Conditional generative adversarial nets for convolutional face generation

Jon Gauthier
Symbolic Systems Program, Natural Language Processing Group
Stanford University
jgauthie@stanford.edu

Abstract

We apply an extension of generative adversarial networks (GANs) [8] to a conditional setting. In the GAN framework, a "generator" network is tasked with fooling a "discriminator" network into believing that its own samples are real data. We add the capability for each network to condition on some arbitrary external data which describes the image being generated or discriminated.

By varying the conditional information provided to this extended GAN, we can use the resulting generative model to generate faces with specific attributes from nothing but random noise. We evaluate the likelihood of real-world faces under the generative model, and examine how to deterministically control face attributes by modifying the conditional information provided to the model.

1. Introduction

Deep learning has been proven in recent years to be an extremely useful tool for discriminative tasks. Through layers of linear transforms combined with nonlinearities, these systems learn to transform their input into an ideal representation across which we can draw clear decision boundaries. Systems with discriminative deep-learning models at their core now yield state-of-the-art results in image classification, speech recognition, and many other popular tasks of modern artificial intelligence.

It remains to be seen how this success might play out in the field of generative models. Popular generative models like the Restricted Boltzmann Machine and its many variants [4, 13, 19] have been used successfully in confined settings such as layer-wise pretraining and some applied tasks. But the overall development of generative models as standalone tools has been largely stunted, due to generally intractable problems which arise during maximum-likelihood estimation (e.g. a very expensive normalization / partition term).

Generative adversarial networks [8] are a recently introduced method for training generative models with neural networks. This approach sidesteps some of the common problems among generative models and adopts a simple SGD training regime. The generative model yielded by a learned GAN can easily serve as a density model of the training data. Sampling is simple and efficient: the network accepts some noise as input and outputs new samples of data in line with the observed training data.

A *conditional* generative adversarial network (hereafter cGAN; proposed with preliminary experiments in [17]) is a simple extension of the basic GAN model which allows the model to condition on external information. This makes it possible to engage the learned generative model in different "modes" by providing it with different contextual information.

In this paper we further develop the conditional generative adversarial network on a face image dataset. We show positive results in using the incorporated conditional data to deterministically control particular attributes of faces sampled from the model. We also demonstrate that the model benefits (in terms of test set likelihood) from this external input data. Using such conditional data as "priors" on generation, the model is evidently able to better navigate the space of possible outputs.

2. Related work

There is a long line of work in generative models for deep learning. The basis for much recent work comes from Hinton and Salakhutdinov [13], who learned compressed codes for images using a stack of Restricted Boltzmann Machines. RBMs, used as generative models in a layer-wise pretraining routine [12], led to substantial success in many deep learning tasks. They have fallen out of favor in recent years due to difficulties in training and likelihood estimation.

More recent work has focused on autoencoders and their capacity as generative models. Vincent *et al*. [21] established the denoising autoencoder (DAE) model, which learns to reconstruct empirical data X from noised inputs $\widetilde{X}$. Later theoretical results [1, 3, 5] explored procedures for sampling from learned denoising autoencoders. At a high level, the

Code available at `github.com/hans/adversarial`.

sampling process follows a Markov chain, where we alternate between sampling reconstructed values $P(X \mid \widetilde{X})$ and noise $C(\widetilde{X} \mid X)$. The Markov chain has a stationary distribution which matches the empirical density model established by the training data.

This work has been further developed under the label of *generative stochastic networks* [3, 18], where the process of noising $C(\widetilde{X} \mid X)$ is generalized to one of decoding into a hidden state using observed data. A related recent idea is the *variational autoencoder* [15], which uses neural networks to map between observed and hidden state (latent variables) during EM as a variational approximation of an expensive posterior.

As an alternative to these autoencoder models, Goodfellow *et al*. [8] proposed a different approach known as the *generative adversarial net* (GAN). The model in this paper is a straightforward extension of the generative adversarial net, and we describe this prior work in much detail below in Section 3. There is a clear contrast between autoencoders as generative models and the GAN approach in sampling new data. Whereas autoencoders require a special Markov chain sampling procedure, drawing new data from a learned GAN requires only real-valued noise input.

Mirza and Osindero [17] implemented a conditional extension to generative adversarial nets and demonstrated some preliminary experiments on MNIST, along with an application to image tagging. This paper builds alongside their work in a complementary way. We examine more formally how conditional information might be incorporated into the GAN model and look further into the process of GAN training and sampling.

