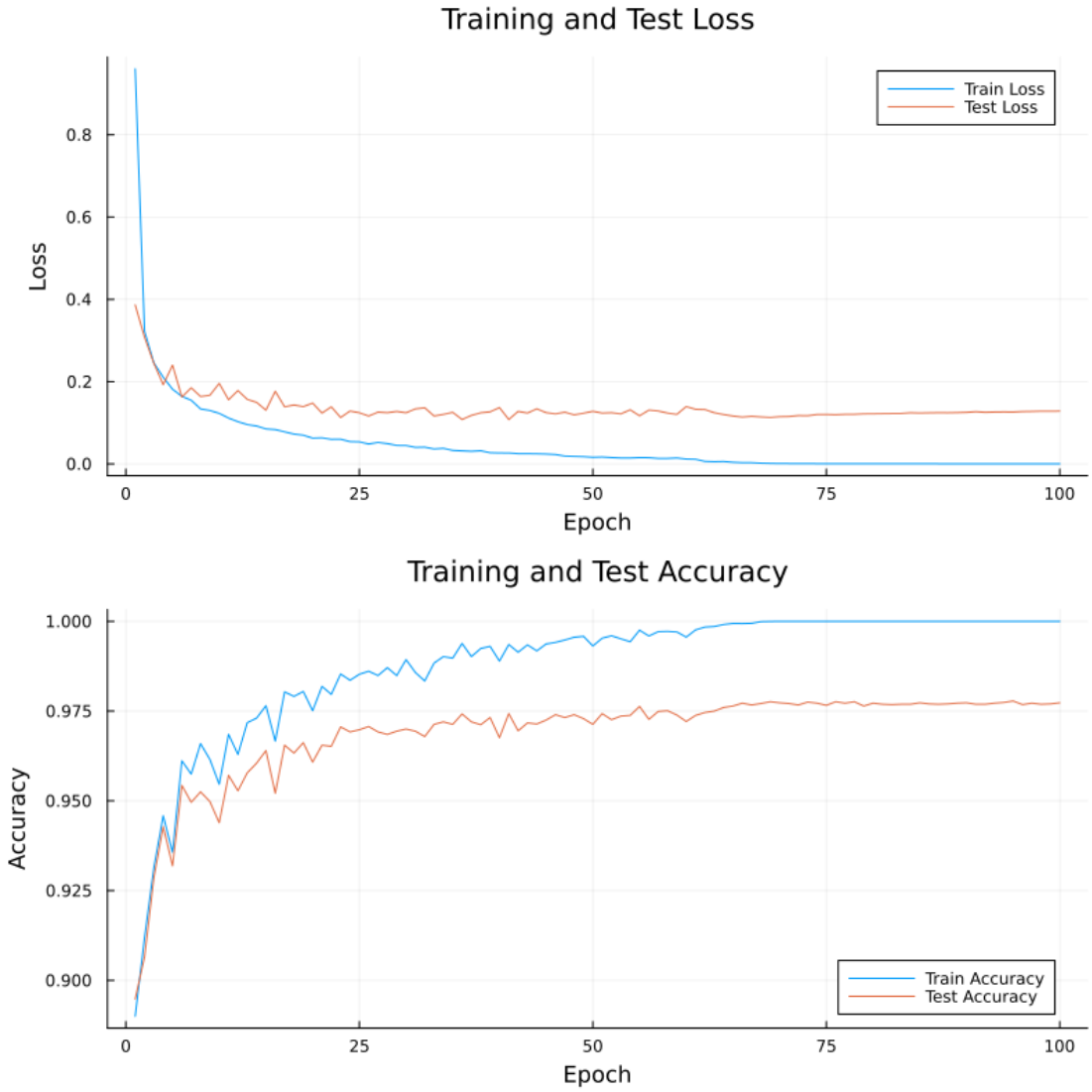


Figure 5.9: Loss and accuracy plots for a model built on dynamical low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.01.

Figure 5.9 for the second model with a learning rate of 0.01 shows that the evolution of the losses and accuracies shows more fluctuations than in Figure 5.8 for the

same model with a learning rate of 0.001 until around 63 epochs. After that, the evolutions of the losses and accuracies are almost straight lines. The gaps between the plots are more significant. After 100 epochs, the training and the test losses are approximately 0 and 0.13, respectively. Similarly, the training and the test accuracies are approximately 100 and 97.7 percent, respectively.

These results are summarized in table 5.3.

Neural Network based on	**Learning Rate**	**Rank**	**Train and Test Accuracies (%)**	**Epochs**	**Runtime per epoch (s)**	**Total Time Taken (s)**
Dynamical low-rank layers	0.001	20	99.6 and 97.0	100	6.58	657.66
Dynamical low-rank layers with momentum and gradient clipping	0.001	20	99.86 and 96.63	200	7.01	1401.45
Dynamical low-rank layers with momentum and gradient clipping	0.01	20	100 and 97.7	100	5.54	553.60

Table 5.3: Performance of dynamical low-rank layers on the MNIST dataset.

5.4 Rank-adaptive dynamical low-rank layer on the MNIST dataset

The results are based on the performance of two neural network models consisting of two rank-adaptive dynamical low-rank layers and one custom dense layer. All three layers have the same parameter values for input and output nodes, as well as activation, like the models from sections 5.1, 5.2, and 5.3. The input and the hidden layers have a maximum rank of 40 and a truncation tolerance of 5×10^{-3}. Other values of maximum rank such as 30 and 50 were also tested but, the best performance was given by a maximum rank of 40. Additionally, the truncation

tolerance of 5×10^{-2} and 5×10^{-4} were also tested. But, the best performance was given by a truncation tolerance of 5×10^{-3}.

The first model consists of layers with a regular SGD update in the training algorithm. It is trained with a learning rate of 0.001. The time taken for training and evaluation is 1471.38 seconds for 100 epochs.

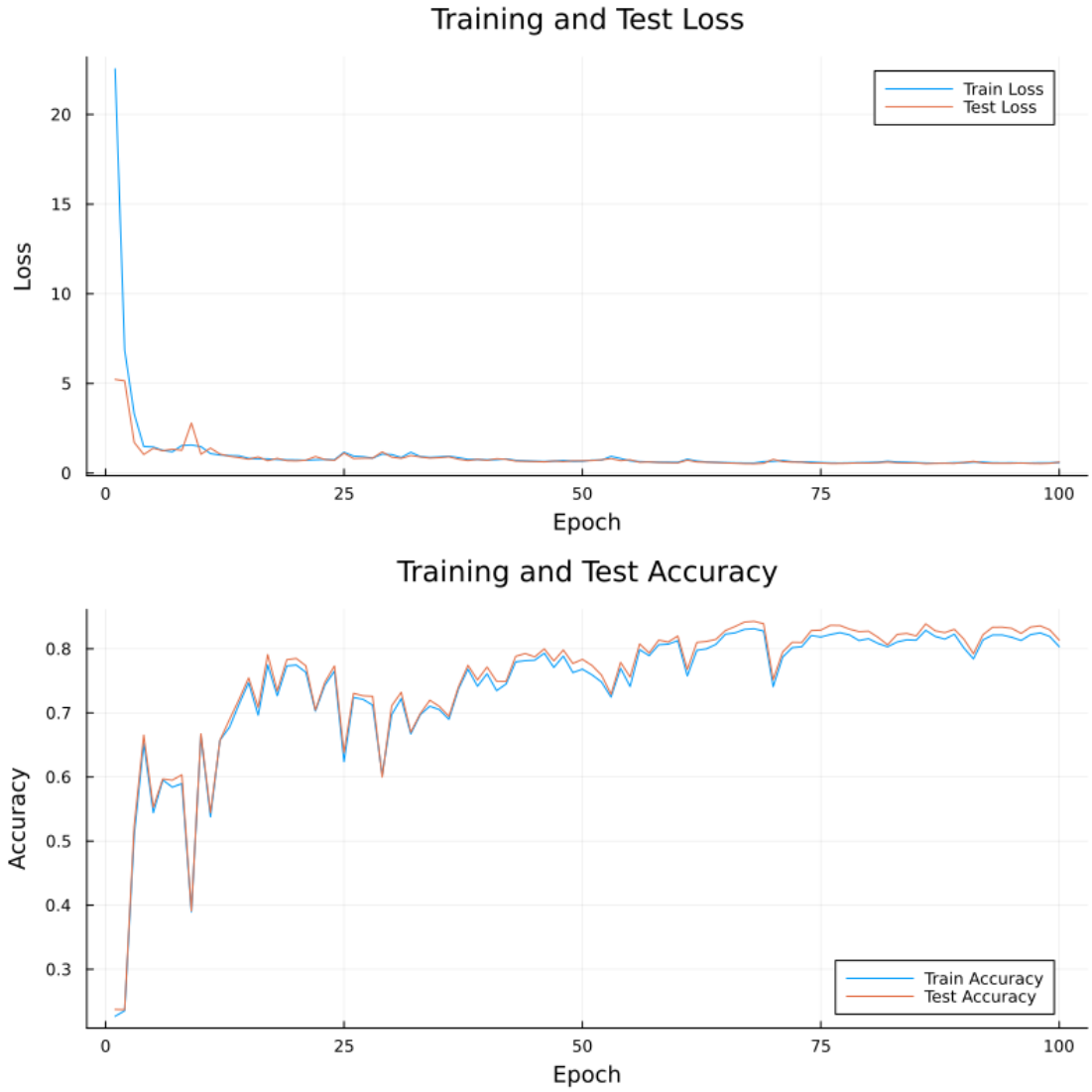


Figure 5.10: Loss and accuracy plots for a model built on rank-adaptive dynamical low-rank layers with regular update mechanism.

Figure 5.10 for the first model shows that the evolution of the losses exhibits few

fluctuations, while the accuracies exhibit more fluctuations, with a significant dip at the 9^{th} epoch. After 100 epochs, the training and the test losses are approximately 0.59 and 0.60, respectively. After 100 epochs, the training and the test accuracies are approximately 80.33 and 81.35 percent, respectively.

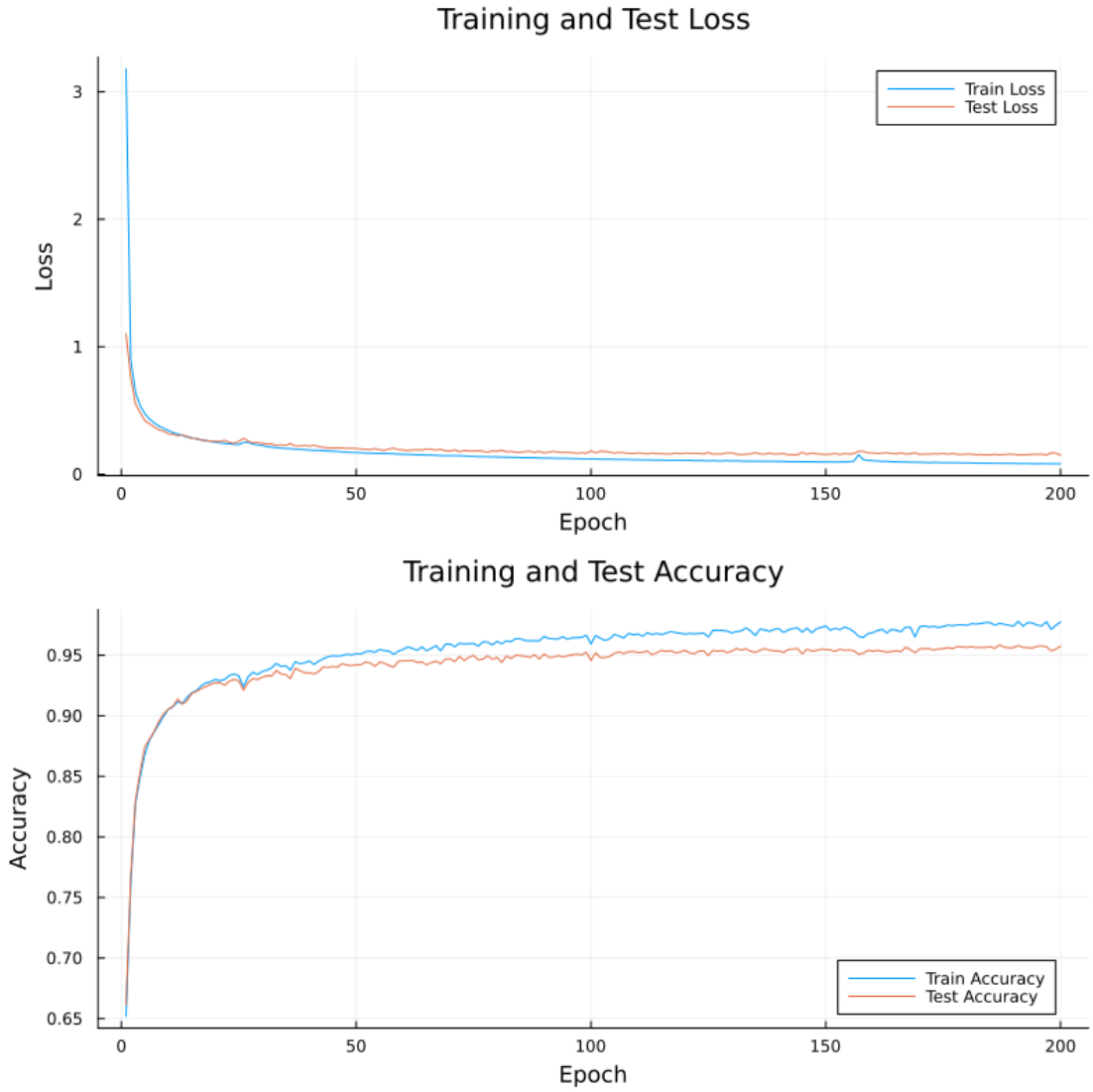


Figure 5.11: Loss and accuracy plots for a model built on rank-adaptive dynamical low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.001.

