On the Effects of Batch and Weight Normalization in Generative Adversarial Networks

Sitao Xiang[1] **Hao Li**[1, 2, 3]
[1]University of Southern California
[2]Pinscreen
[3]USC Institute for Creative Technologies
sitaoxia@usc.edu hao@hao-li.com

Abstract

Generative adversarial networks (GANs) are highly effective unsupervised learning frameworks that can generate very sharp data, even for data such as images with complex, highly multimodal distributions. However GANs are known to be very hard to train, suffering from problems such as mode collapse and disturbing visual artifacts. Batch normalization (BN) techniques have been introduced to address the training problem. However, though BN accelerates training in the beginning, our experiments show that the use of BN can be unstable and negatively impact the quality of the trained model. The evaluation of BN and numerous other recent schemes for improving GAN training is hindered by the lack of an effective objective quality measure for GAN models. To address these issues, we first introduce a weight normalization (WN) approach for GAN training that significantly improves the stability, efficiency and the quality of the generated samples. To allow a methodical evaluation, we introduce a new objective measure based on a squared Euclidean reconstruction error metric, to assess training performance in terms of speed, stability, and quality of generated samples. Our experiments indicate that training using WN is generally superior to BN for GANs. We provide statistical evidence for commonly used datasets (CelebA, LSUN, and CIFAR-10), that WN achieves 10% lower mean squared loss for reconstruction and significantly better qualitative results than BN.

1 Introduction

Despite their prevalent use in Generative Adversarial Networks (GAN), the effects of Batch Normalization (BN) have not been examined carefully. The use of BN is typically justified by its perceived training speedup and stability [8, 13], but the generated samples often suffer from visual artifacts and limited variations (mode collapse). The lack of evidence that BN always improves GAN training is partly due to the unavailability of quality metrics for GAN models.

Being puzzled by this acceleration technique, we propose a methodical evaluation of GAN models and assess their abilities to generate large variations of samples (mode coverage). The idea is to use gradient descent on the latent code to reconstruct test images using a separate training dataset, which does not include the test samples. First a GAN model is learned from the training dataset, then for each test image, we optimize iteratively for the latent code via gradient descent so that the output of the GAN converges toward the test image. Our quality measure consists of the mean squared Euclidean distance between the test samples and the generated one.

Our experiments show that the reconstruction error correlates directly with the visual quality of the generated samples, Though still time consuming, the proposed approach is much more efficient than existing log-likelihood-based evaluation methods. Our evaluation technique is therefore convenient for monitoring the progress during training.

We show that BN generally accelerates training in early stages, and can increase the success rate of GAN training for certain datasets and network structures where a non-normalized model could often fail. In many cases though, BN can cause the stability and generalization power of the model to decrease drastically.

Inspired by the work [14], we introduce a modified Weight Normalization (WN) technique for GAN training. Using the same sets of experiments, we found that our WN approach can achieve faster and more stable training than BN, as well as generate equal or higher quality samples than GAN models without normalization. We believe that our proposed WN technique is superior than BN in the context of GANs.

2 Related work

Batch Normalization Batch Normalization (BN) [8] is a technique to accelerate the training of deep neural networks and has been shown to be effective in various applications. In the context of GANs, it has been first introduced in the influential work on DCGAN [13] and has since become a common practice, as listed in this comprehensive overview of GAN techniques [4].

To summarize, BN takes a batch of samples $\{x_1, x_2, \ldots, x_m\}$ and computes the following:

$$y_i = \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} \cdot \gamma + \beta \quad ,$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the means and standard deviations of the input batch and γ and β are the learned parameters. As a result, the output will always have a mean β and a standard deviation γ, regardless of the input distribution. Most importantly, the gradients must be back-propagated through the computation of $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$.

Weight Normalization Weight Normalization (WN) is a different acceleration approach recently introduced by [14], but has not been explored within the context of GANs. For a linear layer

$$\mathbf{y} = \mathbf{W}^T \mathbf{x} + \mathbf{b} \quad , \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{n \times m}$ and $\mathbf{b} \in \mathbb{R}^m$, weight normalization performs a reparameterization $\mathbf{W}$ with $\mathbf{V} \in \mathbb{R}^{n \times m}$ and $\mathbf{g} \in \mathbb{R}^m$:

$$\mathbf{w}_i = \frac{g_i}{||\mathbf{v}_i||} \cdot \mathbf{v}_i \quad , \tag{2}$$

where $\mathbf{w}_i$ and $\mathbf{v}_i$ are the i-th column of $\mathbf{W}$ and $\mathbf{V}$, respectively. Similar to BN, the computation of $||\mathbf{v}_i||$ is taken into account when computing the gradient with respect to $\mathbf{V}$.

Although presented as a reparameterization that modifies the curvature of the loss function, the main idea is to simply divide the weight vectors by their norms. A very similar idea has been proposed around the same time, under "normalization propogation" (NormProp) by [2]. While the effectiveness of this technique has been illustrated on various experiments in [14] and [2], they did not investigate this acceleration approach for GANs. As detailed in Section 3, we propose a modified version of Weight Normalization to improve the training of GAN models.

GAN Evaluation The most widely used method for evaluating GANs [5, 6] is to visually inspect the quality of generated samples. In particular, overfitting is detected by sampling from the GAN and then finding the closest training samples to see if they are sufficiently different, and by interpolating latent codes. In addition, the log-likelihood of the training set can be estimated by generating a large amount of samples and fitting a Gaussian Parzen window. As discussed in [15], these techniques are not particularly effective, as subtle overfittings are hard to visualize, and the amount of samples that need to be generated for accurate log-likelihood estimation is intractable. The quality of GANs can also been evaluated indirectly by measuring the classification accuracy using features extracted by a GAN discriminator [13]. Beside the quality of GANs, various formulations of training loss have been proposed to monitor the improvements during the training process [1, 3].

3 Weight Normalization for GAN Training

We propose a modified formulation of the weight normalization approach introduced by [14]. A notable deficiency of the original WN technique is that, in its simplest form, it does not normalize the mean value of the input. In [14] this is solved by augmenting WN with a version of BN, that only normalizes the mean of the input but not the variance. While their experiments showed an improved performance for the CIFAR-10 classification task compared to plain WN, it gave worse results for several of our experiments (See appendix D.2). Hence, we chose to not include this augmentation in our approach.

In [2], the authors attempt to solve this problem by enforcing a zero-mean, unit-variance distribution throughout the network. Their method consists of first using a restricted form of Weight Normalization with a scale factor $\mathbf{g} = \mathbf{1}$ and a bias $\mathbf{b} = \mathbf{0}$, where,

$$y = \frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}||} \quad . \tag{3}$$

For simplicity, we consider here a single output neuron. The training data is normalized so that the input to the network has zero mean and unit variance. The mean and variance of the output of each nonlinear layer (ReLU in this case) is evaluated using a closed form under the assumption that the input to the preceding linear layer is from a multivariate standard normal distribution. The mean and variance is then used to correct the distribution of the output:

$$y = \frac{1}{\sigma}\left[\mathrm{ReLU}\left(\frac{\mathbf{w}^T \mathbf{x}}{||\mathbf{w}||}\right) - \mu\right] \quad , \tag{4}$$

where

$$\mu = \sqrt{\frac{1}{2\pi}} \quad \text{and} \quad \sigma = \sqrt{\frac{1}{2}\left(1 - \frac{1}{\pi}\right)}$$

are the mean and standard deviation of the distributions after ReLU when $\mathbf{x}$ is from a multivariate standard normal distribution. The output y in equation 4 would also have zero mean and unit variance.

Notice that this ad-hoc fix does not really achieve its goal. Firstly, as mentioned in [2], the closed form mean and variance is only an approximation since the correctness of this derivation requires the input to be normal distributed, which does not strictly hold beyond the first layer. More critically, after deriving equation 4 and fixing μ and σ, an extra affine transformation needs to be learned after the weight-normalized linear layer and before the succeeding non-linear layer, in order to avoid decreasing the set of functions that can be represented by the network, which is analogous to BN. The formulation then becomes:

$$y = \frac{1}{\sigma}\left[\mathrm{ReLU}\left(\frac{\gamma\left(\mathbf{w}^T \mathbf{x}\right)}{||\mathbf{w}||} + \beta\right) - \mu\right] \quad . \tag{5}$$

Their derivation for μ and σ is for the restricted case, when there is no learned affine transformation, i.e. when $\gamma = 1$ and $\beta = 0$. When this restriction is relaxed, the result would be invalid, even if the condition on $\mathbf{x}$ does hold. We could make μ and σ functions of γ and β to fix this error, but the resulting gradient computations with respect to the parameters would be overly complex, since these functions also need to be taken into account.

