UNIVERSITY of CALIFORNIA
Santa Barbara

Dynamical Systems and Neural Networks: From Implicit Models to Memory Retrieval

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Sean Jaffe

September 2024

The dissertation of Sean Jaffe is approved:

Professor Shiyu Chang

Professor Francesco Bullo, Co-Chair

Professor Ambuj Singh, Co-Chair

September 2024

For my family.

Acknowledgements

I have the great pleasure of having two extremely supportive co-advisors. I would like to thank Professor Ambuj Singh in particular for his belief in my success and his lasting support. I would also like to thank Professor Francesco Bullo for his kindness, warmth, and contagious enthusiasm. I thank both my advisors for everlasting wisdom and immense patience. I thank Professor Shiyu Chang for his support as member of my committee.

I would like to thank the many peers in both the Dynamo lab and the Bullo lab, including Saber, Kevin, Giulia, Marco, Emilio, Robin, Veronica, Gilberto, Anand, Yohan, Abed, Haraldaur, Omid, Hongyuan, Arlei, Furkan, Sikun, Mert, Zexi, Marrianne, Christos, Rasta, Saurabh, Kha-Dinh and Deniz. I would especially like to thank Sasha Davydov, my primary collaborator, without whom this dissertation would not have been possible.

I thank Lindy, Brian, Evan, Melanie, and everyone in my extended family.

I would especially like to thank my grandfathers. First, Steve Jaffe for his encouragement and love; And Jerry, who built the Solow name, had great empathy for those less fortunate than him, and loved poetry.

Curriculum Vitæ

Sean Jaffe

Education

2024 Ph.D., Computer Science, University of California, Santa Barbara

2019 Master's of Science, Computer Science, University of Washington, Seattle, WA

2017 Bachelor's of Science, Computer Science, University of Washington, Seattle, WA

Professional Experience

2023 *Research Intern*, Gilead

2018 *Research Intern*, Nvidia

2017 *Research Intern*, Xevo

2016 *SDE Intern*, Docusign

Publications

Sean Jaffe, Alexander Davydov, Deniz Lapsekili, Ambuj Singh, and Francesco Bullo. Learning neural contracting dynamics: Extended linearization and global guarantees. Under Reivew, 2024.

Alexander Davydov, Sean Jaffe, Ambuj Singh, and Francesco Bullo. Retrieving k-nearest memories with modern Hopfield networks. In NeurIPS Workshop on Associative Memory & Hopfield Networks,

Sean Jaffe, Ambuj K. Singh, and Francesco Bullo. Idkm: Memory efficient neural network quantization via implicit, differentiable k-means, 2023.

Teaching

UCSB: Data Structures and Algorithms

UCSB: Computational Science

UCSB: Intro c++

UCSB: Data Science

University of Washington: Machine Learning

University of Washington: Computational Biology

University of Washington: Discrete Math

University of Washington: Algorithms

University of Washington: Intro Java

Service

Mentor: Pipelines. Andrea Higham, Eduardo Santos.

Mentor: Central Coast Data Science Fellowship. Sage Holter, Nadine Pearson, Dale Reilley, Roman Tellez.

Member at Large: Equity and Access in Algorithms, Mechanisms, and Optimization Reading Group

Abstract

Dynamical Systems and Neural Networks: From Implicit Models to Memory Retrieval

by

Sean Jaffe

The field of machine learning has seen great strides in the past two decades, leading to groundbreaking advancements in various domains. Yet many challenges remain in AI research. Scalability remains a persistent hurdle, with current models struggling to efficiently manage increasing amounts of data and complexity. The challenge of scalability is further highlighted by the power of the human brain, which, with its remarkable efficiency, underscores the limitations of current machine learning models. Finally, both industry and academia have faced considerable difficulties in advancing robotics, particularly in enabling neural networks to interact effectively with the physical world.

In this work, we explore how the lens of dynamical systems can address all three of these challenges. We tackle memory constraints by framing a quantization algorithm as a fixed-point equation, enabling efficient differentiation. We investigate an energy-based, biological model of human memory and reinterpret the widely used self-attention mechanism through this model. Lastly, we leverage contraction theory to train a neural network that can follow a trajectory with stability and robustness. In the first two problems, we utilize dynamical systems to differentiate neural networks, while in the latter, we employ neural networks to learn a dynamical system. Through our understanding of dynamical systems, we build on theoretical advancements and practical applications in AI, offering new insights into both memory optimization and robust robotic manipulation.

Contents

1 Introduction **1**

2 Memory Efficient Neural Network Quantization via Implicit, Differentiable k-Means **6**

2.1 Introduction 6

2.2 Related Work 8

2.2.1 Compact neural Networks 8

2.2.2 Weight Sharing and Implicit Neural Networks 8

2.2.3 Quantization of Neural Networks 9

2.3 Neural Network Quantization 10

2.3.1 Neural Network Quantization via Gradient Descent 11

2.3.2 Soft-k-means for Better Gradients 11

2.3.3 Memory Complexity of Differentiating Soft-k-means 14

2.4 Proposed Method 14

2.4.1 Defining the Fixed Point Equation 15

2.4.2 Deriving the Gradient of the Fixed Point Equation 16

2.4.3 Faster Gradient Computation with Jacobian-Free Backpropagation 18

2.5 Experiments 20

2.5.1 Small Convolutional Neural Network on MNIST 20

2.5.2 Resnet18 on CIFAR10 22

2.6 Discussion 23

3 Retrieving k-Nearest Memories with Modern Hopfield Networks **25**

3.1 Introduction 25

3.2 Methods 26

3.3 Numerical Experiments 30

3.4 Comparative Discussion to Alternative Approaches 35

3.5 Mathematical Analysis 37

3.6 Energy Interpretation of Update Rule 40

3.7 Conclusion and Future Directions 44

4 Learning Neural Contracting Dynamics: Extended Linearization and Global Guarantees 45
4.1 Introduction . . . 45
4.1.1 Related Works . . . 47
4.1.2 Contributions . . . 49
4.2 Preliminaries . . . 51
4.2.1 Contracting Linear Systems . . . 52
4.2.2 Contractivity and Exponential Stability for Nonlinear Systems . . . 53
4.3 Methods . . . 54
4.3.1 Motivation . . . 54
4.3.2 Parametrization of $A(x, x^*)$ . . . 56
4.3.3 Latent Space Representation . . . 59
4.3.4 On the Interdependence of Diffeomorphism and Learned Dynamics 60
4.3.5 Choice of Diffeomorphism . . . 62
4.3.6 Training . . . 63
4.4 Comparisons to Other Models . . . 64
4.5 Experiments . . . 66
4.5.1 Metric . . . 68
4.5.2 Datasets . . . 68
4.5.3 Results . . . 71
4.5.4 Universal Representation . . . 72
4.6 Limitations . . . 72
4.7 Conclusions . . . 73

5 Conclusion 74

Bibliography 76

Chapter 1

Introduction

Neural networks have become the dominant tool across various fields of machine learning, driving significant advancements in areas such as natural language processing (NLP), computer vision, and beyond. Their success is partly due to the development of novel architectures, like transformers, which have revolutionized tasks such as language translation and image recognition. Equally important is the role of modern computational infrastructure, which, with its massive scale, has enabled the training of increasingly complex models.

The enhanced capabilities of large neural networks, such as language models and other advanced architectures, have introduced significant challenges related to scalability and memory efficiency. As these models grow more complex and expansive, they require increasingly large amounts of computational power and memory, often leading to inefficiencies that hinder their practical deployment. For example, state-of-the-art models like

GPT-3 that has 175 billion parameters and GPT-4's estimated 1.7 trillion parameters [1] require immense computational resources for training and deployment. Running a single forward pass or even storing such a model can be computationally prohibitive, especially on hardware that is not specifically designed for such tasks, like GPUs with limited memory capacity. The sheer size of these models presents challenges not only in terms of training but also in deployment, where executing these networks in real-time becomes a significant hurdle.

While companies developing large language models have marketed their products as possessing "super artificial intelligence," advanced neural networks still exhibit clear deficiencies when compared to the human brain. The brain has mastered complex tasks like language processing and visual perception, and executes them with far less energy The stark difference in computational efficiency between the brain and artificial neural networks has spurred research into "biologically inspired" neural networks, which aim to close this gap by mimicking the efficiency of biological systems in computational models.

Despite the remarkable progress in NLP and computational sensing, there are still considerable challenges in fields like physical manipulation and robotics. Tasks such as navigation, manipulation, and trajectory tracking remain difficult for neural networks, as they require these systems to interact with and adapt to the physical world in real-time. These challenges cannot be solved merely by adding more computational power. While the brain manages these tasks with minimal computational resources, even the most advanced artificial systems struggle to achieve similar robustness and reliability, regardless

[1] www.technologyreview.com

of the amount of compute available.

This dissertation addresses these challenges through the lens of dynamical systems. All complexity in the world, including the behavior of neural networks, can be viewed as dynamical systems. By adopting this perspective, we can better understand and address the issues of scalability, memory efficiency, and stability in neural networks.

Many modern machine learning techniques, including physics-informed neural networks (PINNs) [68], implicit neural networks [7], [28], and ODE-based networks [16], utilize iterative processes that are deeply connected to dynamical systems. These processes are particularly evident in data related to physical systems. By differentiating these processes as dynamical systems, we can employ novel differentiation techniques, such as those based on the implicit function theorem [51], to achieve more efficient and accurate model training.

Moreover, the underlying architecture of neural networks themselves can be seen as dynamic systems. Neurons do not simply apply discrete filters; instead, they generate continuous signals that interact within a network, creating a complex, interconnected system of computation. Various models of neural networks, inspired by this dynamic nature, have been developed to enhance their performance and efficiency.

Robotics, one of the most challenging applications of neural networks, inherently involves dynamical systems. Long before the advent of neural networks, robotics relied on principles of dynamics for tasks like trajectory tracking. While neural networks have enabled the learning of more complex trajectories, this often comes at the cost of robustness and stability. By enforcing structural constraints on neural networks, grounded in dynamical systems theory, we can reintroduce stability guarantees, making these models more reliable for real-world robotic applications.

This dissertation explores these themes in detail, providing insights into how a dynamical systems approach can address some of the most pressing challenges in modern neural network research.

- Chapter 2 investigates the challenge of storing large neural networks in memory. The specific challenge of memory quantization is addressed. Quantization is a memory reduction technique that replaces full-precision weights to their nearest weight in a small dictionary of reference weights. Clustering algorithms like k-means have been used to find the reference weights and their assignments. In order for the weights to consider both the quantization objective and the performance objective, the clustering algorithm can be made differentiable and injected into the training loop. though, this is a computationally expensive procedure. We can reduce the computational constraints of differentiable k-means by treating the clustering as a discrete time dynamical system with a local fixed point. The dynamical systems framing of the algorithm enables the use of implicit differentiation, which induces a factor of memory savings with no performance cost.

- Chapter 3 investigates a biologically plausible model for human associative memory, the Hopfield model introduced by [39]. The Hopfield model, and it's descendant Dense Associative Memory [53], are energy based models which stores memories in the wells of an energy landscape created by a stored, low-dimensional, matrix of memories. An input is evolved by minimizing it's energy until a well and its associated memory is found. The update to an energy state is identical to the 'attention' mechanism found in transformers and in our quantization layer in chapter 2. We expand on the Hopfield model by showing how to alter the model for retrieval of multiple memories. Our method introduces a generalization of the softmax operator, which returns can return a matrix of distinct, stochastic vectors. This new operator is analogous to a multi-headed attention, that only requires the memory of a single head. We demonstrate the effectiveness of this operator in a vision-transformer.

- Chapter 4 discusses the task of robust, robotic trajectory tracking. Many areas of robotics require following complex, dynamic paths in a stable and robust fashion. A neural network can learn a trajectory fairly easily, but at the cost of stability and robustness. We learn a neural network dynamical system that is constrained to be contracting, a stronger form of stability that is equipped with robustness guarantees. Unlike prior work, our method directly parameterizes the system, rather than its jacobian, making it computationally efficient. Additionally, our method can represent systems with asymmetric Jacobians, allowing for the representation of systems with oscillatory components like pendulums.

Chapter 2

Memory Efficient Neural Network Quantization via Implicit, Differentiable k-Means

2.1 Introduction

The capabilities of deep neural networks has grown in step with their size. The size of such neural networks is a major draw-back when used on edge-devices with limited memory and energy capacity. Building small neural networks that achieve high performance is thus an essential task for real-world applications.

Works such as MobileNet [41] and SqueezeNet [42] aim to design explicitly compact neural structures. Efficient architectures can be found using neural architecture search, as in EfficentNet [79]. Weight sharing such as repeated layers [20] methods also yield compact models. Implicit neural networks [7], [28] take weight sharing to the extreme, implementing the behavior of deep neural networks with single recurrent layers.

Another means to generate compact neural networks is to compress large neural networks. Compression methods include pruning [87], knowledge-distillation [37], and quantization. We focus on quantization, the aim of which is to limit a neural network to using a limited number of unique weights. The quantized weights are generally found with a clustering routine.

DKM [19] achieved state-of-the-art results in quantization. The authors introduce a soft-k-means algorithm which uses an attention-based mechanism for assignment. DKM incorporates soft-k-means into the training loop and evaluates the loss of the model after clustering. Quantizing while training allows their algorithm to explicitly train weights which perform well after clustering. The soft assignments produce better gradients than hard assignments would. A major drawback is that DKM must store the result of every clustering iteration for the backward pass. This memory requirement is so restrictive, the clustering algorithm has to be stopped before convergence.

We solve the memory constraint of DKM by implementing the soft-k-means algorithm as a Deep Equilibrium Network (DEQ) [7]. DEQs use implicit differentiation to calculate the gradient for a layer independently of the forward pass. Implicit differentiation has been used in many novel recurrent architectures [28], [17]. We employ implicit differentiation to calculate the gradient of the clustering algorithm in *fixed memory* with respect to the number of iterations. Additionally, with Jacobian-Free Backpropagation [31], we can calculate the gradient in *fixed time*. In total, our method can perform the k-means routine to greater precision while requiring less memory and time to calculate the gradient. As a result, we are able to perform extreme quantization on large models such as Resnet18 without expensive cloud infrastructure.

2.2 Related Work

2.2.1 Compact neural Networks

A number of works design neural networks to be explicitly compact, such as MobileNet [41], EfficientNet [79], and SqueezeNet [42], and ShuffleNet [96].

We are interested in compressing already-trained, high-performing neural networks.[66] provides a thorough overview of neural network compression. Among the most well-studied neural network compression methods are pruning [87], knowledge-distillation [37] and weight sharing.

2.2.2 Weight Sharing and Implicit Neural Networks

[20] finds it beneficial to use the same weight in multiple layers. [7] take this to the extreme with their DEQ, which outputs the solution to a fixed point equation determined by a single neural network layer. Calculating the gradient of the DEQ requires implicit differentiation, which we discuss in section 2.4.2.[28] structure their DEQ as a state-space

model. [92] and [44] solve for the gradient of implicit, state-space models using monotone and non-euclidean monotone operator splitting methods, respectively. The Neural ODE [17] is another implicit model that uses a neural network to parameterize an ODE.

2.2.3 Quantization of Neural Networks

An overview of quantization is provided in [73]. The literature on neural network quantization is vast. Quantization can roughly be split into two categories, post-training quantization (PQT), and quantization aware training (QAT). PQT methods, such as [8], [58], and [30], quantize a pre-trained neural network without re-evaluation. QAT on the other hand, adjusts the model weights as it quantizes. For example [34] uses pruning, quantization, and Huffman coding, with intermediary retraining steps.

Other QAT methods directly consider the gradient of the output of the quantized model to update the unquantized model. This is inherently difficult, as hard quantization is not differentiable. [11] introduce the widely used Straight-Through Estimator to approximate the gradients of hard non-linearities. Other works use cluster-based regularization [94], [85]. Another work, [77], uses Product Quantization (PQ) to cluster subvectors of the weights.

Neural networks can also be quantized heterogeneously layer-to-layer. [89] use reinforcement learning to identify hardware-specifc clustering schemes. [18] employs a meta-neural network to approximate the gradient of the non-quantized weights.[67] uses an iterative training strategy which freezes layers subject to high instability. [24] uses properties of the spectrum of the hessian to determine quantization schema.

