курс «Глубокое обучение»

Генеративные состязательные сети

Александр Дьяконов

21 ноября 2022 года

Генеративные модели

Автокодировщики

Pixel CNN / Pixel RNN

Потоки

Генератор и дискриминатор – Adversarial idea

Настройка GAN, советы по настройке GAN

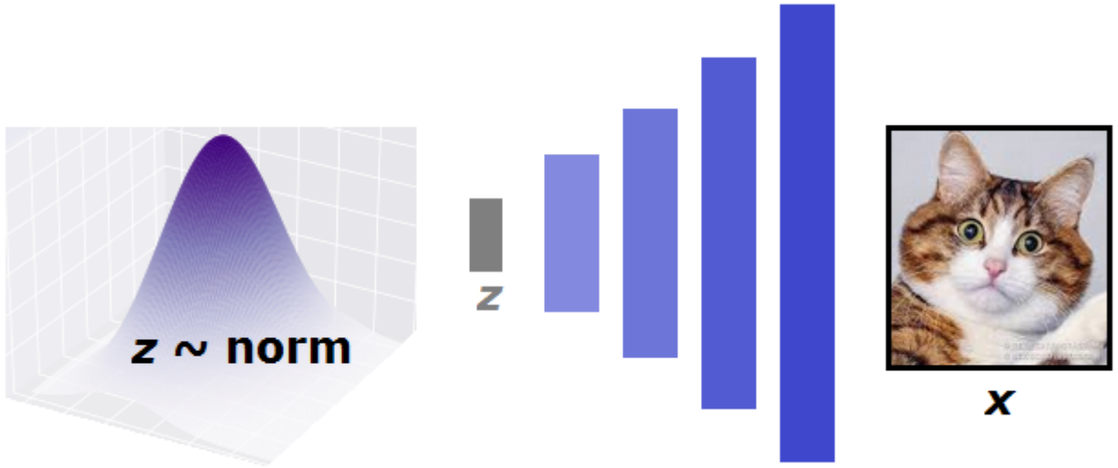
Нестабильность «non-saturating game», Mode-Collapse

Least Square GAN (LSGAN)

Wasserstein GAN (WGAN)

WGAN-GP

Генеративная модель

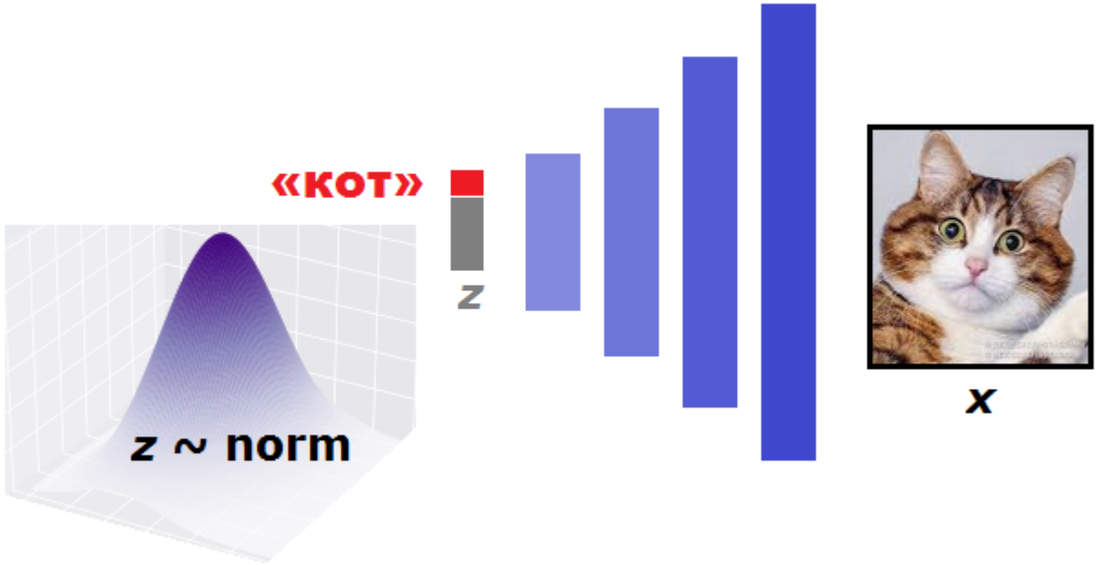


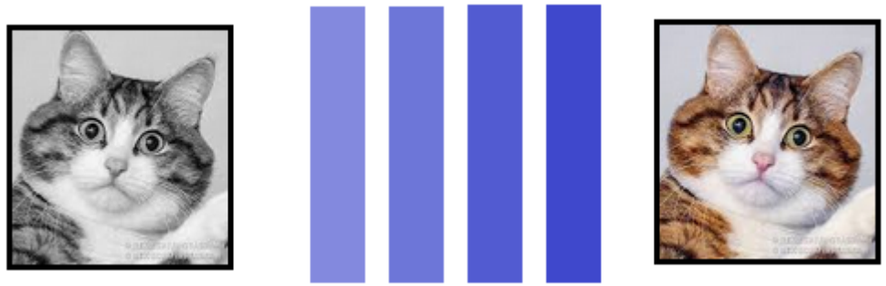
Проблема – как сделать отображение «шум → что-то разумное»

дискриминативная постановка понятнее: различаем несколько категорий

генеративная – порождение!

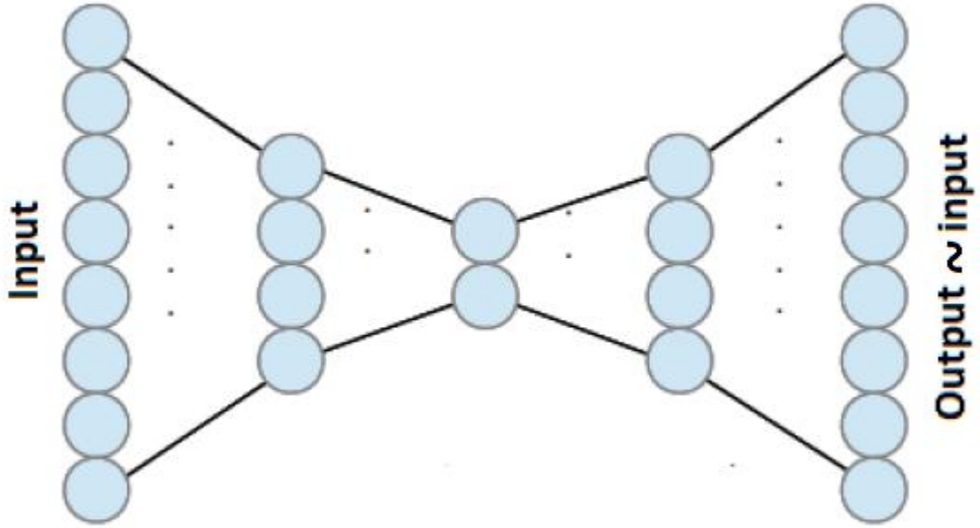
Условная генеративная модель (Conditional Generative Model)



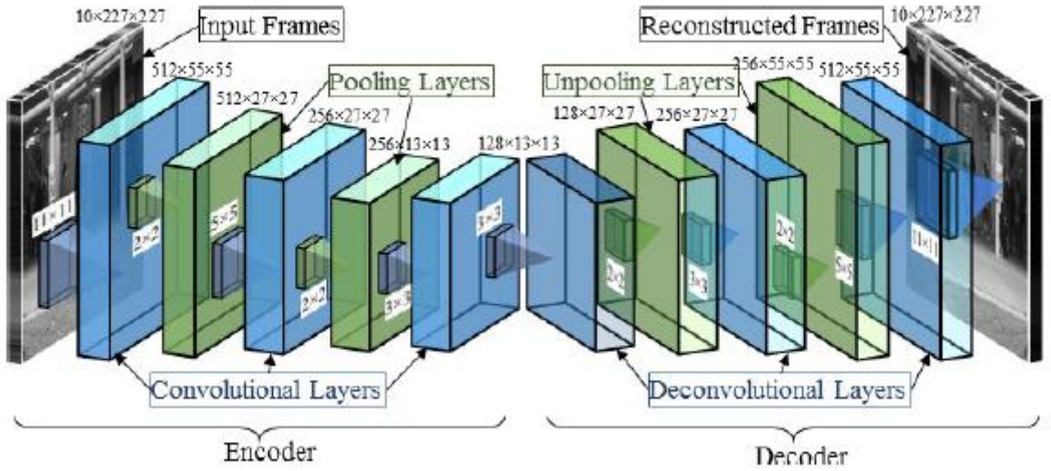


Автокодировщики (Auto-encoders) / понижающие (undercomplete)

Полносвязный с узким горлом (bottleneck)



Свёрточный

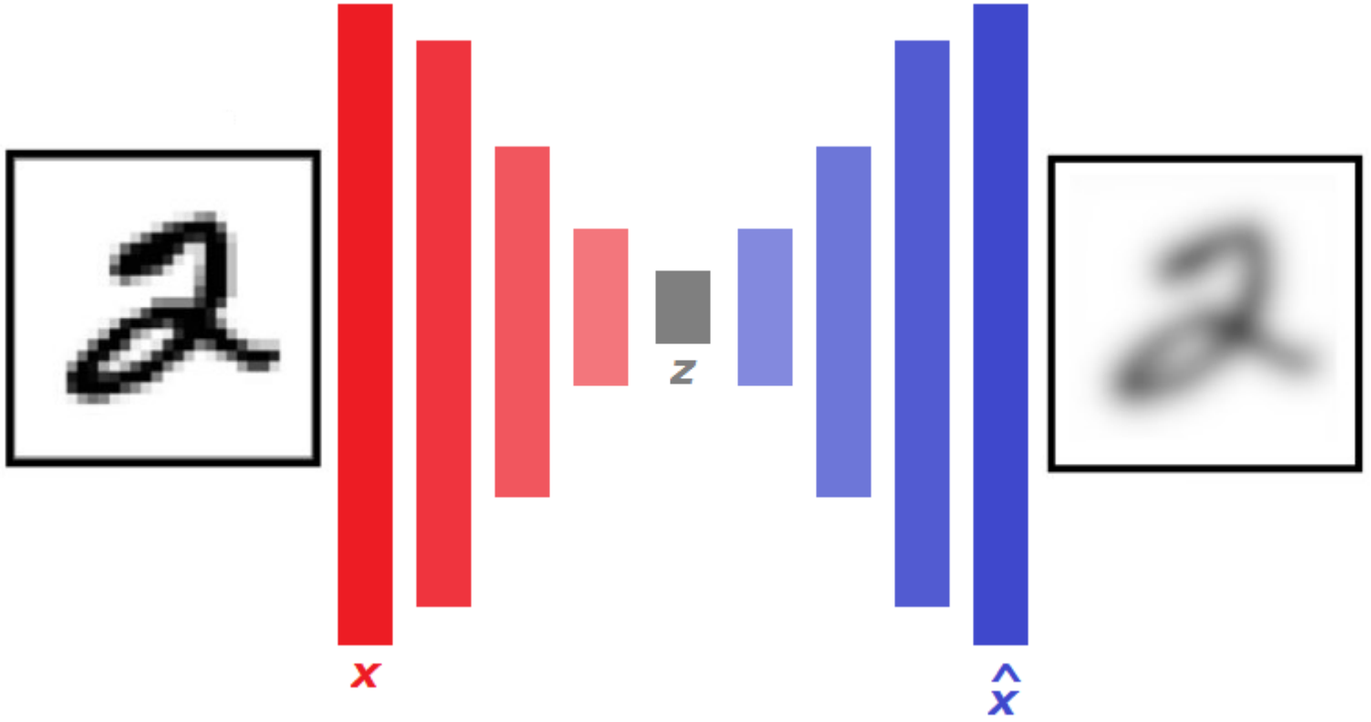


Знакомая архитектура «кодировщик $p_{\text{enc}}(z \mid x)$ – декодировщик $p_{\text{dec}}(x \mid z)$»

encoder-bottleneck-decoder // reconstruction loss

сжимаем информацию в латентное пространство,
пытаемся её восстановить

Глубокие автокодировщики – обучение



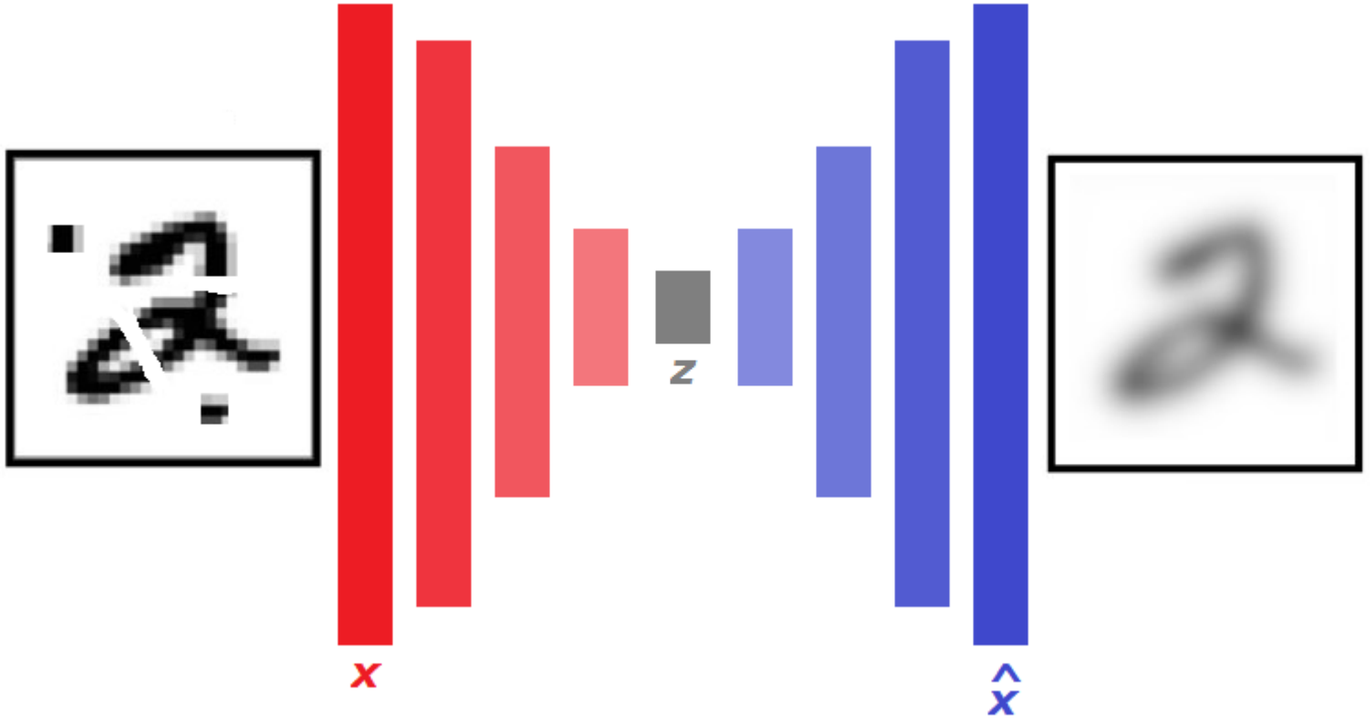
воспроизводим вход ~ reconstruction error / loss + regularizer

Минутка кода: свёрточный автокодировщик

```
import torch.nn as nn
import torch.nn.functional as F

class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1) # conv layer (depth from 1 --> 16), 3x3 kernels
        self.conv2 = nn.Conv2d(16, 4, 3, padding=1) # conv layer (depth from 16 --> 4), 3x3 kernels
        self.pool = nn.MaxPool2d(2, 2) # pooling layer to reduce x-y dims by two; kernel and stride of 2
        self.t_conv1 = nn.ConvTranspose2d(4, 16, 2, stride=2) # a kernel of 2 and a stride of 2 will
        self.t_conv2 = nn.ConvTranspose2d(16, 1, 2, stride=2) # increase the spatial dims by 2

    def forward(self, x):
        ## encoder ##
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)  # compressed representation
        ## decoder ##
        x = F.relu(self.t_conv1(x))
        x = F.sigmoid(self.t_conv2(x)) # output layer (with sigmoid for scaling from 0 to 1)

        return x

model = ConvAutoencoder()
```

https://github.com/udacity/deep-learning-v2-pytorch/blob/master/autoencoder/convolutional-autoencoder/Convolutional_Autoencoder_Solution.ipynb

Denoising Autoencoder



увеличение выборки с помощью дополнения к ней зашумлённых изображений (зашумления разного типа)

$$\| x - g(f(\tilde{x})) \| + R(f, g) \to \min$$

- **больше данных**
- **правильнее формируемые признаки**

Выученные фильтры «шумоудоляющего» автокодировщика

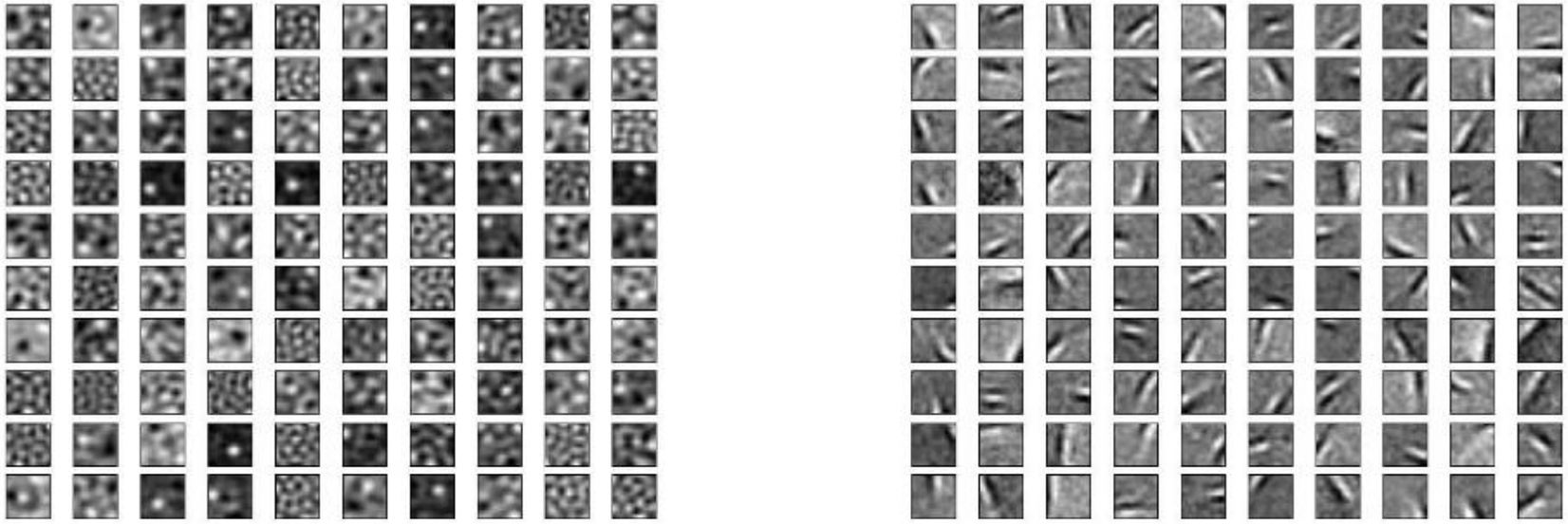
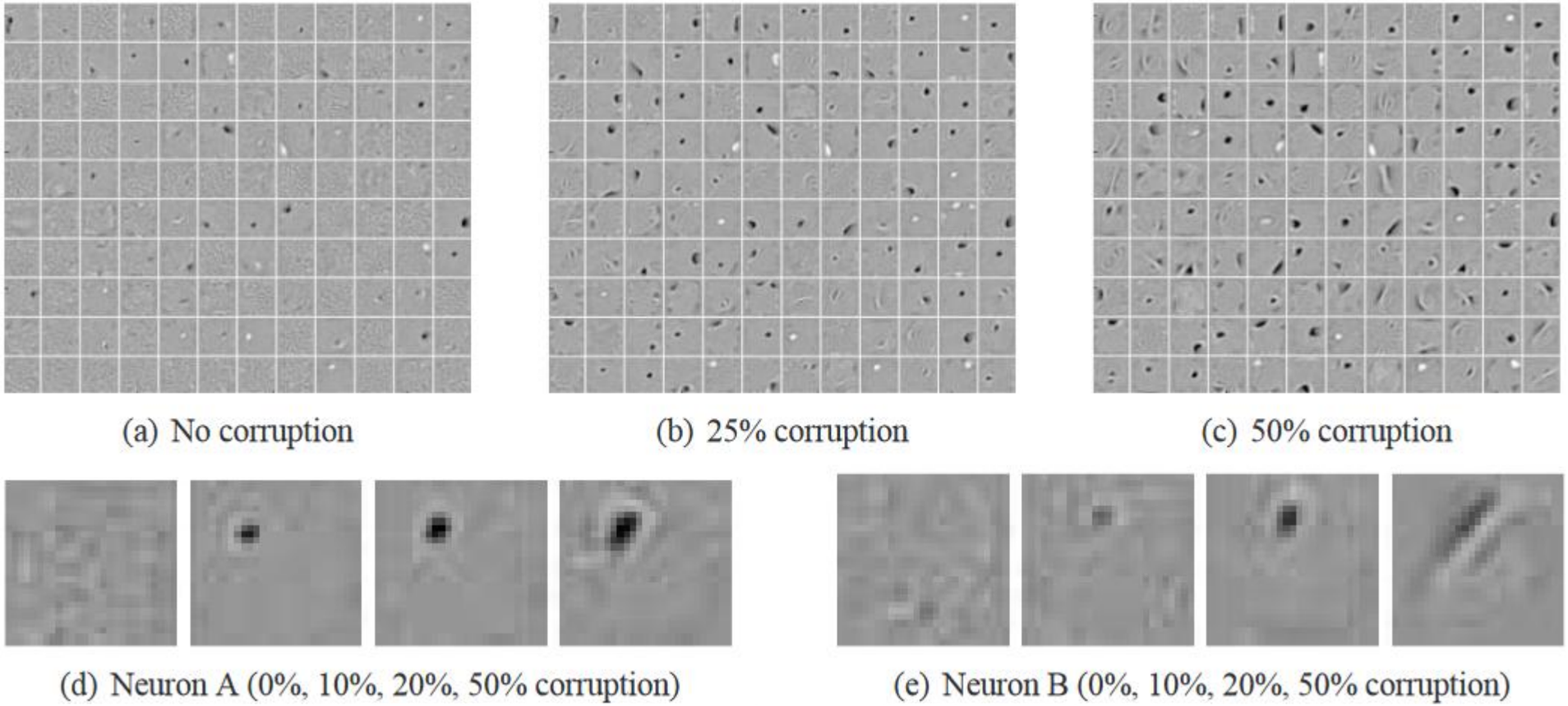


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

https://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf

Выученные фильтры «шумоудоляющего» автокодировщика



(a) No corruption (b) 25% corruption (c) 50% corruption

(d) Neuron A (0%, 10%, 20%, 50% corruption) (e) Neuron B (0%, 10%, 20%, 50% corruption)

Figure 8: Filters learnt by denoising autoencoder on MNIST digits, using zero-masking noise. *(a-c)* show some of the filters learnt by denoising autoencoders trained with various corruption levels ν. Filters at the same position in the three images are related only by the fact that the autoencoders were started from the same random initialization point in parameter space. *(d)* and *(e)* zoom in on the filters obtained for two of the neurons. As can be seen, with no noise, many filters remain similarly uninteresting (undistinctive almost uniform random grey patches). As we increase the noise level, denoising training forces the filters to differentiate more, and capture more distinctive features. Higher noise levels tend to induce less local filters, as expected. One can distinguish different kinds of filters, from local blob detectors, to stroke detectors, and character parts detectors at the higher noise levels.

Сокращающие автокодировщики – Contractive Autoencoders (CAE)

идея – сделать представления менее чувствительными к небольшим изменениям данных

$\Rightarrow$ **производные по входу должны быть** ≈ 0

А «denoising autoencoders» была устойчивость к зашумлению!

$$\| x - g(f(x)) \| + \lambda \| J(x) \|_F^2$$

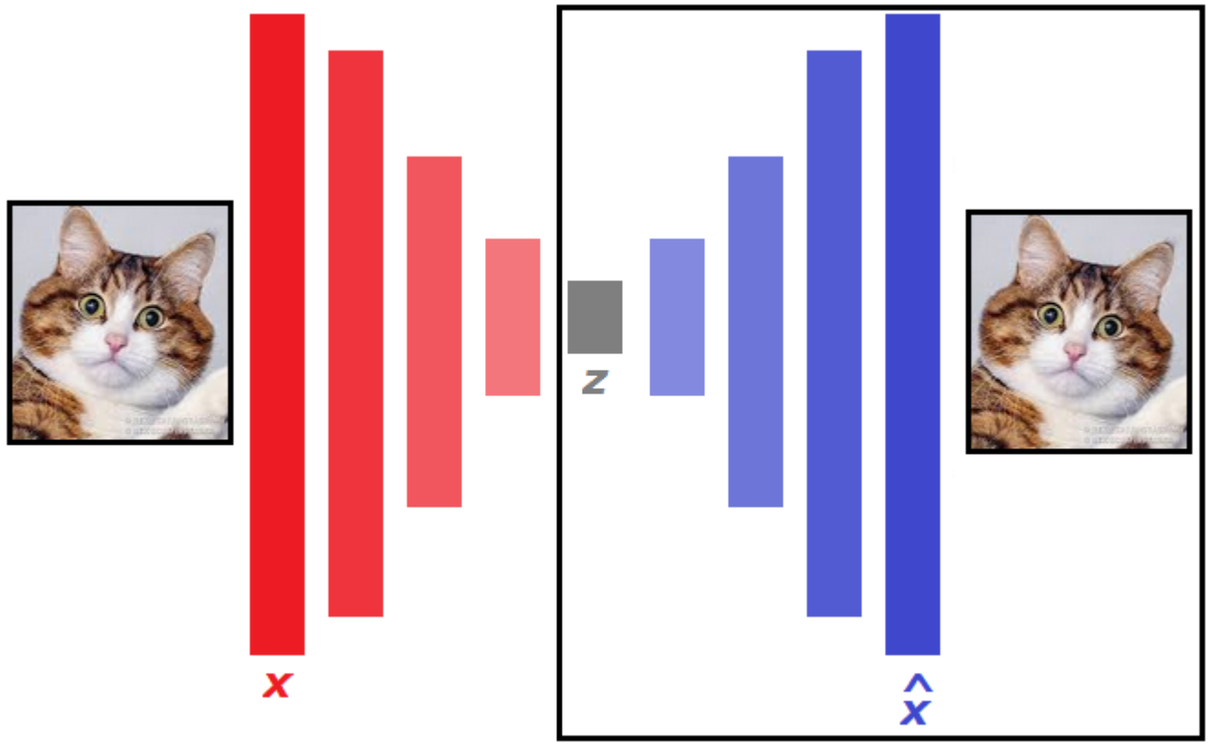
штраф – квадрат нормы Фробениуса $\| \cdot \|_F^2$ **Якобиана** J

$$\| J(x) \|_F^2 = \sum_{ij} \left(\frac{\partial f_j(x)}{\partial x_i} \right)^2$$

производные берутся в скрытом слое $f(x)$

Salah Rifai et al «Contractive Auto-Encoders: Explicit Invariance During Feature Extraction» // https://icml.cc/Conferences/2011/papers/455_icmlpaper.pdf

Минусы стандартного автокодировщика



вряд ли выделенная часть будет генеративной моделью, а хотелось бы…

мы хотим генерировать то, чего не было в обучении ⇒

1) «компактность» латентного пространства (где брать прообразы объектов?)