As we cannot hope to strictly enforce a zero-mean unit-variance distribution, we propose to use a simpler approximation instead. Note that with ReLU-like nonlinearity (i.e. ReLU, leaky ReLU and parametric ReLU) we have $\mathrm{ReLU}(ax) = a \cdot \mathrm{ReLU}(x)$ when $a \geq 0$. In equation 5, when $\gamma < 0$, we can always invert the direction of $\mathbf{w}$ and take the negative of γ. Hence, without loss of generality, we can assume $\gamma \geq 0$. Then equivalently, equation 5 can be written as

$$y = \frac{\gamma}{\sigma}\left[\mathrm{ReLU}\left(\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} + \frac{\beta}{\gamma}\right) - \frac{\mu}{\gamma}\right] \quad . \tag{6}$$

The purpose of μ and σ is to cancel out the non-zero mean and non-unit variance introduced by ReLU and the affine transformation (i.e. β and γ). Instead of deriving a complex formulation, we simply set $\mu = \beta$ and $\sigma = \gamma$, and re-formulate the equation using $\alpha = -\frac{\beta}{\gamma} = -\frac{\mu}{\gamma}$. Equation 6 becomes:

$$y = \mathrm{ReLU}\left(\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} - \alpha\right) + \alpha \quad . \tag{7}$$

Note that we can now separate out the restricted weight normalized layer from equation 7. We call the remaining part "Translated ReLU (TReLU)":

$$\begin{aligned}\mathrm{TReLU}_\alpha(x) =& \mathrm{ReLU}(x - \alpha) + \alpha \\ =& \begin{cases} x & (x \geq \alpha) \\ \alpha & (x < \alpha) \end{cases} ,\end{aligned}$$

where α is a learned parameter. It is more commonly referred to as a "threshold layer". We chose this name to reflect the fact that other ReLU-like nonlinear functions can be used to give translated leaky and parametric ReLU layers. Here, we translate the data by $-\alpha$, apply the nonlinear function, then translate the data back (by α). By using TReLU instead of adding bias to the previous layer, we prevent (to a certain degree) a large mean from being introduced into the distribution.

Such simplification does reduce the set of functions that can be represented by the network. Allowing the learning of an affine transformation at the last weight-normalized layer recovers the expressiveness of the entire stack of layers (see appendix A for proof). From now on, "strict weight-normalized layers" will refer to layers without affine transformations (Equation 3), while layers with a learned affine transformation

$$y = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} \cdot \gamma + \beta$$

are referred as "affine weight-normalized layers". Both types are called "weight-normalized layers".

4 Evaluation Method

For many generative models, the reconstruction error on the training set is often explicitly optimized in some form (e.g., Variational Autoencoders [9]). It is therefore natural to evaluate generative models using a reconstruction error metric (squared Euclidean distance) measured on a test set. In the case of GANs, given a generator G and a set of test samples $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, the reconstruction loss of G on X is defined as

$$\mathcal{L}_{\mathrm{rec}}(G, X) = \frac{1}{m}\sum_{i=1}^{m} \min_z ||G(z) - x^{(i)}||^2 \quad .$$

Since there is no way to directly infer the optimal z from x, we use an alternative method: starting from an all-zero vector, we perform gradient descent on the latent code to find one that minimizes the Euclidean distance between the sample generated from the code and the target one.

Since the code is optimized instead of computed from a feed-forward network, the evaluation process is time-consuming. Thus, we avoid performing this evaluation at every training iteration when monitoring the training process, and only use a reduced number of samples, as well as gradient descent steps. Only for the final trained model, we perform an extensive evaluation on a larger test set, with a larger number of steps.

Table 1: Network structure for discriminators (left) and generators (right). First three columns: type of layers for vanilla, BN and WN models, respectively. Fourth column: kernel size, stride, padding and number of output channels of convolution layer.

vanilla	BN	WN		vanilla	BN	WN	
Conv	Conv	SWNConv	4, 2, 1, 64	Conv	Conv	SWNConv	5, 1, 0, 1024
-	-	-		-	BN	-	
PReLU	PReLU	TPReLU		PReLU	PReLU	TPReLU	
Conv	Conv	SWNConv	4, 2, 1, 128	Conv	Conv	SWNConv	4, 2, 1, 512
-	BN	-		-	BN	-	
PReLU	PReLU	TPReLU		PReLU	PReLU	TPReLU	
Conv	Conv	SWNConv	4, 2, 1, 256	Conv	Conv	SWNConv	4, 2, 1, 256
-	BN	-		-	BN	-	
PReLU	PReLU	TPReLU		PReLU	PReLU	TPReLU	
Conv	Conv	SWNConv	4, 2, 1, 512	Conv	Conv	SWNConv	4, 2, 1, 128
-	BN	-		-	BN	-	
PReLU	PReLU	TPReLU		PReLU	PReLU	TPReLU	
Conv	Conv	SWNConv	4, 2, 1, 1024	Conv	Conv	SWNConv	4, 2, 1, 64
-	BN	-		-	BN	-	
PReLU	PReLU	TPReLU		PReLU	PReLU	TPReLU	
Conv	Conv	AWNConv	5, 1, 0, 1	Conv	Conv	AWNConv	4, 2, 1, 3
Sigmoid	Sigmoid	Sigmoid		Sigmoid	Sigmoid	Sigmoid	

Intuitively, in order to generate the test samples, that are not in the training set, the generator must learn the distribution of the training samples, but not memorize and overfit on them. Consequently, the reconstruction loss is a quantitative alternative to just visually inspecting the samples. It allows us to detect overfitting, as well as measure the ability to generate novel but plausible samples.

5 Experiments

5.1 Setup

We conducted experiments on image generation tasks, using 160×160 patches of the CelebA dataset [11] and evaluated the performance of DCGAN-based models [13]. We compared three models: (1) trained without any normalization as a reference (the non-normalized or "vanilla" model), (2) with Batch Normalization ("BN model"), and (3) with our formulation of Weight Normalization ("WN model"). The network is structured in the following way: for the discriminator, we use successive convolution layers with kernel size 4, stride 2, padding 1 and output features doubling that of the previous layer, starting from 64 features in the first layer. We add convolution layers until the spatial size of the feature map is sufficiently small (5×5). We then add one final convolution layer with stride 1, zero padding and kernel size 5 (equaling the size of the last feature map). For the generator, we reverse this structure and use transposed convolution layers.

As per common practice, Batch Normalization is not applied to the first layer of the discriminator, nor to the last layers of both the discriminator and generator. Weight normalization is used for every layer. For the last layer of both discriminator and generator, we use affine weight-normalized layers (AWNConv) while for every other layer we use strict weight-normalized layers (SWNConv). Parametric ReLU (PReLU) is used for vanilla and batch-normalized models and Translated Parametric ReLU (TPReLU) for weight-normalized models. Slope and bias parameters are learned per-channel. The length of the code is 256 for all models. The architectures are summarized in table 1.

Additional details regarding the implementation of weight normalized layers are discussed in appendix B.

All models are optimized with RMSProp [16], with a learning rate of 10^{-4}, $\alpha = 0.9$, $\varepsilon = 10^{-6}$ and a batch size of 32. Specifically for the BN model, we use separate batches for true samples and generated samples when training the discriminator, as suggested by [4]. After each parameter update, we clip the learned slope of parametric ReLU layers to $[0, 1]$.

There are a total of 202,599 images in CelebA dataset. We randomly selected 2,000 images for evaluation and used the rest for training. During the training, we perform a "running evaluation" for every 500 training iterations, on a randomly selected and fixed subset of 200 test samples. The

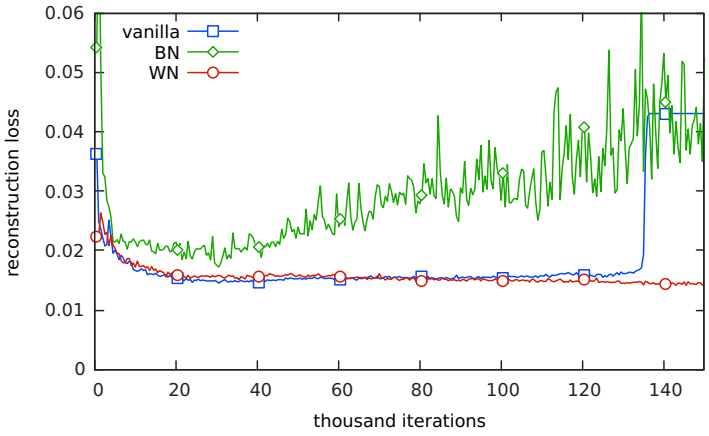

Figure 1: Running reconstruction loss during training.

Table 2: Optimal reconstruction loss of the models

Model	Optimal iteration	Running loss	Final loss
vanilla	30,500	0.014509	0.006171
BN	30,500	0.017199	0.006355
WN	463,000	0.013010	0.005524

optimal code is found by performing gradient descent for 50 steps, starting from a zero vector. Again we use RMSProp, with a learning rate of 0.01.

