Object Representations as Fixed Points: Training Iterative Refinement Algorithms with Implicit Differentiation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Object-centric learning has been motivated by developing algorithms that infer independent and symmetric entities from perceptual input. This often requires the use of iterative refinement procedures that break symmetries among equally plausible explanations for the data, but most prior works differentiate through the unrolled refinement process, which can make optimization exceptionally challenging. We observe that such iterative refinement methods can be made differentiable by means of the implicit function theorem, and develop an implicit differentiation approach that improves the stability and tractability of training such models by decoupling the forward and backward passes. This connection enables us to apply recent advances in optimizing implicit layers to not only improve the stability and optimization of the slot attention module in SLATE, a state-of-the-art method for learning entity representations, but do so with constant space and time complexity in backpropagation and only one additional line of code.

1 Introduction

One of the challenges for machine learning – particularly when working with complex real-world problems such as interpreting visual scenes – is defining and performing inference in models that are able to capture that complexity. Generative models attempt to do so by defining probability distributions that characterize the equivalently complex process by which data are generated, postulating latent parameters as intermediate steps in that process. We are interested in generative models that assume each observed datapoint x is generated from a *set* $\boldsymbol{\lambda}$ of latent parameters. For example, recent work has modeled objects in visual scenes by treating the representations of those objects as parameters of mixture model components [18, 28].

In these latent variable models, inference is typically done by breaking symmetry via a random initial guess $\boldsymbol{\lambda}_0$ and then iteratively updating the value of $\boldsymbol{\lambda}$, with variants of the classical Expectation-Maximization (EM) algorithm [11] being prominent examples. Recent works have replaced the analytical updates of EM with a neural network f that directly computes the update as $\boldsymbol{\lambda} = f(\boldsymbol{\lambda}, x)$, resulting in a class we call **iterative refinement algorithms**. These algorithms have predominantly been applied to learning object-centric representations of visual scenes [18-20, 26, 28, 37, 39, 40, 44]. All are trained by differentiating through the unrolled iterations of f. The state-of-the-art is the slot attention module from Locatello et al. [28], which serves as the focus of our paper.

Despite their conceptual elegance, it has been difficult to scale iterative refinement methods beyond modeling simple static scenes or short video sequences because differentiating through the unrolled forward iteration makes training unstable. Fig. 1b shows that the spectral norm of the Jacobian of f gradually increases over the course of training, which has empirically been observed to cause training

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

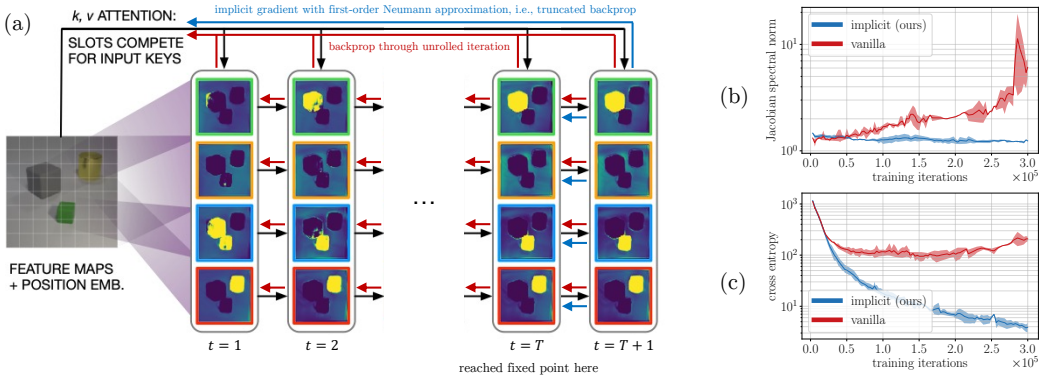


Figure 1: **Overview.** We address efficient training of symmetric factorized latent variable models that encapsulate iterative inference procedures, such as the slot attention model in (a), whose illustration is adapted from Locatello et al. [28]. Vanilla slot attention backpropagates gradients through the unrolled iterative inference procedure, which leads to training instabilities as shown by the growing Jacobian spectral norm (b). Implicit slot attention uses the first-order Neumann approximation of the implicit gradient, which simply truncates the backpropagation, leading to substantially more effective training, as shown in (c).

instabilities [7]. Such instabilities result in sensitivity to hyperparameter choices (e.g., number of inference iterations) and have motivated adding optimization tricks such as gradient clipping, learning rate warm-up, and learning rate decay, all of which make such models more complex and harder to use, restrict the model from optimizing its learning objective fully, and only temporarily delay instabilities that still emerge in later stages of training.

To approach this problem, we observe that previous methods have not taken advantage of the fact that f can be viewed as a fixed point operation. Thus, f can be trained with **implicit differentiation** applied at the fixed point, without backpropagating gradients through the unrolled iterations [8, 12]. We then leverage tools developed for implicit differentiation in neural models for improving the training of these iterative refinement algorithms.

