The Pennsylvania State University
The Graduate School
College of Engineering

PRINCIPLES OF RIEMANNIAN GEOMETRY IN NEURAL NETWORKS

A Dissertation in
Mechanical and Nuclear Engineering
by
Michael Hauser

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2018

The dissertation of Michael Hauser was reviewed and approved* by the following:

Asok Ray
Distinguished Professor of Mechanical Engineering and Mathematics
Graduate Faculty of Electrical Engineering
Dissertation Advisor, Co-Chair of Committee

Shashi Phoha
Research Professor
Graduate Faculty of Electrical Engineering
Director, Division of Information Sciences and Technology
Applied Research Laboratory
Co-Chair of Committee

Daniel Haworth
Professor of Mechanical Engineering

Kenneth Jenkins
Professor Emeritus of Electrical Engineering

Karen Thole
Distinguished Professor of Mechanical Engineering
Head of the Department of Mechanical Engineering

*Signatures are on file in the Graduate School.

Abstract

The first part of this dissertation deals with neural networks in the sense of geometric transformations acting on the coordinate representation of the underlying data manifold from which the data is sampled. It forms part of an attempt to construct a formalized general theory of neural networks in the setting of algebraic and Riemannian geometry. Geometry allows for a rigorous formulation, and therefore understanding, of neural networks as they act on data manifolds, which is certainly of great importance as these tools become ubiquitous in engineering applications. From this perspective, the following theoretical results are developed and proven for feedforward networks. First it is shown that residual neural networks are finite difference approximations to dynamical systems of first order differential equations, as opposed to ordinary networks that are static. This implies that the network is learning systems of differential equations governing the coordinate transformations that represent the data. Second it is shown that a closed form solution of the metric tensor on the underlying data manifold can be found by backpropagating the coordinate representation through the neural network. This is formulated in a formal abstract sense as a sequence of pullback / Lie group actions on the metric fibre space in the principal and associated bundles on the data manifold, where backpropagation is shown to be the pullback / Lie group actions on tensor bundles. A model based on perturbation theory is developed and used to understand how neural networks treat testing data differently than training data.

The second part of this dissertation makes use of neural networks for forecasting probability distributions of time series in terms of discrete symbols that are quantized from real-valued data. The developed framework formulates the forecasting problem into a probabilistic paradigm as $h_\Theta : X \times Y \mapsto [0,1]$ such that $\sum_{y \in Y} h_\Theta(x,y) = 1$, where X is the finite-dimensional state space, Y is the symbol set, and Θ is the set of model parameters. The main advantage of formulating probabilistic forecasting in the symbolic setting is that density predictions are obtained without any significantly restrictive assumptions, such as second order statistics. The efficacy of the proposed method has been demonstrated by forecasting probability distributions on chaotic time series data collected from a laboratory-scale experimental apparatus.

Contents

List of Figures viii

List of Tables xiv

List of Symbols xv

Acknowledgments xvii

Chapter 1
Introduction 1
1.1 Neural Networks in Machine Learning . . . 3
1.2 Neural-Geometric Intuitions . . . 4
1.3 Group Structures on Data Manifolds . . . 9
1.4 Mathematical Notations . . . 11
1.5 Neural Probabilistic Forecasting . . . 11
1.6 Layout of Dissertation . . . 12

Chapter 2
Neural Network Models 14
2.1 Supervised models . . . 14
2.1.1 Standard and Convolution Feed Forward Neural Networks . . . 15
2.1.2 Recurrent Neural Networks . . . 16
2.1.2.1 Long Short-Term Memory Neural Networks . . . 17
2.2 Unsupervised models . . . 18
2.2.1 Restricted Boltzmann Machines . . . 18
2.2.1.1 Deep Belief Networks . . . 19
2.2.2 Autoencoders . . . 19
2.2.2.1 Deep Autoencoders . . . 20
2.2.3 Generative Adversarial Networks . . . 20
2.2.4 Locally Linear Embeddings . . . 21

2.2.5 t-distributed Stochastic Neighbor Embedding 22
2.3 Training neural networks 24
2.3.1 Cost functions 24
2.3.1.1 Cross-entropy 24
2.3.1.2 L_2 error 25
2.3.2 Gradient descent 25
2.3.3 Error Backpropagation 25

Chapter 3
Smooth Manifolds **27**
3.1 Topological Spaces 27
3.2 Topological Manifolds 30
3.2.1 Manifolds 30
3.2.2 Bundles 33
3.2.3 Tangent Spaces 37
3.2.4 Cotangent spaces 42
3.2.5 Tensor Spaces 44
3.2.6 Push-forward and Pull-back 45
3.2.7 Tensor Fields and Modules 48
3.2.8 Lie Groups 49
3.2.9 Principal and Associated Bundles 51
3.2.9.1 Principal Bundles 51
3.2.9.2 Associated Bundles 55
3.3 Riemannian Manifolds 58
3.3.1 Pullback of general (r, s)-tensor bundles 60
3.3.1.1 Properties of Riemannian Manifolds 61

Chapter 4
The Geometrical Anatomy of Neural Networks **63**
4.1 Coordinate representation of the data manifold 64
4.1.1 Discussion 69
4.1.1.1 Coordinate representations of data manifolds 69
4.1.1.2 Effect of batch size on set connectedness and topology 70
4.2 Coordinate representation of the metric tensor 72
4.3 Coordinate representation of the frame bundle 77
4.4 Limiting scenarios of C^k network architectures 81
4.4.1 Discussion 85
4.4.1.1 Neural networks with C^k differentiable coordinate transformations 85

4.4.1.2 Effect of number of layers on the separation process . 87
4.5 Perturbation theory on neural networks 88
4.5.1 The special case of a residual network 91

Chapter 5
Probabilistic forecasting **93**
5.1 Probabilistic Forecasting with Deep Networks 93
5.1.0.1 Mathematical preliminaries 93
5.1.1 Time series symbolization 94
5.1.2 Neural networks 94
5.1.2.1 Restricted Boltzmann Machines 94
5.1.2.2 Deep neural networks 95
5.1.3 Autoregressive moving average (ARMA) 97
5.1.3.1 Developed algorithm 98
5.1.3.2 Results and discussion 99
5.1.3.3 Discussion 104
5.2 Probabilistic Forecasting with LSTM Networks 106
5.2.1 Mathematical preliminaries 106
5.2.1.1 Probabilistic formulation 106
5.2.1.2 Long Short-Term Memory 107
5.2.1.3 Time series symbolization 108
5.2.2 Algorithm Development 108
5.2.2.1 Mathematical formulation 109
5.2.2.2 Testing three neural network architectures 111
5.2.3 Results and Discussion 111
5.2.3.1 Description of the Combustor Apparatus 112
5.2.3.2 Forecasting Physical Phenomena 112
5.2.4 Discussions 117

Chapter 6
Future Work and Conclusions **119**
6.1 Future Work 119
6.2 Conclusions 122

Appendix A
Proofs from Chapter 3 **123**
A.1 Introduction 123
A.2 Topological Spaces 123
A.2.1 The standard topology is a topology 123
A.2.2 The induced topology is a topology 123

A.3 Manifolds 124
A.3.1 Pointwise $+_{T_pM}$ and $\cdot_{T_pM}$ close in T_pM 124
A.3.2 $\dim T_pM = \dim M$ 124

Appendix B
Additional Structures **126**
B.1 Introduction 126

Bibliography **129**

List of Figures

1.1 Enormous variety exists in sizes, shapes, colors, lighting and pose of very similar objects. 4

1.2 A 32×32 pixel image of the letter A, scaled and rotated. When embedding/compressing the image with locally linear embedding there are two underlying dimensions required to reproduce the image, namely scale and angle of rotation. The underlying topology of this manifold is thus the cylinder $M = \mathbb{R} \times S^1$. As a topological space, there is insufficient structure to represent the images. When endowed with a coordinate system acting locally on $U \subset M$ such that $x : U \to x(U) \subset \mathbb{R}^2$ defined by $x : p \mapsto x(p) := (r, \theta)$, then each image $p \in M$ can be represented in coordinates (r, θ), as seen in the scatter plot on the right. 6

1.3 Word embeddings preserve intuitive notions of distances of words. . . 8

1.4 A Figure borrowed from Bojanowski *et al.* "Optimizing the Latent Space of Generative Networks". In the embedding representation, the averages of a, b and c are taken and operated to create a new image $a - b + c$. On an intuitive level, c represents a woman's face without sunglasses, where as $a - b$ represent a man's face with sunglasses minus a man's face without sunglasses, i.e. just sunglasses. Thus performing the group operation vector addition $c + (a - b)$ of a woman's face without sunglasses plus sunglasses should yield a woman's face with sunglasses. Notice that vector addition only makes sense in flat space $\mathbb{R}^d$. 10

3.1 A topological manifold $(M, \mathcal{O})$ with coordinate systems (U, x) and (V, y) such that $U \cap V \neq \phi$. The coordinate transformation $y \circ x^{-1} : x(U \cap V) \to \mathbb{R}^{\dim M}$ is k-times differentiable for a $\mathcal{C}^k$-differentiable manifold. 31

3.2 Let $\varphi : M \to N$ be a map, where $(M, \mathcal{O}_M, \mathcal{A}_M)$ and $(N, \mathcal{O}_N, \mathcal{A}_N)$ are $\mathcal{C}^k$-manifolds. Then φ is called differentiable at $p \in M$ if for some $(U, x) \in \mathcal{A}_M$ with $U \ni p$ and some chart $(V, y) \in \mathcal{A}_N$ with $V \ni \varphi(p)$, the map is $\mathcal{C}^k$ as a map from $y \circ \varphi \circ x^{-1} : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim N}$. Displayed as well is the means of transforming coordinates from x and y to $\tilde{x}$ and $\tilde{y}$ to produce the map $\tilde{y} \circ \varphi \circ \tilde{x}^{-1} : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim N}$. 32

3.3 The Mobius strip is an example of a manifold that is not globally homeomorphic to $\mathbb{R}^d$. As a bundle, where the base space is S^1 and the fibres are the intervals $[-1, 1]$, it cannot be written globally as a Cartesian product, i.e. it is not a trivial fibre bundle. 34

3.4 A fibre bundle (E, π, M, F). The base manifold is M, the fibres are F and the entire space E is the fibre bundle. In this figure, $\dim M = 2$, $\dim F = 1$ and $E = M \times F$ so this is a trivial fibre bundle. At a given point $p \in M$ the corresponding fibre is $F_p = \pi^{-1}(\{p\})$. Along this fibre, the projection map $\pi : E \to M$ takes any point on the fibre and maps it to the base point on the manifold $p \in M$. A section is a map $\sigma : M \to E$ such that $\pi \circ \sigma = 1_M$, and in this figure would be any 1-dimensional fibre-valued field on the base manifold. 35

3.5 Two bundles (E, π, M) and (E', π', M') are said to be isomorphic as bundles if this diagram commutes, i.e. there exists bijective, continuous maps $u : E' \to E$ and $f : M' \to M$ such that $\pi \circ u = f \circ \pi'$. 36

3.6 The composition of a curve $\gamma : \text{preim}_\gamma(U) \to U$ and a function $f : U \to \mathbb{R}$ yields $f \circ \gamma : \text{preim}_\gamma(U) \to \mathbb{R}$. The derivative of this function can be taken using standard calculus. In coordinates $x : U \to x(U)$, the map is $(f \circ x^{-1}) \circ (x \circ \gamma) : \text{preim}_\gamma(U) \to \mathbb{R}$ which is the same thing since x is a homeomorphism on $U \in \mathcal{O}$. 40

3.7 Visualization of the push-forward map. We have $X \in T_pM$ is such that $X : \mathcal{C}^\infty(M) \tilde{\to} \mathbb{R}$, and $\varphi_* : T_pM \tilde{\to} T_{\varphi(p)}N$ so $\varphi_* : X \tilde{\to} (\varphi_* X) \in T_{\varphi(p)}N$ is such that $(\varphi_* X) : \mathcal{C}^\infty(N) \tilde{\to} \mathbb{R}$ is defined by how it operates on the smooth function $f \in \mathcal{C}^\infty(N)$ as $(\varphi_* X)(f) := X(f \circ \varphi)$, where $f \circ \varphi : M \to \mathbb{R}$ so $(f \circ \varphi) \in \mathcal{C}^\infty(M)$, and hence X can act on it. . . . 45

3.8 The map $f : M \to N$ is called ρ-equivariant if this diagram commutes. 52

3.9 Given bundles $P \xrightarrow{\pi} M$ and $P' \xrightarrow{\pi'} M'$, the maps $u : P \to P'$ and $f : M \to M'$ are called principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' g$. 54

3.10 Given bundles $P \xrightarrow{\pi} M$, $P' \xrightarrow{\pi'} M'$ and Lie group homomorphism $\rho : G \to G'$ such that $\rho(g_1 \bullet_G g_2) = \rho(g_1) \bullet_{G'} \rho(g_2)$, the maps $u : P \to P'$ and $f : M \to M'$ are called generalized principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' \rho(g)$. 55

3.11 A principal G-bundle $P \overset{\triangleleft G}{\longleftarrow} P \overset{\pi}{\to} M$ is called trivial if there exists a principal bundle map u to $M \times G \overset{\triangleleft' G}{\longleftarrow} M \times G \overset{\pi_1}{\to} M$ given by $(x, g) \triangleleft' g' := (x, g \bullet g')$ and $\pi_1 : M \times G \to M$ such that $\pi_1 (x, g) := x$. 55
3.12 The relations between principal fibre bundles (left side) and trivial fibre bundles (right side) can be seen. 56
3.13 The principal bundle isomorphism (left side) is such that $\pi' \circ u = h \circ \pi$ and $u (p \triangleleft g) = u (p) \triangleleft' g$, while the induced associated bundle (right side) is such that $\tilde{u} ([p, f]) := [u (p) , f]$ and $\tilde{h} (m) := h (m)$ for all $p \in P, f \in F$ and $m \in M$. 57

4.1 Coordinate systems $x^{(l+1)} := \varphi^{(l)} \circ ... \circ \varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)}$ induced by the coordinate transformations $\varphi^{(l)} : x^{(l)} (M) \to \left(\varphi^{(l)} \circ x^{(l)}\right) (M)$ learned by the neural network. The pullback metric $g_{x^{(l)}(M)} (X, Y) := g_{\left(\varphi^{(l)} \circ x^{(l)}\right)(M)} \left(\varphi_*^{(l)} X, \varphi_*^{(l)} Y\right)$ pulls-back (i.e. backpropagates) the coordinate representation of the metric tensor from layer $l+1$ to layer l, via the pushfoward map $\varphi_*^{(l)} : T x^{(l)} (M) \to T \left(\varphi^{(l)} \circ x^{(l)}\right) (M)$ between tangent spaces. 64
4.2 Untangling and linearly separating the same spiral manifold with 2-dimensional $\mathcal{C}^1$ neural networks with different types of nonlinear coordinate transformations. The x and y axes are the two nodes of the neural network at a given layer l, where layer 0 is the input representation and layer 5 is the output representation for linear classification. Both networks exhibit smooth layerwise coordinate transformations whose coordinates are slight perturbations from the previous layer. The data does not change, only the layer-to-layer coordinate representation of the data changes. Depending on the types of coordinate transformations available to the network, the final coordinate representation of the manifold varies. 65
4.3 Layerwise coordinate transformations with a $\mathcal{C}^1$ (residual) network, used to change the shape of the input to match an output via ℓ^2 minimization. The coordinate transformations take place (approximately) smoothly over the network as the next layer is a slight perturbation from the previous. Also note that if distances at the output are measured by the Euclidean metric, then to preserve the metric properties from the input, the output coordinate space becomes non Cartesian. 70

4.4 The effect of batch size on coordinate representation learned by the same 2-dimensional $\mathcal{C}^1$ network, where layer 0 is the input representation, and both examples achieve 0% error. A basic theorem in topology says continuous functions map connected sets to connected sets. A small batch size of 300 during training sparsely samples from the connected manifold and the network learns overfitted coordinate representations. With a larger batch size of 1000 during training the network learns a simpler coordinate representation and keeps the connected input connected throughout. 71

4.5 A $\mathcal{C}^1$ network with a hyperbolic tangent activation function separating the spiral manifold. Additionally, balls of constant radius $ds = \sqrt{g_{a_l b_l}(x^{(l)}) dx^{(l)a_l} dx^{(l)b_l}}$ at different points are shown. In the output coordinates, they are measured by standard Euclidean distance in Equation 4.7, and so the balls are "round". The coordinate representation of the metric tensor is backpropagated through the network to the input by Equations 4.8 and 4.9; so distances on the data manifold can be measured with the input coordinates, and so the balls are not round. The balls are represented as ellipses, with the principal components of the metric tensor acting as the principal components of the ellipses. These balls can also be interpreted as forming an $\epsilon - \delta$ relationship across layers of the network, where an ϵ-ball at one layer corresponds to a δ-ball at the previous layer. 73

4.6 Untangling the same spiral with 2-dimensional neural networks with different constraints on smoothness. The x and y axes are the two nodes of the neural network at a given layer l, where layer 0 is the input data. The $\mathcal{C}^0$ network is a standard network, while the $\mathcal{C}^1$ network is a residual network and the $\mathcal{C}^2$ network also exhibits smooth layerwise transformations. All networks achieve 0.0% error rates. The momentum term in the $\mathcal{C}^2$ network allows the red and blue sets to pass over each other in layers 3, 4 and 5. Figure 4.6d has the identity connection for all layers other than layer 6. 86

4.7 The effect of number of layers on the separation process of a $\mathcal{C}^1$ neural network. In Figure 4.7a it is seen that the Δl is too large to properly separate the data. In Figures 4.7b and 4.7c the Δl is sufficiently small to separate the data. Interestingly, the separation process is not as simple as merely doubling the parameterization and halving the partitioning in Equation 4.25 because this is a nonlinear system of ODE's. This is seen in Figures 4.7b and 4.7c; the data are at different levels of separation at the same position of layer parameterization, for example by comparing layer 18 in Figure 4.7b to layer 36 in Figure 4.7c. 87

5.1 The deep neural network architecture used for this study consists of an input layer of size 10, two hidden layers of size 22 and an output layer of size 10. The 10 dimensional input layer corresponds to a signal history of size 10, and the 10 dimensional output layer corresponds to the alphabet size of the symbol set. 100
5.2 Example result of ARMA forecasting, where blue is the true RMS over an extended period of time and red is the predicted RMS starting at time step 700 and going until time step 764. In this example the space was partitioned into $K = 10$ equal disjoint regions and 11 of the first 13 forecasted classes were correctly predicted by ARMA, corresponding to about 0.03 seconds. 101
5.3 Probabilistically forecasting instabilities. The red curves are the true RMS and the grayscale map indicates predictions of what class the combustion instability belongs to, black being probability 0 and white being probability 1. 102
5.4 A comparison of the forecasting abilities of the three models, being the neural network model with cross-entropy minimization, the expectation over the posteriors of that neural network, and finally the ARMA model. The neural network model is labeled correct if the $\arg\max_k P(Y = k|X = x)$ is correct. The expectation of the neural network is $\hat{y} = \sum_{i=1}^{K} \bar{y}_i P(Y = y_i|X = x)$ and is labeled correct if the expected trajectory falls in the same partition region as the true trajectory. 103
5.5 Schematic of the LSTM neural network stucture. The nonlinear activations, σ and $\tanh$, act elementwise on their respective input vectors. Similarly, multiplication and addition blocks operate on their input pairs elementwise. 107
5.6 Schematic diagram of the combustor apparatus 112
5.7 Forecasts from the neural probabilistic framework; black corresponds to low probability and white corresponds to high probability. The LSTM and feed forward networks are compared, as well as forecast lengths of 4 and 10 time-steps. Solid red lines are the true trajectory while dotted blue lines are the expectation over the forecasted probability distributions. 114
5.8 Test error rates for different combinations of forecast length and number of symbols. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate. When averaged over all pairwise combinations, the relative reductions in error can be seen in Table 5.1. 115

5.9 Weighted error rates for different combinations of forecast length and number of symbols, where the weighting factor is determined by the distance between the predicted symbol and the true symbol (determined by partition centroid). The weighted error can be interpreted as the average distance between the predicted and true symbol, scaled between 0 and 1. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate with a threshold value of 0.065. When averaged over all pairwise combinations, the relative reductions in weighted error can be seen in Table 5.2. 116

List of Tables

5.1 Relative reduction in error. 116
5.2 Relative reduction in weighted error. 116

List of Symbols

X	Data input random variable
Y	Data label random variable
P	Probability
M	Manifold with points $p \in M$
$\mathcal{O}$	Topology on M
U	Open subset $U \subseteq M$ where $U \in \mathcal{O}$
$\bullet$	Group operation $\bullet : M \times M \to M$ of manifold M
$\tilde{\to}$	The $\tilde{}$ denotes that the mapping is linear
x	General coordinate system on $U \subseteq M$ such that $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$
$x^{(l)}$	Neural coordinate system at layer l on the manifold $x^{(l)} : U \to x^{(l)}(U)$
$\varphi^{(l)}$	Neural coordinate transformation at layer l on the manifold $\varphi^{(l)} : x^{(l)}(U) \to \left(\varphi^{(l)} \circ x^{(l)}\right)(U)$
$x^{(0)}$	Input Cartesian coordinate system $x^{(0)} : U \to x^{(0)}(U) \subseteq \mathbb{R}^{\dim M}$
$x^{(L)}$	Output learned coordinate system $x^{(L)} : U \to x^{(L)}(U) \subseteq \mathbb{R}^{\dim M}$
$x^{(l+1)}$	Coordinate system at layer $l+1$ induced by the layerwise sequence of learned coordinate transformations $\varphi^{(0)}, \varphi^{(1)}, ..., \varphi^{(l)}$ $x^{(l+1)} : U \to x^{(l+1)}(U) := \left(\varphi^{(l)} \circ \varphi^{(l-1)} \circ ....\varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)}\right)(U)$

$\left(\frac{\partial}{\partial x^a}\right)_p$	Canonically induced basis of T_pM from coordinates $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$ for $a = 1, 2, ...\dim M$
$d_p x^a$	Canonically induced covector basis of T_p^*M from $\left(\frac{\partial}{\partial x^a}\right)_p$ for $a = 1, 2, ...\dim M$
(M, g)	Riemannian manifold with metric $g : T_pM \times T_pM \to \mathbb{R}$ with the notation $g\left(x^{(l)}\right)_{a_l b_l} := g\left(\left(\frac{\partial}{\partial x^{(l)}}\right)_{a_l}, \left(\frac{\partial}{\partial x^{(l)}}\right)_{b_l}\right)$
h_Θ	Neural network function $h_\Theta : X \to Y$ with parameters Θ

Acknowledgments

First and foremost I would like to thank my family for their endless support while I've gone through all of my years of school.

I'd like to thank my friends, labmates and advisors at the University of Toronto. I first developed my foundational skills in mathematics and physics at UofT, and it has been invaluable ever since when switching between different fields of applied mathematics. I'd like to thank Fredric Schuller for his course titled "The Geometrical Anatomy of Theoretical Physics", of which Chapter 4 of this dissertation is named after.

I would like to thank my lab mates over the years that I've worked with and have taught me so much and have been so enormously helpful. These include, but are not limited to, Pannathai Khamdaengyodtai, Yue Li, Pritthi Chattopadhyay, Yiwei Fu, Jan Petrich, Najah Jasim, Samer Saab and Sean Gunn.

I would like to thank Albert Einstein for his beautiful theory of General Relativity, which was absolutely inspirational for developing a geometric theory of neural networks. In a similar manner I would like to thank Noam Chomsky. Both his careers as a critic of the political economy of power structures, as well as a mathematical linguist, were built on looking past the disinformation, rooting out presuppositions and asking the correct questions; skills he's articulated so well in his books and lectures.

I would like to thank my advisor at the Applied Research Laboratory, Shashi Phoha. The broad knowledge she has of everything from statistics to successful grant writing has been, and will continue to be, extremely helpful for the rest of my life.

Finally I would like to thank Asok Ray. Joining his laboratory at the Pennsylvania State University was one of the best decisions of my life. His extensive knowledge across so many diverse fields of mathematics, physics and engineering is truly unique. The freedom and encouragement he gave me to pursue topics that I am interested in, while always guiding me to make sure I don't go completely off the rails, has helped me immensely in becoming an independent researcher.

The work reported in this dissertation was supported by the Walker Fellowship from the Applied Research Laboratory, Division of Information Science and Technology, and by the U.S. Air Force Office of Scientific Research under Grant No. FA9550-15-1-0400.

Dedication

I dedicate this to my family.

Chapter 1
Introduction

This dissertation brings together what are usually considered two distinct branches of mathematics; namely graphical models and geometric manifolds. These seemingly unrelated branches of mathematics come together in a beautiful way when studying neural networks and how they operate on data manifolds. Geometric interpretations of neural networks form a minority perspective in machine learning; it is this minority perspective that this study develops, showing that there exists a rich connection between neural networks and Riemannian geometry. *The principal objective of this dissertation is to understand and formulate neural networks at the structural level of a Riemannian manifold.* This prescription of finding and studying the minimum mathematical structure is common in physics, where for example the mathematical structure required to understand Quantum Mechanics is Hilbert spaces [1], and for General Relativity it is Riemannian manifolds [2].

According to the MIT Technology Review in April, 2017, "No one really knows how the most advanced algorithms [neural networks] do what they do. That could be a problem." [3]. Neural networks lack a clear mathematical formulation, often understood by analogies from digital signal processing [4–6], data compression [7, 8], density estimation [9,10], biology [11,12], and vague references to geometry [13,14]. Perhaps the strangest analogy for understanding neural networks comes from motivating dropout regularization by the role of sexual reproduction in evolution [15]. The large majority of research into neural networks goes into their implementation, with the purpose to improve their performance. Far less serious work exists for understanding what they are actually doing. As neural networks increase in complexity [16–18] it will only become more difficult to understand how they operate. Luckily neural networks are built from a relatively small number of building blocks, so if we can thoroughly understand these

building blocks then we can begin to understand the complete network.

The purpose of this study is to form part of an attempt in closing the enormous gap that exists between *how well* neural networks function and *how* neural networks function. Experimental results consistent over decades of research and across many research groups suggest that neural networks, from a machine learning perspective, deserve a serious geometric formulation [13, 19–22], one which is lagging far behind the levels of implementation these models currently receive. The main body of work contained in this thesis is in giving neural networks a rigorous geometric formulation.

The preface to the text *Syntactic Structures* [23], which is often credited as mathematically formalizing the field of linguistics, is quoted verbatim:

> The search for rigorous formulation in linguistics has a much more serious motivation than mere concern for logical niceties or the desire to purify well-established methods of linguistic analysis. Precisely constructed models for linguistic structure can play an important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact source of this inadequacy and, consequently, gain a deeper understanding of linguistic data. More positively, a formalized theory may automatically provide solutions for many problems other than those for which it was explicitly designed. Obscure and intuition-bound notions can neither lead to absurd conclusions nor provide new and correct ones, and hence they fail to be useful in two important respects.

For exactly the same purposes outlined in the preface of *Syntactic Structures* it is necessary to have a rigorous mathematical formulation of neural networks. Additionally the two important roles of a precisely constructed model, namely 1. discovering an unacceptable conclusion and modifying the model and 2. automatically providing solutions for unintended problems, has already born results. The first led to a better understanding of how the definition of a data manifold is related to the model discovering the data manifold. The second is the error backpropagation algorithm is shown to have a much deeper mathematical meaning than how it is commonly understood, which is sequential applications of the chain rule. Instead it is shown that backpropagation is an example of a non-abelian gauge theory in differential geometry, and can be used to find the closed form solution to the metric tensor on the data manifold. More generally it

can be used to find the closed form solution of *any* covariant tensor or tensor density bundle on the data manifold that is initialized in the output coordinates, such as the error gradient or metric tensor.

For the engineer it is absolutely imperative to understand what the tool is doing before using it and building on it. As these tools become ever more entrenched in our everyday lives it becomes ever more important to understand what they are doing. A simple example is, do you trust an autonomous blackbox algorithm to drive your car when one in a thousand times when it encounters a person, it misidentifies the person as some insignificant object and it is completely unknown *why* it misidentified the person as such. From a serious engineering perspective this is unacceptable, and this dissertation hopes to improve the engineering understanding of neural networks.

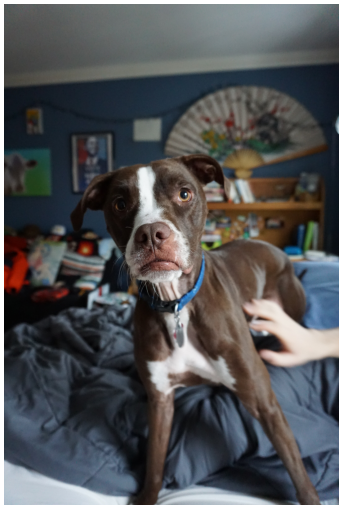
Engineering applications of neural networks are being developed and implemented at a very fast pace. These tools are being used in many tasks that involve computer vision, natural language processing and reinforcement learning. Many aspects of self driving cars, home automation and other challenging engineering problems require these tools at their most fundamental levels.

1.1 Neural Networks in Machine Learning

In machine learning, neural networks are a collection of parametric models for both supervised as well as unsupervised learning. The goal of supervised learning is usually to find parameters Θ for a classification model $h_\Theta : X \to Y$ such that the model can be interpreted as $P(Y|X = x) := h_\Theta(x)$, where X is the set of input data, Y are the set of labels and $P(Y|X = x)$ is the discrete histogram over different values of $y \in Y$. More precisely, the supervised model is a function $P(Y = y|X = x) := h_\Theta(x, y)$ held to the constraint such that $\sum_{y \in Y} h_\Theta(x, y) = 1$ for all $x \in X$. This definition then induces a probability distribution by defining $P(Y|X = x) := h_\Theta(x) := h_\Theta(x, \cdot)$, where the y-argument slot is left open.

The goal of unsupervised learning is to find parameters Θ such that the model can be interpreted as $P(X = x) := h_\Theta(x)$. For example if $x \in X$ is an image, then the high level understanding of $P(X = x)$ is that this is the probability that the image is a natural image as opposed to, say, just noise. Supervised and unsupervised learning are related by Bayes rule $P(Y|X)P(X) = P(X, Y)$, where $P(X, Y)$ is called a generative model.

(a) A small, gray poodle.



(b) A medium, brown boxer.

Figure 1.1: Enormous variety exists in sizes, shapes, colors, lighting and pose of very similar objects.

1.2 Neural-Geometric Intuitions

Arguably the area in which neural networks have been most successful is in computer vision. In computer vision, the model $h_\Theta : X \to Y$ takes as input images $x \in X$ and labels them $y \in Y$. Computer vision is a difficult field for many reasons. One reason is because the images are often largely oversampled in rather arbitrary coordinates, so for example a $100 \times 100 \times 3$ red-green-blue image $x \in X \subset [0, 255]^{100\times100\times3} \subset \mathbb{R}^{100\times100\times3}$ is a $30,000$-dimensional vector. Simply shifting the pixels over one row or column will move the $30,000$-dimensional vector an enormous amount in $\mathbb{R}^{100\times100\times3}$-space. Slightly more complex operations, such as rotating the image, can have an even larger effect on the vector. Even more difficult is the enormous variety of objects that exist. For example, there is no one way of writing the digit "7", just as there is no one type of "dog". Dogs exist with extreme variability such as size, shape, color, breed, pose, orientation, lighting, etc. as seen in Figure 1.1. When identifying a "dog" in an image, the model should be able to account for these possible dimensions of variability, as well as many other dimensions we intuitively understand but cannot list.

These difficulties exist because we are *implicitly* representing dogs in an RGB coordinate system. RGB is a useful coordinate system for displaying images on a monitor, but to also expect RGB to be a useful coordinate system for representing

the abstract concept of a dog is too much beyond what RGB was designed to do. Convolutional neural networks [4, 6] learn hierarchical channels [5, 24] that better embed the images, and thus learn the image manifold. In these learned coordinates a small shift from translating or rotating the image leads to a small shift in the learned coordinate representation [5]. This means that the coordinate representation in the output is more similar to how we intuitively understand images than the coordinate representation in the input.

From a mathematical structural level, an important point should be made here. When studying data in machine learning, these very simple arguments imply that data must be studied *at least* at the mathematical structural level of a manifold. Roughly speaking, a manifold is a topological space that is locally homeomorphic to $\mathbb{R}^d$, and the homeomorphism is a coordinate system. It is insufficient to study data from only the structural level of a topological space. This is because whenever one is working with real world data, that data was collected *with respect to some coordinate system.*

For example with an image of a dog, the image is implicitly using an RGB Cartesian coordinate system. If the images are through a fish-eye lens, the camera is now viewing the world through an RGB curvy-linear coordinate system. If all of the images are from a rotated upside down camera, the images are taken from an upside down coordinate system. In all of these cases, *the dog has not changed, only the coordinate representation of the dog has changed.* With a fixed coordinate system, changing the orientation of the dog, or the lighting, or using a different breed of dog, we are sampling a different point of the manifold of allowable dog images, where each possible dimension of change corresponds to a dimension of the manifold.

This phenomena is seen Figure 1.2, with the example of the image of the letter A being scaled and rotated, the dimension of the underlying data manifold is 2. This follows from the fact that, after a proper embedding/encoding of the image of the letter A via locally linear embedding [22], the only variations allowed in the images is scaling and rotation, i.e. it has the topology of $\mathbb{R} \times S^1$. Thus if $p \in \mathbb{R} \times S^1$, then this image can be represented by the coordinates $x : \mathbb{R} \times S^1 \to x(\mathbb{R} \times S^1)$ defined by, for example, $x : p \mapsto x(p) := (r, \theta) \subseteq \mathbb{R}^2$. Another coordinate system $y : \mathbb{R} \times S^1 \to y(\mathbb{R} \times S^1) \subseteq \mathbb{R}^{32\times 32}$ can be used to map the point to an actual 32×32 image, and transforming between the two coordinate representations is performed by the coordinate transformation $\varphi : y(\mathbb{R} \times S^1) \to x(\mathbb{R} \times S^1)$, where φ^{-1} embeds the 2-dimensional point in $\mathbb{R}^{32\times 32}$. To reiterate, we can represent each point $p \in \mathbb{R} \times S^1$ with

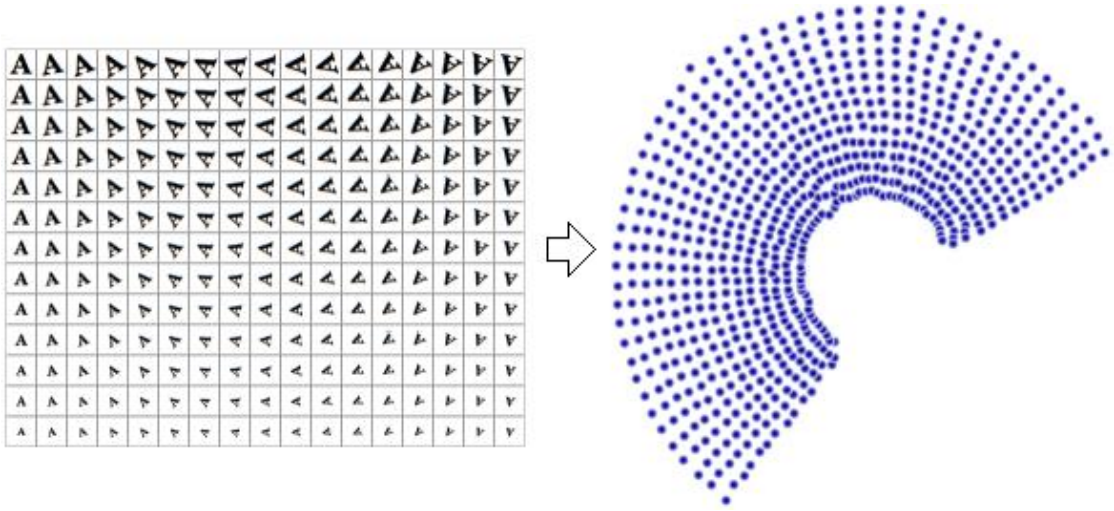


Figure 1.2: A 32×32 pixel image of the letter A, scaled and rotated. When embedding/compressing the image with locally linear embedding there are two underlying dimensions required to reproduce the image, namely scale and angle of rotation. The underlying topology of this manifold is thus the cylinder $M = \mathbb{R} \times S^1$. As a topological space, there is insufficient structure to represent the images. When endowed with a coordinate system acting locally on $U \subset M$ such that $x : U \to x(U) \subset \mathbb{R}^2$ defined by $x : p \mapsto x(p) := (r, \theta)$, then each image $p \in M$ can be represented in coordinates (r, θ), as seen in the scatter plot on the right.

the coordinate system $x(p) = (r, \theta) \in \mathbb{R}^2$, or with the coordinate system $y(p) \in \mathbb{R}^{32 \times 32}$. From a differential geometric perspective both representations of $p \in M$ are equivalent since an isomorphism learned by locally linear embedding exists, but in terms of data compression the x-system is preferred over the y-system.

A topological space is too weak a mathematical structure to account for this since a topological space has no notion of a coordinate system; the stronger conditions of a manifold is required. Inquires into the topology of the data manifold are always performed through a coordinate representation of the data manifold.

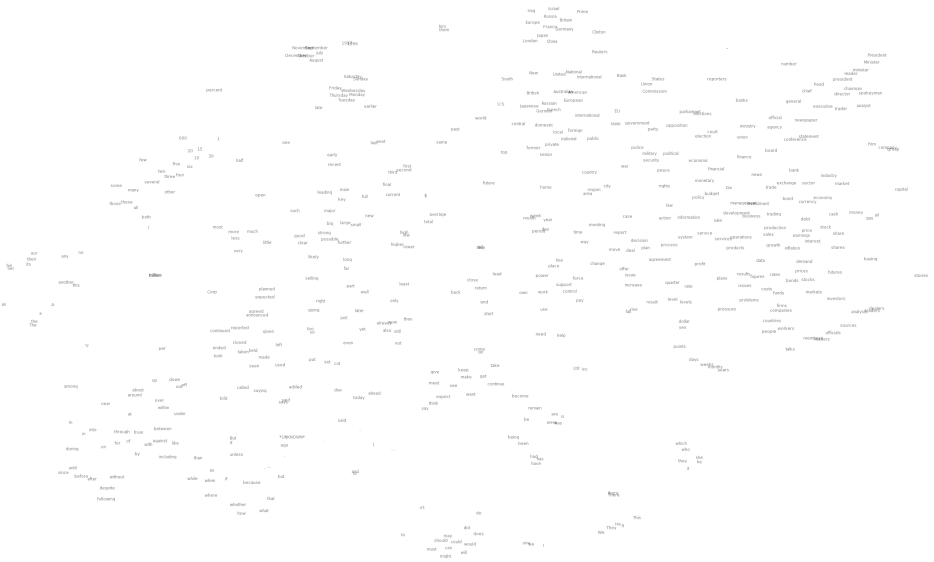
Notice as well that even with the extreme levels of variability that exist in the same kinds of objects, for example in Figure 1.1, on an intuitive level different breeds of dogs are similar, just as different kinds of cars are similar, but dogs are very different from cars. *Thus on an intuitively we have a geometric notion of distance between objects.* The feature embedding space learned by neural networks, and many types of manifold learning algorithms, preserve these intuitive notions of distance, as seen in Figures 1.2 and 1.3. The case of neural networks is particularly interesting because even without

explicitly requiring the neural network to preserve metric properties such as distance in the cost function (as compared to locally linear embedding [22]), they still tend to learn a feature embedding representation that preserves these intuitive notions of distance. This now implies that the minimum mathematical structure for how neural networks function must have at least the structure of topological manifold possessing a metric; namely a Riemannian manifold.

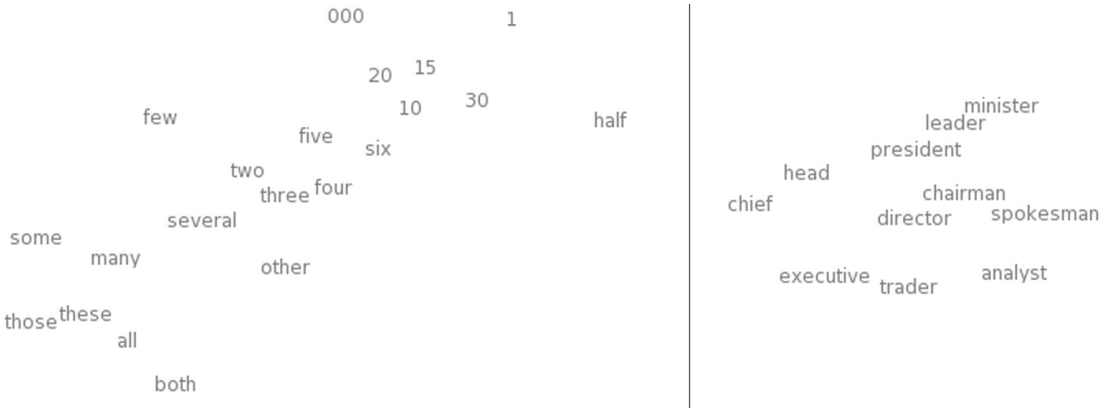
Machine learning as a field was revolutionized in 2012 when instead of hard coding feature detectors such as in the Histogram of Oriented Gradients [26] and Scale Invariant Feature Transform [27], neural networks were used to learn the feature detectors to account for the extreme variability that exists in objects [4, 6]. These convolutional neural networks take inspiration from digital signal processing by learning a hierarchy of match filters [5]; thus instead of hard coding the filters, the model learns the filters from data. This is necessary because if hard coding feature detectors that account for the variability of only dogs is difficult, then hard coding feature detectors for all types of objects and all of their orientations is not feasible.

Convolutional neural networks take inspiration from the visual cortex as well, such as the Hierarchical Model and X [11]. Within the V1 region of the visual cortex of the primate brain, the simple cells respond only to very simple shapes such as oriented vertical and horizontal lines, where as more complex cells in later regions respond to more complex visual features [28]. This was initially discovered by Hubel and Wiesel in 1962 when studying the visual cortex of an anesthetized cat [29] for which they won the Nobel prize. Cats were used because they have very good vision, and they were anesthetized to slow down other neural activity that could be taking place and introducing noise into the measurements.

Similar hierarchical representations take place virtually everywhere. In *Godel, Escher, Bach: an Eternal Golden Braid*, Hofstadter describes how scientific subjects can be grouped together based on the level of fundamental assumptions [30]. For example, the physiologist understands how organs function together for the animal on the highest level. A level below a biologist understands how molecules function together for the organs. A level below a chemist understands how molecules function together for the cells, and finally a level below a physicist understands how atoms function together for the molecules. Each of these levels is self-contained, and has difficulty communicating with the other levels. So just as it does not make sense to represent and understand an image in pixels, it does not make sense to represent and understand an animal in terms



(a) A global view of word embeddings learned by the neural network [19].



