Double Descent and Other Interpolation Phenomena in GANs

Lorenzo Luzi
Rice University
enzo@rice.edu

Yehuda Dar
Rice University
ydar@rice.edu

Richard G. Baraniuk
Rice University
richb@rice.edu

Abstract

We study overparameterization in generative adversarial networks (GANs) that can interpolate the training data. We show that *overparameterization can improve generalization performance and accelerate the training process*. We study the generalization error as a function of latent space dimension and identify two main behaviors, depending on the learning setting. First, we show that overparameterized generative models that learn distributions by minimizing a metric or f-divergence do not exhibit double descent in generalization errors; specifically, *all* the interpolating solutions achieve the same generalization error. Second, we develop a new pseudo-supervised learning approach for GANs where the training utilizes pairs of fabricated (noise) inputs in conjunction with real output samples. Our pseudo-supervised setting exhibits double descent (and in some cases, triple descent) of generalization errors. We combine pseudo-supervision with overparameterization (i.e., overly large latent space dimension) to accelerate training while performing better, or close to, the generalization performance without pseudo-supervision. While our analysis focuses mostly on linear GANs, we also apply important insights for improving generalization of nonlinear, multilayer GANs.

1 Introduction

Generative adversarial networks (GANs) [1] are a prominent concept for addressing data generation tasks in contemporary machine learning. GANs learn a data generator model that produces new instances from a data class that is represented by a set of training examples. A GAN's generator network is trained in conjunction with a discriminator network that evaluates the generator's ability and directs it towards better performance. GANs have an intricate design and training philosophy; however, significant work is still needed to satisfactorily understand both practice and theory.

A key aspect that complicates the understanding of GANs is that, like many other deep learning architectures, they are highly complex models with typically many more parameters than the number of training data samples. This promotes the assumption that GANs are *overparameterized models* that can be trained to interpolate (i.e., memorize) their training examples. Yet, overparameterized GANs are capable of generating high quality data beyond their training datasets. The analysis of overparameterized machine learning is a highly active research area that is mainly focused on supervised learning problems such as regression [2, 3, 4, 5] and classification [6, 7, 8]. *The study of overparameterization in the unsupervised learning and data generation problems relevant to GANs is uncharted territory that we are first to explore in this paper.*

This paper develops a new framework for the study of generalization and overparameterization in linear GANs. *We examine the generalization of linear GANs at different parameterization levels by varying the latent space dimension*, which in a GAN is the dimension of the input (random noise) vectors to the data generator. This is a practical way of controlling the parameterization of our models, since we do not need to consider modifying the width or depth of the generator network.

Preprint. Under review.

Our framework leads us to the following key insights on how the generalization performance of overparameterized linear GANs is affected by the training approach.

First, GAN training via **minimization of a distribution metric or f-divergence** results in *unsatisfactory generalization* performance when the generator model is overparameterized and interpolates its noisy training data. Specifically, we prove that under such a training process *all* overparameterized solutions have the same generalization performance. Moreover, the best generalization is obtained by an underparameterized solution with the same dimension as the true latent space dimension of the data, which is usually unknown. This set of interpolating solutions which have constant test error establishes a new generalization behavior that does not exist in the current literature on overparameterized machine learning.

Second, our theoretical studies inspire a new **pseudo-supervised training** regime for GANs and show that it can *improve generalization performance in overparameterized settings* where interpolation of noisy training data occurs. Our pseudo-supervised approach selects a subset (or all) of the training data examples and individually associates them with random (noise) vectors that act as their latent representations (i.e., the inputs given to the generator to yield the respective training data). Pseudo-supervision *accelerates the training process* and improves generalization by reducing the number of effective degrees of freedom in overparameterized GAN learning (although in many cases the learned GAN can still interpolate the training data). We develop several implementations for the pseudo-supervised optimization objective and examine their respective generalization behaviors, which we show to include *double descent* and also *triple descent* of generalization errors as a function of the latent space dimension of the learned GAN.

Third, encouraged by our new insights into linear GANs, we explore their implications for *nonlinear, multilayer* GANs. Specifically, we implement and study our pseudo-supervised learning scheme for a gradient-penalized Wasserstein GAN [9] on the MNIST digit dataset of binary images. Our results demonstrate that pseudo-supervised learning significantly improves generalization performance and accelerates training when compared to training the same GAN without pseudo-supervision.

2 Related work

GANs [1] have been very successful in modeling complex data distributions, such as distributions of images[10, 11, 12]. These models are usually trained by having two competing networks: a generator network which attempts to approximate the data distribution and a discriminator network which attempts to classify between data from the training set and generated data. The objective function can either be an f-divergence [1, 13, 14] or a metric [15, 9] and is typically minimized by the generator while simultaneously being maximized by the discriminator. This minmax game can be unstable [16, 17] and is hard to analyze in full generality; therefore we turn to linear GANs.

Recent work [18] has studied GANs with linear generators, quadratic discriminators, and Gaussian data (this has been named the LQG setting). In this setting, the objective loss is the 2-Wasserstein distance between two Gaussian distributions $\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$:

$$\mathcal{W}_2(\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)) = \sqrt{\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_2^2 + \mathrm{Tr}(\boldsymbol{\Sigma}_1) + \mathrm{Tr}(\boldsymbol{\Sigma}_2) - 2\mathrm{Tr}\Big(\big(\boldsymbol{\Sigma}_1^{\frac{1}{2}}\boldsymbol{\Sigma}_2\boldsymbol{\Sigma}_1^{\frac{1}{2}}\big)^{\frac{1}{2}}\Big)}. \quad (1)$$

This distance is well known [19, 20] and is even used in the calculation of the well known evaluation metric FID [21] in the GAN literature. One result in the LQG setting [18] is that the principal component analysis (PCA) solution is an optimal solution for the generator in the minmax optimization.

In supervised problems, it was widely believed that the generalization error behavior as a function of the learned model complexity is completely characterized by the bias-variance tradeoff, i.e., in a supervised setting, the test error goes down and then back up as the learned model is more complex (e.g., has more parameters). Recently, it has been shown that test errors can have a *double descent* shape [22, 23] as a function of the learned model complexity. Specifically, in the double descent shape the test error goes back down when the learned model is sufficiently complex (i.e., overparameterized) to interpolate the training data (i.e., achieve zero training error). Remarkably, the double descent shape implies that the best generalization performance can be achieved despite perfect fitting of noisy training data. Typically, when models have many more parameters than training data, many mappings can be learned to perfectly fit (i.e., interpolate) the supervised pairs of examples. Therefore, a mapping with small norm is a natural (parsimonious) choice and tends to yield low test error even

when the number of parameters is large. The research on overparameterized learning and double descent phenomena has been mostly focused on regression [2, 3, 4, 5] and classification [6, 7, 8] problems. Some work has been done in overparameterized GANs [24] to understand how training stability is affected by increasing the width and depth of networks. By contrast, *we are the first to study generalization performance and double descent behavior in GANs.*

Since linear GANs are associated with PCA, our current study relates to our recent work on overparameterization in PCA [25] showing that, as one relaxes the orthonormal constraints and adds supervision to PCA, double descent emerges. Moreover, if the learning is fully supervised and has no orthonormal constraints, then the problem becomes linear regression that estimates a linear subspace. Hence, one can solve learning problems that are partly supervised and partly orthonormally constrained to obtain solutions to problems that are in-between PCA and linear regression. We will leverage this powerful idea to study overparameterization in linear GANs.

3 Bad generalization: Test errors are constant in the overparameterized regime

3.1 No double descent: The learned distribution has no degrees of freedom

The goal of training GANs and generative models in general is to learn the distribution of the data. This is typically done by minimizing a distance between a fixed (i.e., given) distribution p_f, such as the empirical distribution of the training data, and the generated distribution $p_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$. The training dataset $\mathcal{D}$ includes n examples $\{\mathbf{x}_i\}_{i=1}^n \in \mathbb{R}^d$. The next theorem characterizes interpolating solutions for these kinds of problems.

Theorem 1. *Let P be the set of all probability distributions defined on the measurable space $(\Omega, \mathcal{F})$ equipped with any metric or f-Divergence denoted q. Then, we let our training loss be $\mathcal{L}^{train}(\{\mathbf{x}_i\}_{i=1}^n, \boldsymbol{\theta}) = q(p_f, p_{\boldsymbol{\theta}})$ for $p_f, p_{\boldsymbol{\theta}} \in P$ and the test error is given by $\mathcal{L}^{test}(\boldsymbol{\theta}) = q(p_t, p_{\boldsymbol{\theta}})$ for the true distribution $p_t \in P$. Then, for any interpolating solution $\boldsymbol{\theta}$, i.e., any $\boldsymbol{\theta}$ so that $\mathcal{L}^{train}(\{\mathbf{x}_i\}_{i=1}^n, \boldsymbol{\theta}) = 0$, we have that*

$$\mathcal{L}^{test}(\boldsymbol{\theta}) = L^{test}_{interpolate}$$

where $L^{test}_{interpolate}$ is a non-negative constant that depends on q and p_f.

Proof. Let $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}$ be two interpolating solutions. Since q is a metric or f-divergence, the zero training errors of the interpolating solutions $\boldsymbol{\theta}^*$ and $\boldsymbol{\theta}$ imply that $p_{\boldsymbol{\theta}^*} = p_f$ and $p_{\boldsymbol{\theta}} = p_f$. Thus,

$$\mathcal{L}^{\text{test}}(\boldsymbol{\theta}) = q(p_t, p_{\boldsymbol{\theta}}) = q(p_t, p_f) = q(p_t, p_{\boldsymbol{\theta}^*}) = \mathcal{L}^{\text{test}}(\boldsymbol{\theta}^*).$$

By letting $L^{\text{test}}_{\text{interpolate}} \triangleq q(p_t, p_f) \geq 0$, we get the desired result. □

In other words, there is no double descent behavior because the test error is **constant** in the overparameterized regime of interpolating solutions. This differs from the widely studied regression setup in that we are trying to minimize the distance between two distributions rather than data points drawn from those distributions. In other words, we treat the data itself as a distribution and not as a set of data points. Importantly, this result is not specific to GANs but to any generative model that is trained to minimize the distance between the generated distribution and a fixed distribution.

We now narrow our focus to a specific data model to help understand the constant regime of generalization errors. Recall from Section 2 that in the LQG setting [18], PCA is a solution for the optimal linear generator. Hence, we can study PCA solutions and evaluate them using the 2-Wasserstein metric (Equation (1)) to see the generalization error of the linear generator in the LQG setting. We assume that our training data $\{\mathbf{x}_i\}_{i=1}^n$ are realizations of a random vector $\mathbf{x} \in \mathbb{R}^d$ that satisfies the noisy linear model $\mathbf{x} = \boldsymbol{\Gamma}\mathbf{z} + \boldsymbol{\epsilon}$. Here $\boldsymbol{\Gamma} \in \mathbb{R}^{d \times m}$ is a rank m matrix (for $m < d$), $\mathbf{z} \in \mathbb{R}^m$ is a latent random vector of a zero-mean isotropic Gaussian distribution, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$ is a noise vector. The true latent dimension m is unknown; hence, we will pick $k > 0$ and learn a generator matrix $\mathbf{G} \in \mathbb{R}^{d \times k}$. The true, uncorrupted distribution is a Gaussian with distribution $\mathbf{x}_{\text{true}} = \boldsymbol{\Gamma}\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}\boldsymbol{\Gamma}^\top)$. Thus, if the learned latent dimension k equals the true latent dimension m, then $\mathbf{G} = \boldsymbol{\Gamma}$ is an optimal solution. Since the covariance matrix of $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Gamma}\boldsymbol{\Gamma}^\top + \sigma^2 \mathbf{I}_d)$ is

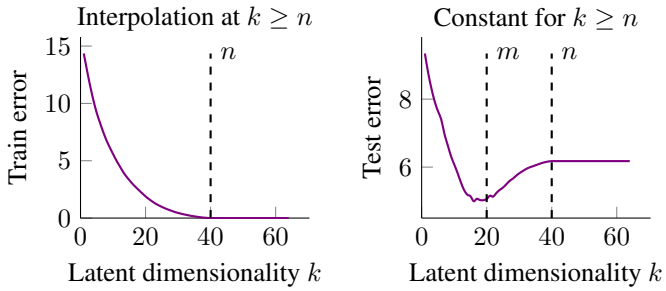

Figure 1: A GAN's test error becomes constant when the model interpolates, i.e., when the latent dimensionality k equals the number of training samples n. Moreover, the test error achieves its minimum when the latent dimensionality k is near the true model's dimensionality m.

the sum of a low rank covariance matrix $\mathbf{\Gamma}\mathbf{\Gamma}^\top$ and a full rank noise covariance matrix, our choice of k will affect how much we overfit to the noise distribution.

