Improved Training of Wasserstein GANs

Ishaan Gulrajani[1], **Faruk Ahmed**[1], **Martin Arjovsky**[2], **Vincent Dumoulin**[1], **Aaron Courville**[1,3]
[1] Montreal Institute for Learning Algorithms
[2] Courant Institute of Mathematical Sciences
[3] CIFAR Fellow
igul222@gmail.com
{faruk.ahmed,vincent.dumoulin,aaron.courville}@umontreal.ca
ma4371@nyu.edu

Abstract

Generative Adversarial Networks (GANs) are powerful generative models, but suffer from training instability. The recently proposed Wasserstein GAN (WGAN) makes significant progress toward stable training of GANs, but can still generate low-quality samples or fail to converge in some settings. We find that these training failures are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to pathological behavior. We propose an alternative method for enforcing the Lipschitz constraint: instead of clipping weights, penalize the norm of the gradient of the critic with respect to its input. Our proposed method converges faster and generates higher-quality samples than WGAN with weight clipping. Finally, our method enables very stable GAN training: for the first time, we can train a wide variety of GAN architectures with almost no hyperparameter tuning, including 101-layer ResNets and language models over discrete data.[1]

1 Introduction

Generative Adversarial Networks (GANs) are a powerful class of generative models that cast the generative modeling problem as a game between two adversary networks: the generator network produces synthetic data given some noise source and the discriminator network discriminates between the generator's output and true data. GANs typically produce very visually appealing samples, but are often very hard to train, and much of the recent work on the subject (Salimans et al., 2016; Arjovsky et al., 2017; Nowozin et al., 2016; Poole et al., 2016; Metz et al., 2016) has been devoted to finding ways of stabilizing training. Despite this, consistently stable training of GANs remains an open problem.

In particular, Arjovsky & Bottou (2017) provide an insightful analysis of the convergence properties of the value function being optimized by GANs. Their proposed alternative, named Wasserstein GAN (WGAN) (Arjovsky et al., 2017), leverages the Wasserstein distance to produce a value function which has better theoretical properties than Jensen-Shannon divergence-based value functions. This new value function gives rise to the additional requirement that the discriminator (referred to in that work as the critic) must lie within the space of 1-Lipschitz functions, which the authors choose to enforce through weight clipping.

In this paper, we propose an alternative to weight clipping in the WGAN discriminator. Our contributions are as follows:

1. Through experiments on toy datasets, we outline the ways in which weight clipping in the discriminator can lead to pathological behavior which hurts stability and performance.

[1] Code for all of our models is available at `https://github.com/igul222/improved_wgan_training`.

2. We propose *WGAN with gradient penalty*, which does not suffer from the same issues.
3. We show that our method converges faster and generates higher-quality samples than standard WGAN.
4. We show that our method enables very stable GAN training: with almost no hyperparameter tuning, we can successfully train a wide variety of difficult GAN architectures for image generation and language modeling.

2 Background

2.1 Generative adversarial networks

The GAN training strategy is to define a game between two competing networks. The *generator* network maps a source of noise to the input space. The *discriminator* network receives either a generated sample or a true data sample and must distinguish between the two. The generator is trained to fool the discriminator.

More formally we can express the game between the generator G and the discriminator D with the minimax objective:

$$\min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log(D(\boldsymbol{x}))] + \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(1 - D(\tilde{\boldsymbol{x}}))]. \tag{1}$$

where $\mathbb{P}_r$ is the data distribution and $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}), \quad \boldsymbol{z} \sim p(\boldsymbol{z})$ (the input $\boldsymbol{z}$ to the generator is sampled from some simple noise distribution, such as the uniform distribution or a spherical Gaussian distribution).

If the discriminator is trained to optimality before each generator parameter update, then minimizing the value function amounts to minimizing the Jensen-Shannon divergence between the data and model distributions on $\boldsymbol{x}$. Doing so is expensive and often leads to vanishing gradients as the discriminator saturates; in practice, this requirement is relaxed, and the generator and the discriminator are updated simultaneously. The consequence of this relaxation is that generator updates minimize a stochastic lower-bound to the JS-divergence (Goodfellow, 2014; Poole et al., 2016). Minimizing a lower bound can lead to meaningless gradient updates, since pushing down the lower bound doesn't imply that the loss is actually decreasing, even as the bound goes to 0. This inherent problem in GANs of trading off unreliable updates and vanishing gradients is one of the main causes of GAN instability, as thoroughly explored in Arjovsky & Bottou (2017). As shown in Arjovsky et al. (2017), Wasserstein GANs don't suffer from this inherent problem.

Again, the GAN value function by itself is hard to optimize: a discriminator confident in its predictions sees its gradient with respect to its input vanish, which is especially hurtful early on in training. This is why training the discriminator closer to optimality typically degrades the training procedure. To circumvent this difficulty, the generator is usually trained to maximize $\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[\log(D(\tilde{\boldsymbol{x}}))]$ instead. However, this loss function was shown to misbehave as well, in the presence of a good discriminator (Arjovsky & Bottou, 2017).

2.2 Wasserstein GANs

At a more fundamental level, Arjovsky et al. (2017) argue that the Jensen-Shannon divergence, along with other common distances and divergences, are potentially not continuous and thus do not provide a usable gradient for the generator.

An alternative is proposed in the form of the *Earth-Mover* (also called Wasserstein-1) distance $W(q, p)$, which is informally defined as the minimum cost of transporting mass in order to transform the distribution q into the distribution p (where the cost is mass times transport distance). The Earth-Mover distance is shown to have the desirable property that under mild assumptions it is continuous everywhere and differentiable almost everywhere.

The value function for a WGAN is constructed by applying the Kantorovich-Rubinstein duality (Villani, 2008) to obtain

$$\min_G \max_{D\in\mathcal{D}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] - \mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\big[D(\tilde{\boldsymbol{x}})\big] \tag{2}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions and $\mathbb{P}_g$ is once again the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z}), \quad \boldsymbol{z} \sim p(\boldsymbol{z})$. In that case, under an optimal discriminator (called "critic" in the paper, since it's not trained to classify), minimizing the value function with respect to the generator parameters minimizes the Earth-Mover distance between $\mathbb{P}_g$ and $\mathbb{P}_r$.

The WGAN value function results in a critic function whose gradient with respect to its input is much better behaved than its GAN counterpart; consequently optimization of the generator is made easier. Additionally, WGAN has the very desirable property that its value function correlates with the quality of samples, which is not the case for GANs.

An open question is how to effectively enforce the Lipschitz constraint on the critic. Arjovsky et al. (2017) propose to clip the weights of the critic to lie within a compact space $[-c, c]$. The set of functions satisfying this constraint is a subset of the k-Lipschitz functions for some k which depends on c and the critic architecture. In the following sections, we demonstrate some of the issues with this approach and propose an alternative.

