$(\text{Implicit})^2$: Implicit Layers for Implicit Representations

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recent research in deep learning has investigated two very different forms of "implicitness": *implicit representations* model high-frequency data such as images or 3D shapes directly via a low-dimensional neural network (often using e.g., sinusoidal bases or nonlinearities); *implicit layers*, in contrast, refer to techniques where the forward pass of a network is computed via non-linear dynamical systems, such as fixed-point or differential equation solutions, with the backward pass computed via the implicit function theorem. In this work, we demonstrate that these two seemingly orthogonal concepts are remarkably well-suited for each other. In particular, we show that by exploiting fixed-point *implicit layer* to model *implicit representations*, we can substantially improve upon the performance of the conventional explicit-layer-based approach. Additionally, as implicit representation networks are typically trained in large-batch settings, we propose to leverage the property of implicit layers to *amortize* the cost of fixed-point forward/backward passes over training steps – thereby addressing one of the primary challenges with implicit layers (that many iterations are required for the black-box fixed-point solvers). We empirically evaluated our method on learning multiple implicit representations for images, videos and audios, showing that our $(\text{Implicit})^2$ approach substantially improve upon existing models while being both faster to train and much more memory efficient.

1 Introduction

The concept of "implicitness" has recently been applied to various contexts of machine learning research. One particular thread is *implicit representations* [18, 21, 27, 26], which aims to learn a continuous representation of high-frequency, often discretely-measured signals such as large images. Formally, given a spatial-temporal coordinate input $x \in \mathbb{R}^d$, an implicit representation of it is a function $\Phi : \mathbb{R}^d \to \mathcal{C}$, where $\mathcal{C}$ is the space of desired quantity (e.g. color, volume density, distance, etc.) and Φ is usually parameterized with a neural network. Such implicitness has multiple benefits over the conventional discrete (e.g., grid-based) representations; e.g., as Φ is defined on a continuous domain, an implicitly represented input consumes much less storage than the original input; as another example, Φ is differentiable, hence allowing for the computation of higher-order derivatives. However, the training of these Φ networks themselves are usually quite memory-consuming, since we usually deal with very high-resolution images and videos, and training is typically done in full-batch mode (e.g., when training on a batch size of 512×512, a simple 4-layer MLP with 1024 hidden units already requires > 16 GB memory just to store the intermediate activations).

A recent (orthogonal) usage of *implicitness* in deep learning comes from the notion of *implicit layers/models*, where the word is used to characterize model architectures (rather than input representations). The nomenclature comes from the concept of implicit vs. explicit functions: instead of

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

representing a model as an explicit stacking of layers, an implicit model solves a non-linear dynamical systems [10] (e.g., ODEs [7, 9] or fixed-points [1, 31]). Importantly, these implicit models decouple the forward and backward passes, and analytically differentiate via the implicit function theorem [14]. This enables these models to use *constant training memory*, irrespective of the forward pass trajectory. Moreover, recent works on deep equilibrium models [1] have been shown to be able to achieve results on par with the state-of-the-art explicit models (e.g., Transformers [30]) on very large-scale vision or NLP tasks [1, 2]. However, despite training memory efficiency, these implicit layers are also slower to compute (e.g., by 2-3$\times$), since the solvers are usually iterative in nature and ultimately require many function evaluations.

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

Specifically, we propose to combine the best of both worlds by replacing the explicit models used in conventional *implicit representation* learning with implicit layers, whose output is defined by a *fixed-point condition*, thus dubbed $(\text{Implicit})^2$ networks. Formally, let x be the input and F be a (often shallow) layer, the $(\text{Implicit})^2$ approach learns the following implicit representation Φ:

$$\Phi(x) = Wz^\star + b, \text{ where } z^\star = F(z^\star; x) \tag{1}$$

While the modification seems simple, the addition of implicitness in fact yields a much richer set of functions than the ones can be represented by a forward pass of F alone. Importantly, we show that these two "implicitness" complements each other well, especially in two important aspects. First, the unique large-batch training scheme of *implicit representations* as well as the forward/backward decoupling properties of *implicit models* permit us to *amortize* the cost of the iterative solver that would otherwise make implicit models slow to train. We show that by fixed-point reuse (in forward pass) and Jacobian approximation (in backward pass), training an implicit model on implicit representation tasks requires very little computation overhead. Second, implicit models allow us to train these high-frequency input tasks with a much lower memory consumption (while not losing speed or performance), thus significantly lowering the hardware requirement and computation budget of training implicit representations. We evaluate $(\text{Implicit})^2$ on multiple implicit representations for images, videos and audios, showing substantial improvement upon existing competitive (explicit-model) methods like SIREN [26] or MFN [11] using equivalent-sized networks.