3. Approach

We construct an extension of the generative adversarial net to a conditional setting. We begin by briefly summarizing the GAN concept, first introduced in [8], and proceed to formalize the conditional GAN model.

The GAN framework establishes two distinct players, a *generator* and *discriminator*, and poses the two in an adversarial game. The discriminator is tasked with distinguishing between samples from the model and samples from the training data; at the same time, the generator is tasked with maximally confusing the discriminator. We can state the objective here using a minimax value function [8]:

$$\min_G \max_D \left(\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[\log D(\mathbf{x}) \right] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \left[\log(1 - D(G(\mathbf{z}))) \right] \right). \quad (1)$$

We will define the relevant random variables and distributions in the Equation (1) shortly. For the moment we explain the two terms of the equation in prose:

1. Train the discriminator to maximize the probability of the training data.
2. Train the discriminator to minimize the probability of the data sampled from the generator. At the same time, train the generator on the opposite objective (i.e., maximize the probability that the discriminator assigns to its own samples).

The two players, when modeled as MLPs, can be trained in alternation by simple stochastic gradient descent.

The contribution of this paper is to add a *conditioning* ability to this framework. We can establish some arbitrary condition $\mathbf{y}$ for generation, which restricts both the generator in its output and the discriminator in its expected input. We might think of this condition $\mathbf{y}$ as engaging both the generator and discriminator in a particular *mode* of generation or prediction.

We now formalize the GAN concept and the conditional extension. We first define the following input and output spaces, each with an associated probability distribution:

- Z is a *noise space* used to seed the generative model. $Z = \mathbb{R}^{d_Z}$, where d_Z is a hyperparameter. Values $\mathbf{z} \in Z$ are sampled from a noise distribution $p_{\mathbf{z}}(\mathbf{z})$. In our experiments $p_{\mathbf{z}}$ is a simple Gaussian noise model.
- Y is an *embedding space* used to *condition* the generative model on some external information, drawn from the training data. $Y = \mathbb{R}^{d_Y}$, where d_Y is a hyperparameter. Using condition information provided in the training data, we can define a density model $p_{\mathbf{y}}(\mathbf{y})$.
- X is the *data space* which represents an image output from the generator or input to the discriminator. Values are normalized pixel values: $X = [0,1]^W \times [0,1]^H \times C$, where W, H represents the resolution of the input images, and C is the set of distinct color channels in the input images. Using the images in the training data and their associated conditional data, we can define a density model $p_{\text{data}}(\mathbf{x}, \mathbf{y})$ of face images. This is exactly the density model we wish to replicate with the overall model in this paper.

We now define two functions:

- $G : (Z \times Y) \to X$ is the *generative model* (or *generator*), which accepts noise data $\mathbf{z} \in Z$ along with an embedding $\mathbf{y} \in Y$ and produces an image $\mathbf{x} \in X$.
- $D : (X \times Y) \to [0,1]$ is the *discriminative model* (or *discriminator*), which accepts an image $\mathbf{x}$ and condition $\mathbf{y}$ and predicts the probability under condition $\mathbf{y}$ that $\mathbf{x}$ came from the empirical data distribution rather than from the generative model.

The generator G implicitly defines a conditional density model $p_g(\mathbf{x} \mid \mathbf{y})$. [1] We can combine this density model

[1] As we will see later, it is nontrivial to derive likelihoods for this implicit distribution. Sampling, however, is fast and direct through our parametric function G.

with our existing conditional density $p_{\mathbf{y}}(\mathbf{y})$ to yield the joint model $p_g(\mathbf{x}, \mathbf{y})$. Our precise task is to parameterize G such that it replicates the empirical density model $p_{\text{data}}(\mathbf{x}, \mathbf{y})$.