2.3 Neural Network Quantization

In what follows, we let $\|\cdot\|$ denote the 2-norm. Suppose we have a deep neural network f that takes input x and outputs a vector y parameterized by weights $\mathbf{W} \in \mathbb{R}^N$. For a given dataset, $\mathcal{D} = \{(x_i, y_i)\}$, the weight vector w is trained to minimize the loss:

$$\mathcal{L}(\mathbf{W}) = \sum_{(x,y)\in\mathcal{D}} \|f(x, \mathbf{W}) - y\| \tag{2.1}$$

We now employ the Product Quantization setup, as described in [77]. Consider a single layer with weights $W \in \mathbb{R}^n$ that is partitioned into $m = n/d$ sub-vectors $w_1, \dots, w_m$, each of which is an element of $\mathbb{R}^d$. The goal of neural network quantization is to learn a codebook of k, d-dimensional codewords $C = \{c_1, \dots, c_k\}$. Each element of W, w_i is mapped to its closest codeword in C. Define the map $q : \mathbb{R}^d \times \mathbb{R}^{d\times k} \to \mathbb{R}^d$ such that $q(w_i, C)$ yields the closest codeword $c \in C$ to w_i. We also define $\mathbf{q} : \mathbb{R}^{d\times m} \times \mathbb{R}^{d\times k} \to \mathbb{R}^{d\times m}$ to be the map for which $\mathbf{q}(W, C)$ replaces every column w_i of W with $q(w_i, C)$. Given a previously trained W, a codebook C that minimizes the norm between each column in W and their respective quantized weights can be found by minimizing the cost function:

$$\mathcal{C}(C, W) = \sum_{i=1}^{m} \|w_i - q(w_i, C)\|^2 \tag{2.2}$$

The k-means algorithm is guaranteed to converge to a local minimum of this objective. Hence, running k-means directly on previously trained weights is a known method for neural network quantization [34].

2.3.1 Neural Network Quantization via Gradient Descent

In model quantization, we are not just interested in minimizing the change in weights, but rather we want to minimize loss in performance as a result of quantization. To achieve this, we can incorporate the clustering loss (2.2) into the original training loss as follows:

$$\begin{aligned} \mathcal{L}(W) &= \sum_{(x,y)\in\mathcal{D}} \|f(x, \mathbf{q}(W, C)) - y\| \\ \text{with}\, C &= \operatorname*{argmin}_{C} \sum_{i} \|w_i - q(w_i, C)\|^2 \end{aligned} \tag{2.3}$$

The overall loss being minimized is the performance of the model using the quantized weights $\mathbf{q}(W, C)$ with optimal C. If C is found using k-means, a differentiable procedure, this objective can be minimized with stochastic gradient descent (SGD).

2.3.2 Soft-k-means for Better Gradients

While the k-means algorithm consists entirely of differentiable computations, the resulting gradient may lack useful information because of the hard assignment between weights and their respective centers. DKM uses continuous relaxation of the k-means algorithm. Rather than quantizing each w_i to its closest codeword in C, the algorithm quantizes each w_i by a convex combination of codewords. Let $r_\tau : \mathbb{R}^d \times \mathbb{R}^{d\times k} \to \mathbb{R}^d$ be the new quantization map, where $\tau > 0$ is a temperature parameter. The formula for the quantizer r_τ is:

$$r_\tau(w_i, C) = \sum_{j=1}^{k} a_j(w_i, C) c_j \tag{2.4}$$

with

$$a_j(w_i, c) = \frac{\exp(-\|w_i - c_j\|/\tau)}{\sum_{l=1}^{k} \exp(-\|w_i - c_l\|/\tau)} \tag{2.5}$$

$$= \big(\text{softmax}_\tau(\big(- \|w_i - c_1\|, \dots, -\|w_i - c_k\|\big)_j \tag{2.6}$$

The vector of weights $a(w_i, C)$ is the softmax of negative distances between w_i and each codeword in C. Note that if $\tau = 0$, then $r_\tau(w_i, C) = q(w_i, C)$. As before, let $\mathbf{r}_\tau : \mathbb{R}^{d\times m} \times \mathbb{R}^{d\times k} \to \mathbb{R}^{d\times m}$ be the map for which $\mathbf{r}_\tau(W, C)$ replaces every column w_i of W with $r_\tau(w_i, C)$. Let $A : \mathbb{R}^{d\times m} \times \mathbb{R}^{d\times k} \to [0, 1]^{m\times k}$ give the matrix with elements $A_{ij}(W, C) = a_j(w_i, C)$ as defined in equation (2.6). This distance-based attention matrix (introduced by [6]) allows the matrix notation of $\mathbf{r}_\tau$:

$$\mathbf{r}_\tau(W, C) = C \cdot A(W, C)^\top \tag{2.7}$$

The codebook C can be computed with EM. The attention matrix $A(W, C)$ is computed in the E-step. Distances between each weight and codeword are stored in a distance matrix $D(W, C) \in \mathbb{R}^{m\times k}_{\geq 0}$ defined by $d_{ij}(W, C) = \|w_i - c_j\|$. We define *row-wise softmax function* $\text{rowsoftmax}_\tau : \mathbb{R}^{m\times k} \to [0, 1]^{m\times k}$ so that $A(W, C)$ (2.4) is written in matrix form as:

$$A(W, C) = \text{rowsoftmax}_\tau(-D(W, C)). \tag{2.8}$$

Centers are updated in the M-step to be the average of all w, weighted by the center's importance to each w. Formally:

$$c_j^+ := \frac{\sum_{i=1}^m a_j(w_i, C) w_i}{\sum_{i=1}^m a_j(w_i, C)} \tag{2.9}$$

or, in matrix notation:

$$C^+ := \operatorname{diag}(A(W,C)^\top \mathbb{1}_m)^{-1} A(W,C)^\top W^\top \tag{2.10}$$

The algorithm is described fully in algorithm 1.

Algorithm 1 **Soft k-means**

```
Require: $W \in \mathbb{R}^{d \times m}$ with columns $(w_1, \dots, w_m)$, tolerance parameter $\epsilon$
Ensure: cluster centers $C$
1: let $C \in \mathbb{R}^{k \times d}$ be an initial value of the cluster centers
2: repeat
3:     Compute the distance matrix $D \in \mathbb{R}_{\geq 0}^{m \times k}$ by $d_{ij} := \|w_i - c_j\|$, $i \in \{1, \dots, m\}$ and $j \in \{1, \dots, k\}$
4:     Given a temperature $\tau > 0$, compute the attention matrix $A$ by
           $A(W,C) = \operatorname{rowsoftmax}_\tau(-D(W,C))$
5:     Update the cluster centers by
           $C^+ := \operatorname{diag}(A(W,C)^\top \mathbb{1}_m)^{-1} A(W,C)^\top W^\top$
6: until $\|C^+ - C\| < \epsilon$
7: RETURN $C^+$
```

We can redefine loss (2.3) using the soft quantizer r_τ to be:

$$\begin{aligned} \mathcal{L}(W) &= \sum_{(x,y)\in\mathcal{D}} \|f(x, \mathbf{r}_\tau(W, C)) - y\| \\ \text{with}\, C &= \operatorname*{argmin}_C \sum_i \|w_i - r_\tau(w_i, C)\|^2 \end{aligned} \tag{2.11}$$

2.3.3 Memory Complexity of Differentiating Soft-k-means

Suppose $C^*(W)$ is the optimal centers provided by soft-k-means (1). In order to optimize the objective (2.3) via SGD, we will have to compute the gradient of soft-k-means, $\frac{\partial C^*(W)}{\partial W}$. With backpropagation, this calculation requires saving every iteration of the algorithm in the autodiff computation graph and revisiting each iteration in the backward pass. Let $b = \lg(k)$ be the number of bits required to identify a cluster and t be the number of iterations in soft-k-means. The GPU space necessary for a forward pass of a single soft-k-means layer is $\mathcal{O}(t \cdot m \cdot 2^b)$. Thus, the memory required is dependent on the number of iterations t. Our contribution removes the need to store each iteration of soft-k-means in the forward pass for backpropagation, lowering the memory complexity to $\mathcal{O}(m \cdot 2^b)$.

2.4 Proposed Method

Inspired by the work on deep equilibrium networks, as in [7], we propose a method to calculate $\frac{\partial C^*(W)}{\partial W}$ efficiently. We use implicit differentiation to calculate the gradient of soft-k-means using only the centers found at convergence. The intermediate steps are not stored, so we can run the clustering algorithm for as many steps necessary without increasing the memory overhead. Additionally, by using Jacobian-Free Backpropagation(JFB) as in [31], we can calculate the backward pass with significantly reduced time complexity compared to the original backward computation and its corresponding DEQ implementation. In summation, our proposal is more efficient in time and space while providing more precise clustering than prior work.

In section 2.4.1 we derive a fixed point equation whose solution is that of soft-k-means. In section 2.4.2 we show how to take the gradient of that fixed point equation using only the solution. In section 2.4.3, we show how to approximate that gradient with JFB. Our proposed method, IDKM, uses implicit soft-k-means to cluster weights inside of SGD. IDKM is described in full in algorithm (2).

2.4.1 Defining the Fixed Point Equation

To motivate the implicit differentiation, we will first frame the clustering algorithm (1) as a fixed point equation. Thanks to [7], we can derive the implicit computation of the gradient directly from the fixed point equation. Given a weight matrix $W \in \mathbb{R}^{m\times d}$ and a temperature $\tau > 0$, with $\mathbb{1}_m$ being the vector of ones in $\mathbb{R}^m$ and $rs_\tau(-D)$ being shorthand for $\text{rowsoftmax}_\tau(-D(W,C))$ we define a map $F : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times m} \to \mathbb{R}^{d\times k}$:

$$F\big(C, W\big) := \text{diag}(rs_\tau(-D)^\top \mathbb{1}_m)^{-1} rs_\tau(-D)^\top W^\top \tag{2.12}$$

The fixed point problem associated with equation (2.12) is given by:

$$C^* = F(C^*, W) \tag{2.13}$$

The solution C^* is equivalent to the output of algorithm 1 by construction.

Note: While F in equation (2.12) is a function of C, one could also define a function which iterates on A instead of C that recovers the same solution C^* (after one extra evaluation of equation (2.10)).

2.4.2 Deriving the Gradient of the Fixed Point Equation

We utilize Theorem 1 in [7] to derive the gradient of the solution to the fixed point equation (2.13). For simplicity, we derive the gradient in the case when $d = 1$. Let $C^*(W)$ be the solution to the fixed point problem (2.13). We first differentiate both sides of the equilibrium condition: $C^* = F(C^*, W)$:

$$\frac{\partial C^*(W)}{\partial W} = \frac{\partial F(C^*(W), W)}{\partial W} \tag{2.14}$$

$$\frac{\partial C^*(W)}{\partial W} = \frac{\partial F(C^*, W)}{\partial W} + \frac{\partial F(C^*, W)}{\partial C^*}\frac{\partial C^*(W)}{\partial W} \tag{2.15}$$

$$\Rightarrow \Big(I_k - \frac{\partial F(C^*, W)}{\partial C^*}\Big)\frac{\partial C^*(W)}{\partial W} = \frac{\partial F(C^*, W)}{\partial W} \tag{2.16}$$

$$\Rightarrow \frac{\partial C^*(W)}{\partial W} = \Big(I_k - \frac{\partial F(C^*, W)}{\partial C^*}\Big)^{-1}\frac{\partial F(C^*, W)}{\partial W} \tag{2.17}$$

To solve(2.17), we must compute the inverse matrix, which we denote by $M^* \in R^{k\times k}$:

$$M^* = \Big(I_k - \frac{\partial F(C^*, W)}{\partial C^*}\Big)^{-1} \tag{2.18}$$

Note that a sufficient condition for the existence of the inverse matrix above is that $\frac{F(C^*,W)}{\partial C^*}$ has all eigenvalues with magnitude less than one. This is also the condition for the convergence of the fixed point equation (2.13). Even the fixed point does not have global fixed points, we know there are local fixed points around which the matrix $\frac{F(C^*,W)}{\partial C^*}$

behaves well. We can rearrange to get:

$$M^*\Big(I_k - \frac{\partial F(C^*,W)}{\partial C^*}\Big) = I_k \tag{2.19}$$

$$\Rightarrow M^* = \frac{\partial F(C^*,W)}{\partial C^*}M^* + I_k \tag{2.20}$$

Let $G : \mathbb{R}^{k\times k} \to \mathbb{R}^{k\times k}$ be the map defined such that $G(M) = \frac{\partial F(C^*,W)}{\partial C^*})M + I_k$. It is clear from equation (2.20) that M^* is the solution to the fixed point equation:

$$M^* = G(M^*) \tag{2.21}$$

Using naive forward iteration, the iteration (2.21) convergence so long as all eigenvalues of $\frac{\partial F(C^*,W)}{\partial C^*}$ have magnitude less than one. This is also the condition for convergence of the forward iteration (2.13).

We have shown so far how to calculate $\frac{\partial C^*(W)}{\partial W}$ independently of the computation of $C^*(W)$. At runtime, we take a solution $C^*(W)$, plug it into $F(C^*(W), W)$ one time and use autodiff to calculate $\frac{\partial F(C^*,W)}{\partial W}$. We then use some forward iteration solver to find the solution of the fixed point of equation (2.21), which is equivalent to $\Big(I_k - \frac{\partial F(C^*,W)}{\partial C^*}\Big)^{-1}$. Multiplying the two terms will yield $\frac{\partial C^*(W)}{\partial W}$.

To Find M, we simply run the forward iteration above until convergence. For numerical convenience, we use an averaging iteration:

$$M(t+1) = \alpha G(M(t)) + (1-\alpha)M(t) \tag{2.22}$$

The smaller the α, the more likely that the fixed point iteration will converge numerically, but the more iterations it will take to reach convergence. In our method, we set $\alpha = .25$

and if we see the iteration diverge, we start over and divide α by 2.

2.4.3 Faster Gradient Computation with Jacobian-Free Backpropagation

We further improve the efficiency of the gradient calculation by employing Jacobian-Free Backpropagation (JFB), as discussed in [31]. JFB requires expanding the inverse term (2.18) using the Neumann series:

$$M^* = \Big(I - \frac{\partial F(C^*, W)}{\partial C^*}\Big)^{-1} = \sum_{k=0}^{\infty} \Big(\frac{\partial F(C^*, W)}{\partial C^*}\Big)^k \tag{2.23}$$

JFB uses the zeroth-order approximation of the Neumann series such that $M^* = I$. This approximation allows to entirely avoid solving the fixed point equation (2.21). The gradient (2.17) now becomes:

$$\frac{dC^*(W)}{dW} = \frac{\partial F(C^*, W)}{\partial W} \tag{2.24}$$

To motivate JFB further, the implicit function theorem, which enables us to calculate the gradient implicitly, ensures that the gradient of the solution to a fixed point equation is independent of the path taken to find the solution. In the case of the k-means, suppose a solution were found by initializing at an optimal C^*. Following algorithm (1) from C^* would terminate after one iteration. The gradient of this solution path is exactly equation (2.24). The principle of the gradient being independent of the solution path means this resulting gradient is the same as that calculated through any other solution path with any other initial C.

Algorithm 2 IDKM

```
Require: Dataset $\mathcal{D} = \{(x_i, y_i)\}$, pretrained neural network with weights $\mathbf{W}$, learning rate $\lambda$
Ensure: new weights $\mathbf{W}$
 1: while Still training do
 2:    for all $(x, y) \in \mathcal{D}$ do
 3:       Turn off autodiff.
 4:       for all Layer $W \in \mathbf{W}$ do
 5:          Find $C^*(W)$ with soft-$k$ means (1),
 6:       end for
 7:       Turn on autodiff.
 8:       Calculate $\mathcal{L}(\mathbf{W})$ by (2.11).
 9:       for all Layers $W \in \mathbf{W}$ do
10:          Solve for $F$ as in (2.12) one time, recover $\frac{\partial F(C^*(W),W)}{\partial W}$ .
11:          Compute $\left(I - \frac{\partial F(C^*,W)}{\partial C^*}\right)^{-1}$ by fixed point (2.21) or JFB (2.24) for IDKM-JFB.
12:          Update $\frac{\partial C^*(W)}{\partial W}$ as in (2.17)
13:          Complete the calculation of $\nabla_W \mathcal{L}(W)$.
14:       end for
15:       $\mathbf{W}^+ = \mathbf{W} - \lambda \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$
16:    end for
17: end while
18: Return $\mathbf{W}^+$
```

2.5 Experiments

To show a proof of concept of our method, we compare directly against DKM. We evaluate our method on two tasks in multiple compression regimes. In the following experiments, we use a learning rate of 1e-4 and temperature τ, of 5e-4. We use an SGD optimizer with no momentum and train for 100 epochs. We let the clustering algorithm in all methods run until convergence or until 30 iterations, whichever comes first.

k	d	DKM	IDKM	IDKM-JFB
8	1	0.9615	**0.9717**	0.9702
4	1	**0.9518**	0.9501	0.9503
2	1	**0.7976**	0.7701	0.751
2	2	0.5512	**0.5822**	0.5044
4	2	**0.8688**	0.825	0.8444

Table 2.1: Top-1 accuracy of quantized two layer convolutional neural network pretrained on MNIST.