For each model, the best performing network during the training is saved and used for final evaluation. In the final evaluation, we use all 2,000 test samples and perform gradient descent for 2,000 steps. For BN model, we use its inference mode.

Due to a phenomenon in which a model with worse running reconstruction may lead to better final reconstruction, as occurred in additional experiments, we also use the "converged" model for evaluation, in case this converged model produces notably worse running reconstruction loss. However, this did not occur in the main experiment. We consider that training has converged if both the running reconstruction loss and the generated samples stay stable for a sufficient amount of time.

We observed mode collapse issues with both the vanilla and BN models. To reduce the possibility that these observations are caused by random factors, we repeat the training procedure for these models three times. We present the results from the best training instances. Additional training instances of vanilla and BN models can be found in Appendix D.1. The same set of experiments were also conducted on LSUN bedroom and CIFAR-10 datasets (see Appendices D.3 and D.4).

5.2 Reconstruction

The (per-pixel) running reconstruction loss of the three models is shown in Figure 1 for the first 150,000 iterations. The generated samples from both the vanilla and BN models have collapsed. The WN model was trained using 700,000 iterations and converged with a running loss close to the best recorded loss (see Appendix C).

The lowest running reconstruction loss recorded during training, the iteration at which this minimum loss is achieved, and the final reconstruction loss for each model is listed in table 2. WN achieves about 10.5% lower final reconstruction loss than the vanilla model, while for BN the loss is 3% higher. We can also see from the loss curve that, until the vanilla model collapses, BN never achieved a better reconstruction loss. Notice that the performance of the WN model lagged behind the vanilla model in the beginning, in terms of running reconstruction loss.

We also provide qualitative results of the reconstructions. Selected reconstructed samples are compared to the original test samples in figure 2. These samples are selected such that all three

Figure 2: Selected final reconstruction results. From left to right in each group: test sample, vanilla reconstruction, BN reconstruction, WN reconstruction.

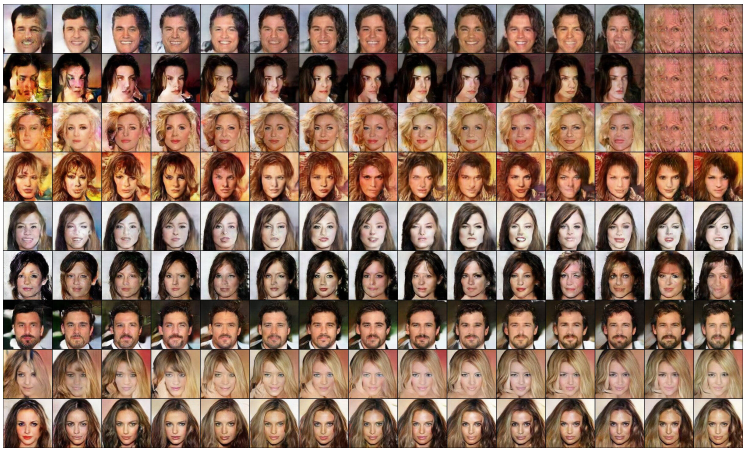

Figure 3: Evolution of samples during training. Top 3 rows: vanilla; Middle 3 rows: BN; Bottom 3 rows: WN. Columns: every 10,000 iterations from 10,000 to 150,000.

models give reasonable results. Additional random samples can be found in Appendix C. The WN model captures details (e.g. facial expression, texture of hair, subtle color variation) much more faithfully. Samples reconstructed by the BN model are significantly blurrier and affected by artifacts.

5.3 Stability

As shown in Figure 1, the reconstruction of vanilla and BN models started to get worse relatively early on during their training, after achieving their optimal reconstruction loss. For the vanilla model, the loss went up slowly, then in a relatively short time around iteration 135,000, the generator collapses and produces the same output, which caused the reconstruction loss to increase suddenly. For the BN model, at around 40,000 iterations, the loss started to show excessive fluctuation. Our WN model however, kept improving steadily until 300,000 iterations and then remained largely stable (see Appendix C).

We can also visualize this (in)stability by checking samples generated from the same code at different iterations, as shown in Figure 3. The WN model is noticeably more stable as samples generated from the same code remain mostly constant across a time scale of 100,000 iterations, and the generated samples are slowly improving, while the other two models produce more random variations. Additional visual analysis and more samples can be found in Appendix C.

We consider the stability of our weight-normalized model to be comparable to Wasserstein GAN [1]. We include a discussion of the connections in Appendix E. Most importantly, our improvement in stability is achieved solely by changing the parameterization of the network, without modifying the

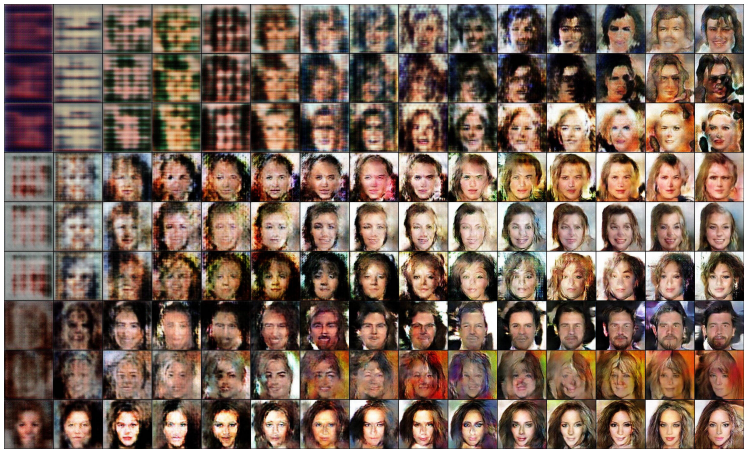

Figure 4: Evolution of samples during early stage of training. Top 3 rows: vanilla; Middle 3 rows: BN; Bottom 3 rows: WN. Columns are samples from iterations 100, 200, 300, 400, 500, 600, 800, 1000, 1200, 1500, 2000, 2500, 3000, 4000 and 5000.

loss function (as in Wasserstein GAN [1], LSGAN [12] and BEGAN [3]), training objective (as in improved WGAN [7] and Softmax GAN [10]), or network structure (as in EBGAN [17]).

5.4 Training Speed

We compare the training speed of the three models by assessing their generated samples during early stages of the training, as illustrated in Figure 4. It is evident that Batch Normalization does accelerate training and the effect of Weight Normalization is comparable. Notice that our WN model can already produce a human face in only 100 iterations.

This accelerated training is mostly useful as a fast sanity check, when monitoring the training progress of deep neural networks. As shown in Figure, the visual quality of the samples, generated by the three models, are comparable at 10,000 iterations, and none of the models achieve a noticeably faster progression than the other. In addition, the ability to generate visually plausible samples earlier on does not necessarily translate into an overall faster improvement of the reconstruction.

Notice that BN allows a higher learning rate. The training of the vanilla and WN models often fail with a learning rate of 0.0002, while the BN model can still be trainable with a learning rate of 0.001. However, we found that an increased learning rate did not accelerate the training of the BN model. Instead, it further harms the stability of the model.

6 Conclusion

We introduced weight normalization for the training of GANs using an alternative formulation than the original work of [14], which achieves superior training performance. Our solution also provides a mathematically more sound formulation than the one of [2].

We also presented an evaluation method for GANs based on the mean squared Euclidean distance between the test samples and the closest generated ones, which are synthesized via gradient descent on a latent code.

We trained variants of DCGAN [13] using different normalization methods for image generation at multiple scales. We found that batch-normalized models perform worse in reconstructing test samples and are less stable during training. In particular, both reconstruction errors and the visual quality can be deteriorated by BN.

However, our formulation of weight normalization improves both reconstruction quality and training stability considerably. Based on our extensive evaluations, we believe that weight normalization should always be used instead of batch normalization when training generative adversarial networks.

Acknowledgments

We would like to thank Kyle Olszewski for proofreading the paper and J.P. Lewis for the valuable feedback. This research is supported in part by Adobe, Oculus & Facebook, Huawei, the Google Faculty Research Award, the Okawa Foundation Research Grant, the Office of Naval Research (ONR) / U.S. Navy, under award number N00014-15-1-2639, the Office of the Director of National Intelligence (ODNI) and Intelligence Advanced Research Projects Activity (IARPA), under contract number 2014-14071600010, and the U.S. Army Research Laboratory (ARL) under contract W911NF-14-D-0005. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ONR, ODNI, IARPA, ARL, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon.

References

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

[2] Devansh Arpit, Yingbo Zhou, Bhargava U Kota, and Venu Govindaraju. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *arXiv preprint arXiv:1603.01431*, 2016.