(b) A local view of some specific clusters [19].

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

(c) A table showing a collection of words and their 10 nearest neighbors [25].

Figure 1.3: Word embeddings preserve intuitive notions of distances of words.

of elementary particles.

In addition to these supervised neural network models, neural network research in unsupervised learning, for example Generative Adversarial Networks [10], suggest serious geometric interpretations. These unsupervised models learn a representation of images of digits such that interpolations in the embedding space produce smoothly changing, plausible looking images [31]. Very roughly speaking, this means there is a flattening of the representation of the data manifold. This is because the interpolation is taking place by Euclidean distance, and for there to be plausible looking images means the embedding space must be Euclidean. To perform a Euclidean interpolation on a non-Euclidean manifold does not make sense and thus produces implausible looking images, as the interpolation will have fallen off the manifold. Similar results were achieved by using the unsupervised Deep Belief Network and interpolating between both images of digits as well as images of faces [13].

1.3 Group Structures on Data Manifolds

In addition to learning a flattened representation of the data manifold, neural networks have also been shown to learn an embedded representation of the data that can "preserve vector offsets" [21]. More mathematically, the network is learning a representation of the data manifold that approximately forms a Lie Group with group operation vector addition [32], a stronger requirement than only requiring a Euclidean manifold. Two illustrative examples are given below.

Using a generative latent optimization (GLO) model [33], it has been shown that the high-level learned representation of some images approximately preserves vector offsets. For example, images of (man with glasses) - (man without glasses) + (woman without glasses) produces a plausible image of a woman with glasses, as seen in Figure 1.4 from Bojanowski *et al.* [33]. This example suggests that the network is learning a flattened representation of an image embedding space with some resemblance to vector Lie Group closure.

A second type of neural network which has also been very successful is the word embedding language model. This neural network learns to embed words into a low-dimensional space [34] to take advantage of linguistic commonalities of certain words, as explained more clearly with the following example:

- Jill walked two miles in Boston.

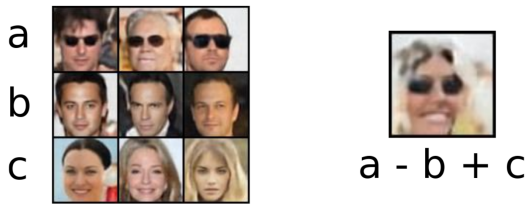


Figure 1.4: A Figure borrowed from Bojanowski *et al.* "Optimizing the Latent Space of Generative Networks". In the embedding representation, the averages of a, b and c are taken and operated to create a new image $a - b + c$. On an intuitive level, c represents a woman's face without sunglasses, where as $a - b$ represent a man's face with sunglasses minus a man's face without sunglasses, i.e. just sunglasses. Thus performing the group operation vector addition $c + (a - b)$ of a woman's face without sunglasses plus sunglasses should yield a woman's face with sunglasses. Notice that vector addition only makes sense in flat space $\mathbb{R}^d$.

- Jane ran three kilometers in Paris.

In this example, Jill/Jane, walked/ran, two/three, miles/kilometers and Boston/Paris all play similar roles in the sentence and can replace each other to form new, syntactically plausible sentences.

It is first seen that the embedded representation of the words preserve some form of a distance measure, where similar words get clustered together and dissimilar words are pushed apart, as seen in Figures 1.3 [19, 25].

Additionally, the embedded representation of these words sometimes preserve vector offsets [21], with the example $x_{apples} - x_{apple} \approx x_{cars} - x_{car} \approx x_{families} - x_{family}$ given for the word embedding vector x_i. This suggests that, just as happens with the GLO model described above, the network learns a flattened representation of a word embedding space with some approximate form of vector Lie Group closure. It is only approximate because although the embedded representation learns a "plural" vector so that $x_{apple} + x_{plural}$ closes to x_{apples}, it will not always close as adding the vectors e.g. $x_{apple} + x_{car}$ is a nonsense construction.

On a general manifold it does not make sense to add different points on the manifold. This only makes sense in Euclidean space $\mathbb{R}^d$ because only in Euclidean space $\mathbb{R}^d$ can one confuse points and vectors. From a structural perspective, manifolds are paracompact, Hausdorff topological sets that are locally homeomorphic to $\mathbb{R}^d$. They have no structural addition operation. A Lie Group is a topological manifold that is also a group. With group operation vector addition, this implies that the manifold is flat. If one tries to use

vector addition on a manifold that isn't flat, there is no guarantee of closure and so one will end up with implausible images or word vectors that do not make sense.

1.4 Mathematical Notations

This dissertation adopts the standard notation of differential geometry, as it is formulating neural networks in this sense.

Einstein notation is used throughout this paper. A raised index in parenthesis, such as $x^{(l)}$, means it is the l^{th} coordinate system while $\varphi^{(l)}$ means it is the l^{th} coordinate transformation. If the index is not in parenthesis, a superscript (contravariant) free index means it is components of a vector, a subscript (covariant) free index means it is components of a covector, and a repeated index means implied summation over the vector components. The $.$ in tensors, such as $A^{a}_{\cdot b}$, ensure consistency in the order of the superscripts and subscripts.

1.5 Neural Probabilistic Forecasting

The second part of this dissertation reviews the development of a neural network based probabilistic forecasting algorithm.

Time series forecasting is a well studied problem across a diverse range of fields [35]. From the perspectives of dynamical systems, accurately forecasting future states can be implemented within a feedback mechanism in the case where control actions can be used to drive the system to a desired state or away from an undesired state. To this end, linear stationary forecasting models [35] have been widely used. For example, ARMA models provide a standard benchmark for linear stationary time series forecasting; if the time series is nonstationary, then autoregressive integrated moving average (ARIMA) models [35] are used to deal with these nonstationary processes.

To overcome the limitations of linear models, forecasting time series with neural networks has been reviewed by Zhang et al. [36]. Neural networks can be made to be highly nonlinear, and with even a single hidden layer they have the ability to approximate any Borel measurable function [37] with a sufficiently large hidden layer. Additionally, it is desirable to replace these single-valued hard predictions by forecasting probability distributions over symbolic states [38] [39].

A forecasting paradigm requires evaluation of both a predicted value and the bounds on its accuracy [40]. Casting the problem in a symbolic-probabilistic setting provides a wide range of generalities; for example, symbols can be interpreted as emanating from a state space. Furthermore, they facilitate probabilistic predictions without prior assumptions on data distributions (e.g. second-order statistics). However, they lack the fidelity that comes with deterministic predictions, although this shortcoming can be partially alleviated by taking an expectation over the prediction space.

Statistical characterizations of symbol sequences have been extensively investigated within the framework of Markov models [41, 42]. Within this framework, symbol emission probabilities are estimated, given a current state of the system. The symbol emission probability matrix has been cast in a Bayesian framework as well [43].

This thesis formulates the probabilistic forecasting problem in a classification sense by using the concept of neural networks such that, conditioned on the time series history, the desired output is a probability over symbols; the symbols partition the range space and each symbol corresponds to a class. This procedure is significantly different from a regression of the mean and its associated quantiles.

The technical approach of the proposed method is to estimate symbol emission probabilities largely analogously to what is done in Markov modeling with stochastic symbolic emissions from given states; however it does so with neural networks. In contrast to a D-Markov model [44, 45] that generates probabilities of emitting certain symbols given a finite history *symbol sequence* of length D, the newly developed model generates symbol emission probabilities based on a finite history of *time series* of length D; in this way the information that is usually lost through course graining is retained.

1.6 Layout of Dissertation

With these motivations, a rigorous geometric formulation of neural networks is developed. This dissertation does not attempt to review all of the extensive literature on neural networks and differentiable manifolds. Instead, it will review only the most important aspects that are necessary to develop this geometric formulation in order to keep the dissertation self-contained.

In Chapter 2 the most important types of neural architectures are formulated. These architectures are used as building blocks to build up the larger, state-of-the-art algorithms. Chapter 3 develops the geometric tools that are used in mathematically rigorous

formulations of non-abelian gauge theories such as General Relativity, Quantum Field Theory and String Theory, with the purpose that they will be used to provide a rigorous formulation of neural networks. The geometric formulation of neural networks will be made in Chapter 4. Chapter 5 reviews new applications of neural networks for probabilistic forecasting. Finally in Chapter 6 future work is suggested as well as concluding remarks are made.

Chapter 2
Neural Network Models

This chapter aims to succinctly provide the necessary background for understanding the standard formulation of neural networks in machine learning in the setting of graphical models, digital signal processing and density estimation. This is by no means a complete review of neural networks, rather a short introduction to provide the reader with a working knowledge of neural networks. The two traditional paradigms of machine learning, supervised and unsupervised modeling, will be reviewed here.

2.1 Supervised models

The goal of this section is to provide a basic working knowledge of neural networks for supervised learning. The goal of the supervised model is to learn parameters Θ for a fixed model structure $h_\Theta : X \to Y$ such that the output is usually interpreted as $P(Y|X = x) := h_\Theta(x)$. The parameters of the model are the learned weights and biases from all the layers $\Theta := \{\left(W^{(l)}, b^{(l)}\right)\}_{l=0}^{L}$, where $\left(W^{(L)}, b^{(L)}\right)$ are the parameters for the softmax classification layer.

The probabilistic output usually comes from the linear softmax classifier:

$$\text{softmax}\left(z^{(L)}\right) := \frac{e^{z^{(L)}}}{\sum_{k=1}^{K} e^{z_k^{(L)}}} \tag{2.1}$$

where $z^{(L)} := W^{(L)} \cdot x^{(L)}$ and $x^{(L)}$ is the final layer input for the softmax classifier.

2.1.1 Standard and Convolution Feed Forward Neural Networks

A standard feed forward neural network is a multilayer perceptron [37,46] and is densely connected:

$$x^{(l+1)} = f\left(W^{(l)} \cdot x^{(l)} + b^{(l)}\right) \tag{2.2}$$

At this point, the $\cdot$ and $+$ operations are just the standard matrix multiplication and vector addition operations of finite dimensional linear algebra, and f is some non-linear function applied elementwise such as ReLu, $\tanh$ or σ. The universal approximator theorems [37,46] are for a squashing non-linearity.

The standard feed forward network is densely connected because there are no *a priori* restrictions on the network connections $W^{(l)}$. Convolution neural networks [4,6] impose a substantial restriction on network connections:

$$x^{(l+1)} = f\left(W^{(l)} * x^{(l)} + b^{(l)}\right) \tag{2.3}$$

where the $*$ is the standard convolution operation from digital signal processing; for example in 2-dimension:

$$\left(W^{(l)} * x^{(l)}\right)_{ij} := \sum_{j'=-J}^{J} \sum_{i'=-I}^{I} W^{(l)}_{i'j'} * x^{(l)}_{i-i'j-j'} \tag{2.4}$$

For an image $x^{(l)}$ of size $m_l \times n_l$ with c_l-channels, the dimension of the entire image is $m_l \times n_l \times c_l$. If the convolution weight matrix has c_{l+1} filters/feature detectors, all of size $u_l \times v_l$, then the matrix $W^{(l)}$ has total dimensions $u_l \times v_l \times c_l \times c_{l+1}$.

Additionally, weights are shared across convolutions. This is because it is both practical to reduce the number of parameters, as well as natural images strongly exhibit statistical stationarity properties. This means that low level features like edges, patterns and object parts exist in different places all over images.

The intuition behind the convolution neural network for computer vision is that nodes should process information on a local level, and so they have a local receptive field. If there exists a type of edge or pattern in one side of the image, that suggests very little for what exists on another side of the image.

For convolution neural networks, entire bodies of literature exist on max/mean/stochastic pooling [4,47], batch normalization [48], types of activation functions [49–51],

dropout regularization [15, 52, 53] etc. These are extremely important topics, but unnecessary to review for the point of this dissertation.

The residual neural network [54, 55] uses a "skip-connection" between network layers. These are currently used in many state-of-the-art implementations across diverse fields of machine learning, such as object detection and tracking in computer vision and natural language processing.

For a standard feed forward network, the residual network takes the following form.

$$x^{(l+1)} = x^{(l)} + f\left(W^{(l)} \cdot x^{(l)} + b^{(l)}\right) \tag{2.5}$$

Residual networks were originally designed [54] with the intuition that it is easier to learn perturbations from the identity map than it is to learn an unreferenced map. Further experiments then suggest that residual networks work well because, during forward propagation and back propagation, the signal from any block can be mapped to any other block [55]. After unraveling the residual network, this attribute can be seen more clearly. From this perspective, the residual network can be understood as an ensemble of shallower networks [56]. In Chapter 4 of this dissertation an alternative understanding of residual networks, supported by experimental results, is presented.

2.1.2 Recurrent Neural Networks

Recurrent neural networks are usually used to process dynamic data. Recurrent networks introduce a time-lagged feedback connection to the standard neural network architecture.

For a time series $\{x_0, x_1, x_2, ..., x_t, ...\}$, the recurrent network processes the data as follows:

$$x_t^{(l+1)} = f\left(W^{(l)} \cdot x_t^{(l)} + U^{(l+1)} \cdot x_{t-1}^{(l+1)}\right) \tag{2.6}$$

The recurrent network is trained using the backpropagation through time algorithm [57] which unrolls the network into an approximate feed forward form so that the standard backpropagation algorithm can be used.

2.1.2.1 Long Short-Term Memory Neural Networks

Long Short-Term Memory (LSTM) neural networks [58] are recurrent neural networks with a specific architecture in the hidden space which allows the network to act as if it has a memory unit it can read-from and write-to. The LSTM has an input x_t, an output h_t and a cell state C_t that acts as the memory and is the central component of the LSTM. It is noted that the sigmoid function $\sigma(z) := 1/(1+\exp(-z))$ has a range of 0 to 1, and the hyperbolic tangent function $\tanh(z) := (\exp(z) - \exp(-z))/(\exp(z) + \exp(-z))$ has a range of -1 to 1.

The candidate cell state $\tilde{C}_t$, its weightings i_t, and the forget gate activations f_t are computed as follows:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i), \tag{2.7}$$

$$\tilde{C}_t = \tanh(W_c \cdot x_t + U_c \cdot h_{t-1} + b_c), \tag{2.8}$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f) \tag{2.9}$$

where W's and U's are appropriate weight matricies and the b's are bias vectors.

With $\odot$ defined to be elementwise multiplication, the updated cell state C_t is then found as:

$$C_t = i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \tag{2.10}$$

and the output is a weighted version of the cell state:

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_f), \tag{2.11}$$

$$h_t = o_t \odot \tanh(C_t) \tag{2.12}$$

Because Equations (5.20), (5.21), (5.22) and (5.24) depend only on x_t, h_t and h_{t-1}, their arguments are computed in parallel.

2.2 Unsupervised models

This section will review the most important unsupervised learning models in neural networks. Additionally, it will review Locally Linear Embeddings (LLE) as well as t-distributed Stochastic Neighbor Embedding (t-SNE). Although LLE and t-SNE are not neural network models, they are enormously important in unsupervised learning and provide valuable intuitions for understanding coordinate representations of data manifolds. These intuitions translate well to intuitions required in Chapter 4 for neural network models.

2.2.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a bipartite, energy-based graphical model [59]. With the sigmoid activation function $\sigma(z) := \frac{1}{1+e^{-z}}$, the total energy of a joint configuration (v, h) (read: visible, hidden) is given by:

$$E(v,h) := -\sum_{i \in \text{visible}} a_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{2.13}$$

where $v_i, h_j \in \{0,1\}$ take binary values, a_i and b_j are biases and w_{ij} are weight connections between visible and hidden states. The probability of a certain energy configuration (v, h) is the following:

$$p(v,h) := \frac{1}{Z} e^{-E(v,h)} \tag{2.14}$$

Requiring the probability to sum to 1 requires the partition function $Z = \sum_{v,h} e^{E(v,h)}$ over all possible visible-hidden state combinations. Similarly, the probability of a visible state existing is $p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$. Maximizing the log-probability of this yields:

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}} \tag{2.15}$$

where the angled brackets denote expectations over the distributions of the subscripts. This gradient is used in the gradient descent search to optimize the model parameters. Gibb's sampling is used in estimating $\langle v_i h_j \rangle_{\text{model}}$ as well as $\langle v_i h_j \rangle_{\text{recon}}$, where $p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij})$ and $p(v_i = 1|h) = \sigma\left(a_i + \sum_j h_j w_{ij}\right)$ are used as the values of h_j and v_i. This algorithm for estimating the gradient in order to update

the parameters is called contrastive divergence.

It is noted that the notation (v, h) is usually used in literature. However, to be consistent with the notation used in the rest of this dissertation we will write states as $\left(x^{(l)}, x^{(l+1)}\right)$, as well as energy $E^{(l)}\left(x^{(l)}, x^{(l+1)}\right)$, which is a function of network parameters $a^{(l)}$, $b^{(l)}$ and $w^{(l)}$.

2.2.1.1 Deep Belief Networks

Deep Belief Networks (DBN) are stacked RBM's [59]. For energy states at layer l given by the following:

$$E^{(l)}\left(x^{(l)}, x^{(l+1)}\right) := -\sum_{i \in \text{visible}} a_i^{(l)} x_i^{(l)} - \sum_{j \in \text{hidden}} b_j^{(l)} x_j^{(l+1)} - \sum_{i,j} x_i^{(l)} x_j^{(l+1)} w_{ij}^{(l)} \tag{2.16}$$

This is initialized at the input for $l = 0$ and trained by contrastive divergence until convergence. The parameters are then held fixed as new data is passed through the previously trained network, creating input data for $l = 1$. This process is continued for as many stacked RBM's as are needed.

2.2.2 Autoencoders

The autoencoder [60] takes the following form:

$$x^{(1)} = f\left(W^{(0)} \cdot x^{(0)} + b^{(0)}\right) \tag{2.17}$$

$$\hat{x}^{(0)} = W^{(0)T} \cdot x^{(1)} + \tilde{b}^{(0)} \tag{2.18}$$

Equation 2.17 is the encoder while Equation 2.18 is the decoder. Note that weights $W^{(0)}$ are shared between the encoder and decoder, with dimensions matching because of the transpose $W^{(0)T}$. If $W^{(0)}$ is of size $n_1 \times n_0$, then for dimensionality reduction one takes $n_1 < n_0$. This is usually trained via an L_2-minimization between the input $x^{(0)}$ and its reconstruction $\hat{x}^{(0)}$.

For a de-noising autoencoder [8, 60, 61], a random selection (usually 50%) of the input rows of $W^{(0)}$ are set to 0. This requires the neural network to reconstruct $\hat{x}^{(0)}$ with only partially available information. This leads to the network learning sparser

features because the separate nodes cannot overly depend on each other to reconstruct the input [53].

2.2.2.1 Deep Autoencoders

A deep autoencoder [8, 60, 61] is an autoencoder with multiple hidden levels. For an autoencoder with L hidden layers, the deep autoencoder takes the following form:

$$x^{(1)} = f\left(W^{(0)} \cdot x^{(0)} + b^{(0)}\right) \tag{2.19}$$
$$x^{(2)} = f\left(W^{(1)} \cdot x^{(1)} + b^{(1)}\right) \tag{2.20}$$
$$\dots$$
$$x^{(L)} = f\left(W^{(L-1)} \cdot x^{(L-1)} + b^{(L-1)}\right) \tag{2.21}$$
$$\hat{x}^{(L-1)} = f\left(W^{(L-1)T} \cdot x^{(L)} + \tilde{b}^{(L-1)}\right) \tag{2.22}$$
$$\dots$$
$$\hat{x}^{(1)} = f\left(W^{(1)T} \cdot \hat{x}^{(2)} + \tilde{b}^{(1)}\right) \tag{2.23}$$
$$\hat{x}^{(0)} = W^{(0)T} \cdot \hat{x}^{(1)} + \tilde{b}^{(0)} \tag{2.24}$$

These deep autoencoders are pre-trained by first training a single layer autoencoder from Equations 2.17 and 2.18. Once that has been sufficiently trained, those parameters are held fixed and the encoder of Equation 2.17 is used to create a new input for a second single layer autoencoder. This process is repeated for as many layers as needed, and is very analogous to training a Deep Belief Network as a sequence of Restricted Boltzmann Machines. Finally, a fine-tuning scheme is used to minimize the L_2-error between the input and reconstruction of Equations 2.19 and 2.24.

2.2.3 Generative Adversarial Networks

Generative Adversarial Networks [10] (GAN's) will be briefly described because they are the state-of-the-art in unsupervised learning with neural networks. Even with very high resolution images [31] these models are able to generate very detailed images and linearly interpolate between these images. From the perspective of geometry, this means that the GAN is learning a flattened representation of the data manifold, since linear interpolation only makes sense on a flat surface; linear interpolation on a curved

surface will fall off of the surface and produce implausible looking images.

The goal of the network is to train two models, a generator $G(x)$ and a discriminator $D(x)$. The goal of the discriminator is to be able to distinguish if an image is real or fake, i.e. minimize $\mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))]$. Similarly, the goal of the generator is to trick the discriminator into thinking its input is real, i.e. minimize $\mathbb{E}_{z\sim p_z(x)}[\log(1-D(G(z)))]$. The overall cost function is then defined as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))] + \mathbb{E}_{z\sim p_z(x)}[\log(1-D(G(z)))] \quad (2.25)$$

In practice, instead of minimizing $\mathbb{E}_{z\sim p_z(x)}[\log(1-D(G(z)))]$ one maximizes $\mathbb{E}_{z\sim p_z(x)}[\log(D(G(z)))]$ as this does not saturate in early training like the minimization problem does.

2.2.4 Locally Linear Embeddings

Locally linear embeddings [22] is a non-parametric manifold learning technique based on k-nearest neighbors [62]. Being built on k-nearest neighbors has two main advantages. The first is it is an enormously flexible type of non parametric non-linearity and can thus model enormous varieties of manifolds. The second is it approximates each region linearly by its nearest neighbor embedding, and thus resembles fundamental definitions of differential geometry; namely local regions of a topological manifold are locally homeomorphic to $\mathbb{R}^n$.

The first step of locally linear embeddings is to minimize the following cost, where there are N data point $x_n \in \mathbb{R}^D$ that lie on or near a manifold of dimension $d < D$ and each data point is reconstructed by $k < D$ nearest neighbors.

$$E(W) = \sum_{n=1}^{N} \left\| x_n - \sum_{i=1}^{k} W_{ni} x_i \right\|^2 \quad (2.26)$$

subject to the constraints that 1.) $W_{ni} = 0$ if x_i is not a nearest neighbor of x_n, and 2.) $\sum_{i=1}^{k} W_{ji} = 1$. The second condition insures that the reconstructions are are invariant to rotations, rescalings and translations. These are exactly the conditions required by the metric axioms on a Riemannian manifold.

The second step is to reconstruct the lower dimensional representation $y_n \in \mathbb{R}^d$ for $d < D$, with the parameters W fixed from the first step.

$$\Phi(Y) = \sum_{n=1}^{N} \left\| y_n - \sum_{i=1}^{k} W_{ni} y_i \right\|^2 \tag{2.27}$$

This simple 2-step optimization procedure finds a non-parametric representation of the manifold that, importantly, preserves angles and distances in the embedding representation.

2.2.5 t-distributed Stochastic Neighbor Embedding

The t-Distributed Stochastic Neighbor Embedding algorithm, where the t stands for Student-t distribution, is a non-parametric, unsupervised technique for dimensionality reduction [63]. t-SNE is based on the Stochastic Neighbor Embedding [64] algorithm, but has two modifications. First, it uses a symmetric version of the SNE algorithm, which allows for simpler gradients and easier optimizations. Second, instead of using a Gaussian distribution in the embedded space, it uses a Student-t distribution which again allows for easier optimization and alleviates the overcrowding problem of the SNE algorithm.

The symmetrized cost function is given as follows:

$$E = KL(P|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.28}$$

where $p_{ii} = q_{ii} = 0$ is required. This is symmetric because between two points i and j, $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$. In both the symmetric SNE as well as t-SNE algorithms, the input conditional probability is measured by a Gaussian:

$$p_{i|j} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq l} \exp\left(-\|x_k - x_l\|^2 / 2\sigma_l^2\right)} \tag{2.29}$$

This becomes symmetric by defining $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$. For symmetric SNE, this same distribution is used in the embedding space, but for t-SNE the Student-t distribution is used:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}} \tag{2.30}$$

The intuition of using the Student-t distribution is that $\left(1 + \|y_i - y_j\|^2\right)^{-1}$ approxi-

mates an inverse square law for large pairwise distances $\|y_i - y_j\|$ in the embedding space. Maaten and Hinton discuss that [63] this is advantageous because joint probabilities are approximately invariant to scale at large distances. Additionally large clusters of points are measured and compared in the same way as individual points, and so at larger scales the optimization is invariant to cluster size.

The embedded data points are initialized randomly, and a gradient descent search over the embedded representation is performed until convergence:

$$\frac{\partial E}{\partial y_i} = 4 \sum_j \left(p_{ij} - q_{ij}\right) \left(y_i - y_j\right) \left(1 + \|y_i - y_j\|^2\right)^{-1} \tag{2.31}$$

In practice momentum as well as other tricks are used to make optimization easier. Several extensions of t-SNE exist. An interesting one in particular uses t-SNE to learn a non-parametric embedding representation, and then uses a neural network to parametrically map input data to the representation learned by t-SNE [65].

2.3 Training neural networks

This section will review, at the most basic level, the standard techniques used to train neural networks. Again this is by no means a complete review of how to properly train neural networks. For this there exist entire bodies of literature [66–70] detailing these ever evolving procedures. The purpose of this section is to familiarize the reader with the basic concepts.

2.3.1 Cost functions

Neural networks, or optimization processes in general, are often trained via a gradient descent search over the parameter manifold to minimize a certain cost [71]. The two most common costs, namely cross-entropy and L_2-error, are explained below.

2.3.1.1 Cross-entropy

The Kullback-Leibler divergence between two probability functions p and q is the number of bits required to construct q given p, thus it is similar to measuring the distance between the two distributions; it is defined as follows [72]:

$$D_{KL}\left(p||q\right) := -\sum_i p\left(i\right) \log\left(\frac{q\left(i\right)}{p\left(i\right)}\right) \tag{2.32}$$

This can be rewritten as follows, where $H\left(p\right) := -\sum_i p\left(i\right) \log p\left(i\right)$ is the Shannon Entropy:

$$D_{KL}\left(p||q\right) = -\sum_i p\left(i\right) \log q\left(i\right) - H\left(p\right) \tag{2.33}$$

In machine learning, p is the true distribution and q is the estimated distribution (where q depends on the parameters Θ). Minimizing the Kullback-Leibler divergence with respect to Θ is equivalent to minimizing $\sum_i p\left(i\right) \log q\left(i\right)$, since $H\left(p\right)$ is independent of q, and thus Θ. The cross-entropy between two distributions is defined as follows:

$$\text{XE}\left(p||q\right) = \sum_i p\left(i\right) \log\left(q\left(i\right)\right) \tag{2.34}$$

The cross-entropy is used when the output is to be interpreted as bits or a probability, such as with the softmax classifier. To have consistent notation with the rest of the dissertation, the cross-entropy loss is written $l\left(y, h_\Theta\left(x\right)\right) := \sum_i y^i \log\left(h_\Theta\left(x^{(0)}\right)^i\right)$, where y is the true distribution/label and $x^{(0)}$ is the input data.

2.3.1.2 L_2 error

The L_2 cost [73] between two points y and $h_\Theta\left(x^{(0)}\right)$ is defined as follows:

$$l\left(y, h_\Theta\left(x^{(0)}\right)\right) := ||y - h_\Theta\left(x^{(0)}\right)||_2^2 \tag{2.35}$$

where $||.||_2$ is the 2-norm. This is an important norm because, unlike other L_p norms, this norm is rotationally-invariant. The 2-norm cost is used when the output is to be interpreted as a point, such as reconstructing an image with an autoencoder.

2.3.2 Gradient descent

The purpose of the gradient descent search is to find the parameter value Θ such that the cumulative loss is minimized over the dataset $D := \{(x_n, y_n)\}_{n=1}^N$:

$$\Theta^* := \arg_{\theta\in\Theta}\min E\left(\Theta\right) := \arg_{\theta\in\Theta}\min \frac{1}{N}\sum_{n=1}^{N} l\left(y_n, h_\Theta\left(x_n^{(0)}\right)\right) \tag{2.36}$$

The gradient descent search updates the parameters $\Theta^{t+1} \leftarrow \Theta^t - \lambda\frac{\partial E}{\partial \Theta}^t$ for epoch t and learning rate λ , which takes a typical value of 0.001. To include a learning rate-decay of decay rate 0.98, one uses $\Theta^{t+1} \leftarrow \Theta^t - 0.98^t\lambda\frac{\partial E}{\partial \Theta}^t$. Additionally, one can include a momentum term of 0.995 into the update rule to average over updates as $\Theta^{t+1} \leftarrow \Theta^t - 0.98^t\lambda\left(0.005\frac{\partial E}{\partial \Theta}^t + 0.995\frac{\partial E}{\partial \Theta}^{t-1}\right)$. Efficiently calculating these derivatives $\frac{\partial E}{\partial \Theta}$ requires the error backpropagation algorithm.

2.3.3 Error Backpropagation

In Chapter 4 a substantial generalized reformulation of the error backpropagation algorithm in a coordinate-free, differential geometric sense is shown. Because of that, this subsection will derive the error backpropagation algorithm in the usual way [73,74].

For a given loss $l\left(y, h_\Theta\left(x^{(0)}\right)\right)$, the error is defined as the expected loss over all data samples. Assuming the data samples are iid, the total error is:

$$E\left(\Theta\right) := \frac{1}{N}\sum_{n=1}^{N} l\left(y, h_{\Theta}\left(x^{(0)}\right)\right) \tag{2.37}$$

The purpose of the error backpropagation algorithm is to efficiently find the derivatives $\frac{\partial E}{\partial \Theta}$ for the gradient descent search $\Theta^{t+1} \leftarrow \Theta^{t} - \lambda \frac{\partial E}{\partial \Theta}$ (up to decay rates, momentum terms and other tricks). Neural networks are nested compositions of non-linear functions, and so the derivatives $\frac{\partial E}{\partial \Theta}$ are found by simply applying the ordinary chain rule of differential calculus sequentially.

At the output, the gradient of the parameters weights are directly calculated:

$$\frac{\partial E}{\partial W^{(L-1)}} = \frac{\partial E}{\partial x^{(L)}} \cdot \frac{\partial x^{(L)}}{\partial W^{(L-1)}} \tag{2.38}$$

where $x^{(L)} := h_{\Theta}\left(x^{(0)}\right)$ and the $\cdot$ is standard matrix multiplication. In practice, one further expands Equation 2.38 by writing $\frac{\partial x^{(L)}}{\partial W^{(L-1)}} = \frac{\partial x^{(L)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}}$, although this becomes notationally burdensome for the purpose of this dissertation. The gradients over the parameter weights for an arbitrary layer $l-1$ can be found by further applying the chain rule on $\frac{\partial x^{(L)}}{\partial W^{(l-1)}}$ as follows:

$$\frac{\partial x^{(L)}}{\partial W^{(l-1)}} = \prod_{l'=L-1}^{l} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right) \cdot \left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right) \tag{2.39}$$

where $\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)$ is a Jacobian matrix and the $\prod_{l'=L-1}^{l}$ operator is sequential matrix multiplication. Instering Equation 2.39 into Equation 2.38 yields the error backpropagation algorithm.

$$\frac{\partial E}{\partial W^{(l-1)}} = \left(\frac{\partial E}{\partial x^{(L)}}\right) \cdot \prod_{l'=L-1}^{l} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right) \cdot \left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right) \tag{2.40}$$

In this form, one has a closed form solution for the gradient with respect to parameter weights at any internal layer $l-1$. Because it is a recursive formula from output to input on the error gradient it is backpropagating errors via the chain rule, thus the rationale for the name of the algorithm.

Chapter 3
Smooth Manifolds

The primary contribution of this dissertation is mathematically formulating neural networks as a branch of Riemannian geometry. The required background knowledge in geometry to rigorously construct this formulation, which is the geometrical anatomy of neural networks, will be reviewed in this chapter.

This chapter will very briefly review spaces with increasing levels of structure. This is to ensure no ambiguity in the later formulations.

3.1 Topological Spaces

Endowing a set with a topology [75–77] is the weakest additional structure on sets that allows the concepts of convergence, connectedness and continuity. These are extremely important concepts, and are required for rigorously discussing manifolds. Additionally, these extremely basic properties are experimentally tested in Chapter 4 and shown to fail in common circumstances.

A topological space is first defined:

Definition 3.1.1. (Topological space) A set M is made into a topological space $(M, \mathcal{O})$ by choosing a collection $\mathcal{O} \subseteq \mathcal{P}(M)$ of subsets of M, provided the choice has been made such that:

1. the trivial subsets are open: $\phi, M \in \mathcal{O}$
2. finite intersections of open sets are open: $U, V \in \mathcal{O} \Rightarrow U \cap V \in \mathcal{O}$
3. arbitrary unions of open sets are open: $\forall \alpha \in A : U_\alpha \in \mathcal{O} \Rightarrow \cup_{\alpha \in A} U_\alpha \in \mathcal{O}$.

There exist many topologies on sets. The topology that allows for the standard formulations of limits in calculus on manifolds is called the standard topology. Only for finite sets can explicit topologies be defined; for infinite sets topologies must be defined implicitly.

Definition 3.1.2. (Standard topology) A subset $U \subset \mathbb{R}^d$ is called open in the standard topology $\mathcal{O}_s$ on $\mathbb{R}^d$ if $\forall p \in U \exists \epsilon > 0 : B_\epsilon(p) := \{q \in \mathbb{R}^d : \sqrt{\sum_{i=1}^{d} (p^i - q^i)^2} < \epsilon\} \subset U$.

Note that one must show that the standard topology $\mathcal{O}_s$ of Definition 3.1.2 satisfies the requirements of Definition 3.1.1 of being a topological space.

In practice when endowing a set with a topology, one usually induces a topology onto the set from a superset. The induced topology is thus defined:

Definition 3.1.3. (Induced topologies) Let $(M, \mathcal{O})$ be a topological space. Let $N \subset M$ Then $\mathcal{O}|_N := \{U \cap N | U \in \mathcal{O}\} \subseteq \mathcal{P}(N)$ is a topology on N, called the induced (subset) topology.

When given two topological spaces, one can take the Cartesian product of the two sets and define a topology on the product set by the product topology.

Definition 3.1.4. (Product topologies) Let $(A, \mathcal{O}_A)$ and $(B, \mathcal{O}_B)$ be topological spaces. Equip $A \times B$ with the product topology $\mathcal{O}_{A\times B}$ defined implicitly by: $U \in \mathcal{O}_{A\times B} : \iff \forall p \in U \exists S \in \mathcal{O}_A \exists T \in \mathcal{O}_B : S \times T \subseteq U$, where $p = (a, b) \in A \times B$ and $a \in S$ and $b \in T$.

Note that one must show that the product topology of Definition 3.1.4 satisfies the requirements of a topological space from Definition 3.1.1

As mentioned earlier, endowing a set with a topology is the weakest additional structure needed to allow for the concepts of convergence, connectedness and continuity.

Definition 3.1.5. (Convergence) A sequence $q : \mathbb{N} \to M$ on a topological space $(M, \mathcal{O})$ is said to converge against a limit point $a \in M$ if $\forall U \in \mathcal{O}$ where $a \in U$ $\exists N \in \mathbb{N} : \forall n > N : q(n) \in U$. The neighborhood $U \ni a$ is called an open neighborhood of a.

Informally, a set is connected if any two points in the set can be connected by a curve on the set. The formal topological definition is given below:

Definition 3.1.6. (Connectedness) A topological space $(M, \mathcal{O})$ is said to be disconnected if there exists two disjoint, non-empty open sets $U, V \in \mathcal{O}$ such that $U \cap V = \phi$ and $U \cup V = M$. Otherwise it is connected.

In set theory, sets are equivalent if there exists a bijection between them. In topology, topological spaces are equivalent if there exists a homeomorphism between them. Continuity is needed to formulate these structure preserving maps in the topology.

Definition 3.1.7. (Continuous function) A map $\varphi : M \to N$ between two topological spaces $(M, \mathcal{O}_M)$ and $(N, \mathcal{O}_N)$ is called a continuous function if $\text{preim}_{\varphi}(V) := \{p \in M | \varphi(p) \in V\} \in \mathcal{O}_M$ for every $V \in \mathcal{O}_N$. A bijection φ is called a homeomorphism if both φ and φ^{-1} are continuous. Two topological spaces between which there exists a homeomorphism are called homeomorphic.

By choosing the topologies in Definition 3.1.7 to be the standard topology of Definition 3.1.2, one obtains the standard $\epsilon - \delta$ definition of continuity in analysis.

The following definitions are needed to define manifolds.

Definition 3.1.8. (Cover) A cover of a set M is a set $\mathcal{U} = \{U_\alpha \subseteq M : \alpha \in A\}$ that is an indexed family of subsets U_α of M such that $M \subseteq \cup_{\alpha \in A} U_\alpha$. The cover is called an open cover if $U_\alpha \in \mathcal{O}$ for all $\alpha \in A$, where $\mathcal{O}$ is the topology on M. The cover $\mathcal{V} = \{V_\beta : \beta \in B\}$ is a called a refinement of the cover $\mathcal{U} = \{U_\alpha \subseteq M : \alpha \in A\}$ if and only if $\forall V_\beta \in \mathcal{V}$ there exists $U_\alpha \in \mathcal{U}$ such that $V_\beta \subseteq U_\alpha$.

Definition 3.1.9. (Paracompact) An open cover $\mathcal{U} = \{U_\alpha \subseteq M : \alpha \in A\}$ of M is locally finite if $\forall p \in M$ there exists $V \ni p$ such that the set $\{\alpha \in A : U_\alpha \cap V \neq \phi\}$ is finite. A topological space $(M, \mathcal{O})$ is called paracompact if every open cover has a locally finite open refinement.

Definition 3.1.10. (Hausdorff) A topological space $(M, \mathcal{O})$ is called Hausdorff if $\forall p, q \in M$ that are distinct there exists open sets $U, V \in \mathcal{O}$, where $p \in U$ and $q \in V$, such that $U \cap V = \phi$.

3.2 Topological Manifolds

Roughly speaking, a topological space that is locally homeomorphic to $\mathbb{R}^d$ is called a topological manifold. The objective of this section is to formally introduce manifolds and their increasing levels of structure [78–80].

3.2.1 Manifolds

This subsection introduces topological manifolds, which is the most important mathematical structure in this thesis, and so it follows that it is very important to clearly understand what they are.

Definition 3.2.1. (Topological Manifold) A paracompact, Hausdorff topological space $(M, \mathcal{O})$ that is locally homeomorphic to $\mathbb{R}^{\dim M}$ is called a $\dim M$-dimensional topological manifold. The homeomorphism $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$ is called a coordinate map, and the pair (x, U) is called a coordinate system.

Manifold are required to be paracompact and Hausdorff so that they share many of the basic properties of Euclidean spaces. For example in Hausdorff spaces, one-point sets are closed and limits of convergent sequences are unique. General topological spaces, such as discrete and chaotic topologies, have very strange properties that are not shared with our usual intuitions of calculus on manifolds.

Often one calls a topological manifold a manifold and leaves out the predicate "topological". The homeomorphism x between $U \in \mathcal{O}$ and $x(U) \subseteq \mathbb{R}^{\dim M}$ is a coordinate map on $U \subseteq M$. This makes it so that, even if the entire set M is not Euclidean, it is locally Euclidean on $U \subseteq M$. Collecting the coordinate systems into a set forms an atlas.

Definition 3.2.2. (Atlas) A $\mathcal{C}^k$ differential atlas on a topological, $\dim M$-dimensional manifold M is a collection $\mathcal{A}$ of charts of M such that:

1. The domains $U \in \mathcal{O}$ of the chart cover M, and
2. If (U, x) and $(V, y) \in \mathcal{A}$ such that $U \cap V \neq \phi$, then $y \circ x^{-1} : x(U \cap V) \to \mathbb{R}^{\dim M}$ is k-times differentiable.

The map $y \circ x^{-1}$ is called a coordinate transformation or chart transition map from the chart (U, x) to (V, y), assuming $U \cap V \neq \phi$. Once we are in charts $x : U \cap V \to x(U \cap V)$

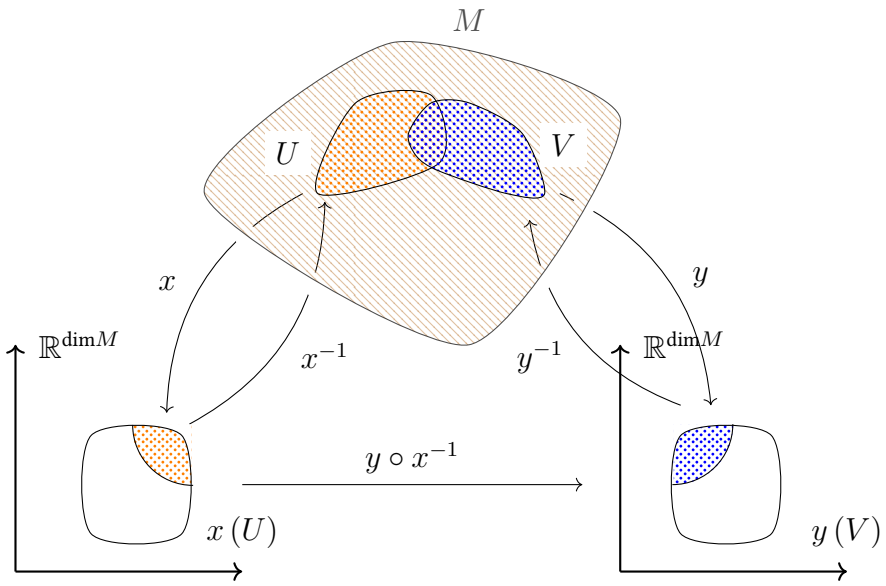


Figure 3.1: A topological manifold $(M, \mathcal{O})$ with coordinate systems (U, x) and (V, y) such that $U \cap V \neq \phi$. The coordinate transformation $y \circ x^{-1} : x\,(U \cap V) \to \mathbb{R}^{\dim M}$ is k-times differentiable for a $\mathcal{C}^k$-differentiable manifold.

or $y : U \cap V \to y\,(U \cap V)$, knowledge of continuity when transitioning between charts with the map $y \circ x^{-1} : x\,(U \cap V) \to y\,(U \cap V)$ only requires knowledge of the topologies on $x\,(U \cap V)$ and $y\,(U \cap V)$, *not* the topology on $U \cap V$. This is important because $x\,(U \cap V)\,, y\,(U \cap V) \subseteq \mathbb{R}^{\dim M}$, and so we can induce the standard topology from $\mathbb{R}^{\dim M}$ onto the sets $x\,(U \cap V)$ and $y\,(U \cap V)$.

