Monotone operator equilibrium networks

Ezra Winston
School of Computer Science
Carnegie Mellon University
Pittsburgh, United States
ewinston@cs.cmu.edu

J. Zico Kolter
School of Computer Science
Carnegie Mellon University
& Bosch Center for AI
Pittsburgh, United States
zkolter@cs.cmu.edu

Abstract

Implicit-depth models such as Deep Equilibrium Networks have recently been shown to match or exceed the performance of traditional deep networks while being much more memory efficient. However, these models suffer from unstable convergence to a solution and lack guarantees that a solution exists. On the other hand, Neural ODEs, another class of implicit-depth models, do guarantee existence of a unique solution but perform poorly compared with traditional networks. In this paper, we develop a new class of implicit-depth model based on the theory of monotone operators, the Monotone Operator Equilibrium Network (monDEQ). We show the close connection between finding the equilibrium point of an implicit network and solving a form of monotone operator splitting problem, which admits efficient solvers with guaranteed, stable convergence. We then develop a parameterization of the network which ensures that all operators remain monotone, which guarantees the existence of a unique equilibrium point. Finally, we show how to instantiate several versions of these models, and implement the resulting iterative solvers, for structured linear operators such as multi-scale convolutions. The resulting models vastly outperform the Neural ODE-based models while also being more computationally efficient. Code is available at http://github.com/locuslab/monotone_op_net.

1 Introduction

Recent work in deep learning has demonstrated the power of *implicit-depth* networks, models where features are created not by explicitly iterating some number of nonlinear layers, but by finding a solution to some implicitly defined equation. Instances of such models include the Neural ODE [8], which computes hidden layers as the solution to a continuous-time dynamical system, and the Deep Equilibrium (DEQ) Model [5], which finds a fixed point of a nonlinear dynamical system corresponding to an effectively infinite-depth weight-tied network. These models, which trace back to some of the original work on recurrent backpropagation [2, 23], have recently regained attention since they have been shown to match or even exceed to performance of traditional deep networks in domains such as sequence modeling [5]. At the same time, these models show drastically improved memory efficiency over traditional networks since backpropagation is typically done analytically using the implicit function theorem, without needing to store the intermediate hidden layers.

However, implict-depth models that perform well require extensive tuning in order to achieve stable convergence to a solution. Obtaining convergence in DEQs requires careful initialization and regularization, which has proven difficult in practice [21]. Moreover, solutions to these models are not guaranteed to exist or be unique, making the output of the models potentially ill-defined. While Neural ODEs [8] do guarantee existence of a unique solution, training remains unstable since the ODE problems can become severely ill-posed [10]. Augmented Neural ODEs [10] improve the stability of Neural ODEs by learning ODEs with simpler flows, but neither model achieves efficient

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

convergence nor performs well on standard benchmarks. Crucial questions remain about how models can have guaranteed, unique solutions, and what algorithms are most efficient at finding them.

In this paper, we present a new class of implicit-depth equilibrium model, the Monotone Operator Equilibrium Network (monDEQ), which guarantees stable convergence to a unique fixed point.[1] The model is based upon the theory of monotone operators [6, 26], and illustrates a close connection between simple fixed-point iteration in weight-tied networks and the solution to a particular form of monotone operator splitting problem. Using this connection, this paper lays the theoretical and practical foundations for such networks. We show how to parameterize networks in a manner that ensures all operators remain monotone, which establishes the existence and uniqueness of the equilibrium point. We show how to backpropagate through such networks using the implicit function theorem; this leads to a corresponding (linear) operator splitting problem for the backward pass, which also is guaranteed to have a unique solution. We then adapt traditional operator splitting methods, such as forward-backward splitting or Peaceman-Rachford splitting, to naturally derive algorithms for efficiently computing these equilibrium points.

Finally, we demonstrate how to practically implement such models and operator splitting methods, in the cases of typical feedforward, fully convolutional, and multi-scale convolutional networks. For convolutional networks, the most efficient fixed-point solution methods require an inversion of the associated linear operator, and we illustrate how to achieve this using the fast Fourier transform. The resulting networks show strong performance on several benchmark tasks, vastly improving upon the accuracy and efficiency of Neural ODEs-based models, the other implicit-depth models where solutions are guaranteed to exist and be unique.

2 Related work

Implicit models in deep learning There has been a growing interest in recent years in implicit layers in deep learning. Instead of specifying the explicit computation to perform, a layer specifies some *condition* that should hold at the solution to the layer, such as a nonlinear equality, or a differential equation solution. Using the implicit function theorem allows for backpropagating through the layer solutions *analytically*, making these layers very memory efficient, as they do not need to maintain intermediate iterations of the solution procedure. Recent examples include layers that compute inference in graphical models [15], solve optimization problems [12, 3, 13, 1], execute model-based control policies [4], solve two-player games [20], solve gradient-based optimization for meta-learning [24], and many others.

Stability of fixed-point models The issue of model stability has in fact been at the heart of much work in fixed-point models. The original work on attractor-style recurrent models, trained via recurrent backpropagation [2, 23], precisely attempted to ensure that the forward iteration procedure was stable. And indeed, much of the work in recurrent architectures such as LSTMs has focused on these issues of stability [14]. Recent work has revisited recurrent backpropagation in a similar manner to DEQs, with the similar aim of speeding up the computation of fixed points [19]. And other work has looked at the stability of implicit models [11], with an emphasis on guaranteeing the existence of fixed points, but focused on alternative stability conditions, and considered only relatively small-scale experiments. Other recent work has looked to use control-theoretic methods to ensure the stability of implicit models, [25], though again they consider only small-scale evaluations.

Monotone operators in deep learning Although most work in the field of monotone operators is concerned with general convex analysis, the recent work of [9] does also highlight connections between deep networks and monotone operator problems. Unlike our current work, however, that work focused largely on the fact that many common non-linearities can be expressed via proximal operators, and analyzed traditional networks under the assumptions that certain of the operators were monotone, but did not address conditions for the networks to be monotone or algorithms for solving or backpropagating through the networks.

3 A monotone operator view of fixed-point networks

This section lays out our main methodological and theoretical contribution, a class of equilibrium networks based upon monotone operators. We begin with some preliminaries, then highlight the

[1] We largely use the terms "fixed point" and "equilibrium point" interchangably in this work, using fixed point in the context of an iterative procedure, and equilibrium point to refer more broadly to the point itself.

basic connection between the fixed point of an "infinite-depth" network and an associated operator splitting problem; next, we propose a parameterization that guarantees the associated operators to be maximal monotone; finally, we show how to use operator splitting methods to both compute the fixed point and backpropagate through the fixed point efficiently.

3.1 Preliminaries

Monotone operator theory The theory of monotone operators plays a foundational role in convex analysis and optimization. Monotone operators are a natural generalization of monotone functions, which can be used to assess the convergence properties of many forms of iterative fixed-point algorithms. We emphasize that the majority of the work in this paper relies on well-known properties about monotone operators, and we refer to standard references on the topic including [6] and a less formal survey by [26]; we do include a brief recap of the definitions and results we require in Appendix A. Formally, an operator is a subset of the space $F \subseteq \mathbb{R}^n \times \mathbb{R}^n$; in our setting this will usually correspond to set-valued or single-valued function. Operator splitting approaches refer to methods for finding a zero in a sum of operators, i.e., find x such that $0 \in (F + G)(x)$. There are many such methods, but the two we will use mainly in this work are *forward-backward* splitting (eqn. A9 in the Appendix) and *Peaceman-Rachford* splitting (eqn. A10). As we will see, both finding a network equilibrium point and backpropagating through it can be formulated as operator splitting problems, and different operator splitting methods will lead to different approaches in their application to our subsequent implicit networks.

Deep equilibrium models The monDEQ architecture is closely relate to the DEQ model, which parameterizes a "weight-tied, input-injected" network of the form $z_{i+1} = g(z_i, x)$, where x denotes the input to the network, injected at each layer; z_i denotes the hidden layer at depth i; and g denotes a nonlinear function which is the same for each layer (hence the network is weight-tied). The key aspect of the DEQ model is that in this weight-tied setting, instead of forward iteration, we can simply use any root-finding approach to find an equilibrium point of such an iteration $z^* = g(z^*, x)$. Assuming the model is stable, this equilibrium point corresponds to an "infinite-depth fixed point" of the layer. The monDEQ architecture can be viewed as an instance of a DEQ model, but one that relies on the theory of monotone operators, and a specific paramterization of the network weights, to guarantee the existence of a unique fixed point for the network. Crucially, however, as is the case for DEQs, naive forward iteration of this model is *not* necessarily stable; we therefore employ operator splitting methods to develop provably (linearly) convergent methods for finding such fixed points.

