10-707: Advanced Deep Learning **Spring 2020**

Lecture 16: March 23

Lecturer: Andrej Risteski *Scribes: Gregory Howe, Samarth Malhotra, George Cazenavette*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

16.1 Intro to GANs

16.1.1 The idea behind GANs

- Matching a distribution on images is hard because we don't have good measures of "distance" between images.
- Intuitively, two images could be very different in pixel space, while "semantically" being the same image.
- Instead of fitting "maximum likelihood", we will instead learn a distribution close to the distribution of the input data: "likelihood-free" model

16.1.2 The GAN Paradigm

- Proposed by Goodfellow et al. '14
- Goal: Learn a distribution close to some distribution we have few samples from.
- Approach: Fit distribution P_g parametrized by neural network g

16.1.2.1 Game theoretic idea

- Generator trained to fool discriminator.
- Discriminator trained to beat generator.

16.1.3 W-GAN formalization

Min-max problem:

- Min-player: generators $g \in G$; Max-player: discriminators $f \in F$
- Samples from image distr. P_{real}. Uniform distribution over samples: $P_{samples}$
- P_g - generator distribution: $Z \sim N(0, I) \to g(Z)$

Training loss: Difference of expectation of f on samples vs generated images

$$\min_{g \in G} \max_{f \in F} |\mathbb{E}_{P_g}[f] - \mathbb{E}_{P_{samples}}[f]|$$

Generator g fools discriminators F (small training loss) if:

$$\forall f \in F, \mathbb{E}_{P_g}[f] \approx \mathbb{E}_{P_{samples}}[f]$$

Discriminators F beat generators if:

$$\forall g \in G, \exists f \in F : \mathbb{E}_{P_g}[f] \not\approx \mathbb{E}_{P_{samples}}[f]$$

16.1.4 Distances d_F

$$d_F(P_{samples}, P_g) = \max_{f \in F} |\mathbb{E}_{P_g}[f] - \mathbb{E}_{P_{samples}}[f]|$$

$$\implies \text{Training loss} = \min_{g \in G} d_F(P_{samples}, P_g)$$

16.1.4.1 Examples

- $F = \{f : |f|_\infty \leq 1\}$: Total variation distance. Measures differences of bounded functions.
- $F = \{f : Lip(f) \leq 1\}$: W1 (Wasserstein, earthmover) distance. Measures differences of 1-Lipschitz functions

If d_F is a metric: $d_F(p, q) \leq 0$ and $d_F(p, q) = 0$ only if $p = q$
Then if we learn P_g such that $d_F(P_g, P_{real}) = 0$, we have that $P_g = P_{real}$

16.1.5 Variants

$$d_F(P_{samples}, P_g) = \max_{f \in F} |\mathbb{E}_{P_g}[\phi(f)] - \mathbb{E}_{P_{samples}}[\phi(1 - f)]|$$

ϕ is Monotone function.

- $\phi = 1$: W-GAN, if F closed under negating ($f \in F \implies -f \in F$)
- $\phi = \log$: DC-GAN (original GAN)

If F is unconstrained, objective can be written as

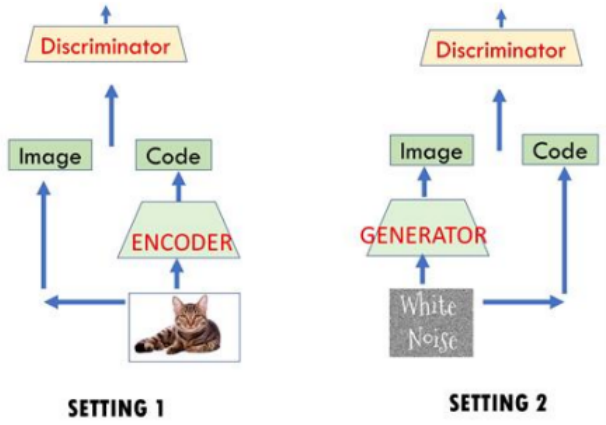
$$\min_{g \in G} KL(P_g || P_{real}) + KL(P_{real} || P_g) - 2 \ln 2$$

Where $KL(P_g || P_{real}) + KL(P_{real} || P_g)$ is the Jensen-Shannon divergence

16.1.6 What about autoencoders?

If we also want to train an encoder E: that is, a network that tries to output the z, s.t. x was generated from z, there is a way to adapt the adversarial setup: Discriminator tried to distinguish between

- Setting 1: samples are $(x, E(x))$
- Setting 2: samples are $(z, G(z))$



In the limit of infinite samples, infinite capacity generators, the distributions of Setting 1 and 2 match.

