Generative Adversarial Networks

Benjamin Striner

CMU 11-785

March 21, 2018

Overview

Overview

- This week
  - What makes generative networks unique?
  - What is a generative adversarial network (GAN)?
  - What kinds of problems can we apply GANs to?
- Next week
  - How do we optimize GANs?
  - What problems do GANs have?
  - What current work is being done?

Generative Networks

Generative vs. Discriminative Networks

- Discriminative networks require inputs X and labels Y and attempt to model the conditional distribution $P(Y \mid X)$.
- Generative networks do not require labels although they may be included. They variously attempt to model $P(X)$, $P(X \mid Y)$, $P(X, Y)$, etc.

Why Generative Networks?

Why would you choose to model $P(X, Y)$ instead of $P(Y \mid X)$?

- Model can still be used to make judgments about $P(Y \mid X)$
- Model can also perform tasks like $P(X \mid Y)$, generating data based on the label
- Provides additional insights into what the model is learning
- *However*, model for $P(X, Y)$ is much harder to learn than model for $P(Y \mid X)$
  - Map from X to Y is typically many to one
  - Map from Y to X is typically one to many
  - Dimensionality of Y is typically $<<$ dimensionality of Y

Performance Differences

It seems easiest to directly solve a given problem. If your task is to determine $P(Y \mid X)$, then why would you want to model $P(X, Y)$ as an intermediate step? Ng and Jordan (2001) shows that generative models can be useful even for traditionally discriminative problems.

To compare generative and discriminative learning, it seems natural to focus on such pairs. In this paper, we consider the naive Bayes model (for both discrete and continuous inputs) and its discriminative analog, logistic regression/linear classification, and show: (a) The generative model does indeed have a higher asymptotic error (as the number of training examples becomes large) than the discriminative model, but (b) The generative model may also approach its asymptotic error much faster than the discriminative model—possibly with a number of training examples that is only *logarithmic*, rather than linear, in the number of parameters. This suggests—and our empirical results strongly support—that, as the number of training examples is increased, there can be two distinct regimes of performance, the first in which the generative model has already approached its asymptotic error and is thus doing better, and the second in which the discriminative model approaches its lower asymptotic error and does better.

VAE Recap

VAE Recap

It is important to compare and contrast GANs with VAEs, which serve a similar purpose but came earlier.

Variational Autoencoders

- Similar to autoencoders
  - Model an encoder $P(Z \mid X)$
  - Model a decoder $P(X \mid Z)$
- Model learns/infers Z; Z is not a part of the training data
- Regularized such that Z matches a prior (lets call it q)
- Since we can sample from Z directly, we can generate samples directly from the latent space

$$\log p(X \mid Z) - KL(p(Z \mid X) \mid\mid q(Z))$$

Generative Adversarial Networks

Generative Adversarial Networks

- Generative adversarial networks (GANs) are relatively new. They have spawned a flurry of activity and progress in recent years. Goodfellow et al. (2014)
- GANs are a new way to build generative models $P(X)$. GANs may have more flexibility and potential than VAEs. They produce sharper and cleaner results than VAEs. However, they are much harder to train and have their own set of issues.

Generator

The generator is equivalent to the decoder of a VAE. It tries to learn $P(X \mid Z)$. The difference is not in the structure of the decoder but how it is trained.

- Inputs are directly sampled from $Q(Z)$ but in a VAE inputs are generated using $p(z \mid x)$
- There is no encoder $p(z \mid x)$, so there is no way to pair x and z
- No true data x is provided when training the generator
- Instead of a traditional loss function, gradient is provided by a discriminator (another network)
- Discriminator weights are frozen while training generator; generator must learn to produce outputs that the discriminator likes

Discriminator

The discriminator attempts to tell the difference between real and fake images. It tries to learn $P(Y \mid X)$, where Y is the label (real or generated) and X is the real or generated data.

- Trained using standard cross entropy loss to assign the correct label (although this has changed in recent GANs)
- Generator weights are frozen while training discriminator; inputs are generated data and real data, targets are 0 and 1
- From generator's point-of-view, discriminator is a black-box loss function

Loss Functions

In traditional GANs, the loss is just cross entropy loss

- Generator wants discriminator to label it as real, so loss is $\mathbb{E}_z - \log(D(G(z)))$
- Discriminator wants to label correctly, so loss is $\mathbb{E}_z - \log(1 - D(G(z)) + \mathbb{E}_x - \log(D(x))$

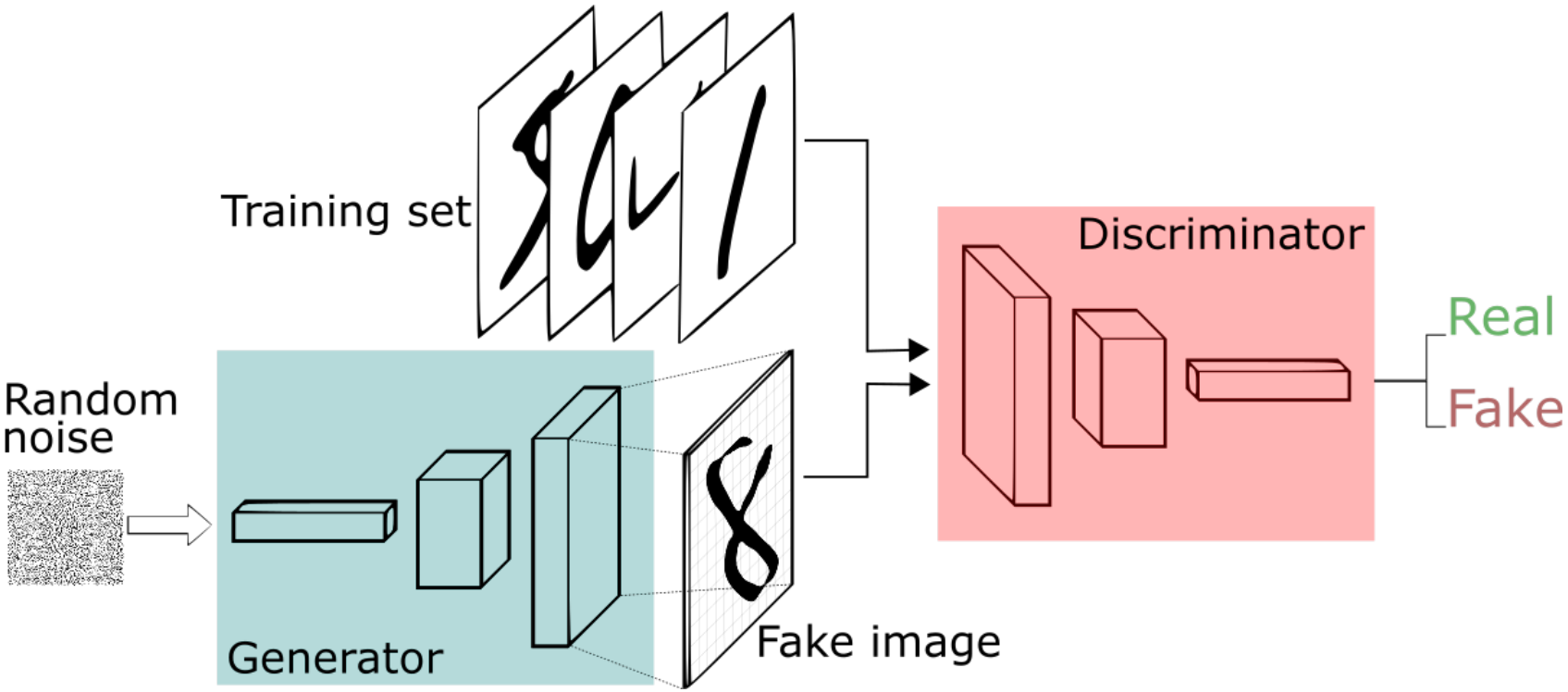
The generator wants to makes the discriminator output a 1. The discriminator wants to output a 0 for generated data but a 1 for real data.

Min-Max Game

The full two-player game can be summarily described by the below.

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

Conceptual Diagram 1 (the simple one)

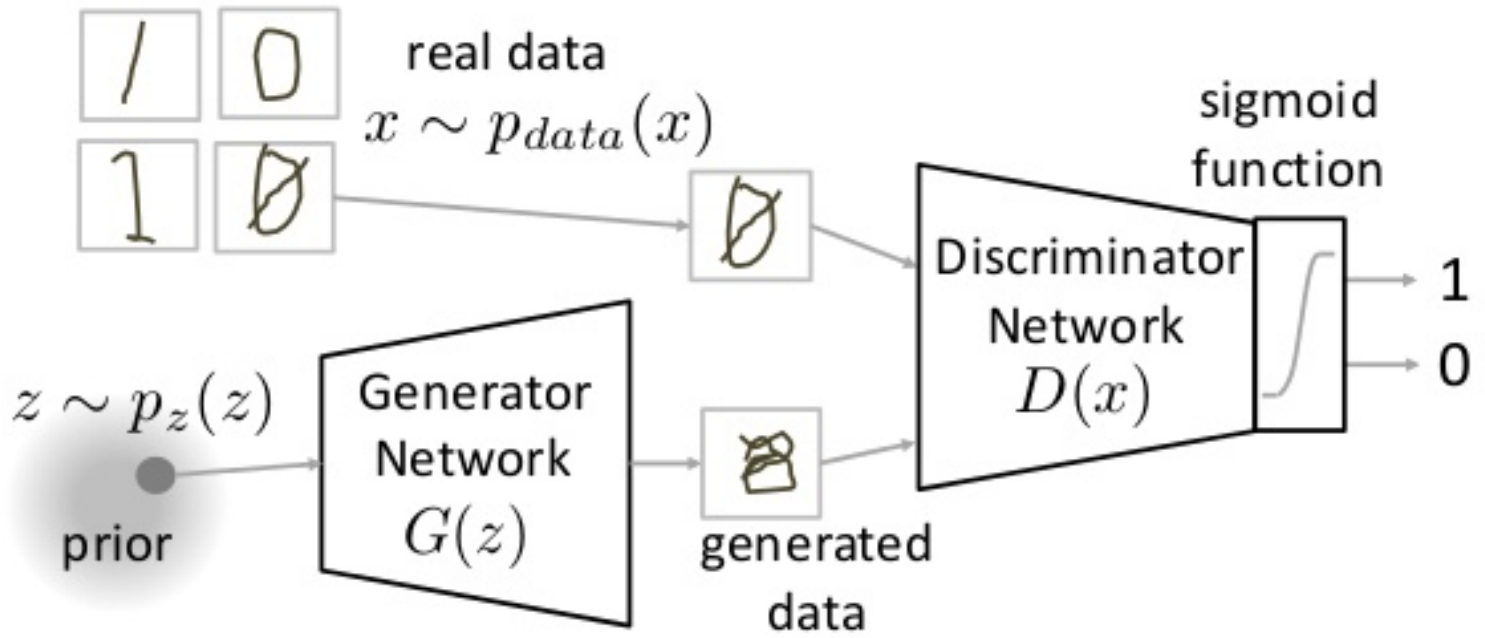


Conceptual Diagram 2 (the math one)

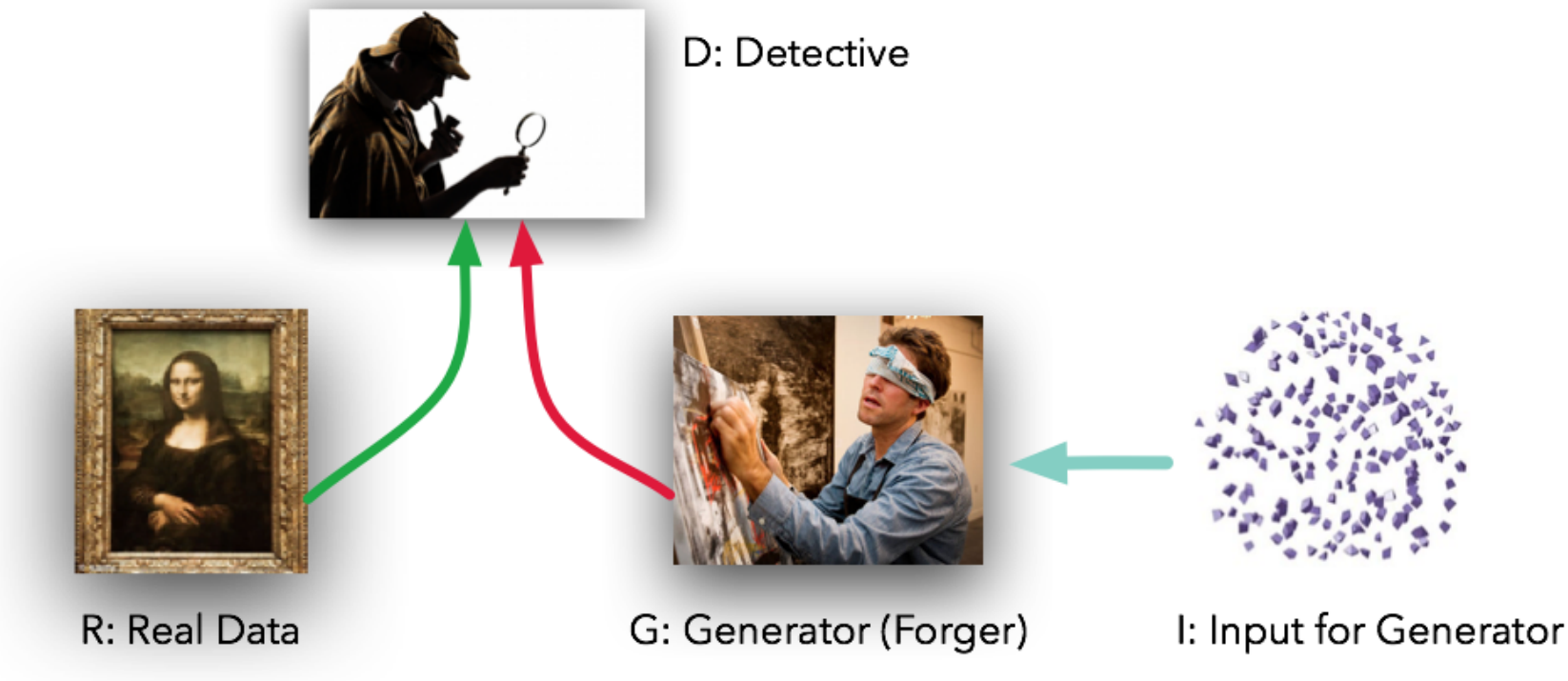
Generative Adversarial Networks

$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$



Conceptual Diagram 3 (the fun one)



See blog post (Nag)

Simultaneous Updates

- It is important to understand that both the generator and discriminator are trying to learn "moving targets". Both networks are trained simultaneously.
- The discriminator needs to update based on how well the generator is doing. The generator is constantly updating to improve performance on the discriminator. These two need to be balanced correctly to achieve stable learning instead of chaos.
- Many experiments on ways to balance the two models. More details next time!