2.3 Properties of the optimal WGAN critic

In order to understand why weight clipping is problematic in a WGAN critic, as well as to motivate our approach, we highlight some properties of the optimal critic in the WGAN framework. We more formally state and prove these in Lemma 1 of the Appendix.

If the optimal critic under the Kantorovich-Rubinstein dual D^* is differentiable, and x is a point from our generator distribution $\mathbb{P}_g$, then there is a point y sampled from the true distribution $\mathbb{P}_r$ such that the gradient of D^* at all points $x_t = (1-t)x + ty$ on a straight line between x and y points directly towards y, meaning $\nabla D^*(x_t) = \frac{y - x_t}{\|y - x_t\|}$.

This implies that the optimal WGAN critic has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$.

3 Difficulties with weight constraints

We find that weight clipping in WGAN leads to optimization difficulties, and that even when optimization succeeds the resulting critic can have a pathological value surface. We explain these problems below and demonstrate their effects.

Although we present experiments while using the specific form of weight constraint implemented in Arjovsky et al. (2017) (hard clipping of the magnitude of each weight), we have also experimented with other implementations of weight constraints (L2 norm clipping, weight normalization, etc.), as well as soft constraints (L1 and L2 weight decay, etc.) and found that they exhibit similar problems.

These are some of the possible mechanisms by which weight clipping hurts WGAN training, but we do not claim that each one always occurs in practice, nor that they are the only such mechanisms. In particular, batch normalization in WGAN interacts with weight clipping in complicated ways which we do not investigate in detail. Sometimes it helps, but at other times (especially with very deep networks) optimization is still difficult. We believe that the problems with batch-normalized WGAN critics are similar to the ones we investigate in this work, and observe in our experiments that our method can successfully train architectures which WGAN with BN cannot.

3.1 Capacity underuse

Optimizing the critic subject to a weight constraint is only an approximation that amounts to optimizing over a small subset of all k-Lipschitz functions. We find that for many problems, even when this optimization converges, the solutions the critic converges to have very different loss surfaces (and hence provide very different gradients) from the optimal one.

This arises from the fact that the optimal critic under the WGAN loss function has gradient with norm 1 almost everywhere (see subsection 2.3). However, under a weight-clipping constraint, most neural network architectures can only attain their maximum gradient norm of k when they learn extremely simple functions. *Consequently, implementing a k-Lipschitz constraint via weight clipping biases the critic toward much simpler functions.*

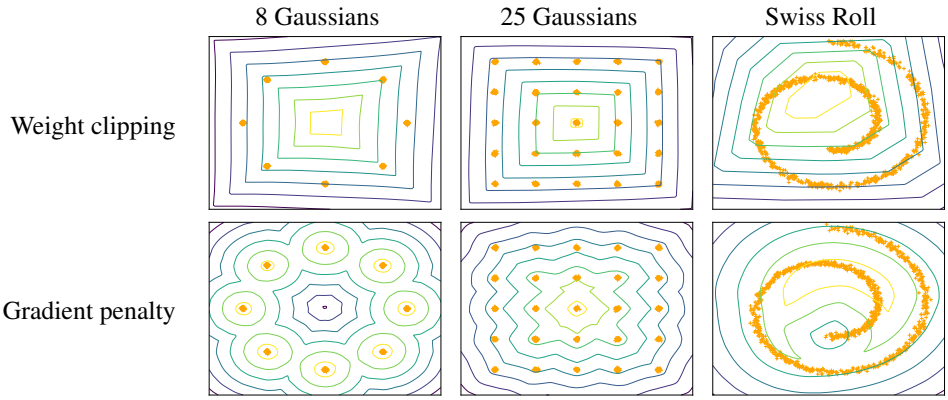
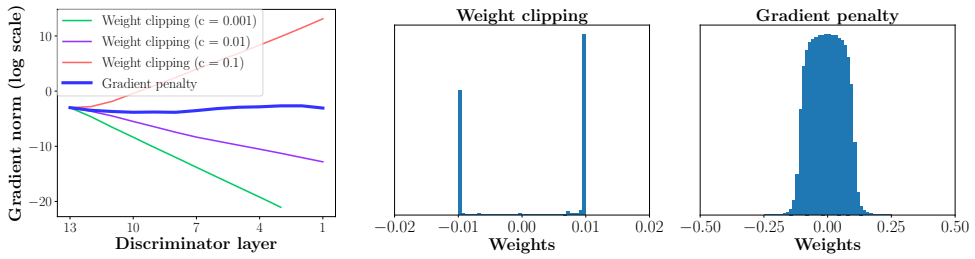


Figure 1: Value surfaces of WGAN critics trained to optimality on toy datasets. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.



(a) (b)

Figure 2: (a) Gradient norms of deep WGAN critics during training on toy datasets. Gradients in WGAN with weight clipping always either explode or vanish, depending on the clipping value. Training with gradient penalty provides stable gradients to earlier layers. (b) Histograms of weight values for WGAN with weight clipping (left) and gradient penalty (right). Weight clipping pushes weights to the extremes of the clipping range, and when this range is high, causes exploding gradients and slows training.

For example, the optimal critic under weight clipping might contain many duplicate hidden units (ignoring symmetries in network architecture) which are then summed to maximize the final scale of the output. This strategy might be optimal in the sense of maximizing the contrast between the real and generated distributions, but the resulting discriminator underutilizes its capacity and yields less useful gradients for training the generator.

To demonstrate this, we train WGAN critics with weight clipping to optimality on several toy distributions, holding the generator distribution $\mathbb{P}_g$ fixed at the real distribution plus unit-variance Gaussian noise. We plot value surfaces of the critics in Figure 1. We omit batch normalization in the critic. In each case, the critic trained with weight clipping ignores higher moments of the data distribution and instead models very simple approximations to the optimal functions. In contrast, our approach does not suffer from this behavior.

3.2 Exploding and vanishing gradients

We observe that the WGAN optimization process is difficult because of interactions between the weight constraint and the cost function, which inevitably result in either vanishing or exploding gradients, depending on the value of the clipping threshold c.

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0.5$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

If the weights are constrained to be too small, the gradient vanishes as we backpropagate through previous layers. This prevents earlier layers in the critic (and the generator) from receiving useful training signal and can make learning very slow for deep nets.

On the other hand, if the weight constraint is too large, we find that the network suffers from *exploding* gradients instead. This is because the training objective encourages all of the weights in the critic to lie at the extremes of their allowed range (we demonstrate this experimentally in Figure 2b).

To demonstrate the presence of vanishing and exploding gradients in weight-clipped WGAN, we train WGAN on the Swiss Roll toy dataset, varying the clipping threshold c in $[10^{-1}, 10^{-2}, 10^{-3}]$, and plot the norm of the gradient of the critic loss with respect to successive layers of activations. Our generator and critic are both 12-layer-deep ReLU MLPs without batch normalization. We show in Figure 2a that for each of these values, the gradient either grows or decays exponentially as we move farther back in the network. We find our method results in more stable gradients that neither vanish nor explode, allowing training of more complicated networks.