As in Equation (1), G and D play a minimax game. The value function is now derived from expectations over three previously described distributions p_{data}, $p_{\mathbf{y}}$, and $p_{\mathbf{z}}$. We combine loss on images sampled from the training data (first term) with loss on images sampled from the generator under conditions $\mathbf{y} \sim p_{\mathbf{y}}(\mathbf{y})$ (second term):

$$\min_G \max_D \left(\mathbb{E}_{\mathbf{x},\mathbf{y} \sim p_{\text{data}}(\mathbf{x},\mathbf{y})} \left[\log D(\mathbf{x}, \mathbf{y}) \right] + \mathbb{E}_{\mathbf{y} \sim p_{\mathbf{y}},\, \mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \left[\log(1 - D(G(\mathbf{z}, \mathbf{y}), \mathbf{y})) \right] \right). \quad (2)$$

In contrast with Equation (1), both terms of the above value function involve some conditional data $\mathbf{y}$ sampled from either the training data or an independent distribution. Note that the second term is an expectation over two independent random variables: the noise $\mathbf{z}$ and the conditional data $\mathbf{y}$. We detail in Section 3.1.1 how $\mathbf{y}$ is sampled in the second case.

We rephrase Equation (2) in terms of cost functions for clarity. Suppose we have a batch $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ of training images $\mathbf{x}_i$ paired with conditional data $\mathbf{y}_i$, and let $\mathbf{z}_i \sim p_{\mathbf{z}}(\mathbf{z})$ be noise data sampled from the noise distribution $p_{\mathbf{z}}$. The cost equation for the discriminator D is a simple logistic cost expression. We wish the discriminator to assign a positive label to true examples $(\mathbf{x}_i, \mathbf{y}_i)$, and a negative label to generated examples $G(\mathbf{z}_i, \mathbf{y_i})$:

$$J_D = -\frac{1}{2n} \left(\sum_{i=1}^{n} \log D(\mathbf{x}_i, \mathbf{y}_i) + \sum_{i=1}^{n} \log \left(1 - D(G(\mathbf{z}_i, \mathbf{y}_i), \mathbf{y}_i)\right) \right). \quad (3)$$

The cost function for G is similar, but only relates to the second term of Equations (2) and (3). Here we want to maximize the probability assigned by the discriminator to samples which come from G:

$$J_G = -\frac{1}{n} \sum_{i=1}^{n} \log D(G(\mathbf{z}_i, \mathbf{y}_i)). \quad (4)$$

3.1. Training

An ideal training process for the above model would be as follows:

1. The generator outputs random RGB noise by default.
2. The discriminator learns basic convolutional filters in order to distinguish between face images and random noise.
3. The generator learns the correct bias (skin tone) and basic filters to confuse the discriminator.
4. The discriminator becomes more attuned to real facial features in order to distinguish between the simple "trick" images from the generator and real face images. Furthermore, the discriminator learns to use signals in the conditional data $\mathbf{y}$ to look for particular triggers in the image.

This process continues ad infinitum, until the discriminator is maximally confused. Since the discriminator outputs the probability that an input image was sampled from the training data, we would expect a "maximally confused" discriminator to consistently output a probability of 0.5 for inputs both from the training data and from the generator.

Note that the conditional data $\mathbf{y}$ plays a key role in this learning process. We would expect the discriminator to first acquire a use for the data $\mathbf{y}$, in that some image attributes specified in $\mathbf{y}$ will help to minimize loss further than a vanilla (non-conditional) GAN might achieve. The generator would then follow up by taking advantage of the $\mathbf{y}$ data soon after the discriminator learns the proper weights for accepting $\mathbf{y}$.

If both G and D are MLPs, we can train the framework by alternating between performing gradient-based updates on G and D. Goodfellow *et al*. [8] suggest training with SGD on D for k iterations (where k is small, perhaps 1) and then training with SGD on G for one iteration. We refer the reader to [8] for full detail on the alternating SGD algorithm, which is used without modification in this paper.

Figure 6 shows an ideal set of loss curves observed while training on data for this experiment, where the discriminator loss trends toward a theoretically maximal confusion.

3.1.1 Condition sampling

We need to sample images from the generator at training time in order to evaluate the two players (see the second term of Equation (2)). This sampling requires both noise $\mathbf{z}$ and conditional data $\mathbf{y}$ as inputs. We can easily sample random noise, but need to be more careful about generating conditional data. A first solution might be to simply provide the generator with conditional data vectors found in the training data. If we drew these directly from the training examples, however, the generator might be able to reach some spurious optimum where it learns to reproduce each training image based on the conditional data input.

To avoid this unfortunate optimum, we randomize conditional data sampling during training. We build a kernel density estimate $p_y(\mathbf{y})$ (also known as a Parzen window estimate) using the conditional values $\{\mathbf{y}_i\}_{i=1}^n$ drawn from the training data. We use a Gaussian kernel, and cross-validate the kernel width σ using a held-out validation set. Samples from this nonparametric density model are used as the inputs to the generator during training.

