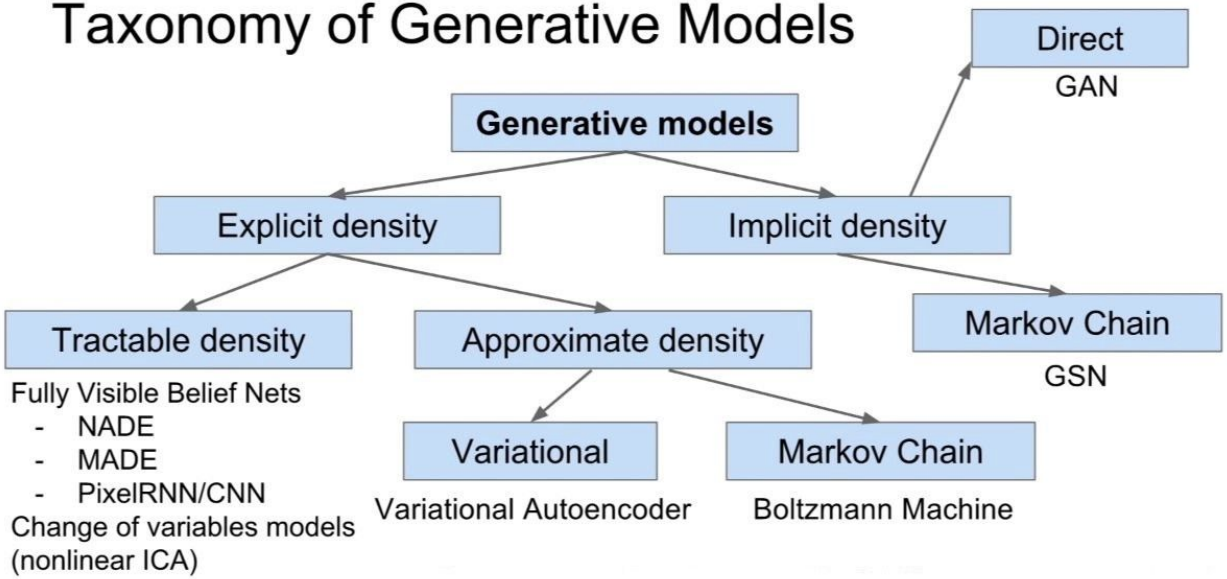
Deep Learning

Lecture 12

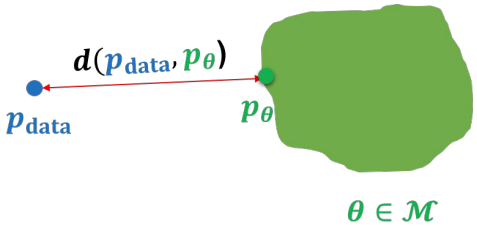
Recap

- Discriminative vs Generative models
- Reparametrization tricks
- Variational Auto Encoder

Taxonomy of Generative Models

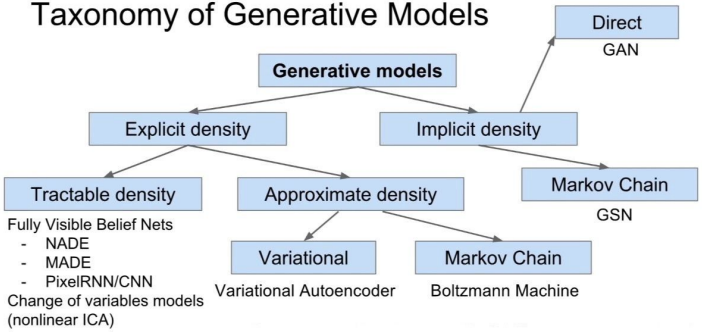


$$\mathbf{x}^{(j)} \sim p_{\text{data}}$$
$$j = 1, 2, \ldots, |\mathcal{D}|$$



Model family

$$\min_{\theta \in \mathcal{M}} d(p_{\text{data}}, p_\theta)$$



- What is the representation for the model family M?
- What is the objective function d(·)?
- What is the optimization procedure for minimizing d(·)?

Density estimation

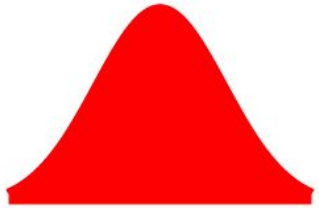
We have a dataset from some distribution p(x). We want to understand its ***nature:***

- Estimate/Approximate the real distribution p(x)
  - VAE (done)
  - Autoregressive models (this lecture)
- Learn to sample from this distribution
  - Generative Adversarial Networks=GAN (This lecture)

Motivation:

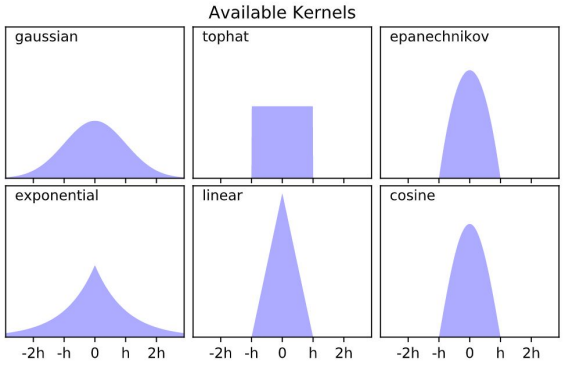
By fitting good data distribution we hope to use this model on downstream inference

Samples from distribution



True distribution

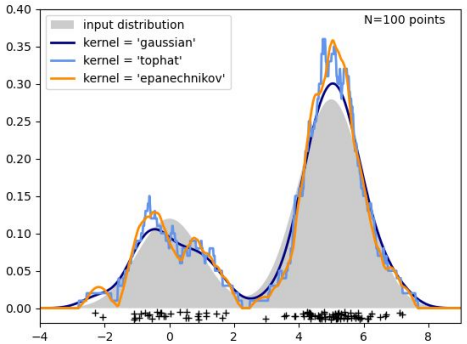
Kernel Density Estimation



Every sample represent some distribution. By merging them we can estimate the real distribution

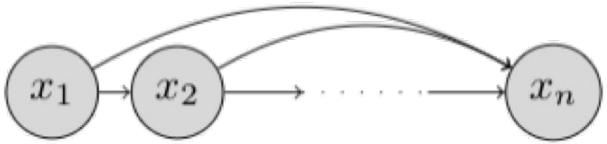
$$p(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

: Nah, I'm too big for you
@Generated cat

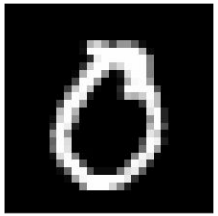


PixelCNN/PixelRNN

The idea is to rewrite full probability using the chain rule



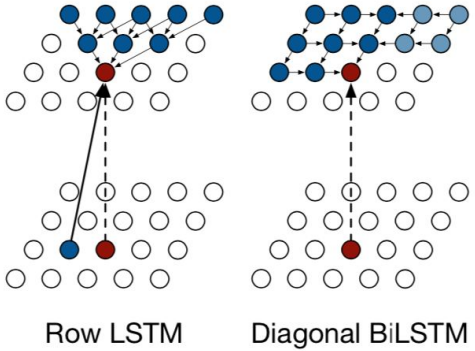
$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$



In this parametrization probability seems to be intractable. Let's parametrize history with some function

$$p(x) = \prod_{i=1}^{n} p(x_i|h = f_\theta(x_{<i-1}), x_{i-1})$$

PixelRNN

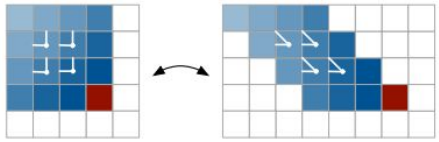


Row LSTM

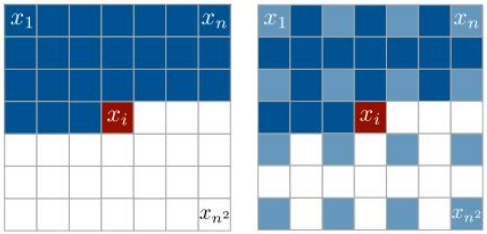
$$h_{i,j} = f\left(h_{i-1,j-1}, h_{i-1,j}, h_{i-1,j+1}, x_{i,j}\right)$$

Diagonal BiLSTM

$$h_{i,j} = f\left(h_{i-1,j}, h_{i,j-1}, x_{i,j}\right)$$

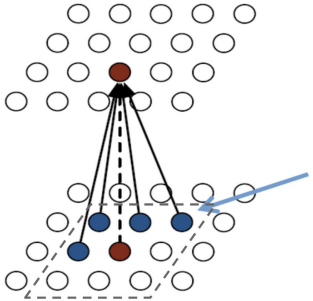


Pixel CNN



Context

Multi-scale context

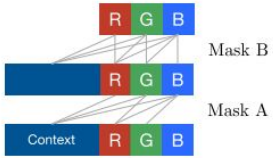


PixelCNN

Masked convolution

1	1	1
1	0	0
0	0	0

How to decompose RGB channels?



