Joint inference and input optimization in equilibrium networks

Swaminathan Gurumurthy
Carnegie Mellon University

Shaojie Bai
Carnegie Mellon University

J. Zico Kolter
Carnegie Mellon University
Bosch Center for AI

Zachary Manchester
Carnegie Mellon University

Abstract

Many tasks in deep learning involve optimizing over the inputs to a network to minimize or maximize some objective; examples include optimization over latent spaces in a generative model to match a target image, or adversarially perturbing an input to worsen classifier performance. Performing such optimization, however, is traditionally quite costly, as it involves a complete forward and backward pass through the network for each gradient step. In a separate line of work, a recent thread of research has developed the deep equilibrium (DEQ) model, a class of models that foregoes traditional network depth and instead computes the output of a network by finding the fixed point of a single nonlinear layer. In this paper, we show that there is a natural synergy between these two settings. Although, naively using DEQs for these optimization problems is expensive (owing to the time needed to compute a fixed point for each gradient step), we can leverage the fact that gradient-based optimization can itself be cast as a fixed point iteration to substantially improve the overall speed. That is, we simultaneously both solve for the DEQ fixed point and optimize over network inputs, all within a single “augmented” DEQ model that jointly encodes both the original network and the optimization process. Indeed, the procedure is fast enough that it allows us to efficiently train DEQ models for tasks traditionally relying on an “inner” optimization loop. We demonstrate this strategy on various tasks such as training generative models while optimizing over latent codes, training models for inverse problems like denoising and inpainting, adversarial training and gradient based meta-learning.

1 Introduction

Many settings in deep learning involve optimization over the inputs to a network to minimize some desired loss. For example, for a “generator” network \( G : \mathcal{Z} \rightarrow \mathcal{X} \) that maps from latent space \( \mathcal{Z} \) to an observed space \( \mathcal{X} \), it may be desirable to find a latent vector \( z \in \mathcal{Z} \) that most closely produces some target output \( x \in \mathcal{X} \) by solving the optimization problem (e.g. \([10][13]\))

\[
\minimize_{z \in \mathcal{Z}} \| x - G_\theta(z) \|_2^2.
\]

As another example, constructing adversarial examples for classifiers \([28][53]\) typically involves optimizing over a perturbation to a given input; i.e., given a classifier network \( g : \mathcal{X} \rightarrow \mathcal{Y} \), task loss

*Correspondence to: Swaminathan Gurumurthy <sgurumur@andrew.cmu.edu>
Code available at [https://github.com/locuslab/JIIO-DEQ](https://github.com/locuslab/JIIO-DEQ)

ℓ : Y → ℝ₊, and a sample x ∈ X, we want to solve

\[
\maximize_{\|\delta\| \leq \epsilon} \ell(g(x + \delta)).
\] (2)

More generally, a wide range of inverse problems [10] and other auxiliary tasks [22, 3] in deep learning can also be formulated in such a manner.

Orthogonal to this line of work, a recent trend has focused on the use of an *implicit layer* within deep networks to avoid traditional depth. For instance, Bai et al. [5] introduced deep equilibrium models (DEQs) which instead treat the network as repeated applications of a single layer and compute the output of the network as a solution to an equilibrium-finding problem instead of simply specifying a sequence of non-linear layer operations. Bai et al. [5] and subsequent work [6] have shown that DEQs can achieve results competitive with traditional deep networks for many realistic tasks.

In this work, we highlight the benefit of using these implicit models in the context of input optimization routines. Specifically, because optimization over inputs itself is typically done via an iterative method (e.g., gradient descent), we can combine this optimization fixed-point iteration with the forward DEQ fixed point iteration all within a single “augmented” DEQ model that *simultaneously* performs forward model inference as well as optimization over the inputs. This enables the models to more quickly perform both the inference and optimization procedures, and the resulting speedups further allow us to *train* networks that use such “bi-level” fixed point passes. In addition, we also show a close connection between our proposed approach and the primal-dual methods for constrained optimization.

We illustrate our methods on 4 tasks that span across different domains and problems: 1) training DEQ-based generative models while optimizing over latent codes; 2) training models for inverse problems such as denoising and inpainting; 3) adversarial training of implicit models; and 4) gradient-based meta-learning. We show that in all cases, performing this simultaneous optimization and forward inference accelerates the process over a more naive inner/outer optimization approach. For instance, using the combined approach leads to a 3.5-9x speedup for generative DEQ networks, a 3x speedup in adversarial training of DEQ networks and a 2.5-3x speedup for gradient based meta-learning. In total, we believe this work points to a variety of new potential applications for optimization with implicit models.

