Conditional Wasserstein Generative Adversarial Networks

Cameron Fabbri
University of Minnesota
Computer Science and Engineering
fabbr013@umn.edu

Abstract

Deep neural networks have been shown to be powerful function approximators, achieving great success across many tasks. While much of the focus has been towards discriminative tasks, recent advances in generative models has brought a surge of research towards this area. Generative Adversarial Networks (GANs) have been able to model complex data distributions such as the set of natural images. While effective, they have been shown to be very hard to train in practice. This work demonstrates how an improvement to the GAN framework can be used in a *conditional* manner, able to restrict their generation according to some alternate information, such as a class label. Additionally, we explore this learned latent space in an informed manner, showing smooth translations between images and attributes.

I. Introduction

Dimensionality reduction is a well studied problem, with applications in many areas including clustering, classification, and latent space exploration through methods such as vector arithmetic or interpolation. Classical dimensionality reduction methods such as SVD and PCA are completely unsupervised. Recent supervised methods using neural networks, such as autoencoders, have been shown to represent data in a lower dimensional space with high accuracy. High dimensional data, such as images, are assumed to live on some natural manifold with properties that make such data much easier to explore. Many methods of image exploration are executed directly on the pixel space. These level of operations, such as image filters (*e.g.* conversion to grayscale, contrast change, sepia toning, etc.), require very little information pertaining to what is represented in the image. More complex, high level operations, such as the altering of class attributes (discrete attribute manipulation), or the changing of an object's size (continuous attribute manipulation), require a more informative representation of the image.

Directly operating on the pixel space (in a fully automated manner) would give unacceptable results, due to the fact that these methods must have some sense of knowledge about the image. Low level image manipulation methods, such as sepia toning, do not need any knowledge of what the image contains. On the other hand, if we were to try and add glasses to an image of a person, inserting glasses into the pixel space would give no guarantees as to whether or not they were correctly placed on the person's face. In order to perform such an operation, we consider introducing alterations in a lower dimensional space, and then generating the new image.

Generative models that are able to accurately capture a complex data distribution have been proven to be very powerful. The most recent wave of success has come with the development of Generative Adversarial Networks (GANs) [1], which aim to approximate the probability distribution function from which some data originates from. This allows the direct sampling from this distribution in order to generate similar, but new data. Although well defined in theory, their use in practice becomes troublesome (see [2] for a very detailed review). There have been a large number of improvements to the vanilla GAN, which we give a brief review of in Section II-E. A simple yet notable extension of the GAN is the conditional GAN (cGAN) [3], which conditions the generation on some extra information, such as a class label. This paper focuses on a variant of the GAN formulation, namely the Wasserstein GAN (WGAN) [4], [5] in a conditional sense. We show that we are able to control image generation conditioned on both discrete and continuous attributes, as well as greatly stabilize training with the use of WGANs.

II. Background

This section serves to give a review of neural networks, convolutional neural networks, and various types of generative models. This work focuses on Generative Adversarial Networks, which are outlined in Section II-E.

A. Neural Networks

To understand a neural network, one must first understand the concept of a perceptron. A perceptron is a single node which computes the summation of its inputs multiplied by a set of trainable weights. This is simply a linear combination of its inputs, and outputs a binary decision function based on whether or not its value passes some threshold. Because many real world problems are extremely nonlinear in nature, the concept of a perceptron evolved to that of a *neuron*. A neuron is similar to a perceptron, but has an additional bias term, as well as an activation function, which adds some form of non-linearity. Figure 1 shows a simple neuron. A feed-forward neural network consists of multiple layers of neurons interconnected in a feed-forward fashion, often referred to as a multi-layered perceptron (MLP). All networks have an input layer and an output layer. Intermediate layers in a network are referred to as *hidden layers*, because they are not directly connected (visible) to outside information.

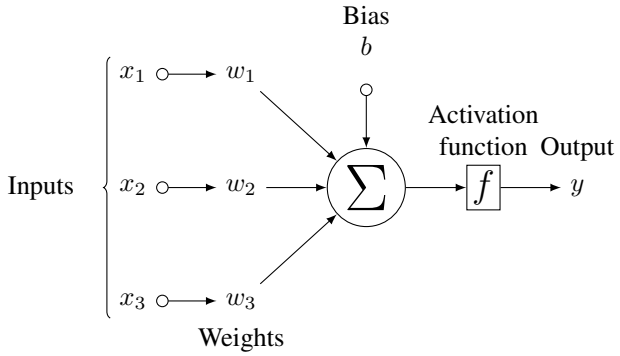

Fig. 1: A single perceptron.

Historically, the sigmoid function was used as a common activation function because it is able to squash a real-valued input to a range between 0 and 1, which is useful in computing probabilities. Nowadays, due to problems with the sigmoid such as the vanishing gradient, the most popular activation function is the Rectified Linear Unit (ReLU) [6]. The ReLU function is a piecewise linear function with two linear pieces, calculated as $f(x) = max(0, x)$. Due to it being nearly linear, it preserves many properties that make linear models easy to optimize with gradient-based methods [7]. Despite the major improvement ReLUs provided, a slight modification to this function, called a Leaky ReLU (lReLU), has seen popular use as well. Often times in a neural network, a ReLU node can "die", which simply means it always outputs a value of 0. When this happens, it is very unlikely to recover because the gradient at 0 is 0, so learning in that neuron will stop. To help alleviate this, the lReLU function is computed as $f(x) = max(\alpha x, x)$, for some $0 < \alpha < 1$.