16.1.7 What affects our choice of F?

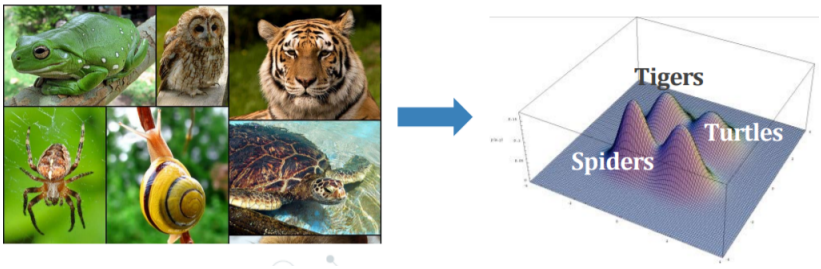
- Statistical considerations: very powerful discriminators (e.g. large neural networks) will require a lot of samples. Weak discriminators will specify a very weak metric: very "different" distributions will look very "similar" to metric. (We have good understanding here)
- Algorithmic considerations: if discriminators are very powerful, gradient information for generator is too weak and can vanish. If they are too weak – metric is weak. (We have very poor understanding of training dynamics here)

16.2 Statistical Questions

16.2.1 Tension: Strength of Discriminators

16.2.1.1 Problems with Small/Weak Discriminators

Mode Collapse



- For any distribution P_{real} any neural network with $\leq$ m parameters can be fooled by a generator with support size $\approx$ m [Arora et al'17, Arora-Risteski-Zhang ICLR'18].
- The above problem isn't due to memorization; as we increase our amount of data, this problem does not go away.
- The discriminator d_f fundamentally cannot distinguish from a small-support distribution and the distribution P_{real}.
- Mode collapse tends to be a large issue because natural/real-life distributions tend to have large supports.

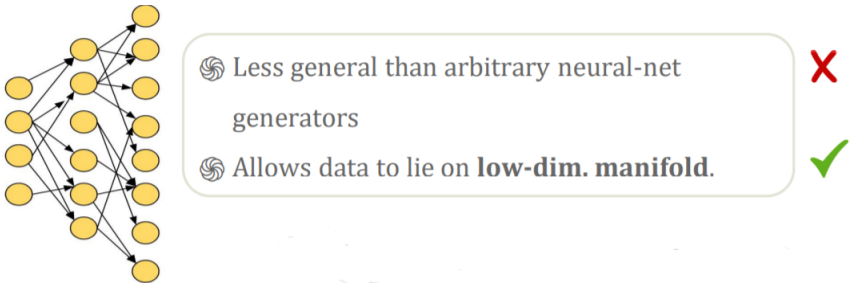
16.2.1.2 Problems with Large Discriminators

Poor generalization

- Loss with a small number of samples differs a lot from the loss with an infinite number of samples ie $d_F(P_{samples}, P_g) \not\approx d_F(P_{real}, P_g)$.

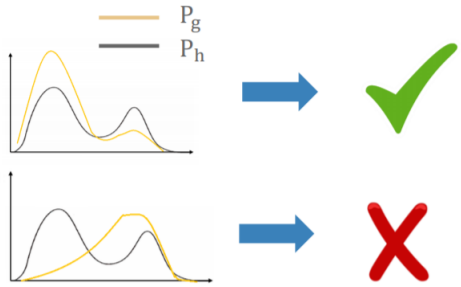
16.2.2 Sweet Spot for Natural Distributions

Let $g \in G$ where G is the set of {1-to-1 neural networks of bounded size }
Fix $P_{real} = P_g$
Let $P_{real} = P_g$ where $g \in G$
This set of generators has the properties



With these restrictions we get the following result
The class of discriminators F have distinguishing power against the generators from G of

$$\forall g, h \in G : d_F(P_g, P_h) \gtrsim W_1(P_g, P_h)$$



16.2.3 Main Result

Theorem (Bai-Ma-Risteski ICLR'19) Small discriminators F with distinguishing power for G = {1-to-1 neural nets of bounded size} exist.