To some extent these problems can be mitigated with batch normalization in the critic, which Arjovsky et al. (2017) use in all of their experiments. However even with batch normalization, we observe that very deep WGAN critics often get stuck in a bad regime and fail to learn.

4 Gradient penalty

As we discussed in the previous section, the use of weight clipping in WGAN can result in undesirable behaviors. We consider an alternative method to enforce the Lipschitz constraint on the training objective: a differentiable function is 1-Lipschitz if and only if it has gradients with norm less than or equal to 1 everywhere, so we would like to directly constrain the gradient norm of our critic function with respect to its input.

Exactly enforcing this constraint is not easily tractable, so instead we enforce a soft version: at certain points sampled from a distribution over the input space $\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}$, we evaluate the gradient of the critic $\nabla_{\hat{x}} D(\hat{x})$ and penalize its squared distance from 1 in the critic loss function. Our new objective for the critic is:

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}\left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2\right]}_{\text{Our gradient penalty}} \tag{3}$$

In the limit of high λ, the optimal critic under our formulation is still the optimal critic under the true Kantorovich-Rubinstein dual. Therefore, given enough capacity on the critic the cost function

for the generator can recover the true Wasserstein distance (which doesn't necessarily happen in a normal WGAN due to the clipping).

Sampling along straight lines The gradient term $||\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})||_2$ is with respect to the points $\hat{\boldsymbol{x}}$, not the parameters of D. We implicitly define the distribution of points $\mathbb{P}_{\hat{\boldsymbol{x}}}$ at which to penalize the gradient by taking straight lines between points in the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$:

$$\epsilon \sim U[0, 1], \boldsymbol{x} \sim \mathbb{P}_r, \tilde{\boldsymbol{x}} \sim \mathbb{P}_g \tag{4}$$

$$\hat{\boldsymbol{x}} = \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}} \tag{5}$$

Our motivation for penalizing the gradient over this distribution comes from the fact that the graph of the optimal critic consists of straight lines connecting points from $\mathbb{P}_r$ and $\mathbb{P}_g$ (see subsection 2.3). Given that enforcing the Lipschitz constraint everywhere is intractable, enforcing it only along these straight lines seems sufficient and experimentally results in good performance.

Hyperparameters Our proposed penalty term introduces one hyperparameter, λ, which controls the trade-off between optimizing the penalty term and the original objective. All experiments in this paper use $\lambda = 10$, which we found to work well across a variety of architectures and datasets ranging from toy tasks to large ImageNet CNNs.

No critic batch normalization Most prior GAN implementations (Radford et al., 2015; Salimans et al., 2016; Arjovsky et al., 2017) make use of batch normalization in both the generator and the discriminator to help stabilize training. However using batch normalization in the discriminator changes the form of the problem: instead of specifying a function mapping a single input to a single output, a discriminator with batch normalization specifies a function mapping from an entire batch of inputs to a batch of outputs (Salimans et al., 2016).

Our penalized training objective is no longer valid in this setting, since we penalize the norm of the critic's gradient with respect to each input independently, and not the entire batch. To resolve this, we simply omit batch normalization in the critic in our models, finding that they perform well without it.

We note that our method does work with normalization schemes which don't introduce correlations between examples in a minibatch such as layer normalization (Ba et al., 2016), weight normalization (Salimans & Kingma, 2016) and instance normalization (Ulyanov et al., 2016). In particular, we recommend layer normalization as a drop-in replacement for batch normalization if desired.

Adam Because our method eliminates difficulties in the optimzation process arising from weight clipping, we find that we can recover the use of momentum-based optimizers, which Arjovsky et al. (2017) report perform poorly with weight-clipped WGAN. In particular, we found that using Adam (Kingma & Ba, 2014) with hyperparameters $\alpha = 0.0001, \beta_1 = 0.5, \beta_2 = 0.9$ in both the critic and the generator worked well. We recommend these settings and use them in all of our experiments.

Two-sided penalty One may wonder why we penalize the norm of the gradient for differing from 1, instead of just penalizing large gradients. The reason is that the optimal critic under the Kantorovich-Rubinstein dual (the WGAN objective) actually has gradients with norm 1 almost everywhere under $\mathbb{P}_r$ and $\mathbb{P}_g$ (see subsection 2.3). Simply penalizing overly large gradients also works in theory, but experimentally we found that this approach converged faster and to better optima.

Quadratic penalty One could have penalized $|\|\nabla D\| - 1|^p$ for some other value of p than 2. However, if $p \leq 1$ we note that pathological behavior can occur. If $\mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{x \sim \mathbb{P}_g}[D(x)] > \lambda \mathbb{E}[|\|\nabla D\| - 1|^p]$ then the network could lower the loss by simply scaling its output by a positive constant without bounds, causing training to diverge. For this reason, we choose to penalize the *squared* difference between the expected gradient norm and 1. As a sanity check, we experimentally tried linear penalties and found that they cause training to diverge.

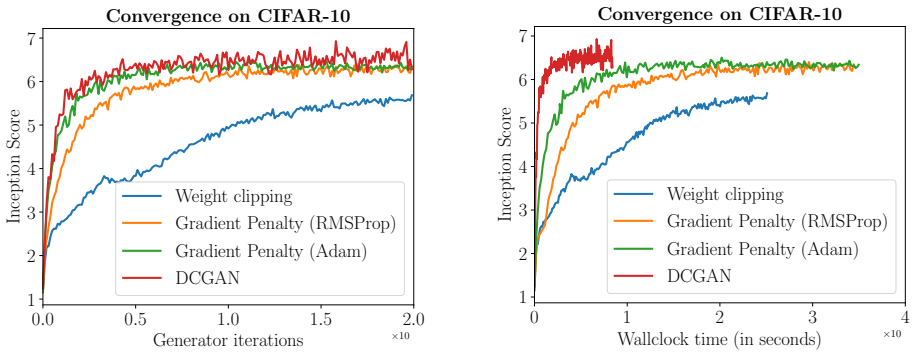


Figure 3: Plots of CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN with gradient penalty and RMSProp (to control for the optimizer), WGAN with gradient penalty and Adam, and DCGAN. Even with the same learning rate, gradient penalty significantly outperforms weight clipping. DCGAN converges faster, but WGAN with gradient penalty achieves similar scores with improved stability.

5 Experiments

In this section we show that WGANs with a gradient penalty enjoy improved convergence speed and stability over WGANs with weight clipping.

The overall observation when comparing with standard GANs was that GANs enjoy better convergence speed *once the architecture is fully stabilized for the problem*. WGANs converge slower, but as we show they enjoy substantially improved stability and scalability. This, together with a loss that correlates with sample quality, allow us to train WGANs with very deep architectures and approach new problems which couldn't be attacked before. We believe this will help extend GAN research toward harder and more important problems.

