Latent Space Growing of Generative Adversarial Networks

Erik Sandström

Master's thesis
2019:E28

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Latent Space Growing of Generative Adversarial Networks

Erik Sandström
Lund University
erik.sandstrm@gmail.com
Advisors: Ted Kronvall, Henning Petzka
Examiner: Kalle Åström

Contents

Master Thesis, Lund University, 2019

1 Abstract

This thesis presents a system, which builds on the Generative Adversarial Network (GAN) framework, with the focus of learning interpretable representations of data. The system is able to learn representations of data that are ordered in regards to the saliency of the attributes, in a completely unsupervised manner. The training strategy expands the latent space dimension while appropriately adding capacity to the model in a controlled way. This builds on the intuition that highly salient attributes are easiest to learn first. Empirical results on the Swiss roll dataset show that the representation is structured in regards to the saliency of the attributes when training the latent space progressively on a very simple GAN architecture. Experiments using a more complex system, trained on the CelebA dataset, scales the idea to a more interesting use case. Experiments using latent space interpolations show that our model successfully structures the latent space with respect to the saliency of the attributes, while also generating at least as real looking images and in less training time, than state-of-the-art methods.

2 Preface

The thesis was conducted during the spring of 2019 at the Centre for Mathematical Sciences at Lund University. First and foremost, I would like to thank my advisors, Ted Kronvall and Henning Petzka, for many fruitful and fun discussions and Martin Trimmel for getting to know you as a great friend from whom I have learned a lot. I would also like to thank everyone else in professor Cristian Sminchisescu's group for being welcoming and fostering my interest in research. I am looking forward to collaborating with you all in the future.

Lastly, I would like to express my gratitude to several people who are my life long supporters. Thank you Eva-Lena Lindén for teaching me that hard work pays off. Thank you Stig Sandström for igniting my interest in science and thank you also Christer Ovrén for inspiring me to become an engineer. Thank you Anna Sandström Löf, Annika Sandström and Matilda Backendal for always believing in me, no matter what.

3 Introduction

Scientific discoveries often start with observations of the real world. Many times, the problem is to explain hidden structure in observed data. In some cases, the structure could perhaps be explained by a differential equation derived from the laws of physics. When studying data with no apparent relation to physics though, other tools need to be used. One such tool, coming from statistics, is called density estimation. Density estimation tries to solve the generic problem of explaining hidden structure in *any* type of data. Density estimation assumes that there exists some true, but unknown, way of explaining the data, which consists of a set of observations, also called samples. Strictly speaking, density estimation assumes that the samples come from an unknown probability density function $p_{data}(x)$ which, at each sample x, describes the likelihood of observing that sample. Adding some more terminology, the data is said to be distributed according to $p_{data}(x)$. Given independent samples from $p_{data}(x)$, density estimation is the process of constructing another density $p_{model}(x;\theta)$ parameterized by θ that approximates $p_{data}(x)$ as closely as possible [8]. This is most often achieved by minimizing some measure of distance between $p_{data}(x)$ and $p_{model}(x;\theta)$ with respect to θ and this process is referred to as training.

Modeling $p_{model}(x;\theta)$ can be done in many ways. One such way is through so-called generative models, which this work focuses on. These models have the ability to generate new data from $p_{model}(x;\theta)$ such that the data looks as if it came from $p_{data}(x)$. In order to produce a sample from $p_{model}(x;\theta)$, an input z to the model, called the latent vector, is often needed. The vector is called latent since it describes that z is a hidden representation of the data sample x. Furthermore, all latent vectors lie in a vector space, which is called the latent space. The latent vector z could be seen as a code, where each entry controls some characteristics of the data sample x. One desirable property of a generative model is to have the ability to specify each entry in z to encode a certain isolated characteristic of the output sample x. For instance, when generating images of faces, it would be beneficial if the entries in z, independently of each other, controlled the sex, hair color, hairstyle, skin color, pose, facial shape, eye color and so on. This would make image generation very

controlled and could help greatly when solving other tasks like classification, localization, anomaly detection and planning, which all need good representations of data. The problem is that most generative models do not yield representations where the entries of z independently control semantic attributes of the data instance. Instead, the entries of z are often entangled, which in practice means that it is impossible to relate the entries to specific isolated attributes of the data. A disentangled latent representation is thus sought. While there exist attempts to formalize the definition of disentanglement [13], using it informally serves our purposes well and in this work, if different directions in the latent space encode semantically independent information, then the latent space is said to be disentangled.

This work focuses on a specific way of disentanglement, where the saliency of the attributes are ordered such that attributes causing most variability in the image are seen as most salient. Representations of this form could potentially be useful for tasks involving style-transfer, but also extend to other applications like data compression and dimensionality reduction. In this work, the sought representation is implicitly inferred by using a novel training process of a generative model, which is trained in an unsupervised manner. Contrary to supervised training, this means that no annotations are required during training. The main contribution is a training scheme of artificial neural networks, GANs specifically, where the latent space dimension grows during the training process in a controlled way. To the best of our knowledge, we are the first ones proposing this idea. The underlying motivation behind this approach is a form of curriculum learning that is imposed by the training process. Curriculum learning originates from the human education system where students are presented easier concepts first and gradually face higher difficulty. The argument is made that by first modeling the problem with a latent space of small dimension, the system is required to first focus on the most salient attributes of the data, which should be easiest to learn at first. Our reasoning relies on the assumption that the most salient attributes of the data lie on a so-called submanifold (of the data manifold) which has high variance. More details will be covered in Section 4.3, but in short, a manifold is a generalization of a surface in a Euclidean space and can serve as a tool to reason about high-dimensional data in a geometric way. Examples of aspects with high data variation could be facial shape, pose and hairstyle. As more latent dimensions become available, aspects of lesser variation, like eye shape and mouth expression are modeled, resulting in disentanglement in the latent space between aspects of high and small data variation. This is similar to principal component analysis (PCA), which linearly orders the data dimensions with respect to data variation. Our model can, however, model nonlinear behavior. It is important to note that our protocol does not imply that each entry in the latent vector z controls an isolated semantic attribute of the output like hairstyle, sex or pose. Rather, the purpose is to order aspects of data variation.

From an optimization perspective, by training the latent space dimensions progressively, the parameters θ of the model are also trained in a progressive manner. The hypothesis is that the training protocol effectively excludes parameter optima that would result in an entangled latent representation and local optima, where the representation is ordered with respect to attribute variability, are thus more easily found (Section 6). Results are shown empirically using latent space walks, where it is clear that our model successfully structures the latent space with respect to the saliency of the attributes, while also generating at least as real looking images and in less training time, than state-of-the-art methods.

The report is outlined as follows. Section 4 presents an overview of generative models and GANs specifically. A brief section on artificial neural networks is also included. Section 5 covers related work, while Section 6 describes, in detail, the developed methodology to grow the latent space of GANs. Several growing strategies are developed and tested in Section 7, which contains experiments on a Swiss roll dataset and the CelebA dataset [18]. Section 8 contains a brief discussion, while Section 9 describes potential directions for future work. The report is concluded in Section 10.

4 Generative Modeling

A generative model can generate data from a distribution $p_{model}(x;\theta)$, such that the generated data is indistinguishable from some true, but unknown, distribution $p_{data}(x)$. This assumes that the parameters θ are tuned adequately. To tune or optimize the parameters, a measure of difference, usually called the loss function or objective, between $p_{model}(x;\theta)$ and $p_{data}(x)$, needs to be minimized with

respect to θ. In statistics, the notion of distance between probability density functions is measured by a divergence, which is a weaker notion than that of a distance, since it may not be symmetric with respect to the data distribution $p_{data}(x)$ and the model distribution $p_{model}(x;\theta)$. Using for instance the Kullback-Leibler divergence (and putting the densities in a certain order) yields a loss function, which creates an optimization problem equivalent to maximum likelihood estimation (MLE) on the observed data [9], i.e.,

$$\min_{\theta \in \mathcal{M}} KL[p_{data}(x)||p_{model}(x;\theta)] = \min_{\theta \in \mathcal{M}} \mathbb{E}_{x \sim p_{data}} \left[log \frac{p_{data}(x)}{p_{model}(x;\theta)} \right], \tag{1}$$

where θ is the collection of parameters that are optimized from a family of models $\mathcal{M}$. The true density $p_{data}(x)$ does not depend on θ and thus the expression can be equivalently written as performing MLE

$$\max_{\theta \in \mathcal{M}} \mathbb{E}_{x \sim p_{data}} [log\, p_{model}(x;\theta)]. \tag{2}$$

Changing the order of $p_{data}(x)$ and $p_{model}(x;\theta)$ in (1) produces a different objective, which can not be phrased as MLE, but in some situations, that objective is also useful [8].

In practice, $p_{data}(x)$ is not available since only discrete samples of $p_{data}(x)$ exist. Calculating the expectation in (2) with respect to the true distribution $p_{data}(x)$ is therefore not possible, but a so-called Monte Carlo estimate of the expectation can be used in practice. A Monte Carlo estimate builds on the fact that

$$\mathbb{E}_{x \sim p_{data}} [log\, p_{model}(x;\theta)] = \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} log\, p_{model}(x_i;\theta), \quad x_i \sim p_{data}. \tag{3}$$

Since in practice, infinitely many samples are not available, a Monte Carlo estimate approximates the true expectation in (2) as

$$\mathbb{E}_{x \sim p_{data}} [log\, p_{model}(x;\theta)] \approx \frac{1}{N} \sum_{i=1}^{N} log\, p_{model}(x_i;\theta), \quad x_i \sim p_{data}. \tag{4}$$

4.1 Implicit and Explicit Generative Models

In order to understand the properties of the specific generative model used in this work, a categorisation of generative models is made. Either the likelihood can be modelled explicitly or implicitly. Starting with explicit generative models, one can note that in order to maximize the likelihood in (4), the likelihood function $p_{model}(x;\theta)$ needs to be explicitly specified from some parameterized family of distributions. This leads to the main advantage of explicit generative models, which is that training is straightforward, since the MLE objective can be used directly. This is achieved by plugging in the function expression of $p_{model}(x;\theta)$ into (4), and optimizing it with an appropriate gradient ascent method. Another advantage of explicit generative models is that the likelihood for a given sample x can be computed directly. Examples of these models include the variational autoencoder and pixelcnn [17], [25]. The main issue with explicit generative models is, however, that the chosen family of distributions both needs to be sufficiently complex to model the often multi-modal data distribution $p_{data}(x)$, while still being tractable to compute the likelihood [8]. Implicit generative models offer a solution to this dilemma. They do not require explicitly specifying the distribution family and thus no assumptions need to be made on $p_{model}(x;\theta)$. As a consequence though, a more difficult optimization problem often ensues, together with the fact that likelihood evaluation is no longer possible. Instead, only samples from $p_{model}(x;\theta)$ can be drawn. An example of an implicit generative model is the generative adversarial network (GAN) [7], which is the main focus of this work.

