CONTENTS

A MORE RELATED WORKS

We briefly review related works in the following categories: normalization and regularization approaches; preconditioning in machine learning; related works on GAN training. Related works on the theoretical analysis of gradient descent for neural nets are briefly discussed in Appendix D.

Normalization and regularization. Bounding the weight matrices during training or at initialization has been studied for a long time. These approaches consider mainly three questions: *what time range* (controlling initial point or all iterates); *what modification* (built-in layer or regularization); and *what goal* (upper bound on Frobenious norm, orthogonal matrix, or control spectrum). We compare a few popular methods in Table 3. To our knowledge, our method is the first one that applies a built-in layer to control the spectrum over all training iterations. In Table 4, we show empirical results of BN, GP, WN, orthogonal regularization, SN, SVD with D-Optimal Regularizer and our PC methods (FPC and APC) on standard CNN for CIFAR-10 and STL-10.

Lin et al. (2020) analyzed why SN works well for GAN, and proposed a variant of SN called bidirectionalSN (BSN). The improvement of BSN over SN is similar to the improvement of Xavier initialization over LeCun initialization: BSN considers both the input dimension and the output dimension and scales the weight tensor by two versions of the reshaped matrices. This technique is somewhat orthogonal to our PC technique, and these two techniques can potentially be combined together. We leave the exploration of the combination to future work.

Preconditioning in machine learning. In a broad sense, any method that aims to reduce the condition number of a matrix can be viewed as preconditioning. Researchers sometimes call a method $\theta \leftarrow \theta - \eta M(\theta)\nabla\mathcal{L}(\theta)$ a "preconditioned" gradient descent, such as Li et al. (2015). In this sense, adaptive gradient methods such as AdaGrad (Duchi et al., 2011) are also viewed as "preconditioned" gradient descent. These methods are universal to any optimization problem and do not utilize the structure of neural nets. In contrast, we apply preconditioning to weight matrices, which rely on the fact that the neural net consists of multiple weight matrices.

Odena et al. (2018) encourage the generator to be well-conditioned. They compute the condition number of the input-output Jacobian, which is based on an earlier understanding of the training (small condition number of input-output Jacobian). In contrast, our result is based on a more recent understanding of the training (small condition number of NTK).

Improving GANs by resolving target issues. Identifying the training issues of GANs is often the first step to understanding. We briefly review three types of issues identified for GAN training.

First: undesirable property of the loss function. Early work on GANs (Goodfellow et al., 2014) studied a loss related to the J-S (Jenson Shannon) distance. Arjovsky & Bottou (2017) pointed out that the J-S distance is ill-defined for two distributions with disjoint support, and proposed to use a loss function based on the Wasserstein distance, which is referred to as WGAN.

Second: cyclic behavior (non-convergence). This issue has been well recognized (Mescheder et al., 2018; Balduzzi et al., 2018; Gidel et al., 2019; Berard et al., 2019). This is a generic issue for min-max optimization: a first-order algorithm may cycle around a stable point, converge very slowly or even diverge. This convergence issue can be alleviated by some advanced optimization algorithms such as optimism (Daskalakis et al., 2018), averaging (Yazıcı et al., 2019) and extrapolation (Gidel et al., 2018).

Third: mode collapse. Mode collapse is long known to be a challenge for GANs (Goodfellow et al., 2014). It means that some "modes" of the true data distribution are not generated, thus the diversity

Method	Range	Modification	Goal	Quantity	
Orthogonal-ini	Initial	–	orthogonal weights	weight	
BatchNorm (BN)	All	built-in layer	bounded norm	postactivation	
WeightNorm (WN)	All	built-in layer	bounded Frobenious norm	weight	
Weight clipping	All	projection	bounded weight norm	weight	
Gradient penalty (GP)	All	regularization	bounded grad norm	gradient	
Orthogonal-reg	All	Regularization	orthogonal weights	weight	
SpectralNorm (SN)	All	Built-in layer	bounded spectral norm	weight	
SVD	All	Regularize	control spectrum	weight	–
PC	**All**	**Built-in layer**	**Control spectrum**	weight	–

Table 3: Comparison of Orthogonal initialization (Xiao et al., 2018), batch normalization (Ioffe & Szegedy, 2015), weight normalization (Salimans & Kingma, 2016), weight clipping (Arjovsky & Bottou, 2017), gradient penalty (Gulrajani et al., 2017), orthogonal normalization (Brock et al., 2016), spectral normalization (Miyato et al., 2018), SVD (Jiang et al., 2019), and our PC layer. The first three methods are originally designed for general-purpose use; the other methods are commonly used in GANs. Note that all methods can be used in both supervised learning and GANs. Their practical performance on common applications may differ. Some notes: BN is originally designed for postactivation (output of each activation layer), but it can also be used for preactivation (input of each activation layer). SVD is actually a mixture of a regularization and a built-in layer: for USV^T SVD regularizes U, V, and either regularizes S or uses a built-in layer to adjust S. For simplicity, we call it "regularize."

of the generated images is not high. The cause of mode collapse is not well understood and there are a few hypotheses, such as improper loss functions (Arjovsky & Bottou, 2017; Arora et al., 2017), weak discriminators (Metz et al., 2017; Salimans et al., 2016; Arora et al., 2017; Li et al., 2018), and inherent property of the loss function (Lin et al., 2018). A few empirical solutions have been proposed, including unrolled GAN (Metz et al., 2017), minibatch discrimination (Salimans et al., 2016) and PacGAN (Lin et al., 2018).

Other GAN Variants. Many different GANs have been proposed over the last few years. These include WGAN (Arjovsky et al., 2017; Arjovsky & Bottou, 2017; Gulrajani et al., 2017), other variants of WGAN (Wu et al., 2019; Kolouri et al., 2018; Adler & Lunz, 2018; Deshpande et al., 2018), least-squares GAN (Mao et al., 2017), f-GAN (Nowozin et al., 2016), and many more (Mroueh & Sercu, 2017; Berthelot et al., 2017; Mroueh et al., 2017; Cully et al., 2017; Li et al., 2017b;a; Salimans et al., 2016; Nowozin et al., 2016; Poole et al., 2016; Metz et al., 2017; Radford et al., 2016; Bengio & LeCun, 2007). As mentioned in the main text, our approach falls into the category of "normalization and regularization." It is hence relatively orthogonal to these approaches.

B Preconditioning: Introduction

Consider a linear system of equations

$$Qw = b,$$

where $Q \in \mathbb{R}^{n \times n}$ is real symmetric, and $b \in \mathbb{R}^{n \times 1}$. Conjugate gradient (CG) is one of the most popular methods to solve the system of equations. It has iteration complexity $O(\sqrt{\kappa(Q)} \log 1/\epsilon)$, where $\kappa(Q)$ is the condition number of Q. For ill-conditioned problems (i.e., large $\kappa(Q)$), the convergence can be slow. Thus, in practice, preconditioned CG is commonly used instead of the original CG.

Suppose there is a certain way to find a preconditioner M that reduces the condition number, i.e., $\kappa(MQ) < \kappa(Q)$. Define $\tilde{Q} = MQ$ and $\tilde{b} = Mb$, then we can solve an alternative problem

$$\tilde{Q}w = \tilde{b},$$

for which CG (and other gradient methods) converges faster. One simple example is Jacobi preconditioning (closely related to whitening in machine learning) where M is a diagonal matrix with $M_{ii} = 1/\sqrt{Q_{i,i}}$.

B.1 POLYNOMIAL PRECONDITIONER

We review the polynomial preconditioners proposed by Johnson et al. (1983). However, we do not directly utilize the polynomials proposed by Johnson et al. (1983) since our setting differs. But we do borrow two lessons, which we will explain at the end of this subsection.

Consider a linear system of equations

$$Qw = b,$$

where $Q \in \mathbb{R}^{n\times n}$ is real symmetric, and $b \in \mathbb{R}^{n\times 1}$. To find a polynomial preconditioner $p(Q)$ such that $g(Q) = p(Q)Q$ is well-conditioned, we only need to find a polynomial p such that $g(x) = p(x)x$ maps $[\lambda_1, \lambda_m]$ to $[1-\epsilon, 1]$. This can be formulated as an approximation theory problem: find a polynomial $g(x)$ that approximates a function f where $f(0) = 0, f([\lambda_1, \lambda_m]) = 1$.

Define

$$P_k = \{p(x) \mid p(x) = \sum_{j=0}^{k} c_j x^j\}, \quad \hat{P}_k = \{g(x) \mid g(x) = \sum_{j=1}^{k} c_j x^j\},$$
$$\hat{P}_k^+ = \{g(x) \in \hat{P}_k \mid g(x) > 0, \forall x \in [\gamma_{\mathrm{L}}, \gamma_{\mathrm{U}}]\}.$$

Here, P_k is the set of all polynomials with degree no more than k, $\hat{P}_k$ contains all elements of P_k that vanish at 0, and $\hat{P}_k^+$ contains all elements of $\hat{P}_k$ that are positive in $[\gamma_{\mathrm{L}}, \gamma_{\mathrm{U}}]$.

Johnson et al. (1983) consider two polynomial preconditioners: minimax and least-squares polynomials. Minimax polynomials are the solution to the following problem:

$$\min_{p\in P_k} \max_{\lambda\in[\gamma_{\mathrm{L}},\gamma_{\mathrm{U}}]} |1 - \lambda p(\lambda)|. \tag{6}$$

Note that there should be an extra constraint that $\lambda p(\lambda) \in \hat{P}_{k+1}$, but as we see shortly the solution automatically satisfies this constraint. Denote $q(\lambda) = \lambda p(\lambda)$, then the above problem can be rewritten into

$$\min_{q\in \hat{P}_{k+1}} \max_{\lambda\in[\gamma_{\mathrm{L}},\gamma_{\mathrm{U}}]} |1 - q(\lambda)|. \tag{7}$$

There is a closed-form solution to the above problem: $q^*(\lambda) = 1 - \frac{T_{k+1}(\mu(\lambda-\gamma_{\mathrm{L}}))}{T_{k+1}(\mu(\lambda-\gamma_{\mathrm{L}}))}$, where $\mu(\lambda) = -1 + 2\frac{\lambda-\gamma_{\mathrm{L}}}{\gamma_{\mathrm{U}}-\gamma_{\mathrm{L}}}$ maps $[\gamma_{\mathrm{L}}, \gamma_{\mathrm{U}}]$ to $[-1, 1]$, and $T_k(x)$ is the Chebyshev polynomial of the first kind satisfying $T_k(\cos(z)) = \cos(kz)$. The first four Chebyshev polynomials are $T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, T_3(x) = 4x^3 - 3x$. This polynomial q^* happens to lie in $\hat{P}_k^+$, thus it is the desired solution.

Johnson et al. (1983) also consider least-squares polynomials, which are the solutions to the problem

$$\min_{q\in \hat{P}_{k+1}} \int_{\gamma_{\mathrm{L}},\gamma_{\mathrm{U}}} |1 - q(\lambda)| w(\lambda) d\lambda. \tag{8}$$

There is a closed-form solution to the above problem since it is a quadratic problem in the coefficients of q. A major theoretical challenge for Johnson et al. (1983) is to find whether the solution is in the set $\hat{P}_{k+1}^+$, i.e., whether it stays positive in $[\gamma_{\mathrm{L}}, \gamma_{\mathrm{U}}]$. Johnson et al. (1983) provided a sufficient condition for the optimal solution to be in $\hat{P}_{k+1}^+$. In particular, for the Jacobian weight function $w(\lambda) = (\gamma_{\mathrm{U}} - \lambda)^\alpha (\lambda - \gamma_{\mathrm{L}})^\beta$ where $\alpha \geq \beta \geq -1/2$, the optimal solution is in $\hat{P}_{k+1}^+$.

As we will explain in the next subsection, we cannot directly borrow the polynomials used by Johnson et al. (1983). Nevertheless, we borrow two lessons for our design. First, the polynomial preconditioner can be designed by solving an optimization problem (either minimax or least squares). Second, they found that least-squares polynomials perform better than minimax polynomials for iterative algorithms. For this reason, we adopt the least-squares polynomials instead of the minimax polynomials.

