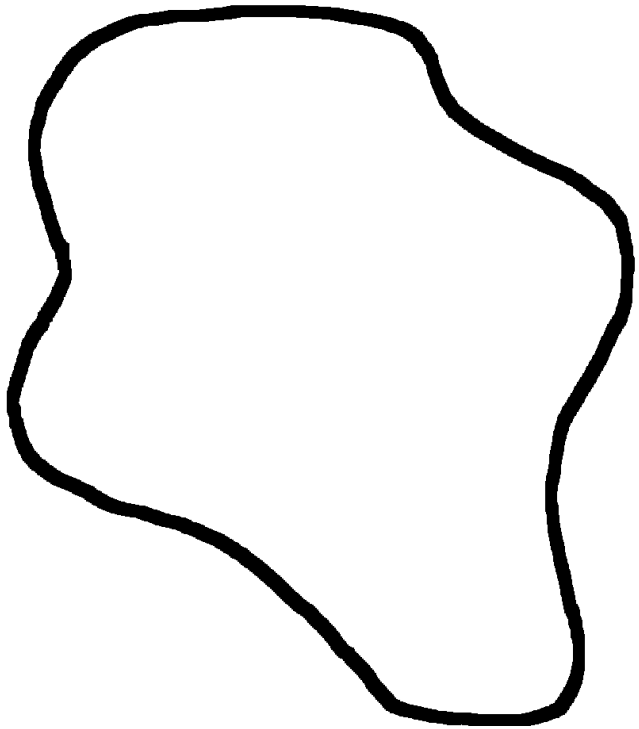
UO 607: Generative Models

Steven Walton
University of Oregon

23 October 2020

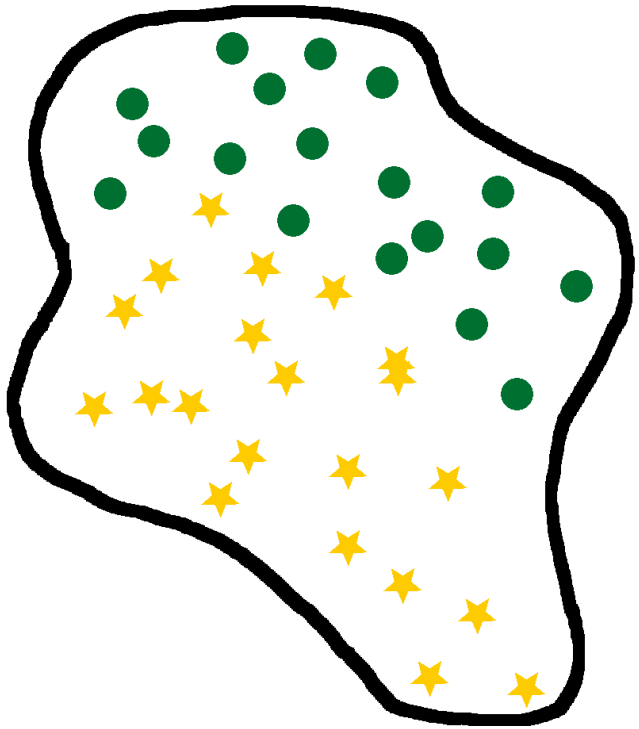
Generators

- Suppose that we have some probability space S.



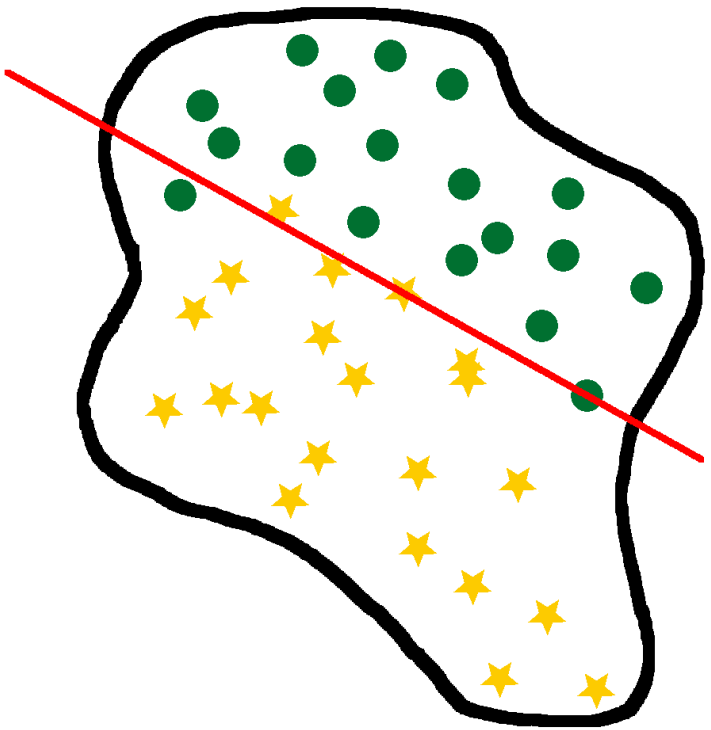
Generators

- Suppose that we have some probability space S.
- S has two variables X (instances) and Y (labels) that represent cats and dogs.



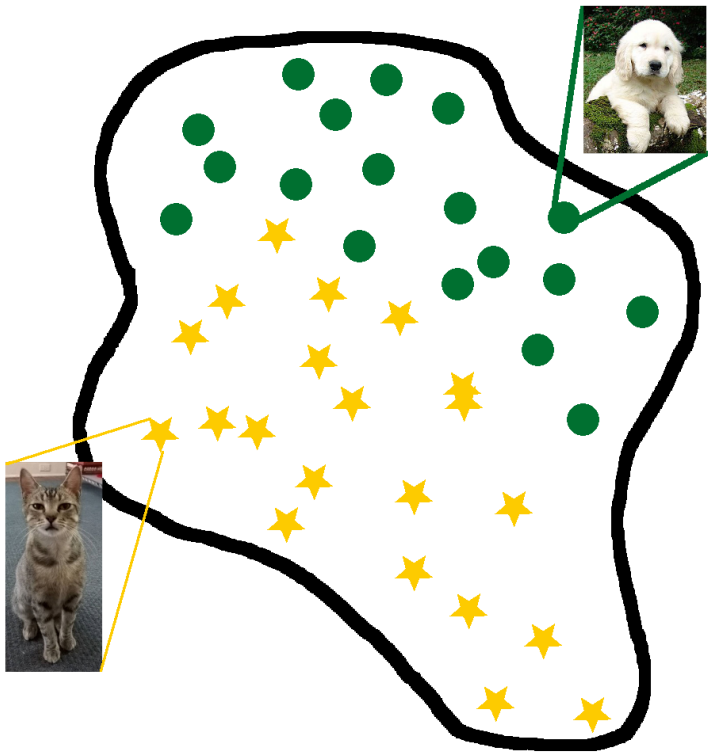
Generators

- Suppose that we have some probability space S.
- S has two variables X (instances) and Y (labels) that represent cats and dogs.
- Often we want to find a boundary between different types of data within a sample space. (Discrimitive Model)



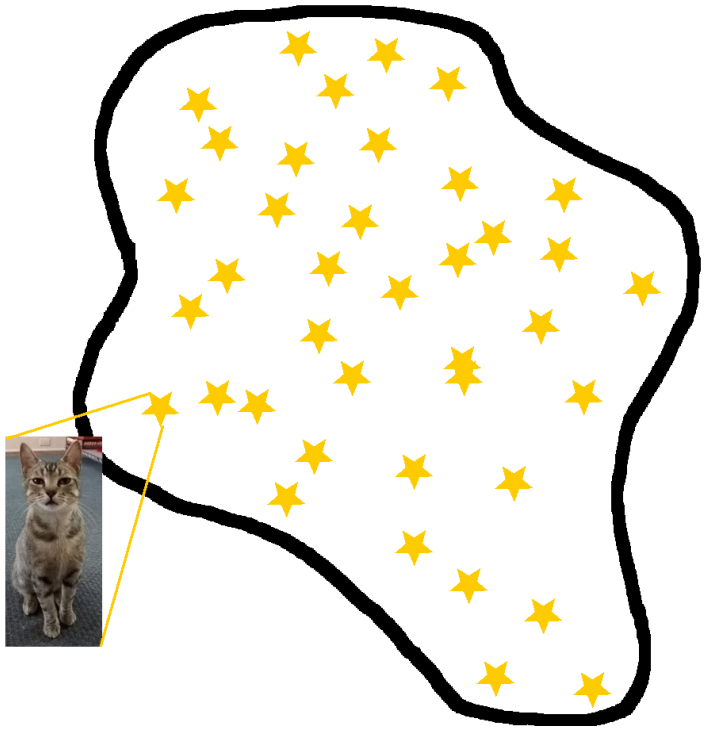
Generators

- Suppose that we have some probability space S.
- S has two variables X (instances) and Y (labels) that represent cats and dogs.
- Often we want to find a boundary between different types of data within a sample space. (Discrimitive Model)
- We might instead want to *sample* from this space and see what images are here.

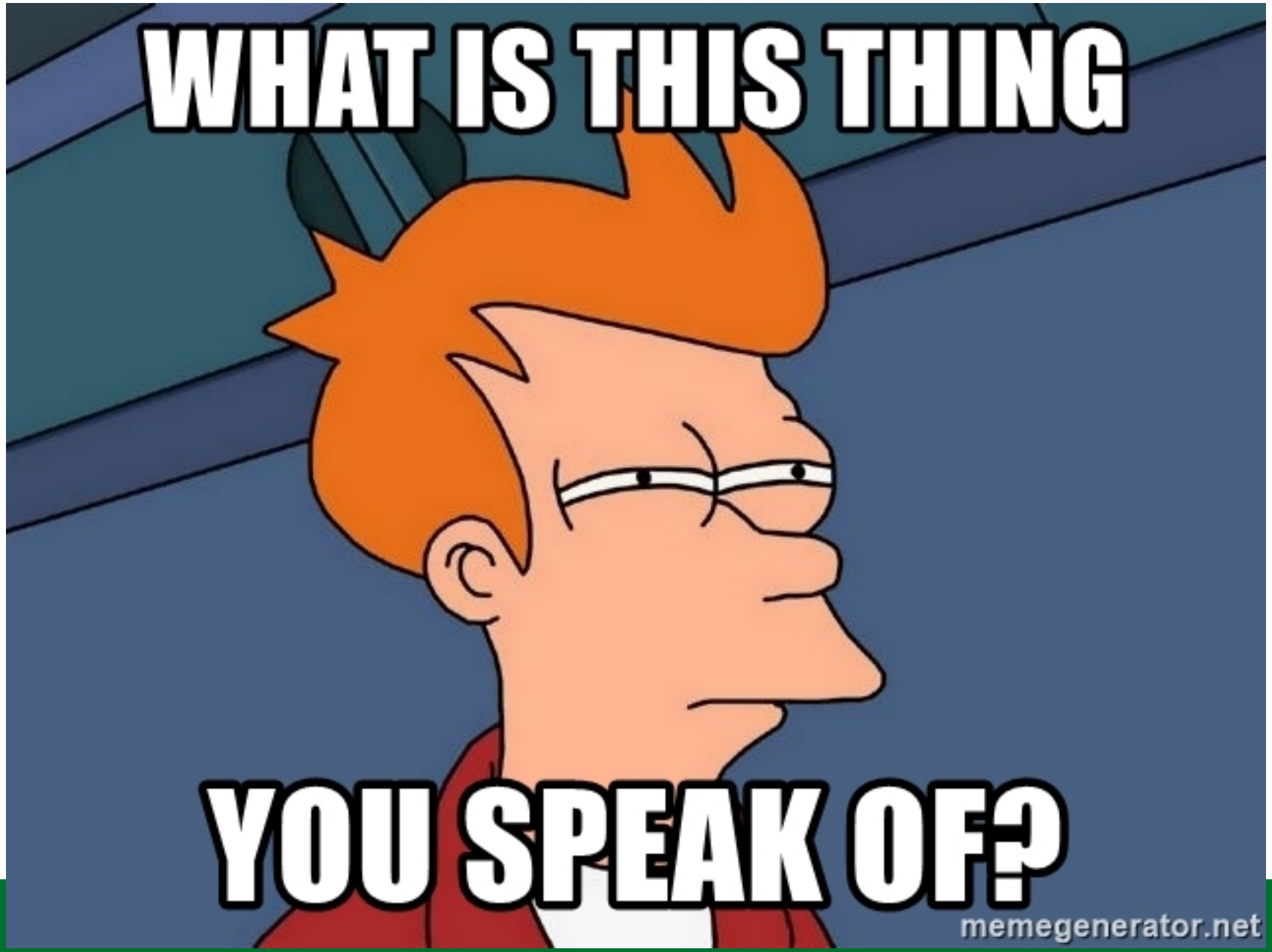


Generators

- Suppose that we have some probability space S.
- S has two variables X (instances) and Y (labels) that represent cats and dogs.
- Often we want to find a boundary between different types of data within a sample space. (Discrimitive Model)
- We might instead want to *sample* from this space and see what images are here.
- Typically we'll only work with a single classification.



What Is A Generator?



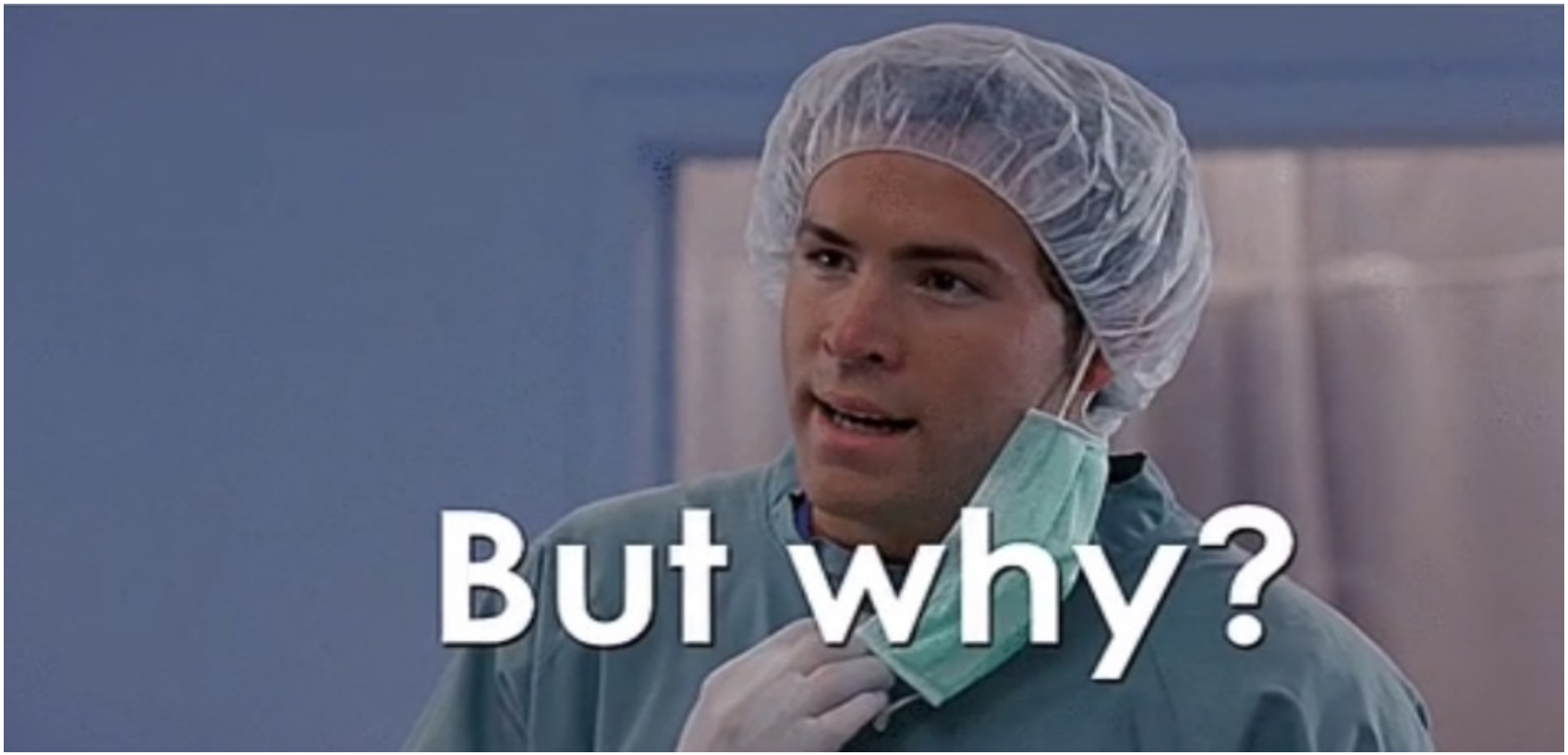
Generators

- In a Discriminitive model we learn a conditional probability $P(Y = \mathit{label} | X = \mathit{image})$ (predict the label given an image).
- In a Generative model we learn to sample for p(x) (produce an image).