3.2 Fixed-point networks as operator splitting

As a starting point of our analysis, consider the weight-tied, input-injected network in which $x \in \mathbb{R}^d$ denotes the input, and $z^k \in \mathbb{R}^n$ denotes the hidden units at layer k, given by the iteration[2]

$$z^{k+1} = \sigma(Wz^k + Ux + b) \tag{1}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinearity applied elementwise, $W \in \mathbb{R}^{n \times n}$ are the hidden unit weights, $U \in \mathbb{R}^{n \times x}$ are the input-injection weights and $b \in \mathbb{R}^n$ is a bias term. An equilibrium point, or fixed point, of this system is some point $z^\star$ which remains constant after an update:

$$z^\star = \sigma(Wz^\star + Ux + b). \tag{2}$$

We begin by observing that it is possible to characterize this equilibrium point exactly as the solution to a certain operator splitting problem, under certain choices of operators and activation σ. This can be formalized in the following theorem, which we prove in Appendix B:

Theorem 1. *Finding a fixed point of the iteration (1) is equivalent to finding a zero of the operator splitting problem* $0 \in (F + G)(z^\star)$ *with the operators*

$$F(z) = (I - W)(z) - (Ux + b), \quad G = \partial f \tag{3}$$

and $\sigma(\cdot) = \text{prox}_f^1(\cdot)$ *for some convex closed proper (CCP) function* f*, where* prox_f^α *denotes the proximal operator*

$$\text{prox}_f^\alpha(x) \equiv \underset{z}{\text{argmin}} \frac{1}{2}\|x - z\|_2^2 + \alpha f(z). \tag{4}$$

[2]This setting can also be seen as corresponding to a recurrent network with identical inputs at each time (indeed, this is the view of so-called attractor networks [23]). However, because in modern usage recurrent networks typically refer to sequential models with *different* inputs at each time, we don't adopt this terminology.

It is also well-established that many common nonlinearities used in deep networks can be represented as proximal operators of CCP functions [7, 9]. For example, the ReLU nonlinearity $\sigma(x) = [x]_+$ is the proximal operator of the indicator of the positive orthant $f(x) = I\{x \geq 0\}$, and tanh, sigmoid, and softplus all have close correspondence with proximal operators of simple expressions [7].

In fact, this method establishes that some seemingly unstable iterations can actually still lead to convergent algorithms. ReLU activations, for instance, have traditionally been avoided in iterative models such as recurrent networks, due to exploding or vanishing gradient problems and nonsmoothness. Yet this iteration shows that (with input injection and the above constraint on W), ReLU operators are perfectly well-suited to these fixed-point iterations.

3.3 Enforcing existence of a unique solution

The above connection is straightforward, but also carries interesting implications for deep learning. Specifically, we can establish the existence and uniqueness of the equilibirum point $z^\star$ via the simple sufficient criterion that $I - W$ is strongly monotone, or in other words[3] $I - W \succeq mI$ for some $m > 0$ (see Appendix A). The constraint is by no means a trivial condition. Although many layers obey this condition under typical initialization schemes, during training it is normal for W to move outside this regime. Thus, the first step of the monDEQ architecture is to parameterize W in such a way that it always satisfies this strong monotonicity constraint.

Proposition 1. *We have $I - W \succeq mI$ if and only if there exist $A, B \in \mathbb{R}^{n\times n}$ such that*

$$W = (1-m)I - A^TA + B - B^T. \tag{5}$$

We therefore propose to simply parameterize W directly in this form, by defining the A and B matrices directly. While this is an overparameterized form for a dense matrix, we could avoid this issue by, e.g. constraining A to be lower triangular (making it the Cholesky factor of A^TA), and by making B strictly upper triangular; in practice, however, simply using general A and B matrices has little impact upon the performance of the method. The parameterization does notably raise additional complications when dealing with convolutional layers, but we defer this discussion to Section 4.2.

3.4 Computing the network fixed point

Given the monDEQ formulation, the first natural question to ask is: how should we compute the equilibrium point $z^\star = \sigma(Wz^\star + Ux + b)$? Crucially, it can be the case that the simple forward iteration of the network equation (1) does *not* converge, i.e., the iteration may be unstable. Fortunately, monotone operator splitting leads to a number of iterative methods for finding these fixed points, which are guaranteed to converge under proper conditions. For example, the forward-backward iteration applied to the monotone operator formulation from Theorem 1 results exactly in a damped version of the forward iteration

$$z^{k+1} = \text{prox}_f^\alpha(z^k - \alpha((I-W)z^k - (Ux+b))) = \text{prox}_f^\alpha((1-\alpha)z^k + \alpha(Wz^k + Ux + b)). \tag{6}$$

This iteration is guaranteed to converge linearly to the fixed point $z^\star$ provided that $\alpha \leq 2m/L^2$, when the operator $I - W$ is Lipschitz and strongly monotone with parameters L (which is simply the operator norm $\|I - W\|_2$) and m [26].

A key advantage of the monDEQ formulation is the flexibility to employ alternative operator splitting methods that converge much more quickly to the equilibrium. One such example is Peaceman-Rachford splitting which, when applied to the formulation from Theorem 1, takes the form

$$\begin{aligned} u^{k+1/2} &= 2z^k - u^k \\ z^{k+1/2} &= (I + \alpha(I-W))^{-1}(u^{k+1/2} - \alpha(Ux+b)) \\ u^{k+1} &= 2z^{k+1/2} - u^{k+1/2} \\ z^{k+1} &= \text{prox}_f^\alpha(u^{k+1}) \end{aligned} \tag{7}$$

where we use the explicit form of the resolvents for the two monotone operators of the model. The advantage of Peaceman-Rachford splitting over forward-backward is two-fold: 1) it typically converges in fewer iterations, which is a key bottleneck for many implicit models; and 2) it converges

[3] For non-symmetric matrices, which of course is typically the case with W, positive definiteness is defined as the positive definiteness of the symmetric component $I - W \succeq mI \Leftrightarrow I - (W + W^T)/2 \succeq mI$.

Algorithm 1 Forward-backward equilibrium solving

$z := 0; \quad \text{err} := 1$
while err $> \epsilon$ **do**
  $z^+ := (1-\alpha)z + \alpha(Wz + Ux + b)$
  $z^+ := \text{prox}_f^\alpha(z^+)$
  $\text{err} := \frac{\|z^+ - z\|_2}{\|z^+\|_2}$
  $z := z^+$
return z

Algorithm 2 Peaceman-Rachford equilibrium solving

$z, u := 0; \quad \text{err} := 1; \quad V := (I + \alpha(I - W))^{-1}$
while err $> \epsilon$ **do**
  $u^{1/2} := 2z - u$
  $z^{1/2} := V(u^{1/2} + \alpha(Ux + b))$
  $u^+ := 2z^{1/2} - u^{1/2}$
  $z^+ := \text{prox}_f^\alpha(u^+)$
  $\text{err} := \frac{\|z^+ - z\|_2}{\|z^+\|_2}$
  $z, u := z^+, u^+$
return z

for any $\alpha > 0$ [26], unlike forward-backward splitting which is dependent on the Lipschitz constant of $I - W$. The disadvantage of Peaceman-Rachford splitting, however, is that it requires an inverse involving the weight matrix W. It is not immediately clear how to apply such methods if the W matrix involves convolutions or multi-layer models; we discuss these points in Section 4.2. A summary of these methods for computation of the forward equilibrium point is given in Algorithms 1 and 2.

3.5 Backprogation through the monotone operator layer

Finally, a key challenge for any implicit model is to determine how to perform backpropagation through the layer. As with most implicit models, a potential benefit of the fixed-point conditions we describe is that, by using the implicit function theorem, it is possible to perform backpropagation without storing the intermediate iterates of the operator splitting algorithm in memory, and instead backpropagating directly through the equilibrium point.

To begin, we present a standard approach to differentiating through the fixed point $z^\star$ using the implicit function theorem. This formulation has some compelling properties for monDEQ, namely the fact that this (sub)gradient will always exist. When training a network via gradient descent, we need to compute the gradients of the loss function

$$\frac{\partial \ell}{\partial (\cdot)} = \frac{\partial \ell}{\partial z^\star} \frac{\partial z^\star}{\partial (\cdot)} \tag{8}$$

where $(\cdot)$ denotes some input to the layer or parameters, i.e. W, x, etc. The challenge here is computing (or left-multiplying by) the Jacobian $\partial z^\star / \partial (\cdot)$, since $z^\star$ is not an explicit function of the inputs. While it would be possible to simply compute gradients through the "unrolled" updates, e.g. $z^{k+1} = \sigma(Wz^k + Ux + b)$ for forward iteration, this would require storing each intermediate state z^k, a potentially memory-intensive operation. Instead, the following theorem gives an explicit formula for the necessary (sub)gradients. We state the theorem more directly in terms of the operators mentioned Theorem 1; that is, we use $\text{prox}_f^1(\cdot)$ in place of $\sigma(\cdot)$.

Theorem 2. *For the equilibrium point $z^\star = \text{prox}_f^1(Wz^\star + Ux + b)$, we have*

$$\frac{\partial \ell}{\partial (\cdot)} = \frac{\partial \ell}{\partial z^\star} (I - JW)^{-1} J \frac{\partial (Wz^\star + Ux + b)}{\partial (\cdot)} \tag{9}$$

where

$$J = \mathsf{D}\, \text{prox}_f^1(Wz^\star + Ux + b) \tag{10}$$

denotes the Clarke generalized Jacobian of the nonlinearity evaluated at the point $Wz^\star + Ux + b$. Furthermore, for the case that $(I - W) \succeq mI$, this derivative always exists.