2 Background and Preliminaries

2.1 Implicit Neural Representations (INR)

While high-frequency data such as images and scene geometry have been traditionally represented in discrete manners (e.g., pixel/voxel grids), recent work has demonstrated the the possibility of replacing them with continuous functions parameterized by multi-layer perceptrons (MLPs) [26, 27]. These deep networks have been used to learn differentiable representations of various forms, such as continuous spatial-temporal image representations [5, 8], high-resolution scene representations [25, 21], and volumetric rendering [18, 28], and have been shown to be significantly more compact than the conventional grid-based approaches. However, training these INR models is not easy. For example, the training of implicit representations are conducted usually in very large batch sizes (e.g., 512×512), which renders these (albeit simple) MLPs very memory-consuming during training. As another example, recent works have shown that typical deep networks with ReLU/tanh/sigmoid non-linearities fail to capture the fine details of an input signal, and resorted to periodic non-linearities [26], Fourier feature encodings [27], or repeated application of non-linear filters (e.g., Gabor wavelets) to the input and then multiply with the features [11]. We introduce below the two latest, highly-competitive models that have been developed on this task, which our work will build on (see Sec. 3.1).

Sine-activated Network for Implicit Representations (SIREN). To overcome the detrimental effect of traditional non-linearities like ReLU/tanh on modeling fine details and higher-order derivatives of the input signals, Sitzmann et al. [26] proposes to use sinusoidal activation functions that

allows explicit supervision on any derivatives of the input signal. The resulting MLPs, though simple, have been shown to achieve state-of-the-art performance in representing images, videos, complex geometries, and more []. Formally, given an input x, SIREN learns an implicit representation by the following L-layer MLP:

$$\Phi(x) = W(g_{L-1} \circ g_{L-2} \circ \cdots \circ g_0)(x) + b \tag{2}$$

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

Multiplicative Filter Networks (MFN). More recently, Fathony et al. [11] proposes to forego the SIREN-like design (i.e., MLPs with periodic activation functions) in favor of multiplicative operations that are proven to be similarly expressive. Specifically, at each layer i of the network, an MFN applies a learnable non-linear filter kernel (paramterized by $\theta^{(i)}$) $g(x; \theta^{(i)})$ on the *original input* $x \in \mathbb{R}^n$, and elementwise multiply it with a *linear transformation* of the features; i.e., formally,

$$\Phi(x) = W h^{[L]} + b, \text{ where } h^{[i+1]} = \big(W_i h^{[i]} + b_i\big) \cdot g(x; \theta^{(i)}) \text{ and } h^{[0]} = g(x; \theta^{(0)}) \tag{3}$$

Fathony et al. [11] in particular proposed two instantiations that admit sinusoids or a Gabor wavelet as the filter g, which are called FourierNet and GaborNet, respectively:

$$\begin{aligned} g_{\text{Fourier}}(x; \theta^{(i)}_{\text{Fourier}}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \\ g_{\text{Gabor}}(x; \theta^{(i)}_{\text{Gabor}}) &= \sin(\omega^{(i)} x + \phi^{(i)}) \cdot \exp\big(-(\gamma^{(i)}/2) \cdot (x - \mu^{(i)})^2\big) \end{aligned} \tag{4}$$

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

2.2 Implicit Models

Whereas explicit models hierarchically stack a sequence of L non-linear operators to build a computation graph for their forward/backward passes (and where L is often called the depth), implicit models are designed to have no prescribed computation graph. Instead, they posit a non-linear dynamical system to solve for. For example, the Neural ODE model [7] solves an initial value problem ODE that intuitively corresponds to infintesimal residual layers [13]. Another example is the recent deep equilibrium (DEQ) models [1] , whose forward pass involves computing the fixed point of a shallow layer F. Importantly, unlike explicit models, these implicit models do not need to store any intermediate activations during training: to produce gradients in the backward pass, we can directly differentiate through their *final outputs* (e.g., by adjoint method for Neural ODEs [7] and implicit diffrentiation for equilibrium models [1]), regardless of how the forward pass was computed. Therefore, the forward and backward passes of implicit models can rely on independent black-box solvers, and these models consume only a constant training memory (as they do not need to store anything). Our work is primarily built on the formulation of deep equilibrium models, which we introduce in detail below.