5.1 CIFAR-10 training speed and sample quality

One advantage of our method over weight clipping is improved training speed and sample quality. To demonstrate this, we train WGANs with weight clipping and our gradient penalty on CIFAR-10 (Krizhevsky, 2009) and plot Inception scores (Salimans et al., 2016) over the course of training (see Figure 3). For WGAN with gradient penalty, we train one model with the same optimizer (RMSProp) and learning rate as WGAN with weight clipping, and another model with Adam and a higher learning rate. Even with the same optimizer, our method converges faster and to a better final score than using weight clipping. Using Adam further improves performance. We also plot the performance of DCGAN (Radford et al., 2015) and observe that although our method converges more slowly (in terms of wall-clock time) than DCGAN, its performance is substantially more stable at convergence.

5.2 Architecture robustness on LSUN bedrooms

To demonstrate the stability of our method's training process, we train a wide variety of GAN architectures on the LSUN bedrooms dataset (Yu et al., 2015). In addition to the baseline DCGAN architecture from Radford et al. (2015), we choose six architectures which are difficult to train:

1. No BN and a constant number of filters in the generator, as in Arjovsky et al. (2017)
2. 4-layer 512-dim ReLU MLP generator, as in Arjovsky et al. (2017)
3. No normalization in either the discriminator or generator
4. Gated multiplicative nonlinearities everywhere, as in van den Oord et al. (2016)
5. $tanh$ nonlinearities everywhere
6. 101-layer ResNet generator and discriminator

Figure 4: Especially difficult GAN architectures trained with different methods. Only WGAN with gradient penalty succeeds in training every architecture.

Table 1: Top: Samples from a WGAN character-level language model trained with our method on sentences from the Billion Word dataset, truncated to 32 characters. The model learns to directly output one-hot character embeddings from a latent vector without any discrete sampling step. Bottom: Samples from the same architecture trained with the standard GAN objective.

WGAN with gradient penalty	
`Busino game camperate spent odea`	`Solice Norkedin pring in since`
`In the bankaway of smarling the`	`ThiS record ( 31. ) UBS ) and Ch`
`SingersMay , who kill that imvic`	`It was not the annuas were plogr`
`Keray Pents of the same Reagun D`	`This will be us , the ect of DAN`
`Manging include a tudancs shat "`	`These leaded as most-worsd p2 a0`
`His Zuith Dudget , the Denmbern`	`The time I paidOa South Cubry i`
`In during the Uitational questio`	`Dour Fraps higs it was these del`
`Divos from The ' noth ronkies of`	`This year out howneed allowed lo`
`She like Monday , of macunsuer S`	`Kaulna Seto consficutes to repor`
`The investor used ty the present`	`A can teal , he was schoon news`
`A papees are cointry congress oo`	`In th 200.  Pesish picriers rega`
`A few year inom the group that s`	`Konney Panice rimimber the teami`
`He said this syenn said they wan`	`The new centuct cut Denester of`
`As a world 1 88 ,for Autouries`	`The near , had been one injostie`
`Foand , th Word people car , Il`	`The incestion to week to shorted`
`High of the upseader homing pull`	`The company the high product of`
`The guipe is worly move dogsfor`	`20 - The time of accomplete , wh`
`The 1874 incidested he could be`	`John WVuderenson seqiivic spends`
`The allo tooks to security and c`	`A ceetens in indestredly the Wat`
Standard GAN objective	
`ddddddddddddddddddddddddddddddd`	`ddddddddddddddddddddddddddddddd`
`ddddddddddddddddddddddddddddddd`	`ddddddddddddddddddddddddddddddd`

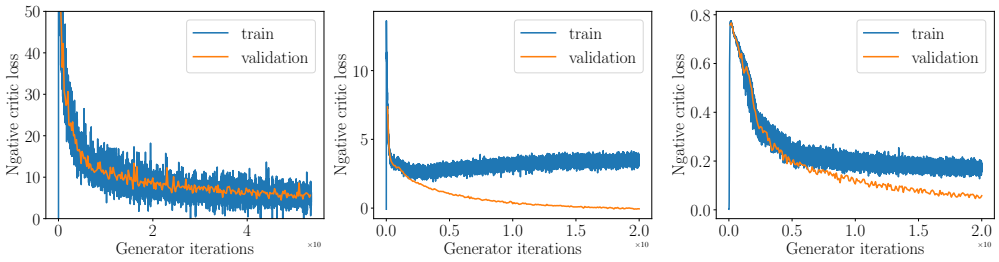
To the best of our knowledge this is the first time very deep residual networks were successfully trained in a GAN setting. For each architecture, we train models using four different GAN procedures: WGAN with gradient penalty, WGAN with weight clipping, DCGAN (Radford et al., 2015), and Least-Squares GAN (Mao et al., 2016). For each objective, we used the default set of optimizer hyperparameters recommended in that work:

- WGAN with gradient penalty: Adam ($\alpha = .0001, \beta_1 = .5, \beta_2 = .9$)
- WGAN with weight clipping: RMSProp ($\alpha = .00005$)
- DCGAN: Adam ($\alpha = .0002, \beta_1 = .5$)
- LSGAN: RMSProp ($\alpha = .0001$) [chosen by search over $\alpha = .001, .0002, .0001$]

For WGAN with gradient penalty, we replace any batch normalization in the discriminator with layer normalization (see section 4, "No batch normalization"). We train each model for 200K iterations and present samples in Figure 4. **WGAN with gradient penalty is the only method which successfully trains every architecture with a single set of default hyperparameters.** For every other training method, some of these architectures are unstable or suffer from mode collapse.

5.3 Character-level language modeling

Using GANs to model language is a challenging task, largely because text is a sequence of discrete tokens and discrete output units in the generator are difficult to backpropagate through. Most past attempts have addressed this with the REINFORCE gradient estimator (Williams, 1992) or continuous approximations to discrete sampling (Jang et al., 2016; Maddison et al., 2016), but training with these methods performs poorly on challenging problems. Consequently, past GAN language modeling work has resorted to pretraining or joint training with a typical supervised maximum-likelihood objective (Yu et al., 2016; Li et al., 2017; Yang et al., 2017; Che et al., 2017; Liang et al., 2017).



(a) (b)

Figure 5: (a) The negative critic loss of our model on LSUN bedrooms converges toward a minimum as the network trains. (b) WGAN training and validation losses on a random 1000-digit subset of MNIST show overfitting when using either our method (left) or weight clipping (right). In particular, with our method, the critic overfits faster than the generator, causing the training loss to increase gradually over time even as the validation loss drops.

Hjelm et al. (2017) also propose a discrete GAN training method but do not evaluate on language modeling.