Several definitions similar to Definition 3.2.2 are itemized below:

- If the charts are $\mathcal{C}^\infty$, then the charts are smooth.
- If the charts are $\mathcal{C}^\omega$, then the charts are analytic (vulgo: Taylor expandable).
- If the charts are complex, then $y \circ x^{-1}$ satisfy the Cauchy-Riemann equations.

Figure 3.1 is helpful for visualizing the definitions on manifolds so far. A manifold M is locally homeomorphic to $\mathbb{R}^{\dim M}$ via the maps $x : U \subseteq M \to x\,(U) \subseteq \mathbb{R}^{\dim M}$ and $y : V \subseteq M \to y\,(V) \subseteq \mathbb{R}^{\dim M}$. The charts (U, x) and (V, y) are in the atlas $\mathcal{A}$. Just as with charts in an everyday sense, when referencing a map of Manhattan, U, which is a subset of the Earth, M, one goes to a specific page of the atlas $\mathcal{A}$ and finds a relevant chart map (U, x) to see the chart representation of Manhattan $x\,(U)$. Additionally, the atlas must be consistent in the sense that the two regions Manhattan,

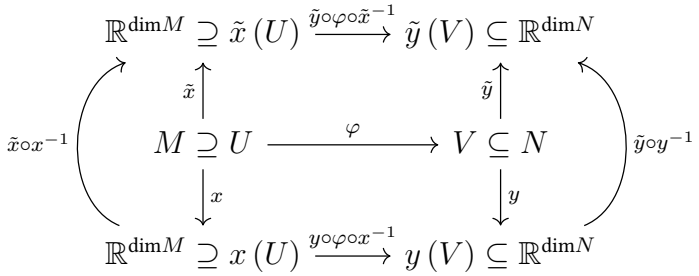


Figure 3.2: Let $\varphi : M \to N$ be a map, where $(M, \mathcal{O}_M, \mathcal{A}_M)$ and $(N, \mathcal{O}_N, \mathcal{A}_N)$ are $\mathcal{C}^k$-manifolds. Then φ is called differentiable at $p \in M$ if for some $(U, x) \in \mathcal{A}_M$ with $U \ni p$ and some chart $(V, y) \in \mathcal{A}_N$ with $V \ni \varphi(p)$, the map is $\mathcal{C}^k$ as a map from $y \circ \varphi \circ x^{-1} : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim N}$. Displayed as well is the means of transforming coordinates from x and y to $\tilde{x}$ and $\tilde{y}$ to produce the map $\tilde{y} \circ \varphi \circ \tilde{x}^{-1} : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim N}$.

U, and Brooklyn, V, line up where they overlap $U \cap V$ in the chart representation $x(U \cap V)$ and $y(U \cap V)$. The level of consistency between chart maps is determined by the chart transition map $y \circ x^{-1} : x(U \cap V) \subseteq \mathbb{R}^{\dim M} \to y(U \cap V) \subseteq \mathbb{R}^{\dim M}$ is e.g. k-times differentiable for a $\mathcal{C}^k$-differentiable manifold.

An important technique used to construct manifolds is to construct them from other manifolds. First we define the submanifold.

Definition 3.2.3. (Submanifold) Let $(M, \mathcal{O})$ be a d-dimensional topological manifold and $N \subseteq M$, then $(N, \mathcal{O}|_N)$ is called a submanifold of $(M, \mathcal{O})$ if it is also a manifold.

A second important way of constructing manifolds is by taking the product of two existing manifolds, thus defining the product manifold.

Definition 3.2.4. (Product manifold) Let $(M, \mathcal{O}_M)$ and $(N, \mathcal{O}_N)$ be $\dim M$ and $\dim N$-dimensional topological manifolds, then $(M \times N, \mathcal{O}_{M\times N})$ is a topological manifold of dimension $\dim M + \dim N$, and is called a product manifold.

In principle one must check that these two definitions satisfy the definition of a manifold.

Definition 3.2.5. (Maximal atlas) A maximal atlas $\mathcal{A}_{\text{max}}$ is an at atlas that contains all $\mathcal{C}^0$ charts.

Theorem 3.2.1. (Whitney) Any maximal $\mathcal{C}^k$-atlas for $k \geq 1$ contains a $\mathcal{C}^\infty$-atlas. And two maximal $\mathcal{C}^k$-atlases that contain the same $\mathcal{C}^\infty$-atlas are already identical.

Remark. (Whitney) The implication of this Whitney theorem is that one does not need to distinguish between $\mathcal{C}^k$ for some $k \geq 1$ and $\mathcal{C}^\infty$, in the above sense.

This theorem suggests we define the following:

Definition 3.2.6. ($\mathcal{C}^k$-manifold) A $\mathcal{C}^k$-manifold is a tuple $(M, \mathcal{O}, \mathcal{A})$ where $(M, \mathcal{O})$ is a topological manifold and $\mathcal{A}$ is a maximal $\mathcal{C}^k$-atlas.

Between two topological manifolds, differentiability can be defined on the chart level with the following definition and the corresponding Figure 3.2.

Definition 3.2.7. (Differentiability between manifolds) Let $\varphi : M \to N$ be a map, where $(M, \mathcal{O}_M, \mathcal{A}_M)$ and $(N, \mathcal{O}_N, \mathcal{A}_N)$ are $\mathcal{C}^k$-manifolds. Then φ is called differentiable at $p \in M$ if for some $(U, x) \in \mathcal{A}_M$ with $U \ni p$ and some chart $(V, y) \in \mathcal{A}_N$ with $V \ni \varphi(p)$, the map is $\mathcal{C}^k$ as a map from $y \circ \varphi \circ x^{-1} : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim N}$.

This can then be used to define diffeomorphisms, which are the structure-preserving-maps across differentiable manifolds.

Definition 3.2.8. (Diffeomorphism) With the conditions from Definition 3.2.7, and $\varphi : M \to N$ is bijective and both φ and φ^{-1} are $\mathcal{C}^\infty$, then φ is called a diffeomorphism. In this case, $(M, \mathcal{O}_M, \mathcal{A}_M)$ and $(N, \mathcal{O}_N, \mathcal{A}_N)$ are called diffeomorphic.

3.2.2 Bundles

A bundle is a very important structure in geometry because it generalizes the Cartesian product in many ways. For example, the manifold $\mathbb{R}^d$ can be written (in fact is defined) as $\mathbb{R}^d := \mathbb{R} \times \ldots \times \mathbb{R}$ d-times. This is not always the case where a manifold can be written simply as a Cartesian product, such as the Mobius strip seen in Figure 3.3. The Mobius strip is only *locally* Euclidean. In cases like this, the generalization of the Cartesian product, namely the fibre bundle, is required. First we need several definitions.

Definition 3.2.9. (Bundle) A bundle is a tuple (E, π, B) in which E and B are topological manifolds and $\pi : E \to B$ is a continuous surjective map. E is called the entire space or total space, B is called the base space and π is called the projection map.

Because E and B are topological manifolds they must have topologies, and the map $\pi : E \to B$ can be understood as a map between topological spaces; thus there exists topological notions of continuity for bundles.

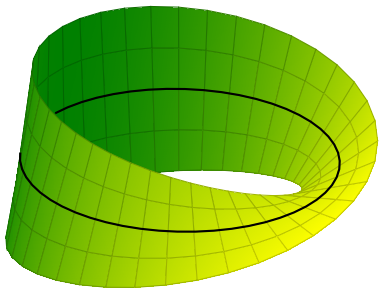


Figure 3.3: The Mobius strip is an example of a manifold that is not globally homeomorphic to $\mathbb{R}^d$. As a bundle, where the base space is S^1 and the fibres are the intervals $[-1, 1]$, it cannot be written globally as a Cartesian product, i.e. it is not a trivial fibre bundle.

A trivial, yet very important example is if M and F are topological manifolds, then define $E := M \times F$ (globally) and $\pi : M \times F \to M$ by $\pi(p, f) = p$, where π is continuous. In this case the projection map π is said to project down down to the first factor.

A nontrivial example is seen in Figure 3.3, if E is the Mobius strip of width from $[-1, 1]$, $M = S^1$ and $\pi : E \to M$ maps points along the width of the Mobius strip $[-1, 1]$ to the center of the strip at 0, which is a point $p \in M = S^1$ of the base space. Locally this is Euclidean and can be written as a product manifold as $U \times [-1, 1]$ for $U \subset M$, but globally it cannot be written as a Cartesian product and thus is only a bundle.

Additional definitions are needed for useful geometric constructions. First the definition of the fibre, which formalizes the notion of the width $[-1, 1]$ of the Mobius strip. Later on the fibre is the space in which the tangent vectors will lie.

Definition 3.2.10. (Fibre) Let (E, π, M) be a bundle and $p \in M$, then the set $F_p := \pi^{-1}(\{p\}) \equiv \text{preim}_\pi(\{p\}) := \{e \in E | \pi(e) = p\}$ is called the fibre at p.

The notion of a fibre is important because Definition 3.2.9 for a bundle does not require the entire space to be constructed from the base space. A fibre is built on the base manifold and the combination forms the entire space in a fibre bundle, as seen in Figure 3.4.

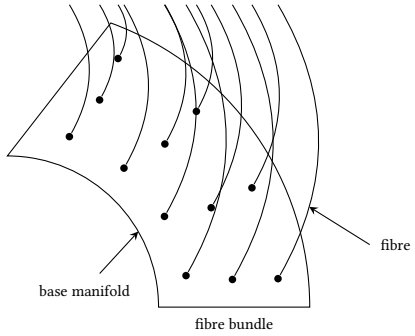


Figure 3.4: A fibre bundle (E, π, M, F). The base manifold is M, the fibres are F and the entire space E is the fibre bundle. In this figure, $\dim M = 2$, $\dim F = 1$ and $E = M \times F$ so this is a trivial fibre bundle. At a given point $p \in M$ the corresponding fibre is $F_p = \pi^{-1}(\{p\})$. Along this fibre, the projection map $\pi : E \to M$ takes any point on the fibre and maps it to the base point on the manifold $p \in M$. A section is a map $\sigma : M \to E$ such that $\pi \circ \sigma = 1_M$, and in this figure would be any 1-dimensional fibre-valued field on the base manifold.

Fibres and bundles however are still a very general structure and very strange things can be considered as satisfying these definitions. For example, at different points on the base space one can have manifolds of different topologies, such as $\mathbb{R}$, $\mathbb{C}$, S^1, S^2, $S^1 \times \mathbb{R}$, $S^1 \times S^1$ etc. Specifically, a bundle can have the form where $M = \mathbb{R}$ and $\pi^{-1}(\{p\}) = S^1 =: F_p$, $\pi^{-1}(\{q\}) = \{0\} =: F_q$ and $\pi^{-1}(\{r\}) = \mathbb{R} =: F_r$, then $F_p \ncong F_q \ncong F_r$. This motivates the definition of the Fibre bundle.

Definition 3.2.11. (Fibre bundle) A fibre bundle is a tuple (E, π, B, F) in which E, B and F are topological manifolds and $\pi : E \to B$ is a surjective map and $\forall p \in M$: $\pi^{-1}(\{p\}) \cong F$ for some manifold F. This is called a fibre bundle with typical fibre F.

In the special case where the entire space is just the ordered product of the base space with the fibre, this is called a trivial fibre bundle.

Definition 3.2.12. (Trivial fibre bundle) A trivial fibre bundle is a fibre bundle (E, π, B, F) in which $E = B \times F$ everywhere.

Remark. (Fibre bundle) A helpful way to understand these definitions is with vector fields on the surface of a sphere S^2. A vector field on the surface of a sphere is a section of a fibre bundle, as at each basepoint $p \in B = S^2$ there exists a vector $v \in F = V$ such that $\sigma(p) = (p, v) \in U \times F \subseteq E$. A different section can give a different vector field. The surjective map $\pi : E \to B$ is then defined as $\pi(p, v) := p$, as projecting the

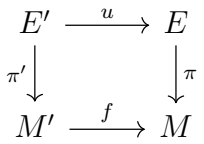


Figure 3.5: Two bundles (E, π, M) and (E', π', M') are said to be isomorphic as bundles if this diagram commutes, i.e. there exists bijective, continuous maps $u : E' \to E$ and $f : M' \to M$ such that $\pi \circ u = f \circ \pi'$.

pair down to the basepoint. If the section is a metric field satisfying the metric axioms, then over the base manifold there is a notion of distance, and thus shape, to the base space $B = S^2$. This is how the notion of shape is formally introduced to a topological manifold, and thus the defining characteristic of a Riemannian manifold.

The purpose of these definitions is to formally introduce and define tensor fields on the tangent spaces of manifolds. A tensor field on the tangent space of a manifold is called a section of a fibre bundle.

Definition 3.2.13. (Section) A section on a fibre bundle (E, π, B, F) is a map $\sigma : B \to E$ such that $\pi \circ \sigma = 1_B$, where 1_B is the identity map on B.

Remark. (Section) A section on a fibre bundle (E, π, B, F) is a map $\sigma : B \to E$ such that $\pi \circ \sigma = 1_B$, where 1_B is the identity map on B. This takes a basepoint $p \in B$ and maps it to the entire space $\sigma : p \to (p, v)$. A different section σ' on the same fibre bundle can map the same basepoint $p \in B$ to a different point in the entire space $\sigma' : p \to (p, v')$, which corresponds to a different vector field.

Remark. (Local representation of a section) Because a fibre bundle (E, π, B, F) is *locally* trivial, i.e. the product of the base space with the fibres on the base space, one can define a function locally for the section $\sigma : U \subset B \to U \times F$ such that $\pi \circ \sigma = 1_B$, where 1_B is the identity map on B. The section in this restricted region is then a function $\sigma : p \to (p, s\,(p))$, where the function is $s : U \to F$. This is true only locally, because globally E is not a product

The structure preserving maps of bundles are called bundle isomorphisms, and holds if Figure 3.5 commutes.

Definition 3.2.14. (Bundle isomorphism) Two bundles (E, π, M) and (E', π', M') are said to be isomorphic as bundles if there exists bijective, continuous maps $u : E' \to E$ and $f : M' \to M$ such that $\pi \circ u = f \circ \pi'$.

The bundle isomorphism exists if Figure 3.5 commutes. In this case, these two bundles (E, π, M) and (E', π', M') are said to be isomorphic as bundles and the maps (u, f) are called bundle isomorphisms, and are the relevant structure-preserving maps for bundles.

Definition 3.2.15. (Pull-back bundle) Consider a bundle (E, π, M), a manifold M' and map $f : M' \to M$. The we can construct the pull-back bundle (E', π', M') as $E' := \{(m', e) \in M' \times E | \pi(e) = f(m')\}$, where $\pi' : E' \to M'$ defined by $\pi'(m', e) := m'$ and $u : E' \to E$ defined by $u(m', e) := e$.

For Definition 3.2.15 of the pull-back bundle, see Figure 3.5 in which the space E' is *defined* by the other pieces of information in the bundle isomorphism.

Remark. (Sections on a bundle pull back to the pull-back bundle) A section $\sigma : M \to E$ on (E, π, M) can be pulled back to form a section $\sigma' : M' \to E'$ on the bundle (E', π', M').

3.2.3 Tangent Spaces

The key feature of differentiable manifolds is in the existence of tangent spaces at each point on the surface of the manifold. This allows the construction of tensor fields on the surface of manifolds, which builds upon the notion of vector fields on $\mathbb{R}^d$ which exists in standard calculus. In order to do this, several definitions are needed.

An important distinction to motivate key definitions used in standard calculus and differentiable geometry is that in differentiable geometry, one wishes to define derivatives in a general, coordinate free way. For example in $\mathbb{R}^2$, one defines the partial derivative of a function $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ at the point (x^1, x^2) as $\partial_1 f(x^1, x^2) := \lim_{\Delta x \to 0} \frac{f(x^1 + \Delta x, x^2) - f(x^1, x^2)}{\Delta x}$. This explicitly depends on the Cartesian coordinate system since the limit is being taken along a curve passing along the x^1-axis. This may make sense in $\mathbb{R}^d$, but on a general manifold there is no preferred direction. One way to generalize the notion of a derivative so that it does not need to occur along coordinate-lines is to define the directional derivative. In standard calculus the directional derivative is the standard gradient dotted with the direction u, i.e. $\partial_u f(p) := (u \cdot \nabla f)(p)$ however this definition still has a dependence on the coordinates through the standard gradient. In order to remove this dependence, we will have to change the definition and define the gradient as $\partial_u f(p) := \lim_{t \to 0} \frac{f(p+tu) - f(p)}{t}$, which makes no reference to coordinates.

To fully remove the dependence on coordinates, first note that we can construct a vector space of functions over $\mathbb{R}$. Let M be a smooth manifold and the vector space of continuous functions over $\mathbb{R}$ is $\left(\mathcal{C}^\infty(M), +_{\mathcal{C}^\infty(M)}, \cdot_{\mathcal{C}^\infty(M)}\right)$ where $+_{\mathcal{C}^\infty(M)}$ and $\cdot_{\mathcal{C}^\infty(M)}$ are defined pointwise, i.e. $+_{\mathcal{C}^\infty(M)} : \mathcal{C}^\infty(M) \times \mathcal{C}^\infty(M) \to \mathcal{C}^\infty(M)$ defined by $\left(f +_{\mathcal{C}^\infty(M)} g\right)(p) := f(p) +_{\mathbb{R}} g(p)$ and $\cdot_{\mathcal{C}^\infty(M)} : \mathbb{R} \times \mathcal{C}^\infty(M) \to \mathcal{C}^\infty(M)$ defined by $\left(\lambda \cdot_{\mathcal{C}^\infty(M)} f\right)(p) := \lambda \cdot_{\mathbb{R}} f(p)$, where $f, g \in C^\infty(M)$ are C^∞ maps $f, g : M \to \mathbb{R}$, $p \in M$ and $\lambda \in \mathbb{R}$.

Now to define the derivative two maps are needed, as seen in Figure 3.6. First a curve $\gamma : \text{preim}_\gamma(U) \to U \subseteq M$ which takes a point $t \in \text{preim}_\gamma(U) \subseteq \mathbb{R}$ that parameterizes the curve and maps it to a point $\gamma(t) \in U \subseteq M$ on the manifold. This is the curve along which the derivative is taken. The second map that is needed is the function $f : U \to \mathbb{R}$ for which the derivative is taken. The composition of these two maps $f \circ \gamma : \mathbb{R} \to \mathbb{R}$ is easily understood, is not dependent on coordinates and the derivative of this function can be defined as the ordinary derivative of standard single variable calculus $(f \circ \gamma)'$. This motivates the following, very important definition of the derivative in differentiable geometry.

Definition 3.2.16. (Tangent vectors) Let $f : U \to \mathbb{R}$ be a smooth, real-valued function on $U \subseteq M$ and $\gamma : \text{preim}_\gamma(U) \subseteq \mathbb{R} \to U$ be a smooth curve on $U \subseteq M$ such that $\gamma(0) = p \in M$. Define the linear derivative operator $X_{\gamma,p} : \mathcal{C}^\infty(M) \tilde{\to} \mathbb{R}$ at p along the curve γ as the linear map $X_{\gamma,p} : f \to X_{\gamma,p}(f)$ defined by:
$X_{\gamma,p}(f) \equiv X_{\gamma,p} f := (f \circ \gamma)'(0) \equiv \frac{d}{dt}|_{t=0}(f \circ \gamma) \equiv \frac{d(f \circ \gamma)}{dt}(0)$.

In differential geometry, $X_{\gamma,p}$ is usually called the tangent vector to the curve γ at the point $p \in M$. It must be shown that $X_{\gamma,p}$ does in fact form a vector space.

Remark. (Smooth curves) A curve $\gamma : \text{preim}_\gamma(U) \to U$ is understood to be continuous and smooth in the topological sense because both U and $\text{preim}_\gamma(U)$ are topological manifolds.

Remark. (Linearity of vector field) One must show that the linear derivative operator $X_{\gamma,p} : \mathcal{C}^\infty(M) \tilde{\to} \mathbb{R}$ at p along the curve γ is in fact linear.

Remark. (Directional derivative) In differential geometry, there is no independent notion of a derivative and a directional derivative. The derivative itself is directional. As mentioned in the opening remarks to motivate this section, this is because there is no preferred direction on a general manifold.

Taking the collection of all tangent vectors at the point $p \in M$ is used for the following very important definition of the tangent vector space at $p \in M$.

Definition 3.2.17. (Tangent vector space) The tangent vector space of a manifold M at the point $p \in M$ is defined as the tuple $\left(T_pM, +_{T_pM}, \cdot_{T_pM}\right)$, where the space $T_pM := \{X_{\gamma,p} | \gamma \text{ smooth curve through } p \in M\}$ is equipped with the addition and scalar multiplication operators $+_{T_pM}$ and $\cdot_{T_pM}$, defined pointwise as follows:

- $+_{T_pM} : T_pM \times T_pM \to T_pM$ defined by:

$$\left(X_{\gamma,p} +_{T_pM} X_{\delta,p}\right)(f) := X_{\gamma,p}(f) +_{\mathbb{R}} X_{\delta,p}(f)$$

- $\cdot_{T_pM} : \mathbb{R} \times T_pM \to T_pM$ defined by:

$$\left(\lambda \cdot_{T_pM} X_{\gamma,p}\right)(f) := \lambda \cdot_{\mathbb{R}} X_{\gamma,p}(f)$$

Remark. (Tangent vectors) The derivative operator elements $X_{\gamma,p} \in T_pM$ are called tangent vectors because they satisfy the axiomatic conditions of vector spaces.

From this point forward, the standard notation X will often be used to denote a vector on the manifold, as opposed to $X_{\gamma,p}$. Therefore it is implicitly understood that whenever we speak of a vector X, it is at a point $p \in M$ of the manifold along a curve $\gamma : \text{preim}_{\gamma}(U) \to U$ where $\gamma(0) = p$.

Definition 3.2.18. (Algebra) A K-vector space $(V, +, \cdot)$ equipped with a product, i.e. a bilinear map $\bullet : V \times V \tilde{\to} V$ is called an algebra $(V, +, \cdot, \bullet)$.

For example $\left(\mathcal{C}^\infty(M), +_{\mathcal{C}^\infty(M)}, \cdot_{\mathcal{C}^\infty(M)}\right)$ is an $\mathbb{R}$-vector space, and define $\bullet_{\mathcal{C}^\infty(M)} : \mathcal{C}^\infty(M) \times \mathcal{C}^\infty(M) \to \mathcal{C}^\infty(M)$ by $\bullet : (f, g) \mapsto \bullet_{\mathcal{C}^\infty(M)}(f, g) := f \bullet_{\mathcal{C}^\infty(M)} g$ where $\left(f \bullet_{\mathcal{C}^\infty(M)} g\right)(p) := f(p) \cdot_{\mathbb{R}} g(p)$ is defined pointwise. Note that $\cdot_{\mathcal{C}^\infty(M)}$ is *scalar*-multiplication, while $\bullet_{\mathcal{C}^\infty(M)}$ is *function*-multiplication. These are very obvious definitions, but it is important to understand at what structural level one is working.

The abstract definition of the derivative is the derivation. To perform a derivation of a function is to derive the derivation.

Definition 3.2.19. (Derivation) A derivation D on an algebra A to an algebra B is a linear map $D : A \tilde{\to} B$ which additionally satisfies the Leibnitz rule $D(f \bullet_A g) = (Df) \bullet_B g +_B f \bullet_B (Dg)$.

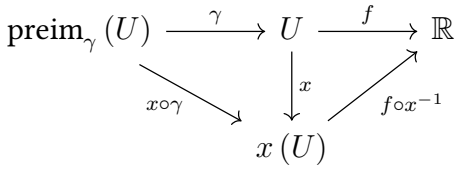


Figure 3.6: The composition of a curve $\gamma : \text{preim}_\gamma(U) \to U$ and a function $f : U \to \mathbb{R}$ yields $f \circ \gamma : \text{preim}_\gamma(U) \to \mathbb{R}$. The derivative of this function can be taken using standard calculus. In coordinates $x : U \to x(U)$, the map is $(f \circ x^{-1}) \circ (x \circ \gamma) : \text{preim}_\gamma(U) \to \mathbb{R}$ which is the same thing since x is a homeomorphism on $U \in \mathcal{O}$.

Important examples of derivations follow:

1. Let $A = \left(\mathcal{C}^\infty(M), +_{\mathcal{C}^\infty(M)}, \cdot_{\mathcal{C}^\infty(M)}, \bullet_{\mathcal{C}^\infty(M)}\right)$ and $B = (\mathbb{R}, +_\mathbb{R}, \cdot_\mathbb{R}, \bullet_\mathbb{R})$. Then any $X \in T_pM$ for which $X : \mathcal{C}^\infty(M) \to \mathbb{R}$ where $X\left(f \bullet_{\mathcal{C}^\infty(M)} g\right) = (Xf) \bullet_\mathbb{R} g +_\mathbb{R} f \bullet_\mathbb{R} (Xg)$.

2. Let $A = \text{End}(V)$ for some vector space V. Define the $\bullet \equiv [.,.]$ operator as $[.,.] : A \times A \tilde{\to} A$ where $[.,.] : (\phi, \psi) \to [\phi, \psi] := \phi \circ \psi - \psi \circ \phi$. The algebra $(\text{End}(V), +, \cdot, [.,.])$ is called a Lie algebra. This algebra satisfies the Jacobi identity, namely $[\phi, [\psi, \rho]] + [\psi, [\rho, \phi]] + [\rho, [\phi, \psi]] = 0$. With this, we can now define a derivation $D_H \equiv [H, .] : A \tilde{\to} A$ where H is some fixed element of A. Then $D_H([A, B]) = [D(A), B] + [A, D(B)]$.

The smooth map $f \in C^\infty(M)$ is such that $f : M \to \mathbb{R}$, and the chart $(U, x) \in \mathcal{A}$ is such that $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$. Then the map $(f \circ x^{-1}) : \mathbb{R}^{\dim M} \to \mathbb{R}$, and so the standard definitions of partial derivatives in calculus ∂_a along the a^{th} coordinates can be used on $(f \circ x^{-1})$, seen in Figure 3.6.

Definition 3.2.20. (Chart induced derivative) Let $f \in C^\infty(M)$ be such that $f : M \to \mathbb{R}$, and the chart $(U, x) \in \mathcal{A}$ be such that $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$. Then define the symbol $\left(\frac{\partial f}{\partial x^a}\right)_p := \partial_a (f \circ x^{-1})(x(p))$, where ∂_a is the partial derivative from ordinary calculus.

The symbol $\left(\frac{\partial}{\partial x^a}\right)_p$ is dangerously simple and one must be careful when using it by always returning to Definition 3.2.20. The symbol $\left(\frac{\partial}{\partial x^a}\right)_p$ operates on functions $f \in C^\infty(M)$, whereas the symbol ∂_a operates on function $f \circ x^{-1} \in C^\infty\left(\mathbb{R}^{\dim M}\right)$ from standard calculus. To be as clear as possible, the chart induced derivative is

$$\left(\frac{\partial f}{\partial x^a}\right)_p := \partial_a \left(f \circ x^{-1}\right)(x(p))$$

where corresponding colors correspond to correspond parts of the definition.

Remark. (Chart induced derivative) Note that in the simplest case where $M = \mathbb{R}^d$ and $x = \mathrm{id}_{\mathbb{R}^d}$ is the identity map on $\mathbb{R}^d$, then the chart induced derivative reduces to the standard derivative from calculus.

Similarly, and following Figure 3.6 closely, the components of the tangent vector $X_{\gamma,p} : \mathcal{C}^\infty(M) \tilde{\to} \mathbb{R}$ acting on $f : M \to \mathbb{R}$ in the chart $x : U \to x(U)$ can be found to be $X_{\gamma,p} f = (f \circ \gamma)'(0) = (f \circ x^{-1} \circ x \circ \gamma)'(0) = \partial_b (f \circ x^{-1})(x(p)) \cdot \left(x^b \circ \gamma\right)'(0)$, but $\partial_b (f \circ x^{-1})(x(p)) = \left(\frac{\partial f}{\partial x^b}\right)_p$. Thus $X_{\gamma,p}$ in the chart x along the curve γ is $X_{\gamma,p} = \left(x^b \circ \gamma\right)'(0) \cdot \left(\frac{\partial}{\partial x^b}\right)_p$.

This suggests the definition of basis vectors $e_a := \left(\frac{\partial}{\partial x^a}\right)_p$ as the tangent vectors from the chart induced curve $\gamma_{(a)}$ at the point $p \in M$, which is the a^{th} component of γ.

The following facts are stated, and their proofs can be found in Appendix A:

- The operators $+_{T_pM}$ and $\cdot_{T_pM}$ close in T_pM, and so T_pM with these operators is a vector space.
- $\dim T_pM = \dim M$.
- For $a = 1, 2, ..., \dim M$ we have that $\lambda^a \left(\frac{\partial}{\partial x^a}\right)_p = 0$ for all $\lambda^a \in \mathbb{R}$ and so the vectors $\left(\frac{\partial}{\partial x^a}\right)_p$ are linearly independent and span the space T_pM, thus they form a basis. This means that any $X \in T_pM$ can be written in coordinates as $X = X^a \left(\frac{\partial}{\partial x^a}\right)_p$, where a given coordinate system $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$ induces the canonical basis $\left(\frac{\partial}{\partial x^a}\right)_p$, and in this basis $\left(\frac{\partial}{\partial x^a}\right)_p$ the vector X has components $X^a \in \mathbb{R}$.

These results imply that $\left(\frac{\partial}{\partial x^1}\right)_p, \left(\frac{\partial}{\partial x^2}\right)_p, ..., \left(\frac{\partial}{\partial x^{\dim M}}\right)_p$ is a generating system for T_pM. For a vector $X \in T_pM$, if $X = X^a \left(\frac{\partial}{\partial x^a}\right)_p$, then the real numbers $X^1, X^2, ..., X^{\dim M}$ are called the components of the vector X with respect to the tangent space basis induced from the chart (U, x).

An especially important consequence for this dissertation is that if coordinates are transformed *nonlinearly* according to $y = y(x)$ (such as in deep neural networks), the basis vectors for a chart representation of $X \in T_pM$ still transform *linearly* according

to $\left(\frac{\partial y}{\partial x}\right)^{a.}_{.b}$. This allows for the generalization of the backpropagation algorithm to allow for backpropagating the coordinate representation of the metric tensor.

The tangent bundle is a bundle of all base point - tangent vector pairs of a manifold.

Definition 3.2.21. (Tangent bundle) The tangent bundle on M is defined as $TM := \cup_{p\in M}\{p\}\cup T_pM := \cup_{p\in M}\{(p,X)\,|p\in M, X\in T_pM\}$, where T_pM is the tangent space of M at the point $p\in M$. Additionally, define the bundle projection $\pi : TM\to M$ by $\pi : (p,X)\mapsto \pi\,(p,X) := p$ where p is the point for which $X\in T_pM$.

We can define another very similar and important type of bundle; namely the frame bundle. The frame bundle is the collection of all basis elements of the tangent space over the entire manifold. It will be shown that the backpropagation algorithm is actually just transforming the frame bundle with coordinate transformations.

Definition 3.2.22. (Frame bundle) The frame bundle on M is defined as $LM := \cup_{p\in M}\{p\}\cup L_pM := \cup_{p\in M}\{p\}\cup\{(e_1,...,e_d)\in T_pM|\,(e_1,...,e_d)$ is a basis of $T_pM\}$, where T_pM is the tangent space of M at the point $p\in M$.

Remark. (Uniqueness of frames) The projection map $\pi : X\mapsto \pi\,(X) := p$ where p is *the* point for which $X\in T_pM$ is unique because the frame bundle is a *disjoint* union $LM := \cup_{p\in M}L_pM$.

Remark. (Frame manifold bundle) Thus far (TM,π,M) is a set bundle, we want to make TM a smooth manifold. To do this, construct a smooth atlas on TM. The procedure for inducing an atlas on TM is outlined below.

- Let $\mathcal{A}_M$ be an atlas on M and take some $(U,x)\in\mathcal{A}_M$.
- Construct $(\text{preim}_\pi\,(U),\xi)$ where $\xi : \text{preim}_\pi\,(U)\to\xi\,(\text{preim}_\pi\,(U))\subseteq\mathbb{R}^{2\text{dim}M}$ defined by $\xi : X\mapsto\xi\,(X) := \left(X^1\,(\pi\,(X)),...,X^{\text{dim}M}\,(\pi\,(X)),X^1,...,X^{\text{dim}M}\right)$. This is an induced chart in $\mathcal{A}_{TM}$.
- Consider the basis of $T_{\pi(X)}M$, namely $\left(\frac{\partial}{\partial x^1}\right)_{\pi(X)},...,\left(\frac{\partial}{\partial x^{\text{dim}M}}\right)_{\pi(X)}$. Then for any $X\in T_{\pi(X)}M$ we have $X = X^a\left(\frac{\partial}{\partial x^a}\right)_{\pi(X)}$ where $X^a\in\mathbb{R}$.

3.2.4 Cotangent spaces

The tangent space T_pM is composed of elements $X\in T_pM$ such that $X : \mathcal{C}^\infty(M)\tilde{\to}\mathbb{R}$. The dual of the tangent space, the cotangent space, denoted $T^*_pM := (T_pM)^*$ has

elements $\omega_p \in T_p^*M$ such that $\omega_p : T_pM \to \mathbb{R}$. It will be shown that the differential of f at the point $p \in M$, namely $d_pf \in T_p^*M$ where $f \in C^\infty(M)$ such that $f : M \to \mathbb{R}$, is the gradient of f. These ideas will be made more precise and developed.

Definition 3.2.23. (Cotangent space) Let M be a smooth manifold. Then the cotangent space $T_p^*M := (T_pM)^*$ is defined to be the dual of the vector space T_pM.

The cotangent space is used to define the gradient of functions.

Definition 3.2.24. (Gradient) Let $f \in \mathcal{C}^\infty(M)$. Then for all $p \in M$ there exists a linear map $d_p : \mathcal{C}^\infty(M) \tilde{\to} T_p^*M$ where $f \mapsto d_pf$ is defined by its action on elements $X \in T_pM$ by $(d_pf)(X) := Xf$.

This linear operator is called the gradient operator at $p \in M$, where d_pf is the gradient of the function f at the point $p \in M$. Note that the gradient is a covector, not a vector.

Often one writes df instead of d_pf, and so it is implicitly understood that this is operating at a point $p \in M$.

Definition 3.2.25. (Chart-induced covector basis) Let $p \in M$ and $(U, x) \in \mathcal{A}$ be a chart with $p \in U$. Then the chart-induced covector basis for $x : U \to \mathbb{R}$ is $dx^a \in T_p^*M$ for all $a = 1, 2, ..., \dim M$. It follows that $dx^a\left(\frac{\partial}{\partial x^b}\right)_p := \left(\frac{\partial}{\partial x^b}\right)_p x^a := \partial_b\left(x^a \circ x^{-1}\right)(x(p)) = \partial_b\left(x \circ x^{-1}\right)^a(x(p)) = \delta^a_{\cdot b}$, where the first equality is from Definition 3.2.24, and the second is from Definition 3.2.20

This is why df in Definition 3.2.24 is called the gradient, because in the chart-induced basis the element acts on the element $X \in T_pM$ as $df(X) := Xf = X^a\left(\frac{\partial f}{\partial x^a}\right)_p$. Thus df is the coordinate free gradient of the function f.

Remark. (Chart-induced covector basis) The tangent space T_pM has an induced canonical basis in coordinates $x : U \to x(U)$ given by $\left\{\left(\frac{\partial}{\partial x^a}\right)\right\}_{a=1}^{\dim M}$. This induced basis for the tangent space T_pM then induces a canonical covector basis for the cotangent space T_p^*M, namely $\{dx^a\}_{a=1}^{\dim M}$.

Remark. (Matrix multiplication) A covector $\omega \in V^*$ is a map $\omega : V \tilde{\to} \mathbb{R}$, so in a vector basis e_b and covector basis ϵ^a one has $\omega(v) = \omega_a\epsilon^a\left(v^be_b\right) = \omega_av^b\epsilon^a(e_b) = \omega_av^b\delta^a_b = \omega_av^a = \hat{\omega} \cdot \hat{v}$, where $\hat{\omega}$ and $\hat{v}$ are lists of numbers and $\cdot$ is row-column matrix multiplication.

Remark. (Changing basis) Let $\tilde{e}_a = A^{a\cdot}_{\cdot b} e_b$ and inversely $e_a = B^{m\cdot}_{\cdot a} \tilde{e}_m$. Then $\omega_a := \omega(e_a) = \omega(B^{m\cdot}_{\cdot a} \tilde{e}_m) = B^{m\cdot}_{\cdot a} \omega(\tilde{e}_m) = B^{m\cdot}_{\cdot a} \tilde{\omega}_m$. Similarly, let $v^a := v(\epsilon^a) = \epsilon^a(v) = \epsilon^a\left(\tilde{v}^b \tilde{e}_b\right) = \epsilon^a\left(\tilde{v}^b A^{m\cdot}_{\cdot b} e_m\right) = \tilde{v}^b A^{m\cdot}_{\cdot b} \epsilon^a(e_m) = \tilde{v}^b A^{m\cdot}_{\cdot b} \delta^a_m = A^{a\cdot}_{\cdot b} \tilde{v}^b$.

We have that $X \in T_pM$ is a map where $X : C^\infty(M) \tilde{\to} \mathbb{R}$, and $\omega \in T^*_pM$ is a map $\omega : T_pM \tilde{\to} \mathbb{R}$. From the tangent and cotangent spaces, one can construct general (k, l) tensor spaces by simply taking the Cartesian product of the tangent and cotangent spaces with themselves k and l times, respectively.

3.2.5 Tensor Spaces

With the mathematical machinery developed so far, it is straight forward to construct general tensor spaces on topological manifolds.

Definition 3.2.26. (Tensor) Let V and V^* be vector and covector spaces. Define the (k, l) tensor as the multilinear map from the Cartesian product of k-cotangent and l-tangents spaces to $\mathbb{R}$, i.e. $T : (V^*)^{\times k} \times (V)^{\times l} \tilde{\to} \mathbb{R}$, where $(V^*)^{\times k} := V^* \times ... \times V^*$ k-times and $(V)^{\times l} := V \times ... \times V$ l-times.

The collection of all tensors then defines a tensor space.

Definition 3.2.27. (Tensor space) Define the tensor space to be $T^k_l V := (V)^{\times k} \times (V^*)^{\times l} := \{T | T \text{ is a } (k, l) \text{ tensor}\}$.

Note that the dual symbol $*$ has been shifted in the tensor in Definition 3.2.26 and the tensor space in Definition 3.2.27. This is because a (k, l) tensor is a multilinear map $T : (V^*)^{\times k} \times (V)^{\times l} \tilde{\to} \mathbb{R}$, and so $T \in (V)^{\times k} \times (V^*)^{\times l} := \{T | T \text{ is a } (k, l) \text{ tensor}\}$.

A tensor $\phi \in T^k_l V$ is a multilinear map $\phi : (V^*)^{\times k} \times (V)^{\times l} \tilde{\to} \mathbb{R}$. For example if $T^1_1 V \equiv V^* \times V$, then $\phi(\omega, v) = \phi\left(\omega_a \epsilon^a, v^b e_b\right) = \omega_a v^b \phi(\epsilon^a, e_b) =: \omega_a v^b \phi^{a\cdot}_{\cdot b}$ in the basis $e_b \in V$ and co-basis $\epsilon^a \in V^*$. From left to right, the first equality comes from expansion in a certain basis and co-basis, the second comes from multi-linearity, and the third is defining standard notation for elements of a tensor. The vector thus has a raised index, while the covector has a lowered index.

In general, tensor spaces are constructed completely independently of differentiable geometry. For differentiable geometry, we take our space $V = T_pM$. The tensor space becomes a vector space when equipped with pointwise addition and multiplication, i.e. $+_{T^k_l V} : (\phi, \psi) \to \left(\phi +_{T^k_l V} \psi\right)(t) := \phi(t) +_{\mathbb{R}} \psi(t)$ and $\cdot_{T^k_l V} : (\lambda, \phi) \to$

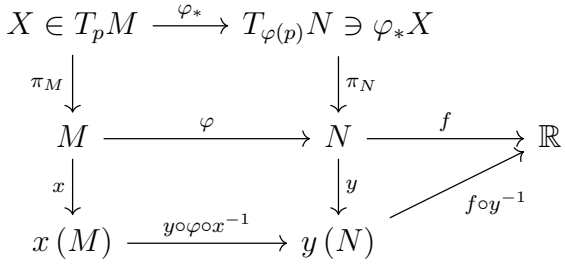


Figure 3.7: Visualization of the push-forward map. We have $X \in T_pM$ is such that $X : \mathcal{C}^\infty(M) \tilde{\to} \mathbb{R}$, and $\varphi_* : T_pM \tilde{\to} T_{\varphi(p)}N$ so $\varphi_* : X \tilde{\to} (\varphi_* X) \in T_{\varphi(p)}N$ is such that $(\varphi_* X) : \mathcal{C}^\infty(N) \tilde{\to} \mathbb{R}$ is defined by how it operates on the smooth function $f \in \mathcal{C}^\infty(N)$ as $(\varphi_* X)(f) := X(f \circ \varphi)$, where $f \circ \varphi : M \to \mathbb{R}$ so $(f \circ \varphi) \in \mathcal{C}^\infty(M)$, and hence X can act on it.

$\left(\lambda \cdot_{T_l^k V} \phi\right)(t) := \lambda \cdot_{\mathbb{R}} \phi(t)$, where $\phi, \psi \in T_l^k V$, $t \in (V^*)^{\times k} \times (V)^{\times l}$ and $\lambda \in \mathbb{R}$. Tensor spaces immediately apply to the work developed so far at a point $p \in M$ if we let $V = T_pM$ and $V^* = T_p^*M$.

3.2.6 Push-forward and Pull-back

This subsection briefly reviews the push-forward and pull-back operators across tangent spaces of topological manifolds. These are enormously important in the development of a generalized theory of neural networks as these are the main tools for transforming geometric tensors across different layers of the neural network. It will be shown that the neural network backpropagation algorithm is in fact a sequence of pull-back operators on the frame bundle of the tangent space.

First we define the push-forward operator, with the relations between functions and maps and spaces seen in Figure 3.7. The push-forward was originally developed to generalize the Jacobian matrix in a coordinate free way.