2 Related Work

Implicit layers. Layers with implicitly defined depth have gained tremendous popularity in recent years [46, 19, 29]. Rather than a static computation graph, these layers define a condition on the output that the model must satisfy, which can represent “infinite” depth, be directly differentiated through via the implicit function theorem [47], and are memory-efficient to train. Some recent examples of implicit layers include optimization layers [16, 1], deep equilibrium models [5, 6, 68, 40, 52], neural ordinary differential equations (ODEs) [14, 18, 61], logical structure learning [67], and continuous generative models [30].

In particular, deep equilibrium models (DEQs) [5] define the output of the model as the fixed point of repeated applications of a layer. They compute this using black-box root-finding methods [5] or accelerated fixed-point iterations [36] (e.g., Broyden’s method [11]). In this work, we propose an efficient approach to perform input optimization with the DEQ by *simultaneously* optimizing over the inputs and solving the forward fixed point of an equilibrium model as a joint, augmented system. As related work, Jeon et al. [36] introduce fixed point iteration networks that generalize DEQs to repeated applications of gradient descent over variables. However, they don’t address the specific formulation presented in this paper, which has a number of practical use cases (e.g., adversarial training). Lu et al. [52] proposes an implicit version of normalizing flows by formulating a joint root-finding problem that defines an invertible function between the input x and output z*. Perhaps the most relevant approach to our work is Gilton et al. [26], which specifically formulates inverse imaging problems as a DEQ model. In contrast, our approach focuses on solving input optimization problems where the network of interest is *already* a DEQ, and thus the combined optimization and forward inference task leads to a substantially different set of update equations and tradeoffs.
Input optimization in deep learning. Many problems in deep learning can be framed as optimizing over the inputs to minimize some objective. Some canonical examples of this include finding adversarial examples [53,45], solving inverse problems [10,13,56], learning generative models [9,72], meta-learning [58,22,74,32] etc. For most of these examples, input optimization is typically done using gradient descent on the input, i.e., we feed the input through the network and compute some loss, which we minimize by optimizing over the input with gradient descent. While some of these problems might not require differentiating through the entire optimization process, many do (introduced below), and can further slow down training and impose massive memory requirements.

Input optimization has recently been applied to train generative models. Zadeh et al. [72], Bojanowski et al. [9] proposed to train generator networks by jointly optimizing the parameters and the latent variables corresponding to each example. Similarly, optimizing a latent variable to make the corresponding output match a target image is common in decoder-only models like GANs to get correspondences [10,59], and has been found useful to stabilize GAN training [71]. However, in all of these cases, the input is optimized for just a few (mostly 1) iterations. In this work, we present a generative model, where we optimize and find the optimal latent code for each image at each training step. Additionally, Bora et al. [10], Chang et al. [13] showed that we can take a pretrained generative model and use it as a prior to solve for the likely solutions to inverse problems by optimizing on the input space of the generative model (i.e., unsupervised inverse problem solving). Furthermore, Diamond et al. [15], Gilton et al. [25], Gregor and LeCun [31] have shown that networks can also be trained to solve specific inverse problems by effectively unrolling the optimization procedure and iteratively updating the input. We demonstrate our approach in the unsupervised setting as in Bora et al. [10], Chang et al. [13], but also show flexible extension of our framework to train implicit models for supervised inverse problem solving.

Another crucial application of input optimization is to find adversarial examples [64] [28]. This manifests as optimizing an objective that incentivizes an incorrect prediction by the classifier, while constraining the input to be within a bounded region of the original input. Many attempts have been made on the defense side [57,55,65,69]. The most successful strategy thus far has been adversarial training with a projected gradient descent (PGD) adversary [53] which involves training the network on the adversarial examples computed using PGD online during training. We show that our joint optimization approach can be easily applied to this setting, allowing us to train implicit models to perform competitively with PGD in guaranteeing adversarial robustness, but at much faster speeds.

While the examples above were illustrated with non-convex networks, attempts have also been made to design networks whose output is a convex function of the input [2]. This allows one to use more sophisticated optimization algorithms, but usually at a heavy cost of model capacity. They have been demonstrated to work in a variety of problems including multi-label prediction, image completion [2], learning stable dynamical systems [44] and optimal transport mappings [54], MPC [12], etc.