B.2 PRECONDITIONING A RECTANGULAR MATRIX: LINEAR REGRESSION

Our problem: how to precondition a rectangular matrix. This differs from the problem considered by Johnson et al. (1983). In this subsection, we explain why we cannot directly apply the polynomials designed by Johnson et al. (1983).

Preconditioning rectangular matrices is often not explicitly discussed in polynomial preconditioning literature. This is partially because for linear models, it is easy to transform a rectangular matrix to a square symmetric matrix. We will discuss this point below.

Consider a linear regression problem

$$\min_{w\in\mathbb{R}^d} F(w) = \|Aw - y\|^2, \tag{9}$$

where $A \in \mathbb{R}^{n\times d}$, $y \in \mathbb{R}^{n\times 1}$. Suppose we want to apply preconditioning to speed up a gradient-based method. There are two different methods: (i) preconditioning A^TA; (ii) preconditioning A.

Method 1: Precondition A^TA. We write the optimality condition of the problem is

$$A^TAw = A^Ty. \tag{10}$$

Thus solving the original problem (Eq. (9)) is equivalent to solving (Eq. (10)). We design a polynomial preconditioner $p(Q)$ for the matrix $Q = A^TA \in \mathbb{R}^{d\times d}$, and obtain a new linear system $p(A^TA)A^TAw = p(A^TA)A^Ty$. The preconditioned GD proceeds as follows: $w_{\text{new}} = w - \alpha p(A^TA)[A^TAw - A^Ty]$.

Note that in practice, we rarely form the matrix A^TA directly since it is time consuming. Instead, we multiply each matrix with a vector sequentially. For instance, when $p(A^TA) = A^TA$, we implement $A^TA[A^TAw - A^Ty]$ as $A^T(A(A^T(Aw - y)))$, where each bracket is a matrix-vector product. From a neural-net perspective, the computation of the gradient consists of: a forward pass that computes $e = Aw - y$; an extra preconditioner $\hat{e} = A(A^T(e))$; a backward pass $A^T\hat{e}$.

Method 2: Precondition A. Now we consider the second choice: precondition A directly. We use a preconditioner $M \in \mathbb{R}^{n\times n}$ and solve a new problem

$$\min_{w\in\mathbb{R}^d} F(w) = \|MAw - My\|^2. \tag{11}$$

The optimality condition of the new problem is

$$A^TM^TMAw = A^TM^TMy. \tag{12}$$

Although Eq. (12) seems different from Eq. (10), it is easy to show that they are equivalent when $M = p(AA^T)$ is a polynomial function of AA^T. In fact, in this case Eq. (12) is equivalent to

$$M^TMA^TAw = M^TMA^Ty.$$

Therefore, applying preconditioner M to A in the original problem is equivalent to applying preconditioner M^TM to A^TA in the optimality condition.

The preconditioned GD proceeds as follows: $w_{\text{new}} = w - \alpha(A^TM^TMAw - A^TM^TMy)$. Again, the real implementation consists of a series of matrix-vector products. It can be decomposed into two steps: a forward pass of computing $\hat{e} = M(Aw - y)$, and a backward pass of computing $A^TM^T\hat{e}$.

Comparison of both methods. For linear regression problems, preconditioning A^TA is slightly better than preconditioning A since the former covers the latter as a special case. More specifically, suppose we use p_1, p_2 to denote the precoditioning polynomial for method 1 and 2 respectively. For method 1, the preconditioned matrix is $Z_1 = p_1(A^TA)A^TA$ while for method 2, the preconditioned matrix is $Z_2 = p_2(A^TA)^2A^TA$. Note that $\{p_2(A^TA)^2A^TA \mid \deg(p_2) \le k\}$ is a proper subset of $\{p_1(A^TA)A^TA \mid \deg(p_1) \le 2k\}$. This means that method 1 can cover method 2 but not the other way. For this reason, there is little motivation for most linear system researchers to study direct preconditioning of A.

Nevertheless, a major advantage of directly preconditioning A is that it can be applied to general parameterized models such as neural nets. The method of preconditioning A^TA relies on the special property of linear regression that A^TA appears in the optimality condition, which does not hold in more general problems.

C Details of Algorithms

C.1 Polynomial fitting algorithm

The whole procedure of identifying polynomial g is summarized in Alg. 1.

Algorithm 1 POLY-FITTING

1: **Input**: a target function $f(x) : \mathbb{R} \to \mathbb{R}$. Default choice: $f(x) = \text{PL}_b(x)$.
2: **Hyper-parameters**: segment $[\gamma_{\text{L}}, \gamma_{\text{U}}]$, degree k, α appearing in $w(\lambda) = \lambda^\alpha$.
3: Solve the convex problem (5) to obtain c^*.
4: **Output**: a polynomial $g(\lambda) = \sum_{t=0}^{k+1} c_i^* \lambda^t$.

Algorithm 2 Fixed Preconditioning (FPC) for Neural Net

1: **Input**: a deep net $D(\theta)$, with $\theta = (W_1, \ldots, W_L)$ the collection of trainable weights. For GAN training, we use the discriminator net D.
2: **Dependent algorithm**: polynomial generating method POLY-FITTING; spectral-norm estimator $\text{SN}(W)$.
3: **Step 1**: generate a polynomial g by POLY-FITTING. This g is a $(2k+1)$-th order polynomial of the form $g(x) = xh(x^2)$, where h is k-th order polynomial.
4: **Step 2**: Build a preconditioned neural net $D_{PC}(\theta) = D(g(\text{SN}(W_1)), \ldots, g(\text{SN}(W_N)))$.
5: **Output**: $D_{PC}(\theta)$.

C.2 Detailed Description of FPC and APC

As stated in the main paper, $g(x)$ are fitted to different piecewise linear functions (Fig. 3) via Alg. 1. Here we provide the formulations of $g_0(x), g_1(x), g_2(x), g_3(x), g_4(x)$ corresponding to the cutoff $b = 1, 0.8, 0.6, 0.4, 0.3$.

$$\begin{aligned} g_0(x) &= x, \\ g_1(x) &= 1.507x - 0.507x^3, \\ g_2(x) &= 2.083x - 1.643x^3 + 0.560x^5, \\ g_3(x) &= 2.909x - 4.649x^3 + 4.023x^5 - 1.283x^7, \\ g_4(x) &= 3.625x - 9.261x^3 + 14.097x^5 - 10.351x^7 + 2.890x^9 \end{aligned} \tag{13}$$

For $g_0(x) = x$, we recover SN; thus SN is a special case of PC. We can even allow continuous transformation from SN to PC: using $g(x; \alpha) = (1+\alpha)x - \alpha x^3$ in PC layer, then for $\alpha = 0$ we have SN and for $\alpha \neq 0$ we have other PC. In our implementation of APC, we do not use continuous change of g, though this can be a potential future research direction.

The detailed procedure of FPC is summarized in Algorithm 2; it just describes how we obtain a new neural net with a fixed PC layer added to each layer. The detailed procedure of APC is summarized in Algorithm 3; it describes how to implement APC during training.

In our implementation of APC, the values of $\tau_0, \tau_1, \tau_2, \tau_3, \tau_4$ are 0, 5, 10, 20, 30 respectively. That is to say, if the condition number $\tilde{\kappa}(W) \leq 5$, no preconditioner will be applied; if $5 < \tilde{\kappa}(W) \leq 10$, g_1 will be applied on the weight W; if $10 < \tilde{\kappa}(W) \leq 20$, g_2 will be applied on the weight W; if $20 < \tilde{\kappa}(W) \leq 30$, g_3 will be applied on the weight W; if $\tilde{\kappa}(W) > 30$, g_4 will be applied on the weight W. We do not heavily tune these values τ_i's; it seems that the performance of APC is relatively robust to different values. In our implementation, we use the average of the condition numbers of the last 5 iterations (to replace Line 9); for simplicity, we do not add this small modification into the table. This may improve the performance a bit, though we did not perform a through ablation study.

C.3 Implementation of PC layer

We discuss a few computational tricks for implementing the PC layer. These tricks can greatly reduce the computation time.

Suppose we decided to use a polynomial $g(x) = xp(x^2)$ to implement a PC-layer.

For a convolutional layer, W is a tensor, and we will follow Miyato et al. (2018) to reshape the tensor into a matrix $\text{flat}(W)$, and apply an SN layer to obtain $A = \text{SN}(\text{flat}(W))$. For the fully connected layer, W is a matrix, and we can define $A = \text{SN}(W)$.

Algorithm 3 Adaptive Preconditioned (APC) Neural Net

Preparation: A neural net $D(\theta)$, where $\theta = (W_1, \dots, W_L)$ is the collection of all weights. An algorithm $\mathcal{A}$ that updates θ and other parameters (e.g. generator parameter), such as SGD and Adam.
Dependent functions: spectral norm estimator $\text{SN}(W)$; preconditioners $g_1, g_2, \dots, g_m$ with increasing preconditioning power, i.e., $g_1(x) \le g_2(x) \le \dots g_m(x), \forall x \in [0, 1]$.
Hyper-parameters: $T_{\text{adj}} \in \mathbb{Z}_+$, $\tau_1 < \dots < \tau_m \in \mathbb{R}_+$.
for $t = 1, 2, \dots$ **do**
  Set the function $g^l(W) = \text{SN}(W)$;
  if $\mod (t, T_{\text{adj}}) == 0$ **then**
    for $l = 1, 2, \dots, L$ **do**
      Compute the condition number $\tilde{\kappa}(W_l)$.
      If $\tilde{\kappa}(W_l) \in [\tau_j, \tau_{j+1}]$, set the function $g^l(W) = g_j(\text{SN}(W))$.
    end for
  end if
  Update parameter θ in the precondtioned neural net $D(g^1(\text{SN}(W_1)), \dots, g^L(\text{SN}(W_L))$, and other parameters by one iteration of the algorithm $\mathcal{A}$.
end for

The first trick is to choose one of the two equivalent expressions depending on whether A is wide or tall. More specifically, note that $g(A) = p(AA^T)A = Ap(A^TA)$. We call $p(AA^T)A$ the AA^T-form and we call $Ap(A^TA)$ the A^TA-form. Although the two forms give the same value, their implementation time can vary a lot: suppose $A \in \mathbb{R}^{n\times m}$ where $n \ge m$, then evaluating A^TA takes $O(m^2n)$ operations while evaluating AA^T takes $O(nm^2)$ operations. In an extreme case where $A \in \mathbb{R}^{1000\times 1}$, A^TA takes 2000 multiplications and 999 additions, while AA^T takes 1000^2 multiplications. Therefore, if A is tall ($n \ge m$), we pick the A^TA-form; otherwise we pick the AA^T-form. We checked a 100×1000 random matrix in PyTorch, and found that AA^T takes time 0.0018 sec while A^TA takes time 0.0078, a 4-times gap.

The second trick, which is quite straightforward, is to store the product $B = A^TA$ or $B = AA^T$, and re-use the product. We will examine the implementation of $p(B) = p(A^TA)$ next.