Our primary contribution is to propose implicit differentiation for training the iterative refinement procedures of symmetric generative models, specifically slot attention. First, we show that slot attention can be cast as a fixed point procedure that can be trained with implicit differentiation, resulting in what we call **implicit slot attention**. Second, we show on the latest state-of-the-art method of this kind, SLATE [37], that using the first-order Neumann approximation of the implicit gradient for the slot attention module yields substantial improvement in optimization. Third, we show across three datasets that, compared to SLATE, our method for training achieves much lower validation loss in training, as well as lower Fréchet inception distance (FID) [23] and mean squared error (MSE) in image reconstruction. Fourth, our method also removes the need for gradient clipping, learning rate decay, learning rate warmup, or tuning the number of iterations, while achieving lower space and time complexity in the backward pass, all with just one additional line of code. Fifth, when integrated with the original slot attention encoders and decoders from Locatello et al. [28], implicit differentiation substantially improves object property prediction and continues to predict intuitive segmentation masks as the vanilla slot attention.

2 Related Work

Our methodological novelty is the adaptation of implicit differentiation techniques for training symmetric factorized object-centric models, where our key insight is that the iterative procedure used for symmetry-breaking reaches a fixed point, thus allowing implicit differentiation to be used. We are not aware that this has been done before, as current methods that perform iterative refinement for object-centric learning [18–20, 26, 28, 37, 39, 40, 44] all differentiate through the unrolled dynamics of the fixed point procedure, which as we show makes them difficult to train.

Our work draws upon innovations in implicit differentiation that have been applied in various other applications besides object-centric learning, such as embedded optimization layers [1, 3], neural ordinary differential equations [10], meta-learning [32], implicit neural representations [24],

declarative layers [17], and transformers [6]. Although we evaluate various technique for implicit differentiation beyond any specific one, our results highlight the benefits of the simplest of these, which is that of using a truncated Neumann approximation [15, 16, 24, 36]. While this technique was used in Zoran et al. [44] without theoretical explanation, we propose implicit differentiation as an explanation for why this technique is beneficial. The closest works to ours is the concurrent work of Zhang et al. [43] which applies implicit differentiation to predicting properties of set elements and finds similar benefits to implicit differentiation as we do. Our approach generalizes their results to the unsupervised setting and scales to high dimensional outputs (e.g. images), taking a crucial step forward toward improving self-supervised learning of symmetric generative models.

3 Background

Our work builds on prior works on iterative refinement and implicit differentiation with deep networks.

Iterative refinement for object-centric learning Current work in object-centric learning has been motivated by developing learning algorithms that infer *independent* and *symmetric* entities from perceptual input. Because mixture models are defined to have *a priori* independent and symmetric mixture components, these methods generally model each datapoint x^n (e.g. image) as a set of independent sensor measurements $x^{n,m}$ (e.g. pixels) which are posited as having been generated from a mixture model whose components represent the entities. Under a clustering lens, the problem reduces to finding the K groups of cluster parameters $\boldsymbol{\theta}^n := \{\theta^{n,k}\}_{k=1}^K$ and cluster assignments $\boldsymbol{\phi}^{n,m} := \{\phi^{n,m,k}\}_{k=1}^K$ that were responsible for the measurements $x^{n,m}$ of the datapoint x^n.

A network f breaks symmetry among components by alternately updating $\boldsymbol{\theta}^n$ and $\boldsymbol{\phi}^{n,m}$ starting from independent randomly initialized $\theta^{n,k}$'s. The slot attention module [28], e.g., computes $\boldsymbol{\theta}_{t+1}^n \leftarrow f\left(\boldsymbol{\theta}_t^n, x^n\right)$, where $\boldsymbol{\phi}^{n,m}$ is updated as an intermediate step inside f. The $\boldsymbol{\theta}^{\boldsymbol{n}}$, called *slots*, serve as input to a downstream objective, e.g. image reconstruction, whose gradients are backpropagated through the unrolling of f. Earlier works applied this approach to binary images [18] and videos [39].

Implicit differentiation Implicit differentiation is a technique for computing the gradients of a function defined in terms of satisfying a joint condition on the input and output. For example, a fixed point operation f is defined to satisfy "find λ such that $\lambda = f(x, \lambda)$," rather than through an explicit parameterization of f. This fixed point λ_* can be computed by simply repeatedly applying f or by using a black-box root-finding solver. Letting $f_{\mathbf{w}}$ be parameterized by weights $\mathbf{w}$, with input x and fixed point λ_*, the implicit function theorem [8] enables us to directly compute the gradient of the loss ℓ with respect to $\mathbf{w}$, using only the output λ_*:

$$\frac{\partial \ell}{\partial \mathbf{w}} = \underbrace{\frac{\partial \ell}{\partial \lambda_*}\left(I - J_{f_{\mathbf{w}}}\left(\lambda_*\right)\right)^{-1}}_{\mathbf{u}^\top} \frac{\partial f_{\mathbf{w}}\left(\lambda_*, x\right)}{\partial \mathbf{w}}, \tag{1}$$

where $J_{f_{\mathbf{w}}}\left(\lambda_*\right)$ is the Jacobian matrix of $f_{\mathbf{w}}$ evaluated at λ_*. Compared to backpropagating through the unrolled iteration of f, which is just one of many choices of the solver, implicit differentiation via Eq. 1 removes the memory cost of storing any intermediate results from the unrolled iteration. Deep equilibrium models (DEQ) Bai et al. [6] represent the class of such functions f parameterized by neural networks, which empirically produce stable fixed points even though the convergence properties of these models remains theoretically not well understood.

