Stabilizing GANs: A Case for Spectral Normalization

Liam Donovan
Department of Mathematics and Statistics, University of Massachusetts Amherst

May 2025

Abstract

Spectral normalization is a technique used to enforce Lipschitz continuity in the discriminator of a Generative Adversarial Network (GAN), improving training stability. This write-up reviews the motivation, mathematical formulation, and empirical effects of spectral normalization, including a comparison with other gradient norm control methods: weight clipping and penalty-based approaches.

1 Introduction and Motivation

Generative Adversarial Networks (GANs) are generative models which aims to iteratively improve samples by taking a **generator**, G and a **discriminator** D and letting them improve each other.

Given some data, $X_1, \dots, X_n \overset{iid}{\sim} P_X$ the generator produces similar samples from a simple sampling distribution P_Z; e.g., a Gaussian. That is, letting $\tilde{P}_X$ be the distribution of "fake samples" (which should look like P_X).

$$G : P_Z \mapsto \tilde{P}_x$$

To ensure that samples are "acceptable", one would define a discriminator, D which outputs a probability of a given sample being "real" (being from P_X, as opposed to being generated by G).

$$D : \mathbb{R}^d \mapsto [0, 1]$$

Since this is a binary classification problem, a natural choice for an objective function is the loss function for classification: **binary cross entropy**, but for the real and the generated data.

$$V(G, D) = E_{x \sim P_X}(\log(D(x)) + E_{z \sim P_Z}(\log(1 - D(G(z))))$$

Note that we drop the negative sign in standard log loss context, making this a maximization problem for the discriminator. The original paper by Goodfellow et al. [2014] does this also.

The discriminator wants to maximize this; wants to increase the second term without sacrificing the first term (reject as many fake samples, still letting the real one's through). On the other hand, the generator wants to minimize this, in order to generate samples which get marked "real" by the discriminator.

So, we have a "fight" between D and G:

$$\min_G \max_D V(G, D)$$

This can be done by gradient decent and gradient ascent (more commonly just gradient descent, for $-V(G, D)$ on the maximization step).

As one would expect, the performance of this procedure is more related to the performance of D than G. Suppose that D is poorly chosen, resulting in many false samples being marked real, so the generator is not incentived to learn. Conversely, suppose that D is very, very strong, then it will always reject fake samples, which also causes no improvement in G. We draw the following analogy:

> *Discriminators are like teachers: one which is too lenient will let poor work pass through, resulting in the students learning nothing. Conversely, one who is too strict will fail every student and the students do not know how to improve. The optimal teacher is challenging but fair; who changes the strictness relative to the level at which the students have already learned.*

Thus, we should consider the optimal form of the discriminator, for a fixed generator.
The optimal discriminator, D^* has the form:

$$D^* = \frac{P_X(x)}{P_X(x) + \tilde{P}_X(x)}$$

Plugging this into the objective function:

$$\max_D V(D,G) = -\log(4) + 2\underbrace{\left(\frac{1}{2}\mathrm{KL}\left(P_X \parallel M\right) + \frac{1}{2}\mathrm{KL}(\tilde{P}_X \parallel M)\right)}_{\mathrm{JS}(P_X \| \tilde{P}_X)}$$

where $\mathrm{JS}(P_X \parallel \tilde{P}_X)$ is the **Jensen-Shannon divergence**, and:

$$M = \frac{1}{2}(P_X + \tilde{P}_X)$$

While JS divergence is bounded and symmetric, it suffers from a key limitation: when P_X and $\tilde{P}_X$ have disjoint support (a common situation early in GAN training), the discriminator can perfectly distinguish real from fake. In this case, the JS divergence reaches its maximum of $\log 2$, and more importantly, the *gradient of the generator's loss vanishes.*

This flat landscape leaves the generator with no signal for how to improve, resulting in stalled learning. Even replacing the KL divergence with other probability divergences like total variation does not avoid this issue — they also saturate when supports do not overlap.

1.1 The Wasserstein-1 Distance: Some History

In 1781, Gaspard Monge posed an "engineering-esque" problem: how to move a pile of dirt into a hole in the most efficient way. This involves finding a mapping from one distribution of mass to another while minimizing the total transport cost. This formulation, now known as the problem of *optimal transport*, required each unit of mass to be moved entirely from one point to another — no splitting allowed. While Monge solved special cases, he was unable to produce a general solution.