3 Joint inference and input optimization in DEQs

Here we present our main methodological contribution, which sets up an augmented DEQ that jointly performs inference and input optimization over an existing DEQ model. We first define the base DEQ model, and then illustrate a joint approach that simultaneously finds its forward fixed point and optimizes over its inputs. We discuss several methodological details and extensions.

3.1 Preliminaries: DEQ-based models

To begin with, we recall the deep equilibrium model setting from Bai et al. [5], but with the notation slightly adapted to better align with its usage in this paper. Specifically, we consider an input-injected layer $f_\theta : \mathcal{Z} \times \mathcal{X} \to \mathcal{Z}$ where $\mathcal{Z}$ denotes the hidden state of the network, $\mathcal{X}$ denotes the input space, and $\theta$ denotes the parameters of the layer. Given an input $x \in \mathcal{X}$, computing the forward pass in a DEQ model involves finding a fixed point $z^*(x) \in \mathcal{Z}$, such that

$$z^*_\theta(x) = f_\theta(z^*_\theta(x), x),$$

which (under proper stability conditions) corresponds to the “infinite depth” limit of repeatedly applying the $f_\theta$ function. We emphasize that under this setting, we can effectively think of $z^*_\theta$ itself
as the implicitly defined network (which thus is also parameterized by \( \theta \)), and one can differentiate through this “network” via the implicit function theorem \[8, 47\]. The fixed point of a DEQ could be computed via the simple forward iteration

\[
z^+ := f_\theta(z, x)
\]

starting at some arbitrary initial value of \( z \) (typically 0). However, in practice DEQ models will typically compute this fixed point not simply by iterating the function \( f_\theta \), but by using a more accelerated root-finding or fixed-point approach such as Broyden’s method \[11\] or Anderson acceleration \[4, 66\]. Further, although little can be said about e.g., the existence or uniqueness of these fixed points in general (though there do exist restrictive settings where this is possible \[68, 59, 23\]), in practice a wide suite of techniques have been used to ensure that such fixed points exist, can be found using relatively few function evaluations, and are able to competitively model large-scale tasks \[5, 6\].

### 3.2 Joint inference and input optimization

Now we consider the setting of performing input optimization for such a DEQ model. Specifically, consider the task of attempting to optimize the input \( x \in X \) to minimize some loss \( \ell : Z \times Y \to \mathbb{R}_+ \).

\[
\minimize_{x \in X} \ell(z^*_\theta(x), y)
\]

where \( y \in Y \) represents the data point. To solve this, we typically perform such an optimization via e.g., gradient descent, which repeats the update

\[
x^+ := x - \alpha \left( \frac{\partial \ell(z^*_\theta(x), y)}{\partial z} \right)^\top
\]

until convergence, where we use term \( z^* \) alone to denote the fixed output of the network \( z^*_\theta \) (i.e., just a fixed output rather than a function). Using the chain rule and the implicit function theorem, we can further expand update \[6\] using the following analytical expression of the gradient:

\[
\frac{\partial (z^*_\theta(x), y)}{\partial x} = \frac{\partial \ell(z^*, y)}{\partial z^*} \frac{\partial z^*}{\partial x} = \frac{\partial \ell(z^*, y)}{\partial z^*} \left( I - \frac{\partial f_\theta(z^*, x)}{\partial z} \right)^{-\top} \frac{\partial f_\theta(z^*, x)}{\partial x}
\]

Thinking about \( z^*_\theta \) as an implicit function of \( x \) permits us to combine the fixed-point equation in Eq. \[4\] (on \( z \)) with this input optimization update (on \( x \)), thus performing a joint forward update:

\[
\begin{bmatrix} z^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} x \\ x - \alpha \left( \frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \left( I - \frac{\partial f_\theta(z, x)}{\partial z} \right)^{-\top} \frac{\partial f_\theta(z, x)}{\partial x} \end{bmatrix}
\]

It should be apparent that, if both iterates converge, then they have converged to a simultaneous fixed point \( z^* \) and an optimal \( x^* \) value for the optimization problem \[5\]. However, simply performing this update can still be inefficient, because computing the inverse Jacobian in Eq. \[7\] is expensive and typically computed via an iterative update – namely, we would first compute the variable

\[
\mu := \left( I - \frac{\partial f_\theta(z, x)}{\partial z} \right)^{-\top} \frac{\partial \ell(z, y)}{\partial z}
\]

via the following iteration (i.e., a Richardson iteration \[60\]):

\[
\mu^+ := \left( \frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \mu + \left( \frac{\partial \ell(z, y)}{\partial z} \right)^\top
\]