Generators

- In a Discriminitive model we learn a conditional probability $P(Y = label|X = image)$ (predict the label given an image).
- In a Generative model we learn to sample for p(x) (produce an image).
- In reality we learn $p_\theta(\hat{x})$ from $p_{true}(x)$

But Why?



But Why?

- We can create images that don't exist.

But Why?

- We can create images that don't exist.
- We can create new data.

But Why?

- We can create images that don't exist.
- We can create new data.
- We can upscale images/videos.

But Why?

- We can create images that don't exist.
- We can create new data.
- We can upscale images/videos.
- We can lean latent variables.

But Why?

- We can create images that don't exist.
- We can create new data.
- We can upscale images/videos.
- We can lean latent variables.
- We can uncover biases.

But Why?

- We can create images that don't exist.
- We can create new data.
- We can upscale images/videos.
- We can lean latent variables.
- We can uncover biases.
- And so much more!

But Why?: Generate Images

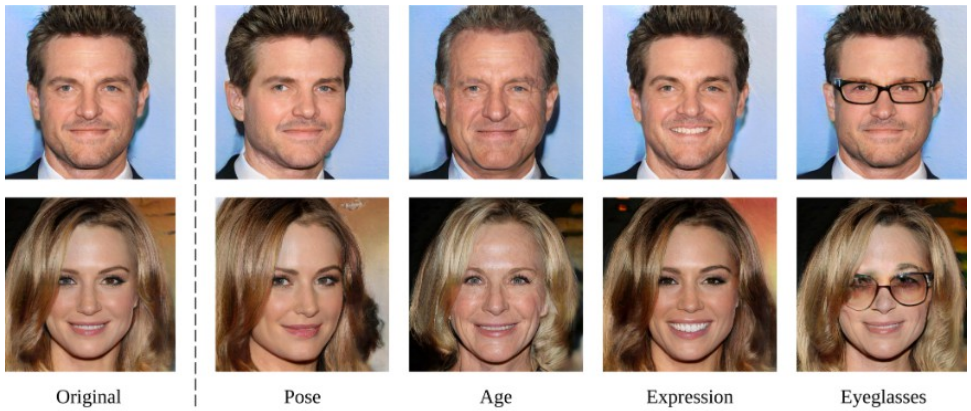
- We can generate cats that don't exist.
- Helps create new data within a dataset.
- Can create fake characters/animals/scenes that don't exist in real life but look realistic (art).
- Don't have to pay models or liability.

But Why?: Upscaling

- We can take lossy images and produce higher quality images.
- We can also take smaller images and make them larger.
- Helpful in compression.
- Restoration of images (also inpainting)
- Make old images/videos look new/modern.

But Why?: Learn Latent Variables

- We can learn the different latent variables in a dataset.
- For example we can learn how to make someone old!
- We could also use this as a form of compression (autoencoders)
- We can add/modify features of someone (add a mostache).

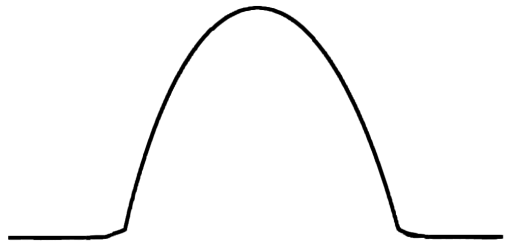


But Why?: Uncovering Bias

- Suppose we have a dataset of faces.

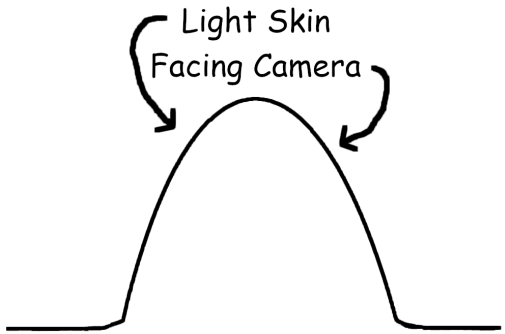
But Why?: Uncovering Bias

- Suppose we have a dataset of faces.
- Might have some distribution, let's say...



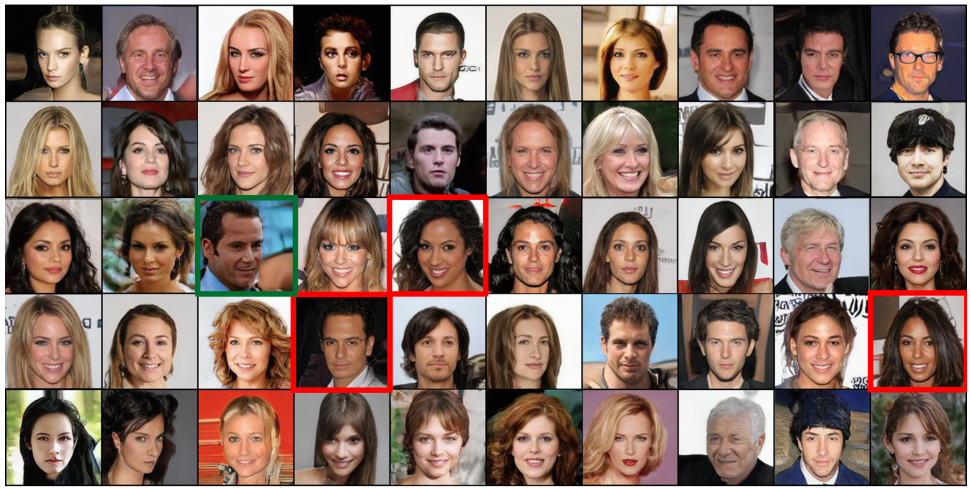
But Why?: Uncovering Bias

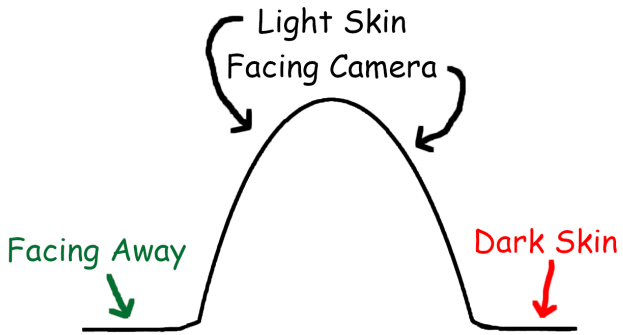
- Suppose we have a dataset of faces.
- Might have some distribution, let's say...
- Most of the faces fit in the main part of the distribution.



But Why?: Uncovering Bias

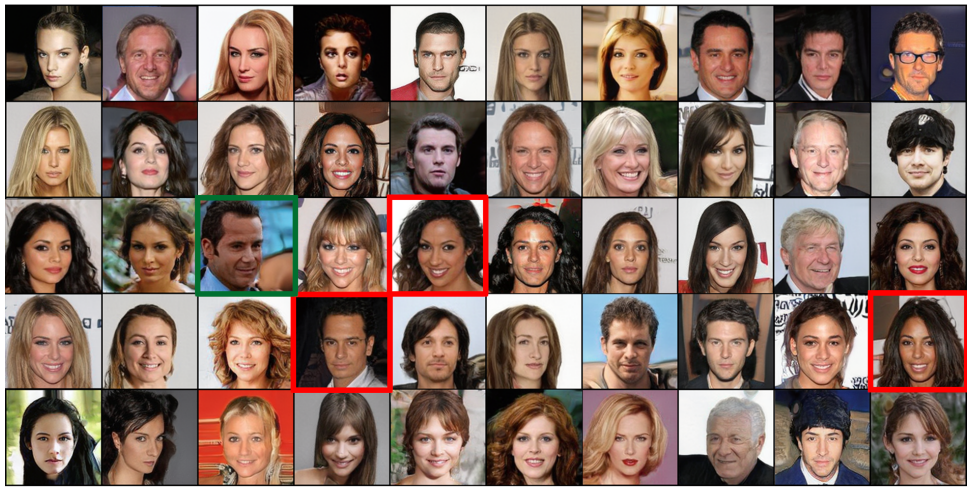
- Suppose we have a dataset of faces.
- Might have some distribution, let's say...
- Most of the faces fit in the main part of the distribution.
- We have outliers in the dataset.

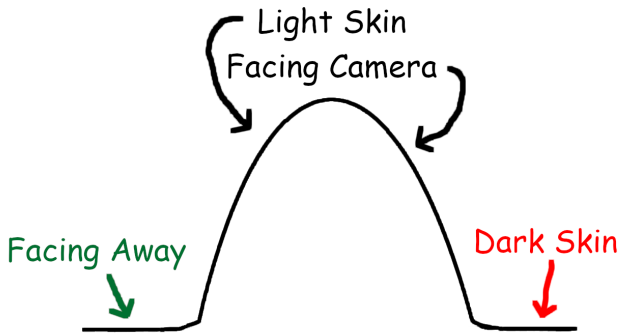




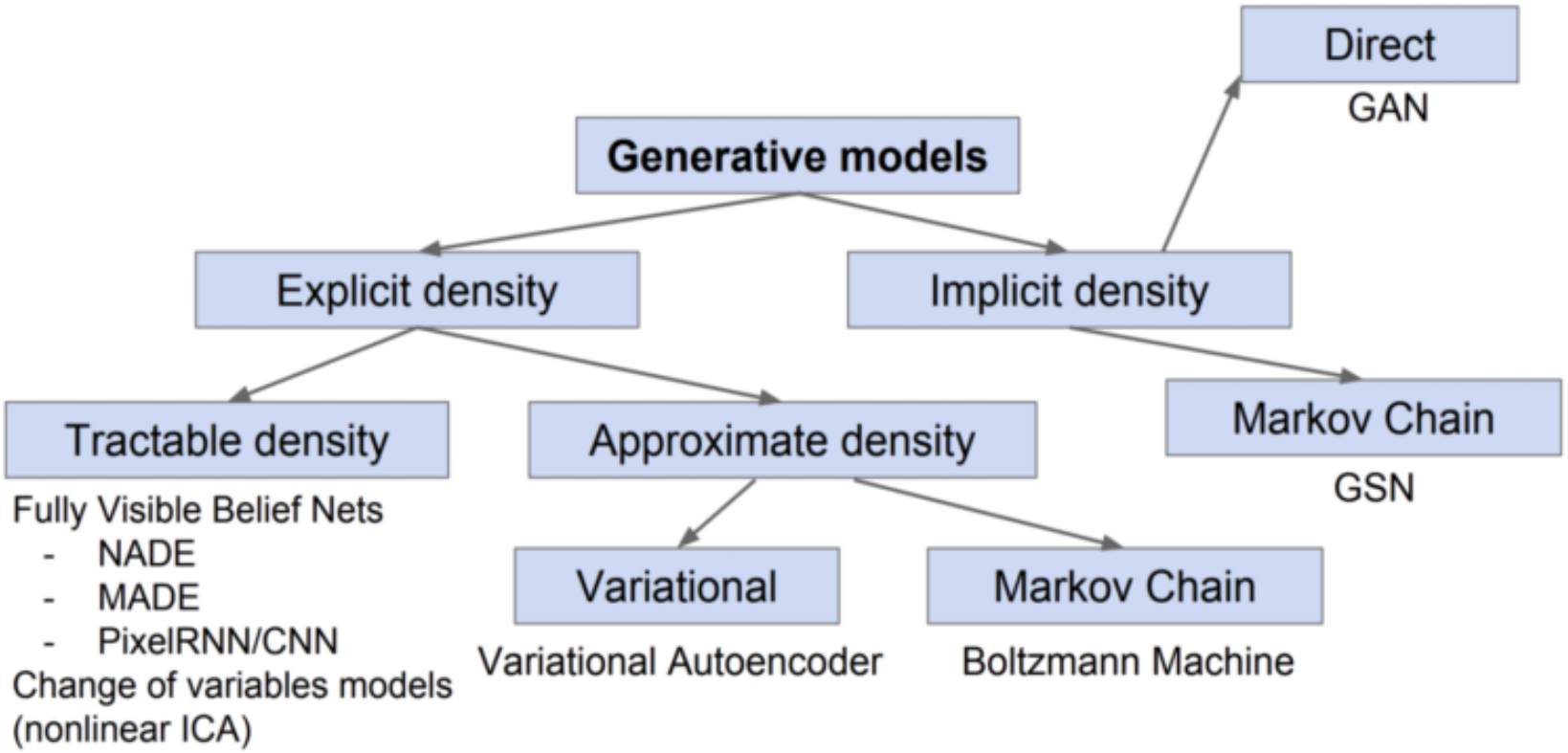
But Why?: Uncovering Bias

- Suppose we have a dataset of faces.
- Might have some distribution, let's say...
- Most of the faces fit in the main part of the distribution.
- We have outliers in the dataset.
- We can now adjust our dataset (gather new data)or potentially generate new images for a classifier if we can't fix the distribution (might not be logistically possible).





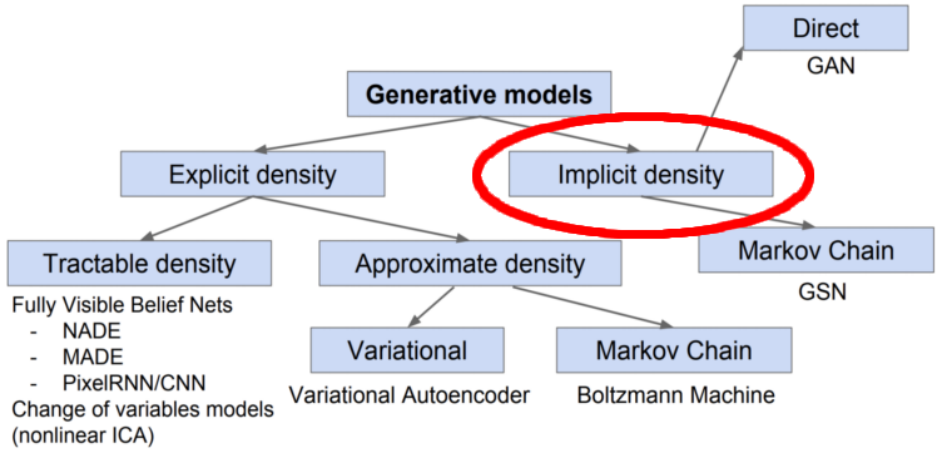
Taxonomy of Generators



source: ian goodfellow

Taxonomy of Generators: Implicit Density

- We want to learn a stochastic procedure to generate data.
- We learn liklihood-free.



source: ian goodfellow

This X Does Not Exist

(a) This Person Does Not Exist (StyleGAN2)

(b) This Cat Does Not Exist (StyleGAN)

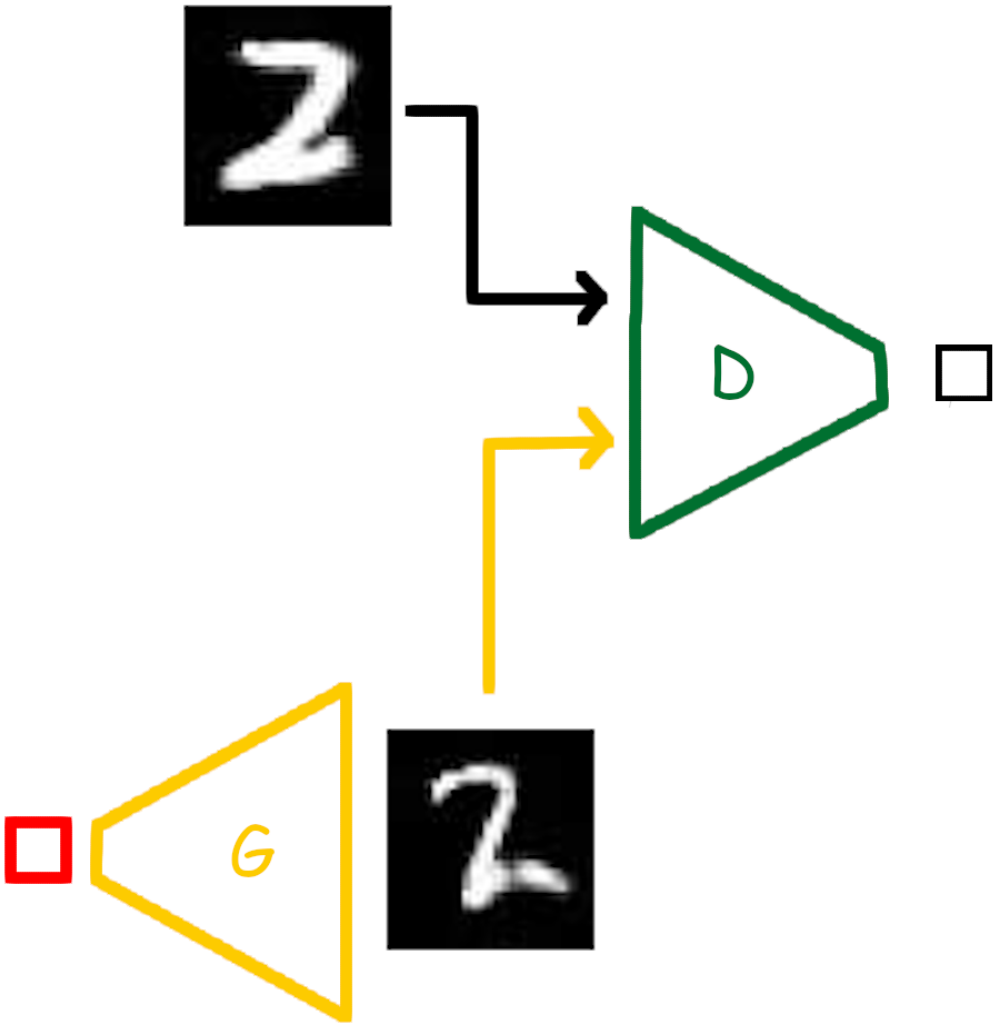
Do Electric Neural Nets Dream Of Anime Shows?