We consider $m < n < d$, i.e., the number of training examples n is higher than the true latent space dimension m, and lower than the data dimension d, for several reasons. Most importantly, data often is assumed to lie on a low dimensional manifold in a higher dimensional space. Thus if $m \geq d$, then the data will have a non-zero probability of being in any open set in $\mathbb{R}^d$, which is clearly not true for many types of data, such as natural images. We also choose to study $m < d$ because it will allow our model to overfit (when the learned latent dimension $k > m$). Now we turn to our choice of n and note that if $n \geq d$, we get the typical U-shaped curve of the bias-variance tradeoff for generalization error as a function of the learned latent dimension k. If $n \leq m$, then the generalization error is just monotonically decreasing in k and is of little interest. For these reasons, we consider only $m < n < d$, which entails new overparameterized settings of great interest.

We train a GAN by picking the top k principal components ($k \leq d$), namely, minimizing the training loss

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathbf{X}) = \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{X}\|_F^2 \tag{2}$$

under the constraint that the $d \times k$ matrix $\mathbf{G}$ has orthonormal columns. Moreover, $\mathbf{X} \in \mathbb{R}^{d \times n}$ denotes the data matrix with n training examples as its columns. If $k > n$, we run out of nonzero principal components and cannot add any more; the learned generator interpolates by producing zero training error. However, the test error will increase if we learn noise, i.e., if the eigenvalues and eigenvectors of $\mathbf{\Gamma}\mathbf{\Gamma}^\top + \sigma^2\mathbf{I}_d$ are corrupted by the noise covariance $\sigma^2\mathbf{I}_d$. Figure 1 shows the train and test errors for the learned model as a function of the learned latent dimension k. We obtain generalization behavior in two stages. First, there is a U-shape with a minimum around $k = m$; then, as the solutions start to interpolate in the overparameterized regime of $k > n$, we observe a constant test error.

To relate this back to Theorem 1, here the training data distribution p_f is $\mathcal{N}(\widehat{\boldsymbol{\mu}}_{\mathbf{x}}, \widehat{\mathbf{\Sigma}}_{\mathbf{x}})$, where $\widehat{\boldsymbol{\mu}}_{\mathbf{x}} \in \mathbb{R}^d$ and $\widehat{\mathbf{\Sigma}}_{\mathbf{x}} \in \mathbb{R}^{d \times d}$ are the empirical mean vector and covariance matrix of the training data, respectively. Roughly speaking, we can think of the generator as learning the true distribution with some noise for the first m components and then just learning noise in the subspace orthogonal to the data; technically, we learn wrong directions in the data for small k if the noise variance σ^2 is very large. In this setting, where the number of training samples n allows us to interpolate, the best that one can do is to try to guess m by using prior knowledge or training multiple models. These solutions are not satisfactory in many scenarios, so we delve deeper into understanding why the test error in the overparameterized regime is constant.

3.2 A pseudometric relaxation of the training loss: Overparameterization yields degrees of freedom that do not affect generalization

The problem in Section 3.1 is that we are optimizing over a metric q, which has a definiteness property, i.e., $q(x, y) = 0$ if and only if $x = y$. If we relax this property, we are left with a pseudometric; similarly, we can relax this property to obtain a non-definite f-divergence. Interestingly, we found that subsampling coordinates of the data is equivalent to using a pseudometric, and we will use subsampling in the next section to control our level of parameterization. We provide a detailed

discussion on the pseudometric formulation in Appendix A since several papers on double descent use feature subsampling to control the parameterization of the model [2, 25, 26].

The learned distribution cannot change once we interpolate, but the learned matrix $\mathbf{G}$ can. Let $\mathbf{U} \in \mathbb{R}^{k \times k}$ be any orthonormal matrix. Then, note that $\mathbf{GU}$ is also a solution which yields the desired distribution. This is because $(\mathbf{GU})(\mathbf{GU})^\top = \mathbf{GG}^\top$ (see Theorem 7.3.11 in [27]) since positive definite matrices are unique up to a orthonormal transformation. This means that we can learn as many matrices $\mathbf{G}$ as there are orthonormal matrices, independent of how much data we have. The number of samples n has no effect on the degrees of freedom of the learned map $\mathbf{G}$, but it does affect the degrees of freedom of which distributions we can learn. More specifically, we are able to learn a distribution whose covariance matrix shares the top n eigenvectors of the sample covariance matrix of the training data distribution p_f. In the next section, we will remove these degrees of freedom and force our optimization to have a smaller and smaller feasible set.

3.3 Double descent: Reducing the number of degrees of freedom through supervision

Since PCA gives us a solution to GANs trained in the LQG setting, we turn to studying overparameterization in PCA to understand overparameterization in GANs. We showed that PCA (with soft orthonormality constraints) does exhibit double descent if supervision was added to the training [25]. This is consistent with our understanding, since supervision will cause the learned map to have fewer degrees of freedom. Our work in [25] focuses on learning a linear subspace for the purpose of dimensionality reduction, therefore we modify our previous idea to work for the purpose of data generation.

Consider a setting where n_{sup} out of the n training examples are given with their true latent vectors. Namely, the training dataset $\mathcal{D}$ includes $n_{\text{sup}} \in \{0, \dots, n\}$ supervised examples $\{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^{n_{\text{sup}}}$ and $n_{\text{unsup}} = n - n_{\text{sup}}$ unsupervised examples $\{\mathbf{x}_i\}_{i=n_{\text{sup}}+1}^{n}$. The training data vectors are organized as the columns of the matrices $\mathbf{X}^{\text{sup}} \in \mathbb{R}^{d \times n_{\text{sup}}}$, $\mathbf{Z}^{\text{sup}} \in \mathbb{R}^{m \times n_{\text{sup}}}$, and $\mathbf{X}^{\text{unsup}} \in \mathbb{R}^{d \times n_{\text{unsup}}}$, respectively.

Since in the ideal setting of this subsection we have true samples of the latent vectors $\mathbf{z}$ that correspond to data points $\mathbf{x}$, this means that we know the true latent space dimension m. Hence, we can control the parameterization of the learned model by choosing a latent dimension $k \leq m$ via subsampling of coordinates in $\mathbf{z}$. Namely, for a set of $k \leq m$ unique coordinate indices $\mathcal{S} \subset \{1, \dots, m\}$, we define $\{\mathbf{z}_{i,\mathcal{S}}\}_{i=1}^{n_{\text{sup}}}$ as the corresponding subvectors of the training data $\{\mathbf{z}_i\}_{i=1}^{n_{\text{sup}}}$. The matrix $\mathbf{Z}_{\mathcal{S}}^{\text{sup}} \in \mathbb{R}^{k \times n_{\text{sup}}}$ has the subsampled vectors $\{\mathbf{z}_{i,\mathcal{S}}\}_{i=1}^{n_{\text{sup}}}$ as its columns.

To train our model, we use a PCA loss term from (2) on the unsupervised portion of the data $\mathbf{X}^{\text{unsup}}$ mixed with a supervised loss term on the supervised portion of the data $\mathbf{Z}^{\text{sup}}, \mathbf{X}^{\text{sup}}$:

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{1}{n_{\text{sup}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{sup}} - \mathbf{X}^{\text{sup}}\|_F^2 + \frac{1}{n_{\text{unsup}}} \|(\mathbf{I}_d - \mathbf{GG}^\top)\mathbf{X}^{\text{unsup}}\|_F^2 \tag{3}$$

for a generator matrix $\mathbf{G} \in \mathbb{R}^{d \times k}$ (that is not explicitly constrained to have orthonormal columns). Unlike the PCA optimization in (2), the optimizations in the current and following subsections do not include any explicit orthonormal constraints on the columns of the learned matrix $\mathbf{G}$. Here, the supervised portion of the data gives us specific information about $\mathbf{\Gamma}$, which we can use to train a better model. This model is trained by minimizing the loss in Equation (3) with gradient descent.

Figure 2 shows that our model exhibits double descent; however, there are a few concerns with the setup. First, we may not have access to the true latent vectors. Second, we can only vary k from 1 to m because we are subsampling coordinates from the true latent vectors in $\mathbb{R}^m$. Third, because $k < m$, we must have $n < m$ to get a peak at $k = n$ which is not the interesting setting where $m < n < d$ (see Section 3.1); thus the double descent here actually does not improve performance at all. This is because we are not overfitting, which happens when we learn eigenvectors in the noise directions orthogonal to the data directions, for which $k > m$. We resolve all these problems with pseudo-supervision, defined in the next section.

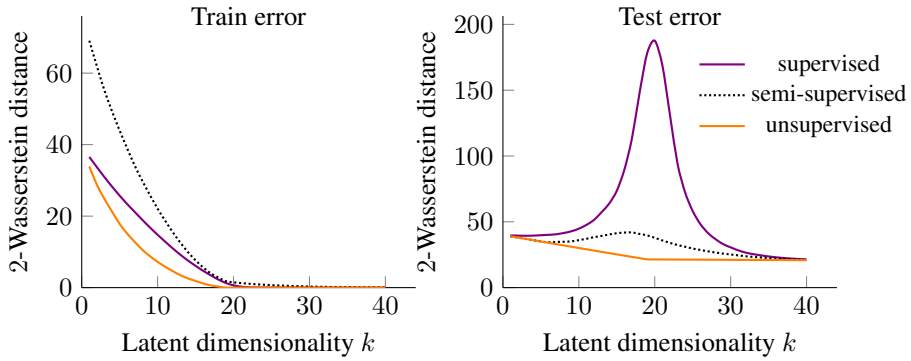

Figure 2: The fully supervised model achieves a peak when the latent dimensionality k is equal to the number of training samples n. The unsupervised model stops changing as soon as it interpolates at $k = n$. The semi-supervised model with $n_{\text{sup}} = 12$ behaves in a way that is somewhat in-between the other two. For other values of n_{sup} and implementation details, see Appendix C.

4 Pseudo-supervision: A practical alternative to supervised GAN training

4.1 Definition of pseudo-supervision

Input-output pairs of points are not realistically available in GAN training, which is unsupervised. Therefore, we will make up latent vectors that correspond to true data points in our training set. Although it may seem odd to partially fabricate training data, there are many advantages to it, starting with not needing access to supervised data. Additionally, we do not need to know the true latent dimensionality m and can study when $k > m$.