PixelCNN

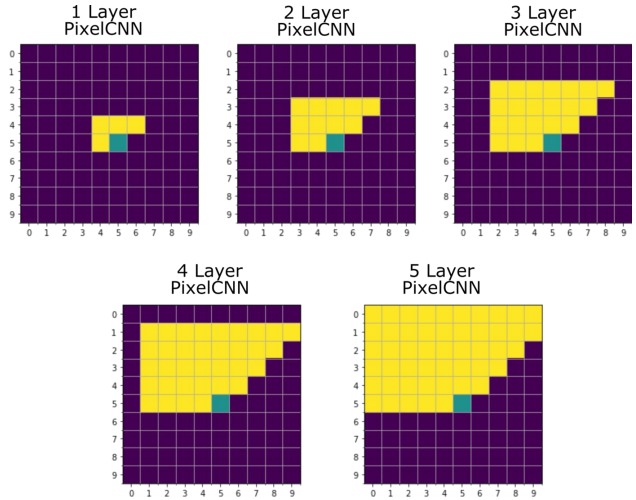


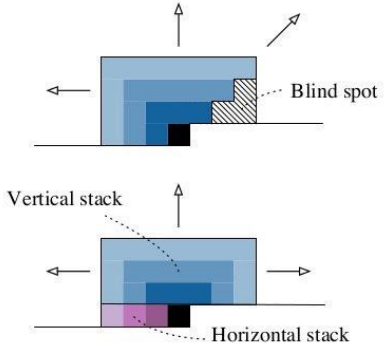
Figure 1. Image completions sampled from a PixelRNN.

PixelCNN

Training

- Take image, reconstruct :)

Inference



0. Set $i = 0$ to represent current iteration and pixel (implicitly we would translate it to the row/col/sub-pixel in the image).
1. Start off with a tensor for your image $\mathbf{x^0}$ with any initialization (it doesn't matter).
2. Feed $\mathbf{x^i}$ into the PixelCNN network to generate distributional outputs $\mathbf{y^{i+1}}$.
3. Randomly sample sub-pixel u_{i+1} from the mixture distribution defined by $\mathbf{y^{i+1}}$ (we only need the subset of values for sub-pixel $i + 1$).
4. Set $\mathbf{x^{i+1}}$ as $\mathbf{x^i}$ but replacing the current sub-pixel with u_{i+1}.
5. Repeat step 2-4 until entire image is generated.

PixelCNN++

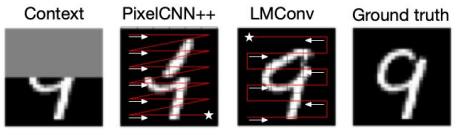
- Output the original PixelCNN is categorical (but pixel have an order)

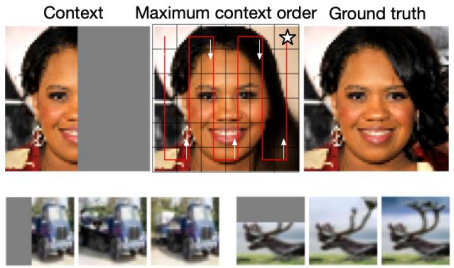
$$P(x|\mu, s) = \begin{cases} \sigma(\frac{x-\mu+0.5}{s}) & \text{for } x = 0 \\ \sigma(\frac{x-\mu+0.5}{s}) - \sigma(\frac{x-\mu-0.5}{s}) & \text{for } 0 < x < 255 \\ 1 - \sigma(\frac{x-\mu-0.5}{s}) & \text{for } x = 255 \end{cases}$$

$$\nu \sim \sum_{i=1}^{K} \pi_i logistic(\mu_i, s_i)$$

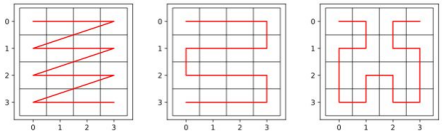
LMConv

$$p(x) = p_\theta\left(x_{\pi(1)}\right) \prod_{i=2}^{D} p_\theta\left(x_{\pi(i)} \mid Pa\left(\mathbf{x}_{\pi(i)}\right)\right)$$

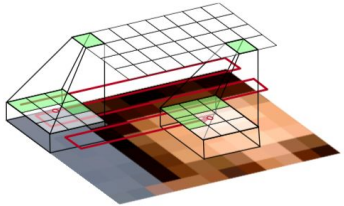




Globally coherent image completion…



+ exotic image generation orders and ensembles…

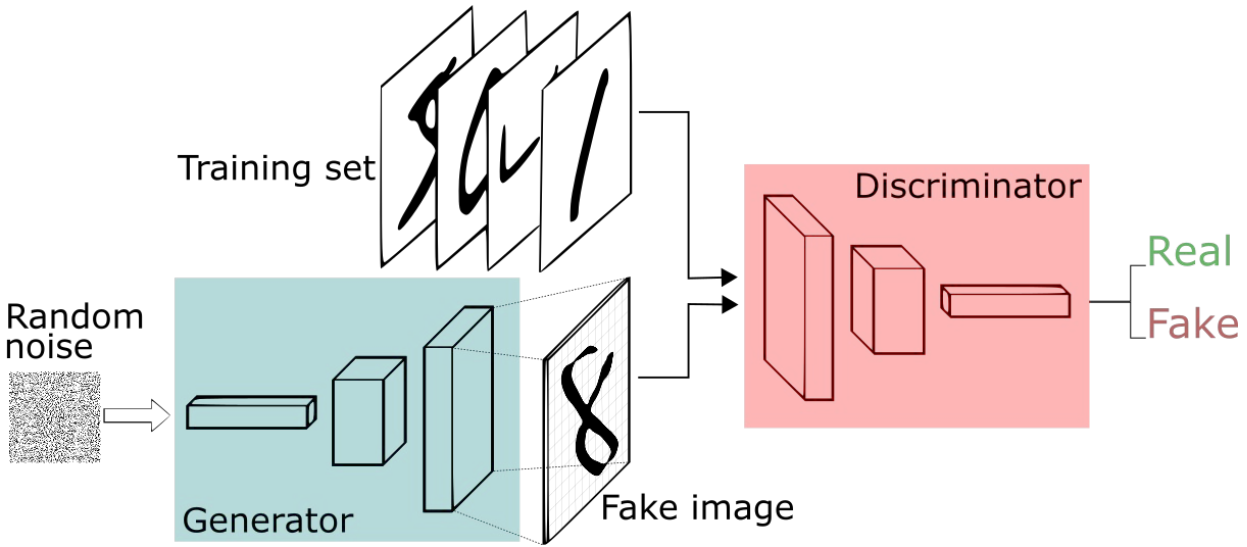


with efficient Locally Masked Convolutions!