Review of the themes and plot of the first episode of the new anime "Toban no Oji-sama" aka "The Master of Balloons," which will debut in Japan this October. Official Summary from the Anime's Official Website | Crunchyroll | Hulu | Funimation | AnimeLab Toban no Oji-sama (The Master of Balloons) | Official Website via Crunchyroll
Synopsis: In a city of recycled materials and discarded trash piles where children with superpowers as rubbish cleaners face off against each other to scrounge up food to survive, a single young boy with the power of "rubbish" dreams of becoming the "King of the Filth."
"The King of the Filth" is a local legend who is said to gather piles of discarded garbage and transform them into treasure. We meet a boy called "Shin." He is the smallest of his gang but has incredible strength. He dreams of the "King of the Filth," and aspires to become him, but has no idea how to go about doing so. Then one day, a gang leader from another district appears and steals the bag that Shin has packed and prepared for a long time, which leads to a big fight. While Shin is desperate to retrieve his bag, he is tempted by a wild idea that "the King of the Filth can give things back." After stealing a balloon from a boy with the power of destruction, he makes a decision...
The theme of this work appears to be "optimism" and "dreams." Child Director and Animator Shigeyasu Yamauchi's earliest work "Niji no Hashi" (= Rainbow Bridge) about an abandoned island in the Pacific Ocean ruled by Children with the strength of darkness, and the drama of psychological issues of a young boy who ran away from home broadcast in March of 2017. The two works depict the struggles of boy Children in totally different ways, and the new work is completely his original creation. His seemingly light hearted and com-

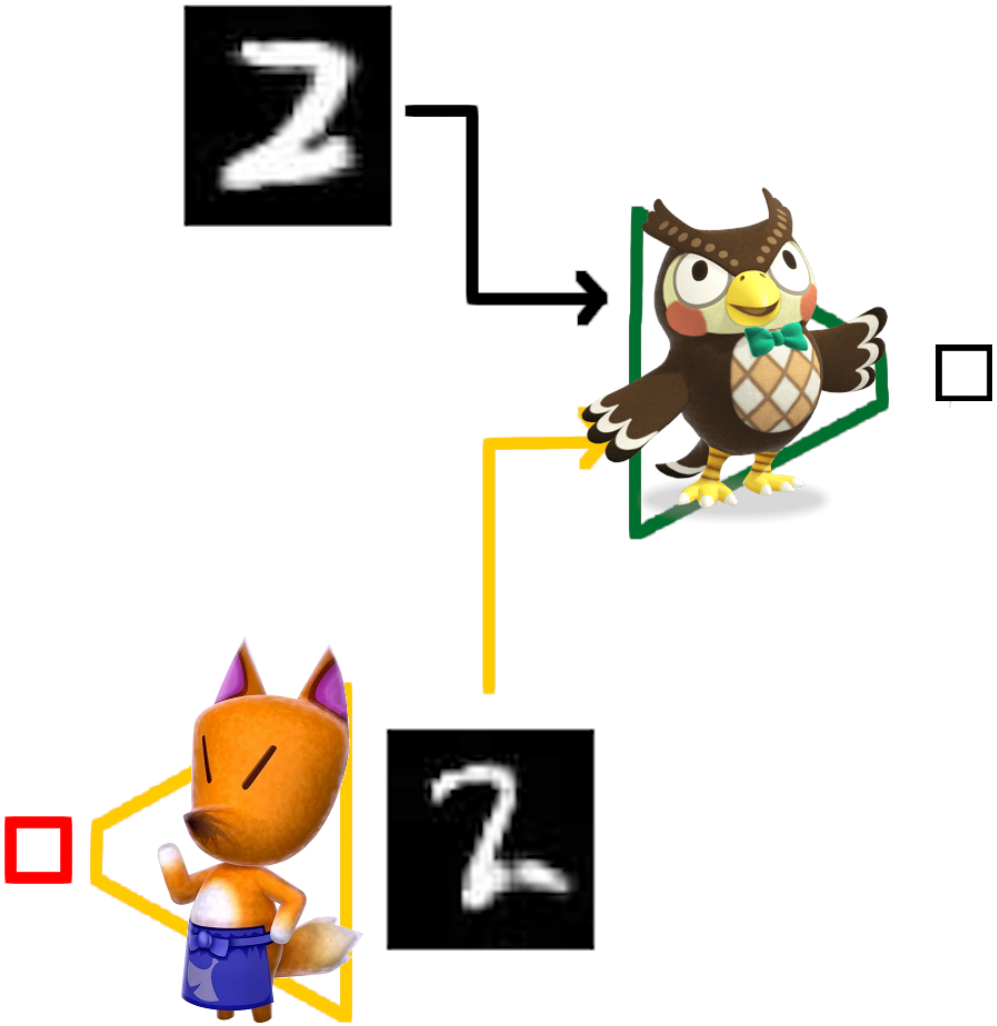
(c) This Anime Does Not Exist (StyleGAN2 + GPT-3)

Generative Adversarial Networks

Generative Adversarial Networks



Generative Adversarial Networks



Generative Adversarial Networks

- Send noise into Generator.
- Send generated image and real images into Discriminator.
- Play mini-max game teaching the Generator to fool the Discriminator.
- When fake image can adaquately fool (a well trained) discriminator we produce quality images.

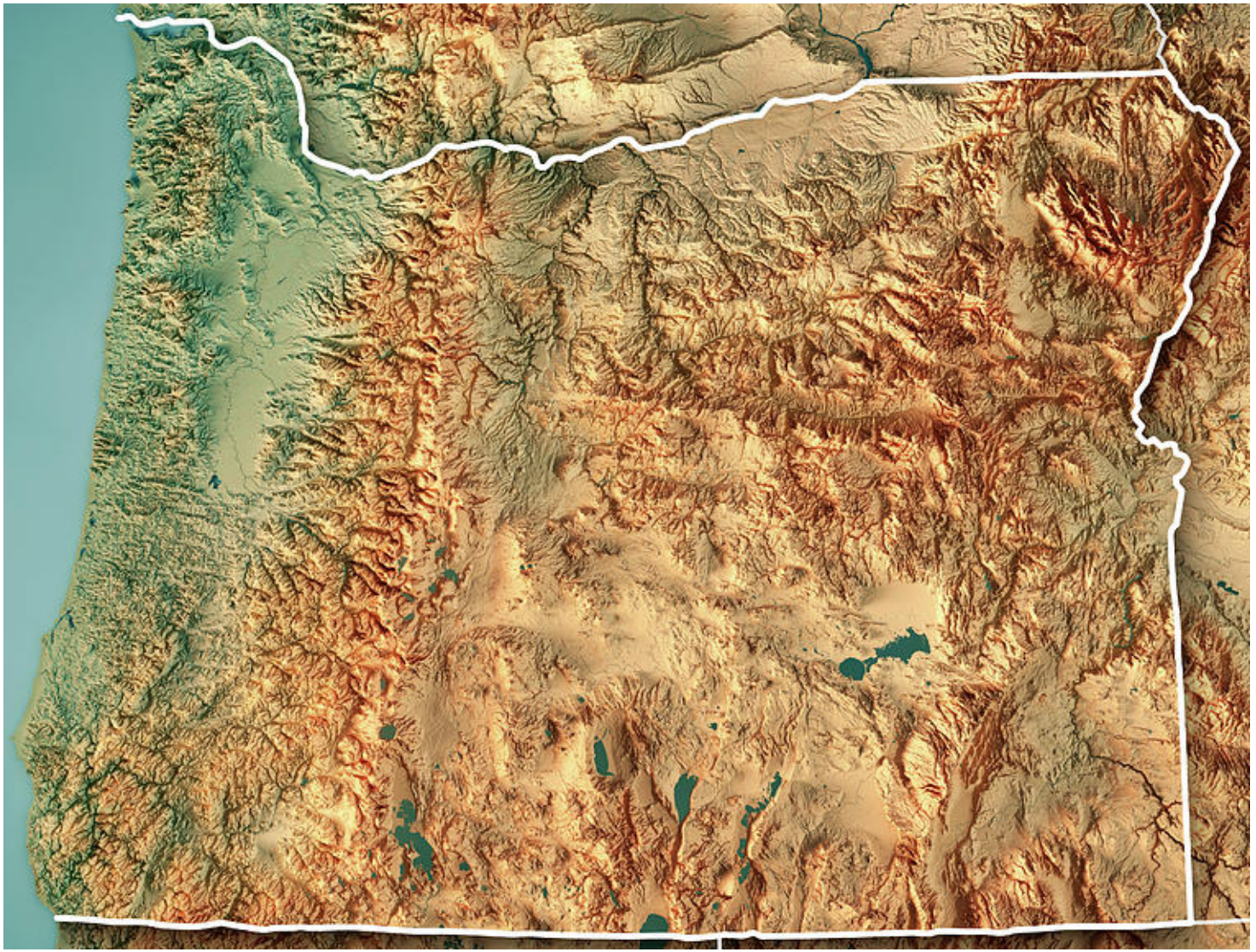
Generative Adversarial Networks

- Send noise into Generator.
- Send generated image and real images into Discriminator.
- Play mini-max game teaching the Generator to fool the Discriminator.
- When fake image can adaquately fool (a well trained) discriminator we produce quality images.
- In practice we train D, then G then repeat.

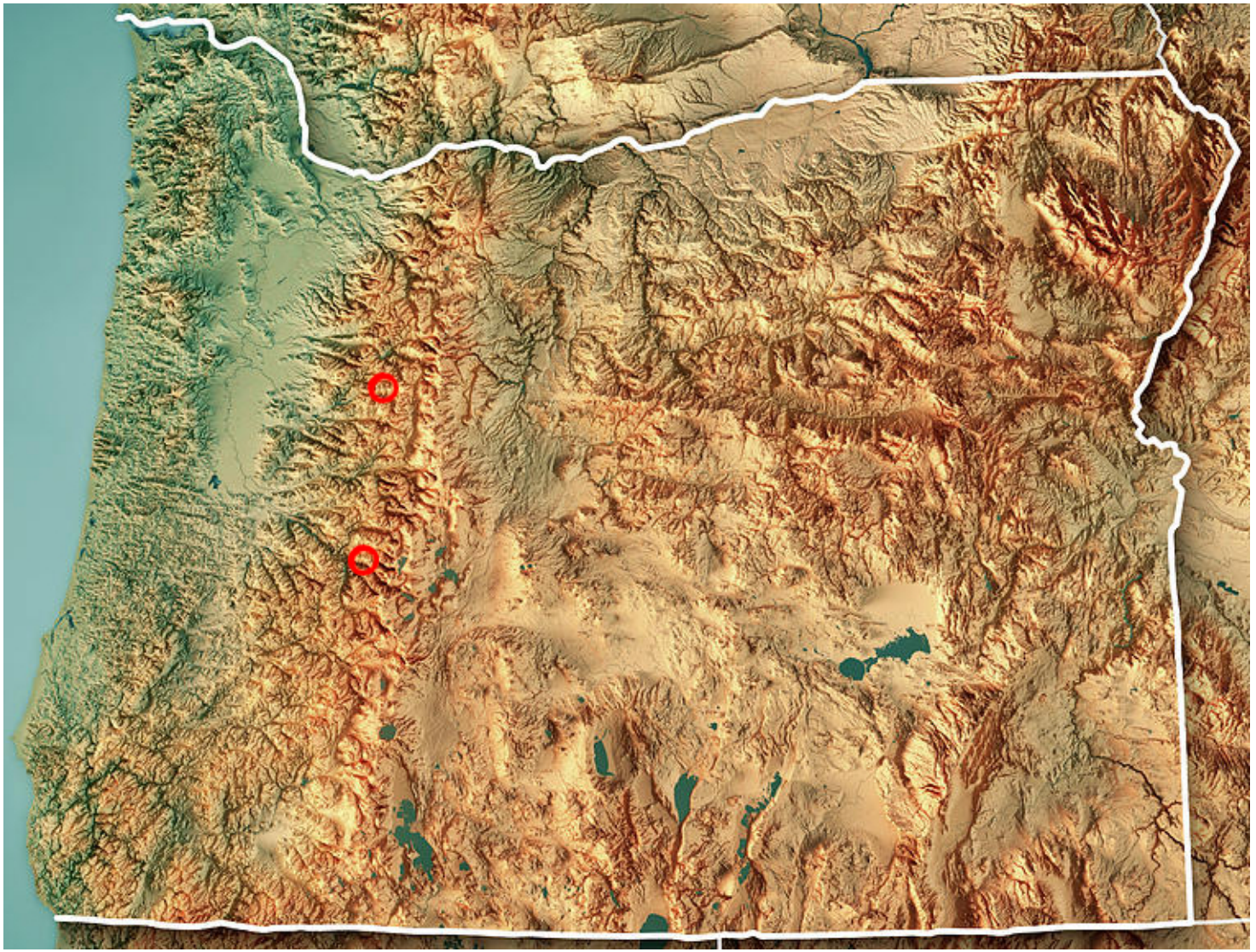
But Why Is This Implicit?

- We didn't learn a density function.
- We just focus on generating samples.
- Don't have an (easy) smooth transition between latent variables.
- Likelihood is not tractable.
- "We're training them to memorize, not generalize" - Ian Goodfellow