Much effort has been put into approximating the inverse-Jacobian term $\left(I - J_{f_{\mathbf{w}}}\left(\lambda_*\right)\right)^{-1}$ which has $\mathcal{O}(n^3)$ complexity to compute. Using notation from Bai et al. [7], Almeida [2], Pineda [31] propose to approximate the $\mathbf{u}^\top$ term in Eq. 1 as the fixed point of the linear system:

$$\mathbf{u}^\top = \mathbf{u}^\top J_{f_{\mathbf{w}}}\left(\lambda_*\right) + \frac{\partial \ell}{\partial \lambda_*}, \tag{2}$$

which can be solved with any black-box solver. However, in the context of applying implicit differentiation to neural networks, this approach has in practice required some expensive regularization to maintain stability [7], so Fung et al. [15], Geng et al. [16], Huang et al. [24], Shaban et al. [36]

propose instead to approximate $(I - J_{f_{\mathbf{w}}}(\lambda_*))^{-1}$ with its Neumann series expansion, yielding

$$\frac{\partial \ell}{\partial \mathbf{w}} = \lim_{T \to \infty} \frac{\partial \ell}{\partial \lambda_*} \sum_{i=0}^{T} J_{f_{\mathbf{w}}}(\lambda_*)^i \frac{\partial f_{\mathbf{w}}(\lambda_*, x)}{\partial \mathbf{w}}. \tag{3}$$

The first-order approximation ($T = 1$) amounts to applying f once to the fixed point λ_* and differentiating through the resulting computation graph. This is not only cheap to compute and easy to implement, but has also been shown empirically [16] to have a regularizing effect on the spectral norm of $J_{f_{\mathbf{w}}}$ without sacrificing performance.

4 Implicit Iterative Refinement

Our main contribution is to propose treating iterative refinement algorithms used for object-centric learning as fixed-point procedures, thereby motivating the use of implicit differentiation to train them.

4.1 Motivation

Our approach is inspired by the structural resemblance between these iterative refinement methods and the Expectation-Maximization algorithm [11], which also has been pointed out by Greff et al. [18] and Locatello et al. [28]. If f were to update the components $\boldsymbol{\theta}^n$ and assignments $\boldsymbol{\phi}^{n,m}$ in a way that monotonically improves the evidence lower bound (ELBO) $\mathcal{L}^n$ on the log-likehood of each image x^n with respect to $\boldsymbol{\theta}^n$ and $\boldsymbol{\phi}^{n,m}$, then we know from Neal and Hinton [29], Wu [42] that such an approach is a fixed point operation whose fixed point locally maximizes $\mathcal{L}^n$. With this interpretation, such iterative refinement algorithms for object-centric learning can be viewed as performing a nested optimization with three levels, where the weights of the slot attention module are optimized across images x^n, the components $\boldsymbol{\theta}^n$ are optimized per-image x^n but across measurements $x^{n,m}$, and the assignments $\boldsymbol{\phi}^{n,m}$ are optimized per-measurement $x^{n,m}$.

How these iterative refinement methods are implemented differs from optimizing the per-datapoint ELBO described above, however. Indeed, both Greff et al. [18] and Locatello et al. [28] implemented ablations to their methods that do implement a variant of Expectation-Maximization on a well-defined mixture model, but found that replacing the update rule with a recurrent neural network empirically performs better at extracting representations.

4.2 Iterative refinement as a fixed point procedure

While in general we do not yet have the theoretical tools to understand whether and how the slots of slot attention optimize a well-defined objective like the ELBO, we empirically find that the forward relative residual of the slots do appear stable (§5.2). This suggests that slot attention can be understood as an instance of a DEQ [6] that uses naive forward iteration to find the fixed point and backpropagates through the iteration to compute the gradient. While the theoretical convergence properties of DEQs are an open question that is beyond the scope of this paper, they have been reported to empirically converge to a fixed point and have successfully been trained with implicit differentiation. The novel implication of this to object-centric methods is that *any* root-finding solver and implicit gradient estimator can in principle be used to train slot attention, and we find that many combinations do train slot attention effectively in §5.2. This insight is one of our main contributions, but in the next section we describe a particular combination that we empirically found effective.