The third trick is to use Horner's method to evaluate the polynomial. The basic idea of Horner's method is that a polynomial can be decomposed into the following form:

$$\sum_{i=0}^{k} a_i x^i = a_0 + x(a_1 + x(a_2 + \dots x(a_{k-1} + a_k x)\dots)).$$

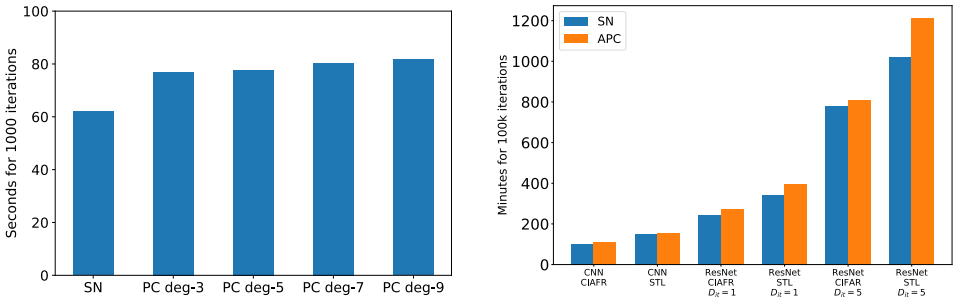
For instance, $a_0 + a_1x + a_2x^2 + a_3x^3 = a_0 + x(a_1 + x(a_2 + a_3x)))$. Suppose $p(x) = \sum_{i=0}^k c_i x^i$. A naïve implementation of

$$p(B) = \sum_{i=0}^{k} c_i B^i$$

requires $\sum_{i=1}^k (i-1) = \frac{k(k-1)}{2}$ matrix multiplications and k additions. Using Horner's method, we only need k matrix multiplications. For a degree 4 polynomial p (corresponding to degree-9 polynomial $g(x) = xp(x^2)$), Horner's method can reduce 6 matrix multiplications to 4 matrix multiplications. It will save more time for higher degree polynomials; e.g., for a degree-15 polynomial ($k = 7$), Horner's method can reduce 21 matrix multiplications to 7 matrix multiplications. Therefore, we have strong power to mimic most target functions with relatively small cost of computation (see C.4 for discussion of computation time). We have not explored too many target functions, and future users of PC-layers can design their own target functions and use higher-degree polynomials.

We summarize the three tricks for computing $p(A) = p(AA^T)A$ as follows:

- Use A^TA-form $Ap(A^TA)$ if A is tall, and the AA^T-form $p(AA^T)A$ if A is wide.
- Store $B = AA^T$ or $B = A^TA$, and compute $p(B)A$ or $Ap(B)$, respectively.
- Use Horner's method to evaluate $p(B)$.



(a) (b)

Figure 4: (a) Computational time (in seconds) for 1000 training iterations of the CIFAR-10 image generator (CNN). (b) Computational time (in minutes) for SN-GAN and APC-GAN for $100k$ iterations and for different experiments.

Example: Suppose we have determined a polynomial $g(x) = c_0x + c_1x^3 + c_2x^5 + c_3x^7$, and want to apply it to a *wide* matrix A. There are two steps: (i) Compute $B = AA^T$; (ii) Compute

$$(c_0 + (c_1 + B(c_2 + B(c_3 + B))))A.$$

If we apply it to the *tall* matrix A, the two steps are: (i) Compute $B = A^TA$; (ii) Compute

$$A(c_0 + (c_1 + B(c_2 + B(c_3 + B)))).$$

C.4 Computation time

We discuss the computation time of POLY-FITTING, FPC and APC. POLY-FITTING solves a convex quadratic problem with dimension being the number of samples (e.g., 300) in $[\gamma_{\mathrm{L}}, \gamma_{\mathrm{U}}] = [0, 1.1]$. This takes little time to run and is done offline.

The extra computation time due to a PC-layer depends on the degree $2k + 1$ of the preconditioning polynomial g. We use a few implementation tricks, including Horner's method for implementing the polynomial evaluation, which only requires $k + 1$ matrix multiplications. See Appendix C.3 for details. In our implementation of FPC with a degree $3, 5, 7$ or 9 polynomial, the actual added time is around $20 - 30\%$ (Fig. 4 (a)) of the original training time of SN-GAN. Interestingly, the extra time for degrees $3, 5, 7, 9$ polynomials are quite close (less than 5% difference).

The computation time of APC is smaller than FPC. Since most layers during practical training are well-conditioned, APC essentially applies nothing to most layers, and only applies PC to few layers. Compared to FPC, APC requires an extra step of computing the condition number of each layer. We only perform this extra step every 1000 iterations, thus the amortized extra cost over all iterations is small (contributing less than 1% to the total time). Fig. 4 (b) shows that the extra time of APC over SN is often less than 10%.

D Global Convergence: Proofs and Other Results

In this section, we will provide a proof of Theorem 1, as well the discussions of the theorem and a few related technical results. We will first review the notion of NTK (neural tangent kernel) and a basic convergence result. Then we prove the global convergence of continuous gradient dynamics under assumptions on minimal singular values of weights. Finally, we prove the global convergence of discrete gradient descent, i.e., Theorem 1.

D.1 Basics: Neural Tangent Kernel and Convergence Lemma

Neural tangent kernel (NTK) is introducd in Jacot et al. (2018) (see also Du et al. (2018)). We briefly review some basics of NTK. The gradient is $\frac{\partial F(\theta; x_i)}{\partial \theta} \in \mathbb{R}^{P \times d_y}$, where P is the number of parameters. Define the Jacobian

$$G(\theta) = \left(\frac{\partial f(\theta; x_1)}{\partial \theta}, \ldots, \frac{\partial f(\theta; x_n)}{\partial \theta} \right) \in \mathbb{R}^{P \times d_y n} \tag{14}$$

and define the *neural tangent kernel* (NTK)

$$K(\theta) = G(\theta)^T G(\theta). \tag{15}$$

Lemma 1 *Define* $\mathcal{W}_{\rm NTK}(\kappa_0) = \{\theta : \lambda_{\min}(K(\theta)) \geq \kappa_0\}$. *Suppose the optimization trajectory* $\theta(t)$ *stays in* $\mathcal{W}_{\rm NTK}(\kappa_0)$ *for* $1 \leq t \leq T$, *then*

$$\|F(\theta(t)) - y\| \leq \|F(\theta(0)) - y\| \exp(-2t\kappa_0), \quad \forall\, 0 \leq t \leq T.$$

Proof of Lemma 1:

Let $e(t) = F(\theta(t)) - y$ and $K(t) = K(\theta(t))$. We have $\frac{de(t)}{dt} = -G(\theta(t))^T G(\theta(t))e(t) = -K(t)e(t)$. Therefore,

$$\frac{d\|e(t)\|^2}{dt} = -2e(t)^T K(t) e(t) \leq -2\lambda_{\min}(K(t))\|e(t)\|^2 \leq -2\kappa_0 \|e(t)\|^2.$$

This implies

$$\begin{aligned}\frac{d(\exp(2t\kappa_0)\|e(t)\|^2)}{dt} &= 2\kappa_0 \exp(2t\kappa_0)\|e(t)\|^2 + \exp(2t\kappa_0)\frac{d\|e(t)\|^2}{dt} \\ &\leq 2\kappa_0 \exp(2t\kappa_0)\|e(t)\|^2 - 2\kappa_0 \|e(t)\|^2 \exp(2t\kappa_0) = 0.\end{aligned}$$

Therefore $\exp(2t\kappa_0)\|e(t)\|^2 \leq \|e(0)\|^2$, i.e.,

$$\|F(\theta(t)) - y\| \leq \|F(\theta(0)) - y\| \exp(-2t\kappa_0).$$

This proofs the lemma. □

A common trick is to show that for ultra-wide neural-nets, all iterates stay in the set $\mathcal{W}_{\rm NTK}$; see, e.g., work by Du et al. (2018); Arora et al. (2019); Lee et al. (2019). However, this requires a very large width, in order to ensure that the weights move little (i.e., the NTK stays positive definite). We assume the weights are well conditioned, thus we allow the weights to freely move away as long as they stay in the well-conditioned region. This permits to provide a simpler proof of global convergence. Note that we do not view this simplicity as a major contribution in theory, since we do need extra assumptions; instead, this simplicity is helpful for non-theory readers to understand the essence of the convergence analysis, and also helpful for them to understand the importance of weight spectrum.

D.2 Convergence of Continuous Dynamics

For linear networks, the expression of the Jacobian is

$$G(\theta) = \left(\frac{\partial f_\theta(x_i)}{\partial W_j}\right) = \begin{bmatrix} x_1 \otimes (W_L W_{L-1} \dots W_2) & \dots & x_n \otimes (W_L W_{L-1} \dots W_2) \\ \dots & \dots & \dots \\ W_{L-1} \dots W_1 x_1 \otimes I & \dots & W_{L-1} \dots W_1 x_n \otimes I \end{bmatrix}.$$

There are $P = d_0 d_1 + \dots + d_{L-1} d_L$ rows, and $n d_L = n d_y$ columns.

The assumption on the architecture is that it is a pyramid-like structure.

Assumption D.1 *(Pyramid net) There exists some* $r \in \{1, \dots, L\}$, *such that* $d_y = d_L \leq d_{L-1} \leq \dots \leq d_r$, *and* $d_x = d_0 \leq d_1 \leq \dots \leq d_r$.

Define

$$\mathcal{W}_{\rm low}(\mu_1, \dots, \mu_L) = \{\theta = (W_1, \dots, W_L) \mid \sigma_{\min}(W_l) \geq \mu_l,\ l = 1, 2, \dots L\}.$$

Given a deep net that satisfies these assumptions, the exponential convergence of the gradient flow defined via $\frac{d\theta(t)}{dt} = -\nabla \mathcal{L}(\theta(t))$ can be shown:

Theorem 2 *Consider a deep linear network that satisfies Assumption D.1. Assume* $n \leq d_x$ *and suppose* $\sigma_{\min}(X) \geq \mu_0 > 0$. *Suppose the optimization trajectory* $\theta(t)$ *of the gradient dynamics satisfy* $\theta(t) \in \mathcal{W}_{\rm low}(\mu_1, \dots, \mu_L), 0 \leq t \leq T$. *Define* $\bar{\lambda} = (\mu_0 \mu_1 \dots \mu_L)^2$, *then*

$$\|F(\theta(t)) - F(\theta^*)\| \leq \|F(\theta(0)) - F(\theta^*)\| \exp(-2t\bar{\lambda}), 0 \leq t \leq T.$$

Below, we provide the proof of the theorem. We present two technical results, and then provide the proof at the end of this subsection (one line proof).

Claim D.1 *Suppose $A_m, A_{m-1}, \ldots, A_1$ are matrices of size $n_m \times n_{m-1}, n_{m-1} \times n_{m-2}, \ldots, n_1 \times n_0$, where $n_m \geq n_{m-1} \geq n_0$. Suppose $\sigma_{\min}(A_i) \geq \mu_i, i = 1, \ldots, m$. Then the product $M = A_m A_{m-1} \ldots A_1$ satisfies $\sigma_{\min}(M) \geq \mu_1 \mu_2 \ldots \mu_m$.*

Proof of Claim D.1: We first prove the following result: for a $k_1 \times k_2$ matrix A and $k_2 \times k_3$ matrix B, where $k_1 \geq k_2 \geq k_3$,

$$\sigma_{\min}(AB) \geq \sigma_{\min}(A)\sigma_{\min}(B). \tag{16}$$

This is proved by the following chain of inequalities:

$$\lambda_{\min}(B^T A^T A B) = \min_{\|u\|=1} u^T B^T A^T A B u \geq \lambda_{\min}(A^T A)\|Bu\|^2 = \lambda_{\min}(A^T A) u^T B^T B u$$
$$\geq \lambda_{\min}(A^T A)\lambda_{\min}(B^T B) = \sigma_{\min}(A)^2 \sigma_{\min}(B)^2.$$

Applying the result multiple times, we immediately obtain the desired result. □

Lemma 2 *Consider a deep linear network that satisfies Assumption D.1. Assume $n \leq d_x$ and $\sigma_{\min}(X) \geq \mu_0$. If a point $\theta = (W_1, \ldots, W_L)$ satisfies $\sigma_{\min}(W_l) \geq \mu_l$ for each l, then $\lambda_{\min}(K(\theta)) \geq (\mu_0 \mu_1 \ldots \mu_L)^2$.*

Proof of Lemma 2: Define $W_{i:j} = W_i W_{i-1} \ldots W_j$. We write the l-th row as $G_l = (W_{l-1:1} X) \otimes (W_{L:l})^T$. which is a $(d_{l-1} d_l \times d_y n)$ matrix. Then we can write the matrix $G(\theta) = [G_1(\theta); G_2(\theta); \ldots; G_L(\theta)]$. According to Assumption D.1, and Claim D.1, we have $\sigma_{\min}(W_{r:1} X) \geq \mu_0 \mu_1 \ldots \mu_r$. and $\sigma_{\min}(W_{L:r+1}^T) \geq \mu_{r+1} \mu_1 \ldots \mu_L$. By Theorem 4.2.15 of Horn et al. (1994), $\sigma_{\min}(W_{r:1} X \otimes W_{L:r+1}^\top) \geq \mu_0 \mu_1 \ldots \mu_L$. Therefore, $K = \sum_{l=1}^L G_l^\top G_l \succeq G_{r+1}^\top G_{r+1} \succeq (\mu_0 \mu_1 \ldots \mu_L)^2 I_n$. □

Proof of Theorem 2: By Lemma 2 and Lemma 1, we immediately obtain the desired result. □

D.3 Convergence of Discrete Time GD: Theorem and Discussion

Motivating works for the result. The idea of bounding spectrum has been around for a long time, but rigorous theory only appears quite recently to our knowledge. Pennington et al. (2017); Xiao et al. (2018; 2020) suggested that the spectrum of the input-output Jacobian or neural tangent kernel (NTK) is important for training. Jacot et al. (2018); Du et al. (2018); Allen-Zhu et al. (2019); Zou et al. (2018) proved that if the NTK stays full rank, then gradient descent on wide neural networks converges to global minima. Directly manipulating the spectrum of NTK is hard, thus one may wonder whether manipulating the spectrum of weight matrices can ensure a full-rank NTK. For multi-layer wide linear neural nets, Hu et al. (2020) proved that starting from orthogonal weight matrices, the initial NTK is full-rank and stays full rank during training, thus converging to a global minimum.

