Unrolled Generative Adversarial Networks

Luke Metz[*]
Google Brain
lmetz@google.com

Ben Poole[†]
Stanford University
poole@cs.stanford.edu

David Pfau
Google DeepMind
pfau@google.com

Jascha Sohl-Dickstein
Google Brain
jaschasd@google.com

Abstract

We introduce a method to stabilize Generative Adversarial Networks (GANs) by defining the generator objective with respect to an unrolled optimization of the discriminator. This allows training to be adjusted between using the optimal discriminator in the generator's objective, which is ideal but infeasible in practice, and using the current value of the discriminator, which is often unstable and leads to poor solutions. We show how this technique solves the common problem of mode collapse, stabilizes training of GANs with complex recurrent generators, and increases diversity and coverage of the data distribution by the generator.

1 Introduction

The use of deep neural networks as generative models for complex data has made great advances in recent years. This success has been achieved through a surprising diversity of training losses and model architectures, including denoising autoencoders (Vincent et al., 2010), variational autoencoders (Kingma & Welling, 2013; Rezende et al., 2014; Gregor et al., 2015; Kulkarni et al., 2015; Burda et al., 2015; Kingma et al., 2016), generative stochastic networks (Alain et al., 2015), diffusion probabilistic models (Sohl-Dickstein et al., 2015), autoregressive models (Theis & Bethge, 2015; van den Oord et al., 2016a;b), real non-volume preserving transformations (Dinh et al., 2014; 2016), Helmholtz machines (Dayan et al., 1995; Bornschein et al., 2015), and Generative Adversarial Networks (GANs) (Goodfellow et al., 2014).

1.1 Generative Adversarial Networks

While most deep generative models are trained by maximizing log likelihood or a lower bound on log likelihood, GANs take a radically different approach that does not require inference or explicit calculation of the data likelihood. Instead, two models are used to solve a minimax game: a generator which samples data, and a discriminator which classifies the data as real or generated. In theory these models are capable of modeling an arbitrarily complex probability distribution. When using the optimal discriminator for a given class of generators, the original GAN proposed by Goodfellow et al. minimizes the Jensen-Shannon divergence between the data distribution and the generator, and extensions generalize this to a wider class of divergences (Nowozin et al., 2016; Sonderby et al., 2016; Poole et al., 2016).

The ability to train extremely flexible generating functions, without explicitly computing likelihoods or performing inference, and while targeting more mode-seeking divergences as made GANs extremely successful in image generation (Odena et al., 2016; Salimans et al., 2016; Radford et al., 2015), and image super resolution (Ledig et al., 2016). The flexibility of the GAN framework has also enabled a number of successful extensions of the technique, for instance for structured prediction (Reed et al., 2016a;b; Odena et al., 2016), training energy based models (Zhao et al., 2016), and combining the GAN loss with a mutual information loss (Chen et al., 2016).

---

[*]Work done as a member of the Google Brain Residency program (g.co/brainresidency)
[†]Work completed as part of a Google Brain internship

In practice, however, GANs suffer from many issues, particularly during training. One common failure mode involves the generator collapsing to produce only a single sample or a small family of very similar samples. Another involves the generator and discriminator oscillating during training, rather than converging to a fixed point. In addition, if one agent becomes much more powerful than the other, the learning signal to the other agent becomes useless, and the system does not learn. To train GANs many tricks must be employed, such as careful selection of architectures (Radford et al., 2015), minibatch discrimination (Salimans et al., 2016), and noise injection (Salimans et al., 2016; Sonderby et al., 2016). Even with these tricks the set of hyperparameters for which training is successful is generally very small in practice.

Once converged, the generative models produced by the GAN training procedure normally do not cover the whole distribution (Dumoulin et al., 2016; Che et al., 2016), even when targeting a mode-covering divergence such as KL. Additionally, because it is intractable to compute the GAN training loss, and because approximate measures of performance such as Parzen window estimates suffer from major flaws (Theis et al., 2016), evaluation of GAN performance is challenging. Currently, human judgement of sample quality is one of the leading metrics for evaluating GANs. In practice this metric does not take into account mode dropping if the number of modes is greater than the number of samples one is visualizing. In fact, the mode dropping problem generally helps visual sample quality as the model can choose to focus on only the most common modes. These common modes correspond, by definition, to more typical samples. Additionally, the generative model is able to allocate more expressive power to the modes it does cover than it would if it attempted to cover all modes.

1.2 DIFFERENTIATING THROUGH OPTIMIZATION

Many optimization schemes, including SGD, RMSProp (Tieleman & Hinton, 2012), and Adam (Kingma & Ba, 2014), consist of a sequence of differentiable updates to parameters. Gradients can be backpropagated through unrolled optimization updates in a similar fashion to backpropagation through a recurrent neural network. The parameters output by the optimizer can thus be included, in a differentiable way, in another objective (Maclaurin et al., 2015). This idea was first suggested for minimax problems in (Pearlmutter & Siskind, 2008), while (Zhang & Lesser, 2010) provided a theoretical analysis and experimental results on differentiating through a single step of gradient ascent for simple matrix games. Differentiating through unrolled optimization was first scaled to deep networks in (Maclaurin et al., 2015), where it was used for hyperparameter optimization. More recently, (Belanger & McCallum, 2015; Han et al., 2016; Andrychowicz et al., 2016) backpropagate through optimization procedures in contexts unrelated to GANs or minimax games.

In this work we address the challenges of unstable optimization and mode collapse in GANs by unrolling optimization of the discriminator objective during training.

2 METHOD

2.1 GENERATIVE ADVERSARIAL NETWORKS

The GAN learning problem is to find the optimal parameters θ_G^* for a generator function $G(z; \theta_G)$ in a minimax objective,

$$\theta_G^* = \operatorname*{argmin}_{\theta_G} \max_{\theta_D} f(\theta_G, \theta_D) \tag{1}$$

$$= \operatorname*{argmin}_{\theta_G} f(\theta_G, \theta_D^*(\theta_G)) \tag{2}$$

$$\theta_D^*(\theta_G) = \operatorname*{argmax}_{\theta_D} f(\theta_G, \theta_D), \tag{3}$$

where f is commonly chosen to be

$$f(\theta_G, \theta_D) = \mathbb{E}_{x \sim p_{data}}\left[\log\left(D\left(x; \theta_D\right)\right)\right] + \mathbb{E}_{z \sim \mathcal{N}(0,I)}\left[\log\left(1 - D\left(G\left(z; \theta_G\right); \theta_D\right)\right)\right]. \tag{4}$$

Here $x \in \mathcal{X}$ is the data variable, $z \in \mathcal{Z}$ is the latent variable, p_{data} is the data distribution, the discriminator $D(\cdot; \theta_D) : \mathcal{X} \to [0, 1]$ outputs the estimated probability that a sample x comes from the data distribution, θ_D and θ_G are the discriminator and generator parameters, and the generator function $G(\cdot; \theta_G) : \mathcal{Z} \to \mathcal{X}$ transforms a sample in the latent space into a sample in the data space.

For the minimax loss in Eq. 4, the optimal discriminator $D^*(x)$ is a known smooth function of the generator probability $p_G(x)$ (Goodfellow et al., 2014),

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}. \tag{5}$$

When the generator loss in Eq. 2 is rewritten directly in terms of $p_G(x)$ and Eq. 5 rather than θ_G and $\theta_D^*(\theta_G)$, then it is similarly a smooth function of $p_G(x)$. These smoothness guarantees are typically lost when $D(x; \theta_D)$ and $G(z; \theta_G)$ are drawn from parametric families. They nonetheless suggest that the true generator objective in Eq. 2 will often be well behaved, and is a desirable target for direct optimization.

Explicitly solving for the optimal discriminator parameters $\theta_D^*(\theta_G)$ for every update step of the generator G is computationally infeasible for discriminators based on neural networks. Therefore this minimax optimization problem is typically solved by alternating gradient descent on θ_G and ascent on θ_D.

The optimal solution $\theta^* = \{\theta_G^*, \theta_D^*\}$ is a fixed point of these iterative learning dynamics. Additionally, if $f(\theta_G, \theta_D)$ is convex in θ_G and concave in θ_D, then alternating gradient descent (ascent) trust region updates are guaranteed to converge to the fixed point, under certain additional weak assumptions (Juditsky et al., 2011). However in practice $f(\theta_G, \theta_D)$ is typically very far from convex in θ_G and concave in θ_D, and updates are not constrained in an appropriate way. As a result GAN training suffers from mode collapse, undamped oscillations, and other problems detailed in Section 1.1. In order to address these difficulties, we will introduce a surrogate objective function $f_K(\theta_G, \theta_D)$ for training the generator which more closely resembles the true generator objective $f(\theta_G, \theta_D^*(\theta_G))$.

2.2 Unrolling GANs

A local optimum of the discriminator parameters θ_D^* can be expressed as the fixed point of an iterative optimization procedure,

$$\theta_D^0 = \theta_D \tag{6}$$

$$\theta_D^{k+1} = \theta_D^k + \eta^k \frac{df(\theta_G, \theta_D^k)}{d\theta_D^k} \tag{7}$$

$$\theta_D^*(\theta_G) = \lim_{k \to \infty} \theta_D^k, \tag{8}$$

where η^k is the learning rate schedule. For clarity, we have expressed Eq. 7 as a full batch steepest gradient ascent equation. More sophisticated optimizers can be similarly unrolled. In our experiments we unroll Adam (Kingma & Ba, 2014).