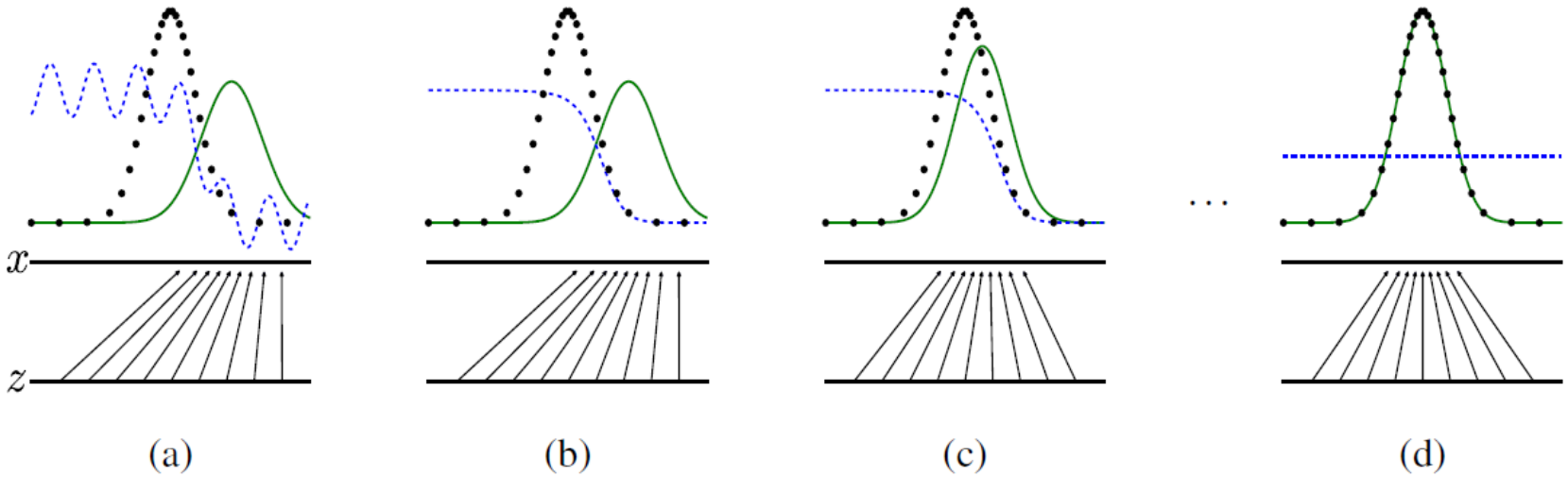
Stationary Point

There is a theoretical point in this game at which the game will be stable and both players will stop changing.

- If the generated data exactly matches the distribution of the real data, the generator should output 0.5 for all points (argmax of loss function)
- If the discriminator is outputting a constant value for all inputs, then there is no gradient that should cause the generator to update

We rarely reach a completely stable point in practice due to practical issues

What does this look like conceptually?

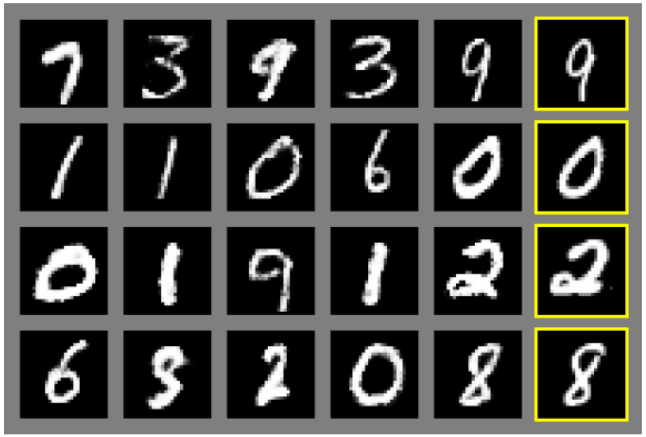


Blue is discriminator, tells the generator where to go. Green is generated data which moves based on discriminator. Dots are real data.

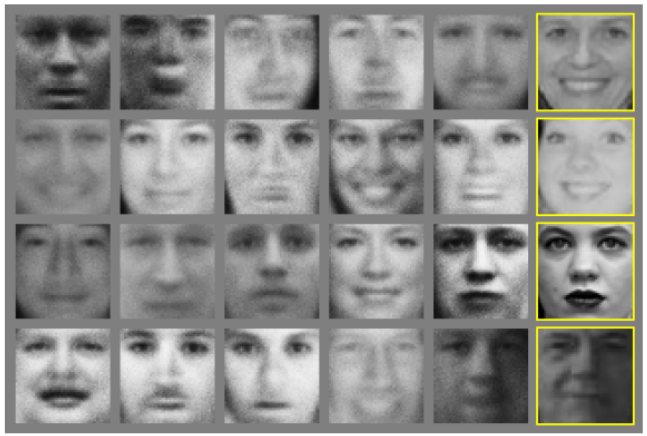
What happens in practice?

- Simplest GAN possible. Generator produces a single 2D point, discrminator is a single neuron, real data is a single point https://www.youtube.com/watch?v=ebMei6bYeWw
- 1-D GAN learning a normal distribution https://www.youtube.com/watch?v=mObnwR-u8pc
- Great video by Ian Goodfellow (40 mins but please watch if you have time) https://www.youtube.com/watch?v=HN9NRhm9waY

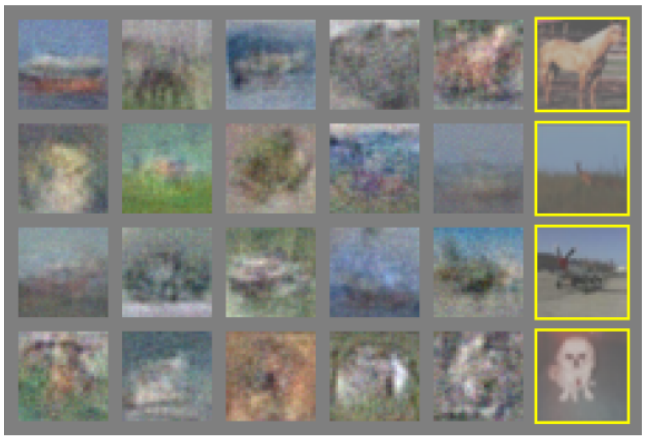
What are the outputs like? (first paper, things get better!)



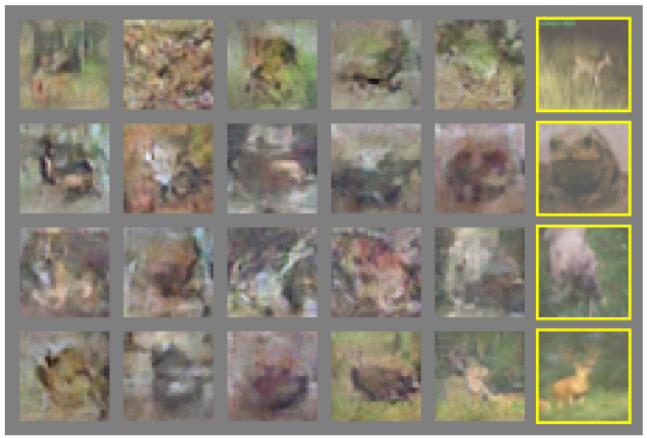
a)



b)



c)



d)

What is the latent space like?

You can interpolate along the hidden space to produce smooth transitions of images.

Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

Key differences between VAEs and GANs

- VAEs are more theoretically grounded than GANs. GANs are more based on *what works*.
  - VAEs are guaranteed to work somewhat; if you have bad hyperparameters or architecture, things will be blurrier than they should be
  - GANs are fragile; with a bad setup chaos will erupt
- GANs traditionally only learn the decoder but there are variations that learn an encoder as well; there are some problems where you want both and some problems where just the decoder will suffice. VAEs learn an encoder/decoder pair
- GAN decoder sees samples from prior $q(z)$, VAE decoder sees samples from model $p(z \mid x)$

Most important difference between VAEs and GANs

- This is the real heart of the discussion but hard to pin down.
  - VAE objective for the decoder is some man-made objective function, like L2 distance between images
  - GAN objective for the generator is some complicated objective function defined by a neural network
- This means a new way of thinking about "distance". We are training networks to minimize the "distance" or "divergence" between generated images and real images.
- Instead of some boring distance metric like L1 or L2, we can make something completely new

Implications of GANs

- Our metric by which we are comparing images is now defined by a neural network. That means any biases implicit in the network are now a part of our distance/loss.
- For example, imagine you have a discriminator that uses max-pooling and is therefore somewhat shift invariant. Your discriminator cannot discriminate based on small shifts, so your loss function is invariant to shifts. That should result in the network generating samples that are slight shifts of real data. Your CNN is really good at looking at local features but not as good at shifts in data.

Loss compared to VAEs

- In a VAE trained with something like L2 distance, the distance between two shifted images can be immense. The VAE will learn to produce roughly average images, which minimize the L2 distance but look blurry.
- The L2 distance is not a measure of how perceptually similar two things are.
- A neural network, with the right architecture, is arguably the definition of perceptual similarity (assuming our visual system is some sort of neural network).

Conceptual Example

Imagine you want to generate completely random white noise.

- This is the hardest task for a VAE to complete. That is because accurately reconstructing the image requires encoding every single pixel of the image. If the VAE is unable to encode everything, it will predict gray, which minimizes the L2 loss.
- This is relatively easy for a generator to produce. It is very hard for the discriminator to tell the difference between one white noise and another. The discriminator can tell the difference between a solid gray mass and white noise easily. The generator will tend to produce white noise.

What can we do with GANs?

What can we do with GANs?

We are going to talk about some of the architectures people have built around GANs.

- Conditional GAN: Mirza and Osindero (2014)
- LapGAN: Denton et al. (2015)
- DCGAN: Radford et al. (2015)
- CatGAN: Springenberg (2015)
- GRAN: Im et al. (2016)
- InfoGAN: Chen et al. (2016)
- AAE: Makhzani et al. (2015)
- BiGAN: Donahue et al. (2016)