To apply the theorem in practice to perform reverse-mode differentiation, we need to solve the system

$$(I - JW)^{-T} \left(\frac{\partial \ell}{\partial z^\star} \right)^T. \tag{11}$$

The above system is a linear equation and while it is typically computationally infeasible to compute the inverse $(I - JW)^{-T}$ exactly, we could compute a solution to $(I - JW)^{-T} v$ using, e.g., conjugate gradient methods. However, we present an alternative formulation to computing (11) as the solution to a (linear) monotone operator splitting problem:

Algorithm 3 Forward-backward equilibrium backpropagation

$u := 0; \quad \text{err} := 1; \quad v := \frac{\partial \ell}{\partial z^*}$
while err $> \epsilon$ **do**
  $u^+ := (1-\alpha)u + \alpha W^T u$
  $u_i^+ := \begin{cases} \frac{u_i^+ + \alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} = \infty \end{cases}$
  $\text{err} := \frac{\|u^+ - u\|_2}{\|u^+\|_2}$
  $u := u^+$
return u

Algorithm 4 Peaceman-Rachford equilibrium backpropagation

$z, u := 0; \quad \text{err} := 1; \quad v := \frac{\partial \ell}{\partial z^*}; \quad V := (I+\alpha(I-W))^{-1}$
while err $> \epsilon$ **do**
  $u^{1/2} := 2z - u$
  $z^{1/2} := V^T u^{1/2}$
  $u^+ := 2z^{1/2} - u^{1/2}$
  $z_i^+ := \begin{cases} \frac{u_i^+ + \alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} = \infty \end{cases}$
  $\text{err} := \frac{\|z^+ - z\|_2}{\|z^+\|_2}$
  $z, u := z^+, u^+$
return z

Theorem 3. *Let $z^\star$ be a solution to the monotone operator splitting problem defined in Theorem 1, and define J as in (10). Then for $v \in \mathbb{R}^n$ the solution of the equation*

$$u^\star = (I - JW)^{-T} v \tag{12}$$

is given by

$$u^\star = v + W^T \tilde{u}^\star \tag{13}$$

where $\tilde{u}^\star$ is a zero of the operator splitting problem $0 \in (\tilde{F} + \tilde{G})(u^\star)$, with operators defined as

$$\tilde{F}(\tilde{u}) = (I - W^T)(\tilde{u}), \quad \tilde{G}(\tilde{u}) = D\tilde{u} - v \tag{14}$$

where D is a diagonal matrix defined by $J = (I + D)^{-1}$ (where we allow for the possibility of $D_{ii} = \infty$ for $J_{ii} = 0$).

An advantage of this approach when using Peaceman-Rachford splitting is that it allows us to reuse a fast method for multiplying by $(I + \alpha(I - W))^{-1}$ which is required by Peaceman-Rachford during both the forward pass (equilibrium solving) and backward pass (backpropagation) of training a monDEQ. Algorithms detailing both the Peaceman-Rachford and forward-backward solvers for the backpropagation problem (14) are given in Algorithms 3 and 4.

4 Example monotone operator networks

With the basic foundations from the previous section, we now highlight several different instantiations of the monDEQ architecture. In each of these settings, as in Theorem 1, we will formulate the objective as one of finding a solution to the operator splitting problem $0 \in (F + G)(z^\star)$ for

$$F(z) = (I - W)(z) - (Ux + b), \quad G = \partial f \tag{15}$$

or equivalently as computing an equilibrium point $z^\star = \text{prox}_f^1(Wz^\star + Ux + b)$.

In each of these settings we need to define what the input and hidden state x and z correspond to, what the W and U operators consist of, and what is the function f which determines the network nonlinearity. Key to the application of monotone operator methods are that 1) we need to constrain the W matrix such that $I - W \succeq mI$ as described in the previous section and 2) we need a method to compute (or solve) the inverse $(I + \alpha(I - W))^{-1}$, needed e.g. for Peaceman-Rachford; while this would not be needed if using only forward-backward splitting, we believe that the full power of the monotone operator view is realized precisely when these more involved methods are possible.

4.1 Fully connected networks

The simplest setting, of course, is the case we have largely highlighted above, where $x \in \mathbb{R}^d$ and $z \in \mathbb{R}^n$ are unstructured vectors, and $W \in \mathbb{R}^{n \times n}$ and $U \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$ are dense matrices and vectors respectively. As indicated above, we parameterize W directly by $A, B \in \mathbb{R}^{n \times n}$ as in (5). Since the Ux term simply acts as a bias in the iteration, there is no constraint on the form of U.

We can form an inverse directly by simply forming and inverting the matrix $I + \alpha(I - W)$, which has cost $O(n^3)$. Note that this inverse needs to be formed only once, and can be reused over all iterations of the operator splitting method and over an entire batch of examples (but recomputed, of course, when W changes). Any proximal function can be used as the activation: for example the ReLU, though as mentioned there are also close approximations to the sigmoid, tanh, and softplus.

4.2 Convolutional networks

The real power of monDEQs comes with the ability to use more structured linear operators such as convolutions. We let $x \in \mathbb{R}^{ds^2}$ be a d-channel input of size $s \times s$ and $z \in \mathbb{R}^{ns^2}$ be a n-channel hidden layer. We also let $W \in \mathbb{R}^{ns^2 \times ns^2}$ denote the linear form of a 2D convolutional operator and similarly for $U \in \mathbb{R}^{ns^2 \times ds^2}$. As above, W is parameterized by two additional convolutional operators A, B of the same form as W. Note that this implicitly increases the receptive field size of W: if A and B are 3×3 convolutions, then $W = (1-m)I - A^TA + B - B^T$ will have an effective kernel size of 5.

Inversion The benefit of convolutional operators in this setting is the ability to perform efficient inversion via the fast Fourier transform. Specifically, in the case that A and B represent circular convolutions, we can reduce the matrices to block-diagonal form via the discrete Fourier transform (DFT) matrix

$$A = F_s D_A F_s^* \tag{16}$$

where F_s denotes (a permuted form of) the 2D DFT operator and $D_A \in \mathbb{C}^{ns^2 \times ns^2}$ is a (complex) block diagonal matrix where each block $D_{A_{ii}} \in \mathbb{C}^{n \times n}$ corresponds to the DFT at one particular location in the image. In this form, we can efficiently multiply by the inverse of the convolutional operator, noting that

$$\begin{aligned} I + \alpha(I - W) &= (1+\alpha m)I + \alpha A^TA - \alpha B + \alpha B^T \\ &= F_s((1+\alpha m)I + \alpha D_A^* D_A - D_B + D_B^*)F_s^*. \end{aligned} \tag{17}$$

The inner term here is itself a block diagonal matrix with complex $n \times n$ blocks (each block is also guaranteed to be invertible by the same logic as for the full matrix). Thus, we can multiply a set of hidden units z by the inverse of this matrix by simply inverting each $n \times n$ block, taking the fast Fourier transform (FFT) of z, multiplying each corresponding block of $F_s z$ by the corresponding inverse, then taking the inverse FFT. The details are given in Appendix C.

The computational cost of multiplying by this inverse is $O(n^2s^2 \log s + n^3s^2)$ to compute the FFT of each convolutional filter and precompute the inverses, and then $O(bns^2 \log s + bn^2s^2)$ to multiply by the inverses for a set of hidden units with a minibatch of size b. Note that just computing the convolutions in a normal manner has cost $O(bn^2s^2)$, so that these computations are on the same order as performing typical forward passes through a network, though empirically 2-3 times slower owing to the relative complexity of performing the necessary FFTs.

One drawback of using the FFT in this manner is that it requires that all convolutions be circular; however, this circular dependence can be avoided using zero-padding, as detailed in Section C.2.

4.3 Forward multi-tier networks

Although a single fully-connected or convolutional operator within a monDEQ can be suitable for small-scale problems, in typical deep learning settings it is common to model hidden units at different hierarchical levels. While monDEQs may seem inherently "single-layer," we can model this same hierarchical structure by incorporating structure into the W matrix. For example, assuming a convolutional setting, with input $x \in \mathbb{R}^{ds^2}$ as in the previous section, we could partition z into L different hierarchical resolutions and let W have a multi-tier structure, e.g.

$$z = \begin{bmatrix} z_1 \in \mathbb{R}^{n_1 s_1^2} \\ z_2 \in \mathbb{R}^{n_2 s_2^2} \\ \vdots \\ z_L \in \mathbb{R}^{n_L s_L^2} \end{bmatrix}, \qquad W = \begin{bmatrix} W_{11} & 0 & 0 & \cdots & 0 \\ W_{21} & W_{22} & 0 & \cdots & 0 \\ 0 & W_{32} & W_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & W_{LL} \end{bmatrix}$$

where z_i denotes the hidden units at level i, an $s_i \times s_i$ resolution hidden unit with n_i channels, and where W_{ii} denotes an n_i channel to n_i channel convolution, and $W_{i+1,i}$ denotes an n_i to n_{i+1} channel, *strided* convolution. This structure of W allows for both inter- and intra-tier influence.

One challenge is to ensure that we can represent W with the form $(1-m)I - A^TA + B - B^T$ while still maintaining the above structure, which we achieve by parameterizing each W_{ij} block appropriately. Another consideration is the inversion of the multi-tier operator, which can be achieved via the FFT similarly as for single-convolutional W, but with additional complexity arising from the fact that the $A_{i+1,i}$ convolutions are strided. These details are described in Appendix D.