4.2 Generative Adversarial Networks

The idea behind GANs is a pair of competing models called generator and discriminator. The generator $G_\theta : \mathbb{R}^k \to \mathbb{R}^d$ (commonly $k < d$) is a complicated nonlinear real-valued differentiable function parameterized by θ, that models $p_{model}(x;\theta)$. The generator takes as input the latent vector $z \in \mathbb{R}^k$, which is sampled from a known, often simple distribution $p(z)$, and generates a sample from $p_{model}(x;\theta)$ by transforming z through G_θ, i.e., $G_\theta(z) = x \sim p_{model}(x;\theta)$. The goal of

G_θ is to produce samples indistinguishable from the true data distribution $p_{data}(x)$. To guide G_θ, another complicated nonlinear real-valued differentiable function parameterized by ϕ, called the discriminator $D_\phi : \mathbb{R}^d \to \mathbb{R}$, tries to distinguish between samples generated by G_θ (labeled as fake) and samples from the true data distribution (labeled as real). The generator is penalized if it generates samples that are recognized as fake by the discriminator and the discriminator receives a penalty if it fails to distinguish between real and fake samples.

To derive the GAN objective function, consider $D_\phi = D_\phi(x)$ to generate a number between 0 and 1 for a given input x. If x comes from $p_{data}(x)$, then the target value for $D_\phi(x)$ is 1 $(label = real)$. If x comes from $p_{model}(x;\theta)$ the target value for $D_\phi(x)$ is 0 $(label = fake)$. The target of $G_\theta(z)$ is the adverse of D_ϕ, which is to generate samples $x = G_\theta(z)$ such that $D_\phi(G_\theta(z)) \approx 1$. Based on this, an objective function is constructed that takes the form

$$\min_{\theta\in\mathcal{M}} \max_{\phi\in\mathcal{F}} \mathbb{E}_{x\sim p_\theta}[log(1 - D_\phi(x))] + \mathbb{E}_{x\sim p_{data}}[log(D_\phi(x))]. \tag{5}$$

Here $p_{model}(x;\theta) = p_\theta$ for brevity. While the objective may look intimidating at first, breaking it down leads to an intuitive optimization problem. Here $\mathcal{M}$ and $\mathcal{F}$ define parametric families of the models. From (5), the objective for G_θ and D_ϕ can be separated. This yields for D_ϕ,

$$\min_{\phi\in\mathcal{F}} -\mathbb{E}_{x\sim p_\theta}[log(1 - D_\phi(x))] - \mathbb{E}_{x\sim p_{data}}[log(D_\phi(x))]. \tag{6}$$

As the second term in (5) does not depend on θ, the generator loss becomes

$$\min_{\theta\in\mathcal{M}} \mathbb{E}_{x\sim p_\theta}[log(1 - D_\phi(x))]. \tag{7}$$

For numerical reasons, the loss in (7) saturates at the beginning of training, when the generator is poor. This yields very slow training due to small gradient magnitudes. To alleviate this problem, instead, the following loss is used for the generator, which together with (6), results in the so-called non-saturated GAN

$$\min_{\theta\in\mathcal{M}} -\mathbb{E}_{x\sim p_\theta}[log(D_\phi(x))]. \tag{8}$$

Finally, thanks to the formulation of the objective in (5) as a sum of expectations, it is possible to estimate the objective by using Monte Carlo estimates, following a recipe similar to (4), instead of exact computation. The objective can thus be optimized without an explicit form of the likelihood $p_{model}(x;\theta)$. Only samples are required from the model.

While here we only discuss the intuitive motivation, the objective in (5) can also be derived from the minimization of a statistical divergence betwen $p_{data}(x)$ and $p_{model}(x;\theta)$. This topic is discussed in Section 4.5 and, building from the latter viewpoint on GANs, in Section 4.6 we discuss how choosing the Wasserstein distance as the divergence leads to the Wasserstein GAN [1].

4.3 The Image Manifold

To find out what specific kinds of functions are useful for G_θ and D_ϕ, the observation is made that any d-dimensional distribution can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function [6]. Recall, however, that the input dimension to the generator G_θ is not necessarily equal to the output dimension. The observation is still applicable though and to see that, some new terminology and reasoning is introduced as follows. First consider a real image of spatial size $n \times n$. Stacking the image matrix as a vector, the image can be seen as to lie in $\mathbb{R}^{n\times n\times 3} = \mathbb{R}^d$ ($\times 3$ due to RGB). The constraint is that the image has to be real-looking and not all vectors in the space $\mathbb{R}^{n\times n\times 3}$ represent valid real images. This constrains the real image to lie on a so-called manifold in $\mathbb{R}^{n\times n\times 3}$. The manifold is a generalization of a surface in a Euclidean space and it describes geometrically how the data is distributed on some nonlinear surface in the high-dimensional space $\mathbb{R}^{n\times n\times 3}$. Digesting the previous sentence is difficult, but in lower dimensions, manifolds appear everywhere. For instance, a sphere is a 2D manifold in $\mathbb{R}^3$ and a ring is a 1D manifold in $\mathbb{R}^2$. A perhaps not very formal definition, but serving a conceptual purpose, is that a manifold is a surface that everywhere locally resembles a Euclidean space. Even more informally, a manifold looks like a plane if one moves close enough to its surface.

The true dimension of the image manifold is unknown, but from a modeling perspective, under mild regularity conditions, the input dimension to the generator determines the dimension of this space.

Therefore, it may be reasonable to assume that the input dimension to the generator should be chosen in a range that respects the dimension of the data manifold. This means that the d-dimensional density $p_{data}(x)$ could be seen as created by a density of a smaller dimension. This concludes the reasoning that if the input dimension to the generator is close enough to the underlying dimension of the real data, any $p_{model}(x;\theta)$ can be constructed as long as G_θ is chosen to be sufficiently complicated.

4.4 Artificial Neural Networks

Building on the reasoning in Section 4.3, this section presents the type of functions used for G_θ and D_ϕ. The reader is assumed to be familiar with neural networks and an introduction to the topic is provided by [21]. Here, only the necessary nomenclature is covered.

Artificial neural networks are a class of parameterized functions $f : \mathbb{R}^n \to \mathbb{R}^m$ that are constructed by composing linear functions f_k and nonlinear functions a_k such that

$$f = a_n \circ f_n \circ a_{n-1} \circ f_{n-1} \circ \ ... \ \circ a_2 \circ f_2 \circ a_1 \circ f_1. \tag{9}$$

Each function $f_k : \mathbb{R}^u \to \mathbb{R}^v$ is parameterized by weights θ_k and forms a so-called layer together with the so-called activation function a_k, that is applied elementwise to the output of f_k. The activation function is what gives the system the ability to model nonlinear behavior and is commonly chosen to be either ReLu, $a(x) = max(0, x)$, or Leaky ReLu (LReLu) defined by $a(x) = max(cx, x)$, c constant. The hyperbolic tangent (tanh) function is sometimes used to constrain the output between $[-1, 1]$.

It can be shown that, in the limit, as more dimensions are added to the layers, neural networks are universal function approximators. This means that with sufficient capacity, they can approximate any given function arbitrarily well [21]. Therefore, they are a useful tool to model G_θ and D_ϕ.

The input vector to each layer is usually visually represented by putting each input entry as a number in a node and stacking the nodes vertically. The output from the layer is then another vertical stack of nodes. Depending on how the layer function defines connections between input nodes and output nodes, different properties of the layer are achieved. If all output nodes are allowed to be any linear combination of all input nodes, the layer is said to be fully connected. The computation is equivalent to matrix multiplication between the weights of the layer, as a matrix, and the input, as a vector. If the weight matrix is forced to be sparse and output nodes can be written as linear combinations of only input nodes that are nearby, the layer is said to be convolutional. The computation is equivalent to sliding a weight matrix, called kernel, across the input, reshaped into an image.

Stacking many fully connected layers form a so-called multilayer perceptron (MLP) while stacking many convolution layers result in a convolutional neural network (CNN).

The optimization scheme for finding the appropriate weights in a neural network model is referred to as training and is done by minimizing a specified objective, also called loss, by following the negative gradient direction of the loss with respect to the weights. These algorithms are referred to as gradient descent methods. In this work, the Adam [16] optimizer and the RMSprop [22] optimizer are used. These methods use an adaptive form for the learning rate, which controls the size of each weight update.

4.5 Connecting the GAN Objective to Divergence Minimization

Recall that the GAN objective was conceptually motivated by (5). No connection was, however, made to a statistical notion of difference between $p_{model}(x;\theta)$ and $p_{data}(x)$. In this section, we will see that the objective in (5) can be generalized to express a multitude of GAN objectives, which all relate to a measure of divergence between $p_{model}(x;\theta)$ and $p_{data}(x)$. The starting point is the fact that some divergences between $p_{model}(x;\theta)$ and $p_{data}(x)$ can be expressed as [9]

$$\max_{\phi \in \mathcal{F}} \mathbb{E}_{x\sim p_\theta}[h_\phi(x)] - \mathbb{E}_{x\sim p_{data}}[h'_\phi(x)]. \tag{10}$$

Here $p_{model}(x;\theta) = p_\theta$ for brevity and h and h' are appropriate real-valued differentiable functions parameterized by ϕ from an appropriate family of models $\mathcal{F}$. This yields a form where the

divergence can be minimized by constructing a minimax objective on the form

$$\min_{\theta \in \mathcal{M}} \max_{\phi \in \mathcal{F}} \mathbb{E}_{x \sim p_\theta}[h_\phi(x)] - \mathbb{E}_{x \sim p_{data}}[h'_\phi(x)]. \tag{11}$$

Setting $h_\phi(x) = log(1 - D_\phi(x))$ and $h'_\phi(x) = -log(D_\phi(x))$ yields the original GAN in (5) and thus (11) is a generalization of (5). Without covering the full details, which can be found in [7], one finds that the objective in (5) for the generator, under the assumption of a perfect discriminator, minimizes the so-called Jensen-Shannon divergence between $p_{data}(x)$ and $p_{model}(x; \theta)$.

Different choices of h, h' and $\mathcal{F}$ yield different divergences that can be used to train GANs and in the next section, a GAN utilizing the so-called Wasserstein distance is presented.

4.6 Wasserstein GAN

Minimizing the Jensen-Shannon divergence between $p_{data}(x)$ and $p_{model}(x; \theta)$ as the GAN objective has been shown to cause diminishing gradients during training, and thus other metrics have been proposed in the literature. The Wasserstein distance has been one of the most popular and this section presents the idea behind the Wasserstein GAN [1].

