Нейронные сети в машинном обучении

Генеративно-состязательные сети

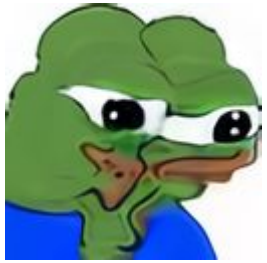
Рогачев Александр

Generative Models

Generative Models Hype

[Stable Diffusion - a Hugging Face Space]

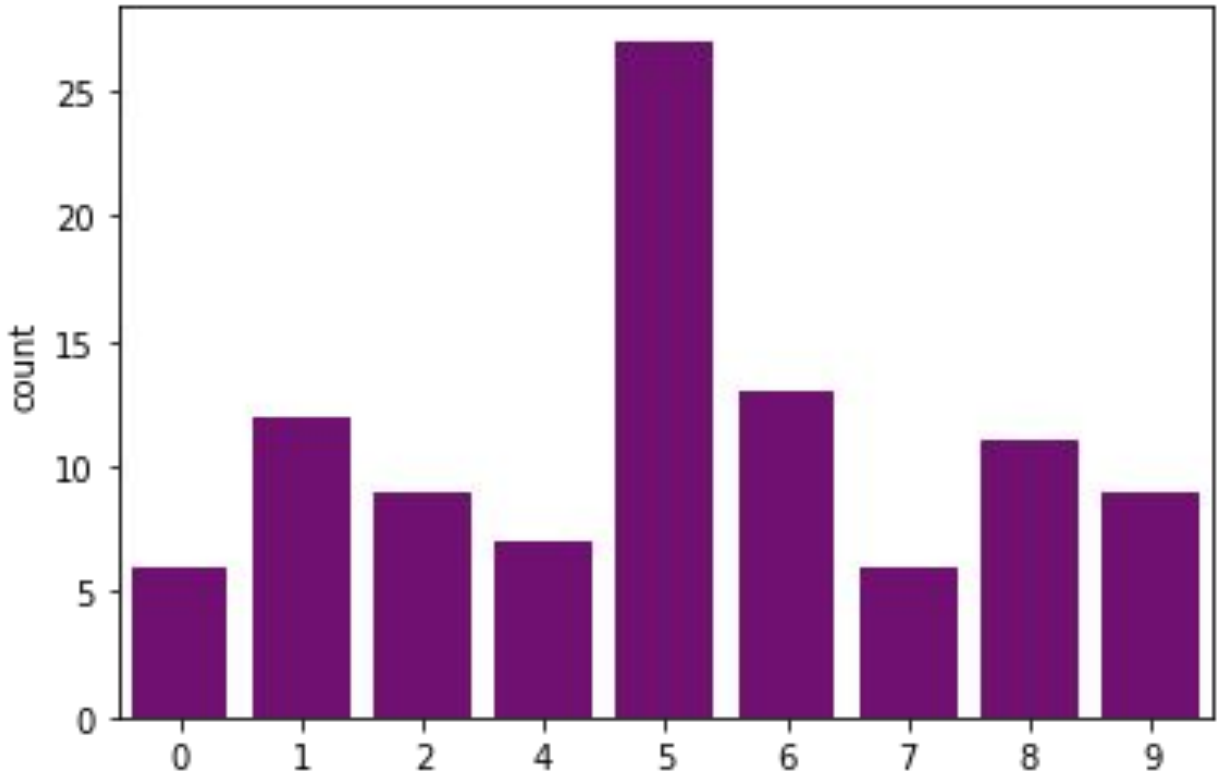
[DALL·E mini by craiyon.com]



[This X Does Not Exist]

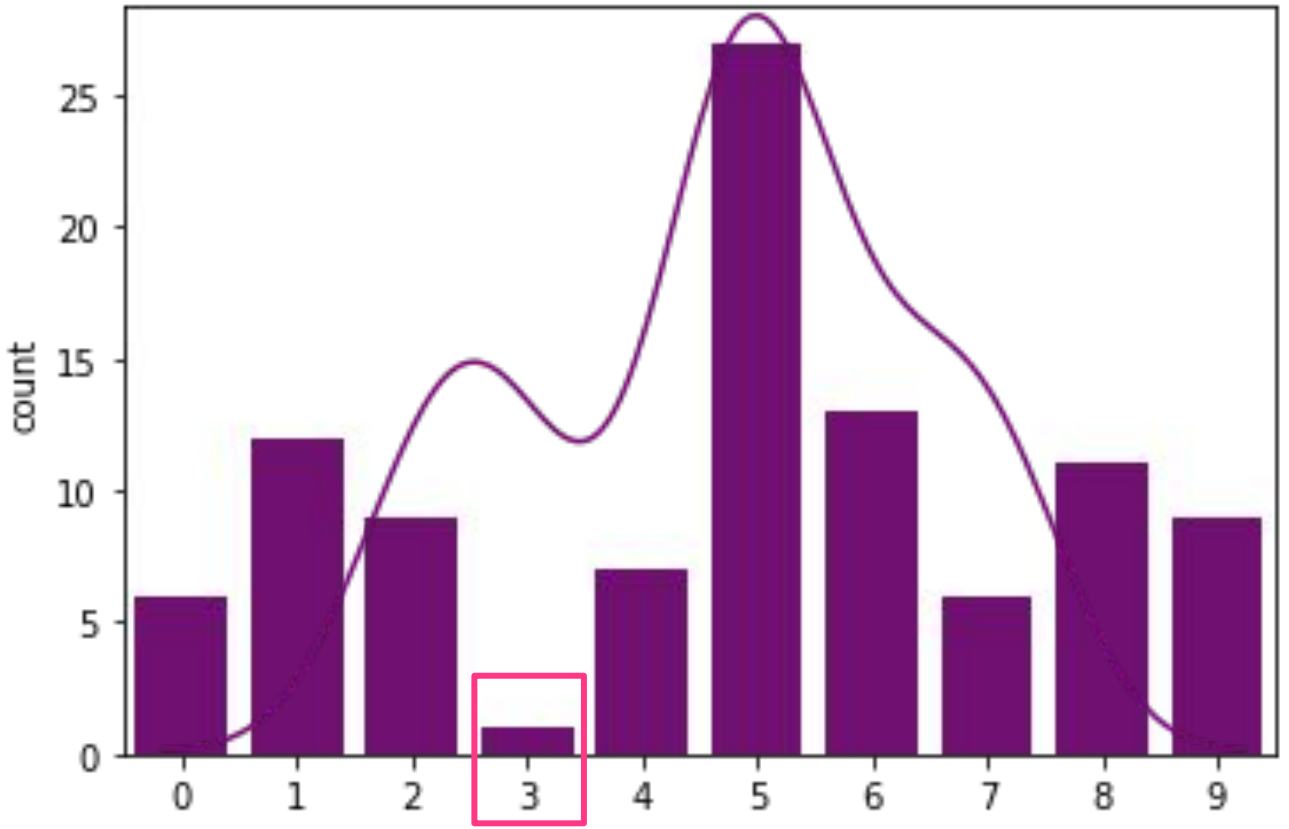
Toy Example

- We have an observed sample with numbers
- We want to generate a new object
- Let's assume there is a probability density p(x)
- We can try to estimate it using our sample and obtain $p_{data}(x)$
- Now we can sample from $p_{data}(x)$, e.g. `np.random.choice(data)`
- **Do we face any problem here?**



Toy Example

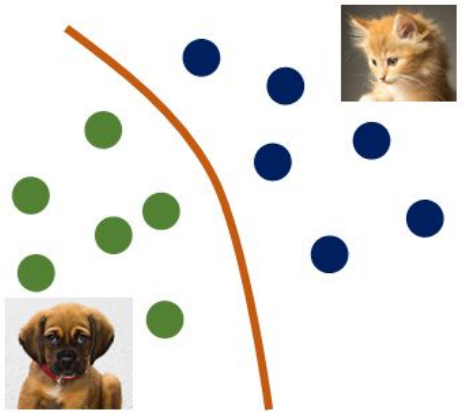
- We do not have “3” in our sample
- Common sense may help us
- Let’s pick up an **interpolation model**
- Now we can sample a new object that was not introduced into our data sample before



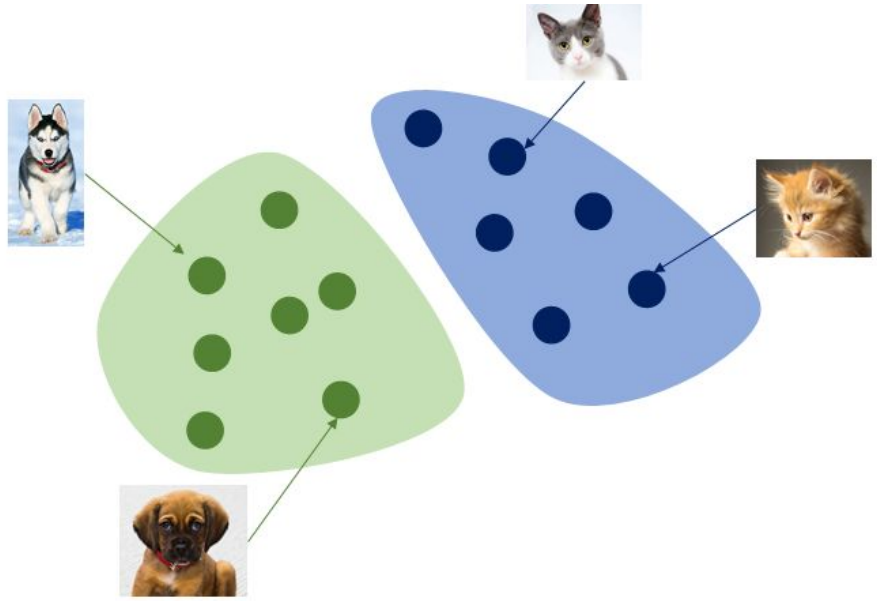
Discriminative VS Generative

For a given observations *x* and targets *y*:

- A discriminative model is a model of the conditional probability **P(y|x)**
- A generative model is a model of the joint probability distribution **P(x,y)**



Discriminative



Generative

https://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

Vanilla GAN

GAN Idea

- Let's assume that we have a **generator G** that samples from a random noise:

$$z \sim N(0;1)$$

$$x = G(z)$$

- G is an instance of NN
- Now we can sample generated objects and try to compare them to real ones
- You can check the batch of cats and give your predictions whether some particular image is real or not
- We can't use backprob in case of labels provided by ourselves, so let's build another network called **discriminator D** to distinguish between the real and generated samples.

Training Vanilla GAN. minimax game

Discriminator

- Classification setting
- Learns to map object into labels (real, fake)
- *D(x)* represents the probability that *x* came from the data rather than it was generated
- *D* maximizes the probability of assigning the correct label to both training examples and samples from *G*
- *D(G(z)) = 0*

Generator

- Reversed bottleneck architecture
- *G(z)* outputs objects we want to generate
- Tries to trick Discriminator *D*
- *D(G(z)) = 1*

D and G play the following two-player minimax game with value function V (G, D):

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

Goodfellow I. J. et al. Generative adversarial networks //arXiv preprint arXiv:1406.2661. – 2014.

Training Vanilla GAN. Algorithm

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:

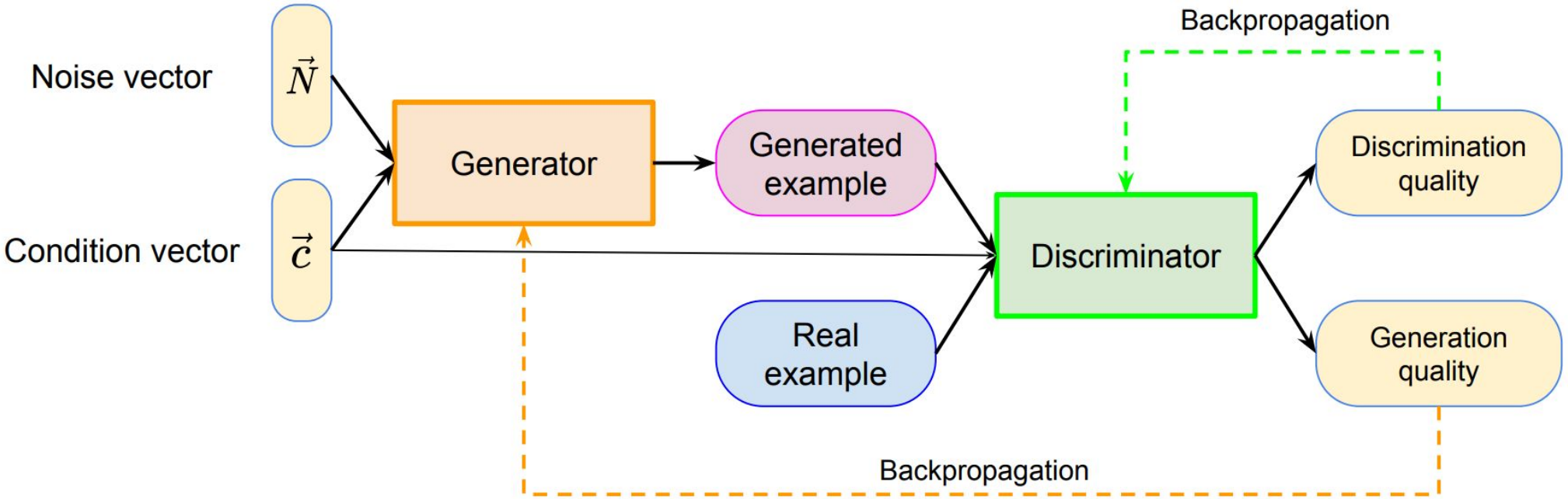
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Goodfellow I. J. et al. Generative adversarial networks //arXiv preprint arXiv:1406.2661. – 2014.