To understand why pseudo-supervision works, consider the supervised scenario discussed in Section 3.3 except with only one supervised sample: $(\mathbf{z}_1, \mathbf{x}_1)$. Now suppose that $\mathbf{z}_{\text{ps}} \in \mathbb{R}^m$ is a completely fabricated sample, independent of $\mathbf{x}_1$. As discussed in Section 3.2, we know that if $\mathbf{G}^{\text{unsup}}$ is the solution to the unsupervised optimization, then so is $\mathbf{G}^{\text{unsup}}\mathbf{U}$ where $\mathbf{U}\mathbf{z}_{\text{ps}} = \mathbf{z}_1$ and $\mathbf{U}$ is an orthonormal matrix. Such a matrix exists if $\|\mathbf{z}_{\text{ps}}\|_2 = \|\mathbf{z}_1\|_2$ because $\mathbf{U}$ is a norm preserving operator. In other words, it doesn't matter if we use $\mathbf{z}_1$ or $\mathbf{z}_{\text{ps}}$ as long as $\|\mathbf{z}_{\text{ps}}\|_2 = \|\mathbf{z}_1\|_2$. Miraculously, by the curse of dimensionality, $\|\mathbf{z}_{\text{ps}}\|_2 = \|\mathbf{z}_1\|_2$ with high probability if d is large enough! This line of reasoning can be extended past one pseudo-supervised example $n_{\text{ps}} = 1$ to $n_{\text{ps}} = k$ (see Appendix B), after which we incur a penalty for learning a bad representation (because we cannot find an orthonormal matrix which will satisfy the conditions above). Therefore, because of positive definite matrix symmetries and the curse of dimensionality, **we can use pseudo-supervision in a very similar way to supervision without actually knowing any additional information.**

In the following subsections we will define several pseudo-supervised settings, in all of which n_{ps} out of the n given training examples $\{\mathbf{x}_i\}_{i=1}^n$ are associated with pseudo (i.e., artificial) latent vectors of dimension $k > 0$ (because the true latent dimension is unknown in general). Specifically, the training dataset $\mathcal{D}$ includes $n_{\text{ps}} \in \{0, \dots, n\}$ pseudo-supervised examples $\{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^{n_{\text{ps}}}$, where $\{\mathbf{z}_i\}_{i=1}^{n_{\text{ps}}}$ are i.i.d. samples of $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, and $n_{\text{unsup}} = n - n_{\text{ps}}$ unsupervised examples $\{\mathbf{x}_i\}_{i=n_{\text{ps}}+1}^{n}$. The training data vectors are organized as the columns of $\mathbf{X}^{\text{ps}} \in \mathbb{R}^{d \times n_{\text{ps}}}$, $\mathbf{Z}^{\text{ps}} \in \mathbb{R}^{k \times n_{\text{ps}}}$, and $\mathbf{X}^{\text{unsup}} \in \mathbb{R}^{d \times n_{\text{unsup}}}$.

4.2 Double descent and superior performance with pseudo-supervision

Our first pseudo-supervised experiment is a straightforward modification of the experiment in Section 3.3. We modify Equation (3) to get the new pseudo-supervised loss

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{1}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}^{\text{ps}}_{\mathcal{S}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1}{n_{\text{unsup}}} \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{X}^{\text{unsup}}\|_F^2, \tag{4}$$

where $\mathbf{X}^{\text{ps}} \in \mathbb{R}^{d \times n_{\text{ps}}}$ and $\mathbf{Z}^{\text{ps}} \in \mathbb{R}^{k \times n_{\text{ps}}}$ are the pseudo-supervised matrices. For $n_{\text{unsup}} = 0$ or $n_{\text{ps}} = 0$, we only use the first or second term in the loss, respectively. We provide a detailed explanation of the gradient calculations and optimization procedure in Appendix C. Note that since the pseudo-supervised latent vectors are completely fabricated, we do not have to subsample their

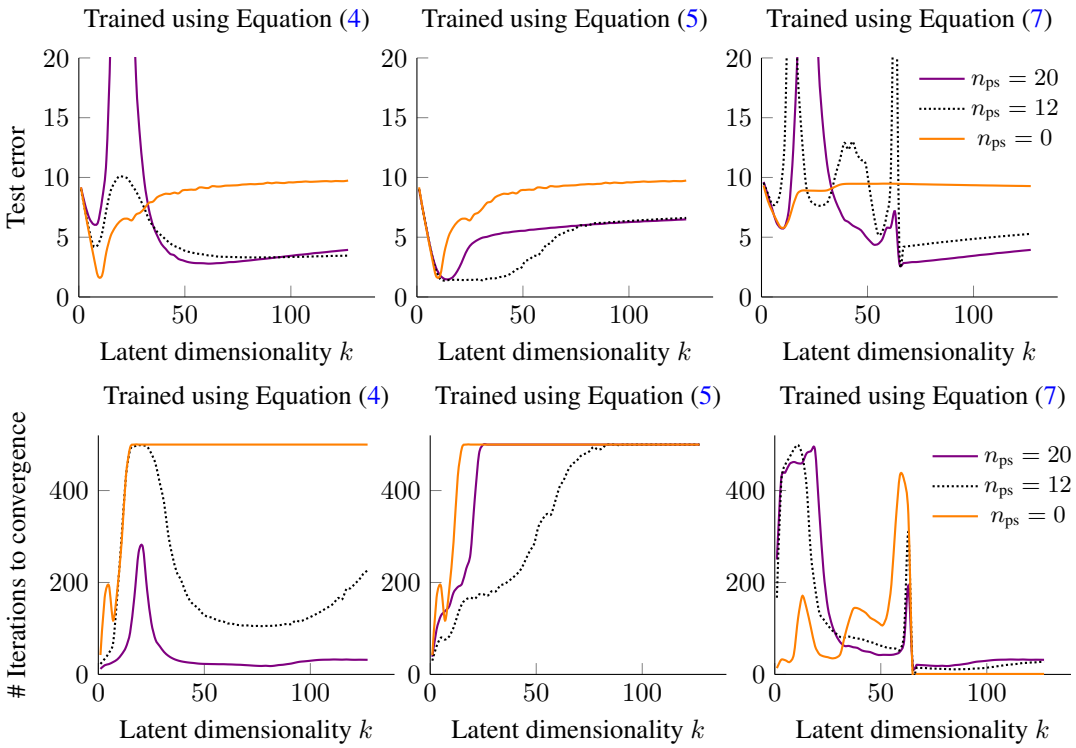

Figure 3: Evaluation of test errors and training convergence speed in learning of linear GANs using the three different training loss formulations in (4),(5),(7). In the first column of subfigures, we use (4) and get double descent that beats the baseline in both generalization performance and convergence speed in the overparameterized range of solutions (the baseline corresponds to the case of no pseudo-supervised training samples $n_{\text{ps}} = 0$). In the second column of subfigures, we use (5) and squash the double descent to get lower generalization error for small latent dimensionality k. In the third column of subfigures, we get triple descent (one peak at $k = n$ and one peak at $k = d$) as well as low generalization errors and extremely fast training speed for large k. In these experiments, the true data is $m = 10$ dimensional, the data space is $d = 64$ dimensional, and we have $n = 20$ total training data samples. The null estimator ($\mathbf{G} = \mathbf{0}_{d\times k}$) achieves an error of approximately 13, so all of these models perform better for large enough k. For additional plots, see Appendix C.

coordinates (i.e., as in the supervised setting of Section 3.3) and we can choose k to be any natural number. As shown in the first column of Figure 3, we achieve a beneficial double descent behavior of test errors. To the best of our knowledge, **this is the first time that double descent has been used beneficially in an unsupervised setting.**

We have extremely low generalization error when $n_{\text{ps}} = n$, even though the loss function does not try to optimize any PCA type loss. When $n_{\text{ps}} = n$, we have $n_{\text{unsup}} = 0$ and the loss $\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \|\mathbf{G}\mathbf{Z}_S^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2$ is completely pseudo-supervised. One would expect this scenario to perform poorly since the pseudo-supervised examples do not provide any information, and indeed it does – for small k. However, when k is large, we perform well, even though the loss does not attempt to minimize the original PCA loss. Thus, instead of guessing the true latent dimension m that is required for good performance in the standard setting of Section 3.1, **we can simply add pseudo-supervision and increase overparameterization to achieve low generalization error!**

As can be seen from the first column of Figure 3, we achieve better generalization performance via double descent phenomena, and we also accelerate training convergence. The accelerated convergence may be partly due to the unsupervised loss dropping off when $n_{\text{ps}} = n$; however, we will address this in the next section by having a more regularized loss function.

4.3 Regularized pseudo-supervision

In the previous section, as n_{ps} increases, the unsupervised term in the loss drops off. This term, in some sense, regularizes the optimization by encouraging the solution to be orthonormal. This is because, if $\mathbf{G}$ has orthonormal columns, then $(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{x} = 0$ for all $\mathbf{x}$ in the columnspace of $\mathbf{G}$. We will then use the full data matrix $\mathbf{X}$ (which is a horizontal concatenation of $\mathbf{X}^{\text{ps}}$ and $\mathbf{X}^{\text{unsup}}$) in the second term of the loss function:

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{1}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1}{n} \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{X}\|_F^2. \tag{5}$$

The results for this optimization are shown in the second column of Figure 3.

This regularized setting with pseudo-supervision outperforms the completely unsupervised setting, but we do not get double descent. This is typical for more regularized problems, as regularization tends to attenuate the double descent phenomenon (see, e.g., for orthonormality constraints in [25], or for ridge regularization in [28, 29]). However, this suggests that the relative importance between the first and second term may significantly impact double descent behavior. More specifically, the only difference between this optimization and the one discussed in Section 4.2 is that the second term uses all the data even when $n_{\text{ps}} > 0$. Thus, we can think of the second term as a regularizer for the loss. On the other hand, we can view the first term as constraining the optimization to fit our pseudo-supervised pairs of points, and thus also a regularizer. Therefore, depending on the point of view, each term can regularize the loss.

Since either of the terms in the training loss in (5) can be perceived as a regularizer, we augment (5) with disproportionate weighting in order to see if this affects the generalization behavior (e.g., the existence of double descent phenomena):

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{\alpha}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1-\alpha}{n} \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{X}\|_F^2, \tag{6}$$

for $\alpha \in [0, 1]$. A figure of the results is shown in Appendix C. Surprisingly, weighting the loss function in this manner actually does achieve double descent which leads to lower test error. We discuss this model here in order to highlight that the relative importance between the pseudo-supervised and unsupervised loss terms can induce double descent behavior. Interestingly, convergence time actually also exhibits double descent, which suggests a compelling question: can overparameterization be used to accelerate training as well as improve generalization?

4.4 Triple descent and huge latent spaces

In the previous experiments we use a similar loss, which indirectly encourages learning orthonormal generator matrices. We can relax this constraint and let our generator learn more complex linear functions by optimizing

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{1}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1}{n} \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)\mathbf{X}\|_F^2, \tag{7}$$

where $\mathbf{G}^\dagger$ is the Moore-Penrose pseudo-inverse of the matrix $\mathbf{G}$. Training this loss may seem similar to the others, but the results are quite different.

With this new loss, we achieve triple descent and desirable generalization and convergence behavior when the latent dimensionality k is larger than the data space dimensionality d (third column of Figure 3). This scenario is most closely related to neural networks because the models that we learn are very general and typically not constrained (e.g., to have orthonormal layers). Moreover, the pseudo-supervised optimization converges to a solution which beats the baseline with few iterations.

5 Nonlinear GANs: Double descent and faster training

In this section we show that double descent can occur in nonlinear, multilayer GANs trained with pseudo-supervision. Finding the right experimental setting for double descent was difficult because the level of parameterization is much harder to quantify in a multilayer network. We still determined the overparameterization solely by modifying the latent dimensionality k and not by making the

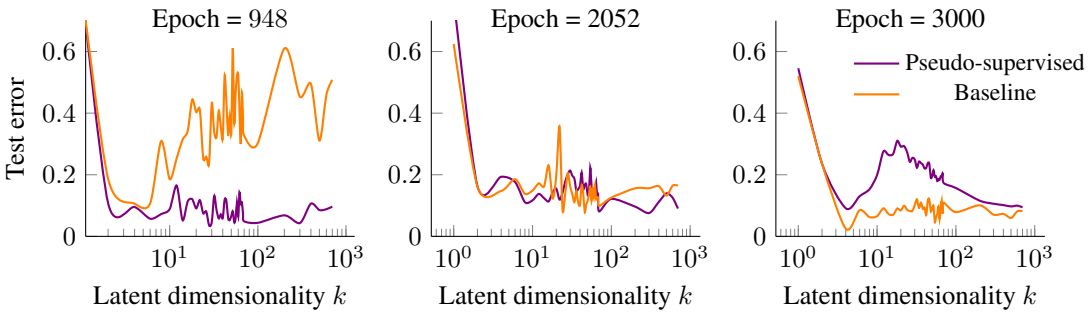

Figure 4: Test errors for multilayer, nonlinear GANs and MNIST digit dataset. On the left we see that the baseline error resembles a noisy version of the test error in Figure 1, characterized by an initial dip and then high levels of error. Our pseudo-supervision training beats the baseline here. As we continue to train (epoch 2052), we see that the baseline error reduces, which may be due to some kind of implicit regularization. On the right, our pseudo-supervised model achieves double descent at epoch 3000. Here the test error is measured by geometry score.

networks wider or deeper. The right side of Figure 4 shows double descent for our pseudo-supervised model. We trained a total of 430 GANs (with different latent dimensionalities and initializations) to make that figure, which is why a study like this would be computationally prohibitive on models that take a significant amount of time to train.