The Wasserstein distance is a distance metric between any two probability density functions. Intuitively, the distance describes the minimum "cost" of turning one distribution into the other, by assuming that the cost is proportional to the amount of probability mass that needs to be moved times the distance it has to be moved. The metric is also called the earth mover's distance, due to the analogy of moving mass around.

The Wasserstein GAN (WGAN) proposes to use the Wasserstein distance as the metric to optimize ϕ and θ. This metric has been shown to provide useful gradients throughout the entire training process. In practice, this means that the generator and discriminator give each other encouraging feedback (non-zero bounded gradient information flow), which in turn leads to stable training. The Wasserstein GAN objective is formed by defining $\mathcal{F}$ such that h_ϕ and h'_ϕ are 1-Lipschitz with respect to the input x and setting $h_\phi(x) = -D_\phi(x)$ and $h'_\phi(x) = -D_\phi(x)$. Here, the discriminator is not restricted to the interval $[0, 1]$, but allowed to take any value in $\mathbb{R}$. The way in which the 1-Lipschitz constraint is enforced in [1] is not optimal. The authors of [10] therefore enforce the 1-Lipschitz constraint with an extra gradient-penalty (GP) term added to the full objective, yielding the WGAN-GP objective as

$$\min_{\theta \in \mathcal{M}} \max_{\phi \in \mathcal{F}} -\mathbb{E}_{x \sim p_\theta}[D_\phi(x)] + \mathbb{E}_{x \sim p_{data}}[D_\phi(x)] - \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}[(||\nabla_{\hat{x}} D_\phi(\hat{x})||_2 - 1)^2], \tag{12}$$

where λ is a hyperparameter and $\hat{x}$ is a point sampled uniformly between a real and a fake example, i.e. $p_{\hat{x}} = t \cdot p_{data}(x) + (1-t) \cdot p_\theta(x), t \in U(0, 1)$. The minimization objectives for the discriminator and generator thus become

$$\min_{\phi \in \mathcal{F}} \mathbb{E}_{x \sim p_\theta}[D_\phi(x)] - \mathbb{E}_{x \sim p_{data}}[D_\phi(x)] + \lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2], \tag{13}$$

and

$$\min_{\theta \in \mathcal{M}} -\mathbb{E}_{x \sim p_\theta}[D_\phi(x)], \tag{14}$$

respectively.

5 Related Work

Recently, there has been rapid improvement in the quality of generated images of GANs [14], [19], [3]. The area of understanding the exact mechanism for controlled image generation is, however, still underexplored. At the centre lies the problem of understanding the structure of the latent space. One of the first works considering this problem with GANs is [4], where ideas from information theory are used to disentangle the image representation by adding an additional latent code to the original latent space. One drawback of the approach is that the prior distribution of each entry in the latent code needs to be specified before training, which may be a difficult task when scaling the system.

Inspired by style-transfer, [15] proposes a generator architecture by feeding in latent variables across many layers and introduces an intermediate latent space with non-convex properties. In [5], the authors also feed in different parts of the latent code at different layers, and use a cascade of convolutional networks within a Laplacian pyramid framework. The authors of [3] study how the choice of the latent space distribution affects image quality and [2] shapes the latent space by using Gaussian mixture models. In [20], the authors provide details of how clustering can be made in the latent space and [23] encourages convexity in the feature space as a means to perform interpolations. The work of [24] tries to disentangle the feature representation for pose-invariant recognition. Given the specific task, the method requires explicit attribute supervision and encodes each attribute as a separate element in the feature vector. Perhaps most similar architecturally to our work is [14] where a progressively growing GAN is proposed, generating different spatial resolutions at different steps of training. Their architecture does, however, not consider learning a disentangled representation. To the best of our knowledge, we are the first to propose a strategy to grow the latent space dimension of GANs.

6 Latent Space Growing of GANs

This section presents in detail the proposed methodology in order to progressively grow the latent space dimension in MLP-based GANs and CNN-based GANs. The MLP-case serves as a toy example where experiments show the motivation for the idea of growing the latent space dimension progressively. The CNN-case shows how to scale the method to more interesting use cases.

It is of initial interest to understand more in-depth the reasoning why it could be beneficial to grow the latent space dimension progressively during training. There are two main arguments.

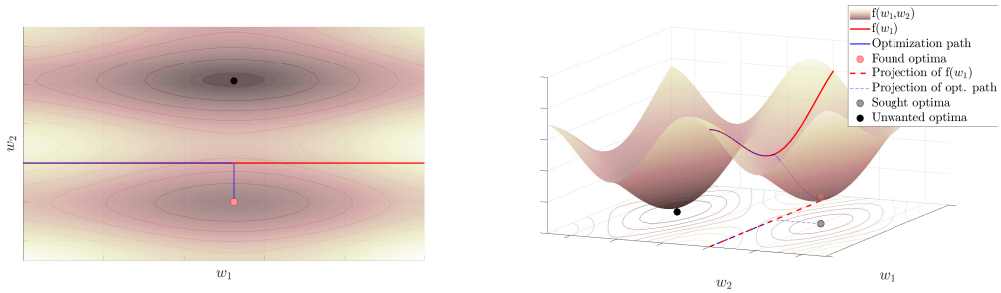
1. Progressively increasing the latent space dimension implies that parameters are added to the objective function progressively. This further implies that local optima are effectively excluded from being reached by the training protocol. Our hypothesis is that entangled optima are excluded by using the proposed approach.

2. By first training on a restricted set of latent dimensions, these dimensions focus on modeling the aspects of most variation in the data which we assume to lie on a submanifold of the data manifold with high variance. As more latent dimensions are added, the added capacity to the model means that also aspects of lesser variation can be taken into account. This argument is shown empirically using experiments on a Swiss roll dataset in Section 7.

Motivated by the first argument, a conceptual example is constructed in order to show that local optima are excluded by training the parameters in a progressive manner. Consider a simple case where the objective function f has at most two optimizable parameters $\{w_1, w_2\}$. First, assume that the function has one optimizable parameter $\{w_1\}$ and that the function is given by the red curve in Fig.1a and 1b. Next, assume that when f has two optimizable parameters ($\{w_1, w_2\}$), the function landscape takes the form given by the complete surface in Fig.1a and 1b. There exist two local optima in the graph given by the black and gray dots. The black optimum is said to be unwanted while the gray is sought. Consider the case where a gradient descent based method is used. These methods require an initialization. If both parameters ($\{w_1, w_2\}$) are trained from scratch, the initial point could be anywhere in $\mathbb{R}^2$. This yields a scheme where it is equiprobable to end up in the black (unwanted) and the gray (sought) optimum.

Now consider first optimizing with respect to $\{w_1\}$. This yields restricting the path to the red curve in Fig.1a and 1b. The local minimum at the bottom of the convex shape is found and the parameter $\{w_2\}$ is added to the optimizer. Now, $\{w_1, w_2\}$ are optimized jointly. Since the gradient at the minimum of the red curve points in the direction of the sought minimum, the sought minimum will always be found. In this way, the unwanted minimum has been excluded from ever being found. It is important to note that this example serves as a conceptual hint. In reality, the loss landscape is high-dimensional and can not be represented accurately visually. The hypothesis posed is that the optima that are excluded from the training scheme yield entangled representations of the latent space. The hypothesis is tested in Section 7 by random walks in the latent space.

Next, the growing strategy of MLP and CNN GANs is described in detail.



(a) (b)

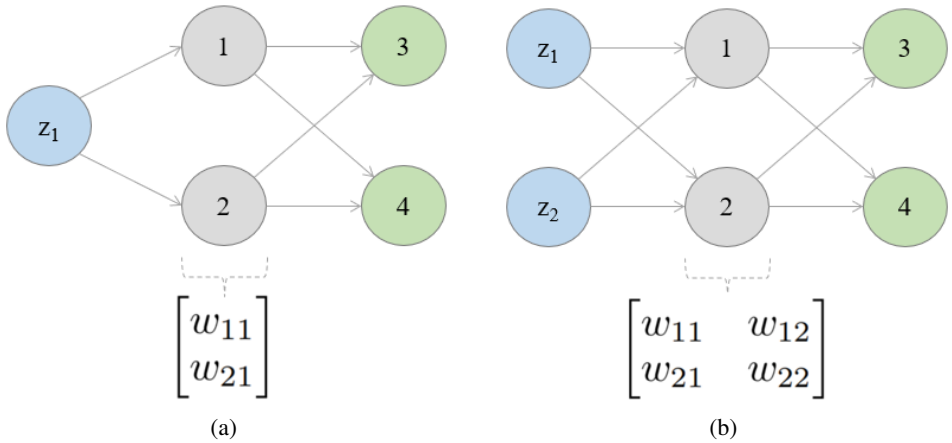
Figure 1: (a) Top view of loss landscape. (b) Loss landscape. By restricting a gradient-based optimizer to first optimize parameter w_1 and then jointly $\{w_1, w_2\}$, the black (unwanted) optimum is excluded from being reached.

6.1 Growing MLP GANs

One can think of several ways to add latent space dimensions to an MLP GAN during training. The simplest way will be explored here and used to show the potential power of this method to produce representations that are ordered with respect to the saliency of the attributes.

Starting with a simple example to understand the basic concept, consider a generator MLP with one hidden layer, one latent space dimension and two output dimensions, given by Fig.2a. If one considers the latent input as a one-component vector $z = [z_1]^T$, the weight matrix of the hidden layer of Fig.2a takes the form of a 2×1 matrix where w_{11} corresponds to weight one in node one while w_{21} corresponds to weight one in node two. The bias terms for each node are excluded for brevity.

Adding a second latent space dimension yields a generator network given by Fig.2b. Each node receives another weight and an additional column is added to the weight matrix of the hidden layer in Fig.2b. The number of bias terms is unchanged and the number of weights in later layers is also unchanged. Take special note of the fact that the architecture of the discriminator network is entirely unchanged using this growing strategy. Using this approach, the general recipe to grow the latent space in any MLP GAN from an n-dimensional latent space to an m-dimensional latent space ($m > n$) is to add $m - n$ columns of weights to the weight matrix of the first layer in the generator. In this work, the new weights are zero initialized.



(a) (b)

Figure 2: (a) An MLP with one hidden layer with two nodes and a one-dimensional latent space and two outputs. (b) An MLP with one hidden layer with two nodes and a two-dimensional latent space and two outputs.