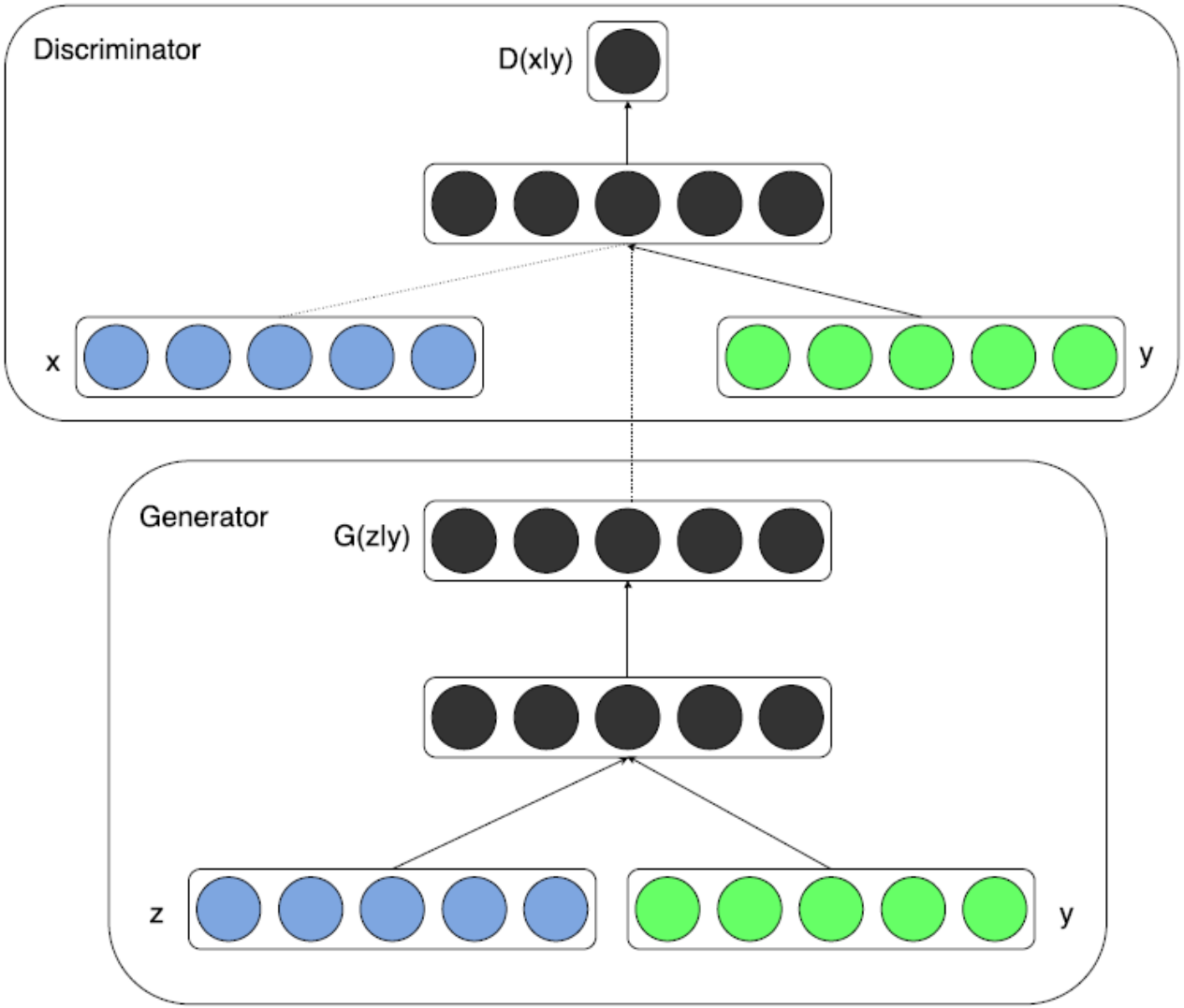
Conditional GANs

Conditional GANs include a label and learn $P(X \mid Y)$ Mirza and Osindero (2014).

- Generator learns $P(X \mid Z, Y)$
- Discriminator learns $P(L \mid X, Y)$

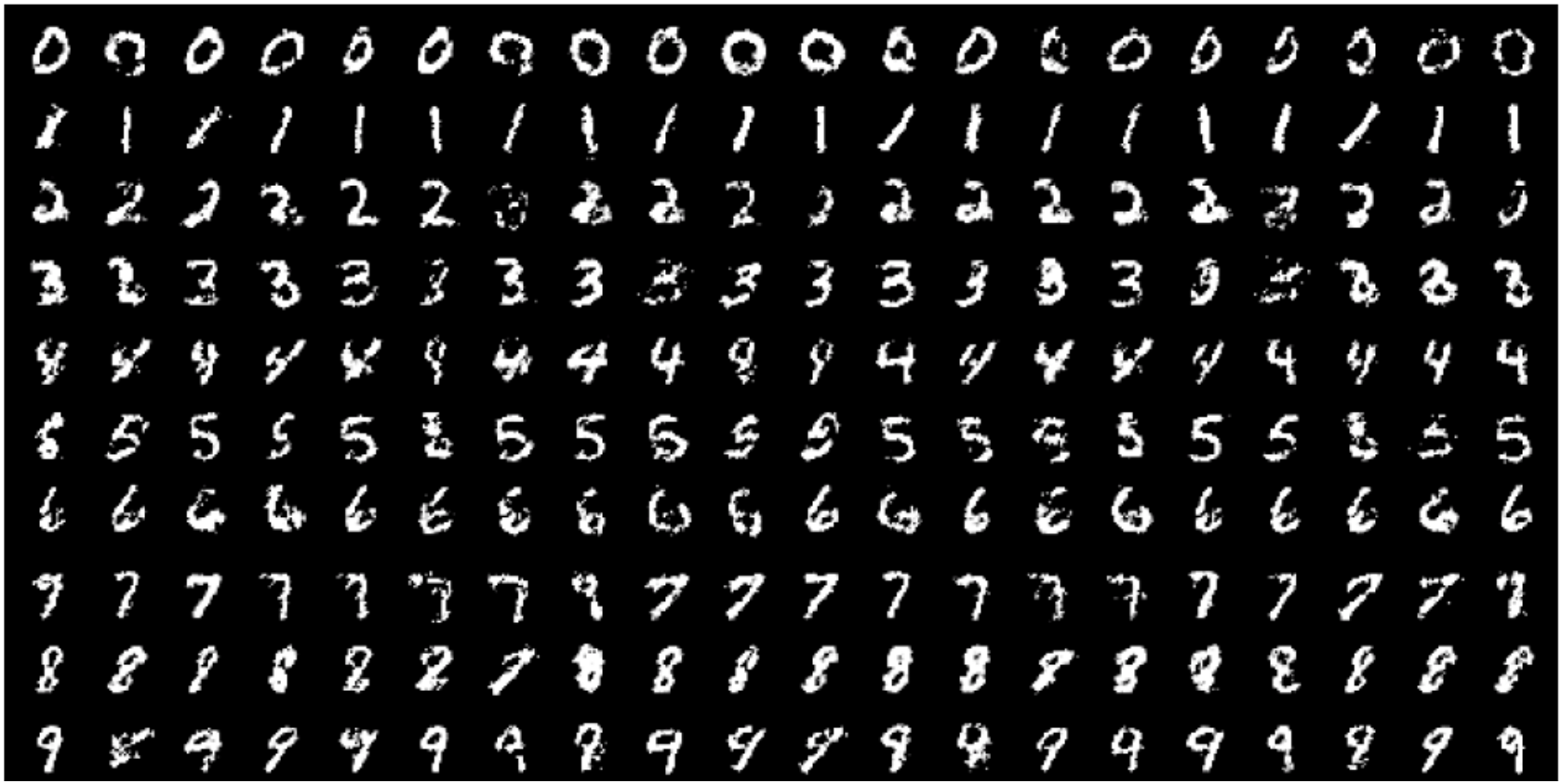
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

CGAN Conceptual Diagram



CGAN Results

Each row is conditioned on a different label. You can use a single neural network to generate all 10 digits by telling it what digit to generate.

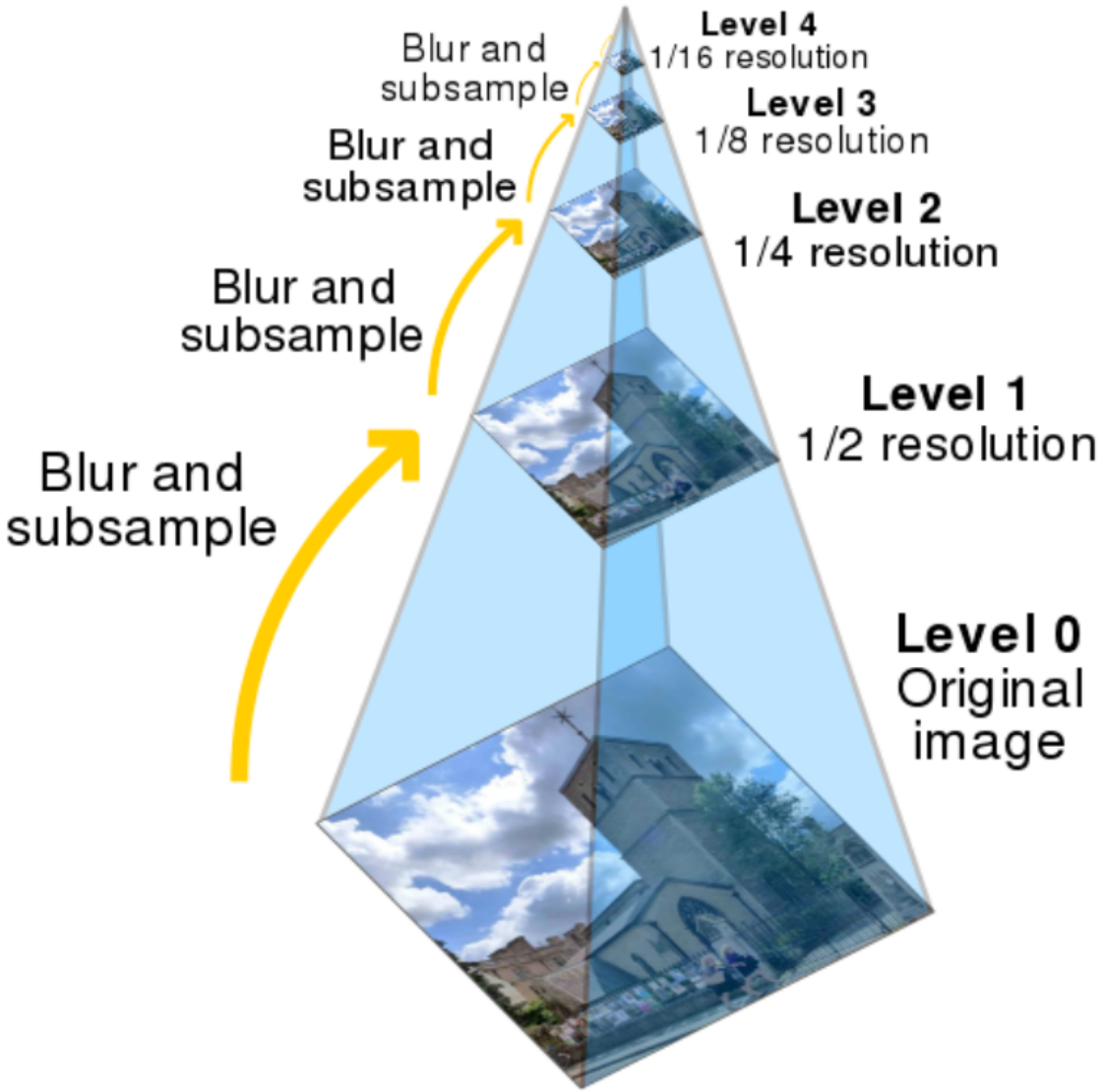


Laplacian-pyramid GANs (LapGANs)

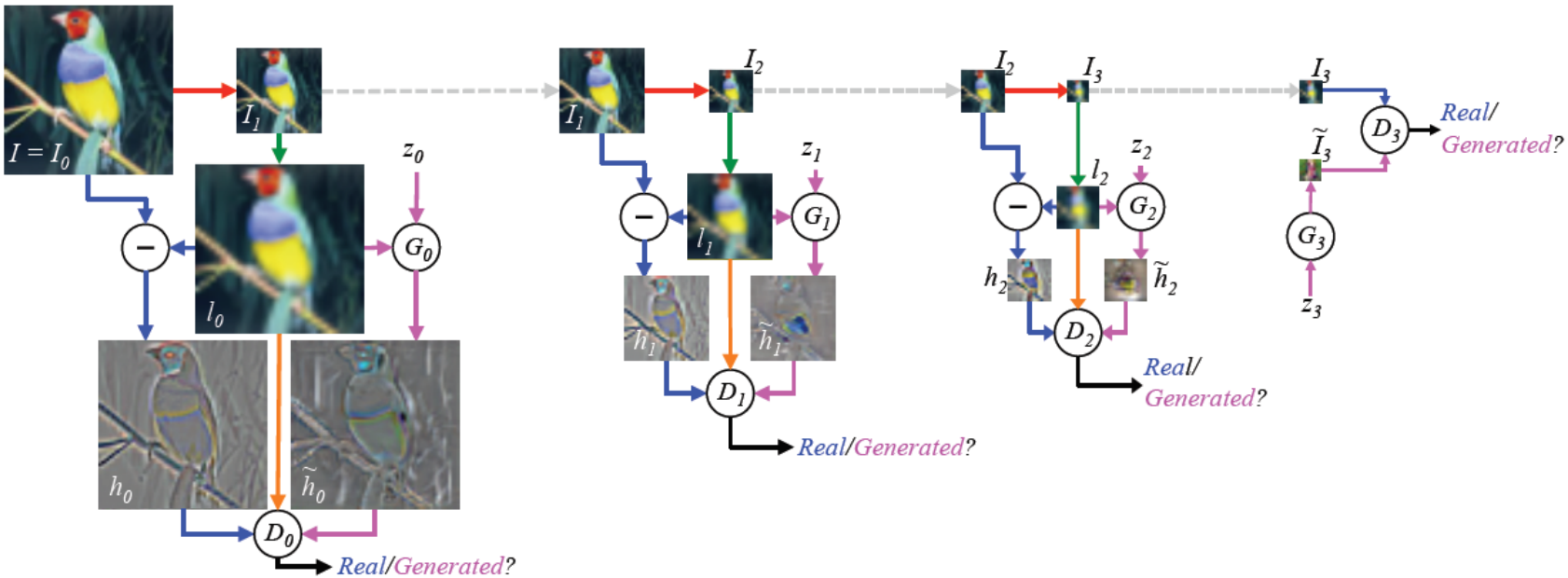
LapGAN utilizes conditional GANs and a trick from image processing known as a "pyramid" Denton et al. (2015).

- Given a blurry image, a single CGAN can generate a slightly less blurry image
- A stack of CGANs can generate a very sharp image

What is a Laplacian Pyramid?