Therefore, to efficiently solve the joint inference and input optimization problem, we propose combining all three iterative procedures into the update

\[
\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f_\theta(z, x) \\ \mu + \left( \frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \mu \left( x - \alpha \left( \frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \mu + \left( \frac{\partial \ell(z, y)}{\partial z} \right)^\top \right) \\ x \end{bmatrix}
\]
Figure 1: Left: Performing each gradient update on DEQ inputs requires a fixed point computation in the forward and backward pass. Right: Solving the 3 fixed points simultaneously as an “augmented” DEQ where the targets $y$ act as input and the function $F$ represents the joint fixed point updates in Eq. 11 we can apply in several settings that we discuss below. This is in contrast to prior works where $f_\theta$ is an explicit deep neural network, where the model forward-pass and optimization processes are disentangled and have to be dealt with separately. We illustrated this in Figure 1 where the figure on the left shows the input optimization naively performed using gradient descent in DEQs v/s the figure on the right which shows the joint updates performed using the augmented DEQ network.

As a final note, we mention that, in practice, just as the gradient-descent update has a step size $\alpha$, it is often beneficial to add similar “damping” step sizes to the other updates as well. This leads to the full iteration over the augmented DEQ

$$
\begin{bmatrix}
z^+ \\
\mu^+ \\
x^+
\end{bmatrix} := 
\begin{bmatrix}
(1 - \alpha_z)z + \alpha_z f(z, x) \\
(1 - \alpha_\mu) \mu + \alpha_\mu \left( \frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \mu + \left( \frac{\partial \ell(z, y)}{\partial z} \right)^\top \\
x - \alpha_x \left( \frac{\partial f_\theta(z, x)}{\partial x} \right)^\top \mu
\end{bmatrix}
$$

Finally, in order to speed up convergence, as is common in DEQ models, we apply a more involved fixed point solver, such as Anderson acceleration, on top of this naive iteration. We analyze the effect of these different root-finding approaches in the Appendix.

Notes on Convergence Our treatment of the above system as an augmented DEQ allows us to borrow results from [68, 7] to ensure convergence of the fixed point iteration. Specifically, if we assume the joint Jacobian of the fixed point iterations we describe are strongly monotone with smoothness parameter $m$ and Lipschitz constant $L$, then by standard arguments (see e.g., Section 5.1 of [62]), the fixed point iteration with step size $\alpha < m/L^2$ will converge. Note that these are substantially weaker rates and constants than required for typical gradient descent or the minimization of locally convex function because the coupling between the three fixed point iterations introduce cross-terms in the joint Jacobian.

Due to these cross-terms, going from the strong monotonicity assumption on the joint fixed point iterations to specific assumptions on $f_\theta$ and $\ell$ is less straightforward. However, empirically, we observed that as long as the step sizes $\alpha$’s were kept reasonably small and the functions $f_\theta$ and $\ell$ were designed appropriately (e.g. $\ell$ respecting notions of local convexity and $f_\theta$ with Jacobian eigenvalues less than 1, etc.) the fixed point iterations converged reliably.

3.3 Interpretation as a primal-dual optimization

While the problem above was introduced as an input-optimization problem, its formulation as a joint optimization problem in the augmented DEQ system (11) can also be viewed as a constrained
optimization problem where the DEQ fixed-point conditions are treated as constraints,
\[
\min_{x,z} \ell(z, y) \quad \text{subject to} \quad z = f_\theta(z, x)
\]
which yield the Lagrangian
\[
\min_{x,z} \max_{\mu} \mathcal{L}(x, z, \mu) \equiv \ell(z, y) + \mu^T (f_\theta(z, x) - z)
\]
and the corresponding KKT conditions
\[
\begin{bmatrix}
\frac{\partial f_\theta(z, x) - z}{\partial \mu(x, z, \mu)} \\
\frac{\partial \mathcal{L}(x, z, \mu)}{\partial x}
\end{bmatrix} = \begin{bmatrix}
\frac{\partial \ell(z, y)}{\partial z} + \mu^T \left( \frac{\partial f_\theta(z, x)}{\partial z} - I \right) \\
\mu^T \frac{\partial f_\theta(z, x)}{\partial z}
\end{bmatrix} = \begin{bmatrix}
0 \\
0
\end{bmatrix} \in \mathbb{R}^{2n+d}
\]
where \(\mu\) are the dual variables corresponding to the equality constraints. Rearranging the terms in the KKT conditions of the above problem, introducing the step size parameters \(\alpha\)'s and treating it as a fixed point iteration gives us the updates in Eq. (11). Indeed, performing such iterations is a variation of the classical primal-dual gradient method for solving equality-constrained optimization problems [20, 34, 17].