6.2 Growing CNN GANs

In this section, five different GAN architectures are explained in detail. The baseline GAN (bGAN) is a standard GAN without any special growing techniques. The layer growing GAN (lgGAN) is the model proposed in [14]. Here, the resolution of the training images is progressively grown and additional layers accommodate the increase in resolution. The baseline with growing GAN (bgGAN) grows the latent space progressively and only the filter channel depth of the first layer is adapted accordingly. The channel growing GAN (cgGAN) takes this a step further. As the latent space dimension is grown, the cgGAN progressively grows the number of input and output channels of each layer of both the generator and discriminator. Finally, the channel and layer growing GAN (clGAN) combines the cgGAN and lgGAN into one system where both channel depth as well as the number of layers of the network grow progressively. The models are summarized in Tab.1.

		Latent Space	
		Not Grown	Grown
Spat. Res.	Not Grown	bGAN	bgGAN, cgGAN
	Grown	lgGAN	clgGAN

Table 1: The growing strategies of all CNN GANs. "Spat. Res." refers to "Spatial Resolution".

The bGAN and lgGAN are used as baselines to evaluate the growing strategies of the latent space. Another important note is that all grown models in Tab.1 end up with the same network architecture when training finishes.

6.2.1 Baseline GAN

The baseline GAN (bGAN) uses the architecture in Tab.2 and 3 and is a standard GAN without growing any parts of the network during training. All models in Tab.1 end up with this architecture at the end of training.

Some clarification of the naming of the layers in Tab.2 and 3 is in order. The first letter in the name refers to $g - generator$ and $d - discriminator$, while the following number is the layer identifier.

GANs have a tendency to only capture a subset of the modes present in the data distribution $p_{data}(x)$. To alleviate this problem, a trick called minibatch standard deviation is used. The details are covered in Section 6.2.6 and other architectural details are covered in Section 6.2.8.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 256$	-
Conv. 4×4	LReLU	$4 \times 4 \times 256$	$g1$
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$g2$
Upsample	-	$8 \times 8 \times 256$	-
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$g3$
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$g4$
Upsample	-	$16 \times 16 \times 128$	-
Conv. 3×3	LReLU	$16 \times 16 \times 64$	$g5$
Conv. 3×3	LReLU	$16 \times 16 \times 64$	$g6$
Upsample	-	$32 \times 32 \times 64$	-
Conv. 3×3	LReLU	$32 \times 32 \times 32$	$g7$
Conv. 3×3	LReLU	$32 \times 32 \times 32$	$g8$
Upsample	-	$64 \times 64 \times 32$	-
Conv. 3×3	LReLU	$64 \times 64 \times 16$	$g9$
Conv. 3×3	LReLU	$64 \times 64 \times 16$	$g10$
Upsample	-	$128 \times 128 \times 16$	-
Conv. 3×3	LReLU	$128 \times 128 \times 8$	$g11$
Conv. 3×3	LReLU	$128 \times 128 \times 8$	$g12$
Conv. 1×1	tanh	$128 \times 128 \times 3$	$g13$

Table 2: Baseline Generator Architecture. The "Output Shape" format is: height $\times$ width $\times$ channels.

Discriminator	Act.	Output Shape	Name
Input Image	-	$128 \times 128 \times 3$	-
Conv. 1×1	LReLU	$128 \times 128 \times 8$	$d14$
Conv. 3×3	LReLU	$128 \times 128 \times 8$	$d13$
Conv. 3×3	LReLU	$128 \times 128 \times 16$	$d12$
Downsample	-	$64 \times 64 \times 16$	-
Conv. 3×3	LReLU	$64 \times 64 \times 16$	$d11$
Conv. 3×3	LReLU	$64 \times 64 \times 32$	$d10$
Downsample	-	$32 \times 32 \times 32$	-
Conv. 3×3	LReLU	$32 \times 32 \times 32$	$d9$
Conv. 3×3	LReLU	$32 \times 32 \times 64$	$d8$
Downsample	-	$16 \times 16 \times 64$	-
Conv. 3×3	LReLU	$16 \times 16 \times 64$	$d7$
Conv. 3×3	LReLU	$16 \times 16 \times 128$	$d6$
Downsample	-	$8 \times 8 \times 128$	-
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$d5$
Conv. 3×3	LReLU	$8 \times 8 \times 256$	$d4$
Downsample	-	$4 \times 4 \times 256$	-
Minibatch std	-	$4 \times 4 \times 257$	-
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$d3$
Conv. 4×4	LReLU	$1 \times 1 \times 256$	$d2$
FC	linear	$1 \times 1 \times 1$	$d1$

Table 3: Baseline Discriminator Architecture. FC - Fully Connected.

6.2.2 Baseline with Growing GAN

Similar to the approach of growing the latent space dimension in an MLP GAN, the bgGAN takes the bGAN, and adapts only layer $g1$ to the size of the latent vector by adding appropriately many input channels to each filter. The network starts with a latent space dimension of 8 and is trained until convergence. Then, the latent space dimension is doubled and layer $g1$ is appropriately grown, adding new trainable weights. In this manner, the latent space is grown in powers of two to dimension 256, i.e. $z_{dim} = \{8, 16, 32, 64, 128, 256\}$.

6.2.3 Channel Growing GAN

Extending on bgGAN, cgGAN involves not only changing the very first layer of the generator, but instead increasing the capacity of all layers in the generator and discriminator progressively during training. This yields a scheme where the number of input and output channels of each layer is grown simultaneously with the latent space growing. The idea is that specific filters in every layer are implicitly targeted to perform a certain task on a specific part of the latent code, which potentially helps in finding useful representations.

The architecture is given by Tab.4 and 5.

Generator	Act.	Output Shape	1	2	3	4	5	6	Name
Latent Vector	-	$1 \times 1 \times$	8	16	32	64	128	256	-
Conv. 4×4	LReLU	$4 \times 4 \times$	8	16	32	64	128	256	$g1$
Conv. 3×3	LReLU	$4 \times 4 \times$	8	16	32	64	128	256	$g2$
Upsample	-	$8 \times 8 \times$	8	16	32	64	128	256	-
Conv. 3×3	LReLU	$8 \times 8 \times$	8	16	32	64	128		$g3$
Conv. 3×3	LReLU	$8 \times 8 \times$	8	16	32	64	128		$g4$
Upsample	-	$16 \times 16 \times$	8	16	32	64	128		-
Conv. 3×3	LReLU	$16 \times 16 \times$	8	16	32	64			$g5$
Conv. 3×3	LReLU	$16 \times 16 \times$	8	16	32	64			$g6$
Upsample	-	$32 \times 32 \times$	8	16	32	64			-
Conv. 3×3	LReLU	$32 \times 32 \times$	8	16	32				$g7$
Conv. 3×3	LReLU	$32 \times 32 \times$	8	16	32				$g8$
Upsample	-	$64 \times 64 \times$	8	16	32				-
Conv. 3×3	LReLU	$64 \times 64 \times$	8	16					$g9$
Conv. 3×3	LReLU	$64 \times 64 \times$	8	16					$g10$
Upsample	-	$128 \times 128 \times$	8	16					-
Conv. 3×3	LReLU	$128 \times 128 \times$	8						$g11$
Conv. 3×3	LReLU	$128 \times 128 \times$	8						$g12$
Conv. 1×1	tanh	$128 \times 128 \times$	3						$g13$

Table 4: Generator architecture of cgGAN. The columns numbered $1-6$ refer to the channel depth at the given training cycle. For clarity of reading the table, if the channel depth remains constant, the space is left blank.

6.2.4 Layer Growing GAN

The lgGAN replicates the methodology used in [14], which has been shown to produce very high-quality images. The fully grown model is given, as previously discussed, by Tab.2 and 3. The lgGAN does not grow the latent space during training and the purpose of this model is to use it in comparison to the clgGAN. In lgGAN, the number of layers is grown progressively. This results in a model that first processes a small spatial resolution, which then gradually increases to the desired spatial size as more layers are added. The network starts with processing a spatial resolution of 4×4. The architecture is given by Tab.6 and 7. An additional subscript is added to the name of each layer. The subscript number signifies the pixel resolution that the architecture outputs. The real images are represented by 4×4 downsampled versions of the full 128×128 images. The GAN is trained until convergence and layers are added to the generator and discriminator to increase the spatial resolution to 8×8, given by Tab.8 and 9.

Discriminator	Act.	Output Shape	1	2	3	4	5	6	Name
Input Image	-	$128 \times 128 \times$	3						-
Conv. 1×1	LReLU	$128 \times 128 \times$	8						$d14$
Conv. 3×3	LReLU	$128 \times 128 \times$	8						$d13$
Conv. 3×3	LReLU	$128 \times 128 \times$	8	16					$d12$
Downsample	-	$64 \times 64 \times$	8	16					-
Conv. 3×3	LReLU	$64 \times 64 \times$	8	16					$d11$
Conv. 3×3	LReLU	$64 \times 64 \times$	8	16	32				$d10$
Downsample	-	$32 \times 32 \times$	8	16	32				-
Conv. 3×3	LReLU	$32 \times 32 \times$	8	16	32				$d9$
Conv. 3×3	LReLU	$32 \times 32 \times$	8	16	32	64			$d8$
Downsample	-	$16 \times 16 \times$	8	16	32	64			-
Conv. 3×3	LReLU	$16 \times 16 \times$	8	16	32	64			$d7$
Conv. 3×3	LReLU	$16 \times 16 \times$	8	16	32	64	128		$d6$
Downsample	-	$8 \times 8 \times$	8	16	32	64	128		-
Conv. 3×3	LReLU	$8 \times 8 \times$	8	16	32	64	128		$d5$
Conv. 3×3	LReLU	$8 \times 8 \times$	8	16	32	64	128	256	$d4$
Downsample	-	$4 \times 4 \times$	8	16	32	64	128	256	-
Minibatch std	-	$4 \times 4 \times$	9	17	33	65	129	257	-
Conv. 3×3	LReLU	$4 \times 4 \times$	8	16	32	64	128	256	$d3$
Conv. 4×4	LReLU	$1 \times 1 \times$	8	16	32	64	128	256	$d2$
FC	linear	$1 \times 1 \times$	1						$d1$

Table 5: Discriminator architecture of cgGAN. The columns numbered $1-6$ refer to the channel depth at the given training cycle. For clarity of reading the table, if the channel depth remains constant, the space is left blank.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 256$	-
Conv. 4×4	LReLU	$4 \times 4 \times 256$	$g1_4$
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$g2_4$
Conv. 1×1	tanh	$4 \times 4 \times 3$	$g3_4$

Table 6: lgGAN generator at resolution 4×4.