More formally if P_{real} is generated by a 1 to 1 neural network with d parameters with poly(d) samples then,

$$d_F(P_{samples}, P_g) \leq \epsilon \Rightarrow W_1(P_{real}, P_g) \leq O(\sqrt{\epsilon})$$

Less formally, if we were able to get our training objective to be small with not to too many samples then we have learned a distribution which is close to the distribution we were trying to learn.

The class of discriminators F are neural networks slightly larger than the generator and with one more layer these properties suffice for this result to go through.

16.3 Algorithmic Questions

16.3.1 Introduction to Training

The high level idea is to use a game theory approach using the best response dynamics:

1. Fix the strategy of player 1
2. Calculate best strategy for player 2 for this strategy
3. Fix the strategy of player 2
4. Calculate best strategy for player 1 for this strategy
5. Repeat until convergence criteria are met

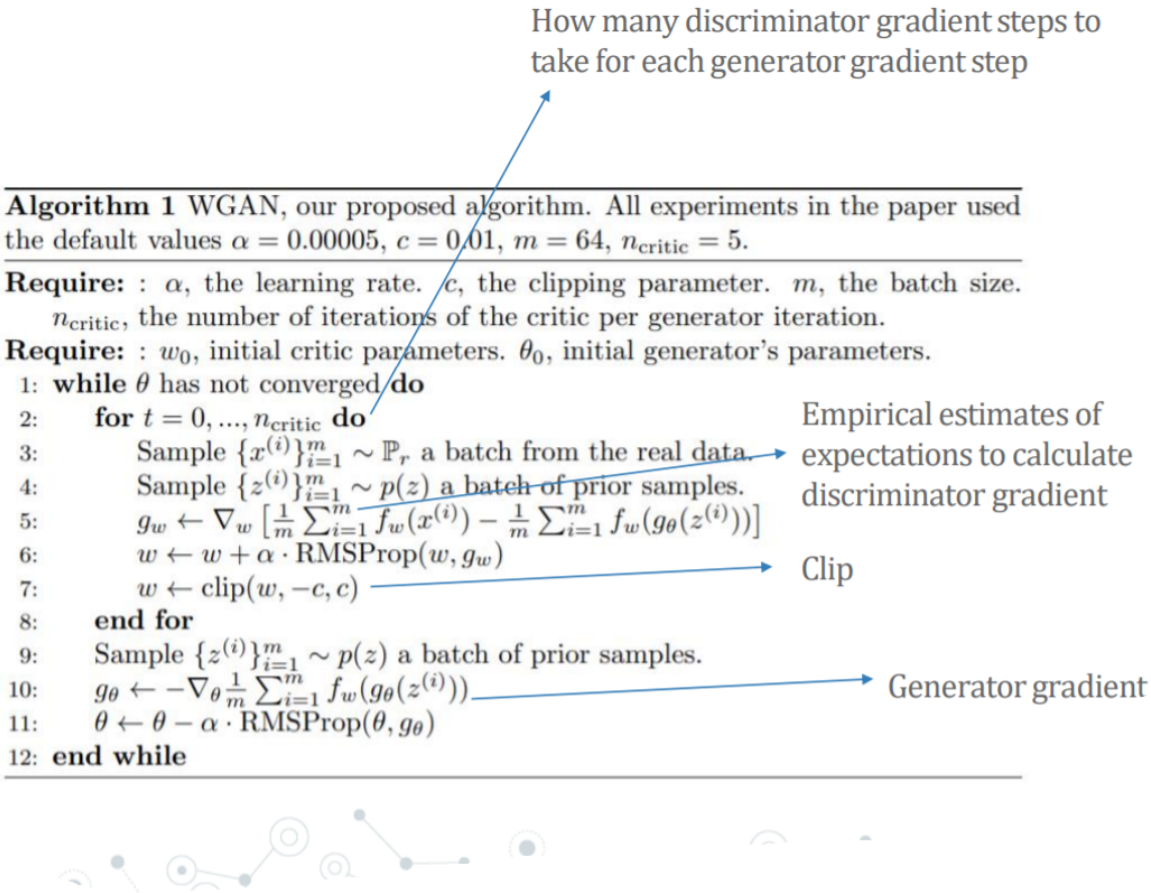
This has been shown to not converge on even simple games, but is a natural algorithm to try for training GANs. We simply set the generator to be player 1 and the discriminator to be player 2.

