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 $G : \mathcal{Z} \to \mathcal{X}$ 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])

$$\underset{z \in \mathcal{Z}}{\text{minimize}} \;\; \|x - G_\theta(z)\|_2^2. \tag{1}$$

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

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

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

$\ell : \mathcal{Y} \to \mathbb{R}_+$, and a sample $x \in \mathcal{X}$, we want to solve

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}} \; \ell(g(x + \delta)). \tag{2}$$

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 adverarial 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 $z^\star$. 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, 39], 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 incentivices 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, 37, 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 it's 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 θ 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 $z^\star(x) \in \mathcal{Z}$, such that

$$z^\star_\theta(x) = f_\theta(z^\star_\theta(x), x), \tag{3}$$

which (under proper stability conditions) corresponds to the "infinite depth" limit of repeatedly applying the f_θ function. We emphasize that under this setting, we can effectively think of $z^\star_\theta$ *itself* as the implicitly defined network (which thus is also parameterized by θ), 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

$$z^+ := f_\theta(z, x) \tag{4}$$

starting at some artibrary initial value of z (typically 0). However, in practice DEQ models will typically compute this fixed point not simply by iterating the function f_θ, 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 $x \in \mathcal{X}$ to minimize some loss $\ell : \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+$.

$$\underset{x \in \mathcal{X}}{\text{minimize}} \;\; \ell(z^\star_\theta(x), y) \tag{5}$$

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

$$x^+ := x - \alpha \left(\frac{\partial \ell(z^\star_\theta(x), y)}{\partial x} \right)^\top \tag{6}$$

until convergence, where we use term $z^\star$ alone to denote the fixed output of the network $z^\star_\theta$ (i.e., just as 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:

$$\frac{\partial \ell(z^\star_\theta(x), y)}{\partial x} = \frac{\partial \ell(z^\star, y)}{\partial z^\star} \frac{\partial z^\star_\theta(x)}{\partial x} = \frac{\partial \ell(z^\star, y)}{\partial z^\star} \left(I - \frac{\partial f_\theta(z^\star, x)}{z^\star} \right)^{-\top} \frac{\partial f_\theta(z^\star, x)}{\partial x} \tag{7}$$

Thinking about $z^\star_\theta$ 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:

$$\begin{bmatrix} z^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f_\theta(z, x) \\ x - \alpha \big(\frac{\partial f_\theta(z,x)}{\partial x} \big)^\top \big(I - \frac{\partial f_\theta(z,x)}{\partial z} \big)^{-\top} \big(\frac{\partial \ell(z,y)}{\partial z} \big)^\top \end{bmatrix} \tag{8}$$

It should be apparent that, if both iterates converge, then they have converged to a simultaneous fixed point $z^\star$ and an optimal $x^\star$ 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 $\mu = \big(I - \frac{\partial f_\theta(z,x)}{\partial z} \big)^{-\top} \big(\frac{\partial \ell(z,y)}{\partial z} \big)^\top$ via the following iteration (i.e., a Richardson iteration [60]):

$$\mu^+ := \left(\frac{\partial f_\theta(z, x)}{\partial z} \right)^\top \mu + \left(\frac{\partial \ell(z, y)}{\partial z} \right)^\top . \tag{9}$$

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

$$\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} f(z, x) \\ \big(\frac{\partial f_\theta(z,x)}{\partial z} \big)^\top \mu + \big(\frac{\partial \ell(z,y)}{\partial z} \big)^\top \\ x - \alpha \big(\frac{\partial f_\theta(z,x)}{\partial x} \big)^\top \mu \end{bmatrix} \tag{10}$$

Like Eq. (8), if this joint process converges to a fixed point, then it corresponds to a simultaneous optimum of both the inference and optimization processes. Such a formulation is especially appealing, as the iteration (10) is *itself* just an *augmented DEQ network* $v^\star_\theta(y)$ (i.e., with input injection y) whose forward pass optimizes on a joint inference-optimization space $v = (x, \mu, z)$. Moreover, we can use standard techniques to differentiate through *this* process, though there are also optimizations

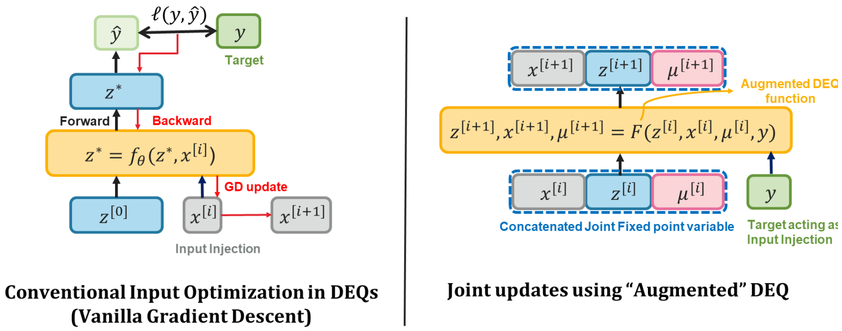

$\ell(y,\hat{y})$
$\hat{y}$
y
Target
z^*
Forward
Backward
$z^* = f_\theta(z^*, x^{[i]})$
GD update
$z^{[0]}$
$x^{[i]}$
$x^{[i+1]}$
Input Injection
Conventional Input Optimization in DEQs (Vanilla Gradient Descent)
$x^{[i+1]}$ $z^{[i+1]}$ $\mu^{[i+1]}$
Augmented DEQ function
$z^{[i+1]}, x^{[i+1]}, \mu^{[i+1]} = F(z^{[i]}, x^{[i]}, \mu^{[i]}, y)$
$x^{[i]}$ $z^{[i]}$ $\mu^{[i]}$
y
Concatenated Joint Fixed point variable
Target acting as Input Injection
Joint updates using "Augmented" DEQ

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_θ 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 α, 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

$$\begin{bmatrix} z^+ \\ \mu^+ \\ x^+ \end{bmatrix} := \begin{bmatrix} (1-\alpha_z)z + \alpha_z f(z,x) \\ (1-\alpha_\mu)\mu + \alpha_\mu \left(\left(\frac{\partial f_\theta(z,x)}{\partial z} \right)^\top \mu + \left(\frac{\partial \ell(z,y)}{\partial z} \right)^\top \right) \\ x - \alpha_x \left(\frac{\partial f_\theta(z,x)}{\partial x} \right)^\top \mu \end{bmatrix} \tag{11}$$

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_θ and ℓ is less straightforward. However, empirically, we observed that as long as the step sizes α's were kept reasonably small and the functions f_θ and ℓ were designed appropriately (e.g ℓ respecting notions of local convexity and f_θ with Jacobian eigenvalues less than 1, etc.) the fixed point iterations converged reliably.