2.5.1 Small Convolutional Neural Network on MNIST

First, we take a small, 2-layer convolutional neural network with 2158 parameters that has been pre-trained on MNIST [56] up to 98.4% top-1 accuracy. We then run our implicit soft-k-means algorithm (IDKM), our algorithm with Jacobian-Free-Backpropagation (IDKM-JFB), and DKM[19]. We vary the values of d and k to experiment with different compression ratios. The purpose of these experiments is to show our methods do not incur a performance drop by calculating the gradient implicitly. So, we let all methods run their clustering algorithm until convergence. We use such a small model because DKM would not be able to cluster until convergence on a large model, and our comparison would be unfair.

k	d	DKM	IDKM	IDKM-JFP
8	1	3900	2560	**1847**
4	1	1723	1380	**1256**
2	1	1748	1299	**1120**
2	2	1711	1316	**1214**
4	2	1584	1418	**1301**

Table 2.2: Time in seconds required to for each method to run for 100 iterations for varying compression schemes. Quantization target is a two-layer convolutional network pre-trained on MNIST.

k	d	IDKM	IDKM-JFP
4	1	122	**95**
2	1	88	**72**
2	2	78	**69**
4	2	96	**83**

Table 2.3: Time in minutes required to for each method to run for 100 iterations for varying compression schemes. Quantization target is ResNet-18.

Our results in Table (2.1) show that IDKM performs fairly evenly to DKM. IDKM-JFB exhibits a slightly greater loss. This is expected because IDKM-JFB approximates the implicit gradient. We additionally show the total training time for each experiment in Table (2.2). IDKM-JFB is the fastest method. Again, this result is expected because IDKM-JFB uses no iteration, implicit or otherwise, to calculate the gradient. Strikingly, IDKM also trains faster than DKM. This means that solving the fixed point equation (2.21) is faster than the autodiff's backpropagation through the soft-k-means iteration. We can claim from these results that, all else being equal, both of our methods are more

k	d	**IDKM**	**IDKM-JFB**
2	1	0.5292	**0.5346**
4	1	**0.897**	0.8961
8	1	**0.9284**	0.9273
2	2	0.3872	**0.4742**
4	2	**0.897**	0.8961
16	4	0.8608	**0.8648**

Table 2.4: Top-1 accuracy on Resnet18 model pretrained on CIFAR10 using our quantization methods. DKM never outperforms random assignment with the maximum iterations allowed by our hardware (5). So, its performance is not reported. Note that when $k = 2$, the number of bits required for storage is 1. With $k = 2$ and $d = 2$, half a bit is used for every weight in the quantized model.

memory efficient and *faster* than DKM. IDKM-JFB allows for a further boost in speed with a small tradeoff in performance.

2.5.2 Resnet18 on CIFAR10

We experiment with clustering the 11,172,032-parmater Resnet18 [35]. We first take a set of publicly-available weights, replace the last layer to yield 10 labels, and fine-tune on CIFAR10 [52] to a top-1 accuracy of 93.2%. We then evaluate our quantization methods under the same compression regimes as before. Unlike with the 2-layer neural network, DKM will run out of memory for all values of k and d tested if more than 5 iterations are used. In our experiments, 5 clustering iterations or fewer will prevent the fully-quantized model from ever beating random. In this setting, it is not just significant that our methods perform well, but rather that they can perform at all. The results of IDKM and IDKM-JFB are shown in Table (2.4).

2.6 Discussion

IDKM and IDKM-JFB remove the number of iterations as a factor in the memory complexity of Quantization methods that use soft-k-means during training. This reduction of memory is enough to justify our value of our methods to the research community. We additionally argue that allowing for more clustering iterations in the forward pass improves the overall performance of the algorithm. Due to hardware limitations, we have yet to compare directly against the presented results in [19]. However, our experiments do allow us to extrapolate that our method will improve performance when used in a higher-resource setting.

In settings where DKM method does not have the memory to run until convergence, [19] simply limit the number of clustering iterations. The authors argue that this is not too restrictive on their performance, as the number of iterations necessary to converge decreases in latter epochs as the model learns well-behaved weights. However, it is still desirable to allow for as many forward iterations as possible. We find a significant decrease in performance when the number of iterations are capped. For Resnet18, DKM was simply not able to learn quantizable weights with 5 iterations, the same cap used by [19]. It's a straight forward argument that DKM will perform better when allowed more clustering iterations. IDKM exhibits comparable behavior to DKM under the same settings. So, just as in our Resnet18 experiments, we can expect that IDKM will outperform DKM when the former is allowed significantly more clustering iterations than the latter in general settings.

Enabling more clustering iterations allows us to consider more thoughtful tuning of the temperature parameter τ. [19] are encouraged to lower the temperature to increase convergence speed. Of course, lowering the temperature lowers the quality of the clusters and of the gradient information. In the future, we would like to explore using higher temperatures equipped with annealing schemes to further improve performance.

IDKM and IDKM-JFB demonstrate extreme compression on a large-scale model using modest hardware. Additionally, our methods' memory efficiency widely expands the reasonable hyperparameter space available to train with soft-k-means, making it very likely we may find a configuration with a large improvement in performance in the large-model, large-dataset regime.

Acknowledgements: This work was supported in part by AFOSR grant FA9550-22-1-0059.

Chapter 3

Retrieving k-Nearest Memories with Modern Hopfield Networks

3.1 Introduction

Hopfield networks [39, 40] are a class of neural networks with an underlying energy function and are typically used as a form of associative memory. Given a partial or noise-corrupted memory, the network updates its state to minimize its energy, converges to a fixed point, and retrieves the full memory. Hopfield networks have garnered interest thanks to recent developments on dense associative memories and modern Hopfield networks. It has been demonstrated that these newer variants possess greater storage capacity compared to their classical counterparts [53, 22, 69, 54]. Further, [69] shows a connection between modern continuous Hopfield networks (MCHNs) and the attention mechanism in transformers [88]. Building upon these breakthroughs in dense associative memories and MCHNs, recent interesting extensions include (i) universal Hopfield networks [64], (ii) kernel interpretations of MCHNs [43], and (iii) Hopfield networks with setwise connections [15]. Finally, Hopfield networks can be used to design new transformer architectures [38].

In this paper, we propose a variant of MCHN to retrieve the k-nearest memories to an input. We call this variant a *k-Hopfield layer*. In short, given a collection of memories $\{\xi_i\}_{i=1}^N$, and input x_0, the k-Hopfield layer retrieves the k memories $\xi_{i_1}, \ldots, \xi_{i_k}$ that are most similar to x_0. We provide a pictorial representation of this procedure in Figure 3.1. Our k-Hopfield layer is built upon a soft top-k operator [3], akin to how MCHNs use softmax as a soft approximation for argmax. In this way, our k-Hopfield layer is differentiable and can be seamlessly integrated into deep learning architectures trained via gradient descent. To the best of our knowledge, the k-Hopfield layer is the first Hopfield-type network to retrieve more than one memory. We demonstrate the k-Hopfield layer's ability to reconstruct obscured memories. We additionally demonstrate the utility of our approach as a multi-head attention substitute that uses fewer trainable parameters with comparable performance on small-scale vision transformers [25].

3.2 Methods

Motivation and ksoftmax: Recall the standard update rule for the modern continuous Hopfield networks (MCHNs)

$$x^+ = \Xi \,\mathrm{softmax}(\beta \Xi^\top x), \tag{3.1}$$

where $\Xi = [\xi_1, \ldots, \xi_N] \in^{n \times N}$ with $\{\xi_i\}_{i=1}^N$ being a collection of memories, and $\beta > 0$ is an inverse temperature parameter. It was shown in [69] that when the memories are sufficiently well separated, the update rule (3.1) in one step retrieves the memory closest to x_0.

We motivate our proposed approach for modifying the update rule (3.1) to retrieve k nearest memories via an intuitive explanation for the effectiveness of (3.1) in memory retrieval. In the limit as $\beta \to \infty$, $\operatorname{softmax}(\beta \Xi^\top x)$ converges to the binary vector with unity in the entry corresponding to the largest entry of the vector $\Xi^\top x$. Then $\Xi \operatorname{softmax}(\beta \Xi^\top x)$ outputs the single memory which is closest to x.

Following the intuitive explanation above, we aim to define a new operation, ksoftmax, which has the property that in the limit as $\beta \to \infty$, $\operatorname{ksoftmax}(\beta \Xi^\top x)$ converges to a binary $n \times k$ matrix where the i-th column corresponds to one of the i-th closest memories with unity in the entry of the vector corresponding to the i-th largest entry of $\Xi^\top x$. Specifically, ksoftmax serves as a smooth and soft approximation for the nonsmooth operator that outputs this binary matrix.

The key observation is the following: finding the index corresponding to the i-th largest entry of a vector may be computed by taking the top-i operator of the vector minus its top-$(i-1)$ operator, where we recall that the top-k operator of a vector x returns a binary vector of the same size as x, but with unity in the entries corresponding to the k-largest entries of x. Since the top-k operator is not differentiable, we replace it by a soft version. We discuss alternative approaches to retrieving k-nearest memories

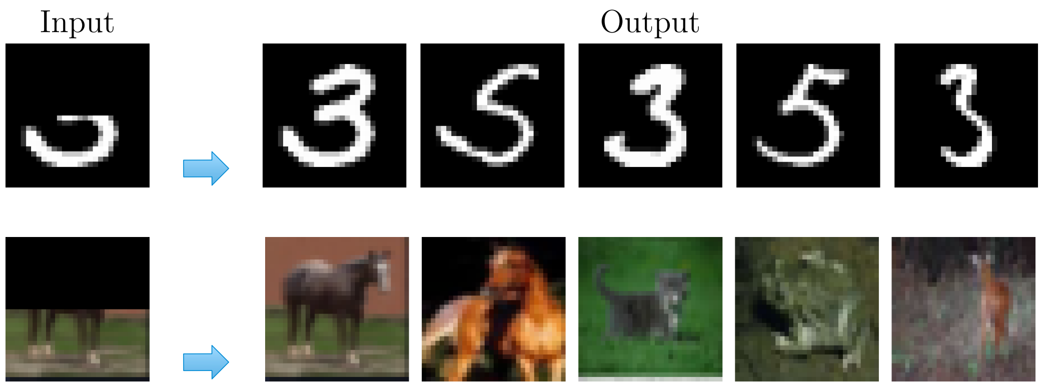


Figure 3.1: Pictorial representation of the k-Hopfield layer with $k = 5$. The left images are the occluded MNIST and CIFAR-10 inputs to the k-Hopfield layer and the right images are the outputs of the k-Hopfield layer, i.e., the k closest stored memories to the occluded input.

in 3.4. In this work, we adopt the approach of [3], which they refer to as the *limited multi-label projection layer*. We will refer to this layer as the *sum-softmax function* in analogy with the classical softmax operation.

Definition 3.2.1. The sum-softmax function, ssm $:^n \times \{1, \ldots, n\} \to^n$ is defined by

$$\mathrm{ssm}(x;k) = \underset{y \in [0,1]^n}{\operatorname{argmin}} \; -x^\top y - H_\mathrm{b}(y), \qquad \text{s.t.} \quad \mathbb{1}_n^\top y = k, \tag{3.2}$$

where $H_\mathrm{b}(y) = -\sum_{i=1}^n (y_i \log(y_i) + (1-y_i)\log(1-y_i))$ is the binary entropy function.

The sum-softmax function and softmax share the following similarity: if $k = 1$ and H_b in the objective function (3.2) is replaced with the entropy function $H(y) = -\sum_{i=1}^n y_i \log(y_i)$, then the solution to the minimization problem is equal to the output of the softmax function. The choice of using H_b rather than H in the objective function encourages less sparse gradients [3]. We establish in 3.5 that the ssm function converges to the top-k operator as $\beta \to \infty$. The forward and backward passes for the ssm function were implemented in Pytorch in [3] and can be directly applied for the problem under consideration.

With the ssm function in hand, we are ready to define the ksoftmax function.

Definition 3.2.2. For $k \leq n$, define the k-softmax function ksoftmax $:^n \to^{n\times k}$ one column at a time, for $i = 1, \ldots, k$, by:

$$\text{ksoftmax}_i(x) = \begin{cases} \text{ssm}(x;1), & \text{if } i = 1, \\ \text{ssm}(x;i) - \text{ssm}(x;i-1), & \text{otherwise} \end{cases} \tag{3.3}$$

Note that since $\text{ssm}(x;i)$ provides a smooth approximation for the binary vector with unity in entries corresponding to the top-i entries of x, then $\text{ksoftmax}_i(x)$ is an approximation to the binary vector with unity in the entry corresponding to the i-th largest entry of x. Moreover, this vector always has nonnegative entries (we prove this intuitive fact in 3.5).

***k*-Hopfield Layer and Practical Considerations:** Given an input $x_0 \in^n$, we propose the following single-step update, which we refer to as a *k-Hopfield layer*:

$$X = \Xi\,\text{ksoftmax}(\beta\Xi^\top x_0). \tag{3.4}$$

Note that although the input x_0 is a vector, the output, $X \in^{n\times k}$ is a matrix where each column corresponds to a memory. The column X_i corresponds to an approximation to the i-th closest memory to the input x_0. Although the update rule (3.4) is a map from ^{n} to $^{n\times k}$, we provide an energy interpretation of the update rule when studied one column at a time in 3.6 (i.e., the dynamical system $x^+ = \Xi\,\text{ksoftmax}_i(\beta\Xi^\top x)$, which takes the i-th column from ksoftmax).

As was studied in [64], similarity measures other than the inner product may be desirable depending on the application domain. To this end, we can modify our update rule to accommodate more general similarity measures:

$$X = \Xi\,\mathrm{ksoftmax}(\beta \mathrm{sim}(\Xi, x)), \qquad \text{where } \mathrm{sim}(\Xi, x)_i = \mathrm{sim}(\xi_i, x), \tag{3.5}$$

where $\mathrm{sim}(\xi_i, x)$ denotes a measure of similarity between the vectors ξ_i and x and is "large" when ξ_i and x are close to one another. Other choices of similarity are the negative squared Euclidean distance, and the negative Manhattan distance, as reported in [64].

Our update rule (3.4) or (3.5) can also readily be used as a first step in a larger dense associative memory. Specifically, given an input $x_0 \in^n$, one can consider the combined update

$$X = \Xi\,\mathrm{ksoftmax}(\beta_1 \Xi^\top x_0), \qquad X^+ = \Xi\,\mathrm{softmax}(\beta_2 \Xi^\top X), \tag{3.6}$$

where the softmax is applied to each column independently. Intuitively, the k-Hopfield update finds k new inputs which are close to the k memories closest to x_0 and then MCHN update retrieves these nearest memories in one step, as was shown in [69].

3.3 Numerical Experiments

Image Reconstruction: In line with the experiments run in [64] and [15], we study the ability of the k-Hopfield layer to reconstruct corrupted memories as a function of both the number of stored memories and k. Specifically, we say that a k-Hopfield layer correctly reconstructs a memory if, given a corrupted input, any of the columns of the output matrix X are sufficiently close to the uncorrupted input. We say that a reconstruction is sufficiently close to the uncorrupted input if the sum-of-squares difference between the two is less than 50.

We embed data from MNIST, CIFAR-10, and Tiny ImageNet as memories into our k-Hopfield layer. We corrupt inputs as follows: if the dataset is (i) either MNIST or Tiny ImageNet, we occlude the top half of the input image, (ii) CIFAR-10, then we occlude the top 10/32 rows of the input image. We use $\beta = 3$ and the negative Manhattan distance as similarity function in all experiments. We report the results for the k-Hopfield layer in additional to the MCHN in Figure 3.2.

We observe in all experiments that using $k = 1$ yields similar performance to using a standard MCHN for reconstruction. Additionally, we can empirically observe that using $k > 1$ provides improved ability to reconstruct the uncorrupted memory. This result makes intuitive sense since increasing k allows the layer to reconstruct multiple memories which are close to the input and check if any of these reconstructed memories are close to the uncorrupted memory.

ksoftmax Multihead Attention: Next, we assess using ksoftmax as a replacement for multihead attention. In this context, k represents the number of heads and the k-Hopfield attention update is

$$\text{kHopfieldAttn(X)} = V\,\text{ksoftmax}(\beta Q^\top K), \tag{3.7}$$

where $K = U_K X, Q = U_Q X$, and $V = U_V X$, denote the keys, values, and queries,

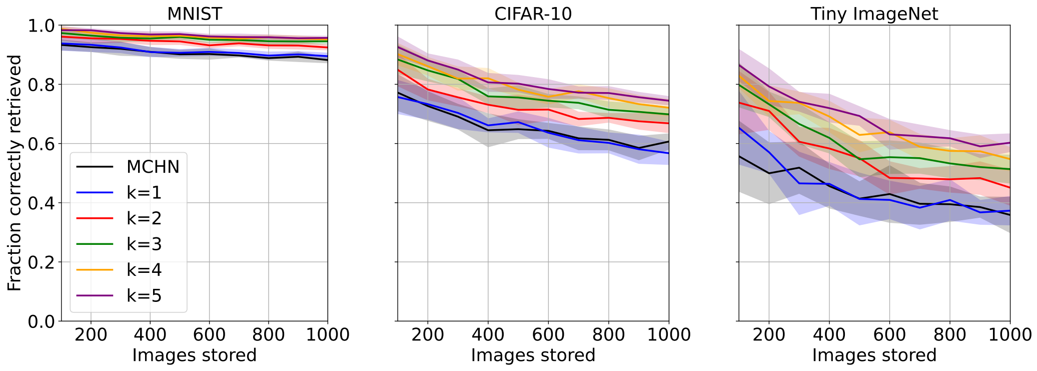


Figure 3.2: Storage capacity of the k-Hopfield layer with varying k and a MCHN [69] as measured by increasing the number of memories. We average the performance over 10 trials with different memories and report the mean and one standard deviation. We observe that $k = 1$ performs comparably to MCHN and increasing k improves storage capacity (higher is better).

respectively, and ksoftmax is applied to each column individually. Compared to standard multihead attention which runs k self-attention updates in parallel and concatenates them, the update (3.7) directly provides k different outputs. Thus, the update (3.7) serves as a substitute for multihead attention, while only requiring the parameters of a single head.