However, it has been shown to be better in practice take one gradient step for generator, then do a few gradient steps for discriminator and repeat.

Going with intuition of Wasserstein distance: we'd like the discriminators to be somewhat Lipschitz. We can easily this by clipping the weights.

16.3.2 Training Algorithm

The algorithm is to do gradient descent over the weights. We use a clipping parameter to monitor the clipping and the rest of the algorithm is simple batch gradient descent using RMSProp which can be changed as needed. We typically use a Guassian to sample from the prior distribution, but this can be modified as needed.



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

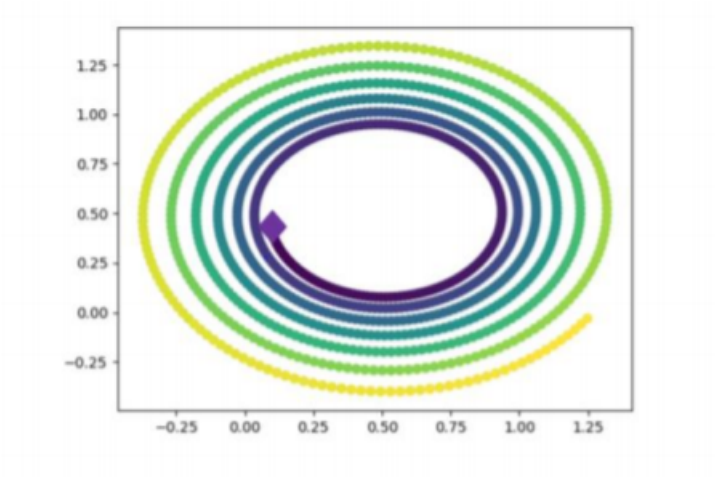
Require: : α, the learning rate. c, the clipping parameter. m, the batch size.
n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

16.3.3 Common Training Problems

1. Unstable Training: The problem we are solving is a min-max problem (also called saddle point problem). Thes optimization for these is typically much less stable than pure minimization.

   One particularly common instantiation is cycling, where the parameters start spiralling away from the optiimum. This is often solved by taking the average of the parameters. The cause of this is as player 1 needs minimize a convex function, but player 2 needs to maximize a concave function.



2. Vanishing gradient: If the discriminator is too good, the generator gradients have a propensity to be small. This is concerning, as to be taking gradients of the Wasserstein/JS/... objective, the discriminator needs to be optimal. Thus, we can't use discriminators which are not that too good.

   However, this is a much smaller problem for more modern GANs. The following graph shows the vanishing gradient well as when we use an already trained discriminator to retrain the generator, the magnitude of the gradient drops heavily.

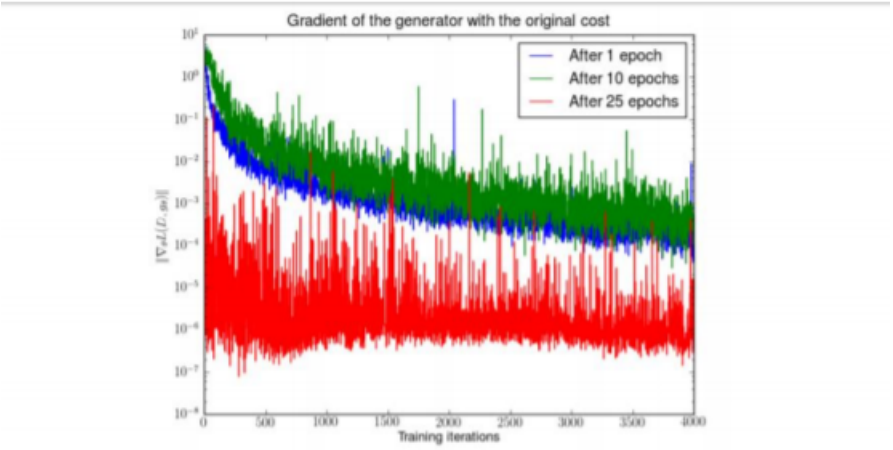
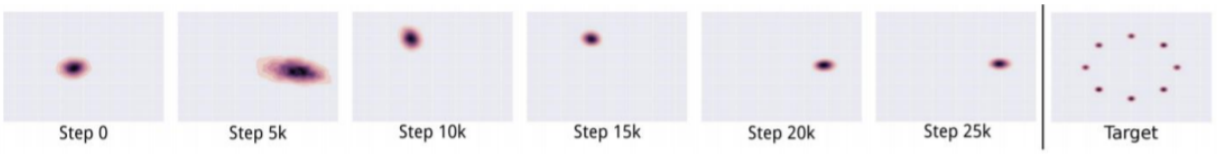