Our Theorem 1 is strongly motivated by the aforementioned progress in neural net theory, and the proof framework somewhat resembles that of (Hu et al., 2020). The main differences with (Hu et al., 2020) are two-fold: first, they assume orthogonal initial point, and we assume arbitrary initial spectrum (as long as weight matrices are full-rank); second, they directly prove convergence to global-min, i.e., $K = \infty$, but we do not provide a bound on K. It is not hard to derive our proof based on (Hu et al., 2020). Our contribution is not on presenting new proof techniques; rather, our contribution mainly lies in proposing PC layer, and this theorem is just a direct motivation for our purpose.

Previous theoretical works on convergence analysis Hu et al. (2020); Jacot et al. (2018); Du et al. (2018); Allen-Zhu et al. (2019); Zou et al. (2018) are mainly interested in proving theoretical convergence and do not directly relate the weight spectrum during training to to convergence rate; note that of course the NTK spectrum is related to the convergence rate but these works do not present direct methods to manipulate NTK spectrum during training. (Xiao et al., 2020), in some sense, provides some "control" of the spectrum of the NTK by analyzing the condition number of the NTK for difference cases, then potentially they provided guidance on how to choose a setting with good

NTK spectrum. However, their result mainly applies to the initial NTK and requires large-width-assumption to show NTK spectrum moves little during training, thus is not directly applicable for controlling the spectrum during practical training.

In light of the above discussion, we choose to present Theorem 1 explicitly, to single out the importance of the weight matrix condition numbers.

Restatement of the theorem. Denote the initial point as $\theta(0)$. Consider gradient descent with constant stepsize

$$\theta(k+1) = \theta(k) - \eta \nabla \mathcal{L}(\theta(t)).$$

For any positive numbers $\tau_1, \ldots, \tau_L$, define the set

$$\mathcal{W}_{\text{up}}(\tau_1, \ldots, \tau_L) = \{\theta = (W_1, \ldots, W_L) \mid \sigma_{\max}(W_l) \leq \tau_l, \forall l\}.$$

We restate Theorem 1 using different notation.

Theorem 3 *(restatement of Theorem 1) Consider a deep linear network that satisfies Assumption D.1. Assume $n \leq d_x$ and suppose $\sigma_{\min}(X) \geq \mu_0 > 0$. Assume*

$$\{\theta(k)\}_{k=0}^K \in \mathcal{W}_{\text{up}}(\tau_1, \ldots, \tau_L) \cap \mathcal{W}_{\text{low}}(\mu_1, \ldots, \mu_L), k = 0, 1, \ldots, K.$$

Assume $\tau_l \geq 1$ and $\mu_l > 0$, $l = 1, \ldots, L$. Denote $e(k) = F(\theta(k); X) - y$. Suppose

$$\beta = L\|X\|_2 \tau_L \ldots \tau_1 \left(\|e(\theta(0))\| + \|X\|_F \tau_L \ldots \tau_1\right), \mu = (\mu_1 \ldots \mu_L)^2 \sigma_{\min}(X)^2.$$

Suppose the learning rate $\eta = \frac{1}{\beta}$. Then we have

$$\|e(k+1)\|^2 \leq (1 - \frac{\mu}{\beta})\|e(k)\|^2, \quad k = 1, \ldots, K. \tag{17}$$

Remark 1: Our result is really about a one-step reduction: if using stepsize $\eta_k = 1/\beta_k$ where $\beta_k = L\|X\|_2 \tau_L \ldots \tau_1 \left(\|e(\theta(k))\| + \|X\|_F \tau_L \ldots \tau_1\right)$, then Eq. (17) holds. For simplicity of presentation, we use a fixed stepsize $1/\beta_0$ where $\beta_0 = \beta$ defined in the theorem statement. As the algorithm converges, the error $\|e(\theta(k))\|$ diminishes to 0, thus the dominant term should be $L\|X\|_2\|X\|_F(\tau_L \ldots \tau_1)^2$. If $\sigma_{\min}(W_l(k))\| \geq \mu_l$ and $\kappa(W_l(k)) \leq \kappa_l$ for $k = 0, 1, \ldots,$ then the asymptotic convergence rate is $1 - \frac{\sigma_{\min}(X)^2(\mu_L \ldots \mu_1)^2}{L\|X\|_2\|X\|_F(\tau_L \ldots \tau_1)^2} \leq 1 - (L\sqrt{n}(\kappa_L \ldots \kappa_1)^2 \kappa(X)^2)^{-1}$. The total number of iterations to achieve error $\|e(T)\| \leq \epsilon$ is at most

$$L\sqrt{n}(\kappa_L \ldots \kappa_1)^2 \kappa(X)^2 \log \frac{1}{\epsilon}.$$

This number is proportional to the product of the squared condition numbers of all weight matrices. Thus reducing the condition numbers can reduce the number of iterations, and consequently increase the convergence speed.

Remark 2: The extra assumption $\tau_l \geq 1$ is added to simplify the original constant $\beta = \sqrt{L}\|X\|_2 \tau_L \ldots \tau_1 \sqrt{\sum_{i=1}^L \frac{1}{\tau_i^2}} \left(\|e(\theta(0))\| + \|X\|_F \tau_L \ldots \tau_1\right)$ to the current form.

Remark 3: We fix $\beta = L\|X\|_2 \tau_L \ldots \tau_1 \left(\|e(\theta(0))\| + \|X\|_F \tau_L \ldots \tau_1\right)$ and pick a fixed stepsize $\eta = 1/\beta$. This is just for simplicity, and we can strengthen the statement a bit like many optimimzation papers. First, we can pick any $\eta < 1/\beta$ and then prove Eq. (17) with a rate that depends on η, μ and β. Second, we can change $\beta = L\|X\|_2 \tau_L \ldots \tau_1 \left(\|e(\theta(0))\| + \|X\|_F \tau_L \ldots \tau_1\right)$ to $\beta \geq L\|X\|_2 \tau_L \ldots \tau_1 \left(\|e(\theta(0))\| + \|X\|_F \tau_L \ldots \tau_1\right)$ and then prove the same result.

Assumption of pyramid networks: Hu et al. (2020) analyze an equal-width linear network with $n \leq d_0$. We analyze a pyramid linear network, which covers the equal-width network as a special case. Our assumption $n \leq d_0$ is the same as that of Hu et al. (2020). This pyramid structure and the assumption $n \leq d_0$ are used in lower bounding the minimum singular value of the NTK matrix. The pyramid structure has been used in neural network theory studies (Nguyen & Hein, 2017). Note that the assumption $n \leq d_0$ is less desirable; anyhow, we follow Hu et al. (2020) and keep this assumption.

Assumption of bounded spectrum for K iterations: To understand the assumption and the essence of the theorem, one needs to understand that there is a certain order of picking parameters. We first

pick an initial point $\theta(0)$, then pick $\tau_1, \dots, \tau_L, \mu_1, \dots, \mu_L$, such that the spectrum of $W_l(0)$ lie in $[\mu_l, \tau_l]$. These τ_i's and μ_i's can be larger than the spectrum range of $W_l(0)$. We then pick β according to τ_i's, L, $\|X\|_2, \|X\|_F$ and the initial loss $e(0)$. Then we run GD with the stepsize $\eta = 1/\beta$ and generate a sequence $\theta(k)$. This sequence starts in the region $R = \mathcal{W}_{\text{up}}(\tau_1, \dots, \tau_L) \cap \mathcal{W}_{\text{low}}(\mu_1, \dots, \mu_L)$, and may stay in R forever, or leave R at some point K. Within the region R, the loss diminishes at a geometric rate. In case $K = \infty$, the algorithm converges to global minimum; even if K is not infinity but quite large, the algorithm can generate very small loss after K iterations. Note, however, that we do not know a priori when the iterates will leave R. This is why other works on neural networks try hard to bound the movement of the weights. We do not attempt to prove the iterates stay in R; instead, we use this as a motivation: if we could improve the spectrum of weights during training, then they may stay in R for longer time, and thus lead to smaller loss values.

Theory for networks with PC layers: Intuitively, adding PC layers can allow the weights to stay in the region R for a longer time, but directly proving this is not easy and requires extra work. One challenge is that the added PC layer has changed the structure of the neural net, thus the gradient has a different form. Note that analyzing gradient descent with normalization layers is typically harder, thus it is not surprising that the analysis of PC layers is also harder than the case without any extra normalization. We leave the theoretical analysis of PC-layers for future study.

D.3.1 Proof of Theorem 1

To prove this result, we need a few bounds.

Claim D.2 *(G spectral norm bound) Consider a deep linear network of any shape. Assume $\sigma_{\max}(X) \le \tau_0$. If a point $\theta = (W_1, \dots, W_L)$ satisfies $\sigma_{\max}(W_l) \le \tau_l$ for each l, then $\lambda_{\max}(K(\theta)) \le L(\tau_0\tau_1 \dots \tau_L)^2$, or equivalently,*

$$\|G(\theta)\|_2 \le \sqrt{L}\tau_0\tau_1 \dots \tau_L.$$

Claim D.3 *(F is Lipschitz continuous) Consider a deep linear network of any shape. If a point $\theta = (W_1, \dots, W_L)$ satisfies $\sigma_{\max}(W_l) \le \tau_l$ for each l, then*

$$\|F(\theta; X) - F(\hat{\theta}; X)\|_F \le \|\theta - \hat{\theta}\|\|X\|_F\tau_L \dots \tau_1\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}.$$

Claim D.4 *(G is Lipschitz continuous) Consider a deep linear network of any shape. Assume $\sigma_{\max}(X) \le \tau_0$. If a point $\theta = (W_1, \dots, W_L)$ satisfies $\sigma_{\max}(W_l) \le \tau_l$ for each l, then $\|G(\theta) - G(\hat{\theta})\|_2 \le \sqrt{L}\tau_L \dots \tau_1\tau_0 \left(\sum_{i=1}^{L}\frac{1}{\tau_i^2}\right)\|\theta - \hat{\theta}\|$.*

Lemma 3 *Consider a deep linear network $F(\theta; x) = W_L \dots W_1 x$. Then*

$$\|\nabla\mathcal{L}(\theta; X) - \nabla\mathcal{L}(\hat{\theta}; X)\| \le \beta(\theta)\|\theta - \hat{\theta}\|, \quad \forall\theta, \hat{\theta} \in \mathcal{W}_{\text{up}}(\tau_1, \dots, \tau_L), \tag{18}$$

where

$$\beta(\theta) = \sqrt{L}\|X\|_2\tau_L \dots \tau_1\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}\left(\|e(\theta)\| + \tau_L \dots \tau_1\|X\|_F\right)}.$$

Here $e(\theta) = vec(F(\theta; X) - Y)$ is the estimation error and we consider θ as a vectorized version of all parameters, thus $\nabla\mathcal{L}(\theta; X)$ is a vector.