B. Deep Neural Networks

Deep learning is a class of neural networks that use many *hidden layers* between the input and output to learn a hierarchy of concepts, often referred to as *deep neural networks* (DNN). As stated in II-A, in a feedforward network, each successive layer uses the output from the previous layer as input, then uses an activation function to obtain a new representation of the input. The output of the network is compared to some given label accoring to some cost function. While this difference is quite intuitive, given any intermediate hidden layer, it becomes less obvious what the output of that layer should be. In order to learn the set of weights and biases connecting successive layers, the error is propagated backwards through the network from the output, in order to optimize a given objective function. This process is known as *backpropagation* [8]. Backpropagation is used in conjunction with an optimization method, such as gradient descent, to efficiently compute the gradients by propagation from the output to input (see *e.g.* [8], [9] for a more in depth review). This allows multi-layer networks to learn a non-linear mapping from a large amount of data. Computational limitations do not allow us to simply optimize this function all at once, and therefore data is often fed into the network in *batches*. Stochastic gradient descent can be used to approximate the true gradient of all the data given a random mini batch. Recent advances in GPUs have provided massive speedups in training due to their ability to parallelize these operations.

C. Convolutional Neural Networks

The most common type of DNN for visual data is the Convolutional Neural Network (CNN), which is designed specifically for multidimensional data with *spatially correlated features*. CNNs incorporate three powerful techniques, which together provide scalability, and some degree of scale and shift invariance. The first is the use of shared weights, which stems from the idea that a feature detector used in one part of an image is almost certainly useful in other parts of the image. This also allows networks to reduce the number of parameters to avoid the curse of dimensionality. The second is the use of local receptive fields. A kernel (sometimes called a filter) is *convolved* across the entire image to produce a *feature map*. Each pixel in the resulting feature map is the result of the kernel convolved with a small area in the input. The use of local receptive fields allow earlier layers in the network to learn low-level features such as edges or corners, which can then be combined in successive layers throughout the network to learn high-level features. The third technique is various forms of subsampling. Convolutions with a stride > 1 inherently subsample, as the resulting output is of a lower dimensionality from the input. Other subsampling techniques, such as the use of pooling layers, provide a strict form of *linear* downsampling. The most common technique, maxpooling, takes the max of an $n \times n$ region of a feature map. One way to interpret this is pushing the most informative information forward.

While the size of the output in a MLP is independent of the size of its input, the produced feature map in a CNN depends on the kernel size and stride. Additionally, padding is often used to pad the input with zeros, simply to allow for spatial dimensions to agree during downsampling. However, the number of feature maps or *depth* of the resulting layer (which corresponds to the *width* of the network as a whole) is arbitrary. While there are many options for architecture design, many rely on heuristics, as well as some theoretical design principles as shown in [10].

D. Deep Generative Models

Since the advent of Alexnet [11], most of the focus has been put on CNNs as a discriminative model, learning a function to map some input data to some desired output label. Only until recently have generative models been more of the focus. Autoencoders, Deep Boltzmann Machines, and Deep Belief Nets are some examples of these class of models. Generative models are most assumed with that of *unsupervised learning*, where we do not have labels for our data. One primary goal in unsupervised learning is to be able to learn a representation powerful enough to accurately

represent the data. Other methods of unsupervised learning are more focused towards the *generative* side, where the goal is to be able to generate a new but similar data point. One issue facing generative models is the need for some sort of randomization somewhere in the model in order for the output to be random (we want to generate something *new*, not just sample from the data we already have) . Because we cannot backpropagate through a random layer in a network, the *reparameterization trick* [12] is introduced. Rather than involving some sense of randomness in our model, we can simply sample randomly from some distribution, and use that as input. More concretely, we can have some generator network G, that takes as input a random variable z sampled from some distribution, *e.g.* $z \sim \mathcal{N}(0, 1)$, and outputs a sample x.

Generative models have been shown to perform well on many tasks such as inpainting [13], image denoising [14], video prediction [15], super resolution [16], volume rendering [17], paired image-to-image translation [18], and unpaired image-to-image translation [19]. Autoencoders have been a popular form of generative model because of their ability to approximate some distribution of observed data, such as imagery. A simple auto-encoder setup may be to learn an encoding z over a set of images, $\mathcal{X}$, with $z \in \mathbb{R}^k$, $X^d \in \mathcal{X}$, with $k < d$. An image $X \in \mathbb{R}^{m \times n}$ is encoded through an encoder CNN to some latent variable $z \in \mathbb{R}^k$, which is then used as input to a decoder CNN to produce $X' \in \mathbb{R}^{m \times n}$. A loss function, such as L_2, is then used to compare the two, and the error is propogated back through the networks. To produce a new sample, a random variable z' can be used as input to the decoder network in place of an encoded z. One obvious issue is the lack of knowledge about this encoded space, making sampling from it difficult. The work of [12] showed that by assuming it is approximately Gaussian, and using that as a constraint on the encoding, we can more easily sample for z'.

