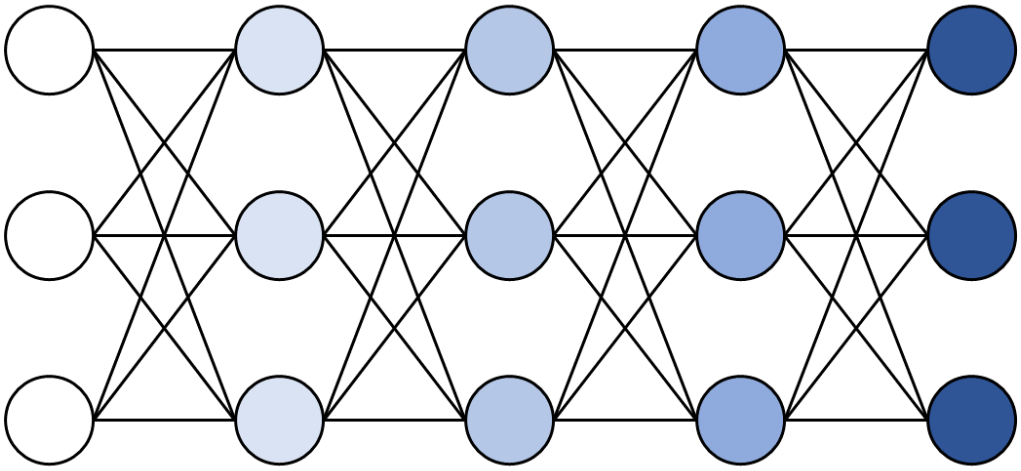


Unstable gradients in deep neural nets

A numerical study on the vanishing gradient problem using dynamical systems theory

Master's thesis in Complex Adaptive Systems

LUDVIG STORM

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Unstable gradients in deep neural nets

A numerical study on the vanishing gradient problem using dynamical systems theory

LUDVIG STORM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Unstable gradients in deep neural nets
A numerical study on the vanishing gradient problem using dynamical systems theory
LUDVIG STORM

Master's Thesis 2020
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2020

Unstable Gradients in Deep Neural Nets
A numerical study on the vanishing gradient problem using dynamical systems theory
LUDVIG STORM
Department of Physics
Chalmers University of Technology

Abstract

In the past decade, deep learning algorithms have gained increased popularity due to their ability to detect and represent abstract features in complex data sets. One of the most prominent deep learning algorithms is the deep neural network, having managed to outperform many state-of-the-art machine learning techniques. While its success can largely be attributes to its depth, this feature also causes it to be difficult to train. One of the main obstacles is the vanishing gradient problem; a phenomenon causing updates to the network to exponentially vanish with depth. The problem is severe enough to have been referred to as a fundamental problem of deep learning [18]. However, simulations reveal that DNNs are able to escape the vanishing gradient problem after having been trained for some time, but the dynamics of this escape are still not understood.

In this work, the underlying dynamics of the escape from the vanishing gradient problem in deep neural networks is explored by means of dynamical systems theory. In particular, the concept of Lyapunov exponents is used to analyse how signals propagating through the network evolve, and whether this has a connection to the vanishing gradient problem. The study is based on results by [16] and [19]. Furthermore, a method to circumvent the vanishing gradient problem, developed in [14] for very wide neural networks, is explored for narrow networks.

The results of this thesis suggest the escape from the vanishing gradient problem is unrelated to what data set the deep neural network is trained on, but is rather a consequence of the training algorithm. Furthermore, it is found that the escape is characterised by the maximal Lyapunov exponent of the network growing from a negative value to a value close to 0. To further explore the underlying dynamics, it is suggested to study the training algorithms in the absence of data. The method of avoiding the vanishing gradient problem, presented by [14], is found to work poorly for narrow neural networks.

Keywords: Vanishing gradient, dynamical system, Lyapunov exponent, dynamical isometry

Acknowledgements

I would like to thank my supervisor and examiner Bernhard Mehlig for the useful discussions and help in figuring out which direction to take with the thesis work, and for taking on this project. I would also like to thank Jan Meibohm for the useful tips on how to interpret the results.

Ludvig Storm, Gothenburg, February 2020

Contents

List of Figures

1

Introduction

1.1 Background

In the past decade, deep learning has gained an increasing popularity in science and engineering due to the upswing in available computer power in recent years (e.g. cheap and powerful GPUs) [6][13][21], and its ability to find and represent highly abstract features in complex data sets, unaided by pre-engineered feature extraction methods necessary for standard machine-learning algorithms [11][6][10][2][21]. Furthermore, deep learning methods have proven to be good at generalisation; the ability for a statistical model to perform well when presented with data not used during training [22]. Due to the ability to handle complex data sets, deep learning methods have found applications in a vast number of fields, including speech recognition, object detection, drug discovery, genomics, sensors, stock market prediction, health care, and remote sensing [23][4][13][17][10]. The backbone of many deep learning methods is the artificial neural network (ANN); a computational structure inspired by biological neuronal networks, consisting of a network of interconnected "neurons" (i.e. simple computational units). By adjusting the connections between the neurons, and altering each neuron's intrinsic bias, an artificial neural network can be trained to complete complex tasks by giving desired output responses to some given inputs. These inputs could, for example, be images of animals, and the output would identify which animal is being portrayed. While a large variety of network architectures exist, by far the most common type is the fully connected, feed-forward architecture. This is because it is present in other architectures, such as convolutional neural networks (CNN) and recurrent neural networks (RNN). The fully connected, feed-forward network consists of a number of layers of neurons, each layer's neurons being connected with the consecutive layer's neurons. If such a network contains more than one intermediate (hidden) layer between the input layer and output layer, it is referred to as a deep neural network (DNN). The terminology reflects the network's ability to identify more abstract (deep) features in a data set than their shallow counterpart.

A problem with DNNs that has been recognised ever since the proposal of such computational structures, is that when more layers are added, the more difficult training becomes [10][11]. The standard procedure of training a neural network is to define an error function which reaches its global minimum once each input yields the correct output. Hence, minimising the error function is tantamount to training the network. Most commonly, a gradient descent algorithm is employed for this

task, as it is efficient and computationally cheap. For each training step, the parameters in each layer is updated such that the point mapped by the error function in parameter space moves in the direction of steepest descent. The updates begin in the final layer and propagate backwards. However, for every layer, the updates become exponentially smaller. This phenomenon is referred to as the vanishing gradient problem (VGP), and it has been referred to as a fundamental problem in deep learning [18].

The success of deep machine learning lies in that DNNs may escape the VGP phase after some training steps. The dynamics of this escape are not fully understood and are being extensively researched [7][16][14][19][9]. By understanding the dynamics, algorithms may be designed to speed up the escape, making training DNNs faster. Training a network through gradient descent amounts to calculating the Jacobian of the network's output with respect to each layer. An approach to studying the VGP and its dynamics is, therefore, to focus on the properties of the Jacobian. In this work, numerical simulations of DNNs are performed to understand the dynamics of how DNNs escape the VGP phase, by means of dynamical systems theory.

1.2 Purpose

To investigate, through numerical simulations, the dynamics of how deep neural networks may escape the vanishing gradient problem, by means of dynamical systems theory.

1.3 Scope

The DNNs considered in this work are randomly initialised and are restricted to the fully connected, feed-forward architecture. Furthermore, the hidden layers will contain the same number of neurons to make the dynamics of data vectors propagating through the network easier to interpret. The thesis will be restricted to treating the vanishing gradient problem, and will not consider its counterpart, the exploding gradient problem, where the updates become exponentially larger per layer. Additionally, only the *tanh* activation function is considered.

1.4 Layout

Here, the structure of the thesis is presented. The first section will cover the theory of DNNs, the vanishing gradient problem, and dynamical systems theory. Then, results from previous works which this thesis's result builds on is summarised. This is followed by a method section, where the experiments conducted are detailed, where the concept of normalised average directionality is introduced. The results are then presented together with a toy model derived from the updating rules for training a DNN, which displays dynamics similar to the escape mechanism. This is followed by a discussion and a conclusion, where further studies are suggested.