In the 1940s, Leonid Kantorovich revisited this problem using the tools of modern measure theory, developed largely through Lebesgue's work in the early 20th century. Kantorovich relaxed Monge's formulation by allowing mass to be *split*, introducing the idea of a joint transport plan — what we now call a *coupling* — between source and target distributions. This relaxation transformed the problem into a convex optimization problem and made it mathematically tractable.

Later developments in measure theory clarified that a probability space is a special case of a measure space: one where the total measure is 1. This recognition allowed the machinery of optimal transport to be applied in probabilistic settings.

In the 1960s, Leonid Vasershtein (whose name was later transliterated as Wasserstein) studied metrics on probability spaces and connected them to the functionals from optimal transport. He defined what is now known as the **Wasserstein-1 distance**:

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right]$$

where γ is a coupling of the probability measures P and Q, and the cost function is the L_1-norm distance between points. The subscript "1" emphasizes the choice of the L_1 metric as the ground cost; in general, one can define W_p distances by using $\|x - y\|^p$ as the cost and taking the p-th root of the expectation.

1.2 The Kantorovich–Rubinstein Duality

Computing the optimal coupling is intractable in higher dimensions. Luckily, Kantorovich and Rubinstein would derive formulation which involves an optimization over a class of functions, instead.

$$W_1(P, Q) = \sup_{\|f\|_{\text{Lip}} \leq 1} (E_{x\in P}(f(x)) - E_{x\in Q}(f(x)))$$

where $\|f\|_{\text{Lip}} \leq 1$ refers to a function being **Lipschitz-1**:

$$|f(x) - f(y)| \leq \|x - y\| \quad \forall x, y$$

That is, the function cannot grow faster than the inputs; its slope is bounded. Seeing the relation between rates of change, this can equivalently be stated:

$$\|\nabla_x f(x)\| \leq 1 \quad \forall x$$

This is better, since functions can be well approximated by neural networks in practice. However, it is infeasible to check the gradient everywhere, so we seek an implementation technique to guarantee this condition.

Since f does not have a codomain of $[0, 1]$, it is not a valid discriminator. It is often called the **critic**. This implementation of GANs is called **Wasserstein-GANs**, often shortened to WGANs (Arjovsky et al. [2017]).

2 Discussion of Earlier Techniques

Recall that a standard **feedforward neural network** is a composition of affine transformations; i.e., matrix vector multiplications with an extra vector sum. Each layer, ℓ, has the form:

$$h_\ell = W_\ell x + b_\ell$$

where x is the output of the previous layer, W is a weight matrix, and b is a bias vector. In the first layer, x corresponds to the input vector.

Then, to ensure that the output is bounded, we then apply a function, called an *activation function*, σ:

$$f_\ell(x) = \sigma_\ell(W_\ell + b_\ell) = \sigma_\ell(h_\ell)$$

h_ℓ is often called the *pre-activation* state at layer ℓ.

The overall network can thus be viewed as a composition of such transformations:

$$f(x) = f_L \circ \cdots \circ f_2 \circ f_1(x)$$

Each layer contributes to the network's overall Lipschitz constant, so we must ensure that each layer's growth is controlled to guarantee the gradient of $f(x)$ is bounded.

Notice that the bias term will vanish under a gradient, thus we can safely ignore b and focus on bounding the behavior of the weight matrix.

2.1 Weight Clipping

A naive approach to achieving the Lipschitz condition is simply to not let the weights in the weight matrix grow too large. For example, after each gradient updating step:

$$W \leftarrow \text{clip}(W, -c, c)$$

for some small constant c. A typical choice is $c = .01$.

Notice this does not *enforce* the Lipschitz condition, it merely *encourages* it. Given a deep enough network, the norm can still vanish or explode. Moreover, since this method is trying to limit the growth for the entire network, each layer's gradient is quite small, leading to slow learning.

2.2 Gradient Penalty

Another approach is the control the gradient directly. This is the approach of **Gradient Penalty** (GP) (Gulrajani et al. [2017]). A corollary of the Kantorovich-Rubinstein duality is that the optimal function is, in practice, piecewise linear between P_X and $\tilde{P}_X$, with each gradient norm being 1. That is, the gradient is either ± 1 (in 1-D).