GAN models

A natural way to set up a likelihood-free objective is to consider the two-sample test, a statistical test that determines whether or not a finite set of samples from two distributions are from the same distribution using only samples from P and Q

GAN


Training set
Discriminator
Real
Fake
Random noise
Generator
Fake image

GAN

Generator

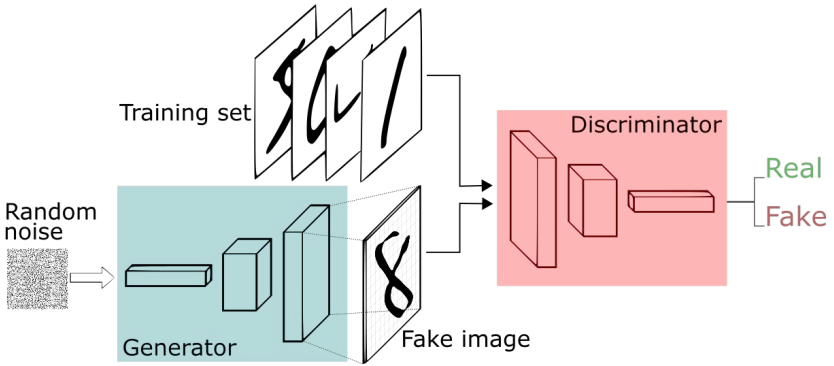
$$G : Z \to X$$

Discriminator

$$D : X \to [0, 1]$$

$p(x)$ - real distribution

$q(x)$ - generated distribution



GAN

Discriminator output

$$\begin{cases} D(x), & x \text{ is real} \\ 1 - D(x), & x \text{ is fake} \end{cases}$$

Loss function for the task:

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x} \sim \mathbf{p}_{\text{data}}} [\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_\phi(G_\theta(\mathbf{z})))]$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

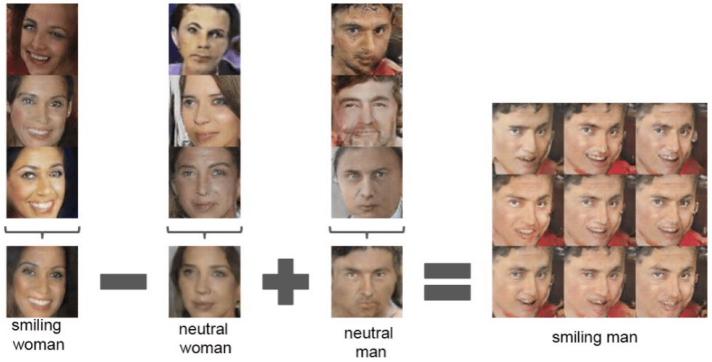
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

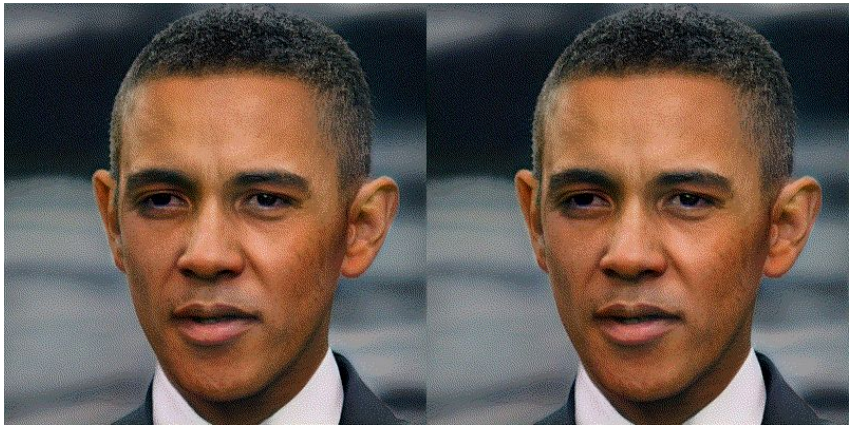
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN samples

Useful properties





Useful properties

Optimal model

Optimal discriminator:

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

$$V(G,D) = \int_x p(x)\log(D(x))dx + \int_z p(g(z))\log(1 - D(g(z)))dz =$$
$$= \int_x p(x)\log(D(x))dx + q(x)\log(1 - D(x))dx$$

Optimal model

Optimal generator (given optimal discriminator)

$$
\begin{aligned}
&V\left(G^*, D^*(G)\right) = \\
&= \int_x p(x) \log \frac{p(x)}{p(x) + q^*(x)} dx + \int_x q(x) \log \frac{q(x)}{p(x) + q^*(x)} dx \\
&= \int_x p(x) \log \frac{1}{2} dx + \int_x q(x) \log \frac{1}{2} dx = 2 \log \frac{1}{2} = -2 \log 2
\end{aligned}
$$

Jensen Shannon divergence

$$
\begin{aligned}
& V\left(G, D^*\right)-V\left(G^*, D^*\right) \\
= & \int_x p(x) \log \frac{p(x)}{p(x)+q(x)} dx+\int_x q(x) \log \frac{q(x)}{p(x)+q(x)} dx \\
& +\int_x p(x) \log 2 dx+\int_x q(x) \log 2 dx \\
= & \int_x p(x) \log \frac{p(x)}{\frac{p(x)+q(x)}{2}} dx+\int_x q(x) \log \frac{q(x)}{\frac{p(x)+q(x)}{2}} dx \\
= & K L\left(p(x) \| \frac{p(x)+q(x)}{2}\right)+K L\left(q(x) \| \frac{p(x)+q(x)}{2}\right) \geq 0
\end{aligned}
$$

GAN problems

Problem: vanishing gradient

$$D(G(z)) \approx 0 => \nabla_{\theta_G} \log(1 - D(G(z)) \approx 0$$

Solution:

$$\mathbb{E}_{z \sim p(z)}[\log(D(G(z)))] \rightarrow \max_G$$

Other options:
Adding noise; Soft labels; Flip labels

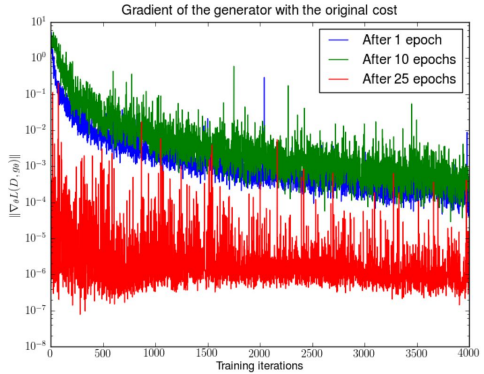
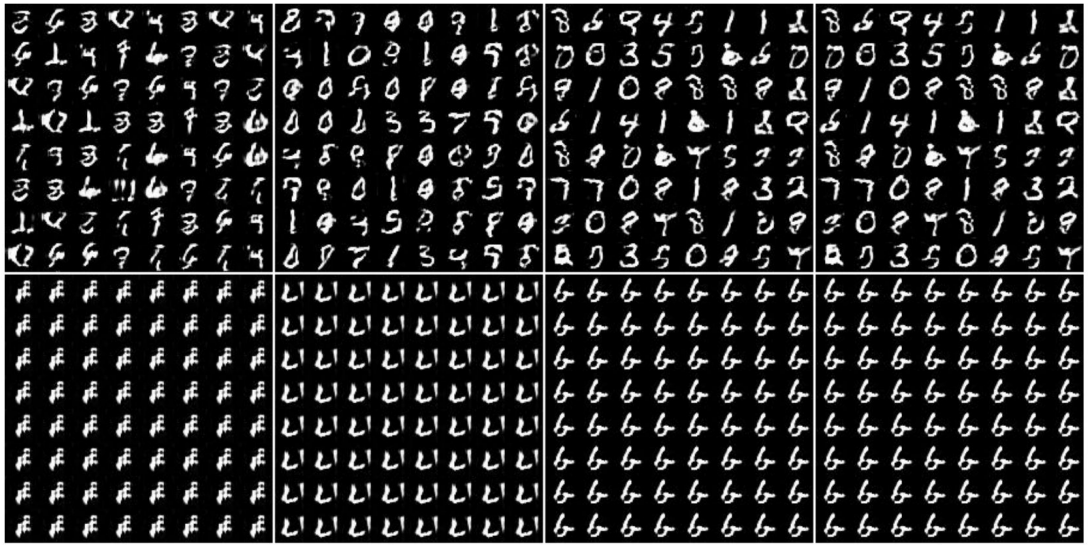