CIFAR-10

Method	Model size	Acc.
Neural ODE	172K	55.3±0.3%
Aug. Neural ODE	172K	58.9±2.8%
Neural ODE†*	1M	59.9%
Aug. Neural ODE†*	1M	73.4%
monDEQ (ours)		
Single conv	172K	**74.0±0.1%**
Multi-tier	170K	72.0±0.3%
Single conv*	854K	82.0±0.3%
Multi-tier*	1M	**89.0±0.3%**

SVHN

Method	Model size	Acc.
Neural ODE‡	172K	81.0%
Aug. Neural ODE‡	172K	83.5%
monDEQ (ours)		
Single conv	172K	88.7±1.1%
Multi-tier	170K	**92.4±0.1%**

MNIST

Method	Model size	Acc.
Neural ODE‡	84K	96.4%
Aug. Neural ODE‡	84K	98.2%
monDEQ (ours)		
Fully connected	84K	98.1±0.1%
Single conv	84K	**99.1±0.1%**
Multi-tier	81K	99.0±0.1%

Table 1: Test accuracy of monDEQ models compared to Neural ODEs and Augmented Neural ODEs. *with data augmentation; †best test accuracy before training diverges; ‡as reported in [10].

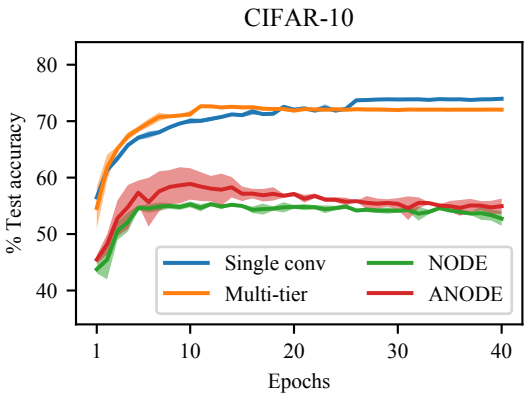

Figure 1: Test accuracy of monDEQs during training on CIFAR-10, with NODE [8] and ANODE [10] for comparison. NODE and ANODE curves obtained using code provided by [10].

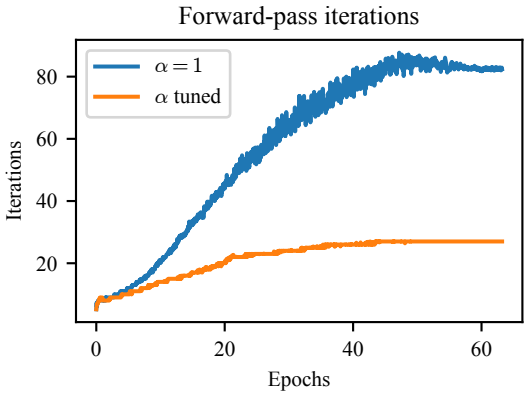

Figure 2: Iterations required by Peaceman-Rachford equilibrium solving over the course of training, for best α and $\alpha = 1$.

5 Experiments

To test the expressive power and training stability of monDEQs, we evaluate the monDEQ instantiations described in Section 4 on several image classification benchmarks. We take as a point of comparison the Neural ODE (NODE) [8] and Augmented Neural ODE (ANODE) [10] models, the only other implicit-depth models which guarantee the existence and uniqueness of a solution. We also assess the stability of training standard DEQs of the same form as our monDEQs.

The training process relies upon the operator splitting algorithms derived in Sections 3.4 and 3.5; for each batch of examples, the forward pass of the network involves finding the network fixed point (Algorithm 1 or 2), and the backward pass involves backpropagating the loss gradient through the fixed point (Algorithm 3 or 4). We analyze the convergence properties of both the forward-backward and Peaceman-Rachford operator splitting methods, and use the more efficient Peaceman-Rachford splitting for our model training. For further training details and model architectures see Appendix E. Experiment code can be found at `http://github.com/locuslab/monotone_op_net`.

Performance on image benchmarks We train small monDEQs on CIFAR-10 [17], SVHN [22], and MNIST [18], with a similar number of parameters as the ODE-based models reported in [8] and [10]. The results (averages over three runs) are shown in Table 1. Training curves for monDEQs, NODE, and ANODE on CIFAR-10 are show in Figure (1) and additional training curves are shown in Figure F1. Notably, except for the fully-connected model on MNIST, all monDEQs significantly outperform the ODE-based models across datasets. We highlight the performance of the small single convolution monDEQ on CIFAR-10 which outperforms Augmented Neural ODE by 15.1%.

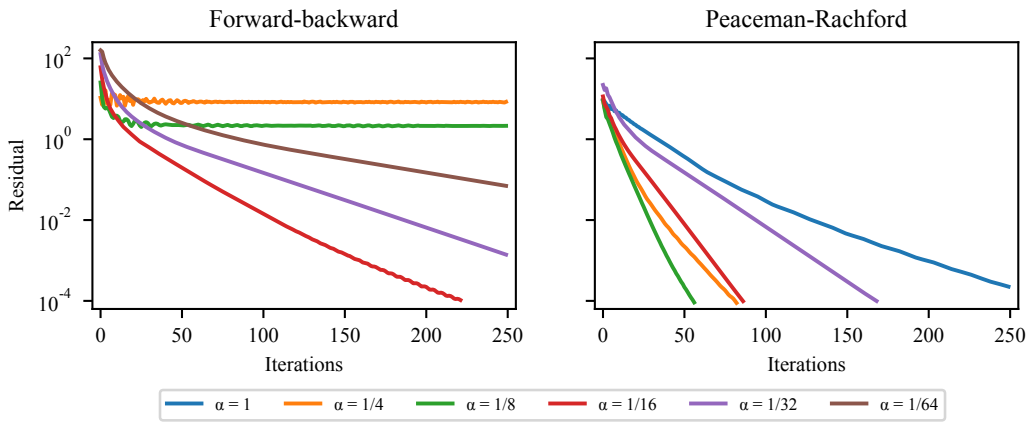

Figure 3: Convergence of Peaceman-Rachford and forward-backward equilibrium solving, on fully-trained model.

We also attempt to train standard DEQs of the same structure as our small multi-tier convolutional monDEQ. We train DEQs both with unconstrained W and with W having the monotone parameterization (5), and solve for the fixed point using Broyden's method as in [5]. All models quickly diverge during the first few epochs of training, even when allowed 300 iterations of Broyden's method.

Additionally, we train two larger monDEQs on CIFAR-10 with data augmentation. The strong performance (89% test accuracy) of the multi-tier network, in particular, goes a long way towards closing the performance gap with traditional deep networks. For comparison, we train larger NODE and ANODE models with a comparable number of parameters (~1M). These attain higher test accuracy than the smaller models during training, but diverge after 10-30 epochs (see Figure F1).

Efficiency of operator splitting methods We compare the convergence rates of Peaceman-Rachford and forward-backward splitting on a fully trained model, using a large multi-tier monDEQ trained on CIFAR-10. Figure 3 shows convergence for both methods during the forward pass, for a range of α. As the theory suggests, the convergence rates depend strongly on the choice of α. Forward-backward does not converge for $\alpha > 0.125$, but convergence speed varies inversely with α for $\alpha < 0.125$. In contrast, Peaceman-Rachford is guaranteed to converge for any $\alpha > 0$ but the dependence is non-monotonic. We see that, for the optimal choice of α, Peaceman-Rachford can converge much more quickly than forward-backward. The convergence rate also depends on the Lipschitz parameter L of $I - W$, which we observe increases during training. Peaceman-Rachford therefore requires an increasing number of iterations during both the forward pass (Figure 2) and backward pass (Figure F2).

Finally, we compare the efficiency of monDEQ to that of the ODE-based models. We report the time and number of function evaluations (OED solver steps or operator splitting iterations) required by the ~170k-parameter models to train on CIFAR-10 for 40 epochs. The monDEQ, neural ODE, and ANODE training takes respectively 1.4, 4.4, and 3.3 hours, with an average of 20, 96, and 90 function evals per minibatch. Note however that training the larger 1M-parameter monDEQ on CIFAR-10 requires 65 epochs and takes 16 hours. All experiments are run on a single RTX 2080 Ti GPU.

6 Conclusion

The connection between monotone operator splitting and implicit network equilibria brings a new suite of tools to the study of implicit-depth networks. The strong performance, efficiency, and guaranteed stability of monDEQ indicate that such networks could become practical alternatives to deep networks, while the flexibility of the framework means that performance can likely be further improved by, e.g. imposing additional structure on W or employing other operator splitting methods. At the same time, we see potential for the study of monDEQs to inform traditional deep learning itself. The guarantees we can derive about what architectures and algorithms work for implicit-depth networks may give us insights into what will work for explicit deep networks.

Broader impact statement

While the main thrust of our work is foundational in nature, we do demonstrate the potential for implicit models to become practical alternatives to traditional deep networks. Owing to their improved memory efficiency, these networks have the potential to further applications of AI methods on edge devices, where they are currently largely impractical. However, the work is still largely algorithmic in nature, and thus it is much less clear the immediate societal-level benefits (or harms) that could result from the specific tehniques we propose and demonstrate in this paper.

Acknowledgements

Ezra Winston is supported by a grant from the Bosch Center for Artificial Intelligence.

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In *Neural Information Processing Systems*, 2019.

[2] L. B. Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial Neural Networks*. 1990.

[3] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 2017.

[4] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8299–8310, 2018.

[5] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 688–699, 2019.

[6] H. H. Bauschke, P. L. Combettes, et al. *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer, 2011.

[7] A. Bibi, B. Ghanem, V. Koltun, and R. Ranftl. Deep layers as stochastic solvers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryxxCiRqYX`.

[8] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[9] P. L. Combettes and J.-C. Pesquet. Deep neural network structures solving variational inequalities. *Set-Valued and Variational Analysis*, pages 1–28, 2020.