2

Theory

2.1 Deep neural networks

An artificial neural network (ANN) is a computational structure inspired by biological neuronal networks. It consists of a collection of connected, simple computational units called neurons, after their biological counterparts. The neurons take inputs from surrounding neurons and fires off a signal if the accumulated input exceeds an intrinsic threshold. Among the most common realisations of ANNs is the fully connected, feedforward architecture, which will be the only architecture considered in this thesis. Here, the neurons are arranged in consecutive layers, where each neuron in layer l is connected to the neurons in layer $l+1$ (see Figure 2.1). The number of neurons in layer l is denoted as N_l. The strength of the connections is determined by real numbers called weights. A signal enters the network through the input layer and propagates forward through the hidden layers until it reaches the output layer. A network consisting of more than one hidden layer is referred to as a deep neural network (DNN). Mathematically, a DNN is a highly nonlinear function which takes an N_0-dimensional input vector $\mathbf{x}^0$ and feeds it through $L > 1$ layers following the dynamics

$$\mathbf{h}^l = \mathbf{W}^l \mathbf{x}^{l-1} + \mathbf{b}^l, \qquad \mathbf{x}^l = g(\mathbf{h}^l), \qquad l = 1, 2 \ldots, L, \tag{2.1}$$

to the final N_L-dimensional output layer L. Here, $\mathbf{W}^l$ is an $N_l \times N_{l-1}$ weight matrix, where the entry $W^l_{ij} \in \mathbb{R}$ models the strength of the connection from the j:th neuron in layer $l-1$ to the i:th neuron in layer l. $\mathbf{b}^l$ is an N_l-dimensional bias vector which

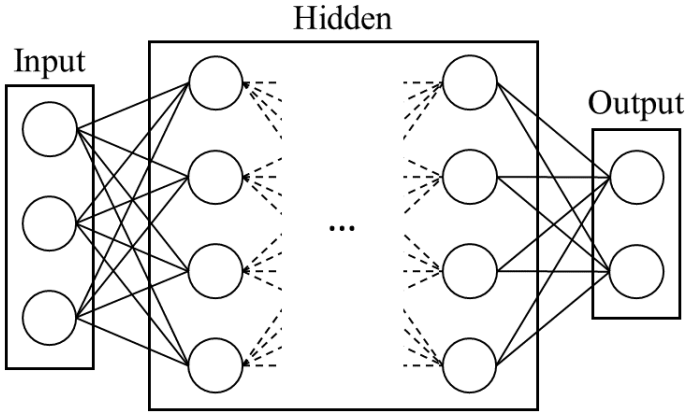


Figure 2.1: Schematic of a fully connected, feedforward neural network.

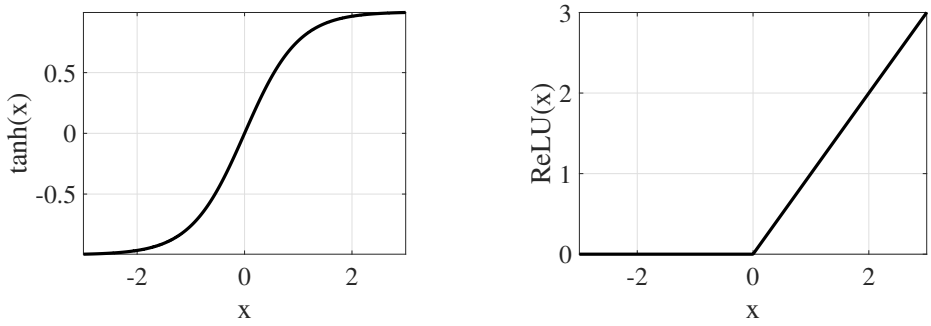


Figure 2.2: Activation functions used in ANNs.

biases the input to the neuron to be either positive or negative. From Equation 2.1, each neuron can be seen as a linear regressor, and from this perspective, the $\mathbf{W}^l$ is the slope and $\mathbf{b}^l$ is the intercept. Finally, $g(\cdot)$ is called the activation function and models the response of a neuron given the inputs from the neurons in the preceding layer. The activation function of a DNN is crucially nonlinear, as setting it to a linear function would effectively reduce its depth to unity. This can be shown using the properties of linearity on Equation 2.1 (see Appendix A). The choice of activation function is important for the learning of a DNN, and a lot of research has been dedicated to the design of these [8][20][1]. Activation functions that have frequently been employed are the hyperbolic tangent (*tanh*) and the rectified linear unit (ReLU), see Figure 2.2. The *width* of a layer is the number of neurons it contains, whereas the *depth* of a neural network is its number of computational layers. As no computation occurs in the input layer, it is excluded from the definition of depth.

When initialising a DNN, the weight matrices and biases must be chosen. This is often done randomly, drawing weights and biases from a normal distribution with a mean of 0 and a variance σ_W^2/N_l and σ_b^2, respectively. The division by N_l is introduced as a way to normalise the input from the previous layer, which is a sum of N_l numbers multiplied by N_l random weights. Hence, the division ensures the input has a variance of σ_W^2. The initialisation of weights and biases can have a major impact on the training performance of a DNN and, as will be explained further, is connected to the VGP.

2.2 Supervised learning and gradient descent

A neural network can be trained to complete a task by using training examples where a particular input has been assigned a desired output. An example of such supervised learning is the recognition of written digits. Sending an image of a written digit into the network, the desired output is for the network to identify which digit the image depicts.

When training a neural network, its weights and biases are adjusted until the network can perform a desired task. The most common approach is to introduce an

error function H which takes the actual output of the network and the desired output, and computes an error based on the average discrepancy between them, such that its global minimum is reached when the error vanishes. Thus, the network learns by minimising the energy function with respect to weights and biases. While the choice of energy function can be important for training a neural network, it is not directly related to the main issue investigated in this thesis, and the results presented are expected to be valid for any choice of error function.

The error function is minimised using some minimisation algorithm, where the most common is the stochastic gradient descent (SGD) algorithm. In this algorithm, the weights and biases are adjusted such that the point mapped by the error function in parameter space travels in the direction of greatest descent:

$$\mathbf{W}^l \to \mathbf{W}^l - \eta \nabla^l_{\mathbf{W}} H, \qquad \mathbf{b}^l \to \mathbf{b}^l - \eta \nabla^l_{\mathbf{b}} H. \tag{2.2}$$

Here, η is the *learning rate* and determines the step size of the weight and bias updates. Calculating the gradients for a generic energy function for $L > 1$ yields

$$\nabla^l_{\mathbf{W}} H = \boldsymbol{\varepsilon}_L \left(\prod_{k=L}^{l+1} \mathbf{D}^k \mathbf{W}^k \right) \mathbf{D}^l \otimes \mathbf{x}^{l-1}, \qquad \nabla^l_{\mathbf{b}} H = \boldsymbol{\varepsilon}_L \left(\prod_{k=L}^{l+1} \mathbf{D}^k \mathbf{W}^k \right) \mathbf{D}^l, \qquad l < L-1 \tag{2.3}$$

where $\mathbf{D}^l$ is a diagonal matrix with entries $D^l_{ij} = g'(h^l_i)\delta_{ij}$. Furthermore, $\boldsymbol{\varepsilon}_L = \frac{\partial H}{\partial \mathbf{x}^L}$ will be referred to as the error vector. For $l = L$, the same equations hold with the product in the paranetheses removed. The product in parentheses present in both expressions in Equation 2.3 is equivalent to the Jacobian of the output layer with respect to the output of layer l, and will be referred to as the input-output Jacobian

$$\mathbf{J}_{m,n} = \frac{\partial \mathbf{x}^m}{\partial \mathbf{x}^n} = \prod_{k=m}^{n+1} \mathbf{D}^k \mathbf{W}^k, \qquad m > n. \tag{2.4}$$

Clearly, the input-output Jacobian is closely related to the learning dynamics of a DNN. Additionally, the Jacobian is a linear approximation of how a vector changes as is passes from layer n to m, and therefore describes how small perturbations to an input vector grows or shrinks. Finally, multiplying $\mathbf{J}_{L,l}$ from the left with $\frac{\partial H}{\partial \mathbf{x}^L}$ describes how the error vector evolves as it propagates through the network.