Latent Space

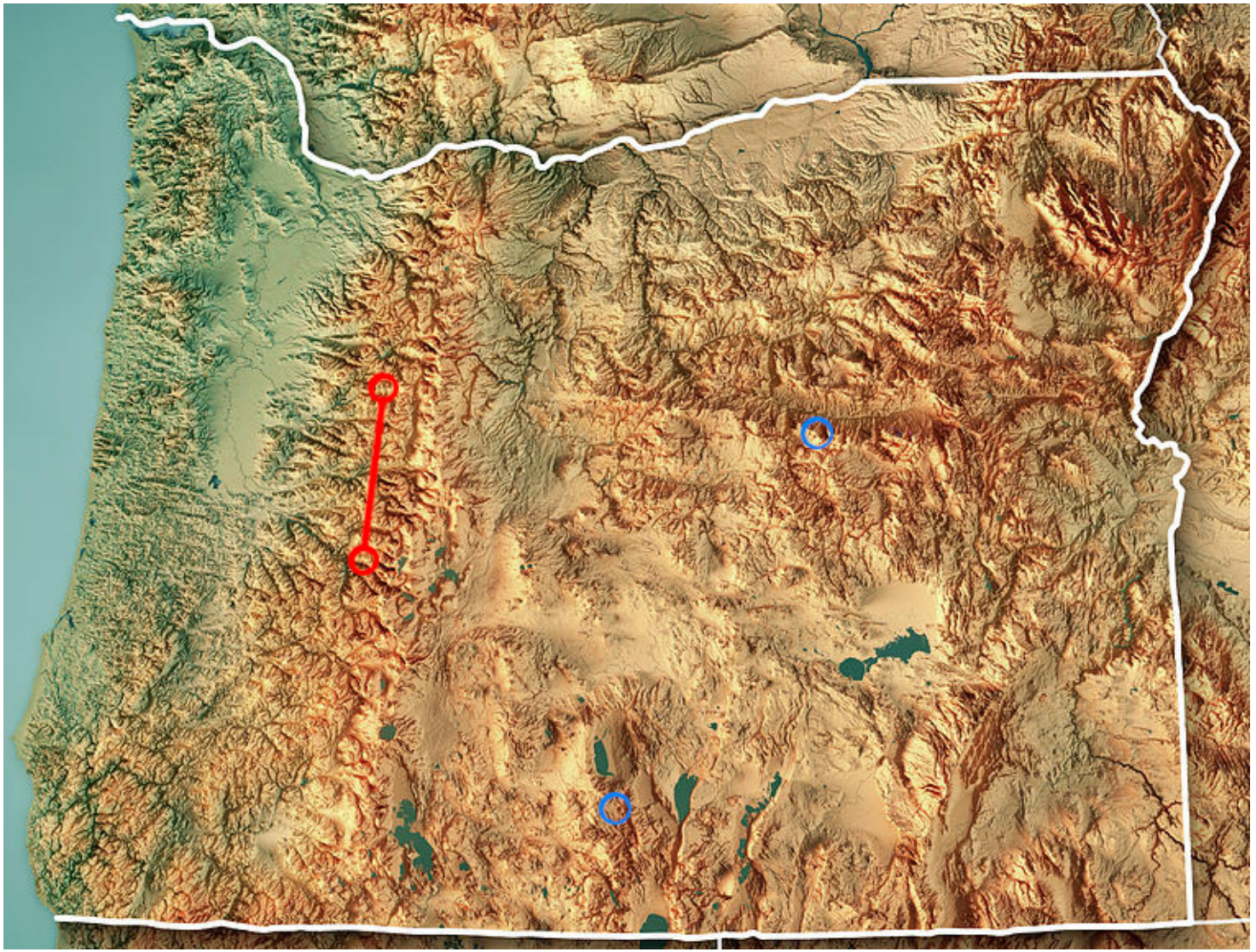


Latent Space

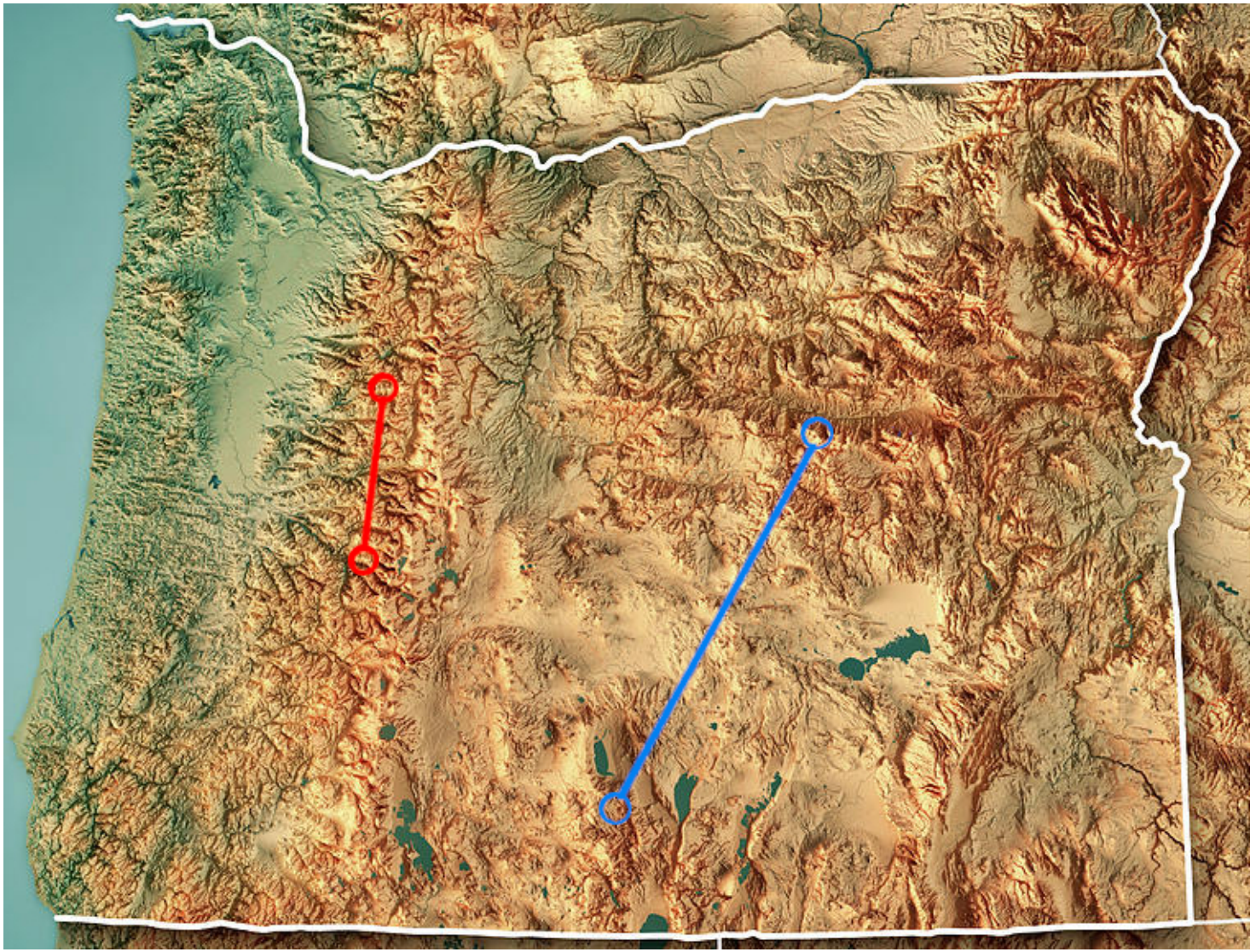


Latent Space

Latent Space



Latent Space



Latent Space

TSNE visualization of vanilla GAN latent space.

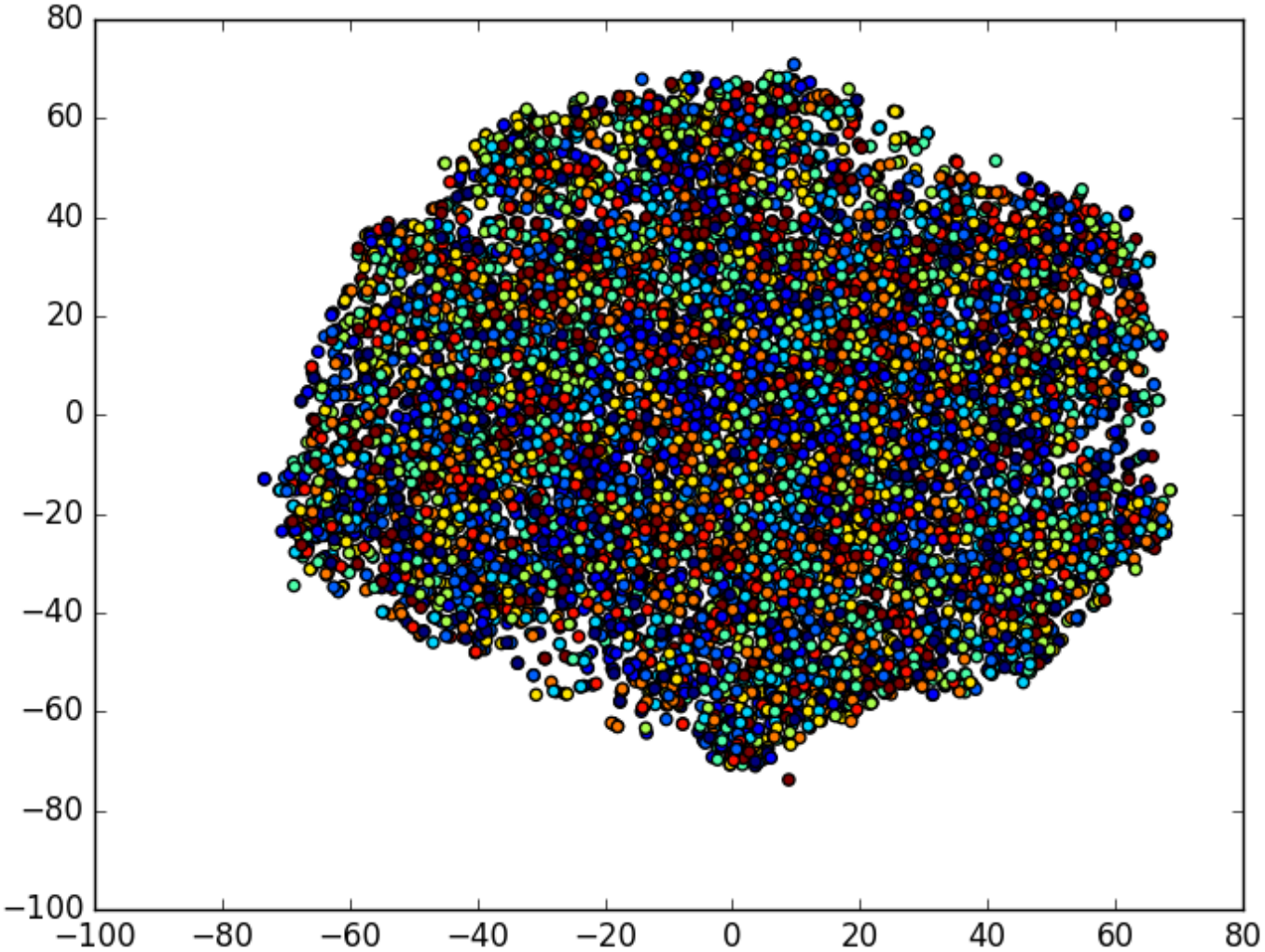


Image from ClusterGAN paper which works on resolving this issue

So Why Use GANs?

- Because they are quick.
- They produce high quality images.
- Can still do many tasks.
- They're well researched.

StyleGAN2

Let's look at a modern GAN and how it works.

Problems With StyleGAN

- GANs typically have “blobs” that are mysteriously produced.
- Often get “GAN Monsters” where parts of images get wildly distorted.
- Certain characteristics are too common. Eyes forward, smile alignment.

Blobs and More Blobs

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

Monsters

(a) Low PPL scores

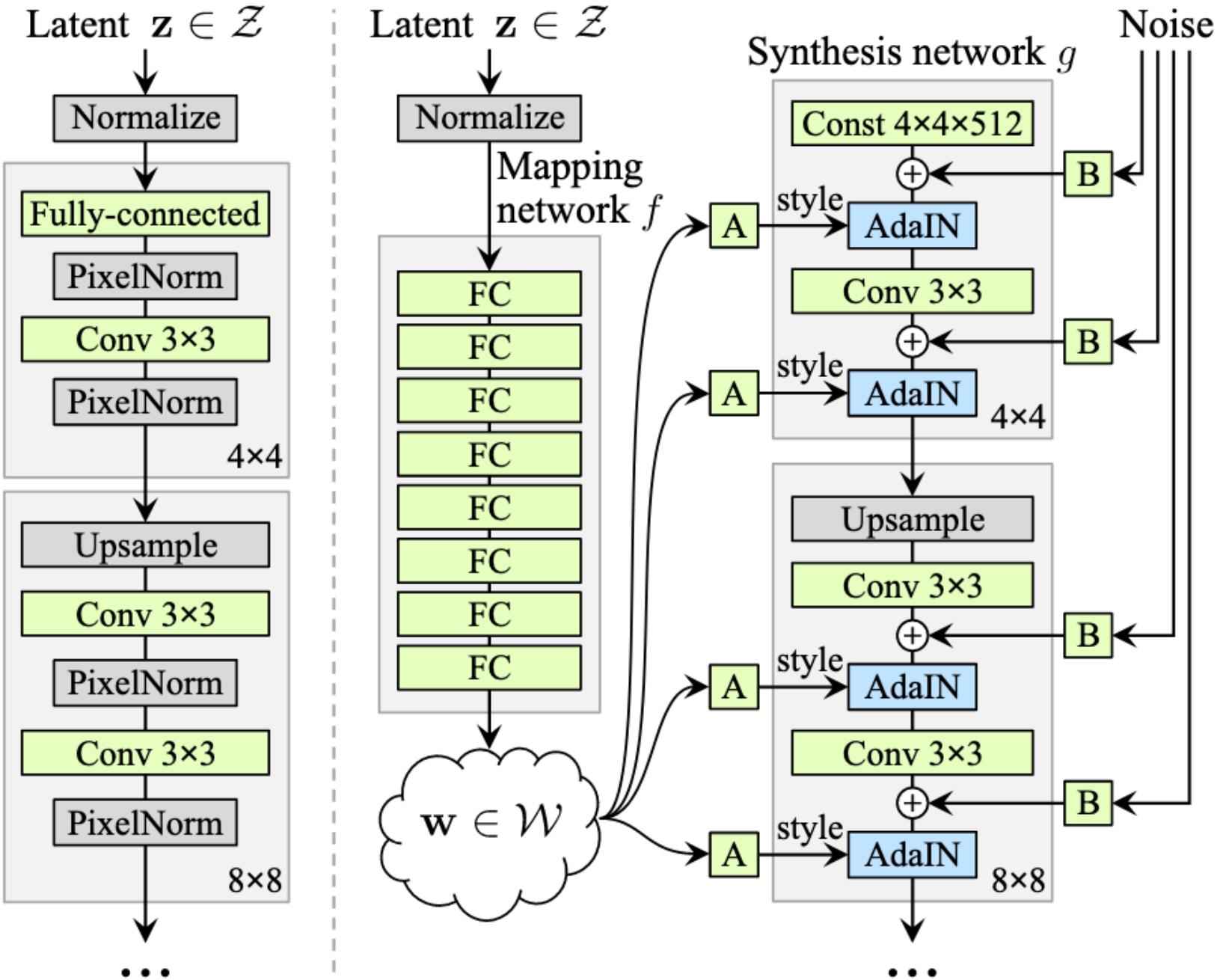
(b) High PPL scores

Figure 4. Connection between perceptual path length and image quality using baseline StyleGAN (config A) with LSUN CAT. (a) Random examples with low PPL ($\leq 10^{\text{th}}$ percentile). (b) Examples with high PPL ($\geq 90^{\text{th}}$ percentile). There is a clear correlation between PPL scores and semantic consistency of the images.

Phase Characteristics

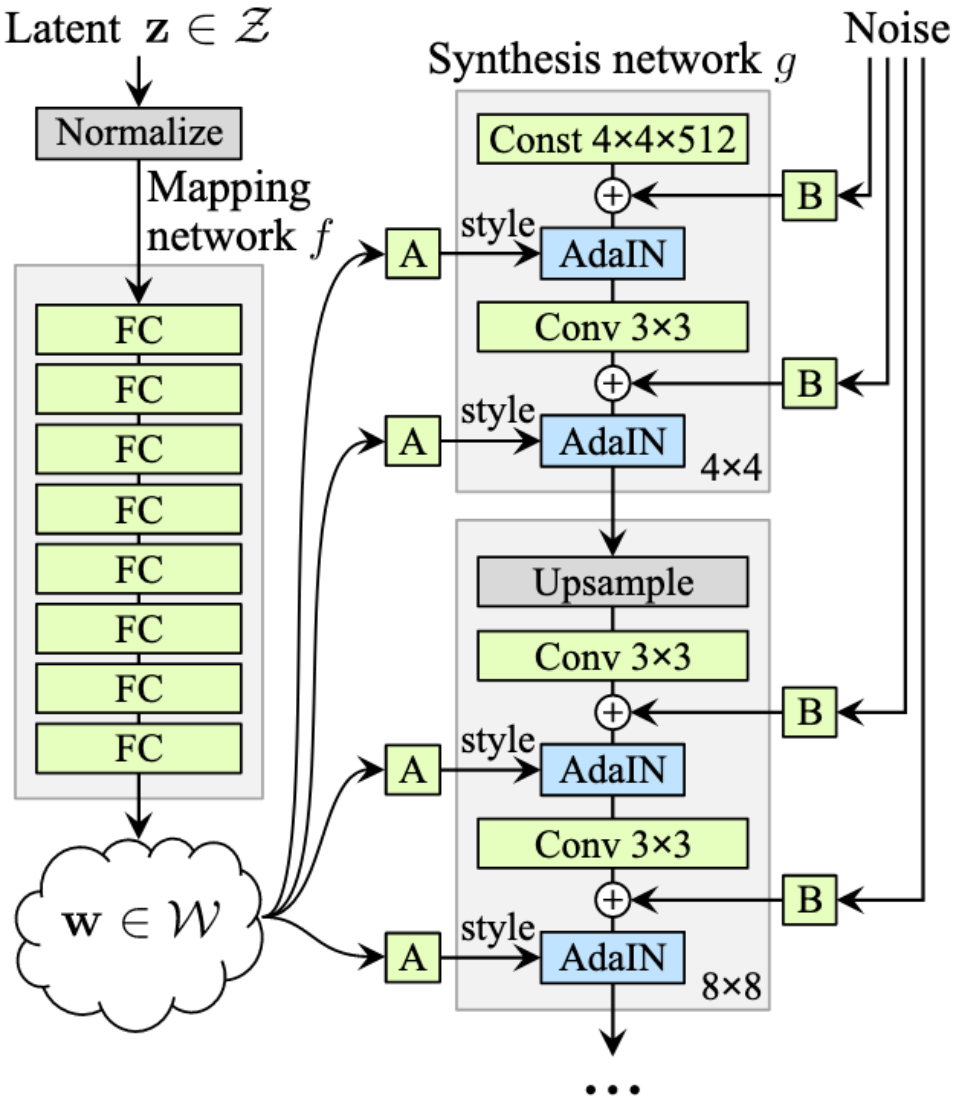
Figure 6. Progressive growing leads to "phase" artifacts. In this example the teeth do not follow the pose but stay aligned to the camera, as indicated by the blue line.