Proof of Theorem 1:

The proof is a simple application of the standard convergence proof of gradient descent. Denote $\phi_k = \|e(k)\|^2$.

We prove by induction. Assume the result holds for $0, 1, \dots, k-1$. Then we have $\mathcal{F}(\theta(k)) \le \mathcal{F}(\theta(0))$, i.e., $e(\theta(k)) \le e(0)$. This implies

$$\beta(\theta(k)) \le \beta. \tag{19}$$

For simplicity, let us denote $g(\theta) = \nabla\mathcal{L}(\theta)$, and $g_k = \nabla\mathcal{L}(\theta(k))$.

Denote $\delta_k = -\eta g_k = w(k+1) - w(k)$. We obtain

$$\begin{aligned}\mathcal{L}(w(k+1)) &= \mathcal{L}(w(k)) + g_k^T\delta_k + \int_0^1 [g(\theta(k)+t\delta_k) - g(\theta(k))]^T\delta_k dt \\ &\leq \mathcal{L}(w(k)) + g_k^T\delta_k + \int_0^1 \|g(\theta(k)+t\delta_k) - g(\theta(k))\|\|\delta_k\| dt.\end{aligned}$$

According to Lemma 3 and Eq. 19, we have $\|g(\theta(k)+t\delta_k) - g(\theta(k))\| \leq \beta(\theta(k))\|t\delta_k\| \leq \beta\|t\delta_k\|$. Plugging into the previous inequality, we get

$$\begin{aligned}\mathcal{L}(w(k+1)) &\leq \mathcal{L}(w(k)) + g_k^T\delta_k + \int_0^1 \beta\|t\delta_k\|\|\delta_k\| dt \\ &= \mathcal{L}(w(k)) - \eta\|g_k\|^2 + \frac{1}{2}\beta\eta^2\|g_k\|^2 \\ &= \mathcal{L}(w(k)) - \frac{1}{2\beta}\|g_k\|^2.\end{aligned}$$

Recall that $\|g(k)\|^2 = \|G(\theta(k))e(\theta(k))\|^2 \geq \lambda_{\min}(G(\theta(k))^T G(\theta(k)))\|e(\theta(k))\|^2 \geq \mu\|e(\theta(k))\|^2 = \phi_k$. Combining the two relations above, and by replacing $\mathcal{L}(w(k)), \mathcal{L}(w(k+1))$ with $2\phi_k, 2\phi_{k+1}$, we get

$$\phi_{k+1} \leq \phi_k - \frac{\mu}{\beta}\phi_k = (1 - \frac{\mu}{\beta})\phi_k.$$

This completes the proof. □

D.3.2 Proof of Claims on Lipschitz Continuity

Proof of Claim D.2:

We have $\|W_{l-1:1}X\| \leq \|W_{l-1}\|\|W_1\|\|X\| \leq \tau_0\tau_1\dots\tau_{l-1}$, and $\|W_{L:l}\| \leq \|W_L\|\dots\|W_l\| \leq \tau_L\tau_{L-1}\dots\tau_l$, thus $\lambda_{\max}(K_l(\theta)) = \lambda_{\max}([\underbrace{(W_{l-1:1}X)^T}_{n\times d_{l-1}}\underbrace{(W_{l-1:1}X)}_{d_{l-1}\times n}])\lambda_{\max}([\underbrace{W_{L:l}}_{d_y\times d_l}\underbrace{W_{L:l}^T}_{d_l\times d_y}]) \leq (\tau_L\tau_{L-1}\dots\tau_l)^2(\tau_0\tau_1\dots\tau_{l-1})^2 = (\tau_0\tau_1\dots\tau_L)^2$. Since this holds for each l, we have $\lambda_{\max}(K(\theta)) \leq L(\tau_0\tau_1\dots\tau_L)^2$. □

Proof of Claim D.3:

$$\begin{aligned}\|F(\theta;X) - F(\hat{\theta};X)\|_F &= \|W_L\dots W_1X - \hat{W}^L\dots\hat{W}^1X\|_F \\ &\leq \|W_L\dots W_1 - \hat{W}^L\dots\hat{W}^1\|_2\|X\|_F \\ &= \|\sum_{l=1}^L \hat{W}^L\dots\hat{W}^{l+1}(W_l - \hat{W}^l)W_{l-1}\dots W_1\|_2\|X\|_F \\ &\leq \tau_L\dots\tau_1\sum_{i=1}^L\frac{1}{\tau_i}\|W_i - \hat{W}_i\|_2\|X\|_F \\ &\leq \tau_L\dots\tau_1\|X\|_F\sqrt{\sum_{i=1}^L\frac{1}{\tau_i^2}}\sqrt{\sum_{i=1}^L\|W_i - \hat{W}_i\|_2^2}.\end{aligned}$$

We can relax $\sqrt{\sum_{i=1}^L\|W_i - \hat{W}_i\|_2^2} \leq \sqrt{\sum_{i=1}^L\|W_i - \hat{W}_i\|_F^2} = \|\theta - \hat{\theta}\|$, thus obtaining the desired result. □

Proof of Claim D.4: Recall that $G(\theta) = [G_1(\theta); G_2(\theta); \dots; G_L(\theta)]$, in which $G_l(\theta) = (W_{l-1:1}X) \otimes (W_{L:l})^T$. Similarly, denote $G_l(\hat{\theta}) = (\hat{W}_{l-1:1}X) \otimes (\hat{W}_{L:l})^T$. We bound $\|G_l(\theta) -$

$G_l(\hat{\theta})\|_2$ as follows:

$$\begin{aligned}
&\|W_{l-1}\dots W_1X - \hat{W}_{l-1}\dots\hat{W}_1X\| \\
=&\|\sum_{i=1}^{l-1}\hat{W}_{l-1}\dots\hat{W}_{i+1}(W_i-\hat{W}_i)W_{i-1}\dots W_1X\| \\
\le&\sum_{i=1}^{l-1}\|\hat{W}_{l-1}\|\dots\|\hat{W}_{i+1}\|\|W_i-\hat{W}_i\|\|W_{i-1}\|\dots\|W_1\|\|X\| \\
\le&\sum_{i=1}^{l-1}\tau_{l-1}\dots\tau_1\tau_0\frac{1}{\tau_i}\|W_i-\hat{W}_i\|
\end{aligned}$$

Combining with $\|W_{L:l}\|_2 \le \Pi_{j=l}^{L}\|W_j\|_2 \le \Pi_{j=l}^{L}\tau_j$, we get

$$\|(W_{l-1:1}X - \hat{W}_{l-1:1}X)\|_2\|W_{L:l}\|_2 \le \tau_L\dots\tau_1\tau_0\sum_{i=1}^{l-1}\frac{1}{\tau_i}\|W_i-\hat{W}_i\|. \tag{20}$$

Similarly,

$$\|\hat{W}_{l-1:1}X\|_2\|W_{L:l} - \hat{W}_{L:l}\|_2 \le \tau_L\dots\tau_1\tau_0\sum_{i=l}^{L}\frac{1}{\tau_i}\|W_i-\hat{W}_i\|. \tag{21}$$

$$\begin{aligned}
\|G_l(\theta)-G_l(\hat{\theta})\|_2 &= \|(W_{l-1:1}X)\otimes(W_{L:l})^T - (\hat{W}_{l-1:1}X)\otimes(\hat{W}_{L:l})^T\|_2 \\
&= \|(W_{l-1:1}X - \hat{W}_{l-1:1}X)\otimes(W_{L:l})^T + (\hat{W}_{l-1:1}X)\otimes(W_{L:l}-\hat{W}_{L:l})^T\|_2 \\
&\le \|(W_{l-1:1}X - \hat{W}_{l-1:1}X)\otimes(W_{L:l})^T\|_2 + \|(\hat{W}_{l-1:1}X)\otimes(W_{L:l}-\hat{W}_{L:l})^T\|_2 \\
&= \|(W_{l-1:1}X - \hat{W}_{l-1:1}X)\|_2\|W_{L:l}\|_2 + \|\hat{W}_{l-1:1}X\|_2\|W_{L:l}-\hat{W}_{L:l}\|_2 \\
&\le \tau_L\dots\tau_1\tau_0\sum_{i=1}^{L}\frac{1}{\tau_i}\|W_i-\hat{W}_i\|_2 \\
&\le \tau_L\dots\tau_1\tau_0\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}\sqrt{\sum_{i=1}^{L}\|W_i-\hat{W}_i\|_2^2} \\
&\le \tau_L\dots\tau_1\tau_0\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}\|\theta-\hat{\theta}\|.
\end{aligned}$$

In the last inequality we used $\sqrt{\sum_{i=1}^{L}\|W_i-\hat{W}_i\|_2^2} \le \sqrt{\sum_{i=1}^{L}\|W_i-\hat{W}_i\|_F^2} = \|\theta-\hat{\theta}\|$.

Now we have

$$\begin{aligned}
\|G(\theta)-G(\hat{\theta})\|_2 &= \sqrt{\lambda_{\max}\left[(G(\theta)-G(\hat{\theta}))^\top(G(\theta)-G(\hat{\theta})\right]} \\
&= \sqrt{\lambda_{\max}\left[\sum_{l=1}^{L}(G_l(\theta)-G_l(\hat{\theta}))^\top(G_l(\theta)-G_l(\hat{\theta})\right]} \\
&\le \sqrt{L}\max_l\|G_l(\theta)-G_l(\hat{\theta})\|_2 \le \sqrt{L}\tau_L\dots\tau_1\tau_0\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}\|\theta-\hat{\theta}\|.
\end{aligned}$$

This completes the proof. □

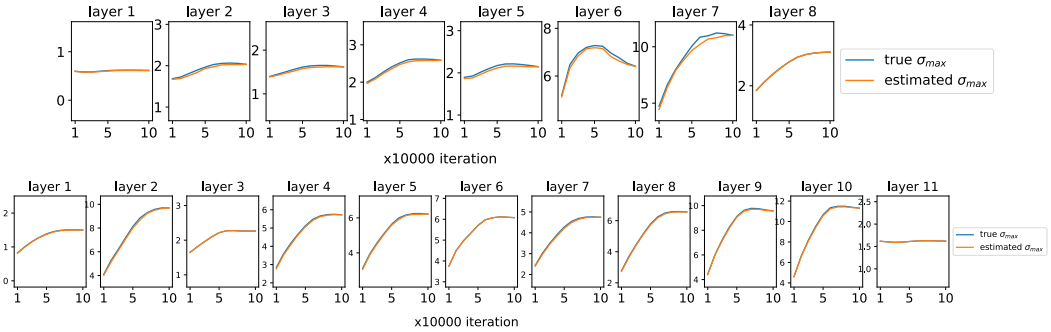


Figure 5: True spectrum and spectrum estimated by SN with one power iteration along the $100k$ iterations training on CIFAR-10 with CNN (above) and ResNet (bottom).

Proof of Lemma 3: Recall that $\nabla\mathcal{L}(\theta; X) = G(\theta)e(\theta)$, where $e(\theta) = \text{vec}(F(\theta; X) - Y)$. When $\theta \in \Omega(c)$, by the definition of $\Omega(c)$ we have $\|e(\theta)\| \le c$. Thus

$$\begin{aligned}\|\nabla\mathcal{L}(\theta; X) - \nabla\mathcal{L}(\hat{\theta}; X)\| &= \|G(\theta)e(\theta) - G(\hat{\theta})e(\hat{\theta})\| \\ &\le \|G(\theta) - G(\hat{\theta})\|_2\|e(\hat{\theta})\| + \|G(\theta)\|_2\|e(\theta) - e(\hat{\theta})\| \\ &= \|G(\theta) - G(\hat{\theta})\|_2\|e(\theta)\| + \|G(\theta)\|_2\|\text{vec}(F(\theta; X) - F(\hat{\theta}; X))\|.\end{aligned}$$

Denote

$$\hat{\tau} = \sqrt{L}\|X\|_2\tau_L \dots \tau_1\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}, \quad \tau_{\text{prod}} = \tau_L \dots \tau_1\|X\|_F,$$

Applying Claim D.3, Claim D.4 and Claim D.2, we get

$$\begin{aligned}&\|\nabla\mathcal{L}(\theta; X) - \nabla\mathcal{L}(\hat{\theta}; X)\| \\ &\le \hat{\tau}\|\theta - \hat{\theta}\|\|e(\theta)\| + \|\theta - \hat{\theta}\|\|X\|_F\hat{\tau}\frac{1}{\sqrt{L}\|X\|_2} \cdot \sqrt{L}\|X\|_2\tau_1 \dots \tau_L \\ &= \|\theta - \hat{\theta}\|\hat{\tau}\left(\|e(\theta)\| + \|X\|_F\tau_1 \dots \tau_L\right) \\ &= \|\theta - \hat{\theta}\|\hat{\tau}\left(\|e(\theta)\| + \tau_{\text{prod}}\right) \\ &\le \|\theta - \hat{\theta}\|\hat{\tau}\left(c + \tau_{\text{prod}}\right).\end{aligned}$$

Since $\beta = \sqrt{L}\|X\|_2\tau_L \dots \tau_1\sqrt{\sum_{i=1}^{L}\frac{1}{\tau_i^2}}\left(c + \tau_L \dots \tau_1\|X\|_F\right)$, the result holds. □

E SUPPLEMENTARY MATERIAL FOR SECTION 3

This section consists of more details of SN, and proofs of two results on spectrum of polynomials.