2.3 Vanishing gradient problem

The VGP arises in DNNs as a consequence of the depth of the network. As the error vector propagates backwards through the network, it may shrink exponentially, causing the gradients in Equation 2.3 to vanish as it approaches the early layers of the network. To see how a vanishing gradient could occur, consider a unit vector $\mathbf{v}$ measured with the L2-norm and introduce the induced matrix 2-norm,

$$\|\mathbf{A}\|_2 \equiv \max_{\|\mathbf{v}\|_2 = 1} \|\mathbf{A}\mathbf{v}\|_2$$

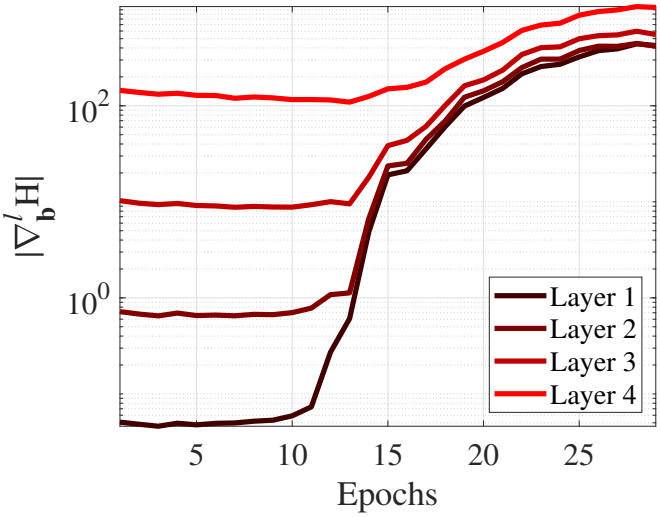


Figure 2.3: Training dynamics of a DNN with 4 hidden layers using tanh activation functions, $\eta = 0.003$ and $\sigma_W = 0.005$, trained on the `MNIST` data set with a batch size of 10.

It can be shown that this norm is equivalent to the maximal singular value of the matrix. Hence, the following inequality can be obtained [5]

$$\|\mathbf{v}\prod_{k=L}^{l+1}\mathbf{D}^k\mathbf{W}^k\|_2 \leq \|\mathbf{v}\|_2\prod_{k=L}^{l+1}\left\|\mathbf{D}^k\right\|_2\left\|\mathbf{W}^k\right\|_2 = \prod_{k=L}^{l+1}\sigma_{max}\left(\mathbf{D}^k\right)\sigma_{max}\left(\mathbf{W}^k\right)$$

where σ_{max} is the maximal singular value of the given matrix. Consider a DNN employing the sigmoid activation function. As the maximal derivative of the sigmoid function is 0.25, the maximal singular value obtained from the $\mathbf{D}^k$ matrices is 0.25 as well. Furthermore, if the variance of the weight matrices has been normalised, the probability of its maximal singular value exceeding unity is small. Thus, the norm of the input vector is expected to decrease exponentially per layer. While other activation functions, such as *tanh* or ReLU, has a maximal derivative of 1, the VGP can still be observed. In Figure 2.3, the effect of the vanishing gradient can be observed in the first 10 training epochs of training a DNN with a depth of 5, using a *tanh* activation function. In the figure, the norm of the updating step to each layer has been plotted. A low weight variance has deliberately been chosen to showcase the effect of the VGP. Ultimately, the aim of the thesis is not to find a solution to the problem, but to understand its dynamics.

2.4 Finite time Lyapunov exponents

The dynamics of the forward propagation of an input vector through the network in Equation 2.1 is naturally interpreted as a dynamical system. Furthermore, the gradient shown in Equation 2.3 can be interpreted as a dynamical system, where the error vector $\boldsymbol{\varepsilon}_L$ propagates backwards from the final layer through the dynamics

$$\boldsymbol{\varepsilon}_{l-1} = \boldsymbol{\varepsilon}_l\mathbf{D}^l\mathbf{W}^l,$$

starting from the initial state $l_0 = L$. Thus, it makes sense to study the VGP using the finite time Lyapunov exponents (FTLE); a quantity frequently employed in dynamical systems theory. Given an initial state vector $\mathbf{x}$ and an infinitesimally perturbed state vector $\mathbf{x} + \delta\mathbf{x}$, the FTLE describes how the perturbation changes in magnitude as the dynamical system evolves. Assuming the states evolve through the function $f^l(\mathbf{x})$, where the superscript l denotes how many steps the system has taken, one may write

$$\delta\mathbf{x}_l = f^l(\mathbf{x} + \delta\mathbf{x}) - f^l(\mathbf{x}) = \frac{df^l(\mathbf{x})}{d\mathbf{x}}\delta\mathbf{x}_0 + \mathcal{O}(\|\delta\mathbf{x}_0\|^2)$$

To put the equation in the context of neural networks, we may take the derivative in the above equation to be the input-output Jacobian in Equation 2.4. As the perturbation is infinitesimal, the second order term can be ignored, and the L2-norm of the perturbation is found to be

$$\|\delta\mathbf{x}_l\|_2 = \sqrt{\langle \mathbf{J}_{l,0}\delta\mathbf{x}_0, \mathbf{J}_{l,0}\delta\mathbf{x}_0\rangle} = \sqrt{\langle \delta\mathbf{x}_0, \mathbf{J}_{l,0}^T\mathbf{J}_{l,0}\delta\mathbf{x}_0\rangle}.$$

Here, $\mathbf{J}_{l,0}^T\mathbf{J}_{l,0}$ is known as the right Cauchy-Green tensor and describes how a perturbation stretches or shrinks as the system evolves. To find the largest change imposed by the system, one may choose the perturbation $\delta\mathbf{x}_0$ to be the maximal eigenvector of the tensor. This yields

$$\max \|\delta\mathbf{x}_l\|_2 = \sqrt{\langle \delta\mathbf{x}_0, \lambda_{max}\delta\mathbf{x}_0\rangle} = \sqrt{\lambda_{max}}\|\delta\mathbf{x}_0\|_2.$$

From this, the maximal FTLE is defined as

$$\Lambda = \frac{1}{l}\ln\sqrt{\lambda_{max}},$$

such that the above equation may be rewritten as

$$\max \|\delta\mathbf{x}_l\|_2 = e^{\Lambda l}\|\delta\mathbf{x}_0\|_2.$$

From this expression, the interpretation of the FTLE becomes clear: a positive exponent results in an exponential growth of the separation, whereas a negative exponent causes exponential convergence.

To calculate the FTLE of the DNNs considered in this work, the method of QR decomposition, as presented in [3], will be employed. Here, the product in Equation 2.4 is calculated as follows: The first matrix in the product from the right is QR decomposed,

$$\mathbf{Q}_1\mathbf{R}_1 = \mathbf{W}^1.$$

Here, $\mathbf{Q}_1$ is an orthogonal matrix, and $\mathbf{R}_1$ is an upper triangular matrix. Then, the second matrix in the product is multiplied with the $\mathbf{Q}$ matrix, and the product is QR decomposed so that the new product forms

$$\mathbf{Q}_2\mathbf{R}_2\mathbf{R}_1 = \mathbf{D}^1\mathbf{W}^1.$$

This procedure continues until the entire product $\mathbf{J}_{L-1,1}$ is formed. The reason the 0:th and L:th layers are not included is because this choice makes $\mathbf{J}$ quadratic, which is easier to interpret. In the end, one obtains

$$\mathbf{J}_{L-1,1} = \mathbf{Q}_{L-1}\mathbf{R}_{L-1}\mathbf{R}_{L-2}\dots\mathbf{R}_1.$$

From [3], it is found that the FTLEs are found from

$$\Lambda_i = \frac{1}{L-1}\sum_{j=1}^{L-1}\log(\mathbf{R}_j)_{ii}, \quad i = 1,\dots,N.$$

2.5 Previous work

The experiments performed in this work build on the works in [16], [14], and [19]. A summary of their results is provided in this section. Common for the articles is to study DNNs in the limit of $N \to \infty$.