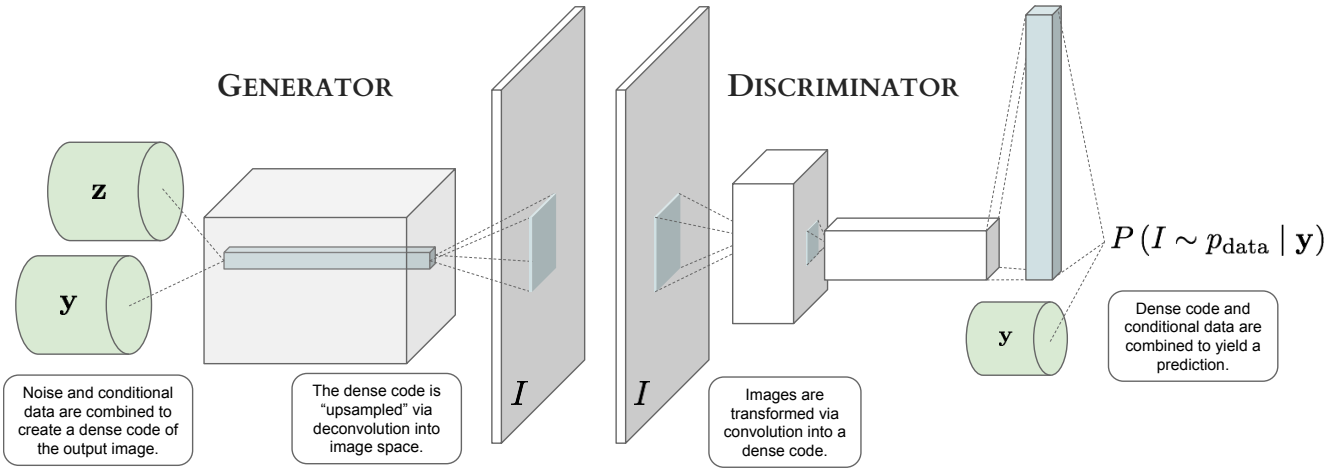


Figure 1: Broad graphical overview of the conditional generative adversarial network (cGAN) framework presented in this paper. Section 3 provides an in-depth detail of the approach, and Section 3.2 in particular details the neural network framework used in the following experiments.

3.2. Model architecture

Figure 1 gives a broad overview of the structure of the cGAN model. We expand on the contents of each model component below.

The generator G is a deconvolutional neural network [7, 22], which runs filters over its inputs and expands rather than contracts the inputs into a new representation of higher spatial dimension. We show a visualization of a deconvolution in Figure 2. In this figure our 3D input space (of dimension $2 \times 1 \times 4$) is deconvolved into an output space of dimension $8 \times 3 \times 1$. Each of the four available kernels are of the size 5×3. We deconvolve with a kernel stride of 3. Note that the spatial dimension expands from 2×1 to 8×3; meanwhile, the depth dimension contracts from 4 to 1 (perhaps a grayscale image).

Each of the 4-dimensional slices (horizontal chains of black and white blocks at left) describes a linear combination of kernels (top left) which yield an output region (right). The closer slice has a strong (white) activation for kernel #1 (blue kernel), and the output space onto which this slice maps is blue. The further slice has a strong activation for kernel #2 (red kernel), and the output space onto which this slice maps is red. Note that because the kernel stride is smaller than the kernel shape, the two output regions overlap.

This deconvolution architecture was successfully used by Goodfellow *et al*. [8] to reconstruct CIFAR-10 images. The deconvolution is exactly the inverse of the convolution operation. The deconvolutional forward pass is calculated just as is the backward pass of a convolutional layer, where a column of a 3D input describes the coefficients for a linear combination of the available filters. The linear combination of the filters forms a single patch, and patches from each such linear combination are overlaid spatially to form the resulting image.

Filter size	Number of filters	Pool shape	Output volume
—	—	—	32×32
8×8	64 ($\times 2$)	4×4	16×16
8×8	64 ($\times 2$)	4×4	7×7
5×5	192 ($\times 2$)	2×2	5×5

Table 1: Convolutional layer structure within the discriminator D. These are maxout convolutions; the number of pieces for each filter is given in parentheses in the above table. Convolution is performed solely on the input image, without using the conditional input $\mathbf{y}$. Padding not shown.