E.1 SPECTRAL NORMALIZATION AND ESTIMATED SPECTRAL NORM

For a given weight matrix W, Miyato et al. (2018) uses one iteration of power iteration to update u and v as follows: $u \leftarrow W_\top v/\|W_\top v\|$ and $v \leftarrow Wu/\|Wv\|$ and let the estimated spectral norm $\tilde{W} = u^T Wu/\|u\|^2$. The spectral norm operator is then defined as $\text{SN}(W) = W/\|\tilde{W}\|$.

Miyato et al. (2018) use "warm start", i.e., the updated u and v at certain iteration k will be used at iteration $k+1$. Thus if W moves slowly (the weight is updated by SGD), then they are essentially using multiple power iteration to estimate $\|W\|_2$. This may be why their estimation of the spectral norm by "only one power iteration" is quite good.

In Fig. 5, we present the figures that show the true spectral norm and the estimated spectral norm by the method of Miyato et al. (2018).

E.2 Proofs of Technical Results

E.2.1 Proof of Claim 3.1

This is a standard proof in linear algebra. For completeness, we include it here. Since Q is a real symmetric matrix, we write its eigenvalue decomposition as $Q = U\Lambda U^T$, where $U \in \mathbb{R}^{m\times m}$ is an orthogonal matrix and $\Lambda \in \mathbb{R}^{m\times m}$ is a diagonal matrix with entries $\lambda_i, i = 1, \ldots, m$. Therefore $Q^j = (U\Lambda U^T)^j = U\Lambda^j U^T$. Suppose $\hat{g}(x) = \sum_{i=0}^k c_i x^i$, then $\hat{g}(Q) = \sum_{i=0}^k c_i Q^i = \sum_{i=0}^k c_i U\Lambda^j U^T = U(\sum_{i=0}^k c_i \Lambda^j)U^T$. This is the eigenvalue decomposition of $\hat{g}(Q)$, where U is orthogonal and $\sum_{i=0}^k c_i\Lambda^j$ is a diagonal matrix with entries $\sum_{i=0}^k c_i\lambda_i^j = \hat{g}(\lambda_i)$. Therefore the eigenvalues of $\hat{g}(Q)$ are $\hat{g}(\lambda_i), i = 1, \ldots, m$.

E.2.2 Proof of Claim 3.2

For any polynomial p, we have

$$p(AA^T)A = Ap(A^TA). \tag{22}$$

The proof is just one line: suppose $p(x) = \sum_{i=0}^k a_i x^i$, then $p(AA^T)A = \sum_{i=0}^k a_i(AA^T)^i A = \sum_{i=0}^k a_i A(A^TA)^i = Ap(A^TA)$.

Suppose the singular values of $g(A) = p(AA^T)A = Ap(A^TA)$ are $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_m$.

Then the eigenvalues of $g(A)^Tg(A)$ $g(A)^Tg(A) = p(A^TA)^TA^TAp(A^TA) = p(A^TA)A^TAp(A^TA) \in \mathbb{R}^{m\times m}$ are $\tilde{\lambda}_1 = \tilde{\sigma}_1^2, \ldots, \tilde{\lambda}_n = \tilde{\sigma}_m^2$.

The eigenvalues of A^TA are $\sigma_1^2, \ldots, \sigma_m^2$, thus by Claim 3.1, the eigenvalues of $p(A^TA)A^TAp(A^TA)$ are $p(\sigma_i^2)\sigma_i^2 p(\sigma_i^2), i = 1, \ldots, m$.

The above two descriptions of the eigenvalues of $g(A)^Tg(A)$ should be the same, thus we have $\{p(\sigma_i^2)\sigma_i^2p(\sigma_i^2) \mid i = 1, \ldots, m\} = \{\tilde{\sigma}_i^2 \mid i = 1, \ldots, m\}$. Since singular values $\tilde{\sigma}_i$'s are non-negative, these $\tilde{\sigma}_i$'s (i.e., the singular values of $g(A)$) are $|p(\sigma_i^2)|\sigma_i = |g(\sigma_i)|, i = 1, \ldots, n$. □

F Experiments: More Details and More Results

We present more details of our experiments and also report further results.

Neural Net Structures: To compare PC and APC with other GAN training mechanisms, we conduct experiments on two low resolution datasets: CIFAR-10 (32×32) and STL-10 (48×48), and three high resolution datasets: LSUN bedroom (128×128), LSUN bedroom (256×256) and LSUN tower (256×256). Low resolution images are generated with both CNN and ResNet structure, following the setting of Miyato et al. (2018). For the high resolution images we use a CNN structure. Architecture details are listed Tab. 7, Tab. 8, Tab. 9 and Tab. 10.

Hyper-parameters: For all experiments, we use a batchsize of 64. For experiments on ResNet with $D_{\text{it}} = 5$, we set $\beta_1 = 0$ and $\beta_2 = 0.9$ in Adam. For others, $\beta_1 = 0.5$ and $\beta_2 = 0.999$ are used in Adam. Unless $D_{\text{it}} = 5$ is specified, $D_{\text{it}} = 1$ is always used across all experiments. We fix the learning rate for both the generator and the discriminator to be 0.0002.

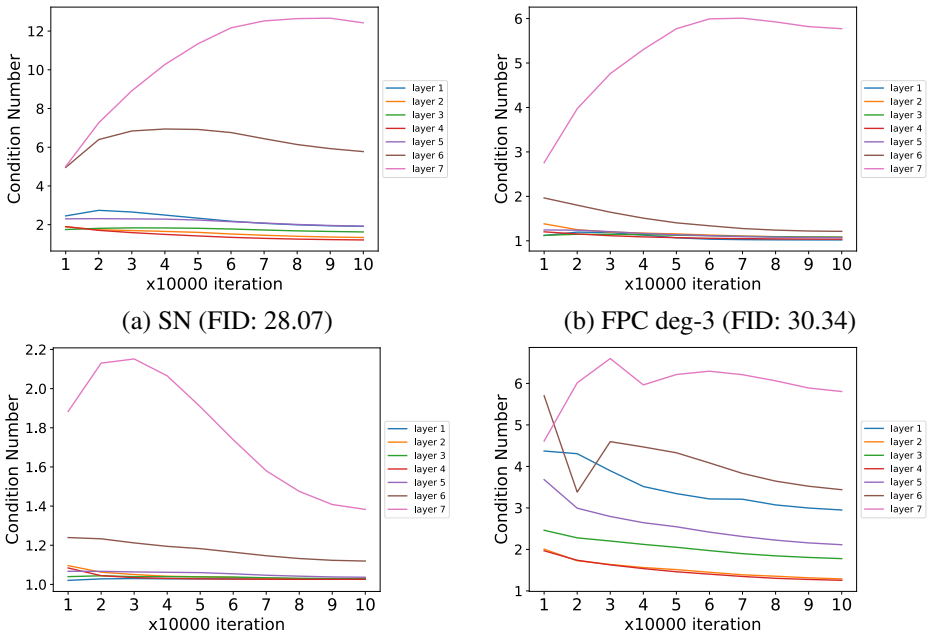
F.1 Results on CIFAR-10 and STL-10 with CNN

For all experiments on the CNN structure, we use the non-saturating log loss suggested by Goodfellow et al. (2014). The log loss is given by

$$L_D^{\text{Log}}(\theta; \hat{G}) = \min_\theta \frac{1}{2n}\left[\sum_i \log(D(x_i)) + \log(1 - D(\hat{G}(z_i)))\right],$$

$$L_G^{\text{Log}}(w; \hat{D}) = \min_w -\frac{1}{n}\sum_i \log(\hat{D}(G_w(z_i))).$$

The results are summarized in Tab. 4. We found that PC deteriorates the performance for both degree 3 and degree 7. More specifically, SN-GAN achieves an FID score 28.07 on CIFAR-10 and 44.06



(a) SN (FID: 28.07)

(b) FPC deg-3 (FID: 30.34)

(c) FPC deg-7 (FID: 32.38)

(d) APC (FID: 26.80)

Figure 6: Condition number evolution of all convolution layers in the CNN during the training on CIFAR-10 for (a) SN, (b) FPC deg-3, (c) FPC deg-7 and (d) APC and the corresponding FID.

on STL-10, which is reasonably good[1]. However, applying PC-GAN with deg-3 precondtioner or deg-7 preconditioner achieves worse or similar FID scores (30.34, 32.38 *vs*. 28.13; 43.89, 46.18 *vs*. 44.06).

We conjecture that PC hurts in this scenario because it is not "proper": conditioning is already good enough, so preconditioning does not help. We verify this conjecture by computing the condition numbers: for SN the largest condition number is around 12, which is indeed quite good. Applying PC-GAN with deg-3 and deg-7 precondtioner reduces the condition numbers to below 6, and to below 2.2, respectively, and the majority of the condition numbers are close to 1. Preconditioning clearly "overshoots." (See Fig. 6 When conditioning is already good, preconditioning only slightly improves the optimization, but the benefit is offset by harming the network's expressivity.

APC experiments support our hypothesis that proper PC can improve the performance. APC achieves an FID of 26.80 on CIFAR-10 and 42.43 on STL-10. If PC hurts the performance, it is likely due to strong preconditioning which reduces the representation power. It can be corrected by properly adjusting PC. A good strategy to apply PC is to apply a proper preconditioner only to not-well-conditioned layers.

F.2 Results on CIFAR-10 and STL-10 with ResNet

And for all experimetns on the ResNet structure, we use the non-saturating hinge loss (Miyato et al., 2018), which is

$$L_D^{\text{Hinge}}(\theta; \hat{G}) = \min_{\theta} \frac{1}{2n}\left[\sum_i \max(0, 1 - D_\theta(x_i)) + \sum_i \max(0, 1 + D_\theta(\hat{G}(z_i))\right],$$
$$L_G^{\text{Hinge}}(w; \hat{D}) = \min_{w} -\frac{1}{n}\sum_i \hat{D}(G_w(z_i)).$$

The results are listed in Tab. 4.

[1]FID 28.07 and 44.06 are reasonably good for this architecture; changing to other architectures such as ResNet can improve to 23 and 41, but we shall compare for a fixed architecture.