By unrolling for K steps, we create a surrogate objective for the update of the generator,

$$f_K(\theta_G, \theta_D) = f(\theta_G, \theta_D^K(\theta_G, \theta_D)). \tag{9}$$

When $K = 0$ this objective corresponds exactly to the standard GAN objective, while as $K \to \infty$ it corresponds to the true generator objective function $f(\theta_G, \theta_D^*(G))$. By adjusting the number of unrolling steps K, we are thus able to interpolate between standard GAN training dynamics with their associated pathologies, and more costly gradient descent on the true generator loss.

2.3 Parameter Updates

The generator and discriminator parameter updates using this surrogate loss are

$$\theta_G \leftarrow \theta_G - \eta \frac{df_K(\theta_G, \theta_D)}{d\theta_G} \tag{10}$$

$$\theta_D \leftarrow \theta_D + \eta \frac{df(\theta_G, \theta_D)}{d\theta_D}. \tag{11}$$

For clarity we use full batch steepest gradient descent (ascent) with stepsize η above, while in experiments we instead use minibatch Adam for both updates. The gradient in Eq. 10 requires backpropagating through the optimization process in Eq. 7. A clear description of differentiation through
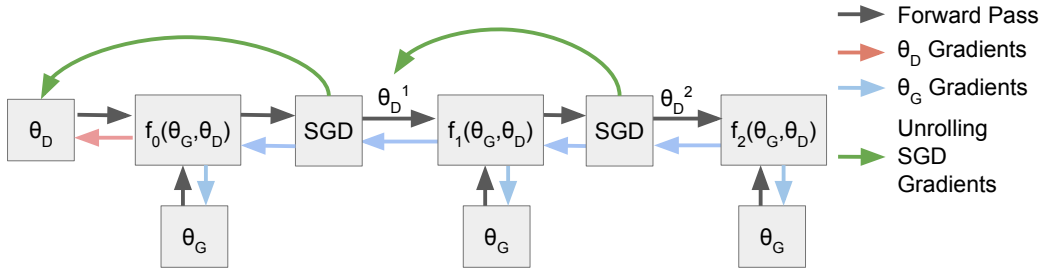
Figure 1: An illustration of the computation graph for an unrolled GAN with 3 unrolling steps. The generator update in Equation 10 involves backpropagating the generator gradient (blue arrows) through the unrolled optimization. Each step k in the unrolled optimization uses the gradients of f_k with respect to θ_D^k, as described in Equation 7 and indicated by the green arrows. The discriminator update in Equation 11 does not depend on the unrolled optimization (red arrow).

gradient descent is given as Algorithm 2 in (Maclaurin et al., 2015), though in practice the use of an automatic differentiation package means this step does not need to be programmed explicitly. A pictorial representation of these updates is provided in Figure 1.

It is important to distinguish this from an approach suggested in (Goodfellow et al., 2014), that several update steps of the discriminator parameters should be run before each single update step for the generator. In that approach, the update steps for both models are still gradient descent (ascent) with respect to *fixed* values of the other model parameters, rather than the surrogate loss we describe in Eq. 9. Performing K steps of discriminator update between each single step of generator update corresponds to updating the generator parameters θ_G using only the first term in Eq. 12 below.

2.4 THE MISSING GRADIENT TERM

To better understand the behavior of the surrogate loss $f_K(\theta_G, \theta_D)$, we examine its gradient with respect to the generator parameters θ_G,

$$\frac{\mathrm{d} f_K(\theta_G, \theta_D)}{\mathrm{d}\theta_G} = \frac{\partial f\left(\theta_G, \theta_D^K(\theta_G, \theta_D)\right)}{\partial \theta_G} + \frac{\partial f\left(\theta_G, \theta_D^K(\theta_G, \theta_D)\right)}{\partial \theta_D^K(\theta_G, \theta_D)} \frac{\mathrm{d}\theta_D^K(\theta_G, \theta_D)}{\mathrm{d}\theta_G}. \tag{12}$$

Standard GAN training corresponds exactly to updating the generator parameters using only the first term in this gradient, with $\theta_D^K(\theta_G, \theta_D)$ being the parameters resulting from the discriminator update step. An optimal generator for any fixed discriminator is a delta function at the x to which the discriminator assigns highest data probability. Therefore, in standard GAN training, each generator update step is a partial collapse towards a delta function.

The second term captures how the discriminator would react to a change in the generator. It reduces the tendency of the generator to engage in mode collapse. For instance, the second term reflects that as the generator collapses towards a delta function, the discriminator reacts and assigns lower probability to that state, increasing the generator loss. It therefore discourages the generator from collapsing, and may improve stability.

As $K \to \infty$, θ_D^K goes to a local optimum of f, where $\frac{\partial f}{\partial \theta_D^K} = 0$, and therefore the second term in Eq. 12 goes to 0 (Danskin, 1967). The gradient of the unrolled surrogate loss $f_K(\theta_G, \theta_D)$ with respect to θ_G is thus identical to the gradient of the standard GAN loss $f(\theta_G, \theta_D)$ both when $K = 0$ and when $K \to \infty$, where we take $K \to \infty$ to imply that in the standard GAN the discriminator is also fully optimized between each generator update. Between these two extremes, $f_K(\theta_G, \theta_D)$ captures additional information about the response of the discriminator to changes in the generator.

2.5 Consequences of the Surrogate Loss

GANs can be thought of as a game between the discriminator (D) and the generator (G). The agents take turns taking actions and updating their parameters until a Nash equilibrium is reached. The optimal action for D is to evaluate the probability ratio $\frac{p_{data}(x)}{p_G(x)+p_{data}(x)}$ for the generator's move x (Eq. 5). The optimal generator action is to move its mass to maximize this ratio.

The initial move for G will be to move as much mass as its parametric family and update step permits to the single point that maximizes the ratio of probability densities. The action D will then take is quite simple. It will track that point, and to the extent allowed by its own parametric family and update step assign low data probability to it, and uniform probability everywhere else. This cycle of G moving and D following will repeat forever or converge depending on the rate of change of the two agents. This is similar to the situation in simple matrix games like rock-paper-scissors and matching pennies, where alternating gradient descent (ascent) with a fixed learning rate is known not to converge (Singh et al., 2000; Bowling & Veloso, 2002).

In the unrolled case, however, this undesirable behavior no longer occurs. Now G's actions take into account how D will respond. In particular, G will try to make steps that D will have a hard time responding to. This extra information helps the generator spread its mass to make the next D step less effective instead of collapsing to a point.

In principle, a surrogate loss function could be used for both D and G. In the case of 1-step unrolled optimization this is known to lead to convergence for games in which gradient descent (ascent) fails (Zhang & Lesser, 2010). However, the motivation for using the surrogate generator loss in Section 2.2, of unrolling the *inner* of two nested min and max functions, does not apply to using a surrogate discriminator loss. Additionally, it is more common for the discriminator to overpower the generator than vice-versa when training a GAN. Giving more information to G by allowing it to 'see into the future' may thus help the two models be more balanced.

3 Experiments

In this section we demonstrate improved mode coverage and stability by applying this technique to five datasets of increasing complexity. Evaluation of generative models is a notoriously hard problem (Theis et al., 2016). As such the de facto standard in GAN literature has become sample quality as evaluated by a human and/or evaluated by a heuristic (Inception score for example, (Salimans et al., 2016)). While these evaluation metrics do a reasonable job capturing sample quality, they fail to capture sample diversity. In our first 2 experiments diversity is easily evaluated via visual inspection. In our later experiments this is not the case, and we will use a variety of methods to quantify coverage of samples. Our measures are individually strongly suggestive of unrolling reducing mode-collapse and improving stability, but none of them alone are conclusive. We believe that taken together however, they provide extremely compelling evidence for the advantages of unrolling.

When doing stochastic optimization, we must choose which minibatches to use in the unrolling updates in Eq. 7. We experimented with both a fixed minibatch and re-sampled minibatches for each unrolling step, and found it did not significantly impact the result. We use fixed minibatches for all experiments in this section.

We provide a reference implementation of this technique at github.com/poolio/unrolled_gan.

3.1 Mixture of Gaussians Dataset

To illustrate the impact of discriminator unrolling, we train a simple GAN architecture on a 2D mixture of 8 Gaussians arranged in a circle. For a detailed list of architecture and hyperparameters see Appendix A. Figure 2 shows the dynamics of this model through time. Without unrolling the generator rotates around the valid modes of the data distribution but is never able to spread out mass. When adding in unrolling steps G quickly learns to spread probability mass and the system converges to the data distribution.

In Appendix B we perform further experiments on this toy dataset. We explore how unrolling compares to historical averaging, and compares to using the unrolled discriminator to update the

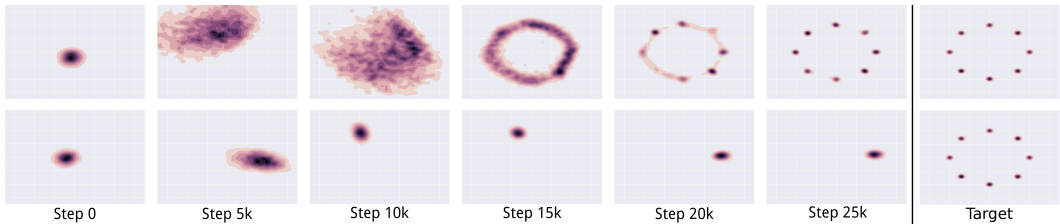| Step 0 | Step 5k | Step 10k | Step 15k | Step 20k | Step 25k | Target |

Figure 2: Unrolling the discriminator stabilizes GAN training on a toy 2D mixture of Gaussians dataset. Columns show a heatmap of the generator distribution after increasing numbers of training steps. The final column shows the data distribution. The top row shows training for a GAN with 10 unrolling steps. Its generator quickly spreads out and converges to the target distribution. The bottom row shows standard GAN training. The generator rotates through the modes of the data distribution. It never converges to a fixed distribution, and only ever assigns significant probability mass to a single data mode at once.