We run just a single deconvolution in this model. Future work might attempt a sequence of deconvolutions with some form of upsampling (or "unpooling" as in [7]) in between deconvolutional layers. In these experiments, the limited depth of the generator evidently helps to prevent overfitting the training data.

The discriminator D is a familiar convolutional neural network, similar to any recent model used in discriminative vision tasks such as image classification. Each convolutional layer has maxout activations [9]. Table 1 gives a full specification of the convolutional section of the discriminator. We treat the final output of the convolutions as a dense code describing the input image.

A crucial design decision is exactly where to insert the conditional information $\mathbf{y}$. As described earlier, we wish to have the input $\mathbf{y}$ engage both the discriminator and generator in different modes of operation. We consider the "foundational" information of each actor — the information it uses to make important choices about generation, or to establish a

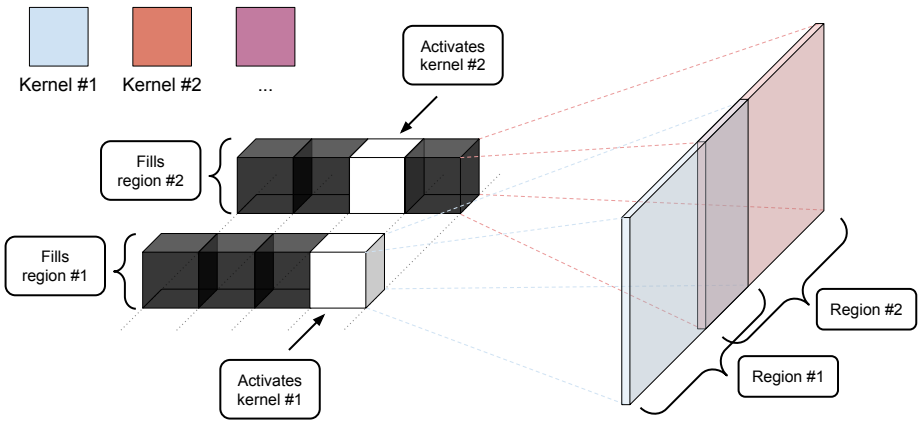


Figure 2: Visualization of a simple deconvolution operation. See Section 3.2 for an explanation of this figure.

decision boundary in discrimination — to be the dense code which appears at the start of the generator process and at the end of the discriminator process. With this in mind, we provide the conditional information $\mathbf{y}$ as input in combination with the dense code at the start of the generator feedforward (before deconvolution) and at the end of the discriminator feedforward (after convolution). See the cylinders labeled with $\mathbf{y}$ in Figure 1 for a visualization.

One could easily imagine different architectures which incorporate the conditional information $\mathbf{y}$. A simple and likely beneficial extension would be to allow for more interaction between the dense codes and the conditional information by adding more hidden layers before deconvolution (in the generator) or before decision (in the discriminator). Because of computational limitations, we restrict this work to the simplest case and leave deeper architectures for future work.

4. Experiment

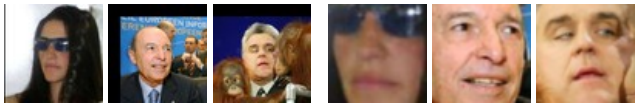
We apply the model described above to a task of *face generation*. In the following experiments we use a noise space of dimensionality $d_Z = 100$.

4.1. Data

The Labeled Faces in the Wild dataset [14] consists of about 13,000 color images of about 5,700 unique subjects in uncontrolled settings. Though originally created for the task of face identification, the data has proven useful in a large number of vision tasks.

Each image has confidence values for a large number of facial expression attributes and related features, detailed in [16], which we will exploit as conditional data $\mathbf{y}$ in these experiments. For example, these attributes include: race (Asian, Indian, black, white), age (baby, child, senior), and emotion (frowning, smiling). There are 73 different attributes in total.

The Labeled Faces in the Wild images have a diverse range of backgrounds, which can range from distracting to severely harmful for our purpose of learning how to generate faces. We avoid this noisy background data by using an automatically cropped version of the dataset from Sanderson and Lovell [20], known as LFWcrop. Figure 3 shows sample images drawn from the Labeled Faces in the Wild dataset and their equivalent cropped forms.