[3] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[4] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to train a gan? tips and tricks to make gans work. `https://github.com/soumith/ganhacks`, 2016. Accessed: 2017-02-26.

[5] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[10] Min Lin. Softmax gan. *arXiv preprint arXiv:1704.06191*, 2017.

[11] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[12] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. *arXiv preprint ArXiv:1611.04076*, 2016.

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

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

[15] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

[16] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

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

A Proof for The Equivalence Between a Non-Normalized and a Strict Weight-Normalized Network with One Affine Weight-Normalized Layer at the End

Consider two networks with $(2n+1)$ layers each. The first network is a non-normalized network, where layers $(2k+1)$ $(0 \leq k \leq n)$ are linear layers and layers $2k$ $(1 \leq k \leq n)$ are ReLU layers. The second network is a weight-normalized network, where layers $(2k+1)$ $(0 \leq k \leq n-1)$ are strict weight-normalized layers, layer $(2n+1)$ is an affine weight-normalized layer, and layers $2k$ $(1 \leq k \leq n)$ are translated ReLU layers.

We make the following claim:

Claim 1. *The aforementioned two networks are capable of representing the same set of functions.*

First, we prove that a linear-and-ReLU combination is equivalent to a strict weight-normalized-and-TReLU combination, if both are augmented by a learned affine transformation at the end.

Lemma 2. *A linear layer, followed by a ReLU layer, followed by an affine transformation, is equivalent to a strict weight-normalized layer, followed by a TReLU layer, then by an affine transformation.*

Proof. For simplicity we consider the case where the first layer has only one output neuron. Then, a linear layer, followed by a ReLU layer, followed by an affine transformation, becomes

$$y = \mathrm{ReLU}\left(\mathbf{w}^T\mathbf{x} + \alpha\right) \cdot \gamma + \beta$$

while a strict weight-normalized layer, followed by a TReLU layer, followed by an affine transformation would be

$$y = \mathrm{TReLU}_{\alpha'}\left(\frac{\mathbf{w'}^T\mathbf{x}}{||\mathbf{w}'||}\right) \cdot \gamma' + \beta'$$

where $\mathbf{w}, \mathbf{w}', \alpha, \alpha', \beta, \beta', \gamma$ and γ' are learned parameters. The transformation

$$\begin{cases} \mathbf{w}' &= \mathbf{w} \\ \alpha' &= -\frac{\alpha}{||\mathbf{w}||} \\ \beta' &= \beta + \alpha \cdot \gamma \\ \gamma' &= ||\mathbf{w}|| \cdot \gamma \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{w} &= \mathbf{w}' \\ \alpha &= -||\mathbf{w}'|| \cdot \alpha' \\ \beta &= \beta' + \alpha' \cdot \gamma' \\ \gamma &= \frac{\gamma'}{||\mathbf{w}'||} \end{cases}$$

establishes a one-to-one correspondence between these two forms. □

We make the following observations:

Lemma 3. *A linear layer preceded by an affine transformation is equivalent to a single linear layer.*

Lemma 4. *A linear layer is equivalent to an affine weight-normalized layer.*

These proofs are trivial. Now we demonstrate the proof of claim 1 by transforming network 1 to network 2:

We perform the following procedure for each k from 1 to $(n-1)$: first, we add an affine transformation between the ReLU layer $2k$ and the linear layer $(2k+1)$. We then exchange the linear layer $(2k-1)$ and the ReLU layer $2k$ with a strict weight-normalized layer and a TReLU layer. The additional linear transformations are then removed. By adding and removing an affine transformation does not change the expressiveness of the network since a linear layer succeeds it. With an affine layer in place, the exchange would not change the expressiveness of the network either.

Finally, we change the last linear layer $(2n+1)$ to an affine weight-normalized layer.

B Implementation Details

Here we provide some implementation details regarding the weight normalized layers.

Note that for strided and transposed convolutional layers, each element in the output tensor receives an input from only a subset of the $c_i \times k_w \times k_h$ elements in the input tensor, which corresponds to the kernel, where c_i is the number of input features and k_w and k_h are the kernel width and height. Ideally, we should perform weight normalization for each of these different subsets of weights separately. In our experiments, we use a simple trick: we compute the norm of the weight for the full kernel as a whole, and divide the norm by $\sqrt{d_w \times d_h}$ where d_w and d_h are horizontal and vertical strides. This norm is used to normalize the weight in all different subsets.

The first layer in the generator deserves some special treatment. While it can be seen as a transposed convolutional layer, (since the spatial size of the input is 1×1), it can also be viewed as a fully connected layer (with shared bias between output elements from the same feature map). These two views do make a difference when Weight Normalization is in place: similar to the case above, each output element actually receives an input from a subset of c_i elements instead of $c_i \times k_w \times k_h$ elements, corresponding to the kernel. Hence, it is more appropriate to implement this layer as a weight normalized fully connected layer, which is what we did in our experiments.

We also found that weight initialization of the first layer of the generator had an impact on the effect of WN. In our experiments, initial weights are drawn uniformly from $[-0.01/\sqrt{c_i}, 0.01/\sqrt{c_i}]$ where c_i is the size of the input which is just the length of the latent code. For the convolutional layers, initial weights are drawn uniformly from $[-1/\sqrt{c_i \cdot k_w \cdot k_h}, 1/\sqrt{c_i \cdot k_w \cdot k_h}]$, as usual.

When computing the norm, a numerical stability term $\varepsilon = 10^{-6}$ is added to the sum of squared weights before taking the square root.

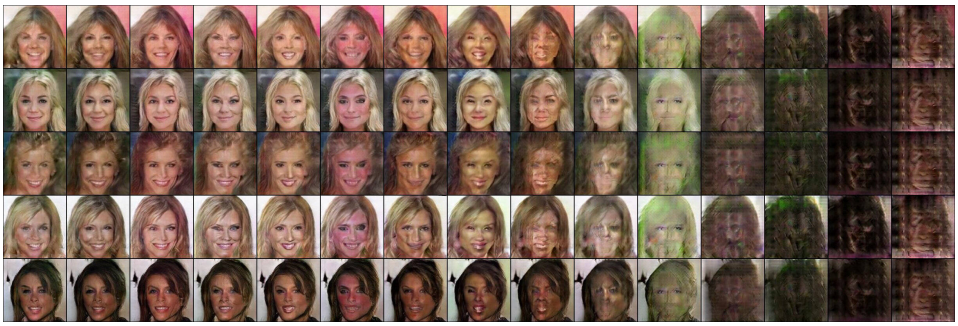

Figure 5: Samples generated by vanilla model, every 100 iterations from 134,000 to 135,400

C Additional Samples and Analysis

Here we show addtitional generated samples from the three models. In addition, we talk about certain issues with GAN training that can only be detected by visually inspecting large amounts of samples.

C.1 Vanilla Model

Figure 6 shows samples generated by the vanilla model from the same set of random codes, at iteration 30,500 (optimal iteration) and 120,000. Samples generated at 120,000 iterations are visually superior on average if each sample is inspected individually, despite a higher reconstruction loss. But notice how the diversity of the samples has decreased: the lower half of the figure is dominated by yellow and brown colors and are darker than the upper half. Even more subtle, the lower half has less variation in facial expressions. The gender diversity is also decreasing: some clear male faces in the upper half become more feminine in the lower half.

When training beyond a certain amount of iterations, the samples start to evolve toward the same direction. While the samples are still different, similar changes can be observed in each iteration. In this process, the difference between samples is gradually lost. This corresponds to the slow and steady increase of the reconstruction loss. At around 130,000 iterations, this process suddenly accelerates and then the model collapses at a certain point, as shown in figure 5.

Interestingly, this appears to be like a reversed behavior of an early stage training. In particular, training usually starts with a code that generates a similar output and changing in a similar way until the synthesized samples start to gain diversity.

C.2 Batch-Normalized Model

Figure 7 shows samples generated by the BN model from the same set of random codes, at iteration 30,500 (optimal iteration) and 110,000. At first sight, it does not show a decrease in diversity as with the vanilla model. But after comparing the samples carefully, we can discover certain repeatedly-occurring features. To see this more clearly, in Figure 8 we picked and rearranged several samples from Figure 7.

We identified two groups from samples generated at iteration 110,000. Within each group, while the appearance of the face varies considerably, almost the exact same expression is produced. When comparing these samples from ones that are generated from the same code at iteration 30,500, we found that a second group is new, while the first group has already existed for a long period of time. This indicates that the BN model had limited diversity even at its optimal iteration.

This is a indication of a different cause for mode collapse: as the training progresses, certain features become dominant. While most samples stay different, more and more start to acquire these dominating features. In the extreme case, only a handful of different possible outputs remain in the end.