Definition 3.2.28. (Push-forward) Let $\varphi : M \to N$ be a smooth map between smooth manifolds. Then the push-forward φ_* of the map φ at $p \in M$ is the linear map $\varphi_* : T_pM \tilde{\to} T_{\varphi(p)}N$, where $\varphi_* : X \tilde{\to} \varphi_*(X) \equiv \varphi_* X$ is defined by how it acts on functions $f \in \mathcal{C}^\infty(N)$ as $(\varphi_* X)(f) := X(f \circ \varphi)$.

The push-forward φ_* is often called the derivative of f at $p \in M$. It is called the push-forward because it pushes forward vectors $X \in T_pM$ to vectors $\varphi_* X \in T_{\varphi(p)}N$. In coordinates and following Figure 3.7, we have

$$\varphi_* \left(\frac{\partial}{\partial x^a} \right)_p = \varphi_* \left(\left(x^{-1} \right)_* \left(\frac{\partial}{\partial x^a} \right)_{x(p)} \right) = \left(y^{-1} \right)_* \left(\left(y \circ \varphi \circ x^{-1} \right)_* \left(\frac{\partial}{\partial x^a} \right)_{x(p)} \right) =$$
$$\left(y^{-1} \right)_* \left(\frac{\partial}{\partial x^a} \left(y \circ \varphi \circ x^{-1} \right)^b \left(\frac{\partial}{\partial y^b} \right)_{y \circ \varphi(p)} \right) = \frac{\partial \left(y \circ \varphi \circ x^{-1} \right)^b}{\partial x^a} \left(\frac{\partial}{\partial y^b} \right)_{\varphi(p)} \tag{3.1}$$

Thus it is shown that the push-forward φ_* is in fact the generalized Jacobian on local regions of manifolds.

Remark. (Change of Coordinates) A special case for the push-forward is when the map $\varphi : M \to M$ is the identity over intersecting chart regions. In this case, the change of coordinates of Equation 3.1 reduces to the following:

$$\left(\frac{\partial}{\partial x^a} \right)_p = \frac{\partial \left(y \circ x^{-1} \right)^b}{\partial x^a} \left(\frac{\partial}{\partial y^b} \right)_p \tag{3.2}$$

For example, with $x, y : \mathbb{R}^2 \to \mathbb{R}^2$, where x is Cartesian coordinates and y is polar coordinates, then $\left(y \circ x^{-1} \right)(p, q) = \left(\sqrt{p^2 + q^2}, \tan 2 \left(q/p \right) \right)$. In this way we are only looking at the coordinate representation of the manifold, as opposed to the manifold itself.

We have that tangent vectors along curves $\gamma : \mathbb{R} \to M$ are pushed forward to tangent vectors along curves $\varphi \circ \gamma : \mathbb{R} \to N$, namely $\varphi_* \left(X_{\gamma,p} \right) = X_{\varphi \circ \gamma, \varphi(p)}$.

Similarly, we define the pull-back operator.

Definition 3.2.29. (Pull-back) Let $\varphi : M \to N$ be a smooth map between smooth manifolds. Then the pull-back φ^* of the map φ at $\varphi(p) \in N$ is the linear map $\varphi^* : T^*_{\varphi(p)} N \tilde{\to} T^*_p M$, where $\varphi^* : \omega \to \varphi^* (\omega)$ is defined by how it acts on vectors $X \in T_p M$ as $\left(\varphi^* \omega \right)(X) := \omega \left(\varphi_* X \right)$.

Remark. (Pull-back) We have that the pull-back φ^* is a map that takes $\omega \in T^*_{\varphi(p)} N \to \varphi^* \omega \in T^*_p M$. The element $\omega \in T^*_{\varphi(p)} N$ is an operator $\omega : T_{\varphi(p)} N \to \mathbb{R}$ by definition of the cotangent space, where $\varphi_* X \in T_{\varphi(p)} N$. Thus the *only* linear way to combine the pieces of information φ^*, ω and X is by letting ω act on $\varphi_* X$, namely $\left(\varphi^* \omega \right)(X) := \omega \left(\varphi_* X \right)$. Similarly, $\varphi^* \omega \in T^*_p M$ is an operator $\varphi^* \omega : T_p M \to \mathbb{R}$

The pull-back operator is extremely important for Chapter 4 as both the generalization of the error backpropagation algorithm as well as the closed form solution of the metric tensor require pulling-back the coordinate representation of tensor fields.

For the mapping $\varphi : M \to N$ there exist several special cases depending on the dimensions of M and N. Loosely speaking, if $\dim N > \dim M$ then the map is called an immersion, and if $\dim M > \dim N$ then the map is called a submersion.

Definition 3.2.30. (Constant rank) Let $\varphi : M \to N$ be a smooth map between smooth manifolds. The rank of φ is defined to be the rank of the linear pushfoward map between tangent spaces, $\varphi_* : T_pM \to T_{\varphi(p)}N$, which is the generalized Jacobian matrix of partial derivatives. If the $\text{rank}\,(\varphi) = k \ \forall p \in M$ is constant, then φ is said to have a constant rank.

With the above definition of constant rank, submersions and immersions are defined:

Definition 3.2.31. (Submersion and immersion) Let $\varphi : M \to N$ be a smooth map between manifolds with constant rank.

- If φ is surjective (or equivalently, $\text{rank}\,(\varphi) = \dim\,(N)$), then it is a submersion.
- If φ is injective (or equivalently, $\text{rank}\,(\varphi) = \dim\,(M)$), then it is an immersion.

An embedding is a special case of an immersion. An embedding is an immersion with the additional requirement that the mapping preserves topological structure.

Definition 3.2.32. (Embedding) The smooth map $\varphi : M \to N$ is an embedding if the following conditions hold:

1. φ is an immersion.
2. $\varphi\,(M) \cong_{\text{top}} N$

Remark. (Embedding) Note that $\left(\varphi\,(M) \cong_{\text{top}} N\right) \Rightarrow \left(\varphi\,(M) \cong_{\text{diff}} N\right)$. This follows from coordinate charts in the atlas are homeomorphisms and the spaces $x\,(\varphi\,(M)) \subseteq \mathbb{R}^{\dim\varphi(M)}$ and $x\,(N) \subseteq \mathbb{R}^{\dim N}$ are equipped with the (induced) standard topology.

A famous result of this is the Whitney Embedding Theorem.

Theorem 3.2.2. (Whitney) Any smooth manifold M can be $(i.)$ embedded in $\mathbb{R}^{2\dim M}$, and $(ii.)$ immersed in $\mathbb{R}^{2\dim M-1}$.

3.2.7 Tensor Fields and Modules

In Section 3.2.5 the concept of a tensor space was introduced, as a multilinear generalization to vector and co-vector spaces. Here we introduce the concept of tensor fields, which will be smooth sections on smooth manifolds, again as a multilinear generalization to vector and co-vector fields. This is a very important construction as both the frame bundle as well as the metric bundle are both tensor fields.

Definition 3.2.33. (Vector field) Let M be a smooth manifold and TM its tangent bundle (TM, π, M) with $\pi : TM \to M$ smooth. A vector field is a smooth section of TM, i.e. $\sigma : M \to TM$ is smooth and $\pi \circ \sigma = \mathrm{id}_M$.

Call the set of all vector fields $\Gamma(TM) := \{\sigma : M \to TM | \pi \circ \sigma = \mathrm{id}_M\}$. Equip $\Gamma(TM)$ with two operations $+_{\Gamma(TM)}$ and $\cdot_{\Gamma(TM)}$ where $\forall p \in M$, $\sigma, \tau \in \Gamma(TM)$ and $f \in \mathcal{C}^\infty(M)$, are such that:

- $+_{\Gamma(TM)} : \Gamma(TM) \times \Gamma(TM) \to \Gamma(TM)$ defined by $\left(\sigma +_{\Gamma(TM)} \tau\right)(p) := \sigma(p) +_{T_pM} \tau(p)$
- $\cdot_{\Gamma(TM)} : \mathcal{C}^\infty(M) \times \Gamma(TM) \to \Gamma(TM)$ defined by $\left(f \cdot_{\Gamma(TM)} \sigma\right)(p) := f(p) \cdot_{T_pM} \sigma(p)$ where $\cdot_{T_pM}$ is scalar multiplication, *not* pointwise ring multiplication.

Recall that $\left(\mathcal{C}^\infty(M), +_{\mathcal{C}^\infty(M)}, \bullet_{\mathcal{C}^\infty(M)}\right)$ is a ring with $\bullet_{\mathcal{C}^\infty(M)} : \mathcal{C}^\infty(M) \times \mathcal{C}^\infty(M) \to \mathcal{C}^\infty(M)$ defined pointwise, whereas $\left(\mathcal{C}^\infty(M), +_{\mathcal{C}^\infty(M)}, \cdot_{\mathcal{C}^\infty(M)}\right)$ is a $\mathbb{R}$-vector space because $\cdot_{\mathcal{C}^\infty(M)} : \mathbb{R} \times \mathcal{C}^\infty(M) \to \mathcal{C}^\infty(M)$ is defined as scalar multiplication. See Appendix B for further clarification. Very succinctly a ring is a field without the $\cdot$ axioms. This is because even though an element $r \in R$ of the ring may not be the zero element, i.e. $r \neq 0_R$, there does not necessarily exist an element r^{-1} such that $r \bullet_R r^{-1} = r^{-1} \bullet r = 1_R$. Take for example the non-zero element $(2, 0, -1/3) \in \mathbb{R}^3$, then with pointwise multiplication the multiplicitive inverse would be $(1/2, \mathrm{NaN}, -3)$ which is *not* an element of the ring $R = \mathbb{R}^3$. This generalizes in a straightforward way to functions.

A vector space V over a field F is an abelian group V with vector addition, with stronger structure imposed such that the group is an F-field. If the field F has the weaker structure of a ring, then it is no longer a vector space but instead a module. See Appendix B for further details.

An important result is that unlike a vector space, a module generally does not have a basis. If the ring is a division ring, then it does have a basis. Also, every field is a division ring so every vector space has a basis. Two important examples are when $M = \mathbb{R}^2$ and when $M = S^2$. When $M = \mathbb{R}^2$, coincidentally $\exists e_1, e_2 \in \Gamma(TM)$ such that $\forall v \in \Gamma(TM)\, \exists! v^1, v^2 \in \mathcal{C}^\infty(M) : v = v^a e_a$. However for $M = S^2$, technically there does not exist an everywhere non-zero everywhere smooth vector field, i.e. "cannot comb a sphere".

3.2.8 Lie Groups

Lie Groups are important because in many settings they provide the information for transforming tensors fields. In the formulation of neural networks provided by this dissertation, these simple group operations are used for transforming the frame bundle (i.e. backpropagation) as well as the metric tensor. The minimal type of structure that this dissertation needs that includes an operator is a group.

Definition 3.2.34. (Group) A group is a tuple $(G, \bullet)$ where G is a set and the map $\bullet : G \times G \to G$ is such that $\bullet : (a, b) \mapsto \bullet(a, b) \equiv a \bullet b$ satisfies the following:

1. Closure, for all $a, b \in G$ we have $a \bullet b \in G$.
2. Associativity, for all $a, b, c \in G$, we have $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
3. Identity, $\exists 1_G \in G$ such that for all $a \in G$ we have $1_G \bullet a = a \bullet 1_G = a$.
4. Inverse element, for all $a \in G \exists a^{-1}$ such that $a \bullet a^{-1} = a^{-1} \bullet a = 1_G$.
5. Commutativity (for Abelian groups), for all $a, b \in G$ we have $a \bullet b = b \bullet a$

Roughly, a group with a set G that has the additional structure that G is a topological manifold is called a Lie Group.

Definition 3.2.35. (Lie Group) A Lie group is a tuple $(G, \bullet)$ such that

1. It is a group with group operation $\bullet$
2. G is a smooth topological manifold and the maps μ and ι are smooth
   (a) $\mu : G \times G \to G$ where $\mu(g_1, g_2) := g_1 \bullet g_2$
   (b) $\iota : G \to G$ where $\iota(g) := g^{-1}$.

Remark. (Lie Group) The maps $\mu : G \times G \to G$ and $\iota : G \to G$ are smooth because, G is a smooth topological manifold, thus $G \times G$ is a smooth topological manifold that inherits a smooth atlas from G, and thus it makes sense to say these maps $\mu : G \times G \to G$ and $\iota : G \to G$ are smooth.

By far the most important example of a Lie group in this thesis is the general linear group.

Definition 3.2.36. (General linear group) Define the topological manifold of dimension $n \times n$ over the reals $\mathbb{R}$ as $G = GL(n, \mathbb{R}) := \{\varphi : \mathbb{R}^n \tilde{\to} \mathbb{R}^n | \det \varphi \neq 0\}$, together with group operation $\bullet \equiv \circ$ composition of matricies / matrix multiplication. Then the tuple $(GL(n, \mathbb{R}), \circ)$ is called the general linear group.

Other important Lie groups in physics are derived from the special orthogonal group. These will be defined, after defining the pseudo-inner product.

Definition 3.2.37. (Pseudo-inner product) Let V be a d-dimensional $\mathbb{R}$-vector space, equipped with a pseudo-inner product $(\cdot, \cdot) : V \times V \tilde{\to} \mathbb{R}$ such that it is (i.) bilinear (ii.) symmetric $(u, v) = (v, u)$ and (iii.) non-degenerate $\forall w \in V : (v, w) = 0 \Rightarrow v = 0$. A stronger requirement than (iii.) is (iii.') positive-definiteness $\forall v \in V : (v, v) \geq 0$ and $(v, v) = 0 \iff v = 0_V$.

With this, define the following structure group, where the signature (p, q) is the number of positive and negative eigenvalues of the metric tensor:

$$O(p, q) := \{\varphi : V \tilde{\to} V | (\varphi(v), \varphi(w)) = (v, w) \, \forall v, w \in V\} \subseteq GL(p + q, \mathbb{R}) \quad (3.3)$$

When we restrict to sets with group element determinants equal to 1 we have the special orthogonal $SO(p, q)$ groups.

Definition 3.2.38. (Special orthogonal group) Define the special orthogonal group to be the set $SO(p, q) := \{\varphi : V \tilde{\to} V | (\varphi(v), \varphi(w)) = (v, w) \, \forall v, w \in V \text{ and } \det \varphi = 1\}$, and note that $SO(p, q) \subseteq GL(p + q, \mathbb{R})$.

Important Lie groups in physics are the Lorentz group $SO(1, 3)$ in General Relativity, $SO(2)$ the electroweak interactions and $SO(3)$ in quantum chromodynamics; both are Yang-Mills theories (non-abelian gauge theories). This thesis deals with the general linear group $GL(\dim M, \mathbb{R})$, or some subset of, induced by the coordinate transformations allowed by the neural network structure.

3.2.9 Principal and Associated Bundles

Very roughly speaking, a principal fibre bundle is a bundle whose fibre is a Lie group. Principal fibre bundles are so immensely important because they allow us to understand any fibre bundle F on which G acts; they are called associated bundles (associated to the principal bundle).

Once expressed in terms of principal fibre bundles and associated fibre bundles, we will see that the process of labeling a basis $\{e_1, ..., e_{\text{dim}M}\}$ of T_pM by lower indicies, and the components of a $X \in T_pM$ by upper indicies, and having the corresponding transformation behavior $\tilde{e}_a = A^{m.}_{.a} e_m$ and $\tilde{X}^a = (A^{-1})^{a.}_{.m} X^m$, will be understood as a right action of $GL\left(\text{dim}M, \mathbb{R}\right)$ on the basis and a left action of $GL\left(\text{dim}M, \mathbb{R}\right)$ on the components.

3.2.9.1 Principal Bundles

Principal bundles are acted on (transformed) by groups. First we define what a group action on a bundle is.

Definition 3.2.39. (Lie Group left action) Let $(G, \bullet)$ be a Lie Group and M a smooth manifold and the smooth map $\triangleright$: $G \times M \to M$ satisfies

1. $e \triangleright p = p \; \forall p \in M$ and $e \in G$ is the group identity element.
2. $h_2 \triangleright (h_1 \triangleright p) = (h_2 \bullet h_1) \triangleright p \; \forall p \in M$ and $\forall h_1, h_2 \in G$.

Any such smooth map $\triangleright$ is called a left G-action on M.

Similarly defined is the right G-action. Note that the triangle, and the arguments of the triangle, have been reversed when compared to the Lie Group left action.

Definition 3.2.40. (Lie Group right action) Let $(G, \bullet)$ be a Lie Group and M a smooth manifold and the smooth map $\triangleleft$: $M \times G \to M$ satisfies

1. $p \triangleleft e = p \; \forall p \in M$ and $e \in G$ is the group identity element.
2. $(p \triangleleft h_1) \triangleleft h_2 = p \triangleleft (h_1 \bullet h_2) \; \forall p \in M$ and $\forall h_1, h_2 \in G$.

Any such smooth map $\triangleleft$ is called a right G-action on M.

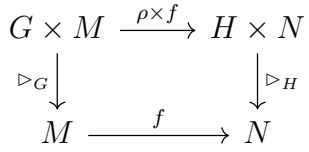


Figure 3.8: The map $f : M \to N$ is called ρ-equivariant if this diagram commutes.

In principal the left and right group actions do not have to be related. An extremely useful case however is when the right action is defined in terms of the left.

Remark. (Lie Group right action) Let $\triangleright$: $G \times M \to M$ be a left G-action, then define the right G-action $\triangleleft$: $M \times G \to M$ in terms of the left as $p \triangleleft h := h^{-1} \triangleright p$ (this is simple to prove that this is a right G-action by showing it satisfies the two requirements and is smooth). The right G-action can be thought of as transforming the basis while the left G-action can be thought of as transforming the tensor components. A vector X has components X^a in the basis e_a. When transforming the basis e_a to $h^{c.}_{.a} e_c$, one must also transform the components X^a to $X^b (h^{-1})^{a.}_{.b}$. Altogether, this ensures that the vector does not depend on the choice of basis: $X = X^a e_a = X^b (h^{-1})^{a.}_{.b} h^{c.}_{.a} e_c = X^b \delta^{c.}_{.b} e_c = X^b e_b = X$.

The following definition, with the corresponding Figure 3.8, gives the condition when two manifolds, and the Lie groups that operate on them, are structurally equivalent, i.e. defines the structure preserving maps.

Definition 3.2.41. (ρ-equivariant) Let $(G, \bullet_G)$ and $(H, \bullet_H)$ be two Lie groups, $\rho : G \to H$ be a Lie group homomorphism, M and N be two smooth manifolds with their respective two left actions $\triangleright_G$: $G \times M \to M$ and $\triangleright_H$: $H \times N \to N$ and $f : M \to N$ be a smooth map. Then f is called ρ-equivariant if $f(g \triangleright_G m) = \rho(g) \triangleright_H f(m)$, where $g \in G$, $m \in M$, $\rho(g) \in H$ and $f(m) \in N$

Several definitions are extremely important when considering Lie groups and how they act on manifolds.

Definition 3.2.42. (Lie Group properties) Let $\triangleleft$: $M \times G \to M$ be a right action.

1. $\forall p \in M$, define its orbit under the action as $\mathcal{O}_p := \{q \in M | \exists h \in G : p \triangleleft h = q\}$.
2. p and q in M are equivalent if they lie on the same orbit, i.e. $p \sim_G q :\iff \exists h \in G : q = p \triangleleft h$.

3. This is an equivalence relation and can be used to construct the quotient set $M/G := M/\sim_G$.

4. $\forall p \in M$ the stabilizer $S_p := \{h \in G | p \lhd h = p\} \subseteq G$.
   (The action $\lhd$ is free if $\forall p \in M$ the stabilizer $S_p = \{e\}$ where $e \in G$ is the identity element.)

Remark. (Lie Group properties) It is helpful to think of the orbit $\mathcal{O}_p$ is the set of coordinate representations of the point $p \in M$ possible by the group G. In physics we are not interested in just the manifold M, we are interested in the manifold M modulo basis representations from group actions induced by coordinates (from the simple fact that the laws of physics should be independent of the coordinate representation). More relevant to this thesis in machine learning, the object "tree" should be independent of coordinate representation.

Definition 3.2.43. (Principal fibre bundles) A bundle (E, π, M) is called a principal G-bundle if

- E is a right G-space (i.e. E is equipped with a right G-action $\lhd$).
- The right action $\lhd$ is free.
- $E \xrightarrow{\pi} M \cong_{\text{bundle}} E \xrightarrow{\rho} E/G$ such that $\rho : E \to E/G$ defined by $\epsilon \mapsto \rho(\epsilon) := [\epsilon]$

  Recall Definition 3.2.14 and Figure 3.5.

Remark. (Bundle isomorphism) Two bundles are said to be isormorphic as bundles if: $E \xrightarrow{\pi} M \cong_{\text{bundle}} E' \xrightarrow{\pi'} M' :\iff \exists$ diffeomorphisms u, f such that $\pi' \circ u = f \circ \pi$.

Remark. (Free action orbit isomorphic to group) Since $\lhd$ is free $\Rightarrow \rho^{-1}([\epsilon]) \cong G$, i.e. if the action is free, then the orbit is isomorphic to the group.

At this point we can use these tools to establish an extremely important example for this dissertation.

- Recall the frame bundle from Definition 3.2.22 is defined as $LM := \cup_{p\in M} L_pM := \cup_{p\in M}\{(e_1, ..., e_d) \in T_pM | (e_1, ..., e_d) \text{ is a basis of } T_pM\}$, where T_pM is the tangent space of M at the point $p \in M$.

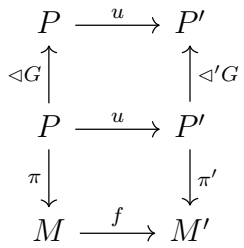


Figure 3.9: Given bundles $P \xrightarrow{\pi} M$ and $P' \xrightarrow{\pi'} M'$, the maps $u : P \to P'$ and $f : M \to M'$ are called principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' g$.

- Equip LM with a smooth atlas $\mathcal{A}_{LM}$ inherited from the atlas $\mathcal{A}_M$ of M. As in Definition 3.2.22, we construct $(\text{preim}_\pi(U), \xi)$ where $\xi : \text{preim}_\pi(U) \to \xi(\text{preim}_\pi(U)) \subseteq \mathbb{R}^{2\text{dim}M}$ defined by $\xi : X \mapsto \xi(X)$ such that $\xi(X) := \left(X^1(\pi(X)), ..., X^{\text{dim}M}(\pi(X)), X^1, ..., X^{\text{dim}M}\right)$.
- Establish a right-$GL(\text{dim}M, \mathbb{R})$ action $\lhd$ on LM. Define $(e_1, ..., e_{\text{dim}M}) \lhd h := (h^{m\cdot}_{\cdot 1} e_m, h^{m\cdot}_{\cdot 2} e_m, ..., h^{m\cdot}_{\cdot \text{dim}M} e_m)$, which is just the ordinary matrix multiplication rule, and $GL(\text{dim}M, \mathbb{R}) = \{h^{m\cdot}_{\cdot n} \in \mathbb{R} | m, n = 1, ..., \text{dim}M; \det h \neq 0\}$. Note that $h \in GL(\text{dim}M, \mathbb{R})$ only keeps $(e_1, ..., e_{\text{dim}M})$ invariant if $h^{a\cdot}_{\cdot b} = \delta^{a\cdot}_{\cdot b}$, so it is a *free* action.
- The bundle $LM \xrightarrow{\pi} M$, with LM having the right action $\lhd$ defined above, is a $GL(\text{dim}M, \mathbb{R})$-principal bundle, because bundle isomorphisms exist such that $\left(LM \xrightarrow{\pi} M\right) \cong_{\text{bundle}} \left(LM \xrightarrow{\pi'} LM/GL(\text{dim}M, \mathbb{R})\right)$.

Definition 3.2.44. (Principal bundle map) Given bundles $P \xrightarrow{\pi} M$ and $P' \xrightarrow{\pi'} M'$, the maps $u : P \to P'$ and $f : M \to M'$ are called principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' g$, i.e. if Figure 3.9 commutes.

A generalization of the principal bundle map is when the Lie group G is allowed to change via Lie group homomorphism $\rho : G \to G'$ such that $\rho(g_1 \bullet_G g_2) = \rho(g_1) \bullet_{G'} \rho(g_2)$.

Definition 3.2.45. (Generalized principal bundle map) Given bundles $P \xrightarrow{\pi} M$, $P' \xrightarrow{\pi'} M'$ and Lie group homomorphism $\rho : G \to G'$ such that $\rho(g_1 \bullet_G g_2) = \rho(g_1) \bullet_{G'} \rho(g_2)$, the maps $u : P \to P'$ and $f : M \to M'$ are called principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' \rho(g)$, i.e. if Figure 3.10 commutes.

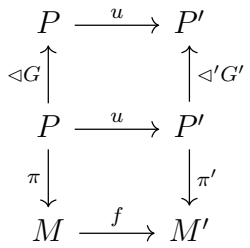


Figure 3.10: Given bundles $P \xrightarrow{\pi} M$, $P' \xrightarrow{\pi'} M'$ and Lie group homomorphism $\rho : G \to G'$ such that $\rho(g_1 \bullet_G g_2) = \rho(g_1) \bullet_{G'} \rho(g_2)$, the maps $u : P \to P'$ and $f : M \to M'$ are called generalized principal bundle maps if $f \circ \pi = \pi' \circ u$ and $u(p \lhd g) = u(p) \lhd' \rho(g)$.

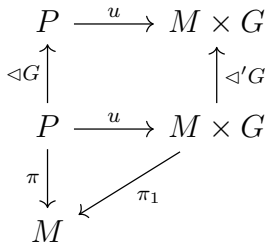


Figure 3.11: A principal G-bundle $P \xleftarrow{\lhd G} P \xrightarrow{\pi} M$ is called trivial if there exists a principal bundle map u to $M \times G \xleftarrow{\lhd' G} M \times G \xrightarrow{\pi_1} M$ given by $(x, g) \lhd' g' := (x, g \bullet g')$ and $\pi_1 : M \times G \to M$ such that $\pi_1(x, g) := x$.

An important case of a principal bundle map is if the map $u : P \to M \times G$ is trivial.

Definition 3.2.46. (Trivial principal bundle map) A principal G-bundle $P \xleftarrow{\lhd G} P \xrightarrow{\pi} M$ is called trivial if there exists a principal bundle map u to $M \times G \xleftarrow{\lhd' G} M \times G \xrightarrow{\pi_1} M$ given by $(x, g) \lhd' g' := (x, g \bullet g')$ and $\pi_1 : M \times G \to M$ such that $\pi_1(x, g) := x$, as seen in Figure 3.11.

The notions and relations between principal fibre bundles and trivial principal bundles can be seen in Figure 3.12.

3.2.9.2 Associated Bundles

The principal bundle provides the Lie group fibres which act on the associated bundle.

Definition 3.2.47. (Associated fibre bundle) Given a G principal bundle and a smooth manifold F on which exists a left G-action $\triangleright$: $G \times F \to F$, the associated fibre bundle (P_F, π_F, M) is defined as follows:

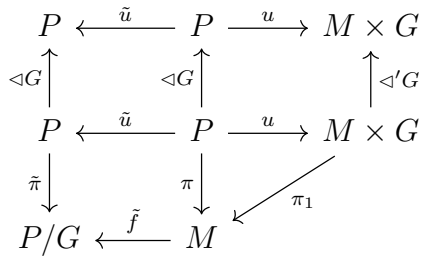


Figure 3.12: The relations between principal fibre bundles (left side) and trivial fibre bundles (right side) can be seen.

- let $\sim_G$ be the relation on $P \times F$ defined as follows:
  $(p, f) \sim_G (p', f') :\Longleftrightarrow \exists h \in G : p' = p \triangleleft h$ and $f' = h^{-1} \triangleright f$. Then $\sim_G$ is an equivalence relation, and thus $P_F := (P \times F) \,/\, \sim_G$ is a quotient space. In other words, the elements of P_F are the equivalence classes $[(p, f)]$, where $p \in P$ and $f \in F$.

- Define $\pi_F : P_F \to M$ by $\pi_F ([(p, f)]) := \pi (p)$. Note that this is well defined since $\pi_F ([(p \triangleleft g, g^{-1} \triangleright f)]) = \pi (p \triangleleft g) = \pi (p) = \pi_F ([(p, f)])$ and $\pi_F : P_F \to M$ is a fibre bundle with typical fibre F.

An extremely important example is given below. This is the general case in which both error backpropagation as well as metric tensor backpropagation are specific cases.

- Let the principal bundle be the frame bundle, defined in the previous section as $P = LM := \bigcup_{p \in M} \{(e_1, ..., e_d) \in T_pM | \, (e_1, ..., e_d) \text{ is a basis of } T_pM\}$ with right-$G = GL(\mathrm{dim}M, \mathbb{R}) = \{h^{m.}_{.n} \in \mathbb{R} | m, n = 1, ..., \mathrm{dim}M; \det h \neq 0\}$ action $\triangleleft$ on LM defined as $(e_1, ..., e_{\mathrm{dim}M}) \triangleleft h := (h^{m.}_{.1} e_m, h^{m.}_{.2} e_m, ..., h^{m.}_{.\mathrm{dim}M} e_m)$, which is just the ordinary matrix multiplication rule.

- Define the fibres of the associated fibre bundle as $F = \left(\mathbb{R}^d\right)^{\times p} \times \left(\mathbb{R}^{d*}\right)^{\times q}$ with left G-action $\triangleright$: $GL(\mathrm{dim}M, \mathbb{R}) \times F \to F$ by
  $(h \triangleright f)^{i_1 ... i_p}{}_{j_1 ... j_q} := f^{\tilde{i_1} ... \tilde{i_p}}{}_{\tilde{j_1} ... \tilde{j_q}} h^{i_1}{}_{\tilde{i_1}} .... h^{i_p}{}_{\tilde{i_p}} (h^{-1})^{\tilde{j_1}}{}_{j_1} .... (h^{-1})^{\tilde{j_q}}{}_{j_q}$.

- Thus $LM_F \xrightarrow{\pi_F} M$ is isomorphic as a bundle to $T^p_q M \xrightarrow{\pi} M$, i.e. the (p, q)-tensor bundle.

Similarly, one can define the (p, q)-tensor ω-density bundle as $(g \triangleright f)^{i_1 ... i_p}{}_{j_1 ... j_q} := \det (g^{-1})^{\omega} f^{\tilde{i_1} ... \tilde{i_p}}{}_{\tilde{j_1} ... \tilde{j_q}} g^{i_1}{}_{\tilde{i_1}} .... g^{i_p}{}_{\tilde{i_p}} (g^{-1})^{\tilde{j_1}}{}_{j_1} .... (g^{-1})^{\tilde{j_q}}{}_{j_q}$. This is necessary for defining

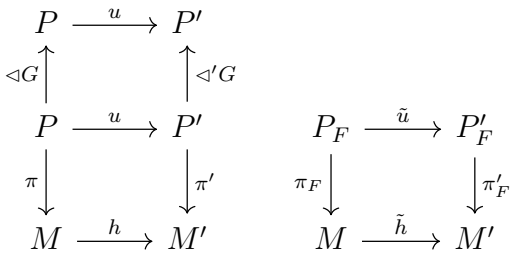


Figure 3.13: The principal bundle isomorphism (left side) is such that $\pi' \circ u = h \circ \pi$ and $u(p \lhd g) = u(p) \lhd' g$, while the induced associated bundle (right side) is such that $\tilde{u}([p, f]) := [u(p), f]$ and $\tilde{h}(m) := h(m)$ for all $p \in P, f \in F$ and $m \in M$.

integration on manifolds and thus formulating physics in terms of minimizing an action in the sense of variational calculus; such as the classical Einstein-Hilbert action [2].

As we will see in the next chapter of this dissertation, error backpropagation takes place when $(p, q) = (0, 1)$, while metric tensor backpropagation takes place when $(p, q) = (0, 2)$. In general this is defined for an arbitrary (p, q) tensor field.

The structure preserving maps between associated fibre bundles are associated bundle maps.

Definition 3.2.48. (Associated bundle maps) Associated bundle maps $\tilde{u} : P_F \to P'_F$ and $\tilde{h} : M \to M'$ between two associated bundles $P_F \xrightarrow{\pi_F} M$ and $P'_F \xrightarrow{\pi'_F} M'$ (where $P_F := P \times F/\sim_G$ and $P'_F := P' \times F/\sim_G$) is a bundle map which can be constructed from a principal bundle map $u : P \to P'$ and $h : M \to M'$ between the underlying principal bundles as $\tilde{u}([p, f]) := [u(p), f]$ and $\tilde{h}(m) := h(m)$ for all $p \in P, f \in F$ and $m \in M$, as seen in Figure 3.13.

Recall a principal G-bundle is called trivial if there exists a principal bundle map $u : P \to P$ and $h : M \to P/G$ between principal bundles $P \xleftarrow{\lhd G} P \xrightarrow{\pi} M$ and $P \xleftarrow{\lhd G} P \xrightarrow{\tilde{\pi}} P/G$, as seen in Figure 3.11. Similarly, define the trivial associated bundle.

Definition 3.2.49. (Trivial associated bundle) An associated bundle is called trivial if the underlying principal bundle is trivial.

3.3 Riemannian Manifolds

This section will review the most important aspects of Riemannian manifolds [78–80] needed in order to understand the geometric formulation of neural networks. This will be straightforward because of the developed definitions of principal and associated fibre bundles.

As mentioned in the introduction with the example from computer vision, a dog is invariant to how the image of the dog is taken. For example if the camera has channels RGB, has a fish eye lens or is taken upside down, the dog is invariant to the coordinate system used to represent the dog. After encoding the object, the coordinate representation of the object contains the information regarding the degrees of freedom available to the object. From this perspective, the minimal mathematical structure required to understand data is on the level of a topological manifold, since the weaker structure of a topological space does not have the notion of coordinates.

Additionally on an intuitive level humans understand that different breeds of dogs are similar to each other, while different types of cars are similar to each other, but dogs and cars dissimilar to each other. This implies that there exists a notion of distance on the manifold of allowable objects. Neural networks learn a representation in which these intuitive notions of distances are preserved, likely due to the fact that feature extraction is largely similar to data compression, and independent compression schemes for dogs and cars are needed to well compress these objects.

The minimum mathematical structure needed to give the notion of shape to a topological manifold is a Riemannian manifold. A Riemannian manifold endows a topological manifold with a smoothly varying inner product on the tangent space of the topological manifold. This allows for the notions of distance and angle to be defined on manifolds, thus giving the manifolds shape.

Roughly speaking, the metric on a manifold is a real valued map $g_p : T_pM \times T_pM \to \mathbb{R}$ where at each point $p \in M$ the map g_p takes elements of the fibre of the associated bundle $T_pM \times T_pM$ to $\mathbb{R}$ by $p \mapsto g_p(X, Y)$ for all differentiable vector fields $X, Y \in T_pM$ on M and $g_p(\cdot, \cdot)$ satisfies the axioms of an inner product. By this construction $g_p \in T_p^*M \times T_p^*M$ is a $(0, 2)$-tensor field/section on M.

Definition 3.3.1. (Inner product) An inner product space is a vector space V over the field K with an inner product $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ that satisfies the following three axioms, for all $X, Y, Z \in V$ and $\lambda \in K$.

- Conjugate symmetry: $\langle X, Y\rangle = \overline{\langle Y, X\rangle}$
- Linearity: $\langle X, \lambda Y\rangle = \lambda \langle X, Y\rangle$ and $\langle X, Y + Z\rangle = \langle X, Y\rangle + \langle X, Z\rangle$
- Positive-definiteness: $\langle X, X\rangle \geq 0$ and $\langle X, X\rangle = 0 \iff X = 0$

Remark. (Inner product) Several remarks are made on the definition of the inner product.

- For Riemannian manifolds, the vector space is the tangent space at each point $p \in M$, i.e. $V = T_pM$.
- For Riemannian manifolds, the field is the reals, i.e. $K = \mathbb{R}$, and so conjugate symmetry reduces to symmetry $\langle X, Y\rangle = \langle Y, X\rangle$.
- For Riemannian manifolds, often the notation $g(\cdot, \cdot) \equiv \langle \cdot, \cdot\rangle : T_pM \times T_pM \to \mathbb{R}$ is used.
- Often in mathematics linearity is assumed in the first argument. Here we assume linearity in both arguments, i.e. bilinearity.
- Riemannian manifolds have positive-definite inner products, as opposed to the weaker notion of semi-positive definite.
- Inner products are often used to induce a norm $||\cdot|| : V \to \mathbb{R}$ by defining the map $||\cdot|| : X \mapsto ||X|| := \sqrt{\langle X, X\rangle}$.

With this definition, the metric on a topological manifold is constructed as a section on the associated bundle.

- Let the principal bundle be the frame bundle, defined in the previous section as $P = LM := \bigcup_{p\in M} \{(e_1, ..., e_d) \in T_pM | \, (e_1, ..., e_d) \text{ is a basis of } T_pM\}$ with right-$G = GL(\text{dim}M, \mathbb{R}) = \{h^{m.}_{.n} \in \mathbb{R} | m, n = 1, ..., \text{dim}M; \det h \neq 0\}$ action $\lhd$ on LM defined as $(e_1, ..., e_{\text{dim}M}) \lhd h := (h^{m.}_{.1} e_m, h^{m.}_{.2} e_m, ..., h^{m.}_{.\text{dim}M} e_m)$, which is just the ordinary matrix multiplication rule.
- Define the fibres of the associated fibre metric bundle as $F = \mathbb{R}^{d*} \times \mathbb{R}^{d*}$ with left G-action $\rhd$: $GL(\text{dim}M, \mathbb{R}) \times F \to F$ by $(h \rhd g)_{ab} := (g \lhd h^{-1})_{ab} := g_{cd} h^{c.}_{.a} h^{d.}_{.b}$.
- The metric tensor is a section on the associated bundle $g : M \to T^*_pM \times T^*_pM$ where $g_p : T_pM \times T_pM \to \mathbb{R}$ is a bilinear inner product.

- Thus $LM_F \xrightarrow{\pi_F} M$ is isomorphic as a bundle to $T^p_q M \xrightarrow{\pi} M$, i.e. the $(0,2)$-tensor bundle.

We are now in a position to formally define the Riemannian manifold.

Definition 3.3.2. (Riemannian manifold) A Riemannian manifold (M, g) is a real smooth manifold M with an inner product, defined by the positive definite metric tensor g, varying smoothly on the tangent space of M.

In a system of local coordinates $x : U \to x(U) \subseteq \mathbb{R}^{\dim M}$, for $p \in U \subseteq M$, the induced basis of the tangent space T_pM is the set $\left\{ \left(\frac{\partial}{\partial x^1}\right)_p, ..., \left(\frac{\partial}{\partial x^{\dim M}}\right)_p \right\}$. The following symbol is then defined $g_{ab}(p) := g_p\left(\left(\frac{\partial}{\partial x^a}\right)_p, \left(\frac{\partial}{\partial x^b}\right)_p\right)$.

With a change of coordinate $y = y(x)$, pulling back this basis with $\left(\frac{\partial y^{\tilde{a}}}{\partial x^a}\right)_p$ yields $\left(\frac{\partial}{\partial x^a}\right)_p = \left(\frac{\partial y^{\tilde{a}}}{\partial x^a}\right)_p \left(\frac{\partial}{\partial y^{\tilde{a}}}\right)_p$. The metric tensor is bilinear, which means that we can write $g_{ab}(p) =_1 g_p\left(\left(\frac{\partial}{\partial x^a}\right)_p, \left(\frac{\partial}{\partial x^b}\right)_p\right) =_2 g_p\left(\left(\frac{\partial y^{\tilde{a}}}{\partial x^a}\right)_p \left(\frac{\partial}{\partial y^{\tilde{a}}}\right)_p, \left(\frac{\partial y^{\tilde{b}}}{\partial x^b}\right)_p \left(\frac{\partial}{\partial y^{\tilde{b}}}\right)_p\right) =_3 \left(\frac{\partial y^{\tilde{a}}}{\partial x^a}\right)_p \left(\frac{\partial y^{\tilde{b}}}{\partial x^b}\right)_p g_p\left(\left(\frac{\partial}{\partial y^{\tilde{a}}}\right)_p, \left(\frac{\partial}{\partial y^{\tilde{b}}}\right)_p\right) =_4 \left(\frac{\partial y^{\tilde{a}}}{\partial x^a}\right)_p \left(\frac{\partial y^{\tilde{b}}}{\partial x^b}\right)_p g_{\tilde{a}\tilde{b}}(p)$, where $=_1$ and $=_4$ follow by definition of the symbol $g_{ab}(p)$, $=_2$ follows by definition of the pullback metric and $=_3$ follows from bilinearity of the metric tensor. The pullback metric is a more general transformation than the left Lie-group action on the metric fibre bundle, which follows from the $=_3$ step, bilinearity.

3.3.1 Pullback of general (r, s)-tensor bundles

One needs the additional structure of a Riemannian manifold in order to pull back one forms. For example, if $\varphi : M \to N$ is a smooth map, $\dim M < \dim N$ and (N, g_N) is a Riemannian manifold, then the pushforward of T_pM at a point $p \in M$, i.e. $\varphi_* : T_pM \mapsto \varphi_*(T_pM)$ is only a subspace of, and thus not equal to $T_{\varphi(p)}N$. In order to fix this, we need to define the normal bundle.