2) плотность латентного пространства + семантическая устойчивость к шумам

Variational Autoencoders (VAE)

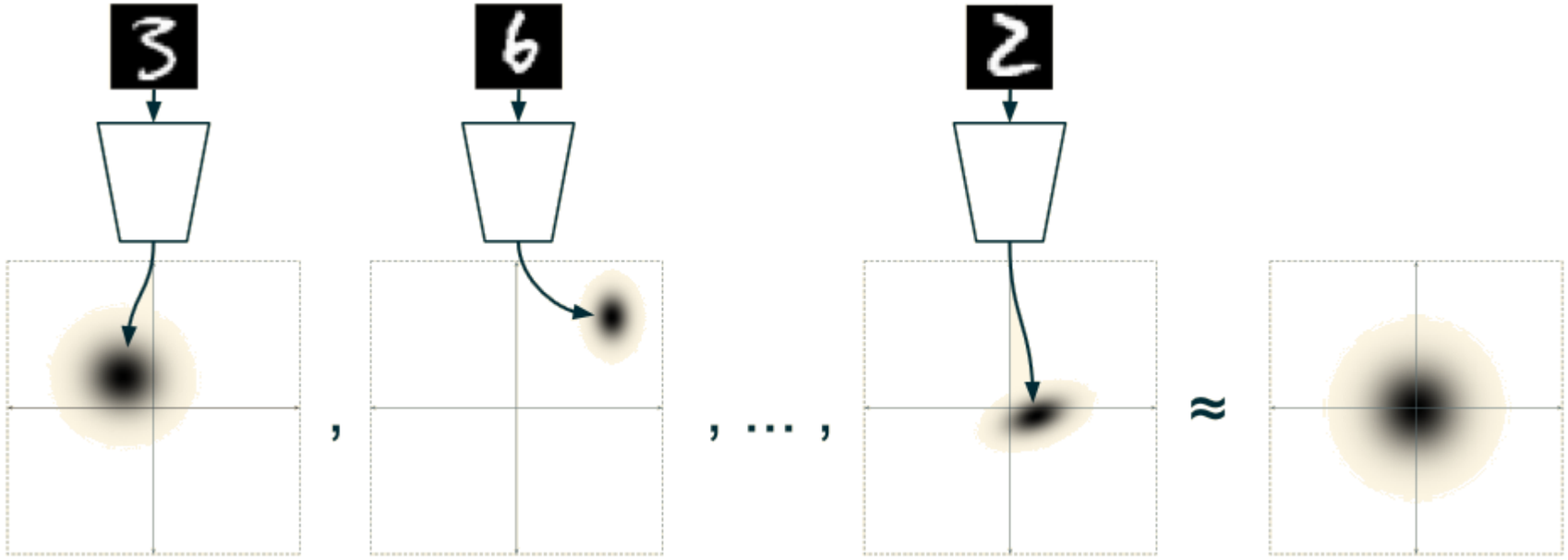
1) «компактность» латентного пространства (где брать прообразы объектов?)

пусть латентное пространство ~ распределение

2) плотность латентного пространства + семантическая устойчивость к шумам

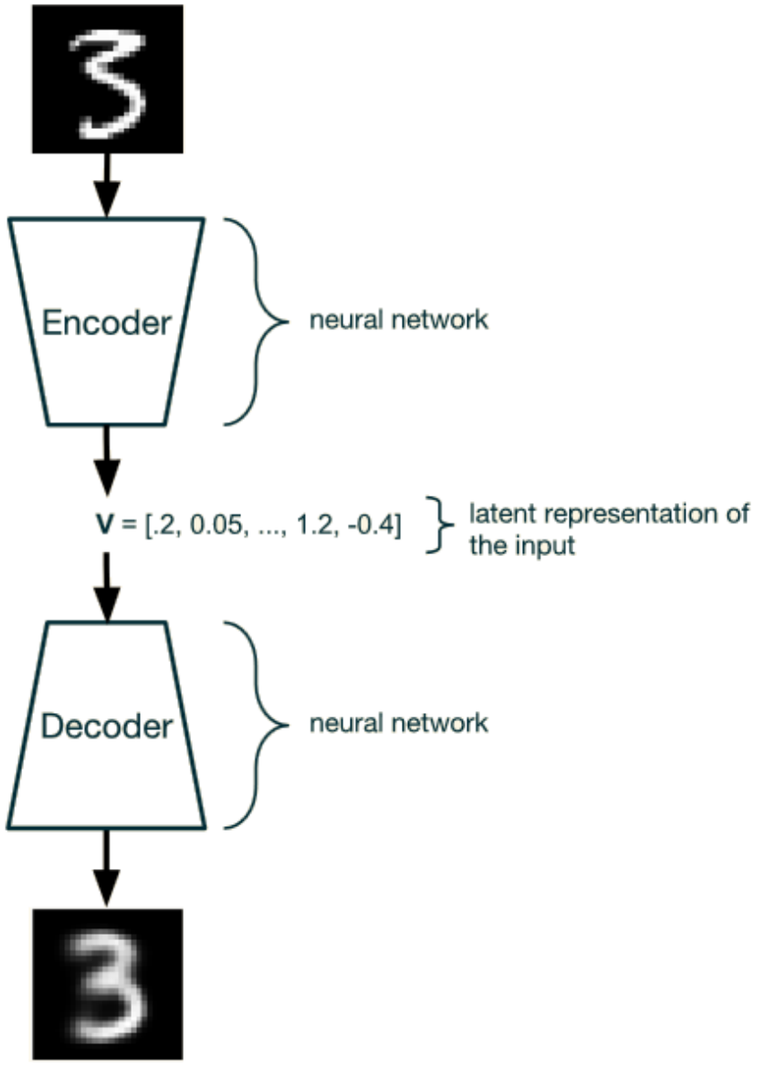
пусть отображение не в вектор, а в распределение (точнее в центр распределения)

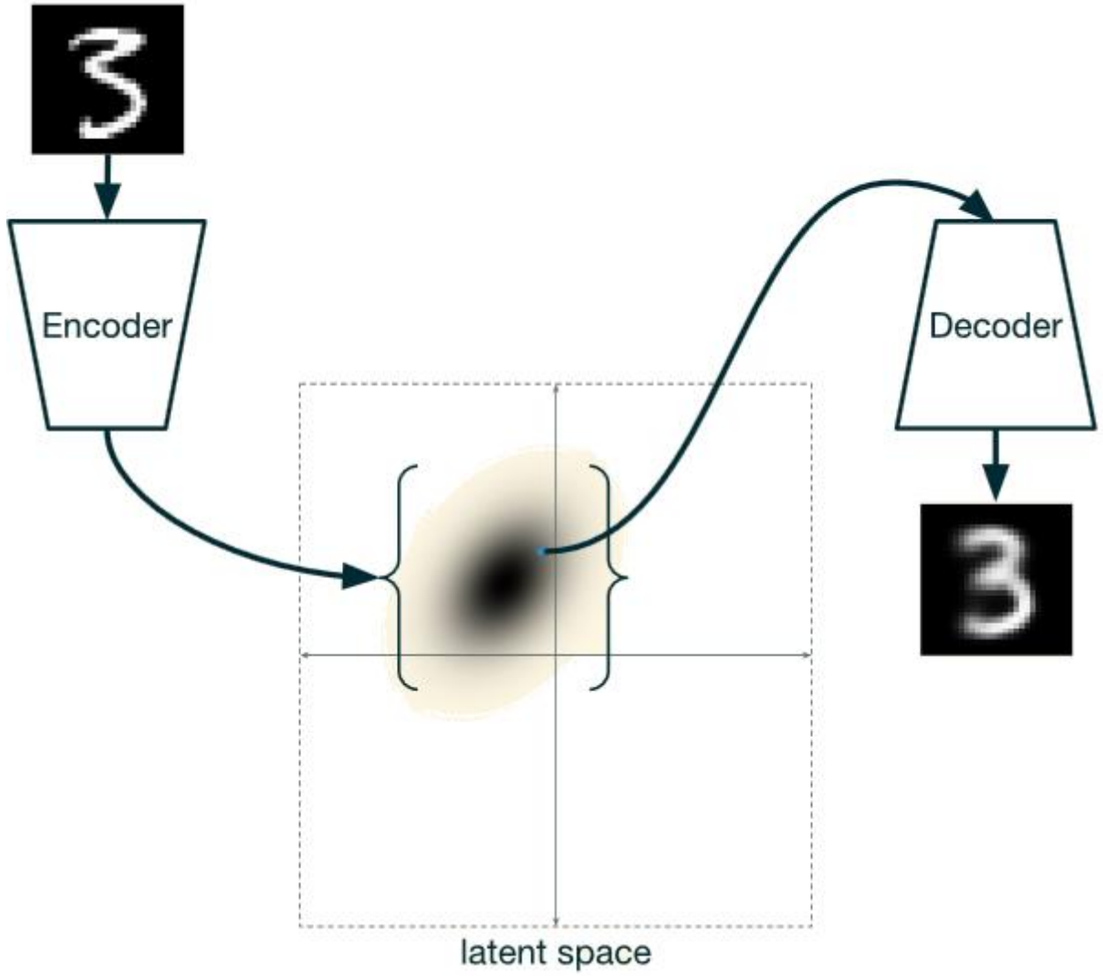
декодировщику подадим любую точку из этого распределения



https://ijdykeman.github.io/ml/2016/12/21/cvae.html

Variational Autoencoders (VAE)

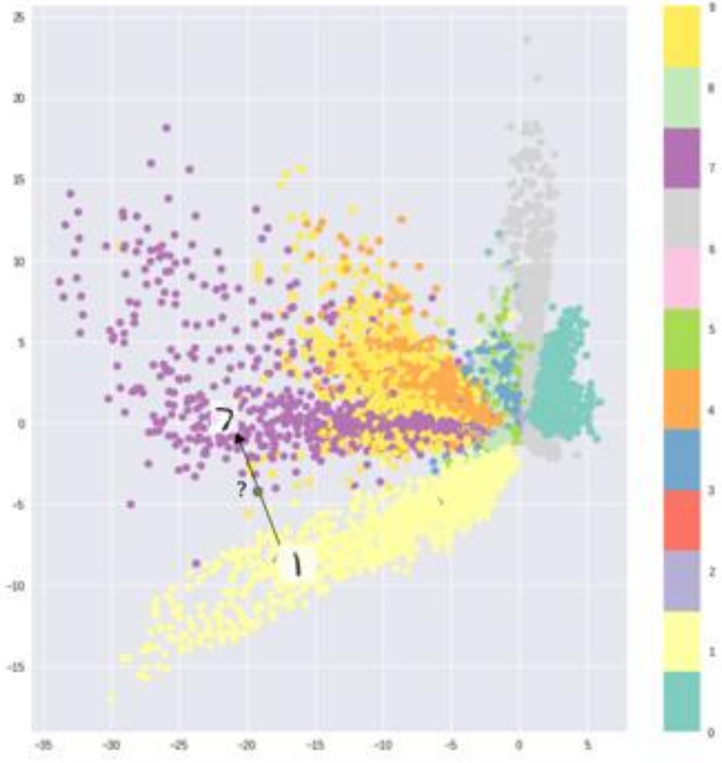




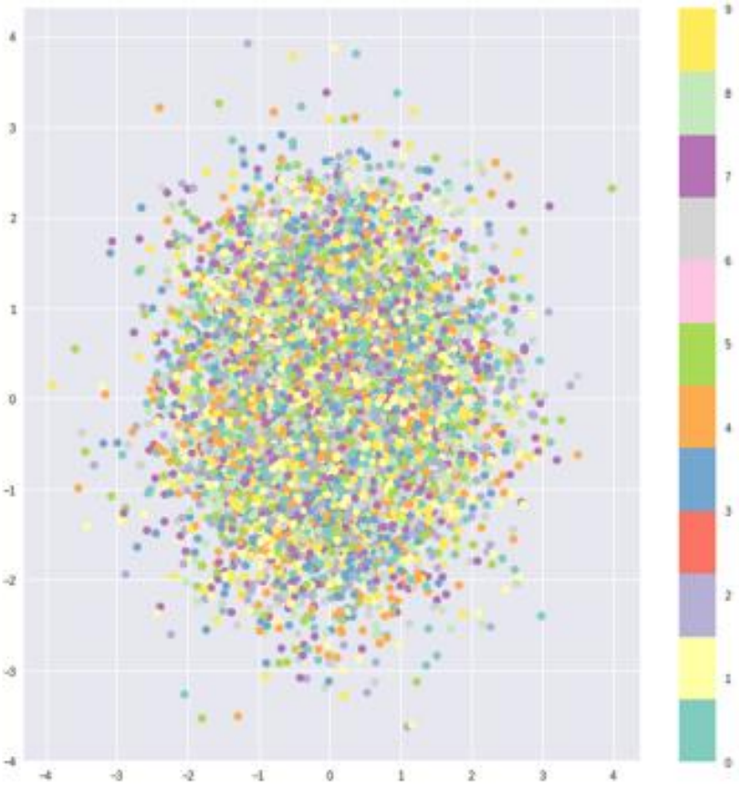
https://ijdykeman.github.io/ml/2016/12/21/cvae.html

Variational Autoencoders (VAE)

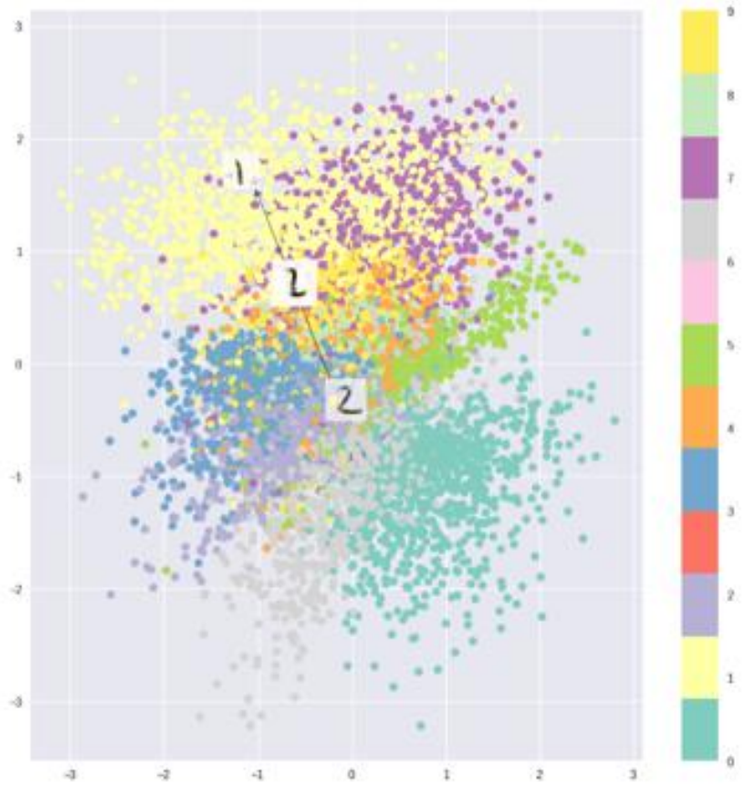
Only reconstruction loss



Only KL divergence

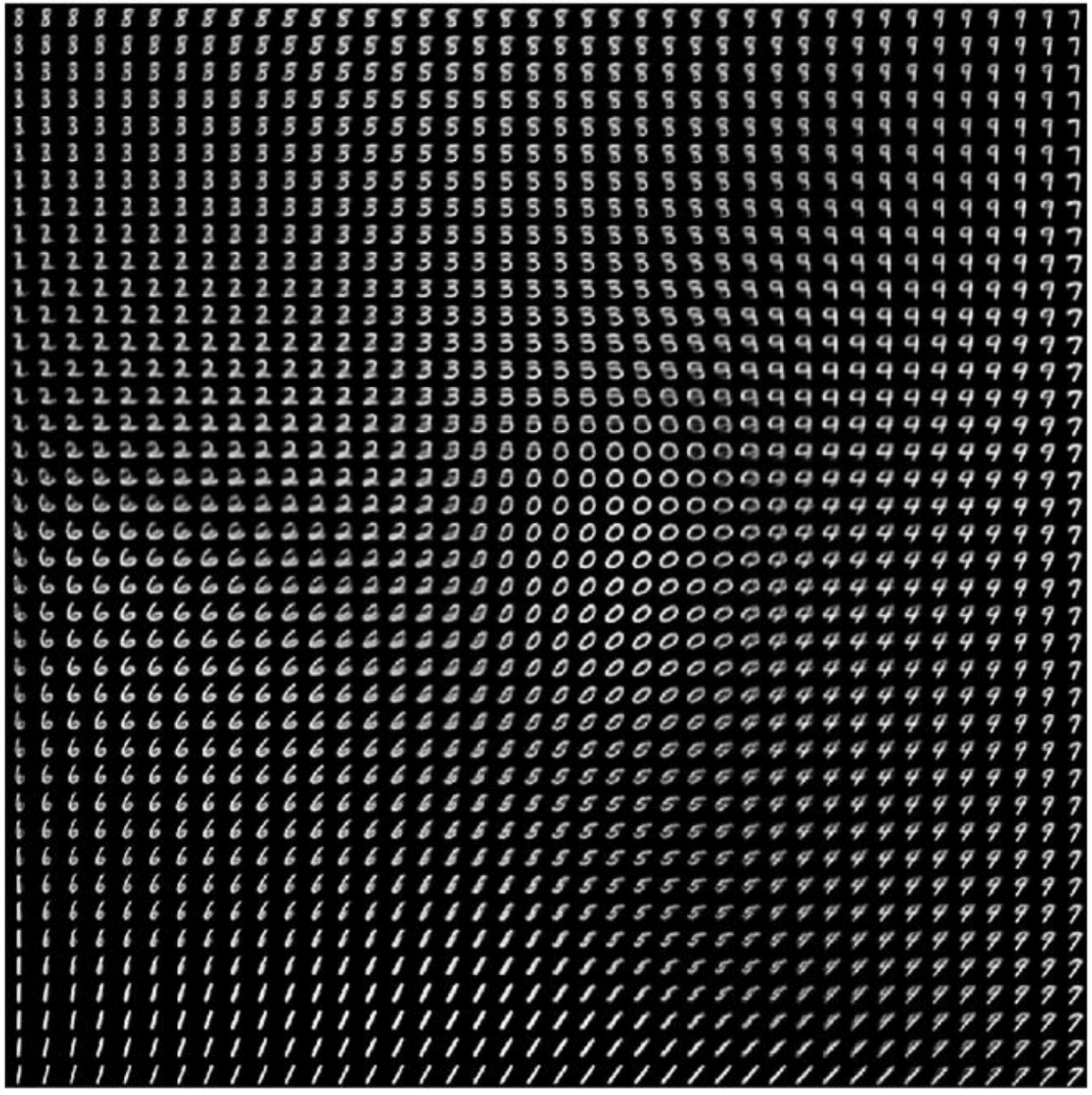


Combination

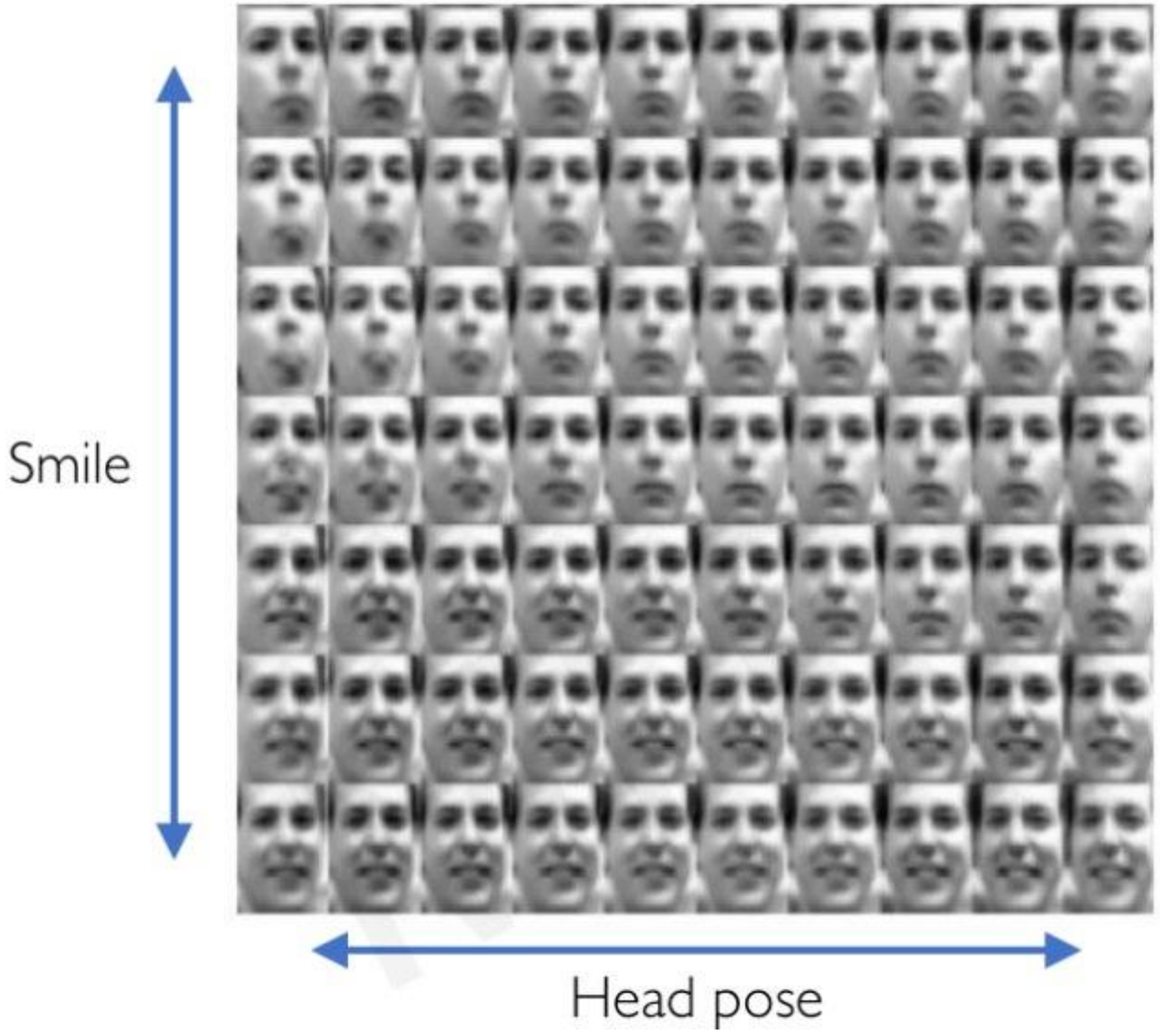


https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Variational Autoencoders (VAE): вариация двух координат z

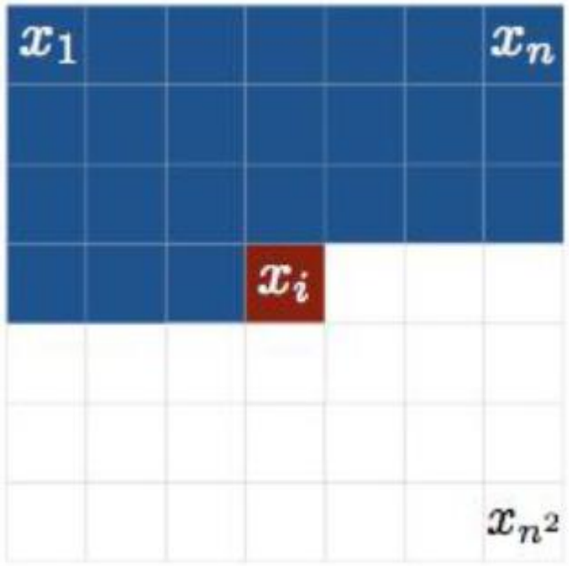


Могут быть интерпретируемые латентные переменные



Pixel CNN / Pixel RNN: Masked Spatial (2D) Convolution

Идея пиксельных сетей

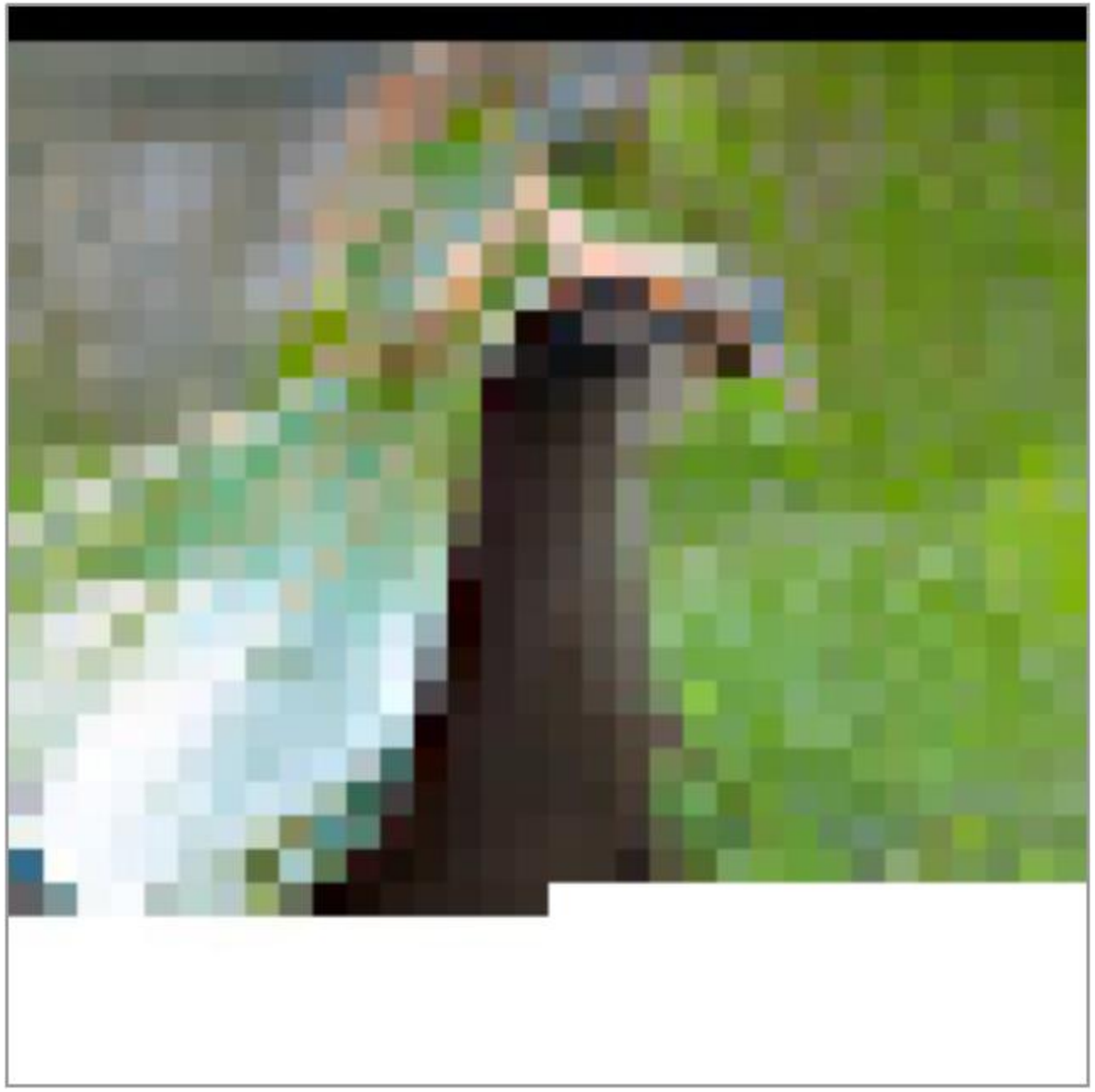


Можно брать свёртки или рекуррентности

+ маскирование (чтобы честная генерация изображения)