We also found that these realistic GANs trained with pseudo-supervision converge to a good solution much faster than they would have without the pseudo-supervision. Figures 4 and 5 show the test errors as training progressed for different latent dimensionalities. The pseudo-supervised models converge much faster and performed very well. They converged to the lowest test error after only about 750 epochs compared to about 1,500 epochs in the baseline case.

The test error in Figure 4 for the baseline had an initial dip then continued up to high levels around epoch 948, suggesting overfitting similar to what we saw in the linear models. We suspect that this overfitting was reduced as we continued to train because of some internal regularization, such as the batch norm in the model.

We performed these experiments with some non-standard procedures to aid in our understanding of generalization and double descent phenomena in GANs. In this work, we are not concerned with training state-of-the-art GANs. For this reason, our experiments are on MNIST [30]. Since MNIST is not very complex, we only use a random subset of 4,096 training data points and perform gradient descent using a gradient penalized Wasserstein GAN[1] (for SGD results, see Appendix D) . Commonly used performance metrics such as FID [21] and IS [16] are made for natural images since they use the Inception v3 [31] model trained on ILSVRC 2012 [32]. Therefore, we use the geometry score [33], which is better suited for MNIST[2]. See Appendix D for more details on the training.

6 Conclusion

We have demonstrated that pseudo-supervision can be used to achieve beneficial double descent phenomena in unsupervised models, specifically in linear GANs and nonlinear, multilayer GANs. Pseudo-supervision can help accelerate training and lower generalization error. This opens up areas of research in understanding overparameterization and double descent behavior in unsupervised models. Moreover, our findings suggest that an empirical study on ImageNet with more complex networks is beneficial to improve state-of-the-art generalization error and convergence speed.

[1]The architecture implementation can be found here under an MIT license
[2]The geometry score implementation can be found here

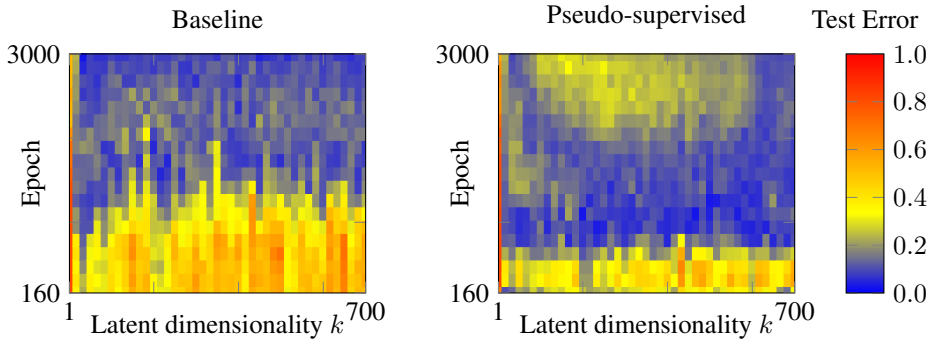

Figure 5: These test error heatmaps for multilayer, nonlinear GANs show that the pseudo-supervised models converge faster than the baseline models. The baseline model has high test error until around epoch 1500, unlike the pseudo-supervised models which have the test error drop off at around epoch 750. The baseline model only beats the pseudo-supervised model later in the training (around epoch 2500), when the pseudo-supervised loss increases and admits a double descent shape. The test error is measured by geometry score here. The k-axis is plotted so that each column corresponds to the next entry for better visualization, even though the spacing is $k \in \{1, 2, 4, 6, \ldots, 70, 100, 200, 300, \ldots, 700\}$.

Acknowledgements

This work was supported by NSF grants CCF-1911094, IIS-1838177, CCF-1730574, and 1842494; ONR grants N00014-18-12571, N00014-20-1-2787, and N00014-20-1-2534; AFOSR grant FA9550-18-1-0478; and a Vannevar Bush Faculty Fellowship, ONR grant N00014-18-1-2047.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[2] M. Belkin, D. Hsu, and J. Xu. Two models of double descent for weak features. *arXiv preprint arXiv:1903.07571*, 2019.

[3] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 2020.

[4] V. Muthukumar, K. Vodrahalli, V. Subramanian, and A. Sahai. Harmless interpolation of noisy data in regression. *IEEE Journal on Selected Areas in Information Theory*, 2020.

[5] S. d'Ascoli, M. Refinetti, G. Biroli, and F. Krzakala. Double trouble in double descent: Bias and variance(s) in the lazy regime. In *International Conference on Machine Learning*, pages 2280–2290. PMLR, 2020.

[6] V. Muthukumar, A. Narang, V. Subramanian, M. Belkin, D. Hsu, and A. Sahai. Classification vs regression in overparameterized regimes: Does the loss function matter? *arXiv preprint arXiv:2005.08054*, 2020.

[7] Z. Deng, A. Kammoun, and C. Thrampoulidis. A model of double descent for high-dimensional binary linear classification. *arXiv preprint arXiv:1911.05822*, 2019.

[8] G. R. Kini and C. Thrampoulidis. Analytic study of double descent in binary classification: The impact of loss. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 2527–2532, 2020.

[9] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*, 2017.

[10] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[11] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[12] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of StyleGAN. *arXiv preprint arXiv:1912.04958*, 2019.

[13] S. Nowozin, B. Cseke, and R. Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. *arXiv preprint arXiv:1606.00709*, 2016.

[14] A. Sarraf and Y. Nie. RGAN: Rényi generative adversarial network. *SN Computer Science*, 2 (1):1–8, 2021.

[15] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training GANs. *arXiv preprint arXiv:1606.03498*, 2016.

[17] L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for GANs do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.

[18] S. Feizi, F. Farnia, T. Ginart, and D. Tse. Understanding GANs in the LQG setting: Formulation, generalization and stability. *IEEE Journal on Selected Areas in Information Theory*, 2020.

[19] C. R. Givens, R. M. Shortt, et al. A class of Wasserstein metrics for probability distributions. *The Michigan Mathematical Journal*, 31(2):231–240, 1984.

[20] I. Olkin and F. Pukelsheim. The distance between two random vectors with given dispersion matrices. *Linear Algebra and its Applications*, 48:257–263, 1982.

[21] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6626–6637, 2017.

[22] S. Spigler, M. Geiger, S. d'Ascoli, L. Sagun, G. Biroli, and M. Wyart. A jamming transition from under-to over-parametrization affects loss landscape and generalization. *arXiv preprint arXiv:1810.09665*, 2018.

[23] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116 (32):15849–15854, 2019.

[24] Y. Balaji, M. Sajedi, N. M. Kalibhat, M. Ding, D. Stöger, M. Soltanolkotabi, and S. Feizi. Understanding overparameterization in generative adversarial networks. *arXiv preprint arXiv:2104.05605*, 2021.

[25] Y. Dar, P. Mayer, L. Luzi, and R. Baraniuk. Subspace fitting meets regression: The effects of supervision and orthonormality constraints on double descent of generalization errors. In *International Conference on Machine Learning*, pages 2366–2375. PMLR, 2020.

[26] Y. Dar and R. G. Baraniuk. Double double descent: On generalization errors in transfer learning between linear regression tasks. *arXiv preprint arXiv:2006.07002*, 2020.

[27] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[28] T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *arXiv preprint arXiv:1903.08560*, 2019.

[29] P. Nakkiran, P. Venkat, S. Kakade, and T. Ma. Optimal regularization can mitigate double descent. *arXiv preprint arXiv:2003.01897*, 2020.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[33] V. Khrulkov and I. Oseledets. Geometry score: A method for comparing generative adversarial networks. In *International Conference on Machine Learning*, pages 2621–2629. PMLR, 2018.

[34] W. Rudin. *Principles of mathematical analysis*. McGraw-hill New York, third edition, 1964.

[35] A. Rényi et al. On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*. The Regents of the University of California, 1961.

[36] C. Villani. *Topics in optimal transportation*. American Mathematical Soc., 2003.

[37] C. Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[38] H. L. Royden and P. Fitzpatrick. *Real analysis*, volume 32. Macmillan New York, 1988.

[39] S. Axler. *Measure, Integration & Real Analysis*. Springer Nature, 2020.

[40] W. Rudin. Real and complex analysis. 1987. *Cited on*, 156, 1987.

[41] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[42] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[43] J. E. Gentle. Matrix algebra. *Springer texts in statistics, Springer, New York, NY, doi*, 10:978–0, 2007.

[44] K. Petersen and M. Pedersen. The matrix cookbook, version 20121115. *Technical Univ. Denmark, Kongens Lyngby, Denmark, Tech. Rep*, 3274, 2012.

Appendices

The appendices below support the main paper as follows. Appendix A provides additional details on how subsampling (or zeroing) coordinates of the data is equivalent to training with a pseudometric as discussed in Section 3.2 of the main paper. In Appendix B we expand on pseudo-supervision and explain when it can be used to mimic supervision. Appendix C includes additional empirical results and details for the linear GAN problems from Sections 3.3 and 4.2 to 4.4 of the main paper. Appendix D provides additional experimental results and details for the multilayer, nonlinear GAN from Section 5 of the main paper; we also include results for SGD-based training of GANs using the complete MNIST dataset.

A Training with pseudometric and subsampling data

A.1 Subsampling the data features

In Section 3.1 we saw that if we optimize an objective function which is a metric [34] or an f-divergence [35, 13], the resulting generalization error will be constant for any interpolating solution. This is due to the definiteness of the metric or f-divergence. In this section we will relax this property for the 2-Wasserstein metric [36, 37]; extensions to this relaxation can be done for f-divergences and other metrics. The resulting mathematical object is called a pseudometric [38], which has been studied thoroughly in the context of L^p metrics in Banach spaces [39, 38].

Definition 1. We denote q_d to be the standard Euclidean metric on $\mathbb{R}^d$ [34]. Let $P(\mathbb{R}^d)$ be the set of all probability distributions defined on the measurable space $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$, where $\mathcal{B}(\mathbb{R}^d)$ is the Borel σ-algebra on $\mathbb{R}^d$ [39, 40]. We denote $\mathcal{W}_d : P(\mathbb{R}^d) \times P(\mathbb{R}^d) \to \mathbb{R}$ to be the 2-Wasserstein metric:

$$\mathcal{W}_d(P, P') = \sqrt{\inf_{\gamma \in \Pi(P,P')} \int_{\mathbb{R}^d \times \mathbb{R}^d} q_d^2(\mathbf{x}, \mathbf{y}) d\gamma(\mathbf{x}, \mathbf{y})},$$

where $\gamma \in \Pi(P, P')$ is any joint distribution of P and P'. For a set $A \subset \{1, \ldots, d\}$, we define the pseudometric $\mathcal{W}_{d,A} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to be the 2-Wasserstein metric on $\mathbb{R}^{d-|A|}$ on the indices not in A. For example, if $P_{2,\ldots,d}, P'_{2,\ldots,d}$ are the marginals (after integrating out the first component) of P and P', respectively, then

$$\mathcal{W}_{d,\{1\}}(P, P') \coloneqq \mathcal{W}_{d-1}(P_{2,\ldots,d}, P'_{2,\ldots,d}).$$

Clearly, $\mathcal{W}_{d,A}$ is a pseudometric as it derives all metric properties from $\mathcal{W}_{d-|A|}$ except the definiteness property.