4.3 Implicit slot attention

We propose **implicit slot attention**: a method for training the state-of-the-art slot attention module [28] with the simplest and most effective method that we have empirically found for approximating the implicit gradient, which is its first-order Neumann approximation (Eq. 3). It can be implemented by simply differentiating the computation graph of applying the slot attention update *once* to the fixed point $\boldsymbol{\theta}_*^n$, where $\boldsymbol{\theta}_*^n$ is computed by simply iterating the slot attention module forward as usual, but *without* the gradient

Table 1: Complexity

	vanilla	implicit
time (forward)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
space (forward)	$\mathcal{O}(n)$	$\mathcal{O}(n)$
time (backward)	$\mathcal{O}(n)$	$\mathcal{O}(1)$
space (backward)	$\mathcal{O}(n)$	$\mathcal{O}(1)$.

```
def step(slots, k, v):
    # compute assignments given slots
    q = project_q(norm_slots(slots))
    k = k * (slot_size ** (-0.5))
    attn = F.softmax(torch.einsum('bkd,bqd->bkq', k, q), dim=-1)
    attn = attn / torch.sum(attn + epsilon, dim=-2, keepdim=True)
    # update slots given assignments
    updates = torch.einsum('bvq,bvd->bqd', attn, v)
    slots = gru(updates, slots)
    slots = slots + mlp(norm_mlp(slots))
    return slots
```

```
def iterate(f, x, num_iters):
    for _ in range(num_iters):
        x = f(x)
    return x

def forward(inputs, slots):
    inputs = norm_inputs(inputs)
    k, v  = project_k(inputs), project_v(inputs)
    slots = iterate(lambda z: step(z, k, v), slots, num_iterations)
    slots = step(slots.detach(), k, v)
    return slots
```

Figure 2: **Code.** The first order Neumann approximation to the implicit gradient adds only one additional line of Pytorch code [30] to the original forward function of slot attention, but yields substantial improvement of optimization. `attn` and `slots` correspond to ϕ and $\boldsymbol{\theta}$ in the text respectively.

tape, amounting to truncating the backpropagation. The time and space complexity of backpropagation for implicit slot attention compared to vanilla slot attention as a function of the number of slot attention iterations n, is shown in Table 1. While other methods for implicit differentiation are more complex to implement, this method requires only one additional line of code (Fig. 2).

5 Experiments

Our main hypothesis is that iterative refinement methods can be trained as DEQs, and that consequently implicit differentiation would improve their optimization. We test this premise by replacing the backward pass of the slot attention module in the state-of-the-art SLATE [37] with the first-order Neumann approximation of the implicit gradient. This requires only one additional line of code to the original slot attention implementation (Fig.2) and in one instance improves reconstruction mean-squared error by almost 7x.

5.1 Experimental setup

We summarize the SLATE architecture, datasets we considered, and relevant implementation details.

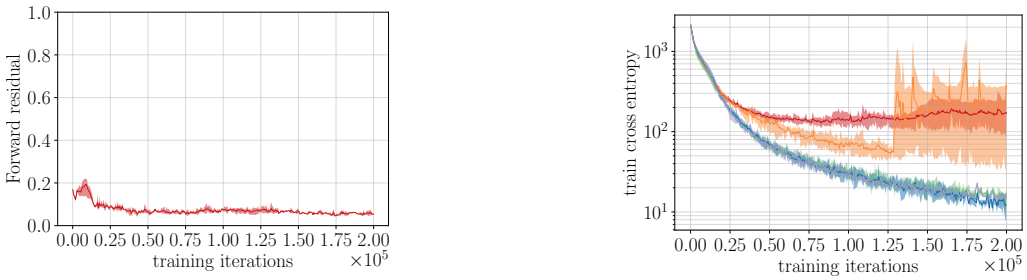
SLATE For image reconstruction, SLATE uses a discrete VAE [33] to compress an input image into a grid of discrete tokens. These tokens index into a codebook of latent code-vectors, which, after applying a learned position encoding, serve as the input to the slot attention module. An Image GPT decoder [9] is trained with a cross-entropy loss to autoregressively reconstruct the latent code-vectors, using the outputted slots from slot attention as queries and the latent code-vectors as keys/values. Gradients are blocked from flowing between the discrete VAE and the rest of the network (i.e. the slot attention module and the Image GPT decoder), but the entire system is trained simultaneously.

Data We consider three datasets: CLEVR-Mirror [37], Shapestacks [21], and COCO-2017 [27], the former two of which were used in the original SLATE paper. We obtained CLEVR-Mirror directly from the SLATE authors and used a 70-15-15 split for training, validation, and testing. We pooled all the data variants of Shapestacks together as Singh et al. [37] did and used the original train-validation-test splits. The COCO-2017 dataset was downloaded from FiftyOne and used the original train-validation-test splits.

Implementation details In all of our experiments we used the same model and training hyperparameters as those used for the ShapeStacks experiment from Singh et al. [37, Table 6], run on an A100 GPU. The only difference is in the image resolution (and consequently number of image tokens) because the largest image size we could train with was 128x128 used for their CLEVR experiment. However, as we show for resolutions of both 96x96 and 64x64, the improvement in optimization appears to be hold across image resolutions.

For a clean comparison, we simply integrated the official SLATE implementation[1] with the solvers and backward gradient hook from the official implementation of deep equilibrium models[2]. Besides implementing logging and evaluation code, as well as the first-order Neumann approximation (Eq. 3), we did not change anything in the above two official implementations otherwise. The SLATE implementation serves as the baseline in all our experiments.