3.3 Iterpretation 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 (10) can also be viewed as a constrained optimization problem where the DEQ fixed-point conditions are treated as constraints,

$$\underset{x,z}{\text{minimize}}\ \ell(z,y)\ \text{ subject to }\ z = f_\theta(z,x) \tag{12}$$

which yield the Lagrangian

$$\underset{x,z}{\text{minimize}}\,\underset{\mu}{\text{maximize}}\ \mathcal{L}(x,z,\mu) \equiv \ell(z,y) + \mu^\top(f_\theta(z,x) - z) \tag{13}$$

and the corresponding KKT conditions

$$\begin{bmatrix} f_\theta(z,x) - z \\ \frac{\partial \mathcal{L}(x,z,\mu)}{\partial z} \\ \frac{\partial \mathcal{L}(x,z,\mu)}{\partial x} \end{bmatrix} = \begin{bmatrix} f_\theta(z,x) - z \\ \frac{\partial \ell(z,y)}{\partial z} + \mu^\top(\frac{\partial f_\theta(z,x)}{\partial z} - I) \\ \mu^\top \frac{\partial f_\theta(z,x)}{\partial x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \in \mathbb{R}^{2n+d} \tag{14}$$

where μ 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 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 $v^\star_\theta(y)$ is that it allows us to abstract away the detailed function of v, and simply train parameters θ of this joint process as an outer optimization problem:

$$\underset{\theta}{\text{minimize}}\ \ell^{\text{outer}}(v^\star_\theta(y), y) \tag{15}$$

where $\ell^{\text{outer}} : \mathcal{X} \times \mathcal{Z} \times \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+$

Given the solutions $x^\star, z^\star$, to the inner problem, computing updates to θ 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.

$$\frac{\partial \ell^{\text{outer}}(v^\star_\theta, y)}{\partial \theta} = \frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\frac{\partial v^\star_\theta}{\partial \theta} = -\frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\left(\frac{\partial K_\theta(v^\star)}{\partial v^\star}\right)^{-1}\frac{\partial K_\theta(v^\star)}{\partial \theta} \tag{16}$$

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,

$$u^\top = -\frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}\left(\frac{\partial K_\theta(v)}{\partial v}\right)^{-1} \iff \frac{\partial \ell^{\text{outer}}(y, v^\star)}{\partial v^\star}^\top + \left(\frac{\partial K_\theta(v)}{\partial v}\right)^\top u = 0 \tag{17}$$

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 ℓ^{outer} is the *same* as ℓ (or in fact where it is the negation of ℓ), 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

$$\ell^{\text{outer}}(y, v^\star_\theta(y)) = \ell(y, z^\star_\theta(y)) \tag{18}$$

so we have that

$$\frac{\partial \ell^{\text{outer}}(y, v^\star_\theta(y))}{\partial \theta} = \frac{\partial \ell(y, z^\star)}{\partial z^\star}\left(I - \frac{\partial f_\theta(z^\star, x^\star)}{\partial z^\star}\right)^{-1}\frac{\partial f_\theta(z^\star, x^\star)}{\partial \theta} = (\mu^\star)^\top\frac{\partial f_\theta(z^\star, x^\star)}{\partial \theta} \tag{19}$$

In other words, we can compute the exact needed derivatives with respect to θ by simply *re-using* the converged solution $v^\star$, 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_θ 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 α 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 $x = \min_{x \in \mathcal{X}} \|D(x) - y\|_2^2$.[2] Moreover, instead of placing explicit regularizations on the latent space $\mathcal{X}$ (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:

$$\begin{aligned} \underset{\theta}{\text{minimize}} \quad & \sum_{i=1}^{n} \|y_i - h_\theta(z_i^\star)\|^2 \\ \text{subject to} \quad & x_i^\star, z_i^\star = \underset{x, z: z = f_\theta(z, x)}{\operatorname{argmin}} \; \|y_i - h_\theta(z)\|^2, \; i = 1, \ldots, n \end{aligned} \tag{20}$$

where h_θ is a final output layer that transform the activations $z^\star$ to the target dimensionality. We train the MDEQ-based f_θ with the JIIO framework on standard 64×64 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.

[2] This 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.

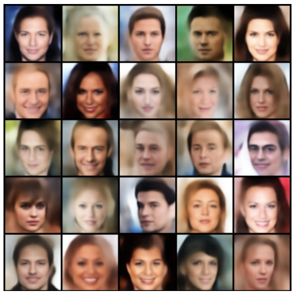


Figure 2: Samples generated with JIIO on a small MDEQ network.

Model	Generation	Reconstruction
VAE [43]	48.12	39.12
RAE [24]	**40.96**	36.01
MDEQ-VAE	57.15	45.81
MDEQ-VAE (w/ JIIO)	-	42.36
JIIO-MDEQ	46.82	**32.52**

Table 1: Comparison of FID scores attained by standard generative models with our method, which performs joint optimization. We use 40 solver iterations (for the augmented DEQ) to train the JIIO model reported in this table.

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 $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$ 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.

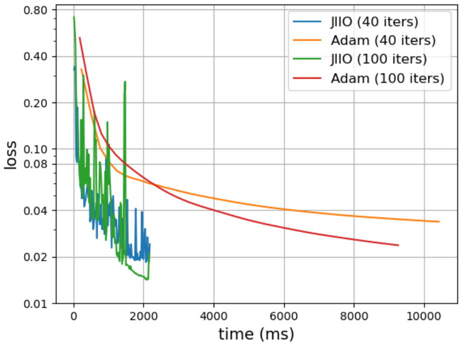


Figure 3: Cost changing with time for Adam v/s JIIO optimization. Tested on models trained with 40 and 100 JIIO iterations respectively

Model	time taken (ms)
PGD : 20 iters	4360
JIIO : 80 iters	1401

Table 2: Time taken to compute adversarial example of a MDEQ model on MNIST

Model	time taken (ms)
Adam : 40 iters	7659
JIIO : 40 iters	862
JIIO : 100 iters	2156

Table 3: Time taken to perform JIIO optimization v/s Adam in the generative modeling/inverse problem experiments