2.5.1 Poole et al.

A mean field approximation for the dynamics of a DNN is developed in [16]. Consider an input vector $\mathbf{x}^{0,a}$ and define the normalised square length as

$$q^l_{aa} = \frac{1}{N_l}\sum_{i=1}^{N_l}(\mathbf{h}^l_i)^2,$$

where the subscript of q^l_{aa} will be explained shortly. It is found that in the limit of $N_l \to \infty$, the normalised square length will develop as

$$q^l_{aa} = \sigma_w^2 \int \mathcal{D}y\; g\left(\sqrt{q^{l-1}_{aa}}y\right)^2 + \sigma_b^2, \quad \text{for} \quad l = 2,\dots,L \tag{2.5}$$

where $\mathcal{D}y = \frac{dy}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}$ so that the result is averaged over the standard Gaussian measure, and $q^0_{aa} = \frac{1}{N_0}\mathbf{x}^{0,a}\cdot\mathbf{x}^{0,a}$. For different parameters σ_w and σ_b, the iterative map in Equation 2.5 will approach some fixed point q^* such that

$$q^* = \sigma_w^2 \int \mathcal{D}y\; g\left(\sqrt{q^*}y\right)^2 + \sigma_b^2.$$

Furthermore, it is found that if two random input vectors $\mathbf{x}^{0,a}$ and $\mathbf{x}^{0,b}$ are sent through the network, the covariance between them will develop according to

$$q^l_{ab} = \sigma_w^2 \int \mathcal{D}y_a\mathcal{D}y_b\; g\left(u_a\right) g\left(u_b\right) + \sigma_b^2 \tag{2.6}$$

where $u_a = \sqrt{q^{l-1}_{aa}}y_a$ and $u_b = \sqrt{q^{l-1}_{bb}}\left[c^{l-1}_{ab}y_a + \sqrt{1-(c^{l-1}_{ab})^2}y_b\right]$, and $c^l_{ab} = q^l_{ab}/\sqrt{q^l_{aa}q^l_{bb}}$ is the correlation between the vectors. Assuming the normalised square length of $\mathbf{x}^{l,a}$ and $\mathbf{x}^{l,b}$ has converged (i.e. $q^l_{aa} = q^l_{bb} = q^*$), we may write $c^l_{ab} = q^l_{ab}(q^*)^{-1}$. This

correlation map has a fixed point at $c^* = 1$, but to determine whether it is stable or not, one must compute the derivative of the map at the fixed point,

$$\chi = \left.\frac{\partial c_{ab}^l}{\partial c_{ab}^{l-1}}\right|_{c=1} = \sigma_w^2 \int \mathcal{D}y \left[g'\left(\sqrt{q^*}y\right)\right]^2.$$

if $\chi < 1$, the fixed point is stable, making two input vectors converge and become fully correlated after some layers. If $\chi > 1$, the point is unstable and the input vectors diverge to become completely de-correlated. As it turns out, χ is closely related to the Jacobian in Equation 2.4. By averaging $\|\mathbf{J}_{l+1,l}\mathbf{u}\|_2/\|\mathbf{u}\|_2$ over random perturbations $\mathbf{u}$ and random initialisations of weights and biases, where $\mathbf{u}$ is some small perturbation to $\mathbf{h}_i^l$, χ is obtained. The quantity is also closely related to the FTLE of the DNN. Essentially, χ reflects how much a random vector shrinks or grows as it passes from layer to layer, meaning that $\chi < 1$ is a direct representation of the VGP. Finally, the case where $\chi = 1$ represents a phase transition between vanishing and exploding gradients and is characterised by that the average singular value of the input-output Jacobian is equal to unity. This idea is explored further in [14].

2.5.2 Pennington et al.

The work in [14] builds on the results of [16]. Here, the goal is to find a way to initialise a DNN so that $\chi = 1$. However, rather than only finding an initialisation such that the *average* singular value of the input-output Jacobian is equal to unity, they employ random matrix theory to try to constrain the *entire* singular value distribution around unity. Doing so would ensure that an input vector maintains its magnitude as is passes through the network. This condition is termed *dynamical isometry*. To constrain the singular values around unity, the spectral eigenvalue density of a matrix is defined as

$$\rho(\lambda) \equiv \left\langle \frac{1}{N}\sum_{i=1}^{N}\delta(\lambda - \lambda_i)\right\rangle$$

where the brackets denote an average over randomly initialised matrices. The Stieltjes transform of a spectral density is

$$G(z) \equiv \int_{\mathbb{R}} \frac{\rho(y)}{z-y}dy, \quad z \in \mathbb{C}\backslash\mathbb{R},$$

and to retrieve the original density, the inverse operation is

$$\rho(\lambda) - \frac{1}{\pi}\lim_{\varepsilon\to 0^+} \text{Im } G(\lambda + i\varepsilon).$$

It was found that for a DNN with a width $N \to \infty$ and weight matrices initialised from a Gaussian distribution, the Stieltjes transform of $\mathbf{J}_{L,0}^T\mathbf{J}_{L,0}$ obeys the relation

$$\sigma_w^{2L}G(Gz + p(q^*) - 1)^L - (Gz - 1) = 0$$

where $G = G(z)$, and $p(q^*)$ is the probability that $g'(\mathbf{h}_i^l) = 1$. For the ReLU activation function, $p(q^*) = 1/2$, whereas for the hard-tanh activation function (i.e. a linear approximation of tanh), $(q^*) = \text{erf}(1/\sqrt{2q^*})$. Using this equation, Pennington et al. found that the maximal eigenvalue of the density spectrum is

$$\lambda_{max} = \left(\sigma_w^2 p(q^*)\right)^L \left(\frac{e}{p(q^*)} L + \mathcal{O}(1)\right)$$

It should be noted that the maximal eigenvalue of $\mathbf{J}_{L,0}^T \mathbf{J}_{L,0}$ is the square of the maximal singular value of $\mathbf{J}_{L,0}$. From the expression above, we see that the maximal singular value grows linearly with each added layer, thus making it impossible to constrain the singular value spectrum around unity and attain dynamical isometry. When the same calculations are performed for a DNN where the weight matrices are instead orthogonal, such that $(\mathbf{W}^l)^T \mathbf{W}^l = \sigma_w^2 \mathbf{I}$, the following result is obtained for the maximal eigenvalue

$$\lambda_{max} = \left(\sigma_w^2 p(q^*)\right)^L \frac{1 - p(q^*)}{p(q^*)} \frac{L^L}{(L-1)^{L-1}}.$$

In this case, the maximal eigenvalue will grow for every added layer when employing the ReLU activation function. However, by using the hard-tanh activation function, it is found that the effect can be cancelled by choosing σ_w and σ_b so that $(1 - p(q^*))/p(q^*)$ becomes sufficiently small, thereby achieving dynamical isometry. Finally, it is shown that a network operating under dynamical isometry is able to avoid the VGP.

2.5.3 Schoenholz et al.

The results in [19] are mainly based on [16]. Using the same framework, they discover a length scale determining how deep information of an input vector may travel before it disappears. The length scales are defined from $|q_{aa}^l - q^*| \sim e^{-l/\xi_q}$ and $|c_{ab}^l - c^*| \sim e^{-l/\xi_c}$, i.e. the rate at which the normalised square length of an input vector and the correlation between two input vectors converge to a fixed point. This rate is calculated by using the iterative map in Equation 2.5 and expanding the equation around q^* given a small perturbation ε^l. This yields

$$\varepsilon^{l+1} = \varepsilon^l \left[\chi + \sigma_w^2 \int \mathcal{D}y \; g''(\sqrt{q^*}y) g(\sqrt{q^*}y)\right] + \mathcal{O}((\varepsilon^l)^2),$$

where the term enclosed in brackets is a constant. Hence, the length scale for a single input vector becomes

$$\xi_q^{-1} = -\log\left[\chi + \sigma_w^2 \int \mathcal{D}y \; g''(\sqrt{q^*}y) g(\sqrt{q^*}y)\right].$$

Similarly, for the correlation between two input vectors, the same procedure is done to obtain

$$\xi_c^{-1} = -\log\left[\sigma_w^2 \int \mathcal{D}y_a \mathcal{D}y_b \; g'(u_a^*) g'(u_b^*)\right].$$

In the case of $\chi < 1$ (i.e. vanishing gradient phase), the length scale becomes $\xi_c^{-1} = -\log \chi$. Additionally, when $\chi = 1$, the length scale diverges to infinity, meaning information can propagate unhindered.