Deep Equilibrium (DEQ) Models. Given an input x and a layer $F(z; x)$ (e.g., a residual block, or a multi-head self-attention layer [30]), a DEQ model aims to approximate an infinite-level stacking of this layer (i.e. $z^{[i+1]} = F(z^{[i]}, x)$ with $i = 1, \ldots, L$ and $L \to \infty$). To achieve this purpose, Bai et al. [1] advocated for directly solving a fixed-point condition defined by F and x:

$$z^\star = F(z^\star; x) \tag{5}$$

where $z^\star$ is called the equilibrium (since intuitively, stacking another $F(\cdot, x)$ layer on top will not change the activation). To solve for the equilibrium point, prior works have had success with various kinds of advanced solvers, such as Broyden's method [6, 1, 17], and Peaceman-Rachford splitting [22, 31, 23]. Then, in the backward pass, we can produce the gradient by the implicit function theorem [14] directly applied on this equilibrium point $z^\star$, with a loss ℓ:

$$\nabla_{(\cdot)} \ell = \frac{\partial \ell}{\partial z^\star} \left(I - J_F\right)^{-1} \frac{\partial F(z^\star; x)}{\partial (\cdot)} \tag{6}$$

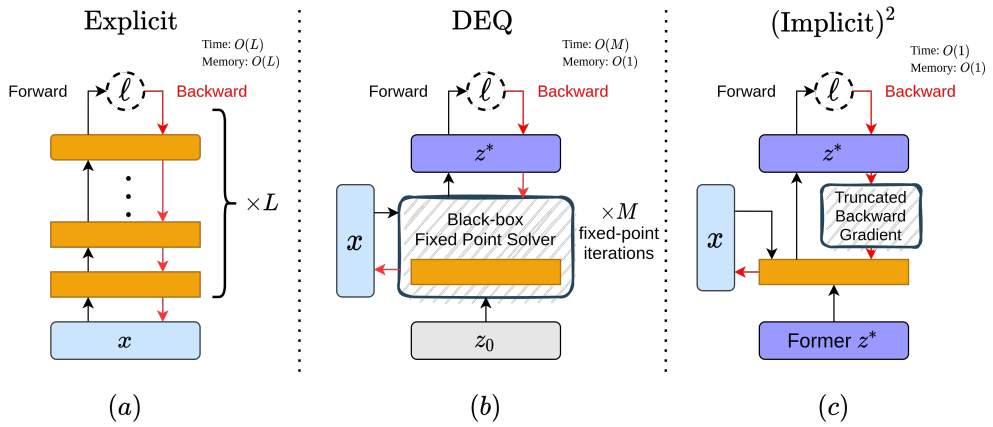

Figure 1: (a) Explicit networks incur $O(L)$ complexity in both time and memory during training. (b) Original DEQ models requires constant memory, yet they take $O(M)$ time in forward/backward passes, where M is the number of fixed-point solver steps. (c) In contrast, an $(\text{Implicit})^2$ Network consumes constant amortized memory *and* time in both forward and backward passes.

where J_F denotes the Jacobian of F evaluated at $z^\star$. As the inverse of a matrix is expensive to compute, in practice the backward pass is usually formulated as another *linear* fixed-point condition in y: $y^\top J_F + \frac{\partial \ell}{\partial z^\star} = y^\top$, which again can be solved via any of the advanced black-box solvers. Thus theoretically, the model only needs to store x, $z^\star$ and $F(z^\star; x)$ in memory at training time, which is equivalent to the memory consumption of merely one F layer. Recent works [1, 2] have demonstrated that DEQ models can scale to high-dimensional tasks and be as competitive as explicit structures like ResNet [13] and Transformers [30] on large-scale computer vision and NLP tasks. However, these implicit models are also slower (typically by $\sim 3\times$) to train, because both forward and backward passes involve numerous iterative steps.

Our work studies the novel combinations of *implicit models* and *implicit representation learning*. Specifically, we show that these two "implicitness" are surprisingly well-suited to each other, as they almost perfectly compensate for each others' drawbacks. We demonstrate that implicit modeling of a simple layer F substantially improve the training speed, memory, *and* performance on implicit representation tasks when compared to the aforementioned state-of-the-art deep explicit networks.

3 $(\text{Implicit})^2$ Networks

As we discussed in Sec. 2.1, learning implicit representation for complex signals is often more effective when projecting the input coordinates onto high-frequency bases (e.g. sinusoidal functions), rendering the designs of these models typically different from deep networks used in more common applications, like image classifications (e.g., ResNets [13]). In this work, we propose to directly build on the designs of these aforementioned state-of-the-art layer architectures by only making minimal modifications, but modeling them as shallow implicit models. We briefly introduce below the two instantiations of our proposed $(\text{Implicit})^2$ approach, which are based on SIREN [26] and MFN [11], respectively, followed by a discussion of how implicit representation tasks can enable us to amortize the cost of training these implicit networks significantly in practice.