Task	**Model**	**Inpainting**	**Denoising** ($\sigma = 0.2$)	**Denoising** ($\sigma = 0.4$)
Supervised	AE	17.9	18.72	18.32
	MDEQ-AE	17.06	18.58	18.49
	JIIO-MDEQ-100	16.90	18.22	17.89
Unsupervised	VAE (Adam) [10]	15.34	15.31	15.24
	MDEQ-VAE (Adam)	16.62	16.96	16.87
	JIIO-MDEQ-40	15.88	17.08	16.03
	JIIO-MDEQ-100	15.87	17.86	17.55

Table 4: Comparison of Median PSNR values for supervised and unsupervised inverse problem solving approaches. The top 3 rows show models that are trained for the specific inverse problem and the latter 5 show pre-trained generative models re-purposed for solving inverse problems

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:

$$\underset{x \in \mathcal{X}}{\text{minimize}} \;\; \|\hat{y} - AG(x)\|_2^2. \tag{21}$$

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):

$$\underset{x,z:z=f_\theta(z,x)}{\text{minimize}} \;\; \|\hat{y} - Ah_\theta(z)\|^2 \tag{22}$$

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 $G_\theta : \mathcal{Y} \rightarrow \mathcal{Y}$ mapping from the corrupted sample $\hat{y}_i = Ay_i$ to the uncorrupted sample y_i by minimizing:

$$\underset{\theta}{\text{minimize}} \;\; \sum_{i=1}^{n} \|y_i - G_\theta(Ay_i)\|^2 \tag{23}$$

Now, instead of modeling G_θ 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:

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \;\; \sum_{i=1}^{n} \|y_i - h(z_i^\star)\|^2 \\ &\text{subject to} \;\; x_i^\star, z_i^\star = \underset{x,z:z=f_\theta(z,x)}{\text{argmin}} \;\; \|Ay_i - Ah(z)\|^2, \; i = 1, \ldots, n \end{aligned} \tag{24}$$

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_θ. 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.

Datasets	**Train (↓) Test (→)**	Clean	PGD	JIIO
MNIST	Clean	99.45 ± 0.03	80.1 ± 1.87	65.88 ± 4.72
	PGD	99.18 ± 0.03	96.53 ± 0.05	95.74 ± 0.04
	JIIO	99.32 ± 0.09	95.74 ± 0.22	96.63 ± 0.58
CIFAR	Clean	78.47 ± 0.94	2.38 ± 0.41	3.71 ± 4.01
	PGD	54.91 ± 1.01	37.4 ± 0.26	36.17 ± 0.55
	JIIO	55.54 ± 0.82	37.31 ± 0.67	37.77 ± 0.84

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

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 ℓ_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

$$\underset{\theta}{\text{minimize}} \; \sum_{i=1}^{n} \max_{\|\delta\|_2 \le \epsilon} \ell(h_\theta(x_i + \delta_i), y_i) \tag{25}$$

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

$$\begin{aligned} &\underset{\theta}{\text{minimize}} \; \sum_{i=1}^{n} \ell(h_\theta(z_i^\star), y) \\ &\text{subject to} \;\; z_i^\star, \delta_i^\star = \underset{z,\delta : z = f_\theta(z, x+\delta), \|\delta\|_2 \le \epsilon}{\text{argmin}} -\ell(h_\theta(z), y), \; i = 1, \ldots, n \end{aligned} \tag{26}$$

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 adversarial training, 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 it's 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

[1] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning (ICML)*, pages 136–145, 2017.

[2] B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 06–11 Aug 2017. URL `http://proceedings.mlr.press/v70/amos17b.html`.

[3] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/ba6d843eb4251a4526ce65d1807a9309-Paper.pdf`.

[4] D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[5] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:690–701, 2019.

[6] S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL `https://github.com/locuslab/mdeq`.

[7] S. Bai, V. Koltun, and Z. Kolter. Stabilizing equilibrium models by jacobian regularization. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 554–565. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/bai21b.html`.

[8] K. Binmore and J. Davies. *Calculus: concepts and methods*. Cambridge University Press, 2001.

[9] P. Bojanowski, A. Joulin, D. Lopez-Paz, and A. Szlam. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

[10] A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning (ICML)*, pages 537–546, 2017.

[11] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 1965.

[12] F. Bünning, A. Schalbetter, A. Aboudonia, M. H. de Badyn, P. Heer, and J. Lygeros. Input convex neural networks for building mpc. *ArXiv*, abs/2011.13227, 2020.

[13] J. R. Chang, C.-L. Li, B. Póczos, B. Vijaya Kumar, and A. C. Sankaranarayanan. One network to solve them all — solving linear inverse problems using deep projection models. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5889–5898, 2017. doi: 10.1109/ICCV.2017.627.

[14] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf`.

[15] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041*, 2017.

[16] J. Djolonga and A. Krause. Differentiable learning of submodular models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/192fc044e74dffea144f9ac5dc9f3395-Paper.pdf`.

[17] S. S. Du and W. Hu. Linear convergence of the primal-dual gradient method for convex-concave saddle point problems without strong convexity. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 196–205. PMLR, 2019.

[18] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf`.

[19] L. El Ghaoui, F. Gu, B. Travacca, A. Askari, and A. Y. Tsai. Implicit deep learning. *arXiv preprint arXiv:1908.06315*, 2, 2019.

[20] H. C. Elman and G. H. Golub. Inexact and preconditioned uzawa algorithms for saddle point problems. *SIAM Journal on Numerical Analysis*, 31(6):1645–1661, 1994.

[21] H.-r. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3):197–221, 2009.

[22] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017. URL `http://proceedings.mlr.press/v70/finn17a.html`.

[23] S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Fixed point networks: Implicit depth models with Jacobian-free backprop. *arXiv:2103.12803*, 2021.

[24] P. Ghosh, M. S. Sajjadi, A. Vergari, M. Black, and B. Schölkopf. From variational to deterministic autoencoders. In *International Conference on Learning Representations*, 2020.

[25] D. Gilton, G. Ongie, and R. Willett. Neumann networks for linear inverse problems in imaging. *IEEE Transactions on Computational Imaging*, 6:328–343, 2020. doi: 10.1109/TCI.2019.2948732.

[26] D. Gilton, G. Ongie, and R. Willett. Deep equilibrium architectures for inverse problems in imaging. *arXiv preprint arXiv:2102.07944*, 2021.

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf`.

[28] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

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

[30] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJxgknCcK7`.

[31] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

[32] S. Gurumurthy, S. Kumar, and K. Sycara. Mame: Model-agnostic meta-exploration. In *Conference on Robot Learning*, pages 910–922. PMLR, 2020.