This pseudometric is constructed by integrating out certain coordinates of the distributions and using a metric on the resulting marginal distributions. Therefore it is possible to have zero distance between two distributions that differ along the coordinates which are integrated out. This is equivalent to subsampling or zeroing out the desired coordinates, which we will shortly show. Thus, for the linear case, we can learn a generator $\mathbf{G}$ which maps our latent space to $\mathbb{R}^d$ and which learns the training data distributions p_f except for the ignored coordinates. Of course, now we have a whole (affine) subspace of matrices $\mathbf{G} \in \mathbb{R}^{d \times k}$ that we can learn. In other words, using a pseudometric, an interpolating solution $\mathbf{G} \in \mathbb{R}^{d \times k}$ forms an affine subspace of $\mathbb{R}^{d \times k}$ if modified along the ignored coordinates. As we will see in Appendix B, we can also transform $\mathbf{G}$ by an orthonormal transformation to get more degrees of freedom than just this affine space. In this setting, the min-norm solution will not project anything on the ignored coordinates.

Theorem 2. *Let P and P' be two distributions defined on $\mathbb{R}^d$. Let $A \subset \{1, \ldots, d\}$ be a subset of the axis indices. We define a new distribution Q_A on $\mathbb{R}^d$ as the product of $|A|$ univariate point masses at 0 and the marginal distribution P_{A^C}. The point masses are located so that the univariate marginals of Q_A are point masses along the coordinates in A. We define Q'_A similarly. Then,*

$$\mathcal{W}_{d,A}(P, P') \coloneqq \mathcal{W}_{d-|A|}(P_{A^C}, P'_{A^C}) = \mathcal{W}_d(Q_A, Q'_A). \tag{8}$$

Proof. An application of Tonelli's Theorem [39] shows that

$$\begin{aligned}
\mathcal{W}_d(Q_A, Q'_A) &= \sqrt{\inf_{\gamma} \int_{\mathbb{R}^d \times \mathbb{R}^d} q_d^2(\mathbf{x}, \mathbf{y}) d\gamma} \\
&= \sqrt{\inf_{\gamma} \sum_{i=1}^{d} \int_{\mathbb{R}^d \times \mathbb{R}^d} |x_i - y_i|^2 d\gamma} \\
&= \sqrt{\inf_{\gamma} \sum_{i \in A} \int_{\mathbb{R}^d \times \mathbb{R}^d} |x_i - y_i|^2 d\gamma + \sum_{i \in A^C} \int_{\mathbb{R}^d \times \mathbb{R}^d} |x_i - y_i|^2 d\gamma} \\
&= \sqrt{\inf_{\gamma} \int_{\mathbb{R}^{2|A|}} \sum_{i \in A} |x_i - y_i|^2 d\gamma_A + \int_{\mathbb{R}^{2(d-|A|)}} \sum_{i \in A^C} |x_i - y_i|^2 d\gamma_{A^C}} \quad \text{(Tonelli)} \\
&= \sqrt{\inf_{\gamma_{A^C}} \int_{\mathbb{R}^{2(d-|A|)}} \sum_{i \in A^C} |x_i - y_i|^2 d\gamma_{A^C}} \quad (*) \\
&= \mathcal{W}_{d-|A|}(P_{A^C}, P'_{A^C}) \\
&= \mathcal{W}_{d,A}(P, P'),
\end{aligned}$$

where γ_A and γ_{A^C} are the joints of the marginals over A and over A^C, respectively. We also use independence when using Tonelli's Theorem, because Q_A and Q'_A are product measures by construction. In $(*)$, we pick γ_A to be the independent joint distribution so that each random variable with index in A is independent. Since each of these random variables is identical, the integral term on the left vanishes and is therefore the minimizer of the infimum. □

Theorem 2 shows that we can train with a pseudometric by simply zeroing the coordinates of the data that we wish to ignore; alternatively, we can also subsample the features so that we keep the features with indices in A^C. This allows us to consider a pseudometric $\mathcal{W}_{d,A}$ which is invariant to the data features with indices in A. Suppose that we instead want $\mathcal{W}_{d,A}$ to be invariant to a specific subspace. It turns out that these two concepts are closely related.

Theorem 3. *Let $V \subset \mathbb{R}^d$ be a subspace spanned by the orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$; the rest of $\mathbb{R}^d$ is spanned by $\mathbf{v}_{m+1}, \ldots, \mathbf{v}_d$ so that $\{\mathbf{v}_i\}_{i=1}^d$ is an orthonormal basis for $\mathbb{R}^d$. We also have a data matrix $\mathbf{X} \in \mathbb{R}^{d \times n}$. Then, we can construct a pseudometric $\mathcal{W}_{d,V}$ to be invariant to the subspace V by replacing the first m rows of $\mathbf{U}^\top \mathbf{X}$ with zeros for $\mathbf{U} = [\mathbf{v}_1 \quad \ldots \quad \mathbf{v}_d] \in \mathbb{R}^{d \times d}$.*

Proof. Let $\mathbf{v} \in V$ be given. Then, we can write $\mathbf{v} = \sum_{i=1}^m c_i \mathbf{v}_i$. Clearly, we have that $\mathbf{U}^\top \mathbf{v} = \sum_{i=1}^m c_i \mathbf{U}^\top \mathbf{v}_i = [c_1 \quad \ldots \quad c_m \quad 0 \quad \ldots \quad 0]^\top$. Similarly, if $\mathbf{w} \in V$ is arbitrary, then we have that $\mathbf{U}^\top \mathbf{w} = [a_1 \quad \ldots \quad a_m \quad a_{m+1} \quad \ldots \quad a_d]^\top$ for some numbers $a_i \in \mathbb{R}$. Hence, by replacing the first m coordinates by 0 we project onto the subspace orthogonal to V. Applying $\mathbf{U}^\top$ to each column of $\mathbf{X}$ is equivalent to computing $\mathbf{U}^\top \mathbf{X}$. □

Thus, without loss of generality, we consider only subsampling feature indices. If we want to ignore a subspace, we simply multiply our data matrix by the correct matrix $\mathbf{U}$.

A.2 Subsampling the latent vector coordinates

In the previous section, we considered subsampling the data features. However, we know that supervision has enabled double descent in PCA-type problems [25]. Thus, we would like to study supervision in the GAN context, as discussed in Section 3.3. In a supervised linear regression setting using the 2-norm loss, we know that we must take a pseudoinverse of the input matrix [41], which induces double descent. In this setting, that is the latent space matrix $\mathbf{Z}$. Therefore, we enable double descent by subsampling the latent vector coordinates. Doing this is very similar to subsampling the features in the data space. For example, if we zero out the first coordinate of the latent distribution, we are essentially zeroing out the subspace corresponding to the first column of the matrix $\mathbf{G}$. Since we learn $\mathbf{G}$, this is a type of adaptive pseudometric procedure, where we learn which subspaces to use and which subspaces to ignore.

B Pseudo-supervision and the curse of dimensionality

This appendix provides further detail regarding the scenario described in Section 4.1. Suppose that $\mathbf{G} \in \mathbb{R}^{d \times m}$ is a solution which provides zero test error. Now, let $\mathbf{z} \in \mathbb{R}^m$ correspond to the true vector which generates $\mathbf{x} \in \mathbb{R}^d$ so that $\mathbf{G}\mathbf{z} = \mathbf{x}$. Now suppose that $\mathbf{z}_{\text{ps}} \in \mathbb{R}^m$ is any vector so that $\|\mathbf{z}_{\text{ps}}\|_2 = \|\mathbf{z}\|_2$. Then, we can find an orthonormal matrix $\mathbf{U} \in \mathbb{R}^{m \times m}$ so that $\mathbf{U}\mathbf{z}_{\text{ps}} = \mathbf{z}$. We see that $\mathbf{G}\mathbf{U}$ is also a solution which gives zero test error, because the covariance matrix of the generated distribution is not changed if we right multiply $\mathbf{G}$ with an orthonormal matrix [27]. However, if we pick $\mathbf{z}_{\text{ps}}$ from $\mathcal{N}(0, \mathbf{I}_m)$ where m is large, we see that $\|\mathbf{z}_{\text{ps}}\|_2 = \|\mathbf{z}\|_2$ with high probability because high dimensional Gaussians concentrate on a thin shell in high-dimensional space [42]. This is typically considered a bad thing, hence its name: the curse of dimensionality. However, here we use the curse of dimensionality to allow fabricated latent vectors $\mathbf{z}_{\text{ps}}$ to mimic supervised latent vectors $\mathbf{z}$. Moreover, we can come up with linearly independent pseudo-supervised latent vectors up to m times, after which we can no longer find an orthonormal matrix $\mathbf{U}$. The more pseudo-supervised samples we have, the fewer matrices $\mathbf{G}$ we can learn, resulting in faster gradient descent convergence since the feasible set is smaller.

We will encounter a problem if $k < m$, i.e., if the latent dimension we pick is lower than the true latent dimension, because we cannot learn a perfect representation (assuming that the linear operator $\mathbf{\Gamma}$ in the data model is full rank). However, if we let k be larger than m, then we can learn a solution which gives us zero test error. Although the true vectors are m-dimensional, we can always learn a generator matrix $\mathbf{G}$ which ignores certain coordinates. For such solutions, we can also construct pseudo-supervised samples up to k times. Therefore the overparameterized regime, where k is large, is very desirable from the pseudo-supervised point of view.

If we fix n_{ps} to some value, note that by the above argument, we will incur a penalty if $k < n_{\text{ps}}$ because we will not be able to find a suitable $\mathbf{U}$. However, if k is larger than m and larger than n_{ps}, we can mimic the behavior of supervised samples because we will be able to find an orthonormal matrix which will transform those pseudo-supervised latent vectors into vectors that equal the true vectors along m coordinates. For this reason, we consider pseudo-supervision when k is large.

C Experiments on linear models and gradient details

C.1 Details regarding linear experiments

In the linear setting, we set $\mathbf{\Gamma} \in \mathbb{R}^{d \times m}$ to be the first $m = 10$ columns of a Hadamard matrix multiplied by $\frac{1}{\sqrt{d}}$, where $d = 64$. Trails using random orthonormal columns for $\mathbf{\Gamma}$ yielded extremely similar results, therefore we only show plots for the Hadamard $\mathbf{\Gamma}$. Then, we create our data by drawing $n = 20$ samples from $\mathbf{\Gamma}\mathbf{z} + \boldsymbol{\epsilon}$, where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_m)$ and $\boldsymbol{\epsilon} = \mathcal{N}(\mathbf{0}, 0.15^2\mathbf{I}_d)$. Our initial matrix $\mathbf{G} \in \mathbb{R}^{d \times k}$ is drawn from an isotropic Gaussian with 0.03 standard deviation. We have $k \in \{1, 3, 5, \ldots, 127\}$ for the pseudo-supervised experiments and $k \in \{1, 2, \ldots, 40\}$ for the supervised experiments. For all these experiments, we have n_{ps} and n_{sup} take values in $\{0, 2, 4, 12, 18, 20\}$.

We perform gradient descent with a maximum of 500 iterations. The initial step size is 0.0001 after which we adaptively pick the current iteration's step size which will reduce the training loss most. We do this by multiplying the current step size by values in $\{0.0000001, 0.000005, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$ and picking the value which will yield the lowest training loss. If the matrix $\mathbf{G}$ does not change more than 0.00001 in Frobenius norm for more than 5 iterations, then the optimization also stops. If the Frobenius norm of the gradient is less than 0.05, then the optimization stops. The gradients are calculated in Appendix C.2.

We run all these experiments 200 times and average the results. For each experiment, we pick a new seed and re-run the same script. Therefore, the pseudo-supervised examples are fixed for each experiment as we vary k and n_{ps}. Hence, the errorbars in Figures 6 to 10 show one standard deviation of how the choice of matrix initialization, pseudo-supervision samples, and data samples all affect the test error.