3.1 $(\text{Implicit})^2$ Network Architctures

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

$$F_{\text{SIREN}}(z; x) = \sin(W(z + \sin(Vx)) + Ux + b) \tag{7}$$

where, following the practice of prior works on deep equilibrium models [1, 31, 23], the transformed input x is added to the hidden representation z (see Fig. 2). Just like the canonical SIREN layer, any order of derivative of this input-injected SIREN variant is also an input-injected SIREN of the same form. Therefore, this proposed variant inherits the same high-frequency and higher-order

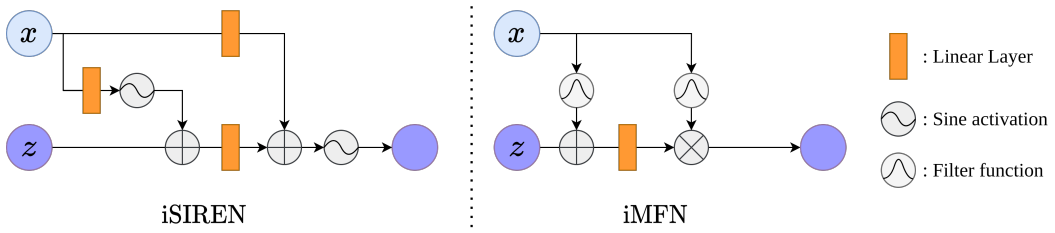

Figure 2: An L-layered function $F^{(L)}$ for implicit layer in iMFN and iSIREN. Note that when $z = 0$, forwarding $F^{(L)}$ once is equivalent to a respective L-layered explicit model.

differentiability properties as SIREN, and we name the implicit model that solves the fixed point of it as *iSIREN*.

Implicit Multiplicative Filter Networks (iMFN) Additionally, we introduce a variant based on the most recent Multiplicative Filter Networks (MFN) [11]. As MFNs only perform linear transformations on the hidden features, which are multiplied with a non-linear filter function on the original input x, we slightly modify the layer design as follows to introduce an additional input injection:

$$F_{\text{MFN}}(z; x) = (W(z + g(x; \theta_2)) + b) \circ g(x; \theta_1) \tag{8}$$

where g denotes the filter function of choice, such as Gabor or Fourier (g_{Gabor} and g_{Fourier} in Eq. (4)) filters [11]. We call the implicit models defined by this layer *iMFN*. Moreover, just like the original MFN [11], we can show that the output of iMFN (which is the fixed point of F_{MFN} is also a linear combination of the non-linear filter functions (see Appendix for proof).

We provide visualizations of these two instantiations of prior architectures in Fig. 2. With these variants, we propose the $(\text{Implicit})^2$ approach to implicit representation tasks as follows:

$$\Phi(x) = W' z^\star + b', \text{ where } z^\star = F(z^\star; x), \text{ and } F \in \{F_{\text{MFN}}, F_{\text{SIREN}}\} \tag{9}$$

3.2 Accelerated Training of $(\text{Implicit})^2$ Networks

Compared to explicit networks, training with implicit models significantly lower the *memory* budget required (due to implicit differentiation). This is especially compelling for training implicit representation tasks, which are often memory-bottlenecked. In this subsection, we describe how one can exploit 1) the large-batch nature of implicit representation training; and 2) the nature of fixed-point optimizations (in both forward and backward), to significantly improve the *speed* of these models, making them eventually both more memory-efficient *and* time-efficient than explicit models (while achieving performance as competitive, or better; see Sec. 4).

Fixed-point Reuse Suppose we fix a training step t and let the correspsonding network parameter (i.e., weights of the network) be given by $\theta_{(t)}$. Typically, a deep equilibrium model solves for the fixed point $z^\star_{(t)}$ given an input x with a black-box fixed-point solver starting with an initial guess $z^{[0]}$ (which is usually $\mathbf{0}$ or a random vector in $\mathcal{N}(0, I)$):

$$z^\star_{(t)} = \mathsf{FixedPointSolve}(F, \theta_{(t)}, x, z^{[0]}) \tag{10}$$

where we note that $z^\star_{(t)}$ does not depend at all on this initial guess $z^{[0]}$ of the optimization. Depending on the quality of the initial guess, the optimization process itself can take a different number of steps. For example, in the unlikely circumstance where we guessed $z^{[0]}$ to be $z^\star_{(t)}$, the fixed point solvers will directly converge at

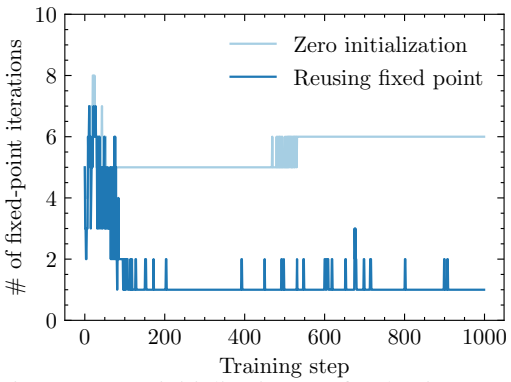

Figure 3: Zero initialization *v.s.* fixed point reuse

iteration 0. However, in practice, this property is rarely used, as given a complex high-diemsnional input x (e.g., a high-resolution image), it is hard to know (or learn) what makes a good initial guess.