GAN conceptual example



Vanilla GAN Global Optimality

- Let's rewrite V using definition of expected value of continuous random variable
- Perform transition from *z* to *x* as *x = G(z)* *(Law of the unconscious statistician)*
- Discriminator maximizes this expression, so let's find it's optimum
- Fix *G*
- For any $(a, b) \in \mathbb{R}^2 \setminus \{0, 0\}$, the function
  $y \to a \log(y) + b \log(1 - y)$
  achieves its maximum in [0, 1] at $\frac{a}{a+b}$
- Use this expression to get $D^*(x)$

$$V(G,D) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz$$
$$= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx$$

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

Medium: proof gan optimal point

Stack Exchange: Optimal Discriminator

Goodfellow I. J. et al. Generative adversarial networks //arXiv preprint arXiv:1406.2661. – 2014.

Vanilla GAN Global Optimality

- Recap KL and JSD
- Think of Jensen-Shannon divergence like symmetric KL divergence
- Let $\boldsymbol{D = D^*(x)}$
- If $p_{data}(x) = p_g(x)$, $D^*(x) = 0.5$
- Thus V(G, D*) = C(G) = log(0.5) + log(0.5)= − log 4
- JSD between two distributions is always non-negative and zero only when they are equal,
- C(G) = − log(4) is the global minimum of and that the only solution
- GAN perfectly replicating the data generating process

$$KL(\pi||p) = \int \pi(\mathbf{x}) \log \frac{\pi(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})} d\mathbf{x}$$

$$JSD(\pi(\mathbf{x})||p(\mathbf{x}|\boldsymbol{\theta})) = \frac{1}{2}\left[KL\left(\pi(\mathbf{x})||\frac{\pi(\mathbf{x}) + p(\mathbf{x}|\boldsymbol{\theta})}{2}\right) + KL\left(p(\mathbf{x}|\boldsymbol{\theta})||\frac{\pi(\mathbf{x}) + p(\mathbf{x}|\boldsymbol{\theta})}{2}\right)\right]$$

$$\begin{aligned} C(G) &= \max_D V(G, D) \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned}$$

$$C(G) = -\log(4) + KL\left(p_{\text{data}}\left\|\frac{p_{\text{data}} + p_g}{2}\right.\right) + KL\left(p_g\left\|\frac{p_{\text{data}} + p_g}{2}\right.\right)$$

$$\boxed{C(G) = -\log(4) + 2 \cdot JSD\left(p_{\text{data}}\|p_g\right)}$$

Goodfellow I. J. et al. Generative adversarial networks //arXiv preprint arXiv:1406.2661. – 2014.

Vanilla GAN

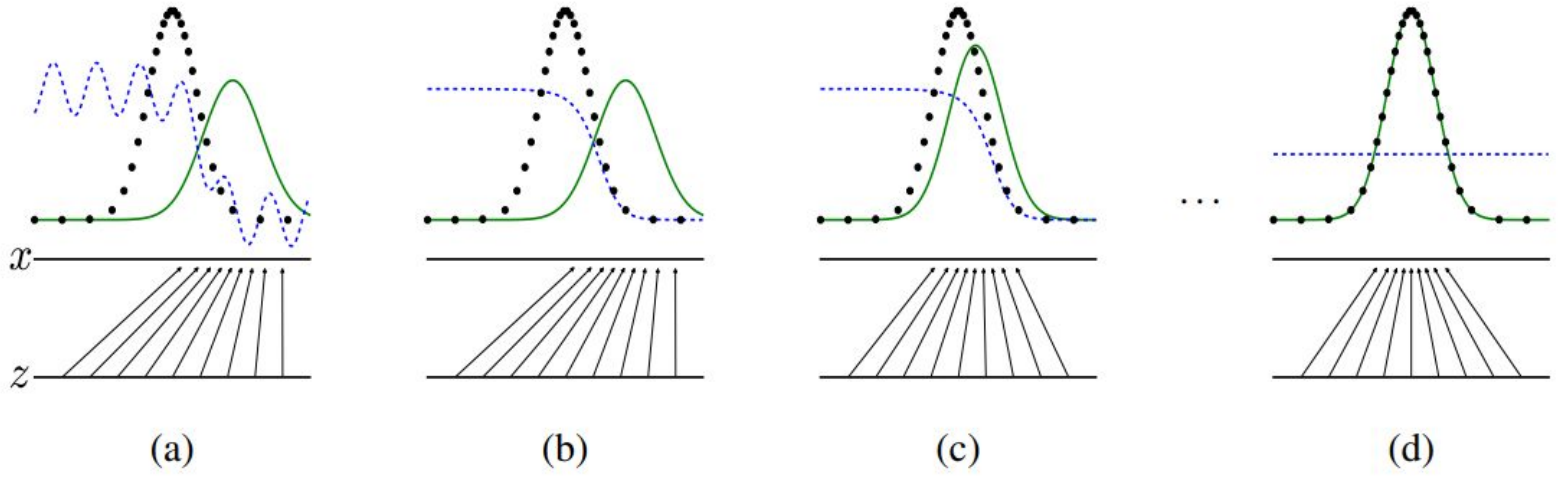


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Vanilla GAN Results

Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

GAN Problems

Vanishing gradients

- During early epochs G is weak
- D can reject samples with high confidence
- log(1 − D(G(z))) saturates

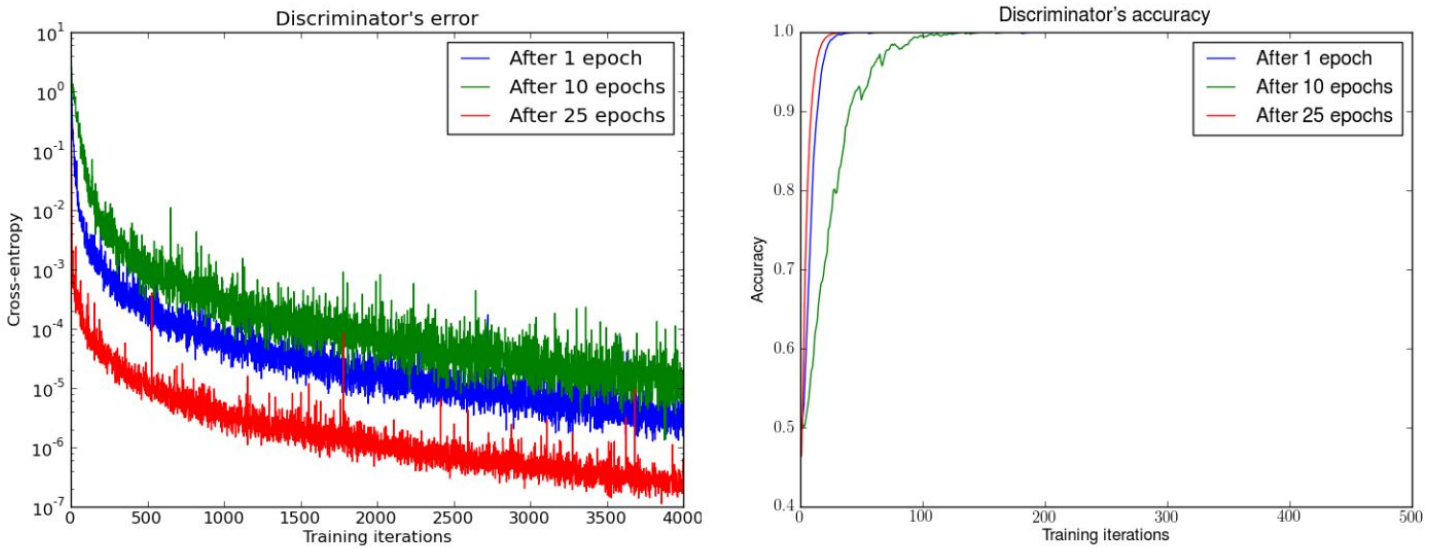
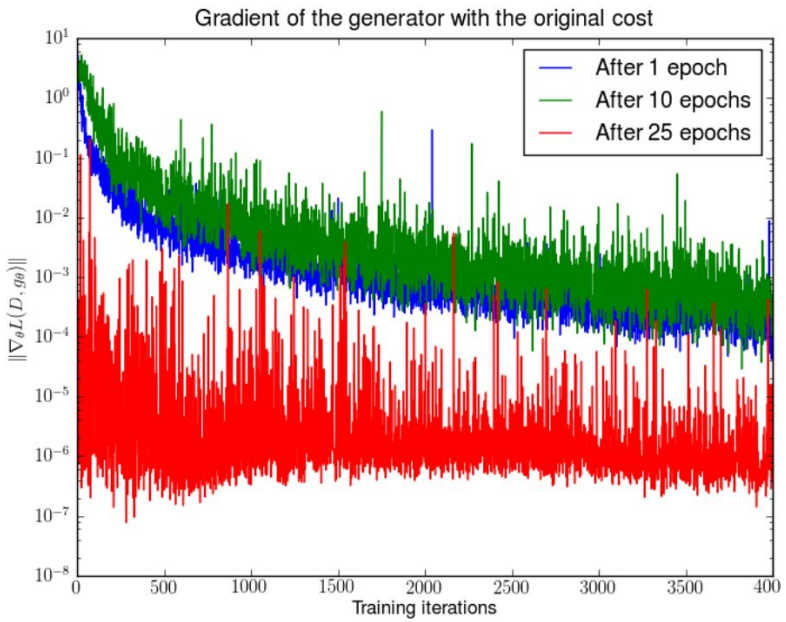


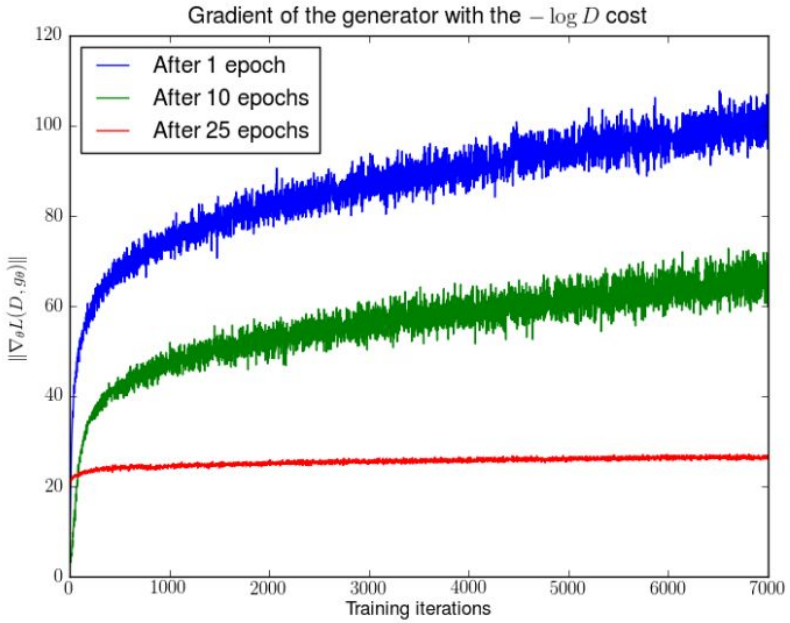
Figure 1: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch. We see the error quickly going to 0, even with very few iterations on the discriminator. This even happens after 25 epochs of the DCGAN, when the samples are remarkably good and the supports are likely to intersect, pointing to the non-continuity of the distributions. Note the logarithmic scale. For illustration purposes we also show the accuracy of the discriminator, which goes to 1 in sometimes less than 50 iterations. This is 1 even for numerical precision, and the numbers are running averages, pointing towards even faster convergence.

Arjovsky M., Bottou L. Towards principled methods for training generative adversarial networks //arXiv preprint arXiv:1701.04862. – 2017.

Vanishing gradients

- Maximize log D(G(z)) instead of minimizing log(1 − D(G(z)))
- While new version of gradient doesn't necessarily suffer from vanishing gradients, it does cause massively unstable updates

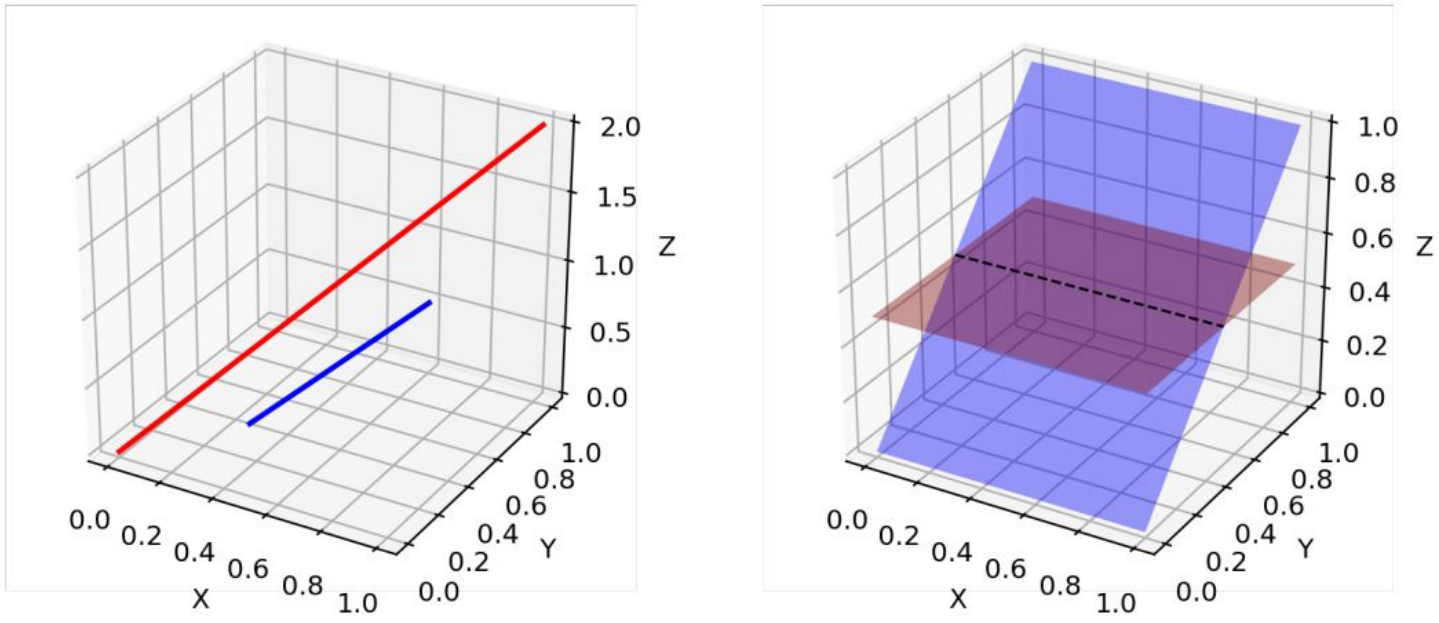




[Arjovsky M., Bottou L. Towards principled methods for training generative adversarial networks //arXiv preprint arXiv:1701.04862. – 2017.]