An open research problem is to determine an appropriate loss function. While L_2 and L_1 are popular choices, they often result in blurry images, and may not even be a correct metric for visual quality. For example, a small shift in the pixelspace may result in no difference to the human eye, but would have a large loss in the L_2 space. A recent class of generative models proposes to instead use an *adversarial* loss in place of existing loss functions in the form of a neural network.

E. Generative Adversarial Networks

Generative Adversarial Networks (GANs) [1] represent a class of generative models based on a game theory scenario in which a generator network G competes against an adversary, D. The goal is to train the generator network to generate samples that are indistinguishable from the true data, p_{data}, by mapping a random input variable $z \sim p_z$ to some x. This mapping can be represented as $G(z; \theta_g)$, where G is a MLP with weights θ_g, and z is a random variable sampled from some distribution, *e.g.* $z \sim \mathcal{N}(0, 1)$. The discriminator, $D(x; \theta_d)$, is represented by a second MLP with weights θ_d, and outputs a scalar representing the probability that a sample x came from p_{data} rather than from G. The two networks are trained simultaneously, with D being trained to maximize the probability of correctly distinguishing whether or not a sample came from p_{data} or from G, and G being trained to fool D by minimizing $log\ (1 - D(G(z)))$. This can be represented by the following objective function:

$$\min_G \max_D \mathbb{E}_{x \sim p_{data(x)}}[log\ D(x)] + \mathbb{E}_{z \sim p_z(z)}[log\ (1 - D(G(z)))] \tag{1}$$

In practice however, early on in training D will be able to easily discriminate between real and generated samples. This is due to the fact that G's weights are randomly initialized, and the generated image is conditioned only on input noise. Images generated early in training obviously will essentially be noise, which will clearly look very different than those from the true data set. The discriminator will easily be able to differentiate between real and generated samples, causing $log\ (1 - D(G(z)))$ to saturate. Therefore, in practice it is more common to instead train G to maximize $log(D(G(z)))$, which is the same as minimizing $log(1-D(G(z)))$, but with much stronger gradients [1].

Conditional GANs (cGANs) [3] introduce a simple method to give varying amounts of control to the image generation process by some extra information y (e.g a class label). This is done by simply feeding y to the generator and discriminator networks, along with z and x, respectively. The new objective function then becomes:

$$\min_G \max_D \mathbb{E}_{x,y \sim p_{data(x,y)}}[log\ D(x, y)] + \mathbb{E}_{z \sim p_z(z), y \sim p_{data}(y)}[log\ (1 - D(G(z, y)))] \tag{2}$$

Conditional GANs have been shown to work well not only with attributes, but also by conditioning on an image itself to perform image-to-image translation [18].

GANs aim to approximate the probability distribution function that some data is assumed to be drawn from. The original formulation in [1] showed that treating the discriminator as a classifier minimizes the Jensen-Shannon (JS) divergence between the true data distribution and the one assumed by the generator. This shows to be extremely unstable and difficult to train in practice. See [2] for a detailed review. Much of the recent work has been focused towards improving the stability and quality of GANs by improving the quality of the gradients from the discriminator (*i.e.* using a loss function other than classification). Energy-based GANs [20] view the discriminator as an energy function which attributes low energy near the data manifold and high energy elsewhere. This is a particularly interesting approach, as it models the discriminator as an autoencoder. This means that just given the true data, the discriminator would be able to learn the distribution on its own, whereas the discriminator in a vanilla GAN would not, because a one class classifier does not make sense. Least Squares GAN [21] (LSGANs) adopts a least squares loss function for the

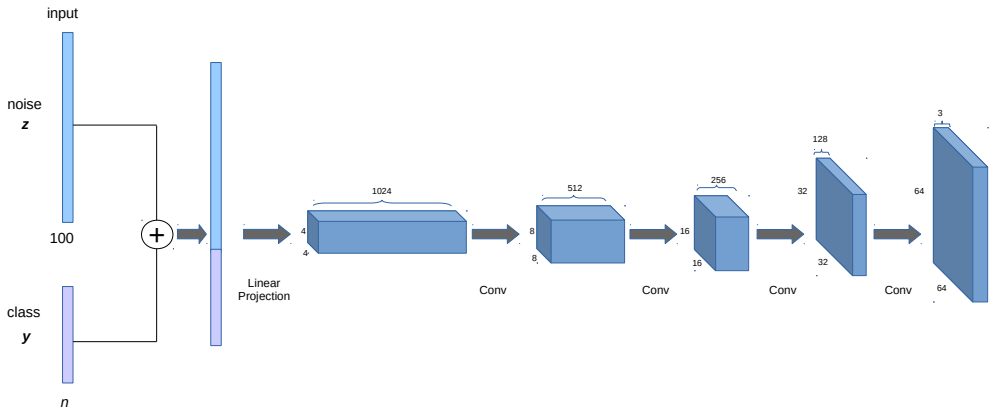

Fig. 2: Our generator architecture. A 100 dimensional vector z is randomly sampled from a normal distribution and concatenated onto an attribute vector y. The result is projected into a convolutional representation. We use transpose convolutions in order to upsample to $64 \times 64 \times 3$.

discriminator, which was shown to minimize the Pearson $\mathcal{X}^2$ divergence. Fisher GAN [22] defines the discriminator as an Integral Prabability Metric which discriminates between two probability distributions. The Wasserstein GAN [4] (WGAN) shows that the Wasserstein-1 distance, or Earth Mover (EM), may be a more appropriate distance in which to measure the two distrubutions. For this work, we focus on an improved method for the Wasserstein GAN, and show a comparison with the vanilla GAN.