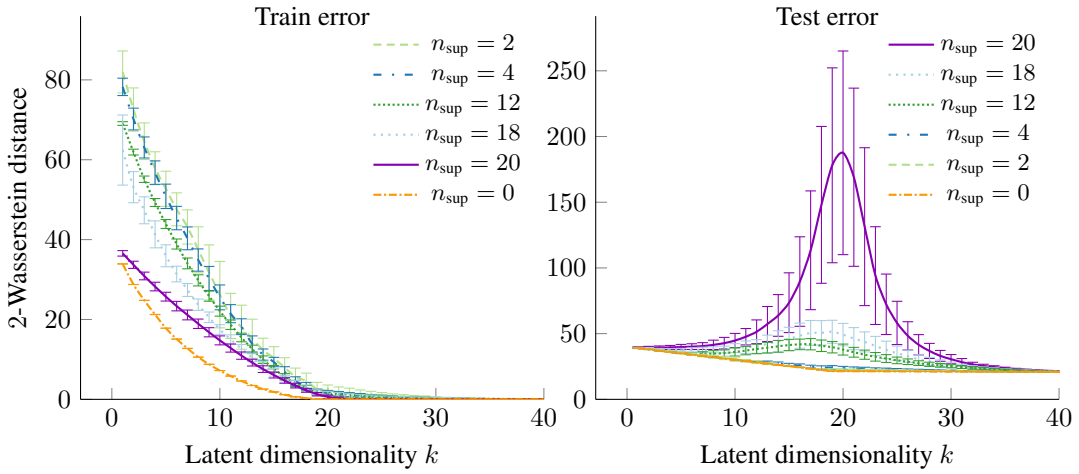

Figure 6: In this figure, we minimize the loss in Equation (3). The legends are displayed in the same order as the curves appear on the plot for clarity. This figure is a more detailed version of Figure 2.

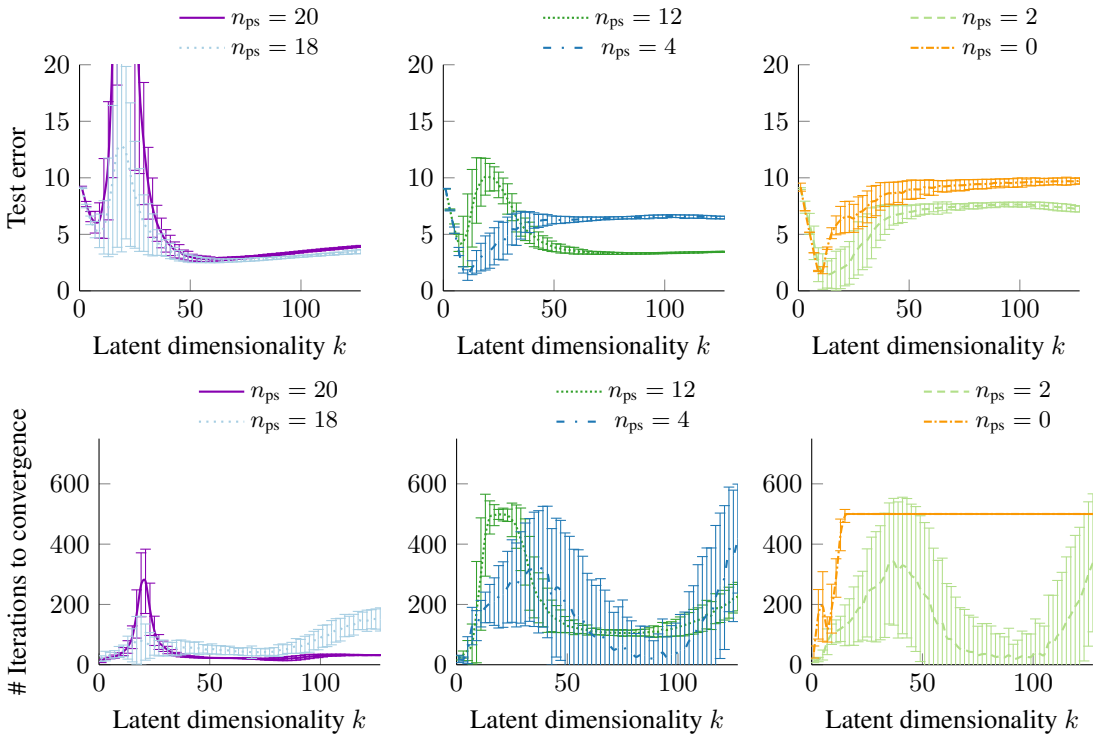

Figure 7: In this figure, we minimize the loss in Equation (4). This figure is a more detailed version of the first column of Figure 3. We show results for six different pseudo-supervision levels (i.e., n_{ps} values). For visual clarity, each subfigure includes results for only two n_{ps} values.

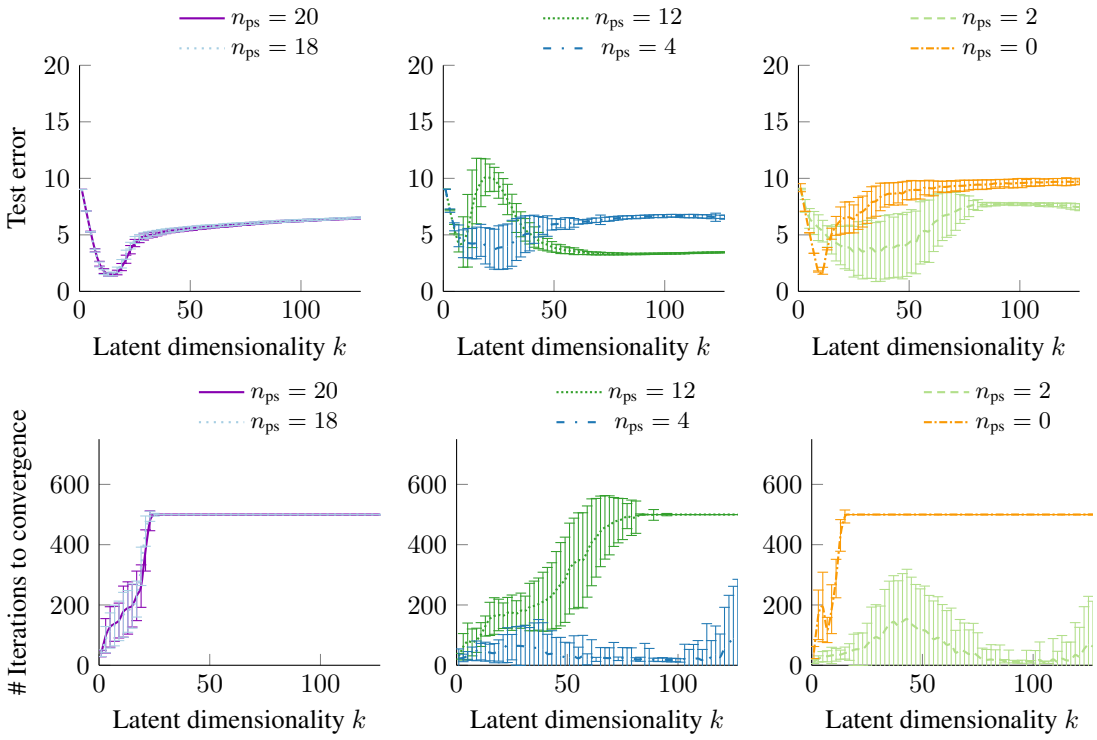

Figure 8: In this figure, we minimize the loss in Equation (5). This figure is a more detailed version of the center column of Figure 3. We show results for six different pseudo-supervision levels (i.e., n_{ps} values). For visual clarity, each subfigure includes results for only two n_{ps} values.

C.2 Gradient calculations

The losses introduced in Equations (4) to (6) all have similar forms, so we only show what is the gradient for Equation (4) and the other ones are easily obtained. For completeness, we restate the loss:

$$\mathcal{L}^{\text{train}}(\mathbf{G}, \mathcal{D}) = \frac{1}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1}{n_{\text{unsup}}} \|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\top)\mathbf{X}^{\text{unsup}}\|_F^2.$$

The gradient of the first term is

$$\begin{aligned}
\nabla_{\mathbf{G}} \frac{1}{n_{\text{ps}}} \|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 &= \frac{1}{n_{\text{ps}}} \nabla_{\mathbf{G}} \|(\mathbf{Z}_{\mathcal{S}}^{\text{ps}})^\top \mathbf{G}^\top - (\mathbf{X}^{\text{ps}})^\top\|_F^2 && \text{(Frobenius transpose invariance)} \\
&= \frac{1}{n_{\text{ps}}} \left(\nabla_{\mathbf{G}^\top} \|(\mathbf{Z}_{\mathcal{S}}^{\text{ps}})^\top \mathbf{G}^\top - (\mathbf{X}^{\text{ps}})^\top\|_F^2 \right)^\top && \text{(Section 4.2.3 of [43])} \\
&= \frac{1}{n_{\text{ps}}} \left(2\mathbf{Z}_{\mathcal{S}}^{\text{ps}} \left((\mathbf{Z}_{\mathcal{S}}^{\text{ps}})^\top \mathbf{G}^\top - (\mathbf{X}^{\text{ps}})^\top \right) \right)^\top \\
&= \frac{2}{n_{\text{ps}}} (\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}})(\mathbf{Z}_{\mathcal{S}}^{\text{ps}})^\top.
\end{aligned}$$

The gradient of the second term in the considered loss function is a bit more tricky. We simplify it first to get

$$\begin{aligned}
\|(\mathbf{I}_p - \mathbf{G}\mathbf{G}^\top)\mathbf{X}_{\mathcal{S}}^{\text{unsup}}\|_F^2 &= \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top (\mathbf{I}_p - \mathbf{G}\mathbf{G}^\top)(\mathbf{I}_p - \mathbf{G}\mathbf{G}^\top)\mathbf{X}_{\mathcal{S}}^{\text{unsup}}) \\
&= \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top (\mathbf{I}_p - 2\mathbf{G}\mathbf{G}^\top + \mathbf{G}\mathbf{G}^\top\mathbf{G}\mathbf{G}^\top)\mathbf{X}_{\mathcal{S}}^{\text{unsup}}) \\
&= \|\mathbf{X}_{\mathcal{S}}^{\text{unsup}}\|_F^2 - 2\text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}) + \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top\mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}})
\end{aligned}$$

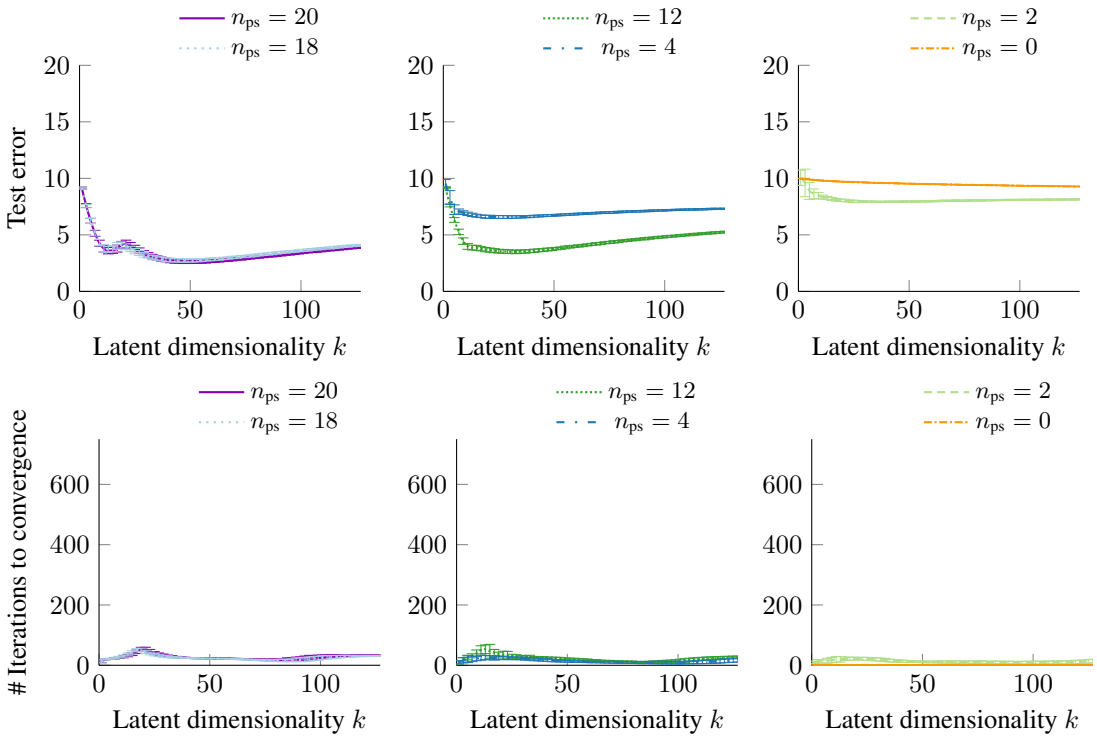

Figure 9: In this figure, we minimize the loss in Equation (6). This figure shows that you can achieve good performance and double descent behavior if you weigh the pseudo-supervised and unsupervised terms in the loss disproportionately. Note that in the $n_{\text{ps}} = 0$ case, we are effectively reducing the step size by making α large. In these experiments, we picked $\alpha = 0.98$. We show results for six different pseudo-supervision levels (i.e., n_{ps} values). For visual clarity, each subfigure includes results for only two n_{ps} values.