3
Methods

The experiments presented in this section are divided into two parts. Firstly, simulations concerning the initialisation of DNNs are presented. Their purpose is to test whether dynamical isometry, as presented in [14] for $N \to \infty$, can be applied to narrow neural networks. The second part focuses on how DNNs manage to escape the vanishing gradient phase. All training is performed on either the `MNIST` data set of handwritten digits or the `CIFAR-10` data set. As such, all input layers will contain 784 or 1024 neurons respectively to accommodate for the 28×28 or 32×32 pixels of the `MNIST` and `CIFAR-10` images, and all output layers will have 10 neurons, as each data set has 10 training targets. All computation is done using Python and the NumPy library.

3.1 Dynamical isometry in narrow DNNs

Following the procedure of [14], a DNN with hard-tanh activation functions is initialised with orthogonal weight matrices with elements drawn from a uniform distribution such that $(\mathbf{W}^l)^T\mathbf{W}^l = \sigma_w^2\mathbf{I}$. The neuron biases are drawn from a Gaussian distribution with variance σ_b^2. The depth of the network is arbitrarily chosen to $L = 10$, as it is deep enough to exhibit the vanishing gradient phase and is not too computationally demanding. Based on this depth, values for σ_w and σ_b are chosen according to the theory established in [14] to obtain dynamical isometry for infinitely wide networks. These values are set to be $\sigma_w^2 = 1.05$ and $\sigma_b^2 = 2.01 \times 10^{-5}$. The networks are trained using SGD with a batch size of 10 and a learning rate of $\eta = 0.003$.

3.1.1 Width dependence of singular value distribution

As a leading assumption, Pennington et al. assume a large network width when exploring the possibility for dynamical isometry in DNNs. As the singular value distribution of a random matrix is dependent on its dimension, the results in [14] may be different if networks of small widths are considered. To observe this change, the singular value distribution of DNNs is simulated for $N = 10,\ 30,\ 50$, and 70 using the parameters given above. The simulation is carried out enough times to obtain 1000 samples of each choice of dimension.

3.1.2 Width dependence of learning rate

To see the effect the width has on how fast the network can be trained, the average number of training steps required to reach a 25% test accuracy is calculated for the same dimensions as above. The result is averaged over 100 iterations.

3.2 Escape from vanishing gradient phase

In this section, numerical simulations for understanding the dynamics of how a DNN escapes the vanishing gradient phase are presented. As the aim of the experiments is to analyse the dynamics of a DNN when it is able to escape the VGP, the initial variances of the weights and biases are selected to generate the escape within a training time of 100 epochs. As such, the initialisation is deliberately chosen to display the VGP instead of an initialisation which could have largely avoided the problem. This is done to so that relatively small and computationally cheap networks can be used, saving simulation time. The networks are trained using SGD with a learning rate of $\eta = 0.003$ and a batch size of 100. All networks utilise the *tanh* activation function.

3.2.1 Evolution of Lyapunov exponent

As described in section 2.5, the VGP phase is characterised by the input-output Jacobian having a negative maximal FTLE. In this experiment, the FTLE is monitored in a network starting in the VGP phase and which manages to escape it. The test is run on a network with depth $L = 5$ and widths $N = 10$, 30, and 50. The weights and biases were drawn from a Gaussian distribution with variances set to $\sigma_W^2 = 0.015$ and $\sigma_b^2 = 0$ for the `MNIST` data set, and $\sigma_W^2 = 0.030$ and $\sigma_b^2 = 0$ for the `CIFAR`-10 data set. The maximal FTLE is calculated using QR decomposition as presented in section 2.4.

3.2.2 Normalised average directionality

In [19], the hypothesis that a DNN in the VGP phase only allows information to travel a finite depth was presented. If this is the case, information about the input may not reach the output, and the updates to the network become meaningless. To measure this, the directionality of the training parameters of the network is monitored. Here, directionality refers to a tendency for the training parameters to move in a preferred direction in parameter space during one training epoch. If information about the data set passes through the network, the SGD algorithm will change the parameters in a way that moves the error function towards a minimum, giving rise to some preferred direction of movement during a training epoch. If the updates are random, no preferred direction of movement should be present and the directionality is low. To quantify the concept, the update vectors added to the biases in a layer $\nabla_{\mathbf{b}}^l H$ (see Equation 2.2) are added up during one training epoch. In the case of the `MNIST` data set with 60000 images, using a batch size of 100, this means adding 600 updating vectors. As only the direction of the updating vector is

of interest, each updating vector is normalised to unity before being added. Finally, the length of the resulting vector is divided by the number of terms in the sum. The norm of this vector is referred to as the normalised average directionality (NAD) of the training epoch and is a number in the range $[0, 1]$. If NAD $= 1$, each updating vector pointed in the same direction, whereas a 0 indicates that no direction is preferred. In equation form, this becomes

$$NAD = \left\| \frac{1}{T} \sum_{t=1}^{T} \mathbf{v}_t \right\|_2$$

where $\mathbf{v}_t$ is the normalised version of the updating vector to the biases in a given layer during time step t in a training epoch. T is the total number of training steps, which in the case of SGD is the number of samples in a data set divided by the batch size. In the experiment, only the NAD of each layer is calculated over the number of training epochs required for the DNN to escape the VGP. The same parameters used for the maximal FTLE are chosen for this experiment.

4
Results

4.1 Initialisation of singular values

The results in this section explore how well the methods of dynamical isometry in [14] can be transferred to DNNs of finite width.

4.1.1 Impact of width on singular value distribution

In Figure 4.1 the singular value distributions of four DNNs with different widths are shown, initialised to display dynamical isometry. As is clear from the figure, a smaller width leads to a distribution of singular values with larger tails, and a more significant portion of the singular values reside close to 0. Thus, narrow DNNs are expected to display a worse training performance than wider networks, due to a greater probability for the singular values to be far away from unity.

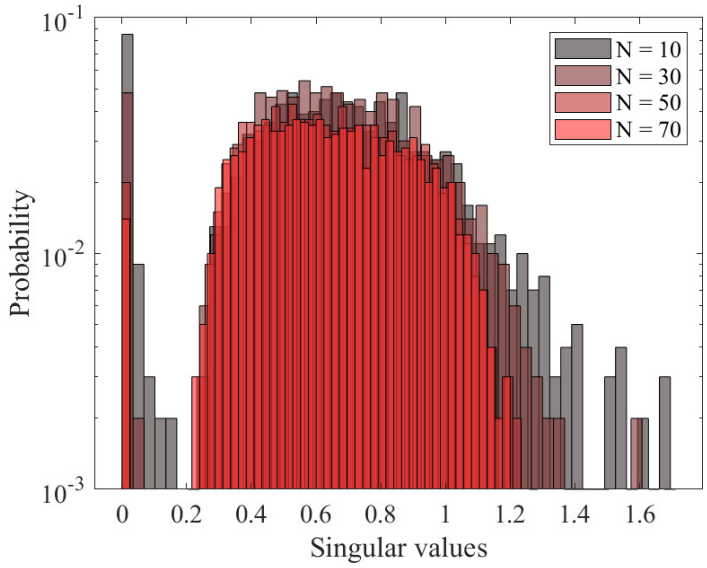


Figure 4.1: Singular value distribution for 4 DNNs with different widths. The histogram is plotted with a logarithmic vertical axis to better see the tails of the distribution. The networks have 10 hidden layers and the weight matrices are initialised to be orthogonal with a variance $\sigma_w^2 = 1.05$. The biases are initialised from a Gaussian distribution with mean 0 and variance $\sigma_b^2 = 2.05 \times 10^{-5}$. A *tanh* activation function is being used.