Figure 2: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale.

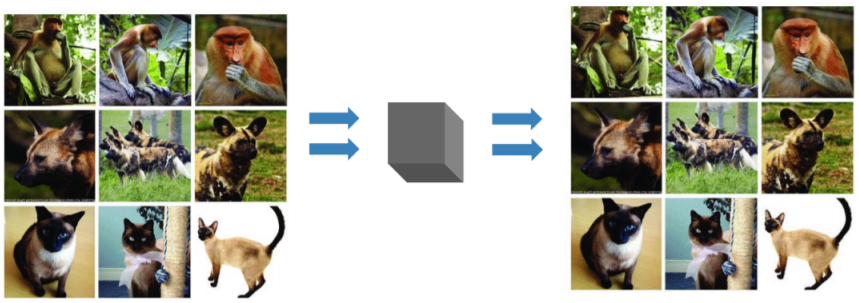
3. Mode collapse: This occurs when the training only recovers some of the modes of the underlying distribution. It is not at all clear if this is a statistical or algorithmic problem. This is the hardest problem to solve for GANs.



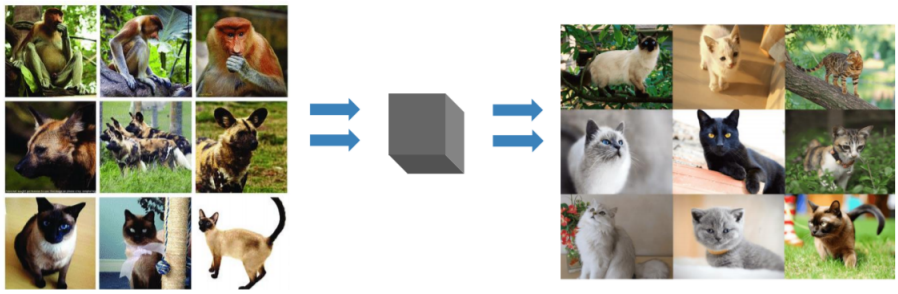
16.4 Evaluating GANs

16.4.1 Introduction

The worry with GANs is if they are actually learning the distribution as they can clearly generate photo-realistic images. The following are input images and output images from a GAN.



In this case, it is clearly memorizing training examples. Ideally we would want it to generate new images of examples.



In this case, it is only generating images of cats and not other animals, thus there is mode collapse and it is missing modes of the distribution. This is also known as having a small support for the learned distribution.

Solving this and evaluating GANs is hard as we cannot use likelihood based models since they would end up being ill defined. Even in academic papers, frequently, the evaluation is done by visually comparing samples. However, we can still try to test for some common failure modes. We shall show 3 common methods of doing this:

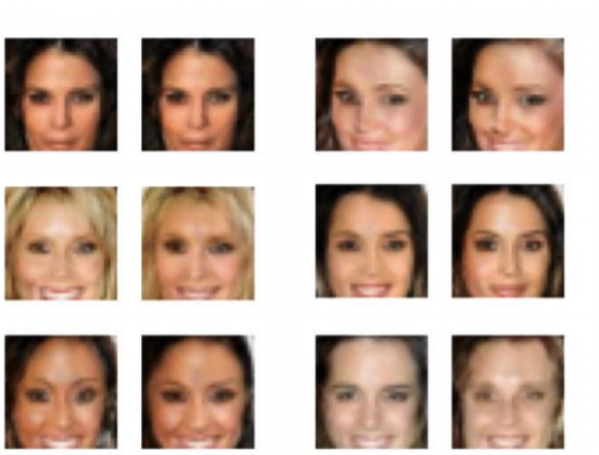
16.4.2 Birthday Paradox

We first define the birthday paradox: Birthday Paradox: If there are 23 people in a group, there is a $> \frac{1}{2}$ chance that two of them share a birthday. The more general version of this is: Suppose a distribution is uniform over N images. Then Pr[sample of size $\sqrt{N}$ as a duplicate image] $> \frac{1}{2}$. Thus, we can use this to evaluate the support size of our GANs. The idea was actually was actually worked on by Professor Risteski himself!