To test the k-Hopfield attention update, we train small vision transformers (ViT) [25]

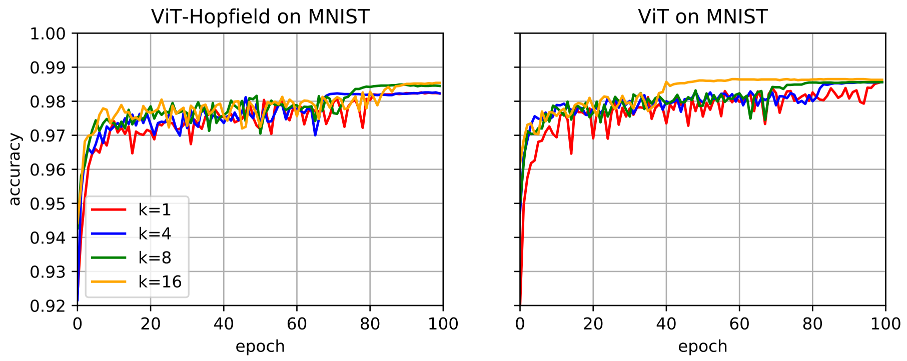


Figure 3.3: ViT training curves on MNIST with k-Hopfield attention (ViT-Hopfield) and standard multi-headed attention (ViT). k corresponds to the number of attention heads. We observe that performance of both methods is comparable, even though our method requires fewer parameters.

with standard multihead attention and with k-Hopfield attention on MNIST (Figure 3.3), CIFAR-10 (3.4), CIFAR-100 (3.5) and Tiny ImageNet [93] (3.6). We use the ViT class from https://github.com/lucidrains/vit-pytorch. We train with patch size of 4, embedding dimension of 128, MLP dimension of 512, and depth of 2 for MNIST and depth of 6 for CIFAR-10, CIFAR-100, and Tiny Imagnnet. For β, we set it equal to the usual transformer normalization constant $1/\sqrt{d}$, where d is the dimension of the head. We use an Adam optimizer with learning rate of 0.001 and no dropout. For the ssm function, we use a modified version of the implementation in https://github.com/locuslab/lml, where we edited the forward pass to use a Newton solver rather than a bracketing method.

We observe that both multiheaded attention and k-Hopfield attention perform comparably on MNIST (3.3). Hopfield-ViT performs slightly worse on the CIFAR datasets (3.4, 3.5), and exhibits less variance in accuracy across the different number of heads. We note, however, that the overall performance of both methods on CIFAR and Tiny ImageNet datasets is poor. Vision transformers are known to be difficult to train on small datasets [59]. Successfully training ViTs requires techniques such as distillation and smart initialization [82, 83]. We do not use these techniques as we are interested in comparison of the attention mechanism. We can see in our experiments on tiny ImageNet (3.6), performance again is comparable between the two methods. However, adding additional heads does not directly correspond to improved performance in either method, an artifact of the fact that ViT's have a lot of trouble with datasets like tiny ImageNet which have few samples per class.

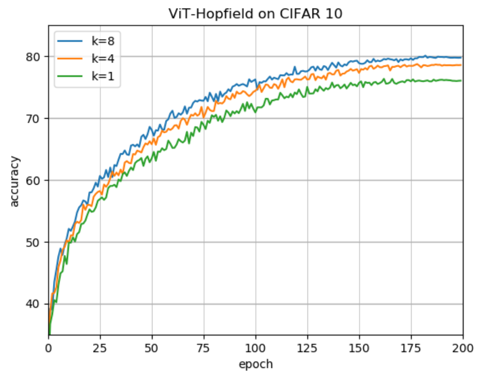


Figure 3.4: Vision transformer training curves on CIFAR-10 with k-Hopfield attention(ViT-Hopfield) and standard multi-headed attention(ViT). k corresponds to the number of attention heads. We observe that performance of both methods is comparable for k=4 heads, even though our method requires fewer parameters.

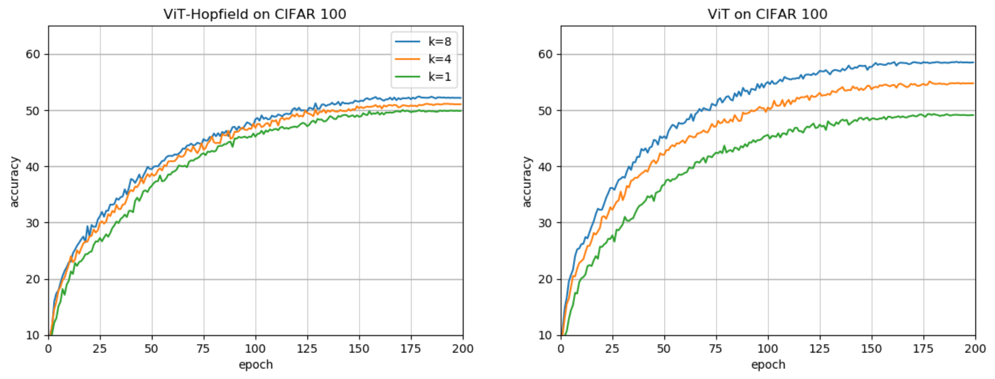


Figure 3.5: Vision transformer training curves on CIFAR-100 with k-Hopfield attention(ViT-Hopfield) and standard multi-headed attention(ViT).

We additionally propose the k-Hopfield for information retrieval. Contextual information for large language models are provided by information retrievers, which select multiple items from large vector databases that store compressed embeddings. The k-Hopfield layer naturally retrieves multiple items in a differentiable manner, allowing for the training of embeddings.

At inference time, we must harden the assignments produced by ksoftmax and we do not multiply the assignments by the original memories so that we can index directly to

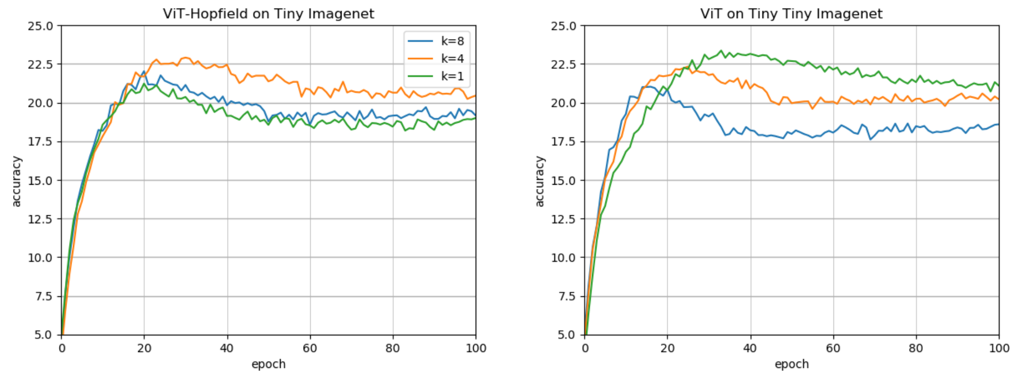


Figure 3.6: Vision transformer training curves on Tiny ImageNet with k-Hopfield attention(ViT-Hopfield) and standard multi-headed attention(ViT). k corresponds to the number of attention heads. We observe similar performance for every configuration, including networks with 4 and 8 heads.

memory. The equation for the ith memory retrieved, ξ_i, becomes:

$$\xi_i = \text{argmax}\left(\text{ksoftmax}(\beta\Xi^\top x)_i\right) \tag{3.8}$$

We leave the training of embeddings for future work. To evaluate the k-Hopfield for performing image retrieval. In our experiments, the memories are set to be 1000 images from MNIST, CIFAR-10, and Tiny ImageNet, respectively. The k-Hopfield layer is given each of theses 1000 images with added Gaussian noise and tasked with retrieving the index of the k-closest images. Recall@k is the percentage of retrieved items that are relevant. We say an item is relevant if it is one of the true, k-closest memories to the un-noised input. Results for different k and variance are shown in (3.7).

3.4 Comparative Discussion to Alternative Approaches

One alternative to retrieving k-nearest memories, for example, would be to retrieve one memory, and then remove that memory and retrieve again. Our method is better than

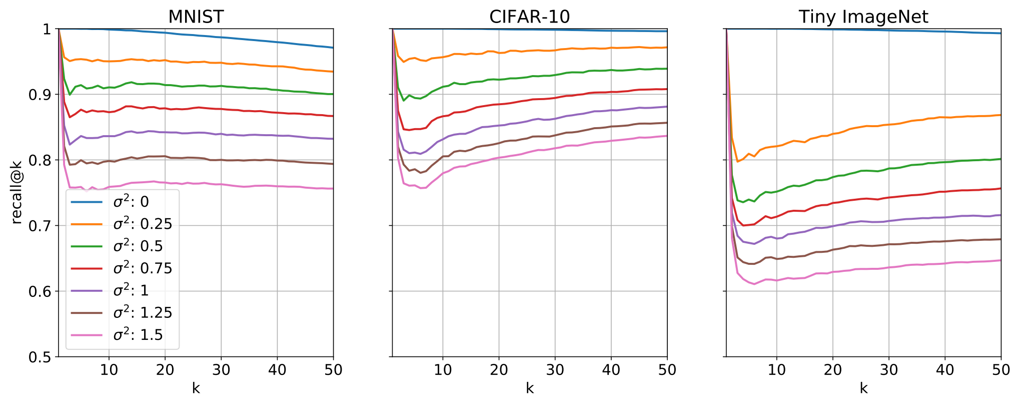


Figure 3.7: Recall@k for different k and noise.

this one in two respects. First, our method is inherently differentiable, while iteratively removing memories is not. Second, a key function of Hopfield networks is the ability to retrieve superpositions of memories, or "in-between" memories, referred to as spin-glass states [39] or metastable states [69]. The retrieval of superpositions is what makes Hopfield networks suitable for continuous learning tasks. There is no intuitive way to remove a superposition from memory if it was retrieved.

An alternative naive approach requires increasing the dimension of the state space to be $n \times k$ and defining new "memories" which are combinations of the original ones. This approach requires combinatorially many new memories compared to original ones and becomes quickly intractible.

Finally, there is a growing literature on soft top-k operators [61, 3, 95, 75]. While we adopt the approach of [3], an interesting future direction of research would evaluate the performance of alternative soft top-k operators in k-Hopfield layers.

3.5 Mathematical Analysis

Proposition 3.5.1. Let $x \in^n$ and suppose $x_{[k]} > x_{[k+1]}$, where $x_{[i]}$ denotes the i-th largest entry of x. Then

$$\lim_{\beta\to\infty} \operatorname{ssm}(\beta x; k) = \operatorname{topk}(x), \qquad \text{where } \operatorname{topk}(x)_i = \begin{cases} 1, & \text{if } x_i \in \{x_{[1]}, \dots, x_{[k]}\} \\ 0, & \text{otherwise} \end{cases}. \tag{3.9}$$

Proof. Since the optimization problem is permutation invariant, we assume, without loss of generality, that $x_1 \geq x_2 \geq \cdots \geq x_n$. Following the analysis from [3], the Lagrangian for the minimization problem arising from ksoftmax is given by

$$L(y, \lambda; \beta) = -\beta x^\top y - H_{\mathrm{b}}(y) + \lambda(k - \mathbb{1}_n^\top y).$$

The first-order optimality condition is $\nabla_y L(y^\star, \lambda^\star; \beta) = \mathbb{0}_n$ and yields the conditions $-\beta x_i + \log(y_i^\star/(1 - y_i^\star)) - \lambda^\star = 0$ for all $i \in \{1, \dots, n\}$. These conditions can be expressed succinctly as

$$y^\star = \sigma(\beta x + \lambda^\star \mathbb{1}_n), \tag{3.10}$$

where σ denotes the logistic function $\sigma(z) = 1/(1 + e^{-z})$, which acts elementwise. Moreover, to find the optimal dual variable, we substitute (3.10) into the constraint $\mathbb{1}_n^\top y^\star = k$ to find the condition

$$\sum_{i=1}^{n} \sigma(\beta x_i + \lambda^\star) = k. \tag{3.11}$$

Note that the map $\lambda \mapsto \sum_{i=1}^n \sigma(\beta x_i + \lambda)$ is continuous and monotonically increasing so that, if there is an interval $[\underline{\lambda}, \overline{\lambda}]$ such that $\sum_{i=1}^n \sigma(\beta x_i + \underline{\lambda}) \leq k$ and $\sum_{i=1}^n \sigma(\beta x_i + \overline{\lambda}) \geq k$, then there exists $\lambda^\star \in [\underline{\lambda}, \overline{\lambda}]$ satisfying (3.11).

We now aim to show that as $\beta \to \infty$, $y^\star \to \mathrm{topk}(x)$. To this end, let $\varepsilon > 0$ be given. We will prove that there exists $\beta > 0$ sufficiently large such that $y_i^\star \geq 1 - \varepsilon$ for $i \in \{1, \dots, k\}$ and $y_i^\star \leq \varepsilon$ for $i \in \{k+1, \dots, n\}$[1]. To this end, let $M = \sigma^{-1}(1-\varepsilon), m = \sigma^{-1}(\varepsilon)$ and pick[2]

$$\beta \geq \max\left\{0, \frac{M - m}{x_k - x_{k+1}}, \frac{M - \sigma^{-1}\left(\frac{\varepsilon}{n-k}\right)}{x_k - x_{k+1}}, \frac{\sigma^{-1}(1 - \varepsilon/k) - m}{x_k - x_{k+1}}\right\}.$$

Note that (1) for $y_i^\star \geq 1 - \varepsilon$ with $i \in \{1, \dots, k\}$, it is sufficient that $\lambda^\star \geq M - \beta x_k$ and (2) for $y_i^\star \leq \varepsilon$ with $i \in \{k+1, \dots, n\}$, it is sufficient that $\lambda^\star \leq m - \beta x_{k+1}$. In other words, to prove the result, we need to establish that there exists $\lambda^\star \in [M - \beta x_k, m - \beta x_{k+1}]$ satisfying the condition (3.11). First note that the interval $[M - \beta x_k, m - \beta x_{k+1}]$ is nonempty because $\beta \geq (M - m)/(x_k - x_{k+1})$. To establish the existence of $\lambda^\star$, we leverage the continuity and monotonicity of the map $\lambda \mapsto \sum_{i=1}^n \sigma(\beta x_i + \lambda)$. First, we evaluate this map at the left endpoint of our interval to obtain

$$\begin{aligned}
\sum_{i=1}^{n} \sigma(\beta x_i + M - \beta x_k) &= \sum_{i=1}^{k-1} \sigma(\beta(x_i - x_k) + M) + \sigma(M) + \sum_{i=k+1}^{n} \sigma(\beta(x_i - x_k) + M) \\
&\overset{(a)}{\leq} k - \varepsilon + \sum_{i=k+1}^{n} \sigma(\beta(x_{k+1} - x_k) + M) \\
&= k - \varepsilon + (n-k)\sigma(\beta(x_{k+1} - x_k) + M) \\
&\overset{(b)}{\leq} k - \varepsilon + \varepsilon = k,
\end{aligned}$$

where inequality (a) holds because $\sigma(z) \leq 1$ for all z, because $\sigma(M) = 1 - \varepsilon$, and because

[1] If $\epsilon \geq 1$, any choice of $\beta \geq 0$ will suffice, so we further assume $\epsilon < 1$.

[2] Note that since σ is the logistic function, σ^{-1} exists on the domain $]0, 1[$ and is given by $\sigma^{-1}(x) = \log(x/(1-x))$.

$\beta(x_i - x_k) \leq \beta(x_{k+1} - x_k)$ for all $i \in \{k+1, \dots, n\}$ since $\beta \geq 0$. Inequality (b) holds because $\beta \geq (M - \sigma^{-1}(\varepsilon/(n-k)))/(x_k - x_{k+1})$.