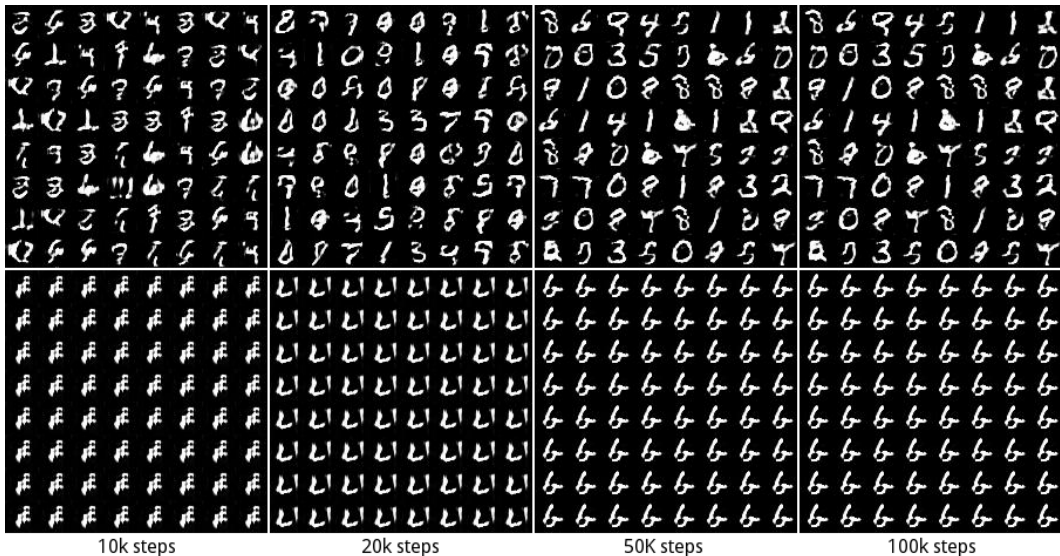| 10k steps | 20k steps | 50K steps | 100k steps |

Figure 3: Unrolled GAN training increases stability for an RNN generator and convolutional discriminator trained on MNIST. The top row was run with 20 unrolling steps. The bottom row is a standard GAN, with 0 unrolling steps. Images are samples from the generator after the indicated number of training steps.

generator, but without backpropagating through the generator. In both cases we find that the unrolled objective performs better.

3.2 PATHOLOGICAL MODEL WITH MISMATCHED GENERATOR AND DISCRIMINATOR

To evaluate the ability of this approach to improve trainability, we look to a traditionally challenging family of models to train – recurrent neural networks (RNNs). In this experiment we try to generate MNIST samples using an LSTM (Hochreiter & Schmidhuber, 1997). MNIST digits are 28x28 pixel images. At each timestep of the generator LSTM, it outputs one column of this image, so that after 28 timesteps it has output the entire sample. We use a convolutional neural network as the discriminator. See Appendix C for the full model and training details. Unlike in all previously successful GAN models, there is no symmetry between the generator and the discriminator in this task, resulting in a more complex power balance. Results can be seen in Figure 3. Once again, without unrolling the model quickly collapses, and rotates through a sequence of single modes. Instead of rotating spatially, it cycles through proto-digit like blobs. When running with unrolling steps the generator disperses and appears to cover the whole data distribution, as in the 2D example.

Discriminator Size	Unrolling steps	0	1	5	10
1/4 size of D compared to G	Modes generated	30.6 ± 20.73	65.4 ± 34.75	236.4 ± 63.30	**327.2 ± 74.67**
	KL(model $\|$data)	5.99 ± 0.42	5.911 ± 0.14	4.67 ± 0.43	**4.66 ± 0.46**
1/2 size of D compared to G	Modes generated	628.0 ± 140.9	523.6 ± 55.768	732.0 ± 44.98	**817.4 ± 37.91**
	KL(model $\|$data)	2.58 ± 0.751	2.44 ± 0.26	1.66 ± 0.090	**1.43 ± 0.12**

Table 1: Unrolled GANs cover more discrete modes when modeling a dataset with 1,000 data modes, corresponding to all combinations of three MNIST digits (10^3 digit combinations). The number of modes covered is given for different numbers of unrolling steps, and for two different architectures. The reverse KL divergence between model and data is also given. Standard error is provided for both measures.

3.3 MODE AND MANIFOLD COLLAPSE USING AUGMENTED MNIST

GANs suffer from two different types of model collapse – collapse to a subset of data modes, and collapse to a sub-manifold within the data distribution. In these experiments we isolate both effects using artificially constructed datasets, and demonstrate that unrolling can largely rescue both types of collapse.

3.3.1 DISCRETE MODE COLLAPSE

To explore the degree to which GANs drop discrete modes in a dataset, we use a technique similar to one from (Che et al., 2016). We construct a dataset by stacking three randomly chosen MNIST digits, so as to construct an RGB image with a different MNIST digit in each color channel. This new dataset has 1,000 distinct modes, corresponding to each combination of the ten MNIST classes in the three channels.

We train a GAN on this dataset, and generate samples from the trained model (25,600 samples for all experiments). We then compute the predicted class label of each color channel using a pre-trained MNIST classifier. To evaluate performance, we use two metrics: the number of modes for which the generator produced at least one sample, and the KL divergence between the model and the expected data distribution. Within this discrete label space, a KL divergence can be estimated tractably between the generated samples and the data distribution over classes, where the data distribution is a uniform distribution over all 1,000 classes.

As presented in Table 1, as the number of unrolling steps is increased, both mode coverage and reverse KL divergence improve. Contrary to (Che et al., 2016), we found that reasonably sized models (such as the one used in Section 3.4) covered all 1,000 modes even without unrolling. As such we use smaller convolutional GAN models. Details on the models used are provided in Appendix E.

We observe an additional interesting effect in this experiment. The benefits of unrolling increase as the discriminator size is reduced. We believe unrolling effectively increases the capacity of the discriminator. The unrolled discriminator can better react to any specific way in which the generator is producing non-data-like samples. When the discriminator is weak, the positive impact of unrolling is thus larger.

3.3.2 MANIFOLD COLLAPSE

In addition to discrete modes, we examine the effect of unrolling when modeling continuous manifolds. To get at this quantity, we constructed a dataset consisting of colored MNIST digits. Unlike in the previous experiment, a single MNIST digit was chosen, and then assigned a single monochromatic color. With a perfect generator, one should be able to recover the distribution of colors used to generate the digits. We use colored MNIST digits so that the generator also has to model the digits, which makes the task sufficiently complex that the generator is unable to perfectly solve it. The color of each digit is sampled from a 3D normal distribution. Details of this dataset are provided in Appendix F. We will examine the distribution of colors in the samples generated by the trained GAN. As will also be true in the CIFAR10 example in Section 3.4, the lack of diversity in generated colors is almost invisible using only visual inspection of the samples. Samples can be found in Appendix F.

Unrolling steps	0	1	5	10
JS divergence with 1/4 layer size	0.073 ± 0.0058	0.142 ± 0.028	$\mathbf{0.049 \pm 0.0021}$	0.075 ± 0.012
JS divergence with 1/2 layer size	0.095 ± 0.011	0.119 ± 0.010	$\mathbf{0.055 \pm 0.0049}$	0.074 ± 0.016
JS divergence with 1/1 layer size	0.034 ± 0.0034	0.050 ± 0.0026	0.027 ± 0.0028	$\mathbf{0.025 \pm 0.00076}$

Table 2: Unrolled GANs better model a continuous distribution. GANs are trained to model randomly colored MNIST digits, where the color is drawn from a Gaussian distribution. The JS divergence between the data and model distributions over digit colors is then reported, along with standard error in the JS divergence. More unrolling steps, and larger models, lead to better JS divergence.

Figure 4: Visual perception of sample quality and diversity is very similar for models trained with different numbers of unrolling steps. Actual sample diversity is higher with more unrolling steps. Each pane shows samples generated after training a model on CIFAR10 with 0, 1, 5, and 10 steps of unrolling.

In order to recover the color the GAN assigned to the digit, we used k-means with 2 clusters, to pick out the foreground color from the background. We then performed this transformation for both the training data and the generated images. Next we fit a Gaussian kernel density estimator to both distributions over digit colors. Finally, we computed the JS divergence between the model and data distributions over colors. Results can be found in Table 2 for several model sizes. Details of the models are provided in Appendix F.

In general, the best performing models are unrolled for 5-10 steps, and larger models perform better than smaller models. Counter-intuitively, taking 1 unrolling step seems to hurt this measure of diversity. We suspect that this is due to it introducing oscillatory dynamics into training. Taking more unrolling steps however leads to improved performance with unrolling.