Alternatively, mode collapse can suddenly happen, as is the case for the "failed" training instance 3 in Appendix D.1. We show the samples that are generated when the model collapses in Figure 9. Notice that there was no clear sign of decreased diversity prior to the sudden collapse.

Figure 6: Random samples generated by vanilla model. Upper half: iteration 30,500. Lower half: Iteration 120,000.

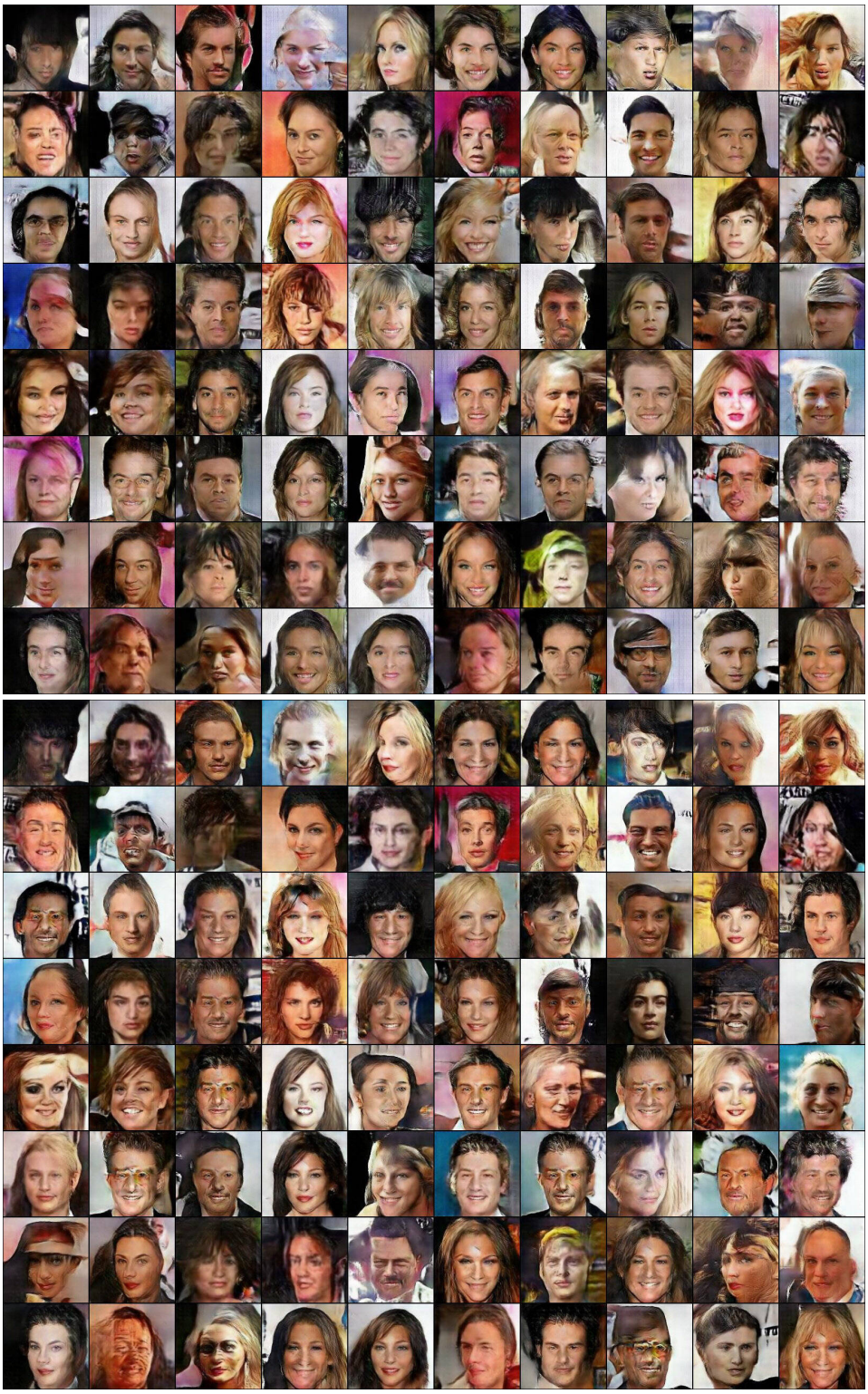

Figure 7: Random samples generated by BN model. Upper half: iteration 30,500. Lower half: Iteration 110,000.

Figure 8: Selected samples from figure 7. Row 1 and 3 are from iteration 30,500. Row 2 and 4 are corresponding samples from iteration 110,000.

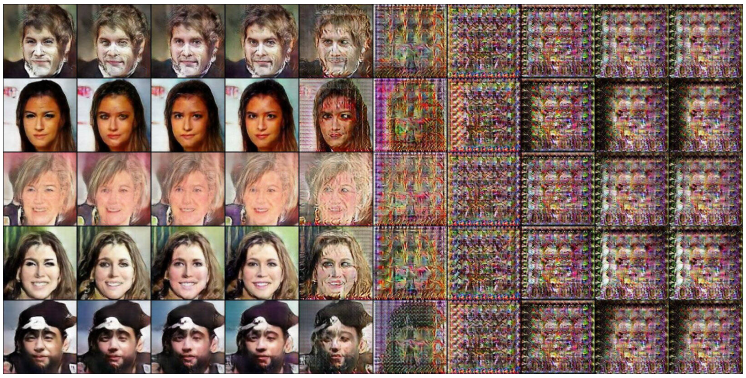

Figure 9: Samples generated by BN model instance 3, every 100 iterations from 38,300 to 39,200

C.3 Weight-Normalized Model

Figure 10 illustrates random samples generated by the optimal WN model. In terms of diversity, it is not ideal, as the samples still show a lack of color variation and an unbalanced gender ratio compared to the ground truth distribution. However, they show more variations than the vanilla model and less subtle recurring features compared to the BN model. The individual samples are of higher quality on average as well. Also note the relative low rate of "failed" or highly implausible samples.

Figure 11 shows the running reconstruction loss recorded during the whole training process of the WN model. The loss remains nearly constant after 300,000 iterations, which demonstrates the stability of the WN model.

C.4 Random Reconstructed Samples

Figure 12 shows a random selection of reconstructed samples.

Figure 10: Random samples generated by WN model at iteration 463,000

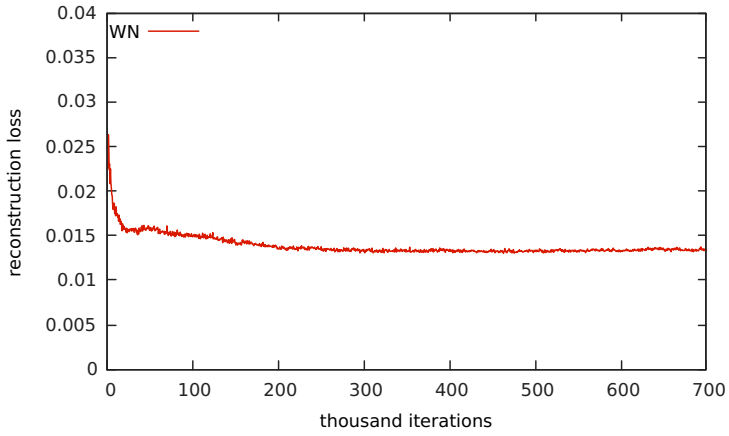

Figure 11: Running reconstruction loss during training of WN model

Table 3: Optimal reconstruction loss of repeated training of vanilla model.

Instance	Optimal iteration	Running loss
vanilla-1	30,500	0.014509
vanilla-2	34,500	0.014703
vanilla-3	31,000	0.014734

D Additional Experiments

D.1 Additional Training Instances of Vanilla and BN Models

Figures 13 and 14, and Tables 3 and 4 show the reconstruction loss recorded during all training instances of the vanilla and batch-normalized models.

We can see that the three instances of vanilla model gave almost identical loss curves. They achieved similar optimal loss at similar times, and the mode collapse also happened around the same time.

In addition to the instability observed for each training instance, the batch-normalized models also showed "meta-instability" as the behavior differed considerably between each training instance. Notably, in the third instance, mode collapse happened very early on. The training did recover to some extent, but the model was never able to regain the same sampling diversity as before the mode collapse.

D.2 Additional Models

For completeness, we also compare different formulations of Weight Normalization. The first one is a full-affine WN model, constructed from the WN model by replacing all strict weight-normalized layers by affine weight-normalized layers and all TPReLU layers by PReLU layers. The second one is a model with Weight Normalization plus mean-only Batch Normalization, as used in [14], constructed by taking the affine WN model and adding mean-only Batch Normalization at those places where regular Batch Normalization layers would be used in the BN model.

Table 4: Optimal reconstruction loss of repeated training of batch-normalized model.