The second model consists of layers with additional momentum and gradient clipping in the training algorithm. When the model is trained with a learning rate of

0.001, the time taken for training and evaluation is 2753.39 seconds for 200 epochs. Likewise, when the model is trained with a learning rate of 0.01, the time taken for training and evaluation is 1360.24 seconds for 100 epochs.

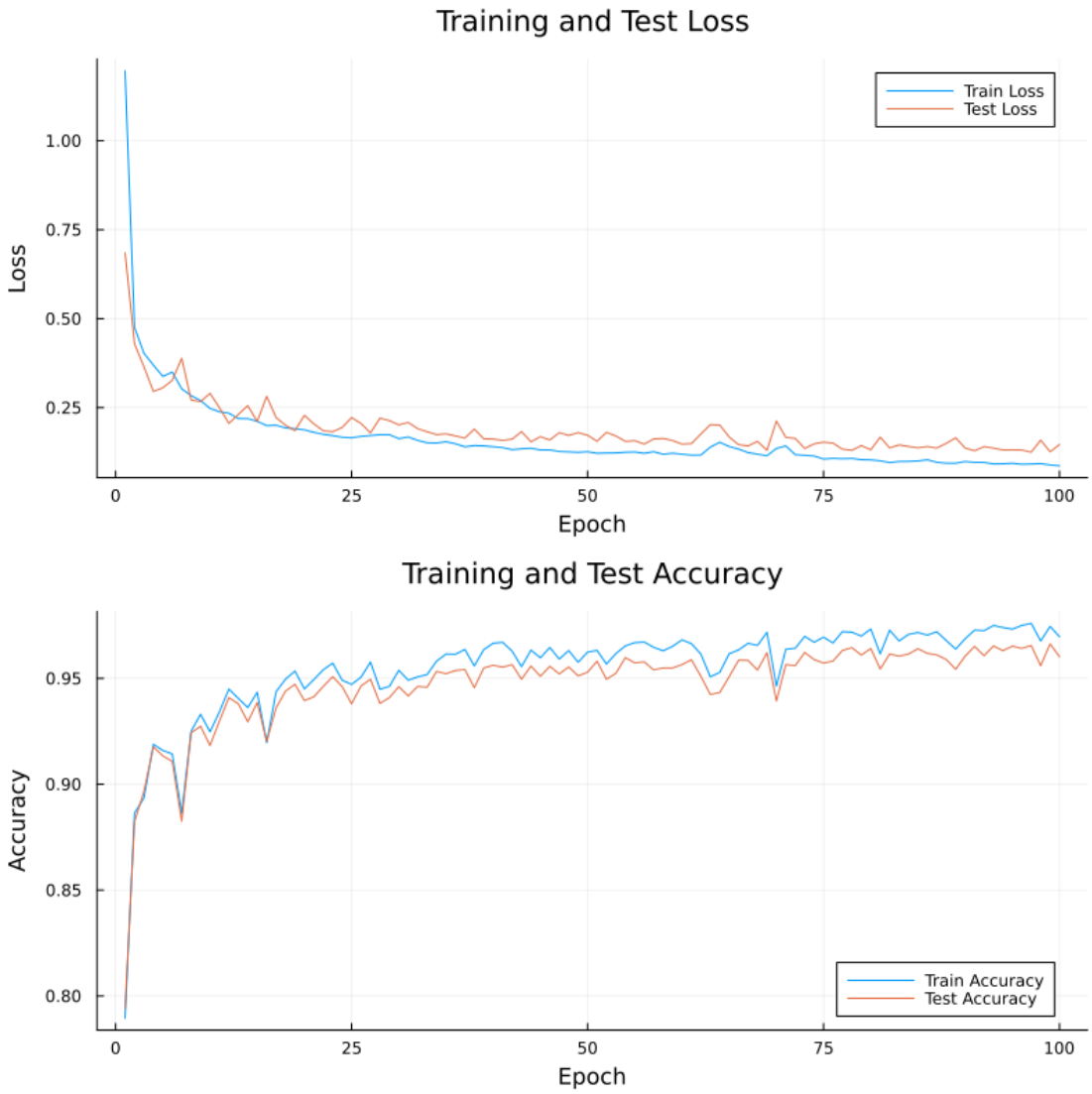


Figure 5.12: Loss and accuracy plots for a model built on rank-adaptive dynamical low-rank layers with additional momentum and gradient clipping in the update mechanism. The learning rate is 0.01.

Figure 5.11 for the second model with a learning rate of 0.001 shows that the evolution of the losses and accuracies are smoother than that in Figure 5.10. The losses are evolving smoothly, whereas the evolution of the accuracies exhibits slight

fluctuations. After 200 epochs, the training and the test losses are approximately 0.08 and 0.15, respectively. Similarly, the training and the test accuracies are approximately 97.74 and 95.75 percent, respectively.

Figure 5.12 for the second with a learning rate of 0.01 model shows that the evolution of the losses and accuracies shows more fluctuations than that in Figure 5.11 for the same model with a learning rate of 0.001. After 100 epochs, the training and the test losses are approximately 0.09 and 0.15, respectively. Similarly, the training and the test accuracies are approximately 96.97 and 96.02 percent, respectively.

These results are summarized in table 5.4.

Neural Network based on	**Learning Rate**	**Rank**	**Train and Test Accuracies (%)**	**Epochs**	**Runtime per epoch (s)**	**Total Time Taken (s)**
Rank-adaptive dynamical low-rank layers	0.001	max 40	80.33 and 81.35	100	14.71	1471.38
Rank-adaptive dynamical low-rank layers with momentum and gradient clipping	0.001	max 40	97.74 and 95.75	200	13.77	2753.39
Rank-adaptive dynamical low-rank layers with momentum and gradient clipping	0.01	max 40	96.97 and 96.02	100	13.60	1360.24

Table 5.4: Performance of rank-adaptive dynamical low-rank layers on the MNIST dataset.

5.5 Results from Pytorch implementation on MNIST dataset

The results are based on the performance of four different models constructed from the layers defined in Pytorch.

The first model consists of 3 custom dense layers. The input layer has 784 input and 256 output nodes with 'relu' activation. Similarly, the hidden layer has 256 input and 128 output nodes with 'relu' activation. Finally, the output layer has 128 input nodes and ten output nodes with 'linear' activation. The model is trained with a learning rate of 0.001. The time taken for training and evaluation is 757.13 seconds for 40 epochs.

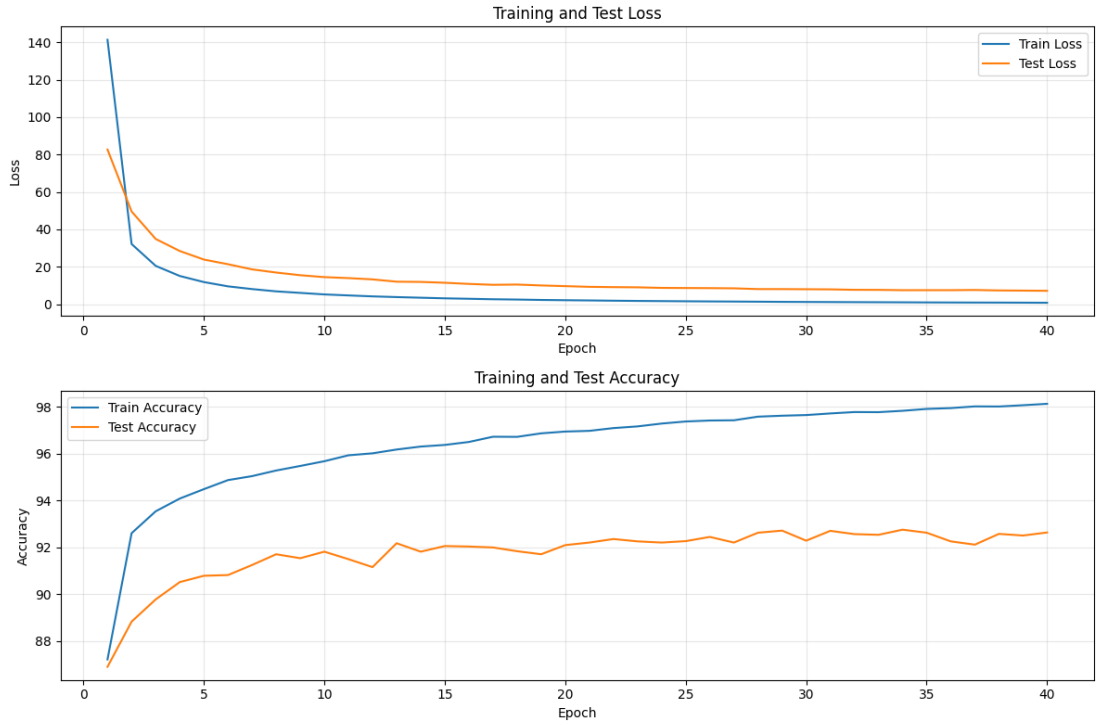


Figure 5.13: Loss and accuracy plots for a model built on custom dense layers.

Figure 5.13 for the first model shows that the losses decrease and the accuracies increase across the epochs. After the 40th epoch, the training loss is almost zero, whereas the test loss is around 7. Similarly, the training accuracy is around 98 percent, and the test accuracy is approximately 92.5 percent. One thing to note is

that the evolution of training and test losses and the training accuracy are smooth. However, the test accuracy fluctuates at some epochs.

The second model consists of 2 vanilla low-rank layers and one custom dense layer. The input layer has 784 input and 256 output nodes with 'relu' activation. Similarly, the hidden layer has 256 input and 128 output nodes with 'relu' activation. Finally, the output layer has 128 input and 10 output nodes with 'linear' activation. This model is further trained with two different input and hidden layer ranks. First, with rank 30, the model is trained with a learning rate of 0.0001. The time taken for training and evaluation is 906.68 seconds for 60 epochs. When the rank is 30, this model cannot show further improvement with additional epochs for a learning rate of 0.001 or higher. The accuracy starts at a value of around 10 percent and remains constant with additional epochs. Second, with rank 10, the model is trained with a learning rate of 0.001. The time taken for training and evaluation is 907.39 seconds for 60 epochs. When the rank is 10, this model cannot show further improvement with additional epochs for a learning rate of 0.01 or higher.

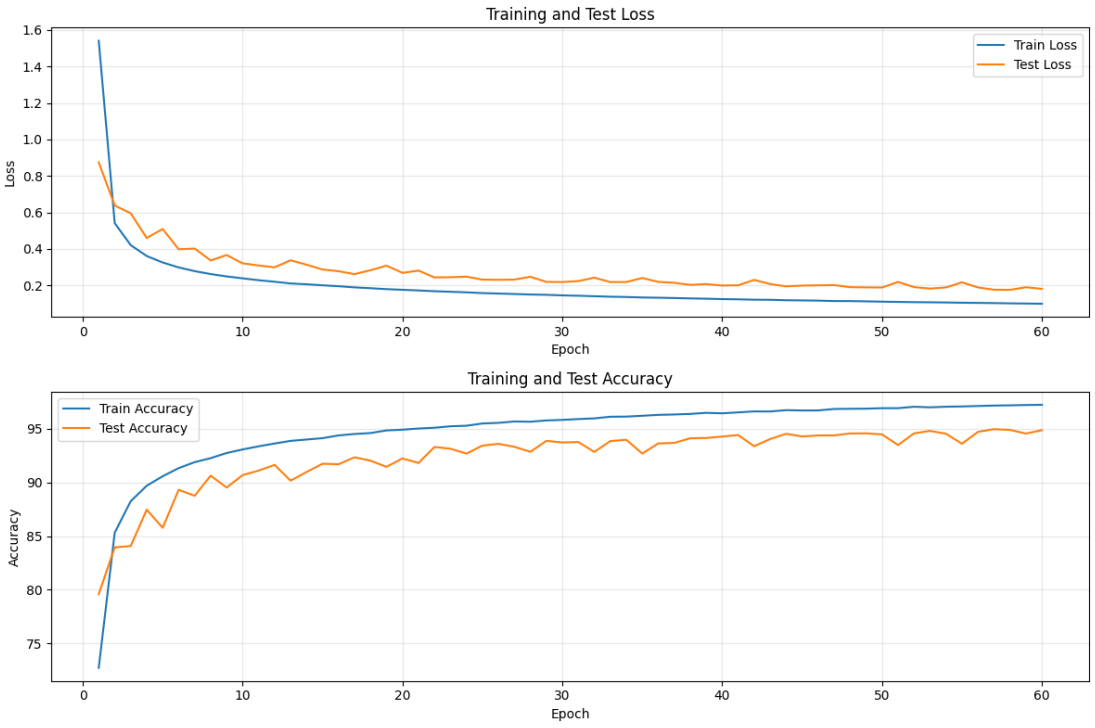


Figure 5.14: Loss and accuracy plots for a model built on vanilla low-rank layers with rank 30.

In Figure 5.14 for the second model with rank 30, the train and the test loss decrease to 0.1 and 0.2, respectively, after 60 epochs. Meanwhile, the train and test accuracies increased to 97 and 95 percent, respectively. The evolution of train loss and accuracy is smooth, while the test loss and accuracy fluctuate at different epochs.

The evolution of the losses and accuracies in Figure 5.15 for the second model with rank 10 is almost similar to the evolution of the losses and accuracies from Figure 5.14 for the same model with rank 30. The slight difference is that the test loss is approximately 0.18 after 60 epochs. Also, the evolution of test loss and accuracy exhibits fewer fluctuations.

The third model consists of 2 dynamical low-rank layers and one custom dense layer. All three layers have the same parameter values for input and output nodes, and the activation is like that of the second layer. This model is also trained with two different ranks for the input and the hidden layers but with the same learning rate of 0.01. First, with rank 30, the time taken for training and evaluation is 675.58 seconds for 40 epochs. Second, with rank 20, the time taken for training and evaluation is 620.75 seconds for 40 epochs.