StyleGAN (OG)

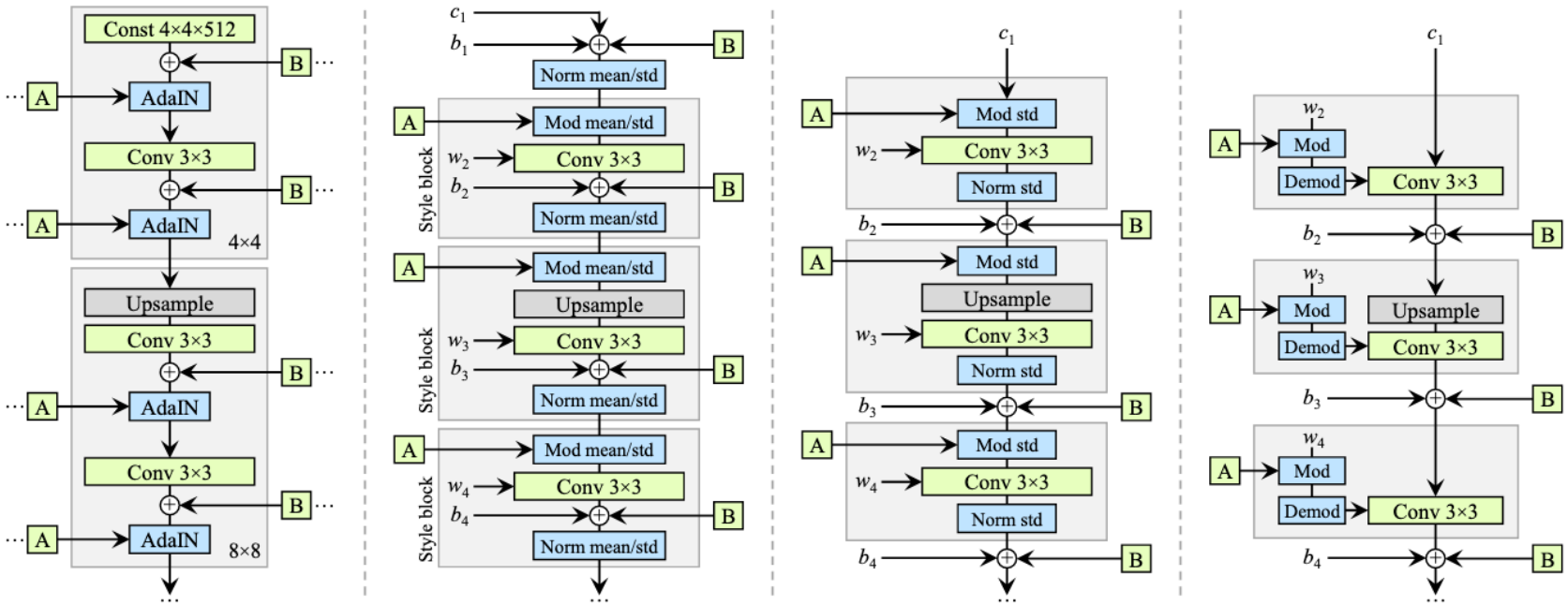

Latent $\mathbf{z} \in \mathcal{Z}$
Normalize
Fully-connected
PixelNorm
Conv 3×3
PixelNorm
4×4
Upsample
Conv 3×3
PixelNorm
Conv 3×3
PixelNorm
8×8
...
Latent $\mathbf{z} \in \mathcal{Z}$
Normalize
Mapping network f
FC
FC
FC
FC
FC
FC
FC
FC
$\mathbf{w} \in \mathcal{W}$
Synthesis network g
Noise
Const 4×4×512
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
4×4
Upsample
Conv 3×3
B
style
A
AdaIN
Conv 3×3
B
style
A
AdaIN
8×8
...

StyleGAN (OG)

- Uses upscaling convolutions, doubling.
- Learns a "style space" (w)
- Introduces Adaptive Instance Normalization (AdaIN)
  $\text{AdaIN}(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$



StyleGAN vs StyleGAN 2



(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

StyleGAN2: What Changed

- Removed AdaIN. Removing this removes the droplet/blobs.
- Lazy Regularization: Split loss and regularization terms and regularize every 16 minibatches.
- Noticed low Perceptual Path Length (PPL) results had better images so introduced a new regularizer to account for this.
- Use skip generator and residual discriminator instead of progressive growing.
- Increase network size (helps get from 512x512 to 1024x1024).

StyleGAN2 Results

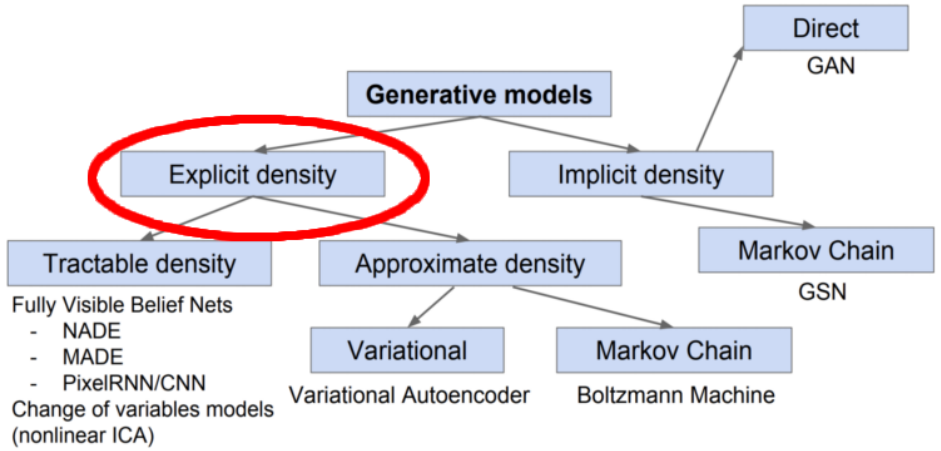
Dataset	Resolution	StyleGAN (A) FID	StyleGAN (A) PPL	StyleGAN2 (F) FID	StyleGAN2 (F) PPL
LSUN CAR	512×384	3.27	1485	**2.32**	**416**
LSUN CAT	256×256	8.53	924	**6.93**	**439**
LSUN CHURCH	256×256	4.21	742	**3.86**	**342**
LSUN HORSE	256×256	3.83	1405	**3.43**	**338**

Table 3. Improvement in LSUN datasets measured using FID and PPL. We trained CAR for 57M images, CAT for 88M, CHURCH for 48M, and HORSE for 100M images.

StyleGAN2 Results

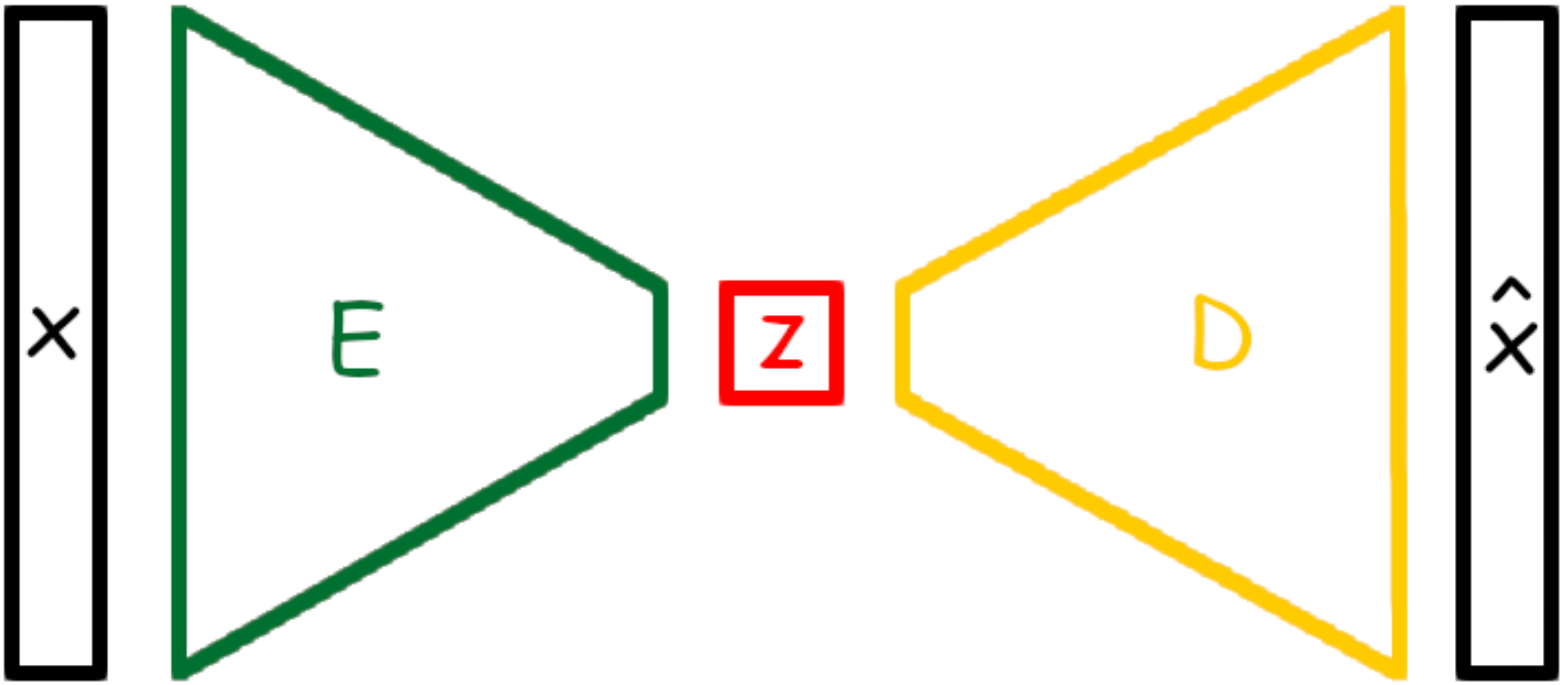
Taxonomy of Generators: Explicit Density

- We want to learn a distribution of data.
- We assume some prior about the distribution of the data.
- We learn the log-liklihood of the function $\log p_\theta(x)$



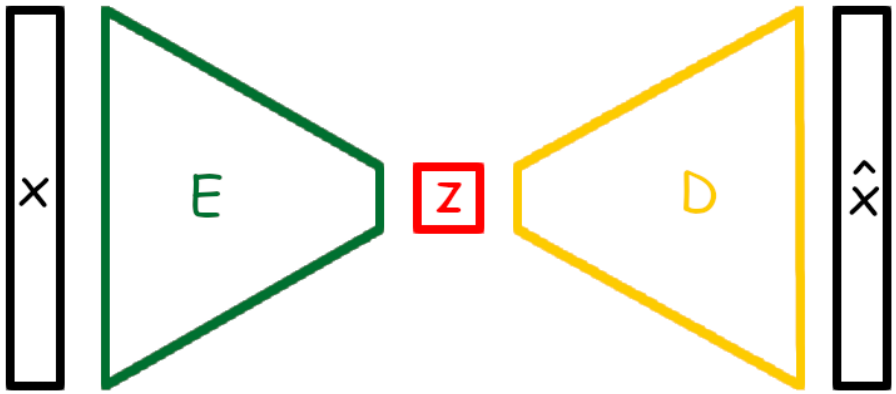
source: ian goodfellow

AutoEncoders



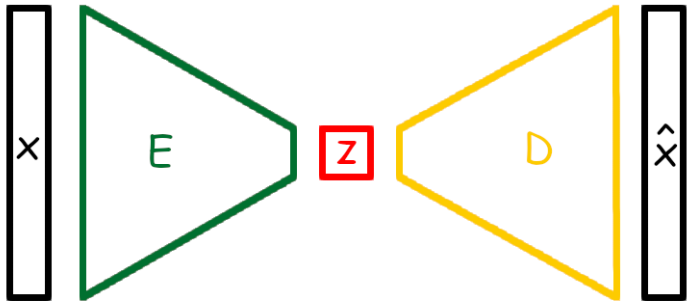
AutoEncoders

- Want to learn latent variable z from x. $\mathbb{R}^n \mapsto \mathbb{R}^m$ where $m < n$
- Compressed representation of the data.
- Since we don't have training data for z, we construct a decoder to do unsupervised learning.



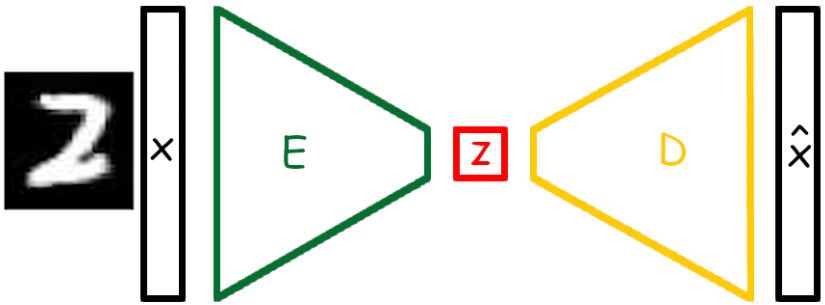
AutoEncoders

- Want to take x and produce $\hat{x}$



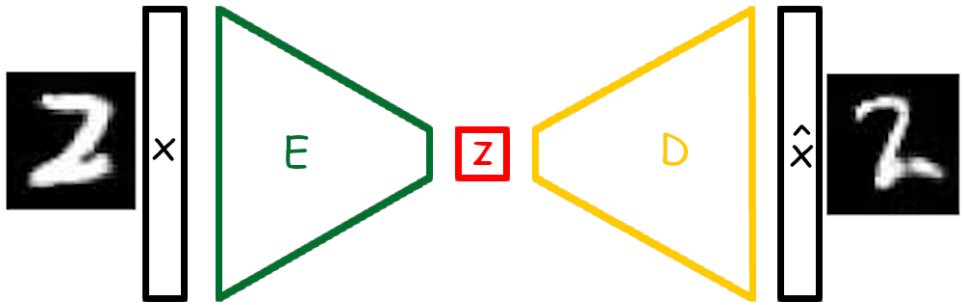
AutoEncoders

- Want to take x and produce $\hat{x}$
- Sample x from dataset and encode to latent space z.



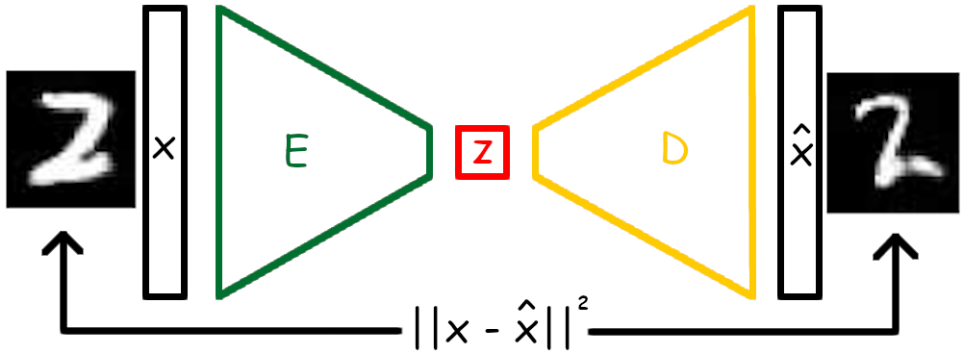
AutoEncoders

- Want to take x and produce $\hat{x}$
- Sample x from dataset and encode to latent space z.
- Decode to sample from z and generate $\hat{x}$



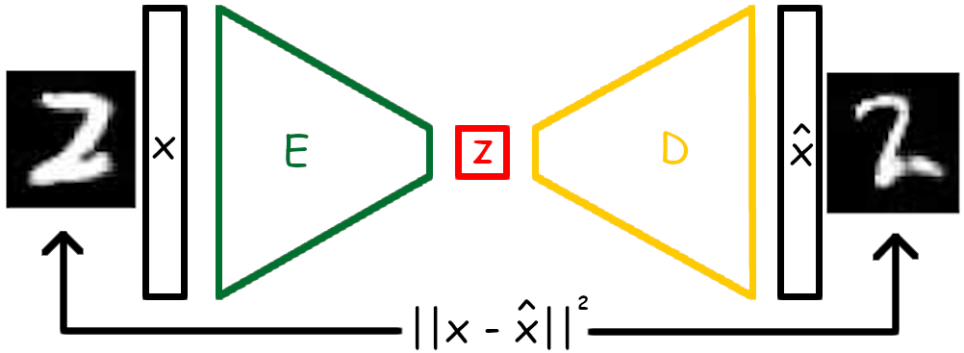
AutoEncoders

- Want to take x and produce $\hat{x}$
- Sample x from dataset and encode to latent space z.
- Decode to sample from z and generate $\hat{x}$
- Regularize $||x - \hat{x}||^2$



AutoEncoders

- Want to take x and produce $\hat{x}$
- Sample x from dataset and encode to latent space z.
- Decode to sample from z and generate $\hat{x}$
- Regularize $||x - \hat{x}||^2$
- Gives us a unsupervised training.