4.1.2 Impact of width on training speed

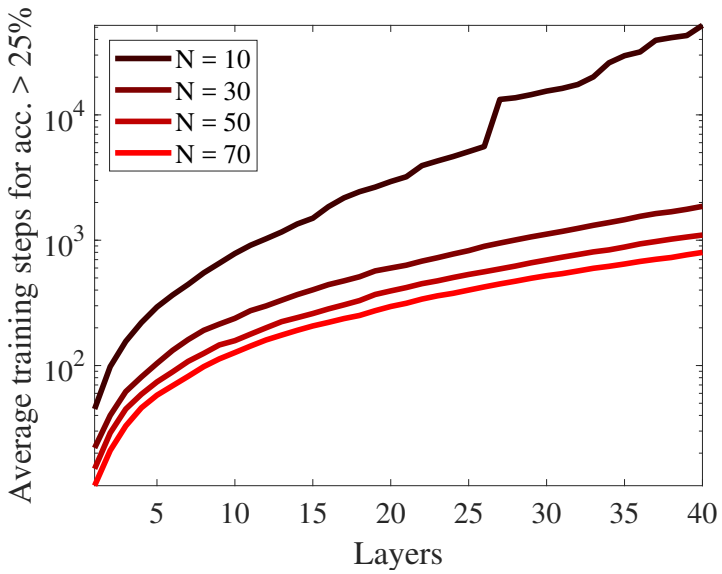


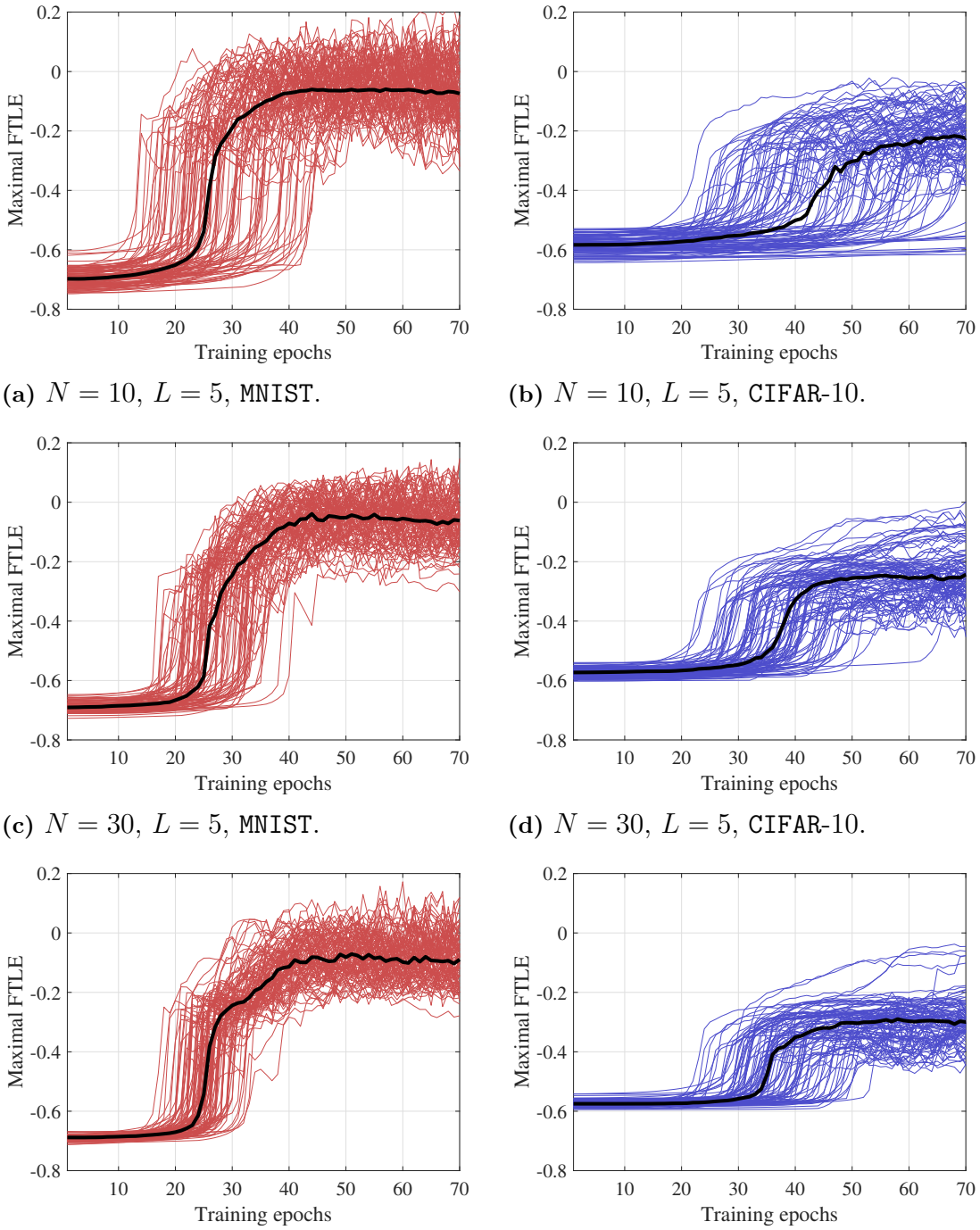
Figure 4.2: Average training steps required to reach an accuracy greater than 25% for different depths and widths. The vertical axis is logarithmic. The initialisation is the same as in Figure 4.1 and details about the training are found in section 3.1.

Using the same set-up, the average amount of training steps to reach 25% testing accuracy is simulated for different network depths. Figure 4.2 shows that an increased width decreases the average training time. This, in conjunction with the previous result, implies that dynamical isometry becomes more difficult to achieve for narrow DNNs.

4.2 Escape from the VGP phase

4.2.1 Dynamics of the maximal FTLE

The dynamics of the maximal FTLE is plotted in Figure 4.3 for the `MNIST` and `CIFAR`-10 data sets. For each case, 100 runs are plotted, together with the median of every training epoch. The median is chosen to represent the general behaviour rather than the mean value, as averaging the runs would have hidden the steep incline present in almost all instances. In all cases, the maximal FTLE is initially negative, indicating that the VGP is present. After an initial phase where the maximal FTLE increases very slightly, a rapid growth is observed. It is further observed, for both data sets, that the distribution of when the rapid growth occurs becomes more narrow for larger network widths. After the rapid increase, the growth saturates to some value closer 0. In other words, the process causes the network to become more dynamically isometric. In Figure 4.4a, the maximal FTLE in one run has been plotted together with the accuracy of the network during the same run. The plot shows that an increase in accuracy only begins once the maximal FTLE has begun to grow.



(a) $N = 10$, $L = 5$, `MNIST`.

(b) $N = 10$, $L = 5$, `CIFAR-10`.

(c) $N = 30$, $L = 5$, `MNIST`.

(d) $N = 30$, $L = 5$, `CIFAR-10`.

(e) $N = 50$, $L = 5$, `MNIST`.

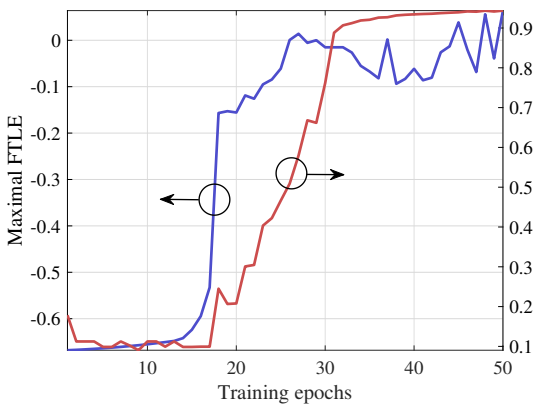
(f) $N = 50$, $L = 5$, `CIFAR-10`.

Figure 4.3: The dynamics of the maximal FTLE is plotted for 3 networks with different widths, trained on the `MNIST` and `CIFAR-10` data sets. 100 iterations are plotted together with the median of each training epoch. The training parameters are given in section 3.2.