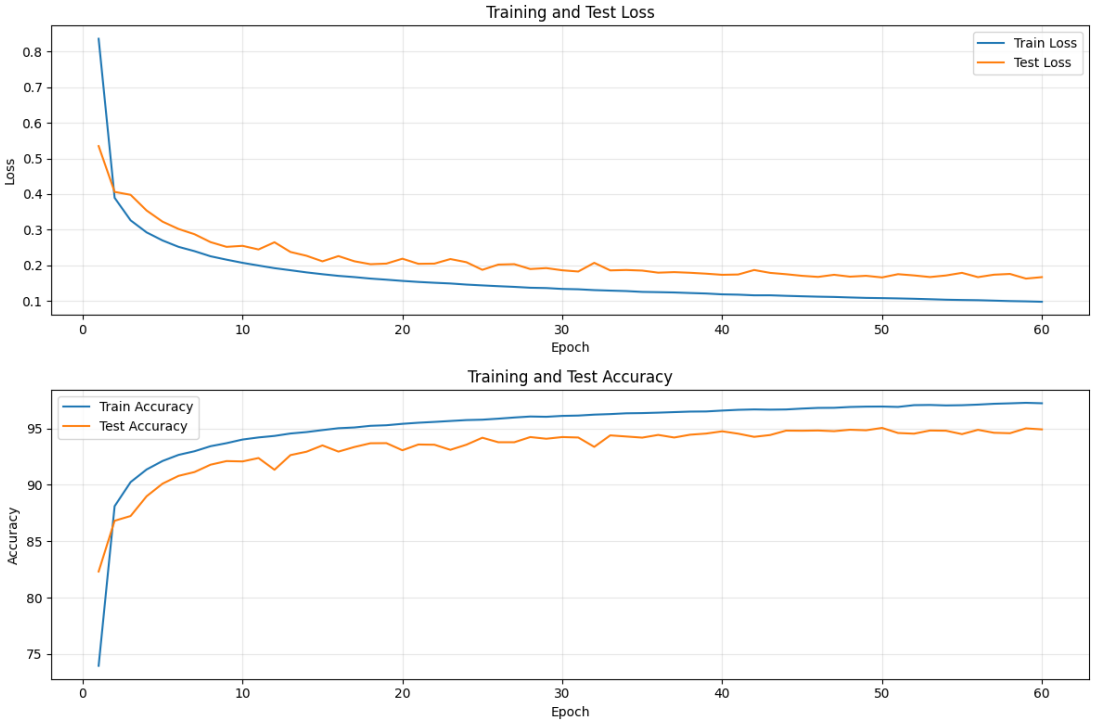


Figure 5.15: Loss and accuracy plots for a model built on vanilla low-rank layers with rank 10.

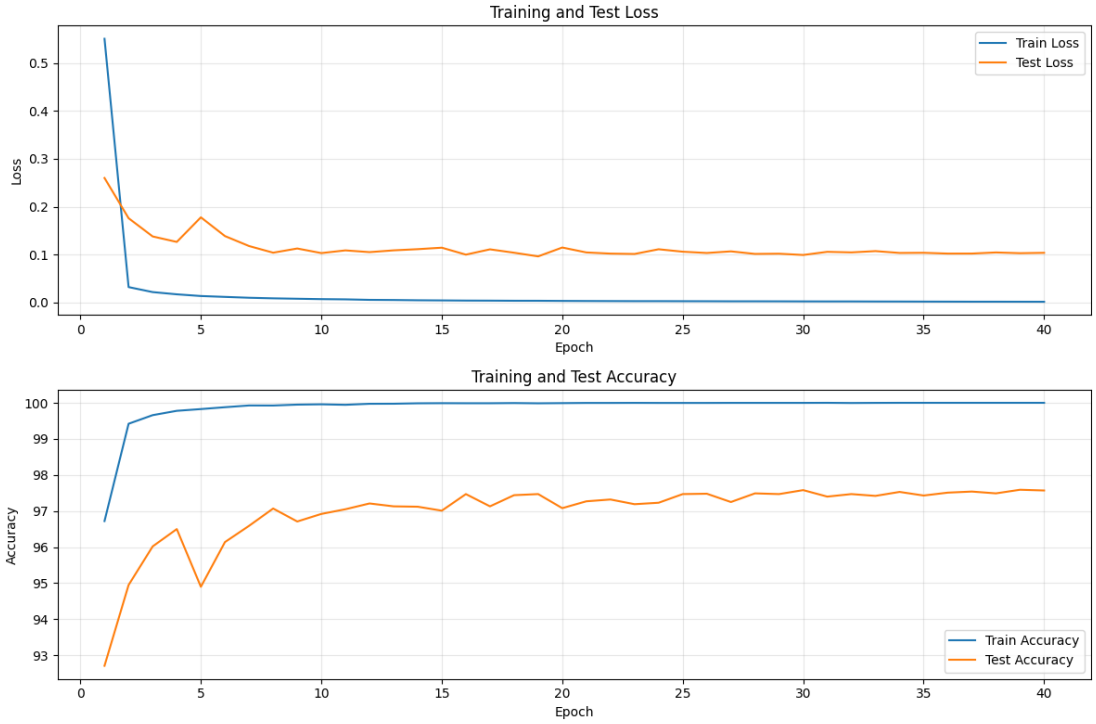


Figure 5.16: Loss and accuracy plots for a model built on dynamical low-rank layers with rank 30.

In Figure 5.16 for the third model with rank 30, the train and the test loss decrease quickly to 0 and 0.1, respectively, after only ten epochs. Meanwhile, the train and test accuracies increase to 100 and 97.60 percent, respectively after 40 epochs. The test loss and accuracy fluctuate at a few epochs.

In Figure 5.17 for the third model with rank 20, the train loss and accuracy approach 0 and 100 percent after 30 epochs. After 40 epochs, the test loss is 0.1, and the test accuracy is 97 percent. Additionally, the test loss and accuracy fluctuate at a few epochs.

The fourth model consists of 2 rank-adaptive dynamical low-rank layers and one custom dense layer. All three layers have the same parameter values for input and output nodes, as well as activation, like the second and third models. The learning rate is 0.01. The input and the hidden layer have a value of 5×10^{-3} for the tolerance parameter. This model is also trained for two different ranks; first, with a maximum rank of 50. The time taken for training and evaluation is 1383.28 seconds for 60 epochs. Second, with a maximum rank of 40. The time taken for

training and evaluation is 1801.03 seconds for 80 epochs.

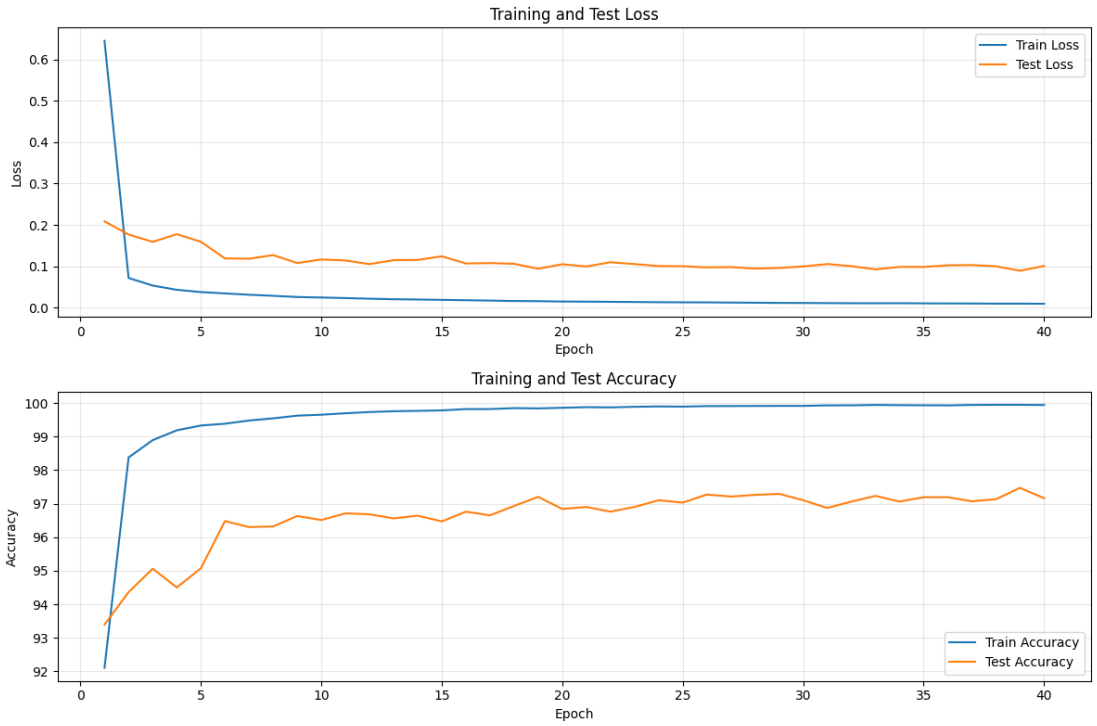


Figure 5.17: Loss and accuracy plots for a model built on dynamical low-rank layers with rank 20.

In Figure 5.18 for the fourth model with a maximum rank of 50, the train and the test loss decrease to approximately 0.13 and 0.17, respectively, after 60 epochs. Meanwhile, the train and test accuracies increased to approximately 95.6 and 95.3 percent, respectively. Like the other models, the test loss and accuracy fluctuate at different epochs.

In Figure 5.19 for the fourth model with a maximum rank of 40, the train and the test loss decrease to approximately 0.13 and 0.17, respectively, after 80 epochs. Meanwhile, the train and test accuracies increased to approximately 98.5 and 96.5 percent, respectively. Compared to Figure 5.18 for the same model with a maximum rank of 50, the test loss and accuracy plots exhibit more fluctuations.

These results are summarized in table 5.5.

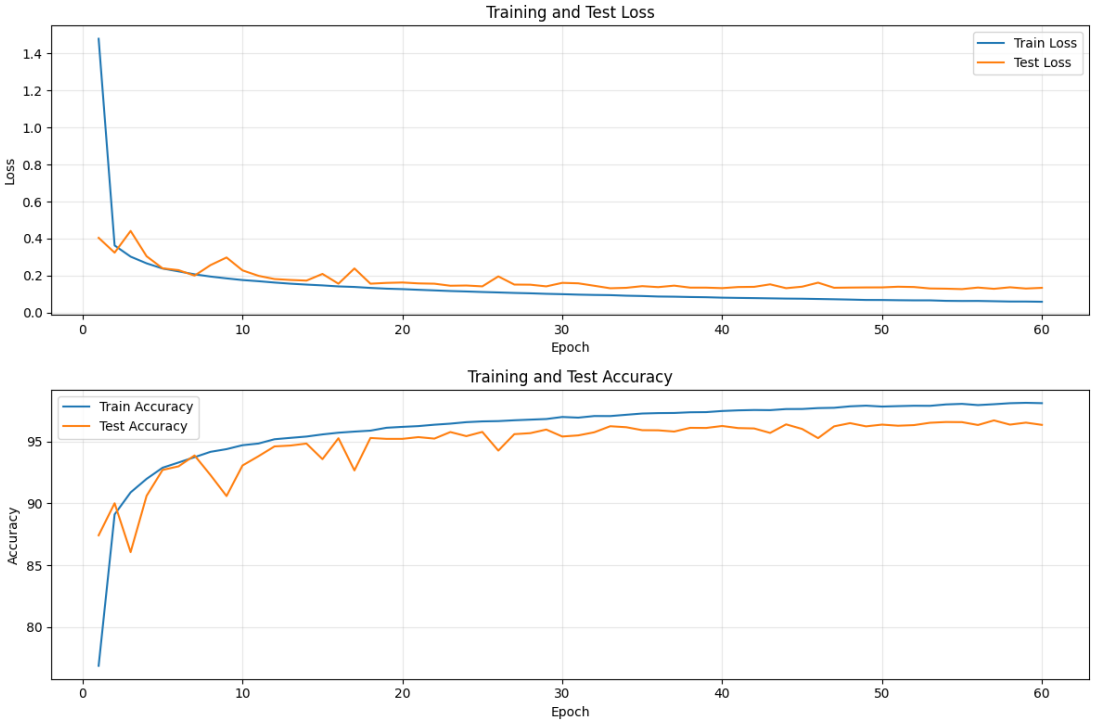


Figure 5.18: Loss and accuracy plots for a model built on rank-adaptive dynamical low-rank layers with a maximum rank of 50.

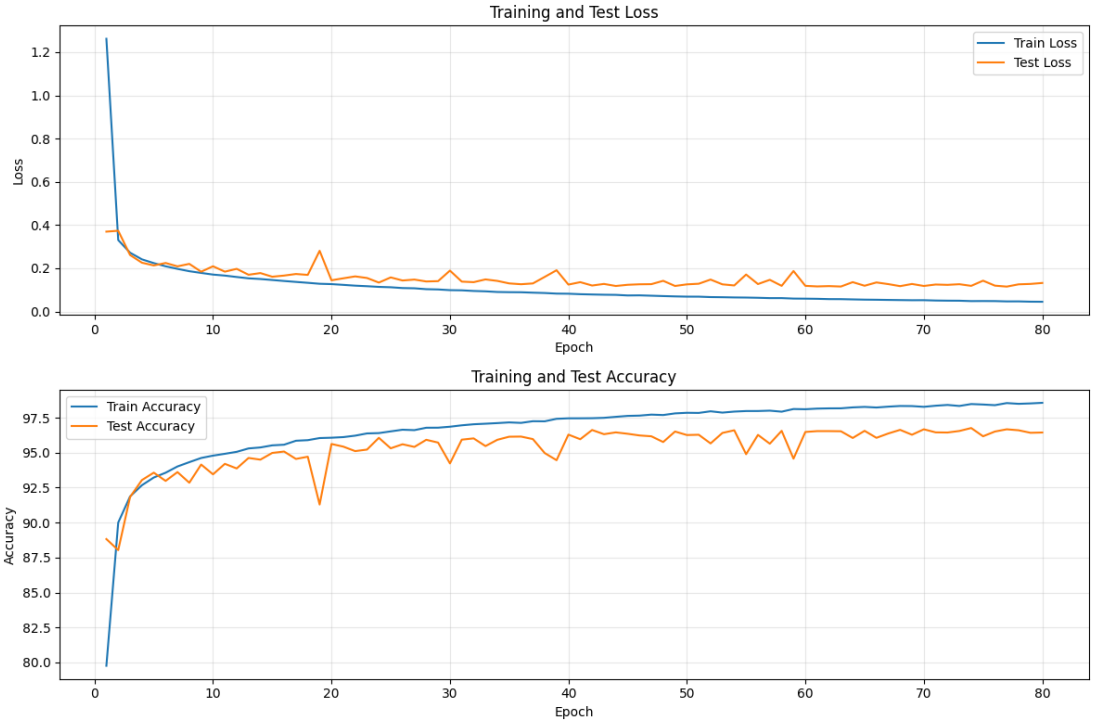


Figure 5.19: Loss and accuracy plots for a model built on rank-adaptive dynamical low-rank layers with a maximum rank of 40.