In optimal transport, the map only needs to be optimal *along the transport path*. Similarly, we claim that we can restrict the search to such paths between the real and generated samples. The enforce the Lipschitz-1 condition, we add a penalizing term to the critic loss:

$$\mathcal{L}(\tilde{P}_X, P_X) = E_{\tilde{x} \in \tilde{P}_X}(f(x)) - E_{x \in P_X}(f(x)) + \lambda \cdot \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} \left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2 \right]$$

where $\hat{x}$ are samples drawn uniformly along straight lines between real and generated points:

$$\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}, \quad \epsilon \sim \mathcal{U}(0, 1)$$

and λ controls the "severity" of the penalty. Empirically, $\lambda = 10$ has shown to perform well.

This method has shown to be quite effective, much more so than weight clipping, but has a few pragmatic shortcomings. Namely, it does not enforce the Lipschitz-1 condition directly, it simply encourages it at the interpolated points. Even if this is achieved, the function may not be Lipschitz-1 away from these points; *it does not result in global Lipschitz-1.* Also, this method requires backpropagation at $\hat{x}$, instead of just at the weights and biases, which can slow down computation.

Observe that neither method actually enforces the Lipschitz-1 condition, directly. Spectral normalization will do so explicitly.

3 Spectral Normalization

In order to directly examine the behavior of the neural network, we define the *Lipschitz norm:*[1]

$$\|f\|_{\text{Lip}} = \sup_{x \neq y} \frac{\|f(x) - f(y)\|}{\|x - y\|}.$$

One may observe that this is simply the Lipschitz constant of f. Thus, we require:

$$\|f\|_{\text{Lip}} \leq 1.$$

By using the triangle inequality [2], one finds:

$$\|f\|_{\text{Lip}} \leq \prod_{\ell=1}^{L} \|\sigma_\ell\|_{\text{Lip}} \cdot \|f_i\|_{\text{Lip}}$$

The Lipschitz constant of many common activation functions, like ReLU, LeakyReLU (if the slope is bounded by 1), and Tanh are Lipschitz-1. Thus, we must bound the Lipschitz constant of each f_i by 1.

Consider one layer, one finds:

$$\|f_\ell\|_{\text{Lip}} = \|\sup_{x \neq 0} \frac{\|W_\ell x\|}{\|x\|} = \|W_\ell\|_2.$$

The 2-norm of a matrix is simply its largest *singular value*. Thus, it is often called the *spectral norm*. Thus, if $\|W\|_2 \leq 1$ at each layer (dropping the subscript), then f will be Lipschitz-1.

The straightforward way to enforce this is by normalizing:

$$\bar{W}_\ell = \frac{W_\ell}{\sigma_{max}(W_\ell)},$$

this is called **spectral normalization** (Miyato et al. [2018]). This method directly enforces the global Lipschitz condition and does not require extra computation in backpropagation. We must however, examine the cost of computing the spectral norm.

3.1 Computing the Spectral Norm by Power Iteration

Computing the largest singular value is often done by *power iteration*. Recall that a singular value of W is nothing but the square root of the largest eigenvalue of $W^T W$.

For a diagonalizable matrix, A: define the iteration

$$v_{k+1} = \frac{Av_k}{\|Av_k\|}$$

for any vector v, it then follows

$$\lim_{k \to \infty} v_k \to \lambda_{max} \approx v_k^T A v_k$$

$W^T W$ is always diagonalizable, so we have:

[1] The Lipschitz "norm" is technically only a semi-norm, as it fails definiteness: it can be zero even when $f \not\equiv 0$, for instance any nonzero constant function. It does, however, satisfy the other norm properties we need.

[2] More precisely, the chain rule for the Lipschitz constant.

1. Initialize a random vector $v_0 \sim \mathcal{N}(0, I)$ and normalize:

$$v_0 \leftarrow \frac{v_0}{\|v_0\|}.$$

2. For $k = 0, 1, \ldots, K-1$, alternate

$$u_{k+1} = W\, v_k, \qquad u_{k+1} \leftarrow \frac{u_{k+1}}{\|u_{k+1}\|},$$
$$v_{k+1} = W^T\, u_{k+1}, \qquad v_{k+1} \leftarrow \frac{v_{k+1}}{\|v_{k+1}\|}.$$

3. Return the estimate

$$\sigma_{\max}(W) \;\approx\; u_K^T (W\, v_K).$$

This generally need not be done more than 1-3 times to achieve a very close approximation, so its cost is very low.