Low Dimensional Supports

- Manifold - a topological space that locally resembles Euclidean space near each point
- Support - A real-valued function f is the subset of the domain containing those elements which are not mapped to zero.
- The dimensions of many real-world datasets only appear to be artificially high.
- Once the theme or the contained object is fixed, the images have a lot of restrictions to follow, i.e., a dog should have two ears and a tail, and a skyscraper should have a straight and tall body, etc.
- These restrictions keep images away from the possibility of having a high-dimensional free form.
- Whenever the generator is asked to a much larger image given a small dimension noise variable input, the distribution of colors over these pixels has been defined by the small random number vector and can hardly fill up the whole high dimensional space.
- Because both p_g and p_r rest in low dimensional manifolds, they are almost certainly disjoint.
- When they have disjoint supports, we are always capable of finding a perfect discriminator that separates real and fake samples correctly



Low dimensional manifolds in high dimension space can hardly have overlaps. (Left) Two lines in a three-dimension space. (Right) Two surfaces in a three-dimension space.

Arjovsky M., Bottou L. Towards principled methods for training generative adversarial networks //arXiv preprint arXiv:1701.04862. – 2017.

Mode collapse

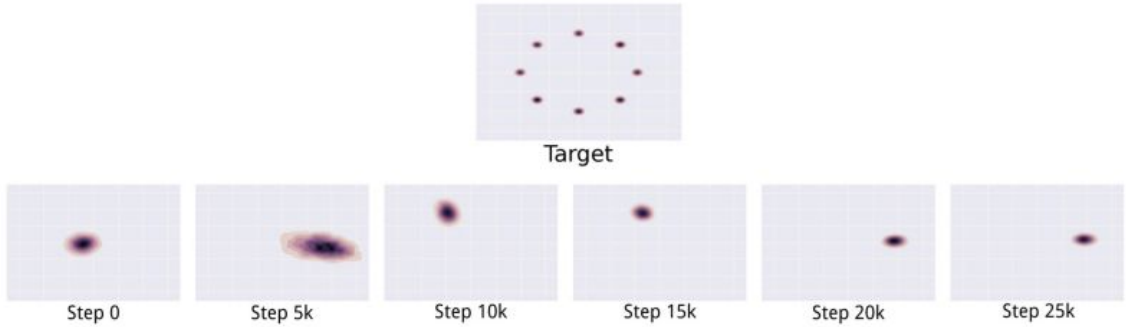


Figure 22: An illustration of the mode collapse problem on a two-dimensional toy dataset. In the top row, we see the target distribution p_{data} that the model should learn. It is a mixture of Gaussians in a two-dimensional space. In the lower row, we see a series of different distributions learned over time as the GAN is trained. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one.

Unrolled Generative Adversarial Networks

Mode collapse. DCGAN

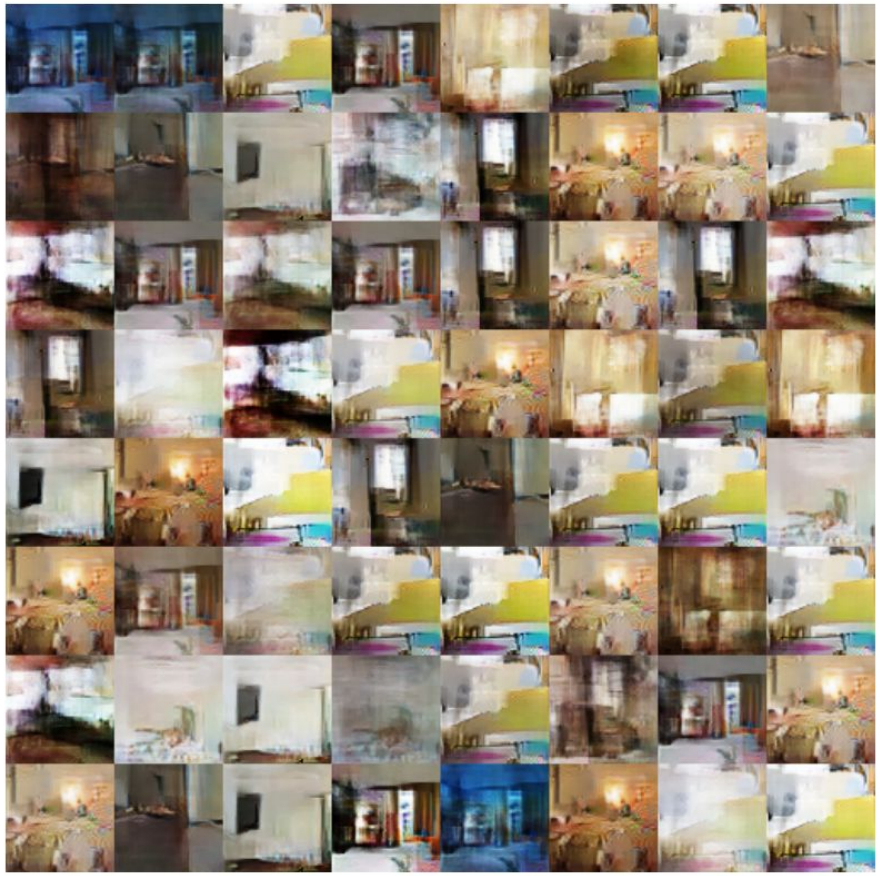
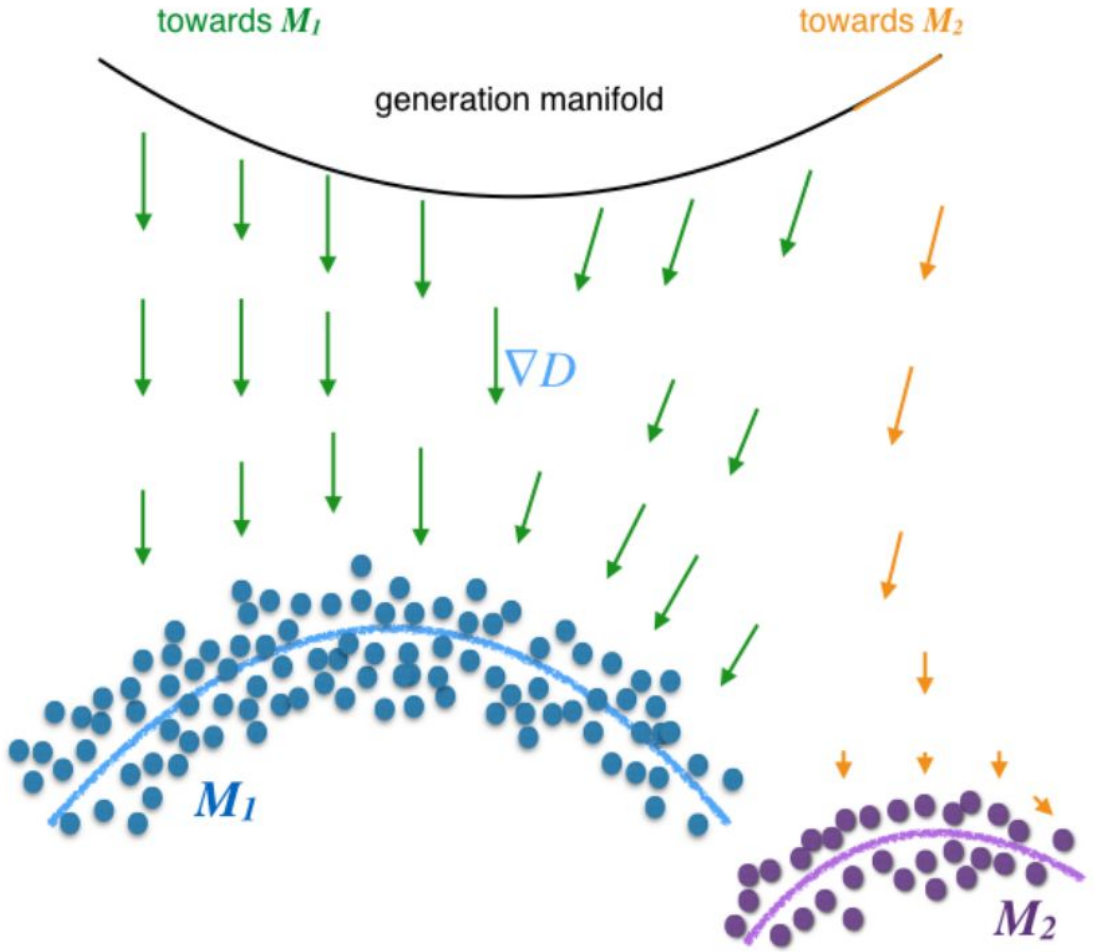


Figure 14: Standard GAN procedure: generator is an MLP with 4 hidden layers of 512 units, discriminator is a DCGAN.

Wasserstein GAN

Mode collapse

- For most z, the gradient of the generator pushes the generator towards the major mode M1 to fool D the most
- *G* may even become independent on *z*
- Only when G(z) is very close to the mode M2 the generator can get gradients to push itself towards the minor mode M2
- As D restarts, it easily finds objects, corresponding to M1
- G finds new mode

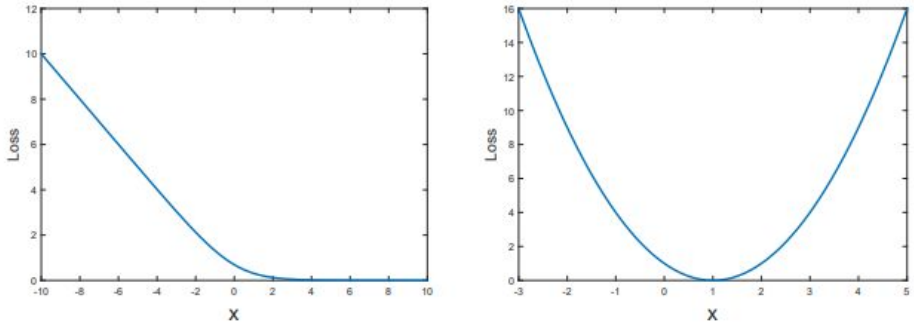


[Mode Regularized Generative Adversarial Networks]

GAN Improvements

LSGAN

- LSGANs adopts least squares loss function for the discriminator.
- Mao X. et al show that minimizing the objective function of LSGAN yields minimizing the Pearson χ^2 divergence.
- a-b coding scheme for the discriminator, where *a* and *b* are the labels for fake data and real data, respectively.
- *c* denotes the value that G wants D to believe for fake data



(a) (b)

Figure 2: (a): The sigmoid cross entropy loss function. (b): The least squares loss function.

$$\min_D V_{\mathrm{LSGAN}}(D) = \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\mathrm{data}}(\boldsymbol{x})}\left[(D(\boldsymbol{x})-b)^2\right] + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-a)^2\right]$$

$$\min_G V_{\mathrm{LSGAN}}(G) = \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[(D(G(\boldsymbol{z}))-c)^2\right],$$

Mao X. et al. Least squares generative adversarial networks //Proceedings of the IEEE international conference on computer vision. – 2017. – С. 2794-2802.