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

Figure 3: Random samples from the Labeled Faces in the Wild dataset [14] ((a), (b), and (c)) and their equivalents (d), (e), and (f) in the LFWcrop [20] dataset.

Concretely, we draw from Labeled Faces in the Wild training examples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. Here $\mathbf{x}_i$ is a $32 \times 32 \times 3$ RGB image, and $\mathbf{y} \in \mathbb{R}^{d_Y}$ is a real-valued vector describing facial attributes as outlined above. We experiment with different subsets of attributes, and our final settings for $\mathbf{y}$ are detailed in Section 4.3.

We randomly split the dataset into training, validation, and test splits. The validation set is used for hyperparameter selection and monitoring during training, and the test set is used for final evaluation.

4.2. Vanilla GAN

We first train a generative adversarial net as in [8], omitting the conditional information $\mathbf{y}$. The rest of the model architecture remains exactly the same. Figure 4 visualizes the evolution of the generator in this model. We sampled four arbitrary noise values $\mathbf{z}$ before training. After each training epoch we sampled images using these four noise values and saved the output images. The figure shows notable checkpoints in the learning of the process of the generator. Before

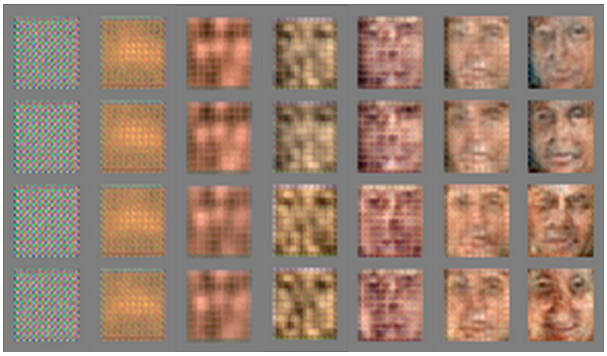


Figure 4: Output of the generator G as it trains on cropped face images. We fix four noise samples $\mathbf{z}$ (one per row) and repeatedly sample from the generator using these noise values as training progresses. The numbers below each column show the time (epoch) at which the sample was made.

any training the generator outputs RGB noise. After a small number of epochs it learns to reproduce skin color; soon after it learns the positions of basic facial features and textures.

There are several items which are worth noting to better understand how a typical image GAN learns:

There is a clear "mosaic" pattern in the early images. The generator deconvolution has a stride of three pixels, while each deconvolutional kernel is a 5×5 map. The generator clearly struggles in coordinating these kernels in regions in which they overlap. We can see this struggle in the grid-lines of discoloration in the early samples of Figure 4. Future work on deconvolution might explore new forms of regularization or parameter sharing in order to better coordinate the work done by the independent deconvolutional kernels.

This GAN consistently underfits its training data. The sampled output from the learned GAN in this experiment (and others) mostly depict white males, and fails to represent many other axes of variation in the training data. This is most likely the result of the model taking advantage of a skew in the training examples. We could probably resolve most of our underfitting problems in this case by increasing the generator capacity (i.e., number of deconvolutional kernels). Our experiments are limited here due to computational restrictions.

Recent theoretical work [10] suggests, however, that GAN models may systematically underfit their training data. The performance ceiling established by the analysis in [10] is unclear.

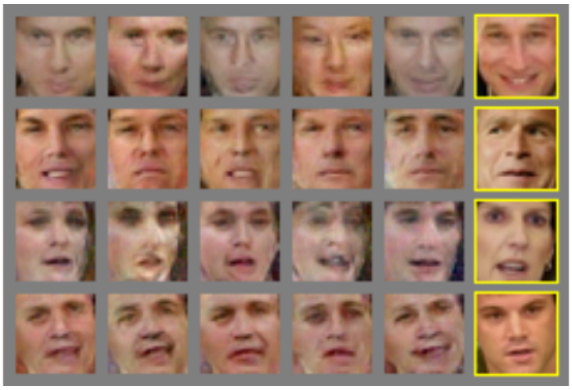


Figure 5: Samples from a learned cGAN generator. See Section 4.3 for details.