Therefore, these implicit model's aforementioned memory efficiency comes at a cost: without good initial guesses, we must spend more iterative updates in order to secure convergence to a good enough estimate of the fixed point in practice, thus getting a slower model. However, we argue that the training on many implicit representation tasks, which uses the entire batch of data (e.g., to fit an RGB-color image implicit representation, each pixel of this image is a "sample" of this full batch), can compensate this issue by providing us very reasonable initial guesses. In particular, training in the full-batch mode allows us to have direct access to *all* of the fixed ponits $z^\star_{(t)}$ for all inputs x in the training set after this step t; and assuming the model updates between iterations t and $t+1$ are small (which is true in practice, such as via SGD with a small learning rate η), one can safely assume the fixed points of the updated layer $F(\cdot; x, \theta_{(t+1)})$ do not deviate much from its current estimate $z^\star_{(t)}$. Formally, $F(\cdot; x, \theta_{(t+1)}) \to F(\cdot; x, \theta_{(t)})$ as learning rate $\eta \to 0$, which implies $z^\star_{(t+1)} \to z^\star_{(t)}$. We empirically verify this on these full-batch training settings (see Fig. 3), where the majority of the fixed-point convergence (to a target level ε, such as 0.01) finishes in exactly 1 step.

However, we note that this do not imply all training iterations will gain a similar level of convergence boost (e.g., see the first 100 steps in Fig. 3). As an simple example, at trianing iteration $t = 0$, we do not have a "previous estimate" at all, which means we still have to solve for $z^\star$ by running the solver for several steps with a neutral initial guess. Thus, we highlight that such fixed point reuse amortizes the cost of later training iterations, and indeed, the majority of the training of the implicit models. Empirically, we found that performing one step of the fixed point solving iteration is already good enough to help train the iSIREN and iMFN networks; i.e., at training step $t+1$, we simply perform

$$z^\star_{(t+1)} = F(z^\star_{(t)}; x, \theta_{(t)}). \tag{11}$$

Truncated Backward Gradient As mentioned in Sec. 2, the backward gradient of implicit models (Eq. (6)) can be alternatively obtained by solving a linear fixed-point system

$$y^\top = y^\top J_F + \frac{\partial \ell}{\partial z^*} \tag{12}$$

so as to avoid expliciting inverting the matrix. However, recent works have suggested that assuming sufficient stability in the DEQ model layer F, we can also approximate $\frac{\partial \ell}{\partial z^*}\left(I - J_F|_{z^*}\right)^{-1}$ by an expansion of it as a Neumann series [15]:

$$y^\top = \frac{\partial \ell}{\partial z^\star} \sum_{i=0}^{\infty} \left(J_F\right)^i \tag{13}$$

For example, a recent work [12] proposes to use the 0-th order approximation and perform a Jacobian-free backward gradient estimation (assuming Lipschitz F); i.e., $y^\top = \frac{\partial \ell}{\partial z^\star}$. Although Fung et al. [12] empirically showed to be effective in image classification tasks, we empirically found this to substantially hurt the training-time performance convergence. Therefore, we proposed to use a truncated backward gradient such that the Neumann series is unrolled for a $T > 0$ steps. In other words, the proposed gradient has the form

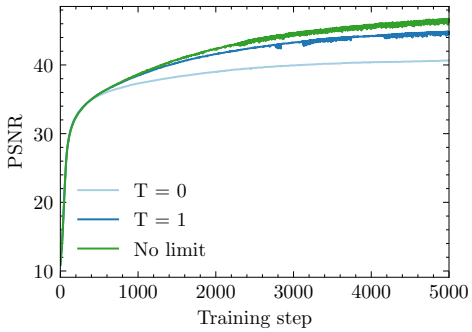

Figure 4: Training PSNR comparison between different levels of gradient approximation.

$$\hat{\nabla}_{(\cdot)}\ell = \frac{\partial \ell}{\partial z^*} \sum_{i=0}^{T} \left(J_F\right)^i \frac{\partial F(z^*; x)}{\partial (\cdot)} \tag{14}$$

In practice, we found $T = 1$ to be already sufficient for implicit representation tasks (see Figure 4, which requires only one evaluation of the vector-jacobian product $\frac{\partial \ell}{\partial z^*} J_F$.