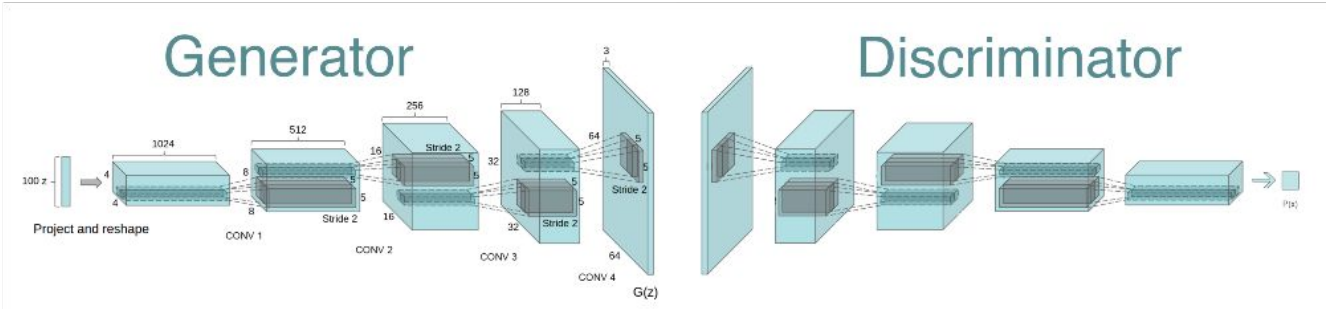
Figure 2: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations.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

GAN problems

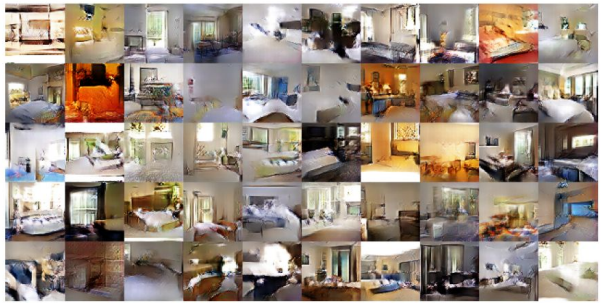
Mode collapse



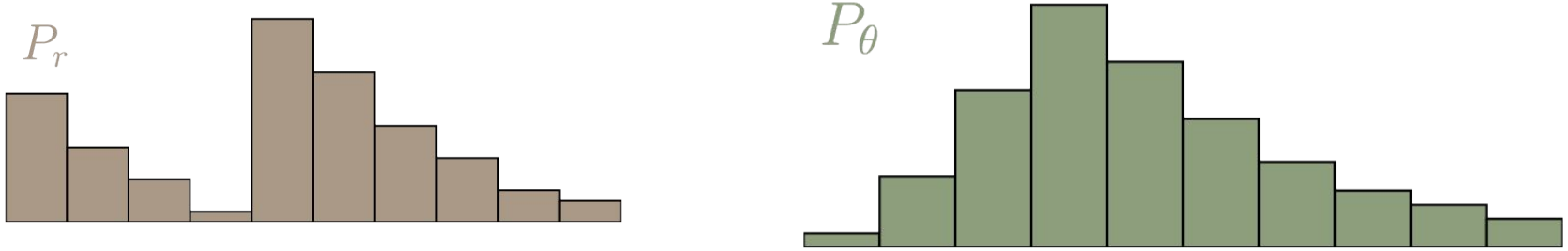
DCGAN



- Replace pooling with strided convolutions
- LeakyReLU instead of ReLU

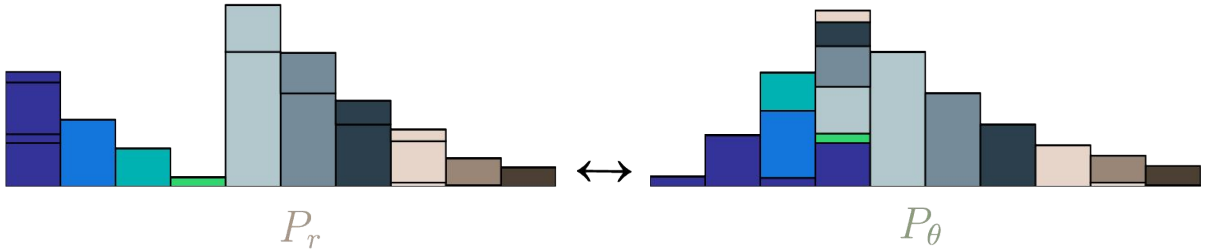


Wasserstein GAN



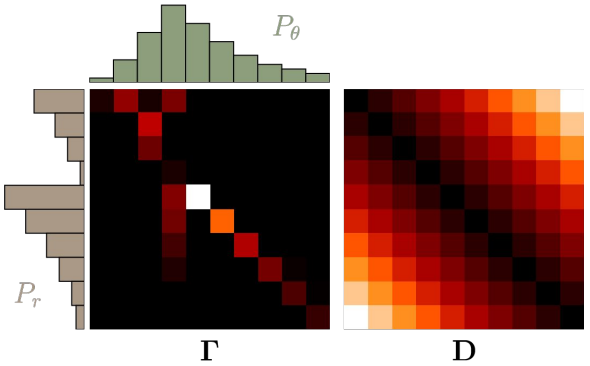
GAN tries to learn create samples close to samples from true distribution = tries to map one distribution to another

Wasserstein GAN

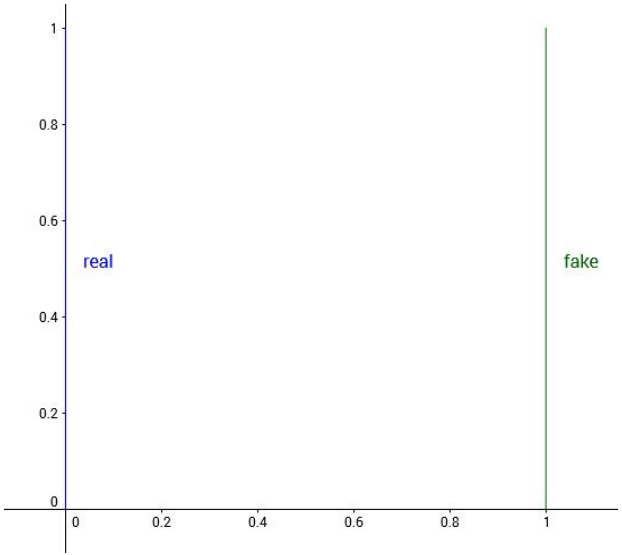


Example of mapping

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|]$$



Example



$$KL\left(p\|q_\theta\right) = KL\left(q_\theta\|p\right) = \begin{cases} +\infty, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$$

$$JS\left(p, q_\theta\right) = \begin{cases} \ln 2, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$$

$$W\left(p, q_\theta\right) = |\theta|$$

Wasserstein distance

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|]$$

f - is L-Lipschitz function

$$f(x) - f(x') \leq L\|x - x'\|$$

Wasserstein distance can be estimated using Kantorovich-Rubenstein duality

$$W(p, q_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q_\theta}[f(x)]$$

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

Wasserstein GAN

- More stable gradient

- No mode collapse:

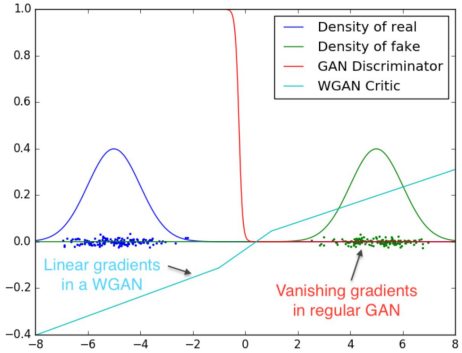
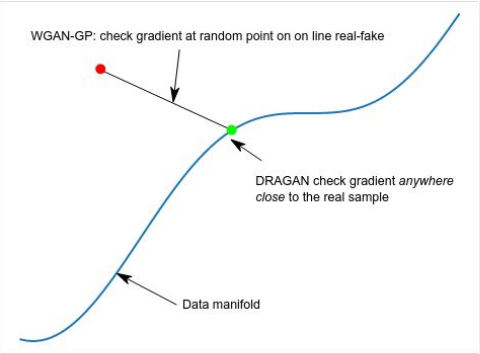