LAPGAN Conceptual Diagram



LAPGAN Generation Chain

(a) (b)

Figure 4: STL samples: **(a)** Random 96x96 samples from our LAPGAN model. **(b)** Coarse-to-fine generation chain.

DCGAN

DCGAN made several improvements allowing GANs to be trained on larger/deeper CNNs Radford et al. (2015). Came after LapGAN and produced even cleaner results. Main techniques:

- Using Leaky ReLUs
- Using Batchnorm
- Using strided convolutions (all-cnn)

DCGAN Architecture

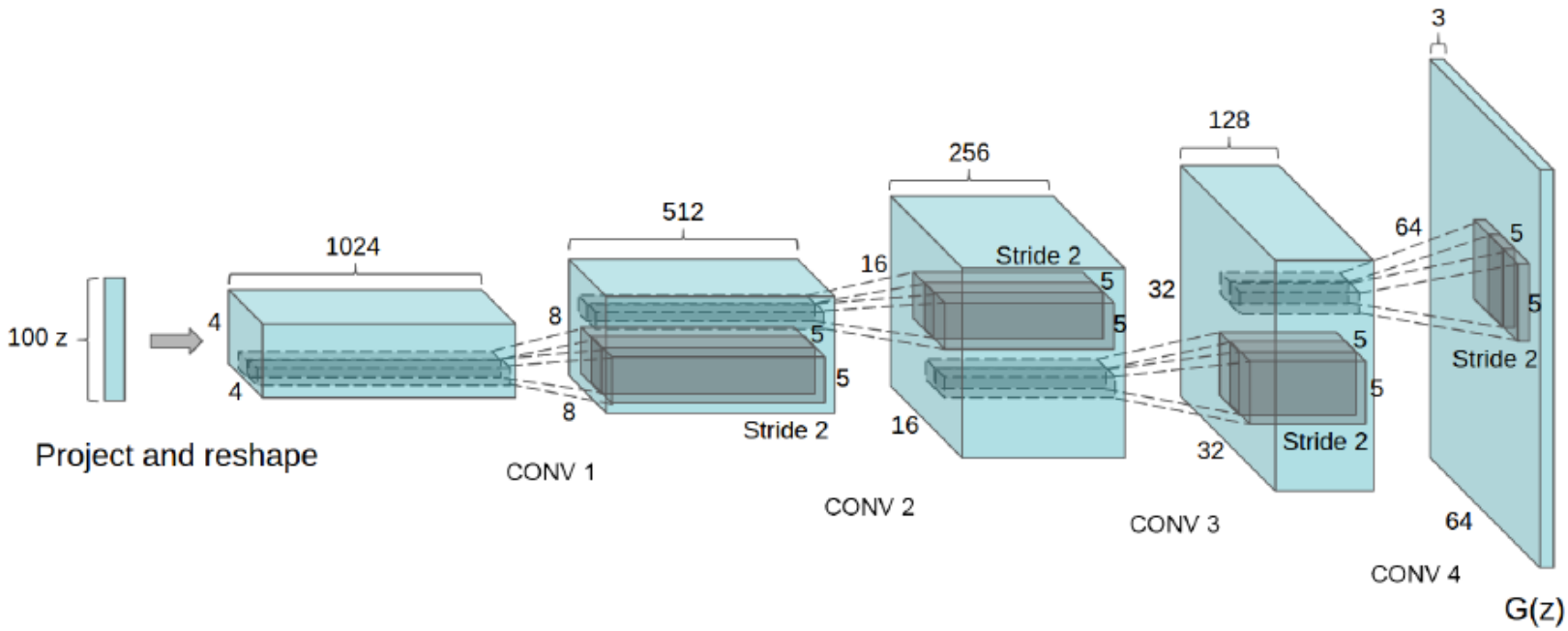
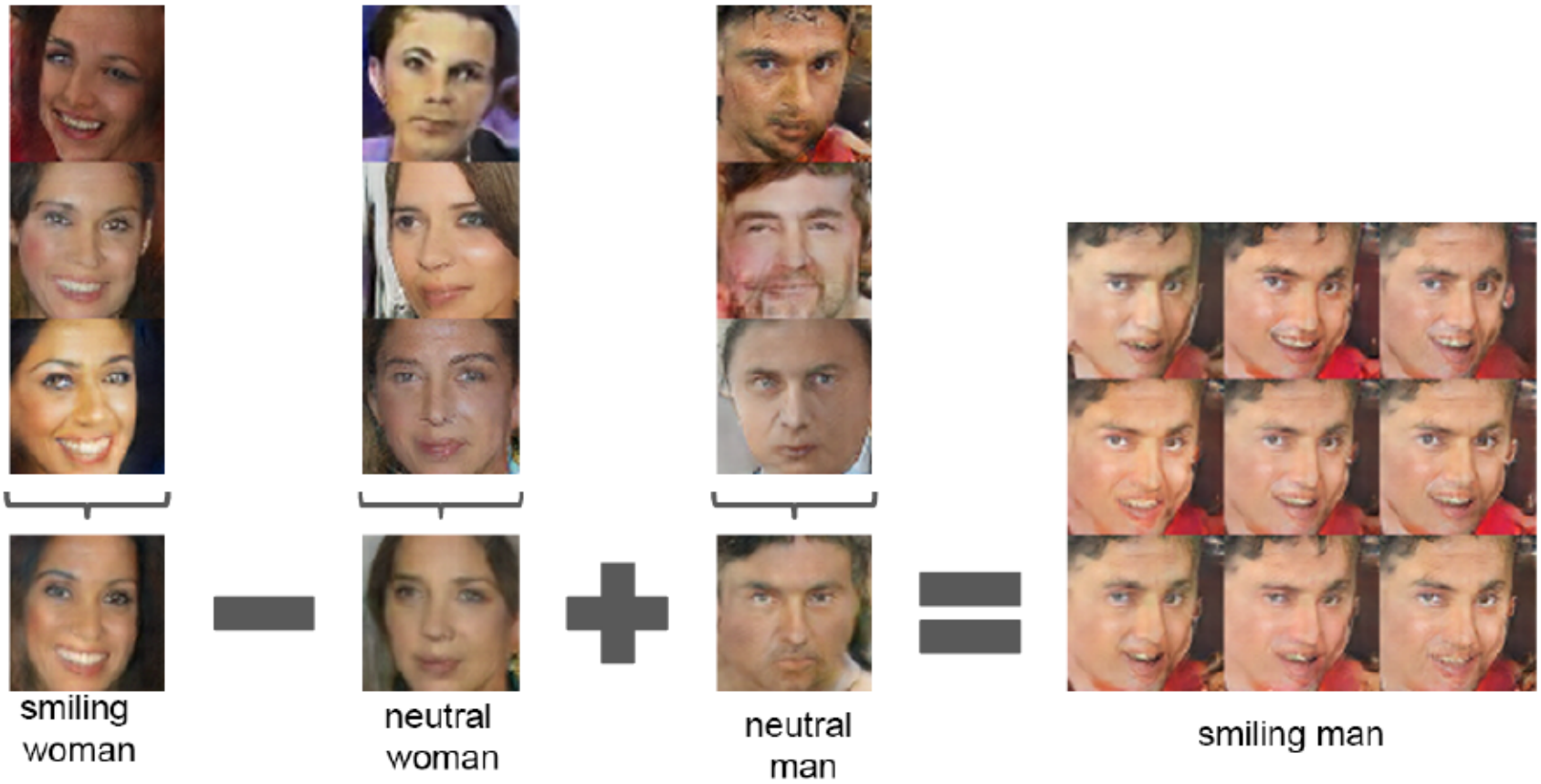


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

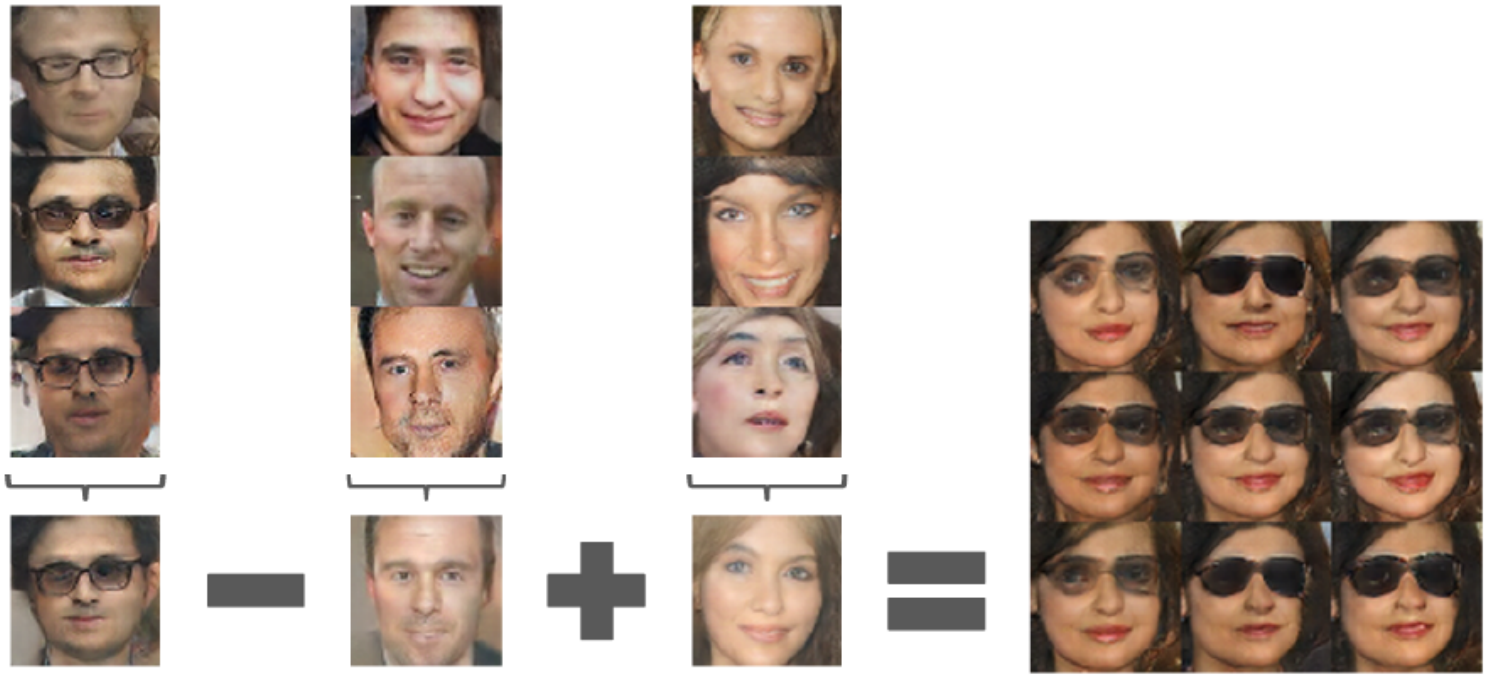
DCGAN LSUN Results

Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

DCGAN Vector Math 1



DCGAN Vector Math 2



CatGAN

CatGAN provides a method for unsupervised or semi-supervised learning of labels. Instead of the typical GAN objective, an entropy-based objective is used. Springenberg (2015)

- Generator tries to fool discriminator into assigning a label (low label entropy)
- Discriminator tries to leave generated samples as uncategorized (high entropy) while real samples are categorized (low entropy)
- Optional supervision on labels can be provided

CatGAN Architecture

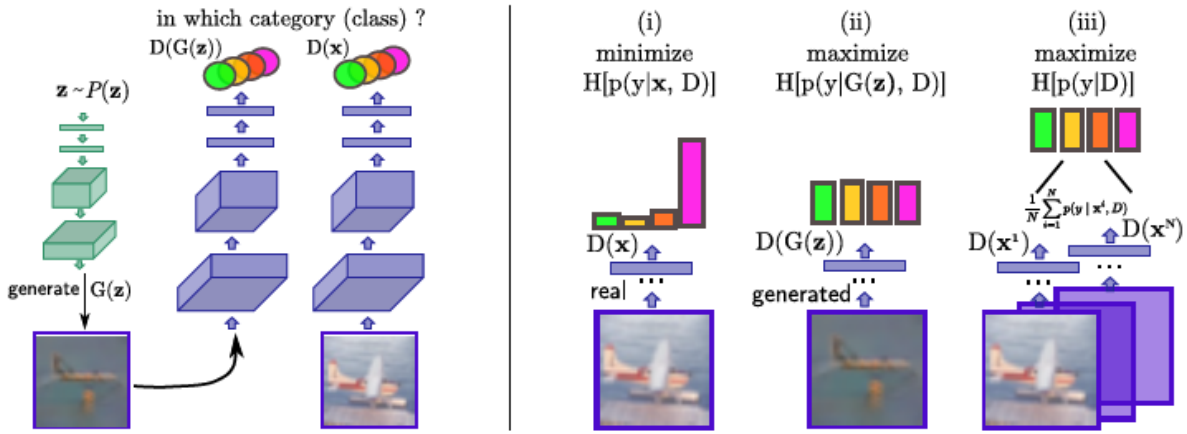


Figure 1: Visualization of the information flow through the generator (in green) and discriminator (in violet) neural networks (left). A sketch of the three parts (i) - (iii) of the objective function $\mathcal{L}_D$ for the discriminator (right). To obtain certain predictions the discriminator minimizes the entropy of $p(y|\mathbf{x}, D)$, leading to a peaked conditional class distribution. To obtain uncertain predictions for generated samples the the entropy of $p(y|G(\mathbf{z}), D)$ is maximized which, in the limit, would result in a uniform distribution. Finally, maximizing the marginal class entropy over all data-points leads to uniform usage of all classes.