Definition 3.3.3. (Normal bundle) Let (M, g) be a Riemannian manifold and $S \subset M$ a Riemannian submanifold. Define, for a given $p \in S$, a vector $n \in T_pM$ to be normal to S whenever $g(n, v) = 0$ for all $v \in T_pS$. Define the normal bundle as follows:

$\mathbf{N}S := \cup_{p\in S} \mathbf{N}_p S := \cup_{p\in S} \{n \in T_pM | g(n, v) = 0 \forall v \in T_pS\}$

Using this definition, we have $T_{\varphi(p)}N = \varphi_*\left(T_pM\right) \bigoplus \mathbf{N}_{\varphi(p)}\varphi\left(M\right)$, and so any $v \in T_{\varphi(p)}N = \varphi_*\left(T_pM\right) \bigoplus \mathbf{N}_{\varphi(p)}\varphi\left(M\right)$ can be projected onto the subspace $\varphi_*\left(T_pM\right)$, and then pulled back via φ_*. In coordinates, this reduces to defining the pullback as the pseudo-inverse of the pushforward φ_*. With this, we have the general pullback of an (r,s)-tensor bundle in coordinates at a point $p \in M$ is given as follows:

$$
\begin{aligned}
&T^{a_1 \ldots a_r}{}_{b_1 \ldots b_s} := \\
&\qquad T\left(dx^{a_1}, ..., dx^{a_r}, \left(\frac{\partial}{\partial x^{b_1}}\right), ..., \left(\frac{\partial}{\partial x^{b_s}}\right)\right) = \\
&T\left(\left(\frac{\partial y}{\partial x}^{-1}\right)^{a_1 \cdot}_{\cdot a'_1} dy^{a'_1}, .. \left(\frac{\partial y}{\partial x}^{-1}\right)^{a_r \cdot}_{\cdot a'_r} dy^{a'_r}, \left(\frac{\partial y}{\partial x}\right)^{b'_1 \cdot}_{\cdot b_1} \left(\frac{\partial}{\partial y^{b'_1}}\right), .. \left(\frac{\partial y}{\partial x}\right)^{b'_s \cdot}_{\cdot b_s} \left(\frac{\partial}{\partial y^{b'_s}}\right)\right) = \\
&\left(\frac{\partial y}{\partial x}^{-1}\right)^{a_1 \cdot}_{\cdot a'_1} \left(\frac{\partial y}{\partial x}^{-1}\right)^{a_r \cdot}_{\cdot a'_r} \left(\frac{\partial y}{\partial x}\right)^{b'_1 \cdot}_{\cdot b_1} \left(\frac{\partial y}{\partial x}\right)^{b'_s \cdot}_{\cdot b_s} T\left(dy^{a'_1}, ..., dy^{a'_r}, \left(\frac{\partial}{\partial y^{b'_1}}\right), ..., \left(\frac{\partial}{\partial y^{b'_s}}\right)\right) \\
&\qquad =: \left(\frac{\partial y}{\partial x}^{-1}\right)^{a_1 \cdot}_{\cdot a'_1} \left(\frac{\partial y}{\partial x}^{-1}\right)^{a_r \cdot}_{\cdot a'_r} \left(\frac{\partial y}{\partial x}\right)^{b'_1 \cdot}_{\cdot b_1} \left(\frac{\partial y}{\partial x}\right)^{b'_s \cdot}_{\cdot b_s} T^{a'_1 \ldots a'_r}{}_{b'_1 \ldots b'_s} \qquad (3.4)
\end{aligned}
$$

This defines the tensor transformation law from $\varphi : M \to N$, where the y-coordinates are in $V \subseteq N$ and the x-coordinates are in $U \subseteq M$. This is very similar to the Lie group actions on the associated fibre bundle, except because the dimensions of the manifolds $\dim M \neq \dim N$, the pushforward matricies $\left(\frac{\partial y}{\partial x}\right)^{b'_1 \cdot}_{\cdot b_1}$ are not square and thus do not have standard linear algebraic inverses $\left(\frac{\partial y}{\partial x}^{-1}\right)^{a_1 \cdot}_{\cdot a'_1}$ as required by the group axiom, and thus the rational for using the generalized Moore-Penrose pseudo inverse.

3.3.1.1 Properties of Riemannian Manifolds

Closed form solutions of the connection 1-form, covariant derivatives and Riemann curvature tensors exist in coordinates, which are all defined from the metric tensor. Thus, once you have the metric tensor, these pieces of information follow immediately. These pieces of information are succinctly defined for completeness.

Definition 3.3.4. (Christoffel symbol) In local coordinates, the Christoffel symbol is given as follows:

$$\Gamma_{abc} := \frac{1}{2}\left(\partial_c g_{ab} + \partial_b g_{ac} - \partial_a g_{bc}\right) \qquad (3.5)$$

Definition 3.3.5. (Covariant derivative) In local coordinates, the covariant derivative of an (r, s) tensor field is given as follows:

$$\begin{aligned}(\nabla_c T)^{a_1 \ldots a_r}{}_{b_1 \ldots b_s} := \partial_c T^{a_1 \ldots a_r}{}_{b_1 \ldots b_s} \\ + \left(\Gamma^{a_1}_{dc} T^{d \ldots a_r}{}_{b_1 \ldots b_s} + \ldots + \Gamma^{a_r}_{dc} T^{a_1 \ldots d}{}_{b_1 \ldots b_s}\right) \\ - \left(\Gamma^{d}_{b_1 c} T^{a_1 \ldots a_r}{}_{d \ldots b_s} + \ldots + \Gamma^{d}_{b_s c} T^{a_1 \ldots a_r}{}_{b_1 \ldots d}\right)\end{aligned} \tag{3.6}$$

Definition 3.3.6. (Riemann curvature tensor) The Riemann curvature tensor is defined:

$$R(U, V) W := \nabla_U \nabla_V W - \nabla_V \nabla_U W - \nabla_{[U,V]} W \tag{3.7}$$

In local coordinates, the Riemann curvature tensor is:

$$R^d{}_{abc} = \frac{\partial}{\partial x^b} \Gamma^d_{ac} - \frac{\partial}{\partial x^c} \Gamma^d_{ab} + \Gamma^d_{bs} \Gamma^s_{ac} - \Gamma^d_{cs} \Gamma^s_{ab} \tag{3.8}$$

Many other extremely important objects exist, such as the Ricci tensor $R_{ab} := R^c{}_{acb}$ and the Ricci scalar $R := R^a_a$, but because these objects are not used in the geometric formulation of neural networks in Chapter 4 they will not be developed.

Chapter 4
The Geometrical Anatomy of Neural Networks

The purpose of this chapter is to formulate neural network actions on data manifolds as a branch of Riemannian geometry [81]. As mentioned, the minimum mathematical structure that has coordinate systems is a topological manifold, and the minimum mathematical structure that endows a topological manifold with shape is a Riemannian manifold. This follows from the fact that a Riemannian manifold is a topological manifold with a smoothly varying inner product on the surface of the manifold.

As mentioned earlier, data manifolds often have an intrinsic shape to them. This is because in the learned embedded representation, features of similar objects are near each other while features of dissimilar objects are far apart. For example, the intra-cluster distance of features of dogs will be small, as will the intra-cluster distance of features of cars, however the inter-cluster distance of these two individual class clusters will be large.

For intuition of data manifolds, taking an image and making it larger or smaller will yield the topological space $\mathbb{R}$. This is because after properly compressing the image the only information left about the image is the size/scale of the image. The topology of the manifold does not necessarily need to be Euclidean however. For example, taking an image and simply rotating it in a circle will yield the topological space S^1. Both scaling and rotating an image will yield the topological space $\mathbb{R} \times S^1$, as seen in Figure 1.2.

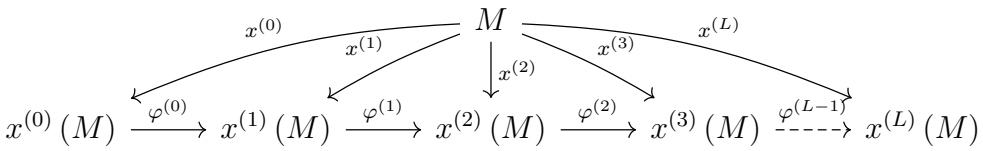


Figure 4.1: Coordinate systems $x^{(l+1)} := \varphi^{(l)} \circ ... \circ \varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)}$ induced by the coordinate transformations $\varphi^{(l)} : x^{(l)}(M) \to \left(\varphi^{(l)} \circ x^{(l)}\right)(M)$ learned by the neural network. The pullback metric $g_{x^{(l)}(M)}(X,Y) := g_{\left(\varphi^{(l)} \circ x^{(l)}\right)(M)}\left(\varphi_*^{(l)} X, \varphi_*^{(l)} Y\right)$ pullsback (i.e. backpropagates) the coordinate representation of the metric tensor from layer $l+1$ to layer l, via the pushfoward map $\varphi_*^{(l)} : Tx^{(l)}(M) \to T\left(\varphi^{(l)} \circ x^{(l)}\right)(M)$ between tangent spaces.

4.1 Coordinate representation of the data manifold

The objective of the neural network is to find a compressed representation of the input data; this holds for both supervised as well as unsupervised learning. The input coordinate representation is rather arbitrary. For example RGB coordinates are used to represent images because images in RGB coordinates can be displayed on a monitor. However RGB is not a good coordinate representation for data compression, which tends to be a flattened representation of the data manifold. For this, the neural network learns a sequence of coordinate transformations to put the data in a flattened form.

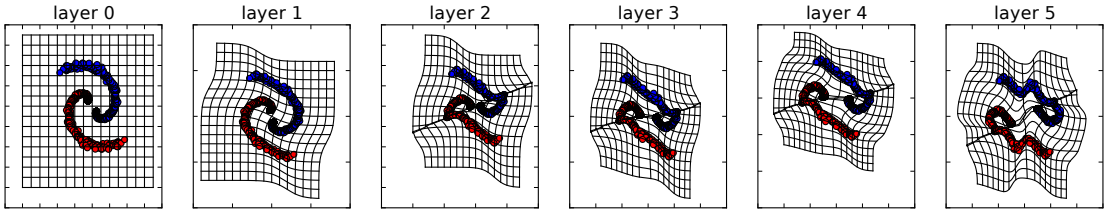
The input coordinates are Cartesian:

$$x^{(0)} : M \to x^{(0)}(M) \subseteq \mathbb{R}^{\dim x^{(0)}(M)} \tag{4.1}$$

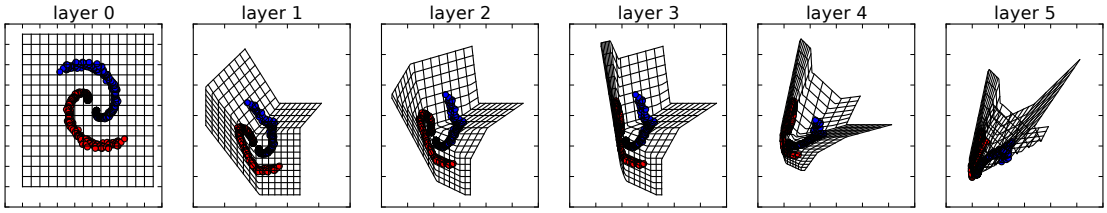
A standard feed forward network has the form $x^{(l+1)} = f\left(x^{(l)}; l\right)$, whereas a residual network has the form $x^{(l+1)} = x^{(l)} + f\left(x^{(l)}; l\right)$ for some nonlinear activation function $f : \mathbb{R}^d \to \mathbb{R}^d$ applied element-wise. If it is fully connected, then $x^{(l+1)} = f\left(W^{(l)} \cdot x^{(l)} + b^{(l)}\right)$ and $x^{(l+1)} = x^{(l)} + f\left(W^{(l)} \cdot x^{(l)} + b^{(l)}\right)$ for the standard and residual cases. Similarly if it is convolutional then $x^{(l+1)} = f\left(W^{(l)} * x^{(l)}\right)$ and $x^{(l+1)} = x^{(l)} + f\left(W^{(l)} * x^{(l)}\right)$ again for the standard and residual cases. The point is all of these transformations can be written in the following form:

$$\varphi^{(l)} : x^{(l)} \to \varphi^{(l)}\left(x^{(l)}\right) \tag{4.2}$$

$$\varphi^{(l)} : x^{(l)}(M) \mapsto x^{(l+1)}(M) := \left(\varphi^{(l)} \circ x^{(l)}\right)(M) \tag{4.3}$$



(a) A $\mathcal{C}^1$ network with a hyperbolic tangent activation function.



(b) A $\mathcal{C}^1$ network with a rectified linear ReLu $(z) := \max(0, z)$ activation function.

Figure 4.2: Untangling and linearly separating the same spiral manifold with 2-dimensional $\mathcal{C}^1$ neural networks with different types of nonlinear coordinate transformations. The x and y axes are the two nodes of the neural network at a given layer l, where layer 0 is the input representation and layer 5 is the output representation for linear classification. Both networks exhibit smooth layerwise coordinate transformations whose coordinates are slight perturbations from the previous layer. The data does not change, only the layer-to-layer coordinate representation of the data changes. Depending on the types of coordinate transformations available to the network, the final coordinate representation of the manifold varies.

More complex networks, such as dense networks [16], can be accommodated into this formalism.

Equation 4.3 is recursive, initialized in Cartesian coordinates by Equation 4.1. Thus the coordinate representation of the data manifold at an arbitrary layer l is explicitly solved:

$$x^{(l+1)} : M \to x^{(l+1)}(M) \tag{4.4}$$

$$x^{(l+1)} : M \mapsto x^{(l+1)}(M) := \left(\varphi^{(l)} \circ .... \circ \varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)}\right)(M) \tag{4.5}$$

Less notationally involved, this is just solving $x^{(l+1)} := \varphi^{(l)}\left(x^{(l)}\right)$ with the initialization $x^{(0)}$, yielding $x^{(l+1)} := \left(\varphi^{(l)} \circ .... \circ \varphi^{(1)} \circ \varphi^{(0)}\right)\left(x^{(0)}\right)$, which itself is just the hidden layer representation at layer $l+1$.

From this perspective, the objective of the neural network is to find a sequence of

coordinate transformations, starting at the input arbitrary coordinates, such that in the output, the data manifold is flat.

The procedure for sequentially transforming the coordinate representation of the manifold can be seen in Figures 4.2a and 4.2b. In Figure 4.2a the hyperbolic tangent $\tanh(z) := \frac{e^z - e^{-z}}{e^z + e^{-z}}$ nonlinearity was used, whereas in Figure 4.2b the $\text{ReLu}(z) := \max(0, z)$ nonlinearity was used. In both cases a C^1 network was used, and all training processes such as batch size and learning rate were kept the same. In the input Cartesian coordinates, the data manifold is seen to not be linearly separable by hyperplane. In the output coordinates the data manifold is flattened and made such that the different classes are linearly separable. It is seen that the hyperbolic tangent nonlinearity smoothly distorts the Cartesian coordinates, whereas the ReLu nonlinearity sharply "folds" the coordinates, such that the entirety of the curvature of the manifold is contained in these finitely countable number of folds; manifolds of this form are called piecewise linear.

Remark. (Embedded representations) Although the network is forward propagating the coordinate representation of the data manifold, within this formulation the data manifold is best represented in the *output* coordinates. The input coordinates are largely arbitrary and oversampled. The output coordinates represent the manifold as flat. Thus it is more intuitive to think of the coordinate representation of the data manifold at the output, where the data manifold is represented in the intuitive coordinates that are *not* arbitrary and oversampled. The network then finds a way to transform the arbitrary coordinates at the input to this better compressed representation at the output. For example, the topological manifold S^1 is intuitively understood in $\mathbb{R}^2$ as a circle in Cartesian coordinates, or in $\mathbb{R}^1$ in polar coordinates. If instead it was represented as a 100×100 RGB image by embedding it in $\mathbb{R}^{100\times100\times3} = \mathbb{R}^{30{,}000}$, this is *not* an intuitive/compressed way to represent this topological manifold. Thus in the reverse direction the neural network is taking the topological manifold S^1 in, for example, $\mathbb{R}^2$ in Cartesian coordinates and embedding it in the arbitrary image coordinates in $\mathbb{R}^{100\times100\times3} = \mathbb{R}^{30{,}000}$.

In practice the number of nodes often changes as one moves through the layers of the neural network. By this formulation, changing the number of nodes is equivalent to changing the number of dimensions the neural network is using to represent the data manifold. We then have that the dimension of the underlying data manifold is the number of dimensions in the smallest, bottleneck layer of the neural network:

$$\dim M := \min_l \dim x^{(l)}(M) \tag{4.6}$$

All other higher dimensional layers are immersion/embedding representations of this lowest dimensional representation.

Being able to change dimensions is important for several reasons, two of which are listed below:

- Neural networks are learning global coordinate systems (the entire manifold at the input is represented in Cartesian coordinates, e.g. all images are represented in RGB) and so going into a higher dimensional space allows some flexibility in representing the manifold. For example, the Whitney embedding theorems, which state that *any* $\dim M$-dimensional topological manifold M can be embedded in $\mathbb{R}^{2\dim M}$, with better lower bounds existing for specific topologies. Thus by moving into higher-dimensional spaces, the network has greater flexibility in manipulating the manifold.

- Neural networks usually have access to a limited number of types of nonlinear coordinate transformations, for example $\tanh$, σ and ReLU. This severely limits the ability of the network to separate the wide variety of manifolds that exist. For example, the networks have difficulty linearly separating the simple toy spirals in Figures 4.6 because they only have access to coordinate transformations of the form $\tanh$. If instead they had access to a coordinate system that was more appropriate for spirals, such as polar coordinates, they could very easily separate the data.

  This is the reason why Locally Linear Embeddings [22] could very easily discover the coordinate charts for the underlying manifold, because k-nearest neighbors is an extremely flexible type of nonlinearity.

  Allowing the neural network to go into higher dimensions by designing them to have large numbers of nodes makes it easier to separate data in this high dimensional space. The current state-of-the-art neural network for image recognition is the dense network [16], which is a very complex way of combining nonlinearities and thus giving it the possibility of well fitting many types of manifolds. This is a far departure from the standard deep neural network which was simply an affine transformation followed by a nonlinear activation.

Remark. (Induced coordinate systems) Important distinctions with the usual intuitions from differential geometry are noted here.

- Usually in differential geometry a coordinate transformation from the $x^{(l)}: U \to x^{(l)}(U)$ coordinate system to the $x^{(l+1)}: V \to x^{(l+1)}(V)$ coordinate system, with $U \cap V \neq \phi$, is defined from these coordinates as $x^{(l+1)} \circ \left(x^{(l)}\right)^{-1}: U \cap V \to \left(x^{(l+1)} \circ \left(x^{(l)}\right)^{-1}\right)(U \cap V)$. Thus a coordinate transformation is usually thought of as the mapping from two already known coordinate systems. With neural networks the coordinate system is *not known a priori*; instead the coordinate system is *induced* by the transformation $\varphi^{(l)}: x^{(l)}(M) \mapsto x^{(l+1)}(M) := \left(\varphi^{(l)} \circ x^{(l)}\right)(M)$ learned by the network, with the objective that the final coordinate system $x^{(L)}(M) := \left(\varphi^{(L-1)} \circ .... \circ \varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)}\right)(M)$ well represents/embeds/compresses the data manifold.

- As mentioned earlier, the coordinate system used in neural networks is a global system. All images use the same neural network for compression; images of dogs do not use a different neural network than images of cars.

For changing dimensions, as mentioned in Chapter 3, this situation is handled by considering submersions, immersions and embeddings. In the developed notation of neural networks, the transformation $\varphi^{(l)}: x^{(l)}(M) \to x^{(l+1)}(M)$, where $x^{(l)}(M) \subseteq \mathbb{R}^{\dim x^{(l)}(M)}$ and $x^{(l+1)}(M) \subseteq \mathbb{R}^{\dim x^{(l+1)}(M)}$. Note that, as mentioned earlier, it is more intuitive to think in the reverse direction, namely $\left(\varphi^{(l)}\right)^{-1}: x^{(l+1)}(M) \to x^{(l)}(M)$. Because $x^{(l+1)}(M)$ and $x^{(l)}(M)$ are topological spaces endowed with the standard topology, it makes sense to talk about smoothness of $\left(\varphi^{(l)}\right)^{-1}: x^{(l+1)}(M) \to x^{(l)}(M)$. The rank of $\left(\varphi^{(l)}\right)^{-1}$ is the rank of the linear map $\left(\varphi^{(l)}\right)^{-1}_{*}: Tx^{(l+1)}(M) \to Tx^{(l)}(M)$, which is the rank of the Jacobian matrix of partial derivatives. The map is said to be of constant rank if the rank is constant over all points in the chart.

- The map $\left(\varphi^{(l)}\right)^{-1}: x^{(l+1)}(M) \to x^{(l)}(M)$ is called a *submersion* if $\left(\varphi^{(l)}\right)^{-1}_{*}$ is surjective at each point. This requirement of a submersion is equivalent to rank $\left(\varphi^{(l)}\right)^{-1} = \dim x^{(l)}(M)$.

- The map $\left(\varphi^{(l)}\right)^{-1}: x^{(l+1)}(M) \to x^{(l)}(M)$ is called an *immersion* if $\left(\varphi^{(l)}\right)^{-1}_{*}$ is injective at each point. This requirement of an immersion is equivalent to rank $\left(\varphi^{(l)}\right)^{-1} = \dim x^{(l+1)}(M)$.

- If the map $\left(\varphi^{(l)}\right)^{-1} : x^{(l+1)}(M) \to x^{(l)}(M)$ is an immersion and preserves the topology of $x^{(l+1)}(M)$ then the mapping is called an *embedding*. Preserving the topology of the space means the map is a homeomorphism onto its image $\left(\varphi^{(l)}\right)^{-1}\left(x^{(l+1)}(M)\right) \subseteq x^{(l)}(M)$.

Remark. (Embeddings and immersions) An embedding is a stronger requirement than an immersion. For example, one can immerse S^1 into $\mathbb{R}^2$ in the normal way as a circle, or in a weird way in which the space crosses over itself. Both ways are immersions, but only the first is an embedding. The second fails to be an embedding where the space crosses over itself, because at this point it is no longer locally homeomorphic to $\mathbb{R}^d$ and thus fails to be a manifold. From the perspective of data compression, a mapping that is an embedding should better preserve information about the data manifold than a mapping that is only an immersion.

4.1.1 Discussion

This subsection will discuss experiments and their results for understanding neural networks as coordinate representations of data manifolds.

4.1.1.1 Coordinate representations of data manifolds

The network is learning a sequence of coordinate transformations, beginning with Cartesian coordinates, to find a coordinate representation of the data manifold that is required by the cost function. This process can be visualized in Figure 4.3. This experiment used a $\mathcal{C}^1$ network with an ℓ^2 minimization between the input and output data, so the group actions on the principal and associated bundles act smoothly along the fibres. Grid lines were not displayed in the other experiments so the specific effects of the other experiments can be more clearly seen.

In the forward direction, beginning with Cartesian coordinates, a sequence of $\mathcal{C}^1$ differential coordinate transformations is applied to find a nonlinear coordinate representation of the data manifold such that in the output coordinates the classes satisfy the cost restraint. In the reverse direction, starting with a standard Euclidean metric at the output Equation 4.7, the coordinate representation of the metric tensor is backpropagated through the network to the input by Equations 4.8-4.9 to find the metric tensor representation in the input Cartesian coordinates.

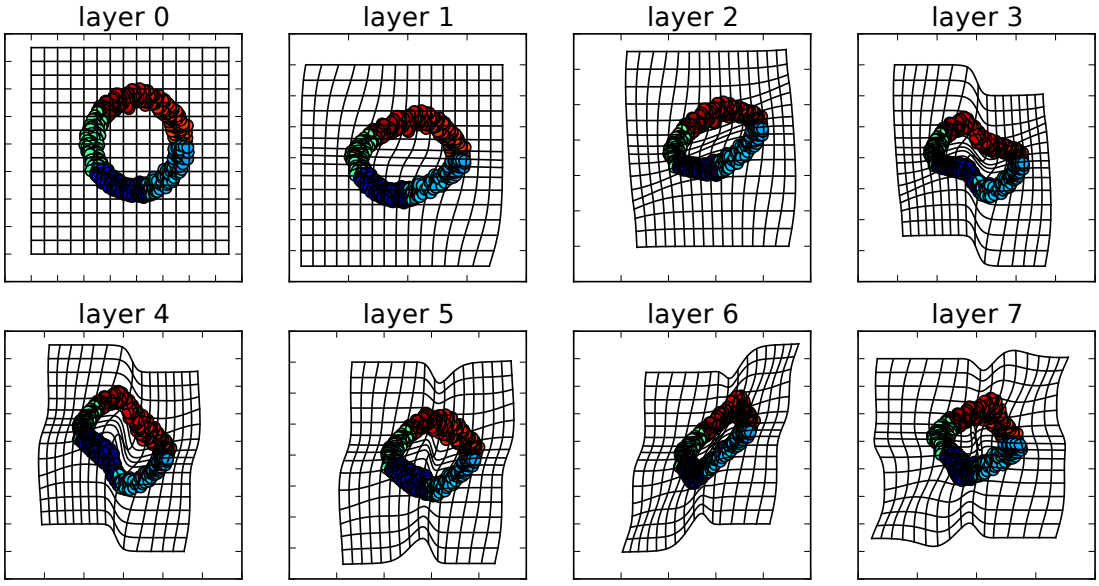
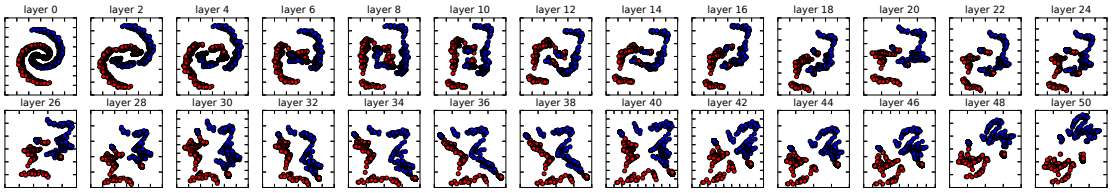


Figure 4.3: Layerwise coordinate transformations with a $\mathcal{C}^1$ (residual) network, used to change the shape of the input to match an output via ℓ^2 minimization. The coordinate transformations take place (approximately) smoothly over the network as the next layer is a slight perturbation from the previous. Also note that if distances at the output are measured by the Euclidean metric, then to preserve the metric properties from the input, the output coordinate space becomes non Cartesian.

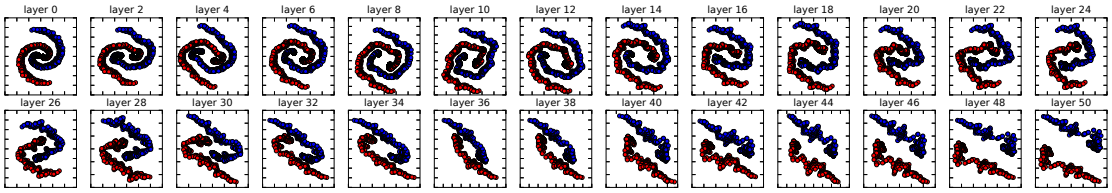
4.1.1.2 Effect of batch size on set connectedness and topology

A basic theorem in topology says that continuous functions map connected sets to connected sets [75]. However, in Figure 4.4a it is seen that as early as layer 4 the continuous neural network is breaking the connected input set into disconnected sets. Additionally, and although it achieves 0% error, it is learning very complicated and unintuitive coordinate system to represent the data in a linearly separable form. This is because during training, with a small batch size of 300 in the stochastic gradient descent search, the underlying manifold was not sufficiently sampled to represent the entire connected manifold and so it seemed disconnected. (In actuality the spaces remain connected because the mapping is continuous, but because of the nonlinear form of the equations the space bifurcates and gives the appearance, with the finite amount of data, that the manifolds become disconnected).

This is compared to Figure 4.4b in which a larger batch size of 1000 was used and was sufficiently sampled to represent the entire connected manifold, and the network was also able to achieve 0% error. The coordinate transformations learned by the neural



(a) A batch size of 300 for untangling data. As early as layer 4 the input connected sets have been disconnected and the data are untangled in an unintuitive way. This means a more complex coordinate representation of the data manifold was learned.



(b) A batch size of 1000 for untangling data. Because the large batch size can well-sample the data manifold, the spiral sets stay connected and are untangled in an intuitive way. This means a simple coordinate representation of the data manifold was learned.

Figure 4.4: The effect of batch size on coordinate representation learned by the same 2-dimensional $\mathcal{C}^1$ network, where layer 0 is the input representation, and both examples achieve 0% error. A basic theorem in topology says continuous functions map connected sets to connected sets. A small batch size of 300 during training sparsely samples from the connected manifold and the network learns overfitted coordinate representations. With a larger batch size of 1000 during training the network learns a simpler coordinate representation and keeps the connected input connected throughout.

network with the larger batch size seems to more intuitively untangle the data in a simpler way than that of Figure 4.4a.

Note that this experiment is in 2-dimensions, and with higher dimensional data the issue of batch size and set connectedness becomes exponentially more important as the curse of dimensionality makes it exponentially more difficult to well-sample the data manifold.

4.2 Coordinate representation of the metric tensor

From the perspective of differentiable geometry, as one moves through the layers of the neural network, the data manifold stays the same but the coordinate representation of the data manifold changes with each successive affine transformation and nonlinear activation. The objective of the neural network is to find a coordinate representation of the data manifold such that the classes are linearly separable by hyperplanes.

If the network has been well trained to encode the data, then by Euclidean distance two input points of the same class may be far apart when represented by the input coordinates but close together in the output coordinates. Similarly, two points of different classes may be near each other when represented by the input coordinates but far apart in the output coordinates. These ideas form the basis of Locally Linear Embeddings [22]. The intuitive way to measure distances is in the output coordinates, which even in the unsupervised case tends to learn a flattened representation of the data manifold [13]. Accordingly, the metric representation in the output coordinates is defined as the standard Euclidean metric:

$$g\left(x^{(L)}\right)_{a_L b_L} := \eta_{a_L b_L} \tag{4.7}$$

The standard notation $g\left(x^{(l)}\right)_{a_l b_l} := g\left(\left(\frac{\partial}{\partial x^{(l)}}\right)_{a_l}, \left(\frac{\partial}{\partial x^{(l)}}\right)_{b_l}\right)$ introduced in Chapter 3 is used here, where $\left(\frac{\partial}{\partial x^{(l)}}\right)_{c_l} \in TM$ being the natural basis in the frame bundle induced by the coordinate system.

The elements of the metric tensor transforms as a tensor with coordinate transformations. This is explained in detail in Chapter 3, and follows from the metric tensor being multilinear in its arguments and thus transforming by left Lie group actions along the associated fibre bundle, as well as by the definition of the pullback metric.

$$g\left(x^{(l)}\right)_{a_l b_l} = \left(\frac{\partial x^{(l+1)}}{\partial x^{(l)}}\right)^{a_{l+1}\cdot}_{\cdot a_l} \left(\frac{\partial x^{(l+1)}}{\partial x^{(l)}}\right)^{b_{l+1}\cdot}_{\cdot b_l} g\left(x^{(l+1)}\right)_{a_{l+1} b_{l+1}} \tag{4.8}$$

The above recursive formula is solved from the output layer to the input, i.e. the coordinate representation of the metric tensor is backpropagated through the network from output to input. The recursive Equation 4.8 is initialized at the output where the coordinate representation of the metric tensor is Euclidean, Equation 4.7, and its solution for any layer l is given as follows:

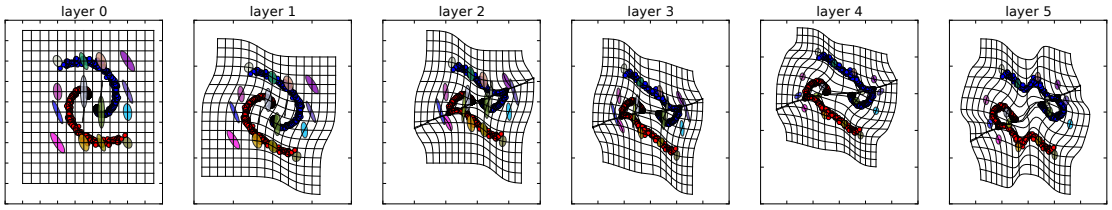


Figure 4.5: A $\mathcal{C}^1$ network with a hyperbolic tangent activation function separating the spiral manifold. Additionally, balls of constant radius $ds = \sqrt{g_{a_l b_l}(x^{(l)}) dx^{(l)a_l} dx^{(l)b_l}}$ at different points are shown. In the output coordinates, they are measured by standard Euclidean distance in Equation 4.7, and so the balls are "round". The coordinate representation of the metric tensor is backpropagated through the network to the input by Equations 4.8 and 4.9; so distances on the data manifold can be measured with the input coordinates, and so the balls are not round. The balls are represented as ellipses, with the principal components of the metric tensor acting as the principal components of the ellipses. These balls can also be interpreted as forming an $\epsilon - \delta$ relationship across layers of the network, where an ϵ-ball at one layer corresponds to a δ-ball at the previous layer.

$$g\left(x^{(l)}\right)_{a_l b_l} = \prod_{l'=L-1}^{l} \left[\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)^{a_{l'+1}\cdot}_{\cdot a_{l'}} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)^{b_{l'+1}\cdot}_{\cdot b_{l'}} \right] \eta_{a_L b_L} \tag{4.9}$$

The procedure for arriving at the closed form solution of Equation 4.9 is very similar to the procedure for arriving at the closed form solution of the error backpropagation algorithm; this is no coincidence and will be elaborated in Section 4.3.

Equation 4.9 is the coordinate representation of the metric tensor in the coordinate system at layer l, namely $x^{(l)} := \varphi^{(l-1)} \circ .... \circ \varphi^{(1)} \circ \varphi^{(0)} \circ x^{(0)} : M \to x^{(l)}(M)$. These coordinates are hard to interpret, other than at the input $l = 0$ where we know the coordinates are Cartesian. Using Equation 4.9 we have the closed form solution of the metric tensor on the data manifold in the input, Cartesian coordinates:

$$g\left(x^{(0)}\right)_{a_0 b_0} = \prod_{l'=L-1}^{0} \left[\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)^{a_{l'+1}\cdot}_{\cdot a_{l'}} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)^{b_{l'+1}\cdot}_{\cdot b_{l'}} \right] \eta_{a_L b_L} \tag{4.10}$$

The neural network actions on the data manifold can be well understood in Figure 4.5, where a C^1 network was used. In the forward direction, as in Figures 4.2a and 4.2b, the coordinate representation of the data manifold are sequentially nonlinearly transformed such that in the output coordinates the classes of the data manifold can be linearly separated by hyperplane. In the reverse direction, the coordinate representation of the

metric tensor is pulled back (backpropagated) through the layers of the network. In the output where the space is flat, a Euclidean metric is used, thus the balls of constant radius are "round." The ellipsoids are generated by using the principal components of the metric tensor at different points in the space. As one pulls back the coordinate representation of the metric tensor and the metric tensor is transformed by sequential Lie group actions, the balls of constant radius are deformed when represented in the new coordinates. This is continued to the input Cartesian coordinates where the metric tensor measures distances appropriate for the coordinate representation of the space.

Remark. (Assumptions on the derivation) For clarity, the major assumptions made to reach the solutions given by Equation 4.9 and 4.10 will be itemized below.

- The biggest assumption made is that at the output the data manifold has a flattened representation, namely Equation 4.7. This assumption is supported however by large amounts of results spanning many research groups over several years [13, 19–22]. Additionally, the more flexible the coordinate transformation used, such as residual networks and dense networks, the more accurate the assumption seems to be that the data manifold representation is flattened at the output. This follows from experiments where, for example, the linear interpolation in the embedding space produces more plausible looking images for these more flexible types of coordinate transformations [31].

- A more minor assumption is that the transformation $\left(\varphi^{(l)}\right)^{-1} : x^{(l+1)}(M) \rightarrow x^{(l)}(M)$ is constant rank. If it is purely ReLu then it is a non-injective surjective function, and thus not a bijection, and not constant rank (since in regions where it is zero it has rank 1, as opposed to the regions where it is greater than zero it has full rank. This is not a problem with the theory, it is a problem with the coordinate transformation. In practice, residual networks or dense networks are used with the ReLu activation functions, in which case the transformation is full rank. Additionally, quite often smooth approximations to the Relu activation, such as the softplus function $f(x) := \ln(1 + e^x)$, are used which are full rank.

  The only way the weight matrix at a given layer would not be full rank would be if the nodes were to learn exactly equivalent filters, and given that the weights are random initialized this is extremely unlikely. Moreover, networks are often designed to ensure this does not happen [5].

- If the network is oddly designed such that the bottle-neck layer is not at the output, but at some other layer of the neural network, then one will be submersing the data manifold (defined at the output) into a lower dimensional space than the dimension of the output. This is a very unintuitive way of designing a neural network, but still allowed so long as the pushforward is of constant rank.

  In practice the data manifold is partly determined by the network; for example, with ReLu transformations the manifold is piecewise linear and thus the curvature is contained in the sharp edges, whereas for a smooth transformation like $\tanh$ the manifold is not piecewise linear and thus the curvature is everywhere. Because of this, the network may be compressing the manifold to the lowest dimensional space inside the network and then embedding it into the higher-dimensional output space, i.e. Equation 4.6.

The data manifold is independent of coordinate representation. At the output where distances are measured by the standard Euclidean metric an ϵ-ball can be defined. The linear element in Equation 4.9 defines the corresponding δ-ball at layer l. This can be used to see what in the input space the neural network says is close in the output space.

To the authors' best knowledge this is the first $\epsilon - \delta$ output-input relation discovered for deep neural networks. A second derivation of this relation is made in Section 4.5 where perturbation theory is applied to deep neural networks, and coming to the same result.

The importance of this $\epsilon - \delta$ relation, in addition to its purely theoretical significance, is that the output coordinates capture the high-level, intuitive relationships that exist between data, whereas the input coordinates are Cartesian and easy to interpret. Thus the $\epsilon - \delta$ relation provides a window into understanding how neural networks function, by telling the user in the input representation what the neural network identifies as similar in the high-level output representation.

This formulation naturally lends itself to be abstracted in a coordinate free manner. Thus one can understand and define backpropagating the metric tensor in a rigorous mathematical sense.

The fibre F in the associated bundle will be the metric tensor space, in which case $F = \left(\mathbb{R}^d\right)^* \times \left(\mathbb{R}^d\right)^*$, where the $*$ denotes the cospace. With this, the left G-action is a map $\triangleright$: $GL\left(d, \mathbb{R}\right) \times F \to F$ defined as $\left(h \triangleright g\right)_{a_l b_l} = \left(g \triangleleft h^{-1}\right)_{a_l b_l} := g_{a_{l+1} b_{l+1}} h^{a_{l+1}}_{\cdot a_l} h^{b_{l+1}}_{\cdot b_l}$, as described in Chapter 3.

Layerwise sequential applications of the left G-action from output to input is thus simply understood:

$$(h_0 \rhd h_1 \rhd ... \rhd h_L \rhd g)_{a_0 b_0} =$$

$$(h_0 \bullet h_1 \bullet ... \bullet h_L) \rhd g_{a_L b_L} = \prod_{l'=L-1}^{0} \left(h^{a_{l'+1} \cdot}_{\cdot a'_l} h^{b_{l'+1} \cdot}_{\cdot b_{l'}} \right) g_{a_L b_L} \quad (4.11)$$

Equation 4.10 is equivalent to Equation 4.11, although Equation 4.11 is formulated in a formal, abstract sense of Lie group elements acting on a tensor bundle without reference to a specific coordinate system, whereas Equation 4.10 is dependent on coordinate systems.

4.3 Coordinate representation of the frame bundle

The error backpropagation algorithm is usually understood as sequentially applying the chain rule on nested compositions of nonlinear functions. The purpose of this section is to show that the error backpropagation algorithm can be naturally incorporated into the formulation of Riemannian geometry as left Lie group actions acting on the principal fibre frame bundle. The frame bundle acts on the error cost functional, and so this formulation says that the frames are pulled back via left Lie group actions. Thus the derivative of the error cost functional is changing throughout the network because it is being represented in different coordinate systems as the Lie group acts on the fibres of the associated bundle. This formulation in terms of Riemannian geometry, although mathematically more difficult, allows for understanding the error backpropagation algorithm in a deeper way than merely saying it is the chain rule and ending the interpretation there.

To quickly review, the pertinent information regarding a Riemannian manifold is itemized below:

- Let the principal bundle be the frame bundle, defined as follows: $P = LM := \bigcup_{p \in M} \{p\} \cup \{(e_1, ..., e_d) \in T_pM | \, (e_1, ..., e_d) \text{ is a basis of } T_pM\}$ with right-$G = GL\left(\text{dim}M, \mathbb{R}\right) = \{h^{m.}_{.n} \in \mathbb{R} | m, n = 1, ..., \text{dim}M; \det h \neq 0\}$ action $\triangleleft$ on LM defined as $(e_1, ..., e_{\text{dim}M}) \triangleleft h := (h^{m.}_{.1} e_m, h^{m.}_{.2} e_m, ..., h^{m.}_{.\text{dim}M} e_m)$, which is just the ordinary matrix multiplication rule.

- Define the fibres of the associated fibre metric bundle as $F = \left(\mathbb{R}^d\right)^* \times \left(\mathbb{R}^d\right)^*$ with left G-action $\triangleright$: $GL\left(\text{dim}M, \mathbb{R}\right) \times F \to F$ by the inverse of the right G-action, namely: $\left(h \triangleright g\right)_{ab} := \left(g \triangleright h^{-1}\right)_{ab} := g_{cd} h^{c.}_{.a} h^{d.}_{.b}$.

- The metric tensor is a section on the associated bundle $g : M \to T^*_pM \times T^*_pM$ where $g_p : T_pM \times T_pM \to \mathbb{R}$ is a bilinear inner product.

- Thus $LM_F \xrightarrow{\pi_F} M$ is isomorphic as a bundle to $T^p_qM \xrightarrow{\pi} M$, i.e. the $(0, 2)$-tensor bundle.

In the coordinates $x^{(l)} : M \to x^{(l)}\left(M\right)$ at layer l, the frame bundle at $p \in M$ is explicitly written as $L_pM = \left\{ \left(\frac{\partial}{\partial x^{(l)}}\right)_{p,1}, ..., \left(\frac{\partial}{\partial x^{(l)}}\right)_{p,\text{dim}M} \right\}$.

For the optimization process, a given loss $l\left(y, h_\Theta\left(x^{(0)}\right)\right)$ is given, and the error is defined as the expected loss over all data samples. Assuming the data samples are iid, the total error is:

$$E\left(\Theta\right) := \frac{1}{N}\sum_{n=1}^{N} l\left(y, h_\Theta\left(x^{(0)}\right)\right) \tag{4.12}$$

As derived in Chapter 2, the purpose of the error backpropagation algorithm is to efficiently find the derivatives $\frac{\partial E}{\partial \Theta}$ for the gradient descent search over parameters $\Theta^{t+1} \leftarrow \Theta^t - \lambda \frac{\partial E}{\partial \Theta}$. Neural networks are nested compositions of non-linear functions, and so the derivatives $\frac{\partial E}{\partial \Theta}$ are found by simply applying the ordinary chain rule of differential calculus sequentially.

$$\frac{\partial E}{\partial W^{(l-1)}} = \left(\frac{\partial E}{\partial x^{(L)}}\right) \cdot \prod_{l'=L-1}^{l} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right) \cdot \left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right) \tag{4.13}$$

In practice, Equation 4.13 is further expanded by writing $\frac{\partial x^{(L)}}{\partial W^{(L-1)}} = \frac{\partial x^{(L)}}{\partial z^{(L-1)}} \cdot \frac{\partial z^{(L-1)}}{\partial W^{(L-1)}}$ so that explicit gradients with respect to parameter weights are derived for the gradient descent search.