4 Empirical Observations

We compare the methods for the MNIST data set:

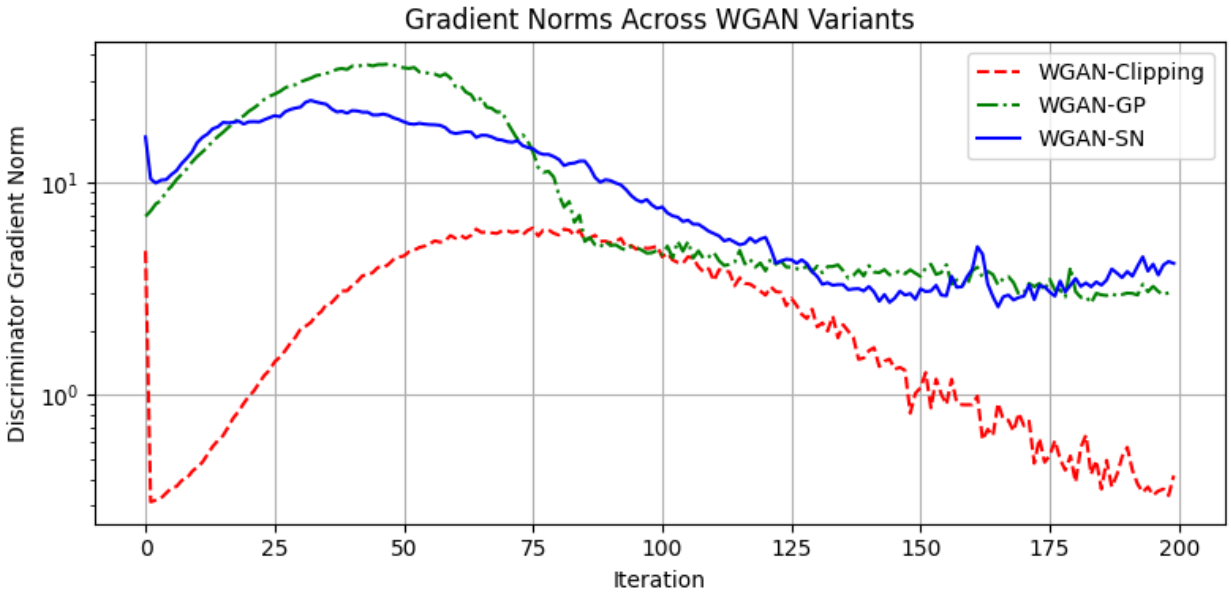

Figure 1: Observe that the SN gradient does has more "controlled" gradient norms across iterations.