LSGAN. Toy example

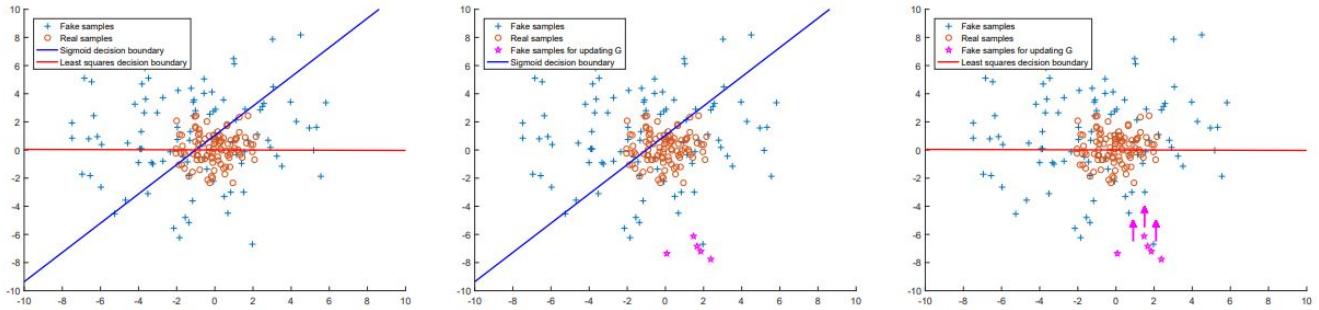


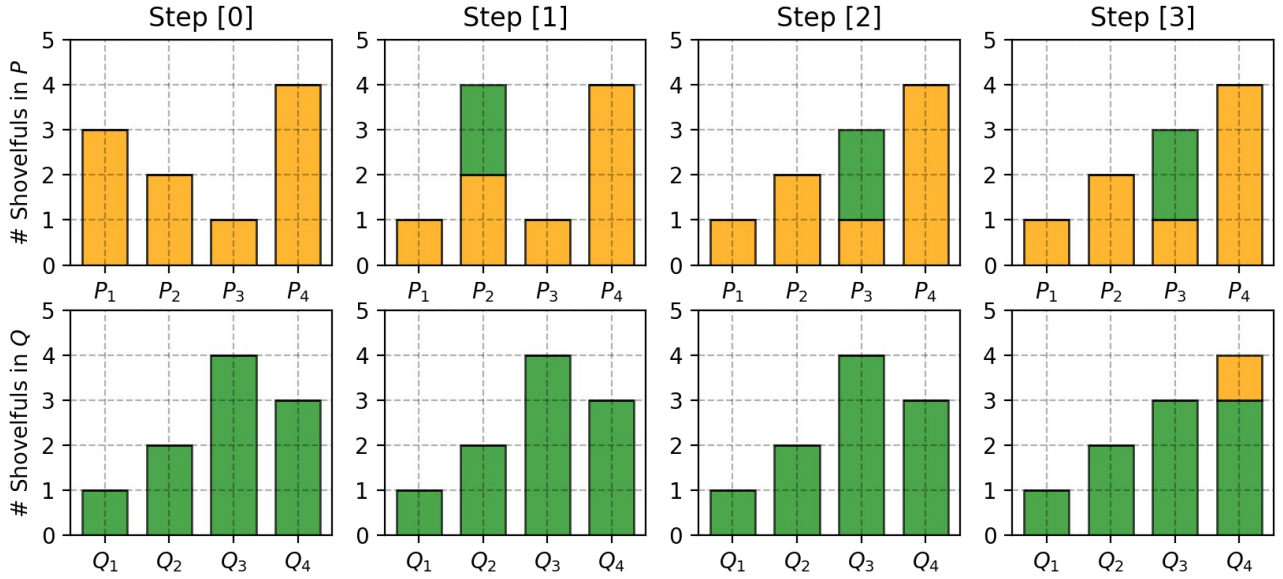
Figure 1: Illustration of different behaviors of two loss functions. (a): Decision boundaries of two loss functions. Note that the decision boundary should go across the real data distribution for a successful GANs learning. Otherwise, the learning process is saturated. (b): Decision boundary of the sigmoid cross entropy loss function. It gets very small errors for the fake samples (in magenta) for updateing G as they are on the correct side of the decision boundary. (c): Decision boundary of the least squares loss function. It penalize the fake samples (in magenta), and as a result, it forces the generator to generate samples toward decision boundary.

Mao X. et al. Least squares generative adversarial networks //Proceedings of the IEEE international conference on computer vision. – 2017. – С. 2794-2802.

Wasserstein distance

- Wasserstein (Earth Mover's) Distance is a measure of the distance between two probability distributions
- Informally it can be interpreted as the minimum energy cost of moving and transforming a pile of dirt in the shape of one probability distribution to the shape of the other distribution
- The cost is quantified by: the amount of dirt moved multiplied by the moving distance

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4 Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$



- First move 2 shovelfuls from P_1 to P_2 => (P_1, Q_1) match up.
- Then move 2 shovelfuls from P_2 to P_3 => (P_2, Q_2) match up.
- Finally move 1 shovelfuls from Q_3 to Q_4 => (P_3, Q_3) and (P_4, Q_4) match up.

$$\text{WD (P, Q)} = 2 \times \text{step(1,2)} + 2 \times \text{step(2,3)} + 1 \times \text{step(3,4)} = 5$$

From GAN to WGAN

WD Continuous

When dealing with the continuous probability domain, the distance formula becomes:

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

In the formula above, $\Pi(p_r, p_g)$ is the set of all possible joint probability distributions between p_r and p_g. One joint distribution $\gamma \in \Pi(p_r, p_g)$ describes one dirt transport plan, same as the discrete example above, but in the continuous probability space. Precisely $\gamma(x, y)$ states the percentage of dirt should be transported from point x to y so as to make x follows the same probability distribution of y. That's why the marginal distribution over x adds up to p_g, $\sum_x \gamma(x, y) = p_g(y)$ (Once we finish moving the planned amount of dirt from every possible x to the target y, we end up with exactly what y has according to p_g.) and vice versa $\sum_y \gamma(x, y) = p_r(x)$.

When treating x as the starting point and y as the destination, the total amount of dirt moved is $\gamma(x, y)$ and the travelling distance is $|x - y|$ and thus the cost is $\gamma(x, y) \cdot |x - y|$. The expected cost averaged across all the (x, y) pairs can be easily computed as:

$$\sum_{x,y} \gamma(x, y)\|x - y\| = \mathbb{E}_{x,y\sim\gamma}\|x - y\|$$

Finally, we take the minimum one among the costs of all dirt moving solutions as the EM distance.

From GAN to WGAN

WD. Toy example

$$\forall(x,y) \in P, x = 0 \text{ and } y \sim U(0,1) \forall(x,y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1)$$

When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

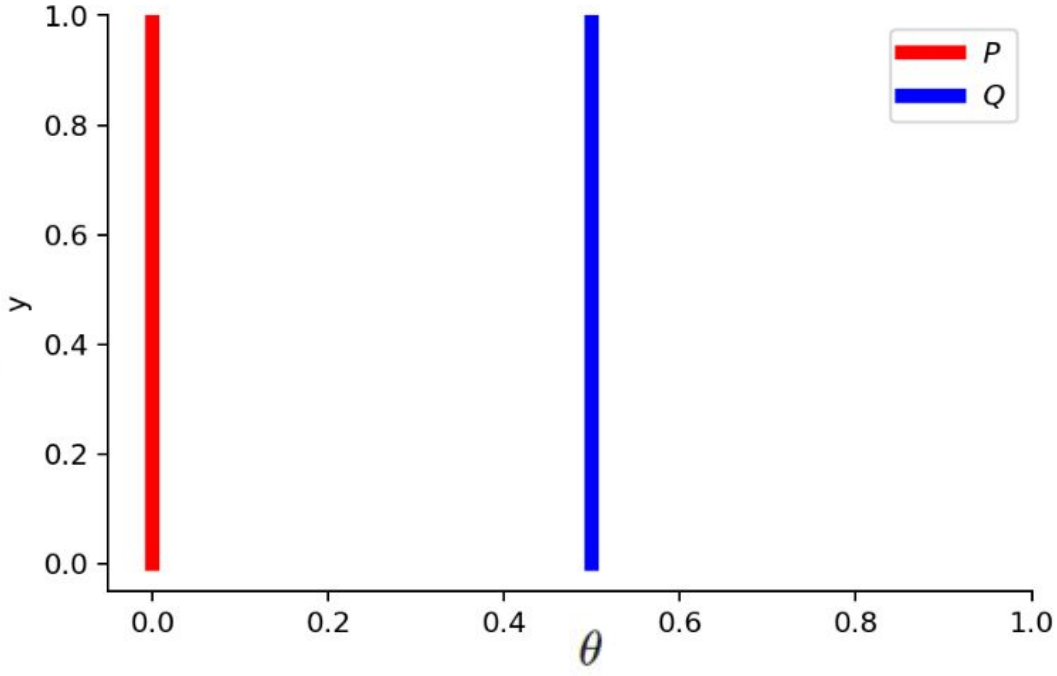
$$D_{JS}(P,Q) = \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

$$W(P,Q) = |\theta|$$

But when $\theta = 0$, two distributions are fully overlapped:

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P,Q) = 0$$

$$W(P,Q) = 0 = |\theta|$$



From GAN to WGAN

WGAN

- It is intractable to exhaust all the possible joint distributions to compute infimum (greatest lower bound)
- Arjovsky et al. proposed a transformation of the formula based on the Kantorovich-Rubinstein duality
- Now we want to measure the least upper bound
- In the modified Wasserstein-GAN, the “discriminator” is trained to learn a function to help compute Wasserstein distance.
- As the loss function decreases in the training, the Wasserstein distance gets smaller and the generator model’s output grows closer to the real data distribution.
- Our discriminator (critic) should be K-Lipschitz continuous.
- If you are interested in how to transfer Wasserstein metric into its dual form according to the Kantorovich-Rubinstein Duality, read this post.

[From GAN to WGAN]

A real-valued function $f : \mathbb{R} \to \mathbb{R}$ is called K-Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in \mathbb{R}$,

$$|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

Transformed WD:

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)]$$

WGAN Objective:

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

WGAN. Algorithm

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 $t = 0, \ldots, n_{\text{critic}}$ **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

From GAN to WGAN

WGAN VS Vanilla-GAN

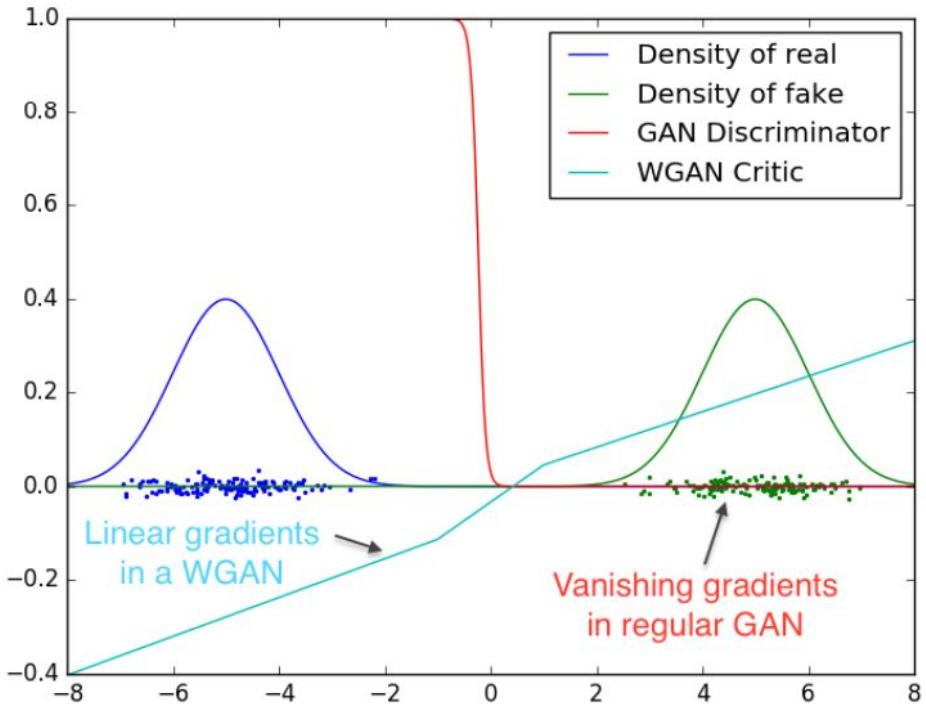
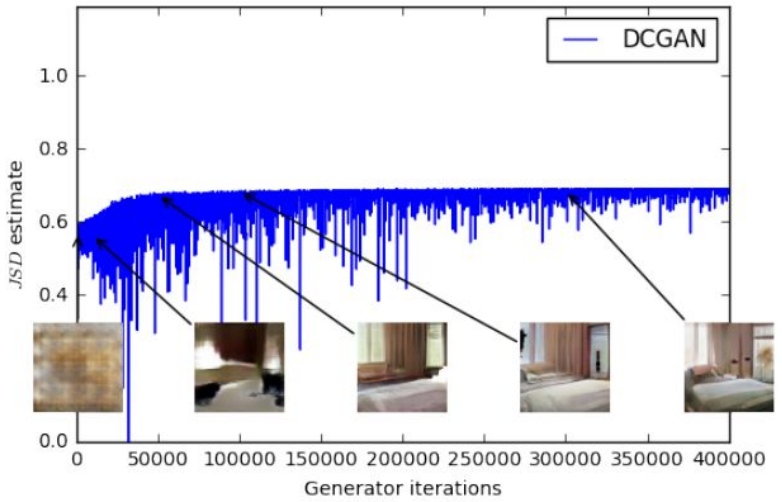


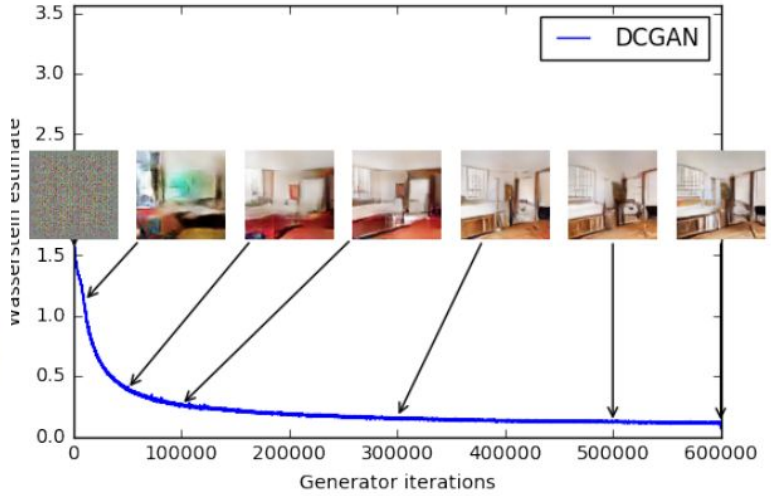
Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