Instance	Optimal iteration	Running loss
BN-1	37,000	0.017349
BN-2	30,500	0.017199
BN-3	16,000	0.018810

Figure 12: Randomly reconstructed test samples. From left to right in each group: test sample, vanilla reconstruction, BN reconstruction, WN reconstruction.

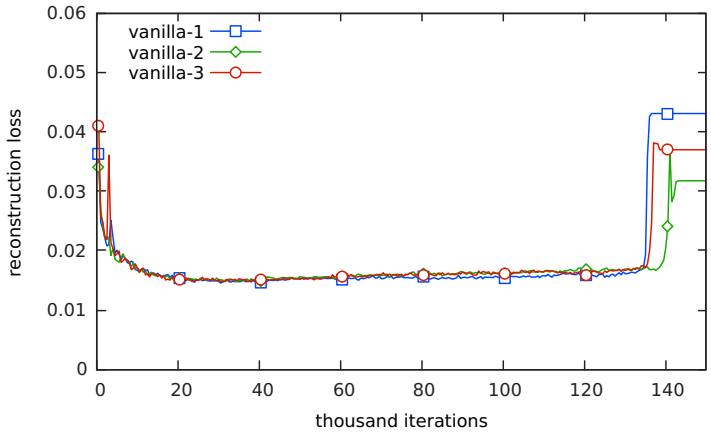

Figure 13: Running reconstruction loss of repeated training of vanilla model.

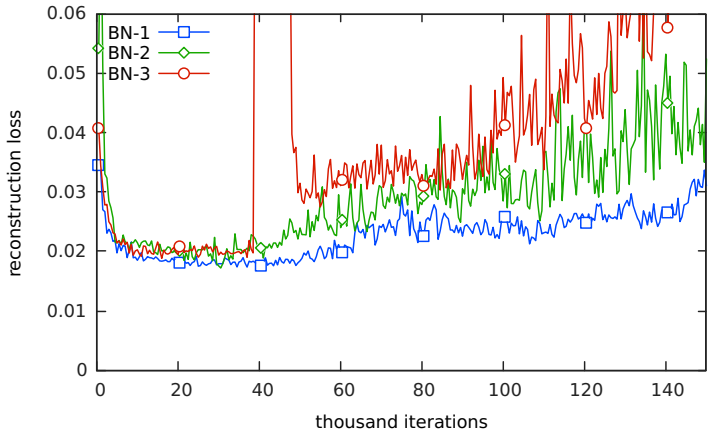

Figure 14: Running reconstruction loss of repeated training of batch-normalized model.

The reconstruction loss for the first 150,000 steps are shown in Figure 15 and Table 5. The results of the WN model are included for comparison.

The WN with mean-only BN model achieved worse reconstruction than the vanilla model. In addition, although less severe than the BN model, it results in similar fluctuations of the BN model. We believe that the major advantage of WN over BN is its independence from batch statistics. By adding mean-only BN, this dependency is re-introduced, which harms the stability of the model.

We are particularly interested in the comparison between the WN model and the affine-WN model, as it compares our formulation of Weight Normalization against the one originally one in [14]. For this purpose, we trained the affine-WN model using also 700,000 iterations. Figure 16 shows the comparison of their reconstruction loss during training.

Table 5: Optimal reconstruction loss of additional models.

Model	Optimal iteration	Running loss
affine-WN	51,000	0.014034
WN+mean-BN	19,500	0.016639
WN	463,000	0.013010

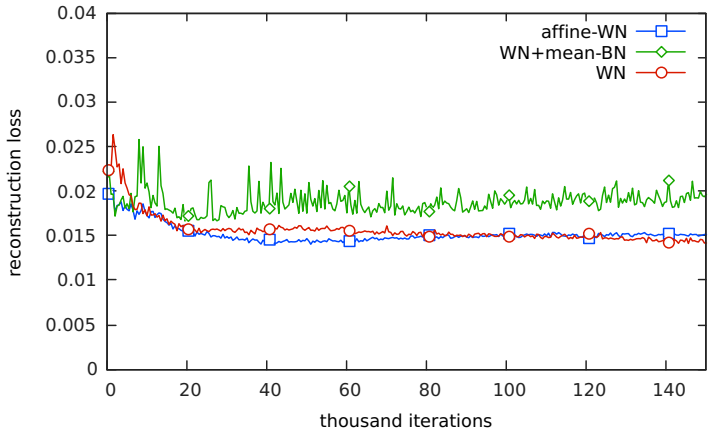

Figure 15: Running reconstruction loss during training of additional models.

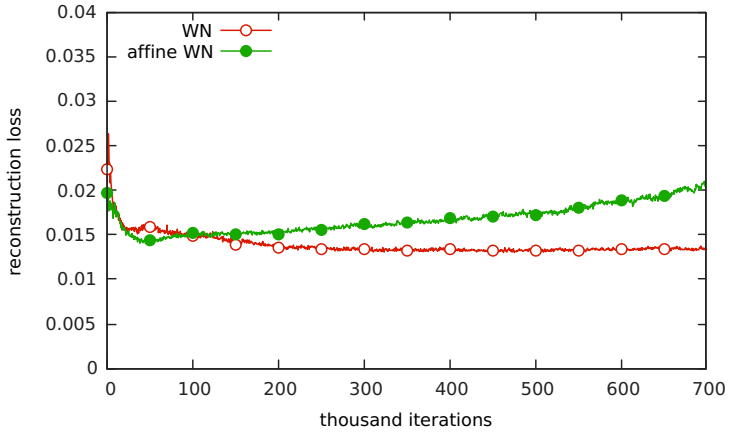

Figure 16: Running reconstruction loss during training of the WN and affine-WN models

After making a quick descent in the beginning, the running reconstruction loss of the affine-WN model start to increase steadily. Unlike the vanilla model, the generated samples kept stable and are of high-quality. Hence, we evaluate both the optimal model and the model at iteration 700,000. The results are compared with the WN model in table 6.

Surprisingly, the affine-WN model at iteration 700,000 yields equally good 2,000-step reconstructions as with iteration 51,000, when the model achieved optimal 50-step reconstruction. Both of them, however, are about 7.5% worse than the WN model.

The difference between the two models reveals something interesting: the WN model is equivalent to the affine-WN model in terms of the set of functions they can express, but uses fewer parameters as the learned affine transformations are removed except for the last layer. By removing these redundant parameters, the simpler model does not tend to degrade as the complex one does. Thus, we suggest to

Table 6: Optimal and converged reconstruction loss of WN and affine-WN models

Model	Iteration	Running loss	Final loss
affine-WN	51,000	0.014034	0.005941
affine-WN	700,000	0.020478	0.005939
WN	463,000	0.013010	0.005525

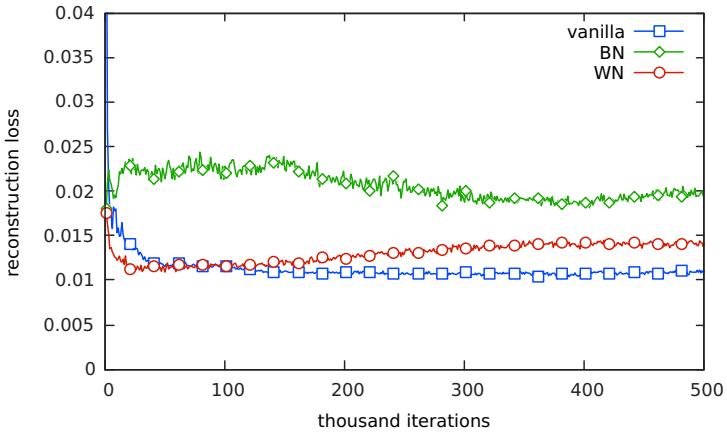

Figure 17: Reconstruction loss during training on CIFAR-10.

Table 7: Optimal and converged reconstruction loss of the models on CIFAR-10

Model	Iteration	Running loss	Final loss
vanilla	387,000	0.010382	0.003413
BN	1,000	0.017987	0.004904
WN	50,000	0.010906	0.003509
vanilla	500,000	0.010948	0.003414
BN	500,000	0.019287	0.005421
WN	500,000	0.014195	0.003269

assess our version of Weight Normalization on other deep learning tasks to see if it improves over the original formulation as well.

D.3 Experiments on CIFAR-10 Dataset

There are 60,000 images (training plus validation) of size 32×32 in the CIFAR-10 dataset. We construct models in similar ways as for CelebA, but begin with 96 output channels for the first convolutional layer and stop further convolutions when the spatial size of the feature map reaches 4×4. The length of the code (256) and training batch size (32) remains the same.

We use 58,000 images for training and 2,000 images for evaluation. During training, evaluation is performed every 1,000 training iterations on 400 images, with 50 gradient descent steps. Final evaluation is performed on the whole test set with 2,000 gradient descent steps.