[1] https://github.com/singhgautam/slate
[2] https://github.com/locuslab/deq



(a) $\frac{\|f_{\mathbf{w}}(\lambda,x)-\lambda\|}{\|f_{\mathbf{w}}(\lambda,x)\|}$

(b) Different solvers

Figure 3: **Can slot attention by trained as a fixed point operation with implicit differentiation?** (3a) The relative residual of the forward iteration of slot attention is close to zero, which motivates its treatment as a fixed point operation. (3b) The forward and backward computations of vanilla slot attention can be swapped out with different solvers, most of which result in the same improved optimization performance. Here we compare with *IB*, *BB*, *IN*, and *BN* variants of implicit differentiation across 4 seeds. Table 2 defines these acronyms.

5.2 Training slot attention with implicit differentiation

The first test to conduct is to see whether slot attention can be trained with implicit differentiation at all. *In summary:* ***Test:*** *Swap the forward pass with a different black-box solver, and train with implicit gradients computed via various gradient estimation methods for Eq. 7.* ***Hypothesis:*** *There exists a (forward solver, implicit gradient estimator) that optimizes cross entropy no worse than backpropagating through unrolled inference.* ***Result:*** *Yes, such a pair exists, and in fact it enables significantly better optimization.*

To conduct this test, we compare vanilla slot attention with four variants trained with implicit differentiation, shown in Table 2, labeled *IB*, *BB*, *IN*, and *BN*. *B* stands for the Broyden solver, used by Bai et al. [6]. We test configurations where the Broyden solver was both used to find the fixed point of slot attention and used to estimate the implicit gradient with Eq. 2. *N* stands for the first-order Neumann approximation for computing the explicit gradient (Eq. 3). Our hypothesis would be tentatively refuted if the cross-entropy learning curves of none of these variants converges as efficiently as the baseline. We train these methods to model the CLEVR-Mirrors dataset with an image resolution of 64x64. The slot attention baseline unrolls the slot attention cell for seven iterations and we also limit the maximum number of Broyden iterations to seven.

Table 2: Different instantiations of implicit differentiation

abbrv.	forward computation	gradient estimation
IB	iteration	Broyden
BB	Broyden	Broyden
IN	iteration	Neumann approximation
BN	Broyden	Neumann approximation

Results and analysis Figure 3 shows that not only is it possible to train slot attention with implicit differentiation, but three out of four of the implicit differentiation configurations from Table 2 optimize significantly better than vanilla slot attention. Whereas vanilla slot attention plateaus at a cross entropy of around 130, the *BB*, *IN*, and *BN* variants all achieve a 10x lower cross entropy loss. Moreover, *BB*, *IN*, and *BN* all follow the same learning curve, suggesting that the optimization improvement is largely due to whether we use implicit differentiation or not, rather than the choice of how we implement the implicit differentiation. The *IB* variant is less stable than the others, and we hypothesize that this can be explained by the tendency for the fixed point of Eq. 2 to become increasingly harder to estimate, an issue discussed in Bai et al. [7].

5.3 Does this improvement generalize across datasets?

We now test whether this improved optimization holds across different datasets. If the improvement was simply due to implicit differentiation being serendipitously useful for 64x64 images CLEVR-Mirror, then we should expect implicit differentiation to not help for other datasets. We focus our attention on *IN* variant to conduct this test because it is the minimal modification to the baseline slot attention for gaining the benefits of implicit differentiation, as it requires adding only one line of code

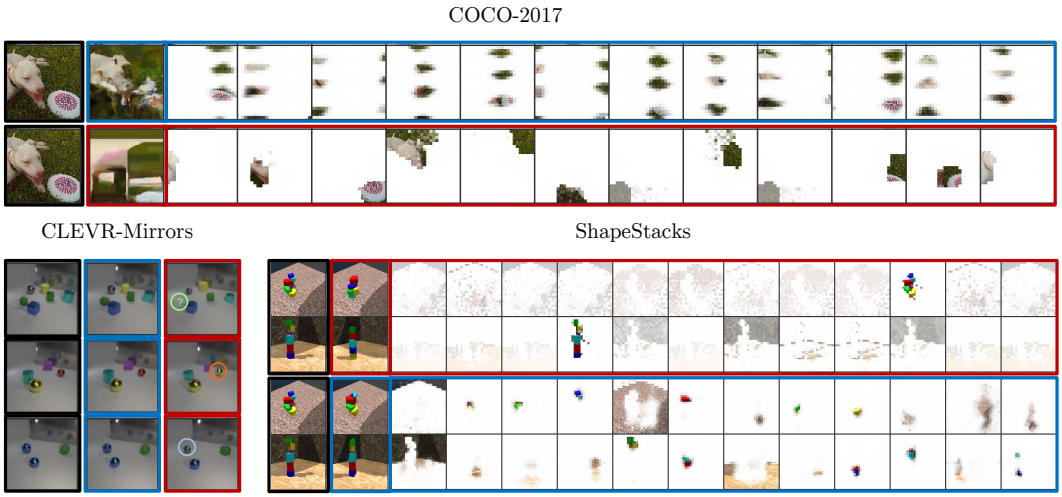


Figure 4: **Qualitative results.** Across three datasets, optimizing SLATE with implicit differentiation leads to improved image reconstructions through the slot bottleneck. Black borders indicate the ground truth image, blue border indicate our method, and red borders indicate vanilla SLATE. The rest of the panels visualize attention masks. In the CLEVR-Mirrors dataset, whereas vanilla SLATE misses objects, changes their size, or changes their color (indicated by the circles), implicit SLATE reconstructs the ground truth more faithfully.