Neural Network based on	Learning Rate	Rank	Train and Test Accuracies (%)	Epochs	Runtime per epoch (s)	Total Time Taken (s)
Custom dense layers	0.0001	Full	98 and 92.5	40	18.93	757.13
Vanilla low-rank layers	0.0001	30	97 and 95	60	15.11	906.68
Vanilla low-rank layers	0.001	10	97 and 95	60	15.12	907.39
Dynamical low-rank layers	0.01	30	100 and 97.60	40	16.44	657.58
Dynamical low-rank layers	0.01	20	100 and 97	40	15.52	620.75
Rank-adaptive dynamical low-rank layers	0.001	max 50	95.60 and 95.30	60	23.05	1383.28
Rank-adaptive dynamical low-rank layers	0.01	max 40	98.50 and 96.50	80	22.51	1801.03

Table 5.5: Performance of full and low-rank neural network models in Python (Pytorch).

5.6 Biochemistry Data

The results are based on the dataset for which 14,000-time voltage pairs are read from the .bka files. In this section, the performance of five different models is presented graphically using two plots for each model. In the first plot, the values of mean absolute error losses are printed for the train, and the test sets across 2000 epochs. The second plot is a scatter plot depicting the values of predicted labels plotted against the actual labels. In this section, the selection of batch size is different than for the MNIST dataset. While working with the stopped-flow dataset, the batch size of 64 gave better performance for the dense layers, on the other hand, the batch size of 32 gave better performance for the low-rank layers. All the layers were tested for the batch sizes of 32 and 64.

The first model consists of 4 built-in dense layers provided by Flux. The input

layer has 28000 input and 512 output nodes with 'relu' activation. Similarly, the first hidden layer has 512 input and 512 output nodes with 'relu' activation. The second hidden layer has 512 input and 256 output nodes with 'relu' activation. Finally, the output layer has 256 input nodes and one output node. The ADAM optimizer is used with a learning rate of 0.001. The time taken for training and evaluation is 1543.8 seconds. Here in this architecture, there is one added layer (512, 512). This layer was added while testing the performance of different values in the architecture. The performance got better after adding the extra hidden layer with same input and output.

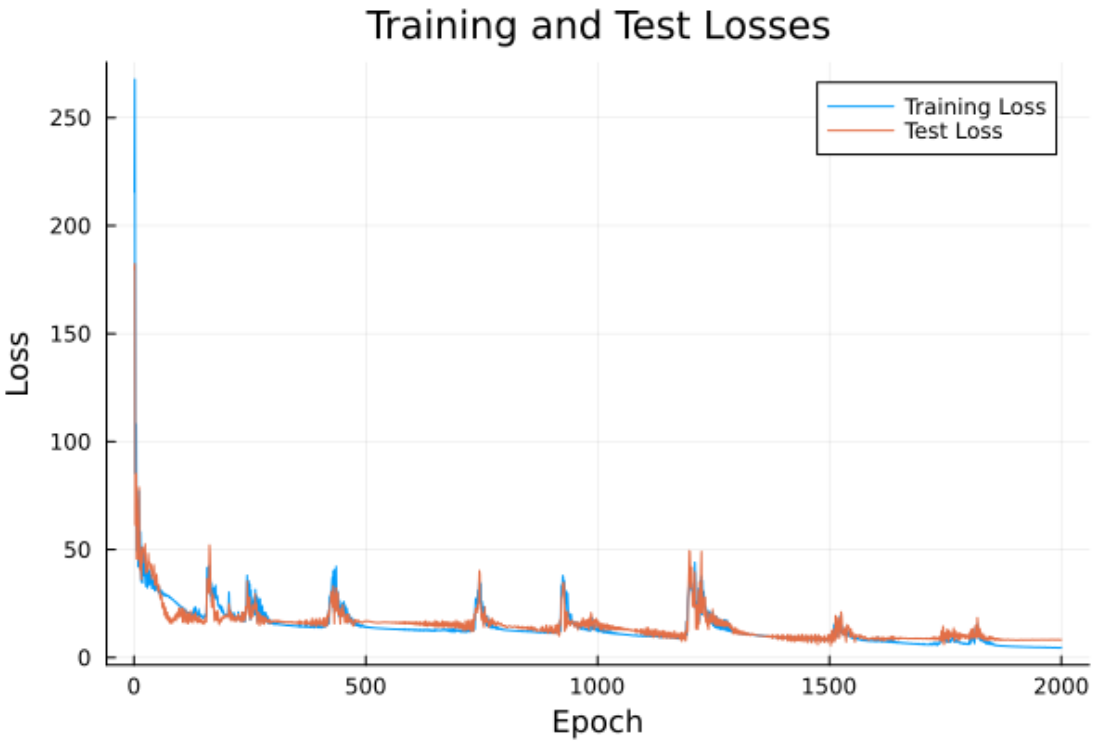


Figure 5.20: Losses for a model based on built-in dense layers.

From Figure 5.20 for the first model, it can be observed that the losses are decreasing with the increasing epochs. However, the learning outcome is unstable, and the evolution of the train and test loss is showing several spikes. But, we can still say that the learning outcome is becoming more stable after 1500 epochs. Even though there are some spikes, they are very short compared to the previous ones.

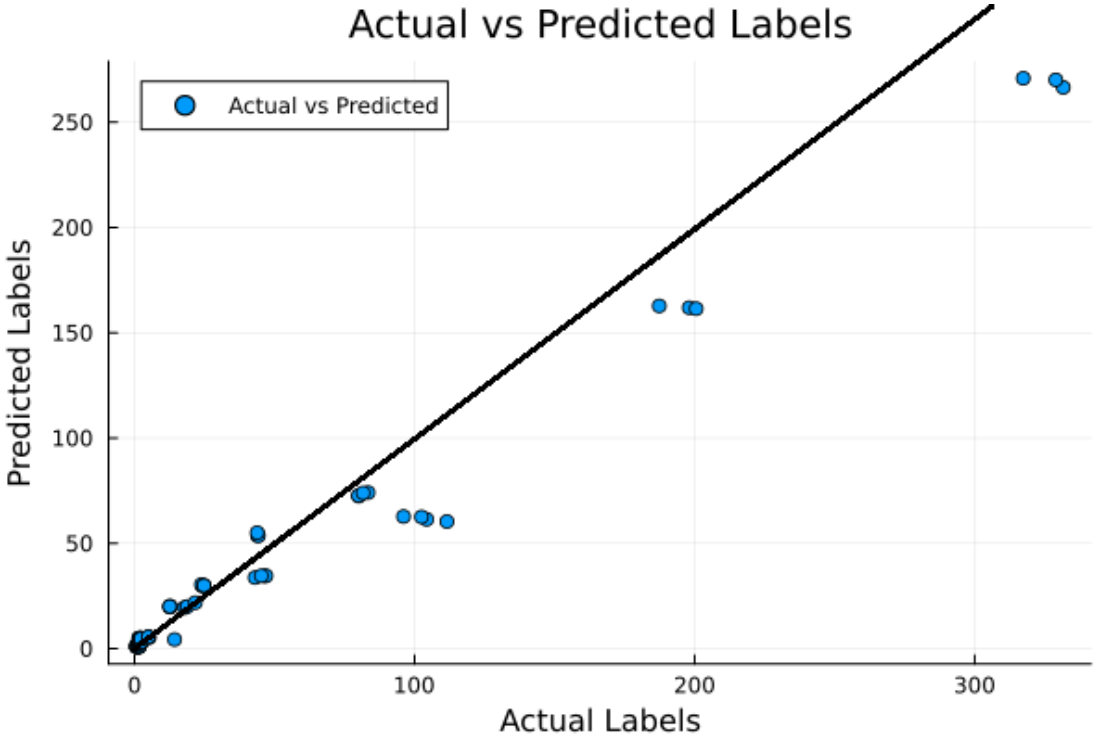


Figure 5.21: Scatter plot of labels for a model based on built-in dense layers.

From Figure 5.21 for the first model, it can be observed that the predicted labels are not accurate, however, they are close to the actual labels. The maximum value of the actual labels is around 325, and the maximum value of the predicted labels is around 270. For the value of the labels until around 80, the plot follows the diagonal. Whereas when the values of the actual labels are around 100, 200, and 320, then the values of the predicted labels are much smaller than the actual ones. Because of this, there are outliers around these values of the actual labels.

The second model consists of 3 custom dense layers defined for this thesis. The input layer has 28000 input and 512 output nodes with 'relu' activation. Similarly, the hidden layer has 512 input and 256 output nodes with 'relu' activation. Finally, the output layer has 256 input nodes and one output node with 'identity' activation. The model is trained with a learning rate of 0.001. The batch size is 64 for the dataloader. The time taken for training and evaluation is 13569.05 seconds. Here, also the model was tested with an added hidden layer but the added layer could not improve the performance like other models.

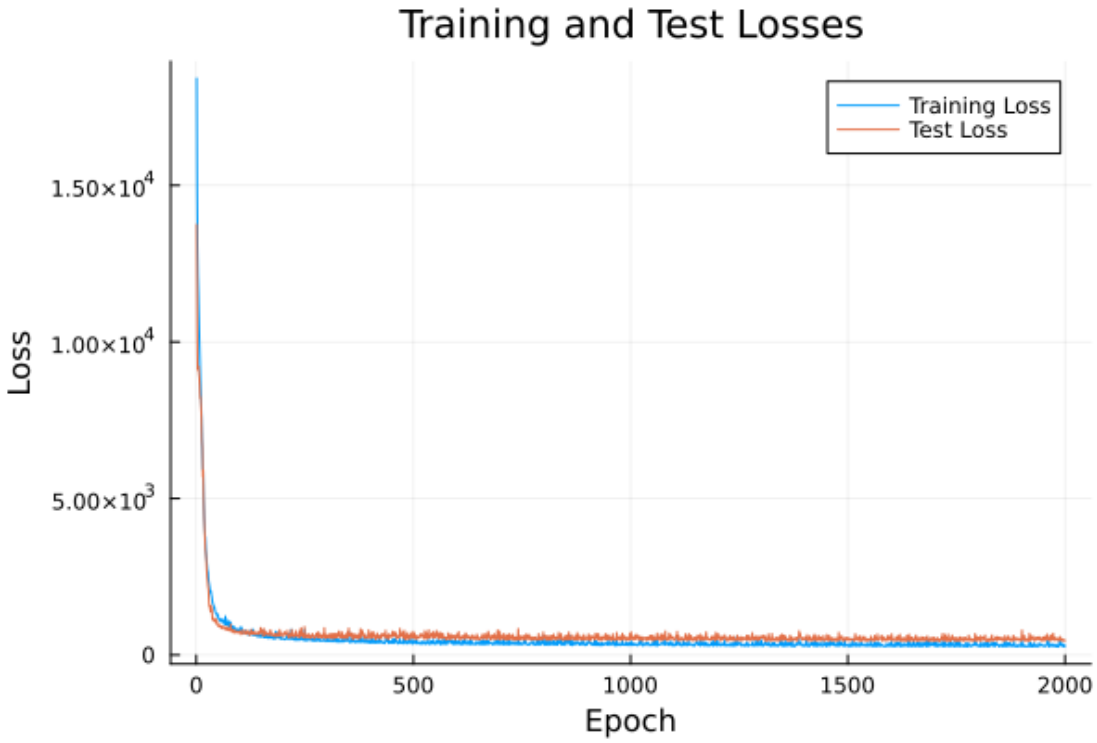


Figure 5.22: Losses for a model based on custom dense layers.

From Figure 5.22, it can be observed that the model cannot perform well for the dataset. The magnitude of the mean absolute error loss is exceptionally high for both the train and test set at the beginning. Although the model tries to decrease the loss, the loss is still high even after 2000 epochs.

Figure 5.23 shows that the values of predicted labels are very far from the actual labels. Many predicted labels have negative values, and since the labels are 'Rate', the values cannot be negative. The range of values for the predicted labels is from -3400 to 1200.

The third model consists of 3 custom vanilla training layers and one custom dense layer defined for this thesis. The input layer has 28000 input and 512 output nodes, with rank 50 and 'relu' activation. Similarly, the first hidden layer has 512 input and 512 output nodes with rank 50 and 'relu' activation. The second hidden layer has 512 input and 512 output nodes with rank 50 and 'relu' activation. Finally, the output layer has 256 input nodes and one output node with 'identity' activation. The model is trained with a learning rate of 0.001. The batch size is 32 for the dataloader. The time taken for training and evaluation is 3683.3 seconds.