1) Wasserstein GANs: It has been shown that even in very simple scenarios the JS divergence does not supply useful gradients for the generator [4]. On the other hand, the EM distance does not suffer from these problems of vanishing gradients. The EM distance or *Wasserstein-1* is defined as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \quad (3)$$

where $\prod(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of joint distributions $\gamma(x, y)$ whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively [4]. Because the infimum is very troublesome, [4] instead proposes to approximate W given a set of K-Lipschitz functions f by solving the following:

$$\max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(G_\theta(z))] \quad (4)$$

Such a function f_w can be found by training a neural network (modeled as the discriminator). To ensure f_w is K-Lipshitz, the weights are clamped to some range (*e.g.* [-0.01, 0.01]), after each gradient update. Two interesting things come from this formulation: 1) The loss reported by the generator and discriminator correlate well with image quality, and 2) unlike the original GAN, where the updates to the generator get worse as the discriminator gets better [2], WGAN actually relies on a perfect discriminator. For this reason, the gradients of the discriminator are updated n times for every 1 update to the generator ($n = 5$ is common in practice). While providing stability in training, this comes with the cost of long training times. Furthermore, [5] showed that the weight clipping leads to potential instability in the discriminator, as well as severely limiting its capacity.

2) Improved Wasserstein GANs: In order to enforce the Lipschitz constraint without clipping the weights of the discriminator, [5] instead penalizes the gradient, leading to a WGAN with gradient penalty (WGAN-GP). More formally, a function is 1-Lipschitz only if it has gradients with a norm of 1 almost everywhere. In order to enforce this, WGAN-GP directly constrains the norm of the discriminator's output with respect to its input. This leads to the new objective function:

$$\max_{w \in W} \mathbb{E}_{z\sim p(z)}[f_w(G_\theta(z))] - \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] + \lambda \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}} f_w(\hat{x})||_2 - 1)^2] \quad (5)$$

where $\mathbb{P}_{\hat{x}}$ is defined as sampling uniformaly along straight lines between pairs of sampled from the true data distribution, $\mathbb{P}_r$, and the distribution assumed by the generator, $\mathbb{P}_g = G_\theta(z)$. This is motivated by the intracability of of enforcing the unit gradient norm constraint everywhere. Because the optimal discriminator consists of straight lines connecting the two distributions (see [5] for more details), the constraint is enforced uniformaly along these lines. As with the original WGAN, the discriminator is updated n times for every 1 update of the generator.

The rest of this paper considers the original GAN and WGAN-GP in the *conditional* sense for directed image alteration and latent space exploration. Expanding on [23], we show the use of WGAN-GP provides more stable training, as well as higher quality generations.

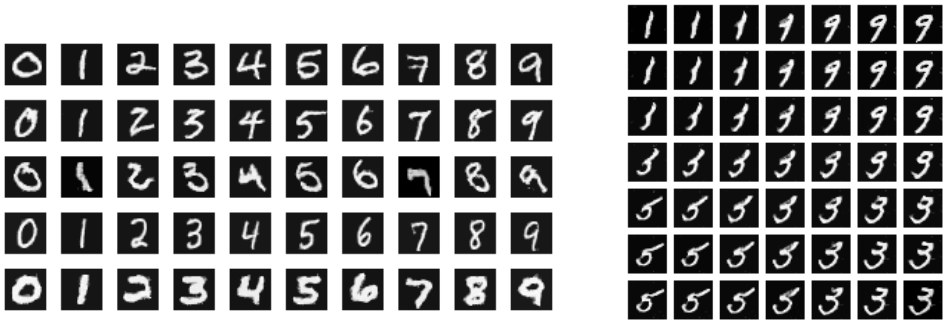

Fig. 3: Left: MNIST styles with attribute swapping. Right: Interpolation between z and y for four generated images (corners).

III. Related Work

Generative models have been showing great success since the advent of GANs. Prior to GANs, a Variational Autoencoder (VAE) was the most popular generative model, which imposes a prior on the encoding space z in order to constrain the space to sample from. One issue facing this approach is the pixel-wise reconstruction error, which introduces blurry images. The work of [24] replaced this reconstruction error with learned features representations from a GAN. Their model was also able to condition on class attributes, showing impressive results. An Adversarial Autoencoder (AAE) was proposed by [25], which is a probabilistic autoencoder that uses a GAN to perform inference. CVAE-GAN [26] combines a VAE with a GAN for synthesizing images in fine-grained categories, such as the face of a specific person. In the work of [27] uses the discriminator of a GAN to also predict the class label of an image, as well as the probability of it being a real or generated image.