Remark. (Information geometry) Note that $W^{(l-1)}$ is a coordinate chart on the parameter manifold [71], *not* the data manifold. To be mathematically rigorous, instead of the partial derivative $\partial_{W^{(L-1)}} \equiv \frac{\partial}{\partial W^{(L-1)}}$ being used, the covariant derivative $\nabla_{W^{(L-1)}}$ should be used. This is because quite often the parameter manifold W that the gradient descent search is being performed over is not Euclidean $\mathbb{R}^{\dim W}$, but instead a general topological manifold. This happens for example when parameter weights are restricted to the surface of a sphere.

Here in Equation 4.13 the $\cdot$ operation is defined to be the ordinary matrix multiplication rule and $\prod_{l'=L-1}^{l}$ is sequential matrix multiplication. To be more explicit for the formulation in geometry, we rewrite Equation 2.38 in Einstein notation:

$$\frac{\partial E}{\partial W^{(l-1)}} = \left(\frac{\partial E}{\partial x^{(L)}}\right)_{a_L} \prod_{l'=L-1}^{l} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)^{a_{l'+1}\cdot}_{\cdot a_{l'}} \left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right)^{a_l} \tag{4.14}$$

In this form it is immediately seen that error backpropagation is a sequence of right G-actions $\prod_{l'=l}^{L-1} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)^{a_{l'+1}\cdot}_{\cdot a_{l'}}$ on the output frame bundle $\left(\frac{\partial}{\partial x^{(L)}}\right)_{a_L}$. This transforms (pulls-back) the frame bundle acting on E to the coordinate system at layer l, and thus

puts it in the same space as $\left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right)^{a_l}$.

Similarly as was done with formulating metric transformations in a formal, abstract sense, the same should be done with the standard error backpropagation algorithm. This is easily done since it is now seen to be right G-actions on the frame bundle.

$$\left(e \lhd h_L \lhd h_{L-1} \lhd ... \lhd h_0\right)_{a_0} =$$

$$e \lhd \left(h_L \bullet h_{L-1} \bullet ... \bullet h_0\right)_{a_0} = \prod_{l'=L-1}^{0} h_{.a'_l}^{a_{l'+1}\cdot} e_{a_L} \quad (4.15)$$

For $e \in LM$ being an element of the frame bundle and $h \in G$ being a group action acting on the frame bundle via right G-action $\lhd$: $LM \times G \to LM$. The sequential right Lie group actions acting on the frame bundle is seen to be at the heart of the backpropagation algorithm. It is seen that the second axiom of group actions, namely $p \lhd h_1 \lhd h_2 = p \lhd (h_1 \bullet h_2)$ is vital in formulating neural networks in this sense of abstract algebra.

Remark. (Backpropagation as group actions on fibre bundles) Similarities and differences between backpropagating the metric tensor and backpropagating the error will be itemized here.

- It is immediately seen that Equation 4.9 and Equation 4.14 are similar because they both have the sequential group actions on the associated and principal bundles, respectively.
  - Equation 4.9 on metric transformations:

$$g\left(x^{(l)}\right)_{a_l b_l} = \prod_{l'=L-1}^{l} \left[\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)_{.b_{l'}}^{b_{l'+1}\cdot}\right] \eta_{a_L b_L}$$

  - Equation 4.14 on frame bundle transformations:

$$\frac{\partial E}{\partial W^{(l-1)}} = \left(\frac{\partial E}{\partial x^{(L)}}\right)_{a_L} \prod_{l'=L-1}^{l} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \left(\frac{\partial x^{(l)}}{\partial W^{(l-1)}}\right)^{a_l}$$

  The metric tensor is rank $(0, 2)$, and so the Lie group action acts doubly on this associated bundle, from the left. This is compared to the frame bundle in which

the Lie group action acts from the right.

- These effects are perhaps more clearly understood in the abstract setting of Equation 4.11 and Equation 4.15.

  - Equation 4.11 on the associated metric bundle transforms:

$$(h_0 \rhd h_1 \rhd ... \rhd h_L \rhd g)_{a_0 b_0} =$$
$$(h_0 \bullet h_1 \bullet ... \bullet h_L) \rhd g_{a_L b_L} = \prod_{l'=L-1}^{0} \left(h^{a_{l'+1} \cdot}_{\cdot a'_l} h^{b_{l'+1} \cdot}_{\cdot b_{l'}} \right) g_{a_L b_L}$$

  - Equation 4.15 on the principal frame bundle transforms:

$$(e \lhd h_L \lhd h_{L-1} \lhd ... \lhd h_0)_{a_0} =$$
$$e \lhd (h_L \bullet h_{L-1} \bullet ... \bullet h_0)_{a_0} = \prod_{l'=L-1}^{0} h^{a_{l'+1} \cdot}_{\cdot a'_l} e_{a_L}$$

In the abstract setting it is immediately seen that the left action $\rhd$: $G \times F \to F$ on the associated bundle defined by $\rhd$: $(h, f) \mapsto h \rhd f$, induced by the right action $\lhd$: $P \times G \to P$ on the principal bundle, defined in terms of the left action by $\lhd$: $(p, h) \mapsto p \lhd h := h^{-1} \rhd p$, is how metric tensor and frame bundle (error gradient) transformations are related; for further details see Chapter 3.

4.4 Limiting scenarios of C^k network architectures

The analysis so far has been quite general, for feedforward networks meeting very weak assumptions. One can impose structure on the network architectures by defining entire classes of coordinate transformations. The following formulation also has the form of differentiable curves/trajectories, but because the number of dimensions often changes as one moves through the network, it is difficult to interpret a trajectory traveling through a space of changing dimensions. A standard feedforward neural network is a $\mathcal{C}^0$ function:

$$x^{(l+1)} := f(x^{(l)}; l) \tag{4.16}$$

A residual network has the form $x^{(l+1)} = x^{(l)} + f(x^{(l)}; l)$. However, because we will eventually take the limit as $L \to \infty$ and $l \in [0, 1] \subset \mathbb{R}$, as opposed to l being only a finitely countable index, the equivalent form of the residual network is as follows:

$$x^{(l+1)} \simeq x^{(l)} + f(x^{(l)}; l)\Delta l \tag{4.17}$$

For a uniform partition of the interval $[0, 1]$, one has $\Delta l = 1/L$ and is implicit in the weight matrix. Being implicit in the weight matrix means that pulling it out as an argument of the nonlinear activation function means that it is a nonlinear parameterization.

Remark. (Differential coordinate transformations) Several peculiarities naturally arise from this problem formulation. Written in this form the residual network in Equation 4.17 hardly seems to be a unique network architecture. For example, instead of taking forward differencing in the definition, in principle one could take central differencing as a higher order finite difference approximation to smoothness over layers. One could also impose smoothness to the curves second and higher derivatives. These smooth differential equations are governing how the coordinates are transforming.

Remark. (Differential group actions) As a corollary to defining smooth differential equations which are governing the coordinate transformations; this will effect the discrete backpropagation algorithm, which is defined on a graph, and in the limiting scenario extend it to allowing smoothly applied group actions on the fibres of the principal and associated bundles.

As mentioned in the above remark, one can define entire classes of coordinate transformations inspired by finite difference approximations of differential equations. These

finite difference equations can be used to construct neural networks that impose k^{th} order differentiable smoothness, as well as varying order finite difference approximations to these equations; three examples of which are given below:

$$\delta x^{(l)} := x^{(l+1)} - x^{(l)} \simeq f(x^{(l)}; l)\Delta l \tag{4.18}$$

$$\delta x^{(l)} := x^{(l+1)} - x^{(l-1)} \simeq f(x^{(l)}; l)2\Delta l \tag{4.19}$$

$$\delta^2 x^{(l)} := x^{(l+1)} - 2x^{(l)} + x^{(l-1)} \simeq f(x^{(l)}; l)\Delta l^2 \tag{4.20}$$

Each of these define a differential equation, but of different order smoothness on the coordinate transformations. Written in this form the residual network in Equation 4.18 is a first-order forward difference approximation to a $\mathcal{C}^1$ coordinate transformation and has $\mathcal{O}(\Delta l)$ error. Network architectures with higher order accuracies can be constructed, such as central differencing approximations of a $\mathcal{C}^1$ coordinate transformation, as in Equation 4.19, to give $\mathcal{O}(\Delta l^2)$ error.

Note that the architecture of a standard feedforward neural network is a static equation, while the others are dynamic. Also note that Equation 4.20 can be rewritten $x^{(l+1)} = x^{(l)} + f(x^{(l)}; l)\Delta l^2 + \delta x^{(l-1)}$, where $\delta x^{(l-1)} = x^{(l)} - x^{(l-1)}$, and in this form one sees that this is a residual network with an extra term $\delta x^{(l-1)}$ acting as a sort of momentum term on the coordinate transformations.

The above finite difference equations can be solved by summing over the domain, as this is a telescoping series. This is analogous to integrating over the continuous velocity of a particle. The solution to the difference equation 4.18 is analogous to the trajectory of a particle moving forward through the layers of the neural network. The discrete trajectory is indexed by layers $l \in \{0, 1, 2, ..., L\}$.

$$x^{(l)} \simeq x^{(0)} + \sum_{l'=0}^{l-1} f(x^{(l')}; l')\Delta l \tag{4.21}$$

Here $x^{(0)}$ is the input data/initial condition and $x^{(L)}$ is the output target prediction. This is the finite difference solution to the finite difference Equation 4.18, and as $\Delta l \to 0$ the finite difference approximation becomes a smooth curve. Given an input data as an initial condition, a trained residual network acts as a differential equation mapping the point along a trajectory to its label. In [55], it is shown that a scalar scaling factor used to break the identity shortcut can lead to exponential signal forward/back propagation.

This is affirmed in Equation 4.18, as including a scalar factor $(1-k)$ in front of the identity term will yield an exponential term of rate k in its solution Equation 4.21.

By the definitions of the $\mathcal{C}^k$ networks given by Equations 4.18-4.20, the right hand side is both continuous and independent of Δl (after dividing), and so the limit exists as $\Delta l \to 0$. Convergence rates and error bounds of finite difference approximations can be applied to these equations. By the standard definition of the derivative, the residual network defines a system of differentiable transformations.

$$\frac{dx^{(l)}}{dl} := \lim_{\Delta l \to 0} \frac{x^{(l+\Delta l)} - x^{(l)}}{\Delta l} = f(x^{(l)}; l) \tag{4.22}$$

$$\frac{dx^{(l)}}{dl} := \lim_{\Delta l \to 0} \frac{x^{(l+\Delta l)} - x^{(l-\Delta l)}}{2\Delta l} = f(x^{(l)}; l) \tag{4.23}$$

$$\frac{d^2x^{(l)}}{dl^2} := \lim_{\Delta l \to 0} \frac{x^{(l+\Delta l)} - 2x^{(l)} + x^{(l-\Delta l)}}{\Delta l^2} = f(x^{(l)}; l) \tag{4.24}$$

Notations are slightly changed, by taking $l = n\Delta l$ for $n \in \{0, 1, 2, .., L-1\}$ and indexing the layers by the fractional index l instead of the integer index n. This defines a *learned* partitioning:

$$\mathcal{P} = \{0 = l\,(0) < l\,(1) < l\,(2) < ... < l\,(n) < ... < l\,(L) = 1\} \tag{4.25}$$

where $\Delta l(n) := l(n+1) - l(n)$ can in general vary with n as the $\max_n \Delta l(n)$ still goes to zero as $L \to \infty$. To reduce notational complexities, this paper will write $\Delta l := \Delta l\,(n)$ for all $n \in \{0, 1, 2, ..., L-1\}$.

Remark. (Inhomogeneous and homogeneous differential transformations) In [82], a deep residual convolution network was trained on ImageNet in the usual fashion except parameter weights between residual blocks at the same dimension were shared, thus reducing the number of parameters by 39%, at a cost to the accuracy of only 0.2%. This is the difference between learning an inhomogeneous first order equation $\frac{dx^{(l)}}{dl} := f(x^{(l)}; l)$ and a (piecewise) homogeneous first order equation $\frac{dx^{(l)}}{dl} := f(x^{(l)})$.

For the boundary value problem with two fixed endpoints, such as $x^{(0)}$ as the input and $x^{(1)}$ as the output, equations of variational form for the action [83, 84] can be constructed:

$$S\left[x^{(l)}\right] := \int_0^1 \mathcal{L}\left[x^{(l)}, \dot{x}^{(l)}, l\right] dl \tag{4.26}$$

This can be solved by minimizing the action, when $\delta S\left[x^{(l)}\right]=0$, which yields the classical Euler-Lagrange equation.

Remark. (Classical action on a neural network) The usual formulation of neural network cost minimizations ignores the trajectory of the particle as it moves through the intermediary layers. Some networks, e.g. GoogLeNet [18], use intermediary layer classifications that alter the intermediary trajectories, although in that framework it is not a condition on the trajectory as opposed to on the predicted error.

Within this formulation, one may be interested in minimizing the distance traveled, in addition to the error between the output and target. It should be noted that it doesn't make sense to minimize the distance the particle travels if the particle is not traveling to the target location.

If the network is taken to be residual as in Equation 4.17, then the Jacobian of the coordinate transformation is found, with $\delta^{a_{l+1}\cdot}_{.a_l}$ the Kronecker delta:

$$\left(\frac{\partial x^{(l+1)}}{\partial x^{(l)}}\right)^{a_{l+1}\cdot}_{.a_l}=\delta^{a_{l+1}\cdot}_{.a_l}+\left(\frac{\partial f\left(x^{(l)};l\right)}{\partial x^{(l)}}\right)^{a_{l+1}\cdot}_{.a_l}\Delta l \tag{4.27}$$

Backpropagating the coordinate representation of the metric tensor requires the sequence of matrix products from output to input, and can be defined for any layer l:

$$P^{a_L\cdot}_{.a_l}:=\prod_{l'=L-1}^{l}\left[\delta^{a_{l'+1}\cdot}_{.a_{l'}}+\left(\frac{\partial f(z^{(l'+1)};l')}{\partial z^{(l'+1)}}\right)^{a_{l'+1}\cdot}_{.e_{l'+1}}\left(\frac{\partial z^{(l'+1)}}{\partial x^{(l')}}\right)^{e_{l'+1}\cdot}_{.a_{l'}}\Delta l\right] \tag{4.28}$$

where $z^{(l+1)}:=W^{(l)}x^{(l)}+b^{(l)}$. With this, taking the output metric to be the standard Euclidean metric η_{ab}, the linear element can be represented in the coordinate space for any layer l:

$$ds^2=P^{a\cdot}_{.a_l}P^{b\cdot}_{.b_l}\eta_{ab}dx^{a_l}dx^{b_l} \tag{4.29}$$

As $L\to\infty$, Equation 4.28 becomes an infinite product of matricies (from our infinite applications of the chain rule). Analogous to the scalar case, $P^{a_L\cdot}_{.a_0}=\prod_{l'=0}^{L-1}(\delta^{a_{l'+1}\cdot}_{.a_{l'}}+A^{a_{l'+1}\cdot}_{.a_{l'}})$ converges if $\sum_{l'=0}^{L-1}||A^{a_{l'+1}\cdot}_{.a_{l'}}||_2$ converges [85]. For example, for a fully connected network with activation $\tanh(z)$, the following inequality holds:

$$\sum_{l'=0}^{L-1} ||A_{.a_{l'}}^{a_{l'+1}\cdot}||_2 = \sum_{l'=0}^{L-1} || \left(\frac{\partial f(z^{(l'+1)}; l')}{\partial z^{(l'+1)}} \right)_{.e_{l'+1}}^{a_{l'+1}\cdot} \left(\frac{\partial z^{(l'+1)}}{\partial x^{(l')}} \right)_{.a_{l'}}^{e_{l'+1}\cdot} \Delta l||_2 \leq$$
$$\sum_{l'=0}^{L-1} 2 \cdot ||W_{.a_{l'}}^{e_{l'+1}\cdot}||_2 \Delta l = 2 \cdot E\left[||W_{.a_{l'}}^{e_{l'+1}\cdot}||_2 \right] < \infty \quad (4.30)$$

where $||.||_2$ is the ℓ^2 norm and $E[.]$ is the expectation. This shows that the infinite sum converges, implying that in the limit Equation 4.28 converges.

In the limit the actions of the coordinate transformations on the metric tensor smoothly transform the metric tensor coordinate representation.

This is interesting from a theoretical perspective as well because these are differential equations governing group actions smoothly along the principle and associated fibres. This is similar in spirit to finding the parallel transport along a curve on a manifold, where the horizontal lift of the curve is generated by starting from an arbitrary curve $\delta : [0, 1] \to P$ defined by $\delta : \lambda \mapsto \delta(\lambda)$, and using a smooth group action $g : \lambda \mapsto g(\lambda)$ to act on δ to produce the lifted curve $\gamma(\lambda) = \delta(\lambda) \triangleleft g(\lambda)$. The suitable curve $g : \lambda \mapsto g(\lambda)$ is the solution to an ordinary differential equation with initial condition $g(0) = g_0$, where g_0 is the unique group element for which $\delta(0) \triangleleft g_0 = p \in P$. The ordinary differential equation for $g : \lambda \mapsto g(\lambda)$ is solved by a path-ordered integral over the local Yang-Mills field [80].

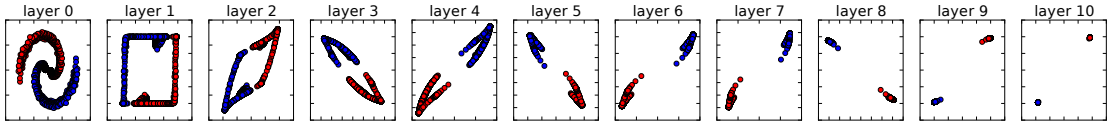
4.4.1 Discussion

This subsection will discuss several numerical experiments and their results studying the $\mathcal{C}^1$ neural network separation process. These experiments all used the hyperbolic tangent nonlinearity for the activation function.

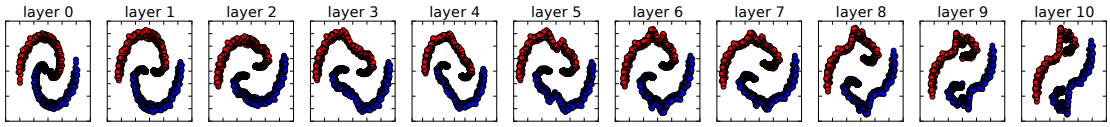
4.4.1.1 Neural networks with C^k differentiable coordinate transformations

As described earlier, k^{th} order smoothness can be imposed on the network by considering network structures defined by e.g. Equations 4.18-4.20. As seen in Figure 4.6a, the standard $\mathcal{C}^0$ network with no impositions on differentiability has very sharp layerwise transformations and separates the data in an unintuitive way.

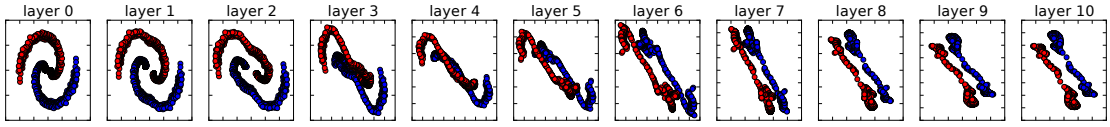
The $\mathcal{C}^1$ (residual) and $\mathcal{C}^2$ networks can be seen in Figures 4.6b and 4.6c. Both exhibit smooth layerwise transformations and separate the data in a more intuitive way. Forward differencing is used for the $\mathcal{C}^1$ network, while central differencing was used for



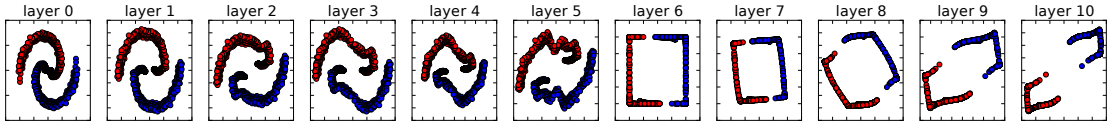
(a) A $\mathcal{C}^0$ network with sharply changing layer-wise particle trajectories.



(b) A $\mathcal{C}^1$ network with smooth layer-wise particle trajectories.



(c) A $\mathcal{C}^2$ network also exhibits smooth layer-wise particle trajectories.

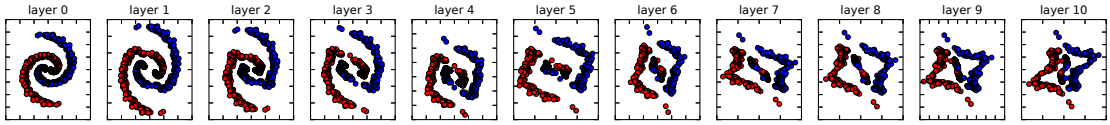


(d) A combination $\mathcal{C}^0$ and $\mathcal{C}^1$ network, where the identity connection is left out in layer 6.

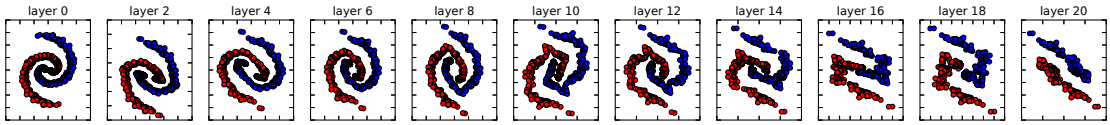
Figure 4.6: Untangling the same spiral with 2-dimensional neural networks with different constraints on smoothness. The x and y axes are the two nodes of the neural network at a given layer l, where layer 0 is the input data. The $\mathcal{C}^0$ network is a standard network, while the $\mathcal{C}^1$ network is a residual network and the $\mathcal{C}^2$ network also exhibits smooth layerwise transformations. All networks achieve 0.0% error rates. The momentum term in the $\mathcal{C}^2$ network allows the red and blue sets to pass over each other in layers 3, 4 and 5. Figure 4.6d has the identity connection for all layers other than layer 6.

the $\mathcal{C}^2$ network, except at the output layer where backward differencing was used, and at the input first order smoothness was used as forward differencing violates causality.

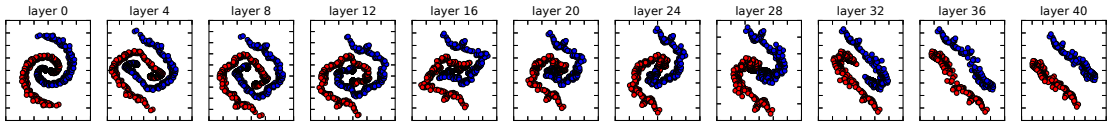
The second derivative is $x^a(l+1) - 2x^a(l) + x^a(l-1) = f^a(x(l); l)\Delta l^2$, which can be rewritten $x^a(l+1) = x^a(l) + f^a(x(l); l)\Delta l^2 + \delta x^a(l-1)$, where $\delta x^a(l-1) = x^a(l) - x^a(l-1)$. In this form one sees that this is a residual network with an extra term $\delta x^a(l-1)$ acting as a sort of momentum term on the coordinate transformations. It is seen in Figure 4.6c that the momentum term in the $\mathcal{C}^2$ network allows the red and blue sets to pass over each other in layers 3, 4 and 5.



(a) A 10 layer $\mathcal{C}^1$ network struggles to separate the spirals and has 1% error rate.



(b) A 20 layer $\mathcal{C}^1$ network is able to separate the spirals and has 0% error rate.



(c) A 40 layer $\mathcal{C}^1$ network is able to separate the spirals and has 0% error rate.

Figure 4.7: The effect of number of layers on the separation process of a $\mathcal{C}^1$ neural network. In Figure 4.7a it is seen that the Δl is too large to properly separate the data. In Figures 4.7b and 4.7c the Δl is sufficiently small to separate the data. Interestingly, the separation process is not as simple as merely doubling the parameterization and halving the partitioning in Equation 4.25 because this is a nonlinear system of ODE's. This is seen in Figures 4.7b and 4.7c; the data are at different levels of separation at the same position of layer parameterization, for example by comparing layer 18 in Figure 4.7b to layer 36 in Figure 4.7c.

4.4.1.2 Effect of number of layers on the separation process

This experiment compares the process in which 2-dimensional $\mathcal{C}^1$ networks with 10, 20 and 40 layers separate the same data, thus experimenting on the Δl in the partitioning of Equation 4.25, as seen in Figure 4.7. The 10 layer network is unable to properly separate the data and achieves a 1% error rate, whereas the 20 and 40 layer networks both achieve 0% error rates. In Figures 4.7b and 4.7c it is seen that at same positions of layer parameterization, for example layers 18 and 36 respectively, the data are at different levels of separation. This implies that the partitioning cannot be interpreted as simply as halving the Δl when doubling the number of layers. This is because the system of ODE's are nonlinear and the Δl is implicit in the weight matrix.

4.5 Perturbation theory on neural networks

This section constructs a model of neural networks using perturbation theory to differentiate between the training and testing phases. The objective of this work is to attempt to understand why neural networks can learn to properly map training data while improperly mapping test data. Assuming that the training data well samples the data manifold, then any testing data point $x^{(0)}$ should be close to some training data point $\bar{x}^{(0)}$, where the closeness is modeled as a perturbation $x^{(0)} = \bar{x}^{(0)} + \delta x^{(0)}$

The inspiration for such a model comes from Locally Linear Embedding [22], where data points are approximated by their nearest neighbors. In this case we model test points by its nearest neighbor with a perturbed error.

A feedforward neural network has the form $x^{(l+1)} = \varphi^{(l)}\left(x^{(l)}\right)$. Assuming that the neural network has been properly trained at mapping input data to output labels, the goal is for the network to be able to map unseen test data to the proper output label. This study develops theoretical bounds on applying perturbations $\delta x^{(0)}$ to trained initial conditions $\bar{x}^{(0)}$, i.e. $x^{(0)} = \bar{x}^{(0)} + \delta x^{(0)}$.

Theorem 4.5.1. A neural network with layerwise transformations given by $x^{(l+1)} = \varphi^{(l)}\left(x^{(l)}\right)$ and input data point for testing $x^{(0)} = \bar{x}^{(0)} + \delta x^{(0)}$, which is perturbed from initial condition for training $\bar{x}^{(0)}$, gets mapped to the following perturbed trajectory, where $\bar{x}^{(l+1)} := \varphi^{(l)}\left(\bar{x}^{(l)}\right)$ is the trajectory determined on the trained data:

$$x^{(l+1)} \approx \bar{x}^{(l+1)} + \prod_{l'=0}^{l} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \delta x^{(0)} \tag{4.31}$$

Proof. We will prove this by induction. First the base case for $l' = 0$, from the initial condition $x^{(0)} = \bar{x}^{(0)} + \delta x^{(0)}$:

$$x^{(1)} = \varphi^{(0)}\left(x^{(0)}\right) = \varphi^{(0)}\left(\bar{x}^{(0)} + \delta x^{(0)}\right) \approx \varphi^{(0)}\left(\bar{x}^{(0)}\right) + \frac{\partial \varphi^{(0)}}{\partial x^{(0)}}\left(\bar{x}^{(0)}\right) \delta x^{(0)} \tag{4.32}$$

Rewriting this, with $\bar{x}^{(1)} := \varphi^{(0)}\left(\bar{x}^{(0)}\right)$ and $\delta x^{(1)} := \frac{\partial \varphi^{(0)}}{\partial x^{(0)}}\left(\bar{x}^{(0)}\right) \delta x^{(0)}$, the forward transformation becomes the following:

$$x^{(1)} \approx \bar{x}^{(1)} + \delta x^{(1)} \tag{4.33}$$

Now assume the case for $l' = l - 1$. This means that with $\bar{x}^{(l)} := \varphi^{(l-1)}\left(\bar{x}^{(l-1)}\right)$ and $\delta x^{(l)} := \frac{\partial \varphi^{(l-1)}}{\partial x^{(l-1)}}\left(\bar{x}^{(l-1)}\right)\delta x^{(l-1)} = \prod_{l'=0}^{l} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(0)}\right) \cdot \delta x^{(0)}$, the forward transformation becomes the following:

$$x^{(l)} \approx \bar{x}^{(l)} + \prod_{l'=0}^{l} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \cdot \delta x^{(0)} \tag{4.34}$$

We now prove the induction step for $l' = l$. Then

$$\begin{aligned} x^{(l+1)} \approx \varphi^{(l)}\left(x^{(l)}\right) &= \varphi^{(l)}\left(\bar{x}^{(l)} + \delta x^{(l)}\right) \\ &\approx \varphi^{(l)}\left(\bar{x}^{(l)}\right) + \frac{\partial \varphi^{(l)}}{\partial x^{(l)}}\left(\bar{x}^{(l)}\right) \cdot \delta x^{(l)} \\ &= \varphi^{(l)}\left(\bar{x}^{(l)}\right) + \frac{\partial \varphi^{(l)}}{\partial x^{(l)}}\left(\bar{x}^{(l)}\right) \cdot \prod_{l'=0}^{l-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \cdot \delta x^{(0)} \\ &= \varphi^{(l)}\left(\bar{x}^{(l)}\right) + \prod_{l'=0}^{l} \frac{\partial \varphi^{(l'-1)}}{\partial x^{(l'-1)}}\left(\bar{x}^{(0)}\right) \cdot \delta x^{(0)} \end{aligned} \tag{4.35}$$

Using the fact that $\bar{x}^{(l+1)} = \varphi^{(l)}\left(\bar{x}^{(l)}\right)$, we have the result:

$$x^{(l+1)} \approx \bar{x}^{(l+1)} + \prod_{l'=0}^{l} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \cdot \delta x^{(0)} \tag{4.36}$$

□

The network is trained to map an input data point $\bar{x}^{(0)}$ to its label $\bar{x}^{(L)} = y$. To see where in the output space the network maps a new unseen data point $x^{(0)} = \bar{x}^{(0)} + \delta x^{(0)}$ in terms of the original data point $\bar{x}^{(0)}$ and its perturbation $\delta x^{(0)}$, Theorem 4.5.1 can be used for the output layer L to find the test point prediction $x^{(L)}$.

$$x^{(L)} \approx \bar{x}^{(L)} + \prod_{l'=0}^{L-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \cdot \delta x^{(0)} \tag{4.37}$$

By using the definition $\delta x^{(L)} := x^{(L)} - \bar{x}^{(L)}$ this is rewritten as follows:

$$\delta x^{(L)} \approx \prod_{l'=0}^{L-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}}\left(\bar{x}^{(l')}\right) \cdot \delta x^{(0)} \tag{4.38}$$

Using the assumption that the neural network was trained properly at mapping inputs $\bar{x}^{(0)}$ to target y, then we have $y = \bar{x}^{(L)}$. This means that Equation 4.38 tells us how far away the neural network was from mapping the test data from where it mapped its neighbor training data.

We clearly see that the neural network transforms perturbations $\delta x^{(0)}$ from training samples $\bar{x}^{(0)}$ by pushing it forward with sequential group actions from the Jacobian matricies $\prod_{l'=0}^{L-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}} \left(\bar{x}^{(l')} \right)$. By taking the norm, we can see rates of convergence:

$$\left\| \delta x^{(L)} \right\|_2 = \left\| \prod_{l'=0}^{L-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}} \left(\bar{x}^{(l')} \right) \cdot \delta x^{(0)} \right\|_2 \tag{4.39}$$

This result can be interpreted in several ways. First it shows that, as the size of the perturbation gets smaller the output prediction size decreases according to a rate determined by the neural network, as expected from continuous functions mapping open sets to open sets. This makes sense because in the limit the data manifold is completely sampled, and the network should have learned to map all points to their labels. Second, it shows that the rate changes in a very non-obvious as the number of layers changes, and the type of neural network changes; it changes depending on the Jacobian group action, which is a subset of the general linear group $GL\left(d, \mathbb{R}\right)$.

This result is also equivalent to backropagating the metric tensor through the network. This can be seen by the following:

$$\begin{aligned}
ds^2 =& \eta_{a_L b_L} dx^{a_L} dx^{b_L} \\
=& \left\| \delta x^{(L)} \right\|_2^2 = \left\| \prod_{l'=0}^{L-1} \frac{\partial \varphi^{(l')}}{\partial x^{(l')}} \left(\bar{x}^{(l')} \right) \cdot \delta x^{(0)} \right\|_2^2 \\
=& \prod_{l'=0}^{L-1} \left[\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)_{\cdot a_{l'}}^{a_{l'+1} \cdot} dx^{a_0} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)_{\cdot b_{l'}}^{b_{l'+1} \cdot} dx^{b_0} \right] \eta_{a_L b_L} \\
=& \prod_{l'=0}^{L-1} \left[\left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)_{\cdot a_{l'}}^{a_{l'+1} \cdot} \left(\frac{\partial x^{(l'+1)}}{\partial x^{(l')}} \right)_{\cdot b_{l'}}^{b_{l'+1} \cdot} \right] \eta_{a_L b_L} dx^{a_0} dx^{b_0} \\
=& g(x^{(0)})_{a_0 b_0} dx^{a_0} dx^{b_0}
\end{aligned}$$

In both cases where the ℓ_2-norm $\left\| \cdot \right\|_2$ is used, this is because the Euclidean metric $\eta_{a_L b_L}$ is used in both equalities.

4.5.1 The special case of a residual network

The analysis so far has been quite general. If the network architecture is residual, then $\varphi^{(l)} : x^{(l)} \mapsto x^{(l)} + f\left(x^{(l)}; l\right)$ for some nonlinear activation function $f : \mathbb{R}^d \to \mathbb{R}^d$, and so the transformation is $x^{(l+1)} = x^{(l)} + f\left(x^{(l)}; l\right)$. Noticing however that the first order Taylor expansion of $x^{(l+1)} = x^{(l)} + \dot{x}^{(l)}\Delta l$, we define the residual network as follows:

$$x^{(l+1)} = x^{(l)} + f(x^{(l)}; l)\Delta l \tag{4.40}$$

for some learned partition $\mathcal{P} = \{0 = l(0) < l(1) < ... < l(n) < ... < l(L) = 1\}$, where in general $\Delta l(n) = l(n+1) - l(n)$ is allowed to vary with n so long as $\max_n \Delta l(n)$ goes to zero as L goes to infinity.

The trajectory of a particle with initial condition $x^{(0)}$ is solved by summing over the telescoping series $x^{(l+1)} - x^{(l)} = f(x^{(l)}; l)\Delta l$ (which in the limit becomes integration):

$$x^{(l)} = x^{(0)} + \sum_{l'=0}^{l-1} f(x^{(l')}; l')\Delta l \tag{4.41}$$

The Jacobian of the Equation 4.40 transformation is found:

$$\left(\frac{\partial x^{(l+1)}}{\partial x^{(l)}}\right)_{.a_l}^{a_{l+1}\cdot} = \delta_{.a_l}^{a_{l+1}\cdot} + \left(\frac{\partial f\left(x^{(l)}; l\right)}{\partial x^{(l)}}\right)_{.a_l}^{a_{l+1}\cdot} \Delta l \tag{4.42}$$

where $\delta_{.a_l}^{a_{l+1}\cdot}$ is the Kronecker-delta. In this case we have

$$\delta x^{a_L} = \prod_{l'=0}^{L-1} \left(\delta_{.a_{l'}}^{a_{l'+1}\cdot} + \left(\frac{\partial f\left(x^{(l')}; l'\right)}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \Delta l\right) \delta x^{a_0} \tag{4.43}$$

This converges because $P_{.a_0}^{a_L\cdot} = \prod_{l'=0}^{L-1}(\delta_{.a_{l'}}^{a_{l'+1}\cdot} + A_{.a_{l'}}^{a_{l'+1}\cdot})$ converges if $\sum_{l'=0}^{L-1} ||A_{.a_{l'}}^{a_{l'+1}\cdot}||_2$ converges [85].

The reverse problem is interesting as well, that is which initial conditions get mapped to the similar outputs? If a matrix A is such that $\lim_{n\to\infty}(I - A)^n = 0$ then A is nonsingular and its inverse may be expressed by a Neumann series $A^{-1} = \sum_{n=0}^{\infty}(I - A)^n$. We have that $\lim_{n\to\infty}(I - A)^n = 0$ because the Kronecker-delta is the identity and the Δl is small. Keeping only first order terms, we have the following approximation of the inverse:

$$\left(\delta_{.a_{l'}}^{a_{l'+1}\cdot} + \left(\frac{\partial f\left(x^{(l')};l'\right)}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \Delta l\right)^{-1}$$
$$\simeq \left(\delta_{.a_{l'}}^{a_{l'+1}\cdot} - \left(\frac{\partial f\left(x^{(l')};l'\right)}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \Delta l\right) \quad (4.44)$$

One could also come to this result by invoking the inverse function theorem.

With $(P_0 P_1 ... P_L)^{-1} = P_L^{-1} ... P_1^{-1} P_0^{-1}$ we can rewrite Equation 4.43 as follows:

$$\prod_{l'=L-1}^{0} \left(\delta_{.a_{l'}}^{a_{l'+1}\cdot} - \left(\frac{\partial f\left(x^{(l')};l'\right)}{\partial x^{(l')}}\right)_{.a_{l'}}^{a_{l'+1}\cdot} \Delta l\right) \delta x^{a_L} \simeq \delta x^{a_0} \quad (4.45)$$

This result, Equation 4.45, tells us the regions in the input space that get mapped to specified regions in the output space/labels via the coordinate transformation learned by the neural network $h_\Theta : X \to Y$.

From differential geometry, if $p \in X$ and $U \subset X$ is an open ball of the input space containing p defined by $U = \{x \in X : \|x - p\|_{g(0)} < \delta x\}$, with coordinate system $x(0)$ and metric $g(0)$, then Equation 4.43 finds the representation of U in the output coordinate system $x(L)$, namely $h_\Theta(U) = \{h_\Theta(x) \in Y : \|x - p\|_{g(0)} < \delta x\}$. The notation $\|.\|_{g(l)}$ denotes the norm induced by the metric at layer l.

Equation 4.45 however goes the reverse direction, if $q \in Y$ and $V \subset Y$ is an open ball of the output space containing q defined by $V = \{y \in Y : \|y - q\|_{g(L)} < \delta y\}$, with coordinate system $x(L)$ and metric $g(L)$, then Equation 4.45 finds the representation of V in the input coordinate system $x(0)$, namely $h_\Theta^{-1}(V) = \{x \in X : \|h_\Theta(x) - q\|_{g(L)} < \delta y\}$.

Note that distance is an invariant to coordinate representation, thus $\|x - p\|_{g(0)} = \|h_\Theta(x) - q\|_{g(L)}$ where $q = h_\Theta(p)$. We are merely defining open sets in one metric space and coordinate system and representing it in another metric space and coordinate system, which is all a neural network is doing. The power of the reverse direction, Equation 4.45, is that we can define similarity on the abstract level at the output space, yet see what this means on the physical level in the input space.

Chapter 5
Probabilistic forecasting

This chapter reviews a developed algorithmic framework for probabilistically forecasting symbol sequences with neural networks.

The probabilistic forecasting problem is formulated in a classification sense using deep neural networks [86] as well as Long Short-Term Memory neural networks [87] such that the desired output is a probability over symbol sequences, with symbols partitioning the range space and each symbol corresponding to a class. This is significantly different from a regression of the mean and its associated quantiles.

Performing probabilistic forecasting in this way has significant advantages over standard probabilistic forecasting techniques, such as ARMA and its derivative models. For example, ARMA assumes second order statistics, where as these developed models assume no distribution of the output space. Second, ARMA is a linear model and assumes linear relationships of the data, where as these developed algorithms can model nonlinear relationships of the data since they are developed from neural networks.

5.1 Probabilistic Forecasting with Deep Networks

5.1.0.1 Mathematical preliminaries

This section details the key elements of the proposed algorithm for probabilistic time series forecasting. It also briefly describes the autoregressive moving average (ARMA) model, which is used as a baseline to compare the results.

5.1.1 Time series symbolization

This section introduces the symbolization procedure, sometimes referred to as coarse graining, of the time series.

The symbolization procedure of time series data is found by constructing a partitioning of the time series feature space. The partition defines mutually exclusive and exhaustive regions over the feature space. Once a partitioning is defined over the feature space, the data is symbolized by mapping each data point to a corresponding symbol which is uniquely identified with that partition region.

For one-dimensional time series, two very common partitioning schemes are the uniform partitioning and the maximum entropy partitioning [88]. Uniform partitioning divides the space into equally sized disjoint regions spanning the space, where as maximum entropy partitioning divides the space such that an equal number of data points belong to each region. For general multi-dimensional time series, k-means clustering [89], Gaussian mixture models or other unsupervised clustering techniques can be used.

5.1.2 Neural networks

This subsection very briefly introduces the key machinery in constructing and training deep neural networks. Many outstanding review material already exists detailing training procedures, for example [66] [67], so only a brief introduction is given here. First we review the Restricted Boltzmann Machine, which acts as the building block for a deep belief network, which is used to initialize the parameters for a deep neural network.

5.1.2.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a bipartite, energy-based probabilistic graphical model [66]. It is used as an unsupervised mechanism to construct a deep belief network, which is itself used to initialize the parameters for a deep neural network [14, 59]. It is difficult to train a deep neural network without initializing the parameters in a region close to optimal, as using backpropagation in the gradient descent search only significantly changes parameters in the higher layers and leaves the lower layer parameters relatively unchanged.

A RBM defines the following distribution:

$$P(x, h) := \frac{e^{-E(x,h)}}{Z} \tag{5.1}$$

where x and h are the observed and hidden variables of the network, $E(x, h)$ is the energy associated to the (x, h) pair, and Z is the partition function. For $x, h \in \{0, 1\}$, then the energy function takes the bilinear, affine form:

$$E(x, h) := -b'x - c'h + h'Wx \tag{5.2}$$

for visible bias b, hidden bias c and weight matrix W. This allows one to factorize the conditional distributions as follows:

$$P(h_i|x) = \frac{e^{c_i + W_{.i}x}}{1 + e^{c_i + W_{.i}x}} \tag{5.3}$$

$$P(x_j|h) = \frac{e^{b_j + W'_{.j}h}}{1 + e^{c_i + W'_{.j}x}} \tag{5.4}$$

The RBM is trained via a special case of Gibbs sampling known as Contrastive Divergence [9], which seeks to find the parameters that find the maximum of the log-likelihood of Equation 5.1, although in the form of free energy from statistical thermodynamics, where the free energy is defined as follows:

$$F(x) := -\log \sum_h e^{-E(x,h)} \tag{5.5}$$

Contrastive Divergence is an approximation of the log-likelihood gradient, and yields the well known update rules:

$$-\Delta w_{ij} :=< h_i x_j >_{mean} - < h_i x_j >_{recon} \tag{5.6}$$

$$-\Delta c_i :=< h_i >_{mean} - < h_i >_{recon} \tag{5.7}$$

$$-\Delta b_i :=< x_i >_{mean} - < x_i >_{recon} \tag{5.8}$$

5.1.2.2 Deep neural networks

A deep neural network is a function $h_\Theta : X \mapsto Y$, and the type of deep neural network used here has the form of nested compostions of a single layer neural network with a

softmax classification layer on top. The softmax classification layer is defined as follows:

$$P(Y = y_i|z) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{5.9}$$

Here Θ is defined to be the parameter set $\Theta := \{(W^{(l)}, b^{(l)})\}_{l=0}^{L-1}$ for layers $l \in \{0, 1, 2, ...L-1\}$. A single layer of a neural network is of the form:

$$z^{(l+1)} := W^{(l)} \cdot x^{(l)} + b^{(l)} \tag{5.10}$$

$$x^{(l+1)} := \sigma(z^{(l+1)}) \tag{5.11}$$

where $\sigma(z^{(l+1)}) := 1/(1 + e^{-z^{(l+1)}})$ is the sigmoid nonlinearity function and is applied element-wise to its argument $z^{(l+1)}$, for layer $l \in \{0, 1, 2, ...L-1\}$. Thus it is seen that Equation 5.10 defines an affine transformation of its input $x^{(l)}$, and Equation 5.11 acts as a nonlinear squashing function of its argument $z^{(l+1)}$. $x^{(0)}$ is taken to be the (pre-processed) data input of the neural network, $x^{(L)}$ is taken to be the output prediction, and all layers between are called hidden layers.

