Joint inference and input optimization in equilibrium networks

Anonymous Author(s)
Affiliation
Address
email

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, and adversarial training.

1 Introduction

Many settings in deep learning involve optimization over the inputs to a network to minimize some desired loss. For example, let's say one has 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. [9, 12])

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

As another example, constructing adversarial examples for classifiers [26, 48] typically involves optimization over a perturbation to a given input, i.e., given a network $g : \mathcal{X} \to \mathcal{Y}$, task loss $\ell : \mathcal{Y} \to \mathbb{R}_+$, 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 so-called inverse problems [9] and other auxiliary tasks [20, 3] in deep learning can also be formulated in such a manner.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

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 the optimization fixed-point iteration *with* the forward DEQ fixed point iteration all within a single "augmented" DEQ that *simultaneously* performs forward inference in the DEQ model and optimization over the inputs. This enables the models to more quickly perform both the inference and optimization procedures, and makes the process fast enough that we can *train* networks that use such "bi-level" fixed point passes. In fact, we also show a close connection between our proposed approach and primal-dual methods for constrained optimization.

We illustrate our methods on three tasks: training DEQ-based generative models while optimizing over latent codes, training models for inverse problems such as denoising and inpainting, and adversarial training of DEQ networks. 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 and and a 3x speedup in adverarial training of DEQ networks. In total, we believe this work points to a variety of new potential applications for optimization in implicit models.

2 Related Work

Implicit Layers Implicit layers have gained tremendous popularity in recent years[42, 18, 27]. They define the computation as a condition on the output that the model must satisfy. Some recent examples of implicit layers include optimization layers [15, 1], deep equilibrium models[5, 6, 60, 36, 47], Neural ODEs [13, 17, 55], logical structure learning [59], and continuous generative models [28].

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 [33]. In this work, we propose an efficient approach to perform optimization over the input to the DEQ by simultaneously optimizing over the inputs and solving the forward fixed point iteration of the DEQ as a joint fixed point iteration. As related work, Jeon et al. [33] 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. [47] 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 $z^\star$) and hence caters to only specific use cases such as modelling normalizing flows. Perhaps the most relevant approach to our work is Gilton et al. [24], which specifically formulates inverse imaging problems as a DEQ model. However, the approaches also differ substantially: our approach is focused 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 very 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 [48, 41], solving inverse problems [9, 12, 50], learning generative models [8, 64], meta-learning [52, 20] etc. For most of these examples, the optimization over the input 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 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. [64], Bojanowski et al. [8] proposed to train generator networks by jointly optimizing over the parameters and the latent

variables corresponding to each example. Similarly, optimizing over the latent space to match a target image is common in decoder-only models like GANs to get correspondences [9, 35], and has been found useful to stabilize GAN training [63]. 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. [9], Chang et al. [12] 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. [14], Gilton et al. [23], Gregor and LeCun [29] 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. [9], Chang et al. [12], 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 [56, 26]. 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 [51, 34, 57, 61]. The most successful strategy thus far has been adversarial training with a projected gradient descent (PGD) adversary [48] 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 [40] and optimal transport mappings [49], MPC [11], 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 more formally 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

$$z_\theta^\star(x) = f_\theta(z_\theta^\star(x), x), \tag{3}$$

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 $z_\theta^\star$ *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 [7, 43].

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 [10] or Anderson acceleration [4, 58]. 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 [60, 53, 21]), 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 $\ell : \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}_+$.

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

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

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

until convergence, where we use term $z^\star$ alone to denote the fixed output of the network $z_\theta^\star$ (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:

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

Thinking about $z_\theta^\star$ 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:

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

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 $\mu = \left(I - \frac{\partial f_\theta(z,x)}{\partial z} \right)^{-\top} \left(\frac{\partial \ell(z,y)}{\partial z} \right)^\top$ via the following iteration (i.e., a Richardson iteration [54]):

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

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

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

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* $v_\theta^\star(y)$ (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 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.

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.

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 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 [19, 31, 16].

3.4 Outer Optimization (Backward Pass)

A notable advantage of formulating the entire joint inference and input optimization problem as an augmented DEQ $v_\theta^\star(y)$ 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:

$$\underset{\theta}{\text{minimize}} \;\; \ell^{\text{outer}}(v_\theta^\star(y), y) \tag{15}$$

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 [38]. 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.

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

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,

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

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

$$\ell^{\text{outer}}(y, v_\theta^\star(y)) = \ell(y, z_\theta^\star(y)) \tag{18}$$

so we have that

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

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 $\ell^{outer} = -\ell$.

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 3 different types of problems that are popular areas of research in machine learning: generative modeling [39, 25], inverse problems [9, 23, 29] and adversarial training [61, 48]. 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 normalizations [32] with group normalization [62] in order to ensure the inner loop 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 [64, 8]; i.e., given a decoder network D, the latent representation x of a sample y (e.g., an image) is $x = \min_{x \in \mathcal{X}} \|D(x) - y\|_2^2$.[1] Moreover, instead of placing explicit regularizations on the latent space $\mathcal{X}$ (as in VAEs), we follow [22] 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:

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

where h 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. [46] and train the model for 50k training steps. We use Fréchet inception distance (FID) [30] 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 [39]. 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.