Regarding the evaluation of the map at the right endpoint of the interval, we compute

$$\begin{aligned}\sum_{i=1}^{n} \sigma(\beta x_i + m - \beta x_{k+1}) &= \sum_{i=1}^{k} \sigma(\beta(x_i - x_{k+1}) + m) + \sigma(m) + \sum_{i=k+2}^{n} \sigma(\beta(x_i - x_{k+1}) + m) \\ &\overset{(c)}{\geq} \sum_{i=1}^{k} \sigma(\beta(x_k - x_{k+1}) + m) + \varepsilon \\ &= k\sigma(\beta(x_k - x_{k+1}) + m) + \varepsilon \\ &\overset{(d)}{\geq} k\Big(1 - \frac{\varepsilon}{k}\Big) + \varepsilon = k,\end{aligned}$$

where inequality (c) holds because $\beta(x_i - x_{k+1}) \geq \beta(x_k - x_{k+1})$ for all $i \in \{1, \dots, k\}$ since $\beta \geq 0$, $\sigma(m) = \varepsilon$, and because $\sigma(z) \geq 0$ for all z. Inequality (d) holds because $\beta \geq \frac{\sigma^{-1}(1-\varepsilon/k)-m}{x_k - x_{k+1}}$. Therefore, we conclude that there exists a $\lambda^\star \in [M - \beta x_k, m - \beta x_{k+1}]$ satisfying condition (3.11). Since $\lambda^\star \geq M - \beta x_k$, by (3.10) we conclude that $y_i^\star \geq 1 - \varepsilon$ for $i \in \{1, \dots, k\}$ and since $\lambda^\star \leq m - \beta x_{k-1}$, we conclude that $y_i^\star \leq \varepsilon$ for $i \in \{k+1, \dots, n\}$. Since $\varepsilon > 0$ was arbitrarily small, the proof is completed. ☐

Proposition 3.5.2. For all $x \in^n$, $k, j \in \{1, \dots, n\}$, with $k \geq j$, the following inequality holds elementwise:

$$\mathrm{ssm}(x; k) \geq \mathrm{ssm}(x; j). \tag{3.12}$$

Proof. Recall from the proof of Proposition 3.5.1 that $\mathrm{ssm}(x; k) =: y^{\star,k}$ is given by $y^{\star,k} = \sigma(x + \lambda^{\star,k})$, where $\lambda^{\star,k}$ is the optimal dual variable and satisfies

$$\sum_{i=1}^{n} \sigma(x_i + \lambda^{\star,k}) = k.$$

Note, however that if $\lambda^{\star,k} \geq \lambda^{\star,j}$, then the result is proved in view of the formulas for $y^{\star,k}$ and $y^{\star,j}$ in view of monotonicity of σ. However, it is straightforward to see that $\lambda^{\star,k} \geq \lambda^{\star,j}$ since the mapping $\lambda \mapsto \sum_{i=1}^{n} \sigma(x_i + \lambda)$ is monotone and $k \geq j$. In other words, for the optimal dual variable to satisfy the constraint, it is necessary that $\lambda^{\star,k} \geq \lambda^{\star,j}$. Thus, the result is proved. $\square$

As a corollary to Proposition 3.5.2, we can see that for any $x \in^n$, ksoftmax$(x) \geq 0$, where the inequality is understood elementwise.

3.6 Energy Interpretation of Update Rule

Proposition 3.6.1. For any $k \in \{1, \ldots, n\}$, the following statements hold:

1. There exists $V_k :^n \to$ such that $\nabla V_k(x) = \text{ssm}(x; k)$. Indeed, one such V_k is given by

$$V_k(x) = \int_0^1 x^\top \text{ssm}(tx; k) dt. \tag{3.13}$$

2. The function V_k in equation (3.13) is convex and its Hessian, $\nabla^2 V_k(x)$, is a Laplacian matrix for all $x \in^n$.

Proof. To establish that ssm is the gradient of a potential function V_k, we show that its Jacobian is symmetric. To this end, leveraging the Lagrangian formulation from Proposition 3.5.1, we know that

$$\mathrm{ssm}(x;k)_i =: y_i^\star(x,\lambda^\star(x)) = \sigma(x_i+\lambda^\star(x)) \quad \text{and} \quad \sum_{i=1}^n \sigma(x_i+\lambda^\star(x)) = k, \tag{3.14}$$

where $\lambda^\star(x)$ denotes the optimal dual variable for the optimization problem. Computing the partial derivatives for $y_i^\star$ yields

$$\frac{\partial y_i^\star}{\partial x_j}(x,\lambda^\star(x)) = \sigma'(x_i+\lambda^\star(x))\Big(\delta_{ij}+\frac{\partial \lambda^\star}{\partial x_j}(x)\Big), \tag{3.15}$$

where δ_{ij} is the Kronecker delta. The evaluation of $\partial\lambda^\star/\partial x_j$ may be computed via the implicit function theorem:

$$\begin{aligned}
&\sum_{i=1}^n \sigma'(x_i+\lambda^\star(x))\Big(\delta_{ij}+\frac{\partial\lambda^\star(x)}{\partial x_j}\Big) = 0 \\
\iff \qquad & \frac{\partial\lambda^\star}{\partial x_j}(x) = -\frac{\sigma'(x_j+\lambda^\star(x))}{\sum_{\ell=1}^n \sigma'(x_\ell+\lambda^\star(x))}.
\end{aligned}$$

Substituting this expression into the equation (3.15) yields

$$\frac{\partial y_i^\star}{\partial x_j}(x,\lambda^\star(x)) = -\frac{\sigma'(x_i+\lambda^\star(x))\sigma'(x_j+\lambda^\star(x))}{\sum_{\ell=1}^n \sigma'(x_\ell+\lambda^\star(x))} + \delta_{ij}\sigma'(x_i+\lambda^\star(x_i)). \tag{3.16}$$

Since $\partial y_i^\star/\partial x_j = \partial y_j^\star/\partial x_i$ for all $i,j \in \{1,\dots,n\}$, we conclude that the Jacobian matrix of $y^\star$ is symmetric. The result is then implied by the Poincaré lemma from differential geometry [1, Theorem 6.4.14]. A direct calculation shows that with V_k defined in (3.13) satisfies $\nabla V_k(x) = \mathrm{ssm}(x;k)$ using the symmetry of partial derivatives.

To see that V_k is convex, we will establish that the Jacobian of $y^\star(x,\lambda^\star(x))$ is positive semidefinite for all x. For $i \in \{1,\dots,n\}$, we use the shorthand $a_i := \sigma'(x_i+\lambda^\star(x))$ and note that for all x and i, $a_i \in \,]0,1/4]$. Then we can see that the diagonal elements of the

Jacobian are positive since

$$\frac{\partial y_i^\star}{\partial x_i}(x, \lambda^\star(x)) = -\frac{a_i^2}{\sum_{j=1}^n a_j} + a_i = a_i\Big(1 - a_i/\sum_{j=1}^n a_j\Big) > 0.$$

We can also see that the off-diagonal elements are negative. Computing row sums of the Jacobian yields

$$\begin{aligned}\frac{\partial y_i^\star}{\partial x_i}(x, \lambda^\star(x)) + \sum_{j\neq i}\frac{\partial y_i^\star}{\partial x_j}(x, \lambda^\star(x)) &= a_i - \frac{a_i^2}{\sum_{\ell=1}^n a_\ell} - \sum_{j\neq i}\frac{a_i a_j}{\sum_{\ell=1}^n a_\ell} \\ &= a_i - \frac{a_i}{\sum_{\ell=1}^n a_\ell}(a_i + \sum_{j\neq i} a_j) = a_i - a_i = 0.\end{aligned}$$

Since the diagonal entries of $\frac{\partial y^\star}{\partial x}$ are positive, the off-diagonal entries are negative, and row sums are equal to zero, we conclude that for all x, the Jacobian matrix is a Laplacian matrix. An application of Gershgorin's circle theorem establishes positive semidefiniteness for all x, which proves convexity of V_k. □

With Proposition 3.6.1 in hand, we are ready to provide an energy interpretation to the k-Hopfield layer rule (3.4). To be more specific, since the rule (3.4) is a map from ^{n} to $^{n\times k}$, it cannot have an energy function that it is minimizing. Instead, we study the update rule one column at a time as follows:

$$x^+ = \Xi\,\mathrm{ksoftmax}_i(\beta\Xi^\top x), \tag{3.17}$$

where $i \in \{1, \ldots, k\}$ and $\mathrm{ksoftmax}_i :^n\to^n$ outputs the ith column of ksoftmax. In other word, in one step, the update rule (3.17) attempts to output the i-th closest memory to x. This update rule defines a map from ^{n} to itself. If $i \neq 1$, we consider the function $E :^n\to^n$

given by

$$E(x) = \frac{1}{2}x^\top x - \frac{1}{\beta}\left(\int_0^1 x^\top(\mathrm{ssm}(t\beta\Xi^\top x; i) - \mathrm{ssm}(t\beta\Xi^\top x; i-1))dt\right). \tag{3.18}$$

Then, the update rule (3.17) is a gradient descent update with unit step size. To see this fact, note that

$$\nabla E(x) = x - \Xi(\mathrm{ssm}(\beta\Xi^\top x; i) - \mathrm{ssm}(\beta\Xi^\top x); i-1)$$

and note that $\mathrm{ssm}(\beta\Xi^\top x; i) - \mathrm{ssm}(\beta\Xi^\top x; i-1) = \mathrm{ksoftmax}_i(\beta\Xi^\top x)$. A gradient descent algorithm on E with stepsize $\alpha > 0$ is

$$x^+ = x - \alpha\nabla E(x) = (1-\alpha)x + \alpha\Xi\,\mathrm{ksoftmax}_i(\beta\Xi^\top x).$$

Picking $\alpha = 1$ yields the stated update rule (3.17).

If instead, $i = 1$, then $\mathrm{ksoftmax}_1(\beta\Xi^\top x) = \mathrm{ssm}(\beta\Xi^\top x; 1)$ and the underlying energy function $E :^n \to^n$ is given by

$$E(x) = \frac{1}{2}x^\top x - \frac{1}{\beta}\int_0^1 x^\top \mathrm{ssm}(t\beta\Xi^\top x; 1)dt. \tag{3.19}$$

Although we do not present a closed-form expression for $V_k(x) = \int_0^1 x^\top \mathrm{ssm}(tx; k)dt$, we provide the following interpretation: since $\mathrm{ssm}(x; k)$ is a smooth approximation for the top-k operator, we believe its underlying potential function V_k is a smooth approximation for the k-max sum function

$$f_k(x) = \sum_{i=1}^{k} x_{[i]},$$

where we recall the notation $x_{[i]}$ denotes the i-th largest element of the vector x. Smooth approximations for k-max sums were studied in [81] and for $k = 1$, one such approximation is the log-sum-exp function, as was studied for MCHNs in [69]. We believe that different approximations can provide different architectures for associative memory systems.

3.7 Conclusion and Future Directions

We propose the k-Hopfield layer which can retrieve k-closest memories to an obscured input in a differentiable manner. We provide a mathematical treatment for the layer and provide an energy-based interpretation. The layer performs well when embedded in small vision transformers while requiring fewer parameters.

Future theoretical work will entail deciphering the provided energy function and convergence guarantees for the update rule. Future experimental work will entail testing on larger scale transformers.

Chapter 4

Learning Neural Contracting Dynamics: Extended Linearization and Global Guarantees

4.1 Introduction

Due to their representation power, deep neural networks have become a popular candidate for modeling continuous-time dynamical systems of the form

$$\dot{x} = \frac{dx}{dt} = f(x), \tag{4.1}$$

where $f(x)$ is an unknown (autonomous) vector field governing a dynamical process. Beyond approximating the vector field f, it is desirable to ensure that the learned vector field is well-behaved. In many robotic tasks like grasping and navigation, a well-behaved system should always reach a fixed endpoint. Ideally, a learned system will still stably reach the desired endpoint even when pushed away from demonstration trajectories. Additionally, the learned system should robustly reach the desired endpoint in the face of uncertainty. In other tasks, such as manufacturing, animation, and human-robot interaction, the learned system must smoothly follow a specific trajectory to its target. Following a trajectory is necessary to ensure function, as is the case with a manufacturing robot dynamically interacting with its environment, and safety, as is necessary with a robot occupying the same space as humans.

To enforce stability guarantees, a popular approach has been to ensure that the learned dynamics admit a unique equilibrium point and have a Lyapunov function, e.g. [63]. While popular, approaches based on Lyapunov functions typically struggle to provide general robustness guarantees since global asymptotic stability does not ensure robustness guarantees in the presence of disturbances. Indeed, input-to-state stability of global asymptotically stable dynamical systems needs to be separately established, e.g., see Chapter 5 in [47].

To ensure robustness and to allow for smooth trajectory following, there has been increased interest in learning contracting dynamics. A dynamical system is said to be contracting if any two trajectories converge to one another exponentially quickly with respect to some metric [60]. If a learned contracting system that admits a demonstration trajectory is pushed off that trajectory, it will follow a new trajectory that exponentially converges to the demonstration trajectory. Additionally, if a system is contracting, it is exponentially incrementally input-to-state stable [84]. If the system is autonomous, it also admits a unique equilibrium, is input-to-state stable, and has two Lyapunov functions that establish exponential stability of the equilibrium [14]. Since establishing contractivity globally requires satisfying a matrix partial differential inequality everywhere, prior works have focused on establishing contractivity in the neighborhood of training data.

4.1.1 Related Works

Stable, but not necessarily contracting dynamics. Numerous works have aimed to learn stable dynamical systems from data including [48, 63, 86, 70, 97, 12, 29]. In [48], the authors introduce the Stable Estimator of Dynamical Systems (SEDS), which leverages a Gaussian mixture model to approximate the dynamics and enforce asymptotic stability via constraints on learnable parameters. In [63], the authors jointly learn the dynamics and a Lyapunov function for the system and project the dynamics onto the set of dynamics which enforce an exponentially decay the Lyapunov function. This projection is done in closed-form and establishes global exponential convergence of the dynamics to the origin. In [86], the authors introduce Imitation Flow where trajectories are mapped to a latent space where states evolve in time according to a stable SDE. In [70], Euclideanizing Flows is introduced where the latent dynamics are enforced to follow natural gradient dynamics [2]. A similar approach is taken in [97] where additionally collision avoidance is considered. Finally, [12] and [29] enforce stability via Koopman operator theory whereby the nonlinear dynamics are lifted to a higher-dimensional space and are enforced to be linear and stable in this space.

Existing works on learning contracting dynamics. Learning contracting vector fields from demonstrations has attracted attention due to robustness guarantees [71, 76, 78, 84, 9]. In [71], the dynamics are defined using a Gaussian mixture model and the contraction metric is parametrized to be a symmetric matrix of polynomial functions. Convergence to an equilibrium is established using partial contraction theory [90]. In [76], the dynamics are defined via an optimization problem over a reproducing kernel Hilbert space; the dynamics are constrainted to be locally contracting around the data points, i.e., there are points in state space where the dynamics are not contracting. In [78] and [84], the authors study controlled dynamical systems of the form $\dot{x} = f(x, u)$, where u is a control input and train neural networks to find a feedback controller such that the closed-loop dynamics are approximately contracting, i.e., contractivity is not enforced but instead lack of contractivity is penalized in the cost function.

Recent work, [9], has proposed a Neural Contractive Dynamical System (NCDS) which learns a dynamical system which is explicitly contracting to a trajectory. NCDS parameterizes contracting vector fields by learning symmetric Jacobians and performing line integrals to evaluate the underlying vector field. NCDS is computationally costly because of this integration. Additionally, the Jacobian parametrization used is overly restrictive, because not all contracting vector fields have symmetric Jacobians. Constraining the vector field to have symmetric Jacobian is equivalent to enforcing that the vector field is a negative gradient flow and contracting with respect to the identity metric [91]. For scalability to higher dimensions, NCDS leverages a latent space structure where an encoder maps the data space to a lower-dimensional space and enforces contracting dynamics in this latent space. Then a decoder "projects" the latent-space dynamics to the full data space. It is then argued that on the submanifold defined by the image of the decoder, the dynamics are contracting.

4.1.2 Contributions

In this paper, we present a novel model for learning deep dynamics with global contracting guarantees. We refer to this model as Extended Linearized Contracting Dynamics (ELCD). To the best of our knowledge, ELCD is the first model to ensure global contraction guarantees. To facilitate the development of this model, we provide a review of contracting dynamics and extended linearization of nonlinear dynamics. Leveraging extended linearization, we factorize our vector field as $f(x) = A(x, x^*)(x - x^*)$, where x^* is the equilibrium of the dynamics. We enforce negative definiteness of the symmetric part of $A(x, x^*)$ everywhere and prove (i) global exponential stability of x^*, (ii) equilibrium contractivity of our dynamics to x^*, and (iii) using a converse contraction theorem, contractivity of the dynamics with respect to some metric.

Since negative definiteness of the symmetric part of A is not sufficient to capture all contracting dynamics, we introduce a latent space of equal dimension as the data space and learn diffeomorphisms between the data space and this latent space. The diffeomorphisms provide additional flexibility in the contraction metric and allow learning of more general contracting dynamics compared to those which are solely equilibrium contracting.

Our example in Section 4.3.4 provides theoretical justification for why the diffeomorphism and the learned contracting model must be trained jointly. In summary, if the diffeomorphism is trained first, and transforms the data to a latent space, the model class may not be expressive enough to accurately represent the true latent dynamics. If the diffeomorphism and dynamics are trained simultaneously, this limitation is overcome. This is in contrast to [9] which trains the diffeomorphism independently as a variational autoencoder and then trains the model after on the transformed data.