This work is an extension of the work done by [5], introducing the Improved Wasserstein method into the cGAN framework. The result of this is a much more stable training environment and higher quality samples for conditional image generation using GANs. The rest of the paper is set up as follows. Section IV discusses the methodology used. Section V shows experiments on three datasets, one not originally shown in [23]. Section VI goes into discussion for how to interpret the results, and finally we conclude in Section VII.

IV. Methodology

Our method is based on the Wasserstein GAN [4], using the gradient penality for ensuring the 1-Lipschitz contsraint of the discriminator [5]. By introducing this into the conditional GAN setting, we are able to alter class attributes for controlled image generation.

A. Conditional GAN

A conditional GAN (cGAN) is very similar to a regular GAN. In its simplest form, some extra information y is concatenated onto z such that the generator is conditioned to include this information in the generation process. In a similar fashion, this information y is also included in the input to the discriminator. It's an unsolved problem as to what the "best" way to include conditional information into a neural network is. Many approaches exaustively search for how this information should be combined, or where in the network it should be included. One approach, which is the one we take in this work, is to include y in the input layer for both the generator and the discriminator. This is a natural choice because, while not having any quantitative measures, one can assume that introducing y earlier in the network will have a greater impact due to the network having more changes to learn from it (*i.e.* information from y will be contained in more layers). Other approaches which we do not explore include introducing y at *every* layer in both networks.

B. WGAN-GP

We chose to use the Improved Wasserstein GAN with Gradient Penalty outlined in Section II-E.2, as it provides more stability while training and overall higher image quality. For all of our experiments, we use the Adam optimizer [28] with parameters $\beta_1 = 0$, $\beta_2 = 0.9$, and a learning rate of $\alpha = 1e-4$. The generator and discriminator are trained in tandem, with the discriminator being updated $n = 5$ times for every generator update.

C. Architecture Details

Our architecture is based on that of [29], and we show the generator in Figure 2. The attribute vector y is concatenated onto z, which is projected into a convolutional representation of size $4\times4\times512$ (this can be thought of as a fully connected layer without an activation function). The rest of network is fully convolutional, and contains batch normalization [30] and a ReLU activation function [6] after each convolution, with an exception of the output layer, which does not use batch norm and uses a TanH activation function. Each convolutional layer uses a kernel size of 5×5 and stride 2×2. For upsampling we chose to use transpose convolutions.

The discriminator network mostly mirrors the generator network with a few exceptions. Due to the gradient penalty with respect to the input, no batch normalization is used. Leaky ReLU activations are used after each layer, except the output which does not have an activation function. Each convolutional layer uses stride a kernel size of 5×5 and stride 2×2, except for the last layer which uses a kernel size of

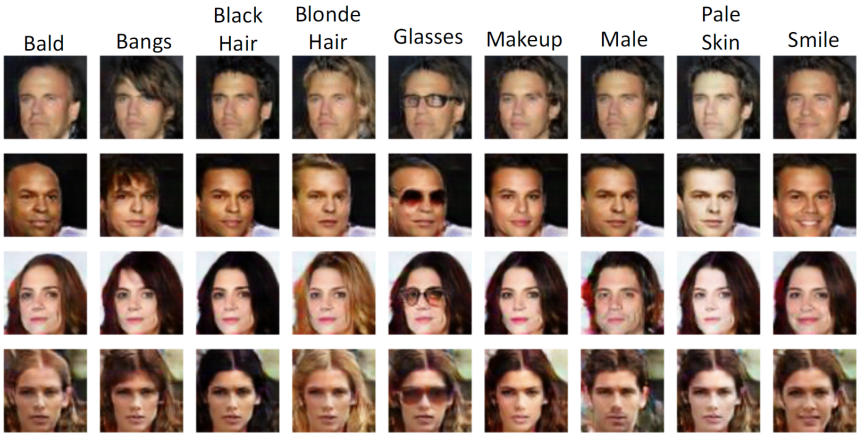

Fig. 4: Swapping CelebA attributes. Each row contains the same z vector, and each column a different y activation for each attribute. The top two rows were generated with the male flag active, and the bottom two with the female flag active.

4×4 and stride 1×1. The resulting output score from the discriminator is of size $4 \times 4 \times 1$, which we take the mean of. In order to to include the d dimensional attribute vector, we concatenate it onto the channel axis of the input image. The resulting input to the first convolutional layer is then $m \times n \times (d+3)$, where m is the image height, n is the image width, and d is the number of attributes in consideration. Images are normalized to a range $[-1, 1]$. Training took approximately 24 hours on a GTX 1080. Implementation was done using the Tensorflow library [31]. [1]