CatGAN Objective

$$H_{\mathcal{X}}\Big[p(y \mid D)\Big] = H\Big[\frac{1}{N}\sum_{i=1}^{N} p(y \mid \mathbf{x}^i, D)\Big],$$
$$H_{G}\Big[p(y \mid D)\Big] \approx H\Big[\frac{1}{M}\sum_{i=1}^{M} p(y \mid G(\mathbf{z}^i), D)\Big], \text{ with } \mathbf{z}^i \sim P(\mathbf{z}). \tag{6}$$

Combining the definition from Equations (4,5,6) we can define the CatGAN objective for the discriminator, which we refer to with $\mathcal{L}_D$, and for the generator, which we refer to with $\mathcal{L}_G$ as

$$\mathcal{L}_D = \max_D H_{\mathcal{X}}\Big[p(y \mid D)\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\Big[H\big[p(y \mid \mathbf{x}, D)\big]\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y \mid G(\mathbf{z}), D)\big]\Big],$$
$$\mathcal{L}_G = \min_G -H_G\Big[p(y \mid D)\Big] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\Big[H\big[p(y \mid G(\mathbf{z}), D)\big]\Big], \tag{7}$$

CatGAN Clustering Results

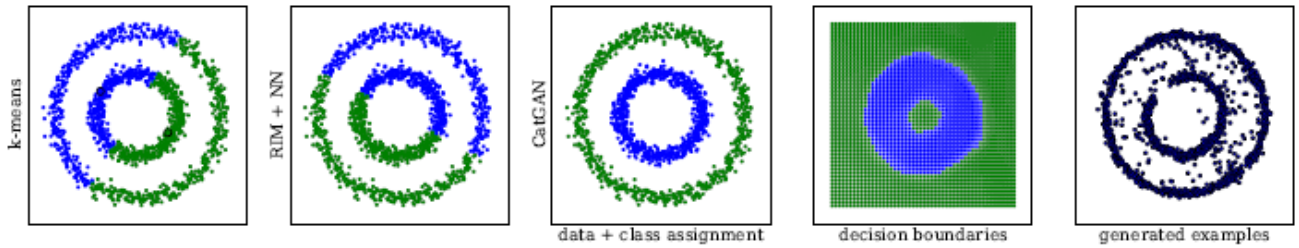
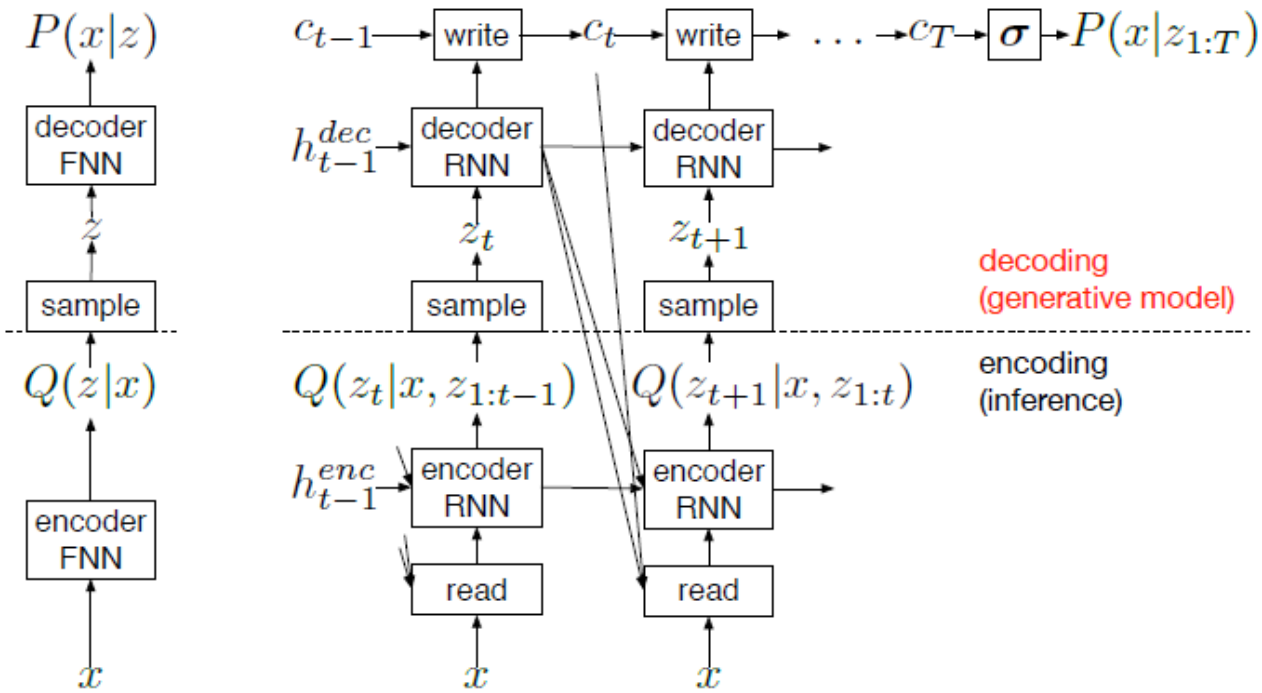


Figure 2: Comparison between k-means (left), RIM (middle) and CatGAN (rightmost three) – with neural networks – on the "circles" dataset with $K = 2$. Blue and green denote class assignments to the two different classes. For CatGAN we visualize class assignments – both on the dataset and on a larger region of the input domain – and generated samples. Best viewed in color.

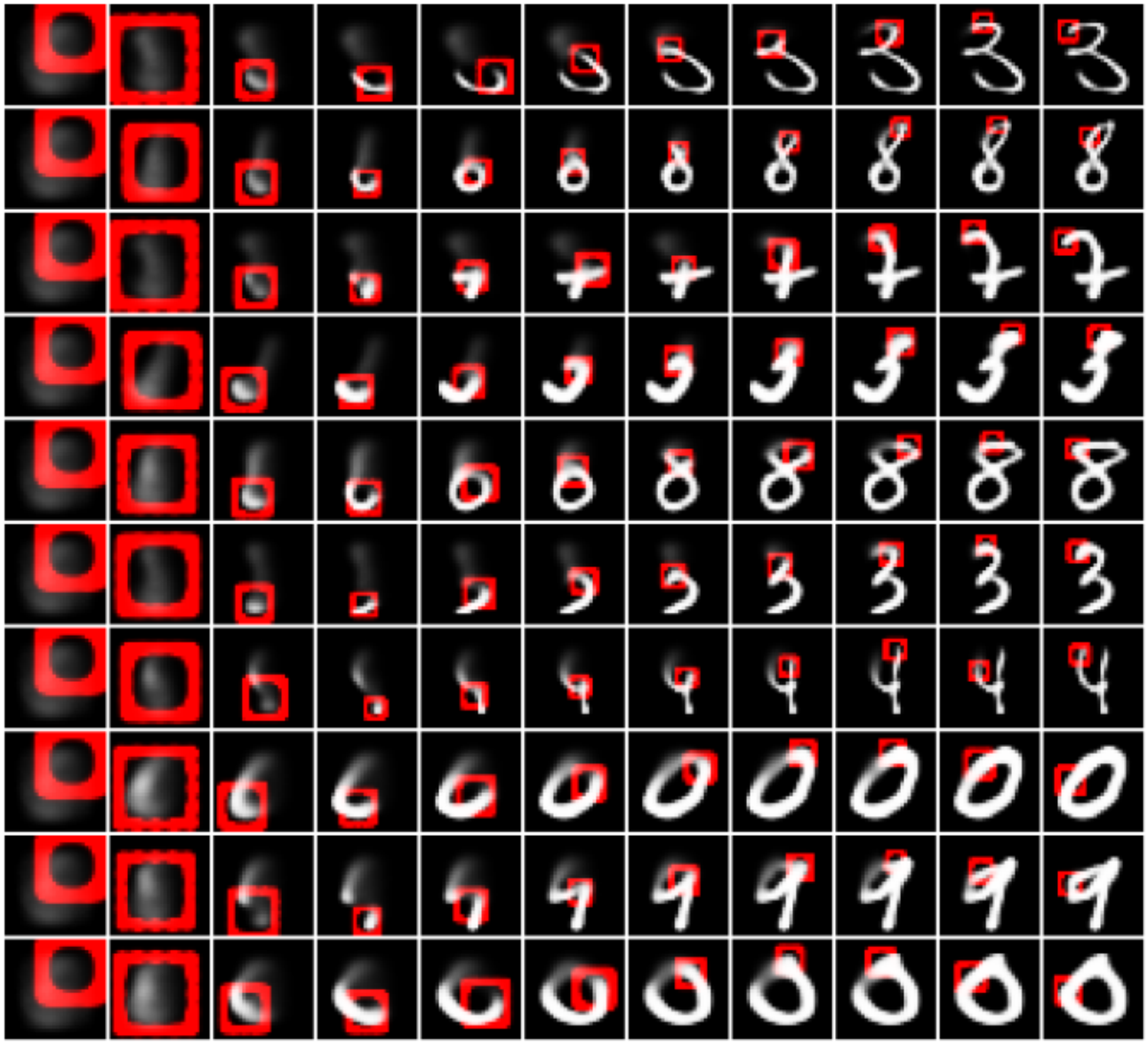
DRAW

DRAW is based on VAEs (not GANs), but it is relevant to the next model (GRAN). DRAW attempts to recurrently draw an image with attention, instead of emitting a single image all-at-once Gregor et al. (2015).

DRAW Conceptual Diagram



DRAW MNIST Attention



DRAW SVHN Attention

Generative Recurrent Adversarial Networks (GRANs)

GRAN can be described most simply as the GAN equivalent of DRAW. It recurrently draws an image, using a GAN. Im et al. (2016)

GRAN Overall Architecture

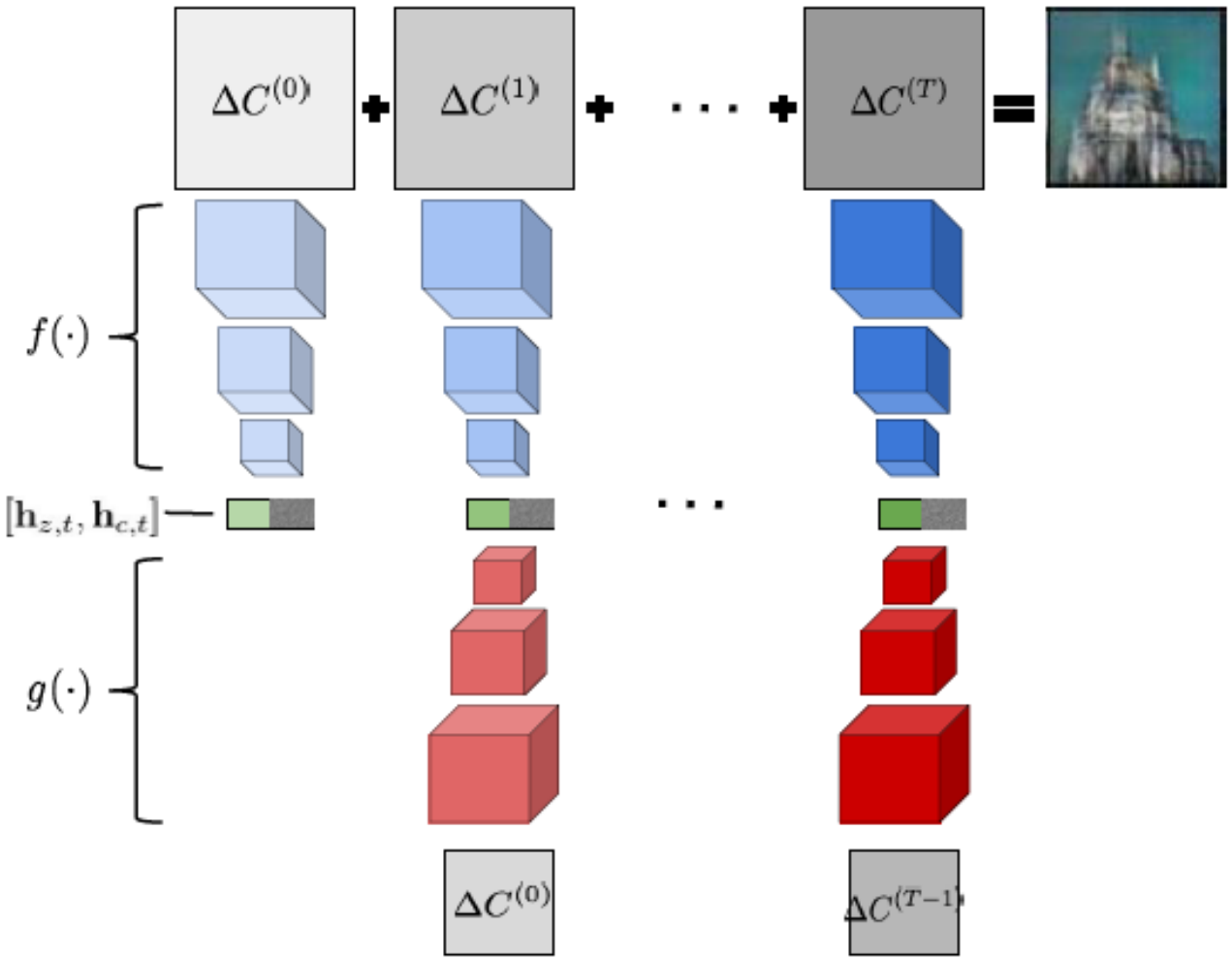


Figure 3. Abstraction of Generative Recurrent Adversarial Networks. The function f serves as the decoder and the function g serves as the encoder of GRAN.