Discriminator	Act.	Output Shape	Name
Input Image	-	$4 \times 4 \times 3$	-
Conv. 1×1	LReLU	$4 \times 4 \times 256$	$d4_4$
Minibatch std	-	$4 \times 4 \times 257$	-
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$d3_4$
Conv. 4×4	LReLU	$1 \times 1 \times 256$	$d2_4$
FC	linear	$1 \times 1 \times 1$	$d1_4$

Table 7: lgGAN discriminator at resolution 4×4.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 256$	-
Conv. 4×4	LReLU	$4 \times 4 \times 256$	$g1_8$ (=$g1_4$)
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$g2_8$ (=$g2_4$)
Upsample	-	$8 \times 8 \times 256$	-
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$g3_8$
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$g4_8$
Conv. 1×1	tanh	$8 \times 8 \times 3$	$g5_8$

Table 8: lgGAN generator at resolution 8×8. The notation gn_8 (= gn_4) means that layer n in the 8×8 resolution model shares parameters with layer n in the 4×4 resolution model.

Discriminator	Act.	Output Shape	Name
Input Image	-	$8 \times 8 \times 3$	-
Conv. 1×1	LReLU	$8 \times 8 \times 128$	$d6_8$
Conv. 3×3	LReLU	$8 \times 8 \times 128$	$d5_8$
Conv. 3×3	LReLU	$8 \times 8 \times 256$	$d4_8$
Downsample	-	$4 \times 4 \times 256$	-
Minibatch std	-	$4 \times 4 \times 257$	-
Conv. 3×3	LReLU	$4 \times 4 \times 256$	$d3_8$ (=$d3_4$)
Conv. 4×4	LReLU	$1 \times 1 \times 256$	$d2_8$ (=$d2_4$)
FC	linear	$1 \times 1 \times 1$	$d1_8$ (=$d1_4$)

Table 9: lgGAN discriminator at resolution 8×8. The notation dn_8 (= dn_4) means that layer n in the 8×8 resolution model shares parameters with layer n in the 4×4 resolution model.

Note that layers $g3_4$ and $d4_4$ are not transferred to the grown network. These layers are, however, used in a smoothing operation in order to gently introduce the new layers, which is covered in Section 6.2.7.

6.2.5 Channel and Layer Growing GAN

The clgGAN combines the ideas from the cgGAN and the lgGAN in order to combine the benefits of both models by both growing the number of layers and channels progressively during training. In this way, the clgGAN grows the latent space dimension as the spatial size is increased, which enables specific filters at each layer to target coarse to fine features naturally.

Starting with a spatial resolution of 4×4 yields the network given by Tab.10 and 11. The grown network is then given by Tab.12 and 13.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 8$	-
Conv. 4×4	LReLU	$4 \times 4 \times 8$	$g1_4$
Conv. 3×3	LReLU	$4 \times 4 \times 8$	$g2_4$
Conv. 1×1	tanh	$4 \times 4 \times 3$	$g3_4$

Table 10: clgGAN generator at resolution 4×4.

Discriminator	Act.	Output Shape	Name
Input Image	-	$4 \times 4 \times 3$	-
Conv. 1×1	LReLU	$4 \times 4 \times 8$	$d4_4$
Minibatch std	-	$4 \times 4 \times 9$	-
Conv. 3×3	LReLU	$4 \times 4 \times 8$	$d3_4$
Conv. 4×4	LReLU	$1 \times 1 \times 8$	$d2_4$
FC	linear	$1 \times 1 \times 1$	$d1_4$

Table 11: clgGAN discriminator at resolution 4×4.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 16$	-
Conv. 4×4	LReLU	$4 \times 4 \times 16$	$g1_8$ ($\supset g1_4$)
Conv. 3×3	LReLU	$4 \times 4 \times 16$	$g2_8$ ($\supset g2_4$)
Upsample	-	$8 \times 8 \times 16$	-
Conv. 3×3	LReLU	$8 \times 8 \times 8$	$g3_8$
Conv. 3×3	LReLU	$8 \times 8 \times 8$	$g4_8$
Conv. 1×1	tanh	$8 \times 8 \times 3$	$g5_8$

Table 12: clgGAN generator at resolution 8×8. The notation gn_8 ($\supset gn_4$) means that a subset of layer n in the 8×8 resolution model shares parameters with layer n in the 4×4 resolution model.

Discriminator	Act.	Output Shape	Name
Input Image	-	$8 \times 8 \times 3$	-
Conv. 1×1	LReLU	$8 \times 8 \times 8$	$d6_8$
Conv. 3×3	LReLU	$8 \times 8 \times 8$	$d5_8$
Conv. 3×3	LReLU	$8 \times 8 \times 16$	$d4_8$
Downsample	-	$4 \times 4 \times 16$	-
Minibatch std	-	$4 \times 4 \times 17$	-
Conv. 3×3	LReLU	$4 \times 4 \times 16$	$d3_8$ ($\supset d3_4$)
Conv. 4×4	LReLU	$1 \times 1 \times 16$	$d2_8$ ($\supset d2_4$)
FC	linear	$1 \times 1 \times 1$	$d1_8$ ($\supset d1_4$)

Table 13: clgGAN discriminator at resolution 8×8. The notation dn_8 ($\supset dn_4$) means that a subset of layer n in the 8×8 resolution model shares parameters with layer n in the 4×4 resolution model.

For clarity, the growing process is shown again. The network is grown to resolution 16×16 and the architecture is shown in Tab.14 and 15. This growing recipe is repeated until resolution 128×128.

Generator	Act.	Output Shape	Name
Latent Vector	-	$1 \times 1 \times 32$	-
Conv. 4×4	LReLU	$4 \times 4 \times 32$	$g1_{16}$ ($\supset g1_8$)
Conv. 3×3	LReLU	$4 \times 4 \times 32$	$g2_{16}$ ($\supset g2_8$)
Upsample	-	$8 \times 8 \times 32$	-
Conv. 3×3	LReLU	$8 \times 8 \times 16$	$g3_{16}$ ($\supset g3_8$)
Conv. 3×3	LReLU	$8 \times 8 \times 16$	$g4_{16}$ ($\supset g4_8$)
Upsample	-	$16 \times 16 \times 16$	-
Conv. 3×3	LReLU	$16 \times 16 \times 8$	$g5_{16}$
Conv. 3×3	LReLU	$16 \times 16 \times 8$	$g6_{16}$
Conv. 1×1	tanh	$16 \times 16 \times 3$	$g7_{16}$

Table 14: clgGAN generator at resolution 16×16.

Discriminator	Act.	Output Shape	Name
Input Image	-	$16 \times 16 \times 3$	-
Conv. 1×1	LReLU	$16 \times 16 \times 8$	$d8_{16}$
Conv. 3×3	LReLU	$16 \times 16 \times 8$	$d7_{16}$
Conv. 3×3	LReLU	$16 \times 16 \times 16$	$d6_{16}$
Downsample	-	$8 \times 8 \times 16$	-
Conv. 3×3	LReLU	$8 \times 8 \times 16$	$d5_{16}$ ($\supset d5_8$)
Conv. 3×3	LReLU	$8 \times 8 \times 32$	$d4_{16}$ ($\supset d4_8$)
Downsample	-	$4 \times 4 \times 32$	-
Minibatch std	-	$4 \times 4 \times 33$	-
Conv. 3×3	LReLU	$4 \times 4 \times 32$	$d3_{16}$ ($\supset d3_8$)
Conv. 4×4	LReLU	$1 \times 1 \times 32$	$d2_{16}$ ($\supset d2_8$)
FC	linear	$1 \times 1 \times 1$	$d1_{16}$ ($\supset d1_8$)

Table 15: clgGAN discriminator at resolution 16×16.

6.2.6 Minibatch Standard Deviation

While the architectural strategy of growing layers and channels has been covered, some extra details are required to successfully train the models. This section presents a way that computes the minibatch standard deviation in order to increase image variation.

GANs have a tendency to only capture a subset of the modes present in the data distribution $p_{data}(x)$. *Mode collapse* ensues when GANs focus solely on one such mode, generating identical images for

all latent vectors, and is a result of unhealthy competition between the generator and discriminator. In order to alleviate this potential problem, *minibatch standard deviation* is proposed by [14]. Before describing the idea, the notion of a feature tensor needs to be introduced. A feature tensor is a multidimensional matrix consisting of four dimensions: (batch size, width, height, channels). Each row in any of the architectural tables describes feature tensors with shape (width, height, channels), omitting the batch size. Given a feature tensor, the minibatch standard deviation method calculates the standard deviation for each feature and spatial location over the minibatch. Then, all standard deviations are averaged resulting in one scalar value. This value is replicated to form a constant feature map which matches the spatial resolution (width, height) of the input feature tensor. The feature map is then concatenated at the end of the input feature tensor. The computations are visualized in Fig.3. This operation gives the discriminator an explicit tool to discover if mode collapse is imminent as the standard deviation would be low, while batches of real images have a higher standard deviation.

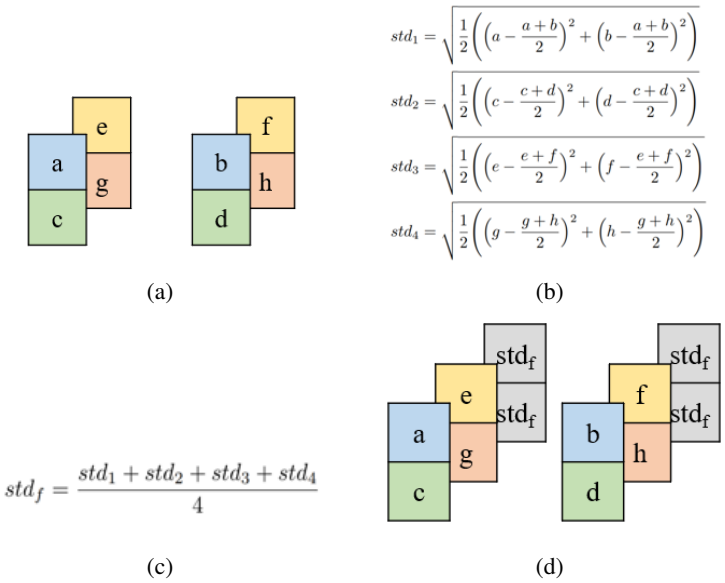


Figure 3: Computation procedure of minibatch standard deviation. (a) One minibatch consisting of two examples with spatial size 2×1 and channel depth 2. (b) The standard deviation is computed across the minibatch for each feature and spatial location. (b) The standard deviations are averaged. (d) The result is concatenated as an additional constant feature map to each example in the minibatch.