In contrast, using our method we train a GAN which generates discrete outputs without resorting to complicated "backprop through discrete variable" methods, maximum-likelihood training, or fine-tuned architectures. We train a character-level GAN language model on the Google Billion Word dataset (Chelba et al., 2013). Our generator is a simple CNN architecture which deterministically transforms a latent vector into a sequence of 32 one-hot character vectors through 1D convolutions. We apply a softmax nonlinearity at the output, but use no sampling step: during training, the softmax output is passed directly into the critic (which likewise, is a simple 1D CNN over sequences of 32 one-hot vectors). When decoding samples, we just take the argmax of each output vector.

We present samples from the model in Table 1. **To our knowledge this is the first general language model trained entirely adversarially without a supervised maximum-likelihood loss.** We see that it makes frequent spelling errors (likely because it has to output each character independently) but nonetheless manages to learn quite a lot about the statistics of natural language.

The difference in performance between WGAN and other GANs can be explained by a simple fact. Consider the simplex $\Delta_n = \{p \in \mathbb{R}^n : p_i \geq 0, \sum_i p_i = 1\}$, and the set of vertices on the simplex (or one-hot vectors) $V_n = \{p \in \mathbb{R}^n : p_i \in \{0, 1\}, \sum_i p_i = 1\} \subseteq \Delta_n$. If we have a vocabulary of size n and we have a distribution $\mathbb{P}_r$ over sequences of size T, we have that $\mathbb{P}_r$ is a distribution on $V_n^T = V_n \times \cdots \times V_n$. Since V_n^T is a subset of Δ_n^T, we can also treat $\mathbb{P}_r$ as a distribution on Δ_n^T (by assigning zero probability mass to all points not in V_n^T).

Now the interesting part is this: $\mathbb{P}_r$ is discrete (or supported on a finite number of elements, namely V_n^T) on Δ_n^T, but $\mathbb{P}_g$ can easily be a continuous distribution over Δ_n^T. The KL divergences between two such distributions are infinite, and so the JS divergence is saturated. In practice, this reflects itself by a discriminator almost instantly learning to say that all samples that don't lie on V_n^T (sequences of one-hot vectors) are fake, and the gradients passed to the generator are meaningless. However, it is easily seen that the conditions of Theorem 1 and Corollary 1 of Arjovsky et al. (2017) are satisfied even on this non-standard learning scenario with $\mathcal{X} = \Delta_n^T$. This means that $W(\mathbb{P}_r, \mathbb{P}_g)$ is still well defined, continuous everywhere and differentiable almost everywhere, and we can optimize it just like in any other continuous variable setting. The way this is manifested in WGANs is in the fact that the Lipschitz constraint doesn't allow sharp transitions on the critic between V_n^T and its neighbourhood: it has to provide a linear gradient from all Δ_n^T towards towards the real points in V_n^T.

5.4 Meaningful loss curves and detecting overfitting

An important benefit of weight-clipped WGANs is that their loss correlates with sample quality and converges toward a minimum. WGANs trained with our method exhibit the same behavior. To show this, we train a WGAN with gradient penalty on the LSUN bedrooms dataset (Yu et al., 2015)

and plot the negative of the critic's loss in Figure 5a. We see that over time the negative critic loss converges as the generator learns to minimize $W(\mathbb{P}_r, \mathbb{P}_g)$.

GANs, like all models trained on limited data, will eventually overfit given enough capacity, but the lack of a meaningful loss metric has made overfitting hard to detect in the past. To explore the loss curve's behavior when the network overfits, we train large unregularized WGANs on a random 1000-image subset of MNIST. We train models with either weight clipping or gradient penalty, and plot the negative critic loss on both the training and validation sets in Figure 5b.

In both models, the two losses diverge, suggesting that the critic overfits and provides an inaccurate estimate of $W(\mathbb{P}_r, \mathbb{P}_g)$, at which point all bets are off regarding correlation with sample quality. However in WGAN with gradient penalty, the training loss gradually increases even while the validation loss drops. Our explanation for the difference in shape between the two plots is that while both models overfit, in WGAN with gradient penalty, the critic overfits faster and more easily than the generator. Precisely, in WGAN with gradient penalty, the critic learns to overestimate $W(\mathbb{P}_r, \mathbb{P}_g)$ faster than the generator learns to minimize the estimate, whereas the opposite is true for WGAN with weight clipping (likely due to the training problems discussed in section 3).

6 Conclusion

In this work, we explored some of the problems with weight clipping in WGAN, and introduced an alternative in the form of a penalty term added to the critic loss. We showed that doing so prevents pathological behavior otherwise caused by weight clipping. Finally, we demonstrated that our method is able to further improve performance over weight-clipped WGAN and successfully train difficult GAN architectures.

Now that we have a stable algorithm for training GANs, we enable exploration of exactly which architectures achieve the best generative modeling performance. In particular, we think our work opens the path for strong modeling performance on large-scale image datasets and language.

Another interesting direction is adapting our penalty term to the standard GAN objective function, where it might stabilize training by encouraging the discriminator to learn smoother decision boundaries.

Acknowledgements

We would like to thank Mohamed Ishmael Belghazi, Zihang Dai, Kyle Kastner, Kundan Kumar, Luke Metz, Alec Radford, Sai Rajeshwar, and Zain Shah for insightful comments.

References

Arjovsky, Martin and Bottou, Léon. Towards principled methods for training generative adversarial networks. 2017.

Arjovsky, Martin, Chintala, Soumith, and Bottou, Léon. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Che, Tong, Li, Yanran, Zhang, Ruixiang, Hjelm, R Devon, Li, Wenjie, Song, Yangqiu, and Bengio, Yoshua. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

Chelba, Ciprian, Mikolov, Tomas, Schuster, Mike, Ge, Qi, Brants, Thorsten, Koehn, Phillipp, and Robinson, Tony. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Goodfellow, Ian J. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515*, 2014.

Hjelm, R Devon, Jacob, Athul Paul, Che, Tong, Cho, Kyunghyun, and Bengio, Yoshua. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431*, 2017.

Jang, Eric, Gu, Shixiang, and Poole, Ben. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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

Krizhevsky, Alex. Learning multiple layers of features from tiny images. 2009.

Li, Jiwei, Monroe, Will, Shi, Tianlin, Ritter, Alan, and Jurafsky, Dan. Adversarial learning for neural dialogue generation. *arXiv preprint arXiv:1701.06547*, 2017.

Liang, Xiaodan, Hu, Zhiting, Zhang, Hao, Gan, Chuang, and Xing, Eric P. Recurrent topic-transition gan for visual paragraph generation. *arXiv preprint arXiv:1703.07022*, 2017.

Maddison, Chris J, Mnih, Andriy, and Teh, Yee Whye. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Mao, Xudong, Li, Qing, Xie, Haoran, Lau, Raymond YK, and Wang, Zhen. Least squares generative adversarial networks. *arXiv preprint arXiv:1611.04076*, 2016.

Metz, Luke, Poole, Ben, Pfau, David, and Sohl-Dickstein, Jascha. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

Nowozin, Sebastian, Cseke, Botond, and Tomioka, Ryota. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, pp. 271–279, 2016.