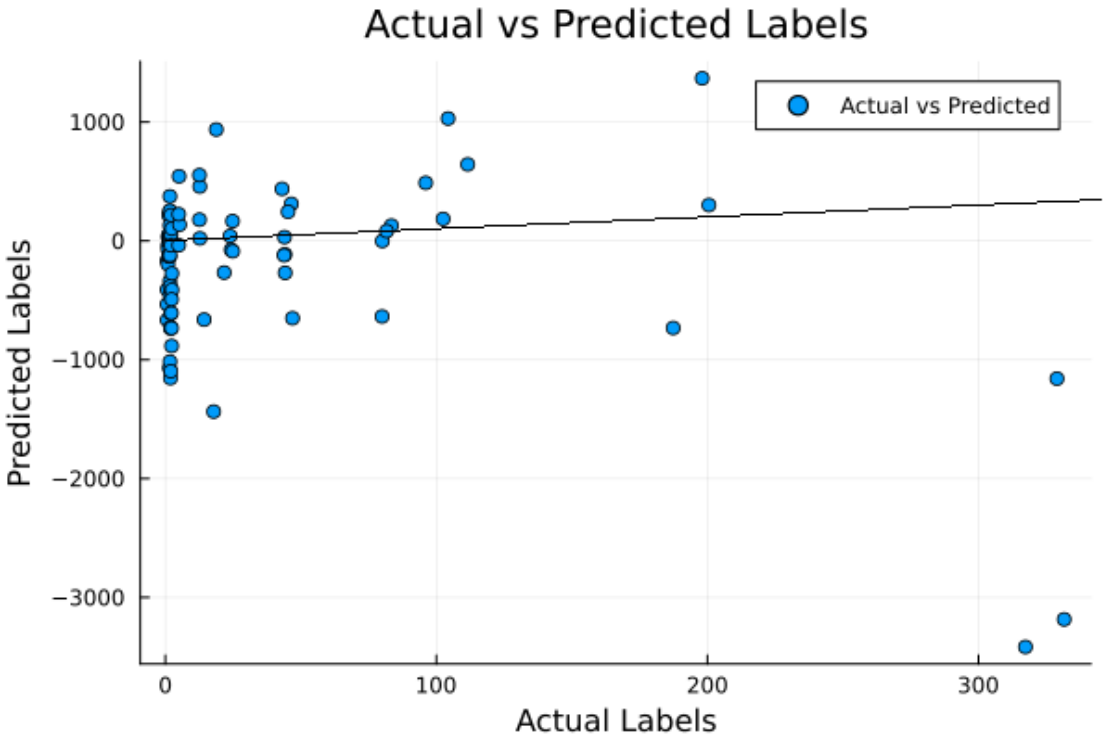


Figure 5.23: Scatter plot of labels for a model based on custom dense layers.

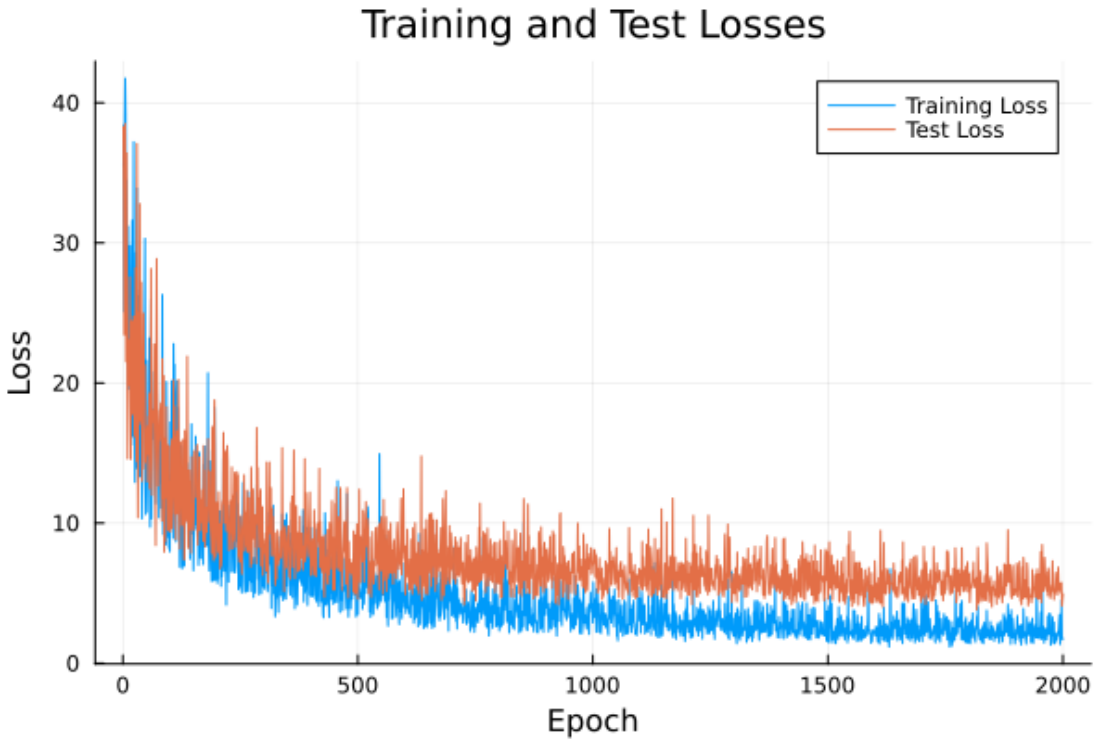


Figure 5.24: Losses for a model based based on vanilla training layers.

From Figure 5.24, it can be observed that the losses are decreasing with increasing epochs. However, the train and test loss plots are full of spikes. Although the plots are full of spikes, the overall trend is decreasing, and the test loss is settling around five after 2000 epochs.

Figure 5.25 shows that the predicted labels are close to the actual labels. The maximum value of the actual labels is around 325, and the maximum value of the predicted labels is around 240. For the value of the labels until around 200, the plot follows the diagonal. Whereas when the values of the actual labels are around 320, the values of the predicted labels are smaller than the actual ones. Because of this, there are outliers around these values of the actual labels.

The fourth model consists of three custom dynamical low-rank layers and one custom dense layer defined for this thesis. All the layers have the same parameters as the third model, with the only difference being that the input and the hidden layers are dynamical low-rank layers. The model is trained with a learning rate of 0.001. The batch size is 32 for the dataloader. The time taken for training and evaluation is 8161.07 seconds.

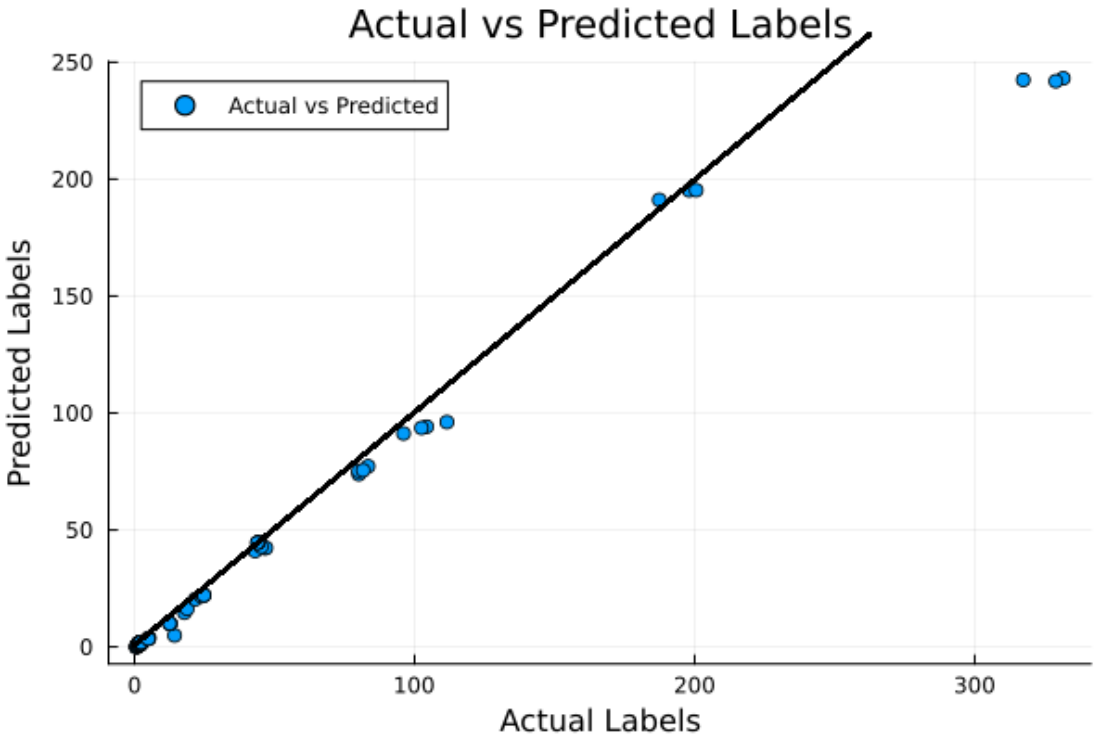


Figure 5.25: Scatter plot of labels for a model based on vanilla training layers.

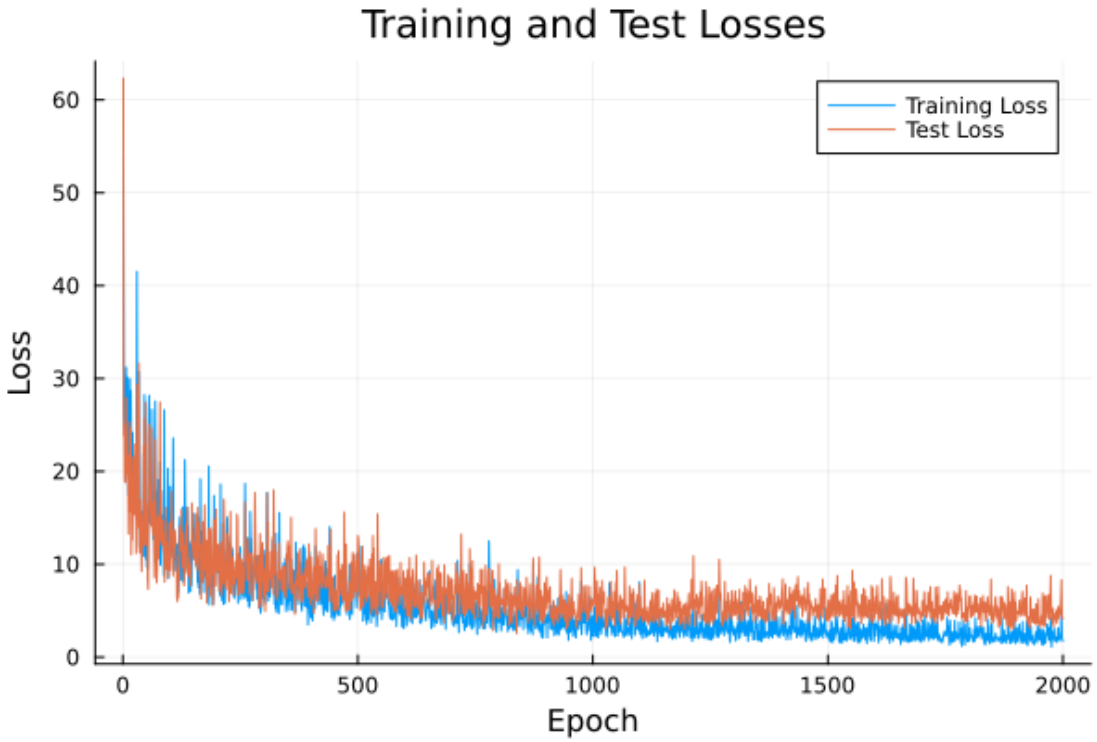


Figure 5.26: Losses for a model based on dynamical low-rank layers.

As observed in Figure 5.26, the train and test loss plots for this fourth model are similar to the ones for the third model. However, the spikes are shorter, and the gap between the two plots is smaller. The test loss is settling around five after 2000 epochs.

The scatter plot of Figure 5.27 is similar to Figure 5.25. The significant difference is that the maximum value of the predicted labels is around 260. Thus, the predicted labels are closer to the actual labels, around 320.

The fifth model consists of 4 built-in dense layers provided by Pytorch. All the layers have the same parameters as the first model. The ADAM optimizer is used with a learning rate of 0.0001. The time taken for training and evaluation is 7298.14 seconds.

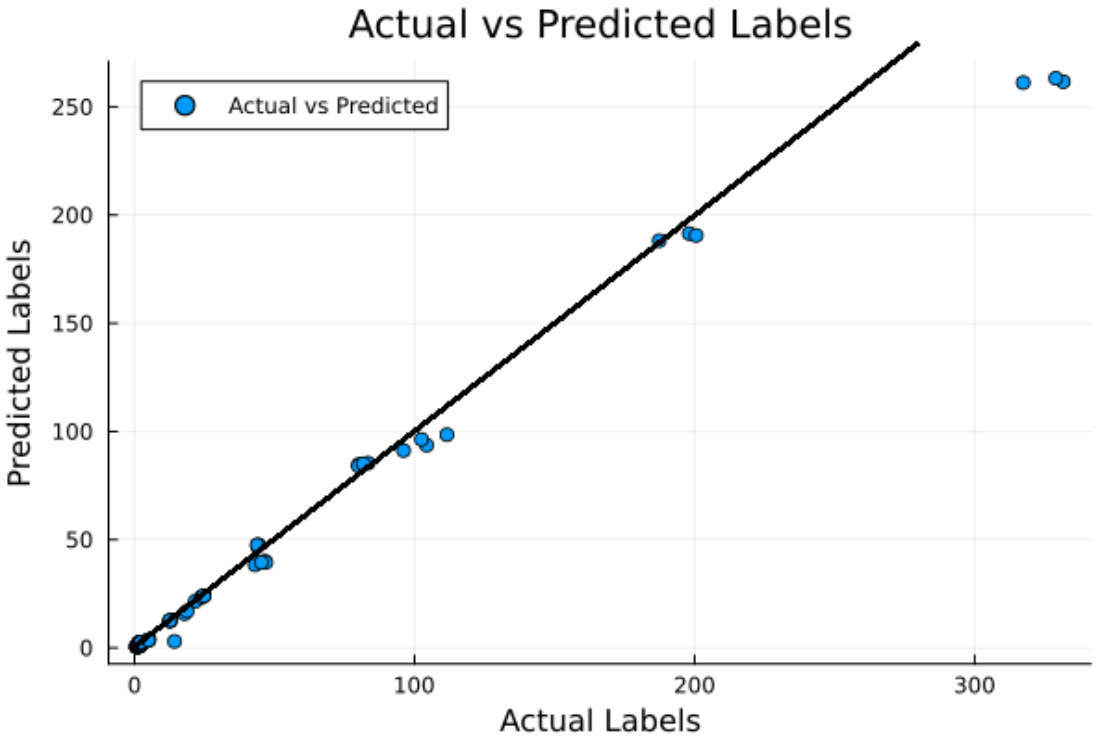


Figure 5.27: Scatter plot of labels for a model based on dynamical-low rank layers.