V. Experiments

A. Datasets

We first experiment on the MNIST dataset [32], which is a collection of handwritten digits. We use the official train test splits (including the validation in train). There are $60,000$ items in the train split, and $10,000$ in the test split. Each image has an associating attribute vector corresponding to what number it is, which we represent with a one-hot vector, $n = 10$. The second dataset we use is CelebA [33], which is comprised of just over 200,000 celebrity faces. We use the aligned and cropped version, and further crop and resize to $(64, 64)$. This dataset consists of 40 attributes, for which we choose $n = 9$, specifically: bald, bangs, black hair, blonde hair, eyeglasses, makeup, gender, pale skin, and smiling. For this we use the official train and validation splits as our training data ($\sim 182,000$ images) and test split ($\sim 20,000$ images). The third and final dataset we consider is the EFIGI Galaxy dataset [34]. This dataset consists of 4458 nearby galaxies with detailed morphologies such as arm strength, visible dust, and spiral arm curvature. Of the attributes, we choose $n = 4$ of them, all of which are continuous, unlike the attributes of the MNIST and CelebA datasets. This shows to be a much more difficult problem. We train all of our networks for 100 epochs, and randomly sample z from a normal distribution with $\mu = -1$ and $\sigma = 1$, *i.e.* $z \sim \mathcal{N}(-1, 1)$.

[1] https://github.com/cameronfabbri/cWGANs

B. MNIST

The generator network is a function $G : z \bigoplus y \in \mathbb{R}^{110} \rightarrow \mathbb{R}^{28 \times 28}$, where $\bigoplus$ represents concatenation. We first test the effect of changing y (the attribute) while keeping z constant, for which results can be seen in Figure 3 (left). What is observed is z controlling the style of writing, while y clearly controls the digit class. Note that these are not images from the train or test split, but completely generated by the network.

One way we can explore the latent space computed by the generator is through interpolation. Figure 3 (right) shows interpolating between 4 images. This shows how we can "walk" along the manifold to display its smoothness. We generate the four corners of the grid, then interpolate between both the z and y vectors. Although the network was only trained on binary attributes for y, interpolation results show that the computed latent space is smooth between attributes, and not simply discrete to those shown in the training set.

C. CelebA

For CelebA we use the same sampling method for z as with MNIST. The CelebA dataset lives in a much larger space than MNIST, as it also contains color. In this case, the generator network is a function $G : z \bigoplus y \in \mathbb{R}^{109} \rightarrow \mathbb{R}^{64 \times 64 \times 3}$. We also have the interesting property of having cases where more than one attribute can be active (*e.g.* a person can be smiling and have blonde hair), whereas with MNIST only one digit label will be active at a time. This adds additional complexity in that the network should learn that combinations such as bald and bangs should not be viable. In Figure 4 we show attribute alteration across generated faces. In each case, the activated attribute is clearly seen in the output image. This qualitative result displays the ability of the network to learn the representation of attributes as they

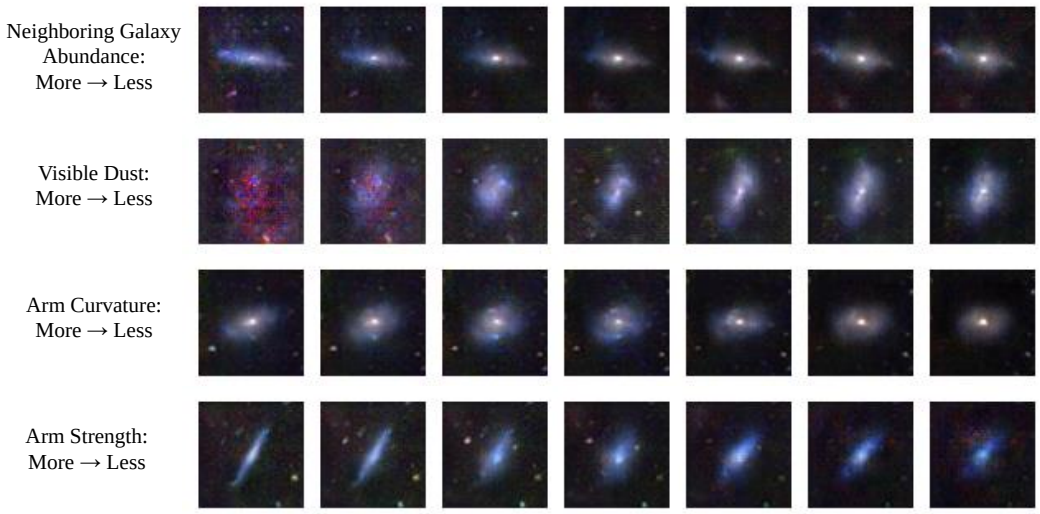

Fig. 5: The result of exploring the latent space for the EFIGI galaxy dataset. For each row we keep z fixed and linearly interpolate between two continuous attributes.

affect faces. Interesting combinations such as male + makeup that were not seen in the dataset look completely realistic.

D. EFIGI

The EFIGI dataset [34] is a set of 4,458 annotated images of galaxies. Of the attributes, we choose 4 due to their strong visual correlations: Arm strength, arm curvature, visible dust, and abundance of neighboring galaxies. All of these are *continuous*, as opposed to the discrete attributes seen in MNIST and CelebA. The generator is then a function $G : z \bigoplus y \in \mathbb{R}^{104} \rightarrow \mathbb{R}^{64\times64\times3}$.