Низкая скорость генерации… но есть улучшения

Pixel CNN / Pixel RNN: Softmax-сэмплирование



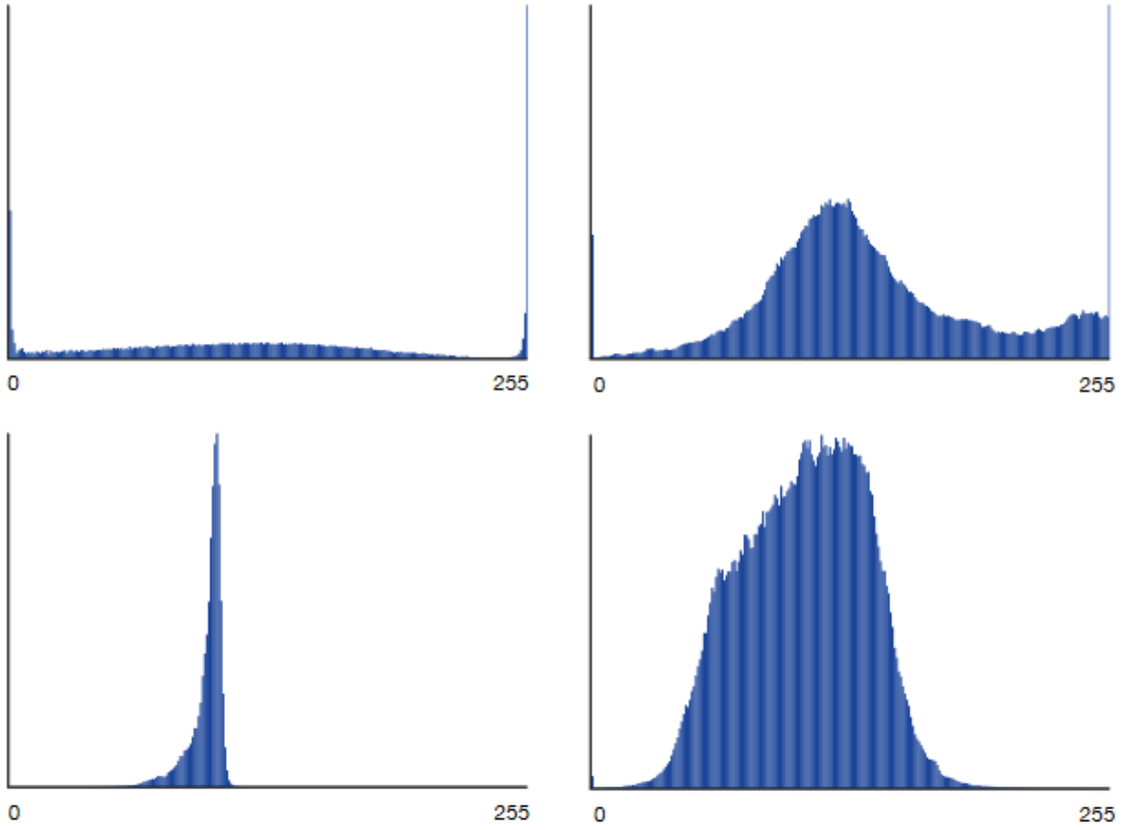
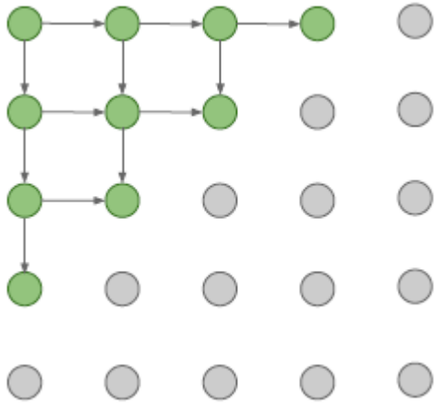


Figure 6. Example softmax activations from the model. The top left shows the distribution of the first pixel red value (first value to sample).

https://arxiv.org/pdf/1601.06759.pdf

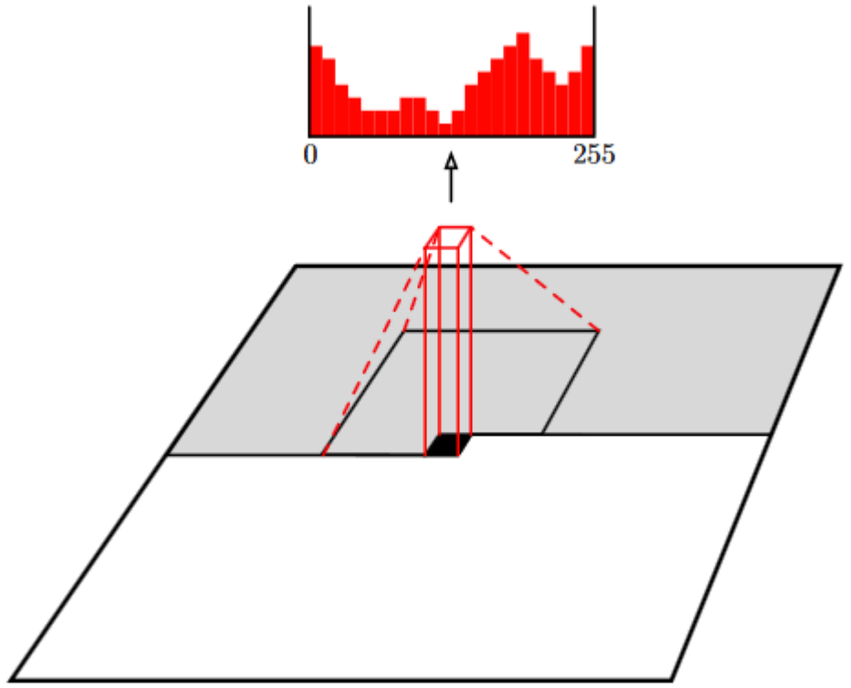
PixelRNN

Генерация изображения из угла (каждый пиксель зависит от предыдущих)



PixelCNN

Генерация изображения из угла (каждый пиксель ~ CNN от предыдущего региона)



PixelRNN

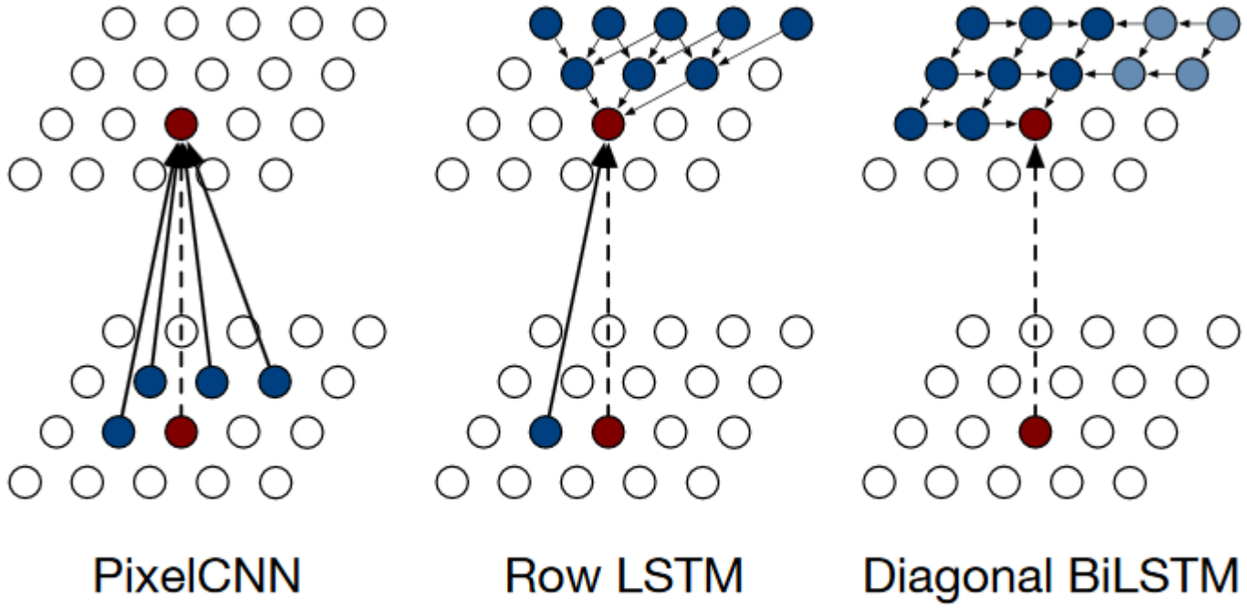


Figure 4. Visualization of the input-to-state and state-to-state mappings for the three proposed architectures.

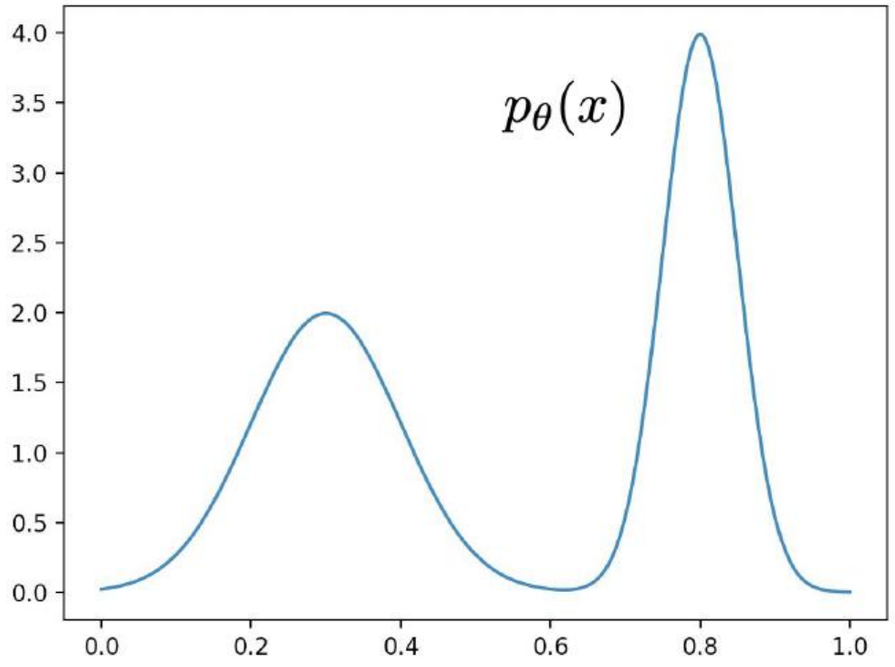
PixelCNN	PixelRNN – Row LSTM	PixelRNN – Diagonal BiLSTM
Full dependency field	Triangular receptive field	Full dependency field
Fastest	Slow	Slowest
Worst log-likelihood	-	Best log-likelihood

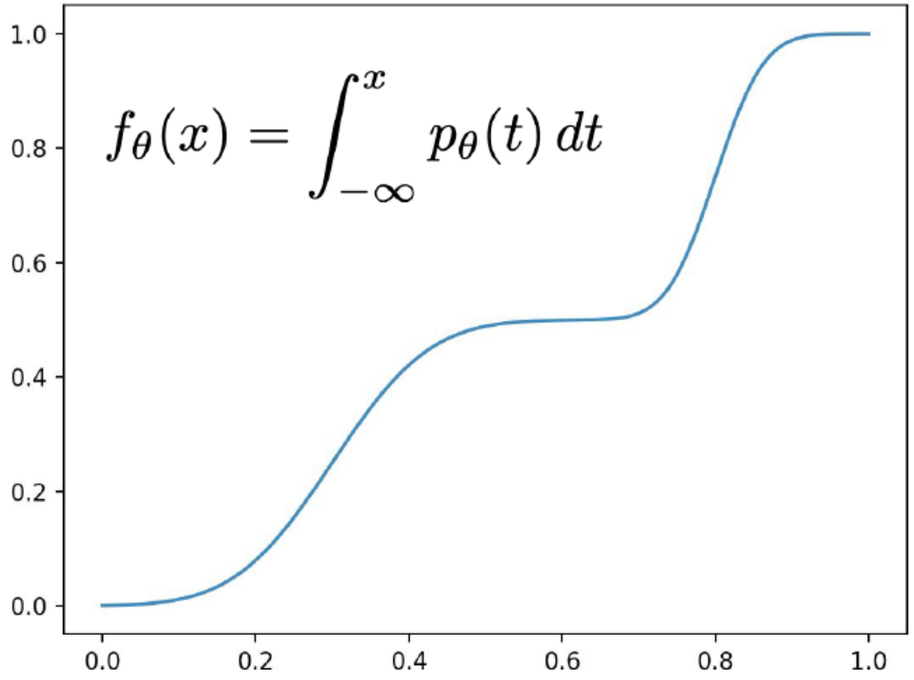
Aaron van den Oord Pixel et al «Recurrent Neural Networks» https://arxiv.org/pdf/1601.06759.pdf

PixelRNN

Figure 9. Image completions sampled from a model that was trained on 32x32 ImageNet images. Note that diversity of the completions is high, which can be attributed to the log-likelihood loss function used in this generative model, as it encourages models with high entropy. As these are sampled from the model, we can easily generate millions of different completions. It is also interesting to see that textures such as water, wood and shrubbery are also inputed relative well (see Figure 1).

Потоки (Flows): идея





CDF переводит область определения (распределение) на отрезок [0, 1]

как мы сэмплируем…

$$z = f(x)$$

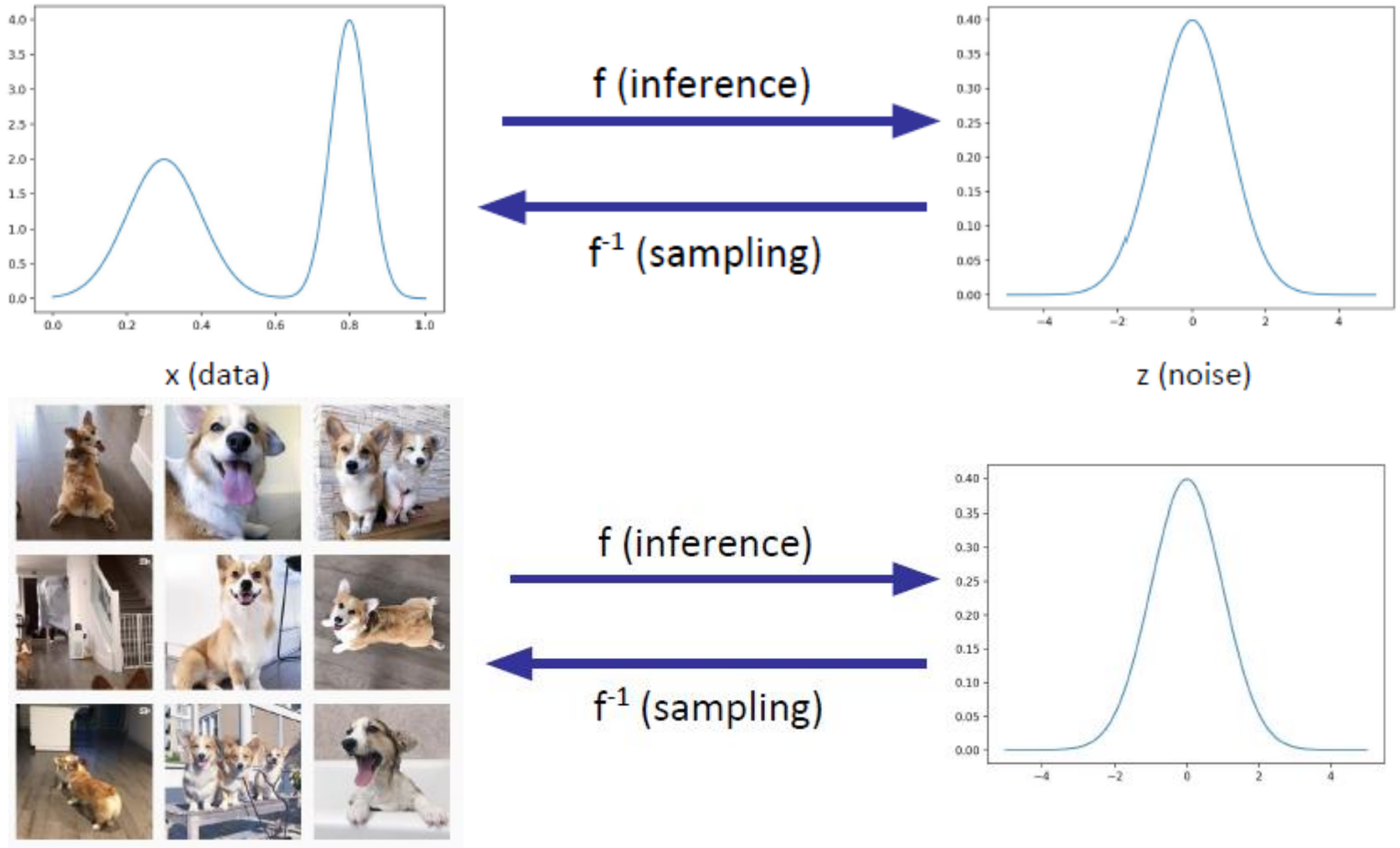
$$z \sim U[0,1]$$

$$x = f_\theta^{-1}(z)$$

CDF ~ обратимое дифференцируемое отображение data → [0, 1]

вместо плотности можно сразу учить соответствующие отображения

Потоки (Flows)



– дифференцируемое обратимое отображение data → шум

переводим распределение данных в базовое (обычно нормальное или равномерное)

x и z должны быть одной размерности

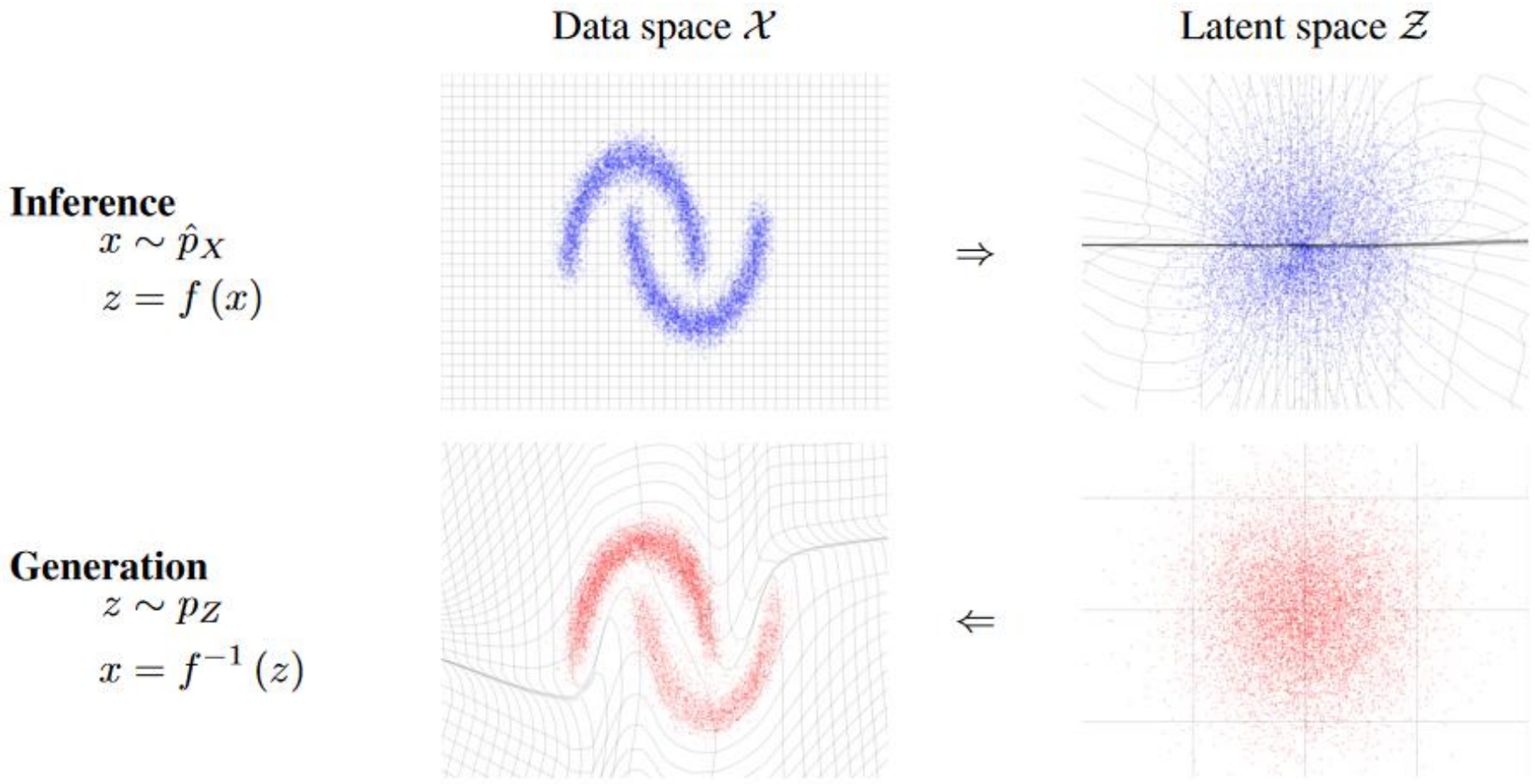


Figure 1: Real NVP learns an invertible, stable, mapping between a data distribution $\hat{p}_X$ and a latent distribution p_Z (typically a Gaussian). Here we show a mapping that has been learned on a toy 2-d dataset. The function $f(x)$ maps samples x from the data distribution in the upper left into approximate samples z from the latent distribution, in the upper right. This corresponds to exact inference of the latent state given the data. The inverse function, $f^{-1}(z)$, maps samples z from the latent distribution in the lower right into approximate samples x from the data distribution in the lower left. This corresponds to exact generation of samples from the model. The transformation of grid lines in $\mathcal{X}$ and $\mathcal{Z}$ space is additionally illustrated for both $f(x)$ and $f^{-1}(z)$.

Применение с обратимыми свёртками: Glow

генеративная потоковая модель на изображениях высокого разрешения

точное значение логарифма правдоподобие

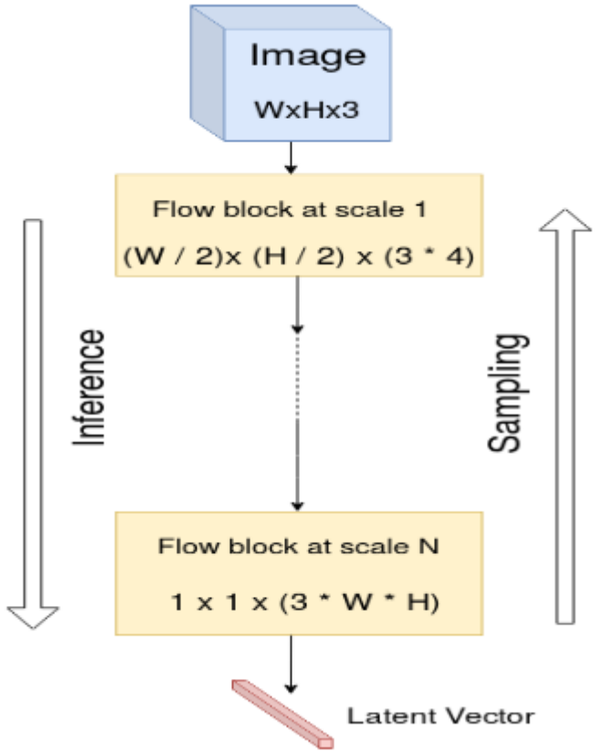
получение латентных переменных

возможность распараллеливания обучения и работы

Figure 1: Synthetic celebrities sampled from our model; see Section 3 for architecture and method, and Section 5 for more results.

Применение с обратимыми свёртками: Glow

Идея:

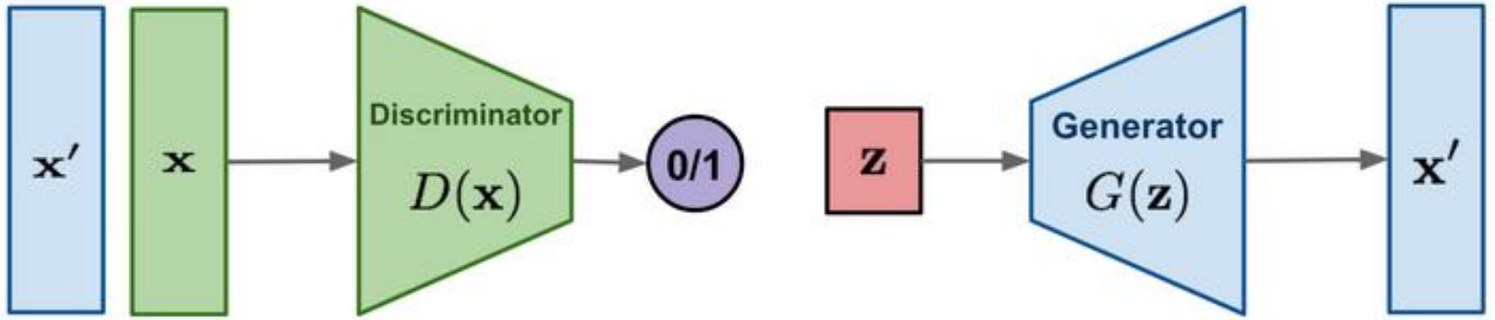


но свёртки должны быть обратимые, чтобы использовать идеологию потоков

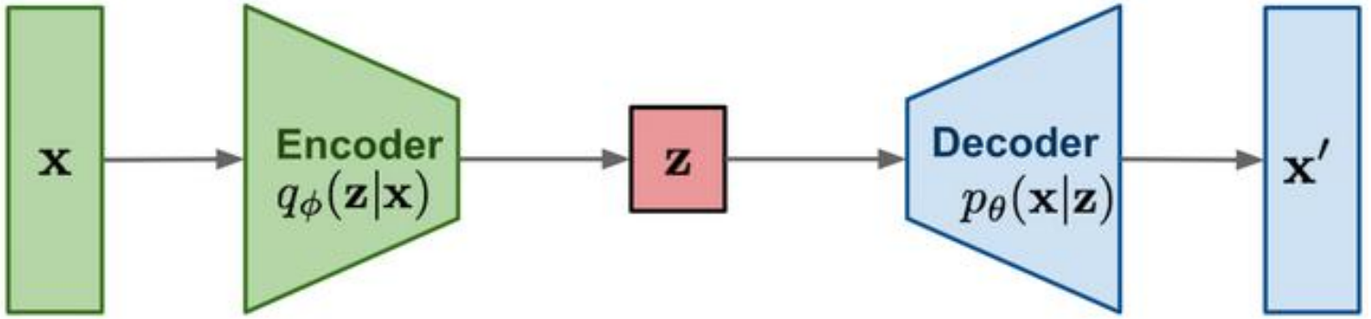
Diederik P. Kingma, Prafulla Dhariwal «Glow: Generative Flow with Invertible 1×1 Convolutions» https://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

Итог и что будет...