Some changes to this method are needed in order to make a smooth transition when growing the number of channels in the cgGAN and clgGAN. Consider the second channel in Fig.3a, i.e. the set $\{e, f, g, h\}$, to be the additional feature map as a result of increasing the number of filters at some layer. As the standard deviation of the new feature channel is not trustworthy in the beginning of training the grown network, a weighted average is constructed such that Fig.3c is replaced by

$$std_f = (1 - \tau)\frac{std_2 + std_2}{2} + \tau\frac{std_3 + std_4}{2}, \tag{15}$$

where $\tau \in \mathbb{R}$ is a smoothing parameter that starts at 0 at the beginning of training the grown network and linearly increases to $1/2$ during training. As a final detail, the filter in layer $d3$ that processes the last feature map as a result of minibatch standard deviation, is restored in the grown network such that the filter still processes the final feature map.

6.2.7 Smoothly Feeding in New Channels and Layers

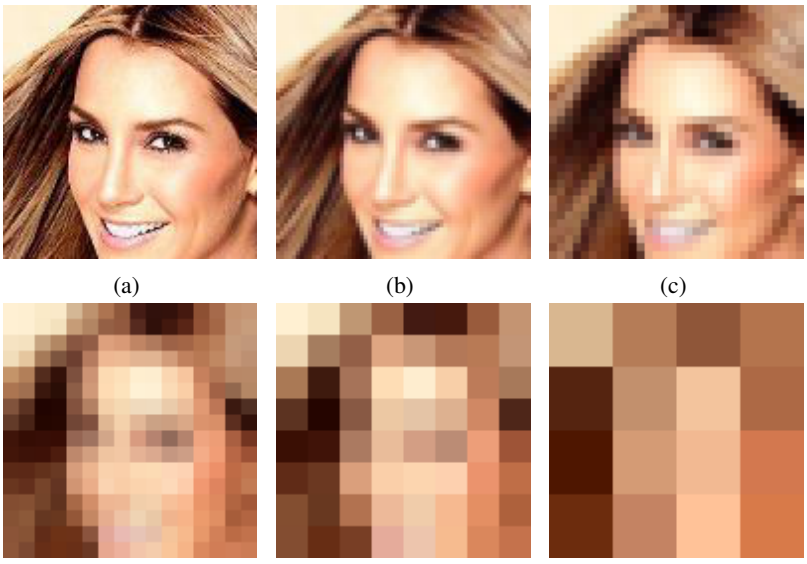
Another design detail that is necessary in order to obtain good results consists of certain operations to both feed in new weights smoothly when the network is grown, and to present the discriminator with finer details as training proceeds.

We start with introducing finer details as training proceeds for the bgGAN and cgGAN. As discussed

in Section 4.3, the dimension of the image manifold is, under mild regularity conditions, specified by the latent space dimension. For small latent space dimensions, the generator has trouble modeling all aspects in images at resolution 128×128, which span a manifold that is too complex for the given latent space dimension. This is because there is not enough capacity in a latent space with few dimensions to express the variation present in images of spatial size 128×128. In practice, this gives too much power to the discriminator and the generator never learns anything useful. To alleviate this problem, the real images are first downsampled to a smaller spatial resolution (causing information loss) and then nearest neighbor upsampled to spatial size 128×128 again. The level of downsampling depends on the current latent space dimension according to Fig.4. This scheme naturally encourages the filters of the network to first focus on learning coarse features in the images before improving their ability to model finer details.

For the lgGAN and clgGAN, the images are not downsampled and upsampled to resolution 128×128. Only downsampling to the appropriate resolution is needed.

There exists an additional trick, also on the topic of adapting the real images fed to the discriminator to the capacity of the network. As the network is grown, the finer resolution real images are smoothly introduced via a weighted average between the current resolution and the previous resolution. When introducing for example resolution 32×32, a snapshot is given by Fig.5. The parameter $\alpha \in \mathbb{R}$ is linearly increased from 0 to 1 during training.



(a) (b) (c)

(d) (e) (f)

Figure 4: An example image from the CelebA dataset at different resolutions. (a) Full 128×128 resolution at latent space dim. 256. (b) 64×64 resolution at latent space dim. 128. (c) 32×32 resolution at latent dim. 64. (d) 16×16 resolution at latent space dim. 32. (e) 8×8 resolution at latent space dim. 16. (f) 4×4 resolution at latent space dim. 8.

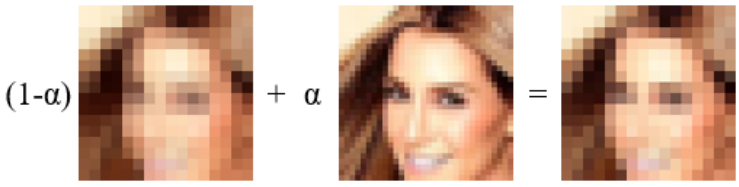


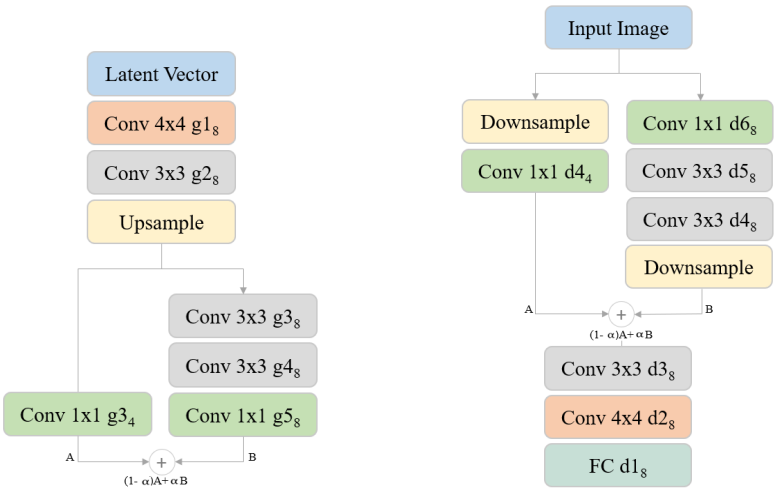
Figure 5: Smoothing scheme at $\alpha = 0.5$ when introducing finer resolution real images. In this example, resolution 32×32 is introduced.

The next training detail to consider is that the new weights need to be smoothly fed in when the network is grown. Contrary to the MLP GAN where the new weights are zero-initialized, the CNN GAN requires more careful initialization. The smoothing operation is done in two ways, depending

on if channels or layers are added during the growing process.

We start with growing the number of channels. This trick is thus used for the bgGAN, cgGAN and clgGAN. When the network is grown, it is crucial that the transition is smooth. If the new weights are given as much priority as the reused weights, the loss landscape will change too drastically and the solution found from the previous network cycle will not remain near a local optimum. To alleviate this problem, the new weights are multiplied at runtime with a smoothing factor $\beta \in \mathbb{R}$ that linearly increases from 0 to 1 during training, i.e. $\hat{w} = \beta \cdot w$, where w is the new weight, and $\hat{w}$ the weight that is used at runtime. When β is 0, the network is identical to the network at the last training iteration of the previous cycle. This trick guarantees that the reused weights serve as a good starting point in the new loss landscape.

Next, we move on to the smoothing operation when the number of layers of the network is grown. This trick is thus used for the lgGAN and clgGAN. Recall from Section 6.2.4 that layers $g3_4$ and $d4_4$ are not transferred to the grown network when the spatial size is increased from 4×4 to 8×8. These layers are, however, used in a smoothing operation in order to gently introduce the new layers. The technique builds on the intuition that the grown network should behave identically to the previous network at the initial step of training. An illustration of the smoothing operation is given by Fig.6a and 6b. The smoothing parameter $\alpha \in \mathbb{R}$ varies linearly from 0 to 1 during training. As a final



(a) (b)

Figure 6: (a) Smoothing operation of the generator using parameter $\alpha \in \mathbb{R}$. (b) Smoothing operation of the discriminator using parameter $\alpha \in \mathbb{R}$.

caveat, when introducing the new layers for the clgGAN, the process described by Fig.6 is slightly modified. The block for layer $g3_4$ is extended to handle 16 input channels instead of 8 and the block for layer $d4_4$ is extended to output 16 channels instead of 8. The new weights required for this modification are also gently introduced via the smoothing operation using β.

6.2.8 Architecture Details

To conclude Section 6, some final training details are given in this section.

A deterministic nearest neighbor upsampling method is used. This refers to replicating each pixel into a 2×2 grid which doubles the input spatially. Downsampling utilizes average pooling. Leaky ReLU with leakiness factor $c = 0.2$ (Section 4.4) is used for all layers except the final layer in the generator and discriminator.

The first layer of the generator is a 4×4 convolution with full padding, meaning that the input is implicitly padded with zeros as far as needed, and an output pixel is generated for every offset where there is an overlap between the input and the kernel. Thus, when the input is 1×1 spatially and the kernel is 4×4, the output is 4×4 too. This is equivalent to a fully connected layer without bias followed by a reshape and then adding the bias, because each of the values in the output tensor is a

unique linear combination of the input values.

All weights are initialized using the so-called He initializer [11]. This initialization procedure is adaptive with respect to the number of input nodes to the specific weight that is initialized.

7 Experiments

In this section, the experimental setup is described in detail as well as comments on the results. The section starts with experiments on a Swiss roll dataset, which covers training of an MLP GAN from scratch versus training it with a progressively growing latent space dimension. The section then proceeds with experiments on the CelebA dataset using the models described in Tab.1 and entails interpolations in the latent space.

The models were implemented in Tensorflow 1.5.0 and trained using Nvidia Geforce GTX Titan and Titan XP GPUs. The code can be found on Github.

7.1 MLP GAN on Swiss Roll

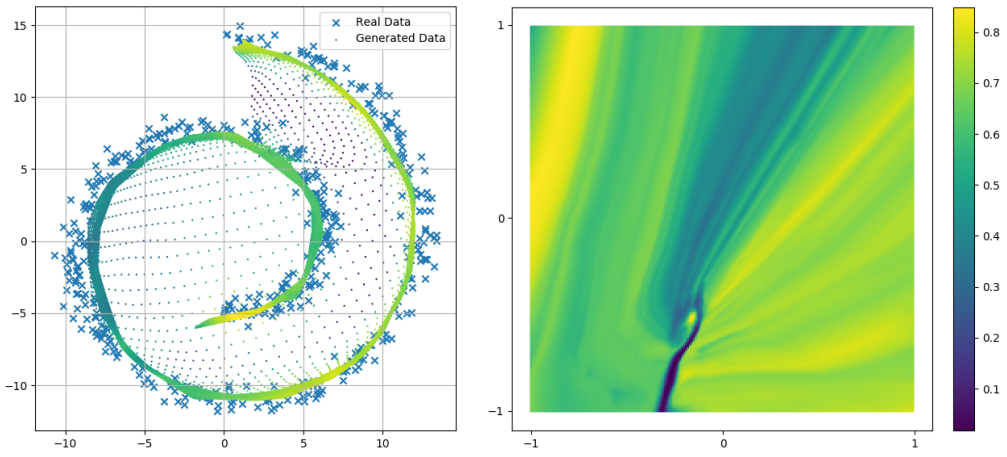
The experiments in this section use the 2D Swiss roll dataset available from the Scikit-learn library. For all experiments, Gaussian noise is added to the Swiss roll with standard deviation 0.5. The latent space distribution $p(z)$ is a uniform distribution from -1 to 1. Three experiments are conducted.