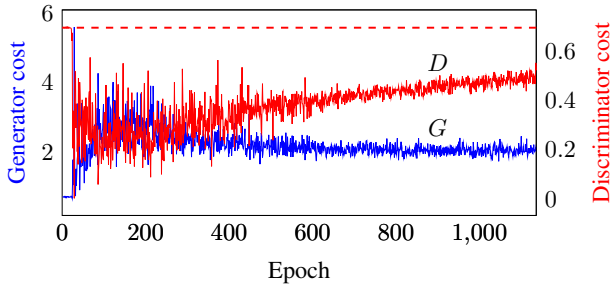


Figure 6: cGAN cost curves on a face image validation set. D (solid red) trends toward maximal confusion (dashed red asymptote) while G cost (blue) continually improves.

4.3. Conditional GAN

We next train the extended cGAN model on the face image dataset, now using both the image data $\mathbf{x}$ and the face attribute data $\mathbf{y}$. In all of the following experiments we use a particular subset of the original face attributes. We eliminate attributes which do not have clear visual effects in the cropped images in our dataset. These attributes would likely be treated as noise by the model (or would be used to overfit the training data) if we retained them. Of an available 73 attributes, we retain $d_Y = 36$ attributes.

Figure 5 shows samples from a cGAN learned on this data. In each row we begin with a conditional data value $\mathbf{y}$ sampled from the empirical distribution $p_{\mathbf{y}}$. In each column we apply random shifts to that value $\mathbf{y}$, and sample an output image. The shifts cause noticeable differences in facial features, though it is evident that the slightly shifted values of $\mathbf{y}$ still lead to similar-looking faces. The outlined face images at the far right show the nearest neighbor in the training data to the sampled image in the second-to-last column; this demonstrates that the generator has not learned to simply overfit the training data.

Figure 6 shows the typical learning process for a cGAN

on the Labeled Faces in the Wild dataset. The cost curves in the figure are calculated on a held-out validation set. The system stalls for an initial $\sim$ 25 epochs,[2] until eventually the adversarial game begins. The discriminator cost (red line) trends toward a theoretical limit of $\ln\left(\frac{1}{2}\right)$, while the generator cost (blue line) continually improves. Perhaps unsurprisingly, we find a correlation between strong positive increases in G performance and decreases in D performance (and vice versa).

4.3.1 Exploring condition space

We are interested in how the conditional data $\mathbf{y}$ might be used to deterministically specify or modify the faces generated by the model. Since in these experiments our conditional data consists of facial attributes, we should be able to tweak axes corresponding to particular attributes and see reasonable changes in the output faces.

Figure 7 shows the result of incremental shifts along particular axes in the condition space Y. We begin by sampling conditional data vectors $\mathbf{y} \sim p_{\mathbf{y}}$. The axes of this conditional data $\mathbf{y}$ are specified in the training data as confidences for particular facial attributes.[3] We concentrate on two axes in Figure 7: axis 5 (SENIOR) and axis 22 (MOUTHOPEN).

The first row of Figure 7 shows samples from the model with no modifications to the original condition data $\mathbf{y}$. In the second row, we add a constant shift to axis 5 (SENIOR) of each condition data vector $\mathbf{y}$ and sample new images. In the third row, we add a constant shift to axis 22 (MOUTHOPEN) of each vector $\mathbf{y}$ and sample once more. The noise data $\mathbf{z}$ is randomly sampled for each column, but is held fixed across rows. This means that all differences in output visible in Figure 7 are due to the shifts in $\mathbf{y}$.

These axes clearly have some deterministic effect on the output image. Many of the samples in Figure 7 become visibly older in the second row, and open their mouths or smile in the third row. Unfortunately, we did not achieve this clear control over image output in many of the other 36 axes of the condition data $\mathbf{y}$. We suspect that the model was unable to take advantage of many of these failed axes, either because they did not have as clear physical manifestations or because they were strongly correlated with the values of other axes.

4.4. Quantitative evaluation

As mentioned earlier in Section 3, while it is quick and simple to sample from a learned GAN / cGAN model, determining the likelihood of a dataset under the learned generator is nontrivial. We follow [8] and establish a Parzen window density estimate using images sampled from the generative model of a learned cGAN. The kernel function of the Parzen estimate is Gaussian, with a width σ which we select using a validation set. We then calculate the negative log-likelihood of a held-out test set under the learned nonparametric density function. Table 2 shows the resulting likelihood values for a standard GAN and the conditional GAN developed in this paper.