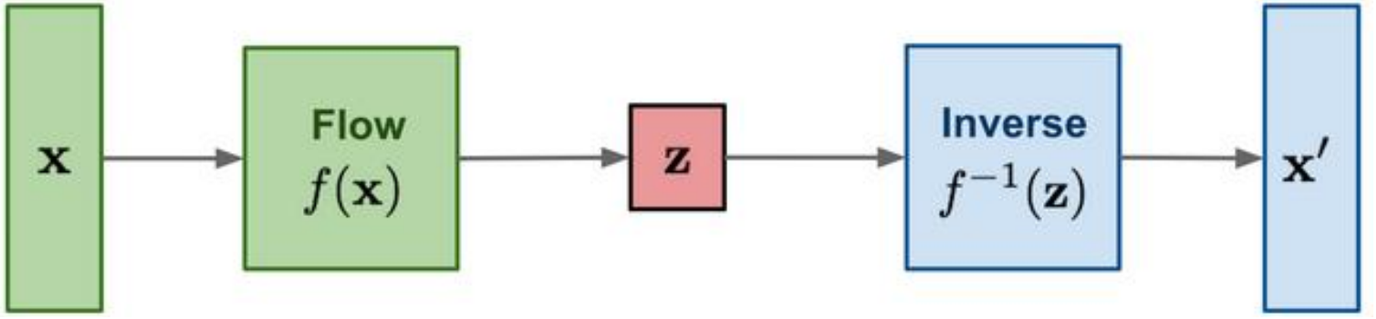
GAN: minimax the classification error loss.



VAE: maximize ELBO.



Flow-based generative models: minimize the negative log-likelihood



https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Generative Adversarial Networks (GAN)

Есть фундаментальная проблема – учим генерировать «котиков»
– как измерить качество модели?

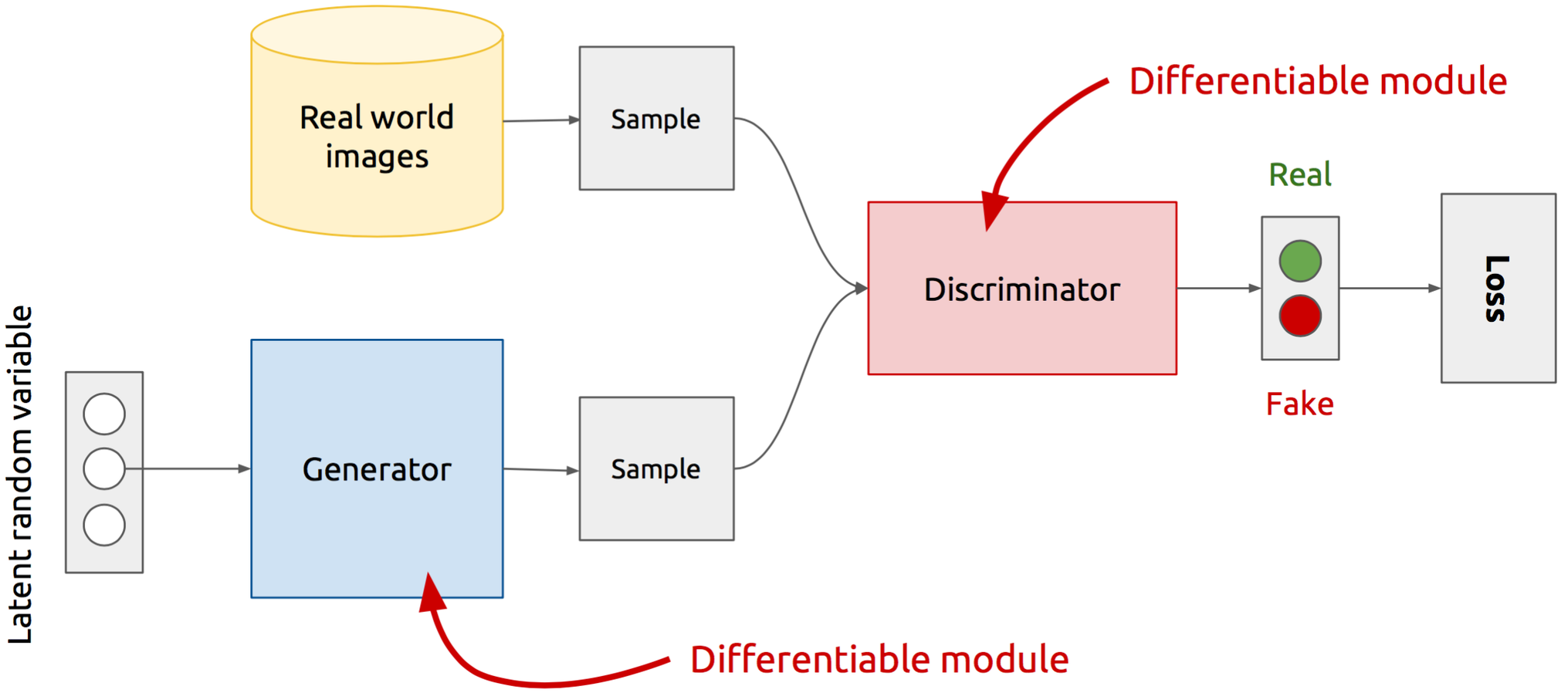
Generative ~ учим генеративную модель
Adversarial ~ в состязательной парадигме
Networks ~ DNN

Модели

Дискриминативные	Генеративные
оценивают P(X|Y)	моделируют P(X)

[Goodfellow et al., 2014]

GAN: Генератор и дискриминатор



Генератор – сеть, которая порождает объект (изображение) из шума

Дискриминатор – сеть, различающая настоящие и сгенерированные объекты

Generative Adversarial Networks (GAN)

- **главное: по сути, дискриминатор – дифференцируемая функция ошибки!**

эту идею можно использовать там, где нет подходящих функций ошибок…

- **в отличие от PixelCNN, VAE нет явной функции плотности!**

т.н. «implicit density model»

- **игровой подход!**

Что могут GAN

- **научились работать со сложными объектами в пространствах высокой размерности**
- **генерация реалистичных объектов**
- **заполнение пропусков (и другие задачи USL)**
- **использование генеративных моделей-ассистентов (при редактировании)**

Задачи

- **улучшение изображений (Image Inpainting)**
- **улучшение звука (Speech Enhancement)**
- **генерация изображений (Image Generation)**
- **супер-разрешение (Super-resolution)**
- **раскраска изображений (Colorization)**
- **творчество (Artwork)**
- **пополнение датасета (Synthetic data), моделирование (Simulation)**

TABLE 2: Applications of GANs discussed in Section 5

Field	Subfield	Method
Image processing and computer vision	Super-resolution	SRGAN [63], ESRGAN [64], Cycle-in-Cycle GANs [65], SRDGAN [66], TGAN [67]
	Image synthesis and manipulation	DR-GAN [68], TP-GAN [69], PG^2 [70], PSGAN [71], APDrawingGAN [72], IGAN [73], introspective adversarial networks [74], GauGAN [75]
	Texture synthesis	MGAN [76], SGAN [77], PSGAN [78]
	Object detection	Segan [79], perceptual GAN [80], MTGAN [81]
	Video	VGAN [82], DRNET [83], Pose-GAN [84], video2video [85], MoCoGan [86]
Sequential data	Natural language processing (NLP)	RankGAN [87], IRGAN [88], [89], TAC-GAN [90]
	Music	RNN-GAN (C-RNN-GAN) [91], ORGAN [92], SeqGAN [93], [94]

https://arxiv.org/pdf/2001.06937.pdf

Идея состязания (Adversarial idea)

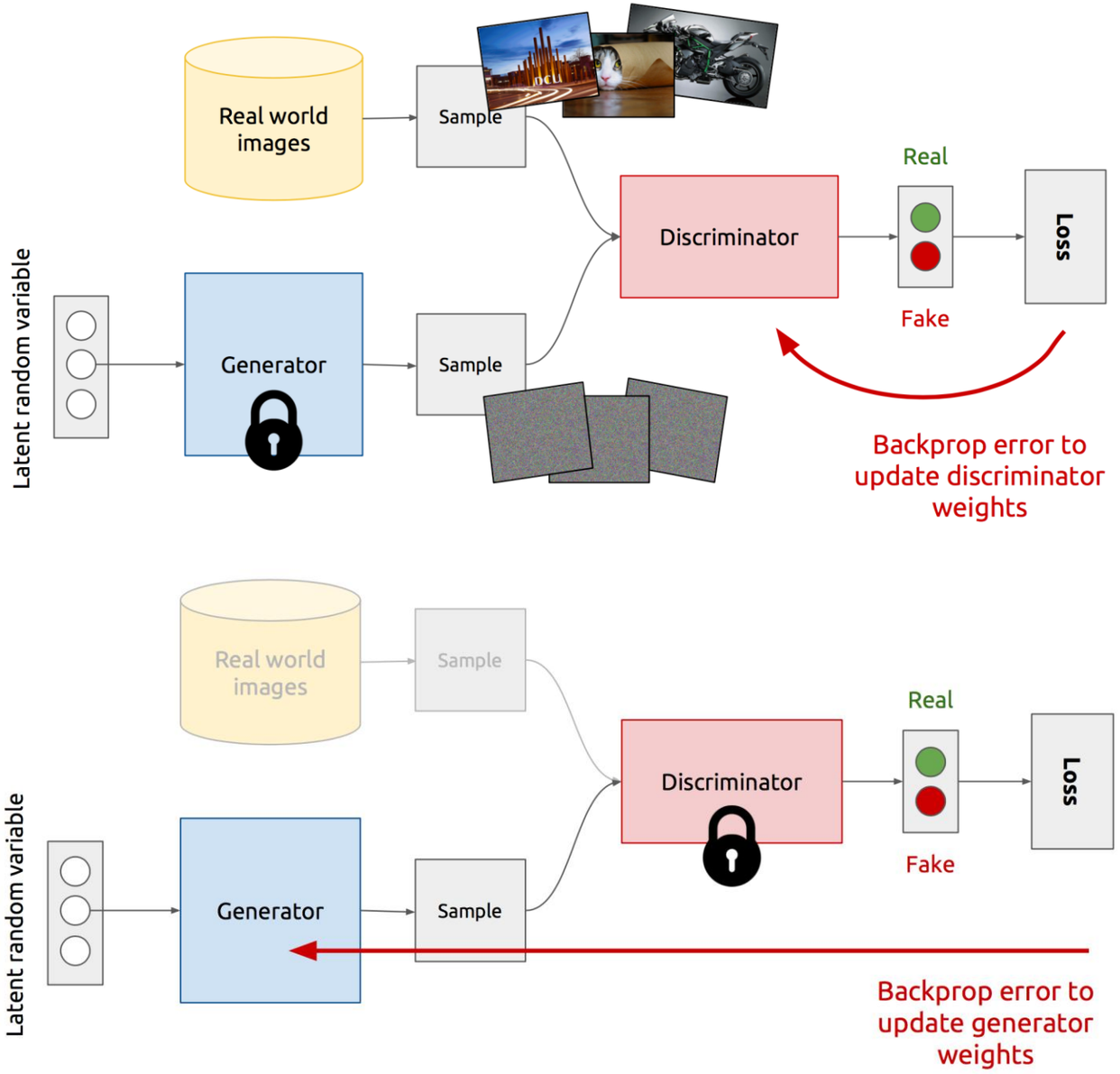
AlphaGo – две сети соперничают друг с другом

Adversarial examples – объекты, которые «неправильно классифицируются»

ex: небольшая модификация объектов какого-то класса,
меняющая результат классификации

«adversarial attacks

GAN: обучение



GAN: обучение – min-max-игра

$$\min_{\theta}\max_{\varphi}\left[\mathbf{E}_{x\sim p_{\text{data}}}\log D_{\varphi}(x)+\mathbf{E}_{z\sim p(z)}\log(1-D_{\varphi}(G_{\theta}(z))\right]$$

Дискриминатор выводит правдоподобие из [0, 1]

$D_{\varphi}(x)$ **– для настоящих данных**

$D_{\varphi}(G_{\theta}(z))$ **– для сгенерированных данных**

Дискриминатор хочет $D_{\varphi}(x)\to 1,\ D_{\varphi}(G_{\theta}(z))\to 0$

Генератор хочет $D_{\varphi}(G_{\theta}(z))\to 1$

Дискриминатор – обычный классификатор

cross entropy loss

Генератор – зависит от того, что генерируем…

Ian J. Goodfellow et al. Generative Adversarial Nets https://arxiv.org/pdf/1406.2661.pdf

GAN: обучение – min-max-игра

Если зафиксировать генератор

$$\min_{\theta}\max_{\varphi}\left[\underbrace{\mathbf{E}_{x\sim p_{\text{data}}}\log D_{\varphi}(x)+\mathbf{E}_{z\sim p(z)}\log(1-D_{\varphi}(G_{\theta}(z))}_{\int(p_{\text{data}}(x)\log D_{\varphi}(x)+p_{G}(x)\log(1-D_{\varphi}(x))\partial x}\right]$$

Взяв производную по D и приравняв к нулю:

$$D^{*}(x)=\frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_{\text{G}}(x)}$$

– оптимальный дискриминатор при фиксированном генераторе

здесь распределение фейков – $p_{\text{G}}(x)\sim x=G_{\theta}(z\,|_{z\sim p(z)})$

GAN: обучение – min-max-игра

Если подставить оптимальный дискриминатор (существенное условие)

$$\int (p_{\text{data}}(x)\log D_\varphi(x) + p_G(x)\log(1-D_\varphi(x))\partial x =$$

$$= \int \left(p_{\text{data}}(x)\log\frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_G(x)} + p_G(x)\log\frac{p_G(x)}{p_{\text{data}}(x)+p_G(x)} \right)\partial x =$$

$$= D_{KL}\left(p_{\text{data}}(x) \,\|\, \frac{p_{\text{data}}(x)+p_G(x)}{2} \right) + D_{KL}\left(p_G(x) \,\|\, \frac{p_{\text{data}}(x)+p_G(x)}{2} \right) - 2\log 2$$

дивергенция Йенсена-Шеннона / JSD (Jensen-Shannon Divergence)

$$\text{JSP}(p \,\|\, q) = \frac{\text{KL}(p \,\|\, (p+q)/2) + \text{KL}(q \,\|\, (p+q)/2)}{2}$$

проблемы: 1) если носители распределений не пересекаются

2) градиенты затухают, когда D обучился

GAN: обучение – min-max-игра

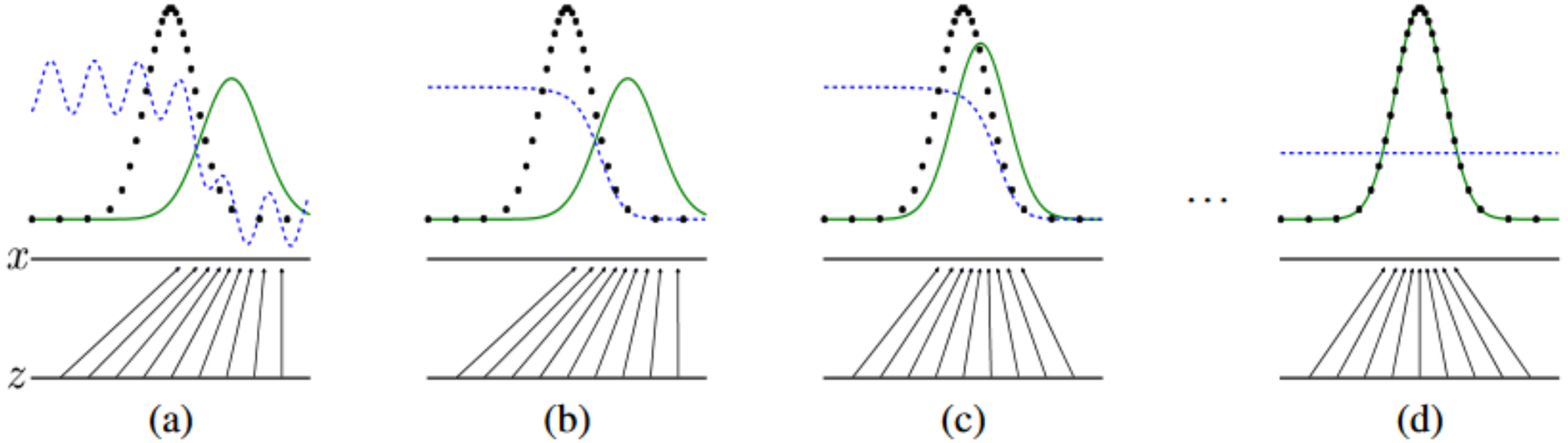


Figure 1: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping $x = G(z)$ 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 $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_g(x)}$. (c) After an update to G, gradient of D has guided $G(z)$ 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. $D(x) = \frac{1}{2}$.

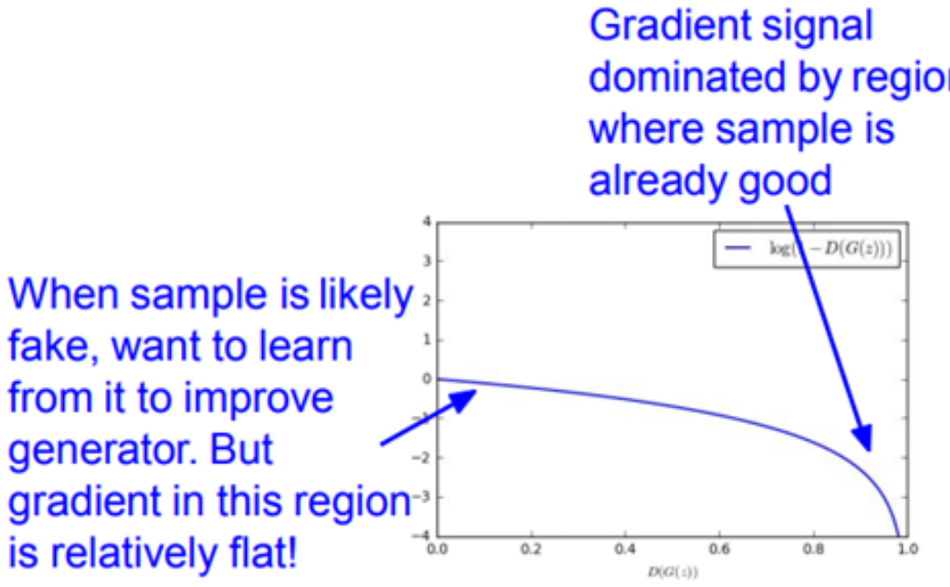
GAN: практическая оптимизация – «non-saturating game»

для оптимизации лучше «non-saturating game»:

$$\max_{\varphi}\left[\mathbf{E}_{x\sim p_{\text{data}}}\log D_{\varphi}(x)+\mathbf{E}_{z\sim p(z)}\log(1-D_{\varphi}(G_{\theta}(z))\right]$$

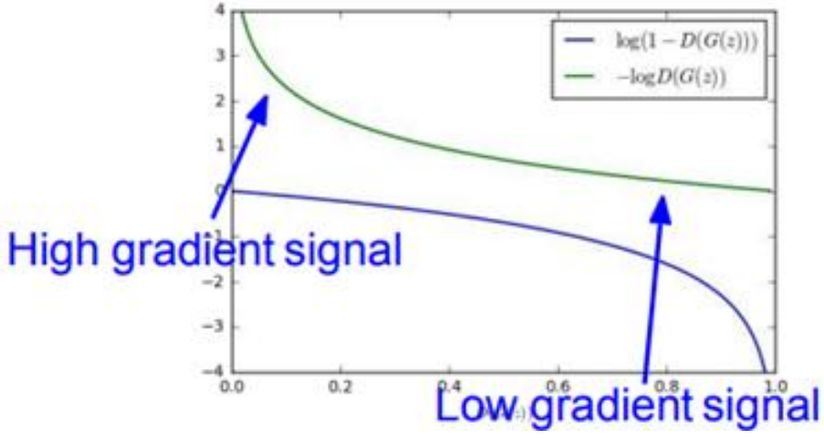
$$\max_{\theta}\mathbf{E}_{z\sim p(z)}\log(D_{\varphi}(G_{\theta}(z)))$$

$$\min_{\theta}\mathbf{E}_{z\sim p(z)}\log(1-D_{\varphi}(G_{\theta}(z)))$$



$$\max_{\theta}\mathbf{E}_{z\sim p(z)}\log(D_{\varphi}(G_{\theta}(z)))$$

вместо min правдоподобия корректности дискриминатора – max правдоподобия того, что дискриминатор ошибался



Алгоритм настройки

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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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 ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

GAN: первые примеры

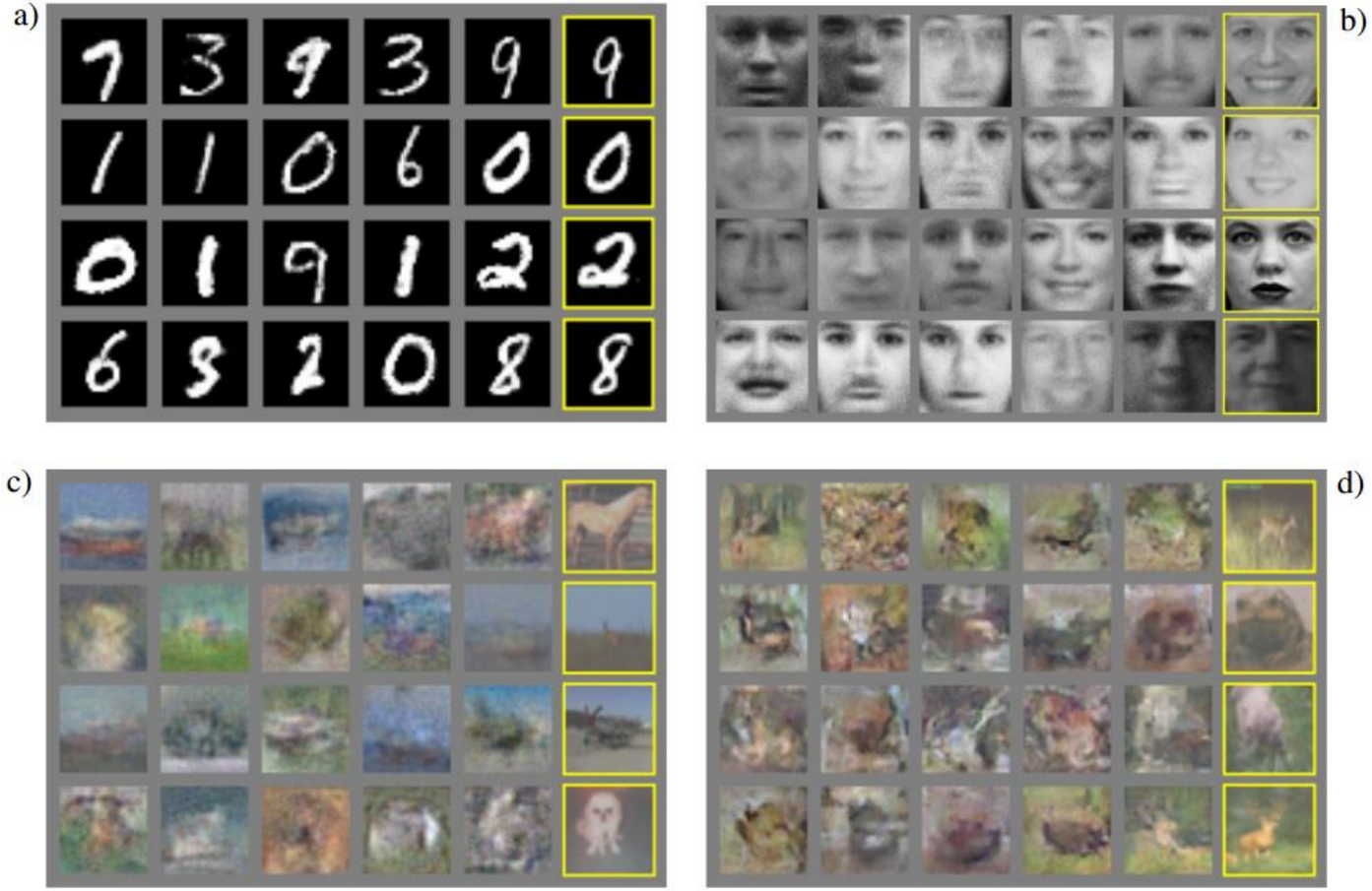


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative 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)

Ian J. Goodfellow et al. Generative Adversarial Nets https://arxiv.org/pdf/1406.2661.pdf

GAN: проблемы первых моделей






Figure 29: GANs on 128 × 128 ImageNet seem to have trouble with counting, often generating animals with the wrong number of body parts.