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

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

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–901, 2016.

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pp. 2226–2234, 2016.

Ulyanov, Dmitry, Vedaldi, Andrea, and Lempitsky, Victor. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

van den Oord, Aaron, Kalchbrenner, Nal, Espeholt, Lasse, Vinyals, Oriol, Graves, Alex, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Villani, Cédric. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Yang, Zhen, Chen, Wei, Wang, Feng, and Xu, Bo. Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*, 2017.

Yu, Fisher, Seff, Ari, Zhang, Yinda, Song, Shuran, Funkhouser, Thomas, and Xiao, Jianxiong. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Yu, Lantao, Zhang, Weinan, Wang, Jun, and Yu, Yong. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.

A Proof about the optimal WGAN critic

Lemma 1. *Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. Then, there is a 1-Lipschitz function f^* which is the optimal solution of*

$$\max_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{y \sim \mathbb{P}_r}[f(y)] - \mathop{\mathbb{E}}_{x \sim \mathbb{P}_g}[f(x)]$$

Let π be the optimal coupling between $\mathbb{P}_r$ and $\mathbb{P}_g$, defined as the minimizer of:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\pi \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathop{\mathbb{E}}_{(x,y)\sim\pi}[\|x - y\|]$$

Where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\pi(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. Then, if f^* is differentiable [2] and $x_t = tx + (1-t)y$ with $0 \leq t \leq 1$,

$$\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y - x_t}{\|y - x_t\|}\right] = 1$$

This, in particular, implies the norms of the gradients are 1.

Proof. Since $\mathcal{X}$ is a compact space, by Theorem 5.10 of Villani (2008), part (iii), we know that there is an optimal f^*. By Theorem 5.10 of Villani (2008), part (ii) we know that if π is an optimal coupling,

$$\mathbb{P}_{(x,y)\sim\pi}\left[f^*(y) - f^*(x) = \|y - x\|\right] = 1$$

Let (x, y) be such that $f^*(y) - f^*(x) = \|y - x\|$. Let $\psi(t) = f^*(x_t) - f^*(x)$. We claim that $\psi(t) = \|x_t - x\| = t\|y - x\|$.

Let $t, t' \in [0, 1]$, then

$$\begin{aligned}
|\psi(t) - \psi(t')| &= \|f^*(x_t) - f^*(x_{t'})\| \\
&\leq \|x_t - x_{t'}\| \\
&= |t - t'|\|x - y\|
\end{aligned}$$

Therefore, ψ is $\|x - y\|$-Lipschitz. This in turn implies

$$\begin{aligned}
\psi(1) - \psi(0) &= \psi(1) - \psi(t) + \psi(t) - \psi(0) \\
&\leq (1 - t)\|x - y\| + \psi(t) - \psi(0) \\
&\leq (1 - t)\|x - y\| + t\|x - y\| \\
&= \|x - y\|
\end{aligned}$$

However, $|\psi(1) - \psi(0)| = |f^*(y) - f^*(x)| = \|y - x\|$ so the inequalities have to actually be equalities. In particular, $\psi(t) - \psi(0) = t\|x - y\|$, and $\psi(0) = f^*(x) - f^*(x) = 0$. Therefore, $\psi(t) = t\|x - y\|$ and we finish our claim.

Let

$$\begin{aligned}
v &= \frac{y - x_t}{\|y - x_t\|} \\
&= \frac{y - ((1-t)x - ty)}{\|y - ((1-t)x - ty)\|} \\
&= \frac{(1-t)(y - x)}{\|(1-t)\|y - x\|} \\
&= \frac{y - x}{\|y - x\|}
\end{aligned}$$

[2] We can actually assume much less, and talk only about directional derivatives on the direction of the line; which we show in the proof always exist. This would imply that in every point where f^* is differentiable (and thus we can take gradients in a neural network setting) the statement holds.

Now we know that $f^*(x_t) - f^*(x) = \psi(t) = t\|x-y\|$, so $f^*(x_t) = f^*(x) + t\|x-y\|$. Then, we have the partial derivative

$$\begin{aligned}
\frac{\partial}{\partial v} f^*(x_t) &= \lim_{h\to 0} \frac{f^*(x_t + hv) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*\left(x + t(y-x) + \frac{h}{\|y-x\|}(y-x)\right) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*\left(x_{t+\frac{h}{\|y-x\|}}\right) - f^*(x_t)}{h} \\
&= \lim_{h\to 0} \frac{f^*(x) + \left(t + \frac{h}{\|y-x\|}\right)\|x-y\| - (f^*(x) + t\|x-y\|)}{h} \\
&= \lim_{h\to 0} \frac{h}{h} \\
&= 1
\end{aligned}$$

If f^* is differentiable at x_t, we know that $\|\nabla f^*(x_t)\| \leq 1$ since it is a 1-Lipschitz function. Therefore, by simple Pythagoras and using that v is a unit vector

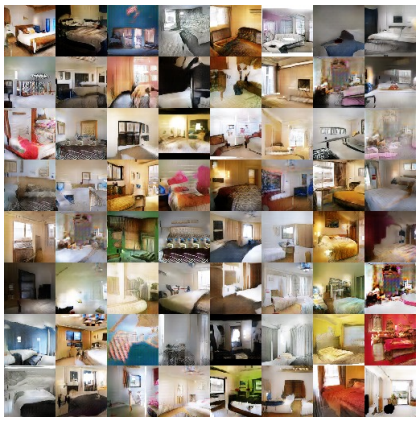
$$\begin{aligned}
1 &\leq \|\nabla f^*(x)\|^2 \\
&= \langle v, \nabla f^*(x_t)\rangle^2 + \|\nabla f^*(x_t) - \langle v, \nabla f^*(x_t)\rangle v\|^2 \\
&= |\frac{\partial}{\partial v} f^*(x_t)|^2 + \|\nabla f^*(x_t) - v\frac{\partial}{\partial v} f^*(x_t)\|^2 \\
&= 1 + \|\nabla f^*(x_t) - v\|^2 \\
&\leq 1
\end{aligned}$$

The fact that both extremes of the inequality coincide means that it was all an equality and $1 = 1 + \|\nabla f^*(x_t) - v\|^2$ so $\|\nabla f^*(x_t) - v\| = 0$ and therefore $\nabla f^*(x_t) = v$. This shows that $\nabla f^*(x_t) = \frac{y-x_t}{\|y-x_t\|}$.

To conclude, we showed that if (x, y) have the property that $f^*(y) - f^*(x) = \|y - x\|$, then $\nabla f^*(x_t) = \frac{y-x_t}{\|y-x_t\|}$. Since this happens with probability 1 under π, we know that $\mathbb{P}_{(x,y)\sim\pi}\left[\nabla f^*(x_t) = \frac{y-x_t}{\|y-x_t\|}\right] = 1$ and we finished the proof.