	CIFAR-10		STL-10	
	Inception Score ↑	FID ↓	Inception Score ↑	FID ↓
Real Dataset	11.24±0.19	5.18	24.45±0.41	5.34
Standard CNN				
BatchNorm	6.27±0.10	49.13	8.01±0.07	50.38
WGAN-GP	6.68±0.06	39.66	8.11±0.09	55.64
Weight Norm	6.96±0.06	37.69	7.12±0.08	55.39
Orthogonal Reg	6.67±0.09	33.41	7.80±0.09	51.27
Spectral Norm	7.42±0.08	28.07	8.32±0.10	44.06
SVD	7.28±0.08	27.74	8.34±0.06	44.29
FPC; deg-3	7.27±0.05	30.34	8.06±0.08	43.89
FPC; deg-7	7.00±0.08	32.38	7.99±0.07	46.18
APC	**7.46**±0.09	**26.80** (27.28±0.34)	**8.42**±0.09	**42.49** (42.90±0.30)
ResNet; $D_{\text{it}} = 1$				
Spectral Norm	4.82±0.05	84.74	4.25±0.02	169.58
Spectral Norm (2x updates)	5.13±0.07	77.85	4.52±0.03	147.90
SVD	7.97±0.87	22.61	8.32±0.02	51.18
SVD (2x updates)	8.01±0.08	20.75	8.96±0.09	38.01
FPC; deg-3	7.89±0.10	21.68	9.22±0.10	40.95
FPC; deg-3 (2x updates)	**8.25**±0.08	20.09	**9.46**±0.14	33.99
FPC; deg-7	8.05±0.13	22.41	9.28±0.16	37.19
FPC; deg-7 (2x updates)	8.16±0.04	**19.31**	9.34±0.11	34.28
APC	8.04±0.09	21.66 (22.21±0.34)	9.27±0.14	44.52 (45.17±0.75)
APC (2x updates)	8.16±0.11	19.53 (19.81±0.27)	9.34±0.14	**34.08** (34.52±0.49)
ResNet; $D_{\text{it}} = 5$				
Spectral Norm	7.87±0.08	23.80	8.87±0.07	36.33
SVD	7.92±0.06	22.31	9.24±0.06	36.85
FPC; deg-3	7.85±0.12	22.60	9.21±0.06	36.02
FPC; deg-7	7.93±0.09	21.79	8.94±0.13	41.96
APC	**8.00**±0.13	**20.32** (20.80±0.35)	**9.24**±0.08	**34.94** (35.73±0.56)

Table 4: Inception score (IS) (higher is better) and Frechét Inception distance (FID) (lower is better) on CIFAR-10 and STL-10. For APC, across 3 runs we report the best score (across runs and iterations) as well as in parenthesis is averaged (across runs) best (across iterations) mean and std.

$D_{\text{it}} = 1$: We run all baselines and PC-GANs for $200k$ iterations. We find that for CIFAR-10, both SVD and PC-GANs can converge within $100k$ iterations. One remarkable point is that APC-$D_{\text{it}} = 1$ can beat the SN-$D_{\text{it}} = 5$ by 2 FID scores (21.66 *vs*. 23.80) using just $100k$ iterations, i.e., using only 1/3 of the training time (Fig. 4(b)). And with $200k$ iterations, APC-$D_{\text{it}} = 1$ can improve SN-$D_{\text{it}} = 5$ by 4 FID scores (19.53 *vs*. 23.80), using 2/3 of the training time. Both SVD and PC-GANs need $200k$ iterations to converge on the STL-10 dataset. APC-GAN-$D_{\text{it}} = 1$ can beat both SVD-$D_{\text{it}} = 1$ and SN-$D_{\text{it}} = 5$ by 4 FID scores (34.08 *vs*. 38.01) and 2 FID scores (34.08 *vs*. 36.66).
$D_{\text{it}} = 5$: We run all experiments for $100k$ iterations. We find $D_{\text{it}} = 5$ is a hard case, since all baselines are well-trained under this setting. But with proper PC layers, APC can still improve the FID scores by approximate 2 scores.

EMA: Since training stability and convergence are two orthogonal problems in GAN training, we claim that applying EMA can further improve the performance of PC. The EMA generator applies a weighted averaging across all the generators along the training process. Specifically, the EMA generator at the t^{th} iteration is obtained as follows:

$$w_{\text{EMA}}^{(t)} = \beta w_{\text{EMA}}^{(t-1)} + (1-\beta) w^{(t)}, \tag{23}$$

where $w_{\text{EMA}}^{(0)} = w^{(0)}$ and $w^{(t)}$ are the original generator parameters. Due to the weighted averaging trick, EMA can speed up convergence. We apply EMA on all baselines and PC-GAN. We set the $\beta = 0.9999$ for all experiments and provide results in Tab. 5.

	CIFAR-10		STL-10	
	Inception Score ↑	FID ↓	Inception Score ↑	FID ↓
Real Dataset	11.24±0.19	5.18	24.45±0.41	5.34
Standard CNN				
Spectral Norm	7.52±0.09	26.76	8.48±0.07	43.68
SVD	7.46±0.09	26.44	8.42±0.13	43.69
FPC; deg-3	7.43±0.11	27.70	8.07±0.11	43.51
FPC; deg-7	7.20±0.12	30.12	8.01±0.12	45.06
APC	**7.64**±0.08	**25.30** (25.83±0.48)	**8.52**±0.06	**41.48** (41.72±0.33)
ResNet; $D_{\text{it}} = 1$				
Spectral Norm	5.11±0.03	80.99	4.37±0.04	166.30
Spectral Norm (2x updates)	5.32±0.01	74.26	4.60±0.03	145.89
SVD	8.36±0.06	19.93	8.78±0.10	49.32
SVD (2x updates)	8.42±0.08	18.23	9.19±0.13	36.85
FPC; deg-3	8.69±0.05	17.04	9.49±0.11	32.95
FPC; deg-3 (2x updates)	8.75±0.10	16.17	9.62±0.10	32.36
FPC; deg-7	8.75±0.13	17.05	9.62±0.15	32.86
FPC; deg-7 (2x updates)	8.78±0.08	16.95	**9.65**±0.10	**31.02**
APC	8.93±0.14	16.20 (16.61±0.41)	9.62±0.15	36.39 (36.73±0.36)
APC (2x updates)	**8.99**±0.10	**16.04** (16.06±0.02)	9.64±0.12	31.05 (31.42±0.38)
ResNet; $D_{\text{it}} = 5$				
Spectral Norm	8.37±0.13	20.98	9.14±0.12	33.06
SVD	8.30±0.07	19.22	9.50±0.07	33.17
FPC; deg-3	8.49±0.11	19.00	9.51±0.13	34.04
FPC; deg-7	8.42±0.06	19.06	9.17±0.11	39.46
APC	**8.63**±0.12	**17.52** (17.90±0.55)	**9.59**±0.12	**31.76** (32.41±0.50)

Table 5: Inception score (IS) (higher is better) and Frechét Inception distance (FID) (lower is better) on CIFAR-10 and STL-10 with EMA trick. For APC, across 3 runs we report the best score (across runs and iterations) as well as in parenthesis averaged (across runs) best (across iterations) mean and std.

	degree-3	degree-5	degree-7	degree-9	APC
CIFAR-10	30.34	30.51	32.38	32.69	26.80
LSUN-bedroom	35.61	35.71	32.43	30.35	31.17

Table 6: FPC-GAN's performance on CIFAR-10 and LSUN bedroom 256 with different degrees.

F.3 Robustness of APC-GAN

We also test the robustness of both SN-GAN and APC-GAN on different choices of hyper-parameters. We study four learning rate (dlr, glr) choices and four choices for the Adam optimizer parameters (β_1, β_2). The setting details are listed in Fig. 7. We keep all other hyper-parameters identical. In Fig. 7, we show the SN-GAN and APC-GAN's Inception Scores (IS) and FID scores on STL-10 with CNN structure using these settings. We observe SN-GAN and APC-GAN are equally robust w.r.t. the learning rate settings. APC-GAN also performs reasonably well with aggressive momentum parameters (setting F and H) while SN-GAN performs less well.

F.4 FPC results with different degrees

To test how sensitive FPC-GAN to different degrees, we conduct a serious of FPC-GAN training with degree 3, 5, 7, 9 on CIFAR-10 and LSUN-bedroom (256×256). The result is listed in Tab. 6. We can see that FPC-GAN is robust to degree overall. But the optimal degree requires tuning. Thus, APC-GAN is a automatic trade-off solution.

F.5 Generated Samples

setting	glr	dlr	β_2	β_2	SN IS	APC IS	SN FID	APC FID
A	1e-4	1e-4	0.5	0.999	8.14±0.09	8.15±0.11	46.38	45.05
B	2e-4	1e-4	0.5	0.999	8.29±0.11	8.39±0.06	43.92	43.68
C	5e-4	1e-4	0.5	0.999	8.28±0.07	8.20±0.08	44.96	43.77
D	1e-3	1e-4	0.5	0.999	8.12±0.08	8.18±0.05	46.08	44.23
E	2e-4	2e-4	0.0	0.9	8.49±0.07	8.46±0.12	42.24	41.29
F	2e-4	2e-4	0.5	0.9	2.82±0.01	7.72±0.09	181.27	57.41
G	2e-4	2e-4	0.5	0.999	8.32±0.12	8.42±0.09	44.06	42.49
H	2e-4	2e-4	0.9	0.999	5.18±0.04	6.62±0.10	95.36	69.31

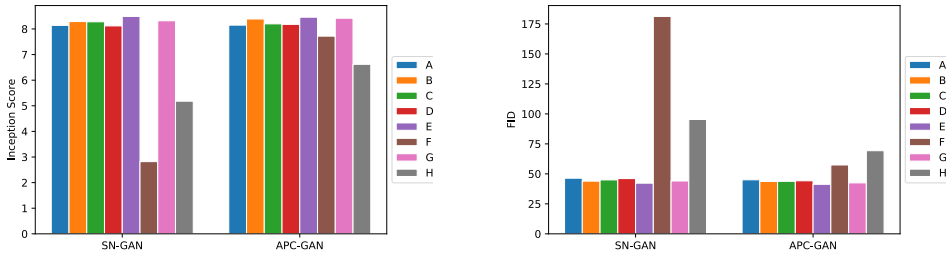


Figure 7: Hyper-parameter settings we studied in the robustness experiments. IS (left) and FID (right) scores on STL-10 with SN-GAN and APC-GAN.

(a) Generator
$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
$128 \to h \times w \times 512$, dense, linear
4×4, stride 2 deconv, 256, BN, ReLU
4×4, stride 2 deconv, 128, BN, ReLU
4×4, stride 2 deconv, 64, BN, ReLU
3×3, stride 1 conv, 3, Tanh

(b) Discriminator
RGB image $x \in [-1, 1]^{H \times W \times 3}$
3×3, stride 1 conv, 64, LReLU 0.1
4×4, stride 2 conv, 128, LReLU 0.1 3×3, stride 1 conv, 128, LReLU 0.1
4×4, stride 2 conv, 256, LReLU 0.1 3×3, stride 1 conv, 256, LReLU 0.1
4×4, stride 2 conv, 512, LReLU 0.1 3×3, stride 1 conv, 512, LReLU 0.1
$h \times w \times 512 \to s$, dense, linear

Table 7: CNN models for CIFAR-10 and STL-10 used in our experiments on image generation. $h = w = 4$, $H = W = 32$ for CIFAR-10. $h = w = 6$, $H = W = 48$ for STL-10.

(a) Generator
$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense, $4 \times 4 \times 256$
ResBlock up 256
ResBlock up 256
ResBlock up 256
BN, ReLU, 3×3 conv, 3 Tanh

(b) Discriminator
RGB image $x \in [-1, 1]^{32 \times 32 \times 3}$
ResBlock down 128
ResBlock down 128
ResBlock 128
ResBlock 128
ReLU
Global sum pooling
dense $\to 1$

Table 8: Regular ResNet models for CIFAR-10 used in our experiments on image generation.

(a) Generator
$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
dense, $6 \times 6 \times 512$
ResBlock up 256
ResBlock up 128
ResBlock up 64
BN, ReLU, 3×3 conv, 3 Tanh

(b) Discriminator
RGB image $x \in [-1, 1]^{48 \times 48 \times 3}$
ResBlock down 64
ResBlock down 128
ResBlock down 256
ResBlock down 512
ResBlock down 1024
ReLU
Global sum pooling
dense $\rightarrow 1$

Table 9: Regular ResNet models for STL-10 used in our experiments on image generation.