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

WGAN-GP

Clipping is bad idea to make neural network a Lipschitz function



$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}\left[D(\tilde{\boldsymbol{x}})\right] - \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathbb{P}_r}\left[D(\boldsymbol{x})\right]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\left[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\right]}_{\text{Our gradient penalty}}.$$

Spectral Normalization

$$f(\boldsymbol{x}, \theta) = W^{L+1} a_L(W^L(a_{L-1}(W^{L-1}(\dots a_1(W^1 \boldsymbol{x}) \dots))))$$

activation function — weights for each layer

$$\|f\|_{\text{Lip}} \leq \|(\boldsymbol{h}_L \mapsto W^{L+1}\boldsymbol{h}_L)\|_{\text{Lip}} \cdot \|a_L\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_{L-1} \mapsto W^L \boldsymbol{h}_{L-1})\|_{\text{Lip}}$$

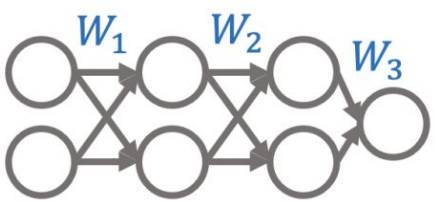
$$\cdots \|a_1\|_{\text{Lip}} \cdot \|(\boldsymbol{h}_0 \mapsto W^1 \boldsymbol{h}_0)\|_{\text{Lip}} = \prod_{l=1}^{L+1} \|(\boldsymbol{h}_{l-1} \mapsto W^l \boldsymbol{h}_{l-1})\|_{\text{Lip}} = \prod_{l=1}^{L+1} \sigma(W^l).$$

$$\bar{W}_{\text{SN}}(W) := W/\sigma(W)$$

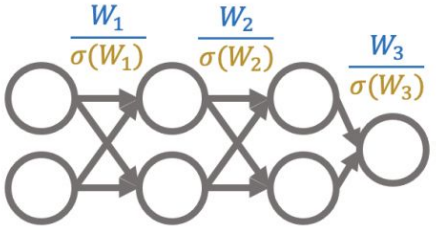
$$\sigma\left(\bar{W}_{\text{SN}}(W)\right) = 1$$

$$\|f\|_{\text{Lip}} = 1$$

(a) Discriminator without SN



(b) Discriminator with SN



Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2 \tag{20}$$

$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2 \tag{21}$$

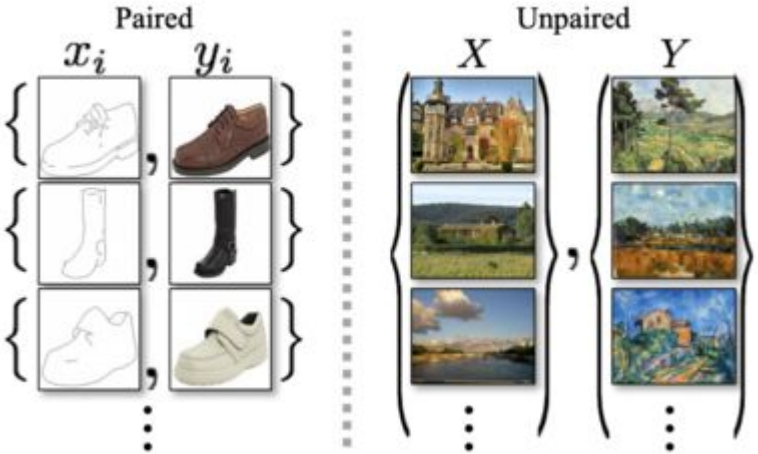
  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

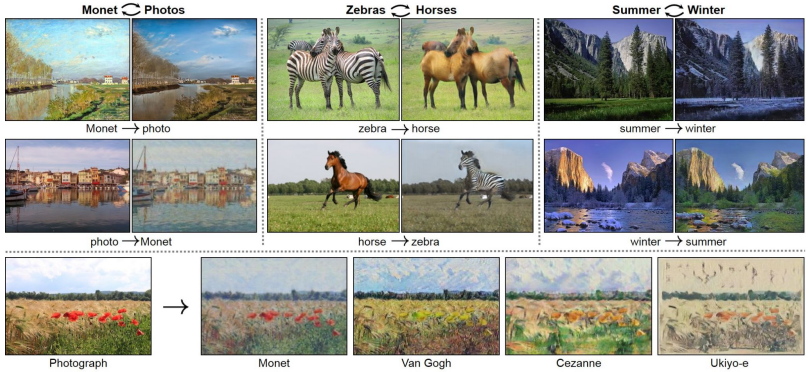
$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l \tag{22}$$

  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M) \tag{23}$$

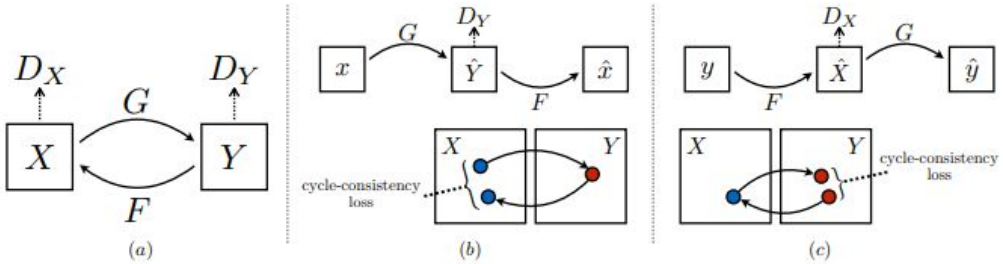
Cycle GAN





Want to map one distribution to another and vice versa. Day <-> Night. Summer <-> Winter

Cycle GAN



$$\min_{F,G,D_{\mathcal{X}},D_{\mathcal{Y}}} \mathcal{L}_{GAN}(G, D_{\mathcal{Y}}, X, Y) + \mathcal{L}_{GAN}(F, D_{\mathcal{X}}, X, Y) + \lambda \left(\mathbb{E}_X[||F(G(X)) - X||_1] + \mathbb{E}_Y[||G(F(Y)) - Y||_1]\right)$$

Recap

- Generative models
- Kernel density estimation
- Autoregressive models (PixelCNN, PixelRNN)
- GAN
- WGAN
- CycleGAN

Metrics

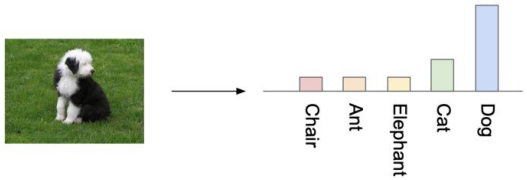
How do automatically measure the performance of generative models?

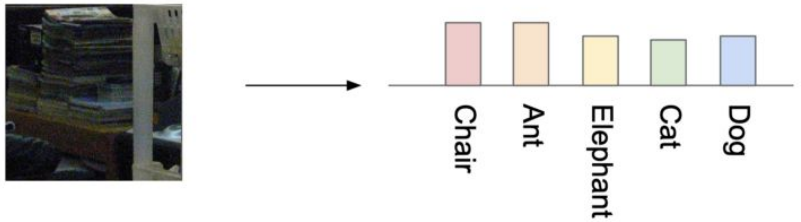
- The images generated should contain clear objects (i.e. the images are sharp rather than blurry), or p(y|x) should be low entropy.

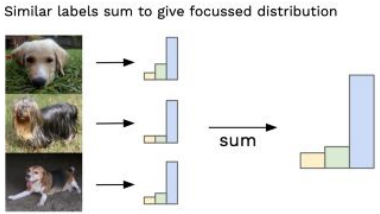
- The generative algorithm should output a high diversity of images from all the different classes in ImageNet, or p(y) should be high entropy.

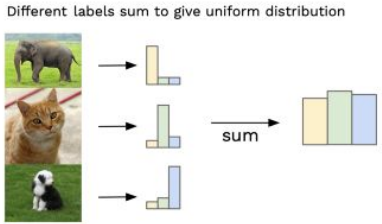
$$\ln(\mathrm{IS}(G)) = I(y; \mathbf{x}) \qquad I(y; \mathbf{x}) = H(y) - H(y \mid \mathbf{x})$$

Inception Score









How to get the distributions?