GRAN Slice Architecture

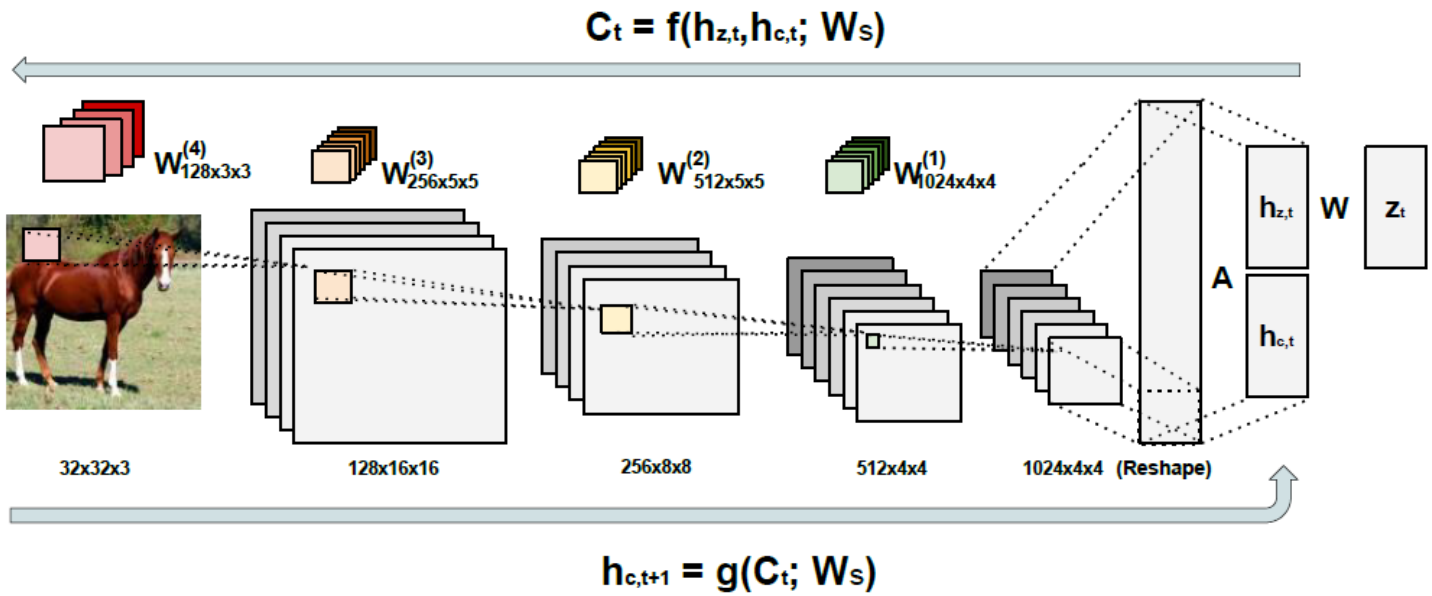


Figure 2. Depiction of single time step component of Generative Recurrent Adversarial Networks architecture layed out. (The numbers of the figures are used for modelling CIFAR10 dataset)

GRAN Results

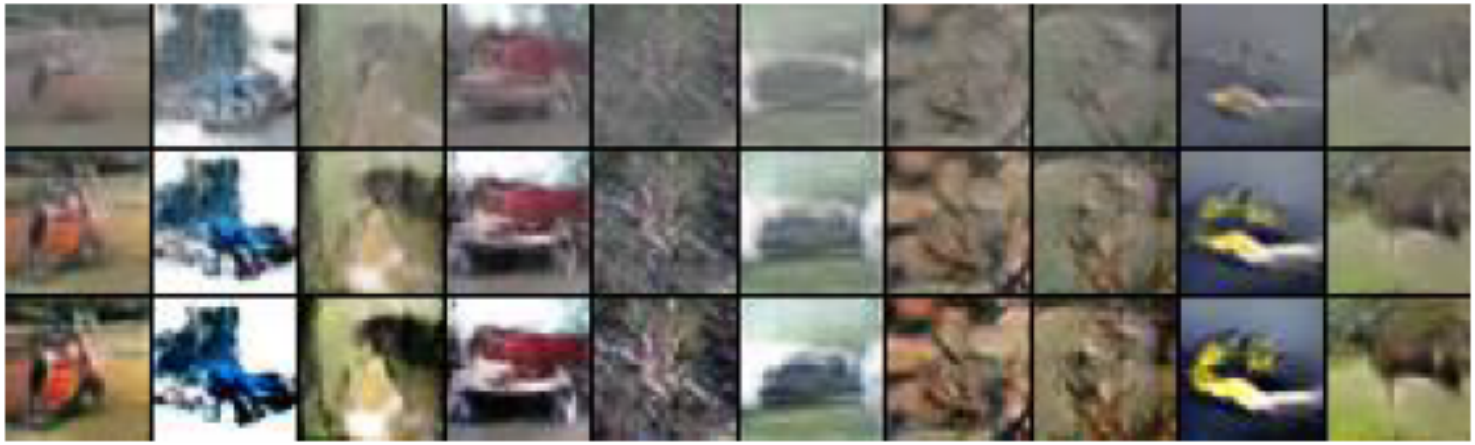


Figure 9. Drawing at different time steps on cifar10 samples.

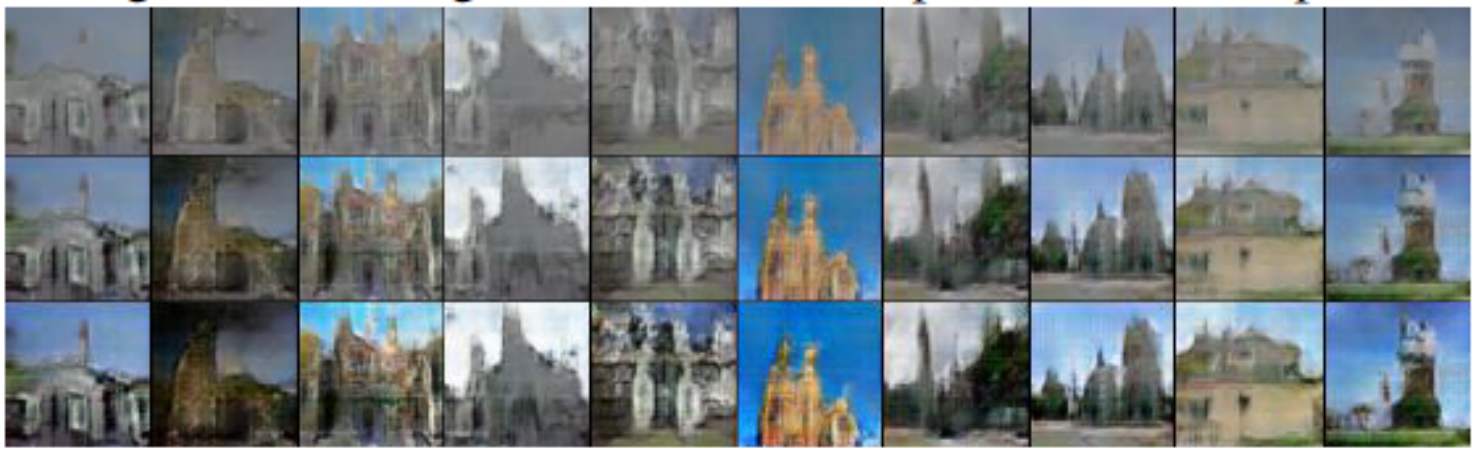


Figure 10. Drawing at different time steps on lsun samples.

InfoGAN

InfoGAN disentangles the hidden representations so that latent dimensions are meaningful. It does this by maximizing the mutual information between the encoding and the output. Roughly speaking, regularize a GAN by adding an encoder function $Q(c \mid x)$. Chen et al. (2016)

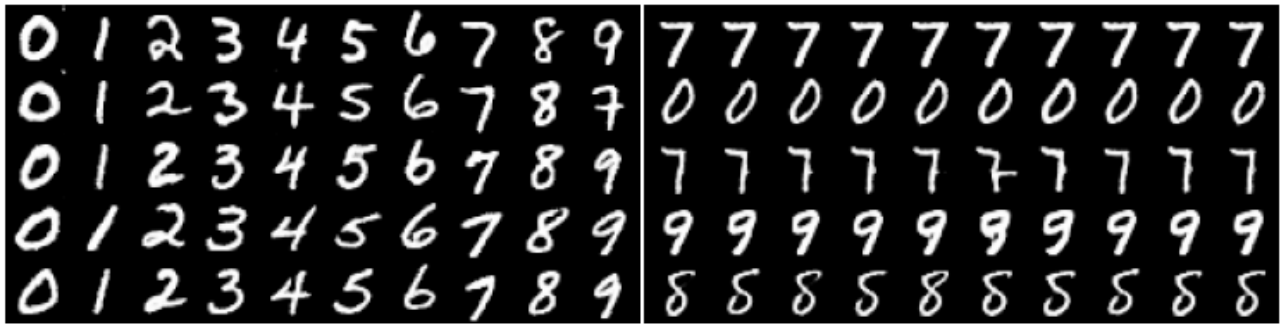
InfoGAN Objective

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Hence, InfoGAN is defined as the following minimax game with a variational regularization of mutual information and a hyperparameter λ:

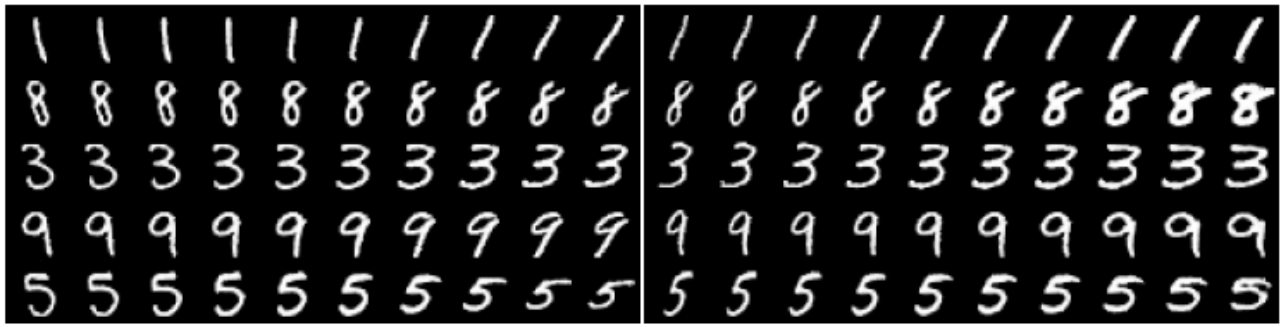
$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D,G,Q) = V(D,G) - \lambda L_I(G,Q) \tag{6}$$

InfoGAN MMIST Results



(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

Figure 2: **Manipulating latent codes on MNIST:** *In all figures of latent code manipulation, we will use the convention that in each one latent code varies from left to right while the other latent codes and noise are fixed. The different rows correspond to different random samples of fixed latent codes and noise. For instance, in (a), one column contains five samples from the same category in c_1, and a row shows the generated images for 10 possible categories in c_1 with other noise fixed.* In (a), each category in c_1 largely corresponds to one digit type; in (b), varying c_1 on a GAN trained without information regularization results in non-interpretable variations; in (c), a small value of c_2 denotes left leaning digit whereas a high value corresponds to right leaning digit; in (d), c_3 smoothly controls the width. We reorder (a) for visualization purpose, as the categorical code is inherently unordered.

InfoGAN Face Results

(a) Azimuth (pose)

(b) Elevation

(c) Lighting

(d) Wide or Narrow

Figure 3: **Manipulating latent codes on 3D Faces:** We show the effect of the learned continuous latent factors on the outputs as their values vary from -1 to 1. In (a), we show that one of the continuous latent codes consistently captures the azimuth of the face across different shapes; in (b), the continuous code captures elevation; in (c), the continuous code captures the orientation of lighting; and finally in (d), the continuous code learns to interpolate between wide and narrow faces while preserving other visual features. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

InfoGAN Chair Results

(a) Rotation (b) Width

Figure 4: **Manipulating latent codes on 3D Chairs:** In (a), we show that the continuous code captures the pose of the chair while preserving its shape, although the learned pose mapping varies across different types; in (b), we show that the continuous code can alternatively learn to capture the widths of different chair types, and smoothly interpolate between them. For each factor, we present the representation that most resembles prior supervised results [7] out of 5 random runs to provide direct comparison.