[33] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017.

[34] M. Hong, M. Razaviyayn, and J. Lee. Gradient primal-dual algorithm converges to second-order stationary solution for nonconvex distributed optimization over networks. In *International Conference on Machine Learning*, pages 2009–2018. PMLR, 2018.

[35] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[36] Y. Jeon, M. Lee, and J. Y. Choi. Differentiable forward and backward fixed-point iteration layers. *IEEE Access*, 9:18383–18392, 2021.

[37] H. Kannan, A. Kurakin, and I. Goodfellow. Adversarial logit pairing. *ArXiv*, abs/1803.06373, 2018.

[38] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks, 2019 ieee. In *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2018.

[39] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[40] K. Kawaguchi. On the theory of implicit deep learning: Global convergence with implicit layers. In *International Conference on Learning Representations (ICLR)*, 2021.

[41] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[42] 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`.

[43] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[44] J. Z. Kolter and G. Manek. Learning stable deep dynamics models. In *NeurIPS*, 2019.

[45] J. Z. Kolter and E. Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, 2018.

[46] J. Z. Kolter, D. Duvenaud, and M. Johnson. Deep implicit layers - neural odes, deep equilibirum models, and beyond. 2020. URL `http://implicit-layers-tutorial.org/`.

[47] S. G. Krantz and H. R. Parks. *The implicit function theorem: History, theory, and applications*. Springer, 2012.

[48] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[49] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[51] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[52] C. Lu, J. Chen, C. Li, Q. Wang, and J. Zhu. Implicit normalizing flows. In *International Conference on Learning Representations (ICLR)*, 2021.

[53] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *ArXiv*, abs/1706.06083, 2018.

[54] A. Makkuva, A. Taghvaei, S. Oh, and J. Lee. Optimal transport mapping via input convex neural networks. In *ICML*, 2020.

[55] A. Nichol and J. Schulman. Reptile: a scalable metalearning algorithm. *arXiv: Learning*, 2018.

[56] G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis, and R. Willett. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020.

[57] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2016. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2015.

[58] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/072b030ba126b2f4b2374f342be9ed44-Paper.pdf.

[59] M. Revay, R. Wang, and I. R. Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020.

[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.

[61] Y. Rubanova, R. T. Chen, and D. Duvenaud. Latent ODEs for irregularly-sampled time series. *arXiv:1907.03907*, 2019.

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

[63] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[64] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. 2014.

[65] G. Tao, S. Ma, Y. Liu, and X. Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/b994697479c5716eda77e8e9713e5f0f-Paper.pdf.

[66] H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[67] P.-W. Wang, P. Donti, B. Wilder, and Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/wang19e.html.

[68] E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. In *Neural Information Processing Systems*, 2020.

[69] E. Wong, L. Rice, and J. Z. Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJx040EFvH`.

[70] Y. Wu and K. He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[71] Y. Wu, J. Donahue, D. Balduzzi, K. Simonyan, and T. Lillicrap. {LOGAN}: Latent optimisation for generative adversarial networks, 2020. URL `https://openreview.net/forum?id=rJeU_1SFvr`.

[72] A. Zadeh, Y.-C. Lim, P. P. Liang, and L.-P. Morency. Variational auto-decoder. *arXiv preprint arXiv:1903.00840*, 2019.

[73] J. Zhang, B. O'Donoghue, and S. Boyd. Globally convergent type-i anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020.

[74] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pages 7693–7702. PMLR, 2019.

A Additional Discussions on Optimization Stability and Speed

A.1 Regularization

As was observed in previous work [7, 68], the stability of convergence of a DEQ is directly related to the conditioning of the Jacobian matrix at the equilibrium point. To that end, we primarily adopt the regularization proposed by [7] which upper bounds the implicit model's stability by estimating their trace with the Hutchinson estimator: $\text{tr}(J_z) = \text{tr}\left(\frac{\partial f_\theta(z,x)}{\partial z}\right) = \mathbb{E}_{\epsilon \in \mathcal{N}(0,I)}[\epsilon^\top J_z^\top J_z \epsilon]$. However, our exact implementation is subtly different from the original proposal in that we regularize the Jacobian matrix at a *randomly chosen* iterate along the optimization trajectory $(x^{(k)}, z^{(k)})$ instead of just the last iterate (x^*, z^*). This modification is especially important as the optimization trajectories become long (which is the case for our problems; e.g., which could take >80 iterations), which essentially encourages the Jacobian to be not only stable at the end but also during the root-solving process. Specifically, with this modification, the outer objective of JIIO becomes :

$$\underset{\theta}{\text{minimize}} \;\; \ell^{\text{outer}}(v_\theta^\star(y), y) + \lambda \mathbb{E}_{\epsilon \in \mathcal{N}(0,1)}[\epsilon^\top J_z^\top J_z \epsilon] \tag{27}$$

where λ is the regularization coefficient, and in practice we sample 1 or 2 ϵ's to produce a Monte-Carlo estimation of the expectation term.

A.2 Choice of Acceleration method

We perform ablation experiments on the choice of the acceleration methods for performing the joint optimization. Figure 4 shows various approaches that can be used to accelerate JIIO applied to a reconstruction task on a pre-trained MDEQ-VAE decoder. Broyden's method treats its solution as a solution to a root finding problem on the KKT conditions instead of as a minimization problem and hence, given the non-convex nature of the problem, could end up chasing arbitrary stationary points. The Anderson based approaches treat the problem as a minimization problem and hence are able to perform much better. Specifically, Type-I Anderson mixing is usually more unstable (a phenomenon that had been discussed in prior work like Fang and Saad [21] and Zhang et al. [73]), and yet manages to attain the lowest loss values in 100 JIIO iterations. Type-II Anderson, on the other hand, allows for much smoother optimization process although it plateaus at higher loss values. However, since speed was an important point of consideration for our experiments, we nevertheless went with Anderson Type-I over Type-II and picked the output of the optimizer using a heuristic that traded off between a small KKT residual and a small cost. Overall, we observed that the criterion for this iterate selection can be somewhat flexible but preferably kept consistent between training and testing runs when we perform JIIO.