Deep networks are used because according to the manifold hypothesis natural data is collected in a much higher dimension than the true underlying manifold in which the data was sampled from. Because the deep neural networks have many nested nonlinearities, the resulting learned function is highly nonlinear and can thus learn highly nonlinear manifolds [37].

To learn the parameter set Θ of the neural network, first the weights are initialized in an unsupervised way via contrastive divergence, as described in section 5.1.2.1. After that, the parameter set is fine-tuned by performing a gradient descent search to minimize a cross-entropy cost functional:

$$l(y, h_\Theta(x^{(0)})) := \sum_{i=1}^{K} y_i \log(h_\Theta(x^{(0)})_i) \tag{5.12}$$

Where the function $h_\Theta(x)_i = P(Y = y_i|x)$ is the learned probability of forecasting symbol y_i.

It is very important to note that in a deterministic regression problem for forecasting the cross-entropy cost functional is taken to be $l(y, h_\Theta(x) := h_\Theta(x) \log(y) + (1 - h_\Theta(x)) \log(1 - y)$ [90]. This is not symbolized as y_i and $h_\Theta(x)_i$ and is not probabilistic,

but instead takes the form as a deterministic regression as y. Similarly, the $l_2(\mathbb{R})$-norm cost functional is also used for regression tasks, as opposed to determining probabilities of new classes/symbols.

5.1.3 Autoregressive moving average (ARMA)

The autoregressive moving average model ($ARMA$) is the combination of an autoregressive (AR) and moving average (MA) model [91] [35]. It is used for both understanding a time series as well as forecasting future values of the time series.

The autoregressive model of order p, $AR(p)$, is a regression of the random variable on its past p values:

$$X_t := c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{5.13}$$

where X_t is the time-series value at time t, c is a model constant, $\{\phi_i\}_{i=1}^{p}$ are model parameters and ϵ_t is white noise.

The moving average model of order q, $MA(q)$, is the moving average of the time series on its past q values:

$$X_t := \mu + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{5.14}$$

where $\mu = E\{X_t\}$ is the expected value of the series X_t, $\{\epsilon_{t-j}\}_{j=0}^{q}$ are i.i.d. white noise and $\{\theta_j\}_{j=1}^{q}$ are model parameters.

The autoregressive moving average model of orders (p, q), $ARMA(p, q)$, is the combination of these two models:

$$X_t := c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{5.15}$$

We measure the performance of $ARMA(p, q)$ by how well it can make out-of-sample forecasting predictions given only in-sample information. Thus in order to make future predictions, we iterate predictions forward until we get to the desired forecasting index.

In order to find the model parameters, the loglikelihood maximized was the conditional sum of squares likelihood, and these values were then used to initialize the

parameters for the exact likelihood from the Kalman filter. This was solved via the limited memory Broyden-Fletcher-Goldfarb-Shanno method [91]

5.1.3.1 Developed algorithm

The proposed algorithm for neural probabilistic forecasting of symbol sequences will now be outlined. The goal of the algorithm is, given a time series interval up to the present $\{x(t-n)\}_{n=0}^{p}$ of length $(p+1)$, can we predict the probability of seeing a symbol y_i at time instance $n+T$, i.e. $P(Y = y_i(n+T)|X = \{x(t-n)\}_{n=0}^{p})$.

The data can thus be organized into input-output pairs, as follows, where p, T, N and i are the time series memory length, future forecasting time step, number of data samples and index of the probability mass function, respectively.

$$D := \{(\{x(t-n)\}_{n=0}^{p}, y_i(t+T))\}_{t=1}^{N} \tag{5.16}$$

As described in section 5.2.1.1, a neural network defines a mapping $h_\Theta : X \mapsto Y$, where X and Y are the input and output spaces and $\Theta = \{(W^l, b^l)\}_{l=0}^{L-1}$ is the parameterization. We will use this model to define probabilities as follows:

$$h_\Theta(\{x(t-n)\}_{n=0}^{p})_i = P(Y = y_i(n+t)|X = \{x(t-n)\}_{n=0}^{p}) \tag{5.17}$$

Training the neural network to make these inferences involves minimizing the cumulative loss over Equation 5.29:

$$L(\Theta; D) := \frac{1}{N}\sum_{n=1}^{N} l(y(n), h_\Theta(x(n))) \tag{5.18}$$

The cross entropy between two probability distributions y and $h_\Theta(x)$ over the same underlying sample space can be intuitively understood as the average number of bits required to identify a realization drawn from the sample space if a coding scheme is used that is optimized for an approximated distribution $h_\Theta(x)$, as opposed to the true distribution y [72]. Thus minimizing the cross entropy between the two distributions is a way of minimizing the distance between them.

In practice, an $l_2(\mathbb{R})$ regularization term is included over the network weights to mitigate overfitting/encourage smoothness in the solution.

Equation 5.28 minimizes Equation 5.31, which outputs a probability distribution at each time step forecasting the symbol generation probabilities. This type of soft

prediction is valuable as it gives confidence in predictions. However, if one is interested in a deterministic formulation, by taking the expectation over the posteriors, one can construct an expected trajectory.

$$\hat{y} = E\{\bar{y}\} = \sum_{i=1}^{K} \bar{y}_i P(Y = y_i | X = x) \tag{5.19}$$

where $\bar{y}_i$ is taken to be the centroid of cluster region i.

It should be clear that this trajectory is very different from a trajectory which derives from an $l_2(\mathbb{R})$-norm cost minimization, which by construction is meant to well match the true trajectory. A neural network from an $l_2(\mathbb{R})$-norm cost minimization directly defines a trajectory, where as by minimizing the cross-entropy in the sense of Equation 5.31 and then defining a trajectory in terms of the expectation over the posteriors as in Equation 5.19, one is indirectly constructing a trajectory.

In addition to Equation 5.19, higher order statistics such as the covariance can be generated to understand the forecasted symbol sequence.

5.1.3.2 Results and discussion

This section presents the results of the proposed algorithm for instability prediction of a fuel lean combustion system, and compare these results to the baseline set by the ARMA model.

Combustion and fluid processes are physically modeled by highly coupled, non-linear partial differential equations. The proposed algorithm is instead based on data-driven techniques for instability prediction. Combustion instabilities are commonly defined by the root-mean-square (RMS) of the pressure signal inside the combustion chamber of a trailing window, with the intuition that a larger RMS value means there exists larger pressure fluctuations, and thus larger instabilities. The goal is, given information about the present, can we predict instabilities in the future to possibly be used in some type of control feedback mechanism for instability mitigation.

The RMS space was partitioned into $K = 10$ equal disjoint regions to define the prediction space/symbol alphabet set. The neural network architecture used can be seen in Figure 5.1. It has a fully connected architecture of 10 input neurons (comprised of the current time element and the history of the previous 9 time elements) to 22 hidden neurons to 22 hidden neurons to 10 output neurons to form symbolic predictions. The hidden layers were chosen to be 22 dimensional because any two embeddings of an

n-manifold are isotopic in $\mathbb{R}^{2n+2}$ [92].

The network was pretrained as a deep belief network and fine tuned using stochastic gradient descent of error backpropagation. The cost functional was to minimize the cross-entropy, thus give the predictions a probabilistic interpretation, and included an l_2 regularization on the network weights to prevent overfitting. The dataset was divided into training, validation and test sets, with all three giving very similar results. This is likely due to the large quantity of data used, as well as the regularization on the network weights implemented.

For comparison, $ARMA(10, 2)$ was used to set a baseline, with this data only using the training and testing sets. A GPU implementation of the neural network was made using the Python library Theano [93], and training would take about 10 minutes for the complete pretraining and fine-tuning procedure.

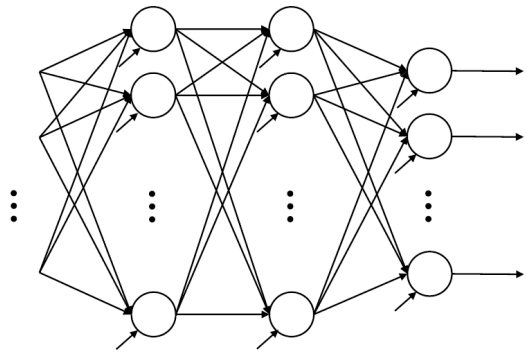


Figure 5.1: The deep neural network architecture used for this study consists of an input layer of size 10, two hidden layers of size 22 and an output layer of size 10. The 10 dimensional input layer corresponds to a signal history of size 10, and the 10 dimensional output layer corresponds to the alphabet size of the symbol set.

A characteristic series of forecasts made by the ARMA model can be seen in Figure 5.2. In this example, the ARMA model is used to predict the RMS values starting at time step 700 and going until time step 764, and 11 of the first 13 forecasted class predictions were correct.

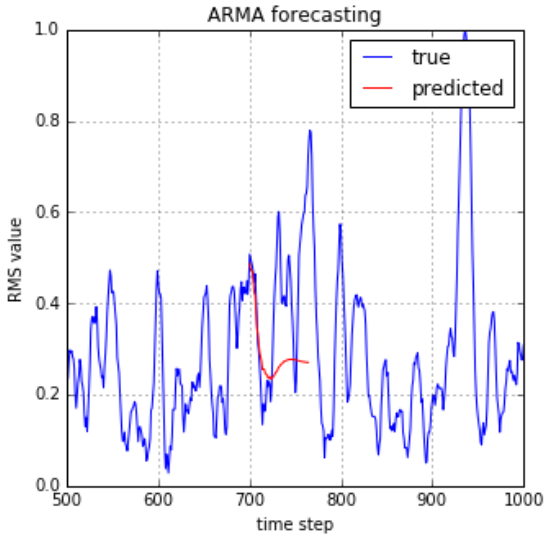


Figure 5.2: Example result of ARMA forecasting, where blue is the true RMS over an extended period of time and red is the predicted RMS starting at time step 700 and going until time step 764. In this example the space was partitioned into $K = 10$ equal disjoint regions and 11 of the first 13 forecasted classes were correctly predicted by ARMA, corresponding to about 0.03 seconds.

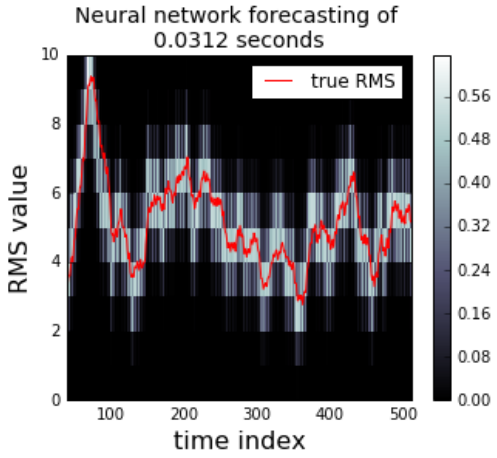
When averaged over all test datasets the forecasting ability of the ARMA model can be seen in Figure 5.4. It is seen that on average the ARMA model has difficulty doing any better than predicting the mean of the time series for future time steps further than about 0.06 seconds in the future to obtain an accuracy of 18.9%. At time steps less than that it still has much difficulty, only obtaining an accuracy of 26.3% at 0.0312 seconds in the future.

From the authors' perspective, there are two main limitations of using the ARMA model for time series prediction. The first is that the ARMA model is a linear model, and as mentioned before the example combustion processes are physically modeled by highly coupled, non-linear equations. This immediately leaves us in the realm of first order perturbed approximations of the constitutive equations.

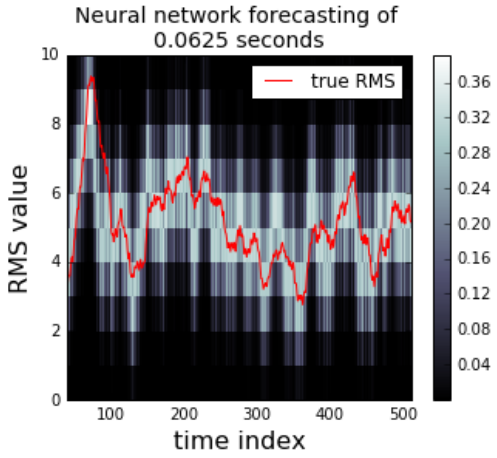
The second limitation of the ARMA model is that it is only providing a deterministic forecast prediction with error bounds, as opposed to the proposed model which generates probabilities over symbols, and does this from the ground up without any simplifications on second order statistics.

Both of these limitations can be overcome by using the proposed algorithm when modeling the system. This is because, first, the nested compositions of simple non-

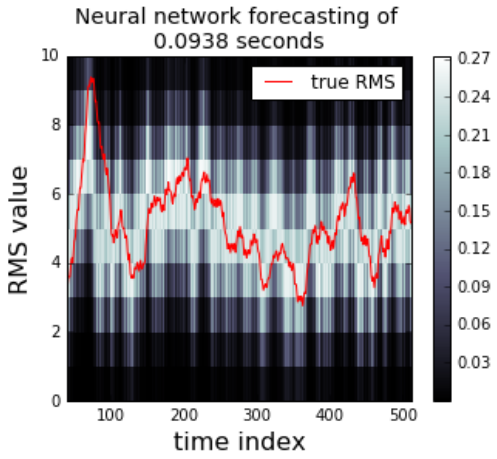
linearities can discover very complex non-linear relationships, and untangle knotted manifolds in high dimensional spaces [92]. And second, by construction the proposed algorithm gives probabilistic inferences to forecasted symbol emissions.



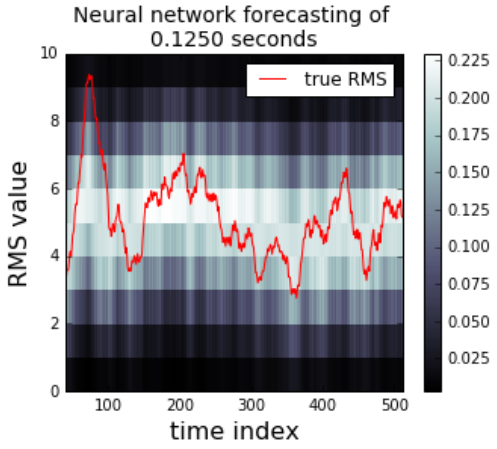
(a) Neural network forecasting 0.0312sec in future



(b) Neural network forecasting 0.0625sec in future



(c) Neural network forecasting 0.0938sec in future



(d) Neural network forecasting 0.1250sec in future

Figure 5.3: Probabilistically forecasting instabilities. The red curves are the true RMS and the grayscale map indicates predictions of what class the combustion instability belongs to, black being probability 0 and white being probability 1.

Characteristic predictive capabilities of the neural network model can be seen in Figures 5.3a, 5.3b, 5.3c and 5.3d. These figures are grayscale maps, where black corresponds to a low class prediction probability and white corresponds to a high class

prediction probability, i.e. $h_\Theta(x)_k = P(Y = k|X = x)$, and the red curve is the true RMS value given as reference. A prediction is labeled correct if the $\arg\max_k P(Y = k|X = x)$ is the same bin that the red curve is in. It is noted that these density forecasts do not necessarily have to be Gaussian.

It is seen in Figure 5.3a, i.e. for forecasting predictions of 0.0312 seconds in the future, the neural network model can very well predict with high confidence which bin the RMS curve will fall in. For Figures 5.3b and 5.3c, i.e. for 0.0625 seconds and 0.0938 seconds in the future, more gray appears in the grayscale map which means the neural network is less confident in its class predictions. And finally for Figure 5.3d, i.e. for 0.1250 seconds in the future, the class prediction probabilities are much more widely distributed, as the network is not confident in any one prediction. This is consistent with intuition, that the further in the future one is forecasting the more uncertainty that exists.

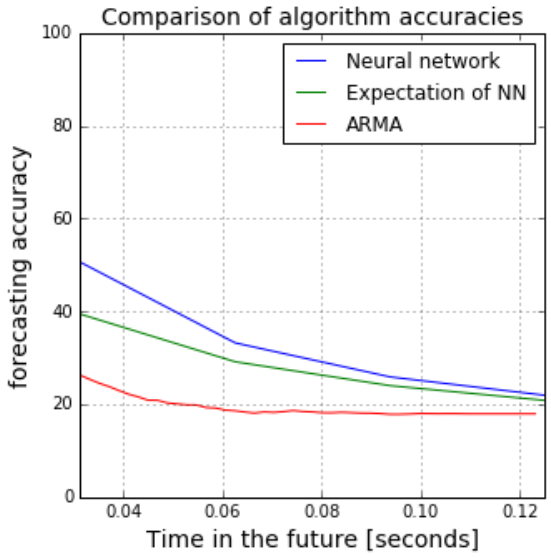


Figure 5.4: A comparison of the forecasting abilities of the three models, being the neural network model with cross-entropy minimization, the expectation over the posteriors of that neural network, and finally the ARMA model. The neural network model is labeled correct if the $\arg\max_k P(Y = k|X = x)$ is correct. The expectation of the neural network is $\hat{y} = \sum_{i=1}^{K} \bar{y}_i P(Y = y_i|X = x)$ and is labeled correct if the expected trajectory falls in the same partition region as the true trajectory.

As previously mentioned, we can use Equation 5.19 to take the expectation of the time series over the posteriors and construct an expected trajectory. We can then take this trajectory, segment it into K equal disjoint regions, and consider the performance

of this type of predictor. A comparison of the three models can be seen in Figure 5.4. It is seen that the neural network model derived from a cross-entropy minimization has the highest performance accuracy over all future time step forecasts, followed by the expectation over the posteriors, and finally the ARMA model.

At these large future time steps, the performance of the ARMA model is comparable to always guessing the mean value of the RMS time series, which suggests the complexity and difficulty of the forecasting problem. However the neural network is able to perform significantly better with improvements in performance rates from 26.3% to 50.6% at 0.0312 seconds, 19.9% to 33.2% at 0.0625 seconds, 18.4% to 25.9% at 0.0938 seconds and finally 18.0% to 21.9% at 0.125 seconds.

The expectation over the posteriors does not perform as well as taking the original $\arg\max_k P(Y = k|X = x)$. This is expected because as mentioned before, the posteriors are not meant to accurately represent the expected value of the trajectory, as this would come from an $l_2(\mathbb{R})$-norm cost minimization. The $l_2(\mathbb{R})$-norm cost minimization however has the disadvantage in that there is no natural way to quantify confidence in predictions, which is overcome by framing the problem in a probabilistic framework.

5.1.3.3 Discussion

A probabilistic means for forecasting symbol sequence using deep neural networks has been developed. The constructed model is of the form of a deep neural network, which is trained to predict class probabilities in the form of symbols. This is in contrast to the usual means of time series forecasting which takes place in a linear 2^{nd}-order fashion, such as autoregressive moving average (ARMA) models or neural networks with an $l_2(\mathbb{R})$-norm cost functional, or a cross-entropy cost functional over a continuous data as opposed to probabilities over symbols. However, by framing the goal as a cross-entropy minimization over symbol probabilities, a probabilistic interpretation is constructed from the ground up. This allows not only accurate forecasting abilities, but also confidence in predictions given by posterior probabilities.

This model has been compared with an ARMA model, which is a standard baseline algorithm in time series forecasting. It is seen that the predictive abilities of the deep neural network significantly outperform that of the baseline algorithm on the tested dataset for forecasting combustion instabilities. This is expected because the ARMA model is limited to linear predictions that assumes 2^{nd}-order statistics, where as the deep neural network with its nested compositions of non-linear activation functions, is

better equipped to discover hierarchical non-linear relationships.

A deterministic trajectory has been derived from the expectation over the posterior predictions. Its forecasting performance has been shown to significantly improve upon those of the ARMA model; however it has not performed as well as taking the $\arg\max_k P(Y = k|X = x)$ of the networks predictions. This is understood as the network was not initially designed to give deterministic predictions and also due to the highly non-linear form of the deep neural network, it was able to better model the underlying non-linear manifold the data was sampled from.

5.2 Probabilistic Forecasting with LSTM Networks

5.2.1 Mathematical preliminaries

This section succinctly presents the key mathematical concepts that are necessary to develop the algorithms.

5.2.1.1 Probabilistic formulation

Neural networks are a general class of algorithms that have nested compositions of affine transformations followed by a simple non-linearity. They can be either strictly feedforward (FF), or include some type of feedback mechanism. Neural networks with feedback are called recurrent and are usually preferred means of modeling temporal dynamics within a neural network framework.

Classification neural networks are mappings $h_\Theta : X \times Y \to [0,1]$ such that $\sum_{y \in Y} h_\Theta(x,y) = 1$, which is usually interpreted as $h_\Theta(x,y) := p(Y = y | X = x)$, where X is the finite-dimensional state space, Y is the finite space of labels (i.e., symbol alphabet) and $\Theta = \left\{ \left(W^l, b^l \right) \right\}_{l=0}^{L-1}$ is the parameterization of architecture class H such that $h_\Theta \in H$. The notation $h_\Theta(x) := h_\Theta(x, \cdot) = p(Y | X = x)$ will be used to represent the output distribution.

Classification neural networks are trained by minimizing a loss function $l(h_\Theta, q)$, where $q : X \times Y \mapsto [0,1]$ such that $\sum_{y \in Y} q(x,y) = 1$ is the true classification model. It is noted that the true distribution q may not be an element of H, i.e. $q \notin H$, but because neural network models are very flexible they may still well approximate q.

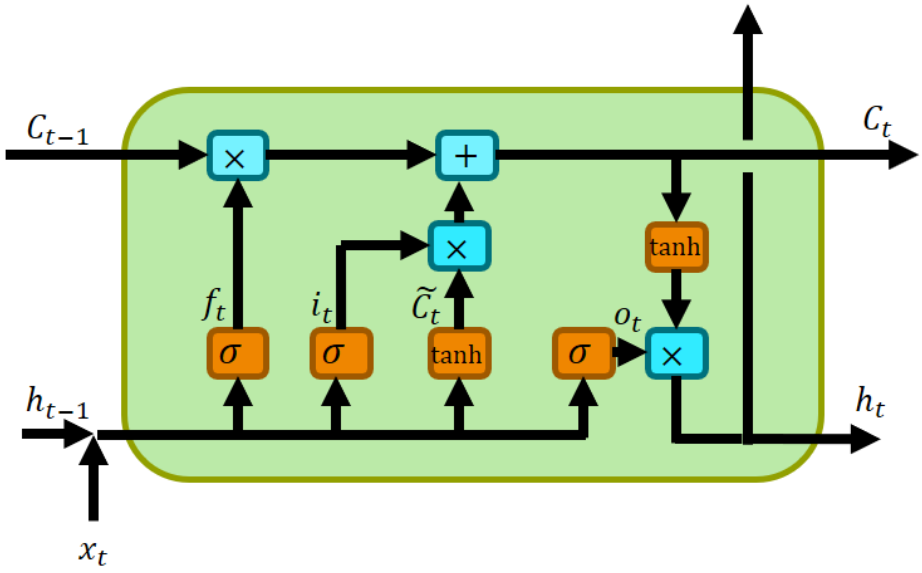


Figure 5.5: Schematic of the LSTM neural network stucture. The nonlinear activations, σ and $\tanh$, act elementwise on their respective input vectors. Similarly, multiplication and addition blocks operate on their input pairs elementwise.

5.2.1.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) neural networks [58, 94] are recurrent neural networks with a specific architecture in the hidden space which allows the network to act as if it has a memory unit it can read from and write to. Figure 5.5 presents a schematic diagram of the LSTM neural network structure for understanding how the different data elements are related-to and operate-on each other.

The LSTM has an input x_t, an output h_t and a cell state C_t that acts as the memory and is the central component of the LSTM. It is noted that the sigmoid function $\sigma(z) := 1/(1+\exp(-z))$ has a range of 0 to 1, and the hyperbolic tangent function $\tanh(z) := (\exp(z) - \exp(-z))/(\exp(z) + \exp(-z))$ has range -1 to 1.

The candidate cell state $\tilde{C}_t$, its weightings i_t, and the forget gate [95] activations f_t are computed as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{5.20}$$

$$\tilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{5.21}$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{5.22}$$

where W's and U's are appropriate weight matricies and the b's are bias vectors.

With $*$ defined to be element-wise multiplication, the updated cell state C_t is then found as:

$$C_t = i_t * \tilde{C}_t + f_t * C_{t-1} \tag{5.23}$$

and the output is otained as a weighted version of the cell state:

$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_f\right), \tag{5.24}$$

$$h_t = o_t * \tanh\left(C_t\right) \tag{5.25}$$

Because Equations (5.20), (5.21), (5.22) and (5.24) depend only on x_t, h_t and h_{t-1}, their arguments are computed in parallel.

5.2.1.3 Time series symbolization

The procedure of time series symbolization is built by constructing a partition of the time series feature space, defining mutually exclusive and exhaustive regions over the feature space. Once a partition is defined over the feature space, the data are symbolized by mapping each data point to a corresponding symbol which is uniquely identified with that partitioning region. With K symbols, the symbol set is defined to be $Y := \{y_1, y_2, ..., y_K\}$.

For one-dimensional time series, two commonly used partitioning schemes are the uniform partitioning and the maximum entropy partitioning [88]. For general multi-dimensional time series, the right column clustering, Gaussian mixture models, or other unsupervised clustering techniques can be used [89]. This work uses uniform partitioning.

5.2.2 Algorithm Development

This section outlines an algorithm for neural probabilistic forecasting of symbol sequences. Given a time series interval up to the present $\{x\left(t - t'\right)\}_{t'=0}^{l-1}$ of length l, the algorithm predicts the probability of the symbol y_k at a time instant $t + T$ as:

$$p\left(Y\left(t + T\right) = y_k | X\left(t\right) = \{x\left(t - t'\right)\}_{t'=0}^{l-1}\right) \tag{5.26}$$

The data can thus be organized into input-output pairs as follows:

$$\mathcal{D} := \left\{ \left(\left\{ x\left(n - t'\right) \right\}_{t'=0}^{l-1}, y\left(n + T\right) \right) \right\}_{n=l-1}^{N} \tag{5.27}$$

Here l is the memory length of the time series obtained from the first minima of the mutual information with a time shifted version of itself [96]); N is the number of data samples; and T is the future forecasting time step. It is noted that $l \ll N$.

The dataset is carefully organized so that the experiments conducted in Subsection 5.2.3.2 use exactly the same conditions for the different neural architectures at a given forecast-length and number-of-symbols pair. This data organization ensures fairness in generating exactly the same conditions for all neural architectures by comparing the plotted true trajectory for a fixed pair; this is seen in Figure 5.7 in Subsection 5.2.3.2 .

5.2.2.1 Mathematical formulation

As described in Section 5.2.1.1, a neural network defines a mapping: $h_\Theta : X \times Y \mapsto [0, 1]$ such that $\sum_{y \in Y} h_\Theta\left(x, y\right) = 1$, where X and Y are the input and label spaces, and $\Theta = \left\{ \left(W^l, b^l \right) \right\}_{l=0}^{L-1}$ is the parameterization.

Because we are dealing with time series of the input and output spaces, the mapping used here is defined as $h_\Theta : \left(X \times T_1\right) \times \left(Y \times T_2\right) \mapsto [0, 1]$ where X is the input space, Y is the symbol alphabet set, and T_1 and T_2 are the forecast shifted time indices of X and Y, respectively. The definition of h_Θ is as follows:

$$h_\Theta\left(\left\{ x\left(t - t'\right) \right\}_{t'=0}^{l-1} \right) := p\left(Y\left(t + T\right) | X = \left\{ x\left(t - t'\right) \right\}_{t'=0}^{l-1} \right) \tag{5.28}$$

The forecasting problem is formulated within a probabilistic framework in terms of the objective function that is chosen as the cross entropy between the true probability distribution $q\left(x, y\right)$ such that, $\sum_{y \in Y} q\left(x, y\right) = 1$, and the model's symbol emission probability distribution $h_\Theta\left(x, y\right) = p\left(Y = y | X = x\right)$. The corresponding loss function is:

$$l\left(h_\Theta\left(x\right), q\left(x\right)\right) := -\sum_{k=1}^{K} q\left(x, y_k\right) \log\left(h_\Theta\left(x, y_k\right)\right) \tag{5.29}$$

For brevity, $x \equiv \left\{ x\left(t - t'\right) \right\}_{t'=0}^{l-1}$ and y_k is realized at time $t + T$. Note that there are K symbols in the symbol set.

The cross entropy between two probability distributions q and h_Θ over the same sample space is intuitively understood as the average number of bits required to identify a realization drawn from the sample space if the coding scheme is optimized for an approximated distribution h_Θ, as opposed to the true distribution q [72]. Thus minimizing the cross entropy between these two distributions is a way of minimizing the distance, measured in bits, between them.

In a deterministic regression problem for forecasting $h_\Theta : X \to Y$, the cost functional of cross entropy is taken to be the loss function [90]:

$$l\left(h_\Theta\left(x\right), y\right) := y \log\left(h_\Theta(x)\right) + \left(1 - y\right) \log\left(1 - h_\Theta(x)\right) \tag{5.30}$$

where it is noted that $h_\Theta\left(x\right) \in \mathbb{R}$ and $y \in \mathbb{R}$ are not probability distributions, because they are actual values and are not symbolized in a finite state space; hence, they are not probabilistic and takes the form as a deterministic regression. Similarly, the $\ell_2(\mathbb{R})$-norm cost functional is also used for regression tasks, as opposed to determining probabilities of new classes and symbols. If used in a probabilistic sense the $\ell_2(\mathbb{R})$-norm cost functional assumes restrictive second-order statistics.

Training the neural network to make inferences involves taking the expected loss over Eq. (5.29).

$$L\left(\Theta; \mathcal{D}\right) := E_{(X,Y)}\left[l\left(h_\Theta\left(x, y\right), q\left(x, y\right)\right)\right] \tag{5.31}$$

The best model is taken to be $\underset{h_\Theta \in H}{\operatorname{argmin}}\, L(\Theta; \mathcal{D})$, which is trained by using Adadelta [97] in backpropagation over 10 epochs with a batch size of 100. The rationale for using Adadelta is delineated below.

1. Adadelta automatically updates the learning rate along each dimension. In this sense it is more user-friendly than standard stochastic-gradient-descent methods that often require the user to manually tune the learning rate.

2. Adadelta is an extension of Adagrad [98], but with the goal to ensure that the learning rate does not monotonically decrease to zero as the number of updates goes to infinity, as is the case with Adagrad.

3. Root mean square propagation (RMSprop) has been tried in preliminary testing as the third update rule, which is an unpublished technique used in Geoff Hinton's

Coursera lecture series on machine learning. RMSprop yielded similar results as Adadelta.

4. While stochastic gradient descent has been used in preliminary testing, Adadelta is consistently found to yield better minima in Eq. (5.31).

5.2.2.2 Testing three neural network architectures

Three different neural network architectures have been tested and compared. The first of these three architectures is a simple feedforward (FF) architecture with three hidden layers. The second is a long short-term memory (LSTM) architecture with a fully connected layer before and after the LSTM hidden layer. The third is a hybrid LSTM-FF architecture, which concatenates the outputs of the third hidden layer of the LSTM and the FF networks. All three architectures use a softmax classifier at the output and have a window size of 8 time stepssx, i.e. $x \in \mathbb{R}^8$ with hidden layer dimensions equal to 50. These three architectures have also been tested with other datasets, all yielding very similar results. Therefore, in the selection of a network architecture, computational considerations are deemed to be very important.

GPU implementations of the above three neural network architectures have been written in the Python library Theano [93, 99, 100]. The training would take, on the average, about 15 minutes per symbol size-forecast length pair. With three architectures and 100 pairs per architecture, the total training time has been about 3 days on the GPU.

5.2.3 Results and Discussion

This subsection presents the results and comparisons of the three proposed neural network algorithms, namely long short-term memory (LSTM), feed-forward (FF), and LSTM-FF running in parallel, on the chaotic experimental data collected from a combustor apparatus that is described below.

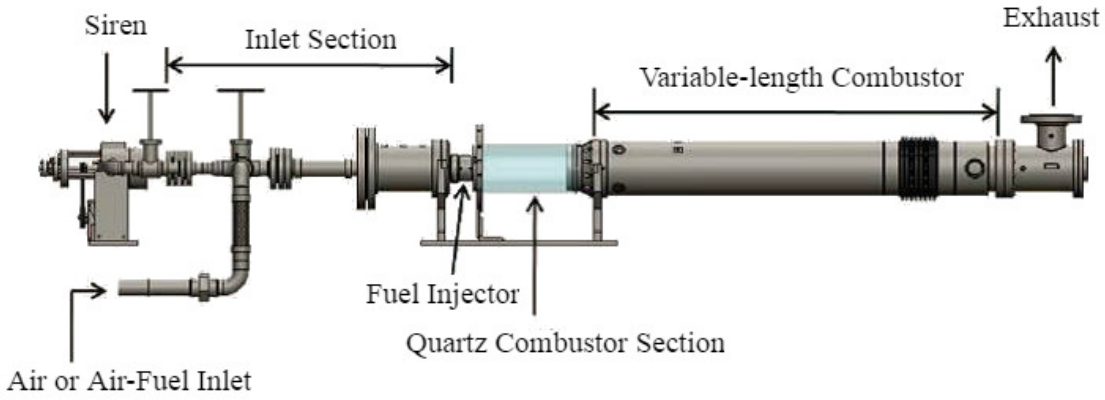


Figure 5.6: Schematic diagram of the combustor apparatus

5.2.3.1 Description of the Combustor Apparatus

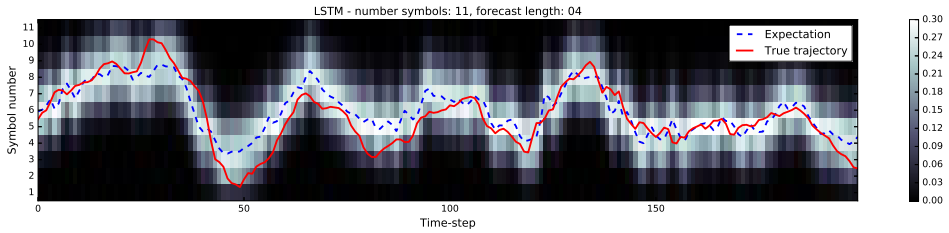
The apparatus is built upon a swirl-stabilized, lean-premixed, laboratory-scale combustor that has been used to perform the experimental investigation. Figure 5.6 shows a schematic diagram of the variable-length combustor, consisting of: an inlet section, an injector, a combustion chamber, and an exhaust section. High pressure air is delivered to the apparatus from a compressor after passing through filters to remove any liquid or solid particles that might be present in the inlet air. The air supply pressure is set to approximately 1.34 MPa using a dome pressure regulator. The air is pre-heated to a maximum temperature of $250\,^{\circ}\mathrm{C}$ by an 88kW electric heater. The fuel for this study is natural gas (approximately 95% methane) that is supplied to the combustor system at a pressure of approximately 1.48 MPa. The flow rates of the air and natural gas are measured by thermal mass flow meters. The desired equivalence ratio and mean inlet velocity is set by adjusting these flow rates with needle valves.

5.2.3.2 Forecasting Physical Phenomena

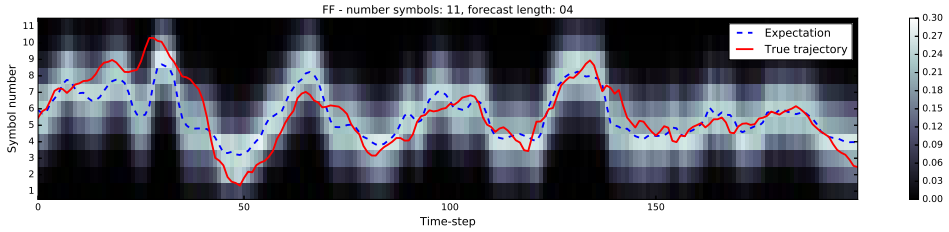
Combustion and fluid processes, which are governed by physical process dynamics that can be represented by highly coupled, nonlinear partial differential equations, generate chaotic pressure signals; the underlying dynamics are very difficult to predict. The proposed algorithms, which serve the purpose of instability prediction, are dynamic data-driven [101], instead of solely relying on constitutive model equations and thermodynamic state relations. The goal here is to predict combustion instabilities for possible usage in a feedback control system to mitigate instabilities.

Tests were conducted at a nominal combustor pressure of 1 atm over a range of operating conditions. At each operating condition, time series of pressure data were collected for 8 seconds at sampling rate 8192 Hz, which is sufficiently long to capture the dynamic characteristics of the underlying process and which, in addition, can be considered to be approximately statistically stationary. Based on these test data, individual models have been compared by averaging over their element-wise relative error rates; reductions in the average error rate between algorithms have been examined.

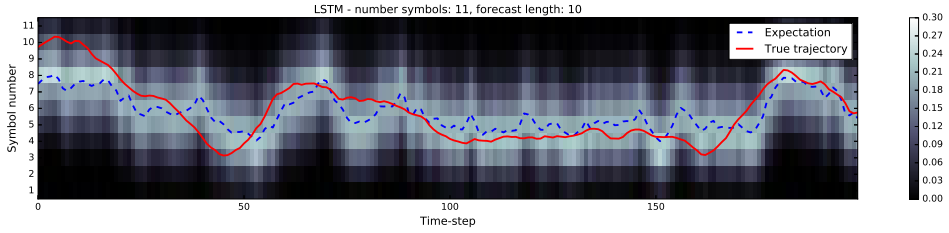
Combustion instabilities are commonly defined by the root-mean-square (RMS) values of the pressure signals inside the combustion chamber of a trailing window, with the intuition that a larger RMS value implies existence of larger pressure fluctuations and thus larger instabilities.



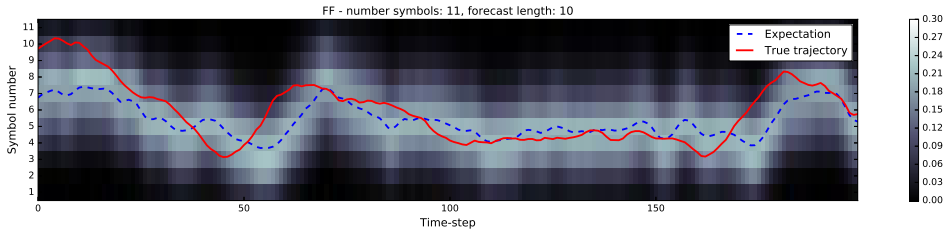
(a) An LSTM with 11 symbols and a forecast length of 4



(b) A FF architecture with 11 symbols and a forecast length of 4



(c) An LSTM with 11 symbols and a forecast length of 10



(d) A FF architecture with 11 symbols and a forecast length of 10

Figure 5.7: Forecasts from the neural probabilistic framework; black corresponds to low probability and white corresponds to high probability. The LSTM and feed forward networks are compared, as well as forecast lengths of 4 and 10 time-steps. Solid red lines are the true trajectory while dotted blue lines are the expectation over the forecasted probability distributions.

Figure 5.7 shows characteristic probabilistic forecasts, where black corresponds to low probability and white corresponds to high probability. It is seen that the LSTM makes more confident predictions than the FF architecture. As the forecast length is increased from 4 to 10 time steps, both architectures are less confident in their predictions and

thus generate more diffuse prediction densities. This is expected because there are more uncertainties in the prediction with increased forecast length. The true trajectory in red in Figure 5.7 is compared with the expectation over the forecasted probability distributions, where the centroid of the partitioned region is taken to be the expected value of the random variable.

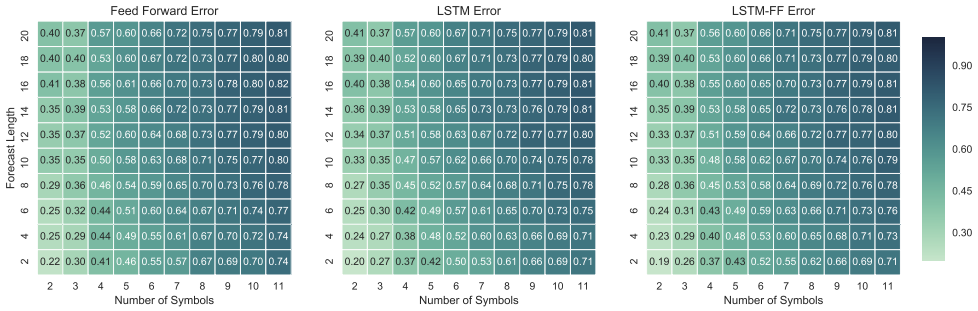


Figure 5.8: Test error rates for different combinations of forecast length and number of symbols. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate. When averaged over all pairwise combinations, the relative reductions in error can be seen in Table 5.1.

Figure 5.8 shows how the different neural network architectures perform across all test data sets. These data sets are composed of all combinations of forecast lengths of $2, 4, 6, \ldots, 20$ and the symbol alphabet size (i.e., number of symbols) being $2, 3, 4, \ldots, 11$, making 100 unique combinations in total.

Increasing the symbol alphabet size (i.e. number of symbols) enhances the fidelity of the model at the possible expense of model accuracy as seen in Figure 5.8, because of either the inherit difficulty in forecasting at such a high resolution or the finite data length [43, 45]. This general trend is seen across all three tested architectures. It is noted that the design parameters (i.e. fidelity, forecast length and accuracy) are application-specific.