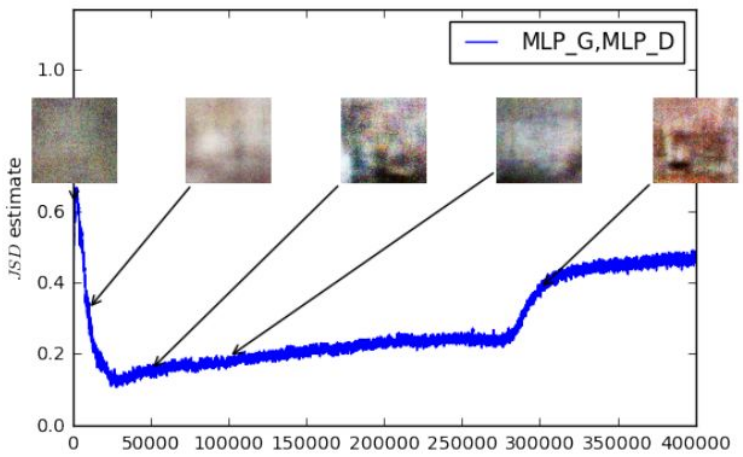
From GAN to WGAN

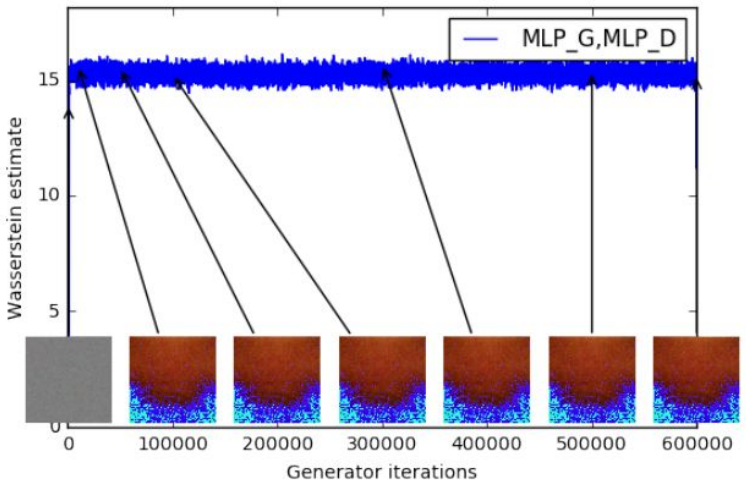
WD-loss vs JS-loss

- JS:
  - Samples get better for the DCGAN but the JS estimate increases or stays constant, pointing towards no significant correlation between sample quality and loss
  - MLP curve goes up and down regardless of sample quality
- WD:
  - Correlation between lower error and better sample quality in case of WGAN
  - Loss is constant and samples are constant as well in case of WGAN-MLP









[Wasserstein GAN]

Lipschitz-1 Condition

- Original paper used Weight Clipping:

$$w \leftarrow \text{clip}(w, -c, c)$$

- Weight clipping makes the critic less expressive and the training harder to converge
- A gradient penalty term can be added into the loss with the weigh λ :

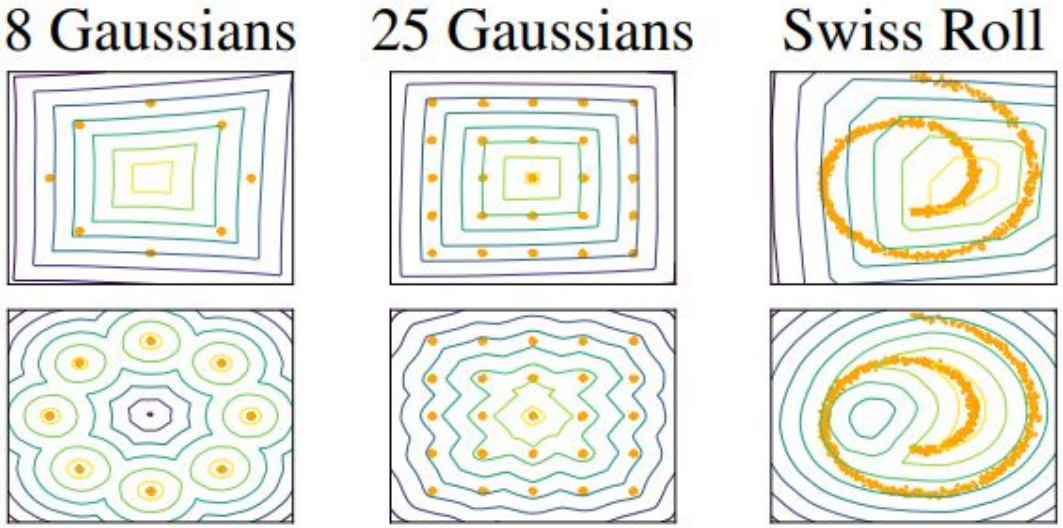
$$\mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]$$

- We can normalize weights using spectral normalization:

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

$$\sigma(A) := \max_{\boldsymbol{h}:\boldsymbol{h}\neq \boldsymbol{0}} \frac{\|A\boldsymbol{h}\|_2}{\|\boldsymbol{h}\|_2} = \max_{\|\boldsymbol{h}\|_2 \leq 1} \|A\boldsymbol{h}\|_2,$$

which is equivalent to the largest singular value of A



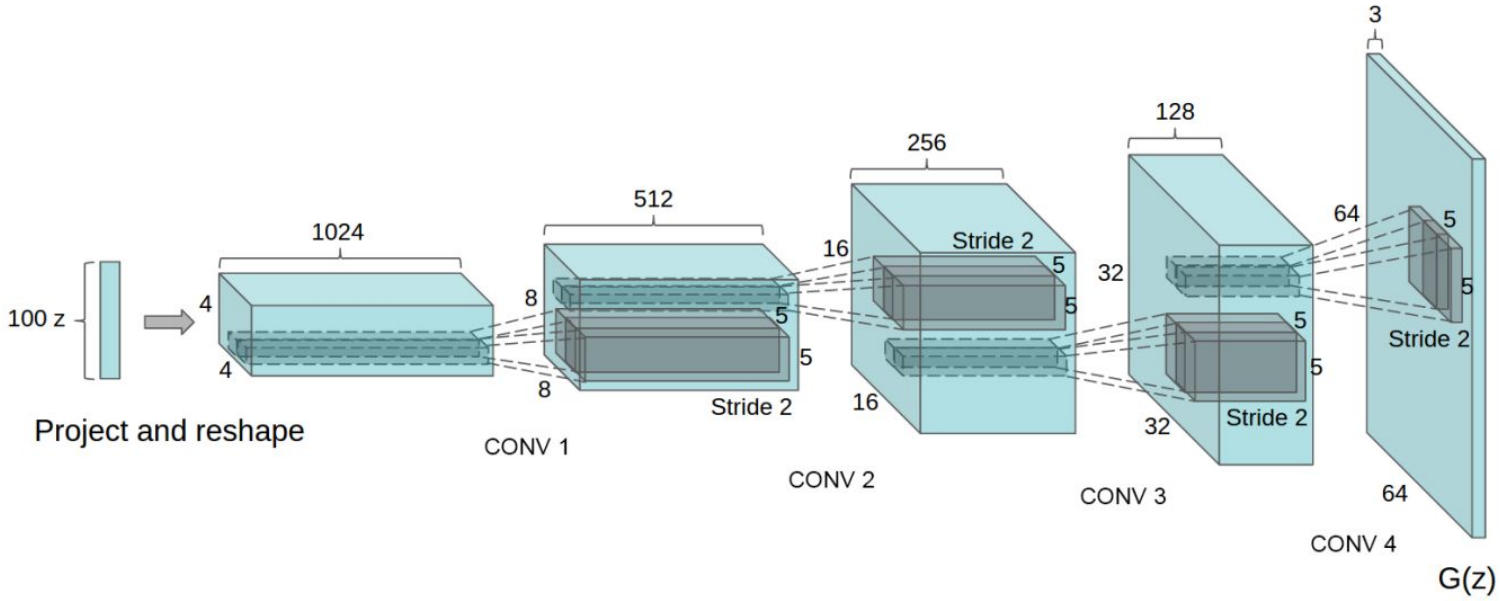
Value surfaces of WGAN critics trained to optimality on toy datasets using (top) weight clipping and (bottom) gradient penalty. Critics trained with weight clipping fail to capture higher moments of the data distribution. The 'generator' is held fixed at the real data plus Gaussian noise.

GAN Applications

DCGAN

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

[Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks]

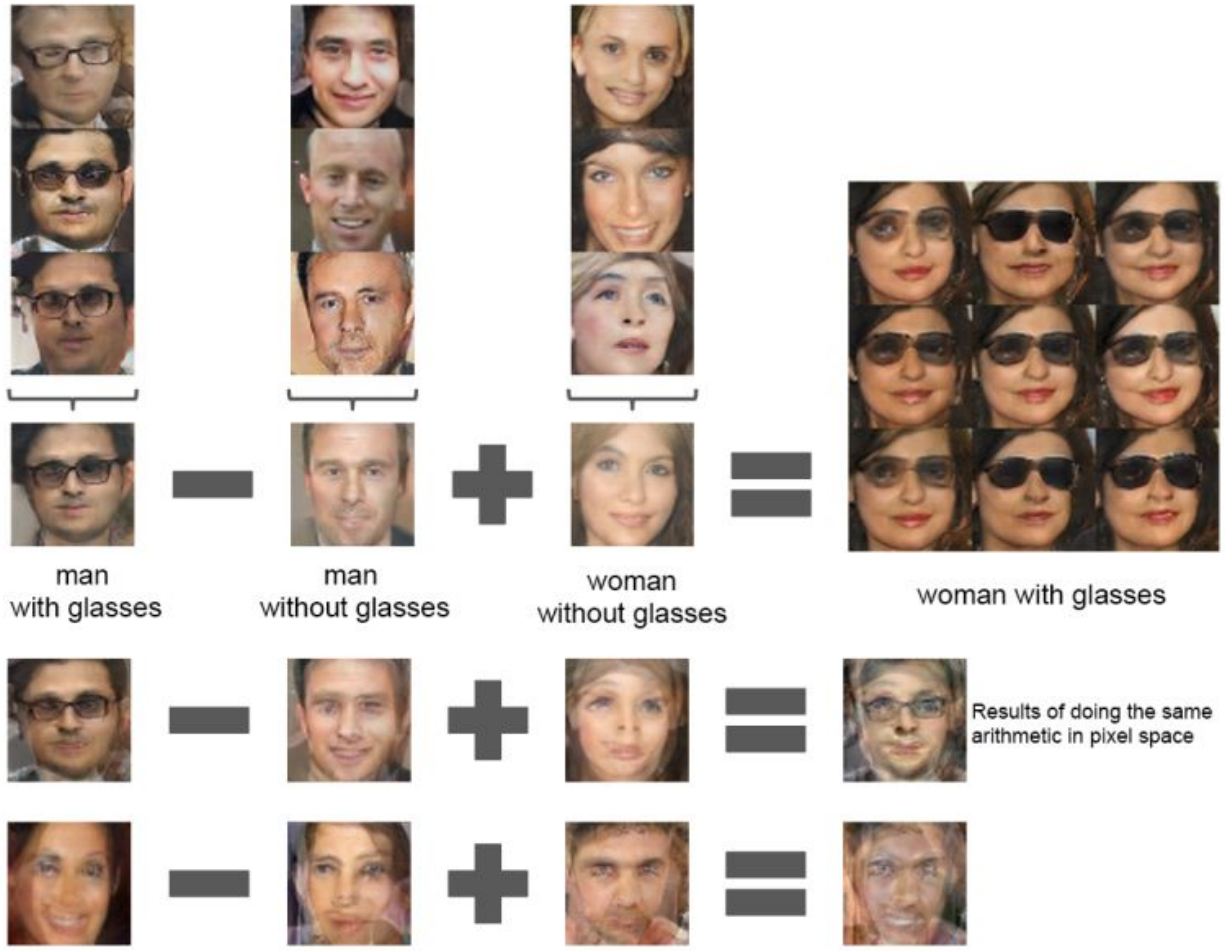


DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert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

DCGAN. Vector arithmetic

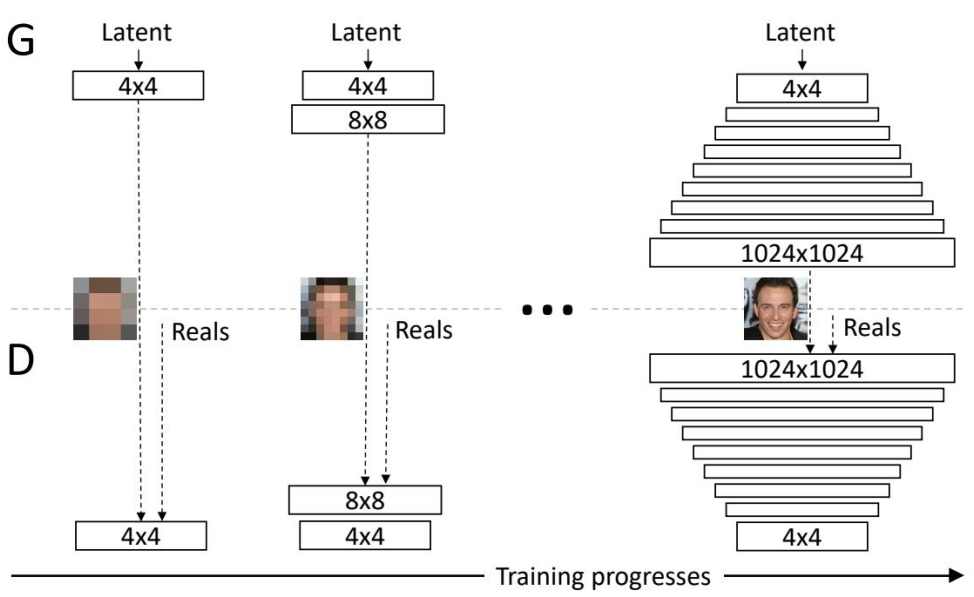
- For each column, the Z vectors of samples are averaged.
- Arithmetic was then performed on the mean vectors creating a new vector Y.
- The center sample on the right hand side is produce by feeding Y as input to the generator.
- To demonstrate the interpolation capabilities of the generator, uniform noise sampled with scale +-0.25 was added to Y to produce the 8 other samples.
- Applying arithmetic in the input space (bottom two examples) results in noisy overlap due to misalignment.