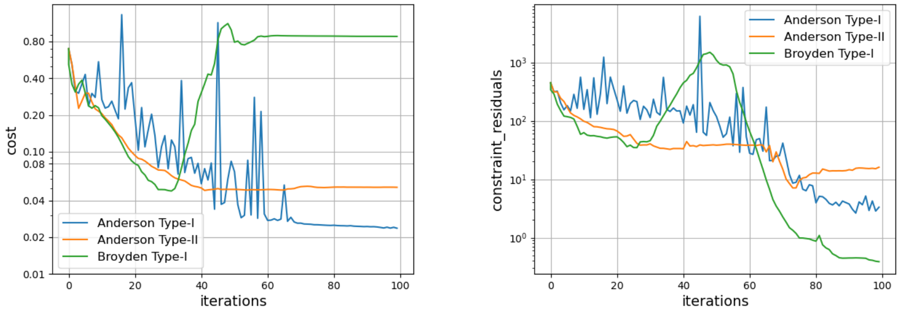


Figure 4: Comparing different acceleration techniques on a MDEQ-VAE decoder

B Supplementary Experiments and Results

B.1 Gradient based Meta-Learning

Gradient (or optimization) based meta-learning defines a bi-level optimization problem where the outer loop learns meta-parameters for a distribution of tasks while the inner loop learns task-specific parameters, typically using a small amount of data. This bi-level structure blends itself naturally into the types of input optimization problems we have been looking at. Specifically, taking few-shot learning as the use case, we are given a collection of tasks $\{\mathcal{T}_i\}_{i=1}^N$, each associated with a dataset $\mathcal{D}_i$, from which we can sample two disjoint sets: $\mathcal{D}_i^{tr} = \{(s_{i,k}^{tr}, y_{i,k}^{tr})\}_{k=1}^K$ and $\mathcal{D}_i^{te}\{(s_{i,k}^{te}, y_{i,k}^{te})\}_{k=1}^K$, where each (s, y) is a data-label pair. Gradient based meta-learning for few shot learning problem can be framed as a bi-level optimization problem with input optimization as the inner loop:

$$\begin{aligned} &\underset{\theta}{\text{minimize}}\ \ell(x_i^\star, h_\theta(x_i^\star, s_{i,k}^{te}), y_{i,k}^{te}), \qquad i = 1, \dots, N \\ &\text{subject to}\ \ x_i^\star = \underset{x_i}{\text{argmin}}\ \ \ell(x_i, h_\theta(x_i, s_{i,k}^{tr}), y_{i,k}^{tr}), \qquad k = 1, \dots, K \end{aligned} \tag{28}$$

where $x_i^\star$ are the task specific parameters inferred during the inner-loop optimization and θ are the meta-parameters. Note that the task specific parameters can be treated as inputs, and thus the inner problem becomes an input optimization problem. In this case, with a DEQ network, the above problem can be modified slightly as:

$$\begin{aligned} &\underset{\theta}{\text{minimize}}\ \ell(x_i^\star, h_\theta(z_{i,k}{}^{\star(te)}), y_{i,k}^{te}), \qquad i = 1, \dots, N \\ &\text{subject to}\ \ x_i^\star, z_{i,k}^\star{}^{(tr)}, z_{i,k}^\star{}^{(te)} = \underset{x_i, z_{i,k}^{(tr/te)} = f_\theta(z_{i,k}^{(tr/te)}, x_i)}{\text{argmin}}\ \ell(x_i, h_\theta(z_{i,k}^{tr}), y_{i,k}^{tr}), \qquad k = 1, \dots, K \end{aligned} \tag{29}$$

Clearly, this modified problem can now to solved using JIIO in the inner loop. Table 6 shows the accuracies obtained by a JIIO-trained DEQ model and various baseline meta-learning approaches like Implicit-MAML [58], MAML [22] and Reptile[55] on the 5-way, 1-shot task on Omniglot [49]. We observe that the JIIO-trained DEQ model achieves comparable accuracy to the baselines. The partially lower accuracy numbers of the DEQ model may be attributed to the fact that we do not differentiate through the fixed point variable $x_i^\star$ while optimizing the outer loop objective. We observed very poor conditioning in our experiments when trying to differentiate through $x_i^\star$ (thus requiring a large number of fixed-point updates for the backward pass and resulting in poor quality gradients) and instead hope to explore that further in future work. Table 7 shows the time taken by JIIO v/s Adam on the DEQ model to perform optimization in the inner loop. Again, we observe that JIIO takes more than 2.7x lesser time to converge, demonstrating the main advantage of using JIIO for input optimization problems with DEQs.

Algorithm/Model	5-way, 1-shot
MAML	98.7
Reptile	97.68
iMAML, GD	99.16
JIIO-DEQ	97.33

Table 6: Accuracy obtained on the 5-way, 1-shot task from the omniglot dataset

Model	time taken (ms)
PGD : 20 iters	28948
JIIO : 100 iters	11836

Table 7: Time taken by JIIO vs. Adam to perform inner-loop optimization in DEQ-based meta-learning tasks

B.2 Adversarial training

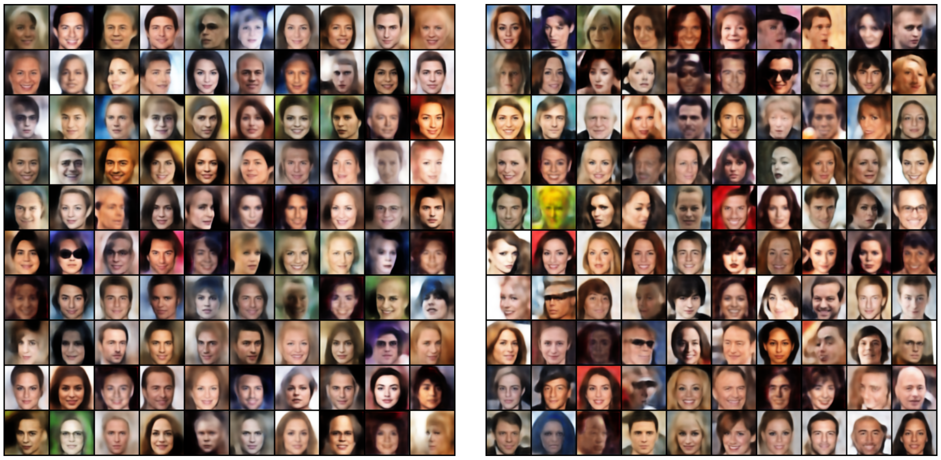
We showed the adversarial training results for L2 perturbations with $\epsilon = 1$ in section 4.3. However, for CIFAR10, it is also common to use $\epsilon = 0.5$ with L2 perturbations. Thus, we also show the robust and clean accuracy of models trained with PGD and JIIO with $\epsilon = 0.5$ for CIFAR10 in Table 8. We again observe that the models trained with JIIO and PGD show similar robust and clean accuracies showing that JIIO is as effective as PGD towards finding adversarial examples while being about 3x faster.