[10] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

[11] L. El Ghaoui, F. Gu, B. Travacca, and A. Askari. Implicit deep learning. *arXiv preprint arXiv:1908.06315*, 2019.

[12] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.

[13] S. Gould, R. Hartley, and D. Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[15] M. Johnson, D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in Neural Information Processing Systems*, 2016.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[17] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[18] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[19] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091, 2018.

[20] C. K. Ling, F. Fang, and J. Z. Kolter. What game are we playing? End-to-end learning in normal and extensive form games. In *International Joint Conference on AI*, 2018.

[21] D. Linsley, A. K. Ashok, L. N. Govindarajan, R. Liu, and T. Serre. Stable and expressive recurrent vision models, 2020.

[22] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[23] F. J. Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Advances in Neural Information Processing Systems*, 1988.

[24] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 2019.

[25] M. Revay and I. R. Manchester. Contracting implicit recurrent neural networks: Stable models with improved trainability. *arXiv preprint arXiv:1912.10402*, 2019.

[26] E. K. Ryu and S. Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1): 3–43, 2016.

[27] L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019.

A Monotone operator theory

We briefly review some of the basic properties of monotone operators that we make use of throughout this work. A *relation* or *operator* (which in our setting will often roughly correspond to a set-valued function), is a subset of the space $F \subseteq \mathbb{R}^n \times \mathbb{R}^n$; we use the notation $F(x) = \{y|(x,y) \in F\}$ or simply $F(x) = y$ if only a single y is contained in this set. We make use of a few basic operators and relations: the identity operator $I = \{(x,x)|x \in \mathbb{R}^n\}$; the operator sum $(F+G)(x) = \{(x, y+z)|(x,y) \in F, (x,z) \in G\}$; the inverse operator $F^{-1}(x,y) = \{(y,x)|(x,y) \in F\}$; and the subdifferential operator $\partial f = \{(x, \partial f(x))| \ \in \mathrm{dom} f\}$. An operator F has Lipschitz constant L if for any $(x,u),(y,v) \in F$

$$\|u - v\|_2 \leq L\|x - y\|_2. \tag{A1}$$

An operator F is monotone if

$$(u-v)^T(x-y) \geq 0, \ \ \forall (x,u),(y,v) \in F \tag{A2}$$

which for the case of F being a function $F : \mathbb{R}^n \to \mathbb{R}^n$ is equivalent to the condition

$$(F(x) - F(y))^T(x-y) \geq 0, \ \ \forall x, y \in \mathrm{dom}\, F. \tag{A3}$$

In the case of scalar-valued functions, this corresponds to our common notion of a monotonic function. The operator F is strongly monotone with parameter m if

$$(u-v)^T(x-y) \geq m\|x-y\|^2, \ \ \forall (x,u),(y,v) \in F. \tag{A4}$$

A monotone operator F is *maximal monotone* if no other monotone operator strictly contains it; formally, most of the convergence properties we use require maximal monotonicity, though we are intentionally informal about this and merely use the established fact that several well-known operators are maximal monotone. Specifically, a linear operator $F(x) = Gx + h$ for $G \in \mathbb{R}^{n \times n}$ and $h \in \mathbb{R}^n$ is (maximal) monotone if and only if $G + G^T \succeq 0$ and strongly monotone if $G + G^T \succeq mI$. Similarly, a subdifferentiable operator ∂f is maximal monotone iff f is a convex closed proper (CCP) function.

The resolvent and Cayley operators for an operator F are denoted R_F and C_F and respectively defined as

$$R_F = (I + \alpha F)^{-1}, \ \ C_F = 2R_F - I \tag{A5}$$

for any $\alpha > 0$. The resolvent and Cayley operators are non-expansive (i.e., have Lipschitz constant $L \leq 1$) for any maximal monotone F, and are contractive (i.e. $L < 1$) for strongly monotone F.

We will mainly use two well-known properties of these operators. First, when $F(x) = Gx + h$ is linear, then

$$R_F(x) = (I + \alpha G)^{-1}(x - \alpha h) \tag{A6}$$

and when $F = \partial f$ for some CCP function f, then the resolvent is given by a proximal operator

$$R_F(x) = \mathrm{prox}_f^\alpha(x) \equiv \underset{z}{\mathrm{argmin}} \, \frac{1}{2}\|x - z\|_2^2 + \alpha f(z). \tag{A7}$$

Operator splitting approaches refer to methods to find a zero in a sum of operators (assumed here to be maximal monotone), i.e., find x such that

$$0 \in (F + G)(x). \tag{A8}$$

There are many such operator splitting methods, which lead to different approaches in their application to our subsequent implicit networks, but the two we use mainly in this work are 1) *forward-backward* splitting, given by the update

$$x^{k+1} := R_G(x^k - \alpha F(x^k)); \tag{A9}$$

and 2) *Peaceman-Rachford* splitting, which is given by the iteration

$$u^{k+1} = C_F C_G(u^k), \ \ x^k = R_G(u^k). \tag{A10}$$

Both methods will converge linearly to an x that is a zero of the operator sum under certain conditions: a sufficient condition for forward-backward to converge is that F be strongly monotone with parameter m and Lipschitz with constant L and $\alpha < 2m/L^2$; for Peaceman-Rachford, the method will converge for any choice of α for strongly monotone F, though the convergence speed will often vary substantially based upon α.

B Proofs

B.1 Proof of Theorem 1

Proof. The proof here is immediate: the forward-backward algorithm applied to the above operators with $\alpha = 1$ corresponds exactly to the network's fixed-point iteration:

$$\begin{aligned} z^{k+1} &= R_G(z^k - \alpha F(z^k)) \\ &= \text{prox}_f^\alpha(z^k - \alpha(I - W)z^k + \alpha(Ux + b)) \\ &= \text{prox}_f^1(Wz^k + Ux + b). \end{aligned}$$

□

B.2 Proof of Proposition 1

Proof. First assume W is of this form. Then clearly

$$(I - W)/2 + (I - W)^T/2 = mI + A^TA \succeq mI. \tag{B1}$$

Alternatively, if $I - W \succeq mI \iff (1 - m)I \succeq (W + W^T)/2$, then

$$(W + W^T)/2 = (1 - m)I - A^TA. \tag{B2}$$

Thus

$$\begin{aligned} W &= (W + W^T)/2 + (W - W^T)/2 \\ &= (1 - m)I - A^TA + B - B^T. \end{aligned}$$

□

B.3 Proof of Theorem 2

Proof. Differentiating both sides of the fixed-point equation $z^\star = \sigma(Wz^\star + Ux + b)$ we have

$$\begin{aligned} \frac{\partial z^\star}{\partial(\cdot)} &= \frac{\partial \,\text{prox}_f^1(Wz^\star + Ux + b)}{\partial(\cdot)} \\ &= J\left(W\frac{\partial z^\star}{\partial(\cdot)} + \frac{\partial(Wz^\star + Ux + b)}{\partial(\cdot)}\right) \end{aligned} \tag{B3}$$

for J defined in (10) (we require the Clarke generalized Jacobian owing to the fact that the nonlinearity need not be a smooth function). Rearranging we get

$$\begin{aligned} (I - JW)\frac{\partial z^\star}{\partial(\cdot)} &= J\frac{\partial(Wz^\star + Ux + b)}{\partial(\cdot)} \\ \Leftrightarrow \frac{\partial z^\star}{\partial(\cdot)} &= (I - JW)^{-1}J\frac{\partial(Wz^\star + Ux + b)}{\partial(\cdot)}. \end{aligned} \tag{B4}$$

To show that this derivative always exists, we need to show that the $I - JW$ matrix is nonsingular. Owing to the fact that proximal operators are monotone and non-expansive, we have $0 \leq J_{ii} \leq 1$. First, letting $\lambda(\cdot)$ denote the set of eigenvalues of a matrix, note that

$$\lambda(I - JW) = \lambda(I - J^{1/2}WJ^{1/2}). \tag{B5}$$

This follows from the similarity transform $\lambda(I - JW) = \lambda(J^{-1/2}(I - JW)J^{1/2})$ for $J > 0$ and the case of $J_{ii} = 0$ follows via the continuity of eigenvalues taking $\lim J_{ii} \to 0$. Now, using the fact that $0 \preceq J \preceq I$, we have

$$\begin{aligned} &\text{Re }\lambda(I - J^{1/2}WJ^{1/2}) \\ &= \text{Re }\lambda(I - J + J^{1/2}(I - W)J^{1/2}) > 0 \end{aligned} \tag{B6}$$

since $I - W \succeq mI$ and $I - J \succeq 0$. □

B.4 Proof of Theorem 3

Proof. We begin with the case where $J_{ii} \neq 0$ and thus $D_{ii} < \infty$. As above, because proximal operators are themselves monotone non-expansive operators, we always $0 \leq J_{ii} \leq 1$, so that $D_{ii} \geq 0$.