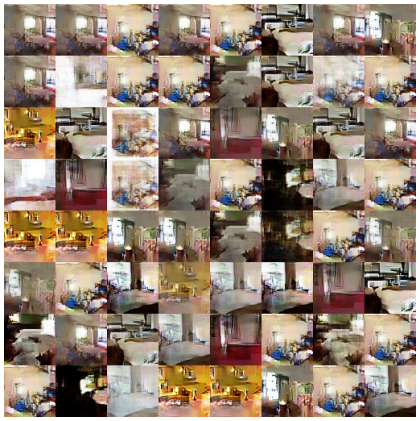
□

B More LSUN samples

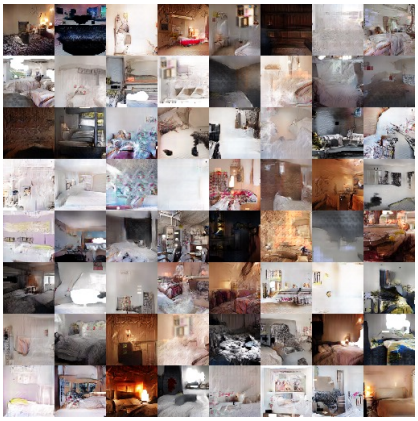


Method: DCGAN
G: DCGAN, D: DCGAN

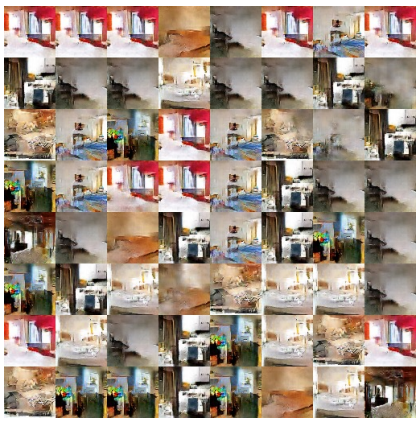
Method: DCGAN
G: No BN and const. filter count



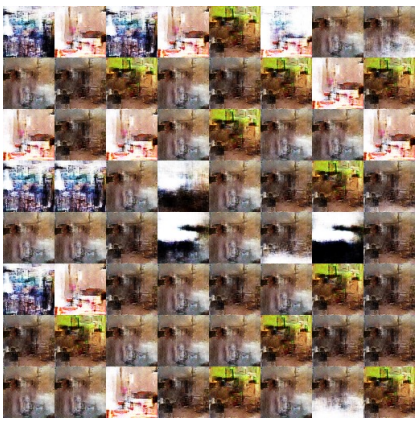
Method: DCGAN
G: 4-layer 512-dim ReLU MLP



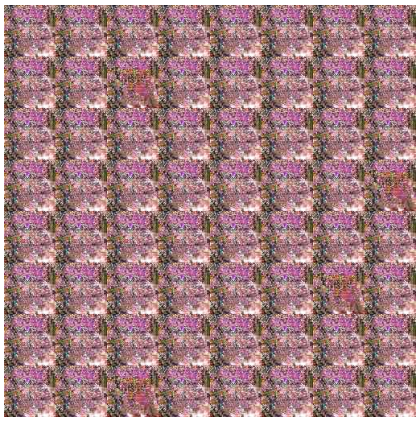
Method: DCGAN
No normalization in either G or D



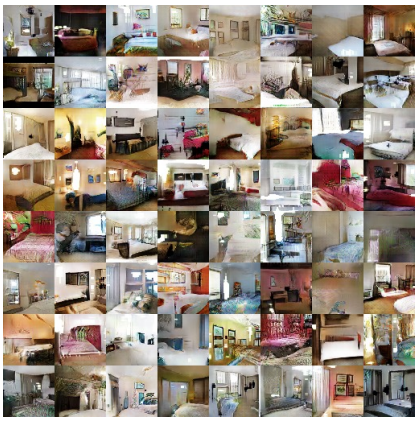
Method: DCGAN
Gated multiplicative nonlinearities



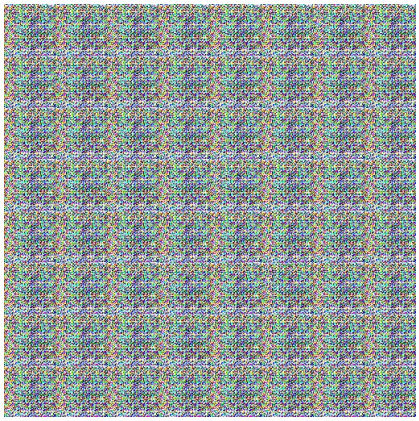
Method: DCGAN
$tanh$ nonlinearities



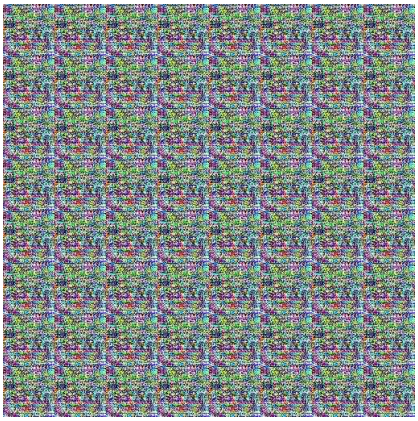
Method: DCGAN
101-layer ResNet G and D



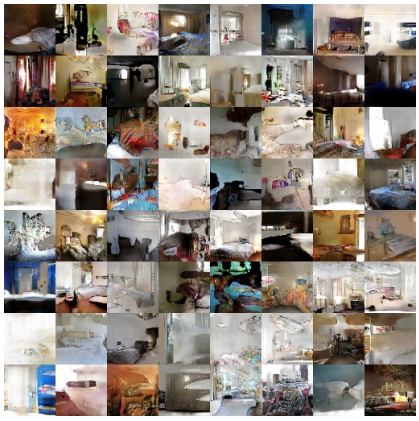
Method: LSGAN
G: DCGAN, D: DCGAN



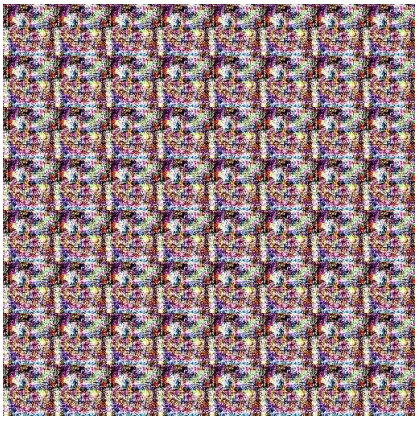
Method: LSGAN
G: No BN and const. filter count