Model	Test set neg. log-likelihood
Vanilla GAN [8]	3024 ± 22
Conditional GAN	$\mathbf{2934 \pm 22}$

Table 2: Negative log-likelihood (lower is better) of a held-out face image test set under the generative models trained in this paper. These likelihood numbers are computed using a Parzen window method described in Section 4.4. Standard error numbers are also provided.

The improved likelihood values shown in Table 2 show that the learned generative model is better able to generate samples with the conditional density model $p_{\mathbf{y}}(\mathbf{y})$ at hand. In sampling with $p_{\mathbf{y}}$, we avoid low-density regions of the empirical model $p_{\text{data}}(\mathbf{x}, \mathbf{y})$; $p_{\mathbf{y}}$ can effectively act as a prior, keeping our generator in better regions of "face space" as it samples.

5. Conclusion

In this paper we developed an extension of the generative adversarial net framework. We added the ability to condition on arbitrary external information to both the generator and discriminator components. We evaluated the model on the Labeled Faces in the Wild dataset, and demonstrated that the conditional information $\mathbf{y}$ could be used as a deterministic control to deterministically control the output of the generator.

The development of a deterministic control slot in the GAN model opens up exciting possibilities for new models and applications. For example, a cGAN could easily accept a multimodal embedding as conditional input $\mathbf{y}$. This $\mathbf{y}$ could be produced by a neural language model, allowing us to generate images from spoken or written descriptions of their content.

Acknowledgements

We owe much credit to Goodfellow *et al*. [8] for introducing the GAN model and releasing its source code on GitHub. Many thanks are due as well to the developers of Pylearn2 [11] and Theano [2, 6], whose contributions are certain to greatly accelerate the progress of the machine learning research community.

[2] We were not able to avoid this stalling, even after extensive hyperparameter tuning. The source of this stall and methods for escaping it are interesting questions worth future research.

[3] To be clear, these are listed as explicit attributes in the training data. We did not find these axes by chance while experimenting with the model.

Figure 7: Results of constant additive shifts along particular axes of the conditional data $\mathbf{y}$. (Full details of this shift are given in Section 4.3.1.) Row 1: randomly sampled images from the generator. Row 2: constant additive shift along the SENIOR axis; faces visibly age in the resulting samples. Row 3: constant additive shift along the MOUTHOPEN axis; faces open mouths / smile.

References

[1] G. Alain and Y. Bengio. What Regularized Auto-Encoders Learn from the Data Generating Distribution. *arXiv:1211.4246 [cs, stat]*, Nov. 2012. arXiv: 1211.4246.

[2] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. 2012. Published: Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

[3] Y. Bengio, E. Laufer, G. Alain, and J. Yosinski. Deep Generative Stochastic Networks Trainable by Backprop. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 226–234, 2014.

[4] Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai. Better Mixing via Deep Representations. *arXiv:1207.4404 [cs]*, July 2012. arXiv: 1207.4404.

[5] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized Denoising Auto-Encoders as Generative Models. In *Advances in Neural Information Processing Systems*, pages 899–907, 2013.

[6] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Austin, TX, June 2010. Oral Presentation.

[7] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to Generate Chairs with Convolutional Neural Networks. *arXiv:1411.5928 [cs]*, Nov. 2014. arXiv: 1411.5928.

[8] 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 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[9] I. Goodfellow, D. Warde-farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1319–1327, 2013.

[10] I. J. Goodfellow. On distinguishability criteria for estimating generative models. *arXiv:1412.6515 [stat]*, Dec. 2014. arXiv: 1412.6515.

[11] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio. Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*, 2013.

[12] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[13] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[14] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[15] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, Dec. 2013. arXiv: 1312.6114.

[16] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *IEEE International Conference on Computer Vision (ICCV)*, Oct. 2009.

[17] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *arXiv:1411.1784 [cs, stat]*, Nov. 2014. arXiv: 1411.1784.

[18] S. Ozair, L. Yao, and Y. Bengio. Multimodal Transitions for Generative Stochastic Networks. Dec. 2013. arXiv: 1312.5578.

[19] R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[20] C. Sanderson and B. C. Lovell. Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference. In *Proceedings of the Third International Conference on Advances in Biometrics*, ICB '09, pages 199–208, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[22] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. *arXiv:1311.2901 [cs]*, Nov. 2013. arXiv: 1311.2901.