Referring to Figure 5.8, Table 5.1 lists the average relative reduction in error over all pairwise combinations, where the relative error reduction is calculated as the mean of $(1 - A./B)$, and $A./B$ is element-wise division of the matricies A and B from Figure 5.8. Using this metric to summarize the results of Figure 5.8, the LSTM performs the best, followed closely by LSTM-FF and the worst performing architecture is the simple feedforward neural network.

Table 5.1: Relative reduction in error.

LSTM / FF	LSTM / LSTM-FF	LSTM-FF / FF
2.57%	0.587%	1.99%

Relative reduction in error calculated as mean$(1 - A./B)$, where $A./B$ is element-wise division of matricies from Figure 5.8.

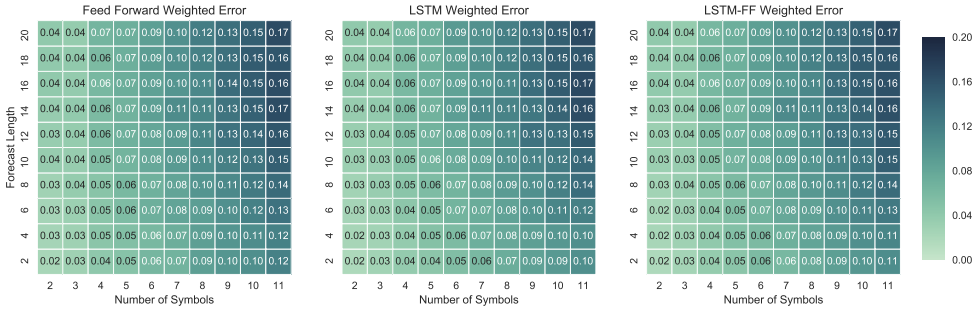


Figure 5.9: Weighted error rates for different combinations of forecast length and number of symbols, where the weighting factor is determined by the distance between the predicted symbol and the true symbol (determined by partition centroid). The weighted error can be interpreted as the average distance between the predicted and true symbol, scaled between 0 and 1. The lighter shade corresponds to a lower error rate while a darker shade corresponds to a higher error rate with a threshold value of 0.065. When averaged over all pairwise combinations, the relative reductions in weighted error can be seen in Table 5.2.

Figure 5.9 shows the errors, weighted by the distance between the predicted symbol and the true symbol. This is perhaps a better performance measure of the architectures because the error rate should be penalized more heavily by the deviation of the predicted symbol from the true symbol. The weighted error can be interpreted as the average distance between the predicted and true symbol, scaled between 0 and 1. The general trend reaffirms that the error rates increase as the forecast length increases, as well as when the number of symbols increases. When averaged over all pairwise combinations, the relative reductions in weighted error can be seen in Table 5.2, where again the relative reduction in weighted error is calculated from Figure 5.9 as mean $(1 - A./B)$, and $A./B$ is element-wise division of matricies A and B.

Table 5.2: Relative reduction in weighted error.

LSTM / FF	LSTM / LSTM-FF	LSTM-FF / FF
3.94%	0.997%	2.98%

Relative reduction in weighted error calculated as mean($1 - A./B$), where $A./B$ is element-wise division of matricies from Figure 5.9.

It is seen that the LSTM algorithm outperforms the FF algorthm, because the recurrent feedback mechanism in LSTM is specifically designed to accommodate time series data. Similarly, the LSTM outperforms the LSTM-FF, likely because the feed forward aspect of LSTM-FF is corrupting the dynamic features learned by LSTM.

5.2.4 Discussions

This work has developed a neural probabilistic framework for forecasting time series from neural network architectures in terms of discrete symbols as opposed to real values without any major limiting assumptions such as second order statistics. Although discrete symbolization is subjected to potential loss of resolution compared to continuous data, it enhances performance robustness especially if the signal-to-noise-ratio (SNR) is not high [102]. In particular, the proposed method yields performance robustness by taking the expectation over the forecasted probablity distribution. This formulation of probabilistic forecasting is suitable for continuously-varying uncertain dynamical systems, in which control actions operate at a slower time scale than that of system dynamics; such systems are prevalent in diverse mechanical engineering applications.

Three neural probabilistic architectures, namely, feedforward (FF), long short-term memory (LSTM), and combined LSTM and FF neural networks running in parallel, have been tested on experimental (time series) data of chaotic pressure oscillations collected from a laboratory-scale combustor apparatus. The performance of these three architectures has been comprehensively compared, where each architecture has been trained for 100 combinations of forecast length and number of symbols, so that application-specific requirements can be optimally chosen. It is found that, on the average, the LSTM performs the best, followed by LSTM-FF, and lastly the FF.

The future research should focus on comparison of the above three neural architectures with AR(I)MA models on user-defined, controlled chaotic datasets. For example, such datasets can be generated from solutions of the Duffing equation that can be studied

as a 2-D time-series trajectory in the phase-space with multi-dimensional symbolization techniques (e.g., k-means), as described in Subsection 5.2.1.3. While there are many other issues that need to be resolved by further theoretical and experimental research, the authors suggest comparison of the present work with various other applications (e.g., Cheng et al. [103]) as a topic of future research.

Chapter 6
Future Work and Conclusions

This chapter states future work to build off results discovered in this thesis, as well as summarizes the results with concluding remarks.

6.1 Future Work

A rigorous mathematical formulation of neural networks in the setting of Riemannian geometry has been developed in this dissertation, as well as a neural-probabilistic forecasting algorithm. In the author's opinion, the following itemized points are some of the most pertinent areas to further develop in future research.

- Future work should further develop this mathematical formulation by extending it to recurrent neural networks, since this analysis was only for feedforward neural networks. Recurrent neural networks are trained by unrolling the network and using the backpropagation through time algorithm; this essentially treats the recurrent network as a feedforward network, and so the Riemannian geometric formulation should remain appropriate in this setting as well. These two tools, feedforward and recurrent neural networks, are generally used as building blocks to construct very complex neural architectures, and so will more complete the mathematical formulation.
- In both the supervised as well as unsupervised cases, the embedding space is found to be Euclidean. In the supervised case this could be due to the fact that classes must be linearly separable by hyperplane, and this implicitly requires the coordinate representation of the manifold be flat. This however does not explain the unsupervised case. The unsupervised case learns a flattened representation of

the manifold with no explicit requirement that the representation be flat (such as in locally linear embedding). Therefore an important question for future research is to explain why the neural network learning a flattened representation of the manifold.

- One of the primary motivations behind the development of differential geometry was to formulate calculus in a coordinate free manner on general topological manifolds. In this sense, all coordinate systems are equivalent. This was one of the guiding principals behind the development of General Relativity and Quantum Field Theory (there should be no preferred frame of reference). However in neural networks, some coordinates do seem to be more appropriate for encoding and flattening data manifolds over others. For example the toy experiments conducted on the separating the spirals suggest that hyperbolic tangent is a poor choice of transformation for representing spirals as the hyperbolic tangent transformations had much difficulty separating the spirals. If instead the coordinates were transformed to polar, then the neural network would have trivially separated the spirals. These comments suggest that future work should investigate what types of coordinate transformations are best for representing what types of topological manifolds.

- Similarly to the above points, future work should investigate ways in which to improve the development of neural architectures given the insights articulated by this neural network formulation in terms of Riemannian geometry. For example, pure hyperbolic tangent or Relu transformations seem ill-suited for flattening many types of manifolds. Therefor, what types of more general transformations exist that have more flexibility, such as residual networks, dense networks and gating functions in long short-term memory networks.

- The experiments of the toy spirals and circle have an interesting resemblance to (classical) string theory, where the spatial variables are over the nodes and the temporal variable is parameterized forward through the network. In string theory, there are open strings and closed strings; both types were looked at in the experiments of Chapter 4. These strings can oscillate on the curved space-layer manifold. In (quantum) string theory, the position and momentum operators are then quantized for the quantum theory of gravity. Referring to the purely classical theory of string theory, further investigating these analogies with neural

networks, as well as neural network analogies with other non-abelian gauge theories, should be a topic of future research.

- Further investigations of the probabilistic forecasting algorithm developed should extend the analysis to multidimensional time-series. The only high level change needed is to simply use a multidimensional partitioning of the prediction space, which can be done using standard tools such as k-means. An end-to-end neural network solution however would be better in the long-term, as neural networks scale much better to high dimensional data than non-parametric methods.

  An interesting and effective means of initializing convolution filters in convolutional neural networks is to use k-means [104], inspired by vector quantization. Ng et al. showed k-means learns similar filters as the convolutional neural network, and can thus use k-means as an unsupervised technique for learning sparse filters [104]. Practically this only works at lower layers, because at higher layers more specialized filters are needed, and so the dimension of the space becomes too large for k-means to function properly. The filters learned by k-means are similar to the first layer features learned by the convolutional neural network [5], which itself is similar to the Gabor-wavelets used in the biologically inspired HMAX algorithm [11].

  From a geometric perspective, this result can be interpreted in the opposite direction; the convolutional neural network is a clustering algorithm learning to cluster the data manifold. Thus the convolutional neural network is learning a hierarchy of clusters to represent the data.

  With these intuitions an unsupervised convolutional neural network such as a deconvolutional neural network [5, 24] or a convolutional deep belief network [105] could be used to cluster the high dimensional data, and then these clusters could then define the partitioning for the symbolization of the high dimensional data.

6.2 Conclusions

The primary purpose of this dissertation was to provide a rigorous mathematical formulation to neural network actions on data manifolds in the language of algebraic and Riemannian geometry. Two major reasons for doing this are described.

First, neural networks are poorly understood, especially in comparison to the level of implementation they currently receive. Development of neural architectures are often guided by vague analogies and intuitions from a diverse range of fields. Lacking a mathematically systematic understanding of neural networks hinders progress as researchers are relegated to trial and error approaches based on their intuitions from analogous, more well understood fields. Additionally, weak mathematical formulations, such as defining neural networks as regression functions minimizing a cost between input and output, may be accurate, but are much too vague to ground further inquiry.

Second, precisely constructed models naturally have better predictive abilities than their more vague counterparts. This dissertation re-derived the error backpropagation algorithm within the developed geometric setting. Additionally, the closed form solution of the metric tensor at any coordinate layer of the network was also derived, with very weak assumptions on the metric geometry of the data manifold. The assumptions to reach this result are supported by years of research published by numerous groups in a multitude of settings. The geometric formulation naturally extends to closed form solutions of any tensor or tensor-density bundle at any layer of the neural network. Thus the precise geometric formulation of neural networks has born results that would have been extremely difficult, if not impossible, to realize had a weaker notion, such as a regression function, been used. This is because the preciseness of the model was used to derive fairly complex results, namely closed form solutions to general tensor and tensor-density bundles on the data manifold.

In addition to codifying geometric principles of neural networks, a neural network solution to probabilistic forecasting was developed. Unlike alternative techniques, the model developed does not make restrictive assumptions on the data distribution such as second order statistics. It was shown that even a simple neural probabilistic forecasting model largely outperforms the baseline ARMA model on the experimental data. The developed neural network model was further expanded by using the long short-term memory recurrent network, and was shown to further improve forecasting accuracy.

Appendix A
Proofs from Chapter 3

A.1 Introduction

This appendix includes some proofs of the more important claims made in Chapter 3.

A.2 Topological Spaces

A.2.1 The standard topology is a topology

1. $\phi \in \mathcal{O}_S$ because $\forall p : (p \in \phi \Rightarrow ...)$ is true because the statement $p \in \phi$ is false. Additionally, $M \in \mathcal{O}_s$ because $B_\epsilon(p) \subseteq M$.

2. Consider $p \in U \cap V$ then $p \in U$ and $p \in V$. But $\exists r \in \mathbb{R}^+ : B_r(p) \subseteq U$ and $\exists s \in \mathbb{R}^+ : B_s(p) \subseteq V$. This implies $B_{\min r,s}(p) \subseteq U$ and $\subseteq V$, and so $B_{\min r,s}(p) \subseteq U \cap V$.

3. Let $U \in C \subseteq \mathcal{O}_s \Rightarrow \forall p \in U \exists r \in \mathbb{R}^+ : B_r(p) \subseteq U \subseteq \cup C$.

A.2.2 The induced topology is a topology

1. $\phi \in \mathcal{O}|_N$ because $\phi = \phi \cap N \in \mathcal{O}_N$ where $\phi \in \mathcal{O}$. Similarly, $N = M \cap N \in \mathcal{O}|_N$ where $M \in \mathcal{O}$.

2. Assume $S, T \in \mathcal{O}|_N$. Then $\exists U \in \mathcal{O} : S = U \cap N$ and $\exists V \in \mathcal{O} : T = V \cap N$. Then $S \cap T = (U \cap N) \cap (V \cap N) = (U \cap V) \cap N \in \mathcal{O}|_N$ since $U \cap V \in \mathcal{O}$.

3. Arbitrary unions of induced topology are in the induced topology.

A.3 Manifolds

A.3.1 Pointwise $+_{T_pM}$ and $\cdot_{T_pM}$ close in T_pM

Show that $+_{T_pM} : T_pM \times T_pM \to T_pM$ defined by $\left(X_{\gamma,p} +_{T_pM} X_{\delta,p}\right)(f) := X_{\gamma,p}(f) +_{\mathbb{R}} X_{\delta,p}(f)$, and $\cdot_{T_pM} : \mathbb{R} \times T_pM \to T_pM$ defined by $\left(\alpha \cdot_{T_pM} X_{\gamma,p}\right)(f) := \alpha \cdot_{\mathbb{R}} X_{\gamma,p}(f)$ close in T_pM.

First, for $+_{T_pM}$. construct curve $\sigma : \mathbb{R} \to M$ such that $\left(X_{\gamma,p} +_{T_pM} X_{\delta,p}\right)(f) = X_{\gamma,p}(f) +_{\mathbb{R}} X_{\delta,p}(f) = X_{\sigma,p}(f)$ by defining
$\sigma(\lambda) := x^{-1} \circ \left(x \circ \gamma +_{C^\infty(M)} x \circ \delta -_{C^\infty(M)} x(p)\right)(\lambda)$. First note that
$\sigma(0) = x^{-1} \circ (x(p) + x(p) - x(p)) = p$. Then the following holds:
$X_{\sigma,p}(f) = (f \circ \sigma)'(0) = \left(f \circ x^{-1} \circ \left(x \circ \gamma +_{C^\infty(M)} x \circ \delta -_{C^\infty(M)} x(p)\right)\right)'(0) =$
$\partial_a (f \circ x^{-1})(x(p)) \left(x^a \circ \gamma +_{C^\infty(M)} x^a \circ \delta -_{C^\infty(M)} x^a(p)\right)'(0) =$
$(f \circ x^{-1} \circ x \circ \gamma)'(0) +_{\mathbb{R}} (f \circ x^{-1} \circ x \circ \delta)'(0) = (f \circ \gamma)'(0) +_{\mathbb{R}} (f \circ \delta)'(0) =$
$X_{\gamma,p}(f) + X_{\delta,p}(f) = \left(X_{\gamma,p} +_{T_pM} X_{\delta,p}\right)(f)$, and so $+_{T_pM}$ closes in T_pM.

Now for $\cdot_{T_pM}$, define $\sigma(\lambda) := \gamma(\alpha \cdot_{\mathbb{R}} \lambda)$. Then the following holds:
$X_{\sigma,p}(f) = (f \circ \sigma)'(0) = (f \circ x^{-1} \circ x \circ \sigma)'(0) = \partial_a (f \circ x^{-1})(x(p)) (x^a \circ \sigma)'(0) =$
$\partial_a (f \circ x^{-1})(x(p)) \, \alpha \cdot_{\mathbb{R}} (x^a \circ \gamma)'(0) = \alpha \cdot_{\mathbb{R}} \partial_a (f \circ x^{-1})(x(p)) (x^a \circ \gamma)'(0) =$
$\alpha \cdot_{\mathbb{R}} X_{\gamma,p}(f)$, and so $\cdot_{T_pM}$ closes in T_pM.

A.3.2 $\dim T_pM = \dim M$

Note that $\dim T_pM$ is a vector-space dimension, where as $\dim M$ is a topological manifold dimension.

Take a chart (U, x) where $p \in U \subseteq M$, and consider $\dim M$-many curves $\gamma_{(a)}$ where $a = 1, 2, ..., \dim M$ and $\gamma_{(a)} : \text{preim}_{\gamma_{(a)}}(U) \to U$ with $\left(x^b \circ \gamma_{(a)}\right)(\lambda) = \delta^b_a$, centered such that $x(p) = 0_{\mathbb{R}^{\dim M}}$.

Calculate the tangent vectors, with $\gamma_{(a)}(0) = p \; \forall a = 1, 2, ..., \dim M$, and define $e_a := X_{\gamma_{(a)},p} \; \forall a = 1, 2, ..., \dim M$.

$e_a(f) = X_{\gamma_{(a)},p}(f) = \left(f \circ \gamma_{(a)}\right)(0) = \left(f \circ x^{-1} \circ x \circ \gamma_{(a)}\right)'(0) =$
$\partial_b (f \circ x^{-1}) \left(x \circ \gamma_{(a)}(0)\right) \left(x^b \circ \gamma_{(a)}\right)'(0) = \partial_b (f \circ x^{-1})(x(p)) \, \delta^b_a =$
$\partial_a (f \circ x^{-1})(x(p))$ where ∂_a is the a^{th} partial derivative of a function $f : \mathbb{R}^{\dim M} \to \mathbb{R}^{\dim M}$.

Now *define* $\frac{\partial f}{\partial x^a} := \partial_a (f \circ x^{-1})$, where $f : M \to \mathbb{R}$ and $\frac{\partial f}{\partial x^a}_p := \partial_a (f \circ x^{-1})(x(p))$.
Remember that $(\partial)_{x(p)} : C^\infty\left(\mathbb{R}^d, \mathbb{R}\right) \to \mathbb{R}$, where as $\left(\frac{\partial}{\partial x^a}\right)_p : C^\infty(M) \to \mathbb{R}$.

In summary, $e_a = \left(\frac{\partial}{\partial x^a}\right)_p$ are the tangent vectors to the chart-induced curves $\gamma_{(a)}$.

Now we must show that any $X \in T_pM$ can be written as $X = X^a \left(\frac{\partial}{\partial x^a}\right)_p$, i.e. the $\left(\frac{\partial}{\partial x^1}\right)_p$, ... , $\left(\frac{\partial}{\partial x^{\dim M}}\right)_p$ are a generating system for T_pM. First note that there exists a smooth curve $\mu : \mathbb{R} \to M$ such that $X = X_{\mu,p}$ and $X_{\mu,p} f = (f \circ \mu)' (0) = (f \circ x^{-1} \circ x \circ \mu)' (0) = \partial_b (f \circ x^{-1}) (x(p)) \left(x^b \circ \mu\right)' (0) =: \left(x^b \circ \mu\right)' (0) \left(\frac{\partial f}{\partial x^b}\right)_p$.

This means that $X_{\mu,p} = \left(x^b \circ \mu\right)' (0) \left(\frac{\partial}{\partial x^b}\right)_p$, and so $\left(\frac{\partial}{\partial x^1}\right)_p$, ... , $\left(\frac{\partial}{\partial x^{\dim M}}\right)_p$ are a generating system for T_pM.

Now we must show that they are linearly independent. We have $\lambda^a \left(\frac{\partial}{\partial x^a}\right)_p f = 0$ and take $f = x^b$, then $\lambda^a \left(\frac{\partial}{\partial x^a}\right)_p x^b = \lambda^a \partial_a \left(x^b \circ x^{-1}\right) (x(p)) = \lambda^a \delta^b_a = \lambda^b = 0$.

Thus $\left(\frac{\partial}{\partial x^1}\right)_p$, ... , $\left(\frac{\partial}{\partial x^{\dim M}}\right)_p$ is a basis of T_pM and the result has been proved.

Appendix B
Additional Structures

B.1 Introduction

This appendix includes additional information of important structures that were left out of Chapter 3.

Definition B.1.1. (Algebraic Field) An algebraic field $(K, +_K, \cdot_K)$ is a set K and two maps $+_K : K \times K \to K$ and $\cdot_K : K \times K \to K$ that satisfy (for all $a, b, c \in K$):

1. Commutativity in $+_K$, $a +_K b = b +_K a$.
2. Associativity in $+_K$, $a +_K (b +_K c) = (a +_K b) +_K c$.
3. Neutral $+_K$ element exists, $\exists 0_K \in K$ such that $a +_K 0_K = 0_K +_K a = a$.
4. Inverse $+_K$ element exists, $\exists\, (-a) \in K$ such that $a +_K (-a) = (-a) +_K a = 0_K$.
5. Commutativity in $\cdot_K$, $a \cdot_K b = b \cdot_K a$.
6. Associativity in $\cdot_K$, $a \cdot_K (b \cdot_K c) = (a \cdot_K b) \cdot_K c$.
7. Neutral $\cdot_K$ element exists, $\exists 1_K \in K$, where $1_K \neq 0_K$, s.th. $\forall a \neq 0_K$ we have $a \cdot_K 1_K = 1_K \cdot_K a = a$.
8. Inverse $\cdot_K$ element exists, $\forall a \neq 0_K \exists\, (a^{-1}) \in K$ s.th. $a \cdot_K (a^{-1}) = (a^{-1}) \cdot_K a = 1_K$.
9. Distributivity of $\cdot_K$ over $+_K$, $a \cdot_K (b +_K c) = (a \cdot_K b) +_K (a \cdot_K c)$.

Note the following on distributivity:

- Distributivity makes sense because both $+_K, \cdot_K : K \times K \to K$ map their arguments to K.
- Distributivity is the only axiom that relates the $+_K$ and $\cdot_K$ operators.
- Distributivity is often written as $a \cdot_K (b +_K c) = a \cdot_K b +_K a \cdot_K c$. This is ambiguous because the right hand side can be interpreted five ways:
  1. $a \cdot_K ((b +_K a) \cdot_K c)$
  2. $a \cdot_K (b +_K (a \cdot_K c))$
  3. $(a \cdot_K b) +_K (a \cdot_K c)$
  4. $((a \cdot_K b) +_K a) \cdot_K c$
  5. $(a \cdot_K (b +_K a)) \cdot_K c$

  That is why we have the elementary school pemdas convention.

A weaker notion of an algebraic field is a ring. In the definition, axioms 5 and 8 are left out to make a direct comparison with the notion of a field.

Definition B.1.2. (Ring) A ring $(R, +_R, \cdot_R)$ is a set R and two maps $+_R : R \times R \to R$ and $\cdot_R : R \times R \to K$ that satisfy (for all $a, b, c \in R$):

1. Commutativity in $+_R$, $a +_R b = b +_R a$.
2. Associativity in $+_R$, $a +_R (b +_R c) = (a +_R b) +_R c$.
3. Neutral $+_R$ element exists, $\exists 0_R \in R$ such that $a +_R 0_R = 0_R +_R a = a$.
4. Inverse $+_R$ element exists, $\exists\, (-a) \in R$ such that $a +_R (-a) = (-a) +_R a = 0_R$.
5.
6. Associativity in $\cdot_R$, $a \cdot_R (b \cdot_R c) = (a \cdot_R b) \cdot_R c$.
7. Neutral $\cdot_R$ element exists, $\exists 1_R \in R$, where $1_R \neq 0_R$, s.th. $\forall a \neq 0_R$ we have $a \cdot_R 1_R = 1_R \cdot_R a = a$.
8.
9. Distributivity (left) of $\cdot_R$ over $+_R$, $a \cdot_R (b +_R c) = (a \cdot_R b) +_R (a \cdot_R c)$. (Right distributivity also holds)

In geometry the most important distinction between the ring and the field comes from axiom 8, existence of a multiplicative inverse for non-zero elements. Consider the set $\mathbb{R}^3$ with addition and multiplication defined point-wise, and so the zero element is $[0,0,0]$ and the one element is $[1,1,1]$. Then the non-zero element $[3,1,-2]$ has multiplicative inverse $[1/3,1,-1/2]$. Now consider the non-zero element $[3,0,-2]$. Even though this is not the zero element $[0,0,0]$, there still does not exists a multiplicative inverse. This simple example generalizes to functions, where non-zero functions do not necessarily have multiplicative inverses. Thus functions on a manifold, with $+$ and $\cdot$ defined point-wise, form a ring, but *not* a field. With functions on a manifold forming a ring, a weaker notion of a K-vector field, namely a module over a ring is defined below:

Definition B.1.3. (R-module) A left R-module, over the ring $(R, +_R, \cdot_R)$, is a tuple $(M, +_M, \cdot_M)$, where $+_M : M \times M \to M$ and $\cdot_M : R \times M \to M$ which satisfy the following (with $u, v, w \in M$ and $a, b \in R$):

1. Commutativity in $+_M$, $u +_M v = v +_M u$.
2. Associativity in $+_M$, $u +_M (v +_M w) = (u +_M v) +_M w$.
3. Neutral $+_M$ element exists, $\exists 0_M \in M$ such that $u +_M 0_M = 0_M +_M u = u$.
4. Inverse $+_M$ element exists, $\exists\, (-u) \in M$ such that $u +_M (-u) = (-u) +_M u = 0_M$.
5. Associativity of $\cdot_R$ and $\cdot_M$, $a \cdot_R (b \cdot_M u) = (a \cdot_R b) \cdot_v u$.
6. Distributivity of scalar multiplication over vector addition, $a \cdot_M (u +_M v) = (a \cdot_M u) +_M (a \cdot_M v)$.
7. Distributivity of scalar multiplication over field addition, $(a +_R b) \cdot_M u = (a \cdot_M u) +_M (a \cdot_M v)$.
8. Unitary scalar multiplication with element $1_R \in R$, $1_R \cdot_M u = u$

In geometry, we will have modules where the vectors are in the tangent spaces and the fields are in the function spaces.

Bibliography

[1] Von Neumann, J. (1955) *Mathematical foundations of quantum mechanics*, 2, Princeton university press.

[2] Misner, C. W., K. S. Thorne, and J. A. Wheeler (2017) *Gravitation*, Princeton University Press.

[3] Knight, W. (2017) "The Dark Secret at the Heart of AI," *MIT Technology Review*.

[4] LeCun, Y., Y. Bengio, et al. (1995) "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, **3361**(10), p. 1995.

[5] Zeiler, M. D. and R. Fergus (2014) "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Springer, pp. 818–833.

[6] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012) "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105.

[7] Tishby, N. and N. Zaslavsky (2015) "Deep learning and the information bottleneck principle," in *Information Theory Workshop (ITW), 2015 IEEE*, IEEE, pp. 1–5.

[8] Hinton, G. E. and R. R. Salakhutdinov (2006) "Reducing the dimensionality of data with neural networks," *science*, **313**(5786), pp. 504–507.

[9] Hinton, G. E. (2002) "Training products of experts by minimizing contrastive divergence," *Neural computation*, **14**(8), pp. 1771–1800.

[10] Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014) "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680.

[11] Riesenhuber, M. and T. Poggio (1999) "Hierarchical models of object recognition in cortex," *Nature neuroscience*, **2**(11), pp. 1019–1025.

[12] Serre, T., L. Wolf, and T. Poggio (2005) "Object recognition with features inspired by visual cortex," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, Ieee, pp. 994–1000.

[13] Bengio, Y., G. Mesnil, Y. Dauphin, and S. Rifai (2013) "Better mixing via deep representations," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 552–560.

[14] LeCun, Y., Y. Bengio, and G. Hinton (2015) "Deep learning," *Nature*, **521**(7553), pp. 436–444.

[15] Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014) "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, **15**(1), pp. 1929–1958.

[16] Huang, G., Z. Liu, K. Q. Weinberger, and L. van der Maaten (2016) "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*.

[17] Chung, J. S., A. Senior, O. Vinyals, and A. Zisserman (2016) "Lip reading sentences in the wild," *arXiv preprint arXiv:1611.05358*.

[18] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2015) "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

[19] Turian, J., L. Ratinov, and Y. Bengio (2010) "Word representations: a simple and general method for semi-supervised learning," in *Proceedings of the 48th annual meeting of the association for computational linguistics*, Association for Computational Linguistics, pp. 384–394.

[20] Socher, R., M. Ganjoo, C. D. Manning, and A. Ng (2013) "Zero-shot learning through cross-modal transfer," in *Advances in neural information processing systems*, pp. 935–943.

[21] Mikolov, T., W.-t. Yih, and G. Zweig (2013) "Linguistic regularities in continuous space word representations." in *hlt-Naacl*, vol. 13, pp. 746–751.

[22] Roweis, S. T. and L. K. Saul (2000) "Nonlinear dimensionality reduction by locally linear embedding," *science*, **290**(5500), pp. 2323–2326.

[23] Chomsky, N. (2002) *Syntactic structures*, Walter de Gruyter.

[24] Zeiler, M. D., D. Krishnan, G. W. Taylor, and R. Fergus (2010) "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, IEEE, pp. 2528–2535.

[25] Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011) "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, **12**(Aug), pp. 2493–2537.

[26] Dalal, N. and B. Triggs (2005) "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, IEEE, pp. 886–893.

[27] Lowe, D. G. (1999) "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, Ieee, pp. 1150–1157.

[28] Mutch, J. and D. G. Lowe (2008) "Object class recognition and localization using sparse features with limited receptive fields," *International Journal of Computer Vision*, **80**(1), pp. 45–57.

[29] Hubel, D. H. and T. N. Wiesel (1962) "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, **160**(1), pp. 106–154.

[30] Hofstadter, D. R. (1980) "Gödel, escher, bach," *Un eterno y grácil bucle*.

[31] Karras, T., T. Aila, S. Laine, and J. Lehtinen (2017) "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*.

[32] Gallian, J. (2016) *Contemporary abstract algebra*, Cengage Learning.

[33] Bojanowski, P., A. Joulin, D. Lopez-Paz, and A. Szlam (2017) "Optimizing the latent space of generative networks," *arXiv preprint arXiv:1707.05776*.

[34] Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003) "A neural probabilistic language model," *Journal of machine learning research*, **3**(Feb), pp. 1137–1155.

[35] Montgomery, D. C., C. L. Jennings, and M. Kulahci (2015) *Introduction to time series analysis and forecasting*, John Wiley & Sons.

[36] Zhang, G., B. E. Patuwo, and M. Y. Hu (1998) "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, **14**(1), pp. 35–62.

[37] Hornik, K., M. Stinchcombe, and H. White (1989) "Multilayer feedforward networks are universal approximators," *Neural networks*, **2**(5), pp. 359–366.

[38] Gneiting, T. (2008) "Editorial: probabilistic forecasting," *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, pp. 319–321.

[39] Gneiting, T. and M. Katzfuss (2014) "Probabilistic forecasting," *Annual Review of Statistics and Its Application*, **1**, pp. 125–151.

[40] Box, G. E., G. M. Jenkins, G. C. Reinsel, and G. M. Ljung (2015) *Time series analysis: forecasting and control*, John Wiley & Sons.

[41] Dupont, P., F. Denis, and Y. Esposito (2005) "Links between probabilistic automata and hidden Markov models, probability distributions, learning models and induction algorithms," *Pattern Recognition*, **38**(9), pp. 1349–1371.

[42] Rozenberg, G. and A. Salomaa (1997) *Handbook of Formal Languages: Beyonds words*, vol. 3, Springer Science & Business Media.

[43] Wen, Y., K. Mukherjee, and A. Ray (2013) "Adaptive pattern classification for symbolic dynamic systems," *Signal Processing*, **93**(1), pp. 252–260.

[44] Ray, A. (2004) "Symbolic dynamic analysis of complex systems for anomaly detection," *Signal Processing*, **84**(7), pp. 1115–1130.

[45] Mukherjee, K. and A. Ray (2014) "State splitting and merging in probabilistic finite state automata for signal representation and analysis," *Signal Processing*, **104**, pp. 105–119.

[46] Kurkova, V. (1992) "Kolmogorov's theorem and multilayer neural networks," *Neural networks*, **5**(3), pp. 501–506.

[47] Zeiler, M. D. and R. Fergus (2013) "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*.

[48] Ioffe, S. and C. Szegedy (2015) "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, pp. 448–456.

[49] Zeiler, M. D., M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, et al. (2013) "On rectified linear units for speech processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, IEEE, pp. 3517–3521.

[50] Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013) "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30.

[51] Nair, V. and G. E. Hinton (2010) "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

[52] Wan, L., M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus (2013) "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, pp. 1058–1066.

[53] Wager, S., S. Wang, and P. S. Liang (2013) "Dropout training as adaptive regularization," in *Advances in neural information processing systems*, pp. 351–359.

[54] He, K., X. Zhang, S. Ren, and J. Sun (2016) "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

[55] ——— (2016) "Identity mappings in deep residual networks," in *European Conference on Computer Vision*, Springer, pp. 630–645.

[56] Veit, A., M. J. Wilber, and S. Belongie (2016) "Residual networks behave like ensembles of relatively shallow networks," in *Advances in Neural Information Processing Systems*, pp. 550–558.

[57] Werbos, P. J. (1990) "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, **78**(10), pp. 1550–1560.

[58] Hochreiter, S. and J. Schmidhuber (1997) "Long short-term memory," *Neural Computation*, **9**(8), pp. 1735–1780.

[59] Hinton, G. E., S. Osindero, and Y.-W. Teh (2006) "A fast learning algorithm for deep belief nets," *Neural computation*, **18**(7), pp. 1527–1554.

[60] Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008) "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, ACM, pp. 1096–1103.

[61] Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010) "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, **11**(Dec), pp. 3371–3408.

[62] Friedman, J., T. Hastie, and R. Tibshirani (2001) *The elements of statistical learning*, vol. 1, Springer series in statistics New York.

[63] Maaten, L. v. d. and G. Hinton (2008) "Visualizing data using t-SNE," *Journal of Machine Learning Research*, **9**(Nov), pp. 2579–2605.

[64] Hinton, G. E. and S. T. Roweis (2003) "Stochastic neighbor embedding," in *Advances in neural information processing systems*, pp. 857–864.

[65] van der Maaten, L. (2009) "Learning a parametric embedding by preserving local structure," *RBM*, **500**(500), p. 26.

[66] Hinton, G. (2010) "A practical guide to training restricted Boltzmann machines," *Momentum*, **9**(1), p. 926.

[67] Bengio, Y. (2009) "Learning deep architectures for AI," *Foundations and Trends® in Machine Learning*, **2**(1), pp. 1–127.

[68] Simard, P. Y., D. Steinkraus, J. C. Platt, et al. (2003) "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, pp. 958–962.

[69] Kanellopoulos, I. and G. Wilkinson (1997) "Strategies and best practice for neural network image classification," *International Journal of Remote Sensing*, **18**(4), pp. 711–725.

[70] Glorot, X. and Y. Bengio (2010) "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

[71] Amari, S.-i. and H. Nagaoka (2007) *Methods of information geometry*, vol. 191, American Mathematical Soc.

[72] Cover, T. M. and J. A. Thomas (2012) *Elements of information theory*, John Wiley & Sons.

[73] Nasrabadi, N. M. (2007) "Pattern recognition and machine learning," *Journal of electronic imaging*, **16**(4), p. 049901.

[74] Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1985) *Learning internal representations by error propagation*, *Tech. rep.*, DTIC Document.

[75] Munkres, J. (2014) *Topology*, Pearson Education.

[76] Fomin, S. *Elements of the theory of functions and functional analysis*.

[77] Spivak, M. (1965) *Calculus on manifolds*, vol. 1, WA Benjamin New York.

[78] Walschap, G. (2012) *Metric structures in differential geometry*, vol. 224, Springer Science & Business Media.

[79] Lee, J. M. (2006) *Riemannian manifolds: an introduction to curvature*, vol. 176, Springer Science & Business Media.

[80] ——— (2003) "Smooth manifolds," in *Introduction to Smooth Manifolds*, Springer, pp. 1–29.

[81] Hauser, M. and A. Ray (2017) "Principles of Riemannian Geometry in Neural Networks," in *Advances in Neural Information Processing Systems*, pp. 2804–2813.

[82] Boulch, A. (2017) "ShaResNet: reducing residual network parameter number by sharing weights," *arXiv preprint arXiv:1702.08782*.

[83] Landau, L., E. Lifshitz, and R. Donnelly (1972) "Mechanics, Vol. I of Course on Theoretical Physics," *American Journal of Physics*, **40**(7), pp. 1050–1051.

[84] Gelfand, I. M., R. A. Silverman, et al. (2000) *Calculus of variations*, Courier Corporation.

[85] Wedderburn, J. H. M. (1934) *Lectures on matrices*, vol. 17, American Mathematical Soc.

[86] Hauser, M., Y. Fu, Y. Li, S. Phoha, and A. Ray (2017) "Probabilistic forecasting of symbol sequences with deep neural networks," in *American Control Conference (ACC), 2017*, IEEE, pp. 3147–3152.

[87] Hauser, M., Y. Fu, S. Phoha, and A. Ray "Neural Probabilistic Forecasting of Symbolic Sequenceswith Long Short-Term Memory," in *Journal of Dynamic Systems, Measurement and Controls, year=2018, organization=ASME*.

[88] Li, Y., P. Chattopadhyay, and A. Ray (2015) "Dynamic data-driven identification of battery state-of-charge via symbolic analysis of input–output pairs," *Applied Energy*, **155**, pp. 778–790.

[89] Hauser, M., Y. Li, J. Li, and A. Ray (2016) "Real-time combustion state identification via image processing: A dynamic data-driven approach," in *2016 American Control Conference (ACC)*, pp. 3316–3321.

[90] Nasr, G. E., E. Badr, and C. Joun (2002) "Cross Entropy Error Function in Neural Networks: Forecasting Gasoline Demand." in *FLAIRS Conference*, pp. 381–384.

[91] Seabold, S. and J. Perktold (2010) "Statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*.

[92] Whitney, H. (1936) "Differentiable manifolds," *Annals of Mathematics*, pp. 645–680.

[93] Theano Development Team (2016) "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, **abs/1605.02688**. URL `http://arxiv.org/abs/1605.02688`

[94] Graves, A. (2012) "Supervised sequence labelling," in *Supervised Sequence Labelling with Recurrent Neural Networks*, Springer, pp. 5–13.

[95] Gers, F. A., J. Schmidhuber, and F. Cummins (2000) "Learning to forget: Continual prediction with LSTM," *Neural Computation*, **12**(10), pp. 2451–2471.

[96] Abarbanel, H. (2012) *Analysis of observed chaotic data*, Springer Science & Business Media.

[97] Zeiler, M. D. (2012) "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*.

[98] Duchi, J., E. Hazan, and Y. Singer (2011) "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, **12**(Jul), pp. 2121–2159.

[99] Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. Goodfellow, A. Bergeron, N. Bouchard, D. Warde-Farley, and Y. Bengio (2012) "Theano: new features and speed improvements," *arXiv preprint arXiv:1211.5590*.

[100] Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010) "Theano: A CPU and GPU math compiler in Python," in *Proc. 9th Python in Science Conf*, pp. 1–7.

[101] Darema, F. (2005) "Dynamic data driven applications systems: New capabilities for application simulations and measurements," in *5th International Conference on Computational Science - ICCS 2005*, Atlanta, GA; USA.

[102] Graben, P. B. (2001) "Estimating and improving the signal-to-noise ratio of time series by symbolic dynamics," *Physical Review E*, **64**(5), p. 051104.

[103] Cheng, L., W. Liu, Z.-G. Hou, and J. Yu (2015) "Neural network based nonlinear model predictive control for piezoelectric actuators," *IEEE Transactions on Industrial Electronics*, **62**(12), pp. 7717–7727.

[104] Coates, A. and A. Y. Ng (2012) "Learning feature representations with k-means," in *Neural networks: Tricks of the trade*, Springer, pp. 561–580.

[105] Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009) "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 609–616.

Michael Hauser

mikebenh@gmail.com • phone number (on request) • www.mbhauser.com

INTERESTS

Machine learning

- Neural networks: residual, convolution, LSTM, language models
- Manifold learning: t-SNE, Locally Linear Embedding, autoencoders, GAN
- Digital signal processing: computer vision, Fourier analysis, vector quantization

Mathematical physics

- Geometry: Riemannian and Algebraic Geometry, Information Geometry
- Physics: General Relativity, Quantum Mechanics, electromagnetics, fluid mechanics, information theory, dynamical systems theory

EDUCATION

Pennsylvania State University, State College, Pennsylvania, USA

- PhD in Mechanical Engineering — Aug 2012 – Feb 2018
  - Thesis: Principles of Riemmanian Geoemetry in Neural Networks
  - Focus: neural networks, Riemannian geometry, manifold learning
- MSc in Mechanical Engineering — Aug 2012 – Jan 2016
  - Thesis: Probabilistic Forecasting of Symbol Sequences with Neural Networks
  - Focus: neural networks, digital signal processing, symbol sequences
- MSc in Electrical Engineering — Aug 2012 – Jan 2016
  - Thesis: D-Markov machine for modeling high-speed video
  - Focus: computer vision, digital signal processing, markov modeling

University of California, Riverside, Riverside, California, USA

- MSc in Mechanical Engineering — Aug 2010 – May 2012
  - Thesis: Analysis of the multidimensional effects in biofilm
  - Focus: mass transfer, fluid mechanics

University of Toronto, Toronto, Ontario, Canada

- BSc (honours) in Physics — Sep 2006 – May 2010
  - Thesis: Inducing orbital angular momentum on light
  - Focus: Quantum optics, electromagnetics
- BSc (honours) in Mathematics — Sep 2006 – May 2010
  - Thesis: Cosmological models of the universe
  - Focus: General Relativity, Quantum Field Theory

WORK EXPERIENCE

Applied Research Laboratory, State College, PA

- Walker Fellow — Oct 2014 – Feb 2018
  - Project: Identify appliances with electric field signal
  - Project: Identify anomalies in 3D printed materials
  - Focus: Digital signal processing, computer vision, machine learning

PUBLICATIONS

[5] M. Hauser and A. Ray, “Principles of Riemannian Geometry in Neural Networks,” in *Advances in Neural Information Processing Systems (NIPS)*, Long Beach, California, USA Dec 2017.

[4] M. Hauser, Y. Fu, S. Phoha, and A. Ray, “Probabilistic forecasting of symbol sequences with Long Short-Term Memory,” *Journal of Dynamic Systems, Measurement and Control*, ASME

[3] M. Hauser, Y. Fu, Y. Li, S. Phoha, A. Ray, “Probabilistic forecasting of symbol sequences with deep neural networks,” in *American Controls Conference (ACC)*, Seattle, Washington, USA, Jul 2017.

[2] M. Hauser, Y. Li, , J. Li, and A. Ray, “Real-time combustion state identification via image processing: A dynamic data-driven approach,” in *American Controls Conference (ACC)*, Boston, Massachusetts, USA, Jul 2016.

[1] M. Hauser, and K. Vafai, “Analysis of the multidimensional effects in biofilm,” *International Journal of Heat and Mass Transfer*, Elsevier, vol. 56, no. 1, pp. 340–349, Jan 2013.