Now, first assuming that $J_{ii} > 0$, and hence $D_{ii} < \infty$, we have

$$\begin{aligned} & u = (I - JW)^{-T} v \\ \Leftrightarrow\ & (I - W^T (I + D)^{-1}) u = v \\ \Leftrightarrow\ & W^{-T} u - (I + D)^{-1} u = W^{-T} v \\ \Leftrightarrow\ & (I + D) W^{-T} u - u = (I + D) W^{-T} v \\ \Leftrightarrow\ & W^{-T} u - u + D W^{-T} u = (I + D) W^{-T} v \\ \Leftrightarrow\ & \tilde{u} - W^T \tilde{u} + D \tilde{u} = (I + D) W^{-T} v \end{aligned} \tag{B7}$$

where we define $\tilde{u} = W^{-T} u$. To simplify the right hand side of this equation and remove the explicit $W^{-T} v$ terms[4] we note that

$$(I - JW)^{-T} = (I - W^T J)^{-1} = I + (I - W^T J)^{-1} W^T J. \tag{B8}$$

Thus, we can always solve the above equation with the v term of the form $W^T J v$, giving

$$(I + D) W^{-T} W^T J v = (I + D) J v = v. \tag{B9}$$

This gives us a (linear) operator splitting problem with the $\tilde{F}$ and $\tilde{G}$ operators given in (14).

To handle the case where $J_{ii} = 0 \Leftrightarrow D_{ii} = \infty$, we can simply take the limit $D_{ii} \to \infty$, and note that all the operators are well-defined for this case. For instance, the resolvent operator

$$R_{\tilde{G}}(u) = (I + \alpha(I + D))^{-1} (u + \alpha v) \tag{B10}$$

and thus

$$R_{\tilde{G}}(u)_{ii} = \frac{u + \alpha v}{1 + \alpha(1 + D_{ii})} \to 0 \tag{B11}$$

as $D_{ii} \to \infty$.

Finally, owing to the fact that $I - W^T \succeq mI$ and $D_{ii} \geq 0$, the $\tilde{F}$ and $\tilde{G}$ operators are strongly monotone and monotone respectively, we conclude that operator splitting techniques applied to the problem will be guaranteed to converge. □

C Convolutional monDEQs

C.1 Inversion via the discrete Fourier transform

First consider the case where $W \in \mathbb{R}^{s^2 \times s^2}$ is the matrix representation of an unstrided (circular) convolution with a single input channel and single output channel. The convolution operates on vectorized $s \times s$ inputs. It is well known that W is diagonalized by the 2D DFT operator $\mathscr{F}_s = F_s \otimes F_s$ where F_s is the Fourier basis matrix $(F_s)_{ij} = \frac{1}{s} \omega^{(i-1)(j-1)}$ and $\omega = \exp(2\pi\iota/s)$. We denote $\iota = \sqrt{-1}$ to avoid confusion with the index i. So

$$\mathscr{F}_s W \mathscr{F}_s^* = D, \tag{C1}$$

a complex diagonal matrix.

Now take the case where $W \in \mathbb{R}^{ns^2 \times ns^2}$ has n input and output channels. Then

$$(I_n \otimes \mathscr{F}_s) W (I_n \otimes \mathscr{F}_s^*) = D = \begin{bmatrix} D_{11} & D_{12} & \cdots & D_{1n} \\ D_{21} & D_{22} & \cdots & D_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ D_{n1} & D_{n2} & \cdots & D_{nn} \end{bmatrix} \tag{C2}$$

where I_n is the $n \times n$ identity matrix and each block $D_{ij} \in \mathbb{C}^{s^2 \times s^2}$ is a complex diagonal matrix. We will denote $\mathscr{F}_{s,n} = I_n \otimes \mathscr{F}_s$.

[4] Although we could solve this operator splitting problem directly, the presence of the $W^{-T} v$ term has two notable downsides: 1) even if the W matrix itself is nonsingular, it may be arbitrarily close to a singular matrix, thus making direct solutions with this matrix introduce substantial numerical errors; and 2) for operator splitting methods that do *not* require an inverse of W (e.g. forward-backward splitting), it would be undesirable to require an explicit inverse in the backward pass.

It is more efficient to consider the permuted form of D

$$\hat{D} = \begin{bmatrix} \hat{D}^1 & 0 & \cdots & 0 \\ 0 & \hat{D}^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{D}^{s^2} \end{bmatrix} \tag{C3}$$

where each block $\hat{D}^k \in \mathbb{C}^{n \times n}$, consists of the kth diagonal elements of all the D_{ij}, that is $\hat{D}^k_{ij} = (D_{ij})_{kk}$. Then inverting or multiplying by $\hat{D}$ reduces to inverting or multiplying by the diagonal blocks, which is amenable to accelerated batch-wise computation in the form of an $s^2 \times n \times n$ tensor. However, the original form (C2) is more convenient mathematically and we use that here.

To perform the required inversion of the operator

$$I + \alpha(I - W) = (1 + \alpha m)I + \alpha A^T A - \alpha B + \alpha B^T \tag{C4}$$

we use the fact that $\mathscr{F}_{s,n}$ is unitary and obtain

$$\begin{aligned} &(1 + \alpha m)I + \alpha A^T A - \alpha B + \alpha B^T \\ &\quad = (1 + \alpha m)\mathscr{F}^*_{s,n}\mathscr{F}_{s,n} + \mathscr{F}^*_{s,n}(\alpha D^*_A \mathscr{F}_{s,n}\mathscr{F}^*_{s,n} D_A - D_B + D^*_B)\mathscr{F}_{s,n} \\ &\quad = \mathscr{F}^*_{s,n}((1 + \alpha m)I + \alpha D^*_A D_A - D_B + D^*_B)\mathscr{F}_{s,n}. \end{aligned} \tag{C5}$$

The inner term here itself has the blockwise-diagonal form (C2). Thus, we can multiply a set of hidden units z by the inverse of this matrix by considering the permuted form (C3), inverting each block $\hat{D}^i$, taking the FFT of z, multiplying each corresponding block of $\mathscr{F}_{s,n} z$ by the corresponding inverse, then taking the inverse FFT.

C.2 Zero padding

One drawback to the above method is that using the FFT in this manner requires that all convolutions be circular. While empirically there is little drawback to simply replacing traditional convolutions with their circular variants, in some cases it may be desirable to avoid this setting, where information about the image may wrap around the borders. If it is desirable to avoid this, we explicitly remove any circular dependence by zero-padding the hidden units with $(k-1)/2$ border pixels, where k denotes the receptive field size of the convolution. This zero padding can then be enforced by simply setting all the border entries to zero within the *nonlinearity* of the network; because setting an element to zero is equivalent to the proximal operator for the indicator of the zero set, such operations still fit within the monotone operator setting.

D Multi-tier monDEQs

D.1 Parameterization

Recall the setting of Section 4.3, with

$$z = \begin{bmatrix} z_1 \in \mathbb{R}^{n_1 s_1^2} \\ z_2 \in \mathbb{R}^{n_2 s_2^2} \\ \vdots \\ z_L \in \mathbb{R}^{n_L s_L^2} \end{bmatrix}, \qquad W = \begin{bmatrix} W_{11} & 0 & 0 & \cdots & 0 \\ W_{21} & W_{22} & 0 & \cdots & 0 \\ 0 & W_{32} & W_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & W_{LL} \end{bmatrix}. \tag{D1}$$

To ensure W has the form $(1-m)I - A^T A + B - B^T$, we restrict both A and B to have the same bidiagonal structure as W. Then the diagonal terms W_{ii} have the form

$$W_{ii} = (1 - m)I - A^T_{ii} A_{ii} - A^T_{i+1,i} A_{i+1,i} + B_{ii} - B^T_{ii} \tag{D2}$$

for $i < L$ and

$$W_{LL} = (1 - m)I - A^T_{LL} A_{LL} + B_{LL} - B^T_{LL}. \tag{D3}$$

To compute the off-diagonal terms $W_{i+1,i}$ note that restricting W to be bidiagonal makes the off-diagonal terms of B redundant. E.g. since $W_{12} = 0$, then

$$\begin{aligned} -A_{21}^T A_{22} - B_{21}^T &= W_{12} = 0 \\ \Rightarrow W_{21} &= -A_{22}^T A_{21} + B_{21} = -2A_{22}^T A_{21}. \end{aligned} \tag{D4}$$

D.2 Inversion via the discrete Fourier transform

Consider W of the form (D1) with convolutions

$$\begin{aligned} W_{ii} &= (1-m)I - A_{ii}^T A_{ii} - A_{i+1,i}^T A_{i+1,i} + B_{ii} - B_{ii}^T \\ W_{LL} &= (1-m)I - A_{LL}^T A_{LL} + B_{LL} - B_{LL}^T \\ W_{i+1,i} &= -2A_{i+1,i+1}^T A_{i+1,i}. \end{aligned} \tag{D5}$$

Here the A_{ii} and B_{ii} terms are unstrided convolutions with n_i input and n_i output channels, while the $A_{i,i+1}$ are strided convolutions with n_i input channels and n_{i+1} output channels.

In order to multiply by $(I + \alpha(I - W))^{-1}$, we use back substitution to solve for x in

$$z = (I + \alpha(I - W))x. \tag{D6}$$

Let $W' = (I + \alpha(I - W))$. The back substitution proceeds by tiers, i.e.

$$\begin{aligned} x_1 &= W_{11}'^{-1} z_1 \\ x_2 &= W_{22}'^{-1}(z_2 - W_{21}' x_1) \\ x_3 &= W_{33}'^{-1}(z_3 - W_{32}' x_2) \\ &\vdots \end{aligned} \tag{D7}$$

Therefore only the diagonal blocks W_{ii}' need be inverted. The inversion of e.g.

$$W_{11}' = (1 + \alpha m)I + \alpha(A_{11}^T A_{11} + A_{21}^T A_{21} + B_{11} - B_{11}^T) \tag{D8}$$

is complicated by the fact that A_{21} is strided, so that it is no longer diagonalized by the DFT. Instead, we perform inversion using the following proposition.