(a) Generator
$z \in \mathbb{R}^{128} \sim \mathcal{N}(0, I)$
$128 \rightarrow 4 \times 4 \times$ 1024, dense, linear
4×4, stride 2 deconv, 512, BN, ReLU
4×4, stride 2 deconv, 256, BN, ReLU
4×4, stride 2 deconv, 128, BN, ReLU
4×4, stride 2 deconv, 64, BN, ReLU
4×4, stride 2 deconv, 32, BN, ReLU
4×4, stride 2 deconv, 16, BN, ReLU
3×3, stride 1 conv, 16, BN, ReLU
3×3, stride 1 conv, 3, Tanh

(b) Discriminator
RGB image $x \in [-1, 1]^{H \times W \times 3}$
4×4, stride 2 conv, 32, LReLU 0.1
4×4, stride 2 conv, 64, LReLU 0.1
4×4, stride 2 conv, 128, LReLU 0.1
4×4, stride 2 conv, 256, LReLU 0.1
4×4, stride 2 conv, 512, LReLU 0.1
4×4, stride 2 conv, 1024, LReLU 0.1
$4 \times 4 \times 1024 \rightarrow 1$, dense, linear

Table 10: CNN models for LSUN 256×256 image generation.

(a) real data (b) Batch Norm

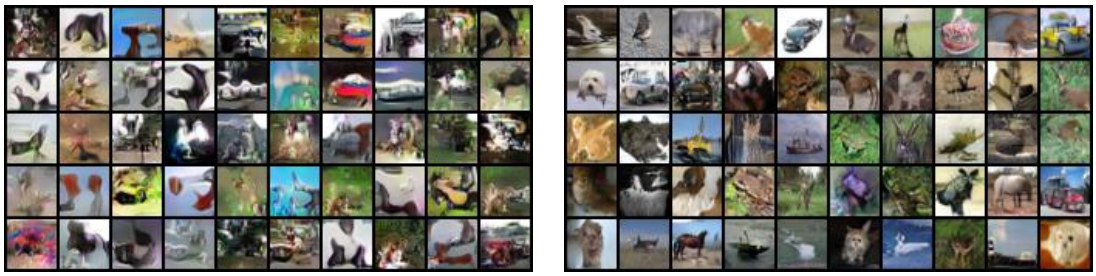
(c) WGAN-GP (d) Weight Norm

(e) Orthogonal Reg (f) Spectral Norm

(g) SVD (h) FPC, deg-3

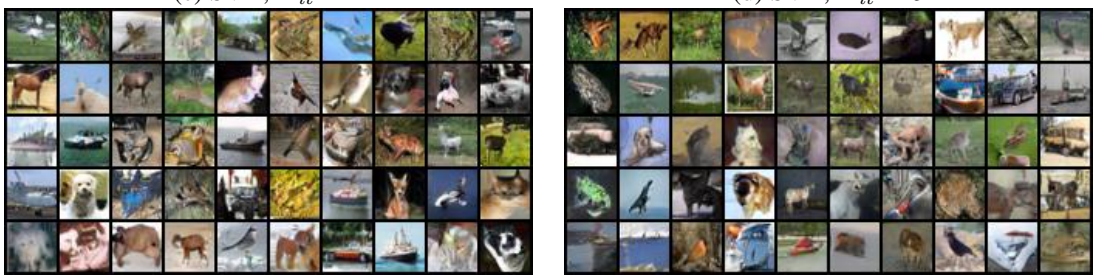
(i) FPC, deg-7 (j) APC

Figure 8: Generated CIFAR-10 samples using a CNN.

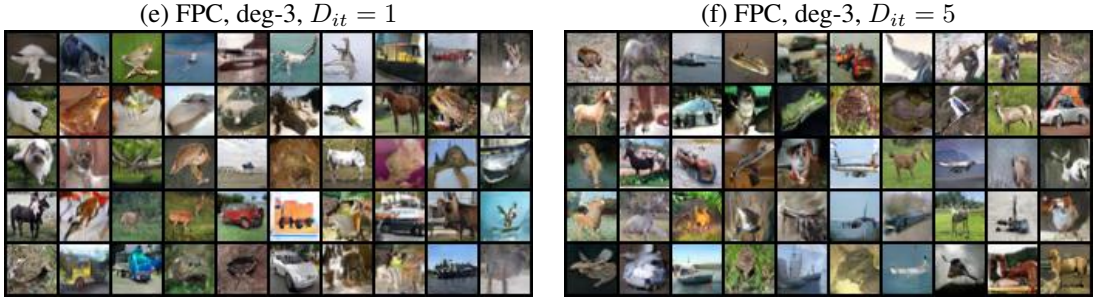


(a) SN, $D_{it} = 1$ (b) SN, $D_{it} = 5$

(c) SVD, $D_{it} = 1$ (d) SVD, $D_{it} = 5$



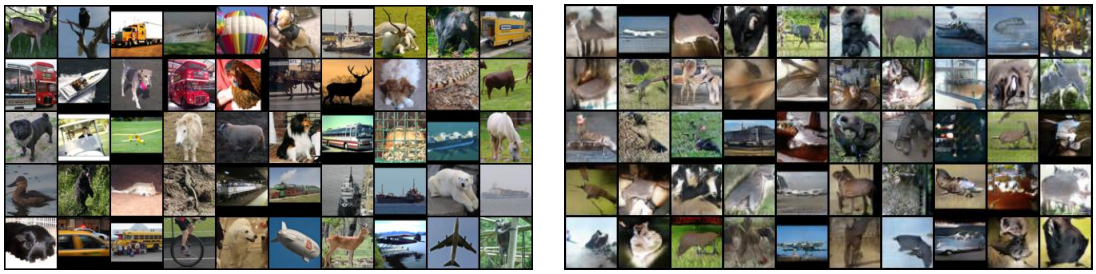
(e) FPC, deg-3, $D_{it} = 1$ (f) FPC, deg-3, $D_{it} = 5$



(g) FPC, deg-7, $D_{it} = 1$ (h) FPC, deg-7, $D_{it} = 5$

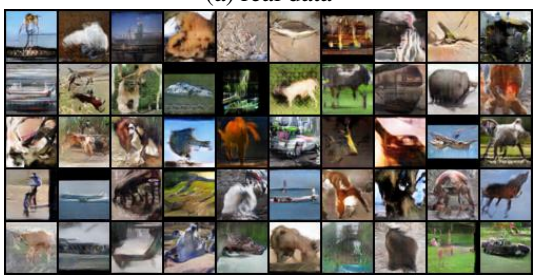
(i) APC, $D_{it} = 1$ (j) APC, $D_{it} = 5$

Figure 9: Generated CIFAR-10 samples using a ResNet with $D_{\text{it}} = 1$ (left) and $D_{\text{it}} = 5$ (right).



(a) real data

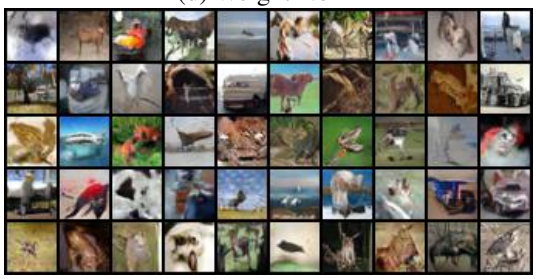
(b) Batch Norm



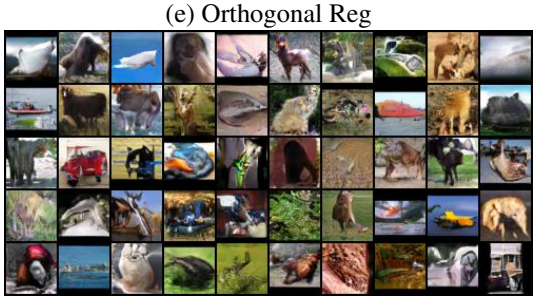
(c) WGAN-GP

(d) Weight Norm

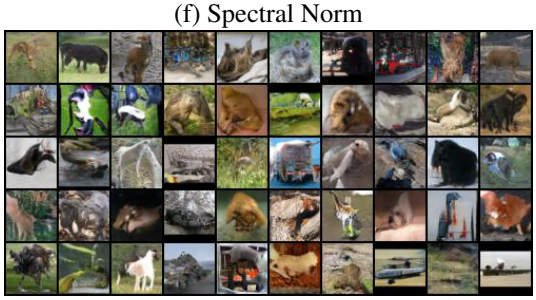
(e) Orthogonal Reg



(f) Spectral Norm



(g) SVD



(h) FPC, deg-3

(i) FPC, deg-7

(j) APC

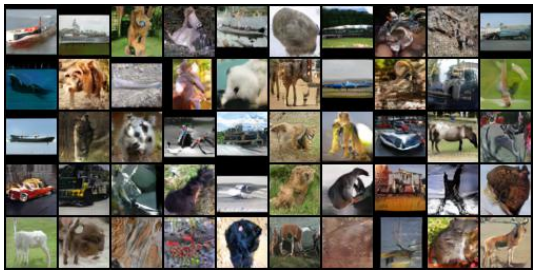
Figure 10: Generated STL-10 samples using a CNN.

(a) SN, $D_{it} = 1$

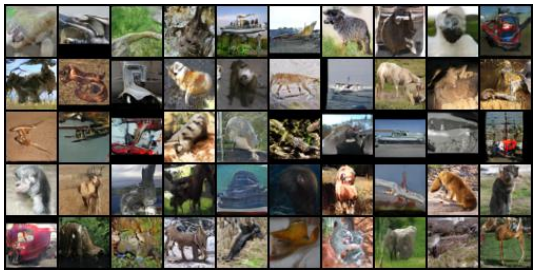
(b) SN, $D_{it} = 5$

(c) SVD, $D_{it} = 1$

(d) SVD, $D_{it} = 5$



(e) FPC, deg-3, $D_{it} = 1$



(f) FPC, deg-3, $D_{it} = 5$

(g) FPC, deg-7, $D_{it} = 1$

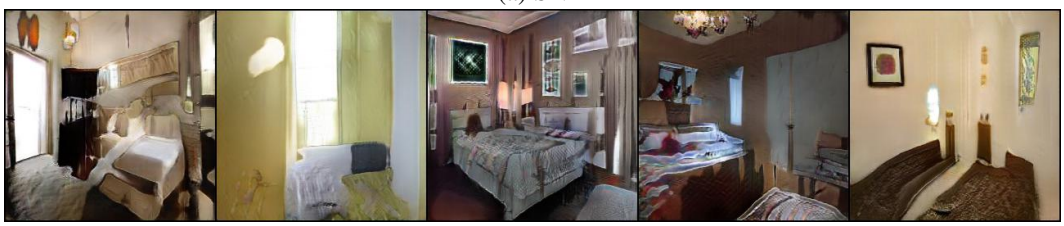
(h) FPC, deg-7, $D_{it} = 5$

(i) APC, $D_{it} = 1$

(j) APC, $D_{it} = 5$

Figure 11: Generated STL-10 samples using a ResNet with $D_{\text{it}} = 1$ (left) and $D_{\text{it}} = 5$ (right).

(a) SN



(b) SVD

(c) FPC, deg-3

(d) APC

Figure 12: Generated LSUN Bedroom 256×256 samples using CNN.

(a) SN

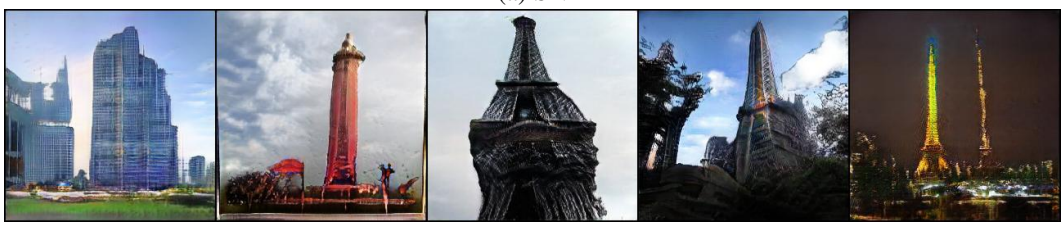
(b) SVD

(c) FPC, deg-3

(d) APC

Figure 13: Generated LSUN Living Room 256×256 samples using CNN.

(a) SN



(b) SVD

(c) FPC, deg-3

(d) APC

Figure 14: Generated LSUN Tower 256×256 samples using CNN.