on top of the baseline slot attention implementation and does not require implementing any black-box solvers. We henceforth refer to the *IN* variant as **implicit slot attention** (and correspondingly implicit SLATE), as described in §4.3. *In summary:* ***Test:*** *Apply implicit slot attention to CLEVR-Mirror, ShapeStacks, and COCO with 96x96 image resolution.* ***Hypothesis:*** *Implicit slot attention significantly improves optimization across all three datasets.* ***Result:*** *Yes it does.*

Results and quantitative analysis Using the two primary metrics used in Singh et al. [37], images generated by implicit SLATE achieve both lower pixel-wise mean-squared error and FID score [23]. The FID score was computed with the PyTorch-Ignite [14] library using the inception network from the PyTorch port of the FID official implementation. All methods were trained for 250k gradient steps. Table 3 compares the FID and MSE scores of the images that result from compressing the SLATE encoder's set of discrete tokens through the slot attention bottleneck, using Image-GPT to autoregressively re-generate these image tokens one by one, and using the discrete VAE decoder to render the generated image tokens. Implicit differentiation significantly improves the quantitative image reconstruction metrics of SLATE across the test sets of CLEVR-Mirrors, Shapestacks, and COCO. In the case of MSE for CLEVR, this is almost a 7x improvement.

Table 3: Quantitative metrics for image reconstruction through the slot bottleneck.

Data	Implicit	Vanilla
CLEVR (FID)	**22.19**	25.89
CLEVR (MSE)	**10.66**	67.04
COCO (FID)	**127.79**	147.48
COCO (MSE)	**1659.15**	1821.75
ShapeStacks (FID)	**34.2**	34.76
ShapeStacks (MSE)	**108.67**	312.14

Qualitative analysis The higher quantitiatve metrics also translate into better quality reconstructions on the test set, as shown in Figure 4. For CLEVR-Mirrors, vanilla SLATE sometimes drops or changes the appearance of objects, even in simple scenes with three objects. In contrast, the generations produced from implicit SLATE match the ground truth very closely. For Shapestacks, implicit SLATE consistently segments the scene into constituent objects. This is sometimes the case with vanilla SLATE on the training and validation set as well, but we observed for both of the seeds we ran for the final evaluation that vanilla SLATE produced degenerated attention maps where one slot captures the entire foreground, and the background is divided among the other slots. The visual complexity of the COCO dataset is much higher than either CLEVR-Mirrors and Shapestacks, and the reconstructions on the COCO dataset are quite poor, for both SLATE's discrete VAE and consequently for the reconstruction through the slot bottleneck. This highlights the gap that still exists between using the state-of-the-art in object-centric learning out-of-the-box and what the community

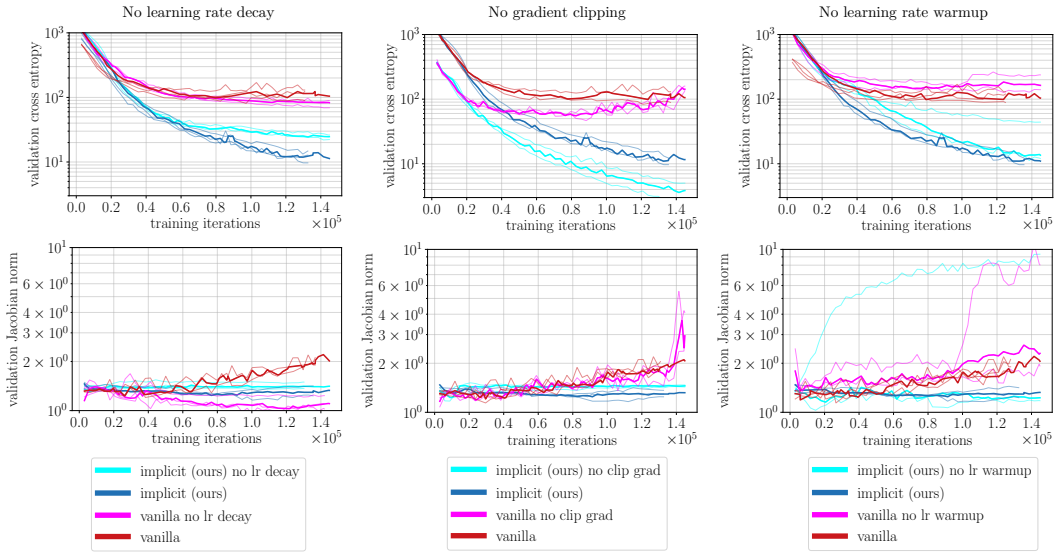
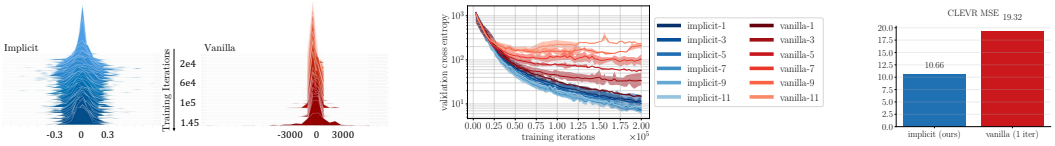