Spectral Normalization We additionally apply spectral normalization on the layers F we use for the implicit models, which ensures that F (for forward passes) and J_F (for backward passes) are contractive mappings, and is able to guarantee unique fixed points under the Banach Fixed Point

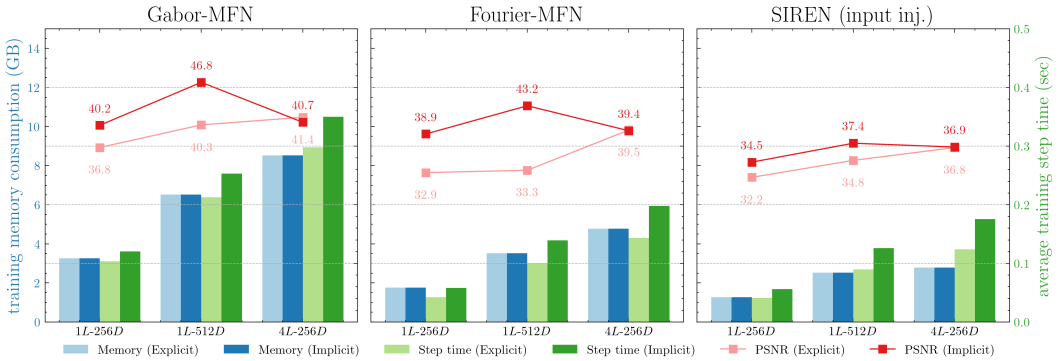

Figure 5: Comparison of step time, memory consumption and PSNR between explicit and implicit models trained on the 512×512 grayscale image.

Theorem [3]. Specifically, we adopt the power-iteration in [19, 4] to ex scale the spectral norms of W in Eq. (7) and (8) after each training iteration if it become > 1, which incurs little additional computation cost [19]. However, we note that contractivity is not always necessary for fixed-point convergence (especially in the presence of stronger fixed-point solvers, as shown in large-scale DEQ models [1]), and weaker regularizers such as weight normalization [24] usually also suffice in practice.

4 Experiments

We evaluate our $(\text{Implicit})^2$ networks on several representation tasks and compare the difference in performance between the implicit and explicit modeling of implicit representations across a variety configurations. For both explicit networks and implicit networks, we use L to denote the number of layers and D to denote the hidden dimensionality of the features. Unless stated otherwise, our implicit models have exactly one F layer (i.e., $L = 1$), in the exact form we described in Sec. 3.1. Our set of experiments is drawn from prior works where competitive implicit representation networks are evaluated on [11, 26]. Overall, our results of learning implicit representation on various domains (including images, videos, audios) suggest that the $(\text{Implicit})^2$ approach offers clear improvement over existing explicit models used for these purposes, where we are able to achieve the same (or better) level of performance while being up to $3\times$ more memory-efficient and $3\times$ more time-efficient. We provide the code in the supplement and the descriptions of the tasks and datasets can be found in Appendix.

4.1 Image Representation

We first evaluate the difference between explicit networks and $(\text{Implicit})^2$ networks on representing a high-resolution 512×512 grayscale image, which is a commonly used goalpost for evaluating implicit representation models from the scikit-image package [29]. In particular, we fit an implicit representation function $\Phi : (x, y) \rightarrow \mathcal{C}$, where $\mathcal{C}$ is the desired color space (and in this case $\mathcal{C} = \mathbb{R}$). We train each of the model for 5000 iterations (under the same setting) using all pixels in the image (i.e., batch size 262,144), and demonstrate the final peak signal-to-noise ratio (PSNR), memory consumption, and average training step time in Fig. 5. The results demonstrates two interesting facts about using $(\text{Implicit})^2$ networks: Compared to standard 4-layer explicit networks $(4L\text{-}256D)$, even a small implicit network $(1L\text{-}256D)$ with approximately %75 less parameters achieve a comparable performance, while requiring only roughly 1/3 of the training time and memory footprint. When given more parameter budget, an implicit network with a similar model size $(i.e., 1L\text{-}512D)$ outperforms the best explicit