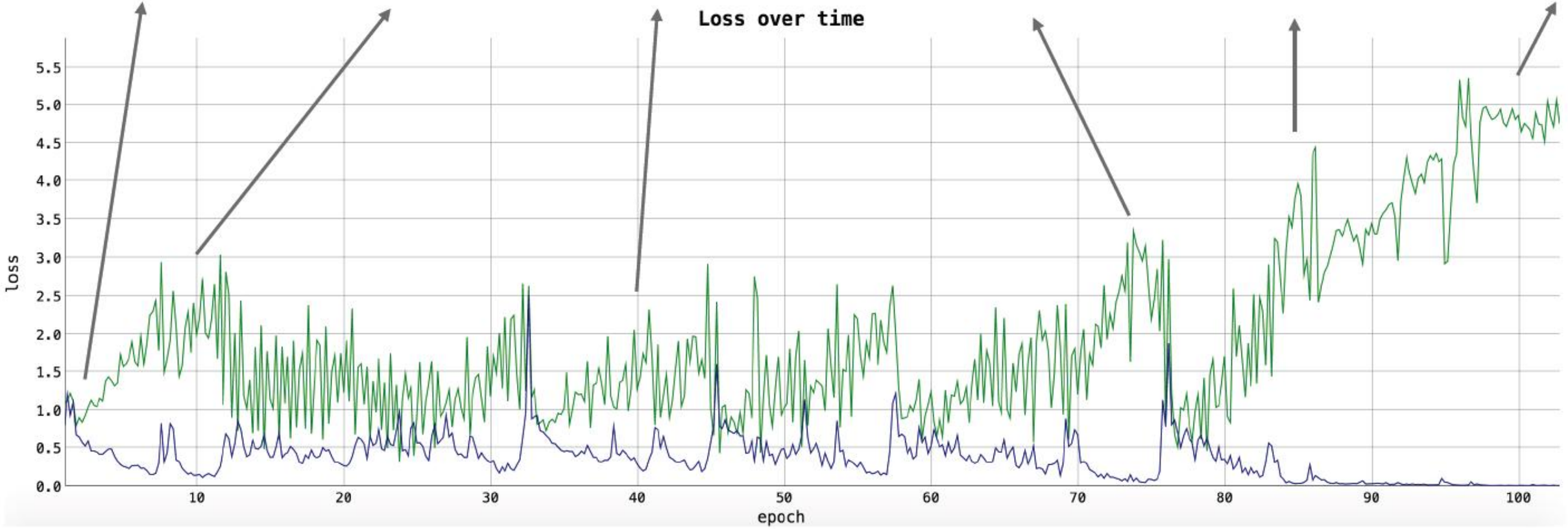
– глобальная структура и неправильное число элементов изображения

Ian Goodfellow «NIPS 2016 Tutorial: Generative Adversarial Networks» https://arxiv.org/abs/1701.00160

Обучение GAN: проблемы на практике





проблемка… вспоминаем про затухание градиентов;)

Обучение GAN: к чему приводит «non-saturating game»

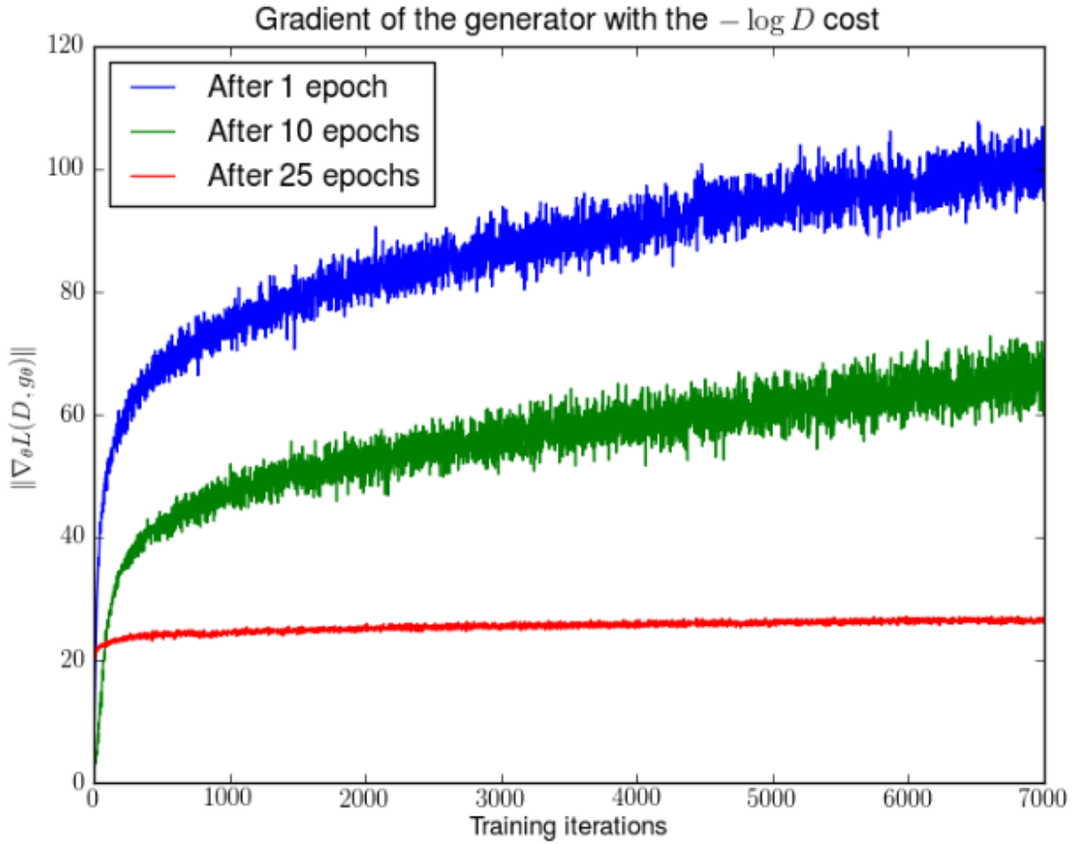


Figure 3: First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the $-\log D$ cost function. We see the gradient norms grow quickly. Furthermore, the noise in the curves shows that the variance of the gradients is also increasing. All these gradients lead to updates that lower sample quality notoriously.

нестабильность – норма градиента (и дисперсия нормы) вырастает!

Обучение GAN: проблемы на практике

Очень сложно обучать

Есть проблемы со стабильностью

плюс дальше опишем сложности

$\Rightarrow$ **много трюков и приёмов**

например для стабильности добавляем шум,
но тогда изображения размываются

в дискриминативной постановке решаем невыпуклую задачу оптимизации – можем сойтись в локальный, а не глобальный минимум

В состояние равновесия можем вообще не попасть

В игровой парадигме хочется найти равновесия, но
SGD не для этого, плохой пример – $\min\limits_{x}\max\limits_{y} xy$

Советы по настройке GAN (не все полезны для всех видов GAN-ов)

1. Нормализация входа (изображения → [–1, +1], выход – tanh)
2. Вместо min(log(1-D(G))) лучше max(log(D(G)))
   приём: меняем метки местами real ↔ fake
3. z сэмплируется не из равномерного, а гауссовского распределения, см. также https://arxiv.org/abs/1609.04468
4. Минибатчи лучше делать чистыми (все real или все fake)
   или специально составляют минибатч Virtual batch normalization (VBN):
   референсные объекты + текущий – лучше считаются статистики + по сравнению с референсными можем справиться с Mode Collapse
5. Не использовать Sparse Gradients (ReLU, Pool), лучше LeakyReLU

   Downsampling: Average Pooling, Conv2d + stride

   Upsampling: PixelShuffle, ConvTranspose2d + stride
6. Лучше размывать метки (label smoothing): 0 → [0, 0.3], 1 → [0.7, 1.2]
   см. потом LSGAN, чаще только 1 размывают

   (чтобы не портить и без того плохие фейки)

Советы по настройке GAN

7. Используйте DCGAN (или гибридные: KL + GAN или VAE + GAN)

8. Используйте трюки из RL (например, Experience Replay)

9. Adam для генератора, SGD для дискриминатора

10. Мониторьте ошибки (ex: loss(D)~0 failure mode – проверяйте градиенты)

11. Добавляйте шум ко входу / к слоям генератора / меткам

12. Дискретные переменные в условных GANax:

используйте Embedding layer, добавляйте как новый канал в изображениях, поддерживайте низкой embedding dimensionality

13. Используйте Dropouts в G

14. Регуляризация нормы градиента

+ специальное слагаемое, чтобы скорректированные параметры не отличались от средних по всем коррекциям (для стабильности)

Советы по настройке GAN

16. Новые loss-функции

дальше подробнее

17. «Соответствие признаков» (Feature matching)

не только смотреть на ответ дискриминатора, но и на схожесть в признаках средних слоёв фейков и реальных объектов

18. Пусть дискриминатор определяет не только фейк / не фейк, а ещё и класс

Salimans T., et al «Improved Techniques for Training GANs» 2016 //
https://arxiv.org/pdf/1606.03498.pdf

How to Train a GAN? Tips and tricks to make GANs work //
https://github.com/soumith/ganhacks

Пример настройки GAN https://poloclub.github.io/ganlab/

GAN в зависимости от функции ошибки в дискриминаторе

функция ошибки	вид GAN
Binary cross-entropy	Vanilla GAN
Least squares (L2)	LSGAN
EM-distance / Wasserstein-1	WGAN
Wasserstein GAN + Gradient penalty	WGAN-GP

другие функции ошибки – один из ключевых способов решения проблем обучения

Least Square GAN (LSGAN)

квадратичная функция ошибки

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

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

борьба с затуханием градиента

Mao et al «Least Squares Generative Adversarial Networks» //
https://arxiv.org/pdf/1611.04076.pdf

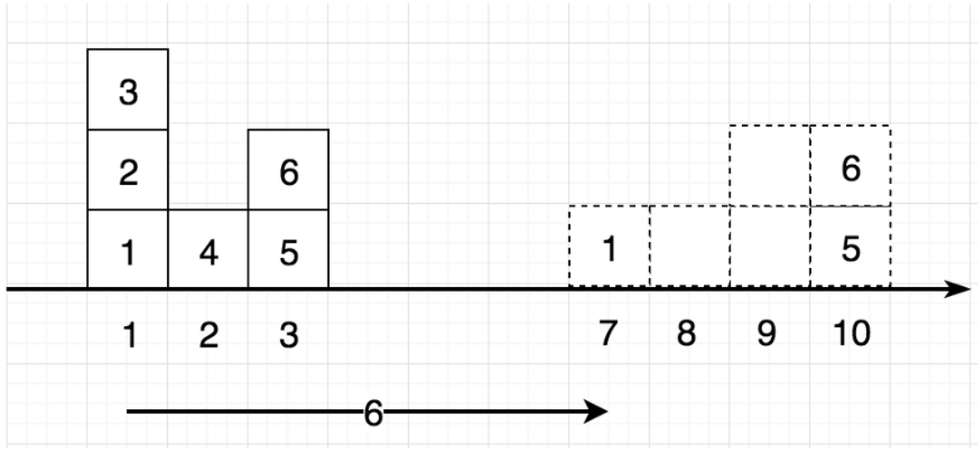
Wasserstein GAN (WGAN)

Earth-Mover (EM) distance (Wasserstein-1)

$$W(p, p') = \inf \mathbf{E}_{(x,y)\sim\Pi(p,p')} \| x - y \|$$

интерпретация – как «выровнять» распределения, если мы как бы переносим кучи песка… тут нет проблемы с пустыми пересечениями носителей

$\Pi(p, p')$ **– семейство совместных распределений с заданными маргинальными**



стоимость переноса = вес·расстояние

Wasserstein GAN (WGAN)

Как вычислять?

$$W(p, p') = \inf \mathbf{E}_{(x,y)\sim\Pi(p,p')} \| x - y \|$$

Двойственность Монжа-Канторовича (Kantorovich-Rubinstein duality)

$$W(p, p') = \sup_{\|f\|_L \le 1} \mathbf{E}_{x\sim p} f(x) - \mathbf{E}_{x\sim p'} f(x)$$

по семейству Липшецевых функций с константой Липшица = 1

На практике для $\| f \|_L \le 1$ обрезаем градиенты (и работает!)

есть теоретический результат,

но тут подвох в том, что гарантируется только

$$\| f \|_L \le \text{const}$$

Martin Arjovsky, Soumith Chintala, Léon Bottou «Wasserstein GAN» // https://arxiv.org/abs/1701.07875

Wasserstein GAN (WGAN)

Обычный GAN: $\min_G \max_D \mathbf{E}_{x\sim p_{\text{data}}(x)} \log D(x) + \mathbf{E}_{z\sim p_z(z)} \log(1 - D(G(x)))$

WGAN: $\min_G W(p_{\text{data}}, p_G) = \min_G \max_f \mathbf{E}_{x\sim p_{\text{data}}(x)}[f(x)] - \mathbf{E}_{z\sim p_z(z)}[f(G(x))]$

Вместо дискриминатора ~ «клиппированная» функция (HC), которую надо $\rightarrow$ max назвали «Критиком», нет сигмоиды и решается задача регрессии – это средство для оценки расстояния между распределениями

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right)\right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right)$

https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

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

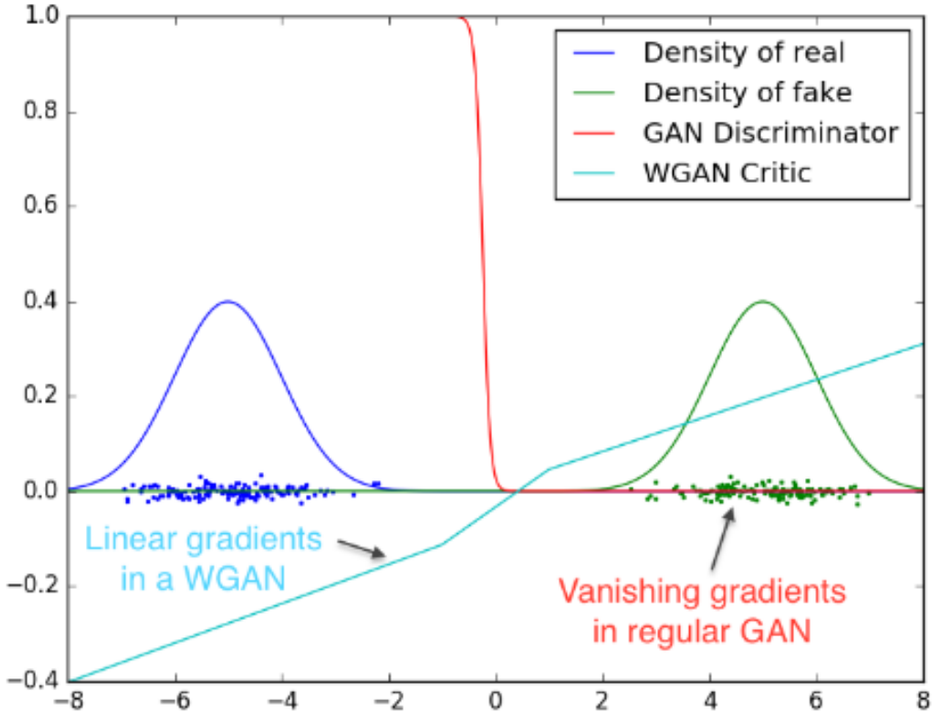


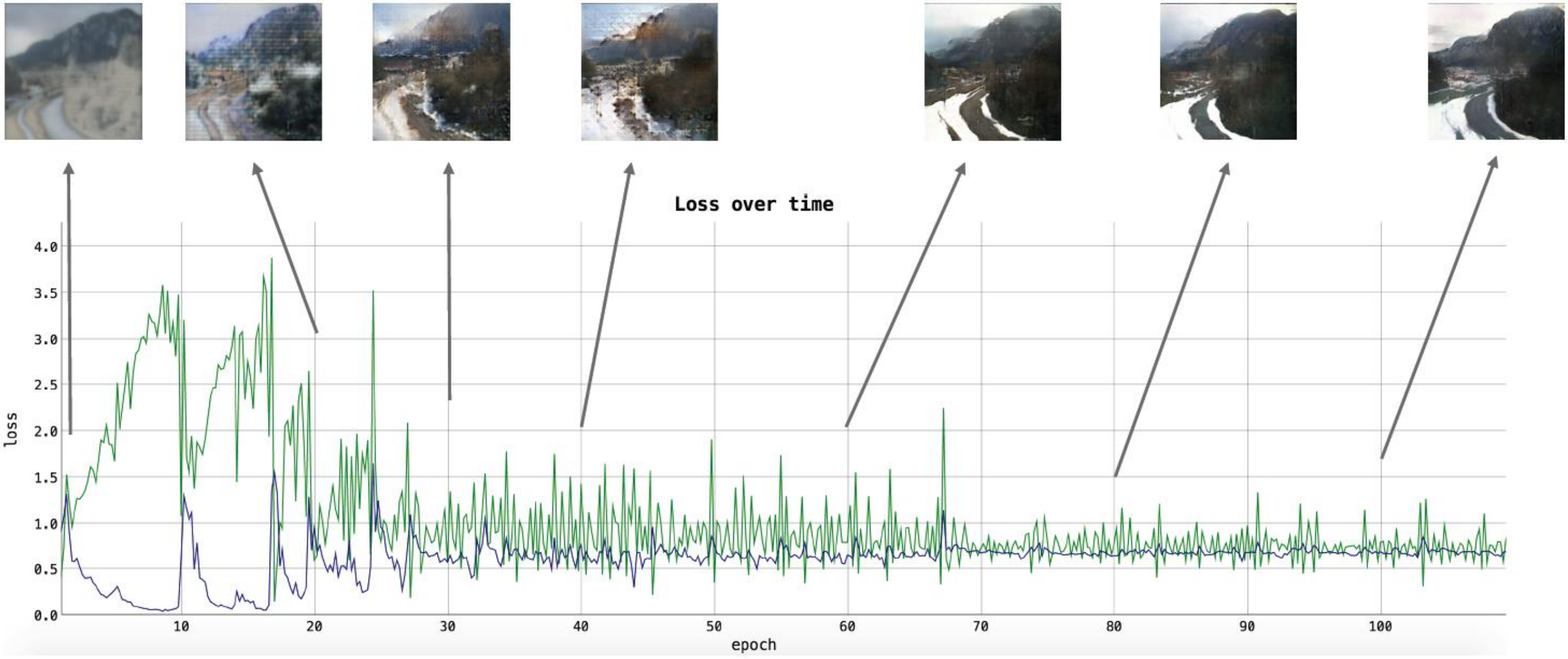
Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

здесь дискриминатор переименовали в критика

нет эффекта коллапса см. дальше

генератор продолжает обучаться, когда критик работает хорошо

Wasserstein GAN



ошибка прямо отражает качество

Figure 5: Algorithms trained with a DCGAN generator. Left: WGAN algorithm. Right: standard GAN formulation. Both algorithms produce high quality samples.

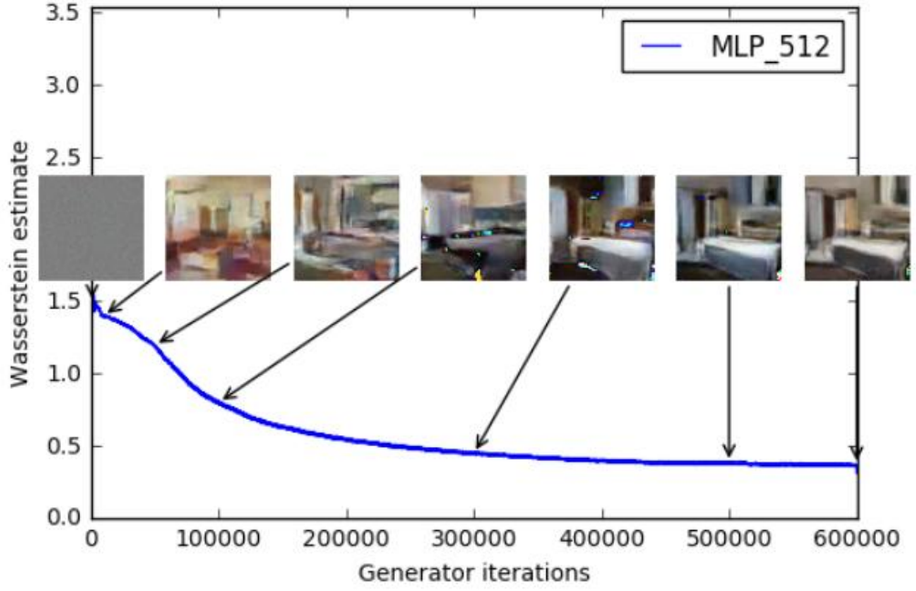
Figure 6: Algorithms trained with a generator without batch normalization and constant number of filters at every layer (as opposed to duplicating them every time as in [18]). Aside from taking out batch normalization, the number of parameters is therefore reduced by a bit more than an order of magnitude. Left: WGAN algorithm. Right: standard GAN formulation. As we can see the standard GAN failed to learn while the WGAN still was able to produce samples.

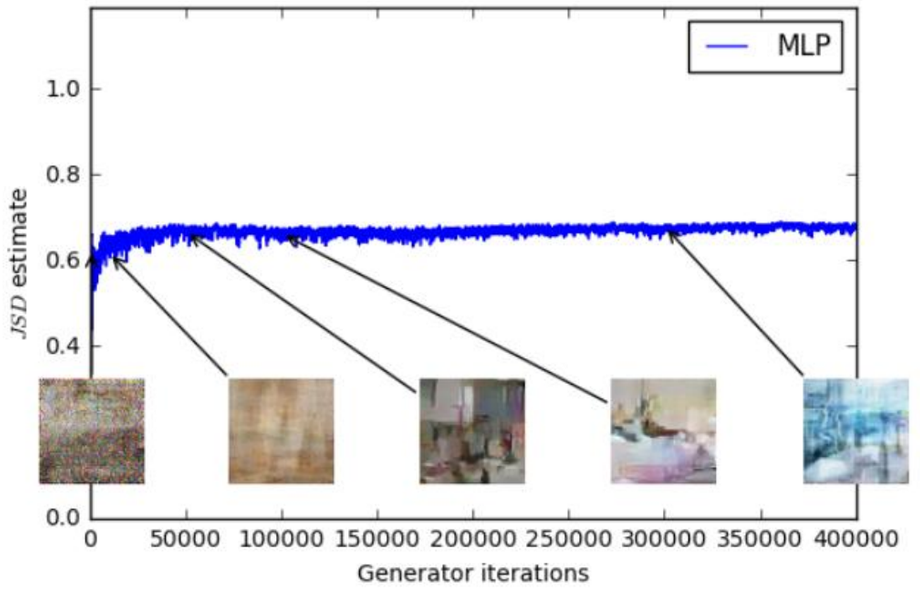
Figure 7: Algorithms trained with an MLP generator with 4 layers and 512 units with ReLU nonlinearities. The number of parameters is similar to that of a DCGAN, but it lacks a strong inductive bias for image generation. Left: WGAN algorithm. Right: standard GAN formulation. The WGAN method still was able to produce samples, lower quality than the DCGAN, and of higher quality than the MLP of the standard GAN. Note the significant degree of mode collapse in the GAN MLP.

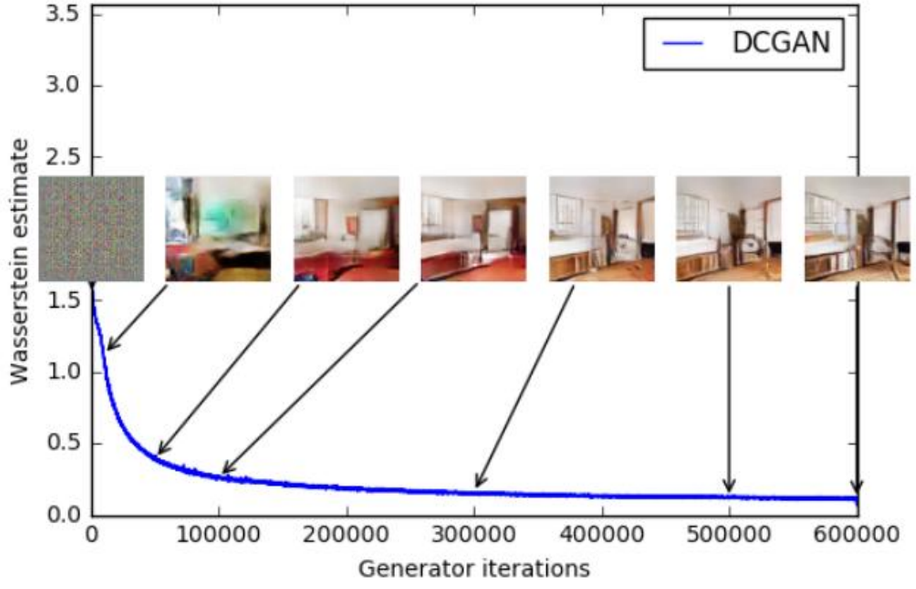
если не применять BN и другие приёмы (ex: удваивание каналов) – всё равно результат «неплохой»

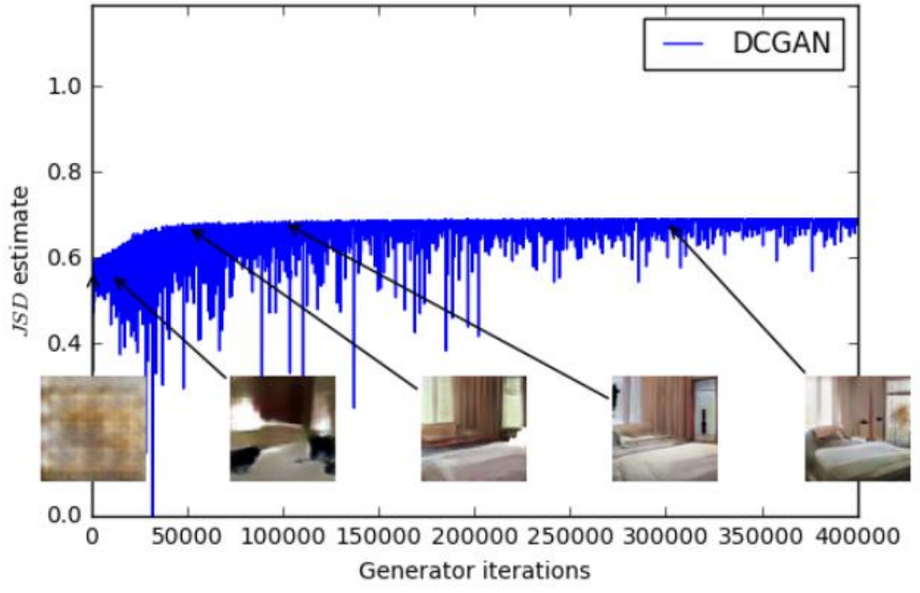
нет mode collapse

Wasserstein GAN: хорошая сходимость до ~min + WE коррелирует с видимым качеством









WGAN-GP

вместо обрезания – мягкий штраф на градиенты – «Gradient Penalty»

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

но штраф хитрый – по точкам из отрезков с концами в разных распределениях
есть вывод, что это обеспечивает липшицевость

нет critic batch normalization

устойчивость к выбору архитектуры

потом ипользовался в SOTA-моделях (Prog GAN, Style GAN и т.п.)

Gulrajani et al «Improved Training of Wasserstein GANs» // https://arxiv.org/pdf/1704.00028.pdf

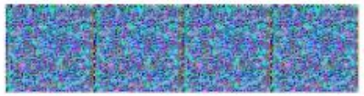
DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)				
G: No BN and a constant number of filters, D: DCGAN				
G: 4-layer 512-dim ReLU MLP, D: DCGAN				
No normalization in either G or D				
Gated multiplicative nonlinearities everywhere in G and D				
tanh nonlinearities everywhere in G and D				
101-layer ResNet G and D				

Figure 2: Different GAN architectures trained with different methods. We only succeeded in training every architecture with a shared set of hyperparameters using WGAN-GP.

WGAN-GP

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

WGAN-GP: исследования на разных архитектурах

успех = inception_score > min_score

Table 1: WGAN-GP's ability to train the architectures in this set.

Nonlinearity (G)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, tanh]
Nonlinearity (D)	[ReLU, LeakyReLU, $\frac{\text{softplus}(2x+2)}{2} - 1$, tanh]
Depth (G)	[4, 8, 12, 20]
Depth (D)	[4, 8, 12, 20]
Batch norm (G)	[True, False]
Batch norm (D; layer norm for WGAN-GP)	[True, False]
Base filter count (G)	[32, 64, 128]
Base filter count (D)	[32, 64, 128]

Table 2: Outcomes of training 200 random architectures, for different success thresholds. For comparison, our standard DCGAN scored 7.24.