Figure 5: **Implicit differentiation removes the need for many optimization tricks.** We ablate three heuristically-motivated optimization tricks from both vanilla SLATE and our method. Whereas removing gradient clipping and learning rate warmup causes vanilla SLATE's training to become unstable, as indicated by the growth of the Jacobian norm of the slot attention cell, our method trains significantly more stably and can take advantage of the larger gradient steps.



(a) **No need for gradient clipping.** (b) **No sensitivity to hyperparameters.** (c) **c.f. 1 iteration**

Figure 6: (a) Without gradient clipping, our implicit differentiation technique keeps gradients small while backpropagating through the unrolled iterations causes gradients to explode. (b) Training with implicit differentiation also is not sensitive to the number of iterations with which to iterate the slot attention cell. (c) Using one iteration for vanilla slot attention trains more stably, but reconstructs more poorly than implicit slate.

may want these methods to do. The attention masks for both vanilla and implicit SLATE furthermore do not appear to correspond consistently to coherent objects in COCO but rather patches on the image that do not immediately seem to match with our human intuition of what constitutes a visual entity.

5.4 Can we simplify the need for optimization tricks?

To further understand the benefits of implicit differentiation, we then ask whether it stabilizes the training of slot attention without the need for optimization tricks like learning rate decay, gradient clipping, and learning warmup. *In summary:* ***Test:*** *Remove learning rate decay, gradient clipping, and learning warmup each from vanilla SLATE and our method.* ***Hypothesis:*** *Our method should not require these tricks to optimize stably.* ***Result:*** *No, it generally does not require these tricks.*

Fig. 5 shows that the ease and stability of training correlates with the spectral norm of the Jacobian of the slot attention cell. Decaying the learning rate regularizes the Jacobian norm from exploding, but it also hurts optimization performance for both our method and vanilla SLATE, as expected. When we remove gradient clipping the Jacobian norm of vanilla SLATE explodes, as do its gradients (Fig. 6a), whereas both stay stable for our method. Lastly, removing learning rate warmup also consistently makes vanilla SLATE's training unstable, whereas it only affects the stability of our method for one out of three seeds. Finally, Fig. 6b shows that implicit slot attention is not sensitive to the number of iterations with which to iterate the slot attention cell, whereas vanilla slot attention is, with more iterations being harder to train.

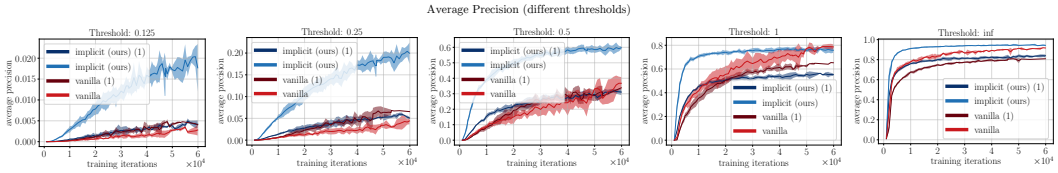


Figure 7: We outperform vanilla slot attention on object property prediction (see Locatello et al. [28]).

Figure 8: Implicit differentiation preserves the quality of predicted segmentations from Locatello et al. [28].

5.5 Does this mean that iterating slot attention for one iteration is enough?

One might be tempted to interpret Fig. 6b to suggest that fewer iterations of vanilla slot attention is sufficient to improve performance. This is not necessarily the case, as vanilla slot attention with one iteration still achieves 2x worse reconstruction MSE than implicit slot attention with seven iterations on CLEVR (Fig. 6c).

Futhermore, Fig. 7 shows that using only a single iteration is not enough to improve optimization of vanilla slot attention for the object property prediction task used by Locatello et al. [28]. We directly modified the released code from Locatello et al. [28], which used three iterations, to use implicit differentiation. Implicit slot attention can instead scale to as many forward iterations as needed.

5.6 Does implicit slot attention still produce intuitive masks with a different architecture?

We sought to check whether implicit differentiation still preserves the quality of the segmentation masks produced by the original slot attention architecture by Locatello et al. [28], which uses a spatial broadcast decoder [41] rather than a transformer decoder as SLATE does. It indeed does (Fig. 8).