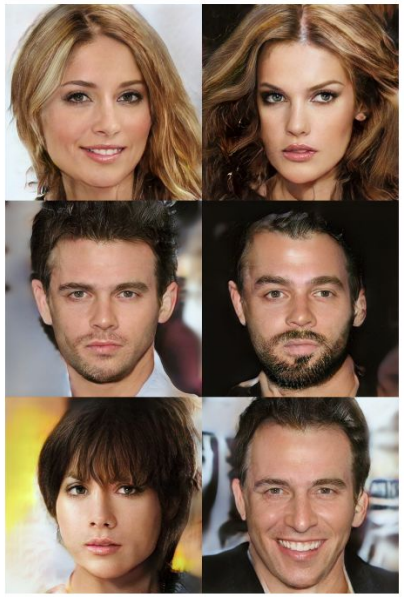
[Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks]



Progressive Growing GAN

- Grow both the generator and discriminator progressively
- Starting from a low resolution, add new layers that model increasingly fine details as training progresses:
  - Train GAN which generate NxN images
  - Increase N
  - Add upsampling layers to G, downsampling layers to D
  - Repeat
- Speeds the training up and greatly stabilizes it

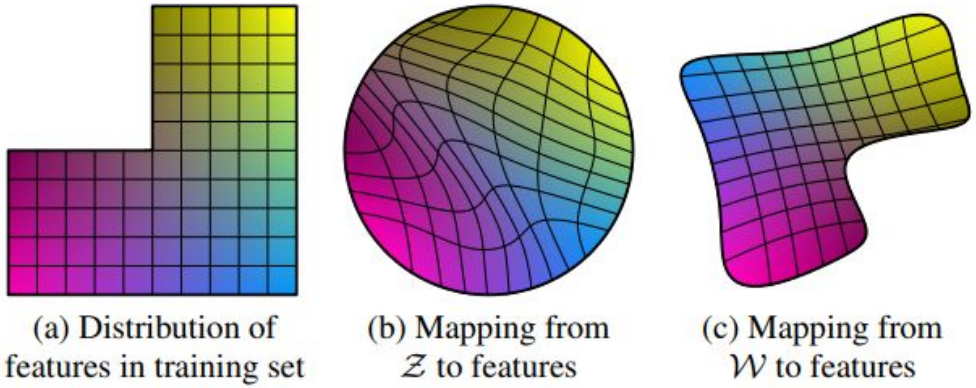




[Progressive Growing of GANs for Improved Quality, Stability, and Variation]

Mapping Network

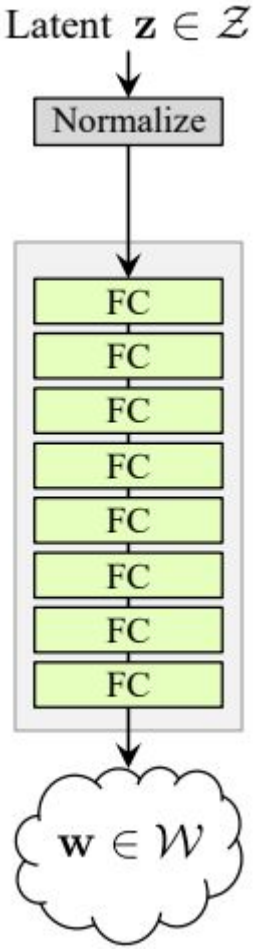
- We expect each component of input vector Z to be responsible for some particular generative factor.
- Introduce Mapping network W = f(Z) to reduce correlations between components of Z.
- (a) An example training set where some combination (e.g., long haired males) is missing.
- (b) This forces the mapping from Z to image features to become curved so that the forbidden combination disappears in Z to prevent the sampling of invalid combinations.
- (c) The learned mapping from Z to W is able to “undo” much of the warping.



(a) Distribution of features in training set

(b) Mapping from $\mathcal{Z}$ to features

(c) Mapping from $\mathcal{W}$ to features



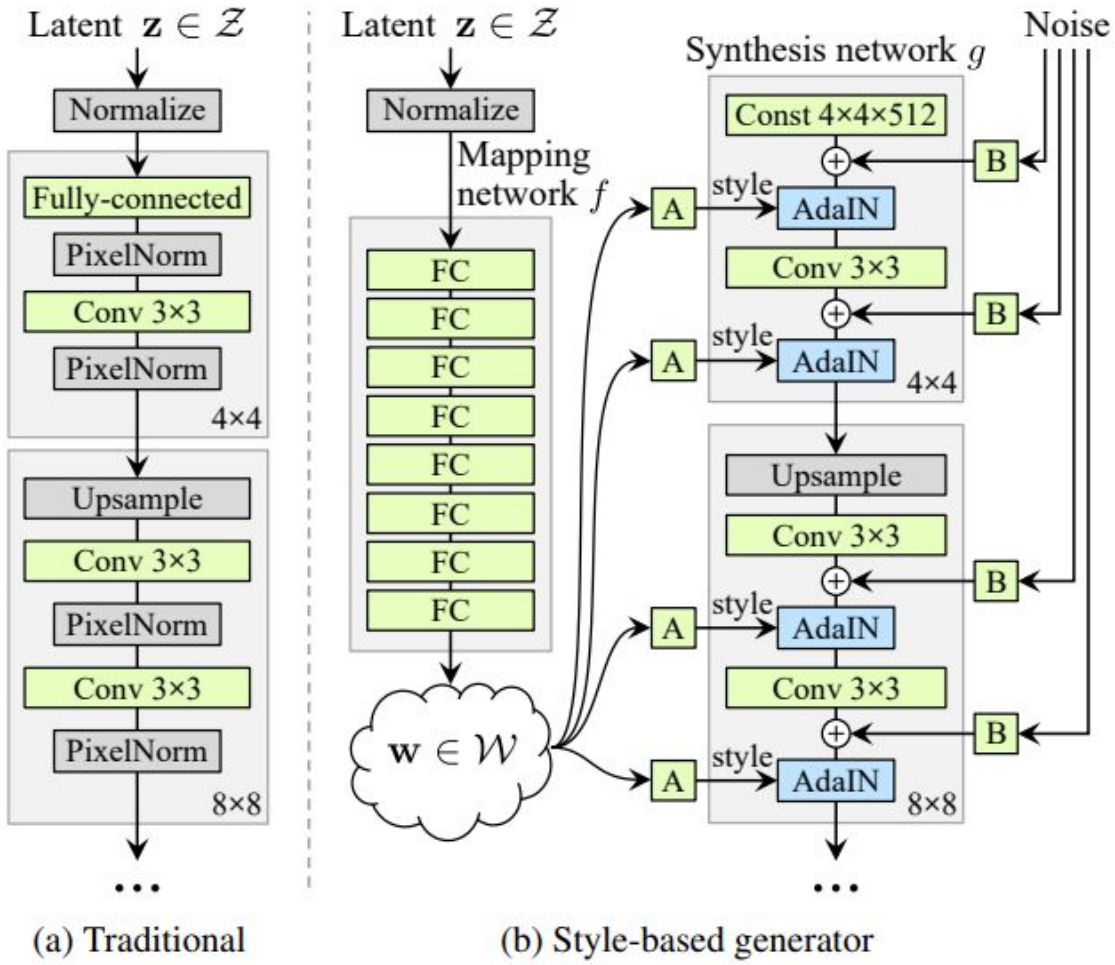
StyleGAN

- Map the input to an intermediate latent space W
- Learned affine transformations specialize W to styles y = (y_s, y_b)
- It controls the generator through adaptive instance normalization (AdaIN) at each convolution layer:

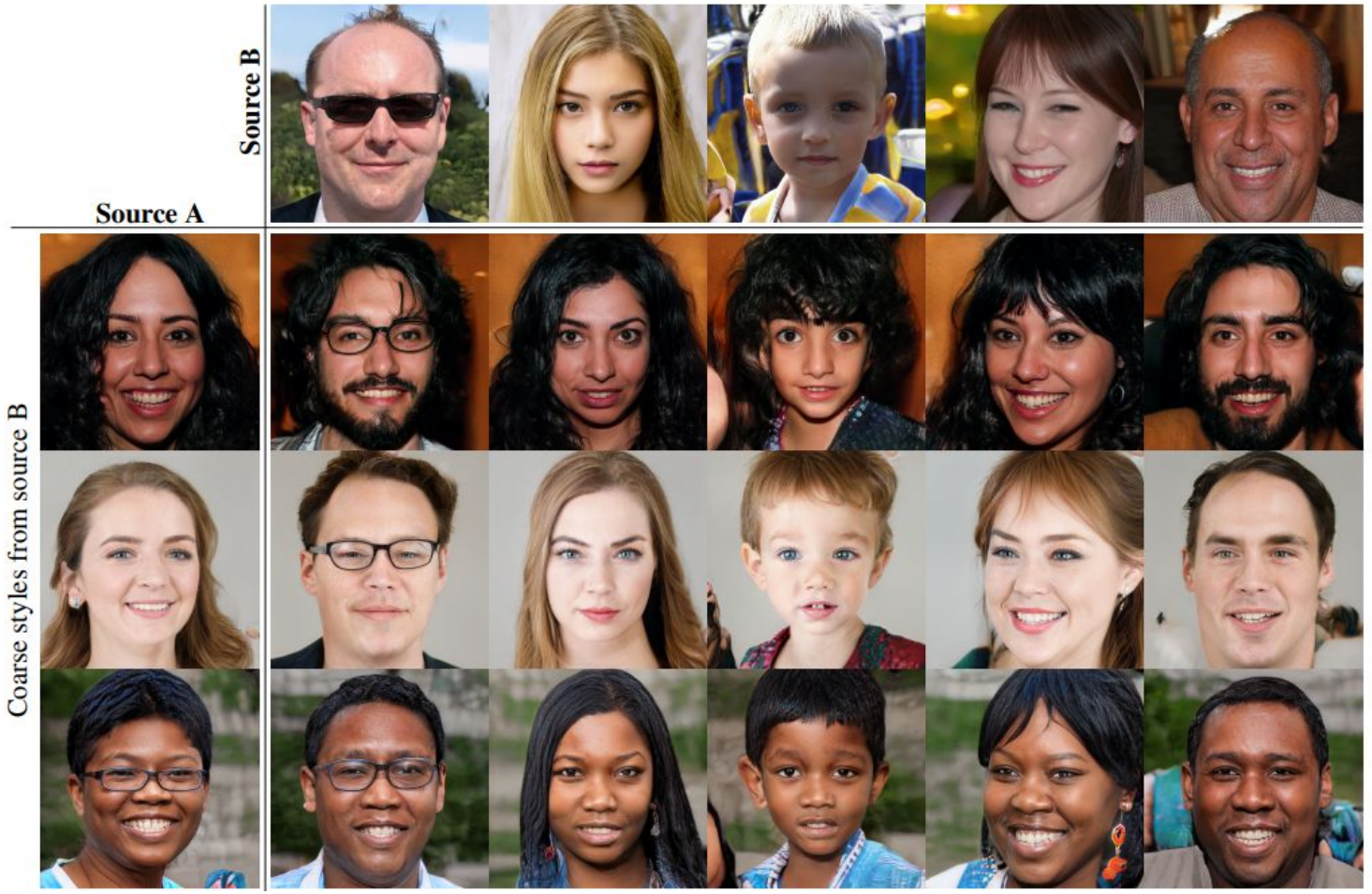
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i}\frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- Gaussian noise is added after each convolution, before evaluating the nonlinearity.
- “A” stands for a learned affine transform, and “B” applies learned per-channel scaling factors to the noise input.
- The output of the last layer is converted to RGB using a separate 1 × 1 convolution.