Min. score	Only GAN	Only WGAN-GP	Both succeeded	Both failed
1.0	0	8	192	0
3.0	1	88	110	1
5.0	0	147	42	11
7.0	1	104	5	90
9.0	0	0	0	200

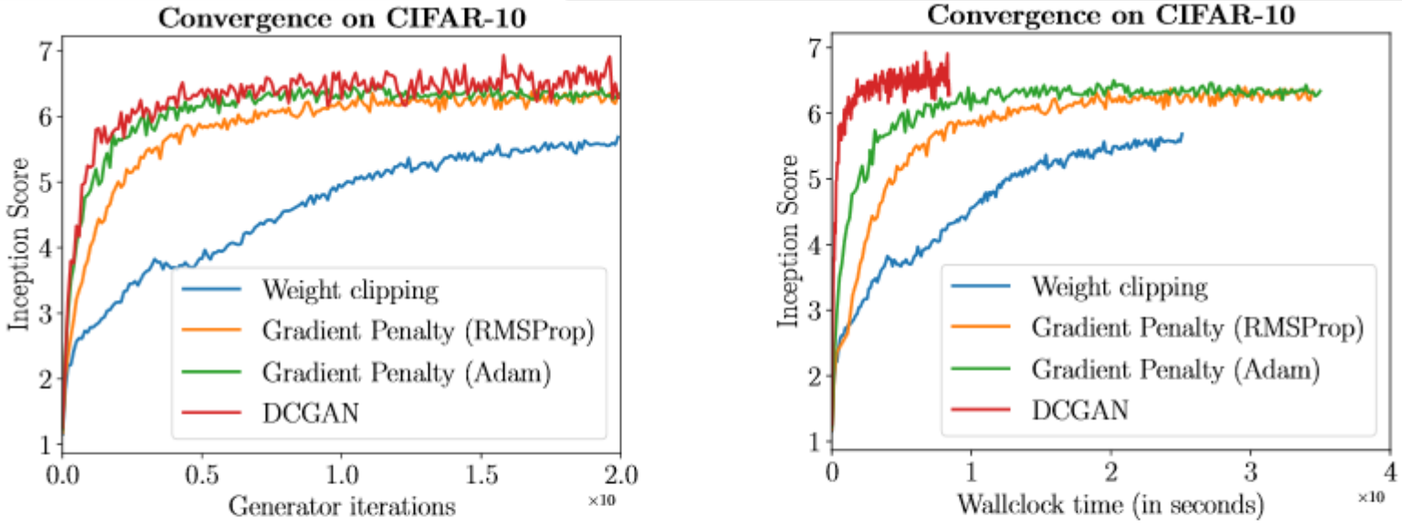


Figure 3: CIFAR-10 Inception score over generator iterations (left) or wall-clock time (right) for four models: WGAN with weight clipping, WGAN-GP with RMSProp and Adam (to control for the optimizer), and DCGAN. WGAN-GP significantly outperforms weight clipping and performs comparably to DCGAN.

GAN: проблемы – Mode-Collapse

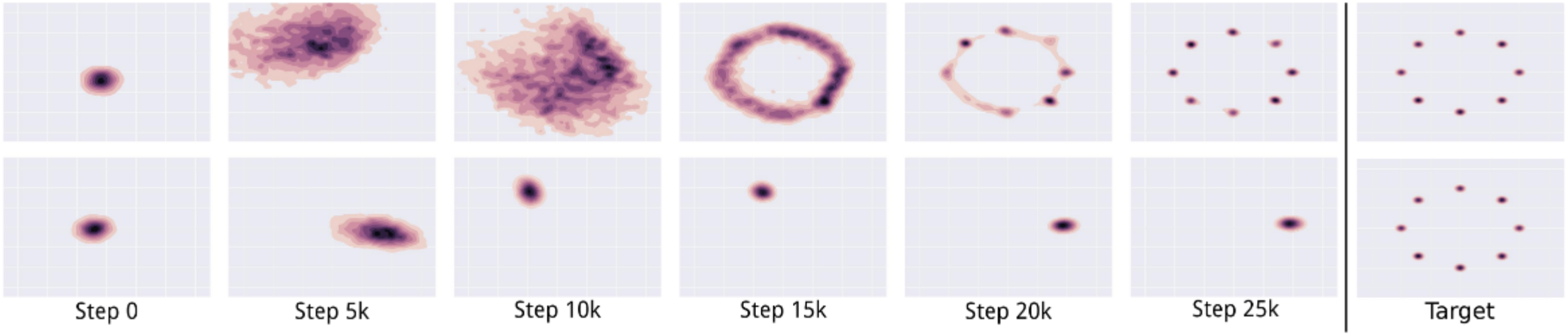


Figure 2: Unrolling the discriminator stabilizes GAN training on a toy 2D mixture of Gaussians dataset. Columns show a heatmap of the generator distribution after increasing numbers of training steps. The final column shows the data distribution. The top row shows training for a GAN with 10 unrolling steps. Its generator quickly spreads out and converges to the target distribution. The bottom row shows standard GAN training. The generator rotates through the modes of the data distribution. It never converges to a fixed distribution, and only ever assigns significant probability mass to a single data mode at once.

нижний ряд – «типичный GAN», нет разнообразия в ответах (sample diversity) **см ниже**

Metz, Luke, et al. «Unrolled Generative Adversarial Networks» //
https://arxiv.org/pdf/1611.02163.pdf

GAN: проблемы – Mode-Collapse

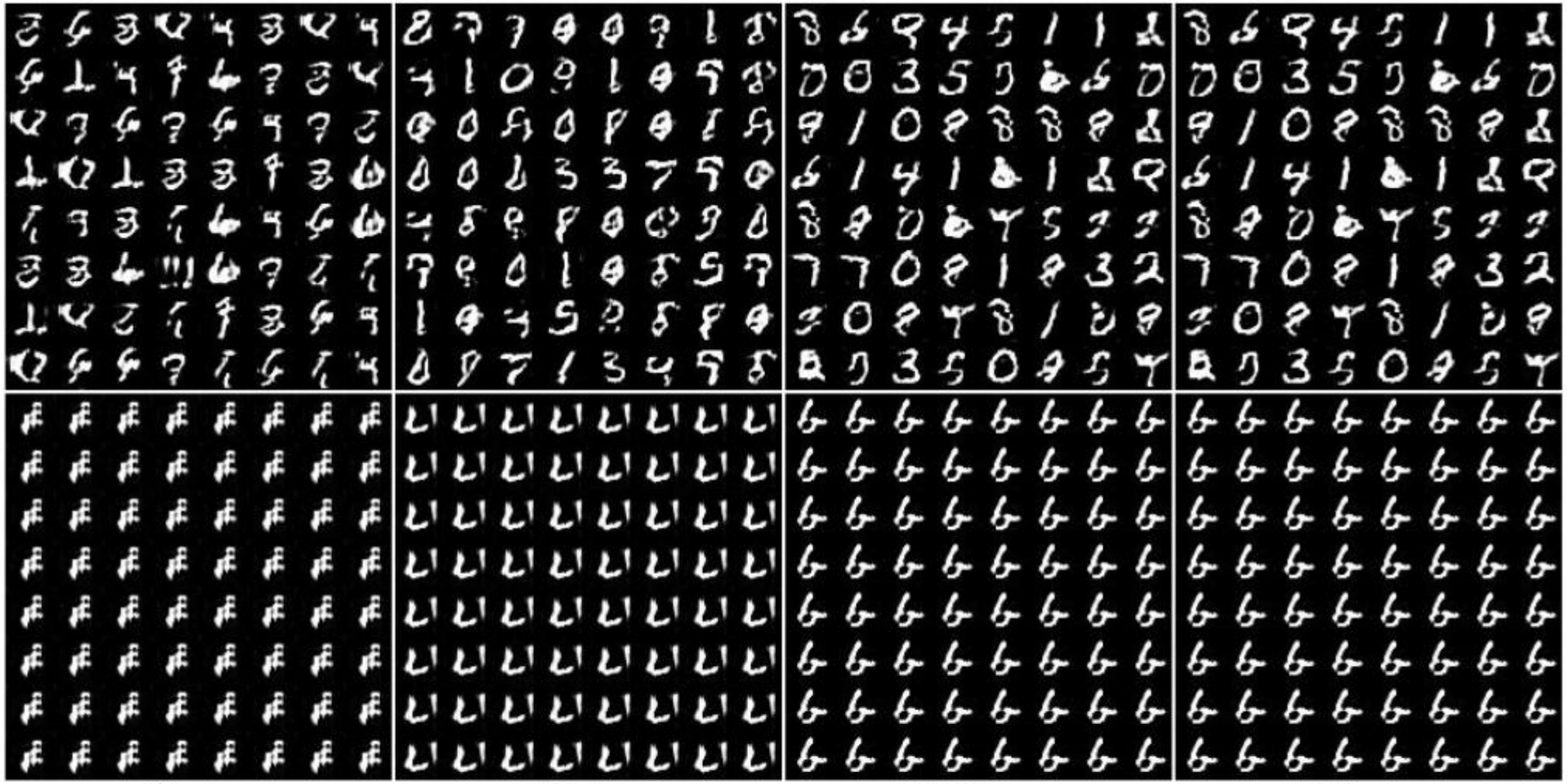


Figure 3: Unrolled GAN training increases stability for an RNN generator and convolutional discriminator trained on MNIST. The top row was run with 20 unrolling steps. The bottom row is a standard GAN, with 0 unrolling steps. Images are samples from the generator after the indicated number of training steps.

GAN: проблемы – Mode-Collapse

решение из статьи – Unrolled GAN

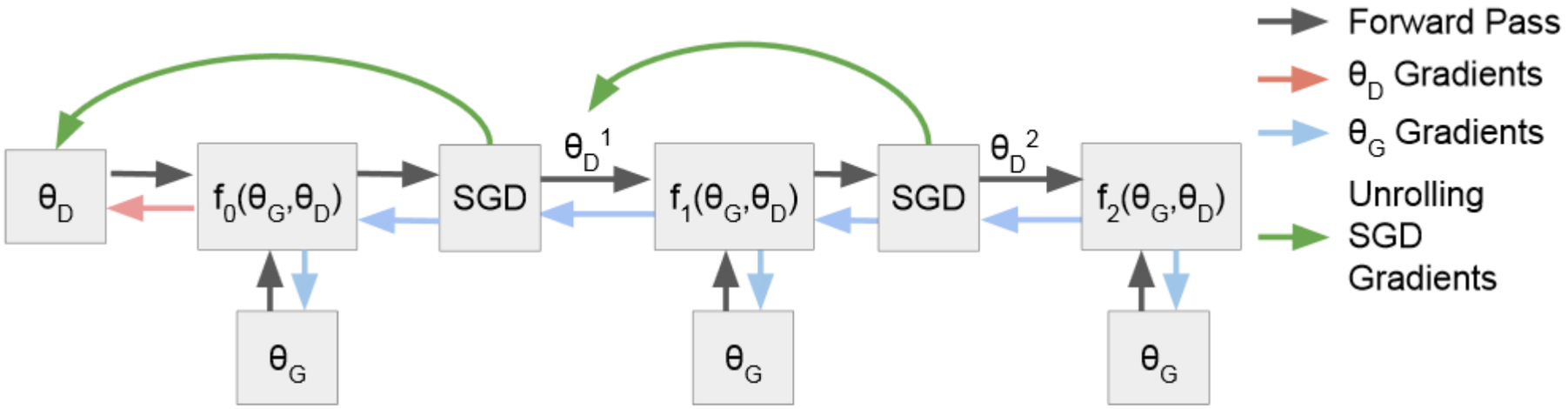


Figure 1: An illustration of the computation graph for an unrolled GAN with 3 unrolling steps. The generator update in Equation 10 involves backpropagating the generator gradient (blue arrows) through the unrolled optimization. Each step k in the unrolled optimization uses the gradients of f_k with respect to θ_D^k, as described in Equation 7 and indicated by the green arrows. The discriminator update in Equation 11 does not depend on the unrolled optimization (red arrow).

ещё решения: организация батчей, W-GANs, LSGANs (было выше) и т.п.

если батч примеров, то дискриминатор видит их однообразность

можно помочь – добавить специальные признаки,

ex: L2-норму разности каждой пары в батче

GAN: проблемы – Mode-Collapse

другое решение – Mixture GAN (MGAN)

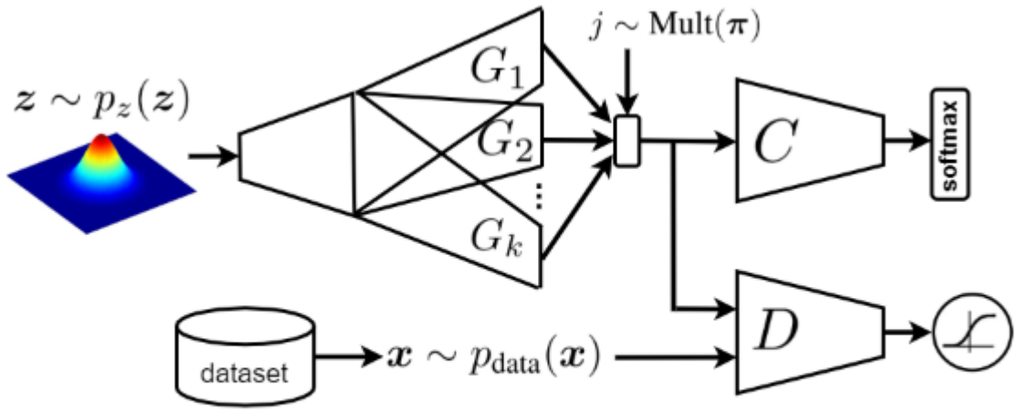


Figure 4. The structure of MGAN.

как бы несколько ГАНов (у каждого своя мода)

Hoang, Quan, et. al. «MGAN: Training generative adversarial nets with multiple generators» // ICLR, 2018.

есть ещё хитрое решение – Bourgain GAN (BourGAN)
(тоже мультимодальность, но в латентном пространстве)

Особенности GAN

генерацию можно распараллелить

нельзя в авторегрессии

визуально кажется лучше других методов

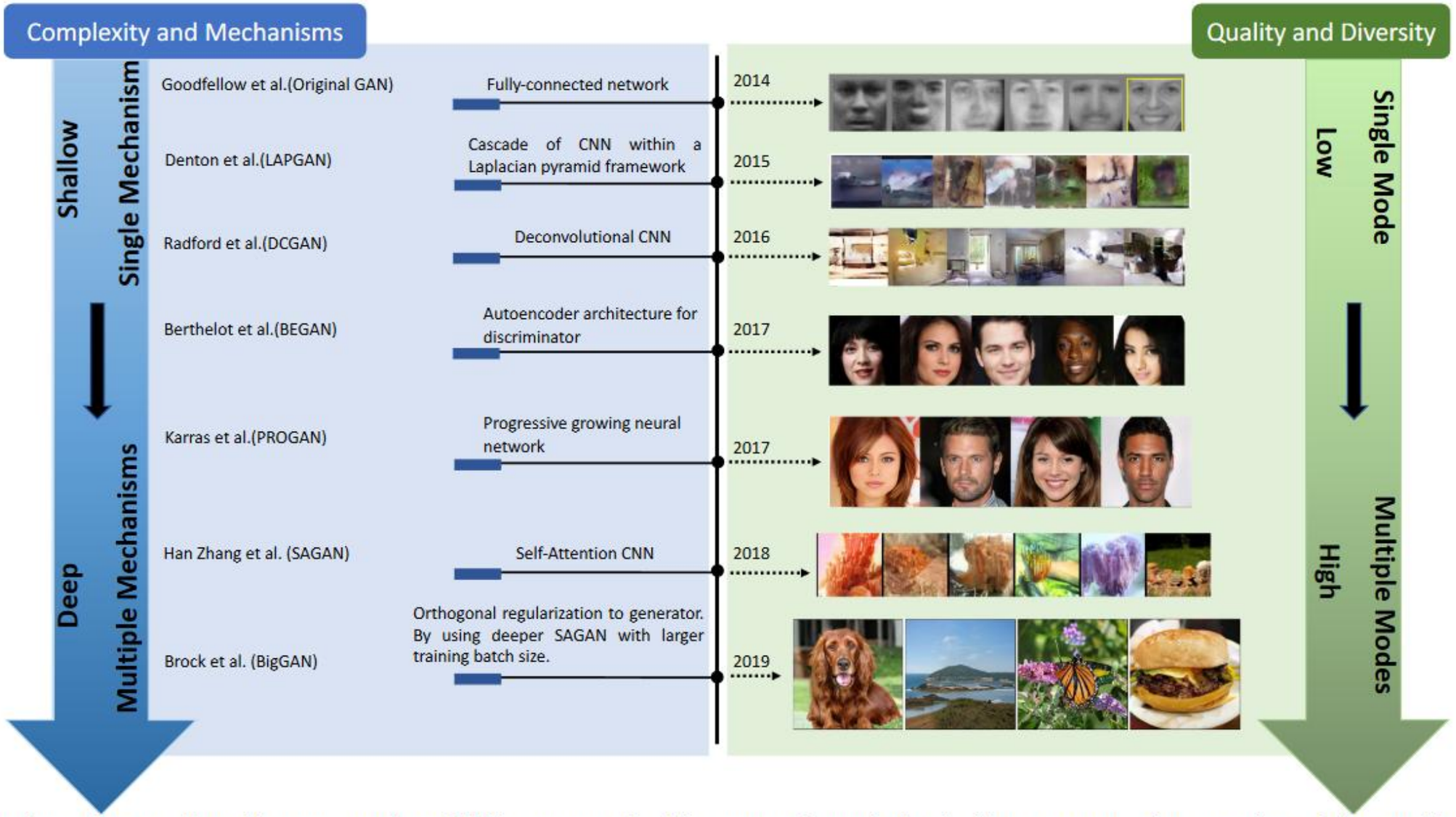


Fig. 3. Timeline of part of architecture-variant GANs present in this paper. Complexity in blue stream refers to size of the architecture and computational cost such as batch size. Mechanisms refer to the number of types of models used in the architecture (e.g., BEGAN uses an autoencoder architecture for its discriminator while a deconvolutional neural network is used for the generator. In this case, two mechanisms are used).

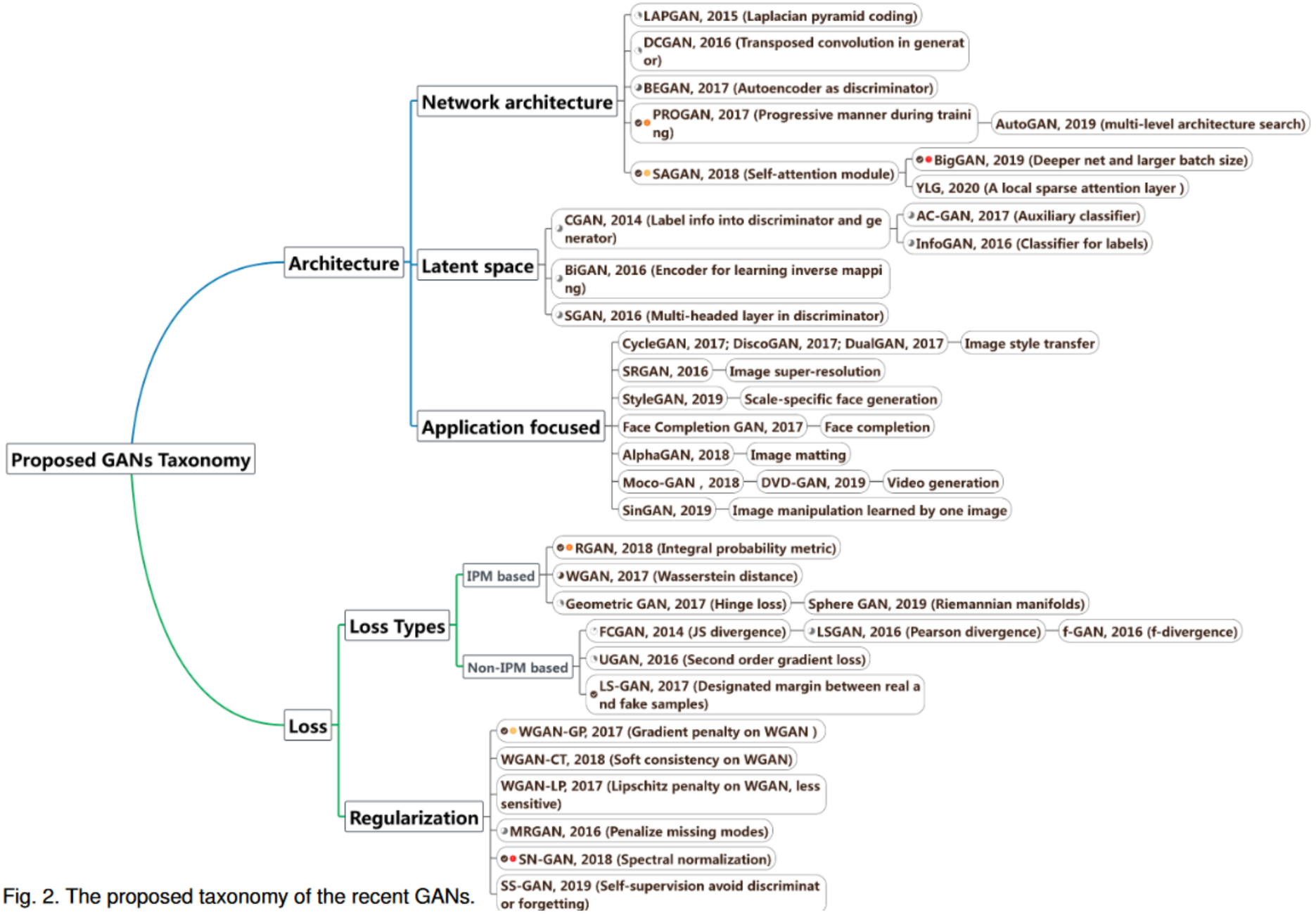


Fig. 2. The proposed taxonomy of the recent GANs.

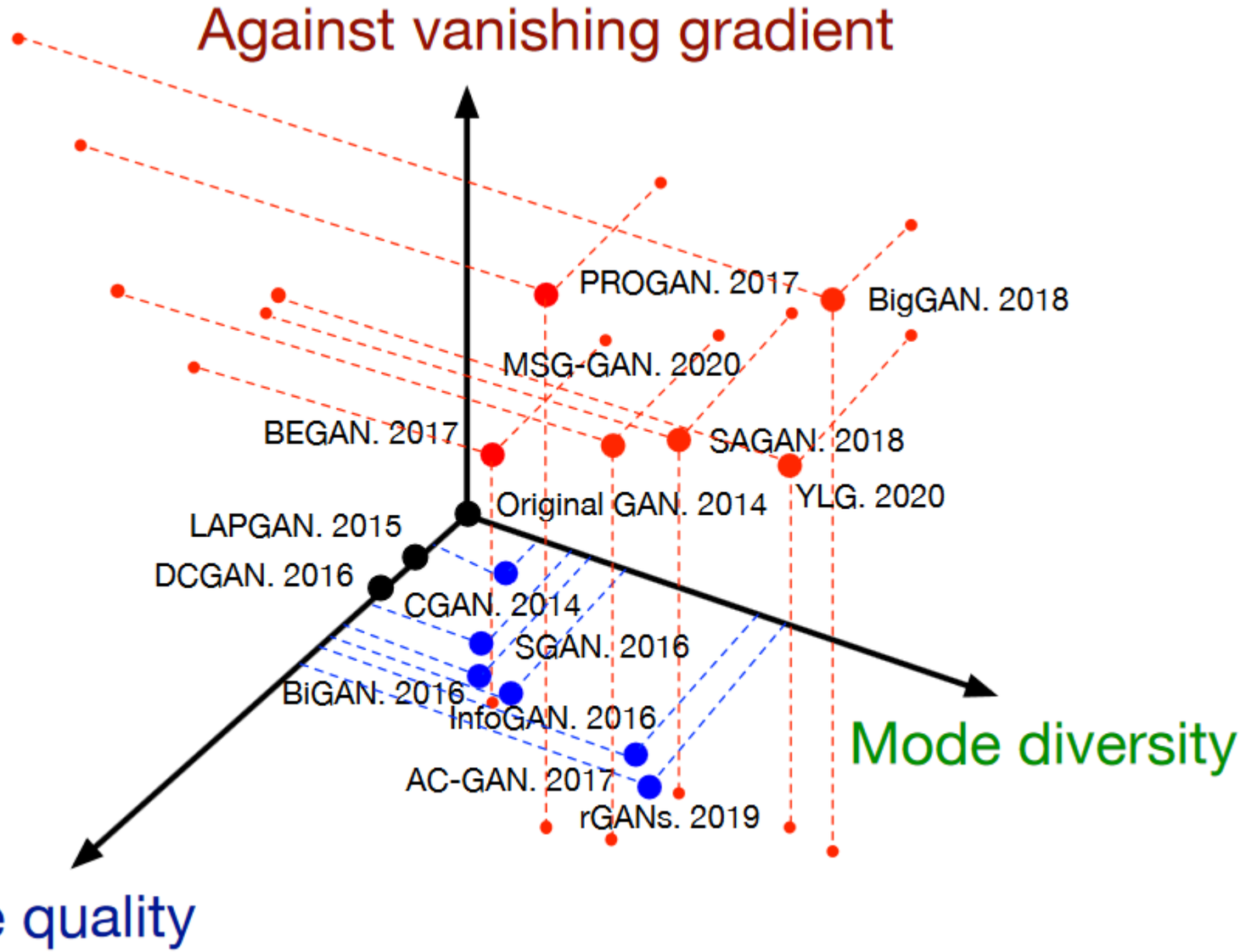


Fig. 18. Summary of recent architecture-variant GANs for solving the three challenges. The challenges are categorized by three orthogonal axes. A larger value for each axis indicates better performance. Red points indicate GAN-variants which cover all three challenges, blue points cover two, and black points cover only one challenge. Quantitative results can be referred to Table 5 in section 8.