3.4 IMAGE MODELING OF CIFAR10

Here we test our technique on a more traditional convolutional GAN architecture and task, similar to those used in (Radford et al., 2015; Salimans et al., 2016). In the previous experiments we tested models where the standard GAN training algorithm would not converge. In this section we improve a standard model by reducing its tendency to engage in mode collapse. We ran 4 configurations of this model, varying the number of unrolling steps to be 0, 1, 5, or 10. Each configuration was run 5 times with different random seeds. For full training details see Appendix D. Samples from each of the 4 configurations can be found in Figure 4. There is no obvious difference in visual quality across these model configurations. Visual inspection however provides only a poor measure of sample diversity.

By training with an unrolled discriminator, we expect to generate more diverse samples which more closely resemble the underlying data distribution. We introduce two techniques to examine sample diversity: inference via optimization, and pairwise distance distributions.

Unrolling Steps	0 steps	1 step	5 steps	10 steps
Average MSE	0.0231 ± 0.0024	0.0195 ± 0.0021	0.0200 ± 0.0023	**0.0181 ± 0.0018**
Percent Best Rank	0.63%	22.97%	15.31%	**61.09%**

Table 3: GANs trained with unrolling are better able to match images in the training set than standard GANs, likely due to mode dropping by the standard GAN. Results show the MSE between training images and the best reconstruction for a model with the given number of unrolling steps. The fraction of training images best reconstructed by a given model is given in the final column. The best reconstructions is found by optimizing the latent representation z to produce the closest matching pixel output $G(z; \theta_G)$. Results are averaged over all 5 runs of each model with different random seeds.

3.4.1 INFERENCE VIA OPTIMIZATION

Since likelihood cannot be tractably computed, over-fitting of GANs is typically tested by taking samples and computing the nearest-neighbor images in pixel space from the training data (Goodfellow et al., 2014). We will do the reverse, and measure the ability of the generative model to generate images that look like specific samples from the training data. If we did this by generating random samples from the model, we would need an exponentially large number of samples. We instead treat finding the nearest neighbor x_{nearest} to a target image x_{target} as an optimization task,

$$z_{\text{nearest}} = \operatorname*{argmin}_{z} ||G(z; \theta_G) - x_{\text{target}}||_2^2 \qquad (13)$$

$$x_{\text{nearest}} = G(z_{\text{nearest}}; \theta_G). \qquad (14)$$

This concept of backpropagating to generate images has been widely used in visualizing features from discriminative networks (Simonyan et al., 2013; Yosinski et al., 2015; Nguyen et al., 2016) and has been applied to explore the visual manifold of GANs in (Zhu et al., 2016).

We apply this technique to each of the models trained. We optimize with 3 random starts using LBFGS, which is the optimizer typically used in similar settings such as style transfer (Johnson et al., 2016; Champandard, 2016). Results comparing average mean squared errors between x_{nearest} and x_{target} in pixel space can be found in Table 3. In addition we compute the percent of images for which a certain configuration achieves the lowest loss when compared to the other configurations.

In the zero step case, there is poor reconstruction and less than 1% of the time does it obtain the lowest error of the 4 configurations. Taking 1 unrolling step results in a significant improvement in MSE. Taking 10 unrolling steps results in more modest improvement, but continues to reduce the reconstruction MSE.

To visually see this, we compare the result of the optimization process for 0, 1, 5, and 10 step configurations in Figure 5. To select for images where differences in behavior is most apparent, we sort the data by the absolute value of a fractional difference in MSE between the 0 and 10 step models, $\left| \frac{l_{0step} - l_{10step}}{\frac{1}{2}(l_{0step} + l_{10step})} \right|$. This highlights examples where either the 0 or 10 step model cannot accurately fit the data example but the other can. In Appendix G we show the same comparison for models initialized using different random seeds. Many of the zero step images are fuzzy and ill-defined suggesting that these images cannot be generated by the standard GAN generative model, and come from a dropped mode. As more unrolling steps are added, the outlines become more clear and well defined – the model covers more of the distribution and thus can recreate these samples.

3.4.2 PAIRWISE DISTANCES

A second complementary approach is to compare statistics of data samples to the corresponding statistics for samples generated by the various models. One particularly simple and relevant statistic is the distribution over pairwise distances between random pairs of samples. In the case of mode collapse, greater probability mass will be concentrated in smaller volumes, and the distribution over inter-sample distances should be skewed towards smaller distances. We sample random pairs of images from each model, as well as from the training data, and compute histograms of the ℓ_2 distances between those sample pairs. As illustrated in Figure 6, the standard GAN, with zero unrolling steps, has its probability mass skewed towards smaller ℓ_2 intersample distances, compared
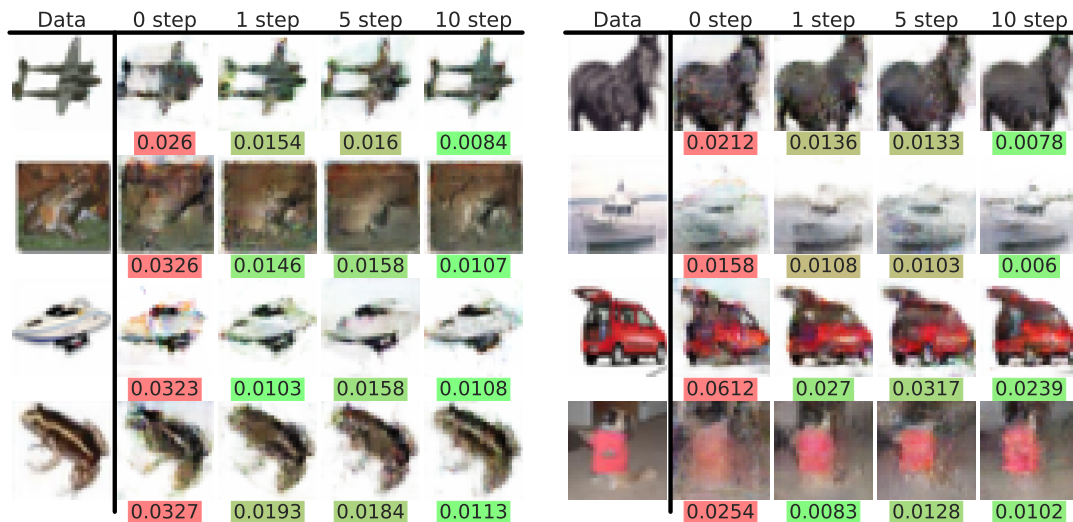
Figure 5: Training set images are more accurately reconstructed using GANs trained with unrolling than by a standard (0 step) GAN, likely due to mode dropping by the standard GAN. Raw data is on the left, and the optimized images to reach this target follow for 0, 1, 5, and 10 unrolling steps. The reconstruction MSE is listed below each sample. A random 1280 images where selected from the training set, and corresponding best reconstructions for each model were found via optimization. Shown here are the eight images with the largest absolute fractional difference between GANs trained with 0 and 10 unrolling steps.

to real data. As the number of unrolling steps is increased, the histograms over intersample distances increasingly come to resemble that for the data distribution. This is further evidence in support of unrolling decreasing the mode collapse behavior of GANs.

4 DISCUSSION

In this work we developed a method to stabilize GAN training and reduce mode collapse by defining the generator objective with respect to unrolled optimization of the discriminator. We then demonstrated the application of this method to several tasks, where it either rescued unstable training, or reduced the tendency of the model to drop regions of the data distribution.

The main drawback to this method is computational cost of each training step, which increases linearly with the number of unrolling steps. There is a tradeoff between better approximating the true generator loss and the computation required to make this estimate. Depending on the architecture, one unrolling step can be enough. In other more unstable models, such as the RNN case, more are needed to stabilize training. We have some initial positive results suggesting it may be sufficient to further perturb the training gradient in the same direction that a single unrolling step perturbs it. While this is more computationally efficient, further investigation is required.

The method presented here bridges some of the gap between theoretical and practical results for training of GANs. We believe developing better update rules for the generator and discriminator is an important line of work for GAN training. In this work we have only considered a small fraction of the design space. For instance, the approach could be extended to unroll G when updating D as well – letting the discriminator react to how the generator would move. It is also possible to unroll sequences of G and D updates. This would make updates that are recursive: G could react to maximize performance as if G and D had already updated.
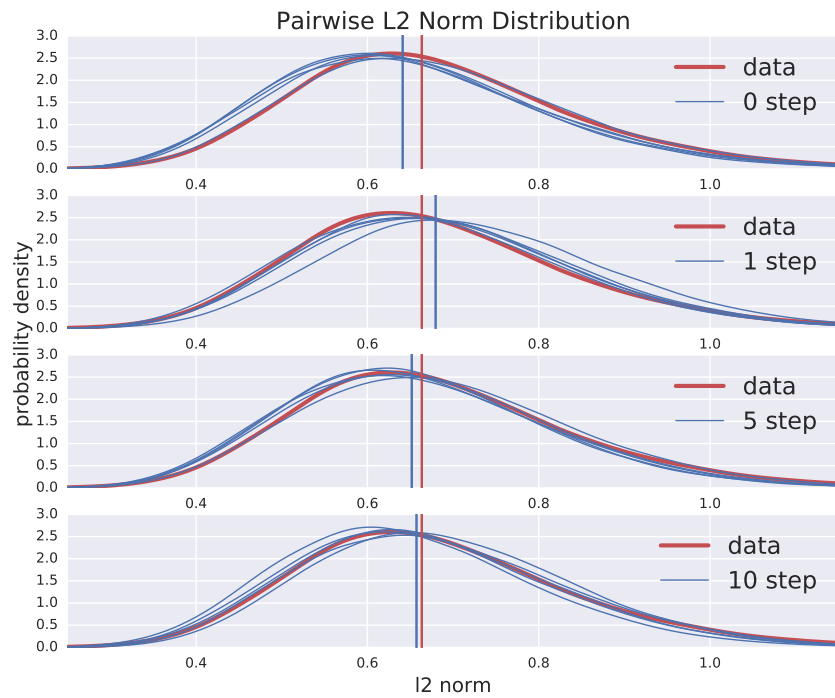
Figure 6: As the number of unrolling steps in GAN training is increased, the distribution of pairwise distances between model samples more closely resembles the same distribution for the data. Here we plot histograms of pairwise distances between randomly selected samples. The red line gives pairwise distances in the data, while each of the five blue lines in each plot represents a model trained with a different random seed. The vertical lines are the medians of each distribution.