Proposition D1. *Let $A \in \mathbb{R}^{n_1 s^2 \times n_1 s^2}$ be an unstrided circular convolution with n_1 input and n_1 output channels, and $B \in \mathbb{R}^{n_2 s^2 \times n_1 s^2}$ a strided circular convolution with n_1 input and n_2 output channels and stride r where r divides s. Then*

$$(A + B^T B)^{-1} = \mathscr{F}_{s,n_1}^* (D_A^{-1} - D_A^{-1} D_B^* (I_{n_2} \otimes K) D_B D_A^{-1}) \mathscr{F}_{s,n_1} \tag{D9}$$

where

$$\begin{aligned} D_A = \mathscr{F}_{s,n_1} A \mathscr{F}_{s,n_1}^*, \quad D_B = \mathscr{F}_{s,n_2} B \mathscr{F}_{s,n_1}^*, \\ K = S^T J (s^2 r^2 I + J^T S D_B D_A^{-1} D_B^* S^T J)^{-1} J^T S \end{aligned} \tag{D10}$$

where $J = 1_{r^2} \otimes I_{s^2/r^2}$ is r^2 stacked identity matrices of size $(s^2/r^2) \times (s^2/r^2)$ and $S = (I_r \otimes S_{s/r,s})$ is a permutation matrix where $S_{a,b} \in \mathbb{R}^{ab \times ab}$ denotes the perfect shuffle matrix defined by subselecting rows of the identity matrix I_{ab}, here given in MATLAB *notation:*

$$S_{a,b} = \begin{bmatrix} I_{ab}(1:b:ab,:) \\ I_{ab}(2:b:ab,:) \\ \vdots \\ I_{ab}(b:b:ab,:) \end{bmatrix}. \tag{D11}$$

Proof. We will show that

$$A + B^T B = \mathscr{F}_{s,n_1}^* (D_A + D_B^* (I_{n_2} \otimes (\frac{1}{s^2 r^2} S^T J J^T S)) D_B) \mathscr{F}_{s,n_1}. \tag{D12}$$

The desired result then follows by applying the Woodbury matrix idenetity.

We start by breaking B into an unstrided convolution B' which can be diagonalized by the DFT and a matrix $U_{r,s}$ which performs the striding on each channel:

$$B = (I_{n_2} \otimes U_{r,s}) B' = (I_{n_2} \otimes U_{r,s}) \mathscr{F}_{s,n_2}^* D_B \mathscr{F}_{s,n_1} \tag{D13}$$

where $U_{r,s} \in \mathbb{R}^{(s^2/r^2)\times s^2}$ is defined by subselecting rows of the identity matrix:

$$U_{r,s} = \begin{bmatrix} I_{s^2}(1:r:s,:) \\ I_{s^2}(rs+1:r:(r+1)s,:) \\ I_{s^2}(2rs+1:r:(2r+1)s,:) \\ \vdots \\ I_{s^2}(s^2-sr+1:r:s^2-s(r-1),:) \end{bmatrix}. \tag{D14}$$

So

$$B^T B = \mathscr{F}^*_{s,n_1} D^*_B (I_{n_2} \otimes (\mathscr{F}_s U^T_{r,s} U_{r,s} \mathscr{F}^*_s)) D_B \mathscr{F}_{s,n_1}. \tag{D15}$$

We want to show that $\mathscr{F}_s U^T_{r,s} U_{r,s} \mathscr{F}^*_s = \frac{1}{s^2 r^2} S^T J J^T S$. Observe that

$$U^T_{r,s} U_{r,s} = (T_{r,s} \otimes T_{r,s}) \tag{D16}$$

where $T_{r,s} \in \mathbb{R}^{s\times s}$ is given by

$$(T_{r,s})_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } i \pmod r = 1, \\ 0 & \text{else.} \end{cases} \tag{D17}$$

Then by the properties of Kronecker product

$$\mathscr{F}_s U^T_{r,s} U_{r,s} \mathscr{F}^*_s = (F_s \otimes F_s)(T_{r,s} \otimes T_{r,s})(F^*_s \otimes F^*_s) = (F_s T_{r,s} F^*_s) \otimes (F_s T_{r,s} F^*_s). \tag{D18}$$

We now show that $(F_s T_{r,s} F^*_s) = L$ where

$$L_{ij} = \begin{cases} \frac{1}{sr} & \text{if } i \equiv j \pmod{s/r}, \\ 0 & \text{else.} \end{cases} \tag{D19}$$

To do so we use several properties of the roots of unity $z^k = \exp(2\pi \iota k/s)$.

1. If $a \equiv b \pmod s$ then $z^a = z^b$.
2. If z is a primitive sth root of unity then z^m is a primitive ath root of unity where $a = \frac{s}{\gcd(m,s)}$.
3. The sum of the sth roots of unity $\sum_{k=0}^{s-1} z^k = 0$ if $s > 1$.

We first compute L_{ij} for the case when $i \equiv j \pmod{s/r}$, or in other words $i = j + \frac{ks}{r}$ for some integer k. We have

$$\begin{aligned} L_{ij} &= \frac{1}{s^2} \sum_{p=1:r:s} \omega^{(i-1)(p-1)} \bar{\omega}^{(p-1)(j-1)} \\ &= \frac{1}{s^2} \sum_{p=0:r:s-1} \exp(2\pi \iota p(i-j)/s) \\ &= \frac{1}{s^2} \sum_{p=0:r:s-1} \exp(2\pi \iota pk/r) \\ &= \frac{1}{s^2} \sum_{p=0}^{\frac{s}{r}-1} \exp(2\pi \iota pk) \\ &= \frac{1}{sr}. \end{aligned} \tag{D20}$$

For the case when $i \not\equiv j \pmod{s/r}$, or in other words $i = j + \frac{ks}{r} + m$ for some integers k and m with $-\frac{s}{r} < m < \frac{s}{r}$, we have

$$\begin{aligned} L_{ij} &= \frac{1}{s^2} \sum_{p=1:r:s} \omega^{(i-1)(p-1)} \bar{\omega}^{(p-1)(j-1)} \\ &= \frac{1}{s^2} \sum_{p=0:r:s-1} \exp(2\pi\imath p(i-j)/s) \\ &= \frac{1}{s^2} \sum_{p=0}^{\frac{s}{r}-1} \exp(2\pi\imath p(i-j)r/s) \\ &= \frac{1}{s^2} \sum_{p=0}^{\frac{s}{r}-1} \exp(2\pi\imath pmr/s) \exp(2\pi\imath pk). \end{aligned} \tag{D21}$$

By property (2), since $\exp(2\pi\imath r/s)$ is a primitive $\frac{s}{r}$th root of unity, then $\exp(2\pi\imath mr/s)$ is a primitive dth root of unity where $d = \frac{s/r}{\gcd(m,s/r)}$. Since d divides s/r, we can split the sum into several sums of dth roots of unity using property (1), each of which will sum to zero by property (3).

$$\begin{aligned} L_{ij} &= \frac{1}{s^2} \sum_{p=0}^{\frac{s}{r}-1} \exp(2\pi\imath pmr/s) \\ &= \frac{1}{s^2} \sum_{q=0}^{\frac{s}{rd}-1} \sum_{p=0}^{d-1} \exp(2\pi\imath(p+qd)mr/s) \\ &= \frac{1}{srd} \sum_{p=0}^{d-1} \exp(2\pi\imath pmr/s) \\ &= 0 \end{aligned} \tag{D22}$$

where the second equality follows from property (1) since $p = p + qd \pmod{d}$ and each sum in the third line is zero by property (3) since $\exp(2\pi\imath mr/s)$ is a primitive dth root of unity.

We now have $\mathscr{F}_s U_{r,s}^T U_{r,s} \mathscr{F}_s^* = L \otimes L$ and it remains to use properties of Kronecker product to show that $L \otimes L = \frac{1}{s^2 r^2} S^T J J^T S$. In particular we need associativity and the fact that for $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{m \times m}$, we have

$$B \otimes A = S_{n,m}(A \otimes B) S_{n,m}^T \tag{D23}$$

where $S_{n,m}$ is the perfect shuffle matrix. Note that $L = \frac{1}{sr} 1_{r \times r} \otimes I_{s/r}$ where $1_{r \times r}$ is the $r \times r$ matrix of all ones. Then

$$\begin{aligned} L \otimes L &= \frac{1}{s^2 r^2} (1_{r \times r} \otimes I_{s/r}) \otimes (1_{r \times r} \otimes I_{s/r}) \\ &= \frac{1}{s^2 r^2} 1_{r \times r} \otimes (I_{s/r} \otimes (1_{r \times r} \otimes I_{s/r})) \\ &= \frac{1}{s^2 r^2} 1_{r \times r} \otimes (S_{s/r,s}((1_{r \times r} \otimes I_{s/r}) \otimes I_{s/r}) S_{s/r,s}^T) \\ &= \frac{1}{s^2 r^2} 1_{r \times r} \otimes (S_{s/r,s}(1_{r \times r} \otimes I_{s^2/r^2}) S_{s/r,s}^T) \\ &= \frac{1}{s^2 r^2} (I_r \otimes S_{s/r,s})(1_{r^2 \times r^2} \otimes I_{s^2/r^2})(I_r \otimes S_{s/r,s}^T) \\ &= \frac{1}{s^2 r^2} S J J^T S^T \end{aligned} \tag{D24}$$

which completes the proof. □

	CIFAR-10			
	Single conv	Multi-tier	Single conv lg.	Multi-tier lg.
Num. channels	81	(16,32,60)	200	(64,128,128)
Num. params	172,218	170,194	853,612	1,014,546
Epochs	40	40	65	65
Initial lr	0.001	0.01	0.001	0.001
Lr schedule	step decay	step decay	1-cycle	1-cycle
Lr decay steps	25	10	-	-
Lr decay factor	10	10	-	-
Max learning rate	-	-	0.01	0.05
Data augmentation	-	-	✓	✓

	SVHN		MNIST		
	Single conv	Multi-tier	FC	Single conv	Multi-tier
Num. channels	81	(16,32,60)	87*	54	(16, 32, 32)
Num. params	172,218	170,194	84,313	84,460	81,394
Initial lr	0.001	0.001	0.001	0.001	0.001
Epochs	40	40	40	40	40
Lr decay steps	25	10	10	10	10
Lr decay factor %	10	10	10	10	10

Table E1: Model hyperparameters. *FC is a dense layer with output dimension of 87.