1. Training a standard MLP GAN with a learning rate that is too high. This experiment shows how the latent space behaves when the solution is in an optimum resulting in poor quality data from the generator. Fig.7a shows the generated dots, colored in the corresponding discriminator value. Fig.7b visually shows the structure of the 2-dimensional latent space where each point is colored by the corresponding discriminator value.
2. Training a standard MLP GAN with a well-tuned learning rate. This experiment shows how the latent space behaves when the solution is in an optimum resulting in good quality data from the generator. Fig.8a shows the generated dots, colored in the corresponding discriminator value. Fig.8b visually shows the structure of the 2-dimensional latent space where each point is colored by the corresponding discriminator value.
3. Training an MLP GAN by progressively growing the latent space dimension from 1 to 2 as described in Section 6.1. The result is given by Fig.9a and 9b. This experiment shows how the latent space behaves when the solution is in an optimum resulting in good quality data from the generator when training the latent space progressively.

One can note in Fig.7a that a local optimum that generates data, which is easily recognized as fake, leads to an entangled latent space in Fig.7b. An entangled latent space may thus not be preferable. In Fig.8a, when training a standard GAN, data more or less indistinguishable from real data is generated. Interestingly, from Fig.8b, the latent space is disentangled without having enforced this in any way during training. Traversing along a diagonal path in the latent space corresponds to walking along the Swiss roll and the diagonal direction thus encodes the direction of most data variation. While the major direction of variation is not aligned along the coordinate axis, a disentangled latent space seems preferable naturally. It is, however, not optimal to express the variation diagonally in the latent space. It is more natural to express the representation such that the variation is encoded along the coordinate axis. This is achieved by training the latent space dimension progressively according to Fig.9. By training the latent space progressively, the major variation in the data is focused from two dimensions to one. This can be seen as a form of dimensionality reduction, while also generating a representation that is more natural for downstream tasks.

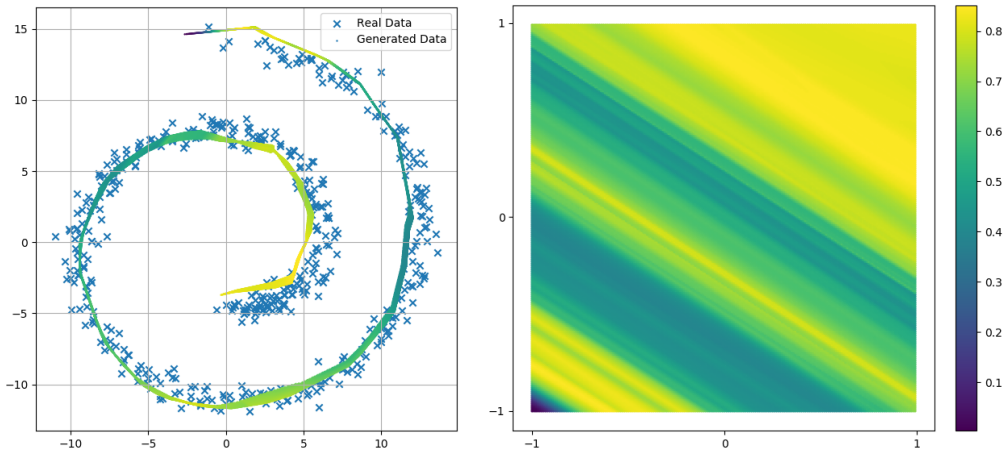
The training details are as follows. The architecture uses four hidden layers with 32 nodes each plus one output layer in both the generator and discriminator. Leaky ReLu is used as activation function with leakiness factor $c = 0.2$ (Section 4.4) for all layers except the output layers where linear activation is used. The optimizer is RMSprop and the minibatch size is 128. The non-saturated GAN loss function from (6) and (8) is used and in total, the discriminator is shown 1.92M real examples during training. This means that in experiment 3, the discriminator sees 1.92/2M real examples when trained using a 1-dimensional latent space and 1.92/2M real examples when trained

using a 2-dimensional latent space. For each minibatch, the generator and discriminator are updated ten times.



(a) (b)

Figure 7: MLP GAN trained from scratch. Learning rate: 5. (a) Generated dots colored in the corresponding discriminator color. The colorbar is in (b). Real crosses in blue. (b) Latent space discriminator values and corresponding colorbar. A local optimum that gives poor quality data leads to an entangled latent space.

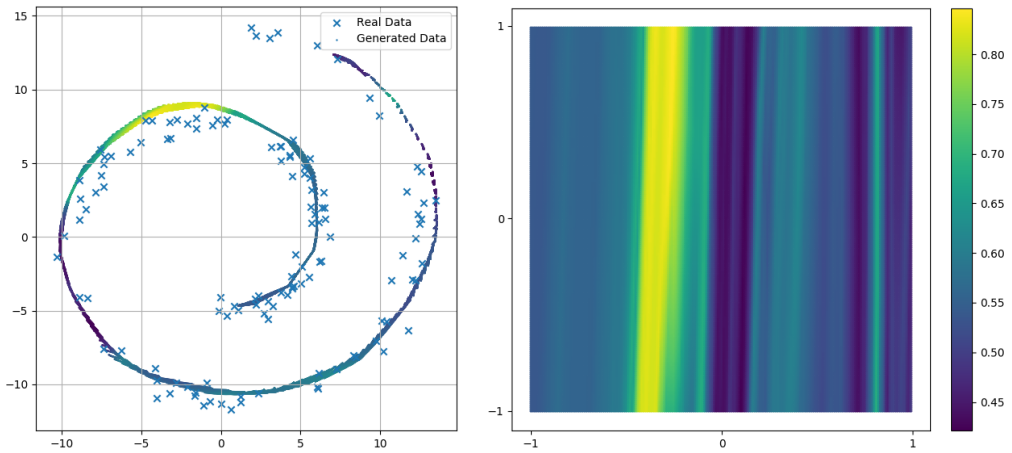


(a) (b)

Figure 8: MLP GAN trained from scratch. Learning rate: 0.001. (a) Generated dots colored in the corresponding discriminator color. The colorbar is in (b). Real crosses in blue. (b) Latent space discriminator values and corresponding colorbar. A local optimum that gives good quality data leads to a disentangled latent space. The direction along which the data varies most is, however, not aligned along the coordinate axis.

7.2 CNN GAN on CelebA

The experiments in this section use the CelebA dataset [18], which consists of 202,599 images of faces. Faces serve as a good starting point when analyzing our method since they provide both large semantic variation, meaning evaluation of our method is visually feasible, while still being a homogeneous class in regards to face placement and size, giving the best conditions for successful learning.



(a) (b)

Figure 9: MLP GAN trained progressively. Learning rate: 0.001. (a) Generated dots colored in the corresponding discriminator color. The colorbar is in (b). Real crosses in blue. (b) Latent space discriminator values and corresponding colorbar. The direction in the latent space corresponding to most data variation is aligned along the coordinate axis.

7.2.1 Latent Space Interpolation

All models are first successfully trained according to the training details given at the end of the section. The experiments are structured as follows. Each model in Tab.1 is evaluated using a latent space random walk protocol. First, recall that the training protocol encourages high attribute variability in the initial entries of the latent vector and gradually lower variability in the remaining entries. In order to test if this property can be shown empirically, the evaluation protocol performs random walks in a restricted latent space determined by specific sets of dimensions that are allowed to vary at a time. The sets of the latent space dimensions that are varied independently of each other are: $\{1-8, 9-16, 17-32, 33-64, 65-128, 129-256\}$. The sets reflect the latent space growing strategy, which starts with an 8-dimensional latent space and then multiplies the dimension by a factor of two for each growing procedure. The specifics of the protocol are as follows. First, a reference latent vector of size 256 is sampled. Then, dimensions $1-8$ are resampled four times. Now, five latent vectors, which only vary in dimensions $1-8$ exist, and linear interpolation is performed from one vector to the other. In the next stage of the protocol, instead of resampling dimensions $1-8$, dimensions $9-16$ are resampled, while keeping dimensions $1-8$ and $17-256$ fixed. In this manner, one can visually inspect the resulting images when interpolation is performed between the sampled vectors. For brevity, only the experiments where dimensions 1-8 and 129-256 vary, are included. Remaining experiments are available as gifs on Github.

We begin the discussion with our baseline models bGAN (a regular GAN) and lgGAN (the model of [14]). Fig.10 and 11 show the results for the bGAN using the interpolation protocol. Each row signifies one random walk between five samples from (a)-(e). In the same way lgGAN: Fig.12-13. It is clear that the variability is proportional to the number of degrees of freedom in the latent space, i.e. the more latent dimensions are varied, the more variability is achieved. This is also expected since the training protocol of these models does not enforce any particular structure on the latent space.

The results from the interpolation protocol for the bgGAN, cgGAN and clgGAN are given by Fig.14-15, Fig.16-17 and Fig.18-19 respectively. Most interestingly, the roles are now reversed compared to the bGAN and lgGAN case. It is clear that the variability is determined by what dimensions are altered, not the number of dimensions. Changing dimensions 1-8 clearly affects the variability more than changing dimensions 129-256. The effect is most evident for the clgGAN. This shows a successfully designed system where the variability is ordered by the dimensions of the latent space. This is analogous to the Swiss roll experiment in Fig.9b where the latent space dimensions are also structured in an order related to the variability of the generated data.

(a) (b) (c) (d) (e)

Figure 10: Generated faces using bGAN where, for each row, only dimensions 1-8 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 11: Generated faces using bGAN where, for each row, only dimensions 129-256 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 12: Generated faces using lgGAN where, for each row, only dimensions 1-8 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 13: Generated faces using lgGAN where, for each row, only dimensions 129-256 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 14: Generated faces using bgGAN where, for each row, only dimensions 1-8 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 15: Generated faces using bgGAN where, for each row, only dimensions 129-256 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 16: Generated faces using cgGAN where, for each row, only dimensions 1-8 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 17: Generated faces using cgGAN where, for each row, only dimensions 129-256 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 18: Generated faces using clgGAN where, for each row, only dimensions 1-8 in the latent vector differ between (a)-(e).