Deep Convolutional Generative Adversarial Networks (DCGAN)

- **Полносвёрточная**

- **Нет пулинга**

в дискриминаторе – strided convolutions

в генераторе – fractional-strided convolutions

- **BN после каждого слоя**

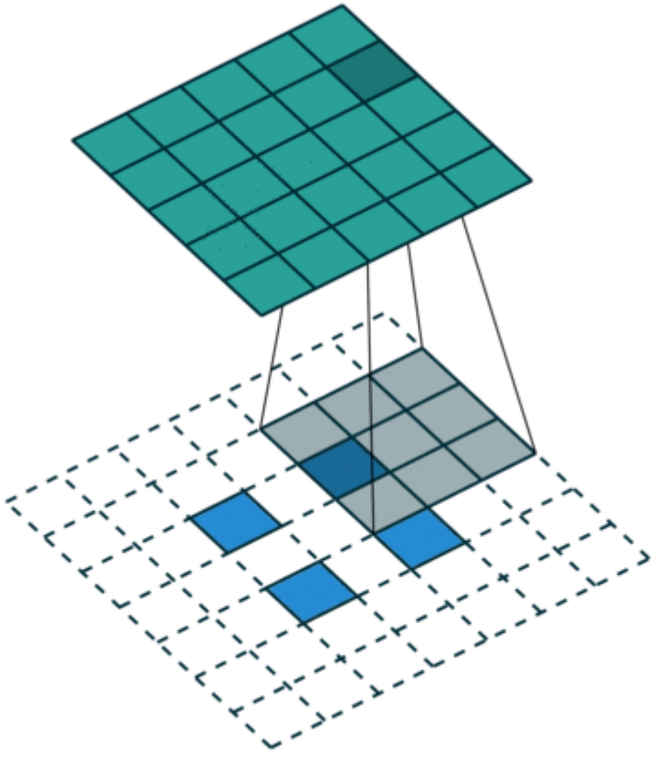
кроме последнего G и первого D

- **FC скрытые слои → свёртки**

- **активации**

генератор: ReLU для скрытых слоёв, Tanh для выходного слоя

дискриминатор: LeakyReLU + sigmoid



fractional-strided convolutions

Впервые удалось использовать CNN для порождения реалистичных картинок

Alec Radford, Luke Metz, Soumith Chintala «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks» https://arxiv.org/abs/1511.06434

Deep Convolutional Generative Adversarial Networks (DCGAN)

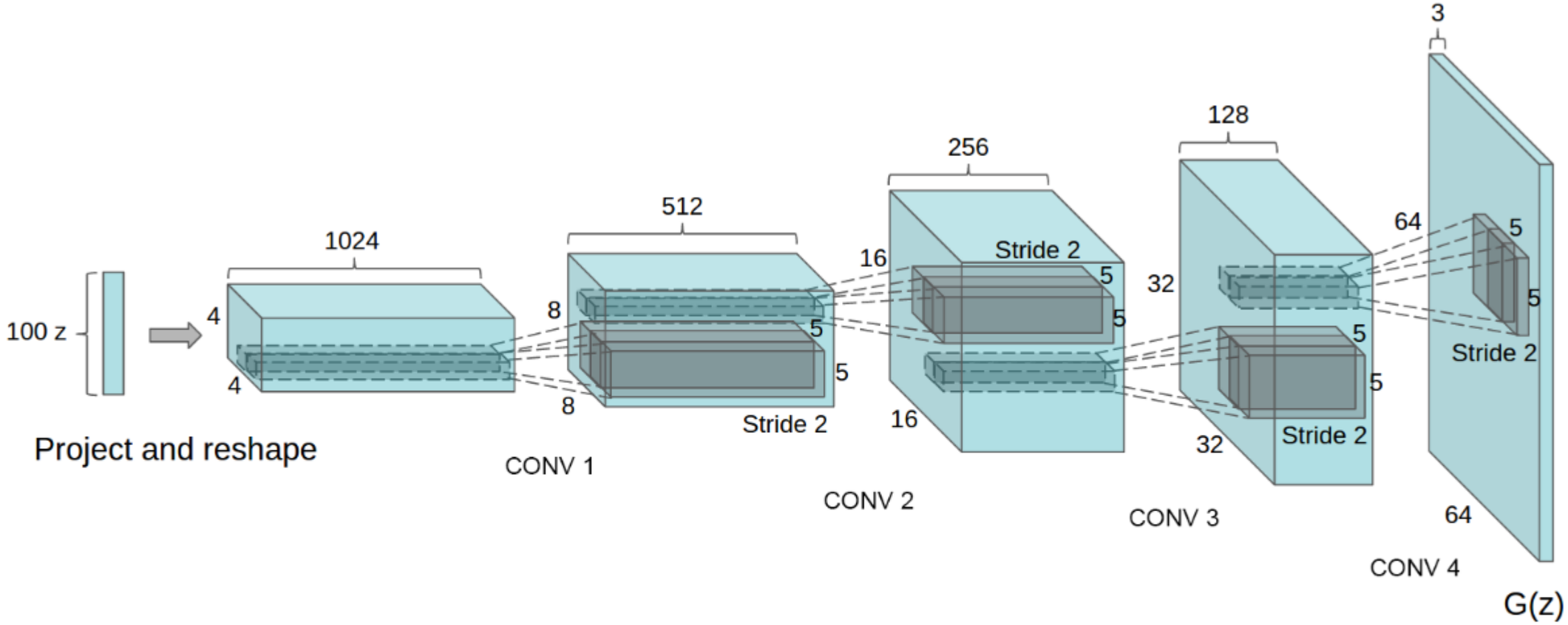


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

Deep Convolutional Generative Adversarial Networks (DCGAN)

Figure 9: Side-by-side illustration of (from left-to-right) the MNIST dataset, generations from a baseline GAN, and generations from our DCGAN .

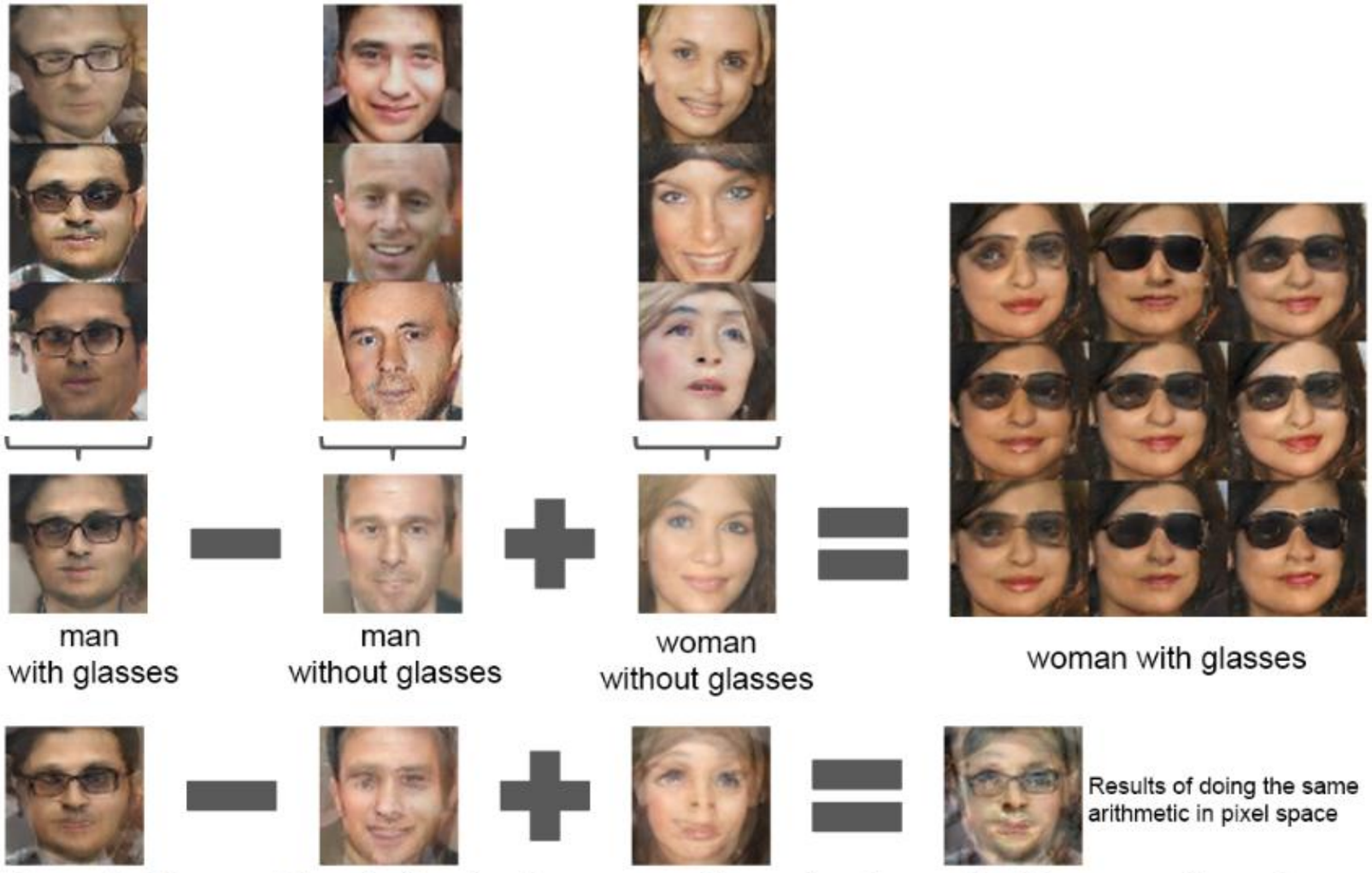


Figure 7: Vector arithmetic for visual concepts. For each column, the Z vectors of samples are averaged. Arithmetic was then performed on the mean vectors creating a new vector Y. The center sample on the right hand side is produce by feeding Y as input to the generator. To demonstrate the interpolation 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.

Figure 8: A ”turn” vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

DCGAN: обложки дисков

Условные состязательные сети (cGAN)

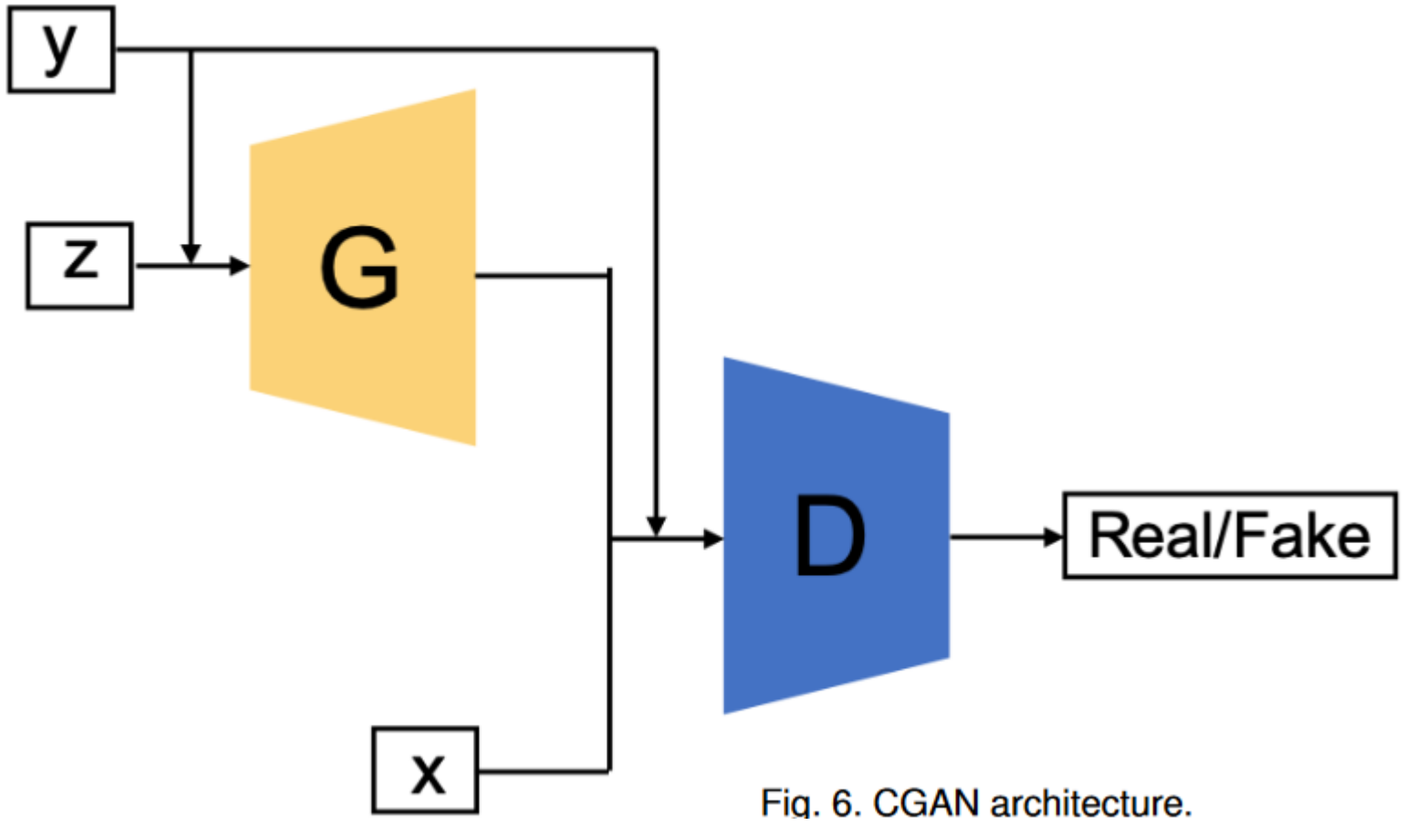
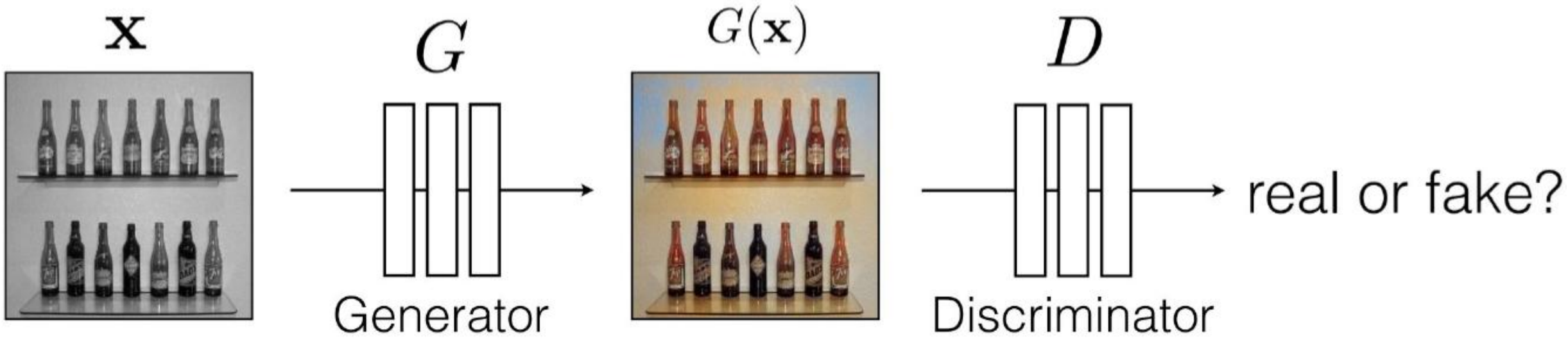


Fig. 6. CGAN architecture.

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_r} \log[D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z}\sim p_\mathbf{z}} \log[1 - D(G(\mathbf{z}|\mathbf{y}))]$$

https://arxiv.org/pdf/1906.01529.pdf

Условные состязательные сети (cGAN)

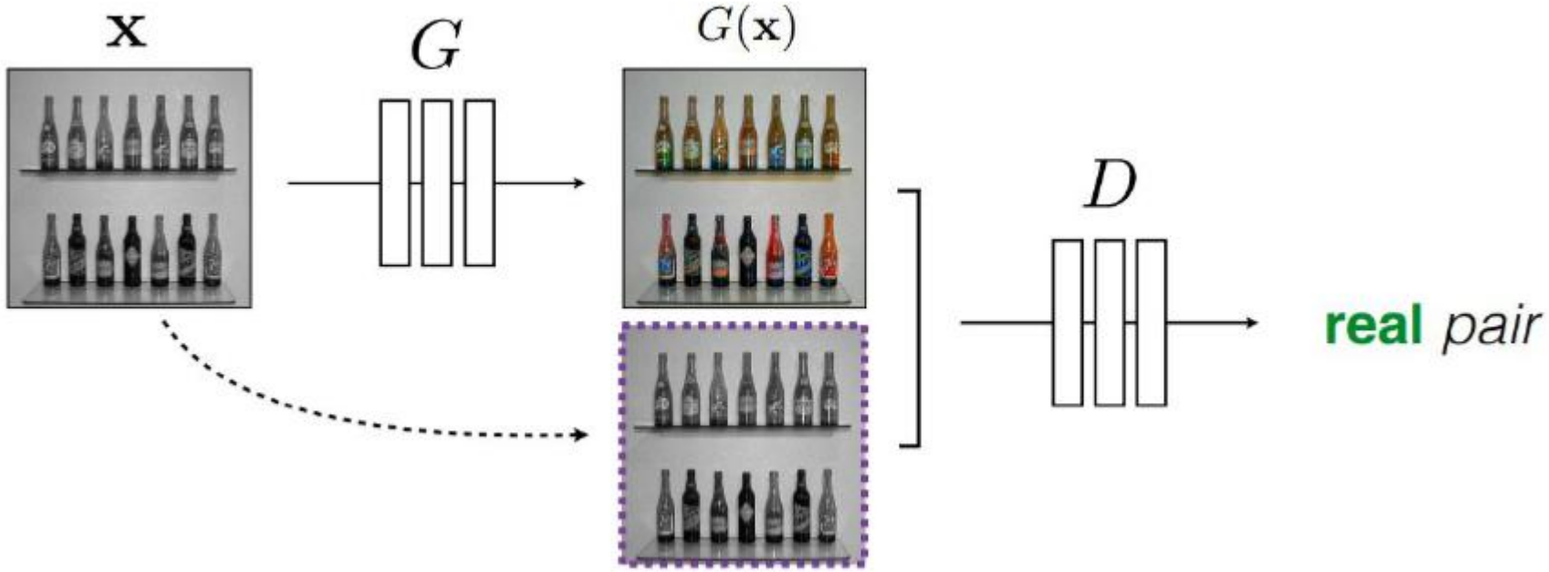


но тут проблема, что сеть начнёт генерировать что-то реальное, например «котиков» и условие будет фиктивным

т.е. тут м.б. отображение не из шума

[Phillip Isola]

Условные состязательные сети (cGAN)



лучше определять реальность/фейковость пары

Pix2pix с условными состязательными сетями (cGAN)

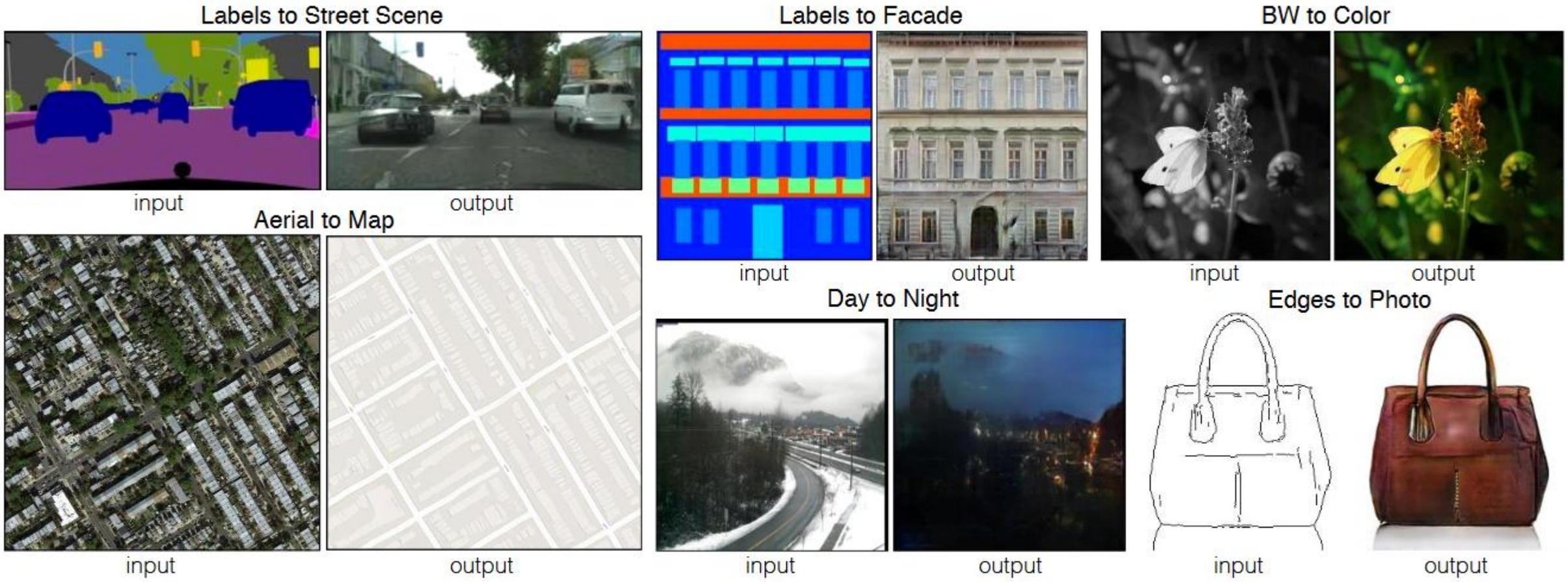


Figure 1: Many problems in image processing, graphics, and vision involve translating an input image into a corresponding output image. These problems are often treated with application-specific algorithms, even though the setting is always the same: map pixels to pixels. Conditional adversarial nets are a general-purpose solution that appears to work well on a wide variety of these problems. Here we show results of the method on several. In each case we use the same architecture and objective, and simply train on different data.

Pix2pix с условными состязательными сетями (cGAN)

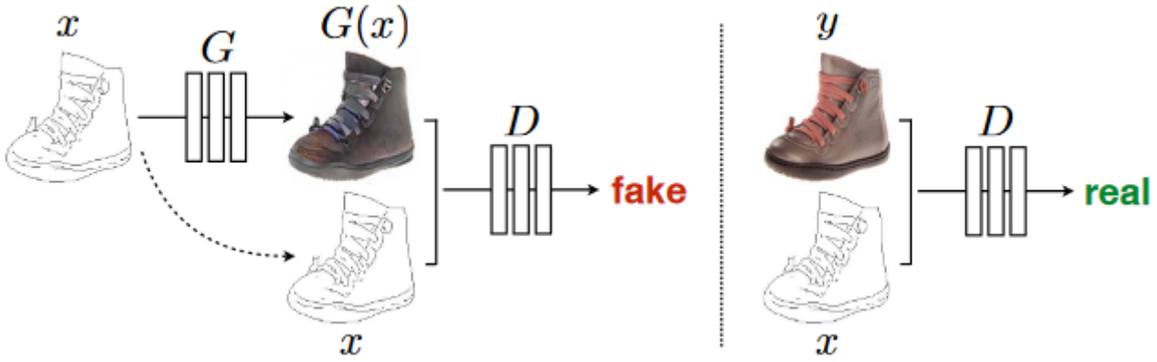


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

нужна размеченная выборка (пары изображений), поэтому такие примеры

- **силуэт**
- **сегментация**
- **стилизация**

U-net для генератора

PatchGAN для дискриминатора – классифицирует кусочки, на всём изображении L1-ошибка

вместо определения фейковости всего изображения смотреть на патчи – быстрее, можно применять к большим изображениям

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x,G(x,z))],$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1] \quad G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G).$$

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros «Image-to-Image Translation with Conditional Adversarial Nets» // CVPR 2017, https://phillipi.github.io/pix2pix/

Figure 4: Different losses induce different quality of results. Each column shows results trained under a different loss. Please see https://phillipi.github.io/pix2pix/ for additional examples.

Figure 8: Example results on Google Maps at 512x512 resolution (model was trained on images at 256 × 256 resolution, and run convolutionally on the larger images at test time). Contrast adjusted for clarity.

Figure 16: Example results of our method on automatically detected edges→handbags, compared to ground truth.

Pix2pix с условными состязательными сетями (cGAN)

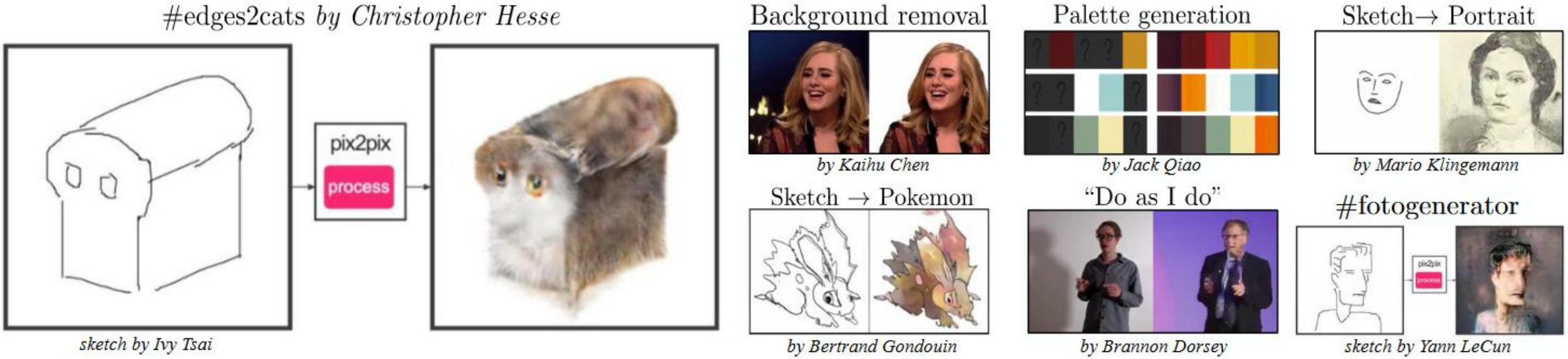


Figure 11: Example applications developed by online community based on our `pix2pix` codebase: *#edges2cats* [3] by Christopher Hesse, *Background removal* [6] by Kaihu Chen, *Palette generation* [5] by Jack Qiao, *Sketch → Portrait* [7] by Mario Klingemann, *Sketch→ Pokemon* [1] by Bertrand Gondouin, *"Do As I Do" pose transfer* [2] by Brannon Dorsey, and *#fotogenerator* by Bosman et al. [4].