REFERENCES

Guillaume Alain, Yoshua Bengio, Li Yao, Jason Yosinski, Eric Thibodeau-Laufer, Saizheng Zhang, and Pascal Vincent. Gsns : Generative stochastic networks. *arXiv preprint arXiv:1503.05571*, 2015.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *arXiv preprint arXiv:1606.04474*, 2016.

David Belanger and Andrew McCallum. Structured prediction energy networks. *arXiv preprint arXiv:1511.06350*, 2015.

Jorg Bornschein, Samira Shabanian, Asja Fischer, and Yoshua Bengio. Bidirectional helmholtz machines. *arXiv preprint arXiv:1506.03877*, 2015.

Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Alex J. Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*, 2016.

Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv: 1612.02136*, 2016.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*, 2016.

John M Danskin. *The theory of max-min and its application to weapons allocation problems*, volume 5. Springer Science & Business Media, 1967.

Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. *arXiv preprint arXiv:1605.08803*, 2016.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, volume 9, pp. 249–256, May 2010.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1462–1471, 2015. URL http://www.jmlr.org/proceedings/papers/v37/gregor15.html.

Tian Han, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. Alternating back-propagation for generator network, 2016. URL https://arxiv.org/abs/1606.08571.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015. URL `http://jmlr.org/proceedings/papers/v37/ioffe15.html`.

Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *arXiv preprint arXiv:1603.08155*, 2016.

Anatoli Juditsky, Arkadi Nemirovski, et al. First order methods for nonsmooth convex large-scale optimization, i: general purpose methods. *Optimization for Machine Learning*, pp. 121–148, 2011.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL `https://arxiv.org/abs/1312.6114`.

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. 2016.

Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network. *arXiv preprint arXiv:1503.03167*, 2015.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016. URL `https://arxiv.org/abs/1609.04802`.

Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. Gradient-based hyperparameter optimization through reversible learning, 2015.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *arXiv preprint arXiv:1605.09304*, 2016.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. *arXiv preprint arXiv:1606.00709*, 2016.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode ad in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.*, 30(2):7:1–7:36, March 2008. ISSN 0164-0925. doi: 10.1145/1330017.1330018. URL `http://doi.acm.org/10.1145/1330017.1330018`.

Ben Poole, Alexander A Alemi, Jascha Sohl-Dickstein, and Anelia Angelova. Improved generator objectives for gans. *arXiv preprint arXiv:1612.02780*, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *NIPS*, 2016a.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text-to-image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016b.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and variational inference in deep latent gaussian models. In *International Conference on Machine Learning*. Citeseer, 2014.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*, 2016.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Satinder Singh, Michael Kearns, and Yishay Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pp. 541–548. Morgan Kaufmann Publishers Inc., 2000.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 2256–2265, 2015. URL `http://arxiv.org/abs/1503.03585`.

Casper Kaae Sonderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszar. Amortised map inference for image super-resolution, 2016. URL `https://arxiv.org/abs/1610.04490v1`.

L. Theis and M. Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems 28*, Dec 2015. URL `http://arxiv.org/abs/1506.03478/`.

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, Apr 2016. URL `http://arxiv.org/abs/1511.01844`.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, abs/1601.06759, 2016a. URL `http://arxiv.org/abs/1601.06759`.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016b.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1756006.1953039`.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Chongjie Zhang and Victor R Lesser. Multi-agent learning with policy prediction. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. Generative visual manipulation on the natural image manifold. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.

Appendix

A   2D Gaussian Training Details

Network architecture and experimental details for the experiment in Section 3.1 are as follows:

The dataset is sampled from a mixture of 8 Gaussians of standard deviation 0.02. The means are equally spaced around a circle of radius 2.

The generator network consists of a fully connected network with 2 hidden layers of size 128 with relu activations followed by a linear projection to 2 dimensions. All weights are initialized to be orthogonal with scaling of 0.8.

The discriminator network first scales its input down by a factor of 4 (to roughly scale to (-1,1)), followed by 1 layer fully connected network with relu activations to a linear layer to of size 1 to act as the logit.

The generator minimizes $\mathcal{L}_G = \log(D(x)) + \log(1 - D(G(z)))$ and the discriminator minimizes $\mathcal{L}_D = -\log(D(x)) - \log(1 - D(G(z)))$ where x is sampled from the data distribution and $z \sim \mathcal{N}(0, I_{256})$. Both networks are optimized using Adam (Kingma & Ba, 2014) with a learning rate of 1e-4 and β_1=0.5.

The network is trained by alternating updates of the generator and the discriminator. One step consists of either G or D updating.

B   More Mixture of Gaussian Experiments

B.1   Effects of time delay / historical averaging

Another comparison we looked at was with regard to historical averaging based approaches. Recently similarly inspired approaches have been used in (Salimans et al., 2016) to stabilize training. For our study, we looked at taking an ensemble of discriminators over time.

First, we looked at taking an ensemble of the last N steps, as shown in Figure App.1.
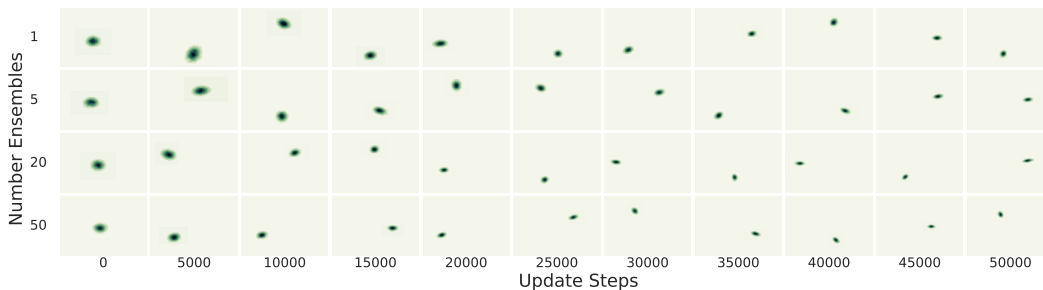

Figure App.1: Historical averaging does not visibly increase stability on the mixture of Gaussians task. Each row corresponds to an ensemble of discriminators which consists of the indicated number of immediately preceding discriminators. The columns correspond to different numbers of training steps.

To further explore this idea, we ran experiments with an ensemble of 5 discriminators, but with different periods between replacing discriminators in the ensemble. For example, if I sample at a rate of 100, it would take 500 steps to replace all 5 discriminators. Results can be seen in Figure App.2.

We observe that given longer and longer time delays, the model becomes less and less stable. We hypothesize that this is due to the initial shape of the discriminator loss surface. When training, the discriminator's estimates of probability densities are only accurate on regions where it was trained. When fixing this discriminator, we are removing the feedback between the generator exploitation
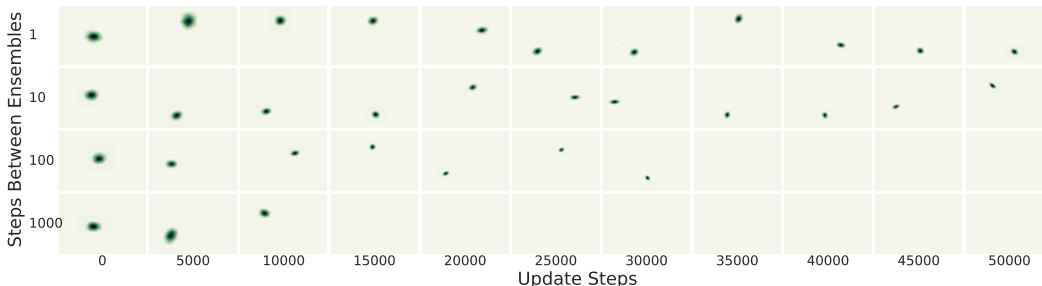
Figure App.2: Introducing longer time delays between the discriminator ensemble results in instability and probability distributions that are not in the window being visualized. The x axis is the number of weight updates and the y axis is how many steps to skip between discriminator updates when selecting the ensemble of 5 discriminators.

and the discriminators ability to move. As a result, the generator is able to exploit these fixed areas of poor performance for older discriminators in the ensemble. New discriminators (over)compensate for this, leading the system to diverge.

B.2 Effects of the second gradient

A second factor we analyzed is the effect of backpropagating the learning signal through the unrolling in Equation 12. We can turn on or off this backpropagation through the unrolling by introducing stop_gradient calls into our computation graph between each unrolling step. With the stop_gradient in place, the update signal corresponds only to the first term in Equation 12. We looked at 3 configurations: without stop_gradients; vanilla unrolled GAN, with stop gradients; and with stop gradients but taking the average over the k unrolling steps instead of taking the final value. Results can be see in Figure App.3.