Adversarial Autoencoders

Adversarial autoencoders are somewhat like a cross between GANs and VAEs. Based on an autoencoder. Makhzani et al. (2015)

- VAE regularizes using KL divergence between Z produced by model and Z from the prior
- AAE regularizes using discriminator loss between Z produced by model and Z from the prior
- Trains both encoder and decoder, unlike GANs; first GAN model to train both encoder and decoder
- Encoding learned by model more accurately matches the prior than with VAEs
- Do not need to analytically calculate KL, so prior can be any arbitrary distribution as long as you can sample from it

Basic AAE Architecture

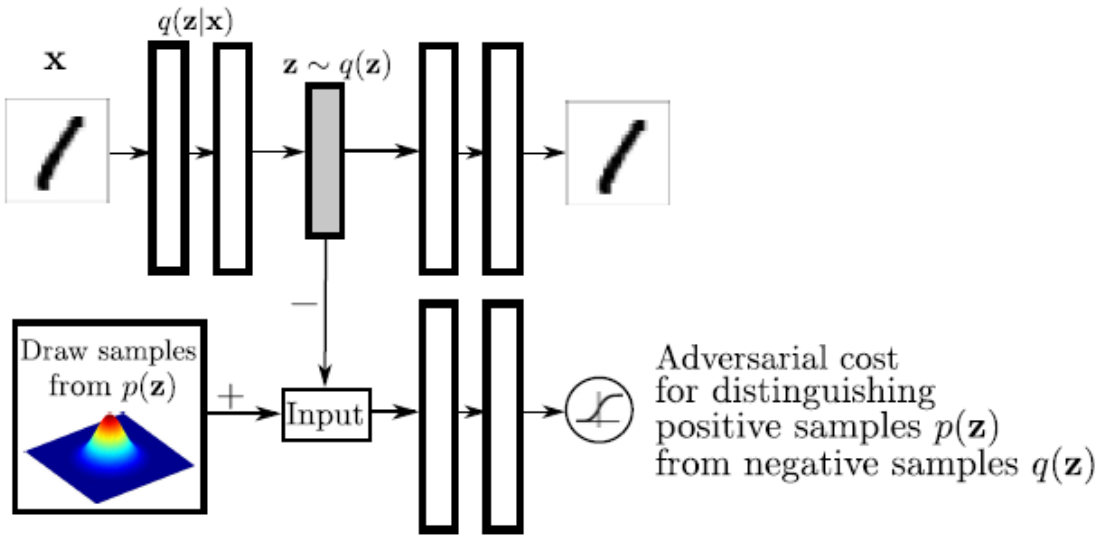


Figure 1: Architecture of an adversarial autoencoder. The top row is a standard autoencoder that reconstructs an image $\mathbf{x}$ from a latent code $\mathbf{z}$. The bottom row diagrams a second network trained to discriminatively predict whether a sample arises from the hidden code of the autoencoder or from a sampled distribution specified by the user.

Comparison with VAE

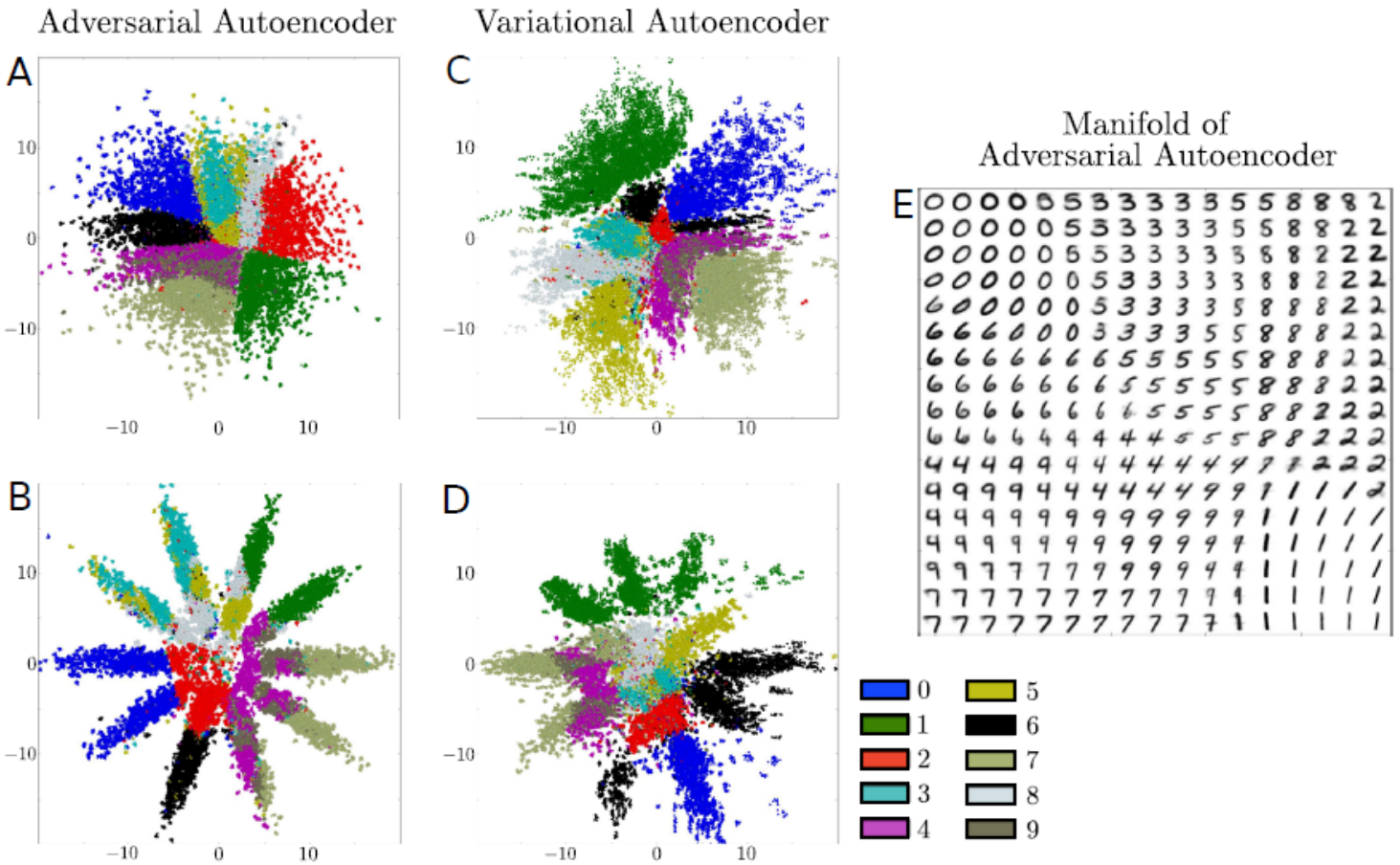


Figure 2: Comparison of adversarial and variational autoencoder on MNIST. The hidden code $\mathbf{z}$ of the *hold-out* images for an adversarial autoencoder fit to (a) a 2-D Gaussian and (b) a mixture of 10 2-D Gaussians. Each color represents the associated label. Same for variational autoencoder with (c) a 2-D gaussian and (d) a mixture of 10 2-D Gaussians. (e) Images generated by uniformly sampling the Gaussian percentiles along each hidden code dimension $\mathbf{z}$ in the 2-D Gaussian adversarial autoencoder.

Arbitrary Priors and can leverage labels

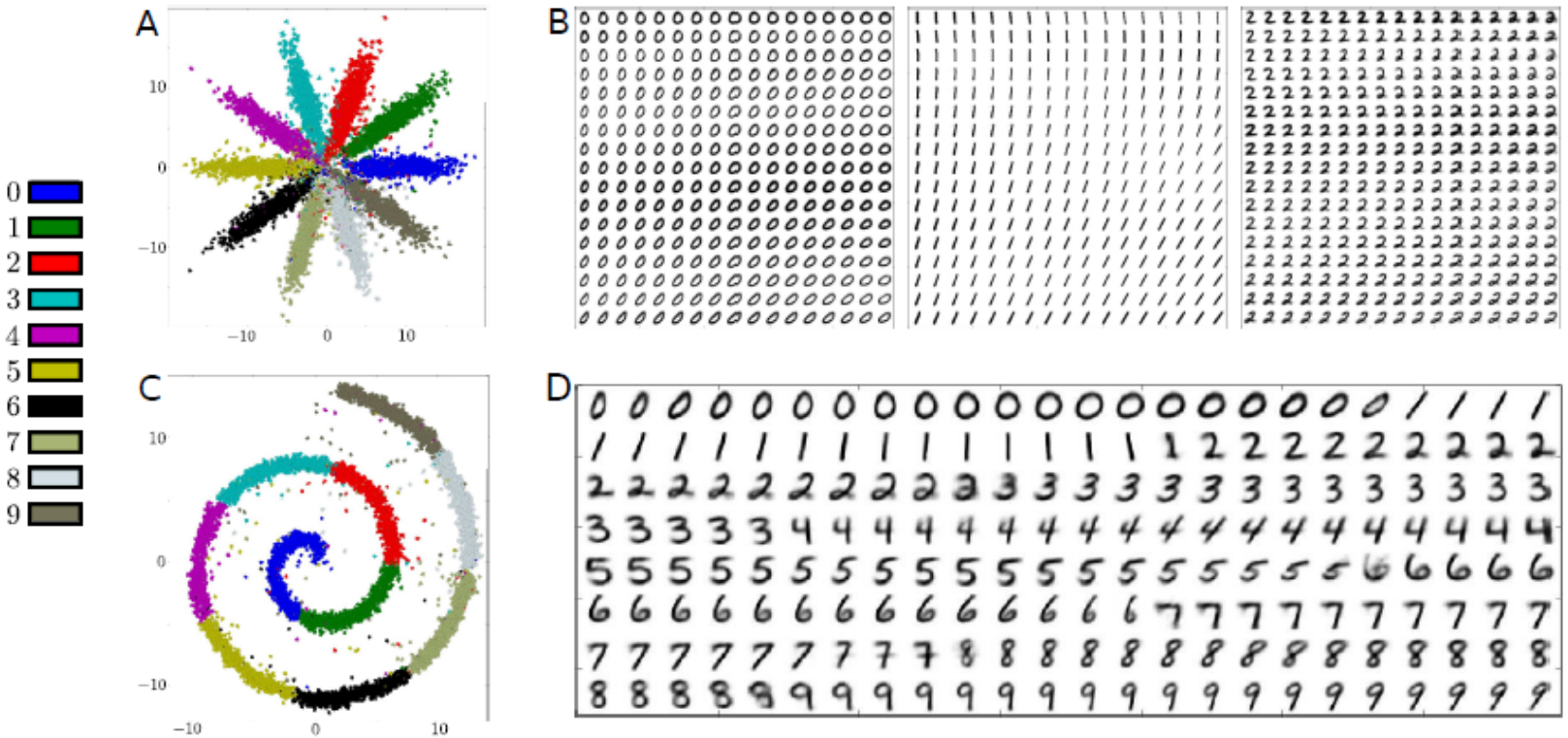


Figure 4: Leveraging label information to better regularize the hidden code. **Top Row:** Training the coding space to match a mixture of 10 2-D Gaussians: (a) Coding space $\mathbf{z}$ of the *hold-out* images. (b) The manifold of the first 3 mixture components: each panel includes images generated by uniformly sampling the Gaussian percentiles along the axes of the corresponding mixture component. **Bottom Row:** Same but for a swiss roll distribution (see text). Note that labels are mapped in a numeric order (i.e., the first 10% of swiss roll is assigned to digit 0 and so on): (c) Coding space $\mathbf{z}$ of the *hold-out* images. (d) Samples generated by walking along the main swiss roll axis.

Semi-Supervised AAE

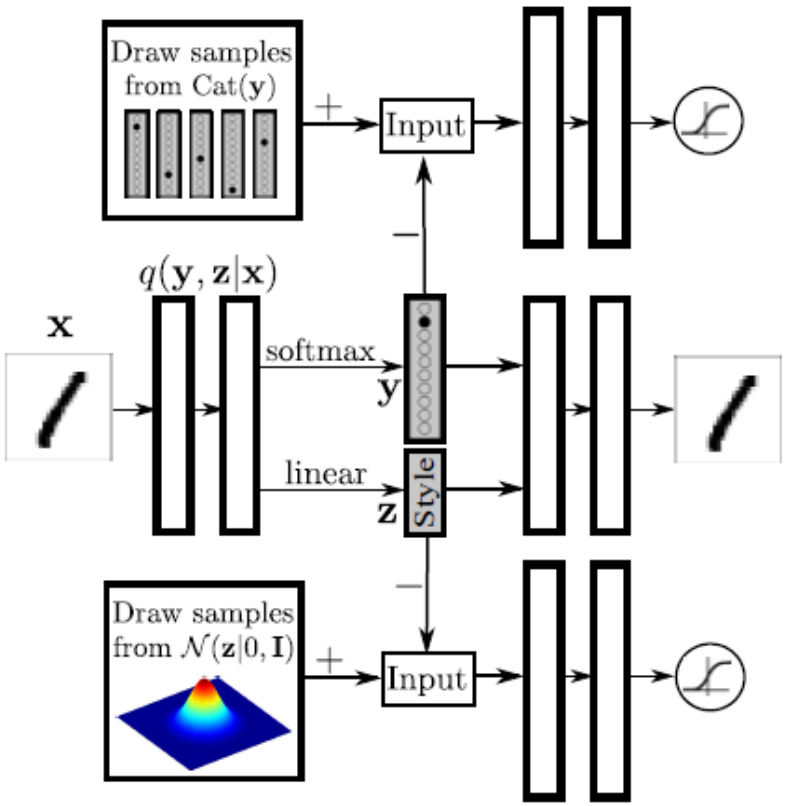


Figure 8: Semi-Supervised AAE: the top adversarial network imposes a Categorical distribution on the label representation and the bottom adversarial network imposes a Gaussian distribution on the style representation. $q(\mathbf{y}|\mathbf{x})$ is trained on the labeled data in the semi-supervised settings.