BN is still the worst model compared to the vanilla and WN models. Now the WN model achieves optimal loss early on, but then becomes worse. On the other hand, the vanilla model keeps improving. However, both models converges, as shown by the flat section in the loss curve, between iterations 400,000 and 500,000. So we take these two models at iteration 500,000 for evaluation in additional to the optimal 50-step models. The BN model does not actually converge, as the rapidly changing "recurring feature", discussed in Section C, occurs. For completeness however, we also take the BN model from iteration 500,000 for evaluation.

The seemingly worse 500,000-iteration WN model turned out to give the best final reconstruction result. A more careful examination of the reconstruction process revealed that the 500,000-iteration WN model achieved better reconstruction than the optimal vanilla model at around 400th reconstruction step. We acknowledge that this exposes a weakness of our evaluation method: performing the reconstruction for too few steps may give inaccurate results, while too many steps would be time-consuming, which makes it unsuitable for training process monitoring.

Figure 18 shows random samples generated by the three models, at their optimal iteration and at iteration 500,000. While the visual quality of samples from all models are good, the results are

Figure 18: CIFAR-10 samples. Top to bottom: vanilla model, BN model, WN model. Left: optimal iteration. Right: iteration 500,000.

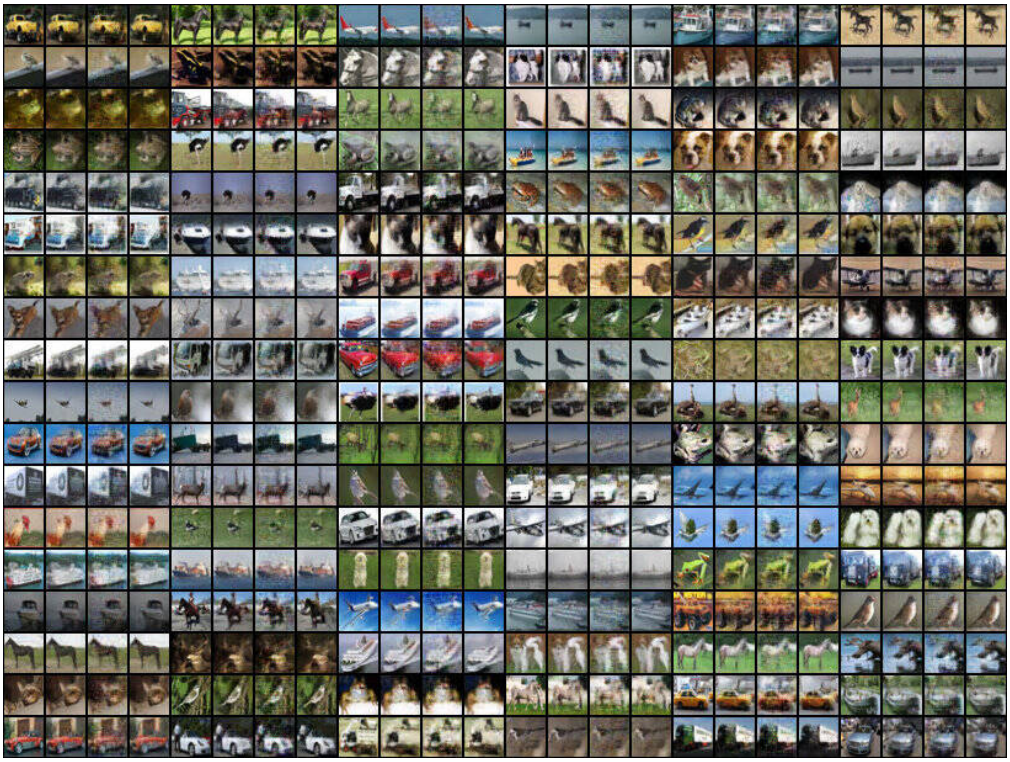

Figure 19: CIFAR-10 reconstructions. Each group, left to right: test sample, vanilla model, BN model, WN model.

Table 8: Reconstruction loss of the models on LSUN bedroom dataset

Model	Optimal iteration	Running loss	Final loss
BN	125,000	0.020943	0.011051
WN	478,000	0.016266	0.008546

consistent in terms of diversity with the analysis in Appendix C. The vanilla samples look dull and are dominated by one color (green); the BN samples show a recurring feature (marked with red border).

Figure 19 shows random test samples and reconstructed ones.

D.4 Experiments on LSUN Bedroom Dataset

There are 3,033,042 images in the bedroom class of the LSUN dataset, with images having 256 pixels on the shorter side. Unlike many published results on this dataset, we use the full-sized images. We crop with centered 256×256 patches but do not down-sample the image. We construct models in a similar way as with CelebA, but stop further convolution when the spatial size of the feature map reaches 4×4 and use a code length of 512. Due to the large size of the images and the network, we reduce the batch sizes to 12 in order to save time and memory.

We use 2,000 images for evaluation and the rest for training. During training, evaluation is performed every 1,000 iterations on 200 images, with 50 gradient descent steps. Final evaluation is performed on the whole test set with 2,000 gradient descent steps.

For the vanilla model, training fails constantly, even when we reduce the learning rate by a factor of 10 (to 10^{-5}), so only the BN and WN models are compared here. The BN model collapsed at iteration 330,800. The WN model was trained with 600,000 iterations. The reconstruction loss is shown in Figure 20 and Table 8.

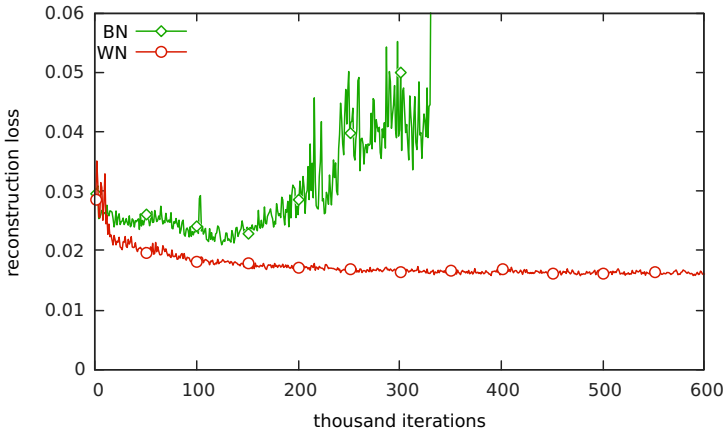

Figure 20: Reconstruction loss during training on LSUN bedroom dataset.

Random samples generated by the two models are shown in Figures 21 and 22. Reconstruction of random samples are shown in figure 23.

In the BN samples, recurring tile-like artifacts are observed. The best quality samples that are generated by the BN model look sharp and clean, while the WN model reproduces details more accurately.