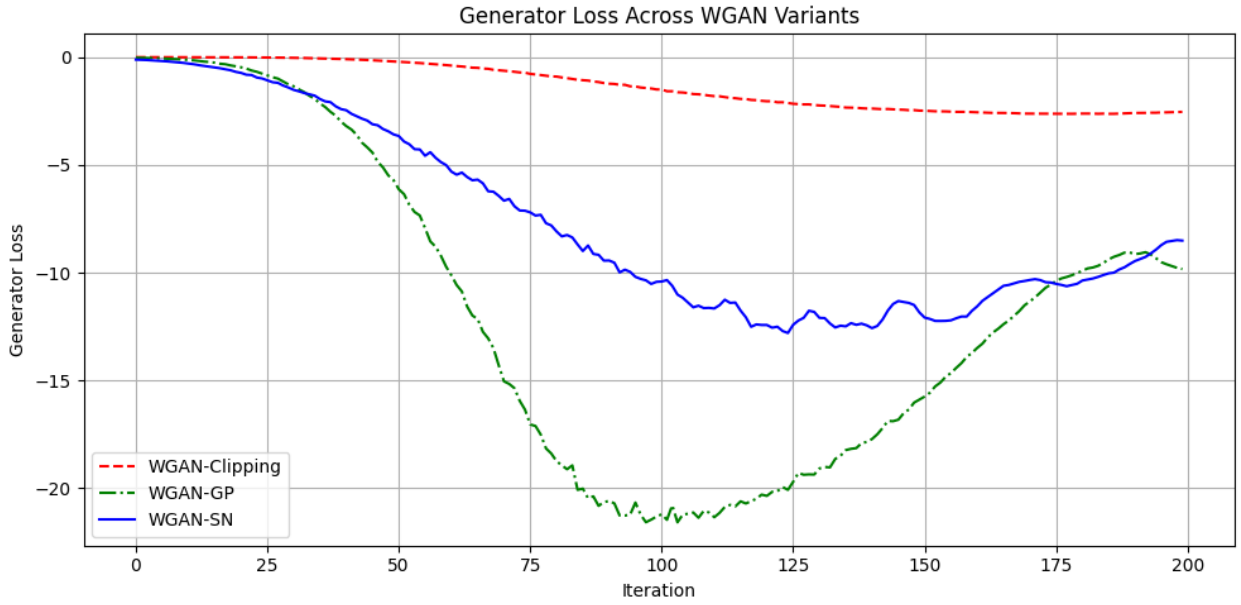

Figure 2: Observe that the GP tends to over correct, while the SN shows a more gradual decrease in generator loss. The weight clipping method shows very little loss over 200 iterations.

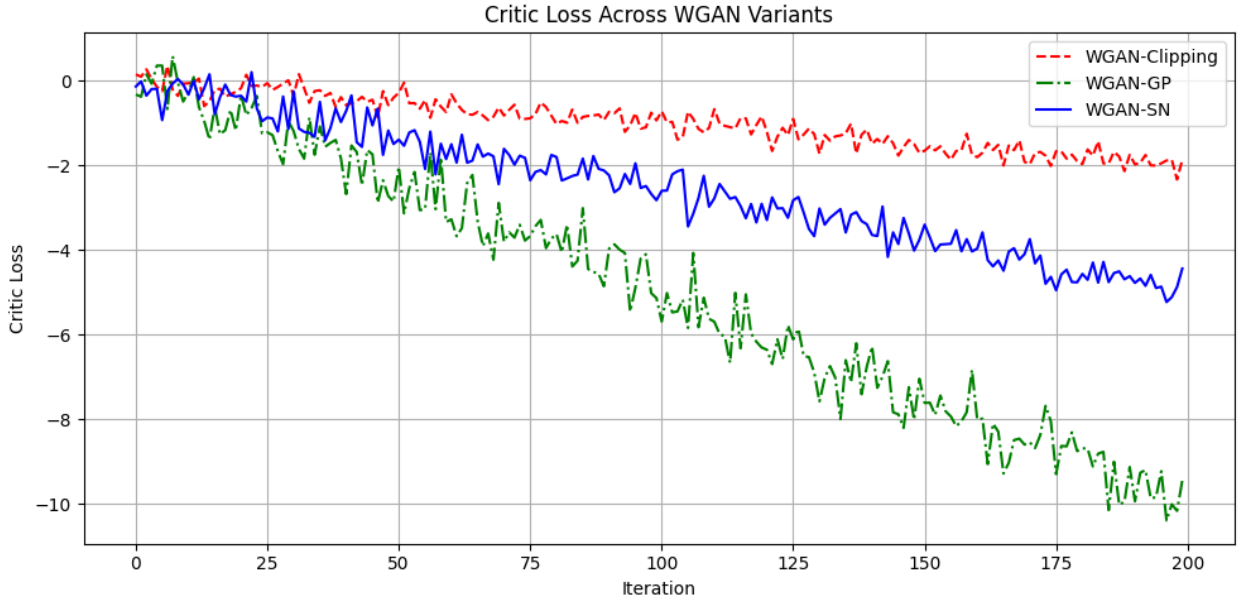

Figure 3: The critic has a more gradual gradient decease in loss across iterations. Here, clipping is likely to under fit, while GP is likely to overfit.

5 Conclusion

Spectral normalization provides a simple yet effective method of enforcing Lipschitz continuity and stabilizing GAN training. Unlike gradient penalties, it introduces no hyperparameters and maintains architectural simplicity. It is also computationally cheaper. SN also, practically, eliminates

the issue of exploding gradients posed by weight clipping by providing a strict bound. It is possible for gradient vanishing, still, specifically if one chooses tanh or the sigmoid activation, since they have Lipschitz constants below 1. There are many adaptations to deal with this, such as simply using ReLU methods, which have Lipschitz constant 1 and methods of bounding the singular values from below.

Most implementations simply use ReLU methods.

References

Martin Arjovsky, Soumith Chintala, and Leon Bottou. Wasserstein gans. *arXiv preprint arXiv:1701.07875*, 2017.

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 (NeurIPS)*, volume 27. Curran Associates, Inc., 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. URL https://arxiv.org/abs/1704.00028.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=B1QRgziT-.