Inception Score problems

Consider two classes with data distributions N(−1, 2) and N(1, 2).

Then, optimal generator will produce -\infty and \infty

What if we use dataset for generation of unique objects which are absent in ImageNet?

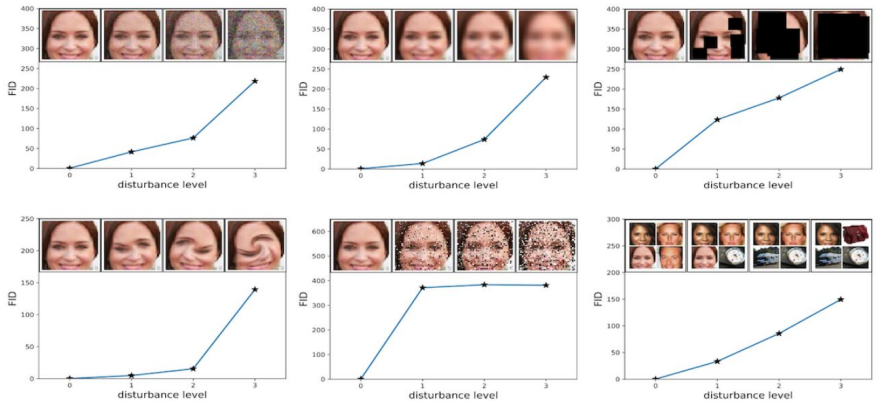
Inception Score

How to calculate Inception Score:

- Create n=10 splits.
- Calculate Inception Score independently on each dataset

Frechet Inception Distance

Real and generated distribution should be close. But it's hard to measure distance between distribution on raw pixels -> let's use the latent representation



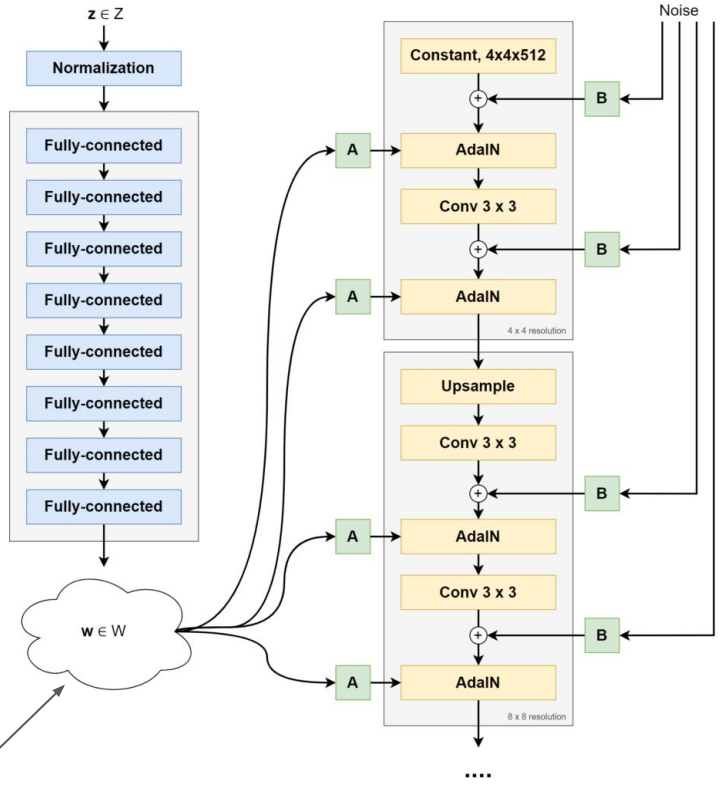
$$FID(r, g) = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r \Sigma_g\right)^{\frac{1}{2}}\right)$$

Style-GAN

Motivation:

- Generators operate as black boxes. Latent spaces of classic GANs were poorly understood
- GANs must be heavily regularized.
- There is little control over image synthesis.

StyleGAN



Disentagled space

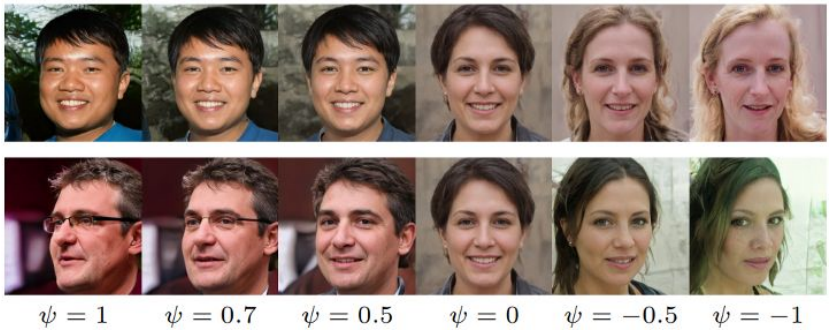
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

Truncation trick

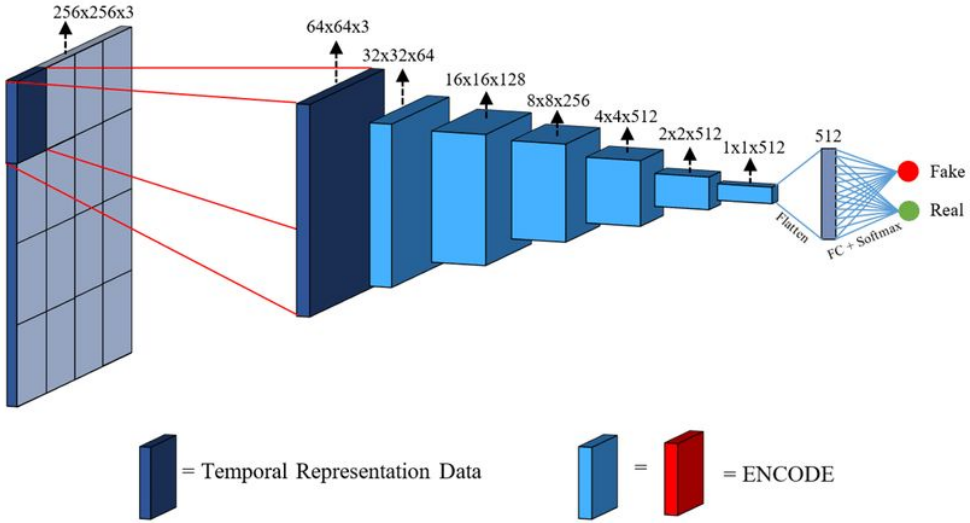
How to choose image for paper?

$$\bar{\mathbf{w}} = \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[f(\mathbf{z})]$$

$$\mathbf{w}' = \bar{\mathbf{w}} + \psi(\mathbf{w} - \bar{\mathbf{w}})$$



Patch GAN



Instead of using all image for classification one can use patches of it. Better modelling of high frequency details

f-GAN

$$D_f(P\|Q) = \int_{\mathcal{X}} q(x) f\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x$$

f - convex
f - semicontinious
f(0) = 1

$$\begin{aligned} D_f(P\|Q) &= \int_{\mathcal{X}} q(x) \sup_{t \in \mathrm{dom}_{f^*}} \left\{ t\frac{p(x)}{q(x)} - f^*(t) \right\} \mathrm{d}x \\ &\geq \sup_{T \in \mathcal{T}} \left(\int_{\mathcal{X}} p(x) T(x) \mathrm{d}x - \int_{\mathcal{X}} q(x) f^*(T(x)) \mathrm{d}x \right) \\ &= \sup_{T \in \mathcal{T}} \left(\mathbb{E}_{x \sim P}[T(x)] - \mathbb{E}_{x \sim Q}\left[f^*(T(x))\right] \right), \end{aligned}$$

f-GAN

Name	$D_f(P\|Q)$	Generator $f(u)$	$T^*(x)$
Kullback-Leibler	$\int p(x) \log \frac{p(x)}{q(x)} \mathrm{d}x$	$u \log u$	$1 + \log \frac{p(x)}{q(x)}$
Reverse KL	$\int q(x) \log \frac{q(x)}{p(x)} \mathrm{d}x$	$-\log u$	$-\frac{q(x)}{p(x)}$
Pearson χ^2	$\int \frac{(q(x)-p(x))^2}{p(x)} \mathrm{d}x$	$(u-1)^2$	$2(\frac{p(x)}{q(x)} - 1)$
Squared Hellinger	$\int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 \mathrm{d}x$	$\left(\sqrt{u}-1\right)^2$	$(\sqrt{\frac{p(x)}{q(x)}} - 1) \cdot \sqrt{\frac{q(x)}{p(x)}}$
Jensen-Shannon	$\frac{1}{2} \int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \mathrm{d}x$	$-(u+1) \log \frac{1+u}{2} + u \log u$	$\log \frac{2p(x)}{p(x)+q(x)}$
GAN	$\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \mathrm{d}x - \log(4)$	$u \log u - (u+1) \log(u+1)$	$\log \frac{p(x)}{p(x)+q(x)}$