Datasets	**Train (↓) Test (→)**	Clean	PGD	JIIO
CIFAR	Clean	78.47 ± 0.94	4.54 ± 1.33	4.85 ± 2.93
	PGD	68.48 ± 0.81	51.77 ± 0.75	50.14 ± 0.74
	JIIO	67.79 ± 2.33	51.39 ± 0.56	51.25 ± 0.66

Table 8: Comparison of adversarial training approaches on L2 norm perturbations with $\epsilon = 0.5$. The rows represent the training procedure and the columns represent the testing procedure

B.3 Generative Models

We provide additional samples and reconstructions from the JIIO-MDEQ model trained on the CelebA 64x64 images in Figure 5. We also tried training the JIIO-MDEQ model with minor modifications (increasing latent dimensions to 384 and switching the downscaling factor to 4) from the original model and training it with 100 inner-loop JIIO iterations. The reconstructions from the resulting model can be seen in Figure 6. We observe that the reconstructions are blurry and speculate that it's likely due to the squared error loss used in training and the limited capacity of the latent space (which is constrained by the size of the post-hoc density model we wish to learn). Salimans et al. [63] have proposed more suitable losses and representation approaches for high dimensional images which we hope to test in future work to scale up our models to large scale generative modeling problems.



(a) Generated Samples

(b) Reconstructions

Figure 5: Generated Samples and Reconstructions obtained from the JIIO-MDEQ generative model trained in Section 4 on CelebA 64x64 dataset.

B.4 Inverse Problems

We showed results on 3 inverse problems in section 4. Figures 7, 8, 9, 10, 11, 12 show examples from unsupervised and supervised trained JIIO-MDEQ models on the tasks. We see that the unsupervised models approach the performance of the supervised alternatives on most tasks, however, the supervised approaches do tend to perform significantly better on the inpainting task.

C modeling and Training Details

In this section, we discuss various modeling decisions and training/testing details for all the tasks.

C.1 Datasets

Generative modeling and inverse problems. For all the tasks in the generative modeling and inverse problems sections, we work with the CelebA 64×64 dataset and use the standard train-

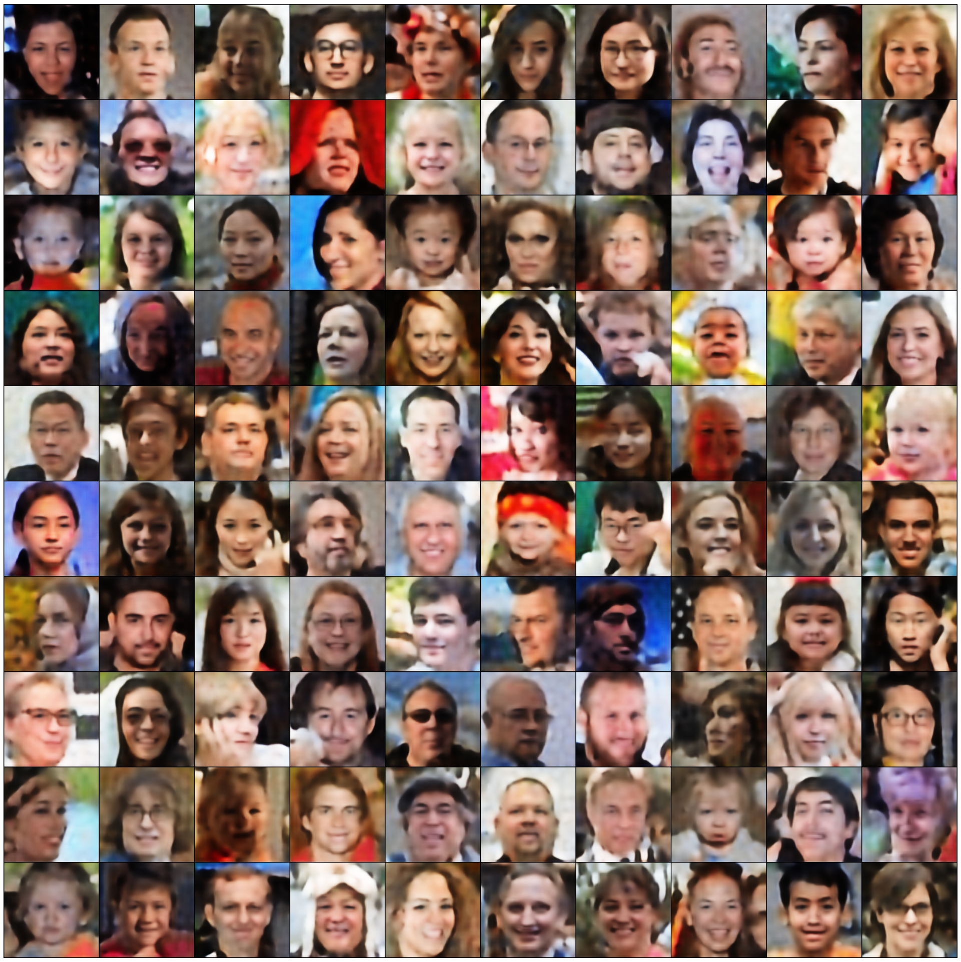


Figure 6: Reconstructions obtained from JIIO-MDEQ trained on 256x256 FFHQ dataset.

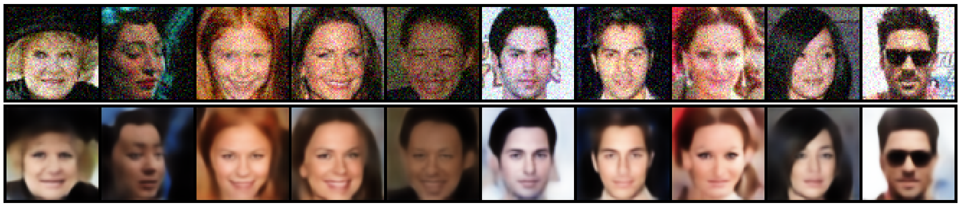


Figure 7: Supervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.2)$: (top) Noisy image; (bottom) Recovered Image

val-test split prescribed in the original paper [51] with 162770 images in the training set, 19867 in the validation set and 19962 in the test set. We follow the procedure in [51] to crop and resize the images to obtain the 64×64 images from the dataset. In section B.3, we also show some preliminary reconstruction results on the FFHQ dataset with 70000 images aligned and cropped as done in [38] and then resized to 256×256. We additionally perform data augmentation on both datasets by performing random horizontal flips on each example.