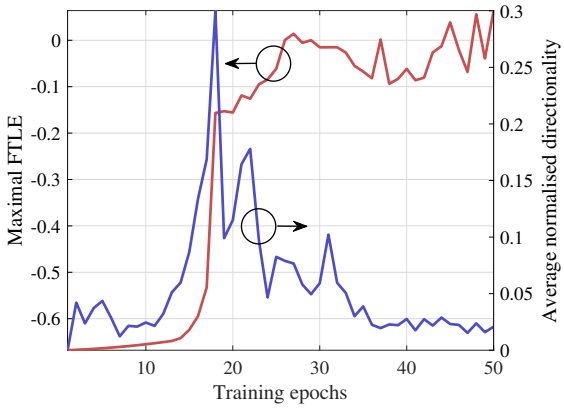
4.2.2 Directionality of training updates

In Figure 4.4b and Figure 4.4c, the NAD of the training epochs is plotted together with the maximal FTLE and accuracy during one training run. Only the direc-

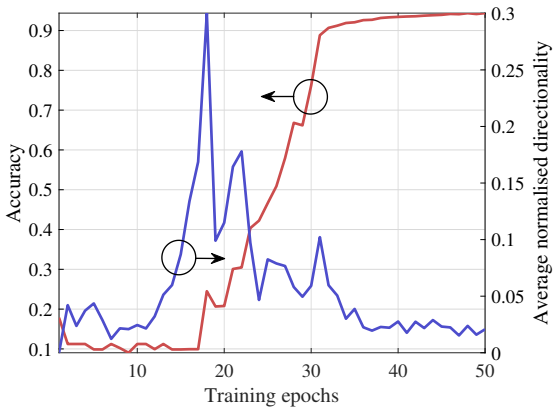
tionality for the first layer has been plotted, as its dynamics are most prominent, although later layers display the same qualitative behaviour. In the initial training phase, the directionality is low, meaning the direction the bias vector of the first layer moves is mainly random. Around the same time the maximal FTLE experiences a rapid growth, an increase in directionality occurs, followed by a slightly less rapid decrease. The directionality returns back to a low value around the same time the accuracy of the network approaches 100%.



(a) Maximal FTLE and accuracy.



(b) Maximal FTLE and NAD.



(c) Accuracy and NAD.

Figure 4.4: In the three plots, the maximal FTLE, accuracy, and NAD are compared for one training run. The arrows indicate which axis refers to which quantity. The training parameters are given in section 3.2.

4.2.3 Toy model

In this section, a toy model is derived which displays similar characteristics to the maximal FTLE in Figure 4.3. The model is derived for a 1-dimensional network to make interpretations of the dynamics more easily interpretable. In the case of a DNN with a constant width of 1, the updating rule for the weights in each layer becomes

$$w^l \to w^l + \eta \frac{\partial H}{\partial w^l}, \qquad b^l \to b^l + \eta \frac{\partial H}{\partial b^l},$$

where

$$\frac{\partial H}{\partial w^l} = \varepsilon \left(\prod_{k=L}^{l+1} g'(h^k) w^k \right) g'(h^l) x^{l-1}, \qquad \frac{\partial H}{\partial b^l} = \varepsilon \left(\prod_{k=L}^{l+1} g'(h^k) w^k \right) g'(h^l). \tag{4.1}$$

A significantly simplified version of the updating rule is derived by ignoring the contribution of the activation function, the error ε, and the signal x. This yields the recurrence relation

$$w^l \to w^l + \eta (w^l)^{L-1}, \qquad b^l \to b^l + \eta (b^l)^{L-1}.$$

Further simplifying the expression such that all weights and biases are equal at all times, the relation becomes

$$w \to w + \eta w^{L-1}, \qquad b \to b + \eta b^{L-1}. \tag{4.2}$$

Letting $\eta = 1$, arbitrarily setting the initial value of w to 0.09 and $L = 6$, the evolution of the logarithm of w becomes similar to the growth of the maximal FTLE, as seen in Figure 4.5. However, the model can only display this behaviour if $L - 1$ is either an odd number or even but with an initial value $w > 0$. This is found by calculating the stability of the recurrence by taking the derivative of the right-hand expression of Equation 4.2 and finding whether it is larger or smaller than unity at the initial value of w.

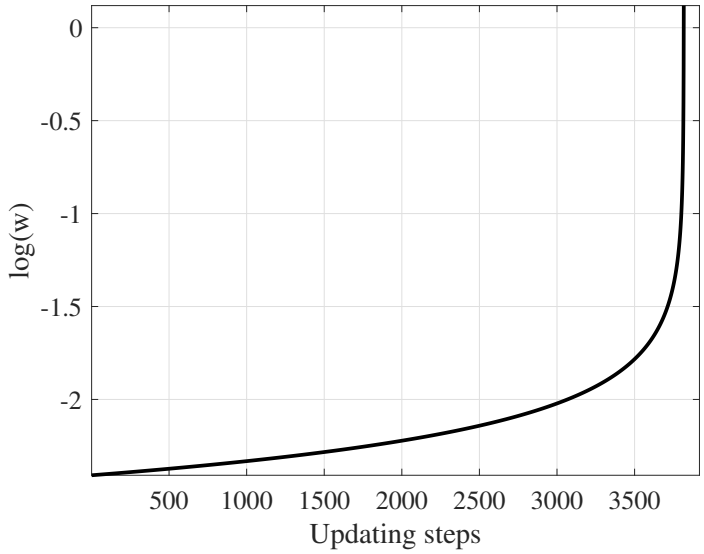


Figure 4.5: The evolution of the logarithm of w. The logarithm is chosen as the maximal FTLE calculated for the real DNN is related to the logarithm of the singular values of the weight matrices.

5
Discussion

5.1 Initialisation of singular values

The exploration of how DNNs can escape the VGP is interesting as an understanding of the dynamics may allow the development of methods to shorten the training time of the networks. However, if it is possible to entirely avoid the VGP, such insights may be redundant. The results of [14] suggest that an appropriately initialised network may display dynamical isometry, and thus allow signals to propagate through the network unhindered. However, the results were derived for wide networks and a similar initialisation for narrow networks may not be possible. The results in this work show that by decreasing the width of the network, the tails of the singular value distribution grow, taking the network further away from dynamical isometry by increasing the probability of having singular values far away from unity. Furthermore, it is shown that the growth of the tails leads to worse training performance, indicating that the vanishing gradient problem cannot be avoided as efficiently. As some networks, such as convolutional neural networks (CNNs), are often implemented with layers with a width of 5 (i.e. convolution layers), and are trained through backpropagation, these results are significant.

It should be noted that the increase in training time may not be solely due to tail growth of the singular value distribution. Instead, the training could be slower because there are fewer training parameters, making it more difficult to fit the network to the data set. However, a DNN with 10 layers and a constant width of 30 trained on the `MNIST` data set contains 31300 training parameters, which is very large amount. Thus, the decreased training rate is not expected to be caused by having too few training parameters.

5.2 Escape from the VGP phase

The dynamics of how a DNN escapes the VGP is revealed by computing how the maximal FTLE evolves as the network is updated. From Figure 4.3, we see that a rapid growth from a negative value to a value closer to 0 occurs after some training epochs. The transition coincides with that the network begins to learning, as shown in Figure 4.4a. Based on the results of [16] and [19], this results suggest that the network goes from a state where information about the input vectors disappears with every layer, to a state where the network is closer to dynamical isometry. Given that no increase in accuracy occurs prior to the rapid growth of the FTLE, it may be

that no information about the input reaches the output. Hence, the network cannot adjust its parameters to improve the accuracy. This hypothesis is further strengthened by the result in Figure 4.4b, where the average normalised direction the biases in the first layer of the network move has been plotted together with the maximal FTLE. If no information about the input vector reaches the output, the error vector ε becomes a random vector. Therefore, the direction in which the parameters of the network move in parameter space should also be random. Indeed, this is shown to be the case in the figure, and only when the maximal FTLE becomes large enough do the updates display any directionality. The reason the directionality decreases after the initial increase is explained by Figure 4.4c; once the network has achieved a high level of accuracy, the parameters are close to a minimum and the absence of a strong gradient causes the updating direction to appear more random.