### 3.4 Outer Optimization (Backward Pass)

A notable advantage of formulating the entire joint inference and input optimization problem as an augmented DEQ \(v^*_\theta(y)\) is that it allows us to abstract away the detailed function of \(v\), and simply train parameters \(\theta\) of this joint process as an outer optimization problem:
\[
\min_{\theta} \ell_{\text{outer}}(v^*_\theta(y), y)
\]
where \(\ell_{\text{outer}} : X \times \mathcal{Z} \times \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+\).

Given the solutions \(x^*, z^*\), to the inner problem, computing updates to \(\theta\) in the outer optimization problem is equivalent to the backward pass of the augmented DEQ and correspondingly is optimized using standard stochastic gradient optimizers like Adam [42]. Thus, as with any other DEQ model, we assume the inner problem was solved to a fixed point, and apply the implicit function theorem to compute the gradients w.r.t the augmented system [11]. This gives us a constant memory backward pass which is invariant to the underlying optimizer used to solve the inner problem.
\[
\frac{\partial \ell_{\text{outer}}(v^*_\theta, y)}{\partial \theta} = \frac{\partial \ell_{\text{outer}}(y, v^*)}{\partial v^*} \frac{\partial v^*_\theta}{\partial \theta} = -\frac{\partial \ell_{\text{outer}}(y, v^*)}{\partial v^*} \left( \frac{\partial K_\theta(v^*)}{\partial v^*} \right)^{-1} \frac{\partial K_\theta(v^*)}{\partial \theta}
\]
where \(v = [x, z, \mu]^\top\) and \(K_\theta(v) = 0\) represents the KKT conditions from [14]. As with the original DEQ, instead of computing the above expression explicitly, we first solve the following linear system to compute \(u\) and then substitute it back in the equation above to obtain the full gradient:
\[
u^\top = -\frac{\partial \ell_{\text{outer}}(y, v^*)}{\partial v^*} \left( \frac{\partial K_\theta(v)}{\partial v} \right)^{-1} \left( \frac{\partial K_\theta(v)}{\partial v} \right)^\top u = 0
\]

Although we can train any joint DEQ in this manner, doing so in practice (e.g., via automatic differentiation), will require double backpropagation, because the definition of \(K_\theta(v)\) above already includes vector-Jacobian products, and this expression will require differentiating again. However, in the case that \(\ell_{\text{outer}}\) is the same as \(\ell\) (or in fact where it is the negation of \(\ell\)), then there exists a substantial simplification of the outer optimization gradient. These cases are indeed quite common, as we may want e.g., to train the parameters of a generative model to minimize the same reconstruction error that we attempt to optimize via the latent variable; or in the case of adversarial examples, the inner adversarial optimization is precisely the negation of the outer objective.

In these cases, we have that
\[
\ell_{\text{outer}}(y, v^*_\theta(y)) = \ell(y, z^*_\theta(y))
\]
so we have that
\[
\frac{\partial \ell_{\text{outer}}(y, v^*_\theta(y))}{\partial \theta} = \frac{\partial \ell(y, z^*)}{\partial z^*} \left( I - \frac{\partial f_\theta(z^*, x^*)}{\partial z^*} \right)^{-1} \frac{\partial f_\theta(z^*, x^*)}{\partial \theta} = (\mu^*)^T \frac{\partial f_\theta(z^*, x^*)}{\partial \theta}
\]
In other words, we can compute the exact needed derivatives with respect to \(\theta\) by simply re-using the converged solution \(v^*\), without the need to double backpropagate through the KKT system. The same considerations, but just negative, apply to the case where \(\ell_{\text{outer}} = -\ell\).
4 Experiments

As our approach provides a generic framework for joint modeling of an implicit network’s forward dynamics and the “inner” optimization over the input space, we demonstrate its effectiveness and generalizability on 4 different types of problems that are popular areas of research in machine learning: generative modeling [43, 27], inverse problems [10, 25, 31], adversarial training [69, 53] and gradient based meta-learning [22, 58] (results for the latter are in the appendix). In all cases, we show that our joint inference and input optimization (JIIO) provides significant speedups over projected gradient descent applied to DEQ models and that the models trained using JIIO achieve results competitive with standard baselines. In all of our experiments, the design of our model layer \( f_\theta \) follows from the prior work on multiscale deep equilibrium (MDEQ) models [6] that have been applied on large-scale computer vision tasks, and where we replace all occurrences of batch normalization [35] with group normalization [70] in order to ensure the inner optimization can be done independently for each instance in the batch. We elaborate on the details of the choice of other hyperparameters and design decisions of our model (such as the damping parameters \( \alpha \) in the update step (11)), as well as that of the datasets for each task in the Appendix.