[A Style-Based Generator Architecture for Generative Adversarial Networks]

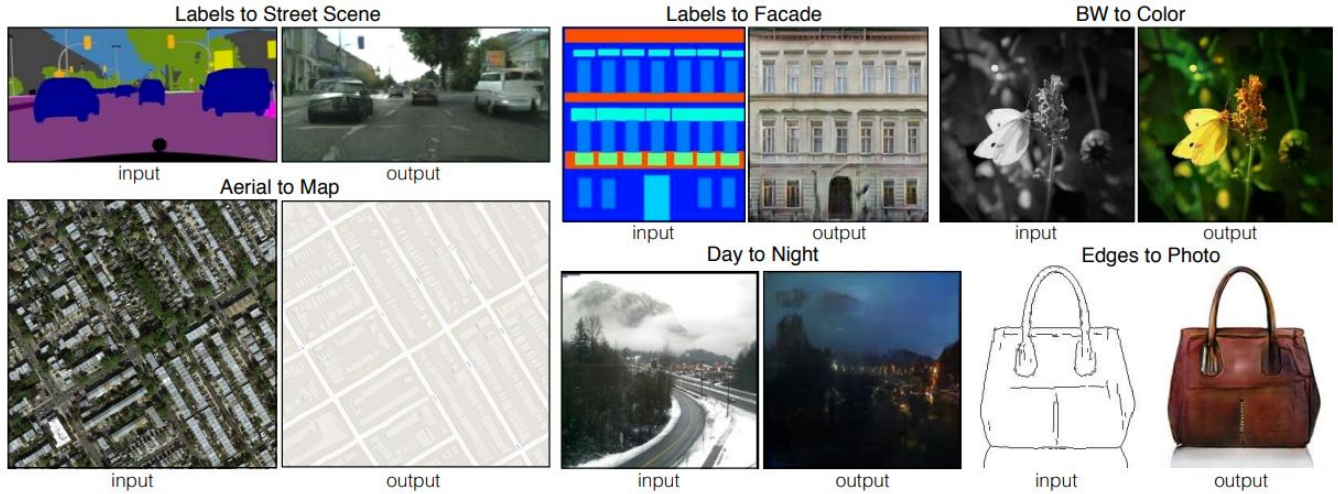


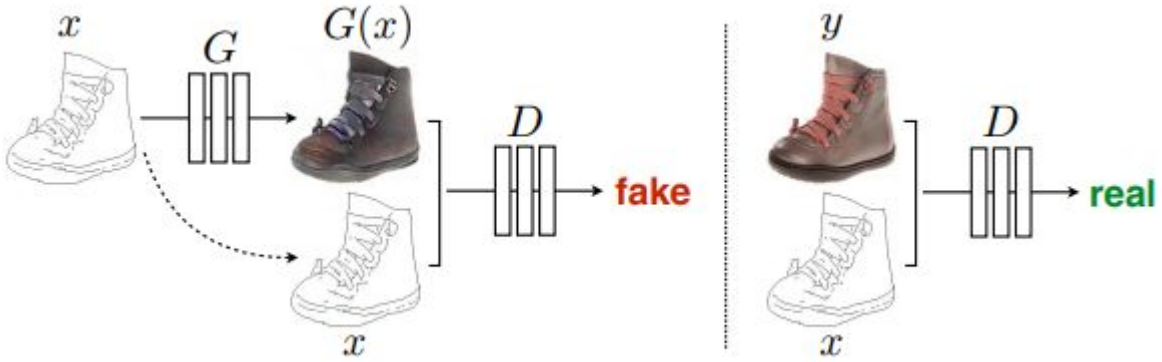
StyleGAN. Results



[A Style-Based Generator Architecture for Generative Adversarial Networks]

Image-to-Image with CGAN





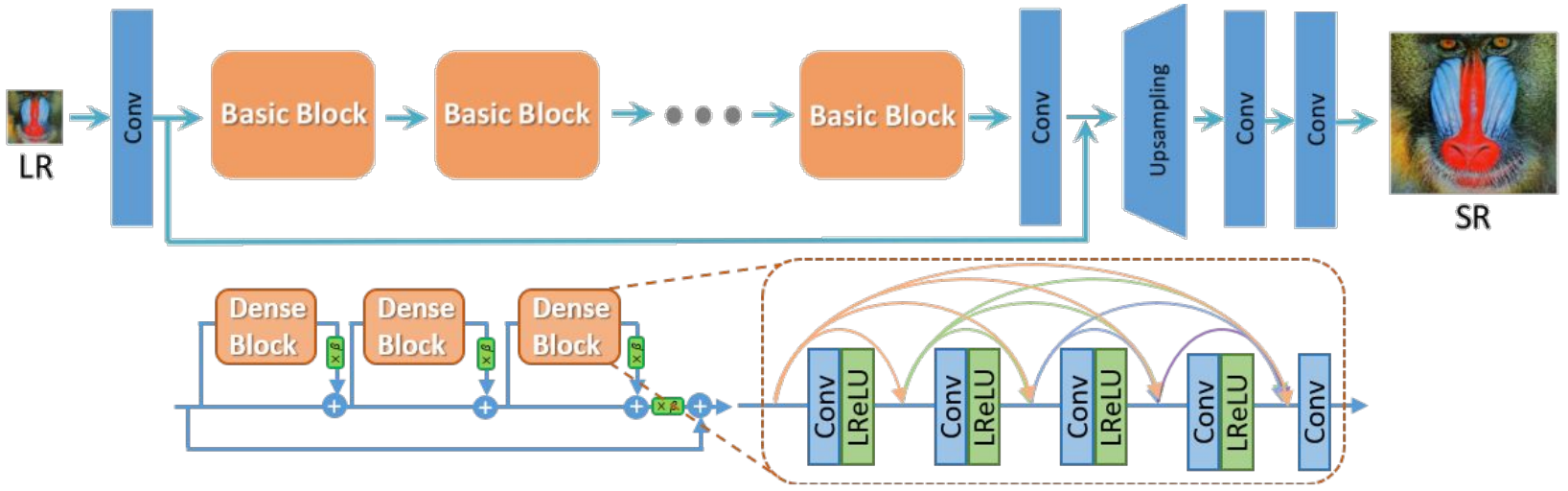
[Image-to-Image Translation with Conditional Adversarial Networks]

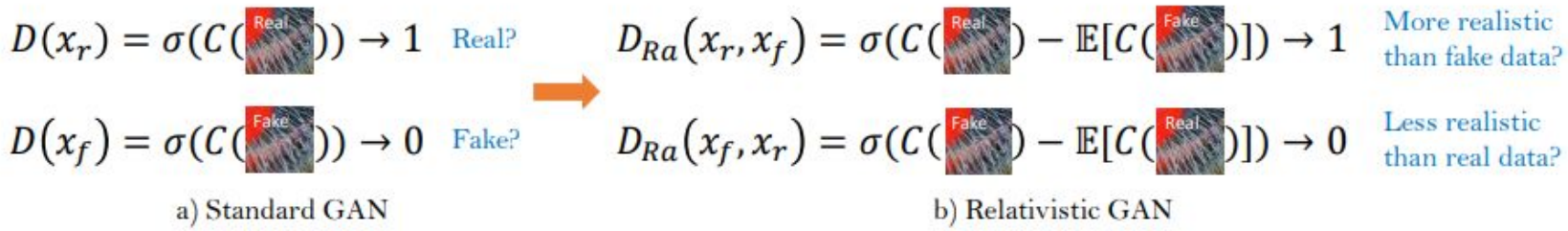
Image-to-Image with CGAN. Results

[Image-to-Image Translation with Conditional Adversarial Networks]

Super Resolution. ESRGAN

- SRResNet-based architecture with residual-in-residual blocks
- Removing BN layers has proven to increase performance and reduce computational complexity in different PSNR-oriented tasks including SR
- Mixture of context, perceptual, and adversarial losses
- Context and perceptual losses are used for proper image upscaling
- Adversarial loss pushes neural network to the natural image manifold using a discriminator network that is trained to differentiate between the super-resolved images and original photo-realistic images
- Different from the standard discriminator D in SRGAN, which estimates the probability that one input image x is real and natural, a relativistic discriminator tries to predict the probability that a real image is relatively more realistic than a fake one





[ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks]

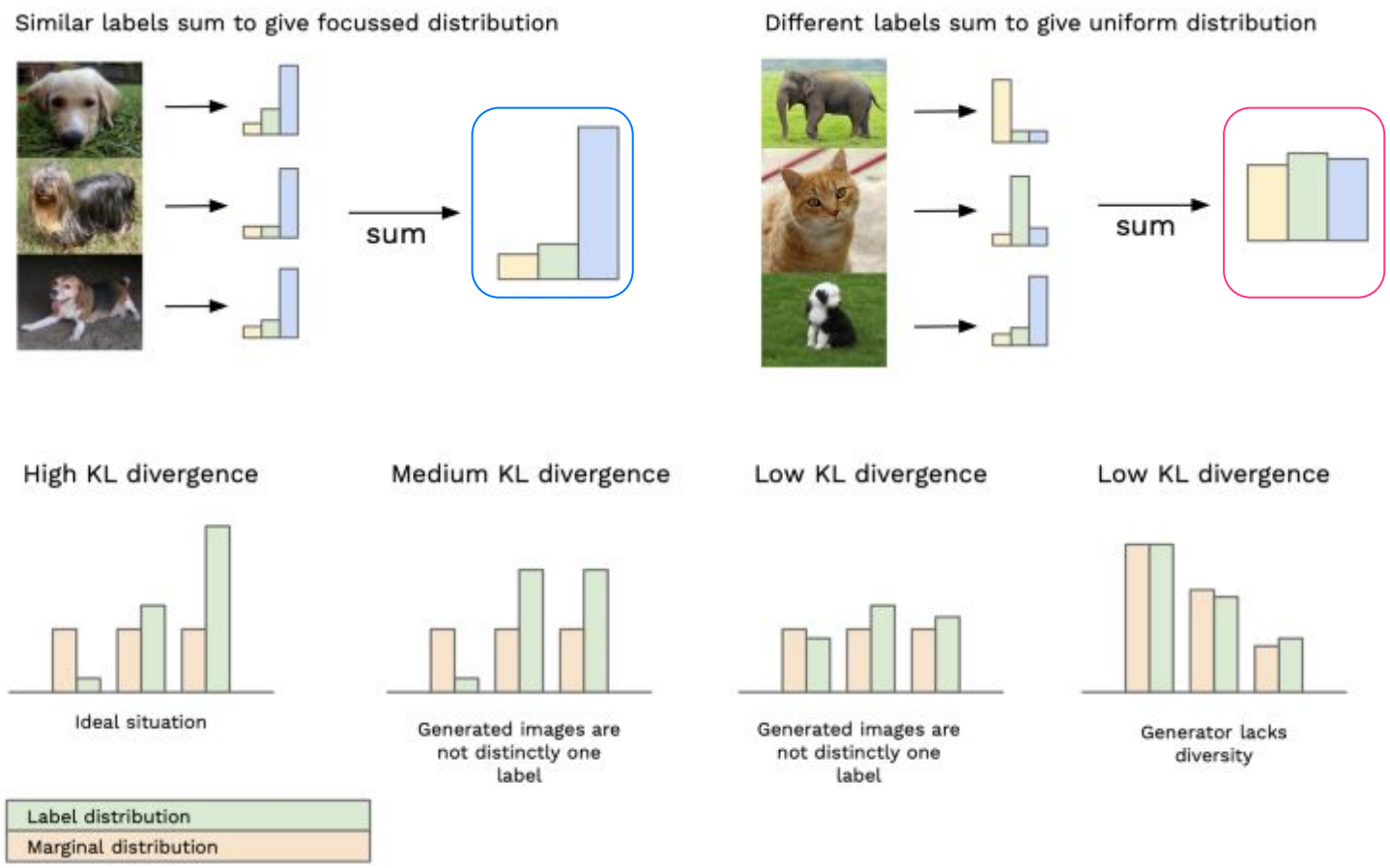
ESRGAN Results

[ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks]

GAN Evaluation

Inception Score

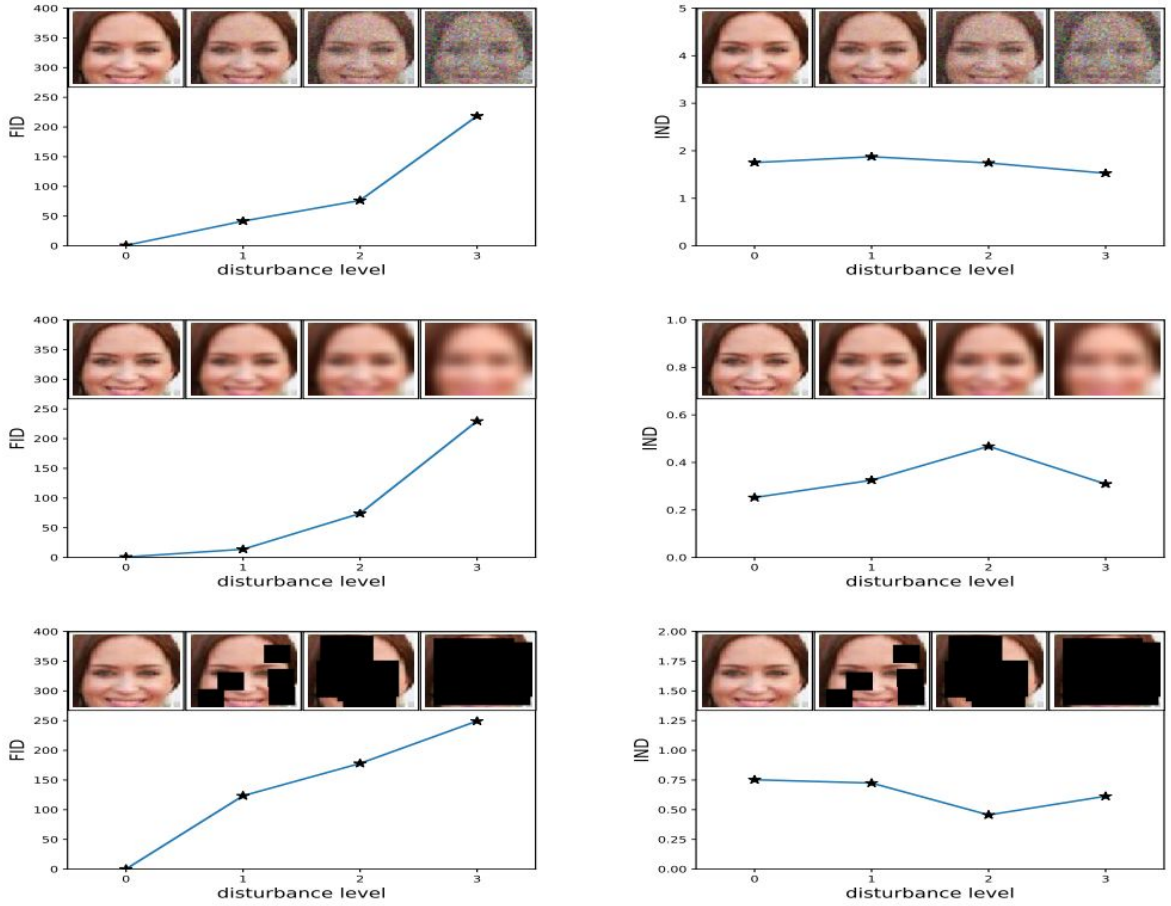
- Correlates well with human evaluation of image quality
- The score measures two things simultaneously:
  - The images have variety (diversity)
  - Each image distinctly looks like something (sharpness)
- Summing the label distributions of images, we create a new label distribution, the marginal distribution
- The marginal distribution tells us how much variety there is in our generator's output
- $\text{IS} = \exp(\mathbb{E}_{\boldsymbol{x}} \text{KL}(p(y|\boldsymbol{x}) \| p(y)))$