Progressive GAN

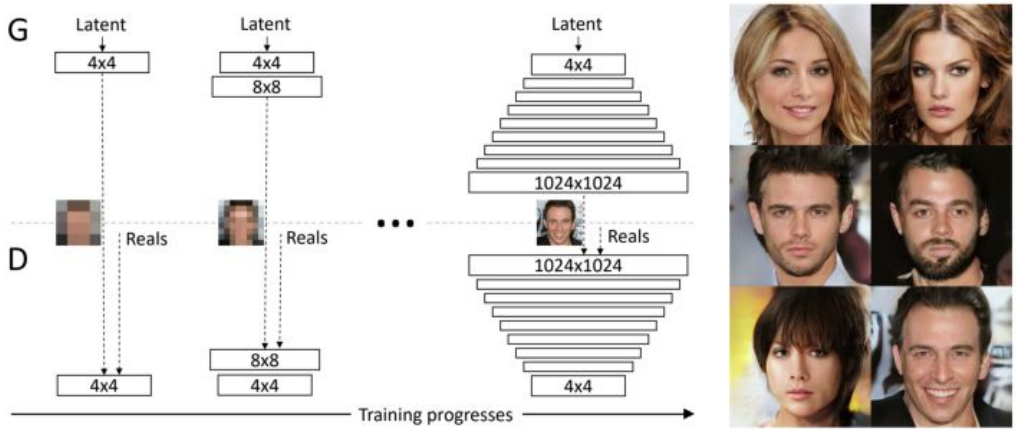
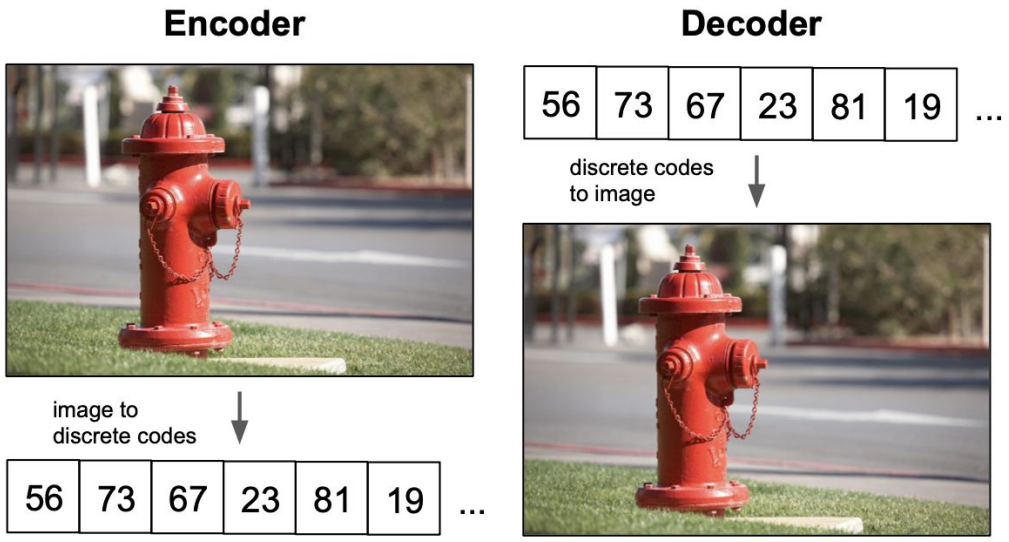
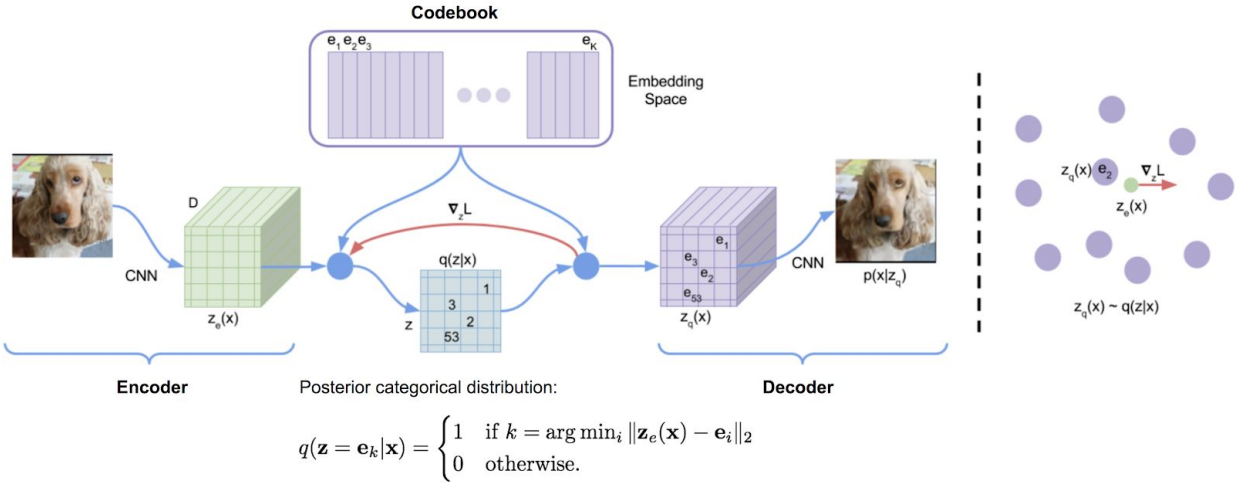


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Vector Quantised (VQ)-VAE



VQ-VAE



$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

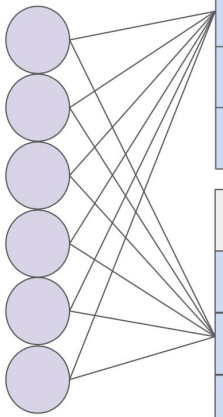
$$\| \log(p(x \mid q(x))) + \| \operatorname{sg}\left[z_e(x)\right] - e \|_2^2 + \beta \| z_e(x) - \operatorname{sg}[e] \|_2^2$$

VQ-VAE

1. an image is fed to the encoder network

...