We implement this as following:

1. Draw sample of size s
2. Heuristically flag possible near-duplicates, can use L_2, L_1 etc.
3. Use human in the loop to verify duplicates

We can use this to evaluate the support size of the following types of GANs. The following image shows this metric on a facial image dataset on some GAN architectures



CelebA (faces): 200k training images

DC-GAN [Radford et al.'15]:
Support size ≈ 250K

BiGAN [Donohue et al.'17] and
ALI [Dumoulin et al.'17]:
Support size ≈ 1M

From this, we can see that we're learning distributions that have a much lower diversity than the underlying distribution.

16.4.3 Interpolation

We study the different latent variables that result in 2 different images. We take 2 images and draw lines between them. In different intervals between these 2 lines we try to figure out what the generator would've generated and try to make sure that it is something sensible that looks like it follows the underlying distribution. The following image shows a good representation of how interpolation should look.

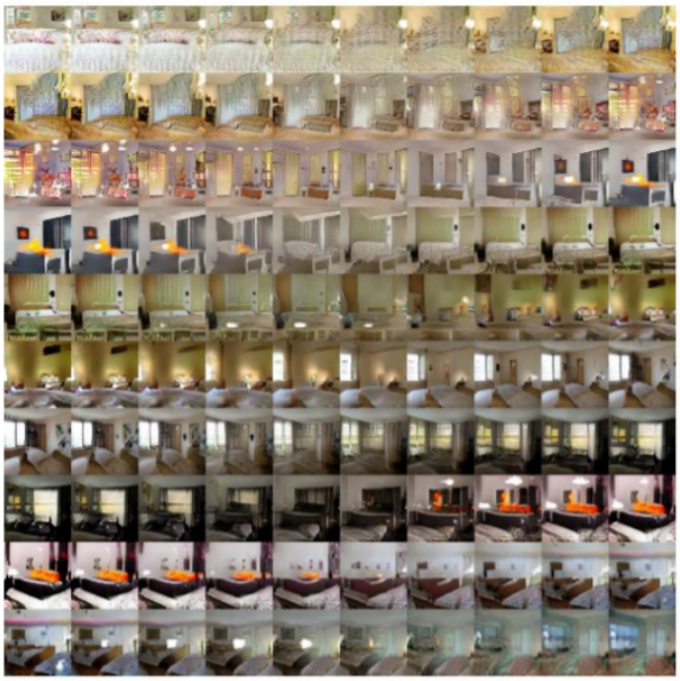


Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

The intuition is that if you are learning meaningful images and don't have sharp transitions from endpoint to another, then you're learning something meaning in the latent space and the lack off sharp transitions implies a lack of copying the training set as well.

16.4.4 Inception Score

Take a trained network, commonly used the Inception architecture, as a labeler for images. Inception gives probability over labels y for sample x: $p(y|x)$.

We want the classifier to be "sure" of the images and the labels it selects the images to be. We can do this by looking for a low entropy in $p(y|x)$. However, we also want the distributions to be diverse, thus we want $\mathbb{E}_{x\sim P_g} p(y|x)$ to have high entropy. Note that we want $H(p(y))$ to be high and $H(p(y|x))$ to be low. Now, let's consider the following:

$$\begin{aligned}\mathbb{E}_{x\sim P_g} KL(p(y|x), p(y)) &= \mathbb{E}_{x\sim P_g}\mathbb{E}_{y\sim P_{y|x}} \log(p(y|x)) - \log(p(y)) \\ &= \mathbb{E}_{x\sim P_g}(H(p(y|x))) + H(p(y))\end{aligned}$$

Thus, the above term is able to capture our goals of entropy for the inception scores. Thus we use the following definition of inception score as exponents make things nicer.

$$\textbf{Inception Score} \ = \exp(\mathbb{E}_{x\sim P_g} KL(p(y|x), p(y)))$$