Figure 5.28: Losses for a model based on built-in dense layers of Pytorch.

In Figure 5.28 as well, the losses are decreasing with increasing epochs. However, the train and test loss plots are full of spikes, some very tall. The trend is declining, and the test loss is settling around three after 2000 epochs.

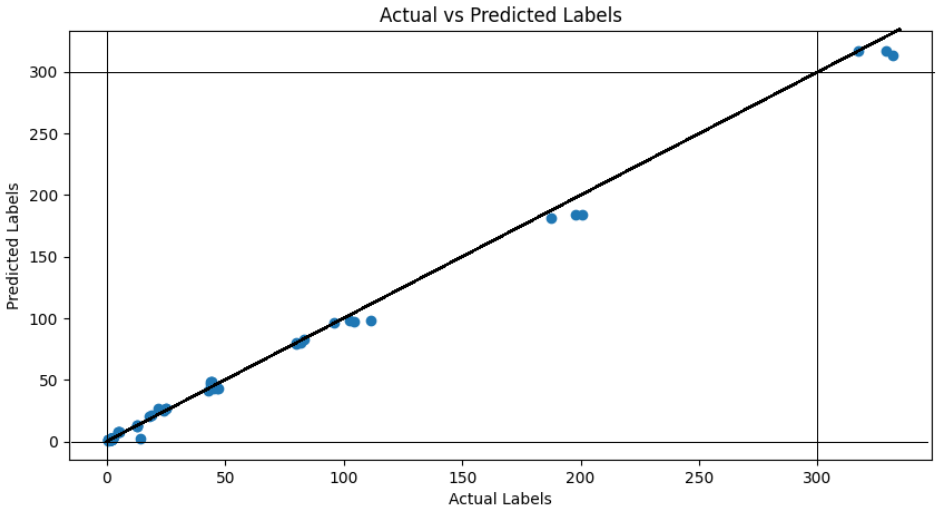


Figure 5.29: Scatter plots of labels for a model based on built-in dense layers of Pytorch.

The scatter plot of Figure 5.29 depicts the best performance among all the models. The points in the plot follow the diagonal line till the end of the plot. Still, there are some outliers, but fewer than the previous models.

These results are summarized in table 5.6.

Neural Network based on	**Learning Rate**	**Rank**	**Loss**	**Epochs**	**Total Time Taken (s)**
Built-in dense layers of Flux	0.001	Full	Approx. 10	2000	1543.8
Custom dense layers with momentum and gradient clipping	0.001	Full	Approx. 300	2000	13569.05
Vanilla low-rank layers with momentum and gradient clipping	0.001	50	approx. 5	2000	3683.3
Dynamical low-rank layers with momentum and gradient clipping	0.001	50	approx. 5	2000	8161.07
Built-in dense layers of Pytorch	0.0001	Full	approx. 3	2000	7298.14

Table 5.6: Performance on stopped-flow biochemistry dataset.

Chapter 6

Discussion

6.1 Resource saving by low-rank neural networks in Julia

The low-rank neural network models based on the low-rank layers defined for this thesis, such as the vanilla low-rank layer, the dynamical low-rank layer, and the rank-adaptive dynamical low-rank layer, consume less computational and memory resources. We can compare the results from sections 5.2, 5.3, and 5.4 with the result observed from the neural network model based on the built-in dense layer of Flux, trained with an SGD optimizer. This result can be found in Figure 6.1. This figure can be considered as the basis for the comparison. If we use the ADAM optimizer instead of the SGD optimizer, the model can converge quickly and cross the 97 percent test accuracy after twenty epochs. However, the custom layer and other low-rank layers defined for this thesis are based on the SGD mechanism for updating the trainable parameters. Thus, we are considering the model trained with SGD optimizer as the base for comparison.

Figure 6.1 for a model built on built-in dense layers shows that the train and test losses are approximately 0.05 and 0.09, respectively, after 200 epochs. Similarly, the train and the test accuracies are approximately 98.57 percent and 97.34 percent, respectively. The time taken is 1134.05 seconds. From sections 5.2, 5.3, and 5.4, we choose the results obtained by the models with additional momentum and gradient clipping trained with a learning rate of 0.001. We are choosing the best results from these sections to compare with those from Figure 6.1. The model based on the vanilla low-rank layer from 5.2 gives training and test accuracies

of approximately 99.95 percent and 96.6 percent, respectively, after 200 epochs by taking 834.63 seconds. Similarly, the train and the test losses are approximately 0.01 and 0.18, respectively. Likewise, the model based on the dynamical low-rank layer from 5.3 gives training and test accuracies of approximately 99.86 percent and 96.63 percent, respectively, after 200 epochs with a training time of 1401.45 seconds. The train and the test losses are approximately 0.01 and 0.16, respectively. Finally, the model based on the rank-adaptive dynamical low-rank layer from section 5.3 gives training and test accuracies of approximately 97.74 percent and 95.75 percent, respectively, after 200 epochs with a training time of 2753.39 seconds. The train and the test losses are approximately 0.08 and 0.15, respectively. We can see that low-rank neural networks also produce results closer to the neural network based on built-in dense layers. On top of that, the neural network based on the vanilla low-rank layer can even produce results faster than the ones based on the built-in dense layer. Vanilla low-rank layers can save a total of 299.42 seconds in comparison to the built-in dense layers to complete 200 epochs. On the other hand, the neural networks based on the dynamical low-rank and rank-adaptive dynamical low-rank layers take more time than those based on built-in dense layers.

These models based on the low-rank layers require fewer computational and memory resources. We can visualize this mathematically. Since the output layer is dense for all the models, we do not focus on the computations of costs. The low-rank neural networks are saving resources in the input and hidden layers. The values of the parameters input and output are 782 and 256 for the input layer. So, a dense layer will have a *weight* matrix of size 256×784 and a *bias* matrix of size 256×1. There will be 200,960 different values to be stored in the matrices. For the vanilla and the dynamical low-rank layers, the value of the rank parameter is 20. In the case of the vanilla low-rank layer, the *weight* matrix is replaced by the U, S, and V matrices of sizes 256×20, 20×20, and 20×784. Moreover, the same size momentum matrices are present for the U, S, V, and *bias* matrices. Thus, there is a total of 42,912 different values to be stored in the matrices. The dynamical low-rank layer also has all these matrices of the vanilla low-rank layer. In addition, it has two more matrices, one of size equal to the U matrix and another of size equal to the V matrix. Thus, there is a total of 63,712 different values to be stored in the matrices. For the rank-adaptive dynamical low-rank layer, the maximum rank is 40. During training, the rank consistently alternated between 20 and 40. For the majority of an epoch, the rank stayed at 20 for four consecutive instances, then shifted to 40 for the next two instances. This alternating pattern of four instances of rank 20 followed by two instances of rank 40, repeated throughout the epoch. However, towards the end of the epoch, the rank stabilized at 20. So, we can say that the number of different values to be stored in the matrices is between

128,512 and 63,712.

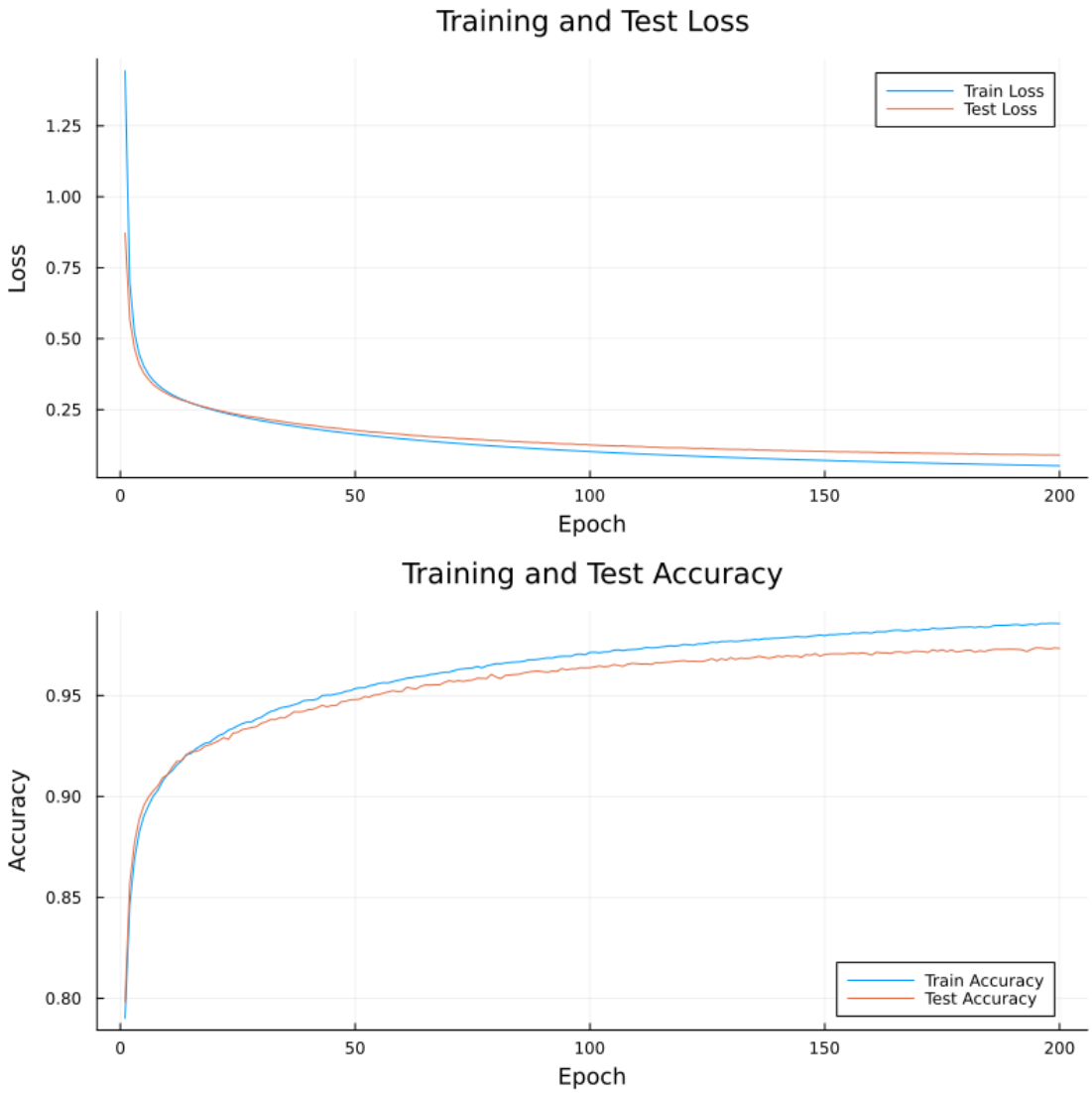


Figure 6.1: Loss and accuracy plots for a model built on built-in dense layers. The learning rate is 0.001, and the optimizer is SGD optimizer.

Similar calculations can also be used to obtain the number of different values stored in the matrices for the hidden layer, which has input and output parameter values of 256 and 128, respectively. The dense layer has 32,896 different values to be stored in the matrices. The vanilla low-rank layer has 16,416 values, and the dynamical low-rank layer has 24,096 values to store in the matrices. So, for

the rank-adaptive dynamical low-rank layer, the number of different values to be stored in the matrices is between 49,536 and 24,096.

With the help of low-rank layers, we can define low-rank neural networks that require far fewer resources but still produce results close to those of full-rank neural networks.

6.2 Comparison of performance of Flux with and without the addition of momentum and gradient clipping

Adding momentum and gradient clipping in the update mechanism can significantly improve the performance of a neural network based on custom layers. This can be observed in the plots from sections 5.1 for models based only on the custom dense layers, 5.2 for models based on vanilla low-rank layers, 5.3 for models based on dynamical low-rank layers, and 5.4 for models based on rank-adaptive dynamical low-rank layer.

Referring to Figure 5.1 from section 5.1, the evolution of the accuracy is unstable and full of fluctuates for the models based on the custom dense layer. The learning rate of 0.001 was chosen because it gave a better result. The accuracy did not increase well when the model was trained with a learning rate of 0.01. The train and the test accuracies started low at around 10 percent and reached only around 35 percent after 20 epochs. Then, after defining models based on the custom dense layers with additional momentum and gradient clipping, the accuracies evolved smoothly, as seen in Figure 5.2. Also, after adding momentum and gradient clipping, the neural network model can give better results even with a learning rate of 0.01, as depicted in Figure 5.3.

For the neural network models in sections 5.2, 5.3, and 5.4 as well, the learning rate was chosen as 0.001 while training with layers without additional momentum and gradient clipping. When the vanilla low-rank, dynamical low-rank, and rank-adaptive dynamical low-rank layers without momentum and gradient clipping were used to design neural network models, the models did not show further improvement with additional epochs when trained with a learning rate of 0.01. For the models based on the vanilla low-rank and the dynamical low-rank layers, the accuracies and losses remained constant at around 10 percent and 32.5, respectively. Meanwhile, for the models based on the rank-adaptive dynamical low-rank layers,

the accuracies and losses remained constant at around 9 percent and 33, respectively.