We initially observed no difference between unrolling with and without the second gradient, as both required 3 unrolling steps to become stable. When the discriminator is unrolled to convergence, the second gradient term becomes zero. Due to the simplicity of the problem, we suspect that the discriminator nearly converged for every generator step, and the second gradient term was thus irrelevant.

To test this, we modified the dynamics to perform five generator steps for each discriminator update. Results are shown in Figure App.4. With the discriminator now kept out of equilibrium, successful training can be achieved with half as many unrolling steps when using both terms in the gradient than when only including the first term.

C RNN MNIST Training Details

The network architecture for the experiment in Section 3.2 is as follows:

The MNIST dataset is scaled to [-1, 1).

The generator first scales the 256D noise vector through a 256 unit fully connected layer with relu activation. This is then fed into the initial state of a 256D LSTM(Hochreiter & Schmidhuber, 1997) that runs 28 steps corresponding to the number of columns in MNIST. The resulting sequence of activations is projected through a fully connected layer with 28 outputs with a tanh activation function. All weights are initialized via the "Xavier" initialization (Glorot & Bengio, 2010). The forget bias on the LSTM is initialized to 1.

The discriminator network feeds the input into a Convolution(16, stride=2) followed by a Convolution(32, stride=2) followed by Convolution(32, stride=2). All convolutions have stride 2. As in (Radford et al., 2015) leaky rectifiers are used with a 0.3 leak. Batch normalization is applied after each layer (Ioffe & Szegedy, 2015). The resulting 4D tensor is then flattened and a linear projection is performed to a single scalar.

Unrolled GAN
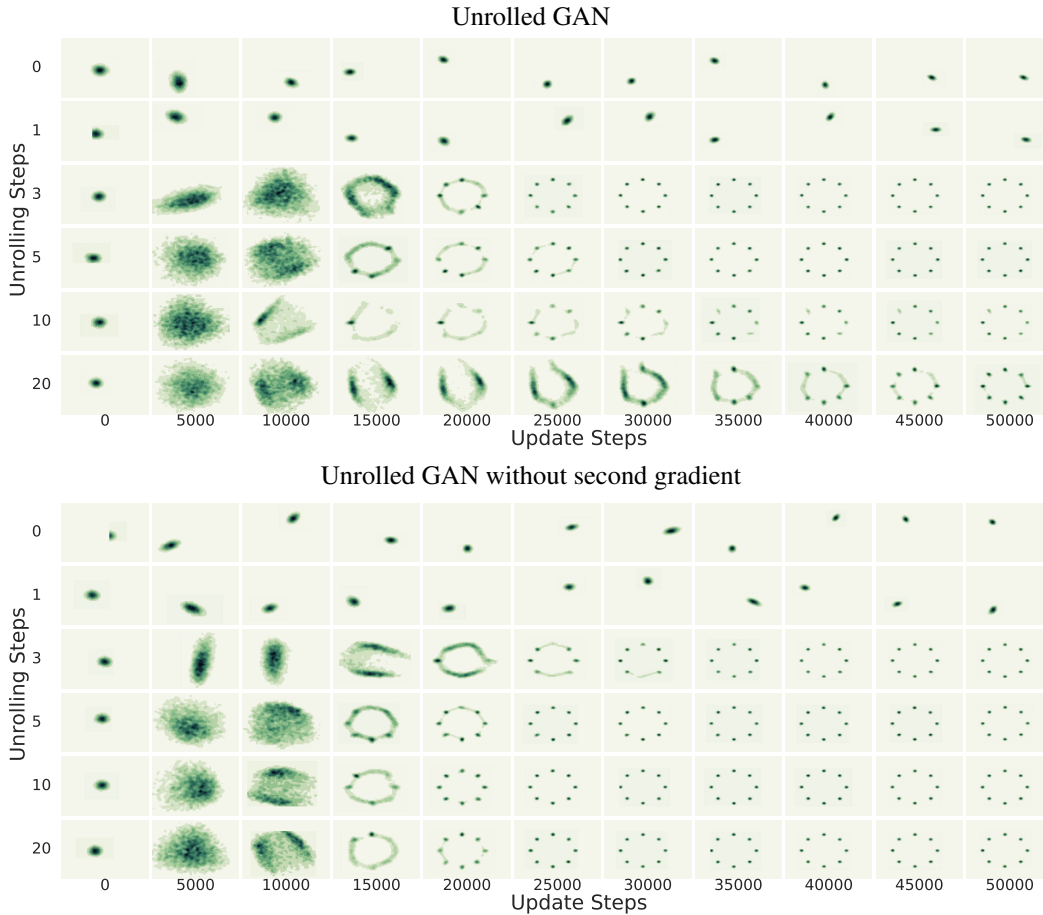

Unrolled GAN without second gradient

Figure App.3: If the discriminator remains nearly at its optimum during learning, then performance is nearly identical with and without the second gradient term in Equation 12. As shown in Figure App.4, when the discriminator lags behind the generator, backpropagating through unrolling aids convergence.

The generator network minimises $\mathcal{L}_G = \log(D(G(z)))$ and the discriminator minimizes $\mathcal{L}_D = \log(D(x)) + \log(1 - D(G(z)))$. Both networks are trained with Adam(Kingma & Ba, 2014) with learning rates of 1e-4 and β_1=0.5. The network is trained alternating updating the generator and the discriminator for 150k steps. One step consists of just 1 network update.

D CIFAR10/MNIST TRAINING DETAILS

The network architectures for the discriminator, generator, and encoder as as follows. All convolutions have a kernel size of 3x3 with batch normalization and leaky ReLU's with a 0.3 leak.

The generator network is defined as:

	number outputs	stride
Input: $z \sim \mathcal{N}(0, I_{256})$		
Fully connected	4 * 4 * 512	
Reshape to image 4,4,512		
Convolution	256	2
Convolution	128	2
Convolution	64	2
Convolution	1 or 3	1

17

Unrolled GAN with 5 G Steps per D

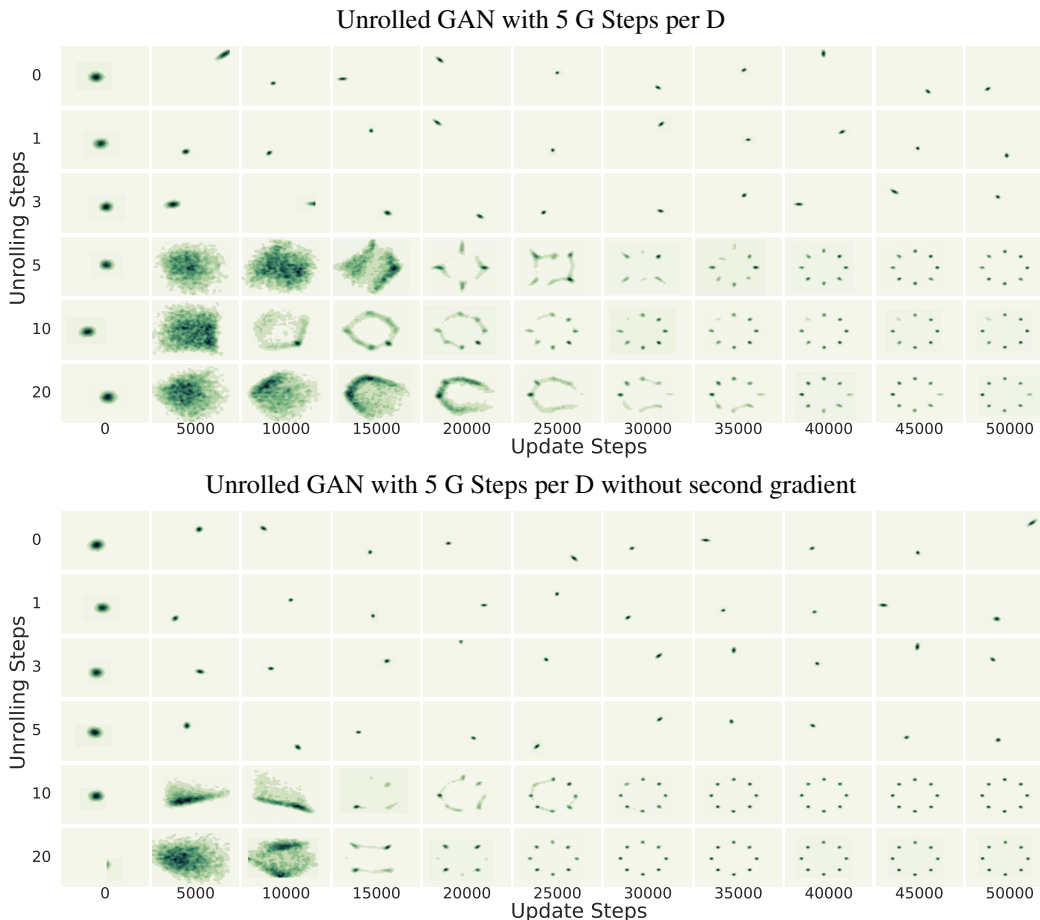Unrolled GAN with 5 G Steps per D without second gradient

Figure App.4: Backpropagating through the unrolling process aids convergence when the discriminator does not fully converge between generator updates. When taking 5 generator steps per discriminator step unrolling greatly increases stability, requiring only 5 unrolling steps to converge. Without the second gradient it requires 10 unrolling steps. Also see Figure App.3.