Adversarial training. We use the well known CIFAR10 [48] and MNIST [50] datasets for our experiments on adversarial training. The CIFAR10 dataset consists of 60k 32×32 color images equally distributed over 10 classes, 50k of which are used for training and 10k for testing. The

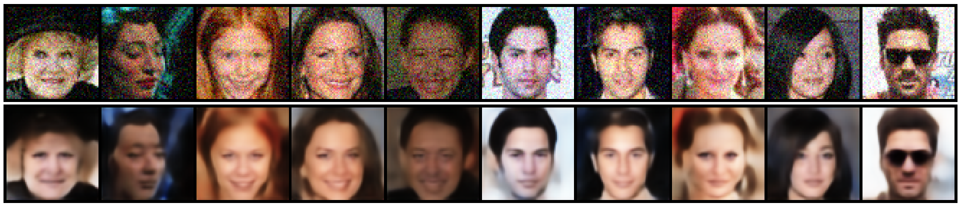


Figure 8: Unsupervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.2)$: (top) Noisy image; (bottom) Recovered Image

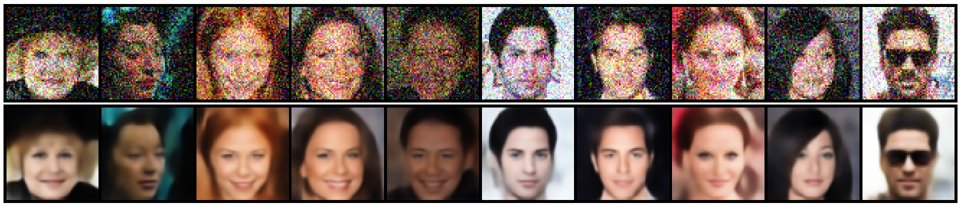


Figure 9: Supervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.4)$: (top) Noisy image; (bottom) Recovered Image

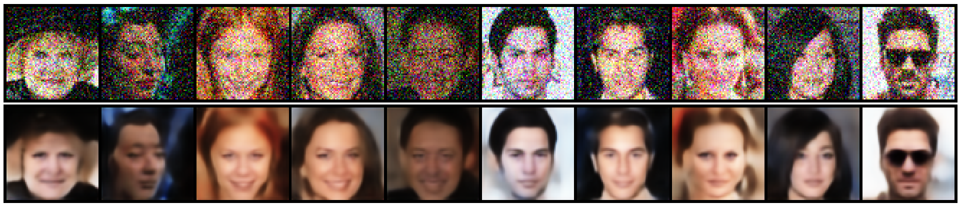


Figure 10: Unsupervised Image Denoising with additive noise sampled from $\mathcal{N}(0, 0.4)$: (top) Noisy image; (bottom) Recovered Image

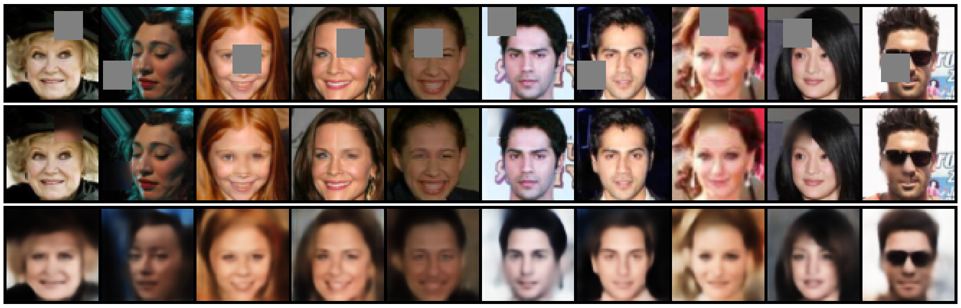


Figure 11: Supervised Image Inpaiting: (top) Incomplete image; (middle) Inpainted image; (bottom) Reconstructed Image

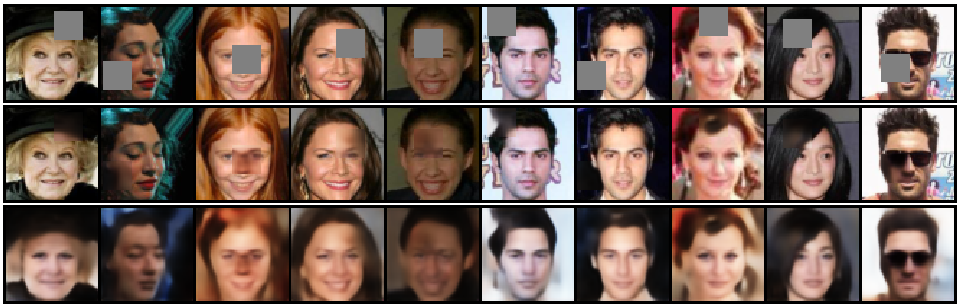


Figure 12: Unsupervised Image Inpaiting: (top) Incomplete image; (middle) Inpainted image; (bottom) Reconstructed Image

MNIST dataset consists of 70k grayscale 28×28 images equally distributed over 10 classes, with 60k used for training and 10k for testing.

Gradient based meta-learning. We use the Omniglot dataset [49] for our experiments with gradient based meta-learning. The Omniglot dataset contains 1623 different handwritten characters from 50 different alphabets, where each image is 28×28 dimensions. We create the tasks and the corresponding meta-training and meta-testing sets using the procedure described in [58].

C.2 Architecture

For all experiments in the paper, we use the multiscale architectures proposed for DEQs (MDEQ) in Bai et al. [6]. In this section, we provide the specific instantiation and hyperparameters of the models used in each of our settings.

Generative modeling and inverse problems. We use a 4 branch MDEQ-LARGE model in all our experiments with a latent vector of size 128 dimensions mapped to the input injection for each branch using a single fully connected layer, while the output layer $h(z)$ merges the outputs of all the branches to obtain the output image as done in the Segmentation Networks in [6]. The hyperparameters of the network are provided in Table 9. Importantly, as mentioned in the experiments section of the paper, we replace all instances of Batch Normalization [35] with Group Normalization [70], in order to make the inner optimization independent for each example. Furthermore, for the MDEQ-VAE and MDEQ-AE baselines, we use a similar architecture for the encoder as well, except that the input injection and output layers are parameterized as in the input injection layers in the MDEQ classification networks in [6]. Additionally, for the baseline VAE, RAE and AE models, we borrow the architectures and code from [24].