2. the encoder outputs distributions over codebook vectors

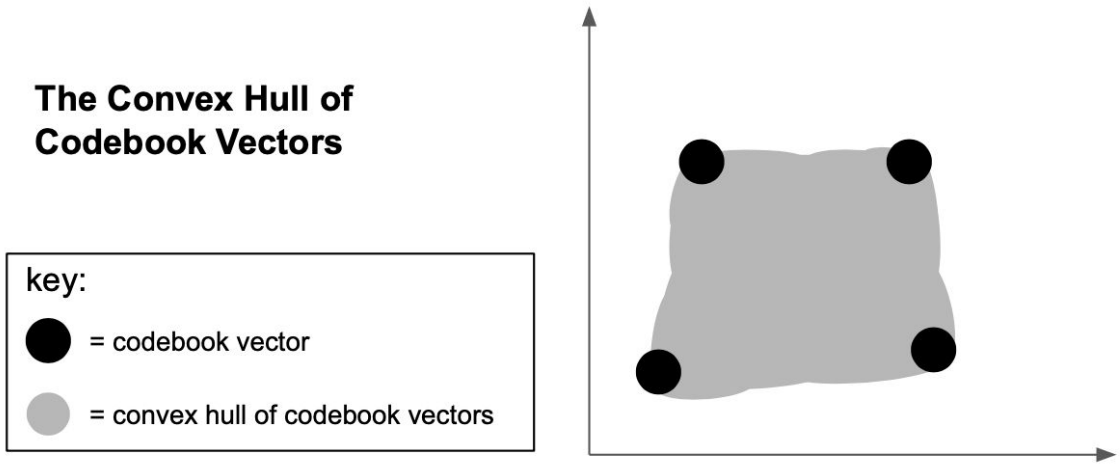


Latent Index	Prob
0	0.2
1	0.3
2	0.4
3	0.1

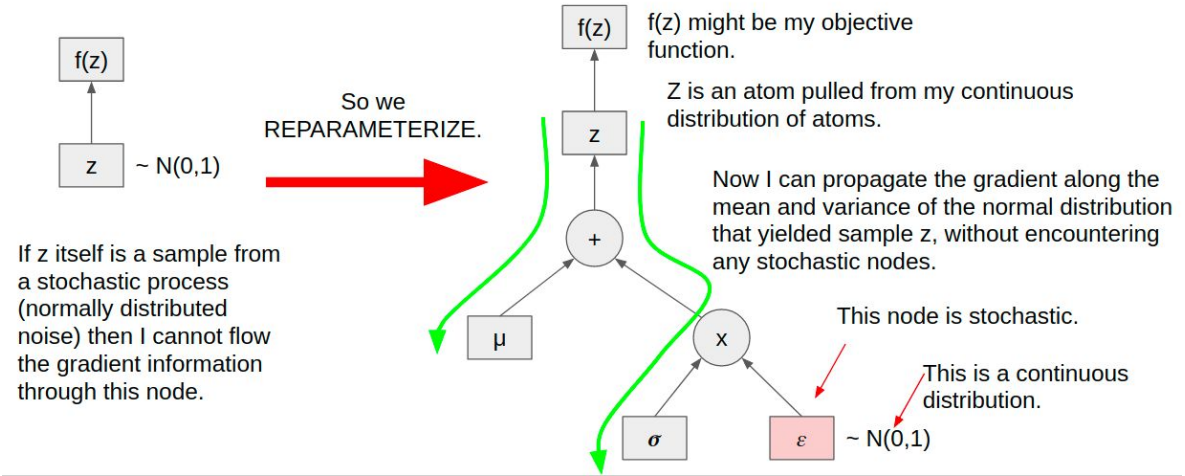
Latent Index	Prob
0	0.4
1	0.2
2	0.15
3	0.25

Latent Index	Codebook Vector
0	[0.01, -2.3, 5.6, 0.04, -0.1, 8.92, 3.24, ...]
1	[5.4, 0.65, 0.2, 4.6, 8.9, -2.43, 0.07, ...]
2	[9.78, 0.67, -3.4, 0.2, -1.0, 7.2, 13.8, ...]
3	[2.45, -8.9, 0.3, 2.04, -0.89, 19.1, 0.3, ...]

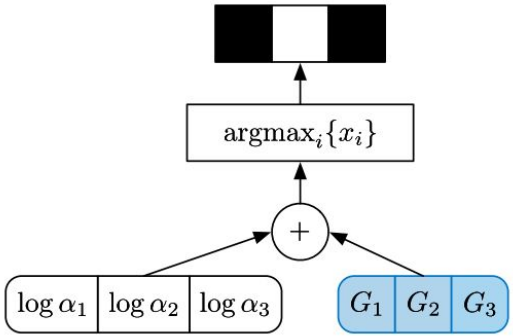
VQ-VAE

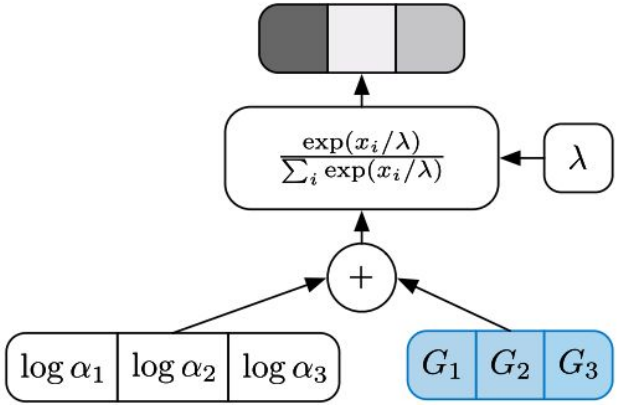

The Convex Hull of
Codebook Vectors
key:
= codebook vector
= convex hull of codebook vectors

Gumbel-Softmax trick



Gumbel-softmax trick





$$y_i = \frac{e^{\frac{(log(\pi_i)+g_i)}{\tau}}}{\sum_{j=1}^{k} e^{\frac{(log(\pi_j)+g_j)}{\tau}}} \text{ for } i = 1, ..., k$$

Gumbel-Softmax

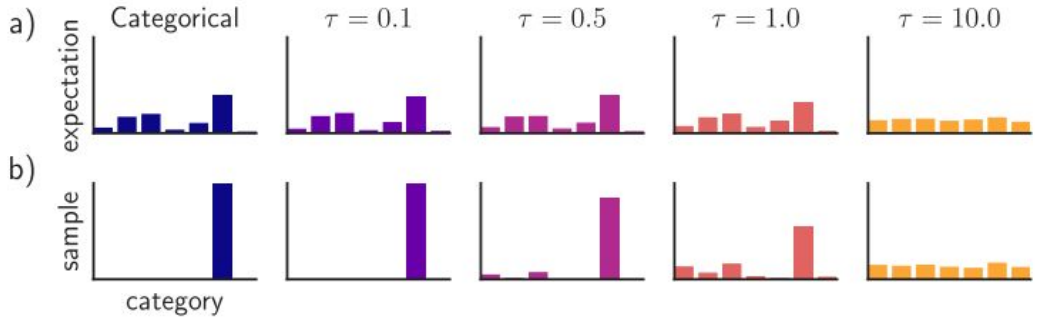
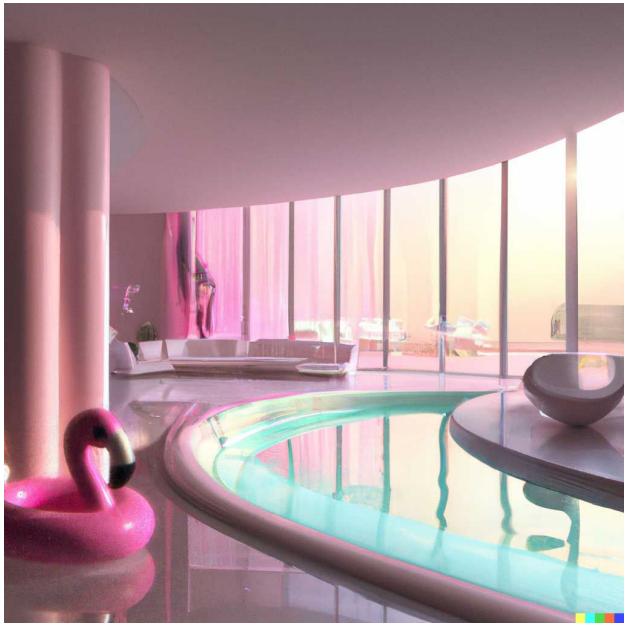


Figure 1: The Gumbel-Softmax distribution interpolates between discrete one-hot-encoded categorical distributions and continuous categorical densities. (a) For low temperatures ($\tau = 0.1, \tau = 0.5$), the expected value of a Gumbel-Softmax random variable approaches the expected value of a categorical random variable with the same logits. As the temperature increases ($\tau = 1.0, \tau = 10.0$), the expected value converges to a uniform distribution over the categories. (b) Samples from Gumbel-Softmax distributions are identical to samples from a categorical distribution as $\tau \to 0$. At higher temperatures, Gumbel-Softmax samples are no longer one-hot, and become uniform as $\tau \to \infty$.

VQ-VAE



Recap