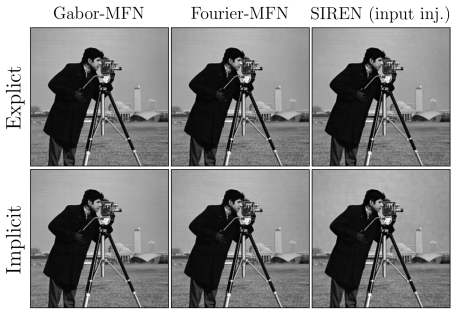

Figure 6: Learned 512×512 grayscale image

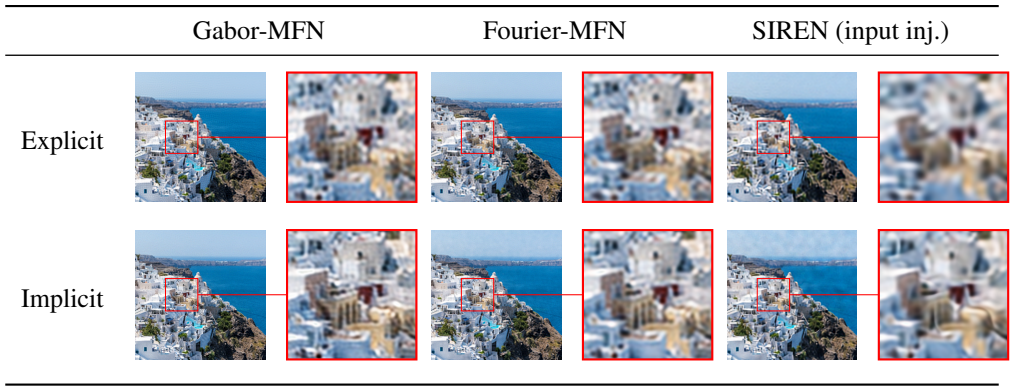

Figure 7: Samples of best performing explicit/implicit models learned on *Natural*

network by a large margin. For completion, we also evaluate an $(\text{Implicit})^2$ network where F also consists of a stack of 4 layers (rather than 1), and observe that the implicit modeling of an already deep structure does not yield much improvement here.

4.2 Image Generalization

In a number of applications [18, 20, 5, 8], implicit representations are used to infer unobserved parts of data. In order to demonstrate the ability for $(\text{Implicit})^2$ networks to generalize well, we train the network on only 25% of the pixels from each image in the *Natural* dataset, following [11], and evaluate PSNR on an unobserved 25% portion of the image. The average PSNR over all 16 high-resolution images in *Natural* is reported in Table 1, and a visualization of the improvement by $(\text{Implicit})^2$ models is shown in Fig. 7. It can be seen that $(\text{Implicit})^2$ networks improves substantially over the explicit baselines, where the best-performing implicit model outperforms the explicit counterpart by > 0.7 in PSNR using the same architecture. Visually, $(\text{Implicit})^2$ networks produces a sharper and shape-consistent representation compared with the best performing explicit network. We observe similar improvements in training speed and memory consumption as in the image representation task (about $3\times$).

	$1L$-$256D$ ($\sim$65K)	$1L$-$512D$ ($\sim$260K)	$4L$-$256D$ ($\sim$260K)
Fourier-MFN	23.27 ± 3.18	23.30 ± 3.05	24.57 ± 3.35
Fourier-iMFN (ours)	$24.88 \pm .44$	$\mathbf{25.19 \pm 3.64}$	24.52 ± 3.33
Gabor-MFN	24.16 ± 3.35	24.68 ± 3.46	24.65 ± 3.38
Gabor-iMFN (ours)	24.91 ± 3.41	$\mathbf{25.42 \pm 3.76}$	24.53 ± 3.33
SIREN (input inj.)	22.88 ± 3.0	24.52 ± 3.28	24.10 ± 3.34
iSIREN (ours)	24.28 ± 3.37	$\mathbf{24.92 \pm 3.58}$	24.05 ± 3.39

Table 1: PSNR (in dB) for all models on image generalization. Numbers in parentheses indicate the approximate number of parameters. The reported mean $\pm$ std is taken over the individual PSNR of the 16 images.

4.3 Audio Representation

Next, we show that the proposed $(\text{Implicit})^2$ network outperforms explicit networks on representing very-high-frequency audio signals. Following [26], we train the models on a 7-second music piece with a sample rate of 44140/sec. We observe that in all cases, the implicit modeling significantly outperforms the respective explicit models. We show the the error reduction results using when Fourier-MFN in Fig 8, with more results available in the appendix. Although an $(\text{Implicit})^2$ model only defines 1 layer,

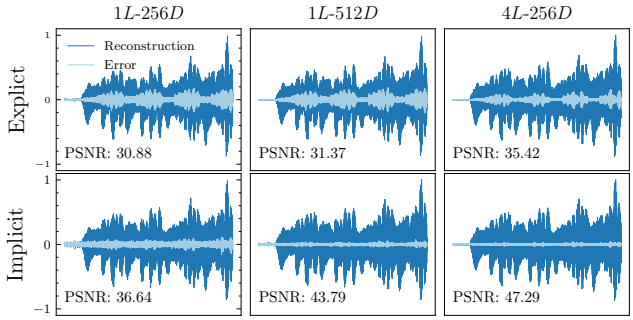

Figure 8: Audio signals represented using Fourier-MFNs. Lower error magnitudes are better.

it is able to outperform (in terms of PSNR) the explicit models by more than 23% (43.79 vs. 35.42), while still retaining the same level of memory- and speed-improvements as before.