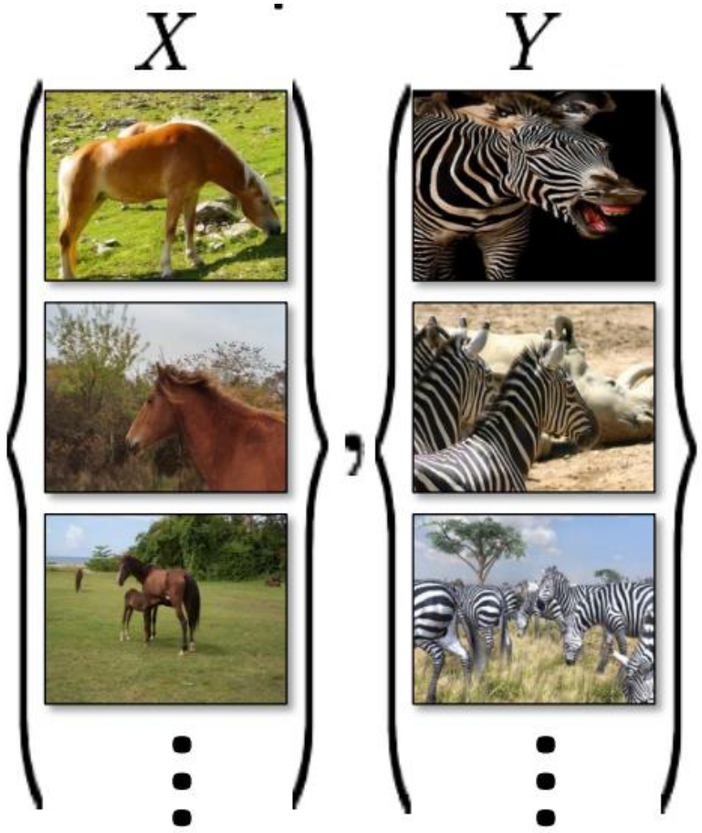
различные результаты сообщества с помощью разработанной библиотеки

Проблема отсутствия выборки

есть размеченная выборка для pix → pix

или может быть дёшево получена, как в рассмотренном примере (получение контуров)

есть просто представители двух множест



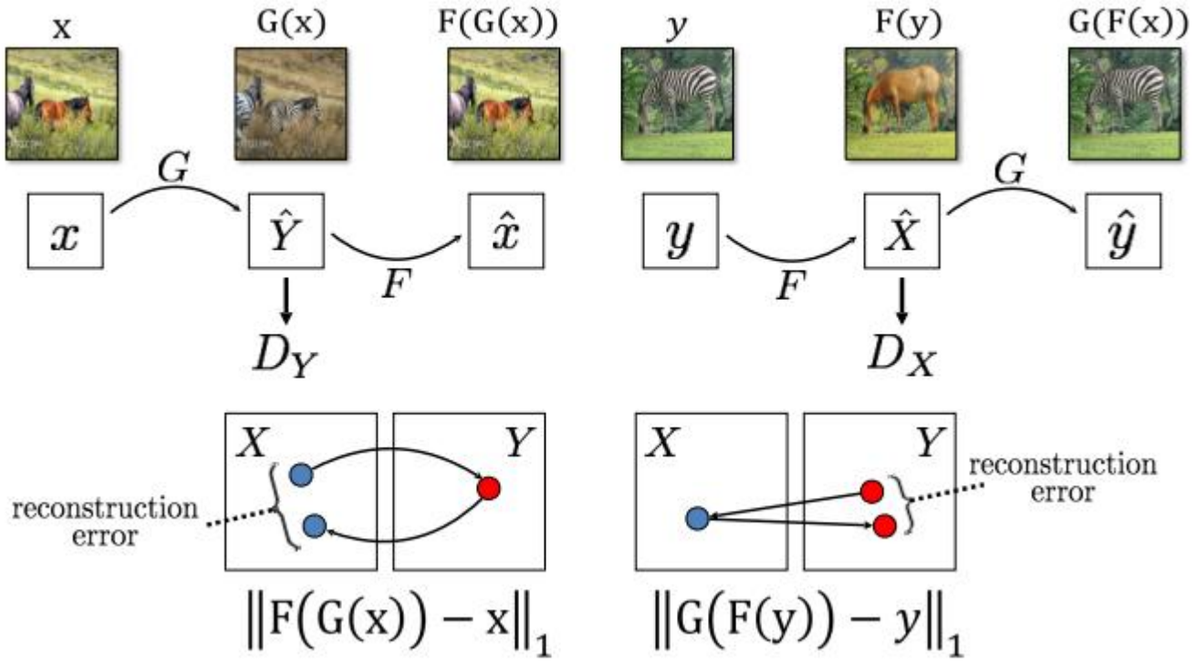
можно ли осуществить трансфер?

CycleGAN: перенос стиля без размеченных данных (пар картинок)

Идея: два генератора (в разные стороны)

Надо, чтобы после работы их подряд получалась исходная картинка

Cycle Consistency Loss



Zhu et al., «Unpaired Image-To-Image Translation Using Cycle-Consistent Adversarial Networks», ICCV 2017

https://junyanz.github.io/CycleGAN/

CycleGAN

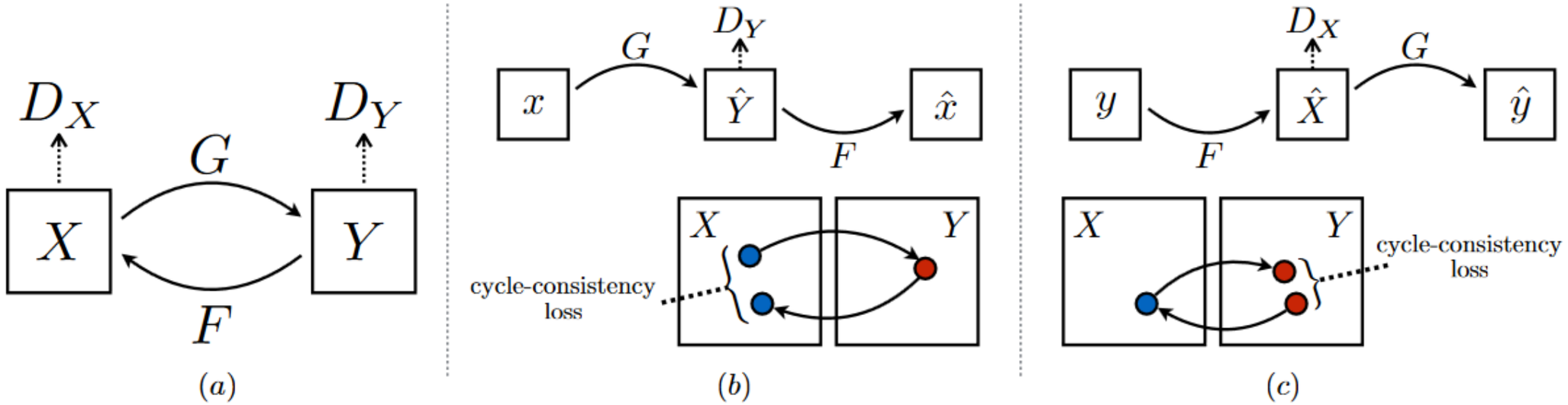


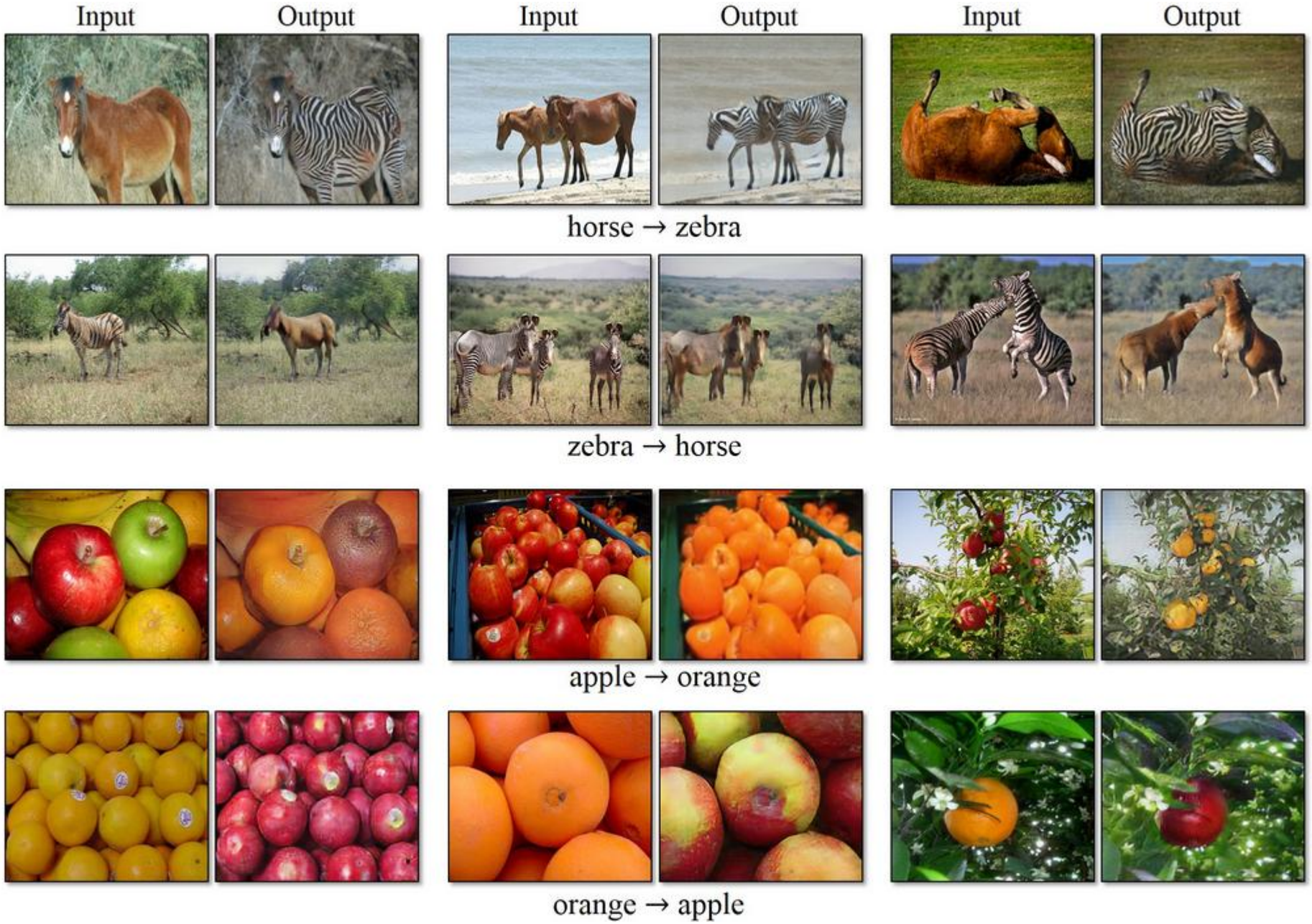
Figure 3: (a) Our model contains two mapping functions $G : X \to Y$ and $F : Y \to X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \to G(x) \to F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \to F(y) \to G(F(y)) \approx y$

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))], \quad (1)$$

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]. \quad (2)$$

Our full objective is:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F), \quad (3)$$


Input
Output
Input
Output
Input
Output
horse → zebra
zebra → horse
apple → orange
orange → apple

CycleGAN

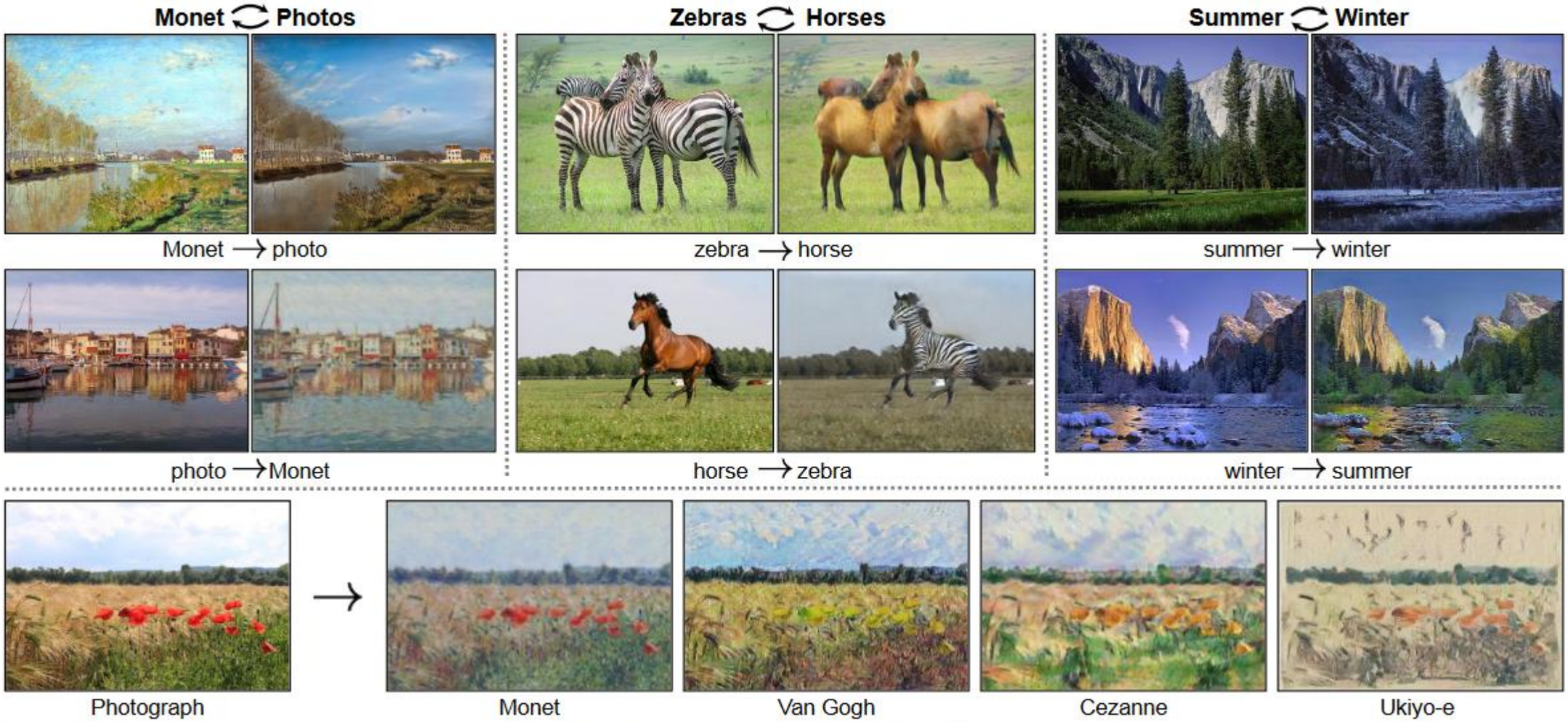


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

CycleGAN: неудачные примеры

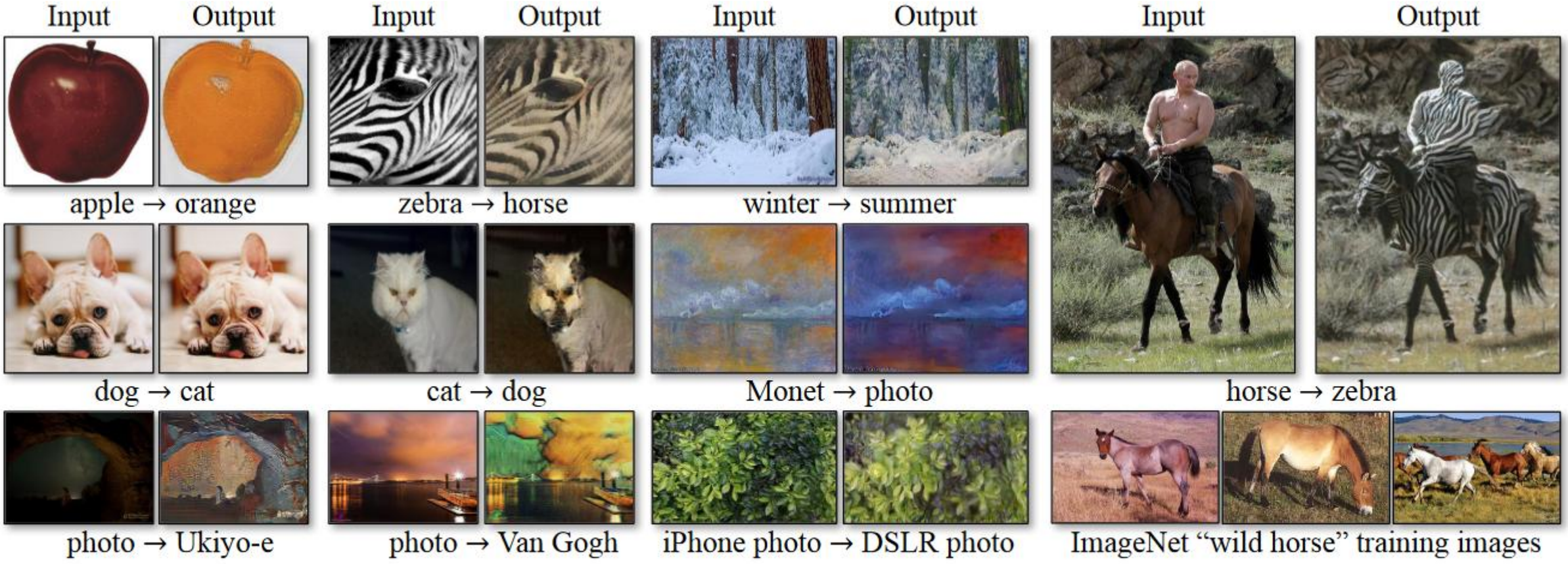


Figure 17: Typical failure cases of our method. Left: in the task of dog→cat transfiguration, CycleGAN can only make minimal changes to the input. Right: CycleGAN also fails in this horse → zebra example as our model has not seen images of horseback riding during training. Please see our website for more comprehensive results.

ProGAN (NVIDIA)

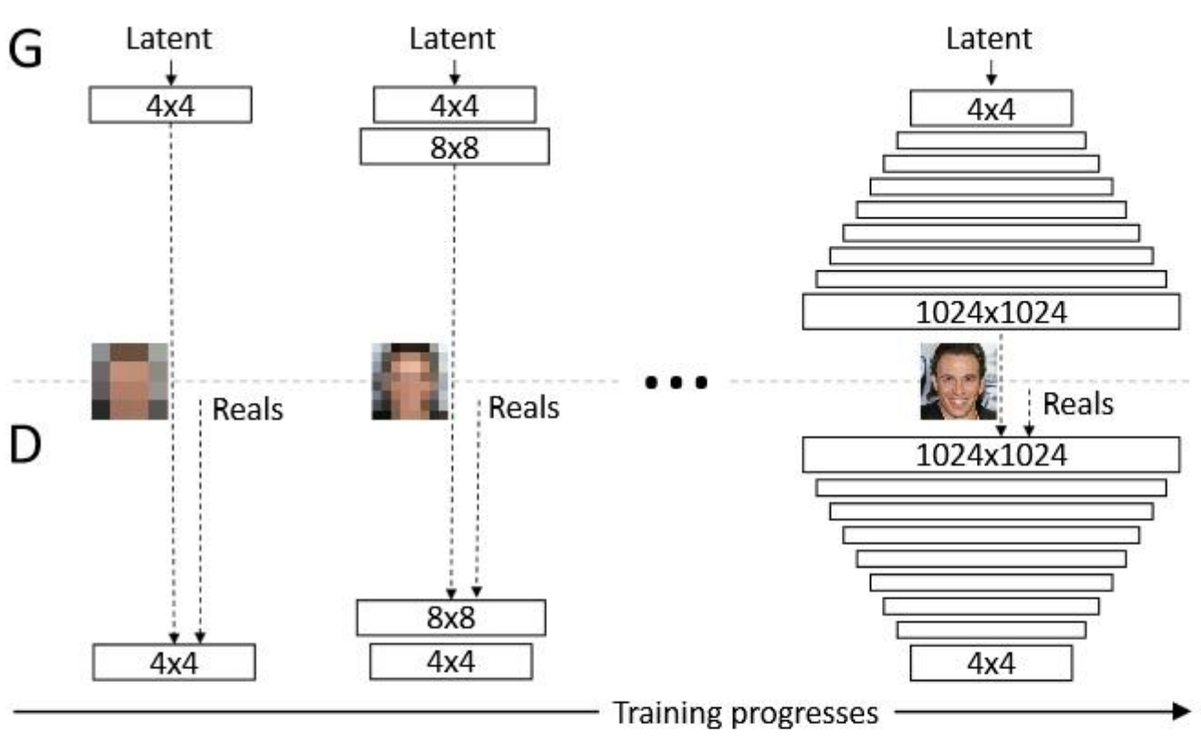


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ 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.

Karras et al., «Progressive Growing of GANs for Improved Quality, Stability, and Variation» // ICLR 2018 https://arxiv.org/abs/1710.10196

ProGAN (NVIDIA)

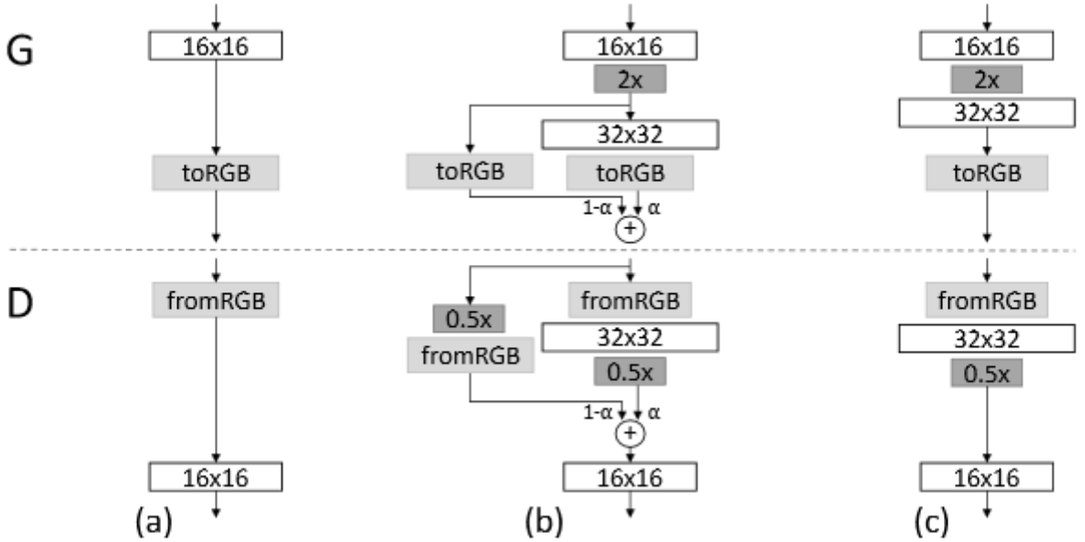


Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from 16×16 images (a) to 32×32 images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1. Here $2\times$ and $0.5\times$ refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The toRGB represents a layer that projects feature vectors to RGB colors and fromRGB does the reverse; both use 1×1 convolutions. When training the discriminator, we feed in real images that are downscaled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

прокидывание связи с меняющимся весом → лучше доучиваться до большего размера

Поэтапно учимся создавать картинки в 4 раза больше до размера 1024×1024

ProGAN (NVIDIA)

Generator	Act.	Output shape	Params
Latent vector	–	512 × 1 × 1	–
Conv 4 × 4	LReLU	512 × 4 × 4	4.2M
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Upsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Upsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Upsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Upsample	–	512 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	1.2M
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Upsample	–	256 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	295k
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Upsample	–	128 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	74k
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Upsample	–	64 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	18k
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Upsample	–	32 × 1024 × 1024	–
Conv 3 × 3	LReLU	16 × 1024 × 1024	4.6k
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 1 × 1	linear	3 × 1024 × 1024	51
Total trainable parameters			**23.1M**

Discriminator	Act.	Output shape	Params
Input image	–	3 × 1024 × 1024	–
Conv 1 × 1	LReLU	16 × 1024 × 1024	64
Conv 3 × 3	LReLU	16 × 1024 × 1024	2.3k
Conv 3 × 3	LReLU	32 × 1024 × 1024	4.6k
Downsample	–	32 × 512 × 512	–
Conv 3 × 3	LReLU	32 × 512 × 512	9.2k
Conv 3 × 3	LReLU	64 × 512 × 512	18k
Downsample	–	64 × 256 × 256	–
Conv 3 × 3	LReLU	64 × 256 × 256	37k
Conv 3 × 3	LReLU	128 × 256 × 256	74k
Downsample	–	128 × 128 × 128	–
Conv 3 × 3	LReLU	128 × 128 × 128	148k
Conv 3 × 3	LReLU	256 × 128 × 128	295k
Downsample	–	256 × 64 × 64	–
Conv 3 × 3	LReLU	256 × 64 × 64	590k
Conv 3 × 3	LReLU	512 × 64 × 64	1.2M
Downsample	–	512 × 32 × 32	–
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Conv 3 × 3	LReLU	512 × 32 × 32	2.4M
Downsample	–	512 × 16 × 16	–
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Conv 3 × 3	LReLU	512 × 16 × 16	2.4M
Downsample	–	512 × 8 × 8	–
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Conv 3 × 3	LReLU	512 × 8 × 8	2.4M
Downsample	–	512 × 4 × 4	–
Minibatch stddev	–	513 × 4 × 4	–
Conv 3 × 3	LReLU	512 × 4 × 4	2.4M
Conv 4 × 4	LReLU	512 × 1 × 1	4.2M
Fully-connected	linear	1 × 1 × 1	513
Total trainable parameters			**23.1M**

Table 2: Generator and discriminator that we use with CELEBA-HQ to generate 1024×1024 images.

ProGAN: хаки, которые применялись

Training configuration	CELEBA						LSUN BEDROOM					
	Sliced Wasserstein distance $\times 10^3$					MS-SSIM	Sliced Wasserstein distance $\times 10^3$					MS-SSIM
	128	64	32	16	Avg		128	64	32	16	Avg	
(a) Gulrajani et al. (2017)	12.99	7.79	7.62	8.73	9.28	0.2854	11.97	10.51	8.03	14.48	11.25	**0.0587**
(b) + Progressive growing	4.62	**2.64**	3.78	6.06	4.28	**0.2838**	7.09	6.27	7.40	9.64	7.60	0.0615
(c) + Small minibatch	75.42	41.33	41.62	26.57	46.23	0.4065	72.73	40.16	42.75	42.46	49.52	0.1061
(d) + Revised training parameters	9.20	6.53	4.71	11.84	8.07	0.3027	7.39	5.51	3.65	9.63	6.54	0.0662
(e*) + Minibatch discrimination	10.76	6.28	6.04	16.29	9.84	0.3057	10.29	6.22	5.32	11.88	8.43	0.0648
(e) Minibatch stddev	13.94	5.67	2.82	5.71	7.04	0.2950	7.77	5.23	3.27	9.64	6.48	0.0671
(f) + Equalized learning rate	4.42	3.28	2.32	7.52	4.39	0.2902	**3.61**	3.32	**2.71**	6.44	4.02	0.0668
(g) + Pixelwise normalization	**4.06**	3.04	**2.02**	**5.13**	**3.56**	0.2845	3.89	**3.05**	3.24	**5.87**	**4.01**	0.0640
(h) Converged	2.42	2.17	2.24	4.99	2.96	0.2828	3.47	2.60	2.30	4.87	3.31	0.0636

Table 1: Sliced Wasserstein distance (SWD) between the generated and training images (Section 5) and multi-scale structural similarity (MS-SSIM) among the generated images for several training setups at 128×128. For SWD, each column represents one level of the Laplacian pyramid, and the last one gives an average of the four distances.

(a) (b) (c) (d) (e*) (e) (f) (g) (h) Converged

Figure 3: (a) – (g) CELEBA examples corresponding to rows in Table 1. These are intentionally non-converged. (h) Our converged result. Notice that some images show aliasing and some are not sharp – this is a flaw of the dataset, which the model learns to replicate faithfully.

Оценивать разнообразие в батчах

Нормализация весов

Нормализация по пикселям в генераторе

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

ProGAN (NVIDIA)

Mao et al. (2016b) (128 × 128) Gulrajani et al. (2017) (128 × 128) Our (256 × 256)

Figure 6: Visual quality comparison in LSUN BEDROOM; pictures copied from the cited articles.

Различные виды GAN

перечень видов со ссылками на первоисточники

https://github.com/hindupuravinash/the-gan-zoo

https://github.com/wiseodd/generative-models

https://github.com/hwalsuklee/tensorflow-generative-model-collections

Ian Goodfellow «NIPS 2016 Tutorial:Generative Adversarial Networks»

https://arxiv.org/pdf/1701.00160.pdf

Поиграться с GAN

https://poloclub.github.io/ganlab/

Jie Gui et al «A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications» // https://arxiv.org/pdf/2001.06937.pdf