[1] 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 1: Samples generated with JIIO on a small MDEQ network.

Model	Generation	Reconstruction
VAE [39]	48.12	39.12
RAE [22]	**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.

We additionally apply JIIO on pre-trained MDEQ-VAEs (i.e., train an MDEQ-based VAE as usual on optimizing ELBO [39], 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.

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 [37]). 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 $9\times$ the time taken by 40 iterations of JIIO, which we used during training and more than $3.5\times$ 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 2 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 is still somewhat unstable (see Fig. 2) and fluctuates more, probably due to the more challenging augmented system that we introduce in JIIO.

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 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. For the inpainting task, we randomly mask a 20x20 window from the image and train the model to adequately fill those missing pixels based on the 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 aim to recover the original image. We use the same datasets and train-test setups as in the generative modelling experiments.

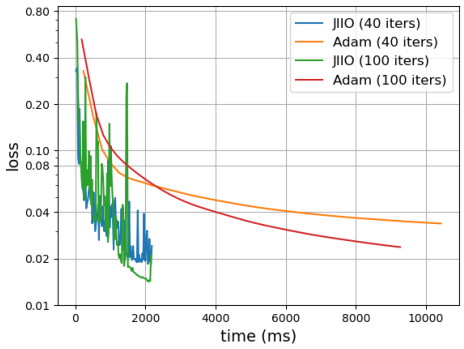

Figure 2: Cost changing with time for Adam v/s JIIO optimization. Tested on models trained with 40 and 100 JIIO iterations respectively

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 modelling/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) [9]	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. [9], Chang et al. [12] 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 $G : \mathcal{X} \to \mathcal{Y}$, 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 $A : \mathcal{Y} \to \mathcal{Y}$ 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 (20):

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

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 $G_\theta : \mathcal{Y} \to \mathcal{Y}$ 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 modelling 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:

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

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

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

We apply our setting to a DEQ-based classifier $h_\theta(x) = h_\theta(z_\theta^\star(x))$ 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

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

Specifically we train MDEQ models on CIFAR10 [44] and MNIST [45] 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 modelling/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 and adversarial training of DEQs. 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, and several limitations of the method as it currently stands. Notably, despite the fact that the augmented joint inference and input optimization DEQ can embed both processess in a "single" DEQ model, we note that in practice these joint models take substantially more iterations to converge (often in the range of 50-100) than traditional DEQs (often in the rate of 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 contributes to challenges in 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.

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.

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

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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`.

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

[15] 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`.

[16] 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.

[17] 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`.

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

[19] 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.

[20] 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`.

[21] 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.

[22] P. Ghosh, M. S. Sajjadi, A. Vergari, M. Black, and B. Schölkopf. From variational to deterministic autoencoders. 2020.

[23] 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.

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

[25] 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`.

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

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

[28] 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`.

[29] 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.

[30] 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.

[31] 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.

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

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

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

[35] 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.

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

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

[38] 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.

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

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

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

[42] 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/.

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

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

[45] 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.

[46] 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.

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

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

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

[50] 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.

[51] 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.

[52] 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.

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

[54] 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.

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

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

[57] 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.

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

[59] 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`.

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

[61] 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`.

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

[63] 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`.

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

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We mention and show that our method provides significant speedups and demonstrate it in the settings mentioned in the abstract and introduction
   (b) Did you describe the limitations of your work? [Yes] Yes, we discuss the limitations in section 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our method provides a general optimization framework to speed up inner loop optimization in DEQ models applied to bilevel optimization problems. The checklist mentions : "On the other hand, if you develop a generic algorithm for optimizing neural networks, you do not need to mention that this could enable people to train models that generate Deepfakes faster.". As a result, we didn't feel the need to include a negative social impacts section
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] we present an algorithm and don't state any theoretical results

(b) Did you include complete proofs of all theoretical results? [N/A] We derive our update rule in section 3.2 but we don't consider it a proof of any theoretical result.

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] we provide the code and the instructions to reproduce the main experiments along with links to datasets we use in the supplementary material

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We don't provide all the details in the main paper for lack of space, but we provide most of the main details in the appendix and reference our code and existing papers for the rest of the details

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] we do provide error bars for our experiments on adversarial training as they were relatively less expensive. However, our experiments with generative models and inverse problems were very computationally expensive and time consuming (each experiment required 4 gpus for 3-4 days) for training. Thus we were unable to run multiple runs with random seeds for all of the experiments.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We do provide rough estimates of the amount of computational resources required for each individual experiment, however, we hadn't kept track of all the partial runs, auxilliary and failed experiments run throughout the course of this project

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We use code and models from 2 repositories and cite their work in our paper and links to their code in the appendix. We also mention and cite all the data sources and the corresponding papers in the appendix and in the paper

(b) Did you mention the license of the assets? [No] The datasets, code bases and their licenses all exist as open source so we just provide citations and links to them in the paper.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code in the supplementary material, but we also expect to release the source code soon after acceptance.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The data sources are all publically available and we thus simply cite them in the paper

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] The datasets we use have been in public use for many years and do not contain such content

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]