Due to the fact we are only considering continuous attributes, visualization becomes easier through interpolation, which results in a smooth transition between images (see Figure X). These qualitative results show the learned latent space is smooth and able to generalize to information not seen during training. A more quantitative result would be, for example, to be able to then predict these attributes given the image, but we leave this for future work.

VI. Discussion

In this section we discuss the implications of a learned latent space. Informative latent spaces can be very powerful, as they provide a means to intelligently sample new data. Ideally, we want a smooth manifold that is able to capture the inherent structure of the data, as opposed to merely memorizing it. GANs have been shown to exhibit very expressive latent spaces, able to perform meaningful vector arithmetic [29]. In the case of conditional GANs, this space must not only capture relations between images, but their attributes as well. Our goal is for the probability distribution approximated by the generator to be smooth enough to capture 'holes' in the dataset (*e.g.* male + makeup attribute).

The CelebA dataset contains only binary attributes associated with each image (*i.e.* there is no 'partially bald' attribute). One may expect the latent space to be partially disjoint to these attributes, only able to capture local 'spikes' in the information according to the attributes provided during training. What we see is quite the opposite. Figure X shows interpolation along the bald attribute, giving a very smooth progression. Although the network was never given information about a partially bald attribute (or a continuous attribute of any type for that matter), the learned representation is able to smoothly transition from one attribute to another. Further latent space exploration can be seen in the Appendix.

Fig. 6: Interpolation along the bald attribute. Although the network was never trained on the attribute of 'partially bald', it is able smoothly interpolate towards it (middle image).

VII. Conclusion

We show a stable generative method for image generation conditioned on attributes. Apart from many other methods, we show experiments that use *continuous* attributes. In comparison to the vanilla GAN loss, WGAN-GP offers a more stable training environment as well as higher quality samples. We also found the latent space computed by WGAN-GP to be overall smoother, and suffer less from mode collapse. Despite the longer training time, we believe these improvements are very advantageous in other applications.

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*, pp. 2672–2680, 2014.

[2] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[3] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[4] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[6] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[8] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[13] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2536–2544, 2016.

[14] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, 2017.

[15] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," *arXiv preprint arXiv:1511.05440*, 2015.

[16] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," *arXiv preprint arXiv:1609.04802*, 2016.

[17] M. Berger, J. Li, and J. A. Levine, "A generative model for volume rendering," *arXiv preprint arXiv:1710.09545*, 2017.

[18] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint arXiv:1611.07004*, 2016.

[19] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint arXiv:1703.10593*, 2017.

[20] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[21] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," *arXiv preprint ArXiv:1611.04076*, 2016.

[22] Y. Mroueh and T. Sercu, "Fisher gan," *arXiv preprint arXiv:1705.09675*, 2017.

[23] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez, "Invertible conditional gans for image editing," *arXiv preprint arXiv:1611.06355*, 2016.

[24] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv preprint arXiv:1512.09300*, 2015.

[25] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[26] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua, "CVAE-GAN: Fine-grained image generation through asymmetric training," *arXiv preprint arXiv:1703.10155*, 2017.

[27] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," *arXiv preprint arXiv:1610.09585*, 2016.

[28] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[32] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[33] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[34] A. Baillard, E. Bertin, V. De Lapparent, P. Fouqué, S. Arnouts, Y. Mellier, R. Pelló, J.-F. Leborgne, P. Prugniel, D. Makarov, *et al.*, "The efigi catalogue of 4458 nearby galaxies with detailed morphology," *Astronomy & Astrophysics*, vol. 532, p. A74, 2011.

VIII. Appendix

Here we provide additional images as well as a more detailed explanation of the architecture designs. In all models, the first linear projection outputs a tensor of size $4 \times 4 \times 512$, which is the input to the first convolution. The concatenation in all generators is done on the first (non batch) dimension. The concatination in all discriminators is done by padding y with zeros and concatenating it depthwise.

A. MNIST Architectures

Generator

Operation	Kernel	Stride	Filters	Batch Norm	Activation
Concatenation				No	
Linear Projection				No	Linear
Convolution	5×5	2	256	Yes	ReLU
Convolution	5×5	2	128	Yes	ReLU
Convolution	5×5	2	1	Yes	ReLU

Discriminator

Operation	Kernel	Stride	Filters	Batch Norm	Activation
Concatenation				No	
Convolution	5×5	2	64	No	Leaky ReLU
Convolution	5×5	2	128	No	Leaky ReLU
Convolution	5×5	2	256	No	Leaky ReLU
Convolution	5×5	2	512	No	Leaky ReLU
Convolution	4×4	1	1	No	Linear

B. CelebA and EFIGI Architectures

Generator

Operation	Kernel	Stride	Filters	Batch Norm	Activation
Concatenation				No	
Linear Projection				No	Linear
Convolution	5×5	2	512	Yes	ReLU
Convolution	5×5	2	256	Yes	ReLU
Convolution	5×5	2	128	Yes	ReLU
Convolution	5×5	2	3	Yes	ReLU