4.4 Video Representation

Lastly, we may add an extra dimension to the input of implicit representation function and try to learn a model for video sequences $\Phi : (t, x, y) \to \mathcal{C}$, where t is the space of time. We aim to represent a 300-frame 512×512 video using each model. The PSNR results and visualizations are shown in Table 2. We observe that, unlike in the image representation setting, since the parameter counts of the models are far less than the size of the video (the smallest model only has as many parameters as 1.3% of all pixel values), model sizes have a more significant impact on the over all performance. But still, in most configurations, we found $(\text{Implicit})^2$ models lead to a consistently non-trivial improvement in performance when compared to their explicit counterparts, while doing so with a much lower memory budget requirement.

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

Table 2: PSNR for the video representation task. The reported mean $\pm$ std is taken over the each frame of the video.

5 Limitations and Directions for Future Work

Our $(\text{Implicit})^2$ approach can be significantly more time-efficient and memory-friendly than explicit networks. However, accelerated training in the forward pass by fixed point reuse (see Sec. 3.2) in practice can be bottlenecked by our ability to store them, such as hardware I/O bandwidth; i.e., it works best when we can fit all or a large portion of the fixed point into memory for quick retrieval. Although such assumption can be easily satisfied for implicit representation learning in many cases, for extremely large data spaces (e.g., in the continuous domain, like Park et al. [21] and Mildenhall et al. [18], where we sample points continuously in $\mathbb{R}^3$; or very long video frames like in 4.4) the fixed point reuse may not always work as well, and could entail more solver steps.

Smarter caching strategies for the fixed points may be developed in the future, where we consider only a subset of the input space for caching throughout the training. For example, one may consider a voxel-grid-based sparse storage for the fixed points, similar to the ones used in [16], such that fixed points of any batch of inputs may be approximately obtained from interpolation of the sparse storage. We leave this direction for future work.

6 Conclusion

In this paper, we propose $(\text{Implicit})^2$ networks, an application of to implicit representation tasks. We show that these two concepts of "implicitness" complements each other well, allowing us to take advantage of their properties (e.g., fixed point reuse) to produce a significantly more efficient training routine and usually more parameter-efficient models for implicit representation learning. We demonstrate through our set of experiments that the *implicit modeling* of *implicit representations* may in many cases be conveniently used as a drop-in replacement for existing state-of-the-art explicit models like SIREN and MFN to further improve the performance and reduce the memory/computation budget required to train these tasks.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[2] S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. math*, 3(1):133–181, 1922.

[4] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR, 2019.

[5] M. Bemana, K. Myszkowski, H.-P. Seidel, and T. Ritschel. X-fields: Implicit neural view-, light- and time-image interpolation. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2020)*, 39(6), 2020. doi: 10.1145/3414685.3417827.

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

[7] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[8] Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020.

[9] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural ODEs. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[10] D. Duvenaud, J. Z. Kolter, and M. Johnson. Deep implicit layers tutorial - neural ODEs, deep equilibirum models, and beyond. *Neural Information Processing Systems Tutorial*, 2020.

[11] R. Fathony, A. K. Sahu, D. Willmott, and J. Z. Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=OmtmcPkkhT`.

[12] 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.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[15] R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning (ICML)*, 2018.

[16] L. Liu, J. Gu, K. Z. Lin, T.-S. Chua, and C. Theobalt. Neural sparse voxel fields. *NeurIPS*, 2020.

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

[18] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020.

[19] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

[20] M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[21] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[22] D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for industrial and Applied Mathematics*, 3(1): 28–41, 1955.

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

[24] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[25] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019.

[26] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[27] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[28] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, et al. State of the art on neural rendering. In *Computer Graphics Forum*, volume 39, pages 701–727. Wiley Online Library, 2020.

[29] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453. URL `https://doi.org/10.7717/peerj.453`.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[31] E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We provide a detailed description in Sec. 5
   (c) Did you discuss any potential negative societal impacts of your work? [No] We did not spot any particular links to negative societal impacts for our work
   (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]
   (b) Did you include complete proofs of all theoretical results? [Yes] We include proofs in the supplement
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 include the code and experimental details in the supplement
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide a detailed specification in the supplement

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report the variance of image generalization results on 16 different images

(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 provide details in the supplement

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]

   (b) Did you mention the license of the assets? [No] We are not aware of particular licenses they are released under

   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The data we used are all publicly available in previous works

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We are not aware of any personally identifiable information or offensive content in the data

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]