which we separate into three terms:

$$
\begin{aligned}
f_1(\mathbf{G}) &= \|\mathbf{X}_{\mathcal{S}}^{\text{unsup}}\|_F^2 \\
f_2(\mathbf{G}) &= -2\text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}) \\
f_3(\mathbf{G}) &= \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}).
\end{aligned}
$$

Clearly, we have that $\nabla_{\mathbf{G}} \|(\mathbf{I}_p - \mathbf{G}\mathbf{G}^\top)\mathbf{X}_{\mathcal{S}}^{\text{unsup}}\|_F^2 = \nabla_{\mathbf{G}} f_1 + \nabla_{\mathbf{G}} f_2 + \nabla_{\mathbf{G}} f_3$ and that $\nabla_{\mathbf{G}} f_1 = 0$. By using some matrix identities, we get that

$$
\begin{aligned}
\nabla_{\mathbf{G}} f_2 &= -2\nabla_{\mathbf{G}} \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}) \\
&= -4\mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G} && \text{((119) from [44])}
\end{aligned}
$$

and

$$
\begin{aligned}
\nabla_{\mathbf{G}} f_3 &= \nabla_{\mathbf{G}} \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}) \\
&= \left(\nabla_{\mathbf{G}^\top} \text{Tr}((\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}})\right)^\top && \text{(Section 4.2.3 of [43])} \\
&= \left(2\mathbf{G}^\top \mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top + 2\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top\right)^\top && \text{((123) of [44])} \\
&= 2\mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{G} + 2\mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}
\end{aligned}
$$

Hence, the gradient of the second term in the considered loss function becomes

$$
\begin{aligned}
\nabla_{\mathbf{G}} \|(\mathbf{I}_p - \mathbf{G}\mathbf{G}^\top)\mathbf{X}_{\mathcal{S}}^{\text{unsup}}\|_F^2 &= \nabla_{\mathbf{G}} f_1 + \nabla_{\mathbf{G}} f_2 + \nabla_{\mathbf{G}} f_3 \\
&= -4\mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G} + 2\mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}\mathbf{G}^\top \mathbf{G} \\
&\quad + 2\mathbf{G}\mathbf{G}^\top \mathbf{X}_{\mathcal{S}}^{\text{unsup}}(\mathbf{X}_{\mathcal{S}}^{\text{unsup}})^\top \mathbf{G}
\end{aligned}
$$

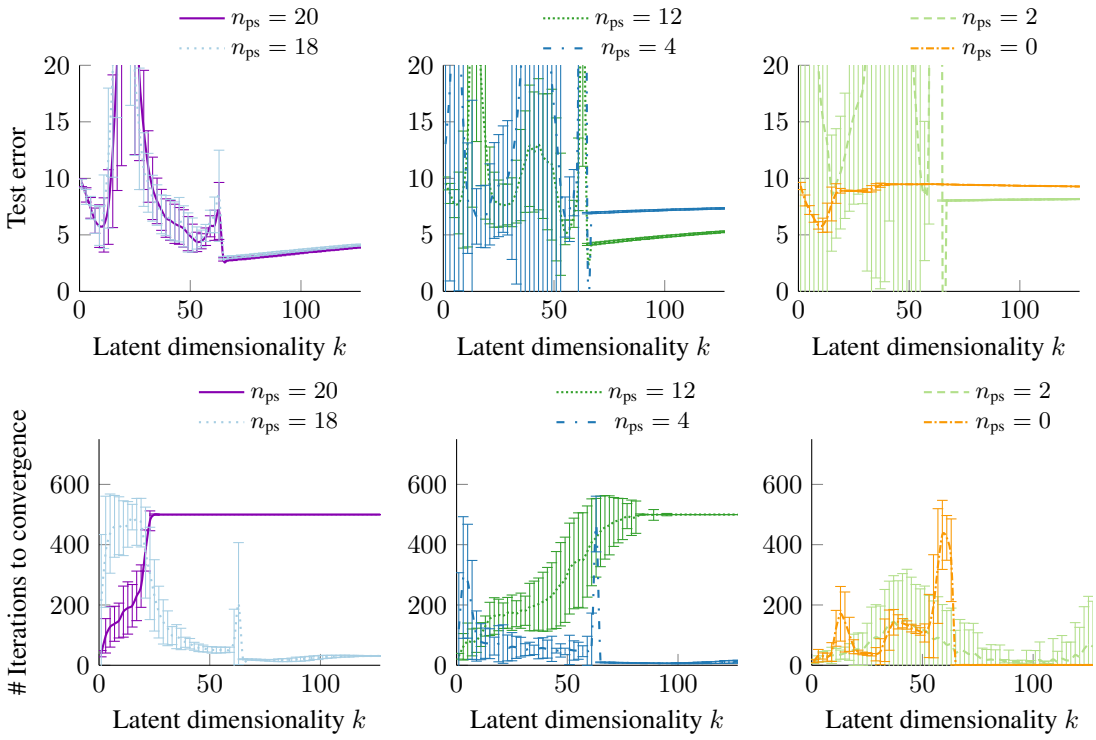

Figure 10: In this figure, we minimize the loss in Equation (7). This figure is a more detailed version of the right column of Figure 3. We show results for six different pseudo-supervision levels (i.e., n_{ps} values). For visual clarity, each subfigure includes results for only two n_{ps} values.

Thus, the total gradient for Equation (4) becomes

$$\begin{aligned}\nabla_{\mathbf{G}}\left(\frac{1}{n_{\text{ps}}}\|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}}-\mathbf{X}^{\text{ps}}\|_F^2+\frac{1}{n_{\text{unsup}}}\|(\mathbf{I}_d-\mathbf{G}\mathbf{G}^\top)\mathbf{X}^{\text{unsup}}\|_F^2\right) &= \frac{2}{n_{\text{ps}}}(\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{sup}}-\mathbf{X}^{\text{sup}})(\mathbf{Z}_{\mathcal{S}}^{\text{sup}})^\top \\ &\quad -\frac{4}{n_{\text{unsup}}}\mathbf{X}^{\text{unsup}}(\mathbf{X}^{\text{unsup}})^\top\mathbf{G} \\ &\quad +\frac{2}{n_{\text{unsup}}}\mathbf{X}^{\text{unsup}}(\mathbf{X}^{\text{unsup}})^\top\mathbf{G}\mathbf{G}^\top\mathbf{G} \\ &\quad +\frac{2}{n_{\text{unsup}}}\mathbf{G}\mathbf{G}^\top\mathbf{X}^{\text{unsup}}(\mathbf{X}^{\text{unsup}})^\top\mathbf{G}.\end{aligned}$$

The gradient for the loss in Equation (7) is similar. Again, we restate the loss for completeness:

$$\mathcal{L}^{\text{train}}(\mathbf{G},\mathcal{D})=\frac{1}{n_{\text{ps}}}\|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}}-\mathbf{X}^{\text{ps}}\|_F^2+\frac{1}{n}\|(\mathbf{I}_d-\mathbf{G}\mathbf{G}^\dagger)\mathbf{X}\|_F^2$$

The gradient of the first term of Equation (7) is the same as in the above result for the loss in Equation (4). The gradient of the second term of Equation (7) requires more work. With $\mathbf{B}=\mathbf{X}\mathbf{X}^\top$

for shorthand and assuming that $\mathbf{G}$ has full column rank, we see that

$$\begin{aligned}
\nabla_{\mathbf{G}}\|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)\mathbf{X}\|_F^2 &= \nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)\mathbf{B}) \\
&= \nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{I}_d - 2\mathbf{G}\mathbf{G}^\dagger + \mathbf{G}\mathbf{G}^\dagger\mathbf{G}\mathbf{G}^\dagger)\mathbf{B}) \\
&= \nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{I}_d - 2\mathbf{G}\mathbf{G}^\dagger + \mathbf{G}\mathbf{G}^\dagger)\mathbf{B}) \\
&= \nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)\mathbf{B}) \\
&= \nabla_{\mathbf{G}}\mathrm{Tr}(\mathbf{B}) - \nabla_{\mathbf{G}}\mathrm{Tr}(\mathbf{G}\mathbf{G}^\dagger\mathbf{B}) \\
&= -\nabla_{\mathbf{G}}\mathrm{Tr}(\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{B}) \\
&= -\nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{B}\mathbf{G}) \\
&= -\nabla_{\mathbf{G}}\mathrm{Tr}((\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{B}\mathbf{G}) \\
&= 2\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{B}\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1} - 2\mathbf{B}\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1}. \quad \text{((126) in [44])}
\end{aligned}$$

Thus, the total gradient for Equation (7) becomes

$$\begin{aligned}
\nabla_{\mathbf{G}}\left(\frac{1}{n_{\text{ps}}}\|\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}}\|_F^2 + \frac{1}{n}\|(\mathbf{I}_d - \mathbf{G}\mathbf{G}^\dagger)\mathbf{X}\|_F^2\right) &= \frac{2}{n_{\text{ps}}}(\mathbf{G}\mathbf{Z}_{\mathcal{S}}^{\text{ps}} - \mathbf{X}^{\text{ps}})(\mathbf{Z}_{\mathcal{S}}^{\text{ps}})^\top \\
&\quad + \frac{2}{n}\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1}\mathbf{G}^\top\mathbf{X}\mathbf{X}^\top\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1} \\
&\quad - \frac{2}{n}\mathbf{X}\mathbf{X}^\top\mathbf{G}(\mathbf{G}^\top\mathbf{G})^{-1}.
\end{aligned}$$

If $\mathbf{G}$ has full row rank instead, one gets a similar gradient expression. During the minimization of the loss in Equation (7), the matrix $\mathbf{G}$ may become close to low rank and make the gradient calculation unstable. For numerical stability of the gradient, we calculate $(\mathbf{G}^\dagger)^\top$ instead of $\mathbf{G}(\mathbf{G}^\top\mathbf{G})^\dagger$.

D Experiments on nonlinear, multilayer GANs on MNIST

In this section we provide details for the experiments on nonlinear, multilayer GANs. One of these experiments is discussed in Section 5 and the other is an additional experiment which is not in the main paper. The details here are relevant to both experiments.

We train a gradient penalized Wasserstein GAN (WGAN-GP) [9] on MNIST [30]. The architecture output directly from PyTorch is shown below with the latent dimensionality changed to k, as it varies in our experiments:

```
Generator(
  (model): Sequential(
    (0): Linear(in_features=k, out_features=128, bias=True)
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Linear(in_features=128, out_features=256, bias=True)
    (3): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace)
    (5): Linear(in_features=256, out_features=512, bias=True)
    (6): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace)
    (8): Linear(in_features=512, out_features=1024, bias=True)
    (9): BatchNorm1d(1024, eps=0.8, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace)
    (11): Linear(in_features=1024, out_features=784, bias=True)
    (12): Tanh()
  )
)

Discriminator(
  (model): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
```

```
    (1): LeakyReLU(negative_slope=0.2, inplace)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2, inplace)
    (4): Linear(in_features=256, out_features=1, bias=True)
  )
)
```

The networks are trained with a gradient penalty weight of $\lambda_{\text{GP}} = 10$. The pseudo-supervised sample pairs were fixed as we varied k so that the plots were comparable. However, we ran both of these experiments over 10 trials, with 10 sets of pseudo-supervised samples corresponding to 10 subsets of the training data. Let us denote $\mathcal{L}_{\text{GP}}$ as the WGAN-GP objective function. We trained the discriminator as usual, and trained the generator with the following modified objective function:

$$\mathcal{L}_G(\mathbf{X}^{\text{batch}}, \mathbf{X}^{\text{ps}}, \mathbf{Z}^{\text{ps}}) = \mathcal{L}_{\text{GP}}(\mathbf{X}^{\text{batch}}) + \|G(\mathbf{Z}^{\text{ps}}) - \mathbf{X}^{\text{ps}}\|_F^2$$

with $\mathbf{X}^{\text{batch}} \in \mathbb{R}^{d \times n_{\text{batch size}}}$, $\mathbf{X} \in \mathbb{R}^{d \times n_{\text{ps}}}$, and $\mathbf{Z}^{\text{ps}} \in \mathbb{R}^{k \times n_{\text{ps}}}$ for generator G. Additionally, one could weigh this pseudo-supervised term more or less, however we found that a weight of 1 was adequate to get our results.