The discriminator network is defined as:

	number outputs	stride
Input: $x \sim p_{data}$ or G		
Transposed Convolution	64	2
Transposed Convolution	128	2
Transposed Convolution	256	2
Flatten		
Fully Connected	1	

The generator network minimises $\mathcal{L}_G = \log(D(G(z)))$ and the discriminator minimizes $\mathcal{L}_D = \log(D(x)) + \log(1 - D(G(z)))$. The networks are trained with Adam with a generator learning rate of 1e-4, and a discriminator learning rate of 2e-4. The network is trained alternating updating the generator and the discriminator for 100k steps. One step consists of just 1 network update.

E   1000 CLASS MNIST

	number outputs	stride
Input: $z \sim \mathcal{N}(0, I_{256})$		
Fully connected	4 * 4 * 64	
Reshape to image 4,4,64		
Convolution	32	2
Convolution	16	2
Convolution	8	2
Convolution	3	1

The discriminator network is parametrized by a size X and is defined as follows. In our tests, we used X of 1/4 and 1/2.

	number outputs	stride
Input: $x \sim p_{data}$ or G		
Transposed Convolution	8*X	2
Transposed Convolution	16*X	2
Transposed Convolution	32*X	2
Flatten		
Fully Connected	1	

F   COLORED MNIST DATASET

F.1   DATASET

To generate this dataset we first took the mnist digit, I, scaled between 0 and 1. For each image we sample a color, C, normally distributed with mean=0 and std=0.5. To generate a colored digit between (-1, 1) we do $I * C + (I - 1)$. Finally, we add a small amount of pixel independent noise sampled from a normal distribution with std=0.2, and the resulting values are cliped between (-1, 1). When visualized, this generates images and samples that can be seen in figure App.5. Once again it is very hard to visually see differences in sample diversity when comparing the 128 and the 512 sized models.

Figure App.5: Right: samples from the data distribution. Middle: Samples from 1/4 size model with 0 look ahead steps (worst diversity). Left: Samples from 1/1 size model with 10 look ahead steps (most diversity).

F.2   MODELS

The models used in this section are parametrized by a variable X to control capacity. A value of X=1 is same architecture used in the cifar10 experiments. We used 1/4, 1/2 and 1 as these values.

The generator network is defined as:

	number outputs	stride
Input: $z \sim \mathcal{N}(0, I_{256})$		
Fully connected	4 * 4 * 512*X	
Reshape to image 4,4,512*X		
Convolution	256*X	2
Convolution	128*X	2
Convolution	64*X	2
Convolution	3	1

The discriminator network is defined as:

	number outputs	stride
Input: $x \sim p_{data}$ or G		
Transposed Convolution	64*X	2
Transposed Convolution	128*X	2
Transposed Convolution	256*X	2
Flatten		
Fully Connected	1	

G   OPTIMIZATION BASED VISUALIZATIONS

More examples of model based optimization. We performed 5 runs with different seeds of each of of the unrolling steps configuration. Bellow are comparisons for each run index. Ideally this would be a many to many comparison, but for space efficiency we grouped the runs by the index in which they were run.
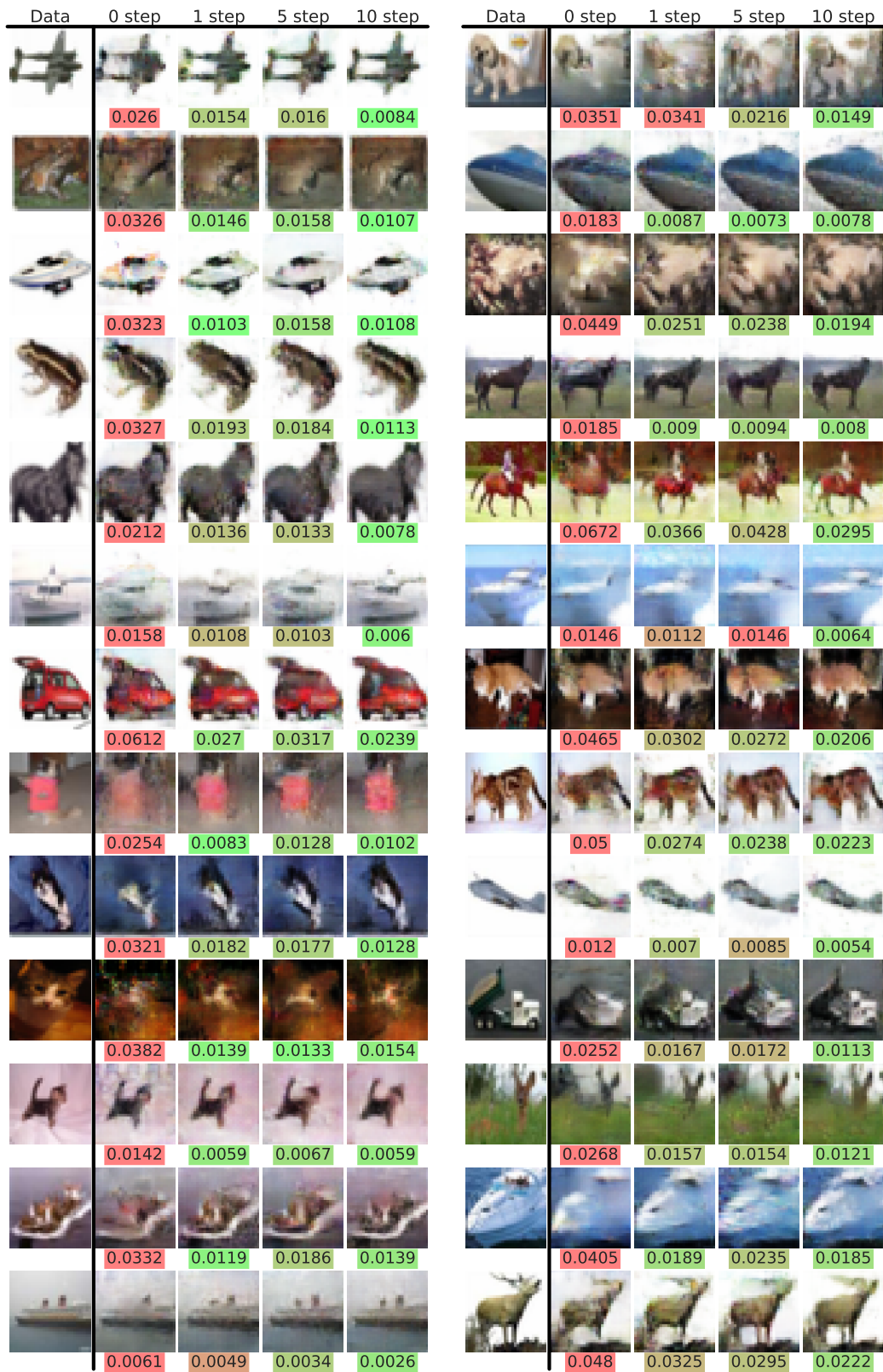
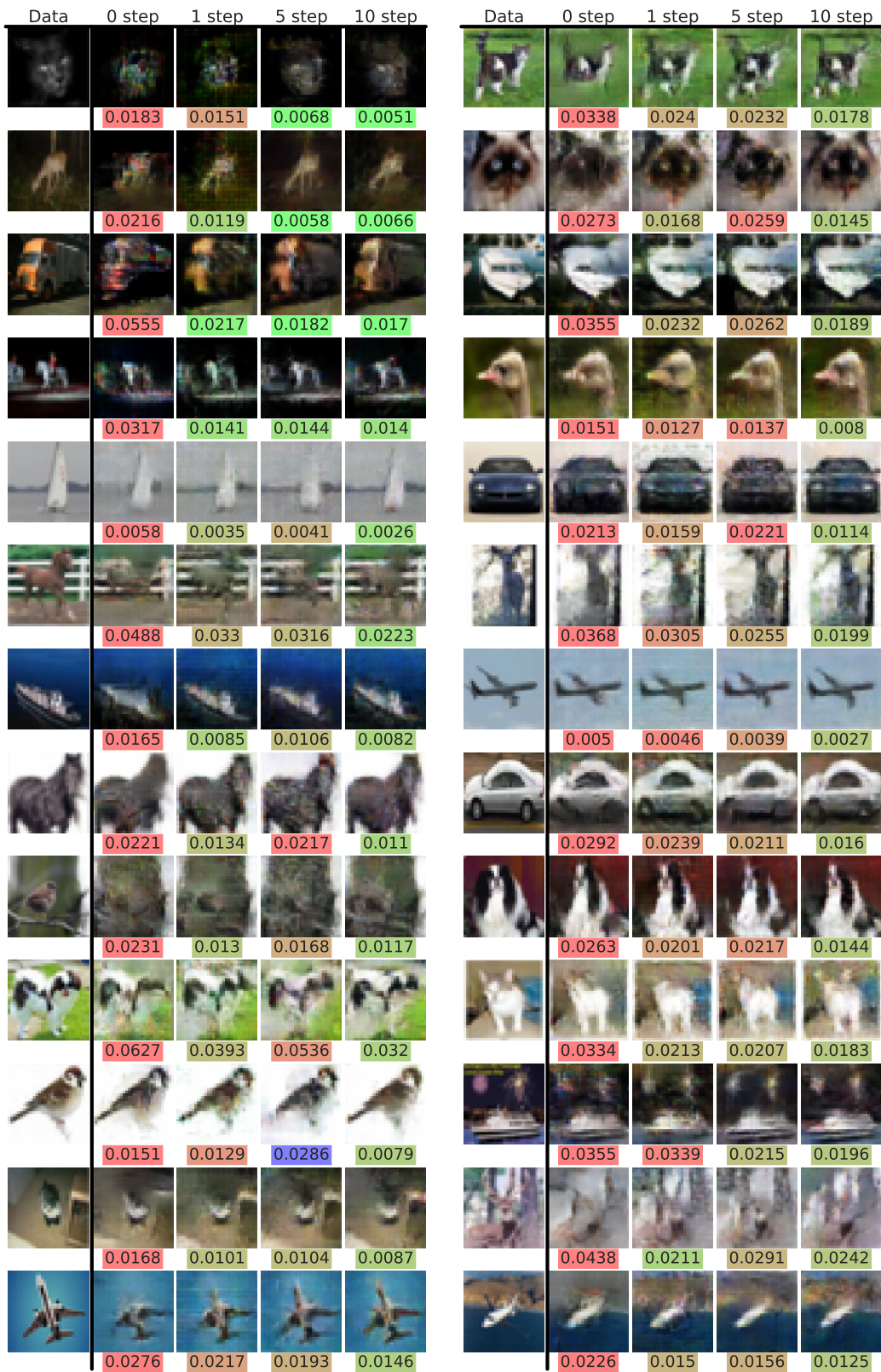Figure App.6: Samples from 1/5 with different random seeds.