(a) (b) (c) (d) (e)

Figure 19: Generated faces using clgGAN where, for each row, only dimensions 129-256 in the latent vector differ between (a)-(e).

Some more comments on the bgGAN, cgGAN and clgGAN are in order. While changing the latent dimensions 1-8 induces more variability than changing dimensions 129-256 for all three models, the effect is least clear for the bgGAN. This may be due to the fact that only one of the filters of one layer of the bgGAN are progressively grown, meaning that most filters in the network can not be learned in a progressive manner. This yields a network that cannot assign certain aspects of the data to specific filters as easily as cgGAN, where the system has a better ability at assigning certain filters to discover coarse to fine features. The cgGAN has, however, problems of its own. While the cgGAN introduces a way to grow the latent space while appropriately adding capacity to the network, it always processes images of the same spatial size. Recall that the real images fed to the discriminator follow a downsampling-upsampling scheme which depends on the latent space dimension. The upsampling step consists of replicating the pixels to form a spatially larger image. As an example, to highlight a problem of the cgGAN, consider the case when the latent space is 8-dimensional and where the generator tries to replicate a real 128×128 image that has first been downsampled to spatial size 4×4 and then upsampled to spatial size 128×128. For the generator, this means that at best, the convolutional layers after resolution 4×4 will perform the identity mapping. This means that the convolutional layers, which process a spatial resolution larger than 4×4, are used and trained, but are in fact completely unnecessary. To solve this problem, the layers are grown using the clgGAN methodology. As the experiments in Fig.18 show, this network induces not only most variability in dimensions 1-8, but also achieves an image quality that outperforms

(a) (b) (c) (d) (e)

Figure 20: Generated faces using a bGAN with latent space dimension 8. Each row represents five distinct points, (a)-(e), between which linear interpolation is performed.

all other models. More experiments need to be conducted in order to determine if the improved image quality is consistent. In order to get a sense of the variability obtained from the clgGAN in Fig.18, a bGAN was trained using only an 8-dimensional latent space and faces were generated using linear interpolations in the latent space given by Fig.20. It is clear that higher variability is achieved in Fig.20 than in Fig.18, but this is at a significant cost regarding image quality. The result is also expected, since the bGAN traverses its entire latent space, while the clgGAN only traverses a restricted set of the 256-dimensional latent space.

7.2.2 Coordinate Axis Walk

In this section, coordinate axis walks are performed in order to evaluate the variation along specific dimensions in the latent space. Specifically, given a 256-dimensional latent vector, one coordinate is varied between $[-0.125, 0.125]$ and the resulting images are captured four times which yields Fig.21-26, where coordinate axes $1, 2, 5, 90, 195, 250$ are varied respectively. Points captured outside of the range $[-0.125, 0.125]$ did not yield accurate images, since the points are too unlikely in the latent distribution $p(z)$.

It is clear that salient attributes are captured by dimensions 1, 2 and 5. While the system is not designed to capture isolated semantic attributes in each dimension, changing dimension 90 seems to isolate the attribute determining how open the eyes are. Changing dimensions 195 and 250 induces very small changes to the hairstyle and minute background alterations. For better visualizations, please visit the Github page.

7.2.3 Training Details

The training details are as follows. A random seed of 547 is used for all experiments. Leaky ReLu is used as activation function with leakiness factor $c = 0.2$ (Section 4.4) for all layers except the output layers of the generator and discriminator. The optimizer is Adam with $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$ and the minibatch size is 16. The WGAN-GP loss function from (13) and (14) is used with $\lambda = 10$. For each minibatch, the generator and discriminator are updated once. In total, all models are trained for 44 epochs. For the models that grow the layers and/or channels, the initial network is trained for 4 epochs, while the grown networks are trained for 8 epochs. When the network is grown, the new layer and/or channel weights are smoothly introduced over 4 epochs by increasing $\beta : 0 - 1$, $\alpha : 0 - 1$ and $\tau : 0 - 0.5$ linearly over 4 epochs. The models are then trained for 4 more epochs to stabilize the weights. The distribution $p(z)$ of the latent space is a 256-dimensional Gaussian with identity covariance matrix that is normalized to unit length. This distribution is used regardless of the latent space dimension of the network. This means, if the latent space is 8-dimensional, 8 elements are extracted from the normalized 256-dimensional latent

(a) (b) (c) (d)

Figure 21: Generated faces using clgGAN where, for each row, only dimension 1 in the latent vector differ between (a)-(d).

(a) (b) (c) (d)

Figure 22: Generated faces using clgGAN where, for each row, only dimension 2 in the latent vector differ between (a)-(d).

(a) (b) (c) (d)

Figure 23: Generated faces using clgGAN where, for each row, only dimension 5 in the latent vector differ between (a)-(d).

(a) (b) (c) (d)

Figure 24: Generated faces using clgGAN where, for each row, only dimension 90 in the latent vector differ between (a)-(d).

vector. The training images are cropped to size 128×128 from the middle of the CelebA images and transformed to range $[-1, 1]$.

8 Discussion

In this section, a brief discussion on training time is given as well as a summary of the advantages to the proposed clgGAN.

An additional advantage of the clgGAN is that it trains faster than any of the other models in Tab.1. The clgGAN uses around 1000 times fewer parameters than the lgGAN at the beginning of training while also maintaining comparatively few parameters over all epochs. The number of parameters over epochs for all models in Tab.1 can be seen in Fig.27a and 27b in logarithmic and linear scale respectively.

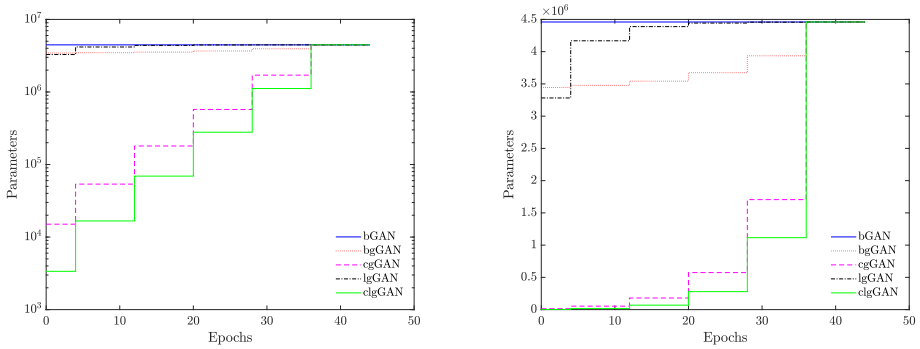
In summary, the clgGAN model offers several sought properties of a generative model.

(a) (b) (c) (d)

Figure 25: Generated faces using clgGAN where, for each row, only dimension 195 in the latent vector differ between (a)-(d).

(a) (b) (c) (d)

Figure 26: Generated faces using clgGAN where, for each row, only dimension 250 in the latent vector differ between (a)-(d).



(a) (b)

Figure 27: Parameter count over epochs for the five models. (a) Logarithmic scale. (b) Linear scale.

1. The image quality appears at least as good as the state of the art model given by [14].
2. The model trains faster than any of the other models in Tab.1.
3. The training scheme induces a beneficial structure on the latent space where the dimensions are ordered in regard to the saliency of the attributes that they encode. This structure could be useful for tasks like dimensionality reduction or transferring coarse to fine attributes from one image to the other, which could be seen as a form of style-transfer.
4. The entire training scheme is unsupervised and does not change the commonly used WGAN-GP loss function.

9 Future Work

The proposed strategy of growing the latent space dimension progressively during training appears to be a promising direction for future research. In this section, some aspects not covered in the thesis are discussed.

First and foremost, objective evaluation techniques are required in order to establish the quality of the images when using the proposed training protocol. The quality of the images can be evaluated using techniques like FID [12]. Orthogonal to evaluating image quality, it is important to measure the variability for each restricted set of latent space dimensions $\{1-8, 9-16, 17-32, 33-64, 65-$

$128, 129 - 256\}$. This can, for instance, be done by generating n images from each restricted set (keeping the other dimensions fixed) and calculating the standard deviation across the sample batch identical to calculating minibatch standard deviation in Section 6.2.6. This yields one scalar value for each set of dimensions, where a high number signifies high variability.

As discussed briefly in Section 8, an application of interest is to use the model for style-transfer. One could, in theory, choose to transfer coarse to fine features from a source image to a reference image by copying specific sets of the latent code from the source image to the latent code of the reference image.

An underlying idea behind the training protocol is to encourage filters early in training to focus on coarse features, while filters added later to the model are encouraged to focus on finer details. This implies that certain dimensions of the latent vector will implicitly encode coarse to fine details. Currently, we believe that the filters may mix more than necessary, which means that there is room for improvement on our method.

10 Conclusion

In conclusion, the purpose of this thesis is to show a potential direction for future research on learning interpretable representations. By gradually increasing the latent space dimension during training of a GAN, while appropriately adding capacity to the model, the dimensions of the latent space are ordered in regard to the saliency of the attributes that they encode. This structure could be useful for tasks like dimensionality reduction or style-transfer. The model also trains faster than and yields images of at least equal quality to state-of-the-art methods.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[2] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum, William T. Freeman, and Antonio Torralba. GAN dissection: Visualizing and understanding generative adversarial networks. *CoRR*, abs/1811.10597, 2018.

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. *CoRR*, abs/1809.11096, 2018.

[4] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.

[5] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *CoRR*, abs/1506.05751, 2015.

[6] Carl Doersch. Tutorial on variational autoencoders, 2016. cite arxiv:1606.05908.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[8] Ian J. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160, 2017.

[9] Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Bridging implicit and prescribed learning in generative models. *CoRR*, abs/1705.08868, 2017.

[10] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, abs/1706.08500, 2017.

[13] Irina Higgins, David Amos, David Pfau, Sébastien Racanière, Loïc Matthey, Danilo J. Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *CoRR*, abs/1812.02230, 2018.

[14] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[15] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[17] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

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

[19] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

[20] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan : Latent space clustering in generative adversarial networks. *CoRR*, abs/1809.03627, 2018.

[21] Michael A. Nielsen. Neural networks and deep learning, 2018.

[22] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[23] Tim Sainburg, Marvin Thielk, Brad Theilman, Benjamin Migliori, and Timothy Gentner. Generative adversarial interpolative autoencoding: adversarial training on latent space interpolations encourage convex latent distributions. *CoRR*, abs/1807.06650, 2018.

[24] Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning gan for pose-invariant face recognition. In *In Proceeding of IEEE Computer Vision and Pattern Recognition*, Honolulu, HI, July 2017.

[25] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

Master's Theses in Mathematical Sciences 2019:E28
ISSN 1404-6342

LUTFMA-3384-2019

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/