E Experiment details

E.1 Model architecture

Recall that a monDEQ is defined by a choice of linear operators W and U, bias b, and nonlinearity σ, and that we parameterize W via linear operators A and B. For all experiments we use $\sigma = \text{ReLU}$. In the fully-connected network A, B and U are dense matrices; in the single-convolution network they are unstrided convolutions with kernel size 3. The structure of the multi-tier network is as described in (D1) and (D5); we use three tiers with unstrided convolutions for U and A_{ii}, B_{ii} and stride-2 convolutions for the subdiagonal terms $A_{i,i+1}$, all with kernels of size 3. The number of channels for single and multi-tier convolutional models varies by dataset, as shown in Table E1.

For all models, the fixed point $z^\star$ is mapped to logits $\hat{y}$ via a dense output layer, and the single convolution model first applies 4×4 average pooling:

$$\hat{y} = W_o z^\star + b_o \quad \text{or} \quad \hat{y} = W_o \,\text{AvgPool}_{4\times4}(z^\star) + b_o.$$

E.2 Training details

Because $W = (1-m)I - A^TA + B - B^T$ contains both linear and quadratic terms, we find that a variant of weight normalization helps to keep the gradients of the different parameters on the same scale. For example, when W is a dense matrix, we reparameterize A^TA as $g\frac{A^TA}{\|A\|^2}$ and B as $h\frac{B}{\|B\|}$, where g and h are learned scalars. When W consists of a single or multi-tiered convolutions, we reparameterize each convolution kernel analogously.

All models are trained by running Peaceman-Rachford with error tolerence ϵ =1e-2, which reduces the number of iterations without impacting performance. The monotonicity parameter m also affects convergence speed since it controls the contraction factor of the relevant operators; consistent with this, we find that Peaceman-Rachford takes longer to converge for smaller m, and use $m = 1$ for all models since model performance is not sensitive to $m \in [0.01, 1]$. We also find that the Lipschitz parameter L of $I - W$ increases during training, changing the optimal α value. We therefore tune $\alpha \in \{1, 1/2, 1/4, \ldots\}$ over the course of training so as to minimize forward-pass iterations.

One detail about stopping criteria for the splitting method: computing the residual $\|z^{k+1} - f(z^{k+1})\|/\|z^{k+1}\|$ requires an additional call to the function f. Therefore during training we instead use the criterion $\|z^{k+1} - z^k\|/\|z^{k+1}\| \leq \epsilon$. The error shown in Figure 3 is the former, while the stopping criterion used in Figures 2 and F2 is the latter. Technically this latter criterion itself depends on both α and L; for different α and L values, having $\|z^{k+1} - z^k\|/\|z^{k+1}\| \leq \epsilon$ implies different bounds on the residual. However, we find that this effect is minimal, so that both stopping criteria work equally well in practice.

	Train examples	Test examples	Image dim.	Num. channels
MNIST	60,000	10,000	28×28	1
SVHN	73,257	26,032	32×32	3
CIFAR-10	50,000	10,000	32×32	3

Table E2: Dataset statistics

Table E1 gives details of the training hyperparameters used for each model. All models are trained with ADAM [16], using batch size of 128. For all but the large CIFAR-10 models, the initial learning rate is chosen from {1e-2, 1e-3} and decayed by a factor of 10 after every 10 or 25 epochs, and the default ADAM momentum parameters are used. All training data is normalized to mean $\mu = 0$, standard deviation $\sigma = 1$.

CIFAR-10 with data augmentation When training large models on CIFAR-10 we use standard data augmentation, consisting of zero-padding the 32×32 images to 40×40, then randomly cropping back to 32x32, and finally performing random horizontal flips. In order to reduce the number of training epochs, we use a single cycle of increasing and decreasing learning rate to achieve super-convergence [27]. The learning rate is increased from 1e-3 to the max learning rate (see Table E1) over 30 epochs, then decreased back to 1e-3 over 30 epochs, then held at 1e-3 for 5 epochs. (The max learning rate is chosen by training for a single epoch while increasing the learning rate until the loss diverges.) The momentum is also decreased from 0.95 to 0.85 over 30 epochs, then back to 0.95 over 30 epochs, then held at 0.95 for 5 epochs. However, we note that the model obtains the same performance when trained with constant learning rate of 1e-3 for around 200 epochs.

E.3 Dataset statistics

MNIST [18] consists of black and white examples of handwritten digits 0-9. SVHN [22] consists of color images of digits 0-9 extracted from house numbers captured by Google Stree View. CIFAR-10 [17] consists of small images from 10 object classes. Dataset statistics are shown in Table E2.

F Additional results and figures

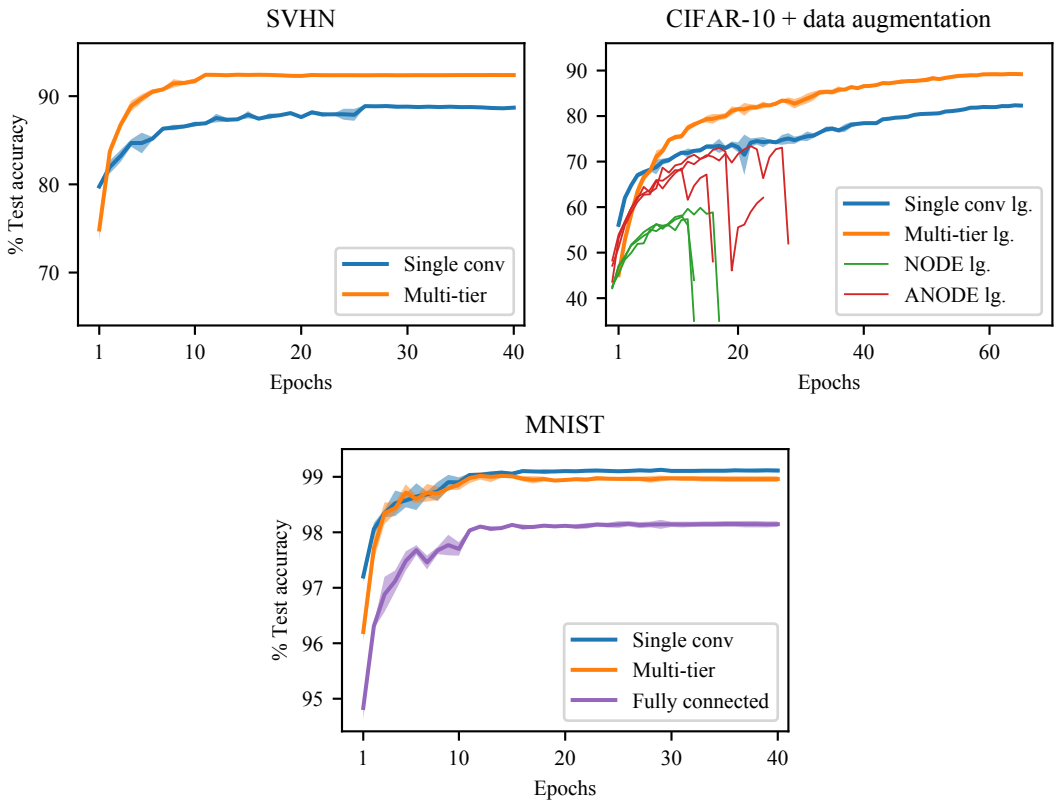

Figure F1: Test accuracy of monDEQs and Neural ODE models during training.

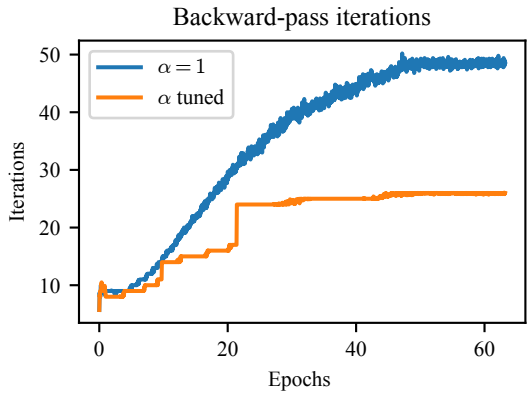

Figure F2: Iterations required by Peaceman-Rachford backprop over the course of training.