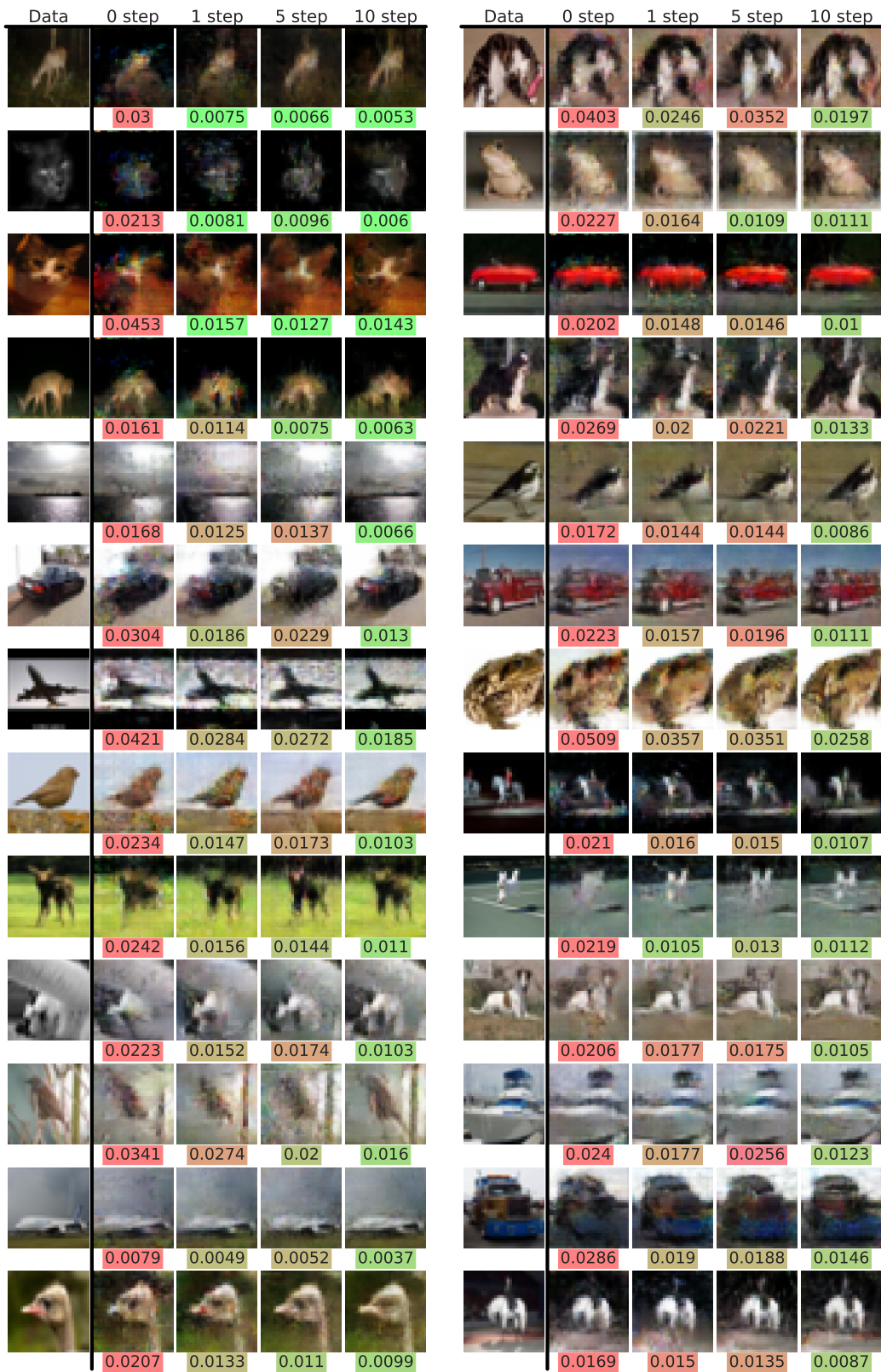Figure App.7: Samples from 2/5 with different random seeds.

Figure App.8: Samples from 3/5 with different random seeds.
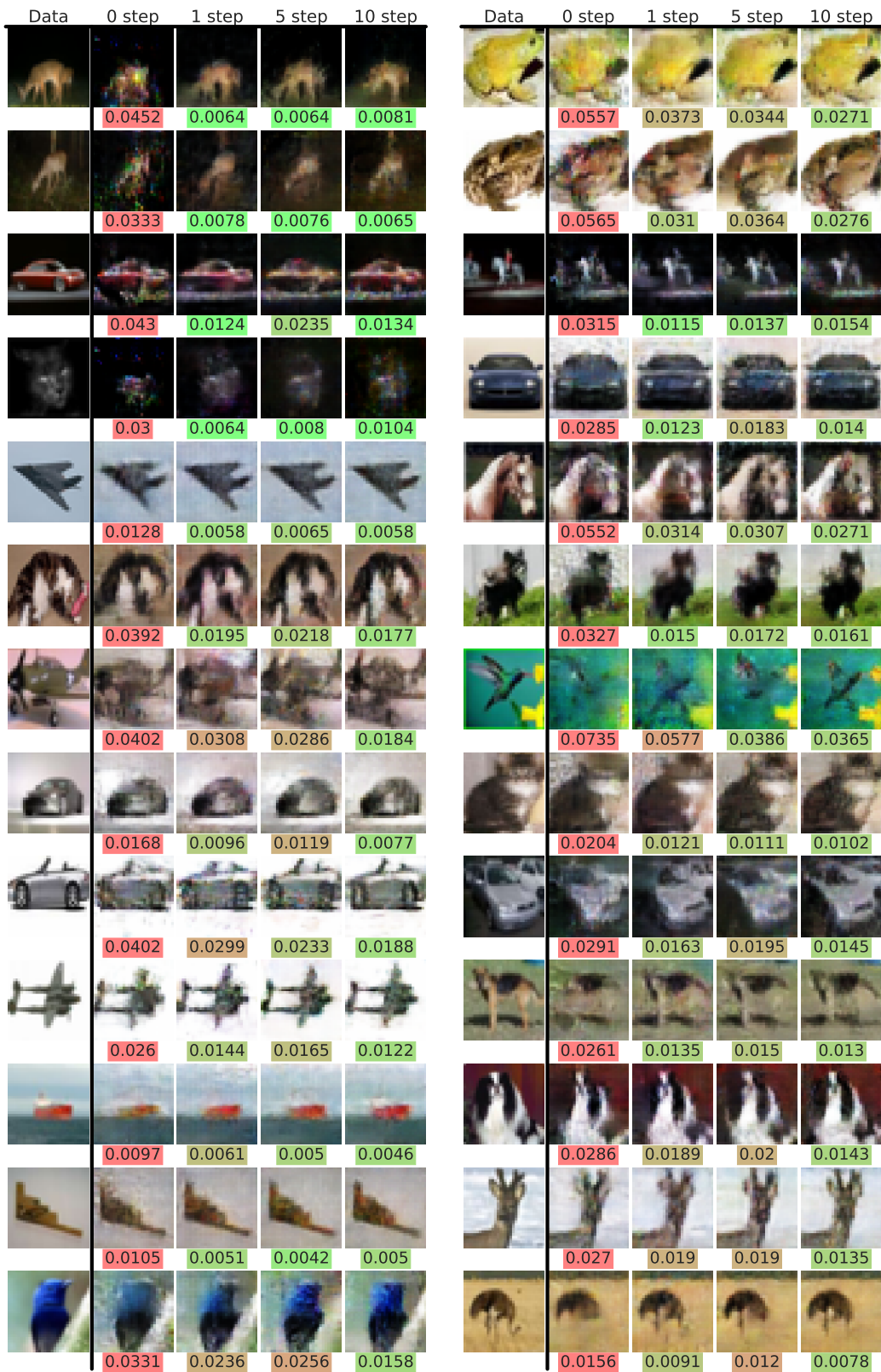
Figure App.9: Samples from 4/5 with different random seeds.

Figure App.10: Samples from 5/5 with different random seeds.