[Improved Techniques for Training GANs]

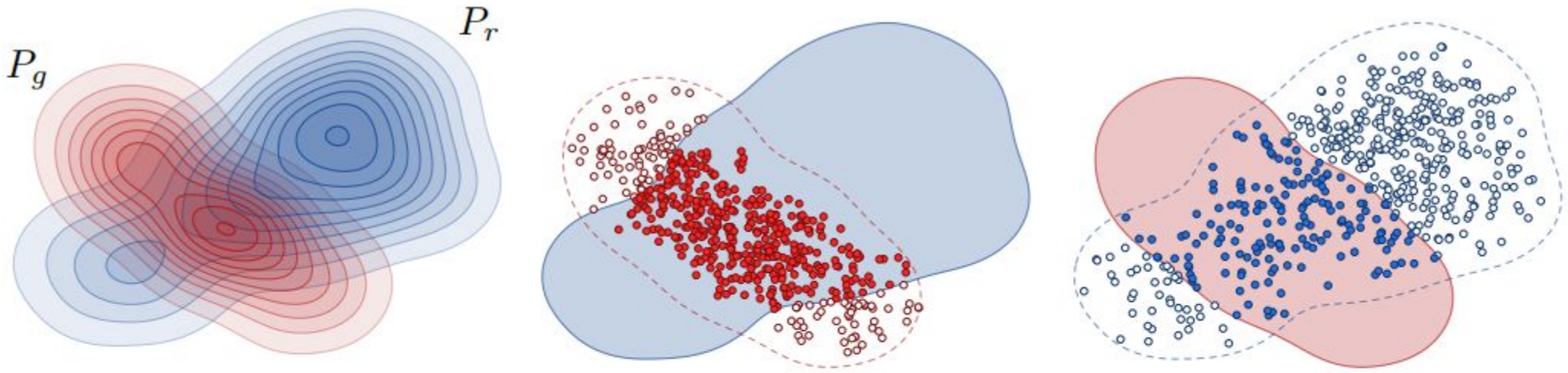
Frechet Inception Distance

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2})$$



[GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium]

Precision-Recall

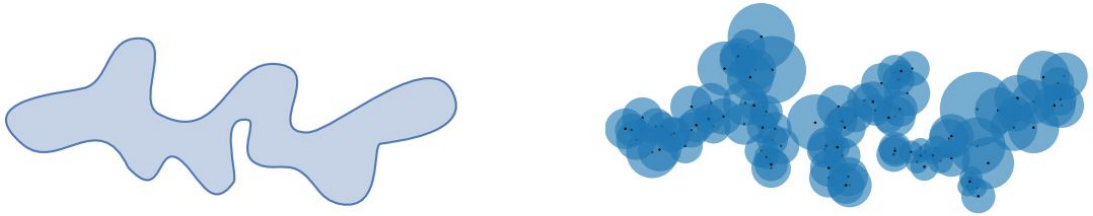


(a) Example distributions (b) Precision (c) Recall

Figure 1: Definition of precision and recall for distributions [25]. (a) Denote the distribution of real images with P_r (blue) and the distribution of generated images with P_g (red). (b) Precision is the probability that a random image from P_g falls within the support of P_r. (c) Recall is the probability that a random image from P_r falls within the support of P_g.

[Improved Precision and Recall Metric for Assessing Generative Models]

Precision-Recall



(a) True manifold (b) Approx. manifold

Figure 2: (a) An example manifold in a feature space. (b) Estimate of the manifold obtained by sampling a set of points and surrounding each with a hypersphere that reaches its kth nearest neighbor.

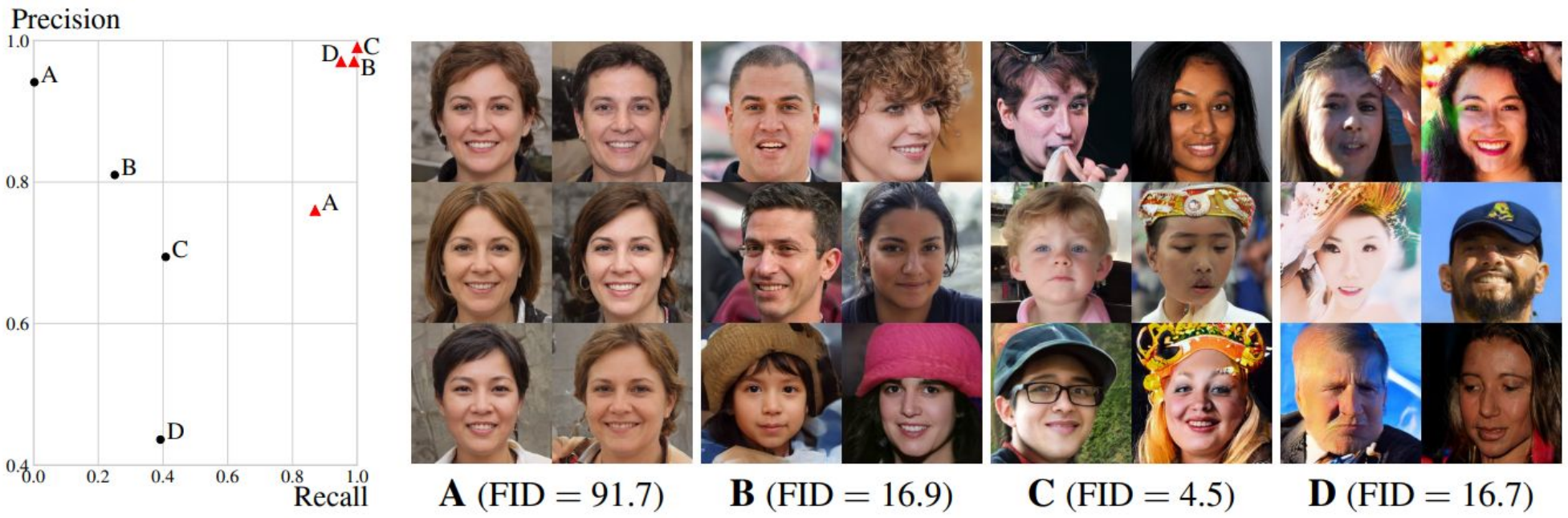
$$f(\phi, \mathbf{\Phi}) = \begin{cases} 1, \text{ if } \left\|\phi - \phi'\right\|_2 \leq \left\|\phi' - \mathrm{NN}_k\left(\phi', \mathbf{\Phi}\right)\right\|_2 \text{ for at least one } \phi' \in \mathbf{\Phi} \\ 0, \text{ otherwise,} \end{cases} \tag{1}$$

where $\mathrm{NN}_k\left(\phi', \mathbf{\Phi}\right)$ returns kth nearest feature vector of ϕ' from set $\mathbf{\Phi}$. In essence, $f(\phi, \mathbf{\Phi}_r)$ provides a way to determine whether a given image looks realistic, whereas $f(\phi, \mathbf{\Phi}_g)$ provides a way to determine whether it could be reproduced by the generator. We can now define our metric as

$$\mathrm{precision}(\mathbf{\Phi}_r, \mathbf{\Phi}_g) = \frac{1}{|\mathbf{\Phi}_g|} \sum_{\phi_g \in \mathbf{\Phi}_g} f(\phi_g, \mathbf{\Phi}_r) \quad \mathrm{recall}(\mathbf{\Phi}_r, \mathbf{\Phi}_g) = \frac{1}{|\mathbf{\Phi}_r|} \sum_{\phi_r \in \mathbf{\Phi}_r} f(\phi_r, \mathbf{\Phi}_g) \tag{2}$$

[Improved Precision and Recall Metric for Assessing Generative Models]

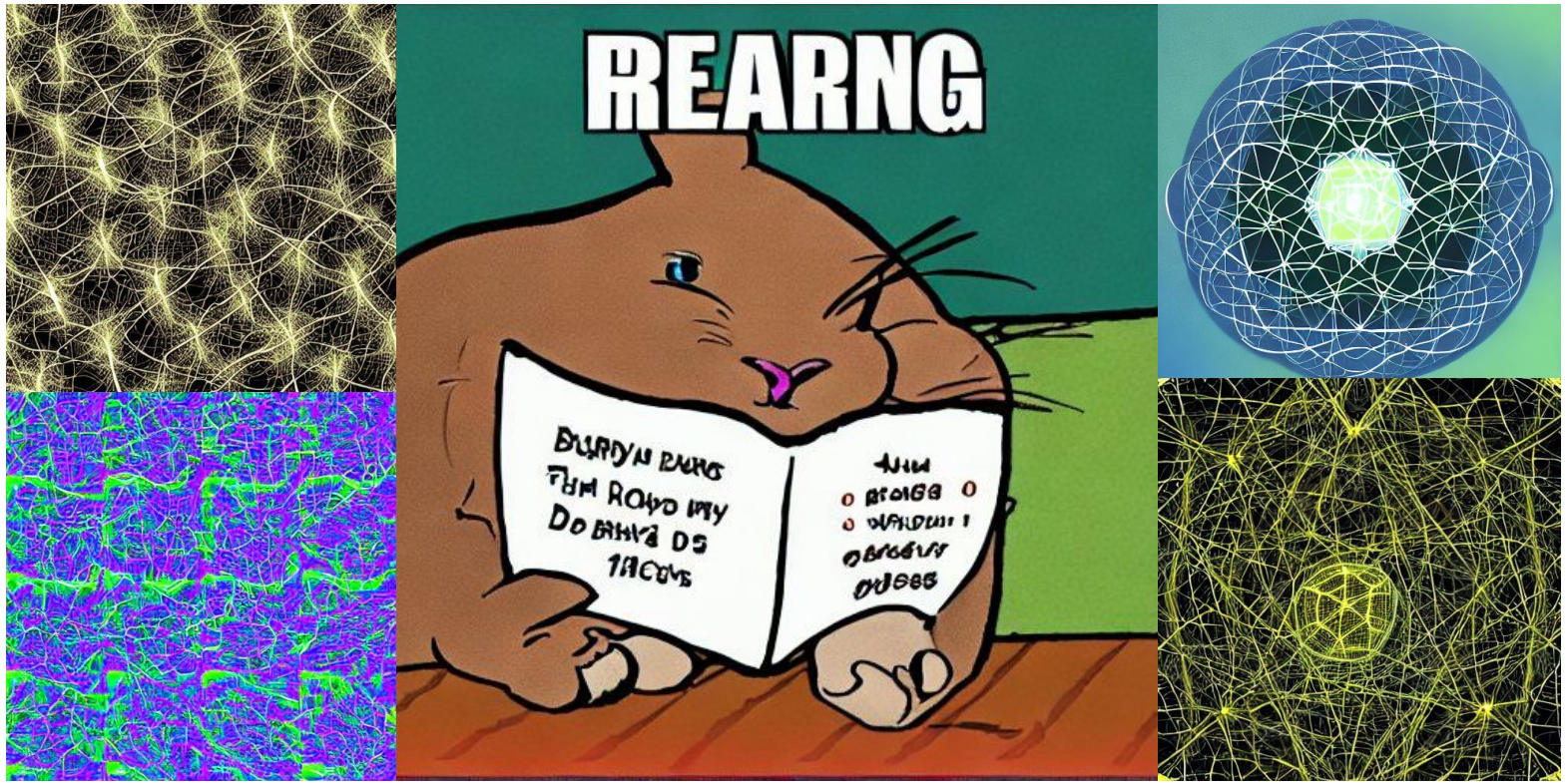
Precision-Recall VS FID



[Improved Precision and Recall Metric for Assessing Generative Models]

Further reading

- [Pros and Cons of GAN Evaluation Measures]
- [Pros and Cons of GAN Evaluation Measures: New Developments]
- [Large Scale GAN Training for High Fidelity Natural Image Synthesis]
- [Why Spectral Normalization Stabilizes GANs: Analysis and Improvements]
- [Alias-Free Generative Adversarial Networks (StyleGAN3)]
- [Towards Real-World Blind Face Restoration with Generative Facial Prior]
- [StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery]



linkedin.com/in/rogachevai

a.rogachev@corp.mail.ru

t.me/airogachev