We introduce below each problem instantiation, how they fit into our methodology described in Sec. 3.2, and the result of applying the JIIO framework compared to the alternative methods trained in similar settings. Overall, our results provide strong evidence of benefits of performing joint optimizations on implicit models, thus opening new opportunities for future research in this direction.

4.1 Generative Modeling

We study the application of JIIO to learning decoder-only generative models that compute the latent representations by directly minimizing the reconstruction loss [72, 9]; i.e., given a decoder network \( D \), the latent representation \( x \) of a sample \( y \) (e.g., an image) is \( x = \min_x \| D(x) - y \|_2^2 \). Moreover, instead of placing explicit regularizations on the latent space \( X \) (as in VAEs), we follow [24] to directly train the decoder for reconstruction (and then after training, we fit the resulting latents using a simple density model, post-hoc, for sampling). Formally, given sample data \( y_1, \ldots, y_n \) (e.g., \( n \) images), the generative model we study takes the following form:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^n \| y_i - h_\theta(z^*_i) \|^2 \\
\text{subject to} & \quad x^*_i, z^*_i = \underset{x, z}{\text{argmin}} \| y_i - h_\theta(z) \|^2, \quad i = 1, \ldots, n
\end{align*}
\]

where \( h_\theta \) is a final output layer that transform the activations \( z^* \) to the target dimensionality. We train the MDEQ-based \( f_\theta \) with the JIIO framework on standard 64x64 cropped images from CelebA dataset, which consists of 202,599 images. We use the standard train-val-test split as used in Liu et al. [51] and train the model for 50k training steps. We use Fréchet inception distance (FID) [33] to measure the quality of the sampling and test-time reconstruction of the implicit model trained with JIIO and compare with the other standard baselines such as VAEs [43]. The results are shown in Table 1. JIIO-MDEQ refers to the MDEQ model trained using our setup with 40 JIIO iterations in the inner loop during training (and tested with 100 iterations). MDEQ-VAE refers to an equivalent MDEQ model but with an encoder and a decoder trained as a VAE. We observe that our model’s generation quality is competitive with, or better than, each of these encoder-decoder based approaches. Moreover, with the joint optimization proposed, JIIO-MDEQ achieves the best reconstruction quality.

We additionally apply JIIO on pre-trained MDEQ-VAEs (i.e., train an MDEQ-based VAE as usual on optimizing ELBO [43], and take the decoder out) for test-time image reconstruction. The result (shown in Table 1) suggests that the reconstructions obtained as a result are better even than the original MDEQ-VAE. In other words, JIIO can be used with general implicit-mode-based decoders at test time even if the decoder wasn’t trained with JIIO.

---

2This notation differs from the “standard” notation of latent variable models (where the latent variable is typically denoted by \( z \)). However, because \( x, y, z \) all have standard meanings in setting above, we change from the common notation here to be more consistent with the remainder of this paper.
One of the key advantages presented by JIIO is the relative speed of optimization over simply running gradient descent (or its adaptive variants like Adam [41]). Table 2 shows our timing results for one optimization run on a single example for various models (averaged over 200 examples). We observe that performing 40 iterations of projected Adam takes more than 9× the time taken by 40 iterations of JIIO, which we used during training and more than 3.5× the time taken by 100 iterations of JIIO which we use for reconstructions at test time (e.g., to produce the results in Table 1 though both of them lead to similar levels of reconstruction loss). Fig 3 shows the reconstruction loss as it evolves across a single optimization run for an MDEQ model trained with JIIO. This again clearly shows that JIIO converges vastly faster (in terms of wall-clock time) than if we handle the inner optimization separately as in prior works, demonstrating the advantage of joint optimization. However, it’s interesting to note that JIIO optimization seems somewhat unstable (see Fig. 3) and fluctuates more as well. This seems to be an artifact of the specific acceleration scheme we use (see more details in Appendix A.2).