Our model, ELCD, directly improves on NCDS in several ways. We parameterize the vector field directly instead of parameterizing its Jacobian. Doing so prevents us from needing to integrate the Jacobian to calculate the vector field and thus speeds up training and inference. ELCD is also more expressive than NCDS because it can represent vector fields with asymmetric Jacobians. Additionally, ELCD is guaranteed to be contracting to an arbitrary equilibrium point, either selected or learned, at all training steps. NCDS, in contrast, must learn the equilibrium point over the course of training. Additionally, NCDS learns an encoder and decoder for a lower-dimensional latent space and thus can only be contracting on the submanifold defined by the image of the decoder. From initial conditions that are not on this submanifold, NCDS may not exhibit contracting behavior. In contrast, since the latent space of ELCD is of the same dimension as the data space, we train diffeomorphisms that ensure global contractivity in the data space. ELCD exhibits better performance than NCDS at reproducing trajectories from the LASA [57], n-link Pendulum, and Rosenbrock datasets. We additionally compare against the Euclideanizing Flow [70], and Stable Deep Dynamics [63] models.

4.2 Preliminaries

We consider the problem of learning a dynamical system $\dot{x} = f(x)$ using a neural network that ensures that the dynamics are contracting in some metric. Going forward, we denote by $Df(x) = \frac{\partial f}{\partial x}(x)$ the Jacobian of f evaluated at x. To this end, we define what it means for a dynamical system to be contracting.

The matrix measure $\mu(A)$ of a matrix $A \in^{n\times n}$ induced by norm $\|\cdot\|$ is given by $\mu(A) = \lim_{h\to o^+} \frac{\|I_n+hA\|-1}{h}$. The matrix measure induced by $\|\cdot\|_2$, in closed-form, is $\mu_2(A) = \lambda_{max}\left(\frac{A+A^\top}{2}\right)$ where λ_{max} is the largest eigenvalue. For the invertible $R \in^{n\times n}$, the weighted, L-p induced matrix measure is given by $\mu_{p,R} = \mu_p(RAR^{-1})$

Definition 4.2.1. A contracting dynamical system is one for which any two trajectories converge exponentially quickly. From [60], for a continuously differentiable map $f :^d \to^d$, the dynamical system $\dot{x} = f(x)$ is contracting with rate $c > 0$ if there exists a continuously-differentiable matrix-valued map $M :^d \to^{d\times d}$ and two constants $a_0, a_1 > 0$ such that for all $x \in^n$, $M(x) = M(x)^\top$ and $a_0 I_d \succeq M(x) \succeq a_1 I_d$ and additionally satisfies for all x

$$M(x)Df(x) + Df(x)^\top M(x) + \dot{M}(x) \preceq -2cM(x). \tag{4.2}$$

The map M is called the *contraction metric* and the notation $\dot{M}(x)$ is shorthand for the $n \times n$ matrix whose (i, j) entry is $\dot{M}(x)_{ij} = \nabla M_{ij}(x)^\top f(x)$. A central result in contraction theory is that any dynamical system $\dot{x} = f(x)$ satisfying (4.2) has any two trajectories converging to one another exponentially quickly [60, 62, 84]. Specifically, there exists $L \geq 1$ such that any two trajectories x_1, x_2 of the dynamical system satisfy

$$\|x_1(t) - x_2(t)\|_2 \leq L\mathrm{e}^{-ct}\|x_1(0) - x_2(0)\|. \tag{4.3}$$

In other words, contractivity establishes exponential *incremental* stability [5].

We note that any contracting dynamical system enjoys numerous useful properties including the existence of a unique exponentially stable equilibrium and exponential input-to-state stability when perturbed by a disturbance. We refer to [60] for more useful properties of contracting dynamical systems and [14] for a recent monograph on the subject.

The problem under consideration is as follows: given a demonstration $\mathcal{D} = \{x_i, \dot{x}_i\}$ consisting of a set of n state, $x_i \in^d$, and velocity, $\dot{x}_i \in^d$, pairs, we aim to learn a neural network $f(x_i) = \dot{x}_i$ that parameterizes a globally contracting dynamical system with equilibrium point x^*, such that $f(x^*) = 0$. In essence, this task requires learning both the vector field and the contraction metric, M such that they jointly satisfy (4.2).

4.2.1 Contracting Linear Systems

Suppose we assumed that the dynamical system we aimed to learn was linear, i.e., $\dot{x} = Ax$ for some matrix $A \in^{d\times d}$ and we wanted to find a contraction metric M which we postulate to be constant $M(x) := M$ for all x. The contraction condition (4.2) then reads

$$M(A + cI_n) + (A + cI_n)^\top M \preceq 0, \tag{4.4}$$

which implies that A has all eigenvalues with $\mathrm{Re}(\lambda(A)) \leq -c$, see Theorem 8.2 in [36]. In other words, for linear systems, contractivity is equivalent to stability.

Although the condition (4.4) is convex in M at fixed A, the task of learning both (A, M) simultaneously from data is not (jointly) convex. Instead, different methods must be employed such as alternating minimization. A similar argument is made in [63] in the context of stability and here we show that the same is true of contractivity.

4.2.2 Contractivity and Exponential Stability for Nonlinear Systems

In the case of nonlinear systems, contractivity is not equivalent to asymptotic stability. Indeed, asymptotic stability requires finding a continuously differentiable Lyapunov function $V :^d \rightarrow \mathbb{R}_{\geq 0}$ which is positive everywhere except at the equilibrium, x^*, and satisfies the decay condition

$$\nabla V(x)^\top f(x) < 0, \quad \forall x \in^d \setminus \{x^*\}. \tag{4.5}$$

One advantage of learning asymptotically stable dynamics compared to contracting ones is that the function V is scalar-valued, while the contraction metric M is matrix-valued. A disadvantage of learning asymptotically stable dynamics compared to contracting ones is that we are required to know the location of the equilibrium point beforehand and no robustness property of the learned dynamics is automatically enforced. To this end, there are works in the literature that enforce an equilibrium point at the origin and learn the dynamics and/or the Lyapunov function under this assumption [63].

In the case of contractivity, existence and uniqueness of an equilibrium point is implied by the condition (4.2) and it can be directly parametrized to best suit the data. At the same time, it is known that globally asymptotically stable dynamics do not enjoy the same robustness properties that contracting ones do, so we prefer to focus our attention to learning contracting dynamics.

4.3 Methods

4.3.1 Motivation

The motivation for our approach comes from the well-known mean-value theorem for vector-valued mappings, which we highlight here.

Lemma 4.3.1 (Mean-value theorem (Proposition 2.4.7 in [1])). Let $f :^d \to^d$ be continuously differentiable. Then for every $x, y \in^d$,

$$f(x) - f(y) = \Big(\int_0^1 Df(\tau x + (1-\tau)y) d\tau \Big)(x - y). \tag{4.6}$$

If $y = x^*$ satisfies $f(x^*) = 0$, then the continuously differentiable map f admits the factorization

$$f(x) = A(x, x^*)(x - x^*), \quad \text{where} \tag{4.7}$$

$$A(x, x^*) = \int_0^1 Df(\tau x + (1-\tau)x^*) d\tau. \tag{4.8}$$

This factorization is referred to as *extended linearization* in analogy to standard linearization, i.e., for x close to x^*, $f(x) \approx Df(x^*)(x - x^*)$. We remark that when $d \geq 2$, extended linearization is not unique, and for a given f, there may exist several A such that $f(x) = A(x, x^*)(x - x^*)$. In other words, (4.8) showcases one valid choice for A such that this factorization holds. Indeed, this nonuniqueness has been leveraged in some prior words, e.g. Section 3.1.3 in [84], to yield less conservative contractivity conditions.

Extended linearization is also sometimes referred to as apparent linearization and the state-dependent coefficient parametrization and has a rich history in nonlinear control theory, specifically in state-dependent linear-quadratic regulator (LQR) design; see the survey [98]. Moreover, it was recently demonstrated in [72] that enforcing this extended linearization structure during learning can be more data-efficient.

Since we know that a contracting vector field admits a unique equilibrium point, x^*, we restrict our attention to learning a matrix-valued mapping $A :^{d} \times^{d} \to^{d \times d}$ and ensuring that this mapping has enough structure so that the overall vector field satisfies the contraction condition (4.2) for some contraction metric M. This task is nontrivial since $f(x) = A(x, x^*)(x - x^*)$ implies that

$$Df(x) = A(x, x^*) + \frac{\partial A}{\partial x}(x, x^*)(x - x^*), \tag{4.9}$$

where $\frac{\partial A}{\partial x} :^{n} \times^{n} \to^{n \times n \times n}$ is a third-order tensor-valued mapping. In what follows, we will show that a more simple condition will imply contractivity of the vector field. Namely, negative definiteness of the symmetric part of $A(x, x^*)$ will be sufficient for contractivity of the dynamical system $\dot{x} = A(x, x^*)(x - x^*)$.

4.3.2 Parametrization of $A(x, x^*)$

We show a simple example of our model, the Extended Linearized contracting Dynamics (ELCD). Let $x \in^d$ be the state variable and $f :^d \to^d$ be a vector field with equilibrium point x^*. As indicated, we parametrize our vector field by its extended linearization

$$\dot{x} = f(x) = A(x, x^*)(x - x^*), \tag{4.10}$$

where now

$$\begin{aligned} A(x, x^*) = &-P_s(x, x^*)^\top P_s(x, x^*) \\ &+ P_a(x, x^*) - P_a(x, x^*)^\top - \alpha I_d \end{aligned} \tag{4.11}$$

$P_s, P_a :^d \times^d \to^{d \times d}$ are neural networks, I_d is the d-dimensional identity matrix, and $\alpha > 0$ is a constant scalar. Note that the symmetric part of $A(x, x^*)$ is negative definite since

$$\frac{A(x, x^*) + A(x, x^*)^\top}{2} = -P_s(x, x^*)^\top P_s(x, x^*) - \alpha I_d$$

and $P_s(x, x^*)^\top P_s(x, x^*)$ is guaranteed to be positive semidefinite.

We prove that a vector field parametrized this way is guaranteed to be (i) globally exponentially stable, (ii) equilibrium contracting as defined in [21], and (iii) contracting in some metric. The key tools are partial contraction theory [90] and a converse contraction theorem.

Theorem 4.3.2 (Equilibrium Contraction and Global Exponential Stability)**.** Suppose the dynamical system $\dot{x} = f(x)$ is parametrized as $f(x) = A(x, x^*)(x - x^*)$ where $A(x)$ is as in (4.11). Then any trajectory $x(t)$ of the dynamical system satisfies

$$\|x(t) - x^*\|_2 \leq e^{-\alpha t}\|x(0) - x^*\|_2. \tag{4.12}$$

Proof. We use the method of partial contraction as in [90]. To this end, define the virtual system

$$\dot{y} = A(x, x^*)(y - x^*). \tag{4.13}$$

We will establish that this virtual system is contracting in the identity metric, i.e., (4.2) is satisfied with $M(x) = I_d$. We see that the Jacobian for this virtual system is simply $A(x, x^*)$ and

$$\begin{aligned} A(x, x^*) + A(x, x^*)^\top &= -2P_s(x, x^*)^\top P_s(x, x^*) - 2\alpha I_d \\ &\preceq -2\alpha I_d. \end{aligned}$$

In other words, in view of (4.3),and since $M(x) = I_d$, any two trajectories $y_1(t)$ and $y_2(t)$ of the virtual system satisfy

$$\|y_1(t) - y_2(t)\|_2 \leq e^{-\alpha t}\|y_1(0) - y_2(0)\|_2.$$

Note that we can pick one trajectory to be $y_1(t) = x^*$ for all t and we can pick $y_2(t) = x(t)$, where $x(t)$ is a trajectory of the nominal system. This then establishes the claim. $\square$

Clearly, the bound (4.12) implies exponential convergence of trajectories of the dynamical system (4.10) to x^*. Moreover, this bound exactly characterizes equilibrium contractivity, as was defined in [21]. Notably, using the language of logarithmic norms, Theorem 33 in [21] establishes a similar result to Theorem 4.3.2 without invoking a virtual system.

Note that although Theorem 4.3.2 establishes global exponential stability and equilibrium contraction, it does not establish global contractivity. Indeed, the contractivity condition (4.2) is not guaranteed to hold with $M(x) = I_d$ without further assumptions on the structure of $A(x, x^*)$ in (4.10). Remarkably, however, due to a converse contraction theorem of Giesl, Hafstein, and Mehrabinezhad, it turns out that one can construct a state-dependent M such that the dynamics are contracting. Specifically, since trajectories of the dynamical system satisfy the bound (4.12), for any matrix-valued mapping $C :^d \to^{d\times d}$ with $C(x) = C(x)^\top \succ 0$ for all x, the matrix PDE

$$M(x)Df(x) + Df(x)^\top M(x) + \dot{M}(x) = -C(x) \tag{4.14}$$

admits a unique solution for each x [32]. In other words, the unique M solving this matrix PDE serves as the contraction metric and will satisfy (4.2) with suitable choice for c. The following proposition provides the explicit solution for M in terms of the solution to the matrix PDE.

Proposition 4.3.3 (Theorem 2.8 in [32]). Let $C :^d \to^{d\times d}$ be smooth and have $C(x) = C(x)^\top$ be a positive definite matrix at each x. Then $M :^d \to^{d\times d}$ given by the formula

$$M(x) = \int_0^\infty \psi(\tau, x)^\top C(\phi(\tau, x))\psi(\tau, x) d\tau, \tag{4.15}$$

is a contraction metric for the dynamical system (4.10) on any compact subset containing x^*, where $\tau \mapsto \phi(\tau, x)$ is the solution to the ODE with $\phi(0, x) = x$ and $\tau \mapsto \psi(\tau, x)$ is the matrix-valued solution to

$$\dot{Y} = Df(\phi(t,x))Y, \quad Y(0) = I_d. \tag{4.16}$$

While it is challenging, in general, to compute the metric (4.15), numerical considerations for approximating it arbitrarily closely are presented in [33]. In practice, one would select $C(x) = I_d$ and evaluate all integrals numerically.

We remark that our parametrization for A in (4.11) is similar to the parametrization for the Jacobian of f that was presented in [9]. There are however a few key differences. Notably, $A(x, x^*)$ may be asymmetric while the Jacobian in equation (3) in [9] is always symmetric. Notably, since the Jacobian in [9] is symmetric and negative definite, the vector field $\dot{x} = f(x)$ is a negative gradient flow, $\dot{x} = -\nabla V(x)$, for some strongly convex function V. On the contrary, our dynamics (4.10) can exhibit richer behaviors than negative gradient flows in view of the asymmetry in $A(x, x^*)$. Additionally, since the dynamics in [9] with their parametrization can be represented as $\dot{x} = -\nabla V(x)$ for some strongly convex V, it is guaranteed to be contracting in the identity metric, $M(x) = I_d$ [91]. On the other hand, our dynamics (4.10) is not necessarily contracting in the identity metric and instead is contracting in a more complex metric given in (4.15).

4.3.3 Latent Space Representation

Realistic dynamical systems and their flows including the handwritten trajectories found in the LASA dataset, are often highly-nonlinear and may not be represented in the form (4.10) or obey the bound (4.12). One solution to these challenges is to transform the system to a latent, possible lower-dimensional, space and learn an ELCD in the latent space.

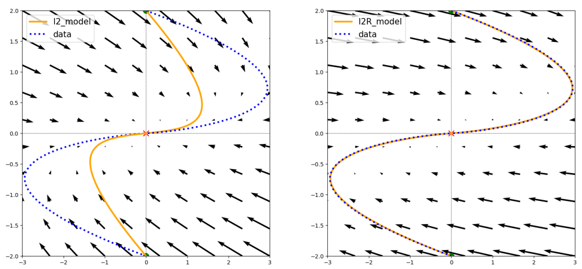
Latent space learning is possible because contraction is invariant under differential coordinate changes. From Theorem 2 of [62], given a dynamical system $\dot{x} = f(x)$, $f :^d \to^d$, with f satisfying (4.2.1), the system will also be contracting under the coordinate change $z = \phi(x)$ if $\phi :^d \to^d$ is a smooth diffeomorphism. Specifically, if $\dot{x} = f(x)$ is contracting with metric M, then the system that evolves z is contracting as well with metric given by $\tilde{M}(x) = D\phi(x)^{-\top} M(x) D\phi(x)^{-1}$, where $z = \phi(x)$ and $D\phi$ is the Jacobian of the coordinate transform. In other words, We can learn vector fields that are contracting in an arbitrary metric by training a vector field f which is parametrized as (4.10) and using a coordinate transform ϕ.

NCDS, [9], treats the coordinate transform as a Variational Autoencoder (VAE) [49]. Their training procedure consists of two steps: first training the coordinate transform with VAE training, then training the function f in the new learned coordinates.

4.3.4 On the Interdependence of Diffeomorphism and Learned Dynamics

To demonstrate the need for a coordinate transform, we consider the task of learning a vector field that fits trajectories generated by the linear system $\dot{x} = Ax$ with $A = \begin{pmatrix} -1 & 4 \\ 0 & -1 \end{pmatrix}$. Clearly, this linear system cannot be represented in the form (4.10) with parametrization (4.11). To see this fact, we observe that $A + A^\top$ is not negative definite and thus no choice of P_s or P_a can represent this linear system. To remedy this issue, one

Figure 4.1: The learned vector field and corresponding trajectories of an ELCD with no transform (left) and a model with a transform (right) when trained on data that is generated from a vector field that is contracting in a more general metric.



can take the linear coordinate transform $z = \phi(x) = Px$, where $P = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}$. Then the z-dynamics read

$$\dot{z} = PAP^{-1}z = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix} z \tag{4.17}$$

and now the symmetric part of PAP^{-1} is negative definite and thus we can find suitable choices of P_s, P_a. Specifically, we can take $\alpha \in (0, 1/2)$, let $P_a(z) = \begin{pmatrix} 0 & 0 \\ 1/2 & 0 \end{pmatrix} z$, and let $P_s(z) = Qz$ where Q is the matrix square root of $\begin{pmatrix} 1 & -1/2 \\ -1/2 & 1 \end{pmatrix} - \alpha I_2$. Note that in this toy example, asymmetry is essential to exactly represent these dynamics in the latent space, z. If we used the parametrization in [9], we would not be able to represent these dynamics since they have an asymmetric Jacobian. We demonstrate this routine in Figure (4.1). We generate two trajectories starting at $(0, 2)$ and $(0, -2)$, and use that data to train two ELCDs, one without and one with a learned, linear transform. Figure 4.1(a) shows the vector field and trajectories corresponding to an ELCD with no transform. The trajectories are forced to the center sooner than the actual data as a consequence of the bound (4.12). Learning a coordinate transform allows the ELCD to learn a system that is contracting in an arbitrary metric and exhibit overshoot. This is demonstrated by the learned system in Figure 4.1(b), which perfectly matches the data.

4.3.5 Choice of Diffeomorphism

There are several popular neural network diffeopmorphisms including coupling layers [23, 70], normalizing flows [86], $\mathcal{M}$-flow [13, 9], spline flows [27], and radial basis functions [10].

A coupling layer $\phi :^d \rightarrow^d$ consists of a neural network $\theta :^k \rightarrow^N$ with $1 < k < d$ and a neural network $g :^N \times \rightarrow$. The transform ϕ maps input $x \in^d$ to $y \in^d$ with the following procedure:

1. Set $y_i = x_i$ for $1 \leq i \leq k$ for some $1 < k < d$.

2. Set $y_i = g(x_{1:k}, x_i)$ for $k \leq i \leq d$

Coupling layers are invertible by doing the above process in reverse, so long as g is invertible. A coupling layer can exhibit a wide variety of behaviors depending on the choice of g. A simple example is the transform, which takes the form $g(x_{1:k}, x_i) = \alpha(x_{1:k}) * x_i + \beta(x_{1:k})$, where α and β are scalar functions. The affine g is clearly invertible.

Polynomial spline curves are another diffeomorphism that are commonly used as g in coupling layers. A polynomial spline has a restricted domain which is divided into bins. A different linear, quadratic [65], or cubic [26] polynomial covers each bin. [27] introduce rational-quadratic splines, which constructs a spline from functions that are the quotient of two quadratic polynomials.

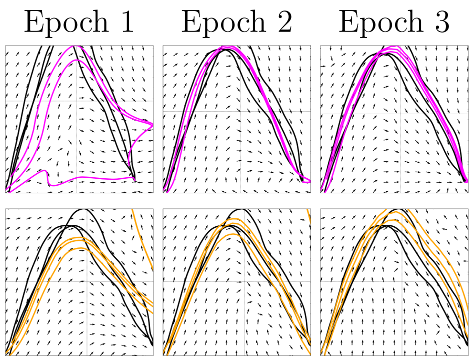
In practice, ϕ is the composition of several of transforms. Because the coupling transform only alters some coordinates, they are often used in conjunction with random permutations. If the latent space dimension is smaller than the data space dimension [9] makes the last composite function of ϕ an 'Unpad' function, which simply removes dimensions to match the latent space. The decoder ϕ is then prepended by a 'Pad' function, which concatenates the latent variable with the necessary number of 'zeros' to match the data space.

Of course, if the latent dimension is smaller than that of the data dimension, then ϕ can not be bijective. However, [9] argue that as long as ϕ is injective and has range over the dataset, then f being contracting in the latent space implies that the learned dynamics are contracting on the submanifold defined by the image of ϕ^{-1}. In other words, NCDS cannot be globally contracting. The consequences of this are shown in Figure (4.2). While NCDS can learn to admit an equilibrium point when on the submanifold, trajectories that fall off the submanifold may diverge. ELCD, in contrast, is contracting to the correct equilibrium point during all phases of training.

4.3.6 Training

Our task is to simultaneously learn the contracting system $f(x)$ and the metric in which f contracts. As previously discussed, the metric is implicitly determined by $\phi(x)$. [9] uses

Figure 4.2: Plots of the vector fields and induced trajectories learned by ELCD (Top) and NCDS (Bottom) after different training epochs. ELCD is contracting always while NCDS may admit multiple equilibrium or diverge.



a two-step method. First, they treat ϕ as a variational autoencoder and minimize the evidence lower bound (ELBO). They then fix the encoder and train the model to evolve the state in latent space. Note that the VAE objective is to make the encoded data to match a standard-normal distribution. Notably, VAE training does not encourage the encoder to transform the data to space in which the data is corresponds to trajectories of a contracting system. If the data is not contracting in the transformed space, a contracting model will not be able to fully fit the data. This explains why, in our implementation of the two step-training, we are unable to reasonably learn the data. For further comparison with NCDS, we will train the encoder and model jointly.

4.4 Comparisons to Other Models

In this section, we describe the models that we compare to, namely Eflow [70], SDD [63], and NCDS [9].

Euclideanizing Flow: Starting with Eflow, they parametrize their dynamical system by

$$\dot{x} = -G_\psi(x)^{-1}\nabla\Phi(\psi(x)), \tag{4.18}$$

where $\psi :^d \to^d$ is a diffeomorphism, $\Phi :^d \to$ is a convex potential function, and $G_\psi(x) = D\psi(x)^\top D\psi(x)$ is the induced Riemmanian metrc. The dynamics (4.18) then has the interpretation of being the steepest descent for $\Phi \circ \psi$ on the Riemmanian manifold defined by metric G_ψ. Specifically, in [70], the convex potential function is defined to be $\Phi(y) = \|y - y^\star\|_2$, where $y^\star = \psi(x^\star)$ and the diffeomorphism is parametrized via a coupling layer [23]. Note that contraction properties of dynamics of these form were studied in [91]. Since Ψ is chosen to be convex but not strongly convex, these dynamics are only *weakly contracting*, i.e., satisfy (4.2) with $c = 0$.

Stable Deep Dynamics: In [63], the authors parametrize both an unconstrained vector field $\hat{f} :^d \to^d$ and a Lyapunov function $V :^d \to \mathbb{R}_{\geq 0}$ via neural networks. Then to enforce global exponential stability, at every point $x \in^d$, they project $\hat{f}(x)$ onto the convex set of points $\{u \in^d \mid \nabla V(x)^\top u \leq -\alpha V(x)\}$. They do this projection in closed-form so that the dynamics have the representation

$$\dot{x} = \hat{f}(x) - \nabla V(x)\frac{\mathrm{ReLU}(\nabla V(x)^\top \hat{f}(x) + \alpha V(x))}{\|\nabla V(x)\|_2^2}. \tag{4.19}$$

Under a smart parametrization of V using input-convex neural networks [4], the authors guarantee global exponential stability of (4.19) to the origin with rate α. Note that the dynamics (4.19) are continuous, but not necessarily differentiable. Due to this potential nonsmoothness, it is theoretically unknown whether these dynamics are contracting (since Theorem 2.8 in [32] requires at least some differentiability).

Neural Contractive Dynamical Systems: In [9], the authors enforce contractivity by directly enforcing negative definiteness of the Jacobian matrix of the vector field by parametrizing

$$Df(x) = -(J_\theta(x)^\top J_\theta(x) + \epsilon I_d), \tag{4.20}$$

where $J_\theta :^d \rightarrow^{d\times d}$ is a neural network. Under this parametrization of the Jacobian, it is straightforward to see that (4.2) holds with $c = \epsilon$ and $M(x) = I_d$ for all x. To recover the vector field from the Jacobian, the fundamental theorem of calculus for line integrals (which can be seen as a version of the mean-value theorem) is utilized to express

$$\dot{x} = f(x) = f(x_0) + \int_0^1 Df((1-t)x_0 + tx)(x - x_0)dt, \tag{4.21}$$

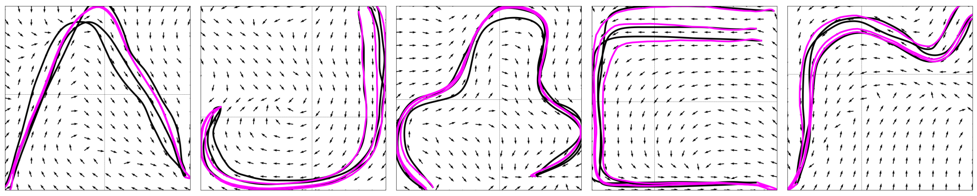
where x_0 and $f(x_0)$ denote the initial condition and its velocity, respectively.

To express richer dynamics, NCDS also consists of an encoder and decoder for a lower-dimensional latent space. The dynamics in the latent space are constrained to be (4.21) and the encoder and decoder are parametrized using a VAE. Since, generally, the latent space is of lower dimension than the data space, the resulting dynamics are only guaranteed to be contracting on the submanifold defined by the image of the decoder.

4.5 Experiments

For all datasets we compare our method against NCDS [9], Stable Deep Dynamics (SDD) [63] and Euclideanizing Flow (Eflow) [70]. For the encoder ϕ in ELCD, we use the composition of two coupling layers composed of rational-quadratic spline. The splines cover the range $\{-10, 10\}$ with 10 bins, and linearly extrapolated outside that range. The parameters of each spline are determined by residual networks, each containing two transform blocks with a hidden dimension of 30. A permutation and linear flow layer is placed before, in the middle, and after the two quadratic spline flow layers. For ELCD, we let the latent dimension size equal the data dimension size. P_a and P_s are implemented as two-layer neural networks with a hidden dimension of 16.

Figure 4.3: Plots of the 2D LASA data. Demonstrations are in black. The learned ELCD trajectories in magenta are plotted along with the learned vector field. The vector field velocities have been normalized for visibility.



As the authors of NCDS have not yet made their code publicly available, we implemented NCDS to the best of our abilities. We used a latent dimension of size 2 for all datasets. This is in-line with the description in [9], and necessary given the extra cost from integrating the Jacobian in higher-dimensional latent spaces. We use the same encoder ϕ as for ELCD with an added un-padding layer as the last function which removes the last $d-2$ dimensions.

All experiments are trained with a batch size of 100, for 100 epochs, with an Adam optimizer and learning rate of 1e-3. All computation is done on CPUs.

4.5.1 Metric

We evaluate trajectories by comparing them against their reference trajectory using the normalized dynamic time warping distance (DTWD) [46]. For two trajectories $x = \{x_i\}_{i\in\{1,\ldots,t_1\}}$, $\bar{x} = \{\bar{x}_i\}_{i\in\{1,\ldots,t_2\}}$ and some distance function $d(\cdot,\cdot)$, let DTWD be

$$\mathbf{DTWD}(x,\bar{x}) = \frac{1}{t_1}\sum_{i=1}^{t_1}\left(\min_{\bar{x}_j\in\bar{x}} d(x_i,\bar{x}_j)\right) + \frac{1}{t_2}\sum_{i=1}^{t_2}\left(\min_{x_j\in x} d(\bar{x}_i,x_j)\right)$$

4.5.2 Datasets

We experiment with the LASA dataset [57], which consists of 30, two-dimensional curves. We use three demonstration trajectories of each curve. As in [9], the first few initial points of each trajectory are omitted so that only the target state has zero velocity. we stack two and four LASA trajectories together to make datasets of 4 and 8-dimensional trajectories, respectively. All data is standardized to have a mean of zero and variance of one. We use in total 10 2D curves, 6 4D curves, and 6 8D curves. Some 2D-LASA trajectories and their respective trained ELCD trajectories and vector fields are visualized in Figure (4.3).

We also experiment with simulated datasets. We simulate 6 trajectories of a 2,4, and 8-link pendulum (4D, 8D, and 16D respectively) each and 4 trajectories of 8D and 16D Riemannian gradient descent dynamics on a generalization of the Rosenbrock function. Each model is trained on all trajectories of the same dimension, and then predictions are made starting from every initial point.

***n*-link pendulum:** The n-link pendulum is a simple mechanical system which is the interconnection of n rigid links under the force of gravity. We specifically assume that there is damping on each of these links, which makes the resulting dynamical system stable and almost all trajectories converge to the downright equilibrium point. In this case, the state of the pendulum is described by the n pairs $(\theta, \dot{\theta}_i)$, where θ is the angle of the i-th link and $\dot{\theta}_i$ is its angular velocity. Thus, this dynamical system is $2n$ dimensional.

As was done in [63], we adapt the code from `http://jakevdp.github.io/blog/2017/03/08/triple-pendulum-chaos/` to generate data for n links.

Rosenbrock datset: The classical Rosenbrock function, see [74]

$$f(x, y) = (1-x)^2 + 100(y-x^2)^2, \tag{4.22}$$

is a nonconvex function with global minimum at $(1, 1)$. Despite its apparent nonconvexity, it is known that a Riemannian gradient descent dynamics, is contracting with respect to a suitable Riemannian metric, see Example 3 in [91] for details. In optimization and in machine learning, the Rosenbrock function is a benchmark for the design of various optimizers and in the study of Riemannian optimization [45, 50].

For a higher-dimensional generalization of the Rosenbrock function, we consider

$$f(x) = \lambda_1(1-x_1)^2 + \sum_{i=2}^{n} \lambda_i (x_i - x_{i-1}^2)^2, \tag{4.23}$$

where $\lambda_i > 0$ are parameters that affect the conditioning of the function. This generalization is still nonconvex and admits several saddle points. Note that the classic Rosenbrock corresponds to $n = 2, \lambda_1 = 1, \lambda_2 = 100$. The unique global minimum of this generalization is at $(1, 1, \ldots, 1)$. The corresponding contracting Riemmanian gradient descent dynamics that find this global minimum are

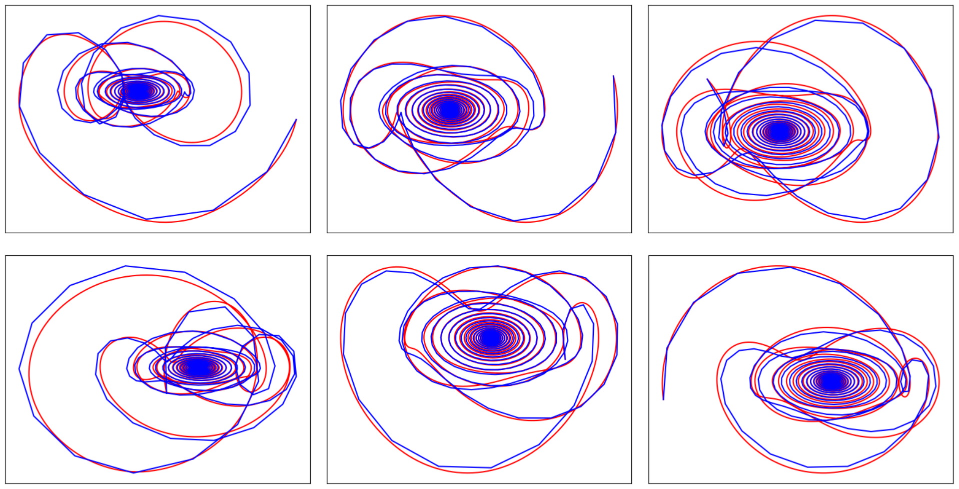
Table 4.1: DTWD on LASA, Pendulum, and Rosenbrock datasets

	SDD	EFlow	NCDS	ELCD
LASA-2D	0.37 ± 0.32	1.05 ± 0.25	0.59 ± 0.61	$\mathbf{0.12 \pm 0.11}$
LASA-4D	2.49 ± 2.4	2.24 ± 0.12	2.19 ± 1.23	$\mathbf{0.80 \pm 0.54}$
LASA-8D	5.26 ± 0.50	2.66 ± 0.63	5.04 ± 0.77	$\mathbf{1.52 \pm 0.61}$
Pendulum-4D	0.49 ± 0.11	0.17 ± 0.01	1.35 ± 2.26	$\mathbf{0.03 \pm 0.01}$
Pendulum-8D	0.75 ± 0.08	0.33 ± 0.01	2.88 ± 0.69	$\mathbf{0.14 \pm 0.03}$
Pendulum-16D	1.86 ± 0.14	0.45 ± 0.01	1.65 ± 0.31	$\mathbf{0.44 \pm 0.09}$
Rosenbrock-8D	NaN	1.90 ± 0.16	2.74 ± 0.15	$\mathbf{1.22 \pm 0.01}$
Rosenbrock-16D	NaN	3.57 ± 0.66	3.68 ± 0.12	$\mathbf{2.57 \pm 0.09}$

$$\dot{x} = -G(x)^{-1}\nabla f(x), \tag{4.24}$$

where $G(x) = D\psi(x)^\top D\psi(x)$, where $\psi :^n \to^n$ is the mapping $\psi(x) = (\sqrt{\lambda_1}(1-x_1), \sqrt{\lambda_2}(x_2 - x_1^2), \ldots, \sqrt{\lambda_n}(x_n - x_{n-1}^2))$. Note that these dynamics were designed via a generalization of the procedure proposed in [91]. To generate data, we choose four initial points and evolved them according to (4.24).

Figure 4.4: Phase plots of the two link-pendulum trajectories (blue) and the trajectories produced by ELCD (red). Each column is a different trajectory. The top row is the first link and the bottom row is the second link. The x-axis is angle and the y-axis is angular velocity.



4.5.3 Results

Table 4.1 presents our results. ELCD performs the best in all taskst. These results shows the benefit of the increased expressiveness allowed by the skew symmetric component of our parametrization. These benefits are more apparent in the pendulum dataset, whose oscillatory patterns inherently have Jacobians with complex eigenvalues, meaning they are not symmetric. A sample of demonstration and learned trajectories is shown in figure (4.4).

The Rosenbrock dynamics are stiff and difficult to learn. In our training of SDD, we observed lack of stability and convergence, resulting in very large DTWDs. Hence, we report NaN for the results of the SDD models on the Rosenbrock dataset. Our model performs the best, there is still room for improvement. Handling dynamics with multpile scales is still an open challenge.

4.5.4 Universal Representation

Our ELCD model equipped with a coupling layer-based transformation can represent any contracting dynamical system. It is known that any smooth nonlinear system with a hyperbolically stable fixed point can be exactly linearized inside its basin of attraction via a suitable diffeomorphism [55]. Our extended-linear parameterization is tasked with learning the stable linear part, while the coupling layers aim to learn and approximate this suitable diffeomorphism. As our ELCD model can universally approximate any linear model, we simply need our coupling layer to universally approximate all diffeomorphisms. Indeed, [80] proved that the coupling layers are universal approximators for diffeomorphisms.

4.6 Limitations

Our method assumes that the underlying dynamics of the data trajectories are contracting. However, the diffeomorphic transform allows for a wide class of stable systems to be represented. Our method is currently implemented for trajectories that converge to the same fixed point. But, this can be overcome by manually changing the fixed point, depending on which trajectory the input data came from. Also, right now our method is limited to dynamics in ^{n}, but we imagine that an extension to more general manifolds should be possible. It is also not obvious how to extend our method to the case of control. Perhaps, one could learn the extended linearization of a dynamical system, with no constraint, then jointly learn a controller and diffeomorphism such that the resulting closed-loop dynamical system is contracting.

4.7 Conclusions

In this paper, we introduce ELCD, the first neural network-based dynamical system with global contractivity guarantees in arbitrary metrics. The main theoretical tools are extended linearization, equilibrium contraction, and a converse contraction theorem. To allow for contraction with respect to more general metrics, we use a latent space representation with dimension of the latent space equal to the dimension of the data space and train diffeomorphisms between these spaces. We highlight key advantages of ELCD compared to NCDS as introduced in [9], including global contraction guarantees, expressible parameterization, and efficient inference. We demonstrate the performance of ELCD on high-dimensional LASA datasets and simulated Pendulum and Rosenbrock dynamics. Our method shows consistent performance across all datasets.

Acknowledgements: This work was supported in part by the NSF Graduate Research Fellowship under grant 2139319, the ARO under grant W911NF-22-1-0233, and by AFOSR award FA9550- 22-1-0059.

Chapter 5

Conclusion

The world is driven by complex dynamical systems, where interactions and processes unfold over time, often with intricate dependencies and feedback loops. It follows that intelligence and learning can be understood and analyzed through the lens of these complex systems. In this dissertation, we explored key challenges in machine learning scalability, the gap between computer and brain performance, and robotics using the framework of dynamical systems.

First, we addressed the issue of scalability by reinterpreting the k-means algorithm as a differential equation. This allowed us to apply the implicit function theorem to derive a memory-efficient quantization algorithm, demonstrating that viewing traditional machine learning methods through the lens of dynamical systems can lead to more efficient and scalable solutions.

We then expanded on a mathematical model for associative memory, drawing connections between this model and the modern self-attention mechanism. Implicitly differentiating through this model allowed us to integrate it in a transformer architecture, emulating multi-headed self-attention and emphasizing the deep connections between biological and artificial neural structures. This work helps bridge the gap between how biological systems store and retrieve information and how these processes can be emulated in artificial systems.

Finally, we demonstrated the training of a neural network capable of following trajectories in a stable and robust manner. This was achieved by constraining the structure of the neural network using contraction theory, resulting in a direct and efficient parameterization that ensures the desired stability properties. This approach underscores the importance of incorporating principles from dynamical systems into the design of neural networks for real-world applications, particularly in robotics.

As Moore's law tapers off and the gains from increased computing power plateau, the future of machine learning research will increasingly depend on a deeper mathematical understanding of learning and its inherent dynamism. This dissertation contributes to this ongoing effort by illustrating how the principles of dynamical systems can be leveraged to address some of the most pressing challenges in the field. By continuing to explore these connections, we can pave the way for more robust, scalable, and biologically inspired machine learning models.

Bibliography

[1] R. Abraham, J. E. Marsden, and T. S. Ratiu. *Manifolds, Tensor Analysis, and Applications*, volume 75 of *Applied Mathematical Sciences*. Springer, 2 edition, 1988.

[2] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[3] B. Amos, V. Koltun, and J. Z. Kolter. The limited multi-label projection layer. *arXiv preprint: 1906.08707*, 2019.

[4] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *International Conference on Machine Learning*, 2017.

[5] D. Angeli. A Lyapunov approach to incremental stability properties. *IEEE Trans. Autom. Control*, 47(3):410–421, 2002.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2015.

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] H. Beik-Mohammadi, S. Hauberg, G. Arvanitidis, N. Figueroa, G. Neumann, and L. Rozo. Neural contractive dynamical systems. In *International Conference on Learning Representations*, 2024.

[10] H. Beik-Mohammadi, S. Hauberg, G. Arvanitidis, G. Neumann, and L. Rozo. Reactive motion generation on learned Riemannian manifolds. *Int J Robotic Research*, 42(10):729–754, 2023.

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[12] P. Bevanda, M. Beier, S. Kerz, A. Lederer, S. Sosnowski, and S. Hirche. Diffeomorphically learning stable Koopman operators. *IEEE Control Systems Letters*, 6:3427–3432, 2022.

[13] J. Brehmer and K. Cranmer. Flows for simultaneous manifold learning and density estimation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 442–453. Curran Associates, Inc., 2020.

[14] F. Bullo. *Contraction Theory for Dynamical Systems.* Kindle Direct Publishing, 1.1 edition, 2023.

[15] T. F. Burns and T. Fukai. Simplicial Hopfield networks. In *International Conference on Learning Representations*, 2023.

[16] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

[17] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.

[18] Shangyu Chen, Wenya Wang, and Sinno Jialin Pan. Metaquant: Learning to quantize by learning to penetrate non-differentiable quantization. *neurips*, 32, 2019.

[19] Minsik Cho, Keivan Alizadeh-Vahid, Saurabh Adya, and Mohammad Rastegari. DKM: Differentiable k-means clustering layer for neural network compression. In *International Conference on Learning Representations*, 2022.

[20] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:6292–6299, 2019.

[21] A. Davydov, S. Jafarpour, and F. Bullo. Non-Euclidean contraction theory for robust nonlinear stability. *IEEE Trans. Autom. Control*, 67(12):6667–6681, 2022.

[22] M. Demircigil, J. Heusel, M. Lwe, Sven Upgang, and F. Vermet. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168:288299, 2017.

[23] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *International Conference on Learning Representations (ICLR)*, 2017. arXiv preprint arXiv:1605.08803.

[24] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[25] A. Dosovitskiy, A. Kolesnikov L. Beyer, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[26] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows, 2019.

[27] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[28] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021.

[29] F. Fan, B. Yi, D. Rye, G. Shi, and I. R. Manchester. Learning stable Koopman embeddings. In *American Control Conference*, pages 2742–2747, 2022.

[30] Jun Fang, Ali Shafiee, Hamzah Abdel-Aziz, David Thorsley, Georgios Georgiadis, and Joseph H Hassoun. Post-training piecewise linear quantization for deep neural networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 69–86. Springer, 2020.

[31] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Jfb: Jacobian-free backpropagation for implicit networks, 2021.

[32] P. Giesl, S. Hafstein, and I. Mehrabinezhad. Computation and verification of contraction metrics for exponentially stable equilibria. *Journal of Computational and Applied Mathematics*, 390:113332, 2021.

[33] P. Giesl, S. Hafstein, and I. Mehrabinezhad. Contraction metrics by numerical integration and quadrature: Uniform error estimate. In *20th International Conference on Informatics in Control, Automation and Robotics*, 2023.

[34] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] J. P. Hespanha. *Linear Systems Theory.* Princeton Univ Press, 2009.

[37] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[38] B. Hoover, Y. Liang, B. Pham, R. Panda, H. Strobelt, D. H. Chau, M. J. Zaki, and D. Krotov. Energy transformer. *arXiv preprint arXiv:2302.07253*, 2023.

[39] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc National Academy of Sciences*, 79(8):2554–2558, 1982.

[40] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proc National Academy of Sciences*, 81(10):3088–3092, 1984.

[41] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[42] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[43] G. Iatropoulos, J. Brea, and W. Gerstner. Kernel memory networks: A unifying framework for memory modeling. In *Advances in Neural Information Processing Systems*, 2022.

[44] Saber Jafarpour, Alexander Davydov, Anton Proskurnikov, and Francesco Bullo. Robust implicit networks via non-Euclidean contractions. *Advances in Neural Information Processing Systems*, 34:9857–9868, 2021.

[45] N. Jaquier and L. Rozo. High-dimensional Bayesian optimization via nested Riemannian manifolds. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[46] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[47] H. K. Khalil. *Nonlinear Systems.* Prentice Hall, 3 edition, 2002.

[48] S. Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957, 2011.

[49] D. P. Kingma and M. Welling. Auto-encoding variational Bayes, 2014.

[50] L. Kozachkov, P. M. Wensing, and J.-J. Slotine. Generalization as dynamical robustness–The role of Riemannian contraction in supervised learning. *Transactions on Machine Learning Research*, 2023.

[51] S. G. Krantz and H. R. Parks. *The Implicit Function Theorem: History, Theory, and Applications*. Birkhäuser, 2002.

[52] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[53] D. Krotov and J. J. Hopfield. Dense associative memory for pattern recognition. In *Advances in Neural Information Processing Systems*, 2016.

[54] D. Krotov and J. J. Hopfield. Large associative memory problem in neurobiology and machine learning. In *International Conference on Learning Representations*, 2021.

[55] Y. Lan and I. Mezi. Linearization in the large of nonlinear systems and Koopman operator spectrum. *Physica D: Nonlinear Phenomena*, 242(1):42–53, 2013.

[56] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[57] A. Lemme, Y. Meirovitch, M. Khansari-Zadeh, T. Flash, A. Billard, and J. J. Steil. Open-source benchmarking for learned reaching motion generation in robotics. *Paladyn, Journal of Behavioral Robotics*, 6(1):000010151520150002, 2015.

[58] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *CoRR*, abs/1711.11294, 2017.

[59] Y. Liu, E. Sangineto, W. Bi, N. Sebe, B. Lepri, and M. De Nadai. Efficient training of visual transformers with small datasets. In *Advances in Neural Information Processing Systems*, 2021.

[60] W. Lohmiller and J.-J. E. Slotine. On contraction analysis for non-linear systems. *Automatica*, 34(6):683–696, 1998.

[61] C. Malaviya, P. Ferreira, and A. F. T. Martins. Sparse and constrained attention for neural machine translation. *arXiv preprint arXiv:1805.08241*, 2018.

[62] I. R. Manchester and J.-J. E. Slotine. Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design. *IEEE Trans. Autom. Control*, 62(6):3046–3053, 2017.

[63] G. Manek and J. Z. Kolter. Learning stable deep dynamics models. In *Advances in Neural Information Processing Systems*, 2019.

[64] B. Millidge, T. Salvatori, Y. Song, T. Lukasiewicz, and R. Bogacz. Universal Hopfield networks: A general framework for single-shot associative memory models. In *International Conference on Machine Learning*, 2022.

[65] Thomas Mller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novk. Neural importance sampling, 2019.

[66] James O'Neill. An overview of neural network compression. *CoRR*, abs/2006.03669, 2020.

[67] Eunhyeok Park and Sungjoo Yoo. PROFIT: A novel training method for sub-4-bit mobilenet models. *CoRR*, abs/2008.04693, 2020.

[68] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[69] H. Ramsauer, B. Schäfl, J. Lehner, P. Seidl, M. Widrich, L. Gruber, M. Holzleitner, T. Adler, D. Kreil, M. K. Kopp, G. Klambauer, J. Brandstetter, and S. Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021.

[70] Muhammad Asif Rana, Anqi Li, Dieter Fox, Byron Boots, Fabio Ramos, and Nathan D. Ratliff. Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems. *CoRR*, abs/2005.13143, 2020.

[71] H. Ravichandar, I. Salehi, and A. Dani. Learning partially contracting dynamical systems from demonstrations. In *Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 369–378, 13–15 Nov 2017.

[72] S. M. Richards, J.-J. E. Slotine, N. Azizan, and M. Pavone. Learning control-oriented dynamical structure from data. In *International Conference on Machine Learning*, 2023.

[73] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. A comprehensive survey on model quantization for deep neural networks. *arXiv preprint arXiv:2205.07877*, 2022.

[74] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175184, 1960.

[75] M. E. Sander, J. Puigcerver, J. Djolonga, G. Peyré, and M. Blondel. Fast, differentiable and sparse top-k: A convex analysis perspective. In *International Conference on Machine Learning*, 2023.

[76] V. Sindhwani, S. Tu, and M. Khansari. Learning contracting vector fields for stable imitation learning, 2018. Arxiv e-print.

[77] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[78] D. Sun, S. Jha, and C. Fan. Learning certified control using contraction metric. In *Conference on Robot Learning*, volume 155, pages 1519–1539, 2021.

[79] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[80] Takeshi Teshima, Isao Ishikawa, Koichi Tojo, Kenta Oono, Masahiro Ikeda, and Masashi Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. *CoRR*, abs/2006.11469, 2020.

[81] M. J. Todd. On max-k-sums. *Mathematical Programming*, 171:489517, 2018.

[82] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2021.

[83] A. Trockman and J. Z. Kolter. Mimetic initialization of self-attention layers. In *International Conference on Machine Learning*, 2023.

[84] H. Tsukamoto, S.-J. Chung, and J.-J. E Slotine. Contraction theory for nonlinear stability analysis and learning-based control: A tutorial overview. *Annual Reviews in Control*, 52:135–169, 2021.

[85] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[86] Julen Urain, Michele Ginesi, Davide Tateo, and Jan Peters. Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows. *CoRR*, abs/2010.13129, 2020.

[87] Sunil Vadera and Salem Ameen. Methods for pruning deep neural networks. *IEEE Access*, 10:63280–63300, 2022.

[88] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[89] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: hardware-aware automated quantization. *CoRR*, abs/1811.08886, 2018.

[90] W. Wang and J. J. Slotine. On partial contraction analysis for coupled nonlinear oscillators. *Biological Cybernetics*, 92(1):38–53, 2005.

[91] P. M. Wensing and J.-J. E. Slotine. Beyond convexity — Contraction and global convergence of gradient descent. *PLoS One*, 15(8):1–29, 2020.

[92] Ezra Winston and J Zico Kolter. Monotone operator equilibrium networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[93] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. 2017.

[94] Junru Wu, Yue Wang, Zhenyu Wu, Zhangyang Wang, Ashok Veeraraghavan, and Yingyan Lin. Deep k-means: Re-training and parameter sharing with harder cluster assignments for compressing deep convolutions. *CoRR*, abs/1806.09228, 2018.

[95] Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister. Differentiable top-k with optimal transport. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[96] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[97] Weiming Zhi, Tin Lai, Lionel Ott, and Fabio Ramos. Diffeomorphic transforms for generalised imitation learning. In Roya Firoozi, Negar Mehr, Esen Yel, Rika Antonova, Jeannette Bohg, Mac Schwager, and Mykel Kochenderfer, editors, *Proceedings of The 4th Annual Learning for Dynamics and Control Conference*, volume 168 of *Proceedings of Machine Learning Research*, pages 508–519. PMLR, 23–24 Jun 2022.

[98] T. imen. State-dependent Riccati equation (SDRE) control: A survey. In *17th IFAC World Congress*, page 37613775, 2008.