Pick Our Latent Space Size Carefully

Dimensionality of latent space → reconstruction quality

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

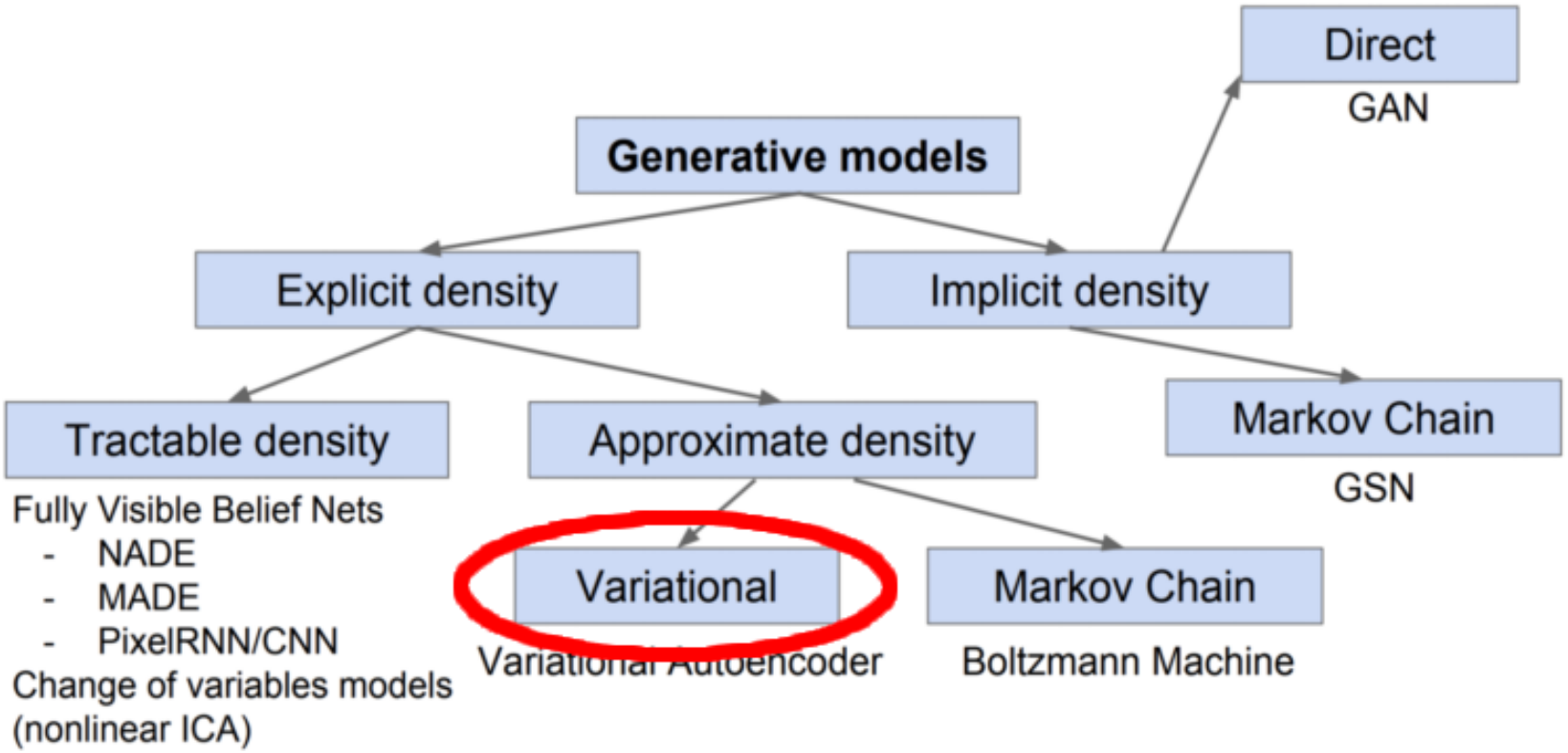
5D latent space

Ground Truth

The BIG Problem!

We can't generate new and unique samples!

Taxonomy of Generators



source: ian goodfellow

Variational Autoencoder (VAE)

- We need a way to generate **new** samples that aren't seen in the dataset.
- Need to add some variance and noise (make non-determanistic).
- Still have to be able to do back propagation.

VAE: Breaking Down the Problem

- Suppose there is some hidden (latent) variable z that produces observation x, $p(z|x)$. **We want to learn this.**

VAE: Breaking Down the Problem

- Suppose there is some hidden (latent) variable z that produces observation x, $p(z|x)$. **We want to learn this.**
- $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$

VAE: Breaking Down the Problem

- Suppose there is some hidden (latent) variable z that produces observation x, $p(z|x)$. **We want to learn this.**
- $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$
- $p(x) = \int p(x|z)p(z)dz$ (intractable!)

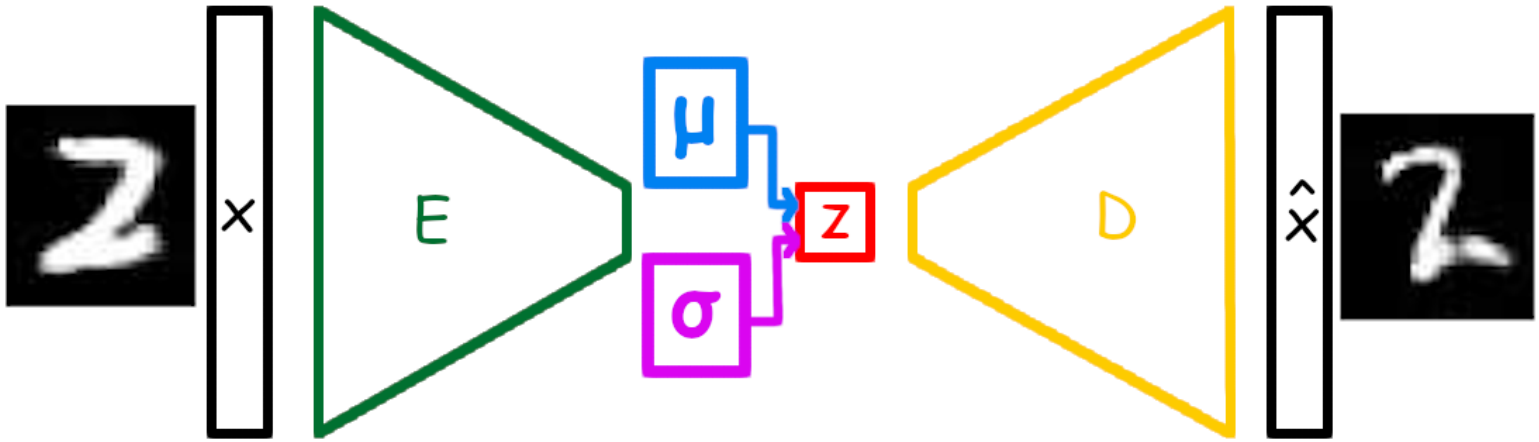
VAE: Breaking Down the Problem

- Suppose there is some hidden (latent) variable z that produces observation x, $p(z|x)$. **We want to learn this.**
- $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$
- $p(x) = \int p(x|z)p(z)dz$ (intractable!)
- We can approximate $p(z|x)$ with another distribution $q(z|x)$ that is tractable.

VAE: Breaking Down the Problem

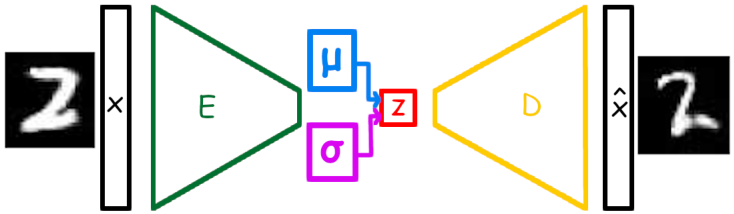
- Suppose there is some hidden (latent) variable z that produces observation x, $p(z|x)$. **We want to learn this.**
- $p(z|x) = \frac{p(x|z)p(z)}{p(x)}$
- $p(x) = \int p(x|z)p(z)dz$ (intractable!)
- We can approximate $p(z|x)$ with another distribution $q(z|x)$ that is tractable.
- We'll then need to measure the difference between p and q.

Variational Autoencoder (VAE)



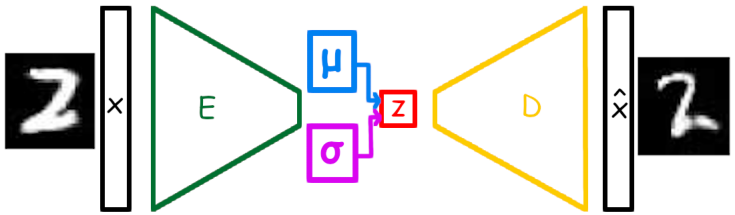
Variational Autoencoder (VAE)

- Encoder computes $p_\phi(z|x)$
  (reconstruction loss, like autoencoder)



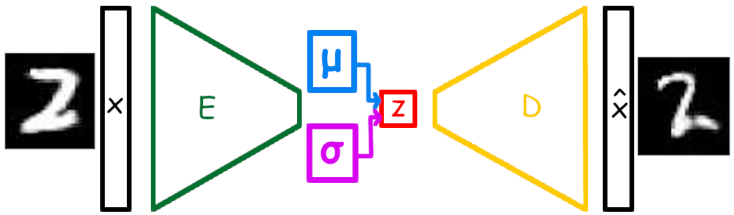
Variational Autoencoder (VAE)

- Encoder computes $p_\phi(z|x)$
  (reconstruction loss, like autoencoder)
- Decoder computes $q_\theta(x|z)$
  (regularization term)



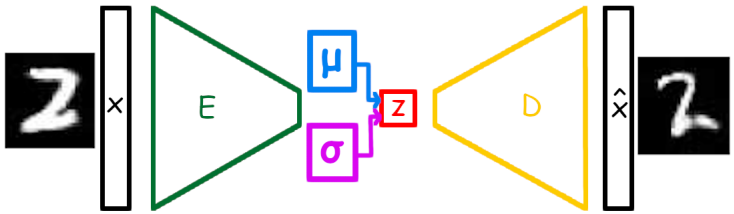
Variational Autoencoder (VAE)

- Encoder computes $p_\phi(z|x)$
  (reconstruction loss, like autoencoder)
- Decoder computes $q_\theta(x|z)$
  (regularization term)
- $L(\phi, \theta, x) = ||x - \hat{x}||^2 + KL(p_\phi(z|x)||p(x))$



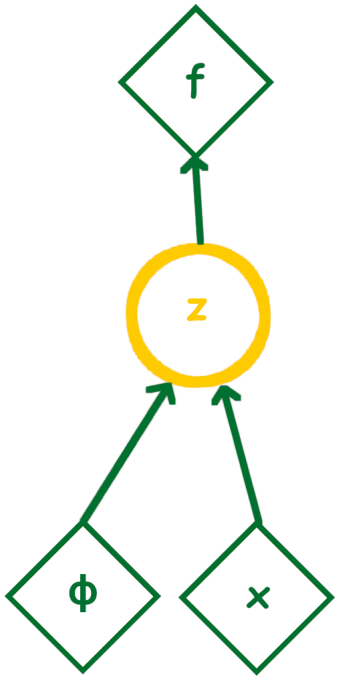
(VAE) Regularization Term

- $L(\phi, \theta, x) = ||x - \hat{x}||^2 + KL(p_\phi(z|x)||p(x))$
- We are using the KL-Divergence between the *inferred latent distribution* ($p_\phi(z|x)$) and the *fixed prior on latent distribution* ($p(z)$).
- What prior? Why not Gaussian? $p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$
- Encourages encoding to distribute evenly around latent space and penalizes when memorizing.
- $KL(p_\phi(z|x)||p(z)) = -\frac{1}{2}\sum(\sigma_j + \mu_j^2 - 1 - \log \sigma_j)$



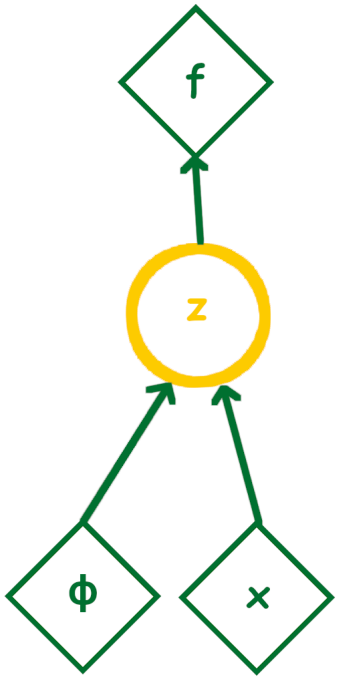
(VAE) Reparameterization

- Ops! We can't backprop through this! z is a stochastic node.



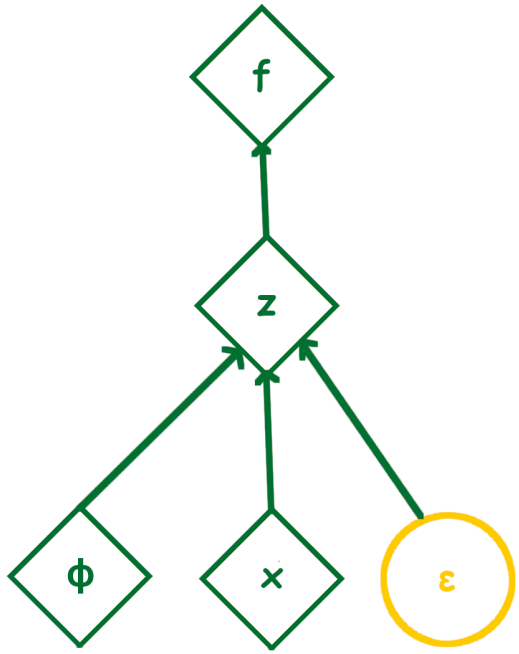
(VAE) Reparameterization

- Ops! We can't backprop through this! z is a stochastic node.
- We can reparameratize.



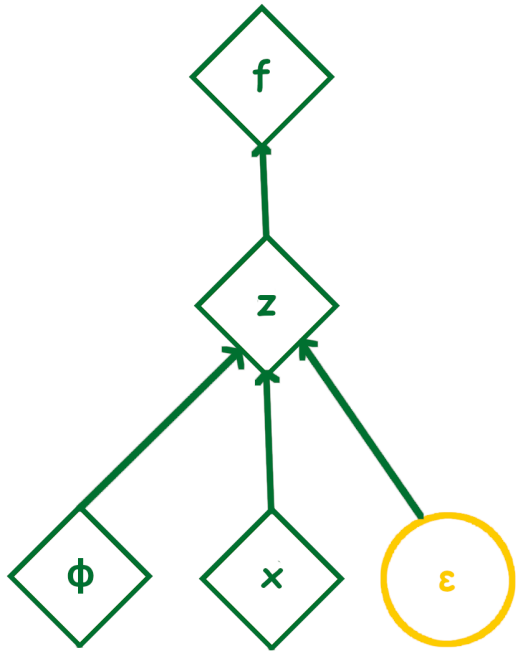
(VAE) Reparameterization

- Ops! We can't backprop through this! z is a stochastic node.
- We can reparameratize.
- $z = \mu + \sigma \odot \varepsilon$
- We let μ and σ be fixed vectors and $\varepsilon \sim \mathcal{N}(0, 1)$



(VAE) Reparameterization

- Ops! We can't backprop through this! z is a stochastic node.
- We can reparameratize.
- $z = \mu + \sigma \odot \varepsilon$
- We let μ and σ be fixed vectors and $\varepsilon \sim \mathcal{N}(0, 1)$
- Now we can backprop through ϕ and x.
- Now we can adjust ε and create new images similar to the ones we learned from.