Method: LSGAN
G: 4-layer 512-dim ReLU MLP

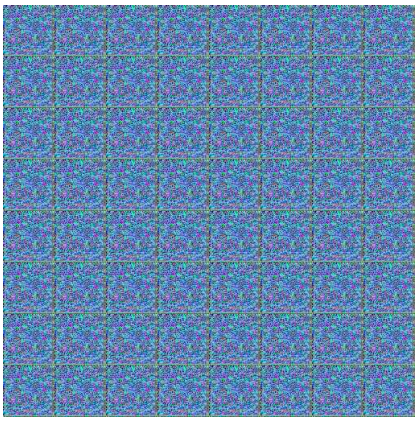


Method: LSGAN
No normalization in either G or D

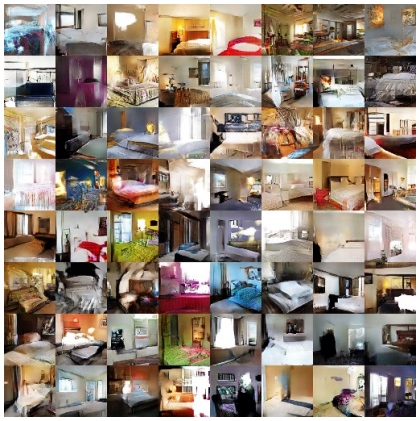


Method: LSGAN
Gated multiplicative nonlinearities

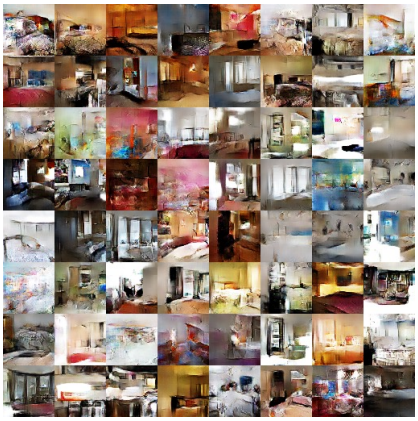
Method: LSGAN
$tanh$ nonlinearities



Method: LSGAN
101-layer ResNet G and D

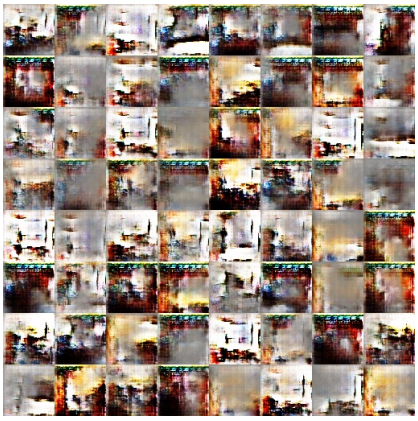


Method: WGAN with clipping
G: DCGAN, D: DCGAN



Method: WGAN with clipping
G: No BN and const. filter count

Method: WGAN with clipping
G: 4-layer 512-dim ReLU MLP

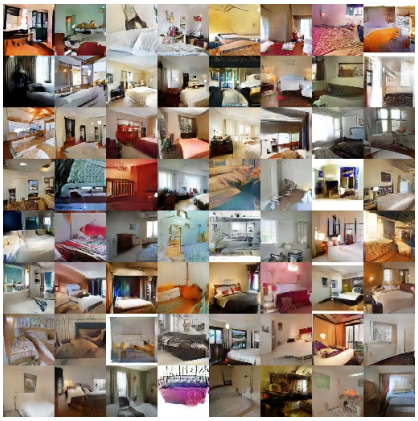


Method: WGAN with clipping
No normalization in either G or D

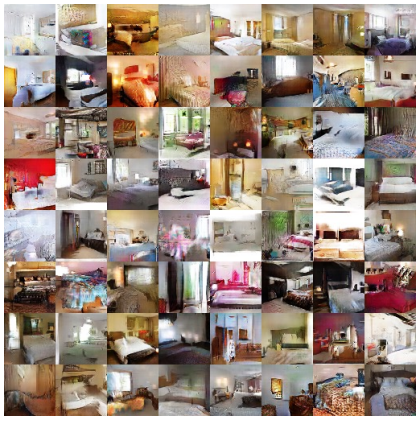
Method: WGAN with clipping
Gated multiplicative nonlinearities

Method: WGAN with clipping
$tanh$ nonlinearities

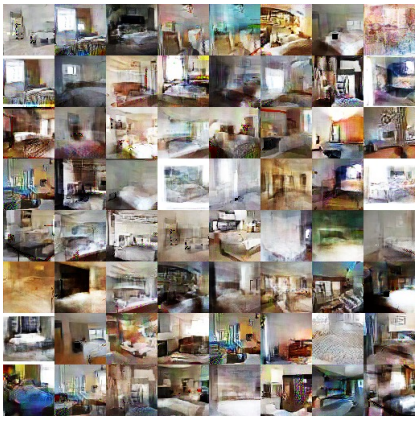
Method: WGAN with clipping
101-layer ResNet G and D



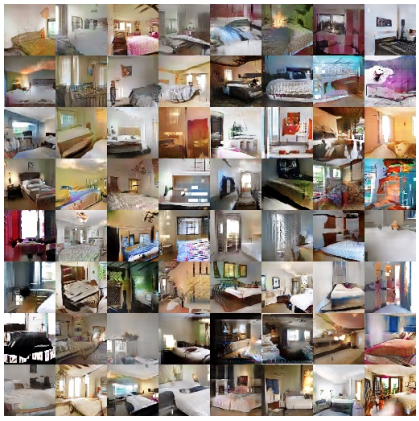
Method: WGAN-GP (ours)
G: DCGAN, D: DCGAN



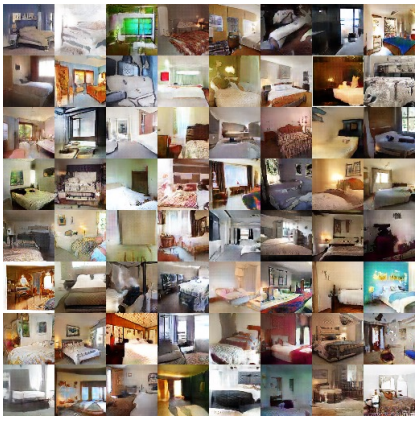
Method: WGAN-GP (ours)
G: No BN and const. filter count



Method: WGAN-GP (ours)
G: 4-layer 512-dim ReLU MLP

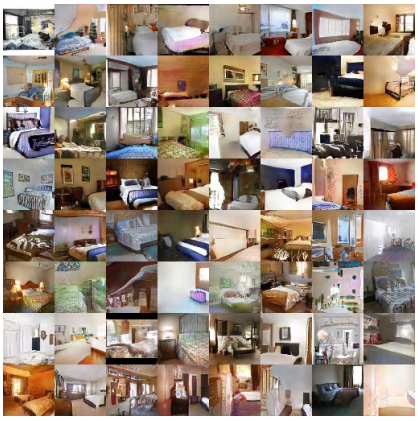


Method: WGAN-GP (ours)
No normalization in either G or D



Method: WGAN-GP (ours)
Gated multiplicative nonlinearities

Method: WGAN-GP (ours)
$tanh$ nonlinearities



Method: WGAN-GP (ours)
101-layer ResNet G and D