If no information can reach the output of the network, the VGP is not solely caused by a vanishing gradient, but also a lack of information reaching the output. Moreover, the growth of the maximal FTLE should not be caused by the structure of the data set, as no information about it can propagate through the network, but rather be an effect of the training algorithm itself. In Figure 4.3, we can see that a similar growth occurs for two different data sets. The dynamics of the maximal FTLE can be divided into three phases: the initial phase, the growth phase, and the saturation phase. The initial phase is controlled by the initialisation of the weights and biases, and if the network is successfully initialised in dynamical isometry, the initial value of the maximal FTLE would be close to 0. The saturation phase can be explained by the presence of $\boldsymbol{\varepsilon}_L$ in Equation 2.3. As the maximal FTLE grows, information can propagate through the network and the training algorithm can minimise the error vector. As the error is minimised, $\boldsymbol{\varepsilon}_L$ becomes smaller, decreasing the sizes of the weight and bias updates until they are close to zero. Ultimately, this stumps the growth of the maximal FTLE. Finally, while the dynamics of the growth phase are less obvious, an illustration of how such a rapid increase could occur is made with the toy model in subsection 4.2.3. As the weights are all assumed to be the same in the toy model, the result may be closer related to RNNs, which also suffer from the VGP but the backpropagation is done through time in the same layer. Moreover, the toy model neglects all random components found in the updating rule for a real DNN, which could greatly alter the dynamics. However, if the weights become large enough for the rapid growth to occur in the toy model, the contribution of the random components may become negligible. More research must be done to determine whether this hypothesis has any grounds.

An interesting phenomenon is seen in Figure 4.3, where the distribution of when the growth occurs becomes more narrow as the width of the network increases. The narrowing of the distribution may be attributed to that the distribution of the initial value of the maximal FTLE becomes more narrow as well. This, in turn, is caused by an averaging effect of having more random parameters in the $\mathbf{W}$ and $\mathbf{D}$ matrices. Interestingly, the narrowing occurs despite the randomness introduced by SGD. This, again, suggests that the rapid growth of the maximal FTLE is not related to which data set is being used.

data set

5.3 Continued research

This work was restricted to only analyse fully connected, feedforward architectures with constant widths. Furthermore, only the *tanh* activation function was being employed. To see whether the dynamics of the maximal FTLE generally occur in DNNs, more testing on other architectures must be performed. For instance, the VGP may behave differently if the width of the network does not remain the same [15]. Additionally, networks utilising the ReLU activation function may display different dynamics, as the ReLU activation function only saturates in one direction. A network architecture that has proven to greatly reduce the VGP is the residual neural network (ResNet) [12], and it could be interesting to analyse how the maximal FTLE evolves in such networks.

6
Conclusion

In this work, the ability of a DNN to escape the VGP when trained using the SGD algorithm is investigated using dynamical systems theory. Furthermore, the method introduced by Pennington et al. to initialising a network in dynamical isometry to avoid the VGP is evaluated for narrow DNNs.

The maximal FTLE found using the input-output Jacobian of a DNN describes how two adjacent inputs either converge or diverge as they propagate through the network. For negative Lyapunov exponents, the inputs converge and as they reach the output they become indistinguishable. Hence, information is unable to propagate through the network, making the network unable to learn. The results of this work show that after some training epochs, the maximal FLTE begins to grow rapidly, moving the DNN closer to dynamical isometry. As a result, the network is able to learn. Furthermore, the results suggest the growth is not related to the data set the DNN is being trained on, but that it is caused by the training algorithm. Hence, to further understand the underlying dynamics of the escape from the VGP, research should be focused on how the updating algorithm behaves in the absence of training data.

The dynamical isometry initialisation is shown to work poorly for narrow networks. As the width of the network becomes smaller, the tails of the singular value distribution become wider, and the probability of having either high or low singular values will increase, making the average learning time for a network slower for smaller networks.

Bibliography

[1] Forest Agostinelli et al. "Learning activation functions to improve deep neural networks". In: *arXiv preprint arXiv:1412.6830* (2014).

[2] Yoshua Bengio, Aaron C Courville, and Pascal Vincent. "Unsupervised feature learning and deep learning: A review and new perspectives". In: *CoRR, abs/1206.5538* 1 (2012), p. 2012.

[3] Luca Dieci, Michael S Jolly, and Erik S Van Vleck. "Numerical techniques for approximating Lyapunov exponents and their implementation". In: *Journal of Computational and Nonlinear Dynamics* 6.1 (2011), p. 011003.

[4] Oliver Faust et al. "Deep learning for healthcare applications based on physiological signals: A review". In: *Computer methods and programs in biomedicine* 161 (2018), pp. 1–13.

[5] Joseph F Grcar. "A matrix lower bound". In: *Linear Algebra and its Applications* 433.1 (2010), pp. 203–220.

[6] Yanming Guo et al. "Deep learning for visual understanding: A review". In: *Neurocomputing* 187 (2016), pp. 27–48.

[7] Boris Hanin. "Which neural net architectures give rise to exploding and vanishing gradients?" In: *Advances in Neural Information Processing Systems*. 2018, pp. 582–591.

[8] Bekir Karlik and A Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.

[9] Janusz Kolbusz, Pawel Rozycki, and Bogdan M Wilamowski. "The study of architecture MLP with linear neurons in order to eliminate the "vanishing gradient" problem". In: *International Conference on Artificial Intelligence and Soft Computing*. Springer. 2017, pp. 97–106.

[10] Martin Längkvist, Lars Karlsson, and Amy Loutfi. "A review of unsupervised feature learning and deep learning for time-series modeling". In: *Pattern Recognition Letters* 42 (2014), pp. 11–24.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[12] Sihan Li et al. "Demystifying resnet". In: *arXiv preprint arXiv:1611.01186* (2016).

[13] Riccardo Miotto et al. "Deep learning for healthcare: review, opportunities and challenges". In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[14] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice". In: *Advances in neural information processing systems*. 2017, pp. 4785–4795.

[15] George Philipp, Dawn Song, and Jaime G Carbonell. "The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions". In: *arXiv preprint arXiv:1712.05577* (2017).

[16] Ben Poole et al. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems*. 2016, pp. 3360–3368.

[17] Daniele Ravı et al. "Deep learning for health informatics". In: *IEEE journal of biomedical and health informatics* 21.1 (2016), pp. 4–21.

[18] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[19] Samuel S Schoenholz et al. "Deep information propagation". In: *arXiv preprint arXiv:1611.01232* (2016).

[20] P Sibi, S Allwyn Jones, and P Siddarth. "Analysis of different activation functions using back propagation neural networks". In: *Journal of Theoretical and Applied Information Technology* 47.3 (2013), pp. 1264–1268.

[21] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[22] C Zhang et al. *Understanding deep learning requires rethinking generalization*. 2018.

[23] Xiao Xiang Zhu et al. "Deep learning in remote sensing: A comprehensive review and list of resources". In: *IEEE Geoscience and Remote Sensing Magazine* 5.4 (2017), pp. 8–36.

A

Appendix 1

Here, a calculation showing how a DNN with a linear activation function loses its depth is presented. The calculation makes use of the properties of linearity.

$$
\begin{aligned}
g(\mathbf{W}^L g(\mathbf{W}^{L-1} g(\dots g(\mathbf{W}^1 \mathbf{x}^0 + \mathbf{b}^1)\dots) + \mathbf{b}^{L-1}) + \mathbf{b}^L) = \\
\left(\prod_{k=L}^{1} \mathbf{W}^k\right) g^{(L)}(\mathbf{x}^0) + \sum_{l=1}^{L-1} \left(\prod_{k=L}^{l+1} \mathbf{W}^k\right) g^{(L-l+1)}(\mathbf{b}^l) + g(\mathbf{b}^L) = \\
\tilde{\mathbf{W}} g^{(L)}(\mathbf{x}^0) + \tilde{\mathbf{b}}
\end{aligned}
$$

Here, $g^{(l)}$ indicates that the function g has been applied l times and

$$
\tilde{\mathbf{W}} = \prod_{k=L}^{1} \mathbf{W}^k, \qquad \tilde{\mathbf{b}} = \sum_{l=1}^{L-1} \left(\prod_{k=L}^{l+1} \mathbf{W}^k\right) g^{(L-l+1)}(\mathbf{b}^l) + g(\mathbf{b}^L), \qquad L > 1.
$$