6 Discussion

The connection we made in this paper between slot attention and deep equilibrium models also highlights various other properties about iterative refinement procedures of symmetric generative models that suggest connections to other areas of machine learning that are worth theoretically developing in the future. First is the connection to the literature on fast weights [5, 25, 35], if we interpret the slots as parameters that are modified during the inner optimization during execution time. Second is the connection to the literature on meta-learning [4, 13, 34, 38], if we interpret the slots as solutions to an inner optimization problem during execution time. The third is the connection to causality, if we interpret the independently generated and symmetrically processed slots as parameterizing independent causal mechanisms [22]. These different fields have their own conceptual and implementation tools that could potentially improve our understanding of how to build better object-centric models and of how objects could potentially be represented in the mind.

Conclusion We have proposed implicit differentiation for training iterative refinement procedures of symmetric generative models. Our results show clear signal that implicit differentiation can offer a significant optimization improvement over backpropagating through the unrolled iteration of slot attention, and potentially any other iterative refinement algorithm, with lower space and time complexity and only one additional line of code. Because it is so simple to apply implicit differentiation to any fixed point algorithm, we hope our work inspires future work to leverage tools developed for implicit differentiation for improving object-centric learning and methods for learning latent structure more broadly.

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. *arXiv preprint arXiv:1910.12430*, 2019.

[2] Luis B Almeida. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning*, pages 102–111. 1990.

[3] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[4] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

[5] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in Neural Information Processing Systems*, 29, 2016.

[6] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

[7] Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Stabilizing equilibrium models by jacobian regularization. *arXiv preprint arXiv:2106.14342*, 2021.

[8] Augustin-Louis Cauchy. Résumé d'un mémoir sur la mécanique céleste et sur un nouveau calcul appelé calcul des limites. *Oeuvres Complétes d'Augustun Cauchy*, 12(48-112):3, 1831.

[9] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020.

[10] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

[11] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[12] David Duvenaud, J. Zico Kolter, and Matthew Johnson. Deep implicit layers tutorial - neural ODEs, deep equilibirum models, and beyond. *Neural Information Processing Systems Tutorial*, 2020.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[14] V. Fomin, J. Anmol, S. Desroziers, J. Kriss, and A. Tejani. High-level library to help with training neural networks in pytorch. `https://github.com/pytorch/ignite`, 2020.

[15] Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. Fixed point networks: Implicit depth models with jacobian-free backprop. *arXiv preprint arXiv:2103.12803*, 2021.

[16] Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34, 2021.

[17] Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019.

[18] Klaus Greff, Sjoerd Van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. *arXiv preprint arXiv:1708.03498*, 2017.

[19] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pages 2424–2433. PMLR, 2019.

[20] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. On the binding problem in artificial neural networks. *arXiv preprint arXiv:2012.05208*, 2020.

[21] Oliver Groth, Fabian B Fuchs, Ingmar Posner, and Andrea Vedaldi. Shapestacks: Learning vision-based physical intuition for generalised object stacking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 702–717, 2018.

[22] Siyuan Guo, Viktor Tóth, Bernhard Schölkopf, and Ferenc Huszár. Causal de finetti: On the identification of invariant causal structure in exchangeable data. *arXiv preprint arXiv:2203.15756*, 2022.

[23] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[24] Zhichun Huang, Shaojie Bai, and J Zico Kolter. Implicit^2: Implicit layers for implicit representations. *Advances in Neural Information Processing Systems*, 34, 2021.

[25] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *Advances in Neural Information Processing Systems*, 34, 2021.

[26] Thomas Kipf, Gamaleldin F Elsayed, Aravindh Mahendran, Austin Stone, Sara Sabour, Georg Heigold, Rico Jonschkowski, Alexey Dosovitskiy, and Klaus Greff. Conditional object-centric learning from video. *arXiv preprint arXiv:2111.12594*, 2021.

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[28] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

[29] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[31] Fernando Pineda. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, pages 602–611, 1987.

[32] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. 2019.

[33] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021.

[34] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[35] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[36] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1723–1732. PMLR, 2019.

[37] Gautam Singh, Fei Deng, and Sungjin Ahn. Illiterate dall·e learns to compose. *arXiv preprint arXiv:2110.11405*, 2021.

[38] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[39] Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

[40] Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456. PMLR, 2020.

[41] Nicholas Watters, Loic Matthey, Christopher P Burgess, and Alexander Lerchner. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv preprint arXiv:1901.07017*, 2019.

[42] C. F. Jeff Wu. On the Convergence Properties of the EM Algorithm. *The Annals of Statistics*, 11 (1):95 – 103, 1983. doi: 10.1214/aos/1176346060. URL https://doi.org/10.1214/aos/1176346060.

[43] Yan Zhang, David W Zhang, Simon Lacoste-Julien, Gertjan J Burghouts, and Cees GM Snoek. Multiset-equivariant set prediction with approximate implicit differentiation. *arXiv preprint arXiv:2111.12193*, 2021.

[44] Daniel Zoran, Rishabh Kabra, Alexander Lerchner, and Danilo J Rezende. Parts: Unsupervised segmentation with slots, attention and independence maximization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10439–10447, 2021.

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]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (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? [N/A]
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] See Fig. 2.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (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]
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? [N/A]
   (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? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

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]