### 4.2 Inverse Problems

We also extend the setup mentioned in section 4.1 directly to inverse problems. These problems, specifically, can be approached as either an unsupervised or a supervised learning problem, which we discuss separately in this section. To demonstrate how JIIO can be applied, we will be using image inpainting and image denoising as example inverse problem tasks, which was extensively studied in prior works like Chang et al. [13], Gilton et al. [26]. For the inpainting task, we randomly mask a 20x20 window from the image and train the model to adequately fill the missing pixels based on the surrounding image context. For the image denoising tasks, we add random gaussian noise \( \varepsilon \sim \mathcal{N}(0,\sigma^2I) \) with \( \sigma = 0.2 \) and \( \sigma = 0.4 \), respectively, to all pixels in the image, and train the model to recover the original image. We use the same datasets and train-test setups as in the generative modeling experiments in Sec. 4.1.
### 4.2.1 Unsupervised inverse problem solving

Bora et al. [10], Chang et al. [13] have showed that we can solve most inverse problems by taking a pre-trained generative model and using that as a prior to solve for the likely solutions to the inverse problems by optimizing on the input space of the generative model. Specifically, given a “generator” network \( G : \mathcal{X} \rightarrow \mathcal{Y} \), mapping from the latent space \( \mathcal{X} \) to an observed space \( \mathcal{Y} \), that models the data generating distribution, they show that one can solve any inverse problem by optimizing the following objective:

\[
\min_{x \in \mathcal{X}} \| \hat{y} - AG(x) \|_2^2.
\]

(21)

where \( \hat{y} = Ay \in \mathcal{Y} \) represents the corrupted data point, \( y \in \mathcal{Y} \) is the uncorrupted data and \( A : \mathcal{Y} \rightarrow \mathcal{Y} \) denotes the measurement matrix that defines the specific type of inverse problem that we try to solve (e.g., for image inpainting, it would be a mask with the missing regions filled in with zeros. For deblurring, it would be a convolution with a gaussian blur operator etc.). They call it unsupervised inverse problem solving. Likewise, we can use the pre-trained generator from section 4.1 to solve most inverse problems by simply solving a slightly modified version of the inner problem in (29):

\[
\min_{x,z} \left\{ z = f_\theta(z,x) \right\} \| \hat{y} - Ah_\theta(z) \|_2^2 \quad \text{(22)}
\]

In table 4, the unsupervised results for VAE and MDEQ-VAE generators are obtained by optimizing (21) using Adam for 40 iterations, while for the JIIO trained models, we optimize (22) with 100 JIIO iterations. JIIO-MDEQ-40 and JIIO-MDEQ-100 refer to JIIO-MDEQ models trained with 40 and 100 inner-loop iterations respectively. The results in Table 4 show that on all 3 problems, JIIO trained generators produce results comparable to the VAE and MDEQ-VAE generators. Moreover, as shown in section 4.1, JIIO also converges much faster than Adam applied to a MDEQ-VAE generator.

### 4.2.2 Supervised Inverse problem solving

While the unsupervised inverse problem solving works reasonably well, we can also learn models to solve specific inverse problems to obtain better performance. Specifically, given uncorrupted data \( y_1, \ldots, y_n \), and the measurement matrix \( A \), we can train a network \( \hat{G}_\theta : \mathcal{Y} \rightarrow \mathcal{Y} \) mapping from the corrupted sample \( \hat{y}_i = Ay_i \) to the uncorrupted sample \( y_i \) by minimizing:

\[
\min_\theta \sum_{i=1}^{n} \| y_i - G_\theta(Ay_i) \|_2^2 \quad \text{(23)}
\]

Now, instead of modeling \( \hat{G}_\theta \) as an explicit network, we could also model it as a solution to the inverse problem in (22) and the resulting parameters can be trained as follows:

\[
\min_\theta \sum_{i=1}^{n} \| y_i - h(z^*_i) \|_2^2
\]

\[
\text{subject to } x^*_i, z^*_i = \arg\min_{x,z : f_\theta(z,x)} \| Ay_i - Ah(z) \|_2^2, \ i = 1, \ldots, n \quad \text{(24)}
\]