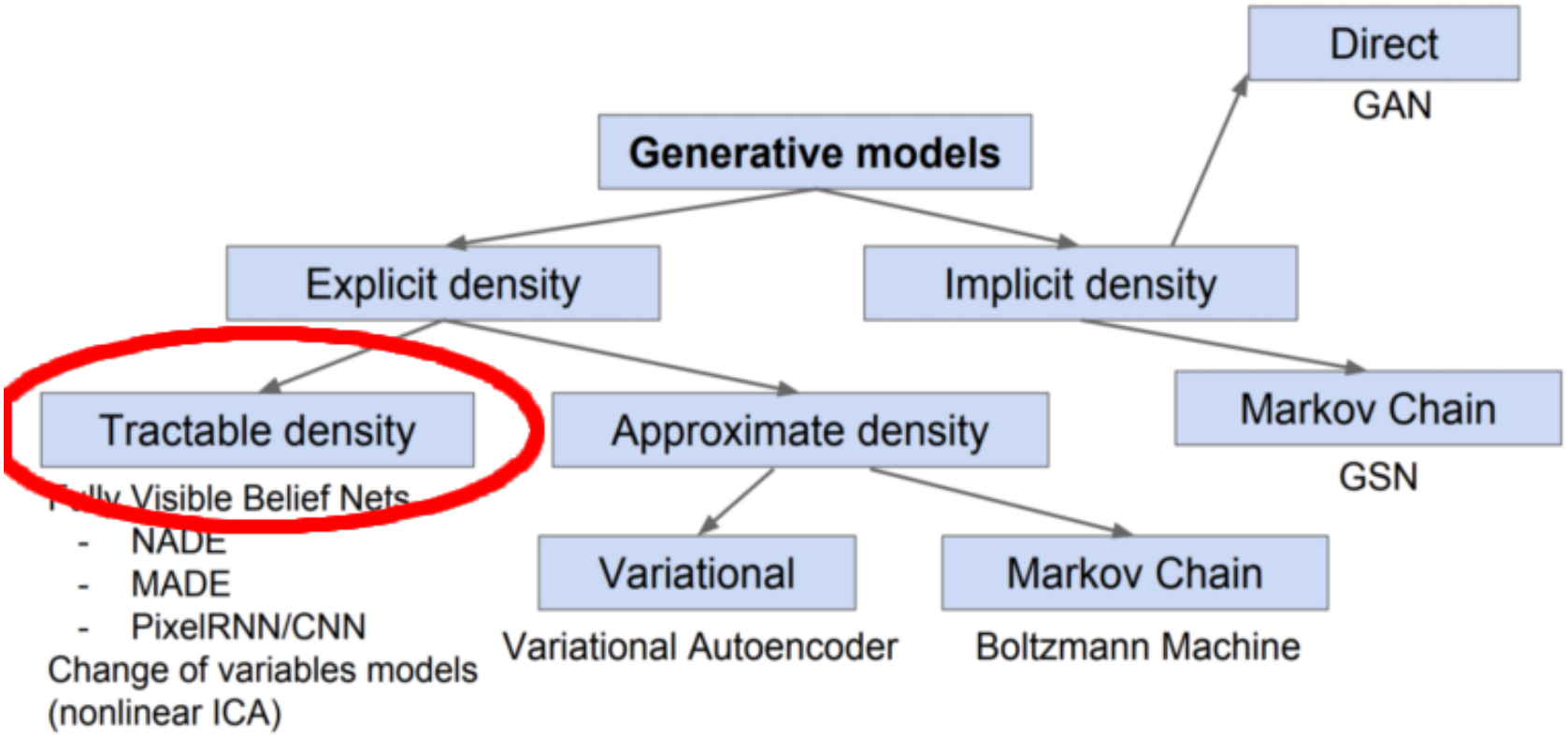
Approximate Density Estimation

- We didn't actually learn $p(x)$ but rather $q(x)$.
- We approximated the density of $p(x)$ with a tractable distribution.
- KL Divergence is always ≥ 0.

Good and Bad of a VAE

- They are very easy to train.
- Often not high quality images.
- You learn an approximate density function (have log-liklihood)

Taxonomy of Generators



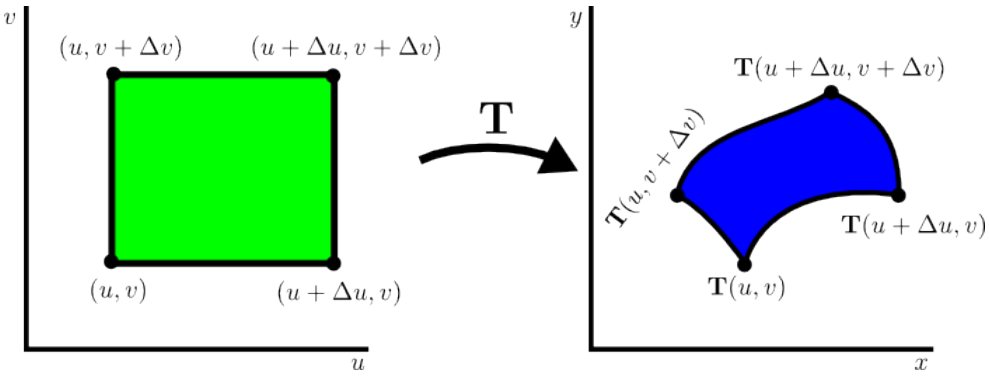
source: ian goodfellow

Density Estimation

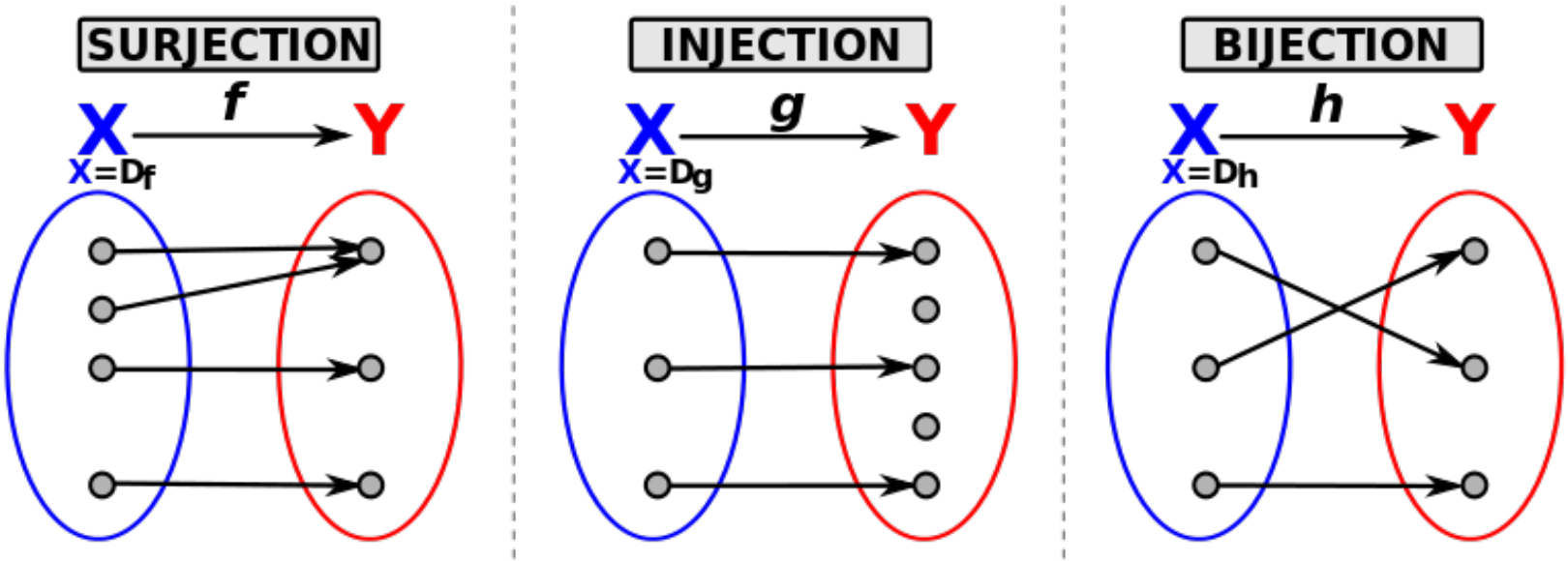
- We've come close to learning density, but not exactly.
- Density is **very** difficult to learn.
- Having true density *should* mean better ability to perform our downstream tasks (generation, density estimation, latent variable inference, etc)

Change Of Variables

- What if we just map to a different space?
- We'll convert one probability distribution to another tractable distribution (such as a Gaussian)



Bijection (One-to-One and On-To)



Change Of Variables

- $\int p(x)dx = \int q(z)dz = 1$ (Definition of PDF)

Change Of Variables

- $\int p(x)dx = \int q(z)dz = 1$ (Definition of PDF)
- $p(x) = q(z)\left|\frac{dz}{dx}\right|$

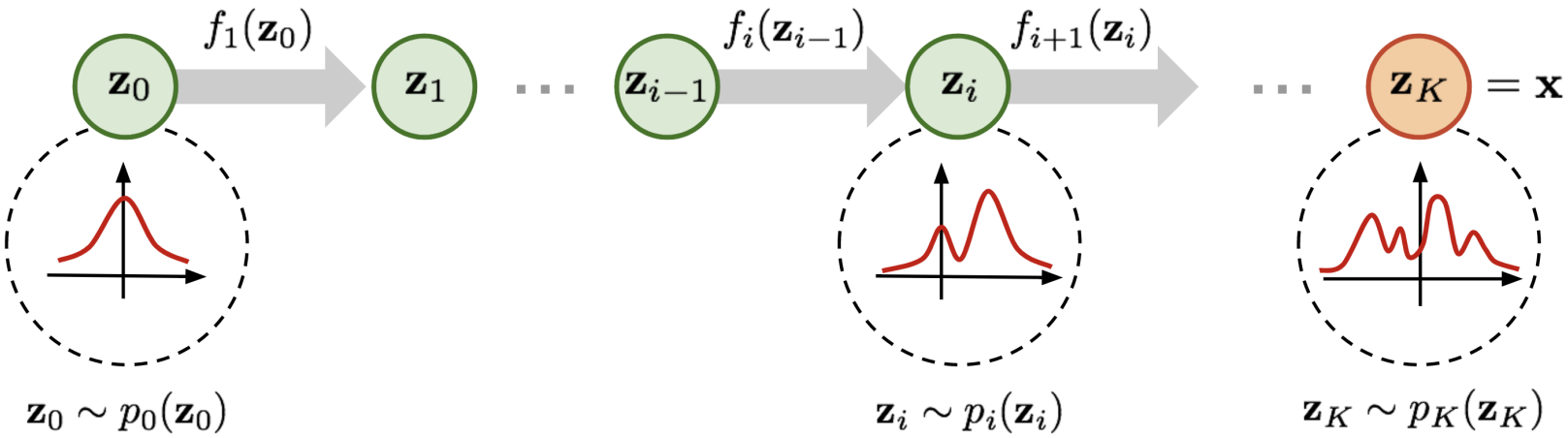
Change Of Variables

- $\int p(x)dx = \int q(z)dz = 1$ (Definition of PDF)
- $p(x) = q(z)\left|\frac{dz}{dx}\right|$
- $= q(f^{-1}(x))\left|\frac{df^{-1}(x)}{dx}\right|$

Change Of Variables

- $\int p(x)dx = \int q(z)dz = 1$ (Definition of PDF)
- $p(x) = q(z)\left|\frac{dz}{dx}\right|$
- $= q(f^{-1}(x))\left|\frac{df^{-1}(x)}{dx}\right|$
- $p(x) = q(f^{-1}(x))\left|det\,\frac{df^{-1}(x)}{dx}\right|$ (using the Jacobian determinant)

Normalizing Flow



Source: lilianweng.github.io

Normalizing Flow

- We need our function $f(x)$ to be easily invertible
- The Jacobian determinant needs to be *easy* to compute.
- BUT we get the exact log-likelihood of $p(x)$ (tractable) and we can train our model on the NLL
  $\mathcal{L}(D) = -\frac{1}{|D|} \sum_{x \in D} log\, p(x)$

NICE: Non-linear Independent Components Estimation

- Key Idea: What if split the the transformation into two blocks? $x \mapsto (x_0, x_1)$

NICE: Non-linear Independent Components Estimation

- Key Idea: What if split the the transformation into two blocks? $x \mapsto (x_0, x_1)$
- Use an easy to invert equation.

$f(x)$	$f^{-1}(y)$
$y_0 = x_0$ $y_1 = x_1 + m(x_0)$	$x_0 = y_0$ $x_1 = y_1 - m(y_0)$

NICE: Non-linear Independent Components Estimation

- Key Idea: What if split the the transformation into two blocks? $x \mapsto (x_0, x_1)$
- Use an easy to invert equation.

$f(x)$	$f^{-1}(y)$
$y_0 = x_0$ $y_1 = x_1 + m(x_0)$	$x_0 = y_0$ $x_1 = y_1 - m(y_0)$

- Jacobian is trivial and triangular.

NICE: Non-linear Independent Components Estimation

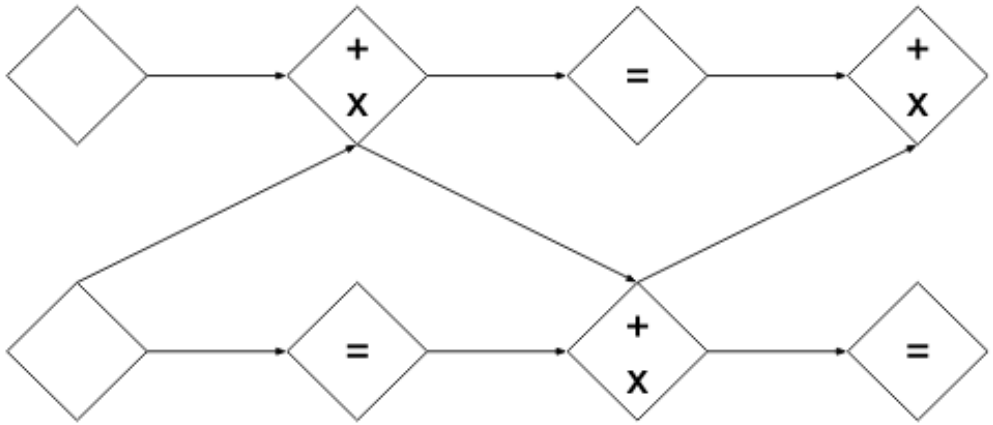
- Key Idea: What if split the the transformation into two blocks? $x \mapsto (x_0, x_1)$
- Use an easy to invert equation.

$f(x)$	$f^{-1}(y)$
$y_0 = x_0$ $y_1 = x_1 + m(x_0)$	$x_0 = y_0$ $x_1 = y_1 - m(y_0)$

- Jacobian is trivial and triangular.
- 2nd Key Idea: Permute x split:
  $x_0 \mapsto (x_{0,0}, x_{0,1}), x_1 \mapsto (x_{1,1}, x_{1,0}), x_2 \mapsto \ldots$

Throwing It Together

- Sample from a tractable distribution $z \sim f(x)$.
- Make $m(x)$ a neural net.
- Use at least 3 layers (permuting x).
- Add a scaling term at the last layer $e^s \odot f(x)$.



NICE: Results

Why Is This Better/Worse?

- We've learned a richer family of priors compared to VAE.
- More difficult to train because we need invertible functions.
- Able to perform complex/accurate downstream tasks because we've better approximated the distribution. (e.g. Inpainting, conceptual compression)

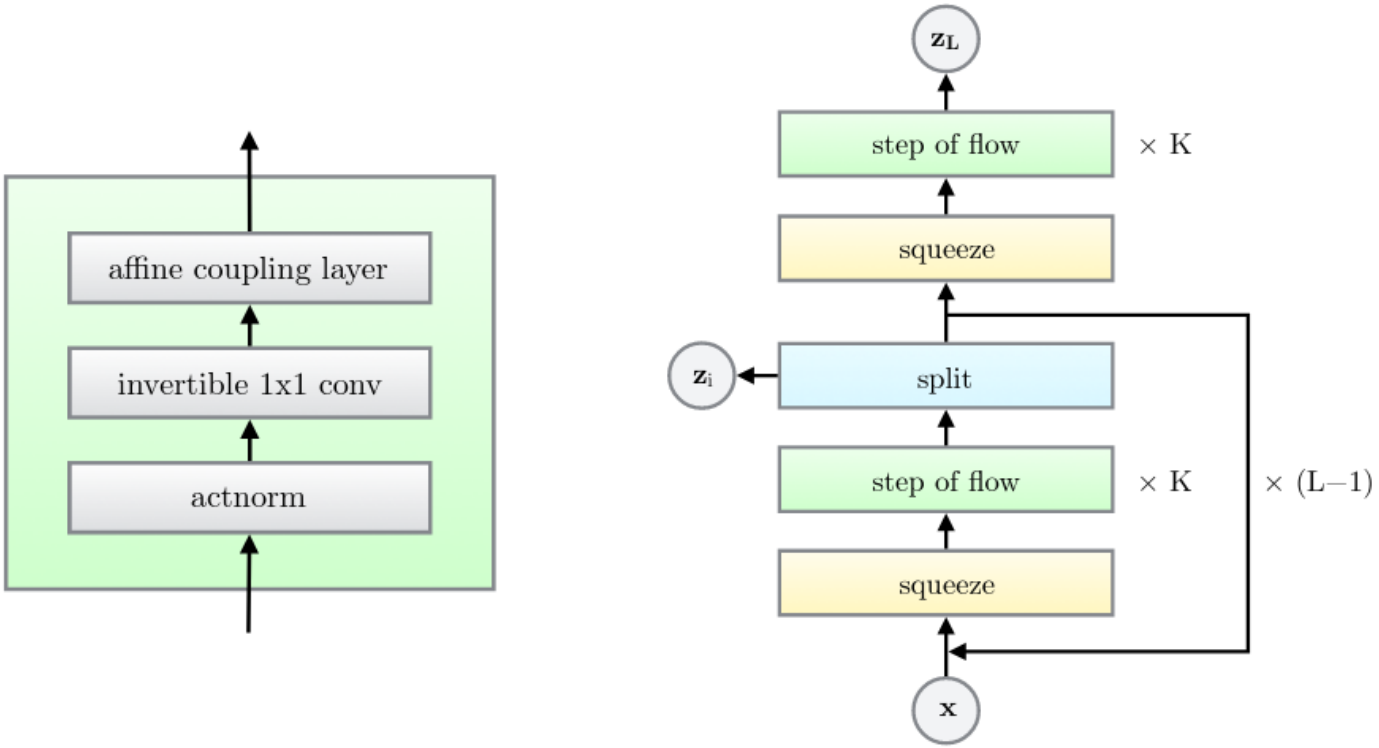
GLOW

- NICE (and RealNVP) do not produce sharp images.
- NICE could not compete with common GAN tasks like face generation.
- Need a new model to get sharp features and produce high quality images.