In sections 5.2 and 5.3, the accuracies and the losses evolved more smoothly after adding momentum and gradient clipping when trained with a learning rate of 0.001. Also, the gap between the train and test losses and accuracies is getting smaller. Whereas in section 5.4, the addition of momentum and gradient clipping made the evolution of losses and accuracies smoother and significantly increased the accuracies. Moreover, these models are also producing good results for a learning rate of 0.01. However, there are more fluctuations in the evolution of the losses and the accuracies than when the models are trained with a learning rate of 0.001. Also, the gap between the train and test accuracies and losses is increased. Still, the models can produce good results with a learning rate of 0.01 after adding momentum and gradient clipping in the layers.

The addition of momentum and gradient clipping is improving Flux's performance greatly. However, these improvements come at a cost. The resource consumed is increased as the layers need to have additional matrices for holding the momentum parameters.

6.3 DLRT: Fixed-rank vs Rank-adaptive

The DLRT algorithm from section 3.2 is implemented in this thesis in two ways: fixed-rank and rank-adaptive. With the help of results from sections 5.3 and 5.4, we can compare the performances of these two approaches. For the fixed-rank implementation, the rank is set to 20. Whereas, for the rank-adaptive implementation, the maximum rank is set to 40, and the truncation tolerance is 5×10^{-3}.

Figure 5.7 from section 5.3 shows that the DLRT algorithm can give a satisfactory result without adding momentum and gradient clipping when implanted as fixed-rank. After 200 epochs the test accuracy is nearly 97 percent. On the other hand, Figure 5.10 from section 5.4 shows that the DLRT algorithm cannot give a satisfactory result without adding momentum and gradient clipping when implemented as rank-adaptive. After 200 epochs, the test accuracy is nearly 81 percent. Besides, the evolution of accuracies is highly unstable at early epochs.

We can recap the results from sections 5.3 and 5.4 for ease of presentation. After adding momentum and gradient clipping to the training algorithm, the evolution of the accuracies is smoother for both the fixed-rank and rank-adaptive imple-

mentation of the DLRT algorithm. From the results of section 5.3, we can see that the fixed-rank implementation gives the test accuracy of 96.63 percent after 200 epochs for a learning rate of 0.001. Similarly, for a learning rate of 0.01, the test accuracy is 97.70 percent after 100 epochs. Likewise, from the results of section 5.4, we can see that the rank-adaptive implementation gives the test accuracies of 95.75 percent after 200 epochs. Here, the values of the rank consistently alternated between 20 and 40 during the training process. Throughout the majority of the epoch, the six-value pattern (20, 20, 20, 20, 50, 50) repeated consistently until the rank stabilized at 20 towards the end. Then, for a learning rate of 0.01, the test accuracy is 96.02 percent. And, the values of the rank consistently alternated between 19 and 38 during the training process. Throughout the majority of the epoch, the six-value pattern (19, 19, 19, 19, 38, 38) repeated consistently until the rank stabilized at 24 towards the end.

In all of the three cases discussed above, the DLRT algorithm produces better results when the rank is kept constant. Another thing to note is that the rank-adaptive approach consumes more resources. The minimum rank is either 19 or 20 and the maximum rank is either 38 or 40 depending upon whether the learning rate is 0.01 or 0.001. Whereas, for the fixed-rank approach, the rank is just 20 for both learning rates. Despite consuming lower resources, the fixed-rank approach is able to produce better results than the rank-adaptive approach. The rank-adaptive approach does extra jobs and consumes more resources, yet, it is not able to beat the results of the fixed-rank approach. So, we can say that if we have to implement the DLRT algorithm, it is better to keep the rank fixed. However, for the fixed-rank approach, we will have to try different values of rank to pick the best one.

Besides the results presented in section 5.4, the model based on rank-adaptive dynamical low-rank layers with additional momentum and gradient clipping was also trained with a maximum rank set to 50, a learning rate of 0.001 and a truncation tolerance of 5×10^{-3}. In this case, the rank alternated between 24 and 50. Throughout the majority of the epoch, the six-value pattern (25, 24, 25, 24, 50, 48) repeated consistently until the rank stabilized at 24 towards the end. If we compare this to the case when the same model with a learning rate of 0.001 and a truncation tolerance of 5×10^{-3}, the rank stabilized at 20 towards the end of an epoch. Thus, for the rank-adaptive dynamic low-rank layer, the value of the lowest rank depends upon the value of the maximum rank. So, we still have to try different values of the maximum rank to get an optimum result. This also supports a fixed-rank approach while implementing the DLRT algorithm.

6.4 Julia vs Python

One of the major goals of this thesis is to compare the implementation of the DLRT algorithm from section 3.2 in Julia (Flux) with that in Python (Pytorch). The results from the sections 5.1, 5.2, 5.3, 5.4 and 5.5 are discussed in this section.

Neural Network based on	**Learning Rate**	**Rank**	**Train and Test Accuracies (%)**	**Epochs**	**Total Time Taken (s)**
Vanilla low-rank layers	0.001	20	99.7 and 96.8	100	378.16
Vanilla low-rank layers with momentum and gradient clipping	0.01	20	99.7 and 97.2	100	418.58
Dynamical low-rank layers	0.001	20	99.60 and 97	100	657.66
dynamical low-rank layers with momentum and gradient clipping	0.01	20	100 and 97.7	100	553.60
Rank- adaptive dynamical low-rank layers	0.001	max 40	80.33 and 81.35	100	1471.38
Rank-adaptive dynamical low-rank layers with momentum and gradient clipping	0.01	max 40	96.97 and 96.02	100	1360.24

Table 6.1: Performance of low-rank neural network models in Julia (Flux).

In the early stages of this thesis, built-in dense layers were used to construct a neural network model to train and evaluate the MNIST dataset. Two different optimizers were used: ADAM and SGD with a learning rate of 0.001. Here, the learning rate of 0.001 was selected because Flux could not train with a learning

Neural Network based on	**Learning Rate**	**Rank**	**Train and Test Accuracies (%)**	**Epochs**	**Runtime per epoch (s)**	**Total Time Taken (s)**
Vanilla low-rank layers	0.0001	30	97 and 95	60	15.11	906.68
Vanilla low-rank layers	0.001	10	97 and 95	60	15.12	907.39
Dynamical low-rank layers	0.01	30	100 and 97.60	40	16.44	657.58
Dynamical low-rank layers	0.01	20	100 and 97	40	15.52	620.75
Rank-adaptive dynamical low-rank layers	0.001	max 50	95.60 and 95.30	60	23.05	1383.28
Rank-adaptive dynamical low-rank layers	0.01	max 40	98.50 and 96.50	80	22.51	1801.03

Table 6.2: Performance of low-rank neural network models in Python (Pytorch).

rate of 0.01 for the SGD optimizer. The results were different in Python and Julia. When the built-in dense layers provided by the Pytorch package were used to construct a model, the number of epochs required for converging was closer for the two optimizers with a difference of only 25. In Python, the model trained on the ADAM optimizer converged with only 20 epochs giving train and test accuracies of 97.23 and 97.24 percent respectively. Similarly, when the model was trained on the SGD optimizer, it converged after 45 epochs giving train and test accuracies of 97.09 and 97.05 percent respectively. However, when the built-in dense layers provided by the Flux package were used to construct a model, the number of epochs required for converging was far from each other for the two optimizers with a difference of 80. In Julia, the model trained on the ADAM optimizer converged with only 20 epochs giving train and test accuracies of 99.06 and 97.50 percent respectively. But, when the model was trained on the SGD optimizer, it converged after 100 epochs giving train and test accuracies of 98.56 and 97.34 percent respectively.

The results from the section 5.5 show that Pytorch takes fewer epochs to converge. The model built on the custom dense layer gives the train and test accuracies of

approximately 98 percent and 92.5 percent just after 40 epochs by taking 757.13 seconds. The results for the low-rank neural network models can be observed in the table 6.2.

On the other hand, the results from sections 5.1, 5.2, 5.3, and 5.4 show that Flux takes more epochs to converge. The model built on the custom dense layer with additional momentum and gradient clipping gives the train and test accuracies of 92.90 and 91.40 percent respectively after 200 epochs. The time taken for training and evaluation is 1535 seconds. The results for the low-rank neural network models can be observed in the table 6.1.

The custom dense layer in both Python and Julia is unable to give an excellent result for the MNIST dataset. However, the Python implementation is able to do it in almost half the time as it takes only 40 epochs in comparison to the Julia implementation which takes 100 epochs. Moreover, the Flux implementation also requires the addition of momentum and gradient clipping which will need additional resources. Whereas, the models based on vanilla low-rank layers in Python are taking more than double the time than the similar models in Julia. However, the models based on the dynamical low-rank and rank-adaptive dynamical low-rank layers in Python are taking a similar time as the corresponding models in Julia. For these models based on low-rank layers, the implementation in Julia requires the addition of momentum and gradient clipping for stable evolution of the accuracies and losses.

There is an interesting response by Python and Julia towards the learning rate when working with low-rank layers. Talking about Python, a neural network model defined on the vanilla low-rank layer cannot show further improvement with additional epochs for a high learning rate value such as 0.01. However, the models defined on the dynamical low-rank and rank-adaptive dynamical low-rank layers can produce good results for the high learning rate of 0.01. On the other hand, for Julia, a neural network model defined on all of the low-rank layers cannot show further improvement with additional epochs for a high learning rate of 0.01. However, after adding momentum and gradient clipping in the training algorithm, the models show further improvement with additional epochs and reach good accuracy.

It can be observed that the Flux package is not as efficient as the Pytorch package when dealing with the SGD optimizer. However, all of the low-rank layers defined in this thesis are based on SGD while updating the parameters during training. This could be the reason behind the larger number of epochs required for converging the low-rank neural network models in Julia. Moreover, the models in Julia

need the help of additional momentum and gradient clipping to produce stable evolution of accuracies and losses.

While writing codes for this thesis, I found it easier to write in Python probably due to the availability of more resources for Python (Pytorch). Additionally, while the Flux package is useful, it is not as efficient as Pytorch when dealing with SGD. We can see that the low-rank methods in Julia take less time than those in Python, however, it requires some extra effort while writing codes in Julia. Also, the addition of momentum and gradient clipping requires extra resources, but the goal is to reduce the consumption of resources.

6.5 Performance on biochemistry data

Five neural network models were defined and used to predict the 'Rate' values for the stopped-flow data. First, four models were described in Julia. The first model was based entirely on the built-in dense layer provided by Flux.The ADAM optimizer was used during training. The second model was wholly based on the custom dense layer defined for this thesis. The third model was based on the vanilla low-rank and custom dense layers. The fourth model was based on the dynamical low-rank training layer and the custom dense layer. The rank-adaptive dynamical low-rank layer was not used to define a model because the fixed-rank approach performed better than the rank-adaptive approach. After observing that the models defined in Julia were not giving excellent results, the Pytorch model was defined to check how it would respond to the dataset.

All the models try to minimize the mean absolute error loss as we train for 2000 epochs. Though the losses of the model constructed from built-in dense layers show many spikes, these spikes are synchronized between the train and the test loss. The loss plots are more stable towards the end. This model predicts the values of the labels that are close to the actual labels. The model constructed from the custom dense layer also tries to minimize the loss over the epochs; however, the loss is so high in the beginning that the loss after 2000 epochs is also high compared to other models. This model is making the worst predictions. The predicted label values deviate significantly from the actual label values. For many test instances, predictions are negative values, which is unacceptable as the labels are chemical reaction rate constants. Though the model based on the custom dense layers produced acceptable results when trained and evaluated on the MNIST dataset, it failed severely when trained and evaluated on the stopped-flow data. This model takes more than eight times the time of the model based on the built-in

dense layer. Based on the time taken and the bad results, it is not recommended to make neural network models based entirely on the custom dense layer when dealing with stopped-flow data.

On the other hand, better results are obtained when the models based on the vanilla low-rank layers and dynamical low-rank layers are trained and evaluated on the stopped-flow data for 2000 epochs. The model based on the vanilla low-rank layer takes more than double the time of the model based on the built-in dense layer. Similarly, the model based on the dynamical low-rank layer takes more than five times the time of the model based on the built-in dense layer. This additional time is needed to perform various steps that keep the matrices low-rank. Besides, additional functionalities like momentum and gradient clipping consume extra time in the update mechanism. If we compare the results from section 5.6, the models based on these low-rank layers outperform those based on built-in dense layers. Among these two low-rank neural network models, the model based on the dynamical low-rank layer performs better. The spikes in the loss plots are smaller, and the values of the predicted labels are closer to the actual labels.

These models based on the low-rank layers require fewer computational and memory resources. We can visualize this mathematically in the same way as in the section 6.1. In this section, the values of the parameters input and output are 28000 and 512 for the input layer. So, a dense layer will have a *weight* matrix of size 512×28000 and a *bias* matrix of size 512×1. There will be a total of 14,336,512 different values that need to be stored in the matrices. For both the low-rank layers, the value of the rank parameter is 50. In the case of the vanilla low-rank layer, U, S, and V matrices have sizes 512×50, 50×50, and 50×28000. Thus, there is a total of 2,857,224 different values to be stored in the matrices. In the case of the dynamical low-rank layer, there is a total of 4,282,824 different values to be stored in the matrices. With similar calculations, the number of different values to be stored in the matrices can also be obtained for the hidden layers. So, for the first hidden layer with a value of 512 for both the input and the output parameters, the dense layer has 262,656 different values to store in the matrices. Similarly, the vanilla low-rank layer has 108,424 different values, and the dynamical low-rank layer has 159,624 values to store in the matrices. Finally, for the second hidden layer, with input and output parameter values of 512 and 256, the dense layer has 131,328 values to store in the matrices. Likewise, the vanilla low-rank layer has 82,312 values, and the dynamical low-rank layer has 120,712 values to store in the matrices. In this way, the neural network models based on low-rank layers enable us to save resources. For the stopped-flow data, the low-rank neural networks save resources and provide better performance than the built-in dense layers.