For the generative modeling experiments, we additionally fit a density model on the inferred latent vectors inferred on the training set for sampling purposes. In our experiments, we fit a VAE with 4 fully connected layers each in the encoder and the decoder with the VAE latent dimension of size 384. The hidden dimensions for all non-latent layers of the VAE are set to 2096. We additionally also fit a 20 component GMM on the latents of the VAE given the larger latent dimension.

Adversarial training. We adopt a 2 branch MDEQ-SMALL model in all our experiments and use the structure of a standard classification network in [6]. The specific hyperparameters are provided in Table 9. As with experiments above, we replace all Batch Normalization layers with Group Normalization.

Gradient based meta-learning. We adopt a 2 branch MDEQ-SMALL model for our experiments and use the classification network proposed in [5]. As with experiments above, we replace all BatchNorm layers with GroupNorm. We additionally feed the task vector x_i through a fully connected layer and use it as a film layer on top of the first GroupNorm in the residual block.

C.3 Joint Inference and Input Optimization (JIIO)

As opposed to vanilla DEQ models, the joint optimization problem in the augmented DEQ requires a larger number of iterations and the corresponding acceleration techniques require larger memory sizes to converge than the forward inference in DEQ models. Moreover, the additional instability also benefits from additional damping. We accelerate the fixed point iterations using (Type-I) Anderson acceleration [4] in all our experiments. We run the optimization for the maximum number iterations as detailed in the experiments and pick the iterate with the least cost for each example in the batch.

Generative modeling and inverse problems. We use a memory size of 40 and observe that further increasing memory sizes can lead to faster convergence (at the cost of additional GPU memory requirements). Additionally, we damp the fixed point iterations with $\alpha = [\alpha_z, \alpha_\mu, \alpha_x] = [0.8, 0.6, 0.01]$. For experiments involving 100 inner loop iterations, we further reduce $\alpha_x = 0.003$ after 65 iterations in order to obtain finer solutions.

Adversarial training. We use a memory size of 20 for all tasks. For the experiments on CIFAR10 and MNIST, we use $\alpha = [\alpha_z, \alpha_\mu, \alpha_\delta] = [0.8, 0.6, 0.1]$ and $\alpha = [\alpha_z, \alpha_\mu, \alpha_\delta] = [0.8, 0.6, 0.6]$ respectively. Additionally, for MNIST, we reduce $\alpha_\delta = 0.2$ after 65 iterations. After each update, we additionally project the iterates δ onto an L2 ball with radius $\epsilon = 1$ in order to ensure the perturbations stay inside the L2 ball around the example.

Gradient based meta-learning. We use a memory size of 10 for this task and set $\alpha = [\alpha_z, \alpha_\mu, \alpha_x] = [0.8, 0.6, 0.04]$. As with the previous experiments we reduce $\alpha_x = 0.01$ after 65 iterations in order to obtain finer solutions. Additionally, we use a task/input vector of size 400 for the meta-learning task.

C.4 Regularization

We use the regularization coefficient $\lambda = 0.1$ and 2 for the generative modeling and inverse problems experiments, and with 40 and 100 JIIO iterations, respectively. We use $\lambda = 0.01$ for the adversarial training experiments and $\lambda = 0.5$ for our meta-learning experiments. For all experiments, we compute the Hutchinson estimator $\mathbb{E}_{\epsilon \in \mathcal{N}(0,1)}[\epsilon^\top J_z^\top J_z \epsilon]$ using 2 samples of ϵ.

C.5 Compute and Runtime

The generative model and inverse problems experiments were trained on 4 RTX-2080 Ti GPUs. The generative modeling experiments were run for 50k training steps. For the inverse problems experiments, we trained models with 100 JIIO iterations for 25k training steps, taking 4.5-5 days for each training run. Adversarial training experiments with JIIO trained models take 9-10 hours for CIFAR10 on 3 RTX-2080 Ti GPUs while taking 5-6 hours on 2 RTX-2080 Ti GPUs for MNIST experiments. The models trained with projected gradient descent take 20-21 hours to train on 3 RTX-2080 Ti GPUs for the CIFAR10 experiments while taking 13-14 hours to train on 2 RTX-2080 gpus for the MNIST experiments. For our meta-learning experiments, the models trained using JIIO used 4 RTX-2080 Ti GPUs for roughly 2 days.

C.6 Space complexity of the method

As pointed out earlier, JIIO has higher memory requirements than the corresponding ADAM version due to the higher memory sizes used in computing the fixed point. For example, in the generative modeling/inverse problems, optimization using JIIO requires 17.45 GB of GPU memory as opposed to vanilla Adam based optimization which simply costs 7.47 GB GPU memory for a batch of 48 images from celebA. However, for the adversarial training problems, the memory requirements were comparable - JIIO requires 2.19 GB memory as opposed to 2.15 for projected gradient descent for a batch with 96 images from CIFAR10.

Hyperparameter	Adversarial Training		Generative Model/Inverse Problems	
	MNIST	CIFAR10	CelebA64	Omniglot
Input Image Size	28×28	32×32	64×64	28×28
Batch Size	96	96	48	80
Optimizer	Adam	Adam	Adam	Adam
(Start) Learning Rate	0.001	0.001	0.001	
Nesterov Momentum	0.9	0.9	0.9	0.9
Weight Decay	0	0	0	0
Number of Scales	2	2	4	2
# of Channels for Each Scale	[24, 24]	[24, 24]	[32,64,128,256]	[64, 128]
Width Expansion (in the residual block)	5×	5×	5×	5×
Normalization (# of groups)	GroupNorm(8)	GroupNorm(8)	GroupNorm(8)	GroupNorm(8)
Weight Normalization	Yes	Yes	Yes	Yes
# of Downsamplings Before Equilirbium Solver	0	0	0	0
Forward Quasi-Newton Threshold T_f	18	18	18	18
Backward Quasi-Newton Threshold T_b	20	20	20	20
Broyden's Method Storage Size m	20	20	20	20
Anderson JIIO Memeory Storage Size M	20	20	40	10
Anderson JIIO Damping α	[0.8, 0.6, 0.1]	[0.8, 0.6, 0.6]	[0.8, 0.6, 0.01]	[0.8, 0.6, 0.04]
Variational Dropout Rate	0.0	0.0	0.0	0.0

Table 9: MDEQ hyperparameters for each task