GLOW: What Changed?

- Add an Actnorm layer.
- Add an invertible convolution layer to learn checkerboarding.
- Make it bigger!

GLOW: What Changed?



(a) One step of our flow.

(b) Multi-scale architecture (Dinh et al., 2016).

Figure 2: We propose a generative flow where each step (left) consists of an *actnorm* step, followed by an invertible 1×1 convolution, followed by an affine transformation (Dinh et al., 2014). This flow is combined with a multi-scale architecture (right). See Section 3 and Table 1.

GLOW: What Changed?

Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\texttt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i, j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert \det(\mathbf{W}) \rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

GLOW: Training

- Set $K = 32$ and $L = 6$ for $(32 * 6 + 32)$ 224 flow steps (NICE had 5!)
- New type of layers (NICE only had the additive coupling)
- Can train on CelebA-HQ (256x256) images.

GLOW: Results

Table 2: Best results in bits per dimension of our model compared to RealNVP.

Model	CIFAR-10	ImageNet 32x32	ImageNet 64x64	LSUN (bedroom)	LSUN (tower)	LSUN (church outdoor)
RealNVP	3.49	4.28	3.98	2.72	2.81	3.08
Glow	**3.35**	**4.09**	**3.81**	**2.38**	**2.46**	**2.67**

Figure 4: Random samples from the model, with temperature 0.7

GLOW vs StyleGAN vs StyleGAN2

GLOW

StyleGAN

StyleGAN2

GLOW Downstream Tasks: Interpolation

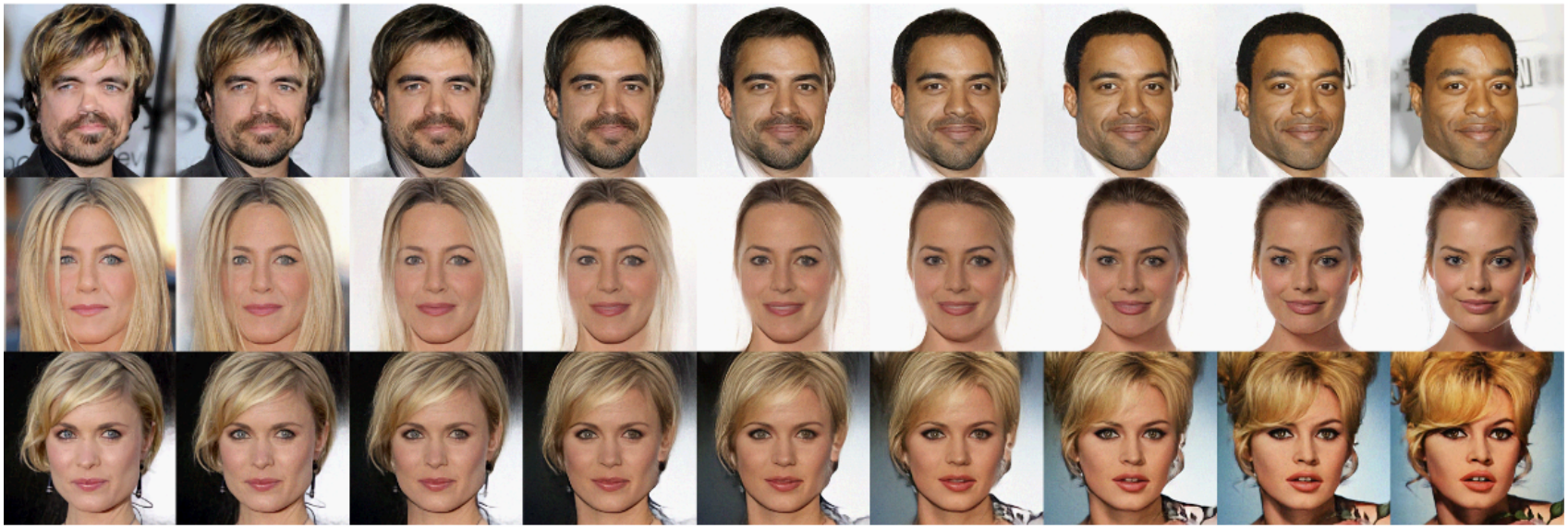


Figure 5: Linear interpolation in latent space between real images

GLOW Downstream Tasks: Feature Manipulation

(a) Smiling

(b) Pale Skin

(c) Blond Hair

(d) Narrow Eyes

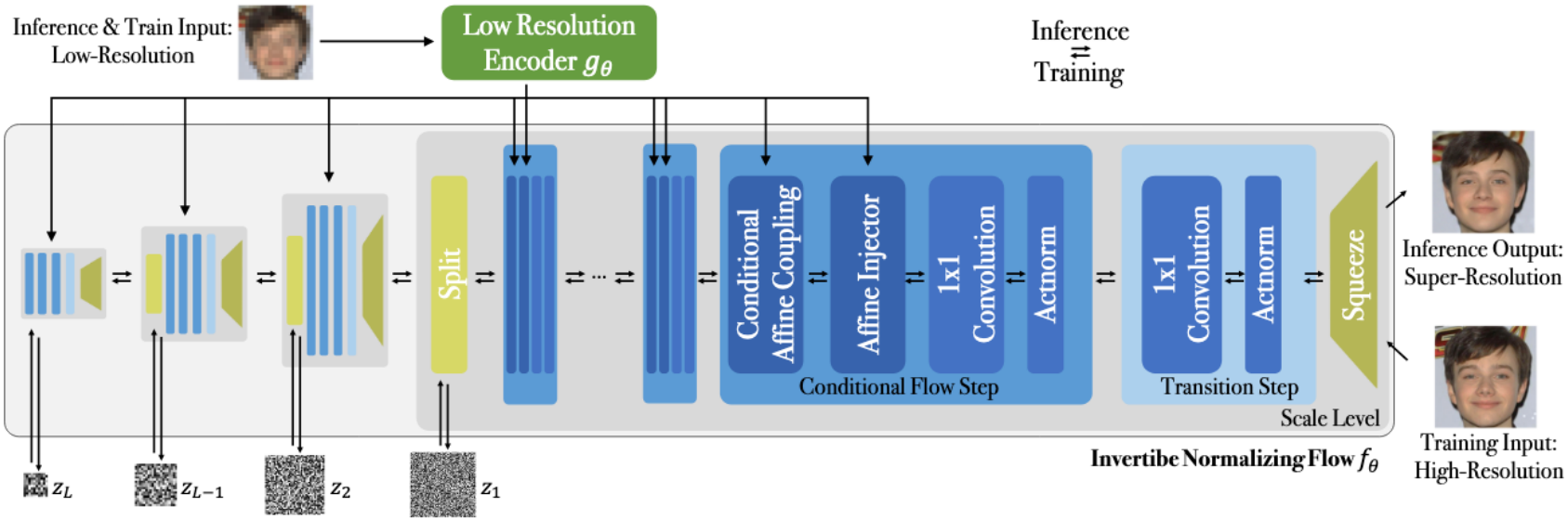
(e) Young

(f) Male

What About Using Flows for Super Resolution?

- Use flows as a replacement for GAN super resolution.
- Training within certain scale uses a normalizing flow.
- When scaling up and down use injective and surjective functions.
- Output is a distribution instead of a single image.

SRFlow

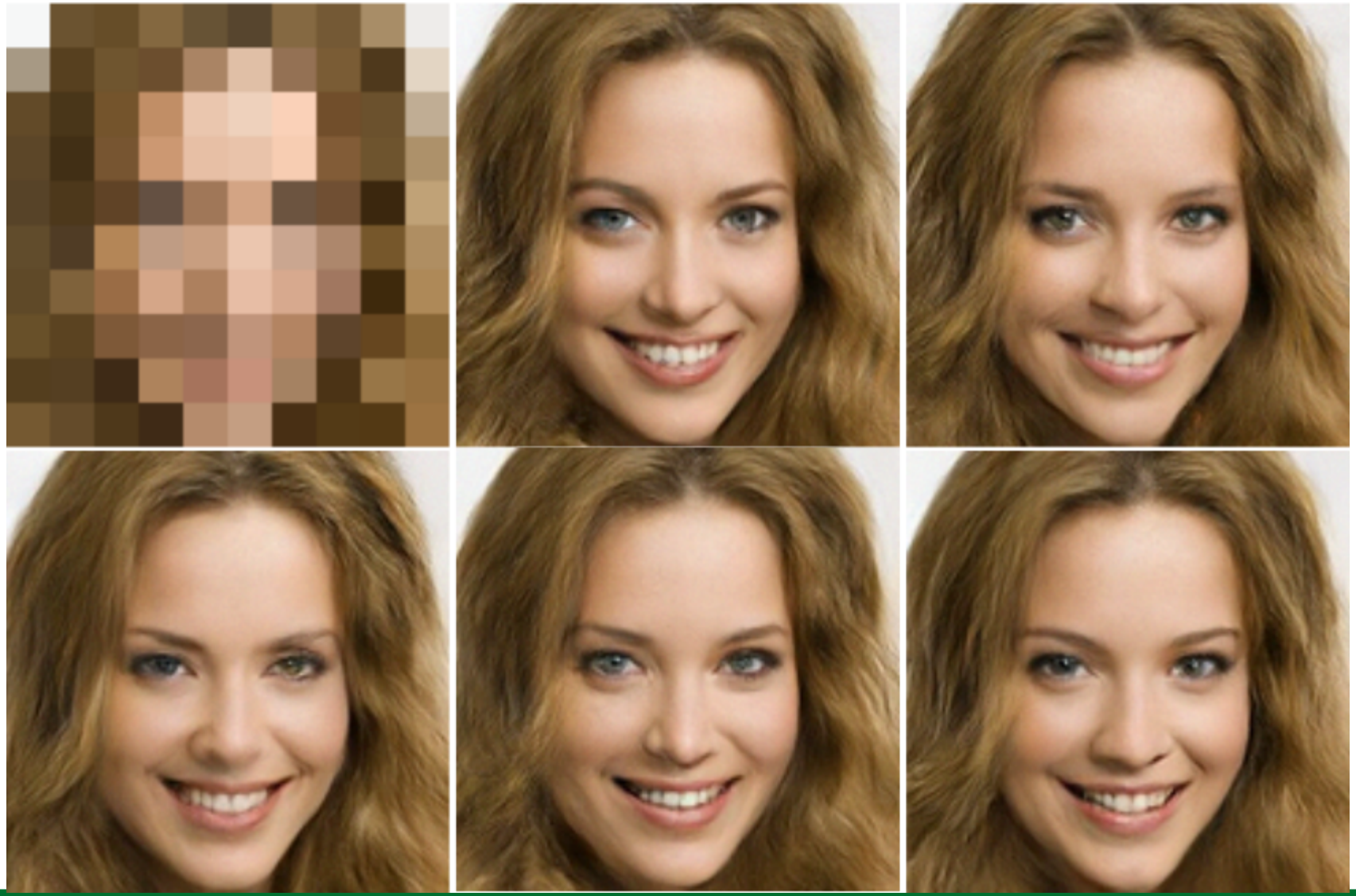

Inference & Train Input:
Low-Resolution
Low Resolution
Encoder g_θ
Inference
⇄
Training
Split
Conditional
Affine Coupling
Affine Injector
1x1
Convolution
Actnorm
Conditional Flow Step
1x1
Convolution
Actnorm
Transition Step
Squeeze
Scale Level
Inference Output:
Super-Resolution
Training Input:
High-Resolution
z_L
z_{L-1}
z_2
z_1
Invertibe Normalizing Flow f_θ

SRFlow: Comparison

Fig. 7. Comparison of our SRFlow with state-of-the-art for 8× face SR on CelebA.

SRFlow: Stochastic Output

SRFlow: Stochastic Output



Recap: Generative Models

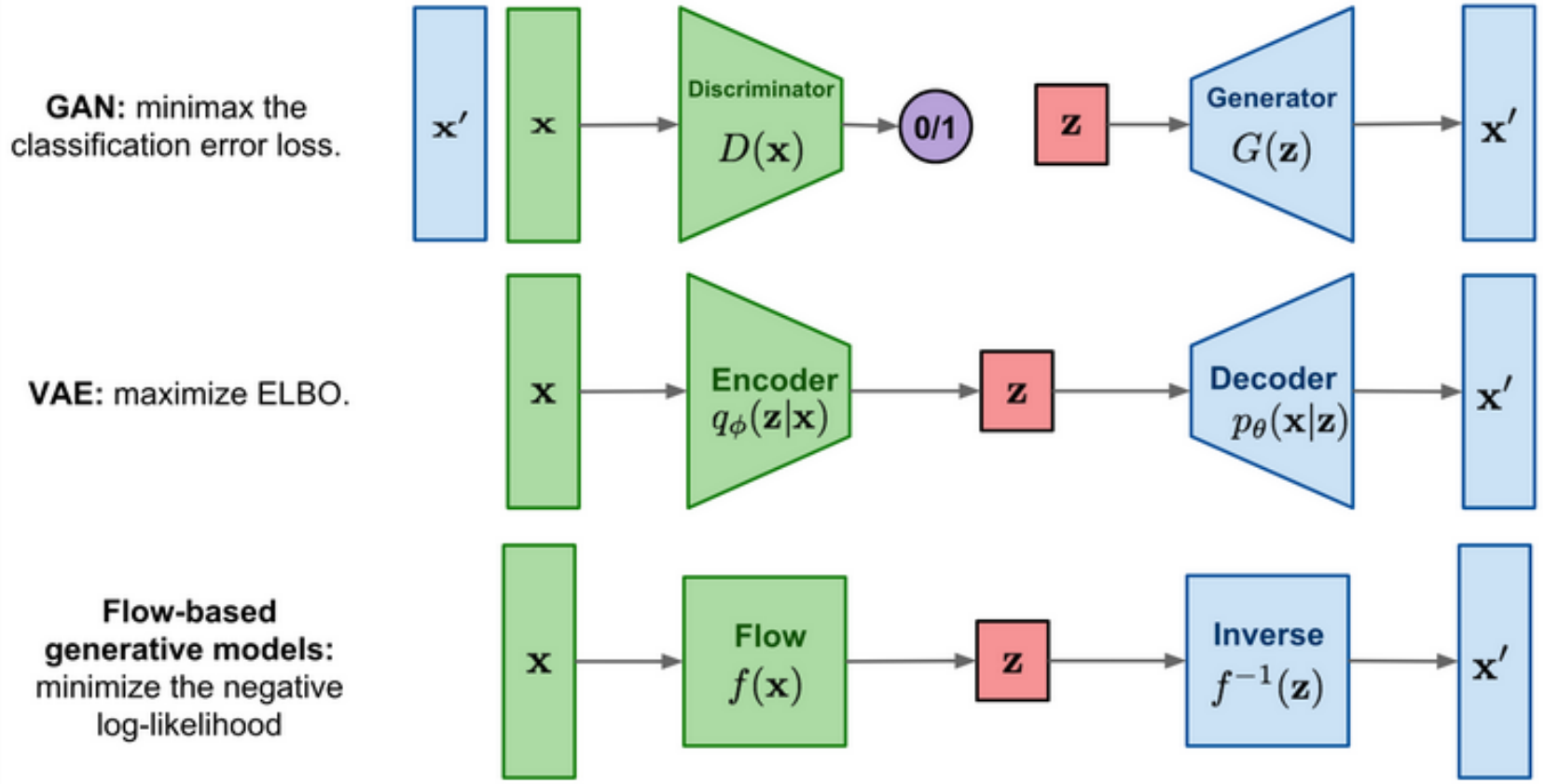


Fig. 1. Comparison of three categories of generative models.