The neural network model based on Pytorch's built-in dense layers has given the best predictions after training with the ADAM optimizer for 2000 epochs. Here, the values of the predicted labels are closer to the actual labels. Moreover, the train and test losses are also quite low; however, the evolution of the losses is not smooth. The evolution of the losses shows significant fluctuations, and there are some high spikes. This model takes more than four times the time of the model based on the built-in dense layer of Flux. The results obtained from this model tell us that even the Pytorch package cannot produce excellent results for the stopped-flow data when a neural network model is designed from the fully connected, dense layers. We are training for 2000 epochs for all the models, which is substantial. We can show that we can reduce resource consumption using low-rank neural networks. However, we must accept that the dense and low-rank layers developed in this thesis cannot produce good results for the stopped-flow data within a reasonable number of epochs.

Chapter 7

Conclusion

This thesis explored the implementation and performance of low-rank methods in neural networks using the Julia programming language. First, the vanilla training and the DLRT algorithm were implemented and evaluated on the MNIST dataset. Then, the results were compared to the ones obtained using the Python programming language. At last, the low-rank neural networks were used to predict the rate constant for the biochemistry dataset. Some interesting results were obtained that demonstrate how the Julia programming language performs for low-rank methods.

It is observed that the Julia Flux package is a viable option to efficiently implement low-rank neural networks. However, only the SGD optimizer can be adapted while defining these low-rank layers. The implementation of the vanilla training and the DLRT algorithm demonstrated significant savings in memory resources. Vanilla low-rank neural networks can even save some time required for training and evaluation. Momentum and gradient clipping need to be added to the training algorithm to make the evolution of the losses and accuracies smoother with lower fluctuations. Besides, higher values of accuracy can be achieved with this addition.

The comparison between the fixed-rank and rank-adaptive approaches to implement the DLRT algorithm shows that it is better to implement the fixed-rank approach as it gives better results while consuming fewer memory resources. Although the rank-adaptive approach can adjust the rank during the training process, it cannot produce results as good as those produced by the fixed-rank approach. Additionally, the neural networks based on the dynamical low-rank layer (fixed-rank) can produce better results than those based on the vanilla low-rank layer. Similarly, the comparison between the performance of Flux and Pytorch also demonstrated some interesting results. The neural network based on the vanilla

low-rank layer performs better with Flux. It can get higher accuracies by taking quite less time. Whereas, in the neural networks based on DLRT algorithms, for both the fixed-rank and the rank-adaptive approaches, Flux cannot save much time in comparison to Pytorch. This is because the low-rank neural networks take more epochs to converge with Flux. After adding momentum and gradient clipping to the training algorithm, the low-rank layers in Flux can produce accuracy similar to the corresponding low-rank layers in Pytorch.

Finally, the application of the low-rank neural networks to the biochemistry dataset revealed promising results. It demonstrated that low-rank neural networks can effectively be applied to complex real-world data to get results similar to full-rank neural networks based on built-in dense layers while saving significant memory resources. In Julia, the low-rank neural networks can produce even better results than the full-rank neural networks. However, the predicted values were still not so accurate when compared to the actual labels.

Overall, the findings of this thesis bring meaningful insights regarding the performance of low-rank methods in the Julia programming language for optimizing neural network training.

Chapter 8

Limitations and Future Work

8.1 Limitations

This thesis has a few limitations which are listed below:

1. Only two datasets are used. To study the behavior of the low-rank methods in Julia, the MNIST dataset is used for training and evaluation. Additionally, the stopped-flow biochemistry data is used to demonstrate the application of the low-rank methods.

2. The application of the low-rank methods on the stopped-flow biochemistry dataset requires 2000 epochs. It is probably because this data is a time-series data.

3. The addition of momentum and gradient clipping in the training algorithm consumes extra memory resources. Specifically, momentum requires the creation of extra matrices.

4. The evaluation metrics used are only accuracy and loss for the MNIST dataset and only loss for the stopped-flow biochemistry dataset.

8.2 Future Work

In this thesis, the low-rank neural networks are trained and evaluated on only the MNIST dataset. A better generalization of the low-rank methods in Julia can be obtained by training and evaluating other standard datasets like the ImageNet1K, and Cifar10 dataset for classification problems, and the IMDB-WIKI, and the Protein Structure dataset for regression problems. Also to show the application of the low-rank methods in Julia, some other complex real-world datasets for both classification and regression problems can be used. Also, in this thesis, only the CPU has been used for training and evaluation. The use of hardware acceleration through GPU can also give us valuable insights.

Since momentum requires extra memory resources, it would be good to find some other optimization techniques that consume less memory resources than momentum. Moreover, new evaluation metrics can be explored apart from the loss and accuracy. Additionally, a wider range of configurations and parameters can be explored to increase the generalizability of the algorithm. Finally, the toughest future work would be to implement the low-rank methods for RNNs so that time series data can be handled easily.

Bibliography

[1] Z. Huang and N. Wang, 'Data-driven sparse structure selection for deep neural networks,' *CoRR*, vol. abs/1707.01213, 2017. arXiv: 1707.01213. [Online]. Available: http://arxiv.org/abs/1707.01213.

[2] E. Denton, W. Zaremba, J. Bruna, Y. LeCun and R. Fergus, 'Exploiting linear structure within convolutional networks for efficient evaluation,' *CoRR*, vol. abs/1404.0736, 2014. arXiv: 1404.0736. [Online]. Available: http://arxiv.org/abs/1404.0736.

[3] S. Han, J. Pool, J. Tran and W. J. Dally, 'Learning both weights and connections for efficient neural networks,' *CoRR*, vol. abs/1506.02626, 2015. arXiv: 1506.02626. [Online]. Available: http://arxiv.org/abs/1506.02626.

[4] D. Blalock, J. J. G. Ortiz, J. Frankle and J. Guttag, 'What is the state of neural network pruning?' *CoRR*, vol. abs/2003.03033, 2020. arXiv: 2003.03033. [Online]. Available: https://arxiv.org/abs/2003.03033.

[5] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, 'Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1,' *CoRR*, vol. abs/1602.02830, 2016. arXiv: 1602.02830. [Online]. Available: http://arxiv.org/abs/1602.02830.

[6] R. Feng, K. Zheng, Y. Huang, D. Zhao, M. Jordan and Z.-J. Zha, 'Rank diminishing in deep neural networks,' eng, *Advances in Neural Information Processing Systems*, vol. 33054-33065, p. 35, 2022.

[7] H. Wang, S. Agarwal and D. Papailiopoulos, 'Pufferfish: Communication-efficient models at no extra cost,' *ArXiv*, vol. abs/2103.03936, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232148049.

[8] G. Ceruti, J. Kusch and C. Lubich, 'A rank-adaptive robust integrator for dynamical low-rank approximation,' eng, *BIT*, vol. 62, no. 4, pp. 1149–1174, 2022, ISSN: 1572-9125.

[9] S. Schotthöfer, E. Zangrando, J. Kusch, G. Ceruti and F. Tudisco, 'Low-rank lottery tickets: Finding efficient low-rank neural networks via matrix differential equations,' in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 20 051–20 063. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/7e98b00eeafcdaeb0c5661fb9355be3a-Paper-Conference.pdf.

[10] Joseph, *Julia flux vs pytorch: Which is the best ai framework?* Published on August 16, 2022. Accessed on Nov 3, 2023, 2022. [Online]. Available: https://reason.town/julia-flux-vs-pytorch/.

[11] J. Bezanson, S. Karpinski, V. B. Shah and A. Edelman, 'Julia: A fast dynamic language for technical computing,' *CoRR*, vol. abs/1209.5145, 2012. arXiv: 1209.5145. [Online]. Available: http://arxiv.org/abs/1209.5145.

[12] B. L. Center, *What are stopped-flows and how do they help chemists and biochemists understand high-speed chemical reactions?* Accessed on December 13, 2023, Jun. 2023. [Online]. Available: https://www.biologic.net/topics/what-are-stopped-flows-and-how-do-they-help-chemists-and-biochemists-understand-high-speed-chemical-reactions/.

[13] S. Raschka and V. Mirjalili, *Python Machine Learning*, 3rd ed. Birmingham, UK: Packt Publishing, 2019.

[14] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 1st ed. O'Reilly Media, 2017.

[16] D. C. Lay, S. R. Lay and J. J. McDonald, *Linear Algebra and its Applications*, 6th ed. New York, NY: Pearson, 2021.

[17] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*, Illustrated. USA: The MIT Press, 2019.

[18] J. Stewart, *Calculus*, 8th ed. USA: Cengage Learning, 2015.

[19] G. Strang, *Introduction to Linear Algebra*, 5th ed. Wellesley, MA: Wellesley-Cambridge Press, 2016. [Online]. Available: https://math.mit.edu/~gs/linearalgebra/ila5/indexila5.html.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Johns Hopkins University Press, 2013.

[21] S. Ruder, 'An overview of gradient descent optimization algorithms,' *CoRR*, vol. abs/1609.04747, 2016. arXiv: 1609.04747. [Online]. Available: http://arxiv.org/abs/1609.04747.

[22] N. Qian, 'On the momentum term in gradient descent learning algorithms,' *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999, ISSN: 1879-2782. DOI: 10.1016/s0893-6080(98)00116-6. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/12662723.

[23] Y. Cheng, D. Wang, P. Zhou and T. Zhang, 'A survey of model compression and acceleration for deep neural networks,' *CoRR*, vol. abs/1710.09282, 2017. arXiv: 1710.09282. [Online]. Available: http://arxiv.org/abs/1710.09282.

[24] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy and B. Ramabhadran, 'Low-rank matrix factorization for deep neural network training with high-dimensional output targets,' in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6655–6659. DOI: 10.1109/ICASSP.2013.6638949.

[25] M. Khodak, N. A. Tenenholtz, L. W. Mackey and N. Fusi, 'Initialization and regularization of factorized neural layers,' *ArXiv*, vol. abs/2105.01029, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233481638.

[26] M. Jaderberg, A. Vedaldi and A. Zisserman, 'Speeding up convolutional neural networks with low rank expansions,' *CoRR*, vol. abs/1405.3866, 2014. arXiv: 1405.3866. [Online]. Available: http://arxiv.org/abs/1405.3866.

[27] Y. A. Ioannou, D. P. Robertson, J. Shotton, R. Cipolla and A. Criminisi, 'Training cnns with low-rank filters for efficient image classification,' *CoRR*, vol. abs/1511.06744, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:11130812.

[28] C. Li and C.-J. R. Shi, 'Constrained optimization based low-rank approximation of deep neural networks,' in *European Conference on Computer Vision*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52951765.

[29] Y. Idelbayev and M. Á. Carreira-Perpiñán, 'Low-rank compression of neural nets: Learning the rank of each layer,' *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8046–8056, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:215716065.

[30] *Julia documentation - the julia language*, https://docs.julialang.org/en/v1/, Accessed on December 14, 2023.

[31] J. Bezanson, A. Edelman, S. Karpinski and V. B. Shah, 'Julia: A fresh approach to numerical computing,' *CoRR*, vol. abs/1411.1607, 2014. arXiv: 1411.1607. [Online]. Available: http://arxiv.org/abs/1411.1607.

[32] J. Bezanson, S. Karpinski, V. B. Shah and A. Edelman, *Why we created julia*, Published on February 14, 2012. Accessed on December 12, 2023, 2012. [Online]. Available: https://julialang.org/blog/2012/02/why-we-created-julia/.

[33] M. Innes, 'Flux: Elegant machine learning with julia,' *J. Open Source Softw.*, vol. 3, no. 25, p. 602, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:53244078.

[34] *Flux: The julia machine learning library*, https://fluxml.ai/Flux.jl/stable/, Accessed on April 21, 2024.

[35] M. Innes, 'Don't unroll adjoint: Differentiating ssa-form programs,' *CoRR*, vol. abs/1810.07951, 2018. arXiv: 1810.07951. [Online]. Available: http://arxiv.org/abs/1810.07951.

[36] G. Ceruti and C. Lubich, 'An unconventional robust integrator for dynamical low-rank approximation,' *Bit Numerical Mathematics*, vol. 62, pp. 23–44, 2022. DOI: 10.1007/s10543-021-00873-0.

[37] Y. LeCun and C. Cortes, *The mnist database of handwritten digits*, https://api.semanticscholar.org/CorpusID:60282629, 2005.

[38] F. Ren, Y. Xu, H. Yang, J. Zhang and C. Lin, 'Frequency domain packet scheduling with stability analysis for 3gpp lte uplink,' *IEEE Transactions on Mobile Computing*, vol. 12, no. 12, pp. 2412–2426, 2013. DOI: 10.1109/TMC.2012.223.

[39] O. Golten, *Personal communication on low rank methods for stopped flow data*, Assisting in the bio-chemical aspect, Oct. 2023.

[40] *Amd ryzen 7 4700u performance review — benchmark*, https://laptopsreviewer.com/product/amd-ryzen-7-4700u/, Accessed on December 13, 2023.

[41] *Releases · julialang/julia*, https://github.com/JuliaLang/julia/releases, Accessed on March 13, 2023.

[42] A. Zhang, Z. C. Lipton, M. Li and A. J. Smola, *Dive into Deep Learning*. Online, 2020, ch. 6, Accessed: 2024-05-14, Chapter 6. [Online]. Available: https://d2l.ai/chapter_builders-guide/.

Appendix A

Evolution of Ranks in Rank-adaptive Approach

A.1 Maximum Rank 40

A.1.1 Learning rate of 0.01

19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19

19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19

19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19

19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19

19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19

19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19
19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 38 38
19 19 19 19 38 38 19 19 19 19 38 38 19 19 19 19 19 19

A.1.2 Learning rate of 0.001

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40

20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40
20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20
20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40	20	20	20	20	40	40

20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20
20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 40 40
20 20 20 20 40 40 20 20 20 20 40 40 20 20 20 20 20 20

A.2 Maximum Rank 50

A.2.1 Learning Rate of 0.001

25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48

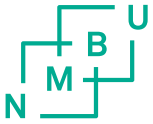
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48

25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48

25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48

25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48

25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25
24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 50 48
25 24 25 24 50 48 25 24 25 24 50 48 25 24 25 24 25 24



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway