

# Equilibrium Approaches to Modern Deep Learning

**Shaojie Bai**

April 2022  
CMU-ML-22-101

School of Computer Science  
Carnegie Mellon University  
Pittsburgh, PA 15213  
United States of America

## **Thesis Committee:**

J. Zico Kolter, <i>Chair</i>	CARNEGIE MELLON UNIVERSITY
Katerina Fragkiadaki	CARNEGIE MELLON UNIVERSITY
Ruslan Salakhutdinov	CARNEGIE MELLON UNIVERSITY
David Duvenaud	UNIVERSITY OF TORONTO
Vladlen Koltun	APPLE

*Submitted in partial fulfillment of the requirements  
for the degree of Doctor of Philosophy.*

Copyright © 2022 Shaojie Bai

This research was sponsored by Robert Bosch GmbH award 0087016732-PCR, a J.P. Morgan AI Ph.D. Fellowship, and a gift fund from Intel.

**Keywords:** Deep Learning, Machine Learning, Neural Network Architectures, Implicit Deep Layers, Fixed-point Solving Algorithms, Deep Equilibrium Models, Implicit Function Theorem, Sequence Modeling

*To my parents, mentors and friends, without whom this thesis would not be possible.*



## Abstract

Deep learning (DL) has become one of the most successful and widely-adopted methods in modern artificial intelligence. Accompanying these successes are also increasingly complex and costly architectural designs, at the foundation of which has been a core concept: *layers*. This thesis challenges this fundamental role of layers, and provides an in-depth introduction to a new, *layer-less* paradigm of deep learning that computes the output as the fixed point of a dynamical system: deep equilibrium (DEQ) models.

First, we introduce the general formulation of deep equilibrium models. We discuss how these models express “infinite-level” neural networks, decouple forward and backward passes, yet with the cost and design complexity of one traditional layer— even in some of the most competitive settings (e.g., language modeling, semantic segmentation, etc.).

Second, we further discuss the challenges and opportunities such an equilibrium approach poses. We show that the DEQ formulation reveals numerous new properties of deep learning that were *long buried* by the traditional layer-stacking scheme. Exploiting them allows us to train and deploy these new and lightweight equilibrium algorithms in ways that significantly complements the existing developments in deep learning, and enables us to improve results on multiple fronts at the state-of-the-art level (e.g., optical flow estimation).

The DEQ approach has already led to a new research area on *implicit deep learning* in the community (e.g., a NeurIPS 2020 tutorial), on both theoretical and empirical ends. We thus conclude this thesis by discussing how future work could further leverage this equilibrium perspective to build more scalable, efficient and accurate *next-generation* DL algorithms, including to scientific computing, which are often characterized by solutions to complex, high-dimensional dynamical systems.



## Acknowledgments

Looking back, my Ph.D. journey was full of coincidences, which constantly reminds me of how fortunate I was. There are a lot moments that I remember dearly, but just a few examples:

- At the beginning of my senior year at CMU, Brandon Amos and I were TAing for CMU’s Distributed Systems course. On the 9th floor of Gates-Hillman Center, he introduced the then-ML-newbie me to his then-advisor Zico Kolter for a senior undergraduate thesis.
- On the first day of my Ph.D., Zico introduced me to Vladlen Koltun to work together on my first-ever Ph.D. research project: temporal convolutional networks (TCNs) for sequence modeling. (It was a time when everyone still uses RNNs/LSTMs.) I still remember Vladlen saying at the end of our first joint meeting, “Shaojie, this is an unusually good project for a first-year Ph.D. student”. But this project still went way beyond the scope and impact of what I could ever hope for and was an immense encouragement to my Ph.D. research.
- When I was preparing for the camera-ready paper and code for our TrellisNet work (which is a TCN+RNN model) in ICLR 2019, I accidentally trained a 50-layer network and tested it with 70 layers, finding this "deeper version" works better. And 100 layers were even better. Zico asked: "What if we have infinite layers?" This accident eventually led to a new research area in deep learning, which this thesis describes.

But not for a single moment have I taken them for granted. During my Ph.D. I had the honor of receiving the support of so many great mentors, collaborators, friends and my family. I learned from so many people I have never met— e.g., in the form of email questions, paper reviews, and GitHub issues.

Among all of these people, the one single person that truly transformed me is my advisor, Zico Kolter. There is no word that can describe my gratitude, but as I have said to numerous people (especially new students): “Zico is the *best* kind of advisor I could have ever hoped for”, and I mean it sincerely. I have been constantly inspired by him not just in research (all the great ideas and insights), but also more generally as a *person* to look up to. Even though half of my Ph.D. overlaps with one of the most chaotic periods in the last decade (the COVID-19 pandemic), Zico’s compassion, enthusiasm, humor and intelligence have been a shining beacon in the dark. I learned not only how to do research, but also do *great* research; not only how to solve a problem, but also *identify* great problems; not only how to work hard, but also *lead* and mentor others. If the last 5 years were how I transformed from a young padawan to a (junior) Jedi, then Zico is my master Yoda!

This thesis is also impossible without the help of my long-time collaborator Vladlen Koltun, whose incredible vision penetrated my entire 5 years of Ph.D. study. In many ways, I have considered Vladlen my “unofficial co-advisor”: who met with me weekly, wrote papers with me, brainstormed with me, and inspired me. My research style was heavily influenced by him, and Vladlen was a witness to all of the

most important research breakthroughs I have had.

I also couldn't imagine what my Ph.D. would be like without all of the internships that I have had. I worked on bridging temporal convolutions and recurrent networks at Intel Labs (2018) with Vladlen, won a global competition (with over 2,700 teams) on predicting molecular properties at BCAI (2019), and learned about improving self-attention mechanism with information granularity control at Facebook AI Research (2020). I am forever grateful for the support and enlightenment received from the great internship mentors I have had: Devin Willmott, Mordechai Kornbluth, Jonathan Mailoa, Michael Auli, Alexei Baevski, Jiatao Gu and Zhouhan Lin.

As a junior researcher, teaching assistant and collaborator, I have enjoyed working with people from around the world in different settings. This includes Brandon Amos, Cem Anil, Philippe Ciuciu, David Duvenaud, Katerina Fragkiadaki, Zhengyang Geng, Roger Grosse, Swaminathan Gurumurthy, Zhichun Huang, Paul Pu Liang, Zhouchen Lin, Zac Manchester, Florian Mannel, Thomas Moreau, Louis-Philippe Morency, Ashwini Pople, Zaccharie Ramzi, Pradeep Ravikumar, Ruslan Salakhutdinov, Yash Savani, Jean-Luc Starck, Yao-Hung (Hubert) Tsai, Makoto Yamada and Xin-Yu Zhang. I am also truly grateful for the great research environment at Carnegie Mellon University (CMU), and all the great colleagues, including Siddharth Ancha, Sebastian Caldas, Devendra Chaplot, Amanda Coston, Saurabh Garg, Chirag Gupta, William Guss, Tim Hsieh, Lisa Lee, Bingbin Liu, Leqi Liu, Biswajit Paria, Adarsh Prasad, Otilia Stretcu, Arun Sai Suggala, Che-Ping Tsai and Chih-Kuan Yeh. I feel incredibly fortunate that the research field I am in love with happens to be (arguably) the most popular research subject in this past decade.

I have also witnessed the unbelievable expansion of the Locus Lab, and am immensely thankful to be able to working alongside with its past and current members: Victor Akinwande, Alnur Ali, Christina Baek, Anna Bair, Filipe de Avila Belbute-Peres, Jeremy Cohen, Jonathan Dinu, Priya Donti, Rizal Fathony, Zhili Feng, Sachin Goyal, Yiding Jiang, Chun Kai Ling, Pratyush Maini, Gaurav Manek, Vaishnavh Nagarajan, Leslie Rice, Mel Roderick, Dylan Sam, Suvansh Sanjeev, Samuel Sokota, Mingjie Sun, Asher Trockman, Po-Wei Wang, Josh Williams, Ezra Winston, Eric Wong, Runtian Zhai and Huan Zhang. I want to thank the administrative staff at SCS, Diane Stidle and Catherine Coeptas, who have been extremely patient and helpful.

Finally, on the personal side, I have always been blessed by the strong support that my family and closest friends have given me: my parents Chunwang Bai and Qihua Du, and friends Yibo Cai, Yutong Chen, Yicheng Fang, Kai Kang, Hongyu Li, Yuxiong Lin, Jing Mao, Shiyao Qu, Jingyi Wu, Yi (Arlene) Wu, Mei Xue, Mengmeng Yang, Zhixian Ye and Yijia (Eka) Zhao. As I'm constantly climbing to the next level and reaching new heights in my life, they have always had my back, bringing me faith at the time of doubt, happiness at the time of dejection, and warmth at the time of hopeless homesickness.

John Archibald Wheeler (Richard Feynman's advisor) once said, "We live on an island surrounded by a sea of ignorance. As our island of knowledge grows, so does the shore of our ignorance." As I embark on a new journey, I know this completion of my doctoral thesis is only the start of my endless exploration of the unknown.







# Contents

- 1 Introduction 1**
  - 1.1 Implicit Perspectives on Deep Learning . . . . . 2
  - 1.2 Our Contributions . . . . . 5
    - 1.2.1 Other Contributions . . . . . 7
  
- I Equilibrium Approach Fundamentals: Infinite Layers to One Layer 9**
  
- 2 Deep Equilibrium Models 11**
  - 2.1 Preliminary: Infinite Layers to a Single Layer . . . . . 12
  - 2.2 Deep Equilibrium Models . . . . . 14
    - 2.2.1 Forward Pass . . . . . 14
    - 2.2.2 Backward Pass . . . . . 15
    - 2.2.3 Universality of Expressivity . . . . . 17
  - 2.3 Instantiations of DEQ . . . . . 18
  - 2.4 Experiments . . . . . 19
    - 2.4.1 Copy Memory Task . . . . . 19
    - 2.4.2 Large-Scale Language Modeling . . . . . 19
  - 2.5 Discussion . . . . . 23
  
- 3 Simultaneous Equilibrium: Modeling Hierarchy Without Layers 25**
  - 3.1 Multiscale Deep Equilibrium Models . . . . . 27
  - 3.2 Integration with Other Deep Learning Techniques . . . . . 28
  - 3.3 Experiments . . . . . 30
    - 3.3.1 Comparing with Prior Implicit Models on CIFAR-10 . . . . . 30
    - 3.3.2 ImageNet Classification . . . . . 31
    - 3.3.3 Cityscapes Semantic Segmentation . . . . . 32
    - 3.3.4 Runtime and Memory Consumption . . . . . 32
    - 3.3.5 Equilibrium Convergence on High-resolution Inputs . . . . . 33
  - 3.4 Discussion . . . . . 34

<b>II</b>	<b>Challenges and Potential Solutions to the Equilibrium Approach</b>	<b>35</b>
<b>4</b>	<b>Challenges to the DEQ Approach</b>	<b>37</b>
4.1	Related Work on Regularizing Implicit Models . . . . .	38
4.2	The Equilibrium Models' Challenges and Discontents . . . . .	39
4.2.1	Existence and Uniqueness of the Fixed Point . . . . .	39
4.2.2	Growing Instability . . . . .	39
4.2.3	Inefficiency Compared to Explicit Networks . . . . .	40
4.2.4	Brittleness to Architectural Choices . . . . .	41
4.2.5	Hidden Cost: Choice of Solver . . . . .	41
4.2.6	Physical Laws of Layer Structure . . . . .	42
4.3	Stabilizing DEQ Models by Jacobian Regularization . . . . .	43
4.4	Experiments . . . . .	45
4.4.1	Visualization with Synthetic Data . . . . .	45
4.4.2	WikiText-103 Language Modeling . . . . .	46
4.4.3	CIFAR-10 and ImageNet Classification . . . . .	47
4.4.4	Effect of Jacobian Regularization on $\rho(J_{f_\theta})$ . . . . .	49
4.4.5	Ablative Analysis and Limitations of the Approach . . . . .	49
4.5	Discussion . . . . .	50
<b>5</b>	<b>Training DEQs with Lightweight Inexact Gradients</b>	<b>51</b>
5.1	Motivation: Inexact Gradients . . . . .	52
5.2	Instantiations: Phantom Gradients . . . . .	54
5.3	Other Inexact Gradients . . . . .	56
5.3.1	SHaring the INverse Estimate (SHINE) from the Forward Pass . . . . .	56
5.3.2	Jacobian-Free Estimation . . . . .	57
5.4	Experiments . . . . .	57
5.5	Discussion . . . . .	60
<b>6</b>	<b>Neural Deep Equilibrium Solvers</b>	<b>63</b>
6.1	Preliminaries: Learning to Optimize, Fixed-point Solvers . . . . .	64
6.2	Neural Deep Equilibrium Solvers . . . . .	65
6.2.1	General Formulation . . . . .	66
6.2.2	Training the Neural Equilibrium Solvers . . . . .	67
6.2.3	Discussion . . . . .	69
6.3	Experiments . . . . .	70
6.3.1	Large-scale Experiments on Vision and Language Tasks . . . . .	71
6.3.2	Training Efficiency of the Neural Solver . . . . .	72
6.3.3	Ablative Studies and Limitations . . . . .	73
6.3.4	Caveats . . . . .	74
6.4	Discussion . . . . .	75

<b>III</b>	<b>Extensions of the Deep Equilibrium Models</b>	<b>77</b>
<b>7</b>	<b>Building Fast and Cheap Deep Equilibrium Optical Flow Estimators</b>	<b>79</b>
7.1	Related Work on Iterative Optical Flow . . . . .	81
7.2	Method . . . . .	81
7.2.1	Preliminaries . . . . .	82
7.2.2	Deep Equilibrium Flow Estimator . . . . .	82
7.2.3	Accelerating DEQ Flows . . . . .	84
7.3	Experiments . . . . .	87
7.3.1	Results . . . . .	88
7.3.2	Performance-Compute Tradeoff . . . . .	89
7.3.3	Ablation Study . . . . .	90
7.3.4	Qualitative Results . . . . .	92
7.4	Limitations . . . . .	92
7.5	Discussion . . . . .	93
<b>8</b>	<b>Implicit<sup>2</sup>: Implicit Models for Implicit Neural Representations</b>	<b>95</b>
8.1	Preliminaries: Implicit Neural Representations (INR) . . . . .	96
8.2	(Implicit) <sup>2</sup> Networks . . . . .	97
8.2.1	(Implicit) <sup>2</sup> Network Architectures . . . . .	98
8.2.2	Accelerated Training of (Implicit) <sup>2</sup> Networks . . . . .	99
8.3	Experiments . . . . .	101
8.3.1	Image Representation . . . . .	101
8.3.2	Image Generalization . . . . .	102
8.3.3	Audio Representation . . . . .	104
8.3.4	Video Representation . . . . .	104
8.3.5	3D Geometry Representation . . . . .	104
8.4	Limitations . . . . .	105
8.5	Discussion . . . . .	106
<b>IV</b>	<b>Conclusion</b>	<b>107</b>
<b>9</b>	<b>A Different Form of Deep Learning</b>	<b>109</b>
9.1	Summary . . . . .	110
9.2	Thoughts and Future Direction . . . . .	111
9.2.1	Are DEQ Models Less “Biological”? . . . . .	112
9.2.2	Community-Wide Effort . . . . .	112
9.2.3	Future . . . . .	113
	<b>Bibliography</b>	<b>115</b>



# List of Figures

- 1.1 Imagine an autonomous vehicle. As it receives and processes streaming camera frames, *each frame* needs to go through the exact same deep network computation graph (say 20 layers). However, the inputs are highly-correlated and almost identical. The layer-based deep learning is causing the model to repeat the same amount of work over and over again. Images from the Cityscapes dataset. . . . . 2
- 1.2 Conventional deep neural networks vs. an implicit deep equilibrium (DEQ) model. A deep equilibrium model defines an underlying dynamical system, and could take any solver path (e.g., Newton, quasi-Newton, etc.) leading to the fixed-point. 5
- 2.1 The behavior of hidden states over infinite unrolling: the activations in TrellisNet and Universal Transformers on sequence inputs with different lengths. The  $y$ -axis denotes  $\|f_\theta(\mathbf{z}) - \mathbf{z}\|$ ; i.e., the change in hidden units as depth increases. Expectedly, longer sequences contain more information and take longer to converge. For Transformers, the hidden unit convergence stops at around 100 layers and the fixed-point oscillation starts. . . . . 13
- 2.2 Comparison of the DEQ with conventional weight-tied deep networks. . . . . 17
- 2.3 A visualization of the results presented in Table 2.3. The equilibrium formulations of temporal convolutional or self-attention layers (i.e., DEQ-TrellisNet and DEQ-Transformer) perform as competitively as conventional deep networks of similar sizes. . . . . 22
- 2.5 DEQs can be accelerated by early stopping, but poorer estimates also hurt performance. . . . . 22
- 2.4 Left: number of Broyden iterations in forward and backward passes gradually grows with epochs. Right: DEQ-Transformer finds the equilibrium in a stable and efficient manner (whereas the deep transformer could oscillate around the fixed point, even when one exists). . . . . 23
- 3.1 The structure of a multiscale deep equilibrium model (MDEQ). *All components* of the model are shown in this figure. MDEQ consists of a transformation  $f_\theta$  that is driven to equilibrium. Features at different scales coexist side by side and are driven to equilibrium simultaneously. . . . . 26
- 3.2 The residual block used in MDEQ. An MDEQ contains only *one* such layer. . . . 27
- 3.3 All resolutions of MDEQ converge *simultaneously*. Larger scale index means higher resolution. . . . . 28

3.4	A visual comparison of MDEQ with prior implicit models and with standard explicit models in computer vision. Equilibrium states at multiple resolutions enable MDEQ to incorporate supervision in different forms. . . . .	29
3.5	Left: test accuracy as a function of training epochs. Right: MDEQ-Small and ANODEs correspond to the settings and results reported in Table 3.1. For all metrics, lower is better. . . . .	31
3.6	Examples of MDEQ-large segmentation results on the Cityscapes dataset. . . . .	32
3.7	Plots of MDEQ’s convergence to equilibrium (measured by $\frac{\ z^{[i+1]} - z^{[i]}\ }{\ z^{[i]}\ }$ ) as a function of the number of times we evaluate $f_\theta$ . As input image resolution grows (from CIFAR-10 to Cityscapes), MDEQ takes more steps to converge with (L-)Broyden’s method. Standard deviation is calculated on 5 randomly selected batches from each dataset. . . . .	33
4.1	Visualizations of DEQ models’ instability and inefficiency problems. “Ours” refers to the regularized DEQ models, which will be introduced in Sec. 4.3. . . . .	40
4.2	Pre- vs. post-LN DEQ-Transformer layer . . . . .	41
4.3	Comparing different architectural modifications of a DEQ-Transformer (first 60K steps). DEQ models are brittle: even slight modifications such as changing the whereabouts of LayerNorm (see Fig. 4.2) or removing weight normalization can cause the model to quickly diverge during training. . . . .	42
4.4	<b>Left:</b> when the slope is less than 1, even the simplest iterative application of $f_\theta$ converges. <b>Right:</b> when slope $> 1$ , the iterative approach may diverge or oscillate, but the fixed point still exists and can be solved for. . . . .	44
4.5	<b>Top:</b> the surface of the $f_\theta(\mathbf{z}; \mathbf{x})$ layer, and the eventual learned equilibria $z^*(x)$ as a function of $x$ (the <b>red</b> dashed line; compare this with the shape of Fig. 4.6). As $\gamma$ grows, the surface is “lifted up” and becomes flat in the $z$ -direction. <b>Bottom:</b> each unique input $x$ defines a slice of the surface, and we perform fixed-point solving on this slice; larger $\gamma$ values flatten the curve and significantly accelerate the convergence to equilibrium. . . . .	46
4.6	Target function $h(x)$ . . . . .	46
4.7	With the proposed regularization, DEQ models are competitive with popular explicit networks in accuracy, memory, and runtime. <b>Lower bars are better.</b> . . . .	48
4.8	Empirical evidence of how our method constrains $\rho(J_{f_\theta})$ . In contrast, insufficient NFEs (e.g., $T=16$ ) at training time cause a DEQ-Transformer model to explode early in the training phase. . . . .	49
4.9	Adding weight decay to the DEQ-Transformer doesn’t help stabilize the convergence. . . . .	50
5.1	Cosine similarity between the phantom and exact gradient in the synthetic setting. As $k$ increases for both UPG and NPG, the cosine similarity also rapidly approaches the true gradient. . . . .	58
5.2	Cosine similarity between the phantom gradient and the exact gradient on CIFAR-10 during training. The horizontal axis corresponds to the cosine similarity, and the vertical axis to the training iterations (steps). . . . .	58



6.1	Pareto curves of the same DEQ with different solvers on WikiText-103 language modeling (on 1 GPU).	63
6.2	6.2a: The canonical Anderson solver is based on a local least-squares solution at each iteration, with $\beta = \beta_k$ set to a constant. 6.2b: Our neural fixed-point solver provides a <i>better initial guess</i> $\mathbf{z}^{[0]}$ and <i>learnable iterative updates</i> .	68
6.3	Visualization of DEQ-Transformer (for WikiText-103 language modeling) initializer and HyperAnderson iterations.	69
6.4	The training procedure of the neural deep equilibrium solver. With a given $f_\theta$ and input $\mathbf{x}$ , we optimize the hypersolver parameters $\omega = \{\phi, \xi\}$ via losses applied on the HyperAnderson iterations and the initializer (see Sec. 6.2.2).	69
6.5	6.5a- 6.5c: Comparisons of DEQs with classic and neural solvers. All speed/accuracy curves within the same plot are benchmarked on the same GPU with the same experimental setting, averaged over 6 independent runs. 6.5d: The overhead of DEQ hypersolver is extremely small, generally requiring only $< 1.2\%$ the (unsupervised) training time and $< 4\%$ the size when compared to the original DEQ models on these large-scale tasks.	71
6.6	Left: Convergence analysis of the hypersolver at inference time. Right: Although trained with a frozen $f_\theta$ , the neural equilibrium solver can also be used to accelerate DEQ training.	73
6.8	Ablations on HyperDEQ (reg.).	73
6.7	Further ablations on the neural solver design (in terms of the $\alpha$ prediction and loss components. Note that we use an unregularized DEQ here (in contrast to Fig. 6.8) to better demonstrate the curve differences, which could otherwise sometimes be small.)	74
7.1	<b>A DEQ flow estimator directly models the flow as a path-independent fixed-point solving process.</b> We propose to use this implicit framework to replace the existing recurrent approach to optical flow estimation. The DEQ flows converge faster, require less memory, are often more accurate, and are compatible with prior model designs (e.g., RAFT and GMA).	80
7.2	A visual comparison of the DEQ flow estimator and the recurrent unrolled flow estimator. After the correlation and context modules (see Sec. 7.2.1), a DEQ flow uses a fast, black-box fixed-point solver (e.g., Anderson) to directly solve for a stable (fixed-point) flow $\mathbf{z}^* = (\mathbf{h}^*, \mathbf{f}^*)$ , and differentiate through $\mathbf{z}^*$ with a cheap inexact gradient. This makes a DEQ flow’s backward pass almost free. In contrast, a recurrent flow estimator has to be unrolled for many steps, and needs to perform BPTT, which is costly in both computation and memory.	82
7.3	(Left) By reusing fixed-point $\mathbf{z}^*$ from the previous frame’s flow estimation, we can “jump start” the subsequent equilibrium solving, essentially amortizing the solver cost and speeding up convergence. (Right) Comparing forward convergence of DEQ and recurrent flow estimators on Sintel videos (50 frames). "DS" stands for deep supervision used by RAFT. DEQ flow with fixed-point reuse converges best; and overall, DEQ flows converge faster than RAFT.	86

7.4	Comparing the training memory, inference speed and performance on Sintel (clean) with image size $436 \times 1024$ . The same model design (based on RAFT) consumes much less memory and computes much quicker than the recurrent counterpart. All results are benchmarked on a single Quadro RTX 8000 GPU. . . .	89
7.5	Performance and convergence stability (measured by absolute residual error) of the DEQ flow. Frequency indicates how many correction terms we pick, with 0 meaning no correction. We also compare with Jacobian regularization in Fig. 7.6. DEQ flows trained with our proposed correction enjoy superior performance and stability. . . . .	90
7.6	<b>Comparison of IFT, Jacobian Regularization and Fixed-Point Correction.</b> Given a limited forward solver budget, the fixed-point correction protocol successfully stabilizes training and shows accelerated fixed-point convergence. . . .	90
7.7	Visualizations of DEQ models’ instability and inefficiency problems. “Ours” refers to the regularized DEQ models, which will be introduced in Sec. 4.3. . . . .	91
7.8	Visualization on the Sintel test set, <code>ambush_1</code> sequence of the clean split. . . .	92
7.9	Visualization on the Sintel test set, <code>bamboo_3</code> sequence of the final split. . . .	92
7.10	Visualization on the Sintel test set, <code>temple_1</code> sequence of the final split. . . .	92
8.1	(a) Explicit networks incur $O(L)$ complexity in both time and memory during training. (b) Original DEQ models requires constant memory, yet they take $O(M)$ time in forward/backward passes, where $M$ is the number of fixed-point solver steps. (c) In contrast, an (Implicit) <sup>2</sup> Network consumes constant amortized memory <i>and</i> time in both forward and backward passes. . . . .	98
8.2	The architecture of one implicit layer $f_\theta(z; x)$ in iMFN and iSIREN. Note that when $z = 0$ , evaluating $f_\theta(z; x)$ once is equivalent to the output of the respective explicit layer. . . . .	99
8.3	Zero initialization <i>v.s.</i> fixed-point reuse in terms of required steps to converge. Further details of this experiment can be found in Sec.8.3.1. . . . .	99
8.4	Training PSNR comparison between different levels of gradient approximation. $T$ denotes truncation length of the inexact gradient (where we use NPG; see Chapter 5), where $T = \infty$ indicates full IFT. . . . .	100
8.5	Comparison of step time, memory consumption and PSNR between explicit and implicit models trained on the $512 \times 512$ grayscale image. . . . .	102
8.6	Learned $512 \times 512$ grayscale image . . . . .	102
8.7	Samples of best performing explicit/implicit models learned on <i>Natural</i> . . . .	103
8.8	Audio signals represented using Fourier-MFNs. Lower error magnitudes (in light blue) are better. . . . .	103
8.9	Normal maps and IoUs of fitted <i>dragon</i> object using Fourier-MFNs in Occupancy Network. The (Implicit) <sup>2</sup> approach performs better than its deep explicit counterpart. . . . .	105

9.1 A simple comparison between conventional explicit deep networks and the implicit equilibrium approach presented in this thesis. These deep equilibrium (DEQ) models do not define a prescribed computation trajectory, but can instead rely on any black-box solvers and be differentiated using only the final output alone (and are thus extremely memory efficient). . . . . 110



# List of Tables

2.1	DEQ achieves strong performance on the long-range copy-memory task. . . . .	19
2.2	DEQ achieves competitive performance on word-level Penn Treebank language modeling (on par with SOTA results, without fine-tuning steps). <sup>†</sup> The memory footprints are benchmarked (for fairness) on input sequence length 150 and batch size 15, which does not reflect the actual hyperparameters used; the values also do <i>not</i> include the memory for word embeddings. . . . .	20
2.3	DEQ-based models are competitive with SOTA deep networks of the same model size on the WikiText-103 corpus, with significantly less memory. <sup>†</sup> See Table 2.2 for more details on the memory benchmarking. Transformer-XL models are not weight-tied, unless specified otherwise. A visualization of this table is also shown in Fig. 2.3. . . . .	21
2.4	Runtime ratios between DEQs and corresponding deep networks at training and inference ( $> 1\times$ implies DEQ is slower). The ratios are benchmarked on WikiText-103. . . . .	21
3.1	Evaluation on CIFAR-10. Standard deviations are calculated on 5 runs. . . . .	30
3.2	Evaluation on ImageNet classification with top-1 and top-5 accuracy. MDEQs were trained for 100 epochs. . . . .	31
3.3	Evaluation on Cityscapes <code>val</code> segmentation. “*” marks the current SOTA. Higher mIoU (mean Intersection over Union) is better. . . . .	31
4.1	Evaluation on WikiText-103. PPL stands for Perplexity. $t_{\text{train}}$ stands for relative training time. <b>JR</b> stands for Jacobian regularization. Memory benchmarked on batch size 15 and excludes the embedding layer at training time. <sup>†</sup> indicates unregularized model hard-stopped at inference time (while still trained with more NFEs). . . . .	47
4.2	Results on CIFAR-10 and ImageNet classification (standard deviation is calculated with 5 runs). <sup>†</sup> indicates unregularized model hard-stopped at inference time. . . . .	48
4.3	Controlled experiments on the strength $\gamma$ of the Jacobian regularization. The NFE value represents the “hard stop” threshold we set for the corresponding DEQ models at inference. . . . .	50
5.1	Complexity comparison. Mem means the memory cost, and $K$ and $k$ denote the solver’s steps and the unrolling/Neumann steps, respectively. Here, $K \gg k \approx 1$ . . . . .	59

5.2	Experiments using DEQ-Transformer and MDEQ on large-scale vision and language tasks. Metrics stand for accuracy(%) $\uparrow$ for image classification on CIFAR-10 and ImageNet, and perplexity $\downarrow$ for language modeling on Wikitext-103. JR stands for Jacobian Regularization (see Chapter 4). $^{\dagger}$ indicates additional steps in the forward equilibrium solver. . . . .	60
5.3	Experiments using implicit graph neural networks on graph tasks. Metrics stand for accuracy(%) $\uparrow$ for graph classification on COX2 and PROTEINS, Micro-F1(%) $\uparrow$ for node classification task on PPI. . . . .	61
6.1	Perplexity (ppl) on WikiText-103 . . . . .	73
7.1	<b>Evaluation on Sintel and KITTI 2015 datasets.</b> We report the Average End Point Error (AEPE) and the F1-all measure for the KITTI dataset (lower is better). “C+T” refers to results that are pre-trained on the Chairs and Things datasets. “S+K+H” refers to methods that are fine-tuned on the Sintel, KITTI, and HD1K datasets. The bold font stands for the best result and the underlined results ranks 2nd. $^{\dagger}$ corresponds to using a 3-step phantom gradient (see Chapter 5). DEQ flow sets SOTA results even w/o attention. . . . .	88
8.1	PSNR (in dB) for all models on image generalization. The reported mean $\pm$ std is taken over the individual PSNR of the 16 images. . . . .	103
8.2	PSNR for the video representation task. The reported mean $\pm$ std is taken over all frames of the video. . . . .	104

# Chapter 1

## Introduction

The past decade has witnessed an explosive rate of growth in the research and development of modern deep learning methods. Central to almost all of them (and perhaps the deep learning universe so far), however, is a *key* concept and basic unit that no model architect can avoid: *layers*. Specifically, the deep models are built by stacking many layers together, which creates a gigantic architecture designed to fit some particular tasks. For example, deep convolutional networks are made up of several convolutional layers and other non-linear or regularizational components like ReLU [175], normalizations [13, 110, 246] and dropout [214]. These components are then connected in multiple ways (e.g., ResNets [96], U-Nets [195]) to extract feature maps, typically following a complicated schedule (e.g., when to downsample/upsample, how many stages, and which layers in each stage). Meanwhile, different kinds of layer designs have emerged, like the multi-head self-attention [233], and graph layers [124, 202]. The most famous AI applications in the past few years, such as high-resolution image synthesis [118], protein structure prediction [117] and text generation [32], all contain hundreds, thousands or more of these basic units.

On a high level, such a *layer-based* view regard deep networks as a huge computation graph, with a prescribed, detailed instructions on how we compute the output from input (like a calculator). However, this creates numerous challenges. First, it is often the model architects' responsibility to construct, as a hyperparameter, the depth and connectivity of a deep network. This quickly adds up complexity to the design, use and testing of these models, especially as they grow large [96, 220, 233, 250]. Second, these networks all rely on an algorithm called gradient backpropagation [89, 197] to train. This entails deep networks to memorize all intermediate layer activations in the forward pass to traverse the computation graph in reverse [46]. This frequently creates a memory footprint bottleneck, since the memory consumption will grow quickly with the architecture depth and reach the hardware limits. Third, such layer composition makes deep networks rather inelastic, as they have to perform the same amount of computation regardless of the complexity of the input (e.g., see Fig. 1.1). For example, we cannot simply skip a layer (unless we add *more layers* that help us control this, like SkipNet [238]), as any such arbitrary removal would make the model function differently from how it was trained.

Despite these disadvantages, layers have been deeply stacked regardless and considered indispensable to modern deep learning for the following reasons:

- **Expressivity**. It was long believed that composing many layers lays the foundation for modeling complex input-output mappings (which are frequently non-linear) [89, 102].

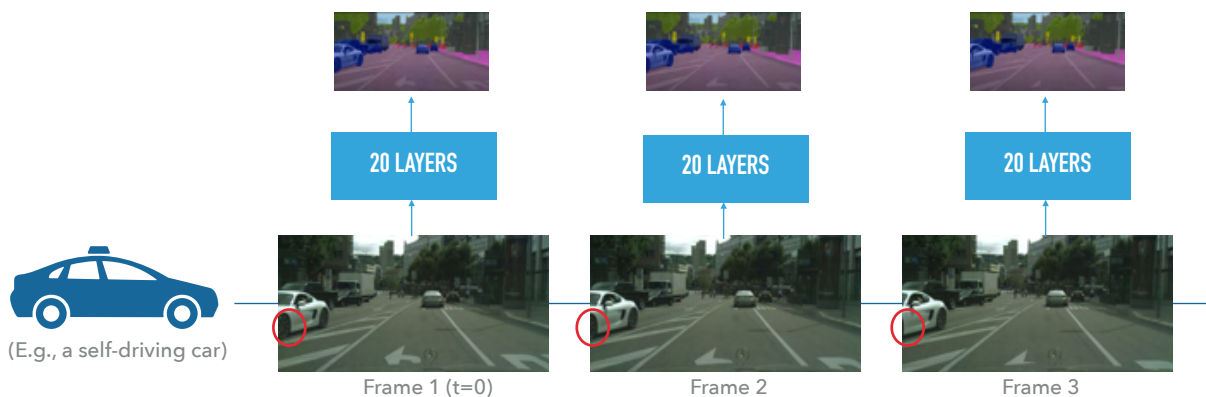


Figure 1.1: Imagine an autonomous vehicle. As it receives and processes streaming camera frames, *each frame* needs to go through the exact same deep network computation graph (say 20 layers). However, the inputs are highly-correlated and almost identical. The layer-based deep learning is causing the model to repeat the same amount of work over and over again. Images from the Cityscapes [53] dataset.

- **Feature hierarchy.** A common view is that layers represent *resolutions*. For example, Lee et al. [136] hypothesized that different levels extract different abstractions of an image.
- **Scalability.** To build large-scale models, we rely on the capability of flexibly connecting a lot of layers; very deep training has been shown to be feasible with techniques like normalizations [13, 110], residual connection [96], etc.

This thesis aims to revisit this fundamental concept of *layers*. A key question that we tackle is the following: ***Do we even need layers at all?***

We propose a new, implicit, *layer-less* approach to deep learning, dubbed the *deep equilibrium (DEQ) models*. Through this equilibrium approach, we essentially present a yet different way of doing deep learning, and how these deep networks can be built and analyzed as *algorithms* (rather than *calculators*). These DEQ models represent infinitely deep neural networks, but with only a single layer that is modeled implicitly (defined slightly later). Such *implicitness* in deep learning, we show, allows us to keep the three aforementioned properties (expressivity, feature hierarchy and scalability in real-world settings) even without layers, while correcting the major drawbacks (e.g., memory footprint) that traditional DL suffers from.

For the rest of this chapter, we first elaborate on what “implicitness” means in a deep learning context, as well as an overview of the prior work related to this direction. Then, we provide a general roadmap of this thesis, which we hope will serve as a blueprint for the past, present and future of deep implicit layers.

## 1.1 Implicit Perspectives on Deep Learning

In this section, we provide a brief survey of the past related works on implicit and continuous perspectives of deep learning methods. As shall be introduced in Chapter 2, the DEQ model can



be viewed as an infinitely deep network, but also a *single*-layer network, with the caveat that this layer is defined implicitly: given input  $\mathbf{x}$  and a (usually parameterized) function  $F$ , the output  $\mathbf{z}^*$  is defined as the value which solves some non-linear equation, i.e.,

$$\mathbf{z}^* = \text{Find } \mathbf{z}^* \text{ such that } F(\mathbf{z}^*; \mathbf{x}) = 0 \quad (1.1)$$

In particular, implicit modeling of hidden states has been explored by the deep learning community for decades, especially in the recurrent network context. Pineda [186] and Almeida [4], for example, studied implicit differentiation techniques for training recurrent dynamics, also known as recurrent back-propagation (RBP). In these cases, the RNNs are structured (e.g., via Lyapunov functions) so that their inference stage is a provably convergent dynamical system, and one needs to solve for the steady state of an RNN sequence (which absorbs the same input at each time step). Following these works, Liao et al. [144] also extend the RBP theory to stabler and more efficient variants based on Neumann series and conjugate gradients (and primarily studied their relationship with truncated backpropagation-through-time (TBPTT) in these RNNs). Zhang et al. [263] similarly also enforce fixed point conditions within RNN architectures. However, these RNNs could only be applied to the extremely limited settings where the same input comes in at every time step, which is rarely the case in practice (e.g., textual data, time-series, etc.).

The implicit approaches to network design have recently attracted renewed interest in very different forms and contexts. Amos and Kolter [6], Gould et al. [90], Johnson et al. [116] all propose to differentiate through an optimization problem (i.e., the arg min operator), thus treating Eq. (1.1) as optimality (e.g., KKT) conditions. Amos and Kolter [6], for instance, proposes to solve a quadratic program (QP) in each individual layer of a deep network; e.g., given hidden state  $\mathbf{z}^{[i]}$  from the previous layer, the layer  $i + 1$  of an OptNet [6] computes the following QP:

$$\mathbf{z}^{[i+1]} = \arg \min_{\mathbf{z}} \frac{1}{2} \mathbf{z}^\top Q(\mathbf{z}^{[i]}) \mathbf{z} + q(\mathbf{z}^{[i]})^\top \mathbf{z} \quad (1.2)$$

$$\text{subject to } A(\mathbf{z}^{[i]}) \mathbf{z} = b(\mathbf{z}^{[i]}) \quad (1.3)$$

$$G(\mathbf{z}^{[i]}) \mathbf{z} \leq h(\mathbf{z}^{[i]}) \quad (1.4)$$

where  $\mathbf{z}^{[i]}$  is the optimization (hidden) variable, and  $Q \succeq 0, q, A, b, G, h$  are parameters that define this QP optimization layer. Differentiation through this layer immediately follows from Eq. (1.1) (when  $Q \succ 0$ ) as we can differentiate through its KKT equation  $K(\mathbf{z}^*, \nu^*, \lambda^*) = 0$ , where  $\nu, \lambda$  are Lagrangian dual variables that correspond to constraints (1.3) and (1.4). With a similar spirit, Wang et al. [237] embeds a optimization-based layer for logical structure learning; de Avila Belbute-Peres et al. [60], Qiao et al. [188] uses these more structured layers to build differentiable physics engines (e.g., one can simulate constrained rigid body dynamics as a linear complementarity problem (LCP) [52, 54] layer in a deep autoencoder network [60]). El Ghaoui et al. [69] looks at such implicit layers in a broad well-posed sense and focuses on training small models via Lagrangian methods. These optimization layers are usually embedded as a specialized layer in a conventional deep architecture, customized for a specific problem domain, and whose strong structural assumptions (e.g., QP [6]) significantly limit their expressivity and scalability.

Another related thread of work formulates Eq. (1.1) to capture differential equations, thus representing a continuous deep neural network. This perspective was first theoretically studied by LeCun et al. [133], with later works propose to interpret ResNet [96] architectures as

discretizations of ordinary differential equation (ODE) solvers to exploit their reversibility and architectural variants. More recently, this interpretation is significantly advanced by the Neural ODE approach [45], which directly uses black-box ODE solvers and adjoint methods for direct differentiation through an ODE solution (and hence, integration with auto-differentiation packages). Specifically, a Neural ODE solves the following initial value problem (IVP) of the hidden state  $\mathbf{z}$ :

$$\frac{\partial \mathbf{z}(t)}{\partial t} = f_{\theta}(\mathbf{z}(t), t), \quad \mathbf{z}(0) = \mathbf{x} \quad (1.5)$$

where  $f_{\theta}$  is a parameterized layer that can take flexible forms, and  $\mathbf{z}^* = \mathbf{z}(T) = \int_0^T f_{\theta}(\mathbf{z}(t), t) dt$  (i.e., computing this continuous network amounts to integrate this layer from  $t = 0$  to  $T$ ). Equivalently, these ODEs admit an implicit general solution  $F(\mathbf{x}, \mathbf{z}^*, T) = 0$ . This Neural ODE formulation has since been improved [67, 121] and successfully applied in many settings, such as fluid dynamics [35] and continuous generative modeling [91]. However, due to the inherent challenge in solving high-dimensional ODEs, these methods are not yet efficient [67, 77, 121] or scalable to more realistic domains (e.g., one needs about 100 ODE solver iterations just for CIFAR-10  $32 \times 32$  image classification).

The work we present in this thesis takes a new approach to Eq. (1.1). Whereas characterizing it as optimality conditions yields optimization-based layers; and differential equations yields Neural ODEs; we introduce a fixed-point equation formulation that yields “infinite-layer” equilibrium feature states; i.e., for a layer  $f_{\theta}$ ,  $F(\mathbf{z}^*, \mathbf{x}) = f_{\theta}(\mathbf{z}^*; \mathbf{x}) - \mathbf{z}^* = 0$ . We hence call the resulting algorithm *deep equilibrium models*. With such formulation, we show that DEQ models:

1. Use exactly **one such standalone implicit layer**  $f_{\theta}$  as the *entire* architecture (in contrast to the traditional layer stacking);
2. **Perform competitively**, or even better, on numerous realistic tasks, such as language modeling, image classification, semantic segmentation, implicit neural representations, and optical flow estimation;
3. Reveal numerous **new properties** (e.g., fixed-point recycle) that were *long buried* by conventional deep learning that allow us to compute deep networks in both memory-wise and computationally efficient ways.

The deep equilibrium models, we show, exemplify a fully implicit deep learning architecture that, unlike these prior exploration, has one and only one implicit layer as the entire architecture, and works on the most competitive level and scale. Fig. 1.2 contrasts the conventional (explicit) deep learning which stacks a huge number of operators, and the implicit DEQ models which solve an underlying dynamical system to model the output.

This thesis provides the first in-depth analysis of the benefits, applications, extensions and challenges that this new approach faces. We demonstrate the significant improvements DEQ models bring to implicit modeling’s *performance, scalability, efficiency, flexibility, representational capacity*, and more. Our research on implicitness suggests a way for us to model infinitely complex concepts (e.g., that of a fixed point) via finite computations, and the *layer-less* approach posits an exciting new paradigm of deep learning computations.

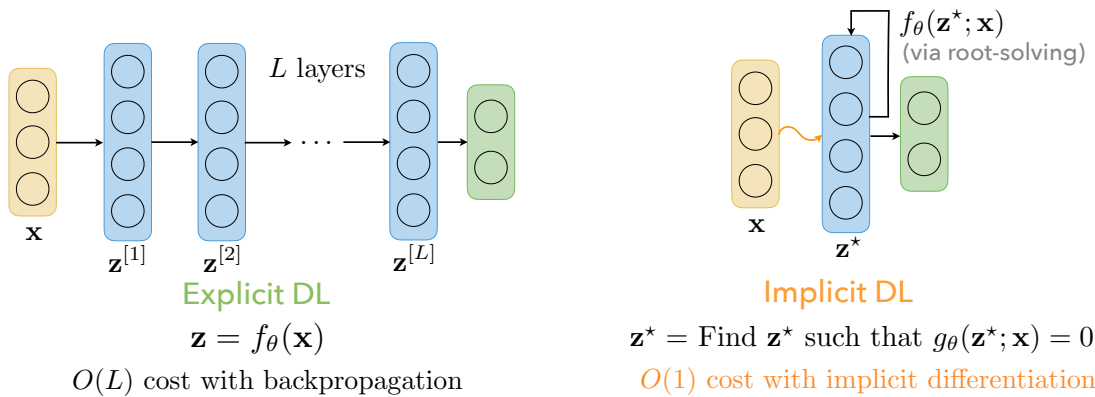


Figure 1.2: Conventional deep neural networks vs. an implicit deep equilibrium (DEQ) model. A deep equilibrium model defines an underlying dynamical system, and could take any solver path (e.g., Newton, quasi-Newton, etc.) leading to the fixed-point.

## 1.2 Our Contributions

In **Part I** of this thesis, we will discuss multiple **findings and motivations that culminate in the fundamentals of deep equilibrium models (i.e., their generic formulations)**. While implicitness has been previously leveraged in deep learning as optimization-driven transformations (Section 1.1), we start from the very successes (and premises) of existing deep learning and their trend: very deep—potentially *infinite-layer*—neural networks. More specifically:

1. In **Chapter 2**, we will derive how infinitely deep neural networks could be represented by an equilibrium network that computes the fixed-point of a layer. We will provide arguments for the universality of this one-layer approach, and (importantly) how such fixed-point computation could be differentiated through *directly* at the final output using implicit function theorem (IFT). We further discuss the implication this has on the forward and backward passes of DL training (which shall be heavily exploited in **Chapter 5, 6 and 7**).
2. Given the general formulation of DEQ models, we will show in **Chapter 2 and 3** how they subsumes a wide range of modern and complex layer designs (e.g., multi-head self-attention [233]) in large-scale realistic settings. In **Chapter 3**, we study how to enable the equilibrium networks, which forego a deep sequence of layers, to be able to represent feature hierarchy. We will expand upon the DEQ construction in **Chapter 2** substantially to introduce simultaneous equilibrium modeling; i.e., we directly optimize for stable representations on all feature scales at the same time, and provide natural interfaces for auxiliary losses and compound training procedures.

Therefore, the first part of this thesis will put a lot of emphasis on the representational power of these equilibrium approaches and their fundamental differences from the traditional deep learning. A powerful message that we hope to send is, “one layer is all you need”.

In **Part II** of this thesis, we discuss more in-depth the implications of such implicit perspective of deep learning. As these equilibrium networks decouple the forward and backward passes of the training process (i.e., one can train the model even just with the final output), we will demonstrate

that this leads to several new challenges and opportunities that traditional neural networks have not faced before. In particular:

1. **Chapter 4** will start by discussing some **novel problems introduced by the equilibrium approach** that do *not* exist in conventional deep learning methods, such as convergence stability and the choice (and cost) of the solver. We will provide a number of empirical evidence that reflects how DEQ models could turn increasingly unstable (i.e., get “deeper”) as training progresses and how this worsens several other problems, while outlining a principle to stabilize the dynamical systems of DEQ models by a regularization-based solution pursuant to these models’ implicitness.
2. The one-layer structure of deep equilibrium models could liberate these models from the costly chain-rule backpropagation process that constitute the learning overhead of conventional deep learning. In **Chapter 5**, we introduce **the notion of approximate gradient (a.k.a. “phantom” gradient or inexact gradient) that allows us to approximate the aforementioned implicit function theorem (IFT) extremely efficiently**. We will theoretically justify the feasibility of these approximations, which render the backward pass of equilibrium models  $5\times$  faster or almost free, a property that conventional neural networks do not have at all.
3. On a parallel thread, these implicit networks also enables decoupling the internal structure of the layer  $f_\theta$  (which controls *representational capacity*) from how the fixed point is actually computed (which impacts *inference-time efficiency*), which is usually via classic techniques such as Broyden’s method [34]. In **Chapter 6** we show that **one can exploit such decoupling and substantially enhance this fixed point computation using a custom neural solver** that can be trained end-to-end in an unsupervised manner.

Combining these discussions on the DEQ models, in **Part III** of this thesis we will demonstrate how these insights can be translated into a variety of applications and extensions (in addition to the large-scale settings Part I & II). We will additionally demonstrate a key advantage of DEQ models in practice, across various data modalities: adaptive computation. While conventional deep networks need to go through a prescribed computation graph regardless of the input complexity, we show that **the equilibrium approach could benefit significantly from highly-correlated data and effectively recycle computations to amortize the cost**. With all of these aforementioned techniques:

1. In **Chapter 7**, we will show that a DEQ-based approach can be multiple times more memory-wise and computationally efficient than, while improving the SOTA performance of, the best conventional deep network on optical flow estimation tasks. We propose DEQ-flow as a new *framework* that is compatible with the prior modeling efforts and replaces the existing recurrent/unrolling procedure completely.
2. In **Chapter 8**, we will show that these implicit models better learn the implicit neural representations (INR) for images, audios, videos and 3D models while with significantly less training time and memory cost.

In **Chapter 9**, we provide a summary of all these contributions, while discussing some

interesting “old questions” related to this new paradigm of deep learning. For example, traditional neural networks were motivated by neurons in the human brain. Are DEQ models less “biological” in any sense? As another example, how can these dynamical-system-view of deep learning best applied in real life dynamical system? We offer some insights for these questions (and for future research) in this final chapter.

With these theoretical and empirical explorations, we hope to be able to propose a different form of deep learning as how this subject has been traditionally studied. **Are layers necessary to deep learning?** This thesis suggests the answer is **no**. Or at least they are not the full picture. We will show that these implicit equilibrium approaches are an important research agenda in that current deep learning have some fundamental ceiling that has to be overcome, and that DEQ models are frequently better by design.

These pioneering work included in this thesis has challenged the long-held view that layer-based hierarchical architectural was a indispensable component of modern deep learning, and has led to a new and quickly growing community called “implicit deep learning”, as well as a NeurIPS 2020 official tutorial “[Deep Implicit Layers](#)” [68].

## 1.2.1 Other Contributions

We also briefly summarize here a list of other contributions during the graduate study that were not extensively discussed in the thesis. Many of the work led (directly or indirectly) to the work on implicit deep learning which this thesis focus on.

**Sequence modeling [15, 16]** . While recurrent networks have long been the dominant force and default toolkit for sequence tasks, we revisit the convolutional approaches to sequence modeling. We present one of the most extensive systematic comparisons of convolutional and recurrent architectures [16] on numerous sequence tasks (from synthetic ones to extremely large-scale ones). Specifically, we distill the best practices in modern ConvNets like residual blocks and dilations to describe a simple *temporal convolutional network* (TCN). Our experimental results indicate that 1) TCN models substantially outperform generic recurrent architectures such as LSTMs and GRUs; and 2) the “infinite sequence memory” advantage of RNNs is largely absent in practice, while TCNs exhibit much longer memory than recurrent architectures with the same capacity. Since its introduction, the proposed generic TCN model has had a phenomenal impact on modeling modern realistic time-series due to its various benefits (e.g., parallelism, good memory retention), and still maintain state-of-the-art level performances in many domains as of today (especially where extremely long-range information is present), such as in speech separation [153, 159], speech recognition [51], speech enhancement [182], genomics modeling [71], text classification [111], lip reading [1], financial time-series [203, 243], dynamic recommender systems [255], human trajectory prediction [173], and many more.

**Deep learning architectures [17, 228]** . We also present studies on the architectural properties of the cutting-edge deep sequence models. In Bai et al. [17], we present trellis networks (TrellisNet), which is a special TCN characterized by weight-tying and direct residual connections from the input layer into deep layers. But on the other hand, we prove that truncated recurrent networks are equivalent to trellis networks with special sparsity structures in their weight matrices.

The TrellisNet architecture therefore bridges two major and seemingly incompatible families of sequence models: recurrent and convolutional networks, and allows us to combine the best practices from both worlds. In addition, in Tsai et al. [228], we study the Transformers from a kernel smoother perspective and perform an in-depth dissection into the individual components of these models' self-attention mechanism and positional encoding.

**Unaligned multimodal machine learning [227]** . A major challenge in modeling multimodal time-series is the fusion of feature representations from multiple modalities (e.g., visual, acoustic, and textual time-series), which are asynchronized and usually require laborious human alignment. We propose Multimodal Transformer (MulT) [227], which uses crossmodal attention to latently adapt unaligned streams from one modality to another. This significantly reduces the requirement for careful feature engineering (which frequently involves lots of domain knowledge) and we show the attention-based multimodal learning can improve over prior methods by 5%-15% consistently.

**Deep learning for scientific computing [30, 205]** . We present graph transformer neural network force field (GTFF) [205] as a computational algorithm for direct prediction of atomic forces in molecular dynamics computer simulations in material systems. Although accurate methodologies exist to calculate the underlying atomic forces and behaviors, they are also extremely expensive because of the tremendous amount of computational resources necessary to apply the approach (e.g., days or weeks per molecule). In contrast, our graph transformer based method can be hundreds of thousands of times faster while losing almost no accuracy. This contribution was made as a part of a Kaggle competition on [Predicting Molecular Properties \[30\]](#), where our method won 1st place out of 2,737 participating teams.

## **Part I**

# **Equilibrium Approach Fundamentals: Infinite Layers to One Layer**





# Chapter 2

## Deep Equilibrium Models

Most modern feedforward deep networks are built on the core concept of *layers*. In the forward pass, each network consists of a stack of some  $L$  transformations, where  $L$  is the depth of the network. To update these networks, the backward passes rely on backpropagating through the same  $L$  layers via the chain rule, which typically necessitates that we store the intermediate values of these layers. The value for  $L$  is usually a hyperparameter and is picked by model designers (e.g., ResNet-101 [96]).

In very general terms, a deep feedforward model can be written as the following iteration:

$$\mathbf{z}^{[i+1]} = f_{\theta}^{[i]}(\mathbf{z}^{[i]}; \mathbf{x}) \quad \text{for } i = 0, 1, 2, \dots, L - 1 \quad (2.1)$$

where  $i$  is the layer index;  $\mathbf{z}^{[i]}$  is the hidden state at layer  $i$ ;  $\mathbf{x}$  is the input data (i.e., we are choosing to explicitly model skip connections, for reasons we explain later); and  $f_{\theta}^{[i]}$  is some nonlinear transformation. As these networks keep getting deeper (e.g., [32, 49, 96, 130]), one might ask: what is the limit of this layer stacking process, and how do we model it?

In this chapter, we propose a new approach to “deep” modeling that addresses these questions. We start in Sec. 2.1 with a preliminary discussion of how we could reduce from a very deep, and potentially infinite-layer network to a one-layer network, corroborated by theoretical and empirical evidence of the rationale behind this reduction. Then in Sec. 2.2, we formally introduce the deep equilibrium (DEQ) models, which are at the core of this entire thesis. This equilibrium method directly computes the fixed point  $\mathbf{z}^*$  of a nonlinear transformation, i.e., the solution to the nonlinear system

$$\mathbf{z}^* = f_{\theta}(\mathbf{z}^*; \mathbf{x}). \quad (2.2)$$

As we shall see, this solution corresponds to the eventual hidden layer values of an *infinite depth* network. But instead of finding this value by iterating the model, we propose to directly (and in practice, more quickly) solve for the equilibrium via any black-box root-finding method. Importantly, we show that DEQ can *directly* differentiate through the fixed point equations via implicit differentiation [128], which does not require storing *any* intermediate activation values. In other words, we can backpropagate through the infinite-depth network while using only *constant* memory, equivalent to a single layer’s activations.

After developing the generic DEQ approach, in the second half of this chapter we study in detail the universality of these layer-*less* DEQ models’ representational capacities, and pro-

vide two specific *instantiations* of this approach on realistic sequence tasks, based on temporal convolutions [17] and multi-head self-attention [58, 61] architectures. With these results, we formally establish the equilibrium approach, thereby offering a novel perspective on deep learning algorithms and challenging the traditional notion of “layer stacking”. The results in this chapter have been previously published in Bai et al. [18].

## 2.1 Preliminary: Infinite Layers to a Single Layer

To achieve our goal to go from infinite-layer networks to one-layer networks, we first broadly consider the class of  $L$ -layer *weight-tied* deep models (with passthrough connections from the input to each layer, also known as *input injections*), which consists of the update

$$\mathbf{z}^{[i+1]} = f_{\theta}(\mathbf{z}^{[i]}; \mathbf{x}), \quad i = 0, \dots, L-1, \quad \mathbf{z}^{[0]} \quad (2.3)$$

In practice, such weight-tying has been generally considered to come with some major benefits. For instance, it acts as a form of regularization and could significantly reduce the model size (as a compression method). While this formulation seems to substantially constrain the class of functions that deep learning could represent and hurt the model expressivity, there have been numerous surprising recent works, in domains like computer vision and sequence modeling, that employ the same transformation in each layer and still achieve results competitive with or better than the state-of-the-art [17, 57, 61, 123, 131, 143, 221]. Such empirical claim is strengthened with the following theoretical statement.

**Theorem 1. (Universality of Weight-tied Deep Networks)** Consider a traditional  $L$ -layer deep network defined by the relation

$$\mathbf{z}^{[i+1]} = \sigma^{[i]}(W^{[i]}\mathbf{z}^{[i]} + \mathbf{b}^{[i]}), \quad i = 0, \dots, L-1, \quad \mathbf{z}^{[0]} = \mathbf{x} \quad (2.4)$$

where  $\mathbf{z}^{[i]}$  denotes the hidden features at depth  $i$ ,  $W^{[i]}$ ,  $\mathbf{b}^{[i]}$  are parameters of the network,  $\sigma^{[i]}$  is the non-linearity at depth  $i$ , and  $\mathbf{x}$  is the original input. Then the same network can be represented by a weight-tied, input-injected network of equivalent depth

$$\tilde{\mathbf{z}}^{[i+1]} = \sigma(W_z \tilde{\mathbf{z}}^{[i]} + W_x \mathbf{x} + \tilde{\mathbf{b}}), \quad i = 0, \dots, L-1. \quad (2.5)$$

where  $\sigma$ ,  $W_z$ ,  $W_x$  and  $\tilde{\mathbf{b}}$  are constant (and shared) over all layers.

*Proof.* The proof is constructive: we build the weight-tied network equivalent to the original network by constructing the relevant matrices using a simple “shift” operation. In particular, we define the network parameters as

$$W_z = \begin{bmatrix} 0 & 0 & \dots & 0 & 0 \\ W^{[1]} & 0 & \dots & 0 & 0 \\ 0 & W^{[2]} & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & W^{[L-1]} & 0 \end{bmatrix}, \quad W_x = \begin{bmatrix} W^{[0]} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \tilde{\mathbf{b}} = \begin{bmatrix} \mathbf{b}^{[0]} \\ \mathbf{b}^{[1]} \\ \vdots \\ \mathbf{b}^{[L-1]} \end{bmatrix}, \quad \sigma = \begin{bmatrix} \sigma^{[0]} \\ \sigma^{[1]} \\ \vdots \\ \sigma^{[L-1]} \end{bmatrix}. \quad (2.6)$$

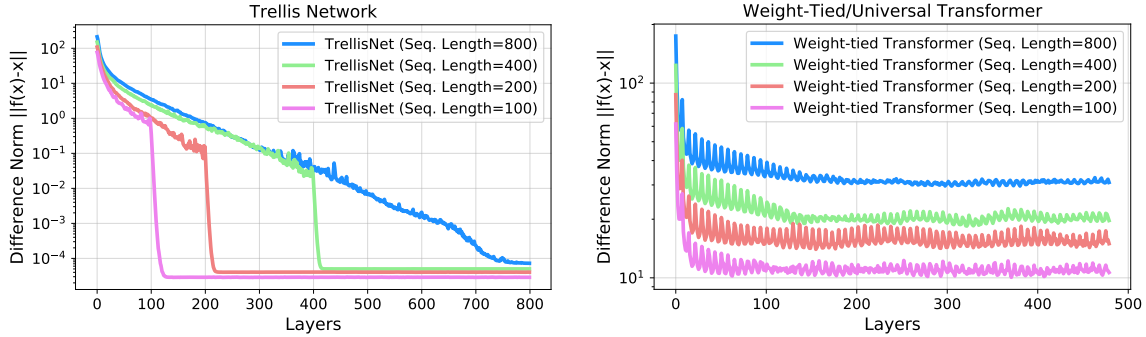


Figure 2.1: The behavior of hidden states over infinite unrolling: the activations in TrellisNet [17] and Universal Transformers [61] on sequence inputs with different lengths. The  $y$ -axis denotes  $\|f_\theta(\mathbf{z}) - \mathbf{z}\|$ ; i.e., the change in hidden units as depth increases. Expectedly, longer sequences contain more information and take longer to converge. For Transformers, the hidden unit convergence stops at around 100 layers and the fixed-point oscillation starts.

It is clear from inspection that after  $L$  applications of the layer, i.e.,

$$\tilde{\mathbf{z}}^{[i+1]} = \sigma(W_z \tilde{\mathbf{z}}^{[i]} + W_x \mathbf{x} + \tilde{\mathbf{b}}) \quad (2.7)$$

using these parameters the hidden vector  $\tilde{\mathbf{z}}$  will take on the value

$$\tilde{\mathbf{z}}^{[L]\top} = \begin{bmatrix} \mathbf{z}^{[1]} & \mathbf{z}^{[2]} & \vdots & \mathbf{z}^{[L]} \end{bmatrix}. \quad (2.8)$$

Thus the weight-tied network computes all the same terms as the original network, using the same depth as the original network, and with a hidden unit size that is just the sum of the individual hidden unit sizes in the original network. This establishes the claim of the theorem. ■

In other words, Theorem 1 shows that any conventional deep feedforward network can be embedded in the *weight-tied, input-injected form* (of a wider layer  $W_z$ ) with equivalent depth. Note that the theorem is not advocating that we *should* do such weight-tying in practice, but that we *can*: expressivity-wise, we do not lose anything by considering these repeated applications of a layer.

Although these weight-tied networks can be unrolled to *any* depth, typically with improved feature abstractions as depth increases [17, 57], in practice almost all such models (and deep nets in general) are stacked, trained and evaluated by unrolling a *pre-determined, fixed number of layers*. Like any deep network, this yields prescribed computation graphs that these deep networks must abide by at training and inference times. One of the primary reasons is the need for the models to store intermediate hidden units for backpropagation—and thus they cannot be trained beyond a certain depth that depends on the available hardware memory.

In principle, the network could have *infinite* depth. This is attained in the limit of unrolling a weight-tied model for an ever higher number of layers. But what is the limit of this process? In practice, for many typical classes of  $f_\theta$  (discussed later), we observe that such weight-tied models tend to converge to a fixed point as depth increases to infinity (e.g., see Fig. 2.1). This reflects a phenomenon of “diminishing returns”: each additional layer has a smaller and smaller

contribution over the feature abstraction (and accordingly, the level of performance), until the network reaches an *equilibrium*:

$$\lim_{i \rightarrow \infty} \mathbf{z}^{[i]} = \lim_{i \rightarrow \infty} f_{\theta}(\mathbf{z}^{[i]}; \mathbf{x}) \equiv f_{\theta}(\mathbf{z}^*; \mathbf{x}) = \mathbf{z}^* \quad (2.9)$$

A heuristic argument behind this convergent phenomenon is that this kind of convergence precisely characterizes the stability of common deep networks: since we have developed network architectures that are already stable for very deep stacking (e.g., ResNet-101 [96] and DenseNet-264 [103]), we have in a sense already biased our design towards layers that tend to stable fixed points (or they will quickly blow up anyway). Such hypothesis and observations provide the key motivation for reducing an infinite-level network to a one-layer version, which we directly characterize by the (implicit) solution to the fixed-point equation (2.9).

## 2.2 Deep Equilibrium Models

The goal of a deep equilibrium (DEQ) model is then to exactly solve for this *fixed-point* expression. However, instead of iteratively stacking  $f_{\theta}$  itself, we advocate for directly solving the solution variable  $\mathbf{z}^*$  in Eq. (2.9) and differentiating through this equilibrium states, which allows us to characterize output as an *implicit* function of the input  $\mathbf{x}$  and parameters  $\theta$ .

### 2.2.1 Forward Pass

Unlike a conventional network where the output is the activations from the  $L^{\text{th}}$  layer, the output of a DEQ is the equilibrium point itself. Therefore, the forward evaluation could be any procedure that solves for this equilibrium point. Conventional deep networks, if they converge to an equilibrium, can be considered a simplest, naïve *iterative solver*:

$$\mathbf{z}^{[i+1]} = f_{\theta}(\mathbf{z}^{[i]}; \mathbf{x}) \quad \text{for } i = 0, 1, 2, \dots \quad (2.10)$$

But alternatively, one can use other methods that provide faster convergence guarantees. For notational convenience, we define the  $g_{\theta}$  as the residual function of  $f_{\theta}$ , and rewrite Eq. (2.9) as  $g_{\theta}(\mathbf{z}^*; \mathbf{x}) - \mathbf{z}^* \rightarrow 0$ . The equilibrium state  $\mathbf{z}^* \in \mathbb{R}^d$  is thus the root of  $g_{\theta}$ , which we can find more easily (and via different trajectories) with Newton’s method or quasi-Newton methods (e.g., Broyden’s method [34]):

$$\mathbf{z}^{[i+1]} = \mathbf{z}^{[i]} - \alpha B g_{\theta}(\mathbf{z}^{[i]}; \mathbf{x}) \quad (2.11)$$

where  $B$  is the inverse Jacobian at  $\mathbf{z}^{[i]}$  (or its low-rank update approximation; the idea is to use gradient-related information about  $f_{\theta}$  to take a smarter step), and  $\alpha$  the step size. But generally, one can exploit any black-box root-finding algorithm to solve for the equilibrium point in the forward pass, given an initial estimate  $\mathbf{z}^{[0]}$  (which we set to  $\mathbf{0}$ ):  $\mathbf{z}^* = \text{RootFind}(g_{\theta}; \mathbf{x})$ .

In practice, the cost of computing the exact inverse Jacobian  $J_{g_{\theta}}^{-1} = (J_{f_{\theta}} - I)^{-1} \in \mathbb{R}^{d \times d}$  can be prohibitive in high dimensions (in terms of both computation and memory storage). Therefore, instead of relying on Newton’s method, we address this using quasi-Newton methods;

for example, Broyden’s method makes low-rank updates to approximate  $J_{g_\theta}^{-1}$  via the Sherman-Morrison formula [207]

$$J_{g_\theta}^{-1}(\mathbf{z}^{[i+1]}) = (J_{f_\theta}(\mathbf{z}^{[i+1]}) - I)^{-1} \approx B^{[i+1]} = B^{[i]} + \underbrace{\frac{\Delta \mathbf{z}^{[i+1]} - B^{[i]} \Delta g_\theta^{[i+1]}}{\Delta \mathbf{z}^{[i+1]\top} B^{[i]} \Delta g_\theta^{[i+1]}}}_{d \times 1} \underbrace{\Delta \mathbf{z}^{[i+1]\top} B^{[i]}}_{1 \times d} \quad (2.12)$$

where  $\Delta \mathbf{z}^{[i+1]} = \mathbf{z}^{[i+1]} - \mathbf{z}^{[i]}$  and  $\Delta g_\theta^{[i+1]} = g_\theta(\mathbf{z}^{[i+1]}; \mathbf{x}) - g_\theta(\mathbf{z}^{[i]}; \mathbf{x})$ . Thus, Broyden’s method allows us to conveniently store  $B$  by only these low-rank updates  $\mathbf{u}, \mathbf{v}$  of iteration:  $B^{[i+1]} = -I + \sum_{k=0}^i \mathbf{u}^{[k]} \mathbf{v}^{[k]\top}$ . Other alternatives include Anderson Acceleration (AA) [8], which mixes the past few steps of the residual by solving a tiny least-squared solution in each step greedily. Other follow-up work on DEQ models have relied on different algorithms, such as Peaceman-Rachford method [244]. We later show in Chapter 6 that this solving process itself can be parameterized and trained, since the dynamical systems here are clearly input-dependent as well.

## 2.2.2 Backward Pass

A major problem with using a black-box fixed-point solver is that we are no longer able to rely on explicit backpropagation through the exact operations in the forward pass. While one can certainly fix an algorithm (say Newton’s method) to obtain the equilibrium, and then store and backpropagate through all the Newton iterations, we provide below an alternative procedure based on the implicit differentiation that is much simpler, requires constant memory, and assumes no knowledge of the forward pass at all.

**Theorem 2. (Gradient of DEQ; Implicit Function Theorem [128])** *Let  $\mathbf{z}^* \in \mathbb{R}^d$  be an equilibrium point, and  $\mathbf{y} \in \mathbb{R}^q$  the ground-truth. Let  $h : \mathbb{R}^d \rightarrow \mathbb{R}^q$  be any differentiable function (e.g., a linear transformation) and let  $\mathcal{L} : \mathbb{R}^q \times \mathbb{R}^q \rightarrow \mathbb{R}$  be a loss function that computes*

$$\ell = \mathcal{L}(h(\mathbf{z}^*), \mathbf{y}) = \mathcal{L}(h(\text{RootFind}(g_\theta; \mathbf{x})), \mathbf{y}). \quad (2.13)$$

*Then the loss gradient with respect to  $(\cdot)$  (a stand-in for any quantity we want to differentiate the fixed point w.r.t.) is*

$$\frac{\partial \ell}{\partial (\cdot)} = - \frac{\partial \ell}{\partial \mathbf{z}^*} (J_{g_\theta}^{-1}(\mathbf{z}^*)) \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial (\cdot)} = \frac{\partial \ell}{\partial h} \frac{\partial h}{\partial \mathbf{z}^*} (I - J_{f_\theta}(\mathbf{z}^*))^{-1} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial (\cdot)}. \quad (2.14)$$

The insight provided by Theorem 2 is at the core of our method and its various benefits. Importantly, the backward gradient through the “infinite” stacking can be represented as one step of matrix multiplication that involves the Jacobian at equilibrium. For instance, an SGD update step on model parameters  $\theta$  would be

$$\theta^+ = \theta - \alpha \cdot \frac{\partial \ell}{\partial \theta} = \theta + \alpha \frac{\partial \ell}{\partial \mathbf{z}^*} (J_{g_\theta}^{-1}|_{\mathbf{z}^*}) \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta}. \quad (2.15)$$

Note that this result is independent of the root-finding algorithm we choose or the internal structure of the transformation  $f_\theta$ , and thus does not require any storage of the intermediate hidden states,

which is necessary for backpropagation in conventional deep networks. We now proceed to formally prove Thm. 2 below.

*Proof.* We first write out the equilibrium sequence condition:  $f_\theta(\mathbf{z}^*; \mathbf{x}) = \mathbf{z}^*$ . By implicitly differentiating two sides of this condition with respect to  $(\cdot)$ :

$$\begin{aligned} \frac{d\mathbf{z}^*}{d(\cdot)} &= \frac{df_\theta(\mathbf{z}^*; \mathbf{x})}{d(\cdot)} = \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial(\cdot)} + \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \mathbf{z}^*} \frac{d\mathbf{z}^*}{d(\cdot)} \\ \implies \left( I - \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \mathbf{z}^*} \right) \frac{d\mathbf{z}^*}{d(\cdot)} &= \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial(\cdot)} \end{aligned}$$

Since  $g_\theta(\mathbf{z}^*; \mathbf{x}) = f_\theta(\mathbf{z}^*; \mathbf{x}) - \mathbf{z}^*$ , we have

$$J_{g_\theta}|_{\mathbf{z}^*} = - \left( I - \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \mathbf{z}^*} \right),$$

which implies

$$\frac{\partial \ell}{\partial(\cdot)} = \frac{\partial \ell}{\partial \mathbf{z}^*} \frac{d\mathbf{z}^*}{d(\cdot)} = - \frac{\partial \ell}{\partial \mathbf{z}^*} (J_{g_\theta}^{-1}|_{\mathbf{z}^*}) \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial(\cdot)}.$$

■

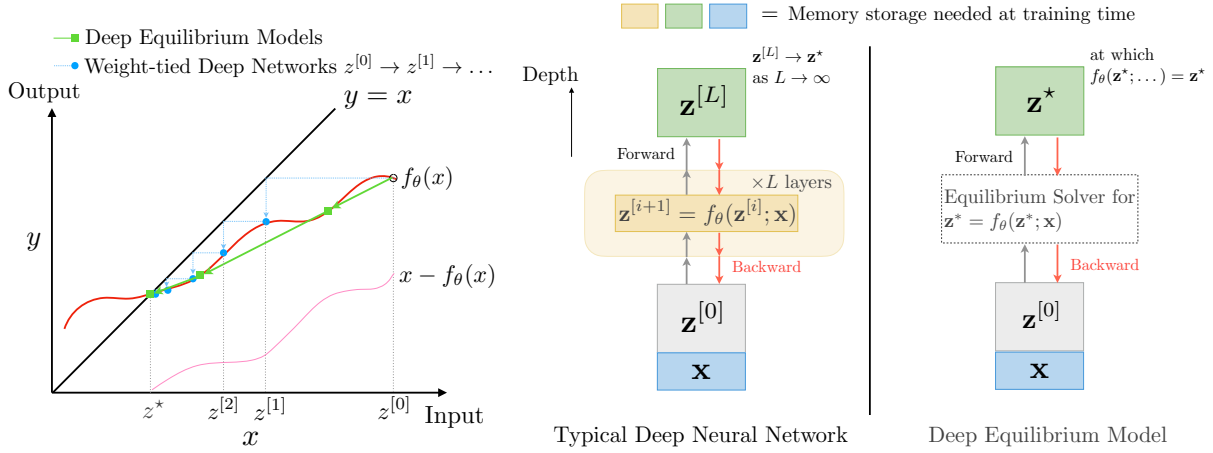
Just like in the forward pass (see Sec. 2.2.1), to compute the implicit differentiation in practice without explicit forming and inverting the Jacobian matrices, we can solve a *linear fixed point system* involving variable  $\mathbf{u} \in \mathbb{R}^d$ :

$$\mathbf{u}^\top = \mathbf{u}^\top J_{f_\theta}(\mathbf{z}^*) + \frac{\partial \ell}{\partial \mathbf{z}^*} \tag{2.16}$$

$$= \mathbf{u}^\top \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \mathbf{z}^*} + \frac{\partial \ell}{\partial \mathbf{z}^*}. \tag{2.17}$$

The solution  $\mathbf{u}^*$  of this linear system will be exactly the portion of implicit gradient highlighted in red in Eq. 2.14, and the vector-Jacobian product  $\mathbf{u}^\top J_{f_\theta}(\mathbf{z}^*)$  can be efficiently computed via autograd packages (e.g., PyTorch [185]). In fact, such a linear system can be solved by any indirect methods that leverage fast matrix-vector products; e.g., we can also use Broyden’s method.

**Memory cost of DEQ.** Fig. 2.2 shows a generic comparison between conventional deep networks and the DEQ approach. Specifically, an important benefit of DEQ is its extreme memory efficiency: to train a deep equilibrium network, we only need to store  $\mathbf{z}^*$  (the equilibrium feature vector),  $\mathbf{x}$  (input) and the  $f_\theta$  itself so that we can solve the backward fixed-point linear system (2.16), where we never construct the  $Nd \times Nd$  Jacobian inverse  $J_{f_\theta}^{-1}$  (where  $N$  is the size of the mini-batch), but only require vector-Jacobian products. Compared to conventional deep networks whose cost scale linearly with their depths, DEQs therefore offer a *constant-memory alternative* which yet models infinite layers. This enables models that previously required multiple GPUs and other implementation-based techniques (e.g., half-precision or gradient checkpointing [46]) to fit easily into even a single GPU.



(a) A simple illustration of solving for an equilibrium point in 2D. (b) A deep equilibrium model operates with black-box forward and backward solvers.

Figure 2.2: Comparison of the DEQ with conventional weight-tied deep networks.

### 2.2.3 Universality of Expressivity

Note that the above formulation yields one fixed-point layer (defined by  $f_\theta$ ). A natural question arises: since a single DEQ layer is as powerful as arbitrary stacked (infinite-level) explicit layers, could we stack these DEQ layers (with potentially *different* classes of transformations) to obtain something even more powerful? The answer, somewhat surprisingly, is no; in fact, a single DEQ layer can model any number of “stacked” DEQ layers as well. This is evidenced formally by the following theorem.

**Theorem 3. (Universality of “single-layer” DEQs.)** Let  $\mathbf{x} \in \mathbb{R}^p$  be the input, and  $\theta^{[1]}, \theta^{[2]}$  the sets of parameters for stable (but potentially different) transformations  $f_{\theta^{[1]}} : \mathbb{R}^r \times \mathbb{R}^p \rightarrow \mathbb{R}^r$  and  $v_{\theta^{[2]}} : \mathbb{R}^d \times \mathbb{R}^r \rightarrow \mathbb{R}^d$ , respectively. Then there exists  $\Gamma_\Theta : \mathbb{R}^{d+r} \times \mathbb{R}^p \rightarrow \mathbb{R}^{d+r}$ , where  $\Theta = \theta^{[1]} \cup \theta^{[2]}$ , s.t.

$$\mathbf{z}^* = \text{RootFind}(g_{\theta^{[2]}}^f; \text{RootFind}(g_{\theta^{[1]}}^v; \mathbf{x})) = \text{RootFind}(g_\Theta^\Gamma; \mathbf{x})_{[-d:]} \quad (2.18)$$

where  $[\cdot]_{[-d:]}$  denotes the last  $d$  feature dimensions of  $[\cdot]$ .

*Proof.* We again provide a constructive proof. Assume  $\mathbf{z}^{[1]*} = \text{RootFind}(g_{\theta^{[1]}}^f; \mathbf{x}) \in \mathbb{R}^r$  is the equilibrium of the first DEQ module under transformation  $f_{\theta^{[1]}}$ . Define  $\Theta = \theta^{[1]} \cup \theta^{[2]}$ , and a new “wider layer”  $\Gamma_\Theta(\mathbf{w}; \mathbf{x}) : \mathbb{R}^{d+r} \times \mathbb{R}^p \rightarrow \mathbb{R}^{d+r}$  by:

$$\Gamma_\Theta(\mathbf{w}; \mathbf{x}) = \Gamma_\Theta \left( \begin{bmatrix} \mathbf{w}^{(1)} \\ \mathbf{w}^{(2)} \end{bmatrix}; \mathbf{x} \right) = \begin{bmatrix} f_{\theta^{[1]}}(\mathbf{w}^{(1)}, \mathbf{x}) \\ v_{\theta^{[2]}}(\mathbf{w}^{(2)}, \mathbf{w}^{(1)}) \end{bmatrix} \quad (2.19)$$

Then  $\mathbf{w}^* = \begin{bmatrix} \mathbf{z}^{[1]*} \\ \mathbf{z}^* \end{bmatrix}$  is a fixed point of  $\Gamma_\Theta(\cdot; \mathbf{x})$ , which completes the proof. ■

The theorem essentially shows that stacking multiple DEQs does not create extra representational power over a single-layer DEQ (as long as  $f_\theta$  is expressive enough), and *one (implicit) layer is all you need*. Indeed, we next show that DEQ models embedded with representative structures of  $f_\theta$  (e.g., based on the Transformer block) typically perform as competitively, or even better, than traditional neural networks of the same size.

## 2.3 Instantiations of DEQ

While the previous analyses of DEQ do not depend on the internal structure of  $f_\theta$ , in this section we briefly highlight two examples of  $f_\theta$  as specific instantiations of DEQ. Both models (TrellisNet [17] and self-attention [61, 233]) achieve state-of-the-art results on various sequence modeling benchmarks (where data are sequences of length  $T$ , denoted  $\mathbf{x}_{1:T}$ ). In the later chapters of this thesis (e.g., Chapter 3, 7 and 8), we shall see more versions of the DEQ models applied to different data modalities.

**Trellis networks.** We briefly introduce the trellis network (TrellisNet) here and refer interested readers to [17] for a detailed description. Generally, TrellisNet is a special kind of temporal convolution that generalizes recurrent neural networks (RNNs) and their variants (e.g., GRUs [50], LSTMs [99]). We can write TrellisNet with convolutional kernel size  $k$ , dilation  $s$ , and nonlinearity  $\psi$  in DEQ form as

$$\begin{aligned} \tilde{\mathbf{x}}_{1:T} &= \text{Input injection (i.e., linearly transformed inputs by Conv1D}(\mathbf{x}_{1:T}; W_x)) \\ f_\theta(\mathbf{z}_{1:T}; \mathbf{x}_{1:T}) &= \psi(\text{Conv1D}([\mathbf{u}_{-(k-1)s}, \mathbf{z}_{1:T}]; W_z) + \tilde{\mathbf{x}}_{1:T}) \end{aligned}$$

where  $\mathbf{u}_{-(k-1)s}$  is typically: 1) the last  $(k-1)s$  elements of the previous sequence’s output (if using history padding [17]); or 2) simply zero-padding.  $[\cdot, \cdot]$  means concatenation along the temporal dimension. Following [17], we use the LSTM gated activation for  $\psi$ .

**Weight-tied transformers.** At a high level, multi-head self-attention transformers [233] are very different from most deep networks. Instead of convolutions or recurrence, a self-attention layer maps the input into  $Q$  (query),  $K$  (key), and  $V$  (value) and computes the attention score between time-steps  $t_i$  and  $t_j$  as  $[QK^\top]_{i,j}$ . This attention score is then normalized via softmax and multiplied with the  $V$  sequence to produce the output. Since the transformer is order-invariant, prior work proposed to add positional embeddings (PE) [58, 233] to the self-attention operation. While referring readers to [58, 61, 233] for more details, we write a transformer block in the DEQ form as

$$\begin{aligned} \tilde{\mathbf{x}}_{1:T} &= \text{Input injection (i.e., linearly transformed inputs by } \mathbf{x}_{1:T} W_x) \\ f_\theta(\mathbf{z}_{1:T}; \mathbf{x}_{1:T}) &= \text{LN}(\phi(\text{LN}(\text{SelfAttention}(\mathbf{z}_{1:T} W_{QKV} + \tilde{\mathbf{x}}_{1:T}; \text{PE}_{1:T})))) \end{aligned}$$

where  $W_{QKV} \in \mathbb{R}^{d \times 3d}$  produces the  $Q, K, V$  for the multi-head self-attention, and LN stands for layer normalization [13]. Note that we add input injection  $\tilde{\mathbf{x}}_{1:T}$  to  $Q, K, V$  in addition to the positional embedding and initialize with  $\mathbf{z}_{1:T}^{[0]} = \mathbf{0}$ . Following prior work [58, 61, 63, 233], we adopt a positionwise feedforward residual block for  $\phi$ . In our implementation, we use the



Table 2.1: DEQ achieves strong performance on the long-range copy-memory task.

	Models (Size)			
	DEQ-Transformer (ours) (14K)	TCN [16] (16K)	LSTM [99] (14K)	GRU [50] (14K)
Copy Memory $T=400$ Loss	<b>3.5e-6</b>	<b>2.7e-5</b>	0.0501	0.0491

memory-augmented transformer proposed by [58], where we feed  $[\mathbf{z}_{-T'}^*, \mathbf{z}_{1:T}]$  (i.e., with history padding of length  $T'$ ) and relative positional embedding  $\text{PE}_{-T':T}$  to the self-attention operation.

Figure 2.2b provides a generic comparison between these conventional weight-tied deep networks and the DEQ approach, highlighting the constant memory requirements of the latter.

## 2.4 Experiments

We evaluate DEQ on both synthetic stress tests and realistic large-scale language modeling (where complex long-term temporal dependencies are involved). We use the two aforementioned instantiations of  $f_\theta$  in the equilibrium approach framework outlined in Sec. 2.2. On both WikiText-103 [164] (which contains  $>100\text{M}$  words and a vocabulary size of  $>260\text{K}$ ) and the smaller Penn Treebank corpus (where stronger regularizations are needed for conventional deep nets) for word-level language modeling, we show that DEQ achieves competitive (or better) performance even when compared to SOTA methods (of the same model size, both weight-tied and not) while using significantly less memory. A more detailed introduction of the tasks and datasets can be found in Bai et al. [18].

**Setting.** Both instantiations of DEQ use Broyden’s method [34], as previously mentioned. For the DEQ-TrellisNet instantiation, we roughly follow the settings of [17]. For DEQ-Transformers, we employ the relative positional embedding [58], with sequences of length 150 at both training and inference on the WikiText-103 dataset.

### 2.4.1 Copy Memory Task

The goal of the *copy memory task* is simple: to explicitly test a sequence model’s ability to exactly memorize elements across a long period of time [10, 16, 260] (see Appendix F in Bai et al. [18]). As shown in Table 2.1, DEQ demonstrates good memory retention (manifested by a significantly lower reconstruction loss) over relatively long sequences ( $T = 400$ ), with substantially better results than recurrent architectures such as LSTM/GRU. This is consistent with the findings in Bai et al. [16], while we note that a DEQ sequence model has only one Transformer layer modeled implicitly.

### 2.4.2 Large-Scale Language Modeling

One issue encountered in prior works that take a continuous view of deep networks [45, 94] is the challenge of scaling these approaches to real, high-dimensional, large-scale datasets. In this

Table 2.2: DEQ achieves competitive performance on word-level Penn Treebank language modeling (on par with SOTA results, without fine-tuning steps [166]). <sup>†</sup>The memory footprints are benchmarked (for fairness) on input sequence length 150 and batch size 15, which does not reflect the actual hyperparameters used; the values also do *not* include the memory for word embeddings.

Word-level Language Modeling w/ Penn Treebank (PTB)				
Model	# Params	Non-embedding model size	Test perplexity	Memory <sup>†</sup>
Variational LSTM [82]	66M	-	73.4	-
NAS Cell [269]	54M	-	62.4	-
NAS (w/ black-box hyperparameter tuner) [162]	24M	20M	59.7	-
AWD-LSTM [166]	24M	20M	58.8	-
DARTS architecture search (second order) [148]	23M	20M	<b>55.7</b>	-
60-layer TrellisNet (w/ auxiliary loss, w/o MoS) [17]	24M	20M	57.0	8.5GB
<b>DEQ-TrellisNet (ours)</b>	24M	20M	57.1	<b>1.2GB</b>

subsection, we evaluate the DEQ approach on some large-scale language datasets and investigate its effectiveness as a practical “implicit-depth” model.

**Performance on Penn Treebank.** Following the set of hyperparameters used by [17] for TrellisNet, we evaluate the DEQ-TrellisNet instantiation on word-level language modeling with the PTB corpus. Note that without an explicit notion of “layer”, we do not add auxiliary losses, as was done in [17]. As shown in Table 2.2, when trained from scratch, the DEQ-TrellisNet achieves a test perplexity on par with the original deeply supervised TrellisNet.

**Performance on WikiText-103.** On the much larger scale WT103 corpus (about 100x larger than PTB), the DEQ-TrellisNet achieves better test perplexity than the original deep TrellisNet. For the Transformer instantiation, we follow the design of the Transformer-XL model [58]. We specifically compare to a “medium” Transformer-XL model (the largest released model that can fit on GPUs) and a “small” Transformer-XL model, while noting that the largest Transformer-XL network has massive memory requirements (due in part to very wide hidden features, batch sizes, and training-time sequence lengths, which would not be decreased by a DEQ) and can only be trained on TPUs [58]. In Table 2.3, we show that the DEQs yield competitive performance, outperforming prior SOTA approaches such as [58] on similar model sizes while consuming much less memory during training.

**Memory footprint of DEQ.** For conventional deep networks with  $L$  layers, the training memory complexity is  $O(L)$  since all intermediate activations are stored for backpropagation. In comparison, DEQs have an  $O(1)$  (i.e., constant in depth) memory footprint due to the root-finding formulation. We benchmark the reduced memory consumption in the last column of Tables 2.2 and 2.3, with controlled sequence lengths and batch sizes for fairness. On both instantiations, the DEQ approach leads to an over 80% (up to 88%) reduction in memory consumption by the model (excluding word embeddings, which are orthogonal to the comparison here). Moreover, we empirically verify (using a 70-layer TrellisNet) that DEQ consumes even less memory than

Table 2.3: DEQ-based models are competitive with SOTA deep networks of the same model size on the WikiText-103 corpus, with significantly less memory. <sup>†</sup>See Table 2.2 for more details on the memory benchmarking. Transformer-XL models are not weight-tied, unless specified otherwise. A visualization of this table is also shown in Fig. 2.3.

Word-level Language Modeling w/ WikiText-103 (WT103)				
Model	# Params	Non-Embedding Model Size	Test perplexity	Memory <sup>†</sup>
Generic TCN [16]	150M	34M	45.2	-
Gated Linear ConvNet [59]	230M	-	37.2	-
AWD-QRNN [165]	159M	51M	33.0	7.1GB
Relational Memory Core [201]	195M	60M	31.6	-
Transformer-XL (X-large, adaptive embed., on TPU) [58]	257M	224M	<b>18.7</b>	12.0GB
70-layer TrellisNet (+ auxiliary loss, etc.) [17]	180M	45M	29.2	24.7GB
70-layer TrellisNet with <i>gradient checkpointing</i>	180M	45M	29.2	5.2GB
<b>DEQ-TrellisNet (ours)</b>	180M	45M	<b>29.0</b>	<b>3.3GB</b>
Transformer-XL (medium, 16 layers)	165M	44M	24.3	8.5GB
<b>DEQ-Transformer (medium, ours)</b>	172M	43M	24.2	<b>2.7GB</b>
Transformer-XL (medium, 18 layers, adaptive embed.)	110M	72M	23.6	9.0GB
<b>DEQ-Transformer (medium, adaptive embed., ours)</b>	110M	70M	<b>23.2</b>	3.7GB
Transformer-XL (small, 4 layers)	139M	4.9M	35.8	4.8GB
Transformer-XL (small, weight-tied 16 layers)	138M	4.5M	34.9	6.8GB
<b>DEQ-Transformer (small, ours)</b>	138M	4.5M	<b>32.4</b>	<b>1.1GB</b>

Table 2.4: Runtime ratios between DEQs and corresponding deep networks at training and inference ( $> 1 \times$  implies DEQ is slower). The ratios are benchmarked on WikiText-103.

DEQ / 18-layer Transformer		DEQ / 70-layer TrellisNet	
Training	Inference	Training	Inference
3.24 $\times$	2.56 $\times$	2.40 $\times$	1.64 $\times$

gradient checkpointing [46], a popular technique that reduces the memory required to train a layer-based model to  $O(\sqrt{L})$ . Note that the DEQ’s memory footprint remains competitive even when compared with baselines that are not weight-tied (a reduction of over 60%), with similar or better accuracy.

**Convergence to equilibrium.** The DEQ models do not have layers. One factor that affects computation time is the number of solver iterations in forward/backward passes’ fixed-point solving, where each step (e.g., Broyden or Anderson step) typically evaluates  $f_\theta$  exactly once (and therefore this is also commonly referred to as the number of functional evaluations (NFEs) in the implicit deep learning literature later). We find that in general the NFEs gradually increases with training epochs, an observation similar to the one reported for training other implicit models like Neural ODEs [45], reflecting a trend of the equilibrium network to “go deeper” as it learns better performance. We will explore this phenomenon later in Chapter 4 and explain how it is correlated to the spectral radius of  $J_{f_\theta}$  during training. Meanwhile, the backward pass requires

## Word-level Language Modeling on WikiText-103 (WT103)

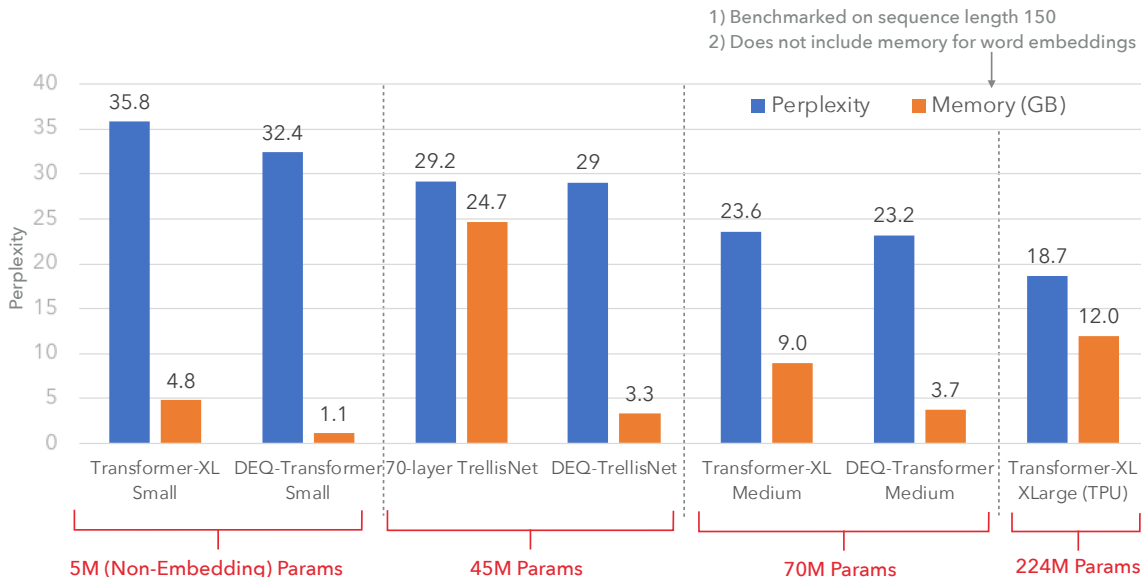


Figure 2.3: A visualization of the results presented in Table 2.3. The equilibrium formulations of temporal convolutional or self-attention layers (i.e., DEQ-TrellisNet and DEQ-Transformer) perform as competitively as conventional deep networks of similar sizes.

much fewer iterations than the forward, primarily due to the simplicity of the linear system in Eq. (2.16). We also find that DEQs can almost always converge to the sequence fixed points, much more efficiently than original weight-tied transformers (which could oscillate rather than converge; see Figure 2.4, right).

We also analyze how the equilibrium approach allows us to trade inference-time efficiency with the quality of the fixed point; i.e., by stopping the black-box fixed-point solver early, we can speed up the inference of a DEQ model at the cost of inaccurate fixed points and thus the performance of the model. Fundamentally, this is because DEQ models decouple the representational capacity (which is solely controlled by  $f_\theta$  (e.g., self-attentional or convolutional)) and the forward computation (which is determined by the solver), a fact that we will further leverage in Chapter 6. Figure 2.5 visualizes this tradeoff on a medium DEQ-Transformer. Note that accuracy quickly diverges when convergence tolerance  $\epsilon$  is too large, suggesting that a poor estimate of the equilibrium can hurt DEQ performances. Table 2.4 also provides approximate runtimes for competitive-accuracy DEQs on WikiText-103. DEQs are typically slower than layer-based deep networks (due to the fixed-point formulation). Accelerating these

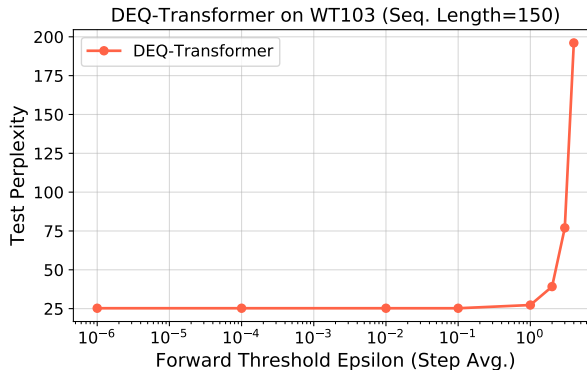


Figure 2.5: DEQs can be accelerated by early stopping, but poorer estimates also hurt performance.

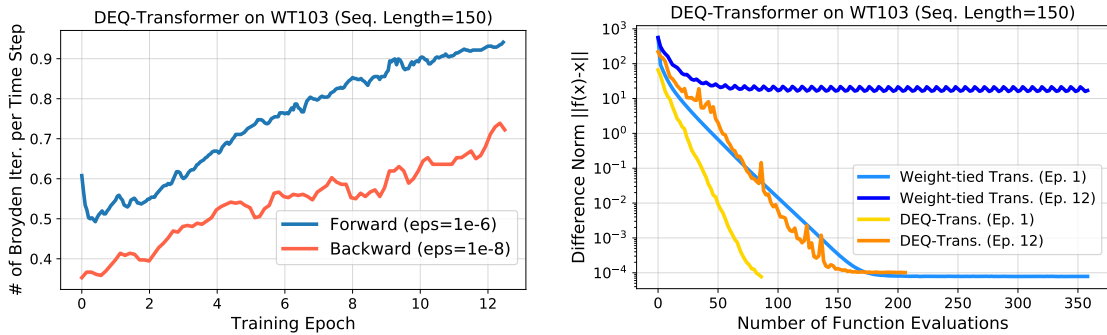


Figure 2.4: Left: number of Broyden iterations in forward and backward passes gradually grows with epochs. Right: DEQ-Transformer finds the equilibrium in a stable and efficient manner (whereas the deep transformer could oscillate around the fixed point, even when one exists).

dynamical systems is an important direction of research [20], and we will shed more light on this in Chapter 4, 5, 6 and 7.

## 2.5 Discussion

Deep networks have predominantly taken the form of stacks of layers. In this chapter, we introduced the generic formulation of deep equilibrium approach (DEQ), which directly solves for the “infinite-level” fixed-point representation of a layer and optimizes this equilibrium for better representations. We also show empirically and theoretically the universality of this one-layer implicit modeling. In particular, DEQ needs only  $O(1)$  memory at training time, is agnostic to the choice of the root solver in the forward pass, and is sufficiently versatile to subsume drastically different architectural choices (e.g., modern, complex layers like Transformer blocks). Our experiments have shown that DEQs have good temporal memory retention on sequences, are able to scale to realistic, large-scale sequence tasks, and perform competitively with, or slightly outperform, SOTA methods.

This chapter provides a general introduction to the nature of the equilibrium approach, which is at the center of this thesis. However multiple problem still remains. For example, layers are traditionally important because they also represent resolutions [136, 179], which are traditionally important for pattern recognition tasks (e.g., computer vision) to build a feature hierarchy. As another example, the experimental results suggest some new challenges (e.g., stability) to these implicit deep networks as we do away with layers. We will tackle at these issues and look at more applications and extensions of this new equilibrium perspective in the chapters to come.



## Chapter 3

# Simultaneous Equilibrium: Modeling Hierarchy Without Layers

State-of-the-art pattern recognition systems in domains such as computer vision and audio processing are almost universally based on multi-layer hierarchical feature extractors [134, 136, 137]. These models are structured in stages: the input is processed via a number of consecutive blocks, each operating at a different resolution [96, 130, 210, 219]. The architectures explicitly express hierarchical structure, with up- and downsampling layers that transition between consecutive blocks operating at different scales. An important motivation for such designs is the prominent multiscale structure and extremely high signal dimensionalities in these domains. A typical image, for instance, contains millions of pixels, which must be processed coherently by the model.

An alternative approach to differentiable modeling, as was introduced in Chapter 2, was implicit deep networks. These DEQ constructions replace explicit, deeply stacked layers with analytical conditions that the model must satisfy, and are able to simulate models with “infinite” depth within a constant memory footprint. *Is implicit deep learning relevant for general pattern recognition tasks?* One clear challenge here is that implicit networks do away with flexible “layers” and “stages”. It is therefore not clear whether they can appropriately model multiscale structure, and be used for cases like compound training (e.g., pre-training and fine-tuning).

Prompted by the flexibility and scalability of the equilibrium approach introduced in the previous chapter, in this chapter, we expand upon the generic DEQ construction substantially to introduce simultaneous equilibrium modeling of *multiple signal resolutions*. As we shall introduce in Sec. 3.1, a Multiscale DEQ (MDEQ) solves for equilibria of multiple streams *simultaneously* by directly optimizing for stable representations on *all* feature scales at the same time. Unlike conventional (explicit) networks, MDEQ does not process resolutions in succession, with higher resolutions flowing into lower ones or vice versa. Rather, the different feature scales are maintained side by side in a single “shallow” model that is driven to equilibrium.

This design brings two major advantages. First, using implicit differentiation, MDEQ has an  $O(1)$  memory footprint during training. This is especially important as pattern recognition systems are memory-intensive. Second, MDEQ rectifies one of the drawbacks of DEQ by exposing multiple feature scales at equilibrium, thereby providing natural interfaces for auxiliary losses and for compound training procedures such as pretraining (e.g., on ImageNet) and fine-tuning (e.g., on segmentation or detection tasks). Multiscale modeling enables a *single* MDEQ to simultaneously

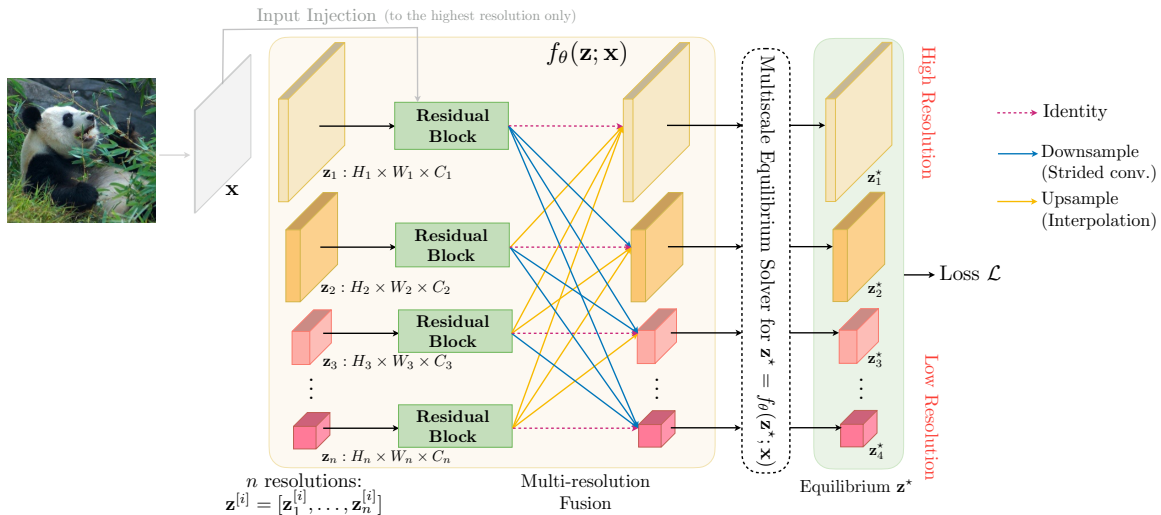


Figure 3.1: The structure of a multiscale deep equilibrium model (MDEQ). *All components* of the model are shown in this figure. MDEQ consists of a transformation  $f_\theta$  that is driven to equilibrium. Features at different scales coexist side by side and are driven to equilibrium simultaneously.

train for multiple losses defined on potentially very different scales, whose equilibrium features can serve as “heads” for a variety of tasks.

We demonstrate the effectiveness of MDEQ via extensive experiments on large-scale vision datasets. Remarkably, this shallow implicit model attains comparable accuracy levels to state-of-the-art deeply-stacked explicit ones. On ImageNet classification, MDEQs outperform baseline ResNets (e.g., ResNet-101) with similar parameter counts, reaching 77.5% top-1 accuracy. On Cityscapes semantic segmentation (dense labeling of 2-megapixel images), ImageNet-pretrained MDEQs match the performance of recent explicit models while consuming much less memory. Our largest MDEQ surpasses 80% mIoU on the Cityscapes validation set, outperforming strong convolutional networks and coming tantalizingly close to the state of the art. This chapter is primarily based on work that appeared in NeurIPS 2020 [19].

**Related work on multiscale networks.** Computer vision is a canonical application domain for hierarchical multiscale modeling. State-of-the-art models for problems in this field are explicitly structured into sequential stages of processing that operate at different resolutions [96, 130, 210, 219]. For example, a ResNet [96] typically consists of 4-6 sequential stages, each operating at half the resolution of the preceding one. A dilated ResNet [257] uses a different schedule for the progression of resolutions. A DenseNet [103] uses different connectivity patterns to carry information between layers, but shares the overarching structure: a sequence of stages. Other designs progressively decrease feature resolution and then increase it step by step [195]. Downsampling and upsampling can also be repeated, again in an explicitly choreographed sequence [176, 218]. Multiscale modeling has been a central motif in computer vision. The Laplacian pyramid is an influential early example of multiscale modeling [36]. Multiscale processing has been integrated with convolutional networks for scene parsing by Farabet et al. [73] and has been explicitly addressed in many subsequent architectures [40, 42, 43, 104, 145, 206, 236, 256, 264].



### 3.1 Multiscale Deep Equilibrium Models

We present the structure of an MDEQ model in Fig. 3.1. As in the previous chapter,  $f_\theta$  denotes the transformation that is (implicitly) iterated to a fixed point,  $\mathbf{x}$  is the input representation provided to  $f_\theta$ , and  $\mathbf{z}$  is the model’s internal state. We omit the batch dimension for clarity, and break down the multiple aspects of the DEQ model in this section.

**Transformation  $f_\theta$ .** The central part of MDEQ is the transformation  $f_\theta$  that is driven to equilibrium. Just like in chapter 2, we keep the design intentionally simple, in which features at each resolution are first taken through a residual block. The blocks are shallow and are identical in structure. At resolution  $i$ , the residual block receives the internal state  $\mathbf{z}_i$  and outputs a transformed feature tensor  $\mathbf{z}_i^+$  at the same resolution. The internal structure of the residual block is shown in Figure 3.2, where we largely adopt the design of He et al. [96]. The residual block at resolution  $i$  can be formally expressed as

$$\begin{aligned} \tilde{\mathbf{z}}_i &= \text{GroupNorm}(\text{Conv2d}(\mathbf{z}_i)) \\ \hat{\mathbf{z}}_i &= \text{GroupNorm}(\text{Conv2d}(\text{ReLU}(\tilde{\mathbf{z}}_i)) + 1_{\{i=1\}} \cdot \mathbf{x}) \\ \mathbf{z}_i^+ &= \text{GroupNorm}(\text{ReLU}(\hat{\mathbf{z}}_i + \mathbf{z}_i)). \end{aligned} \quad (3.1)$$

Following these blocks, the second part of  $f_\theta$  is a multi-resolution fusion step that mixes the feature maps across different scales (see Figure 3.1). The transformed features  $\mathbf{z}_i^+$  undergo either direct upsampling or downsampling from the current scale  $i$  to each other scale  $j \neq i$ . The final output at scale  $j$  is formed by summing over the transformed feature maps provided from all incoming scales  $i$  (along with  $\mathbf{z}_j^+$ ). This forces the features at all scales to be consistent and drives the whole system to a coordinated equilibrium that harmonizes the representations across scales.

**Input Representation.** Notably, only the highest resolution stream (i.e.,  $i = 1$ ) receives an *input injection*  $\mathbf{x}$  (see Eq. (3.1)), which is quite unlike the multiscale (sometimes called pyramidal) input representations used by many explicit vision architectures [43, 73]. The lower resolutions hence start with no knowledge at all about the input, with  $\mathbf{z}_i^{[0]} = \mathbf{0}$  for all  $i$ ; this information will only *implicitly* propagate through them as all scales are gradually driven to coordinated equilibria  $\mathbf{z}^*$  by the (black-box) fixed-point solver.

**(Limited-memory) Multiscale Equilibrium Solver.** In the DEQ, the internal state is a single tensor  $\mathbf{z}$ . The MDEQ state, however, is a *collection* of tensors at  $n$  resolutions:  $\mathbf{z} = [\mathbf{z}_1, \dots, \mathbf{z}_n]$ . Note that this is not a concatenation, as the different  $\mathbf{z}_i$  have different dimensionalities, feature resolutions, and semantics.  $\mathbf{z} = [\mathbf{z}_1, \dots, \mathbf{z}_n]$  is maintained as a collection of  $n$  tensors whose respective equilibrium states (i.e., roots) are solved for and backpropagated through simultaneously (with each resolution inducing its own loss). We present in Figure 3.3 a visualization of the

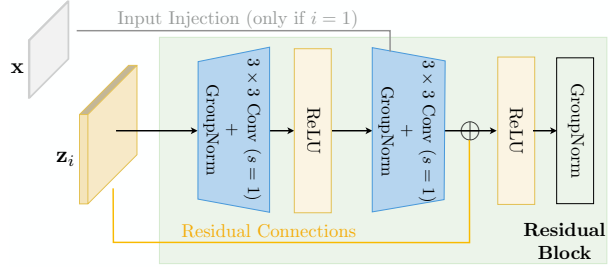


Figure 3.2: The residual block used in MDEQ. An MDEQ contains only *one* such layer.

convergence of all equilibrium streams in an MDEQ applied on CIFAR-10 images. From the figure, we see that 1) all MDEQ resolution streams indeed converge to their equilibria in parallel, with lower-resolution converge faster than higher-resolution streams; and 2) high-resolution feature converges much more quickly in multi-scale setting (pink line) than in the generic single-stream DEQ [18] setting (orange line), suggesting how the multiscale fusion influences the fixed-point convergence of these equilibrium models.

The original Broyden solver was not efficient enough (despite the low-rank updates) when applied to extremely high-dimensional computer vision datasets. For example, in the Cityscapes segmentation task (with  $2048 \times 1024$  resolution images), the Jacobian of a 4-resolution MDEQ at  $\mathbf{z}^*$  is well over 2,000 times larger than its single-stream counterpart in word-level language modeling in chapter 2. To address this, we employed a new solver that is inspired by Limited-memory BFGS (L-BFGS) [147], where we only keep the latest  $m$  low-rank updates at any step and discard the earlier ones.

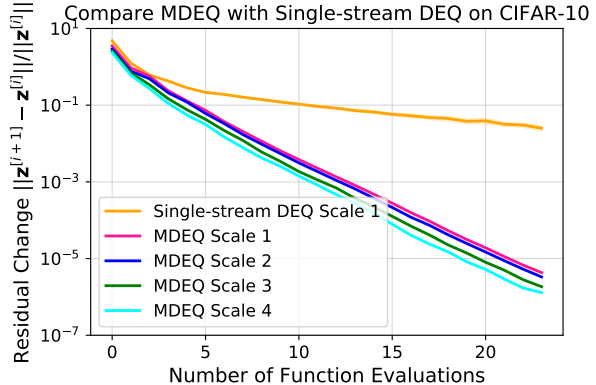


Figure 3.3: All resolutions of MDEQ converge *simultaneously*. Larger scale index means higher resolution.

Figure 3.4 provides a comparison of different modeling options. Prior implicit models (like generic DEQs) assume that a loss is defined on a single stream of implicit hidden states, which has a uniform input and output shape (Figure 3.4b). It is therefore not clear how such a model can be flexibly transferred across structurally different tasks (e.g., pretraining on image classification and fine-tuning on semantic segmentation), or how we can define auxiliary losses [135] since the forward and backward computation trajectories are decoupled.

## 3.2 Integration with Other Deep Learning Techniques

In comparison, MDEQ exposes convenient “interfaces” to its steady-state features at multiple resolutions. One resolution (the highest) can be the same as the resolution of the input, and can be used to define losses for dense prediction tasks such as semantic segmentation. Another resolution (the lowest) can be a vector in which the spatial dimensions are collapsed, and can be used to define losses for image-level labeling tasks such as classification. This suggests clean protocols for training the same model for different tasks, either jointly (e.g., multi-task learning in which structurally different supervision flows through multiple heads) or in sequence (e.g., pretraining for image classification through one head and fine-tuning for semantic segmentation through another), while still using implicit gradients. Overall, the multiscale equilibrium approach significantly generalizes the flexibility of the original, canonical equilibrium model, and permits us to adapt these implicit models to a much broader set of learning settings.

In comparison, MDEQ exposes convenient “interfaces” to its steady-state features at multiple resolutions. One resolution (the highest) can be the same as the resolution of the input, and can be used to define losses for dense prediction tasks such as semantic segmentation. Another resolution (the lowest) can be a vector in which the spatial dimensions are collapsed, and can be used to define losses for image-level labeling tasks such as classification. This suggests clean protocols for training the same model for different tasks, either jointly (e.g., multi-task learning in which structurally different supervision flows through multiple heads) or in sequence (e.g., pretraining for image classification through one head and fine-tuning for semantic segmentation through another), while still using implicit gradients. Overall, the multiscale equilibrium approach significantly generalizes the flexibility of the original, canonical equilibrium model, and permits us to adapt these implicit models to a much broader set of learning settings.

**Caveats.** However, since MDEQ simulates an “infinitely” deep network, such implicitness also calls for care when adapting common deep learning techniques. We provide an exploration of

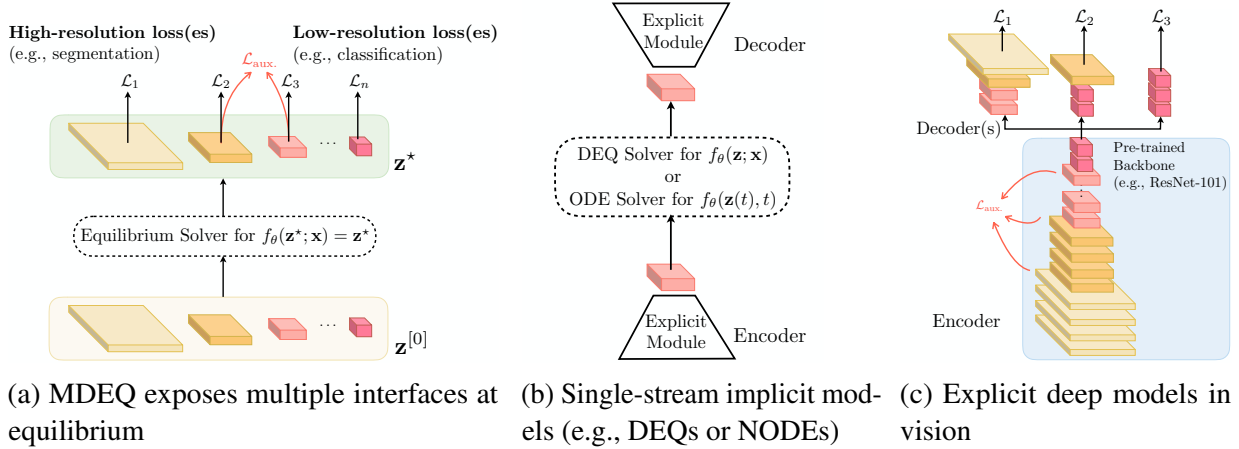


Figure 3.4: A visual comparison of MDEQ with prior implicit models and with standard explicit models in computer vision. Equilibrium states at multiple resolutions enable MDEQ to incorporate supervision in different forms.

such adaptations and their impact on the training dynamics of DEQ models. For example:

- **Normalization.** Layer normalization of hidden activations in  $f_{\theta}$  played an important role in constraining the output and stabilizing DEQs on sequences [18]. A natural counterpart in vision is batch normalization (BN) [110]. However, BN is not directly suitable for implicit models, since it estimates population statistics based on layers, which are implicit in our setting, and the Jacobian matrix of the transformation  $f_{\theta}$  will scale badly to make the fixed point significantly harder to solve for. We therefore use group normalization (GN) [246], which groups the input channels and performs normalization within each group. GN is independent of batch size and offers more natural support for transfer learning (e.g., pretraining and fine-tuning on structurally different tasks). We keep the learnable affine parameters of group normalization in the equilibrium framework.
- **Dropout.** The conventional spatial dropout used by explicit vision models applies a random mask to given layers in the network [214]. A new mask is generated whenever dropout is invoked. Such layer-based stochasticity can significantly hurt the stability of convergence to the equilibrium. In fact, as two adjacent calls to  $f_{\theta}$  most probably will have different Bernoulli dropout masks, it is almost impossible to reach a fixed point where  $f_{\theta}(z^*; x) = z^*$ . We therefore adopt variational dropout [82] and apply the exact same mask at all invocations of  $f_{\theta}$  in a given training iteration. The mask is reset at each training iteration.
- **Convolution and Convergence.** Whereas the original DEQ model focused primarily on self-attention transformations [233], where all hidden units communicate globally, MDEQ models face additional challenges due to the nature of typical vision models. Specifically, they employ convolutions with small receptive fields (e.g., the two  $3 \times 3$  convolutional filters in  $f_{\theta}$ 's residual block) on potentially very large images. In consequence, we typically need a higher number of root-finding iterations to converge to an exact equilibrium. While this does pose a challenge, we find that using the aforementioned strategies of 1) multiscale simultaneous up- and downsampling and 2) quasi-Newton root-finding, drives the model close to equilibrium within a reasonable number of iterations (e.g., see Fig. 3.3).

### 3.3 Experiments

We investigate the empirical performance of MDEQs from two aspects. First, as prior implicit approaches such as Neural ODEs (NODEs) have mostly evaluated on smaller-scale benchmarks such as MNIST [134] and CIFAR-10 ( $32 \times 32$  images) [129], we compare MDEQs with these baselines on the same benchmarks. We evaluate both training-time stability and inference-time performance. Second, we evaluate MDEQs on large-scale computer vision tasks: ImageNet classification [62] and semantic segmentation on the Cityscapes dataset [53]. These tasks have extremely high-dimensional inputs (e.g.,  $2048 \times 1024$  images for Cityscapes) and are dominated by explicit models. More detailed descriptions of the tasks, hyperparameters, and training settings are in Bai et al. [19].

We do note that even with the implicit modeling of layer  $f_\theta$ , the *mini explicit structure* within the design of  $f_\theta$  (e.g., the residual block) is still very helpful empirically in improving the equilibrium representations. All experiments with MDEQs use the limited-memory version of Broyden’s method in both forward and backward passes, and the root solvers are stopped whenever 1) the objective value reaches some predetermined threshold  $\varepsilon$  or 2) the solver’s iteration count reaches a limit  $T$ . On large-scale vision benchmarks (ImageNet and Cityscapes), we downsample the input twice with 2-strided convolutions before feeding it into MDEQs, following the common practice in explicit models [236, 264]. We use the cosine learning rate schedule for all tasks [156].

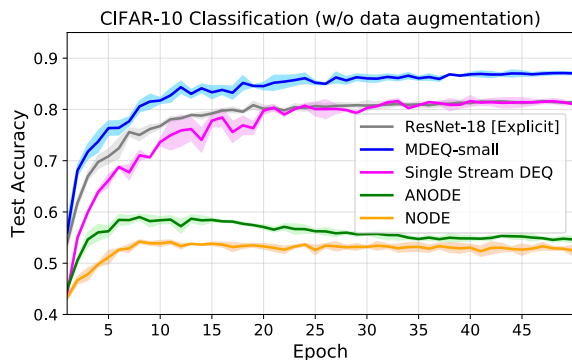
#### 3.3.1 Comparing with Prior Implicit Models on CIFAR-10

Following the setting of Dupont et al. [67], we run the experiments on CIFAR-10 classification (without data augmentation) for 50 epochs and compare models with approximately the same number of parameters. However, unlike the ODE-based approaches, we do not perform downsamplings on the raw images before passing the inputs to the MDEQ solver (so the highest-resolution stream stays at  $32 \times 32$ ). When training the MDEQ model, *all* resolutions are used for the final prediction: higher-resolution streams go through additional downsampling layers and are added to the lowest-resolution output to make a prediction (i.e., a form of auxiliary loss).

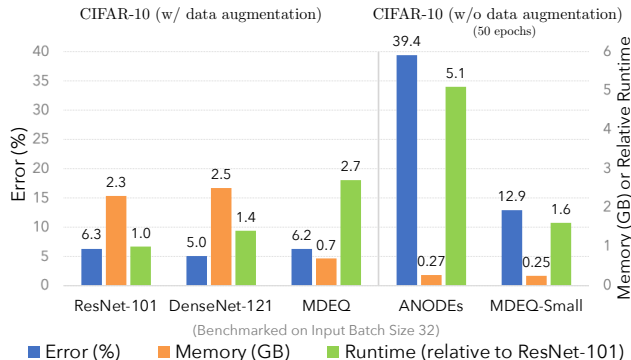
The results of MDEQ models on CIFAR-10 image classification are shown in Table 3.1. Compared to NODEs [45] and Augmented NODEs [67], a small MDEQ with a similar parameter count improves accuracy by more than 20 percentage points: an error reduction by *more than a factor of 2*. MDEQ also improves over the single-stream DEQ (applied at the highest resolution). The training dynamics of the different models are visualized in Figure 3.5a. Finally, a larger MDEQ matches and even exceeds the accuracy of a ResNet-18 with the same capacity: the first time such performance has been demonstrated by an implicit model.

Table 3.1: Evaluation on CIFAR-10. Standard deviations are calculated on 5 runs.

	Model Size	Accuracy
CIFAR-10 ( <i>without</i> data augmentation)		
Neural ODEs [67]	172K	53.7% $\pm$ 0.2%
Aug. Neural ODEs [67]	172K	60.6% $\pm$ 0.4%
Single-stream DEQ [18]	170K	82.2% $\pm$ 0.3%
ResNet-18 [96] [Explicit]	170K	81.6% $\pm$ 0.3%
<b>MDEQ-small (ours)</b>	<b>170K</b>	<b>87.1%</b> $\pm$ 0.4%
CIFAR-10 ( <i>with</i> data augmentation)		
ResNet-18 [96] [Explicit]	10M	<b>92.9%</b> $\pm$ 0.2%
<b>MDEQ (ours)</b>	<b>10M</b>	<b>93.8%</b> $\pm$ 0.3%



(a) Training dynamics of implicit models



(b) Runtime and memory consumption on CIFAR-10

Figure 3.5: Left: test accuracy as a function of training epochs. Right: MDEQ-Small and ANODEs correspond to the settings and results reported in Table 3.1. For all metrics, lower is better.

Table 3.2: Evaluation on ImageNet classification with top-1 and top-5 accuracy. MDEQs were trained for 100 epochs.

	Model Size	top1 Acc.	top5 Acc.
AlexNet [130]	238M	57.0%	80.3%
ResNet-18 [96]	13M	70.2%	89.9%
ResNet-34 [96]	21M	74.8%	91.1%
Inception-V2 [110]	12M	74.8%	92.2%
ResNet-50 [96]	26M	75.1%	92.5%
HRNet-W18-C [236]	21M	76.8%	93.4%
Single-stream DEQ + global pool [18]	18M	72.9%	91.0%
<b>MDEQ-small (ours) [Implicit]</b>	18M	75.5%	92.7%
ResNet-101 [96]	52M	77.1%	93.5%
W-ResNet-50 [259]	69M	78.1%	93.9%
DenseNet-264 [103]	74M	79.7%	94.8%
<b>MDEQ-large (ours) [Implicit]</b>	63M	77.5%	93.6%
Unrolled 5-layer <i>MDEQ-large</i>	63M	75.9%	93.0%
<b>MDEQ-XL (ours) [Implicit]</b>	81M	79.2%	94.5%

Table 3.3: Evaluation on Cityscapes *val* segmentation. “\*” marks the current SOTA. Higher mIoU (mean Intersection over Union) is better.

	Backbone	Model Size	mIoU
ResNet-18-A [152]	ResNet-18	3.8M	55.4
ResNet-18-B [152]	ResNet-18	15.24M	69.1
MobileNetV2Plus [200]	MobileNetV2	8.3M	74.5
GSCNN [222]	ResNet-50	-	73.0
HRNetV2-W18-Small-v2* [236]	HRNet	4.0M	76.0
<b>MDEQ-small (ours) [Implicit]</b>	MDEQ	7.8M	75.1
U-Net++ [267]	ResNet-101	59.5M	75.5
Dilated-ResNet [257]	D-ResNet-101	52.1M	75.7
PSPNet [264]	D-ResNet-101	65.9M	78.4
DeepLabv3 [41]	D-ResNet-101	58.0M	78.5
PSANet [265]	ResNet-101	-	78.6
HRNetV2-W48* [236]	HRNet	65.9M	81.1
<b>MDEQ-large (ours) [Implicit]</b>	MDEQ	53.0M	77.8
<b>MDEQ-XL (ours) [Implicit]</b>	MDEQ	70.9M	80.3

### 3.3.2 ImageNet Classification

We now test the ability of MDEQ to scale to a much larger dataset: ImageNet [62]. As with CIFAR-10 classification, we add a shallow classification layer after the MDEQ module to fuse the equilibrium outputs from different scales, and train on a combined loss.

We benchmark both a small MDEQ model and a large MDEQ to provide appropriate comparisons with a number of reference models, such as ResNet-18, -34, -50, and -101 [96]. Note that MDEQ has only *one layer* of residual blocks followed by multi-resolution fusion. Therefore, to match the capacity of standard explicit models, we need to increase the feature dimensionality within MDEQ. This is accomplished mainly by adjusting the width of the convolutional filter within the residual block (see Figure 3.2).

Table 3.2 shows the accuracy of two MDEQs (of different sizes) in comparison to well-known reference models in computer vision. MDEQs are remarkably competitive with strong explicit models. For example, a small MDEQ with 18M parameters outperforms ResNet-18 (13M parameters), ResNet-34 (21M parameters), and even ResNet-50 (26M parameters). A



Figure 3.6: Examples of MDEQ-large segmentation results on the Cityscapes [53] dataset.

larger MDEQ (64M parameters) reaches the same level of performance as ResNet-101 (52M parameters). This is far beyond the scale and accuracy levels of prior applications of implicit modeling.

### 3.3.3 Cityscapes Semantic Segmentation

After training on ImageNet, we train *the same MDEQs* for semantic segmentation on the Cityscapes dataset [53]. When transferring the models from ImageNet to Cityscapes, we directly use the highest-resolution equilibrium output  $\mathbf{z}_1^*$  to train on the highest-resolution loss. Thus *MDEQ is its own “backbone”*. We train on the Cityscapes `train` set and evaluate on the `val` set. Following the evaluation protocol of Zhao et al. [265] and Wang et al. [236], we test on a single scale with no flipping.

MDEQs attain remarkably high levels of accuracy. They come close to the current state of the art, and match or outperform well-known and carefully architected explicit models that were released in the past two years. A small MDEQ (7.8M parameters) achieves a mean IoU of 75.1. This improves upon a MobileNetV2Plus [200] of the same size and is close to the SOTA for models on this scale. A large MDEQ (53.5M parameters) reaches 77.8 mIoU, which is within 1 percentage point of highly regarded recent semantic segmentation models such as DeepLabv3 [41] and PSPNet [264], whereas a larger version (70.9M parameters) surpasses them. It is surprising that such levels of accuracy can be achieved by a “shallow” implicit model, based on principles that have not been applied to this domain before. Examples of semantic segmentation results are shown in Fig. 3.6.

### 3.3.4 Runtime and Memory Consumption

We provide a runtime and memory analysis of MDEQs using CIFAR-10 data, with input batch size 32. Since prior implicit models such as ANODEs [67] are relatively small, we provide results for both MDEQ and MDEQ-small for a fair comparison. All computation speeds are benchmarked

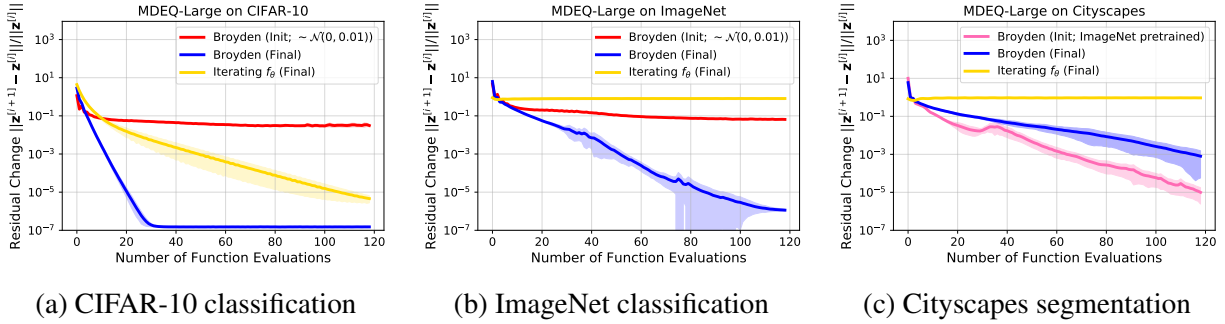


Figure 3.7: Plots of MDEQ’s convergence to equilibrium (measured by  $\frac{\|\mathbf{z}^{[i+1]} - \mathbf{z}^{[i]}\|}{\|\mathbf{z}^{[i]}\|}$ ) as a function of the number of times we evaluate  $f_\theta$ . As input image resolution grows (from CIFAR-10 to Cityscapes), MDEQ takes more steps to converge with (L-)Broyden’s method. Standard deviation is calculated on 5 randomly selected batches from each dataset.

relative to the ResNet-101 model (about 150ms per batch) on a single RTX 2080 Ti GPU. The results are summarized in Figure 3.5b.

MDEQ saves more than 60% of the GPU memory at training time compared to explicit models such as ResNets and DenseNets, while maintaining competitive accuracy. Training a large MDEQ on ImageNet consumes about 6GB of memory, which is mostly used by Broyden’s method. This low memory footprint is a direct result of the analytical backward pass. Meanwhile, MDEQs are generally slower than explicit networks. We observe a  $2.7\times$  slowdown for MDEQ compared to ResNet-101, a tendency similar to that observed in the sequence domain [18]. A major factor contributing to the slowdown is that MDEQs maintain features at all resolutions throughout, whereas explicit models such as ResNets gradually downsample their activations and thus reduce computation (e.g., 70% of ResNet-101 layers operate on features that are downsampled by  $8\times 8$  or more). However, when compared to ANODEs with 172K parameters, an MDEQ of similar size is  $3\times$  faster while achieving a  $3\times$  error reduction.

We will show in Chapter 4 that such slowdown can be significantly mitigated by properly regularizing the multiscale equilibria to be easier to solve for. By encouraging the MDEQ model to optimize for simpler/stabler underlying dynamical systems, we can effectively reduce the NFEs needed for these implicit networks even on large-resolution images (like in ImageNet).

### 3.3.5 Equilibrium Convergence on High-resolution Inputs

As we scale MDEQ to higher-resolution inputs, the equilibrium solving process becomes more challenging. This is illustrated in Figure 3.7, where we show the equilibrium convergence of MDEQ on CIFAR-10 (low-resolution), ImageNet (medium-resolution) and Cityscapes (high-resolution) images by measuring the change of residual with respect to the number of function evaluations. We empirically find that (limited-memory) Broyden’s method and multiscale fusion both help stabilize the convergence on high-resolution data. For example, in all three cases, Broyden’s method (blue lines in Figure 3.7) converges to the fixed point in a more stable and efficient manner than simply iterating  $f_\theta$  (yellow lines). We refer interested readers to further analysis of the multiscale convergence behavior in Bai et al. [19].

## 3.4 Discussion

In this chapter, we introduced multiscale deep equilibrium models (MDEQs): a new class of implicit architectures for domains characterized by high dimensionality and multiscale structure. Unlike prior implicit models, an MDEQ solves for and backpropagates through synchronized equilibria of multiple feature representations at different resolutions. A single MDEQ can be used for different tasks, such as image classification and semantic segmentation. Our experiments demonstrate for the first time that “shallow” implicit models can scale to practical computer vision tasks and achieve competitive performance that matches explicit architectures characterized by sequential processing through deeply stacked layers.

The remarkable performance of such multiscale implicit models brings up a step closer to the core question that we hope to challenge in this thesis. A longstanding assumption that people have had for deep learning is that we *must* have architectural hierarchy (i.e., layer stacking) in order to achieve feature hierarchy (i.e., resolutions) [136]. **We show that this is not correct.** Even without layers, MDEQ models show, we are able to effectively represent feature hierarchy.



## **Part II**

# **Challenges and Potential Solutions to the Equilibrium Approach**



# Chapter 4

## Challenges to the DEQ Approach

In the previous chapters, we briefly went through the underlying formulation of DEQ models, how it represents an “infinite-level” network, and how we can capture feature hierarchy without architectural hierarchy by simultaneous feature equilibria. This equilibrium approach, in particular, is compatible with the designs of modern structured layers and thus demonstrates strong modeling power and scalability, across a wide range of realistic tasks.

However, in this chapter (and as a part of this thesis’ thorough introduction on the equilibrium approaches to deep learning), we divert the attention to the multiple *challenges* of the DEQ approach. In particular, as we enjoy the simplicity of an “layer-less” formulation, what are some of the *new* issues that we could encounter? We outline three main challenges. First, despite their memory efficiency, DEQs are also slower than conventional deep networks that achieve the same level of accuracy. Second, the number of iterations required to solve for the equilibrium quickly grows over the course of training, indicating a trend for approaching instability. Third, the DEQ model is sensitive to architectural choices, and sometimes even small modifications could break the model’s stability of convergence (e.g., the position of layer normalization [13]). Some recent works have tackled this third issue by exploiting *provably convergent layers* via, e.g., monotone operator splitting theories [244], Lipschitz boundedness [192] or an underlying optimization problem [229]. However, these structural solutions rely extensively on specific layer parameterizations, rendering DEQ models unscalable and even more inflexible.

This chapter starts by summarizing and providing empirical evidence on all of these downsides of the equilibrium networks that have so far thwarted many from extending DEQs to both broader applications and more architectural variants. As we shall see in the rest of this thesis, these issues can be mitigated by numerous different approaches (e.g., regularization, recycle computation, approximate gradient, etc.) by exploiting the properties of these implicit models. Therefore, in the second half of this chapter, we will first describe a *regularization-based solution* that directly seeks to improve on DEQ models’ stability, efficiency and flexibility. Importantly, while DEQs have adopted regularizations directly borrowed from explicit deep networks (e.g., recurrent dropout [82] for better generalization), we introduce a simple and theoretically-motivated Jacobian regularization pursuant to DEQ models’ implicitness. We discuss in detail how this Jacobian regularization relates to the contractivity of equilibrium models’ forward non-linear system and backward linear system, and is thus able to effectively stabilize not only their forward but also backward dynamics. There are two immediate benefits of such resulting stability. First, solving

a DEQ requires far fewer iterations than before, which makes regularized DEQs significantly faster than their unregularized counterparts. Second, this model class becomes much less brittle to architectural variants that would otherwise break the DEQ.

We validate the proposed regularization by experiments on both toy-scale synthetic tasks and large-scale real datasets across domains: word-level language modeling on WikiText-103 [164] and high-resolution image classification on the full ImageNet dataset [62]. Empirically, our regularized DEQs are generally  $2\times$  to  $3\times$  faster than prior DEQs, and can be accelerated to be as fast as explicit deep networks in some cases (e.g., ResNets-101 and DenseNets-264). With their  $O(1)$  memory footprint, this further establishes these implicit, equilibrium approaches as a strong competitor to explicit deep architectures.

This chapter is primarily based on a work that appeared in ICML 2021 [20]. We will additionally discuss and compare in Chapter 5, 6 and 7 some other potential solutions to tackle the challenges described in this chapter.

## 4.1 Related Work on Regularizing Implicit Models

Just like explicit deep networks, implicit networks can overfit to the dataset; but additionally, they can also become unstable. For instance, Neural ODEs are essentially modeling infinitesimal steps of a residual block [39, 96], and Grathwohl et al. [91] found that weight decay & spectral normalization [171] are useful (though expensive) in reducing the rapidly growing number of functional evaluations (NFEs) needed to solve for the ODE endpoint. On the other hand, large-scale DEQ networks that we introduced in Chapter 2 and 3 have adopted techniques like weight normalization [199], recurrent (variational) dropout [82], and group normalization (GN) [246] for preventing overfitting and divergence. Nonetheless, all these methods are borrowed from explicit deep networks, where they have long been known to work well. They do not exploit the implicitness of implicit models.

More recently, a few different regularization methods have been introduced to specifically fix the numerous issues of the vanilla Neural ODE and continuous normalizing flow models, such as augmenting the hidden state [67] and regularizing higher-order time derivatives [121]. These methods directly leverage the dynamical system view of Neural ODEs. However, due to the inherent challenge of solving high-dimensional ODEs, these accelerated Neural ODE models can still easily take  $> 100$  forward iterations even on MNIST classification [121], and even more for their backward pass. In comparison, DEQs scale better to high-dimensional tasks (e.g., 25-30 iterations on ImageNet, see Chapter 3) and complex  $f_\theta$  (e.g., a Transformer layer, see Chapter 2). But such extra complexities also make DEQ models harder to regularize; e.g., simply resorting to weight decay doesn't fix the instability of DEQs (see Bai et al. [20]).

The method we will present in this chapter is closely connected to the many prior works that study Jacobian/gradient regularization [66, 77, 100, 146, 178], though these were also motivated differently. Specifically, Hoffman et al. [100], Sokolić et al. [213] regularized the input-output Jacobians of the entire (very deep) explicit classification networks to increase the prediction margin in a robust learning setting (and are thus expensive). Finlay et al. [77] was inspired by a kinetic energy view and possible overfitting of a training-time dynamical system. The method of Linsley et al. [146] targeted (for a Jacobian  $J$ ) a Lipschitzness level  $\lambda$ , used  $\max_i(\mathbf{1}^\top J)_i$  to

approximate the matrix 1-norm, and proposed loss  $L = \|(\mathbf{1}^\top J - \lambda)^+\|_2$ . Yet this approximation is in fact problematic, as it does not provably bound the spectral radius (i.e., stability) at all. For example, matrix

$$J = \begin{bmatrix} 2 & -2 \\ -2 & 2 \end{bmatrix}$$

has loss  $L = 0$  and yet an eigenvalue of 4 (we also empirically verify that this method does not help DEQ models, exactly due to this issue).

## 4.2 The Equilibrium Models’ Challenges and Discontents

Despite the DEQ models’ success in some very challenging tasks, such as Cityscapes semantic segmentation [19, 53], these models suffer from multiple serious downsides. In this section, we provide a summary of some of these problems. While many of them directly lead to our subsequent discussion on the need for regularization, we also believe such systematic discussion in this thesis provides a useful overview for potential future research on further addressing these issues. (We note that after the initial publication of Bai et al. [20], which this chapter is primarily based on, numerous concurrent and follow-up work have proposed different ways of addressing them; e.g., [81, 181, 229, 240]).

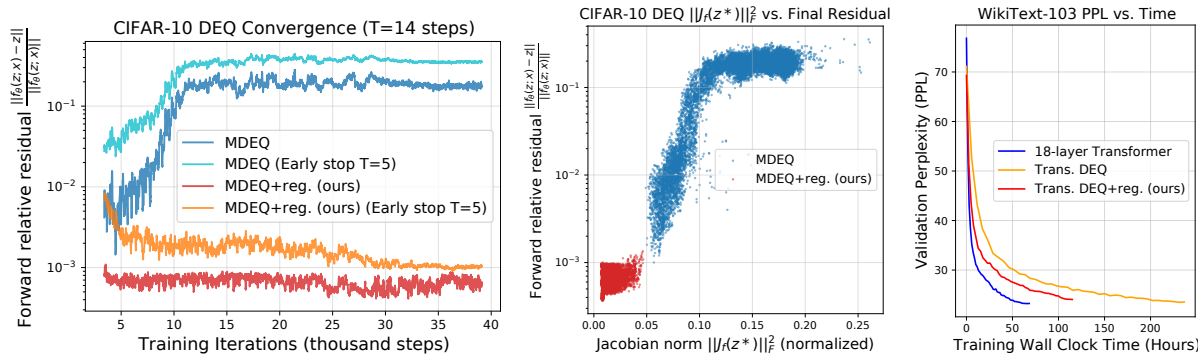
### 4.2.1 Existence and Uniqueness of the Fixed Point

The applications of large-scale DEQ models in chapter 2 and 3 demonstrates how the equilibrium approaches are compatible with modern, structured layers and are thus more applicable to domains where deep learning has been traditionally successful. However, unlike other implicit models such as Neural ODEs, DEQ networks do not have a unique trajectory, and are not generally guaranteed to converge (since the parameters  $\theta$  are constantly updated by SGD). We note that some recent works have indeed begun to examine the stability of the equilibrium models by exploiting provably convergent layers via monotone operator splitting theories [244], Lipschitz boundedness [81, 192], and the Banach fixed-point theorem [120, 184], etc. However, these structural solutions rely extensively on specific layer parameterizations, connections and components, rendering DEQ models unscalable and frequently less flexible.

### 4.2.2 Growing Instability

As previously discussed in Chapter 2, although a DEQ network has no “depth”, a relevant measure of computational efficiency is the number of function evaluations (NFEs) of the layer  $f_\theta(\mathbf{z}; \mathbf{x})$  used by the iterative root solver (e.g., Broyden’s method [34]).

However, one common phenomenon to all prior works on DEQs is that the fixed points are increasingly harder to solve for over the course of model training. In other words, as a DEQ’s performance gradually improves during training, the NFE required to converge to the same threshold  $\varepsilon$  rapidly grows. This observation has been made on different instantiations of equilibrium networks, and regardless of whether the model is provably convergent or not (e.g., [18, 244], where a DEQ at the end of training can take  $> 3\times$  more iterations; also see



(a) The relative residual of an unregularized DEQ’s final output gets worse over training. (b) DEQ’s convergence residual vs. Jacobian norm  $\|J_f\|_F^2$  (same setting as Fig. 4.1a) (c) DEQ training on language modeling as a function of time.

Figure 4.1: Visualizations of DEQ models’ instability and inefficiency problems. “Ours” refers to the regularized DEQ models, which will be introduced in Sec. 4.3.

Fig. 2.4 in chapter 2). Intuitively, such tendency to approach “critical stability” implicitly characterizes an inclination of the model to learn “deeper” networks; so it is unsurprising that unregularized training will keep driving it in this direction. But as a result, the dynamical system only becomes more and more brittle. In practice, one might circumvent this issue by setting a maximum NFE limit so that the solver stops after  $T$  steps, but this could still be risky as the convergence gets more unstable/critical, such a hard stop for the solver cannot guarantee that we are close enough to the fixed point. In the backward pass, for instance, we may consequently be training DEQs with very noisy or even wrong gradients. A similar issue exists for Neural ODEs, though these cannot easily be hard-stopped like DEQs due to the need to accurately trace the ODE flows to their endpoints.

We illustrate this issue on CIFAR-10 in Fig. 4.1a. One can easily see that both forward and backward estimates of the fixed points gets increasingly worse with the training steps (and eventually plateaus in an unstable region where the model keeps yielding bad gradients). Such growing instability is also reflected empirically in the growth of Jacobian norm at equilibrium; i.e.,  $\left\| \frac{\partial f_\theta(z^*; x)}{\partial z^*} \right\|_F$  (see Figure 4.1b), which we discuss further in the later parts of this chapter. Moreover, interestingly, while these plots might suggest simple regularizations like weight decay, we show later that weight decay often makes this stability issue worse for equilibrium networks, and even leads to divergence.

### 4.2.3 Inefficiency Compared to Explicit Networks

A direct ramification of such increase in iterations required (see Section 4.2.2) is the significant increase in both training and inference time for DEQ models.

One advantage of DEQs, as noted previously in Sec. 2.4 of Chapter 2, is that the forward trajectory need not strictly reach the equilibrium. Therefore in a certain sense, we could trade performance for efficiency by stopping at a “good enough” estimate of the equilibrium. However, due to the growing instability problem, this could still be increasingly costly. This causes the existing DEQs to be significantly slower than their explicit network counterparts of comparable

size and performance. E.g., a DEQ-Transformer [18] is over  $3\times$  slower than a deep Transformer-XL [58]; a multiscale DEQ [19] is over  $4\times$  slower than ResNet-101 on ImageNet and over  $7\times$  slower on Cityscapes. Despite their memory efficiency, such slowdown is a roadblock to wider deployment of this class of models in practice. In Figure 4.1c, we visualize this slowdown on the validation set of WikiText-103 language modeling [164] (with comparable model sizes and number of training steps).

We will propose a regularization-based approach to alleviate this issue in this chapter, which aims to improve the dynamical systems that DEQ models learn. But as we shall see, DEQ models’ efficiency could depend strongly on a number of factors, such as the quality of gradient (Chapter 5), the choice of solver (Chapter 6), and the input complexities (Chapter 7 and 8), which suggest a number of different solutions that frequently complement each other.

### 4.2.4 Brittleness to Architectural Choices

The need to have a relatively stable DEQ in order to train it via the implicit function theorem also calls for more careful attention in designing the layer  $f_\theta$ . For example, the largest-scale DEQs [18, 19] that we have presented so far in this thesis all had normalizations [13, 246] at the end of the layer to constrain the output range. How important are these architectural choices? We demonstrate the brittleness of DEQs by ablative studies on the use of layer normalization (LN) or weight normalization (WN) in the DEQ-Transformer model on the large-scale WikiText-103 language modeling task. Specifically, we compare the use of the two most popular Transformer layer designs in the DEQ framework: *pre-LN* and *post-LN*, which simply inserts the LN layers at different parts of the block (see Figure 4.2). These two settings have been extensively studied, used, and compared in the literature [14, 65, 151, 233, 251].

The result is shown in Figure 4.3. Without layer normalization at the end (pink line), a DEQ network (given a max limit NFEs) quickly diverges after 25K training iterations, reflected in both forward and backward divergences. Similarly, without weight normalization (orange line), the model becomes unstable more quickly, with fixed-point solver collapse at around 18K iterations. The original DEQ-Transformer [18] (blue line in Figure 4.3), although not diverged, still suffers from the same increased instability problem as described in Section 4.2.2. These plots are strong indicators that while equilibrium networks work on large scales, they are also relatively inflexible, brittle, and reliant on meticulous architectural designs.

### 4.2.5 Hidden Cost: Choice of Solver

Although DEQ models enjoy constant memory consumption during training time and can use any black-box fixed point solvers in the forward and backward passes, a commonly neglected cost is that introduced by the choice of solver. For example, in Broyden’s method [34] which Bai et al. [18, 19] used, the inverse Jacobian  $J^{-1}$  is approximated by low-rank updates on the identity

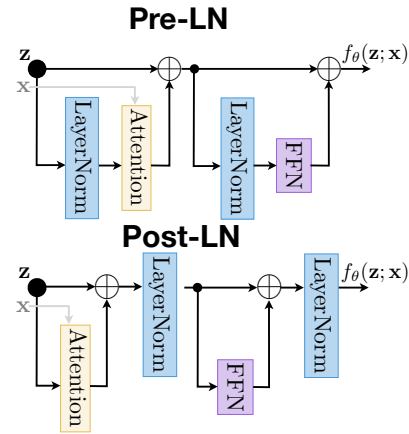
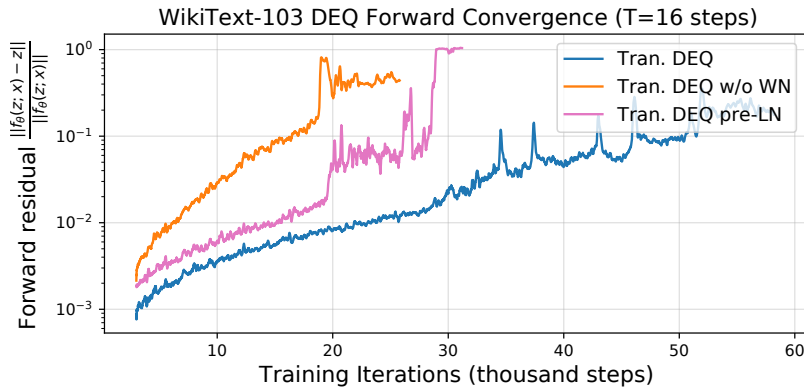
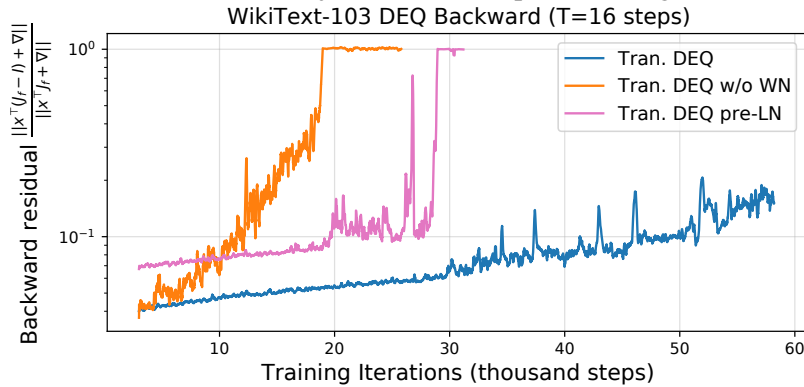


Figure 4.2: Pre- vs. post-LN DEQ-Transformer layer



(a) Forward final objectives of fixed-point convergence



(b) Backward final objectives of fixed-point convergence

Figure 4.3: Comparing different architectural modifications of a DEQ-Transformer (first 60K steps). DEQ models are brittle: even slight modifications such as changing the whereabouts of LayerNorm (see Fig. 4.2) or removing weight normalization can cause the model to quickly diverge during training.

matrix, of the form  $J^{-1} \approx -I + \sum_{i=1}^n \mathbf{u}^{[i]} \mathbf{v}^{[i]\top} = -I + UV^\top$ . As another example, Anderson acceleration (AA) [8] stores and uses the past  $m$  iterations ( $\mathbf{z}^{[n-1]}, \dots, \mathbf{z}^{[n-m]}$ ). Generally, more advanced algorithms (e.g., Newton’s method [11], Halley’s method [2], etc.) incur more costs as we use higher-order information in the fixed-point solving process. While these are typically tiny overheads compared to the number of evaluations of  $f_\theta$  itself, they still add to both computation and memory costs at both training *and inference time* (which conventional deep networks can avoid). Note that this cost depends strongly on the solver; for example, the simplest iterative “solver”  $\mathbf{z}^{[i+1]} = f_\theta(\mathbf{z}^{[i]}; \mathbf{x})$  wouldn’t have any memory cost, but suffers from bad convergence. This issue also highlights the value of faster and stabler convergence, which entails less memory storage overall (e.g., fewer Broyden steps).

## 4.2.6 Physical Laws of Layer Structure

Even if we can accelerate convergence by careful regularizations (see Sec. 4.3) or with the help of more advanced solvers, there are certain “physical laws” that we simply cannot bypass. For



example, if we apply a shallow convolutional DEQ whose layer has a receptive field  $5 \times 5$  on a large image (e.g.,  $1024 \times 1024$ ), it is hard to be able to reach the fixed point with just 6 iterations simply because the model’s receptive field may not broaden sufficiently to cover valuable context (and while we can certainly force contractivity, it would undoubtedly hurt performance substantially as the model will be effectively forced to pay attention to only local context). We shall see an example of this in Sec. 4.4.

However, we note that this physical law constraint depends strongly on the solver choice (e.g., while convolution is “local”, the inverse of a convolutional operator’s Jacobian is a “global” dense matrix, which theoretically makes Newton’s method converge more quickly even on larger images).

**Discussion.** We note that almost all of these challenges and issues are specific to the “*implicitness*” of these models (i.e., they don’t exist in explicit models), and typically do not matter much if our end goal is just to make the model work (e.g., when we are not concerned about speed and time), or to train on small scales. However, as the equilibrium approach starts to be applied on large-scale applications like Cityscapes segmentations, and be compared explicitly with conventional deep learning methods, the severity and accordingly the cost of these challenges also thwart us to advance implicit modeling to broader applications. It is therefore undoubtedly vital to address these challenges as we advance these implicit models to their next phase.

### 4.3 Stabilizing DEQ Models by Jacobian Regularization

In this section, we take a step further in this direction and introduce one strategy to help us cope with some of these challenges, thereby stabilizing and accelerating the equilibrium approach significantly. Specifically, as we have seen in Fig. 4.1a and 4.1b, increasing training steps lead to monotonically growing residual difference and the Jacobian norm at the equilibrium  $\mathbf{z}^*$ , suggesting a lack of proper regularization on DEQ models’ conditioning. We now describe how the Jacobian conditioning is directly related to the stability of equilibrium networks’ forward passes (locally) and backward passes (globally), and harness this relationship to stabilize and accelerate equilibrium models.

**The DEQ Jacobian.** We first recall from Chapter 2 that the forward pass of a DEQ network solves for the fixed point representation  $\mathbf{z}^*$  of a shallow layer  $f_\theta$ ; i.e.,  $\mathbf{z}^* = f_\theta(\mathbf{z}^*; \mathbf{x})$ . Then in the backward pass, one can directly differentiate through the equilibrium by

$$\frac{\partial \ell}{\partial (\cdot)} = \underbrace{\frac{\partial \ell}{\partial \mathbf{z}^*} (I - J_{f_\theta}(\mathbf{z}^*))^{-1}}_{\mathbf{u}^\top} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial (\cdot)}. \quad (4.1)$$

where we can leverage fast vector-Jacobian products to compute the implicit gradient  $\mathbf{u}^\top$  by a *linear* fixed-point system that only depends on this final equilibrium point (same as Eq. (2.16)):

$$\mathbf{u}^\top = \mathbf{u}^\top J_{f_\theta}(\mathbf{z}^*) + \frac{\partial \ell}{\partial \mathbf{z}^*}. \quad (4.2)$$

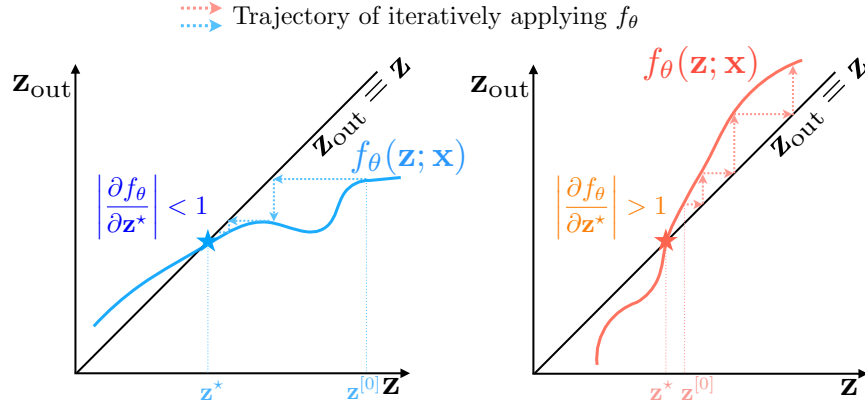


Figure 4.4: **Left:** when the slope is less than 1, even the simplest iterative application of  $f_\theta$  converges. **Right:** when slope  $> 1$ , the iterative approach may diverge or oscillate, but the fixed point still exists and can be solved for.

Now, consider the spectral radius of the Jacobian  $J_{f_\theta} \in \mathbb{R}^{d \times d}$  at the equilibrium:

$$\rho(J_{f_\theta}(\mathbf{z}^*)) = \rho(J_{f_\theta}(\mathbf{z}^*)^\top) = \max(|\lambda_1|, \dots, |\lambda_d|),$$

where  $\lambda_i$ s are eigenvalues. In both the forward and backward passes, this spectral radius directly affects how stable the convergence to the fixed point  $\mathbf{z}^*$  could be in its neighborhood. For instance, in the extreme case where we have a contractive  $\rho(J_{f_\theta}) < 1$ , by Lyapunov linearization theorem even the simplest iterative calls to  $f_\theta(\mathbf{z})$  (in forward, assuming good initial estimate) or  $g(\mathbf{u}) = \mathbf{u}^\top J_{f_\theta}(\mathbf{z}^*) + \frac{\partial \ell}{\partial \mathbf{z}^*}$  (in backward) could converge uniquely, even without advanced solvers. The linear system (4.2), in particular, would enjoy *global* asymptotic stability. However in practice, we don't always, and probably shouldn't, require such a strong contractivity on the dynamical system, which might significantly limit the representational capacity of the model. For example, as shown in Figure 4.4, a fixed point can exist even if  $\rho(J_{f_\theta}) > 1$ ; and we are still able to solve for them using the much stronger root solvers (e.g., Newton or quasi-Newton) than the naïve iterative stackings, which could oscillate or diverge.

**Jacobian regularization.** These connections between  $J_{f_\theta}(\mathbf{z}^*)$  and the forward/backward pass dynamics of DEQs motivate us to append a soft and auxiliary Jacobian term  $\rho(J_{f_\theta}(\mathbf{z}^*))$  to the training objective in order to regularize the model's conditioning. One way of doing this is by spectral normalization, essentially constraining  $\sigma(J_{f_\theta}) = \max_{\|\mathbf{v}\| \leq 1} \|J_{f_\theta} \mathbf{v}\|_2$ . However, explicitly writing out the huge Jacobian and then decomposing it (e.g., by SVD) can be computationally prohibitive, and in the context of DEQs, even power iterations [170, 171] are too expensive due to the successive vector-Jacobian product computations needed. Instead, we propose to regularize the Jacobian through its Frobenius norm since

$$\rho(J_{f_\theta}) \leq \sigma(J_{f_\theta}) \leq \sqrt{\text{tr}(J_{f_\theta} J_{f_\theta}^\top)} = \|J_{f_\theta}\|_F.$$

Importantly,  $\|J_{f_\theta}\|_F$  can be approximated via various unbiased estimators [108, 168, 230]. We adopt the classical Hutchinson estimator [108]; formally, for  $J_{f_\theta} \in \mathbb{R}^{d \times d}$ ,

$$\text{tr}(J_{f_\theta} J_{f_\theta}^\top) = \mathbb{E}_{\epsilon \in \mathcal{N}(0, I_d)} [\|\epsilon^\top J_{f_\theta}\|_2^2], \quad (4.3)$$

which we can approximate by Monte-Carlo estimation (i.e., sampling  $M$  i.i.d.  $\epsilon_i \in \mathcal{N}(0, I_d)$ ). Specifically, prior works [12, 196] have established that the relative error of this estimation diminishes with  $M^{-\frac{1}{2}}$ ; and if we compute the mean over a mini-batch size  $B$ , the overall relative error with respect to  $\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x}), \epsilon \in \mathcal{N}(0, I_d)} [\|\epsilon^\top J_{f_\theta}\|_2^2]$  is expected to further diminish by a factor of  $B^{-\frac{1}{2}}$  [100].

Indeed, empirically, we find that  $M = 1$  already works well since we use relatively large batch sizes. Since our backward iterations already involved computing multiple vector-Jacobian products  $\mathbf{u}^\top J_{f_\theta}$  (see Eq. (4.2)), computing Eq. (4.3) only adds a cost equivalent to that of  $M = 1$  backward steps. The eventual training objective is thus

$$\mathcal{L}_{\text{total}}(\mathbf{z}^*) = \mathcal{L}_{\text{orig}}(\mathbf{z}^*) + \gamma \frac{\|\epsilon^\top J_{f_\theta}(\mathbf{z}^*)\|_2^2}{d}, \quad \epsilon \in \mathcal{N}(0, I_d) \quad (4.4)$$

As we observed in Figure 4.1a, without regularization, a DEQ model that stops after a fixed number  $T$  of solver iterations exhibits increasingly poor convergence, accompanied by a growing  $\|J_{f_\theta}\|_F$  at these fixed points that empirically signals the growing instability. Therefore, by constraining the Jacobian’s Frobenius norm, we encourage DEQs to optimize for stabler and simpler dynamics whose fixed points are easier to solve for, without imposing strong structural constraints on the layer (such as in Fung et al. [81], Park et al. [184], Revay et al. [192], Winston and Kolter [244]).

## 4.4 Experiments

We validate the proposed regularization of DEQ models on multiple fronts. First, we visualize the effect of the proposed Jacobian regularization on a tiny DEQ trained on a synthetic 1D dataset. Second, importantly, we focus on how our method alleviates some of the core problems with DEQs outlined in Section 4.2. Then we show that our method scales to challenging high-dimensional tasks that were presented in the earlier chapters of this thesis: word-level language modeling with the WikiText-103 dataset [164] and image classification with CIFAR-10 and ImageNet [62]. We specifically compare our model with both unregularized DEQ networks and competitive explicit models (e.g., ResNet-101, Transformers) in the same settings, in terms of both space & time efficiency as well as performance. We also explore how Jacobian regularization helps stabilize DEQs over a wider range of architectural choices. Lastly, we perform some ablative studies.

As we found the Jacobian regularization could sometimes hurt performance (see Sec. 4.4.3), we only apply the proposed loss stochastically with a probability  $p$ , and gradually increase this  $p$  or the regularization strength  $\gamma$  (see Eq. (4.4)) over training. The memory and speeds reported are benchmarked across different models on the same setting (e.g., same batch size, sequence length, number of steps, etc.) with the same GPU. We provide more details regarding the tasks, hyperparameters, datasets, hardware, and extra experimental results in Bai et al. [20].

### 4.4.1 Visualization with Synthetic Data

We start by empirically visualizing the effect of our approach on a synthetic dataset. We generated 5096 scalar data pairs  $(x, y)$  using function  $y = h(x) = \frac{3}{2}x^3 + x^2 - 5x + 2\sin(x) - 3 + \delta$  (where  $\delta \in \mathcal{N}(0, 0.05)$ ), the shape of which is shown in Fig. 4.6). The data is subsequently split

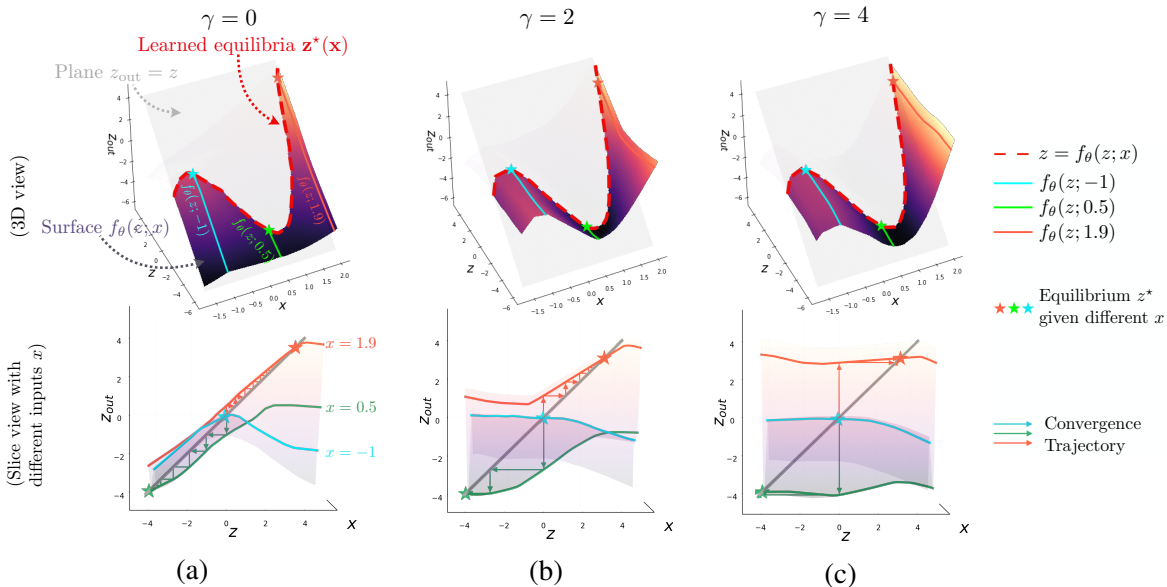


Figure 4.5: **Top:** the surface of the  $f_\theta(\mathbf{z}; \mathbf{x})$  layer, and the eventual learned equilibria  $\mathbf{z}^*(x)$  as a function of  $x$  (the red dashed line; compare this with the shape of Fig. 4.6). As  $\gamma$  grows, the surface is “lifted up” and becomes flat in the  $z$ -direction. **Bottom:** each unique input  $x$  defines a slice of the surface, and we perform fixed-point solving on this slice; larger  $\gamma$  values flatten the curve and significantly accelerate the convergence to equilibrium.

them into 4096 and 1000 training and validation samples, respectively. We then train a tiny DEQ with 200 parameters with the following structure:

$$f_\theta(\mathbf{z}; \mathbf{x}) = W_2^\top \text{ReLU}(W_1 \mathbf{z} + U \mathbf{x} + b), \quad \hat{y} = \mathbf{z}^* \quad (4.5)$$

where we used  $\mathbf{z}, \mathbf{x} \in \mathbb{R}$  and  $W_1, W_2, U \in \mathbb{R}^{50 \times 1}$ . The visualizations of the effect of the Jacobian regularization, with different weights  $\gamma$ , are shown in Figure 4.5. In particular, each input  $x$  defines a slice (i.e., cross-section) of the 3D surface  $z_{out} = f_\theta(z; x)$ ; for example, layer  $f_\theta(z; x)$  when input  $x = -1$  is highlighted in blue. After training, all three settings successfully learned the (almost) identical equilibrium function  $z^*(x)$  (highlighted by the red dashed line) that perfectly fits the target function  $h(x)$ ; but note that surfaces of  $f_\theta$  with  $\gamma = 2, 4$  are “lifted up” significantly compared to the unregularized ( $\gamma = 0$ ) DEQ, which has a steep slope (i.e., large spectral radius in 2D). This slope slows down the fixed-point convergence, as reflected by the zigzag patterns in lower Figure 4.5a. In contrast, the convergences for the  $\gamma > 0$  cases are much faster, and larger  $\gamma$  typically yields flatter surfaces around the equilibrium point.

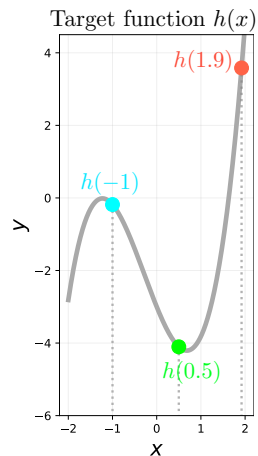


Figure 4.6: Target function  $h(x)$ .

#### 4.4.2 WikiText-103 Language Modeling

We next test the approach on much larger scales, first on the DEQ-Transformer instantiation (see Sec. 2.4) which uses a multi-head self-attention [233] layer as the underlying  $f_\theta(\mathbf{z}; \mathbf{x})$

Table 4.1: Evaluation on WikiText-103. PPL stands for Perplexity.  $t_{\text{train}}$  stands for relative training time. **JR** stands for Jacobian regularization. Memory benchmarked on batch size 15 and excludes the embedding layer at training time. <sup>†</sup> indicates unregularized model hard-stopped at inference time (while still trained with more NFEs).

	Model Size	Perplexity	$t_{\text{train}}$ (relative)	Train NFE	Valid. NFE	Memory
AWD-Quasi RNN [28]	159M	33.0	-	-	-	7.1GB
Relational Memory Core [201]	195M	31.6	-	-	-	-
Megatron-LM [208] [SOTA]	8300M	10.8	-	-	-	-
Transformer-XL (18-layer) [58]	110M	24.1	1×	-	-	9.0GB
DEQ-Transformer (Pre-LN) [18]	98M	[diverged]	N/A	30	N/A	N/A
DEQ-Transformer (Post-LN) [18]	98M	24.0	3.1×	30	30	3.9GB
DEQ-Transformer (Post-LN) <i>early stopped</i>	98M	29.2	3.1×	30	12	3.9GB
DEQ-Transformer (Post-LN) [18]	98M	26.0	2.2×	20	20	3.6GB
DEQ-Transformer (Post-LN) [18]	98M	[diverged]	N/A	15	N/A	3.6GB
<b>DEQ-Transformer (Pre-LN) + JR (ours)</b>	98M	24.5	1.5×	14	14	4.8GB
<b>DEQ-Transformer (Post-LN) + JR (ours)</b>	98M	24.9	1.4×	13	12	4.8GB
<b>DEQ-Transformer (Post-LN) + JR (ours) (trained on <math>T=300</math>)</b>	98M	23.8	2.2×	13	13	6.5GB

function. Although a DEQ-Transformer is able to perform competitively with a deep Transformer-XL [58] in terms of test perplexity, and consumes 60-70% less memory, it is also much slower (about 3×; see Figure 4.1c) and borders on instability. In Table 4.1, we demonstrate how the Jacobian regularization alleviates this. Compared to the original DEQ models, there are two major improvements. First, we significantly reduce the NFEs required for DEQ-Transformer models while maintaining competitive accuracy. Using the Transformer-XL as a time benchmark (1×), the speed of a DEQ-Transformer is significantly accelerated: training time goes from 3.1× to 1.5×. Second, the regularized DEQ model is more flexible with architectural choices. Whereas a Pre-LN DEQ-Transformer (see Figure 4.2) quickly diverges in training even in the presence of a large NFE threshold, the Jacobian regularization resolves this issue and stabilizes the forward/backward convergences consistently (see Figure 4.3 and Table 4.1), eventually reaching 24.5 perplexity. Moreover, while we can early-stop a well-trained unregularized DEQ model at inference time, it hurts generalization performance significantly (e.g., 29.2 ppl with 12 NFEs). Similarly, we find training with NFEs < 30 leads to increasingly bad generalization performance, and when NFEs drops below 20, model training frequently diverge as a result of extremely noisy gradients.

Like the original equilibrium networks, the regularized DEQs are memory efficient, consuming about 45% less training memory than Transformer-XL. Moreover, we find the Jacobian-regularized DEQs reduce over 50% memory consumption of the original DEQs at inference time (when both using Broyden’s method) due to faster/stabler convergence, suggesting its effectiveness in addressing the hidden solver cost issue discussed in Sec. 4.2.5.

### 4.4.3 CIFAR-10 and ImageNet Classification

We additionally conduct experiments on vision tasks using the recent multiscale deep equilibrium networks (MDEQ) [19], which drive multiple feature resolutions to their equilibria simultaneously. Because of the need to maintain high- and low-resoluntional feature maps at all iterative steps and generally higher channel dimensions in  $f_{\theta}$ , MDEQs are substantially slower than conventional

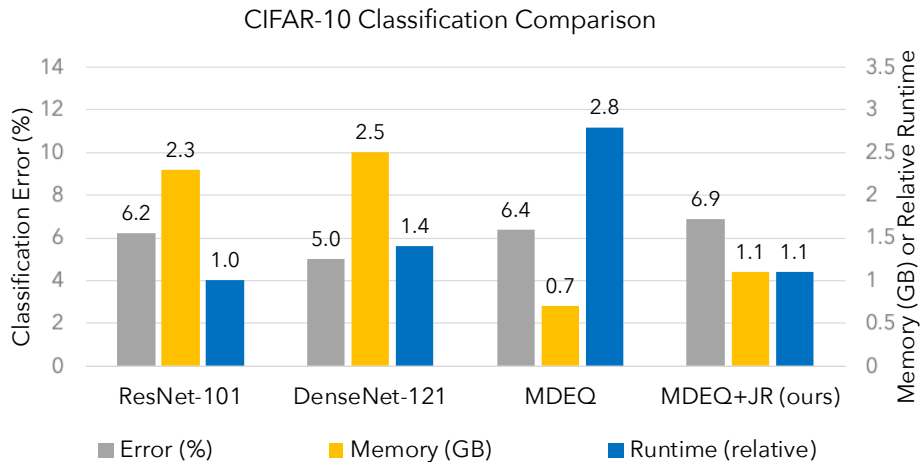


Figure 4.7: With the proposed regularization, DEQ models are competitive with popular explicit networks in accuracy, memory, and runtime. **Lower bars are better.**

networks like ResNets (which operate on progressively downsampled feature maps). This makes acceleration vital to broader adoption of multiscale implicit models.

The results of applying Jacobian regularization on multiscale DEQs for image classification are shown in Table 4.2. On CIFAR-10, whereas the unregularized DEQ models used 17 NFEs to reach the reported competitive level of performance, our DEQ with Jacobian regularization can converge well even within 6 iterations (in fact, we find smaller NFE values still trains, but significantly hurts generalization performance). This improvement is also obvious in Figure 4.1a and 4.1b, where we show that early stopping at threshold  $T = 6$  still yields good convergence with Jacobian regularization. On the much larger-scale ImageNet, where we deal with  $224 \times 224$  images, the factor of reduction in NFEs is not as strong (e.g., from 27 to 14 iterations, due to the receptive field issue; we’ll explain this next in Section 4.4.5) but still yields a roughly  $2\times$  acceleration. This shows that the Jacobian regularization is effective in large-scale computer vision tasks, and in the presence of multiple equilibrium points. However, we also note that as with DEQ-Transformers on WikiText-103, we notice a small slip in accuracy, which may be a result of constraining model parameterizations.

Table 4.2: Results on CIFAR-10 and ImageNet classification (standard deviation is calculated with 5 runs). <sup>†</sup> indicates unregularized model hard-stopped at inference time.

CIFAR-10 classification			
	Size	Accuracy	NFEs
ResNet-18 [96]	10M	93.0 ( $\pm 0.1\%$ )	-
ResNet-101 [96]	40M	93.8 ( $\pm 0.3\%$ )	-
DenseNet-121 [103]	8M	95.0 ( $\pm 0.1\%$ )	-
monotone DEQ [244]	1M	89.4 ( $\pm 0.2\%$ )	24
MDEQ [19]	10M	93.6 ( $\pm 0.2\%$ )	17
MDEQ <i>early stopped</i> <sup>†</sup>	10M	89.1%	6 <sup>†</sup>
<b>MDEQ + JR (ours) [19]</b>	10M	93.1 ( $\pm 0.3\%$ )	<b>6</b>
(Full) ImageNet classification			
	Size	Top-1 Acc.	NFEs
ResNet-18 [96]	13M	70.2%	-
Inception-V2 [110]	12M	74.8%	-
ResNet-50 [96]	26M	75.1%	-
ResNet-101 [96]	52M	77.1%	-
DenseNet-264 [103]	74M	79.7%	-
MDEQ-small [19]	18M	75.4%	27
MDEQ-large [19]	63M	77.5%	30
<b>MDEQ-small + JR (ours)</b>	17M	74.5%	14
<b>MDEQ-large + JR (ours)</b>	62M	76.8%	15

Figure 4.7 provides a visual comparison of different models with respect to three metrics: performance, inference speed, and training memory. These are reported on the CIFAR-10 dataset.

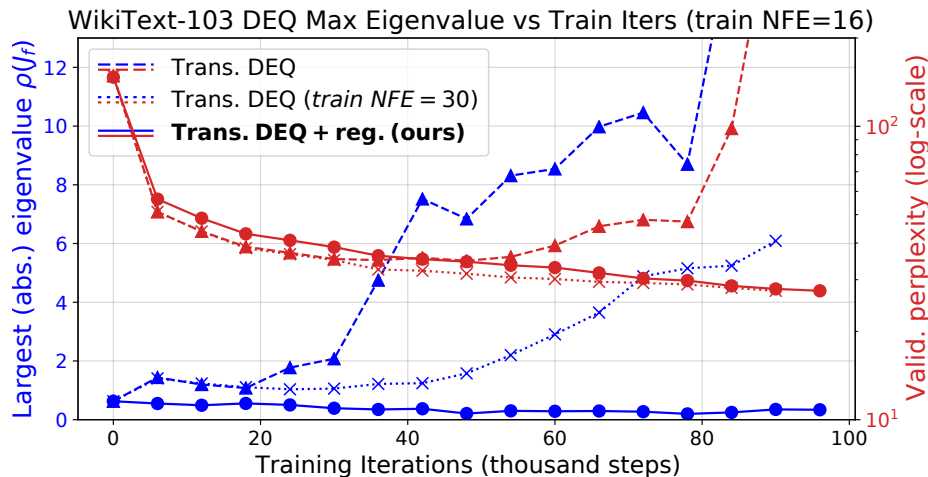


Figure 4.8: Empirical evidence of how our method constrains  $\rho(J_{f_\theta})$ . In contrast, insufficient NFEs (e.g.,  $T=16$ ) at training time cause a DEQ-Transformer model to explode early in the training phase.

Compared to the first version of MDEQ presented in Chapter 3, for the first time, we have an implicit-depth model that runs with a competitive level of speed and accuracy as large explicit networks such as ResNet-101, while consuming much less memory.

#### 4.4.4 Effect of Jacobian Regularization on $\rho(J_{f_\theta})$

In addition to the synthetic study, we also verify that the Jacobian regularization is indeed effectively constraining conditioning of  $J_{f_\theta}$ . Note that the underlying Jacobian matrices are large (e.g.,  $[(B \cdot 110K) \times (B \cdot 110K)]$  in WikiText-103, and  $[(B \cdot 198K) \times (B \cdot 198K)]$  in ImageNet with MDEQ-small) and checking their full spectrum would be infeasible. Therefore, we conduct a study that monitors the average spectral radius  $\rho(J_{f_\theta}(\mathbf{z}^*))$  (i.e., the largest absolute eigenvalue) on the validation set, over the first 100K steps of DEQ training on WikiText-103 using the power method [170]; see Fig. 4.8. Importantly, although  $\|J_{f_\theta}\|_F$  only upper-bounds the spectral radius (see Sec. 4.3), we verify that our proposed regularization does effectively constrain  $\rho(J_{f_\theta})$  (see ●/● paths in Fig. 4.8), thereby making deep equilibrium models more stable. In contrast, an unregularized DEQ with the same few NFEs explodes in both eigenvalue and shortly after also in perplexity (see ▲/▲ paths), and only works if we increase NFEs to 30 (see ×/× paths). In general, we empirically observe that training an unregularized DEQ with insufficient NFEs generally begets extremely noisy gradients, thus leading to faster destabilization and even divergence.

#### 4.4.5 Ablative Analysis and Limitations of the Approach

We continue our discussion with some empirical ablative studies. First, while Grathwohl et al. [91] found weight decay useful for regularizing ODE-based models’ NFEs, we found weight decay generally not effective in stabilizing DEQs and sometimes even counter-productive. This is illustrated in Figure 4.9, where after 50K steps the model started to diverge to  $> 500$  perplexity

and stopped improving. In addition, we also conduct an ablative experiment on how the Jacobian regularization strength  $\gamma$  affects the performance when we constrain NFEs to  $\leq 6$  at inference time, with results shown in Table 4.3 (CIFAR-10 dataset). In general, we find that if  $\gamma$  is too small, the final performance may be good but entails more NFEs. When  $\gamma$  is too large, the accuracy does quickly converge, but the constraint imposed on the model class is too strong and eventually hurts performance (e.g., since the training loss on CIFAR-10 usually overfits to almost 0 towards the end of training, which makes the Jacobian loss dominant instead).

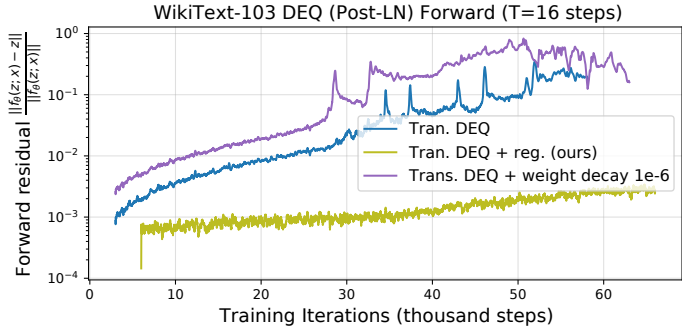


Figure 4.9: Adding weight decay to the DEQ-Transformer doesn’t help stabilize the convergence.

We also highlight two limitations of this approach. First, the addition of Jacobian regularization term does not fundamentally solve the growing instability problem, but only empirically alleviates it. This means that we have to be careful about balancing the main loss objective and this auxiliary objective (see Table 4.3). Second, even with the Jacobian regularization facilitates faster convergence, the physical law issue highlighted in Sec. 4.2.6 still cannot be easily bypassed. This explains why we need more NFEs on ImageNet than on CIFAR-10 (see Table 4.2); it also indicates that while our approach alleviates the brittleness to architectural choices, its effectiveness can still depend on the architecture. This makes global-context alternatives to ConvNets, such as self-attention-based vision layers (e.g., ViT [65, 154]) likely more appealing in the implicit model setting, which we leave for future work.

Table 4.3: Controlled experiments on the strength  $\gamma$  of the Jacobian regularization. The NFE value represents the “hard stop” threshold we set for the corresponding DEQ models at inference.

	NFE=1	NFE=2	NFE=3	NFE=4	NFE=5	NFE=6
$\gamma = 0.1$	82.4%	89.7%	91.9%	92.3%	92.7%	92.9%
$\gamma = 0.6$	<b>85.8%</b>	<b>91.5%</b>	<b>92.7%</b>	<b>93.0%</b>	<b>93.0%</b>	<b>93.1%</b>
$\gamma = 1.2$	84.4%	89.6%	92.2%	92.6%	92.7%	92.7%

## 4.5 Discussion

In this chapter, we summarized the weaknesses and challenges that these “new” equilibrium-based deep learning models face. We have specifically discussed the relationship between the spectral radius of the Jacobian and the stability of forward non-linear and backward linear systems of DEQ models, and provided empirical evidence of the poor conditioning of the Jacobian. This motivates our introduction of Jacobian regularization. Our experiments show that our method significantly alleviates the weaknesses of DEQs, yielding a  $> 2.5\times$  acceleration. This is a major step towards making implicit models more practical and suitable for large-scale real-world applications. Many of those challenges have led to numerous subsequent research explorations, only a tiny portion of which is covered in this thesis (indeed, the Jacobian regularization is just one of the many potential solutions). We discuss some of these other extensions of the DEQ approach next.



## Chapter 5

# Training DEQs with Lightweight Inexact Gradients

Implicit deep networks treat the evolution of the intermediate hidden states as a certain form of dynamics, such as fixed-point equations [18, 19] or ordinary differential equations (ODEs) [45, 67], which represents infinite-level hidden states. This allows us to formulate their forward passes as solving the underlying dynamical systems; e.g., by black-box and advanced root-finding algorithms (see Chapter 2). Importantly, DEQ models could directly differentiate through the final equilibrium point  $\mathbf{z}^*$  (see Sec. 2.2.2):

$$\frac{\partial \ell}{\partial (\cdot)} = \frac{\partial \ell}{\partial \mathbf{z}^*} (I - J_{f_\theta}(\mathbf{z}^*))^{-1} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial (\cdot)}. \quad (5.1)$$

However, this could still induce a heavy memory overhead [80, 155] in practice due to the Jacobian inverse term (highlighted in ). To solve this problem, we have so far relied on the relatively more efficient vector-Jacobian products and solve a Jacobian-based linear fixed-point equation for the backward pass of deep equilibrium models (see Eq. 2.16). But this could still be costly in practice as it relies on another iterative (linear) fixed-point solving process. Without techniques like the Jacobian regularizations introduced in Chapter 4, for example, it could take over 2 weeks to train the largest, state-of-the-art implicit models on ImageNet [198] with 8 GPUs.

In this chapter, we investigate a yet different solution from that proposed in Chapter 4: fast inexact gradients. We propose to directly replace the red Jacobian inverse term in Eq. 5.1 with a lightweight alternative, and directly circumvent the laborious computation that we (and many other work) have depended on so far to yield *the exact gradient* [18, 19, 92, 157, 244]. Importantly, we find that a first-order oracle that produces good gradient estimates is enough to efficiently and effectively train these layer-*less* implicit models (indeed, as we shall see, the fact that DEQ models have one layer is central to this development).

There are a few ways that we can achieve this. Specifically, we will primarily introduce the technique of *phantom gradient*, which is motivated by a Neumann series view of the implicit function theorem [128]. We will analyze the general condition under which these phantom gradients can provide a provably descent direction of the loss landscape, and propose two variations of them for the equilibrium framework (which are based on the the damped fixed-point unrolling and the Neumann series, respectively). Additionally, we show that the stochastic gradient

descent (SGD) algorithm based on the phantom gradient enjoys a sound convergence property as long as the relevant hyperparameters (e.g., the damping factor) are wisely chosen. At the end of our methodology section, we will also briefly introduce some of the other inexact gradient schemes concurrently studied by the community for DEQ models, one based on the forward Broyden matrix [189] (i.e., low-rank updates), and the other one based on the identity matrix [81].

We conduct an extensive set of synthetic, ablation, and large-scale experiments to both analyze the theoretical properties of the phantom gradient and validate its speedup and performances on various tasks, such as ImageNet [198] classification, Wikitext-103 [164] language modeling, and graph classification (COX2, PROTEINS [253]). Overall, our results suggest that: 1) these inexact gradients provides a provably descent gradient direction; 2) they are applicable to large-scale tasks and is capable of achieving a strong performance; and 3) they significantly shorten the training time needed for implicit models roughly by a factor of almost  $2\times$ .

We note that this is generally not possible for explicit, conventional neural networks (except for in certain constrained settings) where each layer decidedly depends on the later layers, which makes chain-rule backpropagation inevitable. This chapter is primarily based on a NeurIPS 2021 work [87] and a ICLR 2022 work [189].

## 5.1 Motivation: Inexact Gradients

A deep equilibrium network solves the solution  $\mathbf{z}^*$  to the following fixed-point equation:

$$\mathbf{z}^* = f_{\theta}(\mathbf{z}^*; \mathbf{x}). \quad (5.2)$$

(In this section, we start by assuming that the function  $\mathbf{z} \rightarrow f_{\theta}(\mathbf{z}; \mathbf{x})$  is a contraction with respect to  $\mathbf{z}$  so that its Lipschitz constant  $L_{\mathbf{z}} < 1$ . We note that this is a setting that has been frequently assumed to be the case in relevant DEQ literature [180, 192].) To train these models, we rely on the implicit function theorem (Eq. (5.1)). However, as we have seen in Chapter 2 and 4, this seemingly simple implicit differentiation suffer from two important issues. First, as the computation and storage of the Jacobian inverse can be prohibitively expensive in high-dimensional settings (e.g., over  $10^6 \times 10^6$  in language modeling, per sequence), we usually compute it by solving a corresponding linear system instead. This is still iterative and costly in nature, since each vector-Jacobian product is about as costly as a forward evaluation of  $f_{\theta}$ . Second, the Jacobian matrix (and thus the Jacobian-inverse) can be numerically unstable when encountering the ill-conditioning issue, which could lead to problems like growing instability and even training divergence (see Sec. 4.2.2).

On a parallel thread, the inexact gradient [25, 78, 81, 86, 204] has been applied in certain prior (constrained) learning protocol, like synthetic gradient [112] (which has been mostly applied to simple tasks such as MNIST classification). Some research has used a moderate gradient noise as a regularization approach [84], which has been shown to play a central role in escaping poor local minima and improving generalization ability [7, 245, 268]. This therefore motivates us to rethink the possibility of replacing the Jacobian-inverse term in the standard implicit differentiation with a cheaper and more stable counterpart as well: since DEQ models have only one (implicit) layer and is agnostic to the forward computation trajectory, it is likely that exact gradient estimate is not always required.

Broadly, for an implicit deep network with parameter  $\theta \in \mathbb{R}^p$  and equilibrium dimensionality  $\mathbf{z}^* \in \mathbb{R}^d$ , given a loss  $\ell$  we define the corresponding inexact gradient as:

$$\widehat{\frac{\partial \ell}{\partial \theta}} := \frac{\partial \ell}{\partial \mathbf{z}^*} \mathcal{A} \quad (5.3)$$

where the matrix  $\mathcal{A} \in \mathbb{R}^{d \times p}$  serves to replace the original  $\frac{\partial \mathbf{z}^*}{\partial \theta} = (I - J_{f_\theta}(\mathbf{z}^*))^{-1} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta}$  term in the implicit function theorem (IFT). It turns out that under certain general conditions of this approximation  $\mathcal{A}$ , these inexact gradients can be guaranteed valid in the deep equilibrium model context, which we provide below as a theorem.

**Theorem 4.** Let  $\sigma_{\max}$  and  $\sigma_{\min}$  be the maximal and minimal singular value of  $\frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta}$ . If

$$\left\| (I - J_{f_\theta}(\mathbf{z}^*)) \mathcal{A} - \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta} \right\| < \frac{\sigma_{\min}^2}{\sigma_{\max}}, \quad (5.4)$$

then the inexact gradient provides an ascent direction of the loss function  $\ell$ , i.e.,

$$\left\langle \widehat{\frac{\partial \ell}{\partial \theta}}, \frac{\partial \ell}{\partial \theta} \right\rangle > 0. \quad (5.5)$$

*Proof.* Denote  $G = \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta} \in \mathbb{R}^{d \times p}$ ,  $\mathbf{v}^\top = \frac{\partial \ell}{\partial \mathbf{z}^*} \in \mathbb{R}^{1 \times d}$  and  $\mathbf{u}^\top = \mathbf{v}^\top (I - J_{f_\theta}(\mathbf{z}^*))^{-1} \in \mathbb{R}^{1 \times d}$ . Let

$$E = (I - J_{f_\theta}(\mathbf{z}^*)) \mathcal{A} - G.$$

If  $\|E\| < \sigma_{\min}^2 / \sigma_{\max}$ , then

$$\left\langle \widehat{\frac{\partial \ell}{\partial \theta}}, \frac{\partial \ell}{\partial \theta} \right\rangle = \mathbf{v}^\top \mathcal{A} G^\top (I - J_{f_\theta}(\mathbf{z}^*))^{-T} \mathbf{v} = \mathbf{u}^\top (G + E) G^\top \mathbf{u} \quad (5.6)$$

$$\geq \|\mathbf{u}^\top G\|_2^2 - \|E\| \|G^\top\| \|\mathbf{u}\|^2 \geq (\sigma_{\min}^2 - \sigma_{\max}^2 \|E\|) \|\mathbf{u}\|^2 > 0 \quad (5.7)$$

which concludes the proof. ■

We can also understand inexact gradient as a replacement for the Jacobian inverse. Suppose we only replace the  $(I - J_{f_\theta}(\mathbf{z}^*))$  term with a matrix  $D \in \mathbb{R}^{d \times d}$  (i.e.,  $\mathcal{A} = D \frac{\partial f_\theta}{\partial \theta}$ ), then condition in inequality (5.4) can be equivalently reduced to the following simpler form:

$$\|(I - J_{f_\theta}(\mathbf{z}^*)) D - I\| < \frac{1}{\kappa^2} \quad (5.8)$$

where  $\kappa$  is the condition number of  $\frac{\partial f_\theta}{\partial \theta}$ .

**How practical is this theorem?** We show next two instantiations of this inexact gradient (which we call *phantom gradients*) for which we can verify the condition in Thm. 4 holds if the hyperparameters in those instantiations are wisely selected.

## 5.2 Instantiations: Phantom Gradients

Note that without altering the fixed point of Eq. (5.2), we could equivalently consider a damped variant naïve fixed point iteration with a damping factor  $\lambda < 1$ :

$$\mathbf{z}^{[i+1]} = \lambda f_\theta(\mathbf{z}^{[i]}; \mathbf{x}) + (1 - \lambda)\mathbf{z}^{[i]}, \quad i = 0, 1, \dots \quad (5.9)$$

(It should be easy to verify that Eq. (5.9) and Eq. (5.2) have the exact same set of fixed points.) Specifically, when  $\lambda$  is properly chosen and  $f_\theta$  is contractive, this infinite series is bounded and thus convergent, yielding the following **Neumann series** expansion of the Jacobian inverse term:

$$(I - J_{f_\theta}(\mathbf{z}^*))^{-1} = \left( I - \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \mathbf{z}^*} \right)^{-1} = \lambda(I + B + B^2 + B^3 + \dots) = \lambda \sum_{t=0}^{\infty} B^t \quad (5.10)$$

where  $B = \lambda J_{f_\theta}(\mathbf{z}^*) + (1 - \lambda)I$ . An equivalent way of interpreting this is, suppose we unroll Eq. (5.9) for a finite  $T$  steps (and assuming  $f_\theta$  is continuously differentiable w.r.t.  $\theta$  and  $\mathbf{z}^*$ ), then as  $T \rightarrow \infty$ , we have  $\lim_{T \rightarrow \infty} \frac{\partial \mathbf{z}^{[T]}}{\partial \theta} = (I - J_{f_\theta}(\mathbf{z}^*))^{-1} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta}$ . This suggests two slightly different variations (up to fixed-point numerical error; see discussion later) of how we could instantiate this inexact gradient. In both cases, we assume that we use some black-box fixed-point solver `RootFind` which yields a fixed point estimate  $\hat{\mathbf{z}}^*$  (which may **not** be exactly the same as the groundtruth fixed point  $\mathbf{z}^*$ , as in practice we stop the solver at some  $\varepsilon$  threshold for efficiency; see Sec. 2).

**Unrolling-based Phantom Gradient (UPG).** Setting  $\mathbf{z}^{[0]} = \hat{\mathbf{z}}^*$  (the fixed point estimate we obtain from the black-box solver), we perform  $\mathbf{z}^{[i+1]} = \lambda f_\theta(\mathbf{z}^{[i]}; \mathbf{x}) + (1 - \lambda)\mathbf{z}^{[i]}$  for  $i = 0, \dots, k - 1$ , and backpropagate through this  $k$ -step unrolling as usual.

Formally, this allows us to define matrix  $\mathcal{A} \in \mathbb{R}^{d \times p}$  in Thm. 4 in the following form:

$$\mathcal{A}_{k,\lambda}^{\text{unr}} = \lambda \sum_{i=0}^{k-1} \left[ \prod_{t=i+1}^{k-1} \left( \lambda J_{f_\theta}(\mathbf{z}^{[t]}) + (1 - \lambda)I \right) \right] \frac{\partial f_\theta(\mathbf{z}^{[i]}; \mathbf{x})}{\partial \theta} \quad (5.11)$$

Intuitively, we perform  $k$  naïve fixed-point iterations at the estimate point  $\hat{\mathbf{z}}^*$  we obtain, and perform backpropagation-through-time (BPTT) on this  $k$ -step unrolling. A PyTorch-style [185] pseudocode describing a sample implementation of UPG is shown in Alg. 1.

**Neumann-series-based Phantom Gradient (NPG).** We can also perform a  $k$ -step truncated Neumann series expansion with a  $B$  matrix (see Eq. 5.10) defined directly by the estimation  $\hat{\mathbf{z}}^*$  (without unrolling like in UPG at all). That is,

$$\mathcal{A}_{k,\lambda}^{\text{neu}} = \lambda(I + B + B^2 + \dots + B^{k-1}) \frac{\partial f_\theta(\hat{\mathbf{z}}^*)}{\partial \theta}, \quad \text{where } B = \lambda J_{f_\theta}(\hat{\mathbf{z}}^*) + (1 - \lambda)I \quad (5.12)$$

We refer interested readers to Appendix B in Geng et al. [87] for implementation of these two phantom gradients. Moreover, for a properly chosen  $\lambda$  (which impacts contractivity), we are able to ensure that condition (5.4) is satisfied when  $k$  is large.

---

**Algorithm 1** Unrolling-based phantom gradient (UPG), PyTorch-style

---

```
# solver: the solver to find  $\mathbf{z}^*$ , e.g., Broyden's method.
# func: the single layer  $f_\theta$  that defines the implicit model.
# z: the input injection  $\mathbf{x}$  to solve  $\mathbf{z}^* = f_\theta(\mathbf{z}^*; \mathbf{x})$ 
# h: the estimated solution  $\hat{\mathbf{z}}^*$  of the single layer.
# k: the UPG unrolling steps  $k$ .
# lambda_: the damping factor  $\lambda$ .
# training: a bool variable that indicates the training or
# inference stage.

# Forward pass (Backward pass is accomplished by automatic
# differentiation)
def forward(z, k, lambda_, training):
    with torch.no_grad():
        h = solver(func, z)

    if training:
        for _ in range(k):
            h = (1 - lambda_) * h + lambda_ * func(h, z)
    return h
```

---

**Discussion.** We would like to clarify that both the UPG and NPG are derived from a truncated form of an infinite series. However, we choose to differentiate between these two instantiations because in practice they can behave very differently with the quality of the fixed-point estimate  $\hat{\mathbf{z}}^*$  from the solver.

- When  $\hat{\mathbf{z}}^*$  is a bad estimation of the fixed point (e.g., when the growing instability issue outlined in Chapter 4 becomes severe), UPG turns into a truncated BPTT (T-BPTT) that is commonly used for RNNs, whereas NPG could be an extremely noisy (or wrong) gradient. This resembles the warmup phase that was used in Bai et al. [18] in DEQ-Transformer training, where we perform a 2-layer unrolling for the first  $\sim 10\text{K}$  training iterations.
- When  $\hat{\mathbf{z}}^*$  is a high-quality fixed-point estimation (i.e.,  $\|\hat{\mathbf{z}}^* - \mathbf{z}^*\| \rightarrow 0$ , which is typically the case when the dynamical system is very stable), UPG and NPG with the same  $k$  and  $\lambda$  should also expect to approach each other.

Moreover, we also characterize the impact of the two major hyperparameters on the precision and conditioning of the inexact gradient  $\mathcal{A}$ . Take NPG (Eq. (5.12)) as an example.

- (i) On the precision of the phantom gradient,
  - a large  $k$  makes the gradient estimate more accurate, as higher-order terms of the Neumann series are included; while
  - a small  $\lambda$  slows down the convergence of the Neumann series because the norm  $\|B\|$  increases as  $\lambda$  decreases.

(ii) On the conditioning of the phantom gradient,

- a large  $k$  impairs the conditioning of  $\mathcal{A}$  since the condition number of  $B^k$  grows exponentially as  $k$  increases; while
- a small  $\lambda$  helps maintain a small condition number of  $\mathcal{A}$  because the singular values of  $J_{f_\theta}(\hat{\mathbf{z}}^*)$  are “smoothed” by the identity matrix.

In a word, a large  $k$  is preferable for a more accurate  $\mathcal{A}$ , whereas a small  $\lambda$  contributes to the well-conditioning of  $\mathcal{A}$ . Practically, these hyperparameters should be selected in consideration of a balanced trade-off between the precision and conditioning of  $\mathcal{A}$ .

We show prove the convergence guarantee of the SGD algorithm using phantom gradient in Geng et al. [87], where we show that under mild conditions (e.g.,  $\ell$  has bounded gradient almost surely, and the phantom gradient is an  $\epsilon$ -approximation) then a DEQ model trained with SGD converges also to an  $\epsilon$ -approximate stationary point in expectation.

## 5.3 Other Inexact Gradients

While this chapter will primarily focus on the two aforementioned of *phantom gradients* (which are based on truncation of an infinite series), we note that they are only instantiations of  $\mathcal{A}$  could be like. There are also other formulations, which we highlight in this section, but refer interested readers to Ramzi et al. [189] and Fung et al. [81] for more results.

### 5.3.1 SHaring the INverse Estimate (SHINE) from the Forward Pass

In a related work [189], we also leverage the fact that in DEQ models are often solved using quasi-Newton algorithms such as Broyden’s method [34] and Anderson acceleration [8, 72], which approximates the inverse of the Jacobian in the direction of steps. Specifically, in those solvers, we update the fixed-point estimation  $\mathbf{z}^{[i]}$  via the quasi-Newton update of the form

$$\mathbf{z}^{[i+1]} = \mathbf{z}^{[i]} - \alpha p^{[i]} = \mathbf{z}^{[i]} - \alpha B^{[i]} \underbrace{(f_\theta(\mathbf{z}^{[i]}; \mathbf{x}) - \mathbf{z}^{[i]})}_{=g_\theta(\mathbf{z}^{[i]}; \mathbf{x}) \text{ in Eq. (2.11)}} \quad (5.13)$$

where the quasi-Newton matrix (e.g., Broyden matrix)  $B^{[i]}$  is an approximation (by low-rank updates performed on top of  $B^{[i-1]}$  that satisfies the secant condition) to the Jacobian inverse  $(I - J_{f_\theta}(\mathbf{z}^{[i]})^{-1}$ . Therefore, one potentially good inexact gradient would then be to recycle  $B^{[i]}$  in the forward pass fixed-point solving as a guess for the backward pass  $(I - J_{f_\theta}(\mathbf{z}^*))^{-1}$  term. Specifically, given  $N$  forward quasi-Newton iterations which yield  $(\mathbf{z}^*, B^{[N]})$ , the SHINE method [189] proposes to produce inexact gradient

$$\mathcal{A}_N^{\text{SHINE}} = B^{[N]} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta} \quad (5.14)$$

This idea is justified with the theoretical guarantees, which we provide informally here and refer readers to Ramzi et al. [189] for details.

**Theorem 5. (Informal)** Under mild assumptions (e.g.,  $\sum_{i=0}^{\infty} \|\mathbf{z}^{[i]} - \mathbf{z}^{\infty}\| < \infty$ , invertible and Lipschitz-continuous  $J_{g_{\theta}}$  near  $\mathbf{z}^*$ , and continuous gradient),

$$\lim_{N \rightarrow \infty} \frac{\partial \ell}{\partial \mathbf{z}^*} \mathcal{A}_N^{SHINE} = \frac{\partial \ell}{\partial \theta} \Big|_{\mathbf{z}^*} \quad (5.15)$$

However, this method depends strongly on the fact that we use a white-box BFGS-alike (e.g., Broyden’s method) forward iterative solver, which may not always be available to us when training deep equilibrium models.

### 5.3.2 Jacobian-Free Estimation

Concurrently, Fung et al. [81] proposes a much simpler approach, where we simply replace the Jacobian inverse approximation with an identity matrix  $I \in \mathbb{R}^{d \times d}$ . The authors showed in [81] that this is equivalent to using a preconditioner on the gradient that is provably descent under certain strong assumptions (e.g., Lipschitzness of  $f_{\theta}$ ). Formally,

$$\mathcal{A}^{\text{JF}} = \frac{\partial f_{\theta}(\mathbf{z}^*; \mathbf{x})}{\partial \theta} \quad (5.16)$$

We note that while the Jacobian-free method is theoretically justified only under strong assumptions, they have been shown to work quite well in practice even when many of these assumptions are not guaranteed but the DEQ dynamical system is overall stable (we will see an example in Chapter 7). This is very important as it means the backward pass of these “infinite-level” neural networks that DEQ represents can be trained with a one-step gradient (i.e., one matrix-vector product), and is therefore almost free-of-charge.

## 5.4 Experiments

In this section, we demonstrate the effectiveness of phantom gradient on deep equilibrium models. We aim to answer the following questions via empirical results: (1) How precise is the phantom gradient? (2) What is the difference between the unrolling-based and the Neumann-series-based phantom gradients? (3) How is the phantom gradient influenced by the hyperparameters  $k$  and  $\lambda$ ? (4) How about the computational cost of the phantom gradient compared with implicit differentiation? (5) Can the phantom gradient work at large-scale settings for various tasks?

We start by introducing three experiment settings. First, in a synthetic setting, we adopt a single fully-connected layer  $f_{\theta}$  of the form  $\mathbf{z}^+ = \sigma(W(\mathbf{z} + \mathbf{x}))$  with spectral normalization [172] (so that the transformation is guaranteed contractive). This *synthetic setting* is *static* and the weights are randomly generated, thus acting as a sanity check to the quality of the inexact gradients. Second, on the CIFAR-10 dataset, we use the MDEQ-Tiny model (see Chapter 3, about 170K parameters) as the *ablation setting* where the training dynamics of DEQ models come into play. Finally, we conduct experiments on large-scale datasets to highlight the improved speed and performances on realistic datasets like ImageNet classification [198] and Wikitext-103 language modeling [164]. All training sources of this work are available at [https://github.com/Gsunshine/phantom\\_grad](https://github.com/Gsunshine/phantom_grad).

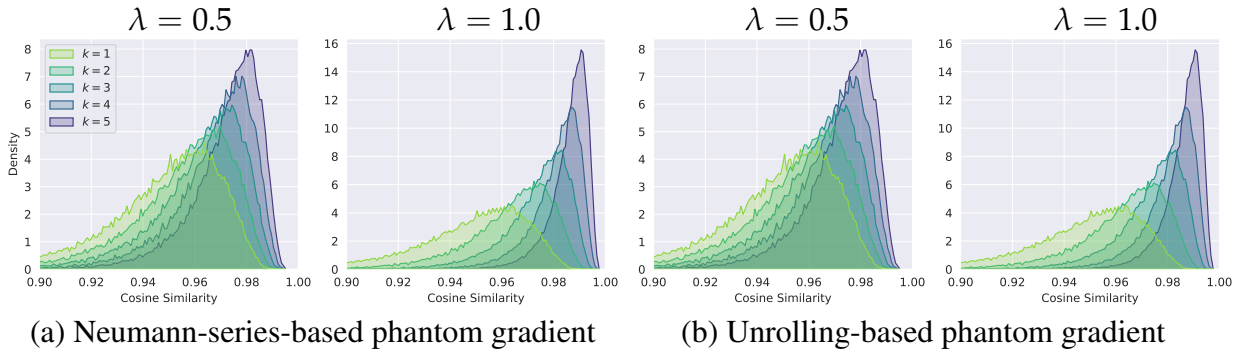


Figure 5.1: Cosine similarity between the phantom and exact gradient in the synthetic setting. As  $k$  increases for both UPG and NPG, the cosine similarity also rapidly approaches the true gradient.

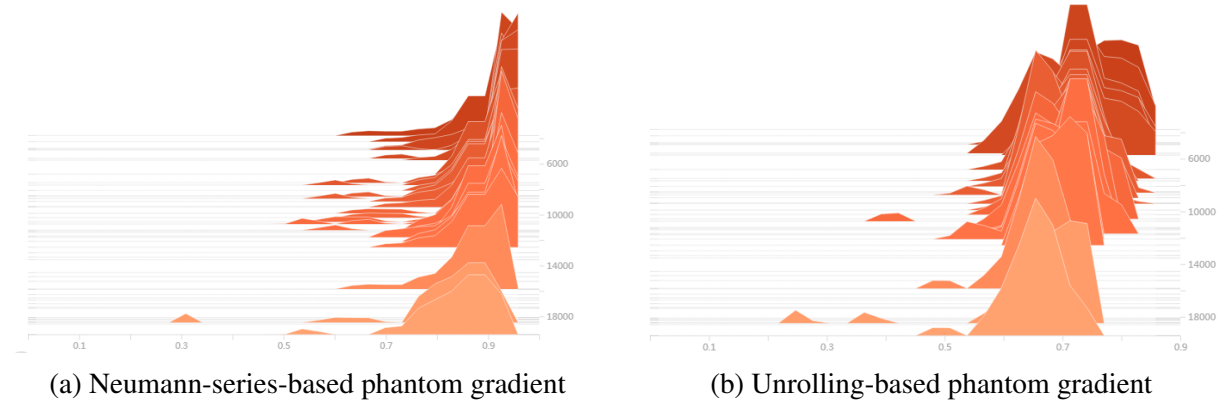


Figure 5.2: Cosine similarity between the phantom gradient and the exact gradient on CIFAR-10 during training. The horizontal axis corresponds to the cosine similarity, and the vertical axis to the training iterations (steps).

**Precision of the Phantom Gradient.** The precision of the phantom gradient is measured by its angle against the exact gradient, indicated by the cosine similarity between the two.

In the synthetic setting, the function  $f_\theta$  is restricted to be a contraction mapping. Specifically, we directly set the Lipschitz constant of  $f_\theta$  to be 0.9, and use up to 100 fixed-point iterations to solve the root  $\mathbf{z}^*$  of Eq. 5.2 until the relative error satisfies  $\frac{\|f_\theta(\mathbf{z}^*; x) - \mathbf{z}^*\|}{\|\mathbf{z}^*\|} < 10^{-5}$ . The inexact gradients produced by UPG and NPG are then compared to the “references” exact gradients, which is estimated by backpropagating through the fixed-point iterations, and cross-validated (with cosine similarity consistently succeeding 0.9999) by implicit differentiation solved with 20 steps of the Broyden’s method [34].

We report the cosine similarity between phantom gradients and the reference exact gradients in this synthetic setting in Fig. 5.1. The synthetic results show that the cosine similarity tends to increase as  $k$  grows for both UPG and NPG as expected, and that a small  $\lambda$  tends to slow down the convergence of the phantom gradient, allowing it to explore in a wider range regarding the angle against the exact gradient. Overall, the inexact and exact gradient directions are highly similar.

We additionally observe this in the ablation setting on CIFAR-10 with an MDEQ-Tiny model,



with the result shown in Fig. 5.2. The plot shows that the phantom gradient still provides good quality ascent direction even in the real training process. Interestingly, the cosine similarity slightly decays as the training progresses, which implies a possibility to construct an adaptive gradient solver for implicit models (e.g., a dynamic schedule on  $k$  and  $\lambda$ ).

**Trade-offs between Unrolling and Neumann.** As mentioned in the discussion of Sec. 5.2, when our fixed point estimation  $\hat{\mathbf{z}}^*$  is exactly the fixed point  $\mathbf{z}^*$ , there is no difference between UPG and NPG. They are both a truncated version of the Neumann series that uses the Jacobian matrix evaluated *exactly* at  $\mathbf{z}^*$ . However, when the numerical error is nontrivial, i.e.,  $\|\hat{\mathbf{z}}^* - f_\theta(\hat{\mathbf{z}}^*; \mathbf{x})\| > 0$ , these two instantiations can behave increasingly differently as this error gap broadens.

We note that a particular benefit of the UPG is its ability to automatically switch between an unrolling-based gradient stage (which can be thought of as a kind of layer-stacking pretraining, or warm-up, which DEQ-Transformer and MDEQ models also frequently use) and an implicit gradient stage (which directly uses the final precise fixed-point estimation). Specifically, when the model is not sufficiently trained or the solver converges poorly (see Bai et al. [19] and Chapter 4), the UPG defines a forward computation graph that is essentially equivalent to a shallow weight-tied network that unrolls to refine the equilibrium states. Therefore in this stage, the phantom gradient serves as a backpropagation-through-time (BPTT) algorithm through the shallow feedforward unrolling. Then, as training progresses, the solver could become more stable and converges to the fixed point  $\mathbf{z}^*$  better, which makes the UPG behave more like the NPG. We argue that such an ability to automatically and adaptively switch training stages is benign to the implicit models’ training protocol, which is also supported by the performance gain empirically.

Note that since in practice we only use a very small value for  $k$  (e.g., 3), this adds still minimal memory overhead to the training procedure, but which is higher than the memory overhead of a canonical implicit differentiation process presented in Chapter 2. We additionally demonstrate in Table 5.1 the time and memory complexity for canonical implicit differentiation (which solves a linear system corresponding to the Jacobian inverse  $(I - J_{f_\theta}(\mathbf{z}^*))^{-1}$ ) and the two forms of phantom gradient (which approximates this term). We note that the Jacobian-free method presented in Sec. 5.3.2 (originally proposed by Fung et al. [81]) corresponds to a special case where  $k = 1$ .

Table 5.1: Complexity comparison. Mem means the memory cost, and  $K$  and  $k$  denote the solver’s steps and the unrolling/Neumann steps, respectively. Here,  $K \gg k \approx 1$ .

Method	Time	Mem	Peak Mem
Implicit	$\mathcal{O}(K)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$
UPG	$\mathcal{O}(k)$	$\mathcal{O}(k)$	$\mathcal{O}(k)$
NPG	$\mathcal{O}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

**Phantom Gradient at Scale.** We conduct large-scale experiments to verify the advantages of the phantom gradient on vision, graph, and language benchmarks. We adopt the UPG in the large-scale experiments as we empirically find it perform better than NPG especially as we scale to the high-dimensional space (see Geng et al. [87] for more details). The results are illustrated in table 5.2 and table 5.3. Using the proposed inexact gradient, we are able to train equilibrium models that match or even surpass the implicit differentiation training protocol on

Table 5.2: Experiments using DEQ-Transformer and MDEQ on large-scale vision and language tasks. Metrics stand for accuracy( $\%$ ) $\uparrow$  for image classification on CIFAR-10 and ImageNet, and perplexity $\downarrow$  for language modeling on Wikitext-103. JR stands for Jacobian Regularization (see Chapter 4).  $\dagger$  indicates additional steps in the forward equilibrium solver.

Datasets	Model	Method	Params	Metrics	Speed
CIFAR-10	MDEQ	Implicit	10M	$93.8 \pm 0.17$	$1.0\times$
CIFAR-10	MDEQ	UPG $\mathcal{A}_{5,0.5}$	10M	$95.0 \pm 0.16$	$1.4\times$
ImageNet	MDEQ	Implicit	18M	75.3	$1.0\times$
ImageNet	MDEQ	UPG $\mathcal{A}_{5,0.6}$	18M	75.7	$1.7\times$
Wikitext-103	DEQ (PostLN)	Implicit	98M	24.0	$1.0\times$
Wikitext-103	DEQ (PostLN)	UPG $\mathcal{A}_{5,0.8}$	98M	25.7	$1.7\times$
Wikitext-103	DEQ (PreLN)	JR + Implicit	98M	24.5	$1.7\times$
Wikitext-103	DEQ (PreLN)	JR + UPG $\mathcal{A}_{5,0.8}$	98M	24.4	$2.2\times$
Wikitext-103	DEQ (PreLN)	JR + UPG $\mathcal{A}_{5,0.8}$	98M	$24.0^\dagger$	$1.7\times$

the state-of-the-art implicit deep networks with a substantial reduction on the training time. The method also complements prior methods that accelerate the equilibrium networks, such as the Jacobian regularizations presented in Chapter 4. And when we only consider the backward pass cost, the speed for multiscale DEQ models can be improved by a remarkable  $12\times$  factor on the large-scale ImageNet classification task.

Similarly, on graph classification and node classification tasks that prior implicit graph networks (which are provably convergent by construction) have done well on [92], we observe that UPG inexact gradients attain similar level of performance but with much cheaper backward pass cost. These results demonstrate that the equilibrium models can benefit significantly from these approximate gradients, and that finding a precise gradient estimation is not often necessary in these one-layer implicit deep architectures (in contrast to conventional deep networks).

## 5.5 Discussion

In this chapter, we explore the possibility of training implicit models via lightweight and inexact gradients that are still in provably ascent gradient directions. We first introduced the two instantiations of the phantom gradient, which were based on a finite-step truncation of an infinite-series view of the Jacobian inverse term. Then, we briefly discussed other potential ways to producing inexact gradients, including by recycling forward quasi-Newton matrices or by a completely Jacobian-free estimation.

Our synthetic and realistic experiments show that implicit deep networks can benefit significantly from the proposed phantom gradient approach. These inexact gradients complements other existing techniques like Jacobian regularization [20] and shows a  $1.7\times$  training acceleration with comparable or better performances than canonical DEQ models on large-scale benchmarks.

We note that non-end-to-end optimization of deep networks is not a completely new subject (e.g., by auxiliary variable [38, 224, 261, 262], or synthetic gradient [56, 112, 132]). However, these methods hardly scale to realistic levels due to their inherent complexities (e.g., synthetic

Table 5.3: Experiments using implicit graph neural networks [92] on graph tasks. Metrics stand for accuracy(%) $\uparrow$  for graph classification on COX2 and PROTEINS [253], Micro-F1(%) $\uparrow$  for node classification task on PPI.

Datasets	Model	Method	Params	Metrics (%)
COX2	IGNN	Implicit	38K	84.1 $\pm$ 2.9
COX2	IGNN	UPG $\mathcal{A}_{5,0.5}$	38K	83.9 $\pm$ 3.0
COX2	IGNN	UPG $\mathcal{A}_{5,0.8}$	38K	83.9 $\pm$ 2.7
COX2	IGNN	UPG $\mathcal{A}_{5,1.0}$	38K	83.0 $\pm$ 2.9
PROTEINS	IGNN	Implicit	34K	78.6 $\pm$ 4.1
PROTEINS	IGNN	UPG $\mathcal{A}_{5,0.5}$	34K	78.4 $\pm$ 4.2
PROTEINS	IGNN	UPG $\mathcal{A}_{5,0.8}$	34K	78.6 $\pm$ 4.2
PROTEINS	IGNN	UPG $\mathcal{A}_{5,1.0}$	34K	78.8 $\pm$ 4.2
PPI	IGNN	Implicit	4.7M	97.6
PPI	IGNN	UPG $\mathcal{A}_{5,0.5}$	4.7M	98.2
PPI	IGNN	UPG $\mathcal{A}_{5,0.8}$	4.7M	97.4
PPI	IGNN	UPG $\mathcal{A}_{5,1.0}$	4.7M	96.2

gradient method estimates the local gradient of neural networks using auxiliary models). In contrast, because a deep equilibrium model has exactly one layer, there is no need chain-rule backpropagation at all and we can directly work with its Jacobian at the final equilibrium point. This is a very important property and as we shall see in Chapter 7, inexact gradients can be of great help as we deploy DEQ models to real-time computer vision problems.



# Chapter 6

## Neural Deep Equilibrium Solvers

In Chapter 4, we introduce a way to make equilibrium models faster by improving their dynamics. That is, by encouraging the model to optimize for a simpler dynamical system, the fixed-point solving process will be faster as a result. In Chapter 5, we show that a different approach to improve training efficiency is to produce faster and approximate gradients.

But *can we make equilibrium models faster by taking advantage of their implicitness?* One benefit of deep equilibrium models’ formulation is the fact that they decouple the internal structure of the layer (which controls *representational capacity*) from how the fixed point is actually computed (which impacts *inference-time efficiency*), which is usually via classic techniques such as Broyden’s method [34] or Anderson acceleration [8]. This is not possible in any explicit model (e.g., ResNet-101 [96]), for which a static and prescribed computation graph is built and executed. Hence, as we showed in Chapter 2, given a trained DEQ, one can trade off inference time and the accuracy of the estimated fixed point by simply reducing the number of solver iterations. This yields a speed/accuracy trade-off curve; e.g., see Fig. 6.1. However, this trade-off (i.e., movements *along* the pareto curves) can be highly risky: as we gradually increase inference speed by compromising the quality of fixed point estimates, model accuracy also degrades drastically.

In this chapter, we show that we can shift the DEQ speed/accuracy trade-off curve by exploiting such decoupling to customize the fixed-point solving process itself. Prior work on equilibrium models relies on classic solvers, which are manually designed and generic (e.g., Broyden’s Method [34]). We propose a tiny, learnable, and *content-aware* solver module that is automatically *customized* to a specific DEQ model. Our *hypersolver* consists of two parts. First, we introduce a learned initializer that estimates a good starting point for the optimization. Second, we introduce a generalized parameterized version of Anderson acceleration [8] that learns the iterative updates as an input-dependent temporal process, and can be trained end-to-end in an *unsupervised* manner (i.e., the hypersolver is unaware of the underlying task). Such a solution is particularly suited

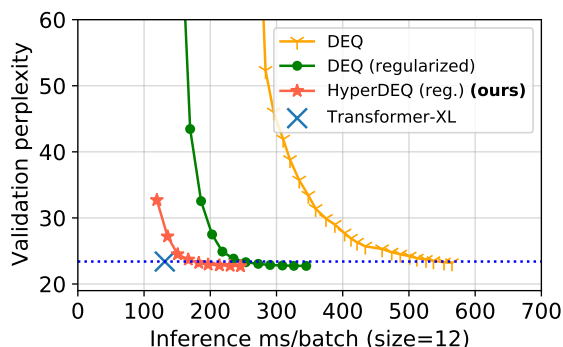


Figure 6.1: Pareto curves of the same DEQ with different solvers on WikiText-103 language modeling (on 1 GPU).

to the implicit model setting, because inference in these models requires *repeatedly* solving for a fixed point of the *same non-linear layer*  $f_\theta$  for different inputs, a task at which our custom neural solver excels. Moreover, the hypersolver consumes a tiny amount of parameters. Since  $f_\theta$  is frozen when the hypersolver is trained, the training is very fast and does not compromise generalization at all.

Our experiments apply this approach to diverse domains with large datasets, including the largest MDEQ applied to Cityscapes segmentation with megapixel images [53]. Our results suggest that neural deep equilibrium solvers add little overhead to training (only taking an extra 0.9-1.1% over the original DEQ’s training time), are extremely compact (about 1-3% of the DEQ’s model size), and lead to a consistent and universal 1.6-2 $\times$  acceleration of inference with no compromise in accuracy. The success of these neural fixed-point solving processes achieves two major objectives, both vital for the quickly growing community studying implicit models: first, we advance these large-scale implicit models to a much more practical level across architectures and input complexities (e.g., almost as fast as Transformers, see Fig. 6.1); and second, we demonstrate how one could exploit this valuable notion of how implicit layers decouple representational capacity and forward computation, a property that directly challenges what traditional deep networks have been about (where forward computation, i.e., layer-stacking, *decides* representational capacity).

This chapter is primarily based on the work that appeared in ICLR 2022 [22].

## 6.1 Preliminaries: Learning to Optimize, Fixed-point Solvers

**Learning to Optimize/Learn.** An important line of work has explored learnable optimization methods. Li and Malik [139, 140] propose to use reinforcement learning (guided policy search) to learn a new *generic* unconstrained continuous optimization algorithm, where the training set consists of numerous randomly generated objective functions. Andrychowicz et al. [9] introduce the “learning to learn” (L2L) framework, where a gradient update rule for the parameters is learned by an LSTM with a pre-defined horizon  $T$  of parameter update steps. However, such approaches [9, 48, 191, 242] have had some difficulty in generalizing to larger tasks due to the need to unroll for a large  $T$  (e.g., 128 [9]). This chapter is related to these prior efforts in L2L, but differs in important ways. First, the L2L framework aims to learn a learning algorithm that will be applied to multiple models and tasks, while we aim to fit the nonlinear dynamics of a specific implicit model. Second, the optimization we tackle is not on the parameter space, but on the hidden unit space; this means that the RNN optimizer used in L2L would not work here, because the fixed points themselves can be of variable sizes at test time (e.g., sequence lengths, image sizes). Third, while L2L methods cannot know a priori what a good “initial guess” of optimal parameters may be, we show that it is reasonable to infer this in the hidden unit space with implicit models. Concurrent to the work presented in this chapter, Venkataraman and Amos [234] studies an RNN-based learnable fixed-point acceleration scheme specifically in the application of convex cone programming. Poli et al. [187] introduce a Neural ODE formulation that adds a learnable residual fitting step to the original solver steps, aiming to approximate the higher-order terms of canonical ODE solvers (e.g., Euler’s method) on each solution checkpoint along the ODE path. Another recent work by Kidger et al. [122] focuses on improving the adjoint method by replacing

---

**Algorithm 2** Anderson acceleration (AA) prototype (with parameter  $\beta$  and  $m$ )

---

- 1: **Input:** initial point  $z^{[0]} \in \mathbb{R}^n$ , fixed-point function  $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , max storage size  $m$
  - 2: **for**  $k = 0, \dots, K$  **do**
  - 3:     1) Set  $m_k = \min\{m, k\}$
  - 4:     2) Compute weights  $\alpha_i^k$  for the **past**  $m_k$  **Anderson steps** s.t.  $\sum_{i=0}^{m_k} \alpha_i^k = 1$ .
  - 5:     3)  $z^{[k+1]} = \beta \sum_{i=0}^{m_k} \alpha_i^k f_\theta(z^{[k-m_k+i]}) + (1 - \beta) \sum_{i=0}^{m_k} \alpha_i^k z^{[k-m_k+i]}$  (AA\_update step)
  - 6: **end for**
  - 7: **Return**  $z^{[K+1]}$  as an estimate for  $z^*$
- 

the usual L2 norm with a more flexible seminorm to make the NODE backward solver faster.

**Fixed-point Solvers for Deep Equilibrium Models.** Previous chapters and prior works have explored a number of techniques for finding the fixed points of DEQs. For example, Bai et al. [18, 19], Lu et al. [157] used Broyden’s method [34], the memory consumption of which grows linearly with the number of iterations since all low-rank updates are stored. Other recent work [68, 88] shifted to Anderson acceleration (AA) [8], a lightweight solver that is provably equivalent to a multi-secant quasi-Newton method [72]. Some specific structural parameterizations (e.g., Lipschitzness or contraction mapping) allows other works to rely on algorithms like Peaceman-Rachford [192, 240, 244]. We briefly introduce Anderson acceleration (AA) here, since our approach will use it as the starting point.

Prototype algorithm 2 illustrates the main idea of Anderson acceleration: we maintain a size- $m$  storage of the most recent steps, and update the iteration as a normalized linear combination of these past iterates with weights  $\alpha_i$  (step 3). In the canonical AA algorithm, the weights are computed in a *greedy* manner at each step to minimize the sum of their linear combination:

$$\alpha^k = \arg \min_{\alpha \in \mathbb{R}^{m_k+1}} \|G^{[k]}\alpha\|_2, \text{ s.t. } \mathbf{1}^\top \alpha = 1, \quad (6.1)$$

where  $G^{[k]} = [g_\theta(\mathbf{z}^{[k-m_k]}) \dots g_\theta(\mathbf{z}^{[k]})]$  are the past (up to  $m + 1$ ) residuals; typically, we choose  $\beta = 1$  and  $m \leq 5$ . Eq. (6.1) can be solved by a least-squares method as  $G^{[k]\top} G^{[k]}$  is only an  $m \times m$  matrix. In all prior works with DEQs [18, 19, 81, 157, 192, 244], the fixed point iteration starts with an initial  $\mathbf{z}^{[0]}$  that is either  $\mathbf{0}$  or a random sample from  $\mathcal{N}(0, I)$ .

## 6.2 Neural Deep Equilibrium Solvers

While classic fixed-point estimation algorithms, as presented in Section 6.1, already work well, they are generic and make minimal assumptions about the specific problem being solved. For example, while multiple papers in optimization literature have acknowledged that tuning  $m$  (and  $m_k$ ) as well as varying  $\beta = (\beta_k)_{k=0, \dots, K}$  for each Anderson iteration  $k$  could accelerate AA’s convergence to the fixed point [8, 72, 235], this is rarely considered in practice because it’s unclear what schedule should be applied to these parameters.

We propose to make fixed-point solvers for DEQ models learnable and content-based, which is made possible by the unique properties of implicit models. First, unlike generic problems,

the nonlinear system for each DEQ is uniquely defined by the input  $\mathbf{x}$  (e.g., an image, etc.):  $\mathbf{z}^*(\mathbf{x}) = \mathbf{z}^* = f_\theta(\mathbf{z}^*; \mathbf{x})$ . This opens the door to learning to make an informed initial guess, followed by content-based iterative updates in the solver. Second, due to implicit models’ disentanglement of representation capacity with forward computation, our goal of improving solvers is decoupled from the original learning goal of the DEQ model itself (i.e., the solver is *not* aware of the original task, such as to predict the class of an image). Hence, we are able to train this neural solver in a *lightweight* and *unsupervised* manner, directly with the help of groundtruth fixed-point solutions (see below).

### 6.2.1 General Formulation

For a given DEQ layer  $f_\theta$  and (possibly random) input  $\mathbf{x}$ , we assume access to its exact fixed point  $\mathbf{z}^* = \mathbf{z}^*(\mathbf{x}) = f_\theta(\mathbf{z}^*, \mathbf{x})$ , which can be obtained by taking a classic solver (e.g., Broyden’s method) and running it for as many iterations as needed (e.g., 100 steps) to a high level of precision.

The overall structure of the hypersolver is shown in Fig. 6.2. We use a tiny neural network parameterized by  $\omega = \{\phi, \xi\}$  (explained below) to learn the initialization and iterative solving process, and unroll the learnable solver for some  $K$  steps to yield a prediction  $\mathbf{z}^{[K]}(\mathbf{x})$ . To train this neural solver, we minimize an objective  $\mathcal{L}(\omega, K)$  (discussed in Sec. 6.2.2) by backpropagating through this  $K$ -step *temporal* process [174, 194]. The original DEQ parameters  $\theta$  are frozen, and only the hypersolver parameters  $\omega$  are trained here. We also do not need the groundtruth label  $y$  (e.g., the class of an image) that corresponds to input  $\mathbf{x}$ , which means these neural equilibrium solvers can also be fine-tuned on the fly after deployment, at inference time.

**Initializer.** The initial values can have a significant impact on the optimization process and its convergence speed. We propose to make an input-based guess with a tiny network  $h_\phi$ :  $\mathbf{z}^{[0]} = h_\phi(\mathbf{x})$ , where  $\phi$  are the parameters. Note that the goal of the initializer is not to solve the underlying problem at all (e.g., to classify an image; we don’t even need the groundtruth label  $y$ ), but only to yield a quick initial estimate. For example, in language modeling, where  $\mathbf{x} \in \mathbb{R}^{T \times d}$  is a length- $T$  sequence, we set

$$h_\phi(\mathbf{x}) = \text{ReLU}(\text{Conv1d}_{k=3}(\mathbf{x}))W, \text{ where } \text{Conv1d}_{k=3} : \mathbb{R}^{T \times d} \rightarrow \mathbb{R}^{T \times p} \quad (6.2)$$

and where  $W \in \mathbb{R}^{p \times q}$ , with  $q$  being the dimension of the fixed point of a single token. We set  $p \ll d$  to be very small (e.g.,  $p = 100$  and  $d = 700$  in the WikiText-103 experiment), so that  $h_\phi$  is tiny and fast to evaluate. Note that this 1-layer initializer by itself has very low expressivity and is usually a poor model for the original task, as we verify in Sec. 6.3.3.

**HyperAnderson Iterations.** We further parameterize the setting of  $\beta_k$  and  $\alpha_i^k$  while following the AA prototype outlined in Alg. 2. In lieu of setting Eq. 6.1 for  $\alpha$  to a least-squares solution over the past few residuals  $G$ , we make both  $\alpha \in \mathbb{R}^{(m_k+1)}$  and  $\beta \in \mathbb{R}$  explicit learnable functions of  $G$  with a neural network  $s_{\xi}(G) : \mathbb{R}^{(m_k+1) \times n} \rightarrow (\mathbb{R}^{(m_k+1)} \times \mathbb{R})$ ; see Alg. 3.

A challenge here is that  $n$  (the dimension of  $\mathbf{z}^*$ ) is typically large in practice, as it is affected by the scale of the input (e.g., in DEQ sequence models [18],  $n$  is over  $1.5 \cdot 10^5$  on a *single* textual



---

**Algorithm 3** HyperAnderson Iterations (parameterized parts highlighted in color)

---

- 1: **Input:** initial point  $\mathbf{z}^{[0]} = h_\phi(\mathbf{x}) \in \mathbb{R}^n$ , (frozen) layer  $f_\theta$ , storage  $G = \mathbf{0} \in \mathbb{R}^{(m+1) \times n}$  with size  $m + 1$ , HyperAnderson network  $s_\xi$ .
  - 2: Define  $g_\theta(\mathbf{z}) = f_\theta(\mathbf{z}) - \mathbf{z}$ . Set  $G[0] = g_\theta(\mathbf{z}^{[0]})$ .
  - 3: **for**  $k = 0, \dots, K$  **do**
  - 4:   Set  $m_k = \min\{m, k\}$  and  $G^{[k]} = G[0:(m_k + 1)] \in \mathbb{R}^{(m_k+1) \times n}$
  - 5:   Compute  $\hat{\alpha}^k, \beta_k = s_\xi(G^{[k]})$ , where  $\hat{\alpha}^k = (\hat{\alpha}_0^k, \dots, \hat{\alpha}_{m_k}^k) \in \mathbb{R}^{(m_k+1)}$
  - 6:    $\alpha^k = \hat{\alpha}^k + \frac{(1-\mathbf{1}^\top \hat{\alpha}^k)}{m_k+1} \cdot \mathbf{1}$  (normalization step)
  - 7:    $\mathbf{z}^{[k+1]} = \beta_k \cdot \mathbf{1}^\top G^{[k]} + \sum_{i=0}^{m_k} \alpha_i^k \mathbf{z}^{[k-m_k+i]}$  (same AA\_update as in Alg. 2, simplified)
  - 8:   Update  $G = \text{concat}(G[1:], [g_\theta(\mathbf{z}^{[k+1]})])$
  - 9: **end for**
  - 10: **Return**  $\mathbf{z}^{[K+1]}$
- 

sequence of length 200). This makes  $s_\xi$  map from an extremely high-dimensional space to a low-dimensional space (e.g.,  $m = 5$ ). To keep  $s_\xi$  fast, small, and applicable to inputs of varying dimensionalities (e.g., sequence length or image size), we propose to first compress each  $g_\theta(\mathbf{z}^{[k]})$  to form a smaller yet still representative version  $\hat{G}^{[k]}$  of  $G^{[k]} = [g_\theta(\mathbf{z}^{[k-m_k]}), \dots, g_\theta(\mathbf{z}^{[k]})]$ . For example, when each  $g_\theta(\mathbf{z}^{[k]})$  is a image feature map residual of dimension  $n = C \times H \times W$ , we can perform global pooling to form a  $C$ -dimensional vector  $\text{Pool}(g_\theta(\mathbf{z}^{[k]}))$  as its compressed version:

$$\hat{G}^{[k]} = [\text{Pool}(g_\theta(\mathbf{z}^{[k-m_k]})), \dots, \text{Pool}(g_\theta(\mathbf{z}^{[k]}))] \in \mathbb{R}^{C \times (m_k+1)}, \quad \text{and predict } \alpha^k, \beta_k = s_\xi(\hat{G}^{[k]}) \quad (6.3)$$

Once we have this representative collection  $\hat{G}^{[k]}$ , we treat it as a mini time-series of length  $(m_k + 1)$  that encodes the latest estimates of the fixed point. We then apply a 2-layer temporal convolution [231] to learn to predict: 1) a relative weight  $\alpha_i^k$  for each of these past residuals  $i \in [m_k]$ ; and 2) the HyperAnderson mixing coefficient  $\beta_k$  for the current iteration. Therefore,  $s_\xi$  shall gradually learn to adjust these parameters  $\alpha$  and  $\beta$  in light of the previous hypersolver steps, and receive gradients from later iterations. We explain the detailed design choices of  $s_\xi$  in Sec. 6.3, while noting that it still *completely captures* the AA prototype (see Alg. 2).

We show in Fig. 6.3 an example of the modules in a DEQ-Transformer’s neural solver. The initializer, as discussed previously, consist of a tiny 1D convolutional module. During the HyperAnderson iterations, we compress the past and current residuals  $G^{[k]} = [g_\theta(\mathbf{z}^{[k-m]}), \dots, g_\theta(\mathbf{z}^{[k+1]})] \in \mathbb{R}^{(m+1) \times T \times q}$  by simply taking the last and most contextualized token of each sequence; i.e.,  $(g_\theta(\mathbf{z}^{[i]}))_T \in \mathbb{R}^q$ . This is reasonable because the design of Transformer block that  $f_\theta$  uses already ensures that the last token contains most of the contextual information of the entire sequence (and this technique is frequently used for textual classification tasks, or in vision transformers [65]).

## 6.2.2 Training the Neural Equilibrium Solvers

One benefit of training hypersolvers on implicit models is that they can be trained in an *unsupervised* manner via  $\mathbf{z}^*(\mathbf{x})$ , which a slower classic method can provide as many as needed, and for

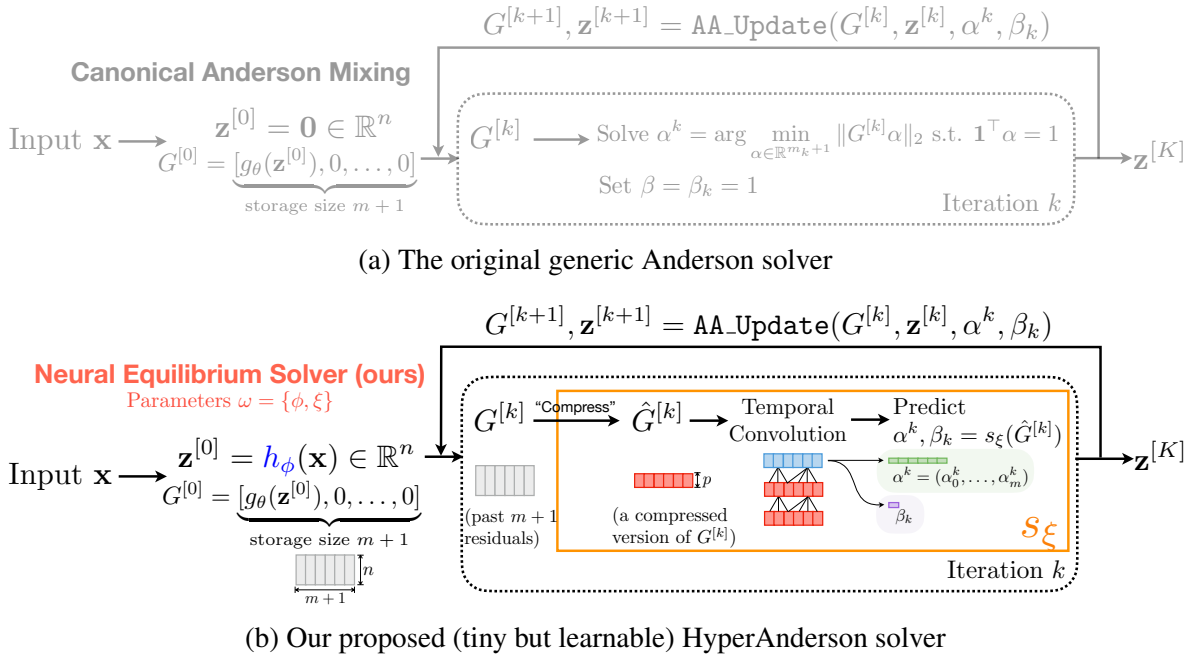


Figure 6.2: **6.2a**: The canonical Anderson solver is based on a local least-squares solution at each iteration, with  $\beta = \beta_k$  set to a constant. **6.2b**: Our neural fixed-point solver provides a *better initial guess*  $\mathbf{z}^{[0]}$  and *learnable iterative updates*.

any given (possibly even random) input tensor  $\mathbf{x}$ . Moreover, unlike NODE solvers [45, 187], a DEQ model does not have a unique trajectory and thus its hypersolvers do not need trajectory fitting at all. All that we need is to drive everything to be as close to  $\mathbf{z}^*$  as possible. As an example, a neural solver could learn to sacrifice progress in earlier iterations if it subsequently converges to the equilibrium faster. Formally, given a hypersolver  $\{h_\phi, s_\xi\}$  that yields a set of states  $(\mathbf{z}^{[k]}, G^{[k]}, \alpha^k, \beta_k)_{k=0, \dots, K}$  (recall  $\mathbf{z}^{[0]} = h_\phi(\mathbf{x})$ ), we introduce 3 objectives for its training.

**Fixed-point Convergence Loss.** The first loss aims to encourage convergence at all intermediate estimates  $\{\mathbf{z}^{[k]}\}_{k=1, \dots, K}$  of the HyperAnderson iterations:  $\mathcal{L}_{\text{conv}} = \sum_{k=1}^K w_k \|\mathbf{z}^{[k]} - \mathbf{z}^*\|_2$ , where  $w_k$  is the weight for the loss from iteration  $k$  such that  $\sum_{k=1}^K w_k = 1$ . We set  $w_k$  to be monotonically increasing with  $k$  such that later iterations are applied a heavier penalty for deviations from the fixed point (see Appendix A of Bai et al. [22] for more details).

**Initializer Loss.** We also train the initializer by maximizing the proximity of the initial guess to the fixed point:  $\mathcal{L}_{\text{init}} = \|h_\phi(\mathbf{x}) - \mathbf{z}^*\|_2$ . We separate this objective from  $\mathcal{L}_{\text{conv}}$  since the initialization is predicted directly from the input  $\mathbf{x}$  and does not go through HyperAnderson updates.

**Alpha Loss.** Although we replace the generic Anderson solver [8] in terms of how  $\alpha^k, \beta_k$  are computed in each iteration, we empirically found it still beneficial to guide the hypersolvers' prediction of  $\alpha$  with an auxiliary loss especially at the start of the training:  $\mathcal{L}_\alpha = \sum_{k=0}^K \|G^{[k]} \alpha^k\|_2$ . In practice, we gradually decay the weight of this loss to 0 as training progresses. We summarize the complete training procedure of a neural solver on top of a DEQ in Fig. 6.4.

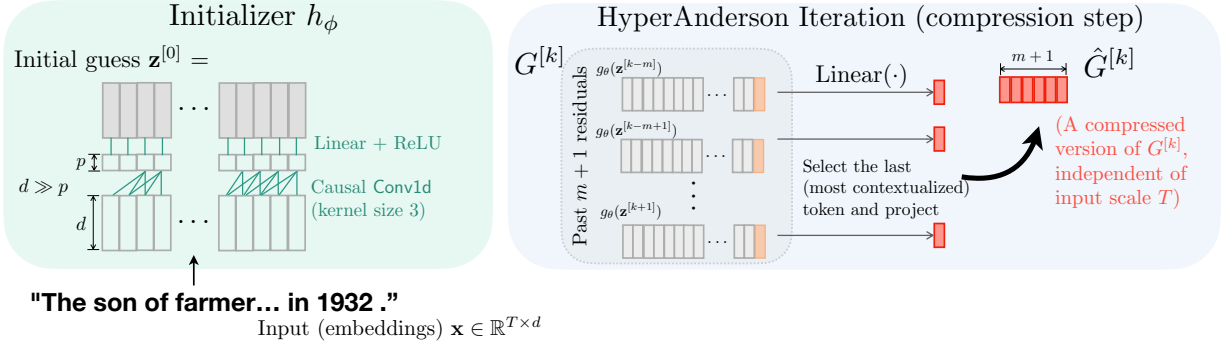


Figure 6.3: Visualization of DEQ-Transformer (for WikiText-103 language modeling) initializer and HyperAnderson iterations.

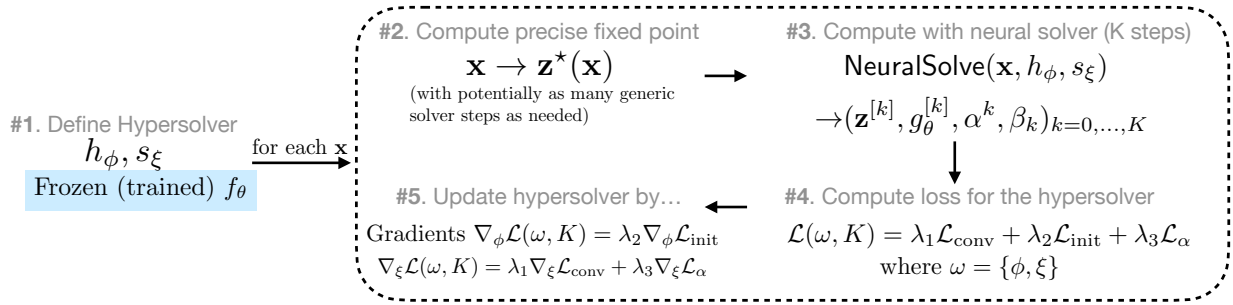


Figure 6.4: The training procedure of the neural deep equilibrium solver. With a given  $f_\theta$  and input  $\mathbf{x}$ , we optimize the hypersolver parameters  $\omega = \{\phi, \xi\}$  via losses applied on the HyperAnderson iterations and the initializer (see Sec. 6.2.2).

## 6.2.3 Discussion

**Complexity of hypersolver.** Note that  $f_\theta$  remains frozen during hypersolver training. This means that for a given DEQ model  $f_\theta$  and input  $\mathbf{x}$ , the fixed point  $\mathbf{z}^*(\mathbf{x}) = f_\theta(\mathbf{z}^*; \mathbf{x})$  also remains the same – we are just trying to learn to find it faster, with a limited  $K$ -iteration budget. Moreover, we designed the initializer  $h_\phi$  and HyperAnderson network  $s_\xi$  to be intentionally simple (e.g, 1 layer with few hidden units), so that *each hypersolver step is even faster than the original Anderson step*, whose main computational overhead occurs in solving the constrained optimization in Eq. 6.1.

These points also highlight the difference between the neural solver and techniques such as model compression [95] or distillation [98], where a pruned/smaller (but still representationally rich) model is trained to match the output and performance of a larger model. Specifically, in our case, as the fixed point  $\mathbf{z}^*$  is determined solely by  $f_\theta$  and  $\mathbf{x}$ , the hypersolver itself does not have much representational capacity, since its only goal is to produce an “educated” initial guess and learnable iterations to facilitate the optimization process. E.g., the 1-layer Conv1d-based initializer Sec. 6.2.1 would be a bad language model by itself since it is tiny and only sees the past 2 tokens (see Sec. 6.3.3 for empirical evidence), yet this limited capacity and context turn out sufficient to guide and substantially improve the solver.

**Training hypersolver via BPTT.** While a generic Anderson solver computes  $\alpha^k$  by optimizing locally with  $G^{[k]}$ , backpropagating through the HyperAnderson steps ensures that the iterative update network  $s_{\xi}$  can receive gradient and learn from later iterations. This is appealing because, arguably, only the output of the  $K^{\text{th}}$  iteration matters in the end. Indeed, we empirically verify via ablation studies in Sec. 6.3 that such learned  $\alpha$  and  $\beta$  predictors already significantly accelerate the convergence process even without the presence of the initializer. Note that as DEQ models’  $f_{\theta}$  layer is typically richly parameterized, the backpropagation-through-time (BPTT) [241] might consume a lot of memory. To limit memory consumption, we use small batch sizes for hypersolver training. (This does not affect the training of the DEQ model itself, which is separate.) We have observed that hypersolver training is highly effective with small batch sizes, as reported in Sec. 6.3. As an alternative solution, since these hypersolvers are very fast to train in practice, one could also use methods such as gradient checkpointing [46].

**Complementarity with DEQ regularizations.** Besides tiny size and fast training, the value and usefulness of neural equilibrium solvers are highlighted by how DEQ models decouple representational capacity and forward solver choice. In particular, our method is orthogonal to prior work that accelerates DEQ models by structural regularization of  $f_{\theta}$  [20, 192, 244] (e.g., Chapter 4) or approximating the Jacobian of  $f_{\theta}$  in the backward pass [81] (e.g., Chapter 5). In Sec. 6.3, we show evidence that our method (which is solver-based) integrates well with regularization approaches (which are  $f_{\theta}$ -based) and yields broad improvements compared to canonical solvers (e.g., Broyden or Anderson methods) regardless of how  $f_{\theta}$  was trained or what structure it uses.

## 6.3 Experiments

As our goal is to show the superiority of the learnable solvers over generic solvers on both performance and efficiency aspects, we compare the movement of the *entire* speed/accuracy pareto curve rather than a single point on the curve. To achieve this purpose, we study the hypersolver on some of the largest-scale experiments that DEQs have been used on in the previous few chapters (which involve models of size almost 100M): WikiText-103 language modeling [164], ImageNet classification [62], and Cityscapes semantic segmentation with megapixel images [53]. Overall, we show that: 1) these *custom* neural solvers bring universal improvement over generic solvers on DEQ models in all scenarios, with a typically  $1.6\text{-}2\times$  speedup at inference and no loss in performance (i.e., the new pareto curves *strictly dominates* old ones); 2) these hypersolvers can be trained very quickly; and 3) these methods complement prior methods such as regularizations on  $f_{\theta}$  to bring these implicit models to a new competitive level. At the end of this section, we also conduct extensive ablative studies on the design of the hypersolver.

Note that since the neural solver training is independent of the DEQ training, we do not need to train the actual DEQ model  $f_{\theta}$  itself, but could instead directly work on top of a pre-trained DEQ model. Therefore, the major hyperparameters in our setting are only the relative weights of the loss objectives (see Sec. 6.2.2 and Appendix A in Bai et al. [22]). We also clarify that the use of hypersolver does implicitly assume local stability around  $\mathbf{z}^*$  for convergence – which we find almost always holds empirically, and can be regularized for [20] (see Chapter 4).

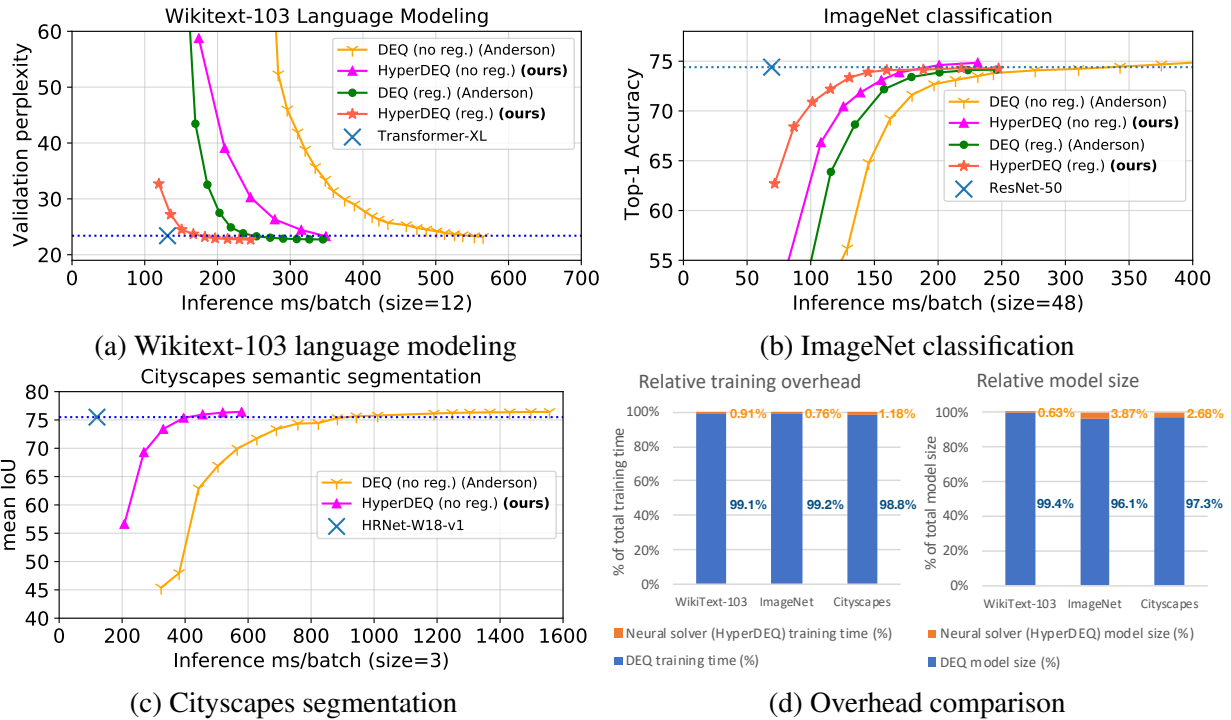


Figure 6.5: 6.5a- 6.5c: Comparisons of DEQs with classic and neural solvers. All speed/accuracy curves within the same plot are benchmarked on the same GPU with the same experimental setting, averaged over 6 independent runs. 6.5d: The overhead of DEQ hypersolver is extremely small, generally requiring only  $< 1.2\%$  the (unsupervised) training time and  $< 4\%$  the size when compared to the original DEQ models on these large-scale tasks.

### 6.3.1 Large-scale Experiments on Vision and Language Tasks

To evaluate the the neural deep equilibrium solvers, we apply them on three largest-scale and highest-dimensional tasks the implicit models have ever been applied on, across the vision and language modalities. In contrast to prior work [20, 45, 244] and chapters that measure the number of function evaluations (NFEs), we directly measure wall-clock inference speed under the exact same experimental settings (e.g., input scale). We elaborate on the detailed experimental settings and the implications of the results below.

**WikiText-103 Language Modeling.** In this experiment,  $f_\theta$  is a Transformer layer [18, 58, 233] and the fixed points  $\mathbf{z}^*$  are (embeddings of) text sequences. We train the neural solver on sequences of length 60 for 5000 steps, and demonstrate its inference-time effect in Figure 6.5a (where we use a validation sequence length of 150). Specifically, compared with the original DEQ-Transformer [18] (Y curve), which uses generic Anderson acceleration [8] or Broyden’s method [34] (both have similar pareto curves; see Fig. 6.8), this same DEQ model solved with our neural approach (dubbed HyperDEQ; see  $\blacktriangle$  curve) achieves significantly better efficiency. Moreover, our method is complementary to prior work that builds faster implicit models by Jacobian regularizations [20, 77]. To demonstrate this, we additionally train a DEQ-Transformer

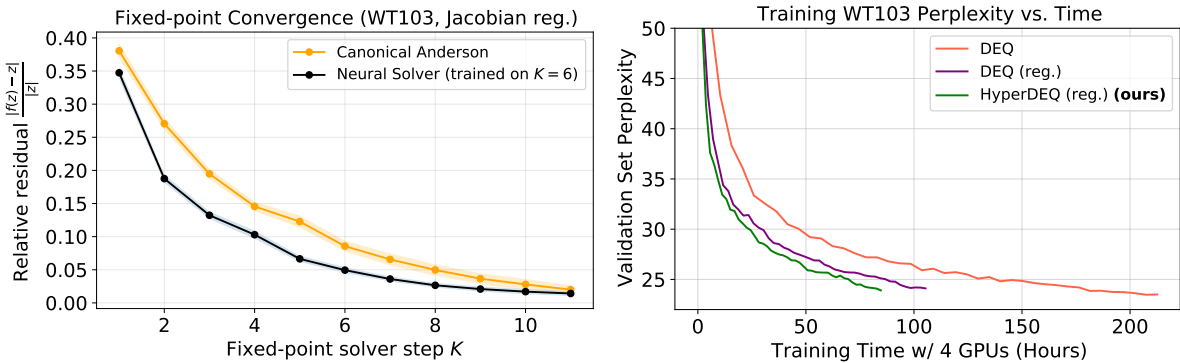
model with Jacobian regularization [20] (● curve), and apply the neural solver on this regularized DEQ (★ curve). This movement of the speed/perplexity curves validates the DEQ property at the core of this paper: *the decoupling of the representational capacity (i.e.,  $f_\theta$ ) and the forward computation (i.e., the solver)*. With everything combined, we bring the performance of implicit Transformer-based DEQs close to the explicit Transformer-XL [58], which is the SOTA architecture on this task.

**ImageNet classification.** We additionally evaluate HyperDEQ on ImageNet classification ( $224 \times 224$  images over 1000 classes), customizing a neural solver on top of a 4-resolutional multiscale DEQ models [19]. We train the HyperDEQ with 12 HyperAnderson iterations, and the speed/accuracy curves are shown in Figure 6.5b (▲ and ★ curves). Note that while Jacobian regularization (● curve) eventually hurts the performance of a multiscale DEQ (cf. Y curve) due to the strong constraint it imposes, the DEQ model with neural solver achieves faster inference without sacrificing any accuracy (since  $f_\theta$ , and thus  $\mathbf{z}^*$ , are identical); e.g., we reach 75.0% accuracy while being almost  $2\times$  faster.

**Cityscapes semantic segmentation.** We also show that our neural solver approach works well in domains where existing regularization-based methods fail (see Sec. 4.2.6 for the “physical law” issue on large images). Specifically, we apply the neural equilibrium solver on Cityscapes semantic segmentation, where the task objective is to label every pixel on a high-resolution (typically  $2048 \times 1024$ ) image with the class of the object that the pixel belongs to. As in the ImageNet and WikiText-103 tasks, we found that there is a consistent gain in using the neural solver over the generic alternative, accelerating fixed-point convergence by more than a factor of 2 (see Figure 6.5c). In contrast, prior methods such as Jacobian regularization [20] presented in Chapter 4 do not work in this setting, due to their dependence on the exact structure of  $f_\theta$ . (Specifically, when  $f_\theta$  is convolution-based and the image is very large, Jacobian regularization that encourages contractivity is *at odds with* the gradual broadening of the receptive field.) Our neural solver is orthogonal to the structure of  $f_\theta$  (which is frozen), and we only improve how the solver functions.

### 6.3.2 Training Efficiency of the Neural Solver

We also provide extra training analysis in Fig. 6.5d. Not only is our approach effective, but the overhead for training the neural solver is also extremely small: the neural solver module is tiny ( $< 4\%$  of the DEQ model size) and requires only about 1% of the training time needed by the original DEQ model (e.g., on WikiText-103, a DEQ requires 130 hours on 4 GPUs; the neural solver requires only about 1.2 extra hours). We believe this is strong evidence that neural solvers are simple, lightweight, and effective tools that take advantage of the decoupling properties of equilibrium models to yield an almost-free acceleration at inference time. We additionally provide empirical evidence on the convergence and generalizability of the neural solver in Fig. 6.6b, where canonical AA is compared with a hypersolver that was *trained* to unroll for  $K = 6$  HyperAnderson steps. As the figure shows, a hypersolver trained with  $K$  steps is able to generalize well beyond, and consistently improves over the canonical solver’s convergence while being more lightweight per step.



(a) The neural equilibrium solver improves upon the convergence path of canonical AA. The speedup of HyperDEQ is a result of this faster convergence and the more lightweight solver step.

(b) Hypersolver can also be used at training time to further accelerate the DEQ model training by alternating their training (in addition to existing methods like Jacobian stabilization).

Figure 6.6: Left: Convergence analysis of the hypersolver at inference time. Right: Although trained with a frozen  $f_\theta$ , the neural equilibrium solver can also be used to accelerate DEQ training.

Interestingly, one can also employ the neural solver to accelerate the DEQ training, but with three caveats: 1) during training the fixed point manifold also keeps changing; 2) we want to amortize the cost of computing “groundtruth”  $z^*$ ; and 3) we still keep the backward implicit differentiation intact. Thus, we propose to train the neural solver  $\{h_\phi, s_\xi\}$  and the DEQ model  $f_\theta$  in an alternating manner (i.e., we take a snapshot of the  $f_\theta$  layer occasionally, and a hypersolver customized toward it). We The results are shown in Fig. 6.6b We empirically observe this leads to a 16-20% DEQ training speedup.

### 6.3.3 Ablative Studies and Limitations

Finally, we perform a series of ablation studies to understand the benefits of multiple components within our design of the neural equilibrium solvers. We use the language modeling task on WikiText-103 for this purpose (where  $f_\theta$  is a Transformer layer), while noting that we’ve noticed similar trends in all other settings. The results are presented in Fig. 6.8. The HyperDEQ with everything combined (initializer,  $\alpha^k$ , and  $\beta_k$  predictions) performs best. Making the Anderson iterations learnable generally improves convergence. Moreover, although simply adding an initializer to a generic solver (+ curve) does not help much, learning and back-propagating through the HyperAnderson iterations makes the initializer quite useful (cf. ■ and ★ curves). We additionally take the learned initializer  $h_\phi$  from HyperDEQ and verify that

Table 6.1: Perplexity (ppl) on WikiText-103

	Model Size	Test ppl
Gated ConvNet [59]	230M	37.2
Transformer-XL [67]	165M	24.2
HyperDEQ (reg.) w/ 12 iters (ours)	98M	<b>23.4</b>
Initializer $h_\phi$ (Conv1d)	0.4M	836.94

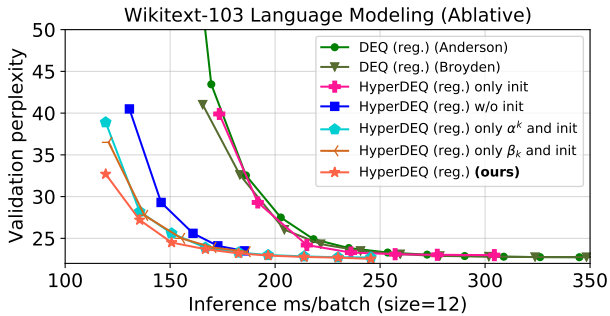
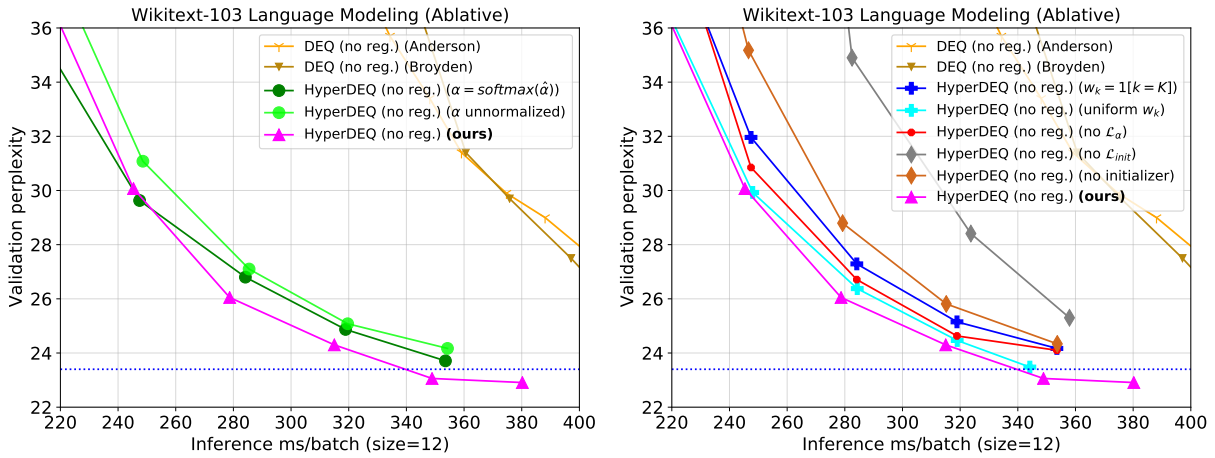


Figure 6.8: Ablations on HyperDEQ (reg.).



(a) Ablations on the importance of  $\alpha$  normalization and learning.  $\blacktriangle$  curve is our proposed method. (b) Ablations on loss components and relative loss weights.  $\blacktriangle$  curve is our proposed method.

Figure 6.7: Further ablations on the neural solver design (in terms of the  $\alpha$  prediction and loss components). Note that we use an unregularized DEQ here (in contrast to Fig. 6.8) to better demonstrate the curve differences, which could otherwise sometimes be small.)

this tiny module is by itself *still a poor language model* (see Table 6.1 and Sec. 6.2.3), but is valuable to our HyperAnderson iterations.

We also perform more ablations on the importance of mixing factor  $\alpha$ 's prediction. In our hypersolver, we propose to “normalize” the predicted  $\hat{\alpha}^k$  by a shift:  $\alpha^k = \hat{\alpha}^k + \frac{(1-\mathbf{1}^\top \hat{\alpha}^k)}{m_k+1} \cdot \mathbf{1}$ . We here compare two alternatives: no normalization at all ( $\bullet$  in Fig. 6.7a), and softmax-based normalization ( $\bullet$  in Fig. 6.7a). Specifically, we find that the exact choice of  $\alpha$  normalization does not affect the overall substantial pareto curve improvement, but ensuring that the  $\alpha^k$  values can be negative while still normalized (i.e., sum to 1) overall benefit the performance.

In the end, we analyze the effect of different loss components in Fig. 6.7b. For the main fixed-point convergence loss  $\mathcal{L}_{\text{conv}}$ , we compare two alternatives of the relative weights applied on the intermediate HyperAnderson steps: only apply  $w_k = 1$  at the final output  $K$  (see  $\blackplus$  curve), or set  $w_k$  to be uniform for all  $K$  iterates (see  $\blackplus$  curve). We empirically find that the hypersolver trained in both scenarios perform well, but the monotonically increasing  $w_k$  which we use (i.e., putting larger emphasis on later iterations) perform best. Moreover, removing either the initializer loss  $\mathcal{L}_{\text{init}}$  (note that we still keep the initializer  $h_\phi$  itself, just don't supervise it; see the  $\blacklozenge$ ) or the alpha loss (the  $\bullet$  curve)  $\mathcal{L}_\alpha$  impacts the performance. Interestingly, removing the  $\mathcal{L}_{\text{init}}$  loss on the initializer yields even worse performance than even removing the initializer itself (the  $\blacklozenge$  curve). However, we note that all of these ablative settings still significantly outperform the speed/accuracy efficiency than the generic solvers, which suggests the overall benefit of using custom learnable fixed-point solvers for implicit models.

### 6.3.4 Caveats

We also note two caveats for our approach. First, as mentioned in Sec. 6.2.3, backpropagating through the HyperAnderson iterations means the memory could grow with the number of steps



$K$  that we run for. However, we don't find this to be problematic in practice, as we observed the training of these hypersolvers to be very insensitive to batch size, and that at inference time hypersolvers *do* easily generalize to iterations  $> K$  (see Fig. 6.6a). Second, though our method brings consistent improvements over generic solvers, in some cases a certain amount of iterations may still be required for good performance (e.g.,  $f_\theta$  is a  $3 \times 3$  convolution and the input is a large image), which we also discussed in Sec. 4.2.6 in Chapter 4.

## 6.4 Discussion

We introduce a neural fixed-point solver in this chapter for the implicit deep networks. The approach is simple, customizable, and extremely lightweight. Unlike prior efforts that regularize the structures or parameterizations of the implicit layer design (usually at the cost of accuracy), we propose to exploit how equilibrium models decouple the representation (i.e.,  $f_\theta$ ) from the forwards computation (i.e., solver). We directly learn a *model-specific* equilibrium solver that provides: 1) better-informed initial guesses; and 2) parameterized iterations that generalize Anderson acceleration and take into account future steps. Our experiments show that these modifications substantially improve the speed/accuracy trade-off across diverse large-scale tasks, while adding almost no overhead to training. We see these encouraging results as a significant step towards making implicit equilibrium models more practical at deployment while complementing the previously introduced approaches in this thesis.



## **Part III**

# **Extensions of the Deep Equilibrium Models**



## Chapter 7

# Building Fast and Cheap Deep Equilibrium Optical Flow Estimators

In this chapter, we will demonstrate a real-world example of how these previously introduced techniques (e.g., constant memory, stabilization, adaptive computation, inexact gradients, etc.) can be integrated to bring cutting-edge applications in optical flow problems.

Optical flow estimation is the classic computer vision task of predicting the pixel-level apparent motions of objects and surfaces between video frames [24, 64, 79, 101, 158, 225, 266]. Learning based approaches to this problem first proposed the use of conventional deep convolutional networks [64, 107, 266], and has shown to outperform classical methods. Recent progress has shown that finite-step, unrolled and recurrent update operations significantly improve the estimation quality, exemplified by the emergence of the RAFT [225] method. Contemporary optical flow models that employ this approach typically rely on a Gated Recurrent Unit (GRU) [50] to *iteratively refine* the flow estimate. This approach was motivated to emulate traditional optimization-based methods, and the update operators defined accordingly have become the standard design for state-of-the-art flow models [70, 113, 115, 217, 225].

Despite their superior performance, these rolled-out recurrent networks suffer from a few drawbacks. First, training these models involves tracking a long hidden-state history in the backpropagation-through-time (BPTT) algorithm [241], which yields a significant computational and memory burden. Therefore, these models tend to scale poorly with larger images and more iterations. Second, although these models were designed to emulate *traditional optimization approaches* which solve for a “stable estimate” with as many steps as needed, the recurrent networks do not directly model such a minimum-energy optima state. Rather, they stop after a predefined  $L$  update steps, and are still trained in a path-dependent way using BPTT. We also show later in Fig. 7.3 that the GRUs frequently oscillate instead of converging.

This suggests a natural connection between the optimization perspectives of optical flows with the equilibrium modeling that this thesis studies. Specifically, in this chapter, we introduce deep equilibrium (DEQ) flow estimators, based on explorations in the previous chapters. Our method functions as a *substantially superior* and natural framework to replace the existing recurrent, unrolling-based flow estimation approach. There are multiple reasons why a DEQ-based approach is preferable. **First**, instead of relying on the naïve iterative layer stacking, DEQ models define their outputs as the fixed points of a single layer  $f_\theta$  using the input  $\mathbf{x}$ , i.e.,  $\mathbf{z}^* = f_\theta(\mathbf{z}^*; \mathbf{x})$ ,

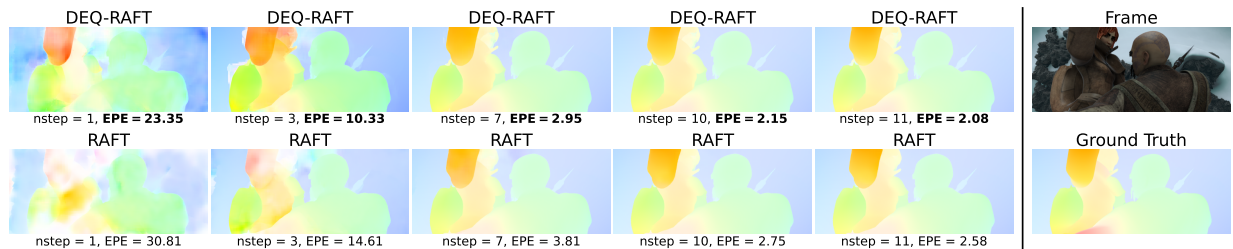


Figure 7.1: A DEQ flow estimator directly models the flow as a path-independent fixed-point solving process. We propose to use this implicit framework to replace the existing recurrent approach to optical flow estimation. The DEQ flows converge faster, require less memory, are often more accurate, and are compatible with prior model designs (e.g., RAFT and GMA).

modeling an “infinite-layer” equilibrium representation. We can directly solve for the fixed point using specialized black-box solvers, e.g., quasi-Newton methods [8, 34], in a spirit much more coherent with the traditional optimization-based perspective [79, 101]. This approach expedites the stable flow estimation process while often yielding better results. **Second**, we no longer need to perform BPTT. Instead, DEQ models can directly differentiate through the final fixed point  $\mathbf{z}^*$  without having to store intermediary states during the forward computation, considerably lowering the training memory cost. **Third**, this fixed-point formulation justifies numerous implicit network enhancements such as 1) fixed-point reuse from adjacent video frames; and 2) inexact gradients [81, 86, 87, 189] (see Chapter 5). The former helps avoid redundant computations, thereby substantially accelerating flow estimations; and the latter makes the backward pass computationally *almost free*! **Fourth**, the DEQ approach is not predicated on any specific structure for  $f_\theta$ . Therefore, DEQ flow is a *framework* that is orthogonal to, and thus directly applicable with, a wide range of these SOTA flow estimation model designs (e.g., RAFT [225], GMA [113], and Depthstillation [3]), and we can obtain the aforementioned computational and memory benefits with even additional gain based on the specific structure of  $f_\theta$ .

In addition to suggesting DEQ flow estimators as a superior replacement to the existing recurrent approach, we also provide an additional method, motivated by the deep supervision technique in computer vision [135, 225, 249], to tackle the longstanding instability challenge of training DEQ networks [18, 20, 45, 244] (see Chapter 4). We propose a novel, sparse fixed-point correction scheme that substantially stabilizes our DEQ flow estimators.

The contributions of the approach presented in this chapter are as follows. **First**, we propose the deep equilibrium (DEQ) approach as a new natural starting point for formulating optical flow methods. A DEQ approach directly models and substantially accelerates the fixed-point convergence of the flow estimation process, avoids redundant computations across video frames, and comes with an almost-free backward pass. **Second**, we show that the DEQ approach is orthogonal to, and thus compatible with, the prior modeling efforts (which focus on the model design and feature extraction) [113, 225] and data-related efforts [217]. With DEQ, these prior arts are now more computationally and memory efficient as well as more accurate. For instance, on KITTI-15 [85] (*train*) a zero-shot DEQ-based RAFT model further reduces the state-of-the-art F1-all measure by 14.0% while using the same underlying RAFT operator. **Third**, we introduce a sparse fixed-point correction scheme that significantly stabilizes DEQ models on optical flow

problems while only adding minimal cost, and show that on flow estimation tasks this approach is even superior to Jacobian-based regularization in Chapter 4.

This chapter is primarily based on the work published in CVPR 2022 [21].

## 7.1 Related Work on Iterative Optical Flow

Although optical flow is a classical problem, there has recently been substantial progress in the area. Earlier methods [27, 33, 101, 239, 258] formulated the optical flow prediction as an energy minimization problems using continuous optimization with different objective terms. This perspective inspired multiple improvements that used discrete optimization to model optical flows, i.e., those based on conditional random fields [163], global optimization [44], and inference on the global 4D cost volume [252]. More recently, with the advancement of deep learning, there have been an explosion of efforts trying to emulate these optimization steps via deep neural networks. For example, a number of optical flow methods are based on deep architectures that rely on coarse-to-fine pyramids [64, 106, 107, 109, 215, 216, 254]. Specifically, recent research efforts have turned to iterative refinements, which typically involves stacking multiple direct flow prediction modules [109, 190]. The RAFT model, which inspired this work, first showed they could achieve state-of-the-art performance on optical flow tasks using a correlation volume and a convolutional GRU update operator that mimics the behavior of traditional optimizers, which tends to converge to a stable flow estimate. Built on top of this recurrent unrolling framework of RAFT, Jiang et al. [113] introduced an additional self-attention-style global motion aggregation (GMA) module prior to the recurrent stage to improve the modeling of the occlusions. Another contemporary work, AutoFlow [217], exploits bilevel optimization to automatically render and augment training data for optical flow. Finally, Jiang et al. [114] proposes to speed up these flow estimators by replacing the dense correlation volume with a sparse alternative.

The focus of this chapter is on a direction that is largely orthogonal to and thus complementary to these modeling efforts. As a part of this thesis, we challenge and improve the “default” *recurrent, unrolled* formulation of training flow estimators *themselves*. With the help of the equilibrium approach, we can maintain a dynamical-system-based, convergent flow estimation method while paying substantially less computation and memory costs.

## 7.2 Method

We start by introducing some preliminaries of existing flow estimators. These modules are typically applied directly on raw image pairs, with the extracted representations then passed into the iterative refinement stage. We use RAFT as the illustrative example here while noting that cutting-edge flow estimators generally share similar structure (i.e., for context extraction and visual correlation computations).

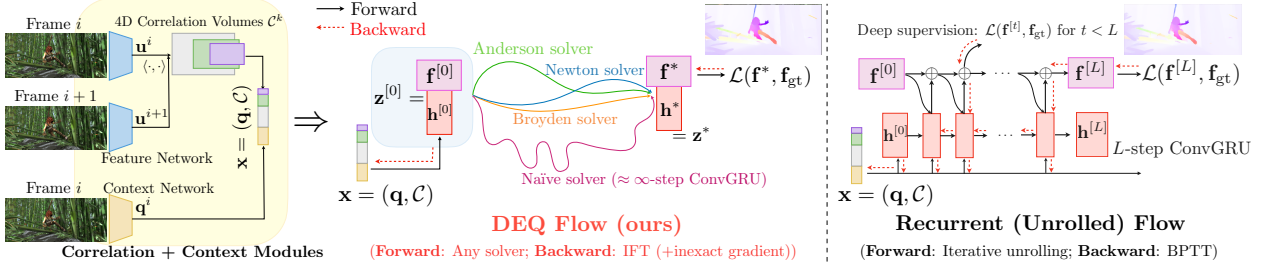


Figure 7.2: A visual comparison of the DEQ flow estimator and the recurrent unrolled flow estimator. After the correlation and context modules (see Sec. 7.2.1), a DEQ flow uses a fast, black-box fixed-point solver (e.g., Anderson) to directly solve for a stable (fixed-point) flow  $\mathbf{z}^* = (\mathbf{h}^*, \mathbf{f}^*)$ , and differentiate through  $\mathbf{z}^*$  with a cheap inexact gradient. This makes a DEQ flow’s backward pass almost free. In contrast, a recurrent flow estimator has to be unrolled for many steps, and needs to perform BPTT, which is costly in both computation and memory.

## 7.2.1 Preliminaries

Given an RGB image pair  $\mathbf{p}^1, \mathbf{p}^2 \in \mathbb{R}^{3 \times H \times W}$ , an optical flow estimator aims to learn a correspondence  $\mathbf{f} \in \mathbb{R}^{2 \times H \times W}$  between two coordinate grids  $\mathbf{c}^1, \mathbf{c}^2$  (i.e.,  $\mathbf{f} = \mathbf{c}^2 - \mathbf{c}^1$ ), which describes the per-pixel motion between consecutive frames in the horizontal ( $dx$ ) and vertical ( $dy$ ) directions. To process the matched image pair, we first encode features  $\mathbf{u}^1, \mathbf{u}^2 \in \mathbb{R}^{C \times H \times W}$  of  $\mathbf{p}^1, \mathbf{p}^2$ , and produce a context embedding  $\mathbf{q}$  from the first image  $\mathbf{p}^1$ . Then, we construct a group of pyramid global correlation tensors  $\mathcal{C} = \{\mathcal{C}^0, \dots, \mathcal{C}^{p-1}\}$ , where  $\mathcal{C}^k \in \mathbb{R}^{H \times W \times H/2^k \times W/2^k}$  is found by first calculating the inner product between all pairs of hyperpixels in  $\mathbf{u}^1$  and  $\mathbf{u}^2$  as  $\mathcal{C}^0$ , i.e.,

$$\mathcal{C}_{ijmn}^0 = \sum_d \mathbf{u}_{ijd}^1 \mathbf{u}_{mnd}^2 \quad (7.1)$$

followed by downsampling the last two dimensions to produce  $\mathcal{C}^k$  ( $k > 0$ ). The correlation pyramid  $\mathcal{C}$  and context embedding  $\mathbf{q}$ , which allow the model to infer large motions and displacements in a global sense, are then passed as inputs into the *iterative refinement* stage.

In this work, we keep the correlation and context computation part intact (see Fig. 7.2) and concentrate on the iterative refinement stage. We refer interested readers to Teed and Deng [225] for a more detailed description of the feature extraction process.

## 7.2.2 Deep Equilibrium Flow Estimator

Due to the inherent challenges of the flow estimation task, prior works have shown that explicit neural networks struggle to predict the flow accurately, requiring a prohibitively large number of training iterations [64]. Recent works [3, 113, 225] have resorted to mimicking the flavor of traditional optimization-based algorithms [101] with RNNs (e.g., convGRUs). However, these methods are still quite different from the traditional methods in a few ways. For example, optimization-based methods 1) have an adaptive and well-defined stopping criteria (e.g., whenever they reach the optima); 2) are agnostic to the choice of solver (e.g., first- or second-order methods); and 3) are essentially path-independent (i.e., the output alone is the only thing we should need).



None of these properties are directly characterized by the finite-step unrolling of recurrent networks.

We propose to close this gap with a DEQ-based approach. Specifically, given the context embedding  $\mathbf{q}$  and the pyramid correlation tensor  $\mathcal{C}$ , a DEQ flow estimator simultaneously solves for the fixed-point convergence of two alternate streams: 1) a latent representation  $\mathbf{h}$ , which constructs the flow updates; and 2) the flow estimate  $\mathbf{f}$  itself, whose updates are generically related as follows:

$$\begin{aligned}\mathbf{h}^{[t+1]} &= \mathcal{H}(\mathbf{h}^{[t]}, \mathbf{f}^{[t]}, \mathbf{q}, \mathcal{C}) \\ \mathbf{f}^{[t+1]} &= \mathcal{F}(\mathbf{h}^{[t+1]}, \mathbf{f}^{[t]}, \mathbf{q}, \mathcal{C}).\end{aligned}\tag{7.2}$$

This formulation captures the form of prominent flow estimator model designs like RAFT [225] or GMA [113]. Formally, the input  $\mathbf{x} = (\mathbf{q}, \mathcal{C})$  and model parameters  $f_\theta = (\mathcal{H}, \mathcal{F})$  jointly define a dynamical system that the DEQ flow model can *directly* solve the fixed-point for using the following flow update equation in its forward pass:

$$(\mathbf{h}^*, \mathbf{f}^*) = \mathbf{z}^* = f_\theta(\mathbf{z}^*; \mathbf{x}) = f_\theta((\mathbf{h}^*, \mathbf{f}^*); \mathbf{x}).\tag{7.3}$$

Intuitively, this corresponds to an “infinite-depth” feature representation  $\mathbf{z}^*$  where, if we perform one more flow update step  $f_\theta$ , both flow estimation  $\mathbf{f}$  and latent state  $\mathbf{h}$  will not change (i.e., an “equilibrium”). As we can leverage much more advanced root solving methods like quasi-Newton methods (see Chapter 2 and Chapter 6) to find the fixed point, the DEQ approach guarantee a much faster (superlinear) and better-quality convergence than if we perform infinitely many naïve unrolling steps (as do recurrent networks but only up to a finite number of steps due to computation and memory constraints). Moreover, as we have shown in the previous chapters and other prior work have demonstrated, the exact structure of  $f_\theta$  subsumes a wide variety of model designs, such as a Transformer block [18, 233], a residual block [19, 96], or a graph layer [92, 149, 184]. Similarly, for the deep equilibrium flow estimator, Eq.(7.2) engulfs exactly the designs of state-of-the-art optical flow models, which we follow and *use without modification*. For example, for Recurrent All-pairs Field Transforms (RAFT) [225], we can write a DEQ-flow version that solves the following fixed-point system:

$$\begin{aligned}\mathbf{x} &= \text{Conv2d}([\mathbf{q}, \mathbf{f}^*, \mathcal{C}(\mathbf{f}^* + \mathbf{c}^0)]) \\ \mathbf{h}^* &= \text{ConvGRU}(\mathbf{h}^*, [\mathbf{x}, \mathbf{q}]) \\ \mathbf{f}^* &= \mathbf{f}^* + \text{Conv2d}(\mathbf{h}^*),\end{aligned}\tag{7.4}$$

where  $\mathcal{C}(\mathbf{f}^* + \mathbf{c}^0)$  stands for the correlation lookup as in RAFT [225]. A different layer design, called the Global Motion Aggregation (GMA) [113], can be similarly instantiated in DEQ flows where we solve for the equilibrium  $\mathbf{z} = (\mathbf{h}, f_\theta)$  that satisfies

$$\begin{aligned}\mathbf{x} &= \text{Conv2d}([\mathbf{q}, \mathbf{f}^*, \mathcal{C}(\mathbf{f}^* + \mathbf{c}^0)]) \\ \hat{\mathbf{x}} &= \text{Attention}(\mathbf{q}, \mathbf{q}, \mathbf{x}) \\ \mathbf{h}^* &= \text{ConvGRU}(\mathbf{h}^*, [\hat{\mathbf{x}}, \mathbf{x}, \mathbf{q}]) \\ \mathbf{f}^* &= \mathbf{f}^* + \text{Conv2d}(\mathbf{h}^*)\end{aligned}\tag{7.5}$$

where  $\text{Attention}(\cdot)$  is the self-attention module, see [113, 233].

In order to update and train such a DEQ flow estimator, as with the previous chapters, we can directly differentiate through this “infinite-level” flow state,  $(\mathbf{h}^*, \mathbf{f}^*)$ , even without any knowledge of the fixed-point convergence trajectory. We write out the (DEQ flow version of) implicit function theorem here again, while referring readers to Chapter 2 for a much more thorough discussion and proof.

**Corollary 1.** (*DEQ-Flow Implicit Function Theorem (IFT); see Chapter 2*) *Given the fixed-point flow representation  $\mathbf{z}^* = (\mathbf{h}^*, \mathbf{f}^*)$ , the corresponding flow loss  $\mathcal{L}(\mathbf{h}^*, \mathbf{f}^*, \mathbf{f}_{gt})$  and input  $\mathbf{x} = (\mathbf{q}, \mathcal{C})$ , the gradient of DEQ flow is given by*

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^*} \left( I - \frac{\partial f_\theta}{\partial \mathbf{z}^*} \right)^{-1} \frac{\partial f_\theta(\mathbf{z}^*; \mathbf{x})}{\partial \theta} \quad (7.6)$$

As a DEQ flow only requires the final equilibrium flow estimation, this means a huge memory reduction: whereas an  $L$ -step recurrent flow estimator takes  $O(L)$  memory to perform BPTT, a DEQ estimator reduces the overhead by a factor of  $L$  to be  $O(1)$  (e.g., RAFT uses  $L = 12$  for training (and many more for inference), so using a DEQ flow can theoretically reduce the iterative refinement memory cost by  $12\times$ ).

To summarize, a DEQ flow’s forward pass directly solves a fixed-point flow-update equation; and its backward pass relies only on the final optimum  $\mathbf{z}^*$ , which make this flow estimation process much more akin to the traditional optimization-based perspective [101].

### 7.2.3 Accelerating DEQ Flows

Formulating optical flow estimation as a deep equilibrium solution also enables us to fully exploit the toolkit from implicit deep learning introduced in the previous chapters. We elaborate below on how these previous techniques (e.g., inexact gradient; see Chapter 5) can substantially help us improve the forward and backward pipeline and significantly simplify the overall overhead of modern flow estimators.

**Inexact Gradients for Training DEQs.** Despite the niceness of the Corollary 1, inverting the Jacobian term could quickly become intractable as we deal with high-dimensional feature maps. While a way around is to solve the linear system by exploiting fast vector-Jacobian products  $\mathbf{g}^\top = \mathbf{g}^\top \frac{\partial f_\theta}{\partial \mathbf{z}^*} + \frac{\partial \mathcal{L}}{\partial \mathbf{z}^*}$  (see Chapter 2), this is still iterative in nature and in practice no cheaper than the forward flow solving process.

As introduced in Chapter 5, recent work [81, 86, 189] suggest that these equilibrium methods’ backward dynamics could typically be trained, and even benefit from, simple approximations of the IFT, while still modeling an “infinite-depth” representation through the fixed-point forward pass. That is, we do not need the exact solution to Corollary 1 to train these networks; instead we could use

$$\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{\partial \widehat{\mathcal{L}}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^*} D \frac{\partial f_\theta(\mathbf{z}^*, \mathbf{x})}{\partial \theta} \quad (7.7)$$

where  $D$  is a Jacobian (inverse) approximation term (see Sec. 5.1, where we analyzed the phantom gradient  $\mathcal{A} = D \frac{\partial f_\theta}{\partial \theta}$ ). Indeed, Sec. 5.2 and 5.3 have shown that there are numerous ways that

we could build this approximation  $D$ ; e.g., using truncation of infinite series (like UPG and NPG), or simply use  $D = I$  which results in a Jacobian-free gradient [81]. Empirically, we find that Jacobian-free gradients already work very well on most of the DEQ flow estimators, which essentially simplifies the backward pass of a DEQ flow to  $\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{\partial \mathcal{L}}{\partial \mathbf{z}^*} \frac{\partial f_\theta(\mathbf{z}^*, \mathbf{x})}{\partial \theta}$ . Therefore, unlike the BPTT-based recurrent framework used by existing flow estimators, a DEQ flow estimator’s backward pass that uses inexact gradient could consist of a single step (and thus is *almost free*)! Empirically, since we almost eliminate the backward pass cost, the inexact gradients significantly reduce the total training time for DEQ flow estimator further by a factor of almost  $2\times$ .

Such capability of using inexact gradients is a direct and unique consequence of the fixed-point formulation, as we proved in Chapter 5, and assumes a certain level of stability for the underlying dynamics.

**Sparse fixed-point correction of DEQ flows.** Another longstanding challenge in training equilibrium models, which was highlighted in Chapter 4, is *the growing instability problem* (Sec. 4.2.2). In the context of flow estimation, this means the stable flow estimate  $\mathbf{z}^* = (\mathbf{h}^*, \mathbf{f}^*)$  could become computationally expensive to reach. This suggests that the optical flow estimation process gets slower during training.

While Sec. 4.3 describes how we could alleviate this issue via Jacobian regularizations (at a small cost to the performance), in this work/chapter, we propose to sparsely apply a fixed-point correction term to stabilize the flow convergence that works even better (see empirical evidence in Bai et al. [21]). Formally, suppose the black-box DEQ flow solver (e.g., Broyden’s method) yields a convergence path  $(\mathbf{z}^{[0]}, \dots, \mathbf{z}^{[i]}, \dots, \mathbf{z}^*)$ , where  $\mathbf{z}^{[0]}$  is the initial guess and  $\mathbf{z}^*$  is the final flow estimate. We then randomly pick  $\mathbf{z}^{[i]} = (\mathbf{h}^{[i]}, \mathbf{f}^{[i]})$  on this path (e.g., can be uniformly spaced), and define our total loss to be

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{main}} + \mathcal{L}_{\text{cor}} = \underbrace{\|\mathbf{f}^* - \mathbf{f}_{\text{gt}}\|_2^2}_{\text{main loss}} + \gamma \underbrace{\|\mathbf{f}^{[i]} - \mathbf{f}_{\text{gt}}\|_2^2}_{\text{fixed-point correction}} \quad (7.8)$$

where  $\gamma < 1$  is a loss weight hyperparameter. This was inspired by the dense step-wise deep supervision used traditional vision literature [135, 225, 249]. However, our application here differs in two significant ways. First, we apply this in a very sparse manner, with our primary goal being correcting instability of a dynamical system. Second, unlike in RAFT, which performs costly BPTT through the RNN chain, this fixed-point correction loss is still *path-independent* and can be understood as a *coarse-grained* fixed-point estimate. Therefore, we could also perform inexact gradient updates on this correction loss as well; i.e.,

$$\frac{\partial \mathcal{L}_{\text{cor}}}{\partial \theta} \approx \gamma \frac{\partial \mathcal{L}_{\text{cor}}}{\partial \mathbf{z}^{[i]}} \frac{\partial f_\theta(\mathbf{z}^{[i]}, \mathbf{x})}{\partial \theta}. \quad (7.9)$$

Empirically, we find this significantly stabilizes the DEQ flow estimator while having no noticeable negative impact on performance. This result is in sharp contrast to existing stabilization methods like Jacobian regularization [20] which 1) apply only locally to  $\mathbf{z}^*$ ; and 2) usually slightly hurt model accuracy (also see the ablation study in Sec. 7.3). Moreover, thanks to the inexact gradient in Eq. (7.9), Eq. (7.9) adds almost no extra computation or memory cost.

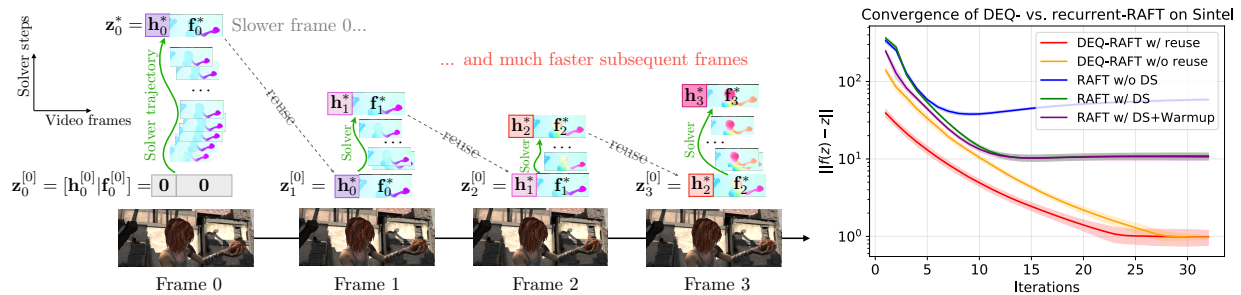


Figure 7.3: (Left) By reusing fixed-point  $\mathbf{z}^*$  from the previous frame’s flow estimation, we can “jump start” the subsequent equilibrium solving, essentially amortizing the solver cost and speeding up convergence. (Right) Comparing forward convergence of DEQ and recurrent flow estimators on Sintel videos (50 frames). "DS" stands for deep supervision used by RAFT. DEQ flow with fixed-point reuse converges best; and overall, DEQ flows converge faster than RAFT.

While our scope is limited to flow estimation in this chapter, we believe this approach suggests a potentially valuable and lightweight solution to the generic instability issue of implicit models, which we leave for future work.

**Fixed-point reuse for better initialization.** The DEQ flow estimator’s unique formulation also inherits many useful properties from the general optimization framework. One of these nice properties is the ability to perform fixed-point reuse to further accelerate flow estimation convergence. The motivation for this comes from the fact that consecutive frames of a video are typically highly correlated. For instance, perhaps only a few objects are moving in the foreground, while most of the other content and background are nearly identical across these adjacent frames. More formally, if  $\mathbf{p}^i$ ,  $\mathbf{p}^{i+1}$ , and  $\mathbf{p}^{i+2}$  are 3 consecutive video frames, then the ground-truth optical flow  $\mathbf{f}_i$  (between  $\mathbf{p}^i$  and  $\mathbf{p}^{i+1}$ ) is usually highly correlated to the next ground-truth optical flow  $\mathbf{f}_{i+1}$ . Thus, when we perform real-time flow estimation with conventional networks like FlowNet [64] and RAFT [225], we frequently perform a lot of *redundant* computations. In contrast, with a DEQ flow, we can recycle the fixed-point solution  $\mathbf{z}_i^*$  of the previous frame, which estimates  $\mathbf{f}_i$ , as the initial guess  $\mathbf{z}_{i+1}^{[0]}$  for the subsequent frame’s fixed-point solver. Intuitively, these DEQ flows are able to automatically adjust their forward optimization by exploiting this more informed initial guess, which facilitates convergence speed. It amortizes the cost of flow estimation over long video sequences, since only frame 0 requires full fixed-point solving while the remaining frames can all recycle their predecessor’s flow. We note that such reuse is related to, but still different from the warm-up scheme of RAFT [225], which only applies to  $\mathbf{f}$ , excludes  $\mathbf{h}$ , and still has to be unrolled for many steps. In our case, because a DEQ flow directly models a fixed point, such an adaptive computation by exploiting the inductive bias of video data is well-justified.

In fact, when using an equilibrium network, such fixed-point recycle can be applied also to many other cases where, like here, the inputs are highly-correlated (recall Fig. 1.1 in Chapter 1) and thus the computations can adapt to the input complexity. We will see another example of fixed-point recycle in Chapter 8 (but in a different way, across training iterations).

Fig. 7.3 shows the practicality of fixed-point reuse on Sintel dataset video sequences. By re-using the fixed point, we can further accelerate the DEQ flow estimator’s inference by a

---

**Algorithm 4** DEQ flow (PyTorch-style). Note that we reuse the fixed point, perform fixed-point correction, and train with Jacobian-free (aka. “1-step”) inexact gradient.

---

```
# solver: fixed-point solver, e.g., Broyden [34]
# func: layer  $f_\theta$  that defines dynamic system
# dist: loss function for fixed point correction
# x: input information  $\mathbf{x}_t = (\mathbf{q}_t, \mathcal{C}_t)$  of frame  $t$ 
# z: fixed-point flow estimation  $\mathbf{z}_t^*$ 
# f: ground truth optical flow  $\mathbf{f}$ 
# freq: frequency of correction.
# gamma: coefficient of correction.
# prev_z:  $\mathbf{z}_{t-1}^*$  of the last frame (if exists)
# training: bool indicating training/inference

# Forward pass (w/ backward pass by autodiff)
def forward(x, f, gamma, freq=1, training=True, prev_z=None):
    with torch.no_grad():
        # Fixed-Point Reuse
        z, z_m = solver(func, x, freq, z0=prev_z)
    if training:
        loss = dist(f, func(z, x))
        # Fixed Point Correction w/ 1-step gradient
        for i in range(freq):
            loss += gamma[i] * dist(f, func(z_m[i], x))
        return z, loss
    return z
```

---

factor of about  $1.6\times$ . Interestingly, while RAFT’s iterative unrolling aims to mimic the iterative convergence, we find its activations usually oscillate at a relatively high level after about 14 update iterations.

To summarize, while a conventional recurrent approach like RAFT needs to be unrolled for some finite  $L$  steps and back-propagated through the same  $L$ -step chain, a deep equilibrium flow estimator: 1) leverages the implicit differentiation (thus path-independent) and requires only  $O(1)$  training memory; 2) uses inexact gradients to reduce the backward pass to  $O(1)$  computation; 3) can take advantage of correlation between adjacent frames to amortize the flow estimation cost across a long sequence; and 4) is directly compatible with prior modeling and data efforts. To help readers better understand, we provide a PyTorch-style pseudo-code for DEQ-flow in Alg. 4.

## 7.3 Experiments

We present the results of our experiments in this section. Specifically, we highlight the computational and memory efficiency of DEQ flow estimators and analyze how the fixed-point correction

Data	Method	Sintel (train)		KITTI-15 (train)		Sintel (test)		KITTI-15 (test)	
		Clean	Final	AEPE	F1-all	Clean	Final	F1-fg	F1-all
C + T	LiteFlowNet [106]	2.48	4.04	10.39	28.5	-	-	-	-
	PWC-Net [215]	2.55	3.93	10.35	33.7	-	-	-	-
	LiteFlowNet2 [107]	2.24	3.78	8.97	25.9	-	-	-	-
	VCN [254]	2.21	3.68	8.36	25.1	-	-	-	-
	MaskFlowNet [266]	2.25	3.61	-	23.1	-	-	-	-
	FlowNet2 [109]	2.02	3.54	10.08	30.0	3.96	6.02	-	-
	RAFT [225]	1.43	2.71	5.04	17.4	-	-	-	-
	DEQ-RAFT-B	1.48	2.81	5.01	16.3	-	-	-	-
	DEQ-RAFT-L	1.40	<u>2.65</u>	4.76	16.1	-	-	-	-
	DEQ-RAFT-H	1.41	2.75	<u>4.38</u>	<u>14.9</u>	-	-	-	-
	DEQ-RAFT-H <sup>†</sup>	1.34	<b>2.60</b>	<b>3.99</b>	<b>13.5</b>	-	-	-	-
	GMA[113]	<b>1.30</b>	2.74	4.69	17.1	-	-	-	-
	DEQ-GMA-B	1.35	2.90	4.84	16.2	-	-	-	-
	DEQ-GMA-L	<u>1.33</u>	2.71	4.72	16.4	-	-	-	-
	C+T+S+K+H	LiteFlowNet2 [107]	(1.30)	(1.62)	(1.47)	(4.8)	3.48	4.69	7.62
PWC-Net+ [216]		(1.71)	(2.34)	(1.50)	(5.3)	3.45	4.60	7.88	7.72
VCN [254]		(1.66)	(2.24)	(1.16)	(4.1)	2.81	4.40	8.66	6.30
MaskFlowNet [266]		-	-	-	-	2.52	4.17	7.70	6.10
RAFT [225]		(0.76)	(1.22)	(0.63)	(1.5)	1.94	<b>3.18</b>	6.87	5.10
DEQ-RAFT		(0.73)	(1.02)	(0.61)	(1.4)	<b>1.82</b>	3.23	<b>6.06</b>	<b>4.91</b>

Table 7.1: **Evaluation on Sintel and KITTI 2015 datasets.** We report the Average End Point Error (AEPE) and the F1-all measure for the KITTI dataset (lower is better). “C+T” refers to results that are pre-trained on the Chairs and Things datasets. “S+K+H” refers to methods that are fine-tuned on the Sintel, KITTI, and HD1K datasets. The bold font stands for the best result and the underlined results ranks 2nd. <sup>†</sup> corresponds to using a 3-step phantom gradient (see Chapter 5). DEQ flow sets SOTA results even w/o attention.

improves the DEQ flow. Our method achieves the SOTA zero-shot performance on both the MPI Sintel [37] dataset and the KITTI 2015 [85] dataset, with an astonishing 21.0% error reduction in the F1-all score and 14.9% improvement in EPE for KITTI-15 (while still using a similar training budget to RAFT [225]).

### 7.3.1 Results

Our quantitative evaluation is presented in table 7.1. Following previous work [113, 225], we first pretrain the DEQ flow model on the FlyingChairs [64] and FlyingThings3D [160] datasets. We then test the model on the training set of MPI Sintel [37] and KITTI 2015 [85] datasets. This model is denoted “C + T”; it evaluates the *zero-shot generalization* of the DEQ flow model. Then, we fine-tune the DEQ flow estimator on FlyingThings3D [160], MPI-Sintel [37], KITTI 2015 [85], and HD1K [127] for the test submission.

The models we train are of exactly the same size as RAFT (5.3M) and GMA (5.9M) except they use DEQ flow formulation instead of recurrent updates. They are denoted as DEQ-RAFT-B

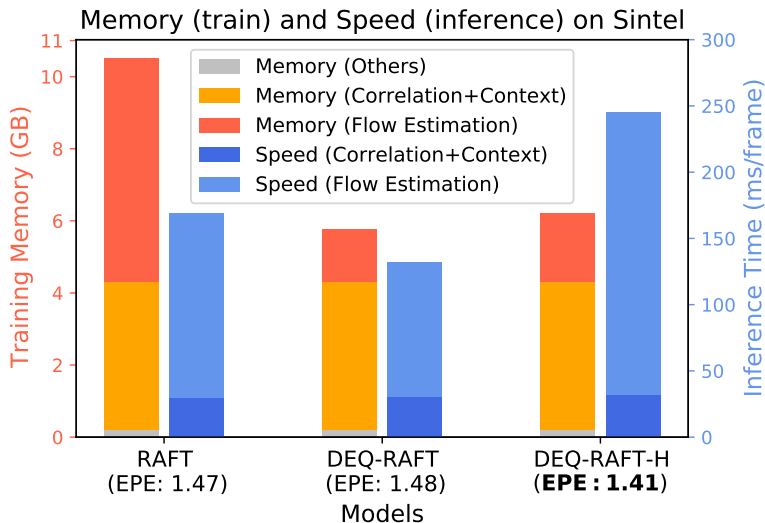


Figure 7.4: Comparing the training memory, inference speed and performance on Sintel (clean) with image size  $436 \times 1024$ . The same model design (based on RAFT) consumes much less memory and computes much quicker than the recurrent counterpart. All results are benchmarked on a single Quadro RTX 8000 GPU.

and DEQ-GMA-B, respectively. Exploiting the memory efficiency of the DEQ flow model (see 7.3.2), we can fit much larger models into the same compute budget of two 11 GB 2080Ti GPUs. To this end, we also trained DEQ-RAFT-L (8.4M) and DEQ-RAFT-H (12.8M) by increasing the the width of hidden layers inside the update operator. We also trained DEQ-RAFT-D (9.4M) by duplicating the ConvGRU within  $f_\theta$ . As shown in fig. 7.4, even the largest DEQ-RAFT-H model only consumes less than half of the flow estimation memory used by a standard-sized RAFT model, while achieving significantly better accuracy (4.38 AEPE and 14.9 F1-all score on KITTI-15, see table 7.1).

### 7.3.2 Performance-Compute Tradeoff

We further verify the aforementioned computational and memory benefits of the DEQ flow model on the Sintel (clean) [37] dataset with a RAFT-based update operator (see Eq. (7.4)) trained on FlyingChairs [64] and FlyingThings3D [160]. The results are shown in Fig. 7.4. Specifically, when training the DEQ flow estimator on Sintel with a batch size of 3 per GPU (the maximum that RAFT can fit with a 11 GB GPU), we observe that the memory cost of the flow estimation process reduces by a factor of over  $4\times$  (red bars). Note that since we keep the rest of the model intact (e.g., correlation pyramid and context extraction; see Sec. 7.2.1), the DEQ flow estimator does not improve those parts of the memory burden, which now becomes the new dominant source of memory overhead. In addition, when we use the model for inference, we follow Teed and Deng [225] using 32 recurrent steps for RAFT (with warm-start), and the Anderson solver for DEQ-RAFT (with reuse), which stops if relative residual falls below  $\varepsilon = 10^{-3}$ . Our results suggest that the DEQ flow converges to an accurate solution, and it is in practice about 20% faster than the RAFT models with the same structure and size (blue bars). Finally, we show that we can exploit such memory savings to build even larger and more accurate flow estimators

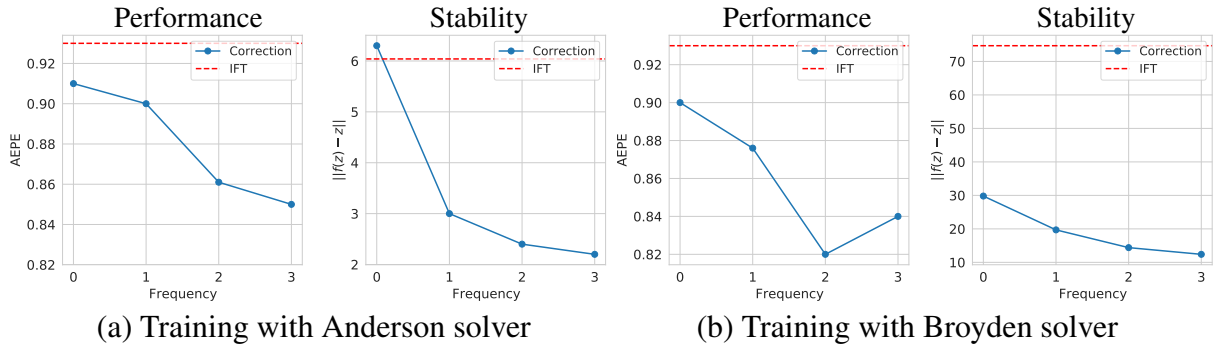


Figure 7.5: Performance and convergence stability (measured by absolute residual error) of the DEQ flow. Frequency indicates how many correction terms we pick, with 0 meaning no correction. We also compare with Jacobian regularization [20] in Fig. 7.6. DEQ flows trained with our proposed correction enjoy superior performance and stability.

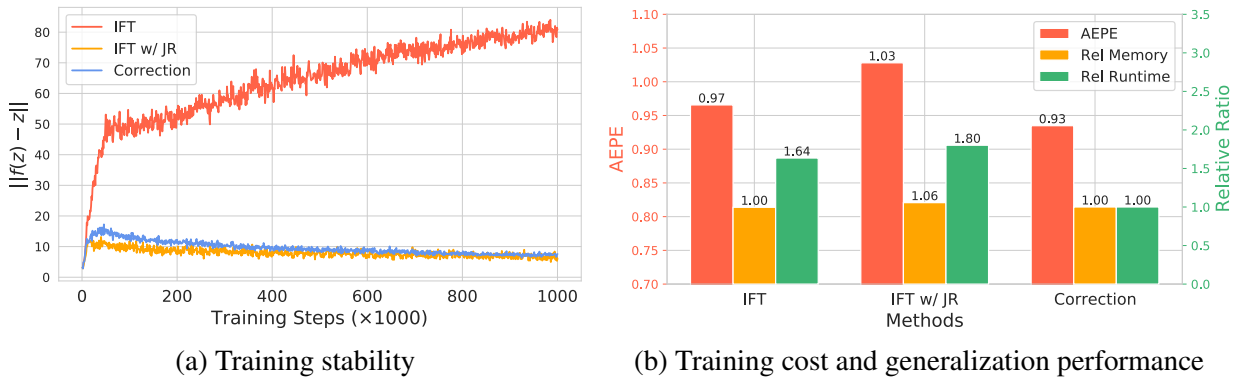


Figure 7.6: **Comparison of IFT, Jacobian Regularization and Fixed-Point Correction.** Given a limited forward solver budget, the fixed-point correction protocol successfully stabilizes training and shows accelerated fixed-point convergence.

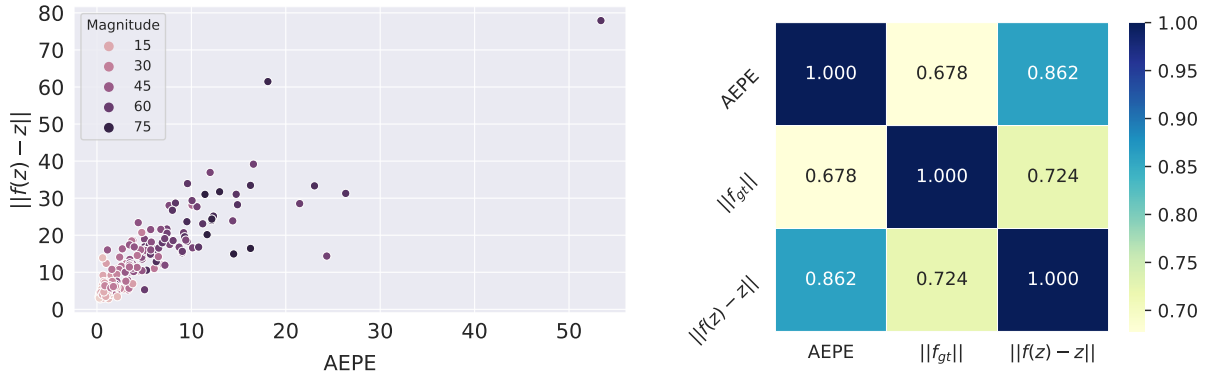
(DEQ-RAFT-H), while still staying well within the compute and memory budget.

### 7.3.3 Ablation Study

In this subsection, we aim to answer the following questions: 1) How useful is the fixed-point correction compared with canonical IFT in performance, stability, and speed? 2) How does the convergence of a DEQ flow correlate with the quality of the flow estimation? As in S7.3.2, we use the model design from RAFT to instantiate our DEQ flow. By default, we conduct the ablation experiments on the FlyingChairs [64] dataset using the default training hyperparameters of RAFT and report the Average End Point Error (AEPE) on its validation split.

**Stabilizing DEQ by Fixed-Point Correction.** As mentioned in Sec. 7.2.3, unregularized canonical DEQ models (as well as other implicit networks like Neural ODEs [45]) typically suffer from a growing instability issue typically symptomized by an increasingly costly forward fixed-point solving process. We perform an ablation experiment to study how our proposed sparse





(a) Correlation between convergence and performance. Harder examples (e.g., those with large motion) typically lead to more challenging fixed-point convergence. (b) Pearson correlation coefficient across per-frame EPE, fixed-point error, and the magnitude of flow.

Figure 7.7: Visualizations of DEQ models’ instability and inefficiency problems. “Ours” refers to the regularized DEQ models, which will be introduced in Sec. 4.3.

fixed-point correction scheme could help alleviate this issue. To understand the scheme’s effect, we train a DEQ flow model using both an Anderson [8] and a Broyden [34] solver with 36 and 24 forward iterations, respectively. For simplicity, we equally divide the solver convergence trajectory into  $r + 1$  segments (where  $r$  is the frequency in fig. 7.5) and impose a correction loss after each trajectory clip. As mentioned in Sec. 7.2.3, we apply Jacobian-free (i.e., one-step) inexact gradient to the correction loss.

We visualize results of DEQ flow models trained with 3 different settings: 1) a DEQ flow trained by IFT directly without an auxiliary correction loss; 2) a DEQ flow trained by inexact gradient without an auxiliary correction loss; and 3) DEQ flows trained by inexact gradient *as well as* 1-3 fixed-point correction terms. Our results are reported in terms of AEPE (which measures performance) and absolute fixed-point residual error  $\|f_{\theta}(\mathbf{z}^*; \mathbf{x}) - \mathbf{z}^*\|_2$  (which measures stability). As shown in Fig. 7.5, our proposed fixed-point correction significantly outperforms the standard IFT training protocol by about 9%, and reduces the fixed-point error by a conspicuous margin, e.g., over 60%. Moreover, we find that this significant improvement in stability quickly diminishes as we apply more corrections; therefore in practice, we usually use one correction term. Together with the inexact gradient, the total training time can be streamlined over 45%, while the backward pass of a DEQ flow is still almost *free*.

**Correlation between Performance and Convergence.** A potential question is whether better fixed-point convergence can lead to better performance. To tackle this, we evaluate the DEQ flow model trained using the standard “C+T” training protocol (see S7.3.1) on the KITTI-15 [85] training dataset. We visualize the per-frame EPE and the convergence measured by the absolute fixed point error in fig. 7.7a and dye the scatter plot with the average norm of per-pixel flow across the frame, which can be understood as an indicator of hardness due to the large displacements. The Pearson correlation coefficient between the fixed-point error and EPE is over **0.86** (see fig. 7.7b)

supporting the claim that convergence is strongly correlated with the flow performance. From fig. 7.7b, we see that hard flows, with large motions are also challenging for a naive solver. This demonstrates the necessity of advanced solvers in DEQ flow estimation.

### 7.3.4 Qualitative Results

We also visualize the flow estimation obtained by DEQ flow in Fig. 7.8- 7.10, using consecutive frames of the MPI Sintel [37] test set.

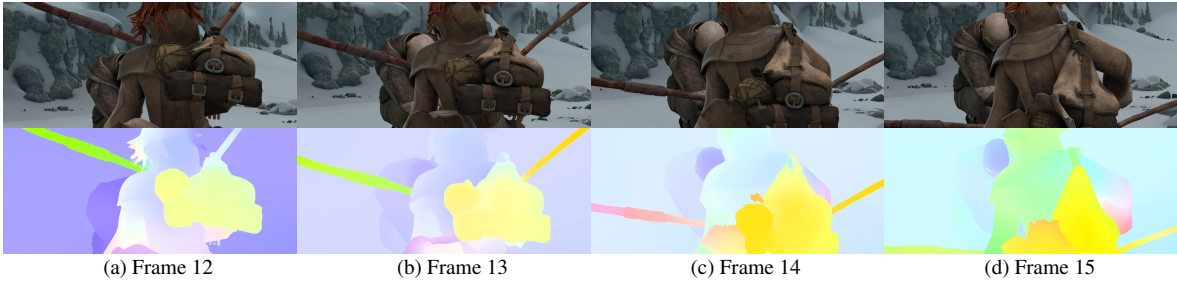


Figure 7.8: Visualization on the Sintel test set, `ambush_1` sequence of the clean split.

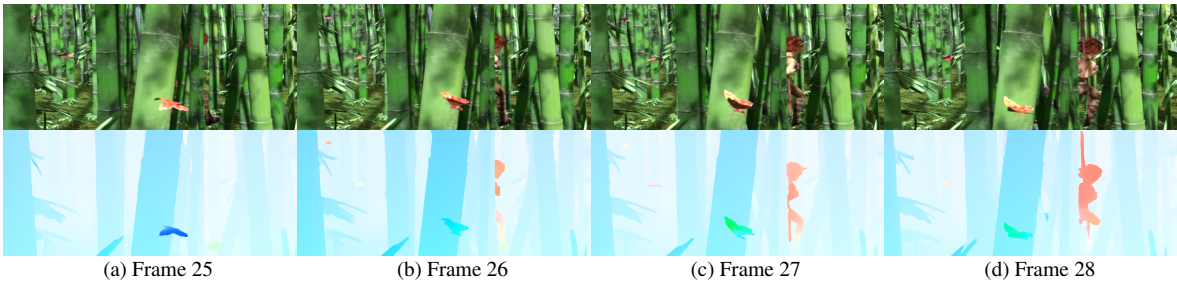


Figure 7.9: Visualization on the Sintel test set, `bamboo_3` sequence of the final split.

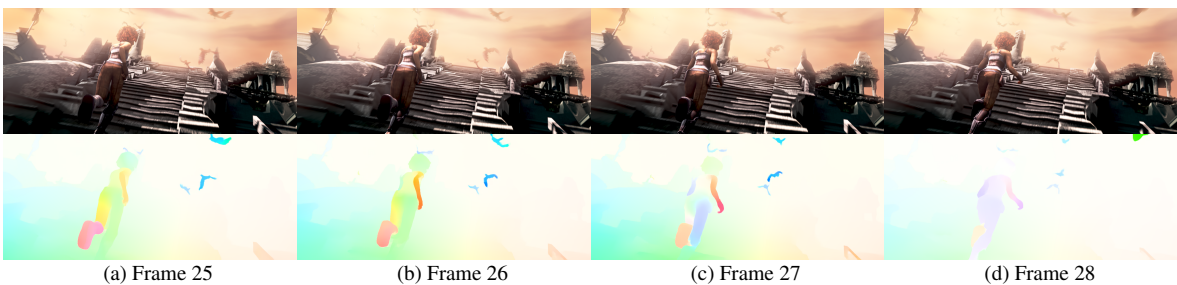


Figure 7.10: Visualization on the Sintel test set, `temple_1` sequence of the final split.

## 7.4 Limitations

The improved performance and efficiency of our approach comes at the cost of a slightly more complex training pipeline. Implementing the naïve unrolled flow estimation as presented in Teed

and Deng [225] and Jiang et al. [113] is simple using most libraries equipped with automatic differentiation that directly handle BPTT. On the other hand, our approach involves some finagling of the training protocol (e.g., fixed-point solvers, IFT, inexact gradients, etc.). To help alleviate this complexity and promote the use of DEQ-flows, we release our code at <https://github.com/locuslab/deq-flow>.

In addition, while DEQ flows provide a novel and more efficient framework to train and use these flow estimators, we still occasionally need to be careful about the stability of this approach. For example, what would happen if the solver *converges poorly* (or even diverges) on a dynamical system? In such case, the behavior of the DEQ flow estimation would not be well-defined. In practice, we rarely observe such instability (as long as we spend enough solver steps); but as we analyzed in Sec. 4, harder examples also typically lead to more lengthy convergence path. We leave a more thorough study of the flow estimation stability to future work.

## 7.5 Discussion

This chapter proposes a new, equilibrium-based framework for modeling optical flow estimation. We show that equipped with the tools introduced in the other chapters of this thesis, a deep equilibrium (DEQ) flow directly models and solves a fixed-point *stable* flow estimate which can be trained and used for inference efficiently. Such DEQ flow framework is orthogonal to, and therefore complements, other modeling- and data-related efforts in the optical flow literature. With experiments we show that we can easily integrate many of these modeling designs in the equilibrium approach and achieve results that are more accurate, faster, *and* multiple times more memory efficient.

We therefore argue that this implicit framework provides a strong (drop-in) replacement for existing recurrent, unrolled update operators used by most cutting-edge flow estimators. The DEQ flows are both more performant and lightweight— both computationally and memory-wise. We believe this suggests an exciting direction for building more efficient, large-scale and accurate optical flow models in the future.



## Chapter 8

# Implicit<sup>2</sup>: Implicit Models for Implicit Neural Representations

The concept of “implicitness” has recently been applied to various contexts of machine learning research. One particular thread is *implicit neural representations* (INR) [169, 183, 212, 223], which aims to learn a continuous representation of high-frequency, often discretely-measured signals such as large images. Formally, given a spatial-temporal coordinate input  $x \in \mathbb{R}^d$ , an implicit representation of it is a function  $\Phi : \mathbb{R}^d \rightarrow \mathcal{C}$ , where  $\mathcal{C}$  is the space of desired quantity (e.g. color, volume density, distance, etc.) and  $\Phi$  is usually parameterized with a neural network. Such implicitness has multiple benefits over the conventional discrete (e.g., grid-based) representations; e.g., as  $\Phi$  is defined on a continuous domain, an implicitly represented input consumes much less storage than the original input; as another example,  $\Phi$  is differentiable, hence allowing for the computation of higher-order derivatives. However, the training of these  $\Phi$  networks themselves are usually quite memory-consuming, since we usually deal with very high-resolution images and videos, and training is typically done in full-batch mode (e.g., when training on a batch size of  $512 \times 512$ , a simple 4-layer MLP with 1024 hidden units already requires  $> 16$  GB memory just to store the intermediate activations).

An orthogonal usage of *implicitness* is the implicit deep learning introduced in this thesis, where the word is used to characterize the nature of model architectures (rather than input representations). As introduced in Sec. 1.1 of Chapter 1, the nomenclature comes from the concept of implicit vs. explicit functions: instead of representing a model as an explicit stacking of layers, an implicit model solves a non-linear dynamical system [68] (e.g., ODEs [45, 67] or fixed-points [18, 244]) and differentiate via the implicit function theorem [128]. Previous chapters have established how these implicit models could achieve competitive results on a wide-range of realistic tasks, as well as the challenges/opportunities they face.

Despite the success of these “implicit” methods in their respective areas, the two concepts so far have rarely intersected: existing models used for implicit neural representations (INR) are neither *implicit* in nature (since they are still stacked multi-layer explicit operators trained end-to-end in the feedforward manner), nor do they exploit any properties of the implicit functions during training (e.g., implicit differentiation). In this chapter, we argue that implicit representations and implicit layers are remarkably well-suited to each other, and their use together can compensate for each other’s drawbacks.

Specifically, we propose to combine the best of both worlds by replacing the explicit models used in conventional INR learning with an implicit layer, thus dubbed (Implicit)<sup>2</sup> networks. Formally, let  $x$  be the input and  $f_\theta$  be an (often shallow) layer, the (Implicit)<sup>2</sup> approach learns the following implicit representation  $\Phi$  of the data  $x$ :

$$\Phi(x) = Wz^* + b, \text{ where } z^* = f_\theta(z^*; x) \quad (8.1)$$

Importantly, we show that these two “implicitness” complement each other well, especially in two important aspects. First, the unique large-batch training scheme of *implicit representations* as well as the forward/backward decoupling properties of *implicit models* permit us to *amortize* the cost of the iterative solver that would otherwise make implicit models slow to train. We show that, similar to the DEQ optical flow case in Chapter 7, an implicit equilibrium model can recycle the fixed-point computations across *training iterations* and leverage inexact gradient (see Chapter 5) to significantly lower the hardware requirement and computation budget needed to train INR. We evaluate (Implicit)<sup>2</sup> on multiple INR tasks for images, videos, and audios, showing substantial improvement upon existing competitive explicit-model methods like SIREN [212] or MFN [75] using equivalent-sized networks. This chapter is primarily based on the work published in NeurIPS 2021 [105].

## 8.1 Preliminaries: Implicit Neural Representations (INR)

While high-frequency data such as images and scene geometry have been traditionally represented discretely (e.g., pixel/voxel grids or mesh points), recent work has demonstrated the possibility of replacing them with continuous functions parameterized by multi-layer perceptrons (MLPs) [212, 223]. These deep networks have been used to learn differentiable representations of various forms, such as continuous spatial-temporal image representations [26, 47], high-resolution scene representations [183, 211], and volumetric rendering [169, 226], and have been shown to be significantly more compact than the conventional grid-based approaches.

However, training these INR models is not easy. For example, the training of implicit representations is usually conducted in very large batch sizes (e.g.,  $512 \times 512$ ), which renders these (albeit simple) MLPs very memory-consuming during training. As another example, recent works have shown that typical deep networks with ReLU/tanh/sigmoid non-linearities fail to capture the fine details of an input signal, and resorted to periodic non-linearities [212], Fourier feature encodings [223], or repeated application of non-linear filters (e.g., Gabor wavelets) to the input and then multiply with the features [75]. We introduce below the two latest, highly competitive models that have been developed on this task, which our work will build on in Sec. 8.2.1.

**Sine-activated Network for Implicit Representations (SIREN).** To overcome the detrimental effect of traditional non-linearities like ReLU/tanh on modeling fine details and higher-order derivative of the input signals, Sitzmann et al. [212] proposes to use sinusoidal activation functions that allow explicit supervision on any derivatives of the input signal. The resulting MLPs, though simple, have been shown to achieve state-of-the-art performance in representing images, videos,

complex geometries, and more [212]. Formally, given an input  $x$ , SIREN learns an implicit representation by the following  $L$ -layer MLP:

$$\Phi(x) = W(g_{L-1} \circ g_{L-2} \circ \dots \circ g_0)(x) + b \quad (8.2)$$

where  $g_i$  is the  $i$ -th layer of the network that computes  $g_i(h^{[i]}) = \sin(W_i h^{[i]} + b_i)$  with a sine non-linearity, and  $h^{[i]}$  denotes the hidden state at  $i$ -th layer.

**Multiplicative Filter Networks (MFN).** More recently, Fathony et al. [75] proposed to forego the SIREN-like design (i.e., MLPs with periodic activation functions) in favor of multiplicative operations that are proven to be similarly expressive. Specifically, at each layer  $i$  of the network, an MFN applies a learnable non-linear filter kernel (parameterized by  $\theta^{(i)}$ )  $g(x; \theta^{(i)})$  on the *original input*  $x \in \mathbb{R}^n$ , and elementwise multiply it with a *linear transformation* of the features; i.e., formally,

$$\Phi(x) = Wh^{[L]} + b, \text{ where } h^{[i+1]} = (W_i h^{[i]} + b_i) \cdot g(x; \theta^{(i)}) \text{ and } h^{[0]} = g(x; \theta^{(0)}) \quad (8.3)$$

Fathony et al. [75] in particular proposed two instantiations that admit sinusoids or a Gabor wavelet as the filter  $g$ , which are called FourierNet and GaborNet, respectively:

$$\begin{aligned} g_{\text{Fourier}}(x; \theta_{\text{Fourier}}^{(i)}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \\ g_{\text{Gabor}}(x; \theta_{\text{Gabor}}^{(i)}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \cdot \exp(-(\gamma^{(i)}/2) \cdot (x - \mu^{(i)})^2) \end{aligned} \quad (8.4)$$

where  $\theta_{\text{Gabor}}^{(i)} = \{\omega^{(i)} \in \mathbb{R}^n, \phi^{(i)} \in \mathbb{R}, \gamma^{(i)} \in \mathbb{R}, \mu^{(i)} \in \mathbb{R}^n\}$  (similar for  $\theta_{\text{Fourier}}^{(i)}$ ), and  $(\mu^{(i)}, \gamma^{(i)})$  denotes the mean and scale of the Gabor filter. These models have shown performance on par with or better than the periodic-nonlinearity-based networks like SIREN on a range of tasks like image and video representations.

This chapter studies the novel combinations of *implicit models* (exemplified by the equilibrium models) and *implicit representation learning* (see Fig. 8.1). Specifically, we show that these two “implicitness” are surprisingly well-suited to each other, as they almost perfectly compensate for each others’ drawbacks. We demonstrate that implicit modeling of a simple layer  $f_\theta$  substantially improves the training speed, memory, *and* performance on implicit representation tasks when compared to the aforementioned state-of-the-art deep explicit networks.

## 8.2 (Implicit)<sup>2</sup> Networks

As we discussed in Sec. 8.1, learning implicit representation for complex signals is often more effective when projecting the input coordinates onto high-frequency bases (e.g. sinusoidal functions), making the designs of these models typically different from deep networks used in more common applications, like image classifications (e.g., ResNets [96]). In this work, we propose to directly build on the designs of these aforementioned state-of-the-art layer architectures by only making minimal modifications, but modeling them as shallow implicit models. We briefly introduce below the two instantiations of our proposed (Implicit)<sup>2</sup> approach, which are based on SIREN [212] and MFN [75], respectively, followed by a discussion of how implicit representation tasks can enable us to amortize the cost of training these implicit networks significantly in practice.

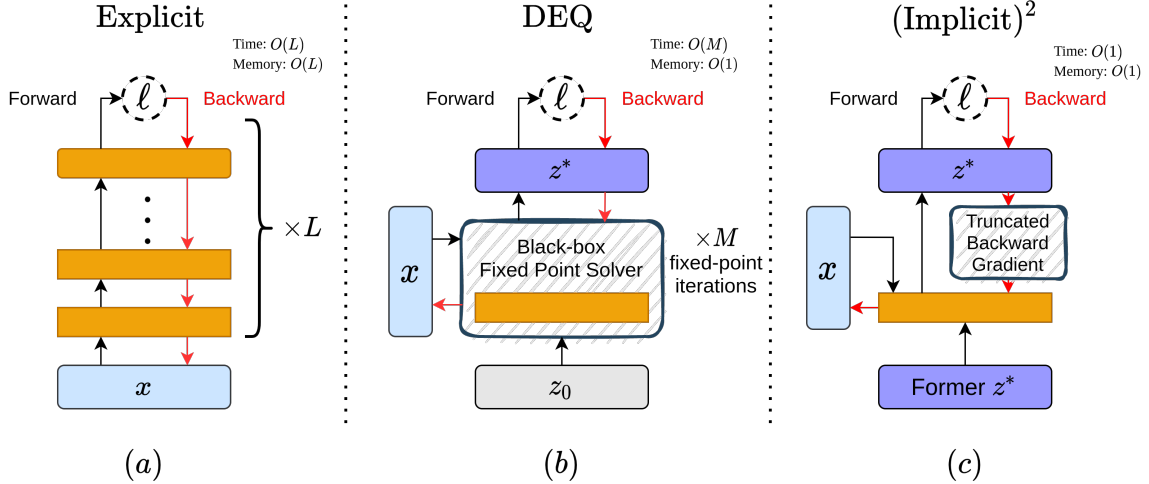


Figure 8.1: (a) Explicit networks incur  $O(L)$  complexity in both time and memory during training. (b) Original DEQ models requires constant memory, yet they take  $O(M)$  time in forward/backward passes, where  $M$  is the number of fixed-point solver steps. (c) In contrast, an  $(\text{Implicit})^2$  Network consumes constant amortized memory *and* time in both forward and backward passes.

### 8.2.1 $(\text{Implicit})^2$ Network Architectures

**Implicit Sine-activated Networks (iSIREN).** Motivated by the success of SIREN, we first propose an *input-injected* variant of SIREN (denoted as *SIREN (input inj.)*) suitable for implicit models, i.e.,

$$f_{\theta}^{\text{SIREN}}(z; x) = \sin(W(z + \sin(Vx)) + Ux + b) \quad (8.5)$$

where, following prior works on deep equilibrium models [18, 192, 244], the transformed input  $x$  is added to the hidden representation  $z$  (see Fig. 8.2). Just like the canonical SIREN layer, any order of derivative of this input-injected SIREN variant is also an input-injected SIREN of the same form. Therefore, the proposed variant inherits the same high-frequency and higher-order differentiability properties as SIREN, and we name the implicit model that solves its fixed-point as *iSIREN*.

**Implicit Multiplicative Filter Networks (iMFN).** Additionally, we introduce a variant based on the recent Multiplicative Filter Networks (MFN) [75]. As MFNs only perform linear transformations on the hidden features, which are multiplied with a non-linear filter function on the original input  $x$ , we slightly modify the layer design as follows to introduce an additional input injection:

$$f_{\theta}^{\text{MFN}}(z; x) = (W(z + g(x; \theta_2)) + b) \circ g(x; \theta_1) \quad (8.6)$$

where  $g$  denotes the filter function of choice, such as Gabor or Fourier ( $g_{\text{Gabor}}$  and  $g_{\text{Fourier}}$  in Eq. (8.4)) filters [75]. We call the implicit models defined by this layer iMFN. Moreover, just like the original MFN [75], we can show that the output of iMFN (which is the fixed-point of  $f_{\theta}^{\text{MFN}}$ ) is also a linear combination of the non-linear filter functions (see Huang et al. [105] for proof).



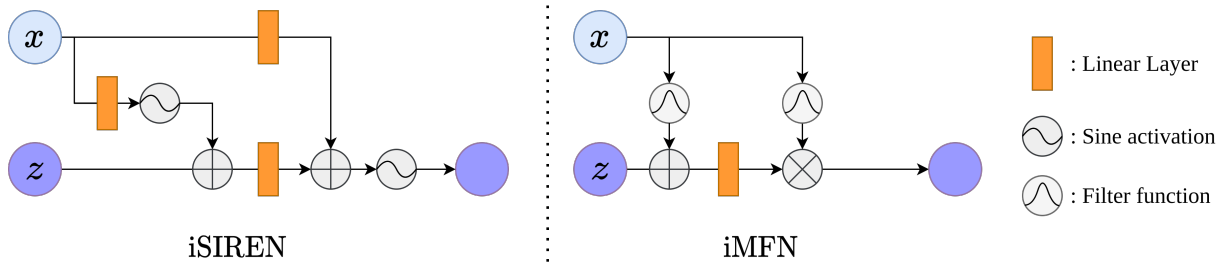


Figure 8.2: The architecture of one implicit layer  $f_\theta(z; x)$  in iMFN and iSIREN. Note that when  $z = 0$ , evaluating  $f_\theta(z; x)$  once is equivalent to the output of the respective explicit layer.

We provide visualizations of these two architectural instantiations in Fig. 8.2. With these variants, we formally introduce the corresponding (Implicit)<sup>2</sup> approaches to implicit representation tasks as follows:

$$\Phi(x) = W'z^* + b', \text{ where } z^* = f_\theta(z^*; x), \text{ and } f_\theta \in \{f_\theta^{\text{MFN}}, f_\theta^{\text{SIREN}}\} \quad (8.7)$$

## 8.2.2 Accelerated Training of (Implicit)<sup>2</sup> Networks

Compared to explicit networks, training with implicit models significantly lowers the *memory* budget required. This is especially compelling for training INR tasks, which are often memory-bottlenecked. In this subsection, we build on the previous chapters and show how we can use these techniques to improve the *speed* of these (Implicit)<sup>2</sup> models, eventually rendering them both more memory-efficient *and* time-efficient than explicit models (while frequently performing better; see Sec. 8.3).

**Fixed-point Reuse.** Suppose we fix a training step  $t$  and let the corresponding network parameter at this snapshot be denoted by  $\theta_{(t)}$ . Typically, a DEQ model solves for the fixed-point  $z_{(t)}^*$  given an input  $x$  with a black-box fixed-point solver starting with an initial guess  $z^{[0]}$  (which is usually  $\mathbf{0}$ ):

$$z_{(t)}^* = \text{RootFind}(f_{\theta_{(t)}}, x, z^{[0]}) \quad (8.8)$$

where  $z_{(t)}^*$  does not depend on this initial guess  $z^{[0]}$  of the optimization. But depending on the quality of the initial guess, the optimization process itself can take a different number of steps. For example, in the unlikely circumstance where we guessed  $z^{[0]}$  to be  $z_{(t)}^*$ , the fixed-point solvers will directly converge at iteration 0. We

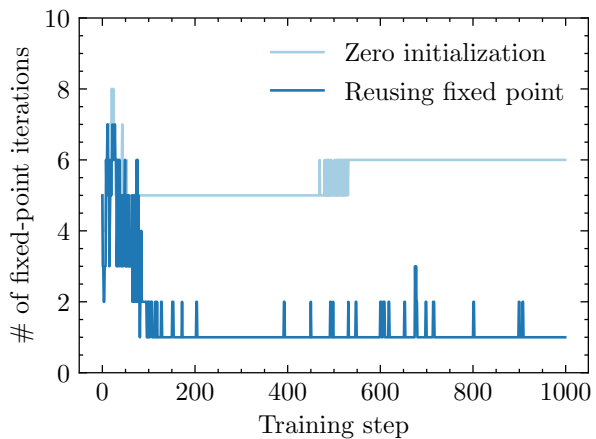


Figure 8.3: Zero initialization v.s. fixed-point reuse in terms of required steps to converge. Further details of this experiment can be found in Sec.8.3.1.

have shown in Chapter 7 another instance of how correlations across high-frequency video frames can also facilitate a better initial guess.

Similarly, we argue that the training on many implicit representation tasks, which uses the entire batch of data (e.g., to fit an RGB-color image with an INR, each pixel of this image is a “sample” of this full batch), also enables us to provide very reasonable initial guesses to our equilibrium networks. In particular, training in the full-batch mode allows us to have direct access to *all* of the fixed-points  $z_{(t)}^*$  for all inputs  $x$  in the training set immediately after step  $t$ . Assuming the model updates between iterations  $t$  and  $t + 1$  are small (which is true in practice, such as via SGD with a small learning rate  $\eta$ ), one can safely assume the fixed-points of the updated layer  $f_{\theta_{(t+1)}}(\cdot; x)$  do not deviate much from its current estimate  $z_{(t)}^*$ . Formally,  $f_{\theta_{(t+1)}}(\cdot; x) \rightarrow f_{\theta_{(t)}}(\cdot; x)$  as learning rate  $\eta \rightarrow 0$ , which implies  $z_{(t+1)}^* \rightarrow z_{(t)}^*$ . We empirically verify this on these full-batch training settings (see Fig. 8.3), where a majority of the fixed-point convergences (to a target residual level  $\varepsilon = 0.01$ ) finish in exactly 1 step.

However, we note that this does not imply all training iterations will gain a similar level of convergence boost (e.g., see the first 100 steps in Fig. 8.3). In particular, at training iteration  $t = 0$ , we do not have a “previous estimate” at all, which means we still have to solve for  $z^*$  by running the solver for several steps with a neutral initial guess. Thus, we highlight that such fixed-point reuse *amortizes* the cost of later training iterations, and indeed, the vast majority of training steps of the implicit models. Empirically, we found that performing 1 step of the fixed-point iteration is sufficient for training the iSIREN and iMFN networks; i.e., at training step  $t + 1$ , (if using naïve iterations) we simply perform

$$z_{(t+1)}^* = f_{\theta_{(t)}}(z_{(t)}^*; x). \quad (8.9)$$

**Inexact Gradient.** We also propose to use inexact gradient to accelerate the training of (Implicit)<sup>2</sup> models further. Specifically, as a recap, recall from Chapter 5 that unrolling-based phantom gradient (UPG) performs  $T$  unrolling steps at the fixed-point estimate, whereas Neumann-based phantom gradient (NPG) (Sec. 5.2) directly uses the Jacobian matrix at the fixed-point estimate and perform a  $T$ -step truncated Neumann-series computation. Moreover:

1. When the forward fixed point solving is very accurate, UPG and NPG approaches each other. In our case, since we can leverage fixed-point reuse and found NFE=1 to be sufficient (see above), this condition is satisfied.
2. When  $T = 0$ , UPG reduces to the Jacobian-free gradient [81] (see Sec. 5.3).

While the  $T = 0$  worked well for the DEQ optical flow estimation (see Chapter 7, we empirically find on INR tasks that Neumann-based phantom gradient  $T = 1$  works better. Note that this is still very cheap—it only requires one evaluation of the

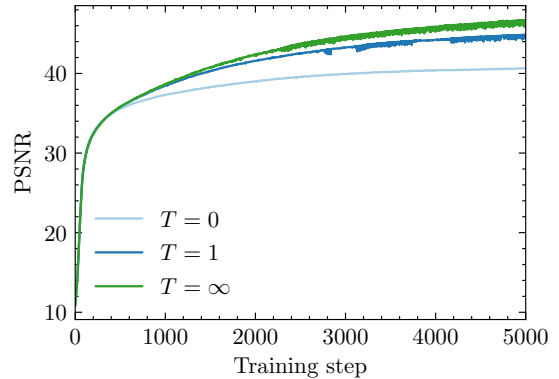


Figure 8.4: Training PSNR comparison between different levels of gradient approximation.  $T$  denotes truncation length of the inexact gradient (where we use NPG; see Chapter 5), where  $T = \infty$  indicates full IFT.

vector-Jacobian product  $\frac{\partial \ell}{\partial z^*} J_{f_\theta}$ . We show in Fig. 8.4 the performance of the model as a result of using different “approximation” levels of the inexact gradient. Formally, the inexact gradient we use has the form

$$\widehat{\frac{\partial \ell}{\partial \theta}} = \frac{\partial \ell}{\partial z^*} \sum_{i=0}^T B^i \frac{\partial f_\theta(z^*; x)}{\partial \theta}, \quad \text{where } B = J_{f_\theta}(z^*) \quad (8.10)$$

**Spectral Normalization** . We additionally apply spectral normalization on the layers  $f_\theta$ , which ensures that  $f_\theta$  (for forward passes) and  $J_{f_\theta}$  (for backward passes) are contractive, and is able to guarantee unique and stable fixed-points due to the Banach Fixed Point Theorem [23]. Specifically, we adopt the power-iteration in [25, 171] to scale the spectral norms of  $W$  in Eq. (8.5) and Eq. (8.6) after each training iteration if it become  $> 1$ , which incurs little additional computation cost [171]. However, we note that weaker regularizers such as weight normalization [199] may also suffice in practice.

## 8.3 Experiments

We evaluate our (Implicit)<sup>2</sup> networks on several representation tasks and compare the difference in performance between the implicit and explicit modeling of implicit representations across a variety of configurations. For both explicit networks and implicit networks, we use  $L$  to denote the number of layers and  $D$  to denote the hidden dimensionality of the features. Unless stated otherwise, our implicit models have exactly one  $f_\theta$  layer (i.e.,  $L = 1$ ), in the exact form we described in Sec.8.2.1. Our set of experiments is drawn from prior works where competitive INR deep networks are evaluated on [75, 212].

Overall, our results of learning INR on various domains (including images, videos, audios) suggest that the (Implicit)<sup>2</sup> approach offers clear improvements over existing explicit models used for these purposes, where we are able to achieve the same or better level of performance while being up to  $3\times$  more memory-efficient and  $3\times$  more time-efficient. Our implementation can be found at <https://github.com/locuslab/ImpSq>.

### 8.3.1 Image Representation

We first evaluate the difference between explicit networks and (Implicit)<sup>2</sup> networks on representing a high-resolution  $512 \times 512$  grayscale image, which is a commonly used goalpost for evaluating implicit representation models from the scikit-image package [232]. In particular, we fit an implicit representation function  $\Phi : (x, y) \rightarrow \mathcal{C}$ , where  $\mathcal{C}$  is the desired color space (and in this case  $\mathcal{C} = \mathbb{R}$ ). We train each model for 5000 iterations (under the same setting) using all pixels in the image (i.e., batch size 262,144), and demonstrate the final peak signal-to-noise ratio (PSNR), memory consumption, and average training step time in Fig. 8.5. The results convey two interesting facts about using (Implicit)<sup>2</sup> networks: Compared to standard 4-layer explicit networks ( $4L-256D$ ), even a small implicit network ( $1L-256D$ ) with approximately 75% fewer parameters achieves a comparable performance, while requiring only roughly 1/3 of the training

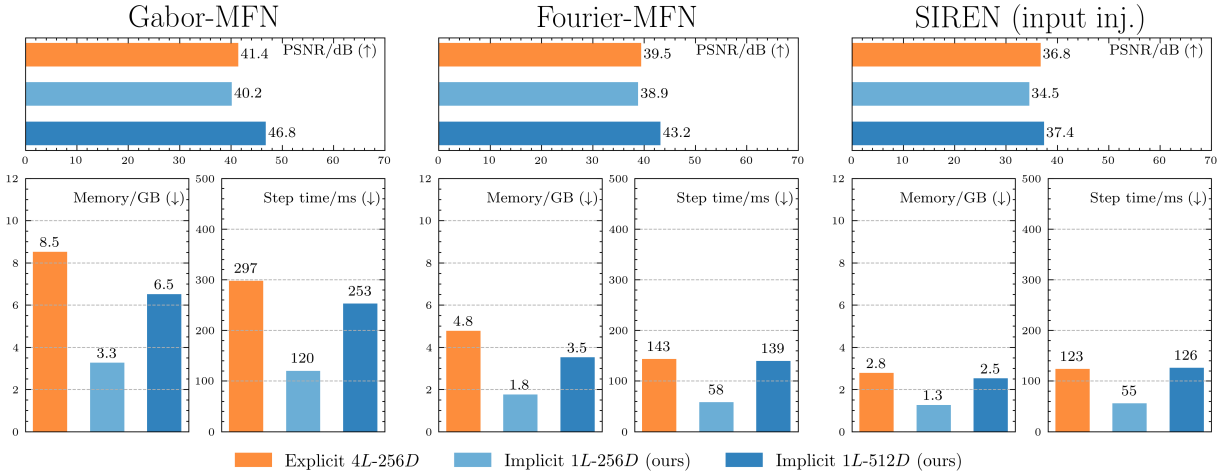


Figure 8.5: Comparison of step time, memory consumption and PSNR between explicit and implicit models trained on the  $512 \times 512$  grayscale image.

time and memory footprint. When given a higher parameter budget, an implicit network with a similar model size (*i.e.*, 1L-512D) outperforms the best explicit network by a large margin.

We also demonstrate the effect of fixed-point reuse and truncated backward gradient using a 1L-512D Fourier MFN on fitting the same  $512 \times 512$  image, with results shown in Fig. 8.3 and 8.4. Specifically, by reusing fixed-points in implicit layers, the solver takes significantly fewer steps to converge for the majority of the training period, making the forward evaluation of implicit layers drastically more efficient. Meanwhile, Fig. 8.4 shows that the 0th-order gradient approximation in [81] (*i.e.*  $T = 0$ ), albeit efficient, could substantially hurt model performance. In contrast, Neumann-based phantom gradient [87] with  $T = 1$  greatly reduces the performance gap with only one additional vector-Jacobian product evaluation, thus achieving a better balance between model efficiency and effectiveness.

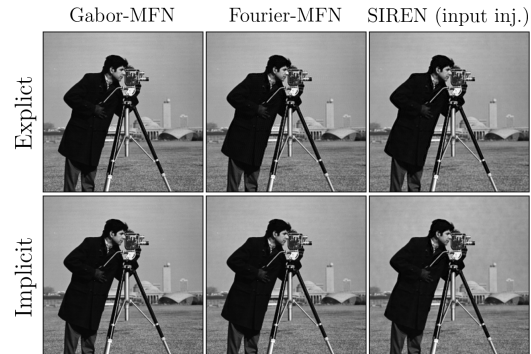


Figure 8.6: Learned  $512 \times 512$  grayscale image

### 8.3.2 Image Generalization

In a number of applications [26, 47, 169, 177], implicit representations are used to infer unobserved parts of data. In order to demonstrate the ability for (Implicit)<sup>2</sup> networks to generalize well, we train the network on only 25% of the pixels from each image in the *Natural* and *Text* dataset, following [75], and evaluate PSNR on an unobserved 25% portion of the image. The average PSNR over all 16 high-resolution images in *Natural* and *Text* is reported in Table 8.1, and a visualization of the improvement by (Implicit)<sup>2</sup> models is shown in Fig. 8.7.

It can be seen that (Implicit)<sup>2</sup> networks improve substantially over the explicit baselines,

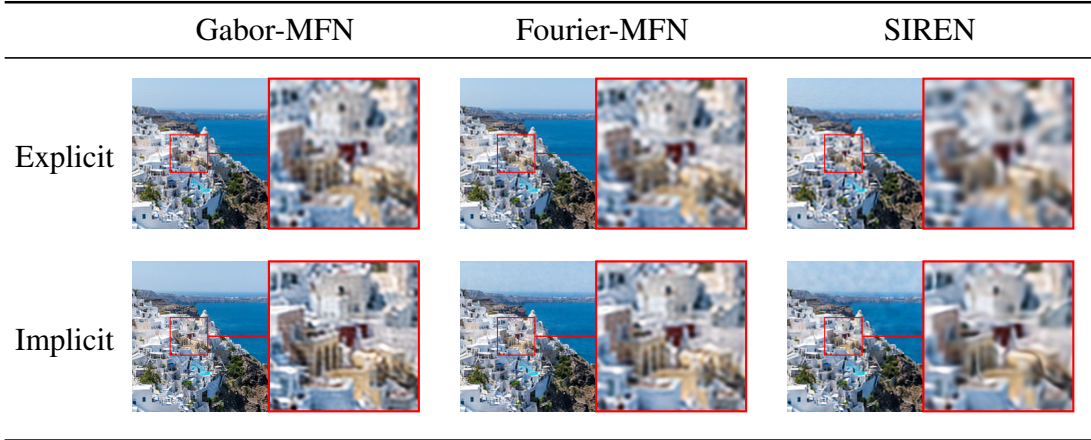


Figure 8.7: Samples of best performing explicit/implicit models learned on *Natural*

Table 8.1: PSNR (in dB) for all models on image generalization. The reported mean  $\pm$  std is taken over the individual PSNR of the 16 images.

	<i>Natural</i>			<i>Text</i>		
	1L-256D	1L-512D	4L-256D	1L-256D	1L-512D	4L-256D
Fourier-MFN	23.27 $\pm$ 3.18	23.30 $\pm$ 3.05	24.57 $\pm$ 3.35	24.64 $\pm$ 2.11	24.84 $\pm$ 2.10	26.67 $\pm$ 2.06
Fourier-iMFN (ours)	24.88 $\pm$ .44	<b>25.19 <math>\pm</math> 3.64</b>	24.52 $\pm$ 3.33	26.90 $\pm$ 2.14	<b>27.19 <math>\pm</math> 1.83</b>	26.48 $\pm$ 2.04
Gabor-MFN	24.16 $\pm$ 3.35	24.68 $\pm$ 3.46	24.65 $\pm$ 3.38	27.19 $\pm$ 2.18	27.74 $\pm$ 2.13	27.57 $\pm$ 2.10
Gabor-iMFN (ours)	24.91 $\pm$ 3.41	<b>25.42 <math>\pm</math> 3.76</b>	24.53 $\pm$ 3.33	27.53 $\pm$ 2.18	<b>28.07 <math>\pm</math> 2.00</b>	27.40 $\pm$ 2.10
SIREN (input inj.)	22.88 $\pm$ 3.0	24.52 $\pm$ 3.28	24.10 $\pm$ 3.34	24.54 $\pm$ 2.19	25.69 $\pm$ 2.18	26.21 $\pm$ 2.19
iSIREN (ours)	24.28 $\pm$ 3.37	<b>24.92 <math>\pm</math> 3.58</b>	24.05 $\pm$ 3.39	26.06 $\pm$ 2.18	<b>26.81 <math>\pm</math> 2.09</b>	26.31 $\pm$ 2.20

where the best-performing implicit model outperforms the explicit counterpart by  $> 0.7$  in PSNR using the same architecture. Visually, (Implicit)<sup>2</sup> networks produce a sharper and shape-consistent representation compared with the best performing explicit network. We observe similar improvements in training speed and memory consumption as in the image representation task (about  $3\times$ ). For completeness, we also evaluate an (Implicit)<sup>2</sup> network where  $F$  also consists of a stack of 4 layers (rather than 1), and observe that the implicit modeling of an already deep structure does not yield much improvement.

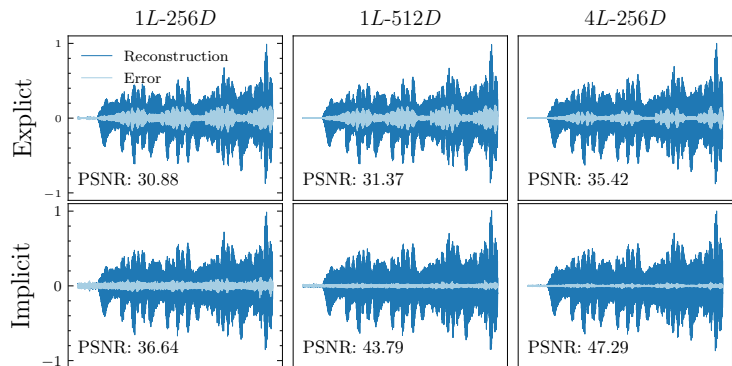


Figure 8.8: Audio signals represented using Fourier-MFNs. Lower error magnitudes (in light blue) are better.

### 8.3.3 Audio Representation

We further show that the proposed (Implicit)<sup>2</sup> network outperforms explicit networks on representing very-high-frequency audio signals. Following [212], we train the models to fit a 7-second music piece. We observe that in all cases, the implicit modeling significantly outperforms the respective explicit models.

In Fig. 8.8, we show the reconstruction results and the respective errors when using Fourier-MFN to fit the audio signal, with more results available in the appendix. Although an (Implicit)<sup>2</sup> model only defines one layer, it is able to outperform explicit models in terms of PSNR by more than 23% (43.79 vs. 35.42), while still retaining the same level of memory and speed improvements as before. We also find that, in this case, having a deeper architecture in an implicit layer (implicit 4L-256D) leads to improved performance.

### 8.3.4 Video Representation

We may add an extra dimension to the input of implicit representation function and try to learn a model for video sequences  $\Phi : (t, x, y) \rightarrow \mathcal{C}$ , where  $t$  is the space of time. We aim to represent a 300-frame  $512 \times 512$  video using each model. The PSNR results are shown in Table 8.2.

Unlike in the image representation setting, we observe that, due to the relatively few parameters compared to the size of the video data (the smallest model only has as many parameters as 1.3% of all pixel values), model sizes have a more significant impact on the overall performance. But still, in most configurations, we found (Implicit)<sup>2</sup> models lead to a consistently non-trivial improvement in performance when compared to their explicit counterparts, while doing so with equivalent or less memory budget.

	1L-1024D	1L-2048D	4L-1024D
Fourier-MFN	24.97 ± 1.08	26.9 ± 0.97	27.64 ± 0.90
Fourier-iMFN (ours)	25.85 ± 1.00	27.7 ± 0.90	<b>28.03 ± 0.95</b>
Gabor-MFN	26.15 ± 1.02	28.18 ± 0.78	<b>29.64 ± 0.81</b>
Gabor-iMFN (ours)	26.45 ± 0.98	28.79 ± 0.77	29.20 ± 1.04
SIREN (input inj.)	25.12 ± 0.96	26.04 ± 0.99	26.52 ± 0.86
iSIREN (ours)	26.03 ± 0.92	27.08 ± 0.93	<b>27.12 ± 0.89</b>

Table 8.2: PSNR for the video representation task. The reported mean ± std is taken over all frames of the video.

### 8.3.5 3D Geometry Representation

To further demonstrate the benefit of (Implicit)<sup>2</sup> networks over their explicit counterparts, we choose three formulations of Fourier-MFN with similar parameter count (i.e. explicit 1L-512D, explicit 4L-256D, and implicit 1L-512D) and fitted them on several 3D object meshes with the point occupancy prediction objective similar to [167] - given input coordinate  $c = (x, y, z)$ , the model  $\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}$  is trained to predict a binary label indicating whether the point corresponding to the coordinate is located inside the target object (0 if the point is outside the object, and 1 if inside). The visualized normal maps of the learned representations on one of the objects are shown in Fig. 8.9. We further evaluated the prediction IoU over test points densely sampled near

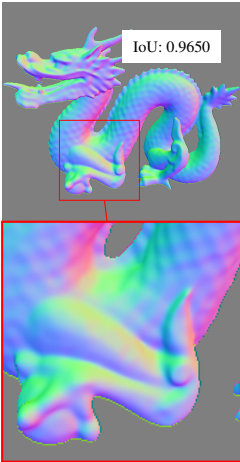
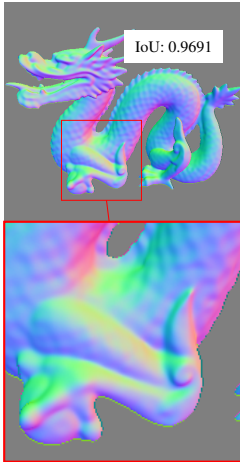
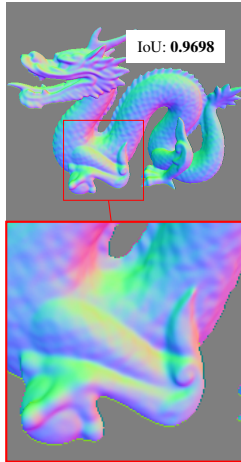
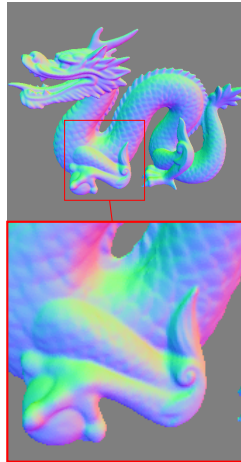
Explicit 1L-512D	Explicit 4L-256D	Implicit 1L-512D	Ground Truth
			

Figure 8.9: Normal maps and IoUs of fitted *dragon* object using Fourier-MFNs in Occupancy Network [167]. The (Implicit)<sup>2</sup> approach performs better than its deep explicit counterpart.

the mesh and present them in the same figure. Experimental details and results on additional 3D models are available in the appendix.

The results show that, with a similar model size, the one-layer network modeled with the (Implicit)<sup>2</sup> formulation is able to capture more details than an explicit model with an identical (1L-512D) or a deeper (4L-256D) structure, and at the same time having a training time and memory advantage similar to the ones in Fig. 8.5. Such an example demonstrates the superior parameter and memory efficiency of (Implicit)<sup>2</sup> networks.

We note that in both the video and 3D geometry tasks, the signal is either of large dimension or defined continuously, where we may not fit the entire data into a single batch. Therefore, the implicit models usually entail more than one step to reach convergence due to the challenge in fixed-point reuse with mini-batch training (see a more detailed analysis in Huang et al. [105]). Nevertheless, our proposed method is still able to boost the training efficiency by leveraging the inexact gradient, while improving upon its explicit counterparts in most cases.

## 8.4 Limitations

The (Implicit)<sup>2</sup> approach can be significantly more time-efficient and memory-friendly than explicit networks at training time; however, at inference time, the implicit models must resort to the normal fixed-point solving. Furthermore, accelerated training in the forward pass by fixed point reuse (see Sec.8.2.2) in practice can be bottlenecked by storage or hardware I/O bandwidth constraints; i.e., it works best when we can fit all or a large portion of the fixed point into memory for quick retrieval. Although such assumption can be easily satisfied for implicit representation learning in many cases, for extremely large data spaces (e.g. Park et al. [183] and Mildenhall et al. [169], where samples are drawn continuously in  $\mathbb{R}^3$ ), the fixed point reuse may not always work

as well and could require more solver steps.

Smarter caching strategies for the fixed points may be developed in the future, where we consider only a subset of the input space for caching throughout the training. For example, one may consider a voxel-grid-based sparse storage for the fixed points, similar to the ones used in Liu et al. [150], such that fixed points of any batch of inputs may be approximately obtained from interpolation of the sparse storage. We leave this direction for future work.

## 8.5 Discussion

In this chapter, we propose (Implicit)<sup>2</sup> networks, which demonstrates yet another example application of equilibrium models—specifically, on implicit neural representation (INR) tasks. We show that these two concepts of “implicitness” complement each other well, allowing us to take advantage of their properties (e.g., fixed point reuse across *training iterations*, in contrast to the reuse across video frames as in Chapter 7) to produce a significantly more efficient training routine and usually more parameter-efficient models for INR learning. We demonstrate through our set of experiments that the *implicit modeling* of *implicit representations* may in many cases be conveniently used as a drop-in replacement for existing state-of-the-art explicit models like SIREN and MFN to further improve the performance and reduce the memory/computation budget required to train these tasks.



**Part IV**  
**Conclusion**



# Chapter 9

## A Different Form of Deep Learning

*Are layers necessary to deep learning?* This was the question that we asked at the beginning of this thesis. For a very long time, it has been assumed that deep learning is powerful because of its deeply stacked architectures. Indeed, when building these modern AI models, there is a set of questions that all model architects must answer; for example: 1) how many layers should there be (e.g., ResNet-50 or -100 [96])? 2) How do we schedule these layers (e.g., U-Net [195] or Pyramid networks [264])? 3) How do we connect different layers (e.g., DenseNet [103] or ResNeXt [248])? The trend of deep learning in the last decade has been to get deeper and deeper, with more and more parameters and increasingly complex structures (e.g., GPT-3 [31]).

But this thesis answers otherwise. While layers have been very successful, we introduce a new arena where we abandon this traditional notion of layer-stacking completely, and model the output of as an implicit, continuous dynamical system at no cost to model capacity (see Fig. 9.1). The goal of this thesis is to re-think, re-identify, and exploit new properties and forms of neural networks that were long buried by the conventional layer-based formulation. We demonstrate a deep (or really, shallow?) equilibrium approach that

- has no layer stacking or computation graph;
- represents an **infinitely deep** conventional neural network, but with only **one layer** modeled implicitly;
- can be directly differentiated through its final output  $\mathbf{z}^*$ , without knowledge of the forward computation (thus completely decoupling the forward and backward passes);
- consumes only  $O(1)$  memory footprint— equivalent to that of just one layer;
- subsumes a wide range of modern, structured and cutting-edge layer designs (e.g., Transformer block);
- can compute adaptively and efficiently in both the forward (e.g., different solver paths, fixed-point reuse, etc.) and in the backward passes (e.g., inexact gradients);
- scales to very high-dimensional space and is able to perform competitively with (or even improves over) large, deeply stacked traditional networks.

We don't believe those layer-less deep learning models will replace the conventional explicit ones, nor is that the main message that this thesis hope to convey. The goal is never to *replace*, but to *complement*: via DEQ models, we hope to demonstrate a **different paradigm** of deep networks

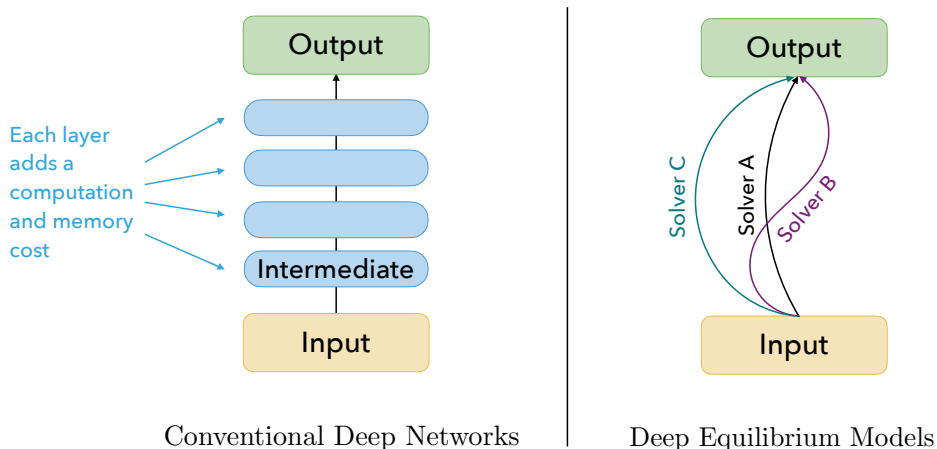


Figure 9.1: A simple comparison between conventional explicit deep networks and the implicit equilibrium approach presented in this thesis. These deep equilibrium (DEQ) models do not define a prescribed computation trajectory, but can instead rely on any black-box solvers and be differentiated using only the final output alone (and are thus extremely memory efficient).

that complements the existing development, and suggests a valuable and fundamental approach to rectify their drawbacks while significantly enhance their efficiency in large-scale, real-world applications.

We next summarize again the in-depth exploration we have had in this thesis.

## 9.1 Summary

This thesis was broken into two parts. In the first part, we present a way for very deep, possible infinite-level, deep networks to be reduced to one level.

- In Chapter 2, we formally introduce the generic formulation of *deep equilibrium (DEQ) models*, which compute the “infinite” limit (i.e., a fixed point) representation of hidden activations. We show that this equilibrium approach is universal in terms of its representational capacity, and how one can directly differentiate through the final equilibrium point, thus enabling DEQ models to consume only the memory equivalent to that of just one layer. We demonstrate the power of this *layer-less* approach with competitive performance on small- and large-scale language modeling tasks.
- In Chapter 3, we further extend the idea of a single “equilibrium” to multiple “equilibria”. While the success of conventional deep networks in pattern recognition tasks (e.g., computer vision) suggests the indispensable role that layers play to represent drastically different resolutions, we demonstrate that such *feature hierarchy* can be modeled even without *architectural hierarchy* of models (i.e., layers). By simultaneously driving a set of representations to a synchronized fixed point, we can effectively model feature abstractions and enable DEQ models to provide natural interfaces that allows it for multi-tasking or compound training. We demonstrate state-of-the-art level performance on realistic image classification and semantic segmentation tasks.

In the second part of this thesis, we build substantially on this formulation, which regards a deep network as a dynamical system rather than a computation graph. We provide a thorough overview of their challenges, present the many new opportunities they bring forth, and leverage these new properties to build better applications in many fronts.

- In Chapter 4, we start by discussing some new challenges that this equilibrium approach poses that were unknown to conventional deep networks— e.g., stability, architectural brittleness, etc. We demonstrate how the equilibrium models “learns to get deeper” (i.e., growing instability), and how these convergences are closely related to the Jacobian conditioning at the fixed point. Then, we outline a new regularization, pursuant to the implicitness of the DEQ models, that stabilize these dynamical systems.
- In Chapter 5, we discuss the implication these one-layer implicit models have on the backward pass gradient generation process. Unlike conventional deep networks which relies on chain-rule backpropagation, we show (both theoretically and empirically), these equilibrium models can be effectively and efficiently trained with much more lightweight and inexact gradients. Using these approximate gradients, the DEQ models’ backward pass could be almost free (e.g., using Jacobian-free methods), while still achieving competitive performances in large-scale experiments.
- In Chapter 6, we bring up the notion of how DEQ models decouple *representational capacity* (which is controlled by  $f_\theta$ ) and *forward computations* (which is determined by the solver). This is in sharp contrast to conventional neural networks, for which the latter is the foundation of the former (i.e., expressivity comes with deep stacking). We show that one can harness such decoupling and parameterize the fixed-point solving process itself, by building *customized*, neural equilibrium solvers. Importantly, these solvers themselves are extremely lightweight and cheap to train, are orthogonal to the parameterization or structure of the single layer  $f_\theta$  itself, and are able to lead to substantial inference-time speedup.
- In Chapter 7 and 8, we show how these equilibrium networks and the toolbox we (and the rest of the community) have built around them can help us build better applications. As two examples, we show that as we do away with layers and abandon the computation graph view of deep learning, these DEQ models can help us build optical flow estimators (Chapter 7) and implicit neural representations (Chapter 8) that are substantially on all three axes: computational cost, memory footprint, and accuracy.

## 9.2 Thoughts and Future Direction

This thesis provides a strong argument that current deep networks have some fundamental ceiling that must be overcome (e.g., computationally), and that deep equilibrium models are frequently better by design— even though they are not outperforming everything yet. The idea presented in this thesis has already led to a new research subarea called **implicit deep learning** in the community, and many of the results presented in the earlier chapters would not be possible without the community wide effort.

We use the rest of this section to discuss some of these community-wide efforts and envision the future research direction of this “newer” deep learning. But before that, we also go down the

memory lane to revisit an interesting “old question”.

### 9.2.1 Are DEQ Models Less “Biological”?

How “biological” are the DEQ models, one might ask, as we replace neuron-stacking with dynamical systems? After all, conventional deep networks were first inspired by biological neurons in the human brain [74, 125, 161] (which started the famous connectionist AI vs. symbolic AI rival), as the neurons constantly propagate signals to the other neurons (i.e., “activations”) and process the received input signals.

But in fact, there is very limited biology, in terms of both mechanism and learning process, of these neural networks. For example, as the Nobel Prize winner Francis Crick famously said in his 1989 paper “The recent excitement about neural networks” [55]:

*“But is this what the brain actually does? Alas, the back-drop nets are unrealistic in almost every respect, as indeed some of their inventors have admitted [...] As far as the learning process is concerned, it is unlikely that the brain actually uses backpropagation.”*

Moreover, he also added that the memory neurons should “have a single layer” that “feeds back on itself” and constantly “adjusting the strengths of all synapses”.

Certainly, the success of modern deep learning itself is a strong argument that deep networks do not need to be like human brain in order to perform well, or even better, such as AlphaGo [209]. But what Francis Crick does point out, is that the current deep learning as we know it (which is based on layers, computation graphs and chain-rule backpropagation) may only be one of the many potential formulations— even just a rudimentary one. Implicit deep learning, exemplified by the deep equilibrium (DEQ) models introduces an alternative formulation, and one that arguably fits better with Crick said.

### 9.2.2 Community-Wide Effort

Since the deep equilibrium models were first introduced in 2019 [18], these pioneering work introduced in this thesis has led to a new and quickly growing community dedicated to study and extend these implicit views of deep learning, as well as a new research subfield called “implicit deep learning”.

Besides the research directions presented in this thesis, for example, the community has studied the DEQ models in numerous contexts, such as in *graph modeling* [92, 138, 149, 184], *gradient dynamics and learning theory* [76, 83, 120], generative modeling [93, 157], approximate gradient [5, 81, 86, 97], provable stability/convergence [141, 180, 192, 193, 229, 244], spiking neural networks [247], out-of-distribution generalization and robustness [142, 192, 240], and many more. Notably, these advancements in continuous and “infinite-level” deep networks (along with the closely related work on Neural ODEs [45]) has led to a NeurIPS 2020 tutorial “[Deep Implicit Layers](#)”. Many ideas of the work presented in this thesis stem from these other explorations.

### 9.2.3 Future

Without a doubt, a lot of questions remain to be answered, and this thesis is likely a just a starting point of these new perspectives on continuous, implicit deep learning approaches. While we have summarized some major challenges and limitations of the DEQ models (e.g., instability, inefficiency (generically), fragility, etc.) in Chapter 4 already, we also identify two major directions where we believe this research could have a huge impact on.

**Implicit Deep Learning and their Practical Applications.** Despite their appealing properties and encouraging results that rival their explicit counterparts, there are still lots of unknowns to these implicit deep algorithms. A very important open problem is, where do we expect to see them completely replace the conventional neural networks; e.g., as a strictly better alternative? A major obstacle here is still the efficiency of fixed-point solving and a lack of more in-depth understanding of how DEQ models' stability relate to their generalization (e.g., what does it mean when the root-solving diverges; what will the computations be like in out-of-distribution (OOD) contexts). We also believe in the importance of gaining a better understanding of how exactly these inexact gradients presented previously facilitate the learning process of implicit models on large-scale practical tasks, and its implication on generalization. This will have a profound impact as we train and deploy these implicit architectures to broader settings, especially without the strong (and often constraining) assumption of a global unique fixed point [192, 244]. Moreover, as these implicit-depth networks model underlying dynamical systems, we expect them to improve over the traditional domains where iterative, optimization-inspired approaches dominate, a domain where we already start to see DEQs play a larger role [21, 229]. Finally, we note that the current computation hardware (e.g., GPUs) are heavily customized toward conventional deep networks to help them execute the computation graphs more efficiently. But as we abandon the notion of layer stacking, it raises a value direction where we could further improve the hardware level integration with AI; e.g., using analogue feedback control circuit, which has a closely connected feedback logic with the DEQ models. We have also already seen some efforts on the software end, such as Google's autodiff frameworks like JAX [29] which makes implicit differentiation significantly easier and more efficient.

**Deep Learning for Scientific Computing.** The research presented in this thesis shows a viable path to make deep learning about solutions to non-linear, high-dimensional and complex dynamical systems. This advancement shed new light on some old questions. Can we leverage these expressive deep learning-based dynamical systems to better learn and capture real-world scientific equilibria, which are often characterized by dynamical systems? For example, recent work on differentiable density functional theory (DFT) found strong connection between the fixed-point implicit networks with the self-consistency iteration to calculate density profile in quantum chemistry [119]. Some work has also explored the connections these equilibrium networks have with the equilibrium in game theory [97]. But more broadly, can we build more scientifically informed neural networks (explicit and implicit ones, or a hybrid) to improve the human knowledge on natural sciences? This direction has shown great prospect as researchers show that DL can significantly accelerate computational fluid dynamics [126], or predict complex protein folding [117]. We believe it is important that we have a more systematic exploration and

understanding of how to design and structure deep learning algorithms for scientific computing, especially with the advent of implicit networks. With these insights, we may substantially advance the frontier of science discovery (especially computationally intractable domains) and their deployment in relevant high-dimensional downstream applications (e.g., drug discovery).



# Bibliography

- [1] Triantafyllos Afouras, Joon Son Chung, and Andrew Zisserman. Deep lip reading: a comparison of models and an online application. *arXiv preprint arXiv:1806.06053*, 2018. [1.2.1](#)
- [2] G Alefeld. On the convergence of halley’s method. *The American Mathematical Monthly*, 88(7):530–536, 1981. [4.2.5](#)
- [3] Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning optical flow from still images. In *Computer Vision and Pattern Recognition (CVPR)*, pages 15201–15211, 2021. [7](#), [7.2.2](#)
- [4] Luis B Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. *Artificial Neural Networks*, 1990. [1.1](#)
- [5] Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *arXiv preprint arXiv:2202.00665*, 2022. [9.2.2](#)
- [6] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, 2017. [1.1](#), [1.1](#)
- [7] Guozhong An. The Effects of Adding Noise During Backpropagation Training on a Generalization Performance. *Neural Computation*, 8(3):643–674, 1996. [5.1](#)
- [8] Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965. [2.2.1](#), [4.2.5](#), [5.3.1](#), [6](#), [6.1](#), [6.2](#), [6.2.2](#), [6.3.1](#), [7](#), [7.3.3](#)
- [9] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Neural Information Processing Systems*, 2016. [6.1](#)
- [10] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016. [2.4.1](#)
- [11] Kendall E Atkinson. *An introduction to numerical analysis*. John wiley & sons, 2008. [4.2.5](#)
- [12] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011. [4.3](#)
- [13] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016. [1](#), [2.3](#), [4](#), [4.2.4](#)
- [14] Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations (ICLR)*, 2019. [4.2.4](#)

- [15] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Convolutional sequence modeling revisited. *International Conference on Learning Representations (ICLR) Workshop*, 2018. 1.2.1
- [16] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271*, 2018. 1.2.1, 2.1, 2.4.1, 2.3
- [17] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *International Conference on Learning Representations (ICLR)*, 2019. 1.2.1, 2, 2.1, 2.1, 2.1, 2.3, 2.3, 2.4, 2.2, 2.4.2, 2.3
- [18] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Neural Information Processing Systems*, 2019. 2, 2.4, 2.4.1, 3.1, 3.2, 3.1, 3.2, 3.3.4, 4.2.2, 4.2.3, 4.2.4, 4.2.4, 4.2.5, 4.1, 5, 5, 5.2, 6.1, 6.1, 6.2.1, 6.3.1, 7, 7.2.2, 8, 8.2.1, 9.2.2
- [19] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Neural Information Processing Systems*, 2020. 3, 3.3, 3.3.5, 4.2, 4.2.3, 4.2.4, 4.2.5, 4.4.3, 4.2, 5, 5, 5.4, 6.1, 6.1, 6.3.1, 7.2.2
- [20] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Stabilizing equilibrium models by jacobian regularizations. In *International Conference on Machine Learning (ICML)*, 2021. 2.4.2, 4, 4.1, 4.2, 4.4, 5.5, 6.2.3, 6.3, 6.3.1, 6.3.1, 6.3.1, 7, 7.2.3, 7.5
- [21] Shaojie Bai, Zhengyang Geng, Yash Savani, and J. Zico Kolter. Deep equilibrium optical flow estimation. In *Computer Vision and Pattern Recognition (CVPR)*, 2022. 7, 7.2.3, 9.2.3
- [22] Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Neural deep equilibrium solvers. In *International Conference on Learning Representations*, 2022. 6, 6.2.2, 6.3
- [23] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922. 8.2.2
- [24] John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994. 7
- [25] Jens Behrmann, David Kristjanson Duvenaud, and Jörn-Henrik Jacobsen. Invertible Residual Networks. In *International Conference on Machine Learning (ICML)*, 2019. 5.1, 8.2.2
- [26] Mojtaba Bermana, Karol Myszkowski, Hans-Peter Seidel, and Tobias Ritschel. X-fields: Implicit neural view-, light- and time-image interpolation. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2020)*, 39(6), 2020. doi: 10.1145/3414685.3417827. 8.1, 8.3.2
- [27] Michael J Black and Padmanabhan Anandan. A framework for the robust estimation of optical flow. In *1993 (4th) International Conference on Computer Vision*, pages 231–236. IEEE, 1993. 7.1
- [28] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2017. 4.1
- [29] James Bradbury, Roy Frostig, Peter Hawkins, Matthew Johnson, et al. JAX: composable transformations of Python+NumPy programs, 2018. URL <http://github.com/google/jax>. 9.2.3

- [30] Lars A Bratholm, Will Gerrard, Brandon Anderson, Shaojie Bai, Sunghwan Choi, Lam Dang, Pavel Hanchar, Addison Howard, Guillaume Huard, Sanghoon Kim, et al. A community-powered search of machine learning strategy space to find nmr property prediction models. *arXiv preprint arXiv:2008.05994*, 2020. [1.2.1](#)
- [31] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4), 1992. [9](#)
- [32] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Neural Information Processing Systems*, 33:1877–1901, 2020. [1](#), [2](#)
- [33] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:500–513, 2011. [7.1](#)
- [34] Charles G Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 1965. [3](#), [2.2.1](#), [2.4](#), [4.2.2](#), [4.2.5](#), [5.3.1](#), [5.4](#), [6](#), [6.1](#), [6.3.1](#), [7](#), [1](#), [7.3.3](#)
- [35] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020. [1.1](#)
- [36] Peter Burt and Edward Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4), 1983. [3](#)
- [37] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *European conference on computer vision*, pages 611–625. Springer, 2012. [7.3](#), [7.3.1](#), [7.3.2](#), [7.3.4](#)
- [38] Miguel Carreira-Perpinan and Weiran Wang. Distributed Optimization of Deeply Nested Systems. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 10–19, 2014. [5.5](#)
- [39] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. *International Conference on Learning Representations (ICLR)*, 2018. [4.1](#)
- [40] Liang-Chieh Chen, Yi Yang, Jiang Wang, Wei Xu, and Alan L. Yuille. Attention to scale: Scale-aware semantic image segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. [3](#)
- [41] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587*, 2017. [3.3](#), [3.3.3](#)
- [42] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Neural Information Processing Systems*, 2018. [3](#)
- [43] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L.

- Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4), 2018. 3, 3.1
- [44] Qifeng Chen and Vladlen Koltun. Full flow: Optical flow estimation by global optimization over regular grids. *Computer Vision and Pattern Recognition (CVPR)*, pages 4706–4714, 2016. 7.1
- [45] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018. 1.1, 2.4.2, 2.4.2, 3.3.1, 5, 6.2.2, 6.3.1, 7, 7.3.3, 8, 9.2.2
- [46] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv:1604.06174*, 2016. 1, 2.2.2, 2.4.2, 6.2.3
- [47] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020. 8.1, 8.3.2
- [48] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning (ICML)*, 2017. 6.1
- [49] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv:1904.10509*, 2019. 2
- [50] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*, 2014. 2.3, 2.1, 7
- [51] Jan Chorowski, Ron J Weiss, Samy Bengio, and Aäron van den Oord. Unsupervised speech representation learning using wavenet autoencoders. *IEEE/ACM transactions on audio, speech, and language processing*, 27(12):2041–2053, 2019. 1.2.1
- [52] Michael Bradley Cline. *Rigid body simulation with contact and constraints*. PhD thesis, University of British Columbia, 2002. 1.1
- [53] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1.1, 3.3, 3.6, 3.3.3, 4.2, 6, 6.3
- [54] Richard W Cottle, Jong-Shi Pang, and Richard E Stone. *The linear complementarity problem*. SIAM, 2009. 1.1
- [55] Francis Crick et al. The recent excitement about neural networks. *Nature*, 337(6203): 129–132, 1989. 9.2.1
- [56] Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding Synthetic Gradients and Decoupled Neural Interfaces. In *International Conference on Machine Learning (ICML)*, pages 904–912, 2017. 5.5
- [57] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In *AAAI*, 2019. 2.1, 2.1

- [58] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019. 2, 2.3, 2.4, 2.4.2, 2.3, 4.2.3, 4.1, 4.4.2, 6.3.1
- [59] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International Conference on Machine Learning (ICML)*, 2017. 2.3, 6.1
- [60] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In *Neural Information Processing Systems*, 2018. 1.1
- [61] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019. 2, 2.1, 2.1, 2.3, 2.3
- [62] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009. 3.3, 3.3.2, 4, 4.4, 6.3
- [63] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019. 2.3
- [64] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2758–2766, 2015. 7, 7.1, 7.2.2, 7.2.3, 7.3.1, 7.3.2, 7.3.3
- [65] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 4.2.4, 4.4.5, 6.2.1
- [66] Harris Drucker and Yann Le Cun. Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, 3(6):991–997, 1992. 4.1
- [67] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In *Neural Information Processing Systems*, 2019. 1.1, 3.3.1, 3.1, 3.3.1, 3.3.4, 4.1, 5, 6.1, 8
- [68] David Duvenaud, J. Zico Kolter, and Matthew Johnson. Deep implicit layers tutorial - neural ODEs, deep equilibrium models, and beyond. *Neural Information Processing Systems Tutorial*, 2020. 1.2, 6.1, 8
- [69] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, and Armin Askari. Implicit deep learning. *arXiv:1908.06315*, 2019. 1.1
- [70] Abdelrahman Eldesokey and Michael Felsberg. Normalized convolution upsampling for refined optical flow estimation. *arXiv preprint arXiv:2102.06979*, 2021. 7
- [71] Gökçen Eraslan, Žiga Avsec, Julien Gagneur, and Fabian J Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389–403, 2019. 1.2.1

- [72] Haw-ren Fang and Yousef Saad. Two classes of multiseccant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009. 5.3.1, 6.1, 6.2
- [73] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 2013. 3, 3.1
- [74] BWAC Farley and W Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, 1954. 9.2.1
- [75] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021. 8, 8.1, 8.1, 8.1, 8.2, 8.2.1, 8.2.1, 8.3, 8.3.2
- [76] Zhili Feng and J Zico Kolter. On the neural tangent kernel of equilibrium models, 2020. URL [https://openreview.net/forum?id=8\\_7yhptEWD](https://openreview.net/forum?id=8_7yhptEWD). 9.2.2
- [77] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ODE. *arXiv:2002.02798*, 2020. 1.1, 4.1, 6.3.1
- [78] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135, 2017. 5.1
- [79] David Fleet and Yair Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006. 7
- [80] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and Reverse Gradient-Based Hyperparameter Optimization. In *International Conference on Machine Learning (ICML)*, pages 1165–1173, 2017. 5
- [81] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Fixed point networks: Implicit depth models with Jacobian-free backprop. *arXiv:2103.12803*, 2021. 4.2, 4.2.1, 4.3, 5, 5.1, 5.3, 5.3.2, 5.4, 6.1, 6.2.3, 7, 7.2.3, 7.2.3, 2, 8.3.1, 9.2.2
- [82] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Neural Information Processing Systems*, 2016. 2.2, 3.2, 4, 4.1
- [83] Tianxiang Gao, Hailiang Liu, Jia Liu, Hridesh Rajan, and Hongyang Gao. A global convergence theory for deep relu implicit networks via over-parameterization. In *International Conference on Learning Representations (ICLR)*, 2022. 9.2.2
- [84] Xavier Gastaldi. Shake-Shake Regularization. *arXiv preprint arXiv:1705.07485*, 2017. 5.1
- [85] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 7, 7.3, 7.3.1, 7.3.3
- [86] Zhengyang Geng, Meng-Hao Guo, Hongxu Chen, Xia Li, Ke Wei, and Zhouchen Lin. Is Attention Better Than Matrix Decomposition? In *International Conference on Learning Representations (ICLR)*, 2021. 5.1, 7, 7.2.3, 9.2.2

- [87] Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34, 2021. 5, 5.2, 5.2, 5.4, 7, 8.3.1
- [88] Davis Gilton, Gregory Ongie, and Rebecca Willett. Deep equilibrium architectures for inverse problems in imaging. *arXiv:2102.07944*, 2021. 6.1
- [89] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 1
- [90] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016. 1.1
- [91] Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations (ICLR)*, 2019. 1.1, 4.1, 4.4.5
- [92] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit Graph Neural Networks. In *Neural Information Processing Systems*, pages 11984–11995, 2020. 5, 5.4, 5.3, 7.2.2, 9.2.2
- [93] Swaminathan Gurumurthy, Shaojie Bai, Zachary Manchester, and J Zico Kolter. Joint inference and input optimization in equilibrium networks. *Neural Information Processing Systems*, 34, 2021. 9.2.2
- [94] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 2017. 2.4.2
- [95] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015. 6.2.3
- [96] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 1.1, 2, 2, 2.1, 3, 3, 3.1, 3.1, 3.2, 3.3.2, 4.1, 4.2, 6, 7.2.2, 8.2, 9
- [97] Howard Heaton, Daniel McKenzie, Qiuwei Li, Samy Wu Fung, Stanley Osher, and Wotao Yin. Learn to predict equilibria via fixed point networks. *arXiv preprint arXiv:2106.00906*, 2021. 9.2.2, 9.2.3
- [98] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015. 6.2.3
- [99] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997. 2.3, 2.1
- [100] Judy Hoffman, Daniel A Roberts, and Sho Yaida. Robust learning with Jacobian regularization. *arXiv:1908.02729*, 2019. 4.1, 4.3
- [101] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981. 7, 7.1, 7.2.2, 7.2.2
- [102] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 1

- [103] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 2.1, 3, 3.2, 4.2, 9
- [104] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018. 3
- [105] Zhichun Huang, Shaojie Bai, and J Zico Kolter. (implicit)<sup>2</sup>: Implicit layers for implicit representations. *Neural Information Processing Systems*, 34, 2021. 8, 8.2.1, 8.3.5
- [106] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 8981–8989, 2018. 7.1, ??
- [107] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. A lightweight optical flow cnn—revisiting data fidelity and regularization. *arXiv preprint arXiv:1903.07414*, 2019. 7, 7.1, ??, ??
- [108] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989. 4.3
- [109] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 2462–2470, 2017. 7.1, ??
- [110] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. 1, 3.2, 3.2, 4.2
- [111] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018. 1.2.1
- [112] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled Neural Interfaces using Synthetic Gradients. In *International Conference on Machine Learning (ICML)*, pages 1627–1635, 2017. 5.1, 5.5
- [113] Shihao Jiang, Dylan Campbell, Yao Lu, Hongdong Li, and Richard Hartley. Learning to estimate hidden motions with global motion aggregation. In *International Conference on Computer Vision (ICCV)*, pages 9772–9781, October 2021. 7, 7, 7.1, 7.2.2, 7.2.2, 7.2.2, 7.2.2, ??, 7.3.1, 7.4
- [114] Shihao Jiang, Yao Lu, Hongdong Li, and Richard Hartley. Learning optical flow from a few matches. In *Computer Vision and Pattern Recognition (CVPR)*, pages 16592–16600, 2021. 7.1
- [115] Shihao Jiang, Yao Lu, Hongdong Li, and Richard I. Hartley. Learning optical flow from a few matches. *Computer Vision and Pattern Recognition (CVPR)*, pages 16587–16595, 2021. 7
- [116] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R



- Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Neural Information Processing Systems*, 2016. 1.1
- [117] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873): 583–589, 2021. 1, 9.2.3
- [118] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Computer Vision and Pattern Recognition (CVPR)*, pages 8110–8119, 2020. 1
- [119] Muhammad F Kasim and Sam M Vinko. Learning the exchange-correlation functional from nature with fully differentiable density functional theory. *arXiv preprint arXiv:2102.04229*, 2021. 9.2.3
- [120] Kenji Kawaguchi. On the theory of implicit deep learning: Global convergence with implicit layers. In *International Conference on Learning Representations (ICLR)*, 2021. 4.2.1, 9.2.2
- [121] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. In *Neural Information Processing Systems*, 2020. 1.1, 4.1
- [122] Patrick Kidger, Ricky TQ Chen, and Terry Lyons. “hey, that’s not an ODE”: Faster ODE adjoints with 12 lines of code. In *International Conference on Machine Learning (ICML)*, 2021. 6.1
- [123] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Deeply-recursive convolutional network for image super-resolution. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1637–1645, 2016. 2.1
- [124] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1
- [125] Stephen C Kleene et al. Representation of events in nerve nets and finite automata. *Automata studies*, 34:3–41, 1956. 9.2.1
- [126] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021. 9.2.3
- [127] Daniel Kondermann, Rahul Nair, Katrin Honauer, Karsten Krispin, Jonas Andrusis, Alexander Brock, Burkhard Gusefeld, Mohsen Rahimimoghaddam, Sabine Hofmann, Claus Brenner, et al. The hci benchmark suite: Stereo and flow ground truth with uncertainties for urban autonomous driving. *CVPR Workshop*, 2016. 7.3.1
- [128] Steven G Krantz and Harold R Parks. *The implicit function theorem: History, theory, and applications*. Springer, 2012. 2, 2, 5, 8
- [129] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 3.3
- [130] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with

- deep convolutional neural networks. In *Neural Information Processing Systems*, 2012. 2, 3, 3, 3.2
- [131] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. 2.1
- [132] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Kording. Learning to solve the credit assignment problem. In *International Conference on Learning Representations (ICLR)*, 2020. 5.5
- [133] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988. 1.1
- [134] Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 1989. 3, 3.3
- [135] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *AISTATS*, 2015. 3.2, 7, 7.2.3
- [136] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning (ICML)*, 2009. 1, 2.5, 3, 3.4
- [137] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Neural Information Processing Systems*, 2009. 3
- [138] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pages 6437–6449. PMLR, 2021. 9.2.2
- [139] Ke Li and Jitendra Malik. Learning to optimize. *arXiv:1606.01885*, 2016. 6.1
- [140] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv:1703.00441*, 2017. 6.1
- [141] Mingjie Li, Yisen Wang, Xingyu Xie, and Zhouchen Lin. Optimization inspired multi-branch equilibrium models. In *International Conference on Learning Representations (ICLR)*, 2022. 9.2.2
- [142] Kaiqu Liang, Cem Anil, Yuhuai Wu, and Roger Grosse. Out-of-distribution generalization with deep equilibrium models. In *International Conference on Machine Learning (ICML) 2021 Workshop on Uncertainty and Robustness in Deep Learning*, 2021. 9.2.2
- [143] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3367–3375, 2015. 2.1
- [144] Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning (ICML)*, 2018. 1.1
- [145] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J.

- Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 3
- [146] Drew Linsley, Alekh Karkada Ashok, Lakshmi Narasimhan Govindarajan, Rex Liu, and Thomas Serre. Stable and expressive recurrent vision models. *arXiv:2005.11362*, 2020. 4.1
- [147] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3), 1989. 3.1
- [148] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 2.2
- [149] Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. EIGNN: Efficient infinite-depth graph neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Neural Information Processing Systems*, 2021. 7.2.2, 9.2.2
- [150] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020. 8.4
- [151] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv:2004.08249*, 2020. 4.2.4
- [152] Yifan Liu, Changyong Shu, Jingdong Wang, and Chunhua Shen. Structured knowledge distillation for dense prediction. In *Computer Vision and Pattern Recognition (CVPR)*, 2019. 3.3
- [153] Yuzhou Liu and DeLiang Wang. Divide and conquer: A deep casa approach to talker-independent monaural speaker separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(12):2092–2102, 2019. 1.2.1
- [154] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Computer Vision and Pattern Recognition (CVPR)*, pages 10012–10022, 2021. 4.4.5
- [155] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing Millions of Hyperparameters by Implicit Differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1540–1552, 2020. 5
- [156] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017. 3.3
- [157] Cheng Lu, Jianfei Chen, Chongxuan Li, Qiuhan Wang, and Jun Zhu. Implicit normalizing flows. In *International Conference on Learning Representations (ICLR)*, 2021. 5, 6.1, 6.1, 9.2.2
- [158] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. Vancouver, 1981. 7
- [159] Yi Luo and Nima Mesgarani. Conv-tasnet: Surpassing ideal time–frequency magnitude masking for speech separation. *IEEE/ACM transactions on audio, speech, and language processing*, 27(8):1256–1266, 2019. 1.2.1
- [160] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Doso-

- vitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4040–4048, 2016. [7.3.1](#), [7.3.2](#)
- [161] James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel Distributed Processing, Volume 2: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, volume 2. MIT press, 1987. [9.2.1](#)
- [162] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations (ICLR)*, 2018. [2.2](#)
- [163] Moritz Menze, Christian Heipke, and Andreas Geiger. Discrete optimization for optical flow. In *GCPR*, 2015. [7.1](#)
- [164] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017. [2.4](#), [4](#), [4.2.3](#), [4.4](#), [5](#), [5.4](#), [6.3](#)
- [165] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arXiv:1803.08240*, 2018. [2.3](#)
- [166] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations (ICLR)*, 2018. [2.2](#)
- [167] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. [8.3.5](#), [8.9](#)
- [168] Raphael A Meyer, Cameron Musco, Christopher Musco, and David P Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 142–155. SIAM, 2021. [4.3](#)
- [169] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, pages 405–421. Springer, 2020. [8](#), [8.1](#), [8.3.2](#), [8.4](#)
- [170] RV Mises and Hilda Pollaczek-Geiringer. Praktische verfahren der gleichungsauflösung. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 9(1):58–77, 1929. [4.3](#), [4.4.4](#)
- [171] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. [4.1](#), [4.3](#), [8.2.2](#)
- [172] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR)*, 2018. [5.4](#)
- [173] Abdualloh Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. Social-

- stgcn: A social spatio-temporal graph convolutional neural network for human trajectory prediction. In *Computer Vision and Pattern Recognition (CVPR)*, pages 14424–14432, 2020. [1.2.1](#)
- [174] Michael C Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3(4):349–381, 1989. [6.2.1](#)
- [175] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010. [1](#)
- [176] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision (ECCV)*, 2016. [3](#)
- [177] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Computer Vision and Pattern Recognition (CVPR)*, pages 3504–3515, 2020. [8.3.2](#)
- [178] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv:1802.08760*, 2018. [4.1](#)
- [179] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2(11):e7, 2017. [2.5](#)
- [180] Chirag Pabbaraju, Ezra Winston, and J. Zico Kolter. Estimating lipschitz constants of monotone deep equilibrium models. In *International Conference on Learning Representations (ICLR)*, 2021. [5.1](#), [9.2.2](#)
- [181] Avik Pal, Alan Edelman, and Christopher Rackauckas. Mixing implicit and explicit deep learning with skip deqs and infinite time neural odes (continuous deqs). *arXiv preprint arXiv:2201.12240*, 2022. [4.2](#)
- [182] Ashutosh Pandey and DeLiang Wang. Tcnn: Temporal convolutional neural network for real-time speech enhancement in the time domain. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6875–6879. IEEE, 2019. [1.2.1](#)
- [183] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Computer Vision and Pattern Recognition (CVPR)*, June 2019. [8](#), [8.1](#), [8.4](#)
- [184] Junyoung Park, Jinhyun Choo, and Jinkyoo Park. Convergent graph solvers. In *International Conference on Learning Representations (ICLR)*, 2022. [4.2.1](#), [4.3](#), [7.2.2](#), [9.2.2](#)
- [185] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019. [2.2.2](#), [5.2](#)
- [186] Fernando J Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural Information Processing Systems*, 1988. [1.1](#)
- [187] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *arXiv:2007.09601*, 2020. [6.1](#), [6.2.2](#)

- [188] Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *International Conference on Machine Learning (ICML)*, 2020. 1.1
- [189] Zaccharie Ramzi, Florian Mannel, Shaojie Bai, Jean-Luc Starck, Philippe Ciuciu, and Thomas Moreau. Shine: Sharing the inverse estimate from the forward pass for bi-level optimization and implicit models. In *International Conference on Learning Representations (ICLR)*, 2022. 5, 5.3, 5.3.1, 5.3.1, 5.3.1, 7, 7.2.3
- [190] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In *Computer Vision and Pattern Recognition (CVPR)*, pages 4161–4170, 2017. 7.1
- [191] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2016. 6.1
- [192] Max Revay, Ruigang Wang, and Ian R Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020. 4, 4.2.1, 4.3, 5.1, 6.1, 6.1, 6.2.3, 8.2.1, 9.2.2, 9.2.3
- [193] Max Revay, Ruigang Wang, and Ian R Manchester. Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness. *arXiv preprint arXiv:2104.05942*, 2021. 9.2.2
- [194] AJ Robinson and Frank Fallside. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering Cambridge, MA, 1987. 6.2.1
- [195] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 1, 3, 9
- [196] Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 15(5):1187–1212, 2015. 4.3
- [197] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 1
- [198] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet: Large Scale Visual Recognition Challenge. *International Journal on Computer Vision (IJCV)*, 115(3):211–252, 2015. 5, 5.4
- [199] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Neural Information Processing Systems*, 2016. 4.1, 8.2.2
- [200] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Computer Vision and Pattern Recognition (CVPR)*, 2018. 3.3, 3.3.3
- [201] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Neural Information Processing Systems*, 2018. 2.3, 4.1
- [202] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele

- Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20 (1):61–80, 2008. 1
- [203] Rajat Sen, Hsiang-Fu Yu, and Inderjit Dhillon. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *arXiv preprint arXiv:1905.03806*, 2019. 1.2.1
- [204] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated Back-propagation for Bilevel Optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1723–1732, 2019. 5.1
- [205] BAI Shaojie, Jeremy Zieg Kolter, Mordechai Kornbluth, Jonathan Mailoa, and Devin Willmott. Graph transformer neural network force field for prediction of atomic forces and energies in molecular dynamic simulations, March 18 2021. US Patent App. 16/569,308. 1.2.1
- [206] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4), 2017. 3
- [207] Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 1950. 2.2.1
- [208] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv:1909.08053*, 2019. 4.1
- [209] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017. 9.2.1
- [210] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 3, 3
- [211] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Neural Information Processing Systems*, 2019. 8.1
- [212] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Neural Information Processing Systems*, 2020. 8, 8, 8.1, 8.1, 8.2, 8.3, 8.3.3
- [213] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017. 4.1
- [214] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 15, 2014. 1, 3.2
- [215] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow

- using pyramid, warping, and cost volume. In *Computer Vision and Pattern Recognition (CVPR)*, pages 8934–8943, 2018. 7.1, ??
- [216] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Models matter, so does training: An empirical study of cnns for optical flow estimation. *arXiv preprint arXiv:1809.05571*, 2018. 7.1, ??
- [217] Deqing Sun, Daniel Vlasic, Charles Herrmann, V. Jampani, Michael Krainin, Huiwen Chang, Ramin Zabih, William T. Freeman, and Ce Liu. Autoflow: Learning a better training set for optical flow. *Computer Vision and Pattern Recognition (CVPR)*, pages 10088–10097, 2021. 7, 7, 7.1
- [218] Shuyang Sun, Jiangmiao Pang, Jianping Shi, Shuai Yi, and Wanli Ouyang. FishNet: A versatile backbone for image, region, and pixel level prediction. In *Neural Information Processing Systems*, 2018. 3
- [219] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 3, 3
- [220] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 1
- [221] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3147–3155, 2017. 2.1
- [222] Towaki Takikawa, David Acuna, Varun Jampani, and Sanja Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In *International Conference on Computer Vision (ICCV)*, 2019. 3.3
- [223] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Neural Information Processing Systems*, 2020. 8, 8.1
- [224] Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training Neural Networks Without Gradients: A Scalable ADMM Approach. In *International Conference on Machine Learning (ICML)*, pages 2722–2731, 2016. 5.5
- [225] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision (ECCV)*, pages 402–419. Springer, 2020. 7, 7, 7.2.1, 7.2.2, 7.2.2, 7.2.2, 7.2.2, 7.2.3, 7.2.3, ??, ??, 7.3, 7.3.1, 7.3.2, 7.4
- [226] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020. 8.1
- [227] Yao-Hung Hubert Tsai, Shaojie Bai, Paul Pu Liang, J. Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov. Multimodal transformer for unaligned multimodal language



- sequences. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2019, page 6558. NIH Public Access, 2019. 1.2.1
- [228] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: A unified understanding of transformer’s attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019. 1.2.1
- [229] Russell Tsuchida, Suk Yee Yong, Mohammad Ali Armin, Lars Petersson, and Cheng Soon Ong. Declarative nets that are equilibrium models. In *International Conference on Learning Representations*, 2022. 4, 4.2, 9.2.2, 9.2.3
- [230] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of  $\text{tr}(f(a))$  via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017. 4.3
- [231] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016. 6.2.1
- [232] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, Francois Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014. 8.3.1
- [233] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. 1, 2, 2.3, 2.3, 3.2, 4.2.4, 4.4.2, 6.3.1, 7.2.2, 7.2.2
- [234] Shobha Venkataraman and Brandon Amos. Neural fixed-point acceleration for convex optimization. *arXiv preprint arXiv:2107.10254*, 2021. 6.1
- [235] Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011. 6.2
- [236] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao. Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 3, 3.3, 3.2, 3.3, 3.3.3
- [237] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning (ICML)*, 2019. 1.1
- [238] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 1
- [239] Andreas Wedel, Thomas Pock, Christopher Zach, Horst Bischof, and Daniel Cremers. An improved algorithm for tv-l 1 optical flow. In *Statistical and Geometrical Approaches to Visual Motion Analysis*, 2008. 7.1
- [240] Colin Wei and J. Zico Kolter. Certified robustness for deep equilibrium models via interval bound propagation. In *International Conference on Learning Representations*, 2022. 4.2, 6.1, 9.2.2

- [241] Paul J Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10), 1990. 6.2.3, 7
- [242] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning (ICML)*, 2017. 6.1
- [243] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: Deep generation of financial time series. *Quantitative Finance*, 20(9):1419–1440, 2020. 1.2.1
- [244] Ezra Winston and J. Zico Kolter. Monotone operator equilibrium networks. In *Neural Information Processing Systems*, 2020. 2.2.1, 4, 4.2.1, 4.2.2, 4.3, 4.2, 5, 6.1, 6.1, 6.2.3, 6.3.1, 7, 8, 8.2.1, 9.2.2, 9.2.3
- [245] Jingfeng Wu, Wenqing Hu, Haoyi Xiong, Jun Huan, Vladimir Braverman, and Zhanxing Zhu. On the Noisy Gradient Descent that Generalizes as SGD. In *International Conference on Machine Learning (ICML)*, pages 10367–10376, 2020. 5.1
- [246] Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision (ECCV)*, 2018. 1, 3.2, 4.1, 4.2.4
- [247] Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Yisen Wang, and Zhouchen Lin. Training feedback spiking neural networks by implicit differentiation on the equilibrium state. *Neural Information Processing Systems*, 34, 2021. 9.2.2
- [248] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves ImageNet classification. In *Computer Vision and Pattern Recognition (CVPR)*, 2020. 9
- [249] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *International Conference on Computer Vision (ICCV)*, 2015. 7, 7.2.3
- [250] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 1
- [251] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tiejian Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*, pages 10524–10533, 2020. 4.2.4
- [252] Jia Xu, René Ranftl, and Vladlen Koltun. Accurate optical flow via direct cost volume processing. *Computer Vision and Pattern Recognition (CVPR)*, pages 5807–5815, 2017. 7.1
- [253] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015. 5, 5.3
- [254] Gengshan Yang and Deva Ramanan. Volumetric correspondence networks for optical flow. In *Advances in Neural Information Processing Systems*, pages 793–803, 2019. 7.1, ??, ??
- [255] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenberg, and Jure Leskovec. Hierarchical temporal convolutional networks for dynamic recommender

- systems. In *The world wide web conference*, pages 2236–2246, 2019. 1.2.1
- [256] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations (ICLR)*, 2016. 3
- [257] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 3.3
- [258] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l1 optical flow. In *DAGM-Symposium*, 2007. 7.1
- [259] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv:1605.07146*, 2016. 3.2
- [260] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan R Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. In *Neural Information Processing Systems*, 2016. 2.4.1
- [261] Ziming Zhang and Matthew Brand. Convergent Block Coordinate Descent for Training Tikhonov Regularized Deep Neural Networks. In *Neural Information Processing Systems*, 2017. 5.5
- [262] Ziming Zhang, Yuting Chen, and Venkatesh Saligrama. Efficient Training of Very Deep Neural Networks for Supervised Hashing. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1487–1495, 2016. 5.5
- [263] Ziming Zhang, Anil Kag, Alan Sullivan, and Venkatesh Saligrama. Equilibrated recurrent neural network: Neuronal time-delayed self-feedback improves accuracy and stability. *arXiv:1903.00755*, 2019. 1.1
- [264] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 3, 3.3, 3.3, 3.3.3, 9
- [265] Hengshuang Zhao, Yi Zhang, Shu Liu, Jianping Shi, Chen Change Loy, Dahua Lin, and Jiaya Jia. PSANet: Point-wise spatial attention network for scene parsing. In *European Conference on Computer Vision (ECCV)*, 2018. 3.3, 3.3.3
- [266] Shengyu Zhao, Yilun Sheng, Yue Dong, Eric I Chang, Yan Xu, et al. Maskflownet: Asymmetric feature matching with learnable occlusion mask. In *Computer Vision and Pattern Recognition (CVPR)*, pages 6278–6287, 2020. 7, ??, ??
- [267] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. UNet++: A nested U-Net architecture for medical image segmentation. In *MICCAI*, 2018. 3.3
- [268] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The Anisotropic Noise in Stochastic Gradient Descent: Its Behavior of Escaping from Sharp Minima and Regularization Effects. In *International Conference on Machine Learning (ICML)*, pages 7654–7663, 2019. 5.1
- [269] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 2.2