Figure 21: LSUN samples generated by the BN model at iteration 125,000

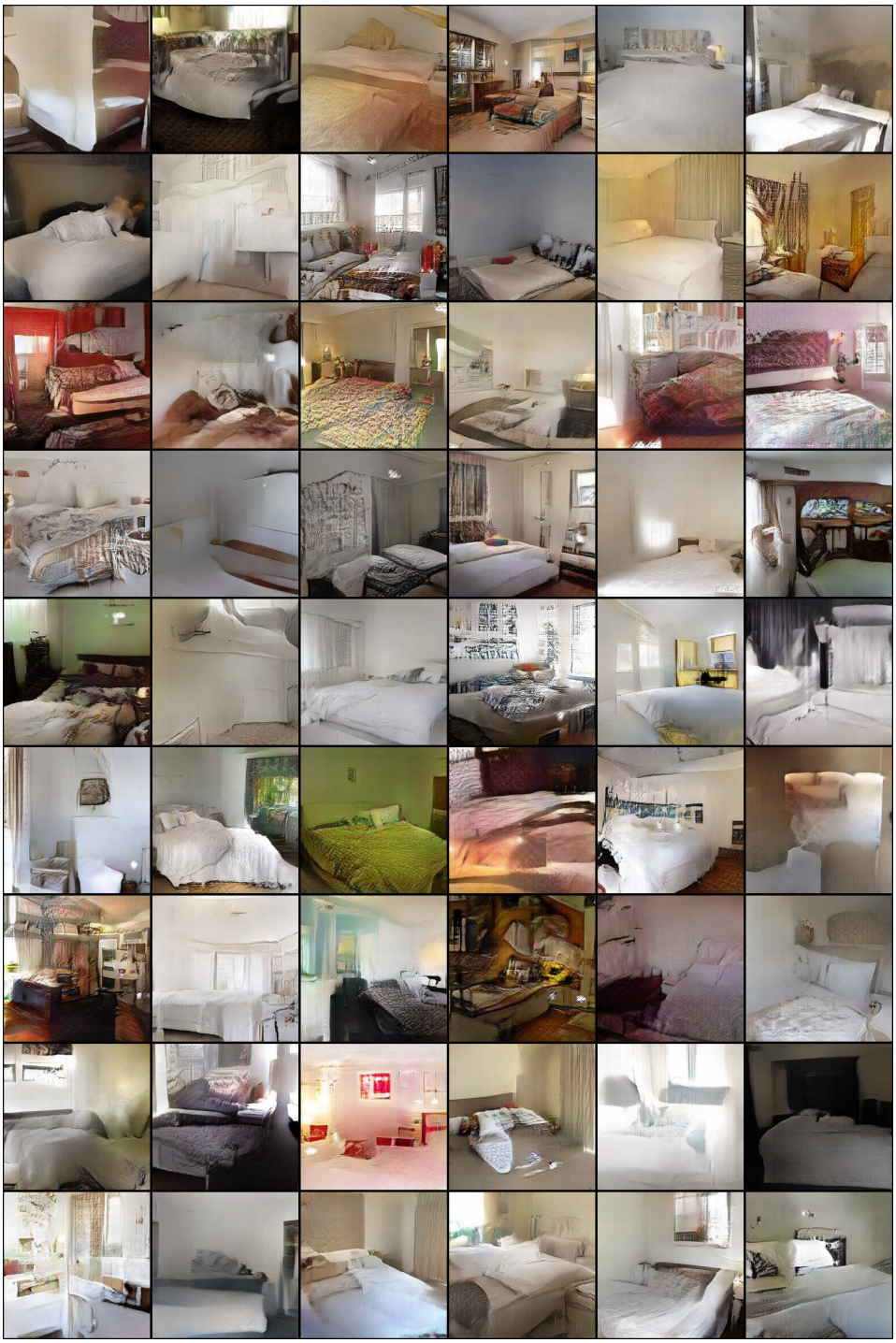

Figure 22: LSUN samples generated by the WN model at iteration 299,000

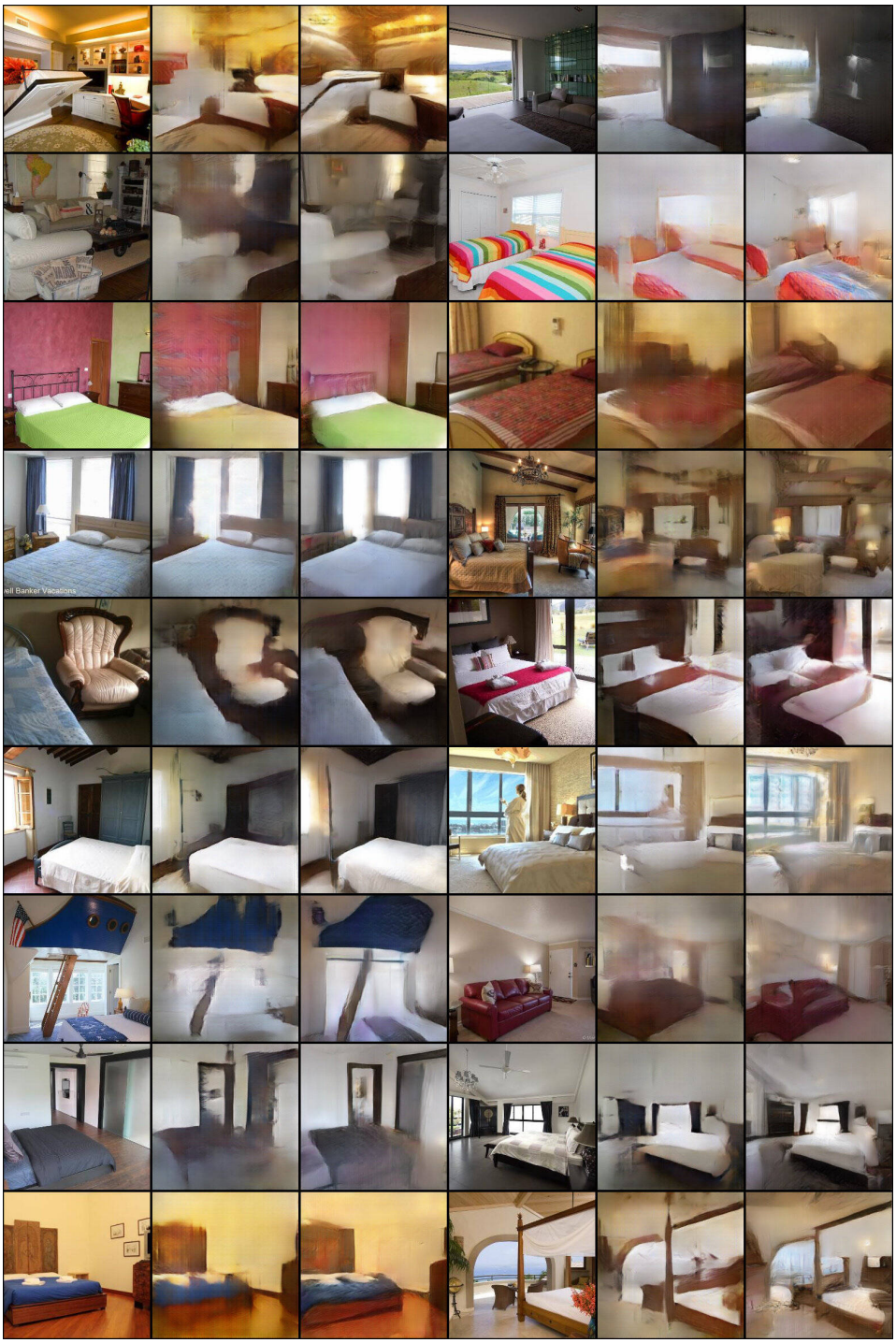

Figure 23: LSUN reconstructions. Each group, left to right: test sample, BN model, WN model.

E Connection to Wasserstein GAN

For Wasserstein GANs [1], the discriminator is replaced with a critic, that is K-Lipschitz-continuous for some constant K and only depends on the structure of the network. To achieve this, they clipped the parameters of the critic network to a small window $[-0.01, 0.01]$ after each parameter update during training.

We claim that our weight-normalized discriminator is Lipschitz-continuous with a small modification:

Claim 1. *The weight-normalized discriminator proposed in this paper is K-Lipschitz-continuous for some constant K if the sigmoid layer is removed and the only affine weight-normalized layer is replaced by a strict weight-normalized layer.*

To see this, we first prove the following lemma:

Lemma 2. *For a strict weight-normalized layer*

$$y_i = \frac{\mathbf{w_i}^T \mathbf{x}}{||\mathbf{w_i}||} \quad ,$$

where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{n \times m}$ is the weight matrix and $\mathbf{w}_i$ the i-th column of $\mathbf{W}$. If the loss function of the network is L, then

$$\sum_{i=1}^{n} \left| \frac{\partial L}{\partial x_i} \right| \leq \sqrt{n} \cdot \sum_{i=1}^{m} \left| \frac{\partial L}{\partial y_i} \right| \tag{8}$$

Proof.

$$\begin{aligned}
\sum_{i=1}^{n} \left| \frac{\partial L}{\partial x_i} \right| &= \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\left| \frac{\partial y_j}{\partial x_i} \right| \cdot \left| \frac{\partial L}{\partial y_j} \right| \right) \\
&= \sum_{j=1}^{m} \left(\left| \frac{\partial L}{\partial y_j} \right| \sum_{i=1}^{n} \left| \frac{\partial y_j}{\partial x_i} \right| \right) \\
&= \sum_{j=1}^{m} \left(\left| \frac{\partial L}{\partial y_j} \right| \sum_{i=1}^{n} \frac{|w_{ij}|}{||\mathbf{w}_j||} \right) \\
&\leq \sum_{j=1}^{m} \left(\left| \frac{\partial L}{\partial y_j} \right| \cdot \sqrt{n} \right) \\
&= \sqrt{n} \cdot \sum_{j=1}^{m} \left| \frac{\partial L}{\partial y_j} \right|
\end{aligned}$$

□

For a strict weight-normalized convolution layer with c_I input channels and kernel size $k_W \times k_H$, change $\sqrt{n}$ in inequality 8 to $\sqrt{c_I \cdot k_W \cdot k_H}$.

Note that in our implementation, the learned slope of parametric ReLU layers are clipped to $[0, 1]$, so the following becomes obvious:

Lemma 3. *For a TPReLU layer with input $\mathbf{x}$ and output $\mathbf{y}$ in $\mathbb{R}^n$,*

$$\sum_{i=1}^{n} \left| \frac{\partial L}{\partial x_i} \right| \leq \sum_{i=1}^{n} \left| \frac{\partial L}{\partial y_i} \right|$$

Now it is easy to see that claim 1 is true since for each layer, the sum of absolute value of gradients grows by at most a constant factor. So with such a modification, our discriminator changes into a WGAN critic.