As shown in Table 4, this yields models competitive with their autoencoder based counterparts, while being better than all the unsupervised approaches. Each of the baseline models in the supervised section of Table 4 are trained by simply optimizing (23) with the corresponding model replacing \( G_\theta \). However, note that given the models here are trained on specific inverse problems, one would have to train a new model for each new problem as opposed to the unsupervised approach.
Table 5: Comparison of adversarial training approaches on L2 norm perturbations with $\epsilon = 1$. The rows represent the training procedure and the columns represent the testing procedure.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Train (%)</th>
<th>Test (%)</th>
<th>Clean</th>
<th>PGD</th>
<th>JIIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>Clean</td>
<td>99.45 ± 0.03</td>
<td>80.1 ± 1.87</td>
<td>65.88 ± 4.72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PGD</td>
<td>99.18 ± 0.03</td>
<td>96.53 ± 0.05</td>
<td>95.74 ± 0.04</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIIO</td>
<td>99.32 ± 0.09</td>
<td>95.74 ± 0.22</td>
<td>96.63 ± 0.58</td>
<td></td>
</tr>
<tr>
<td>CIFAR</td>
<td>Clean</td>
<td>78.47 ± 0.94</td>
<td>2.38 ± 0.41</td>
<td>3.71 ± 4.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PGD</td>
<td>54.91 ± 1.01</td>
<td>37.4 ± 0.26</td>
<td>36.17 ± 0.55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JIIO</td>
<td>55.54 ± 0.82</td>
<td>37.31 ± 0.67</td>
<td>37.77 ± 0.84</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Adversarial Training

Although the previous two tasks were based on image generation, we note that our approach can be used more generally for input optimization in DEQs and illustrate that by applying it to $\ell_2$ adversarial training on DEQ-based classification models. Specifically, given inputs $x_i, y_i; i = 1, \ldots, n$, adversarial training seeks to optimize the objective

$$\min_{\theta} \sum_{i=1}^{n} \max_{\|\delta\|_2 \leq \epsilon} \ell(h_\theta(x_i + \delta_i), y_i)$$

We apply our setting to a DEQ-based classifier $h_\theta(x) = h_\theta(z^*_\theta(x))$ where $z^*_\theta = f_\theta(z^*_\theta, x)$. In this setting, we embed the iterative optimization over $\delta$ (with projected gradient descent) into the augmented DEQ, and write the problem as

$$\min_{\theta} \sum_{i=1}^{n} \ell(h_\theta(z^*_i), y)$$

subject to $z^*_i, \delta^*_i = \arg\min_{z, \delta: x = f_\theta(z, x + \delta), \|\delta\|_2 \leq \epsilon} -\ell(h_\theta(z), y), i = 1, \ldots, n$

Specifically we train MDEQ models on CIFAR10 [48] and MNIST [50] using adversarial training against L2 attacks with $\epsilon = 1$ for 20 epochs and 10 epochs respectively using the standard train-val-test splits. Table 5 shows the robust and clean accuracy of models trained using PGD and JIIO adversarial training and vanilla training. We find that models trained using JIIO have comparable robust and clean accuracy on both datasets. Furthermore, when tested on models trained without adversarial training, we observe that JIIO serves as a comparable attack method to PGD. Table 4-2 shows the time taken to find the adversarial example for a single image of MNIST using 20 iterations of PGD and 80 iterations of JIIO. We again observe more than 3x speedups when using JIIO over using PGD while obtaining competitive robust accuracy. However, note that, unlike previous experiments in generative modeling/inverse problems, performing adversarial training with truncated JIIO iterations would lead to significant reduction in robust performance due to the adversarial setting.

5 Concluding remarks

We present a novel optimization procedure for jointly optimizing over the input and the forward fixed point in DEQ models and show that, for the same class of models, it is $3 - 9 \times$ faster than performing vanilla gradient descent or Adam on the inputs. We also apply this approach to 3 different settings to show its effectiveness: training generative models, solving inverse problems, adversarial training of DEQs and gradient based meta-learning. In the process, we also introduce an entirely new type of decoder only generative model that performs competitively with it’s autoencoder based counterparts.

Despite these features, we note that there is substantial room for future work in these directions. Notably, despite the fact that the augmented joint inference and input optimization DEQ can embed both processes in a “single” DEQ model, in practice these joint models take substantially more iterations to converge as well (often in the range of 50-100) than traditional DEQs (often in 10-20 iterations), and correspondingly often use larger memory caches within methods like Anderson acceleration. Thus, while we make the statement that these augmented DEQs are “just” another DEQ, this relative difficulty in finding a fixed point likely adds challenges to training the underlying models. Thus, despite ongoing work in improving the inference time of typical DEQ models, there is substantial room for improvement here in making these joint models truly efficient.
6 Acknowledgements

Swaminathan Gurumurthy and Shaojie Bai are supported by a grant from the Bosch Center for Artificial Intelligence. We further thank Zhichun Huang and Taylor Howell for their help with some implementations and for various brainstorming sessions and feedback throughout the course of this project. We would also like to thank Vladlen Koltun for brainstorming ideas especially in the initial stages of the project.

References


[60] L. F. Richardson. Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 210(459-470):307–357, 1911.