Disentangling Content and Style Architecture

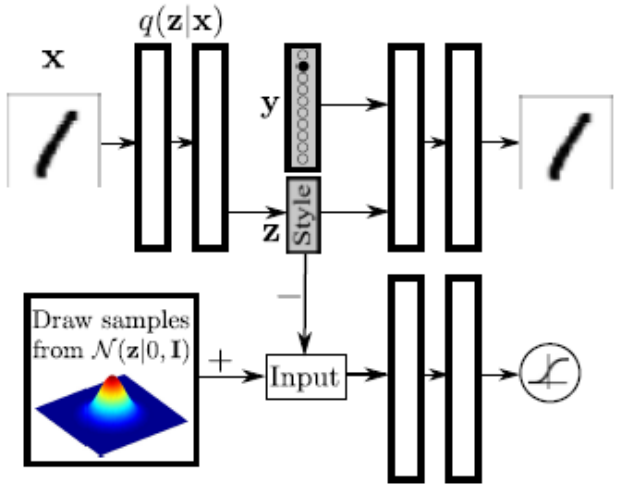
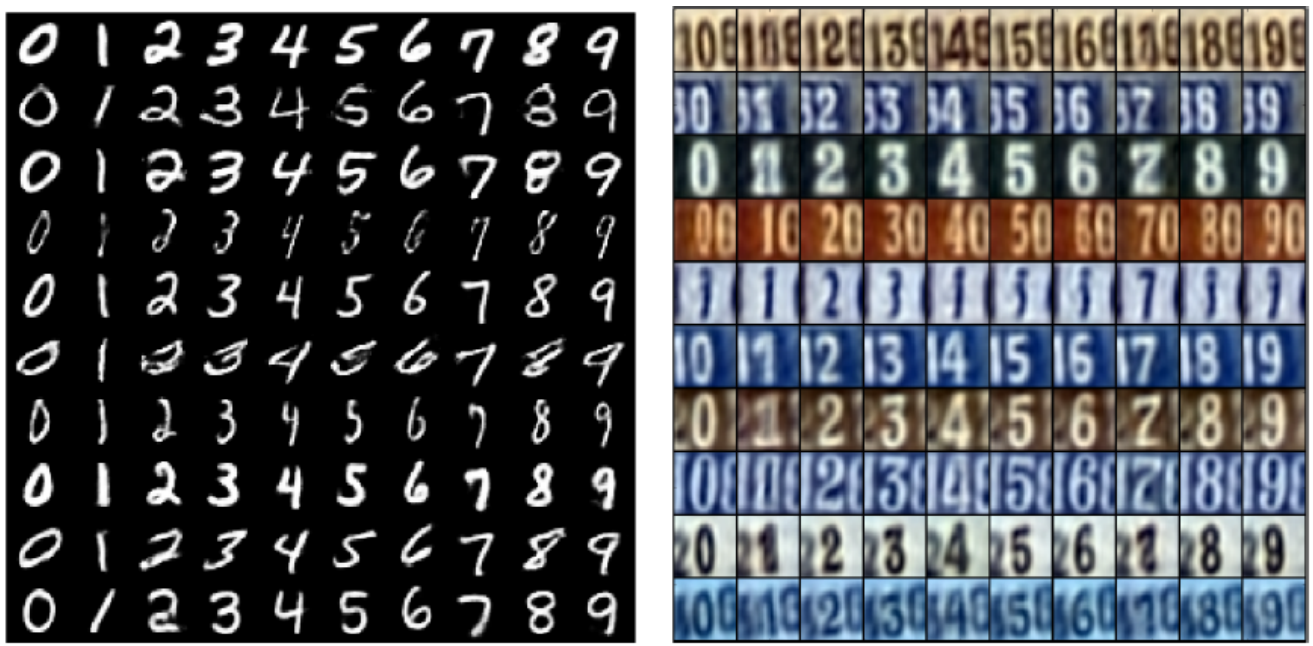


Figure 6: Disentangling the label information from the hidden code by providing the one-hot vector to the generative model. The hidden code in this case learns to represent the style of the image.

Disentangling Content and Style Results



(a) MNIST (b) SVHN

Figure 7: Disentangling content and style (15-D Gaussian) on MNIST and SVHN datasets.

Bidirectional GANs (BiGANs)

BiGANs are the first successful attempt to learn an encoder/decoder pair using purely GANs. Donahue et al. (2016)

- Encoder models $P(Z \mid X)$
- Decoder models $P(X \mid Z)$
- Discriminator takes a pair of x and z. It has the following task:
  - assign 1 to pairs of x=real data and y=encoding of real data
  - assign 0 to pairs of x=decoded prior samples and y=prior samples

BiGAN Architecture

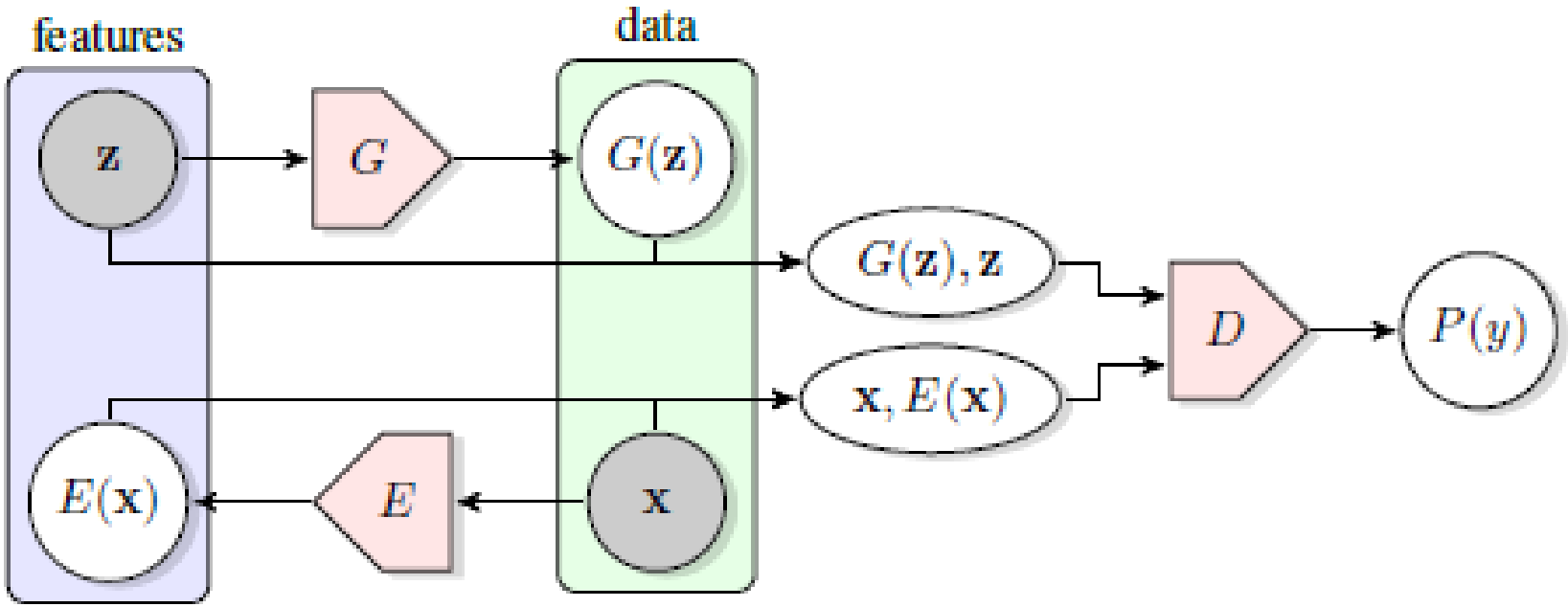


Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

BiGAN Objective

The BiGAN training objective is defined as a minimax objective

$$\min_{G,E} \max_{D} V(D, E, G) \tag{2}$$

where

$$V(D, E, G) := \mathbb{E}_{\mathbf{x} \sim p_{\mathbf{X}}} \Big[\underbrace{\mathbb{E}_{\mathbf{z} \sim p_E(\cdot|\mathbf{x})} [\log D(\mathbf{x}, \mathbf{z})]}_{\log D(\mathbf{x}, E(\mathbf{x}))} \Big] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{Z}}} \Big[\underbrace{\mathbb{E}_{\mathbf{x} \sim p_G(\cdot|\mathbf{z})} [\log (1 - D(\mathbf{x}, \mathbf{z}))]}_{\log(1 - D(G(\mathbf{z}), \mathbf{z}))} \Big]. \tag{3}$$

BiGAN Results

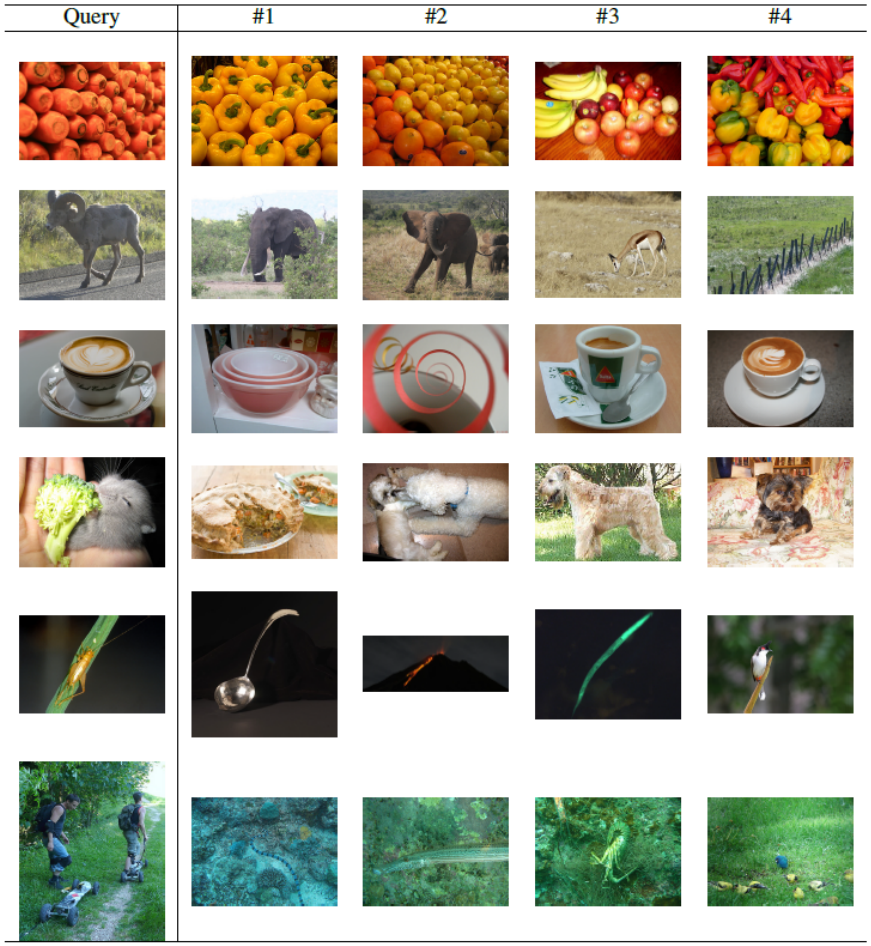


Figure 5: For the query images used in Krähenbühl et al. (2016) (left), nearest neighbors (by minimum cosine distance) from the ImageNet LSVRC (Russakovsky et al., 2015) training set in the *fc6* feature space of the ImageNet-trained BiGAN encoder E. (The *fc6* weights are set randomly; this space is a random projection of the learned *conv5* feature space.)

Optimization Issues

Generative Modeling is Solved!

Seems too good to be true? What is missing?

Optimization Issues

The main problem with GANs is that they are tricky to train. There are many tricks to train them better but some of those tricks do away with what makes GANs so special.

Unrolled Generative Adversarial Networks

GANs have trouble reaching a stationary state. By making decisions based on the opponents reactions, the model can achieve a steady state. Metz et al. (2016)

Improved Techniques for Training GANs

A large collection of techniques including minibatch discrimination and label smoothing. Salimans et al. (2016)

Wasserstein GAN

A theoretically grounded improvement on GANs but the implementation in practice tends to not work so well. Arjovsky et al. (2017)

Improved Training of Wasserstein GANs

An improvement on WGANs that appears to train much better and is the current standard. Gulrajani et al. (2017)

References

Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes, 2001.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *ArXiv e-prints*, June 2014.

Dev Nag. Generative adversarial networks (gans) in 50 lines of code (pytorch). `https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-py`

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014. URL `http://arxiv.org/abs/1411.1784`.

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. URL `http://arxiv.org/abs/1506.05751`.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL `http://arxiv.org/abs/1511.06434`.

J. T. Springenberg. Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks. *ArXiv e-prints*, November 2015.

Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *CoRR*, abs/1602.05110, 2016. URL `http://arxiv.org/abs/1602.05110`.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015. URL `http://arxiv.org/abs/1511.05644`.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016. URL `http://arxiv.org/abs/1605.09782`.

Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015. URL `http://arxiv.org/abs/1502.04623`.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *ArXiv e-prints*, June 2016.

Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016. URL `http://arxiv.org/abs/1611.02163`.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016. URL `http://arxiv.org/abs/1606.03498`.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, January 2017.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. *ArXiv e-prints*, March 2017.