Discriminator

Operation	Kernel	Stride	Filters	Batch Norm	Activation
Concatenation				No	
Convolution	5×5	2	64	No	Leaky ReLU
Convolution	5×5	2	128	No	Leaky ReLU
Convolution	5×5	2	256	No	Leaky ReLU
Convolution	5×5	2	512	No	Leaky ReLU
Convolution	4×4	1	1	No	Linear

C. Additional Examples

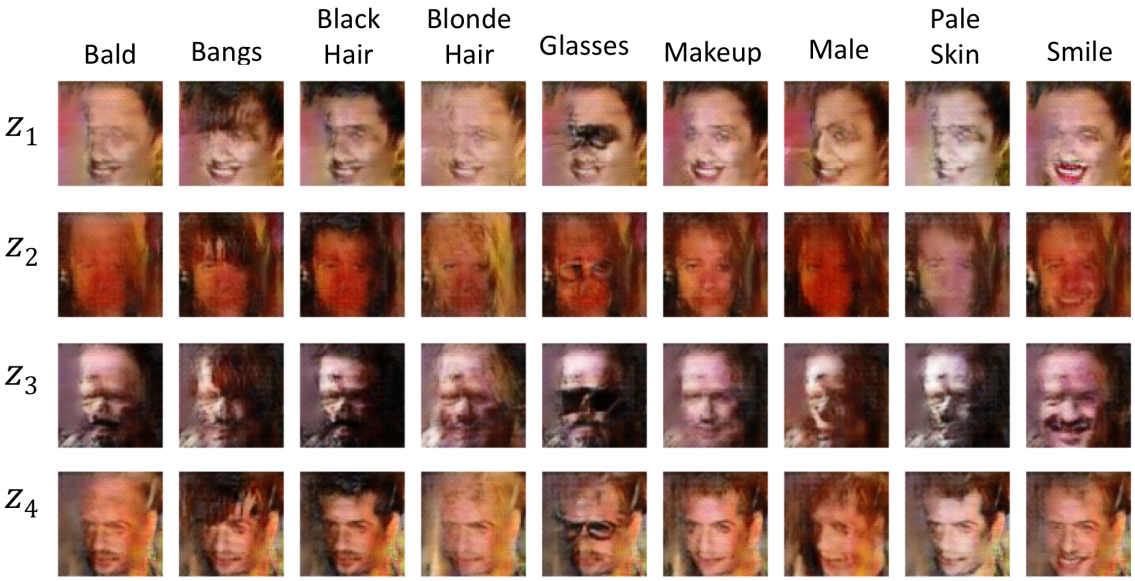

Fig. 7: Swapping CelebA attributes using the vanilla GAN loss with the *same* architecture. Attributes are present, although visual quality is much worse compared to WGAN-GP.

We can also interpolate in the same manner seen in Figure 3 between faces. Figure 8 shows interpolation using a regular GAN.

Fig. 8: Interpolation between four images using the vanilla GAN loss. Each corner is an image generated using randomly selected attributes. Interpolation along z and y generated the intermediate images.

Fig. 9: Interpolation between four images using the WGAN-GP loss. Each corner is an image generated using randomly selected attributes. Interpolation along z and y generated the intermediate images.

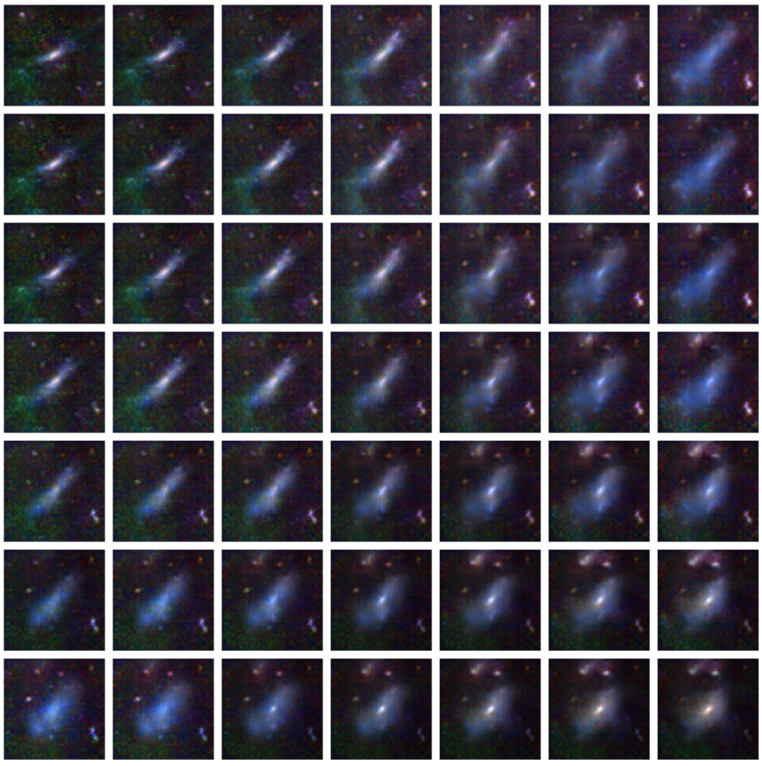

Fig. 10: Interpolation between four galaxies using the WGAN loss. Each corner is an image generated using randomly selected attributes. Interpolation along z and y generated the intermediate images.