For all of our experiments with nonlinear, multilayer GANs, we have a batch size of 4096, an ADAM learning rate of 0.0002, ADAM hyperparameters $\beta = (0.5, 0.999)$, and clip value of 0.01. We train the discriminator 5 times per iteration. The optimizer values are used for both generator and discriminator. In the main paper, we train for 3000 iterations and use 4096 total samples from the training data so that we are performing gradient descent instead of stochastic gradient descent (SGD). We also do experiments for SGD in Appendix D.1.

We measure test error with geometry score [33] as it is better suited for MNIST than other performance measures, such as Fréchet Inception Distance [21] and Inception Score [16] which are better suited for natural images. For our calculation of the geometry score, we pick $L_0 = 32, \gamma = \frac{1}{1000}, i_{\text{max}} = 100$, and $n = 100$ as done in the original paper when computing scores for the MNIST dataset. Moreover, we generate 10,000 images and compare these generated images to the MNIST test set, which also contains 10,000 images.

In Figure 11 we provide errorbars for Figure 4 to show that pseudo-supervision lowers variance.

D.1 Pseudo-supervision with stochastic gradient descent

In this section, we train a WGAN-GP just as above except for two changes: we train using all the training data (60000 samples) for 200 iterations using SGD. We only train for 200 iterations because now each epoch has about 15 batches instead of the single batch in the previous section.

Our results are shown in Figure 12 and Figure 13. We see that with SGD, we lose the double descent but consistently beat the baseline. We also converge faster than the baseline, but not as fast as the pure gradient descent setting. Moreover, we reduce the variance in the test error across experiments drastically compared to the baseline.

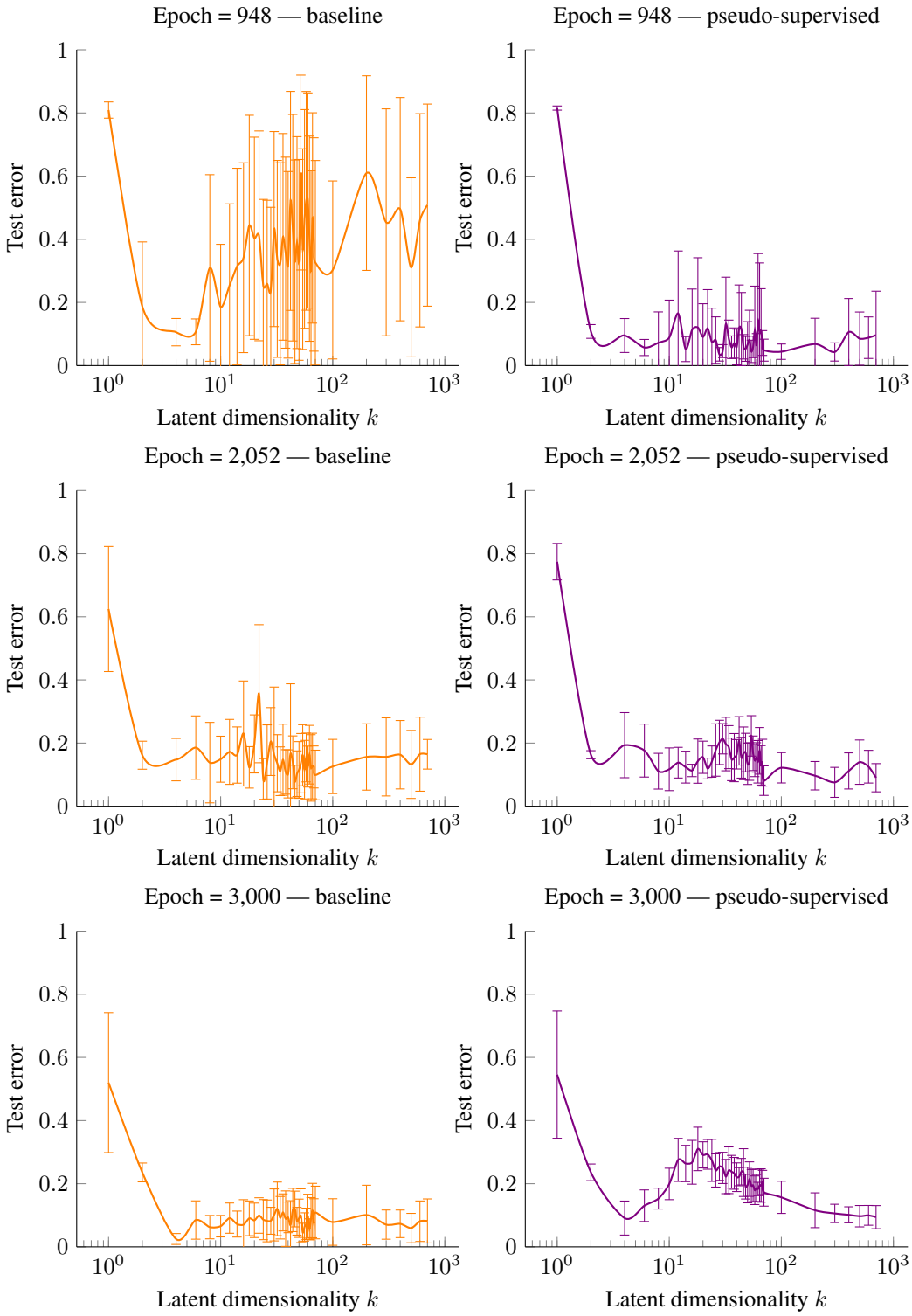

Figure 11: In this figure, we train a WGAN-GP on MNIST using gradient descent on a subset of 4096 training images. This figure is a more detailed version of Figure 4.

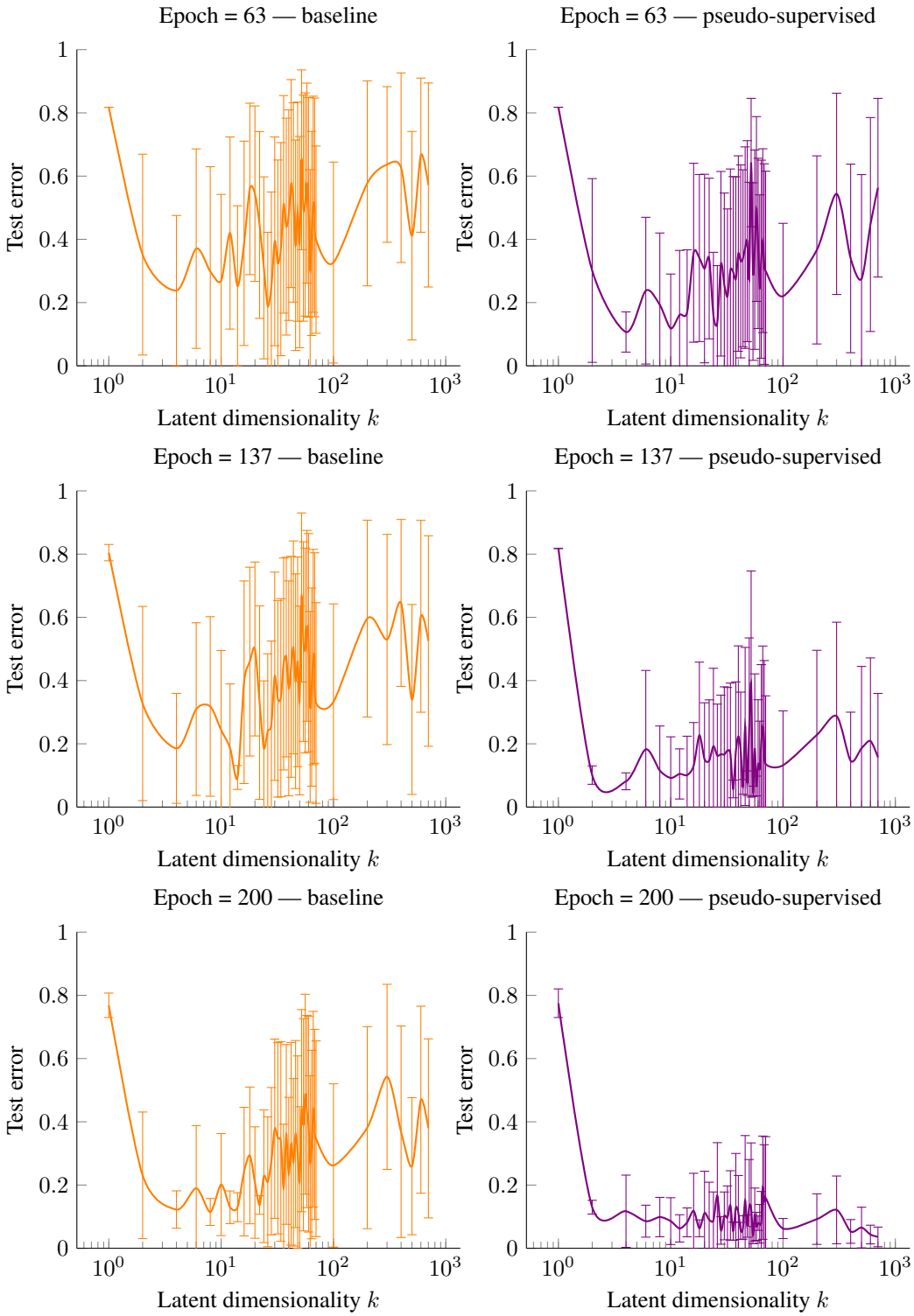

Figure 12: In this figure, we train a WGAN-GP on the full MNIST dataset using SGD. The pseudo-supervised GAN has much lower variance and outperforms the baseline later in training. Convergence speed is also faster for the pseudo-supervised model.

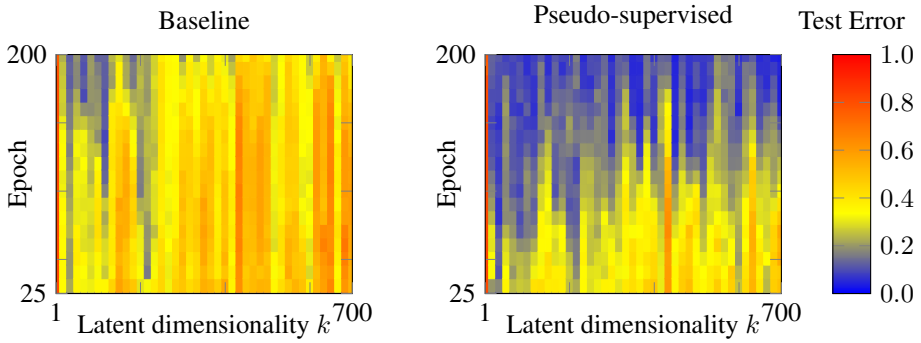

Figure 13: In this figure, we train a WGAN-GP on the full MNIST dataset using SGD. The pseudo-supervised GAN converges to a low error much faster than the baseline. Just as in Figure 5, each point in the heatmap is an average test error over 10 networks The test error is measured by geometry score here. The k-axis is plotted so that each column corresponds to the next entry for better visualization, even though the spacing is $k \in \{1, 2, 4, 6, \ldots, 70, 100, 200, 300, \ldots, 700\}$.