Лекция 8. GAN. Генеративные состязательные нейросети

Бабич Павел Васильевич

ИММиКН

2026

Цели лекции и маршрут

1. Понять ограничения likelihood-based моделей и мотивацию GAN.
2. Вывести minimax-цель, оптимальный дискриминатор и связь с JS-дивергенцией.
3. Разобрать нестабильность обучения и Wasserstein GAN.
4. Сравнить DCGAN, conditional GAN, Pix2Pix, CycleGAN и StyleGAN.
5. Реализовать основные компоненты на PyTorch.

Маршрут: VAE $\rightarrow$ другие семейства моделей $\rightarrow$ GAN $\rightarrow$ условная генерация и перенос домена.

Напоминание: что оптимизирует VAE?

Латентная модель: $z \sim p(z)$, $x \sim p_\theta(x \mid z)$.

$$\log p_\theta(x) \geq \underbrace{\mathbb{E}_{q_\phi(z|x)} \log p_\theta(x \mid z)}_{\text{качество реконструкции}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z))}_{\text{регуляризация}}.$$

- Для обучения нужно выбрать вероятностную модель $p_\theta(x \mid z)$.
- ELBO является нижней оценкой, а не точным log-likelihood.
- Простая пиксельная модель наблюдений может неадекватно описывать сложные изображения.

Почему хорошая реконструкция не равна реализму?

Если $p_\theta(x \mid z) = \mathcal{N}(\mu_\theta(z), \sigma^2 I)$, то

$$-\log p_\theta(x \mid z) = \frac{1}{2\sigma^2}\|x - \mu_\theta(z)\|_2^2 + C.$$

При неоднозначном восстановлении минимум ожидаемого MSE достигается условным средним:

$$\arg\min_a \mathbb{E}[\|X - a\|_2^2 \mid z] = \mathbb{E}[X \mid z].$$

Интуиция

Усреднение нескольких правдоподобных вариантов может дать визуально неправдоподобное изображение. Это проблема конкретного likelihood и ограничений модели, а не теорема о всех VAE.

Какие альтернативы существовали?

Семейство	**Принцип обучения**	**Компромисс**
Autoregressive	$\sum_j \log p_\theta(x_j \mid x_{<j})$	Точный likelihood, последовательное сэмплирование
VAE	Вариационная нижняя оценка	Быстрое сэмплирование, приближённый вывод
Normalizing flows	Замена переменных	Точная плотность, обратимость
Energy-based	Энергия $E_\theta(x)$	Сложность нормировки и сэмплирования
GAN	Дискриминатор различает выборки	Нет необходимости вычислять $p_\theta(x)$

Diffusion/score-based модели изучаются отдельно; они тоже дают альтернативный путь к генерации.

Autoregressive и flows: две полезные точки сравнения

Авторегрессионная факторизация:

$$p_\theta(x) = \prod_{j=1}^{D} p_\theta(x_j \mid x_1, \ldots, x_{j-1}).$$

Normalizing flow: $x = f_\theta(z)$, $z \sim p_Z$; f_θ обратимо:

$$\log p_X(x) = \log p_Z(f_\theta^{-1}(x)) + \log \left| \det J_{f_\theta^{-1}}(x) \right|.$$

GAN: достаточно уметь генерировать $x = G_\theta(z)$; плотность $p_\theta(x)$ может быть недоступна.

PyTorch: генератор как отображение шума

Ключевой переход: вместо decoder, задающего likelihood, обучаем отображение $G_\theta : \mathbb{R}^d \to \mathbb{R}^D$.

$$z \sim \mathcal{N}(0, I), \qquad \tilde{x} = G_\theta(z).$$

Что меняется?

Нам нужен обучающий сигнал для G_θ, который сравнивает *распределения выборок*, а не пары $(x, \hat{x})$.

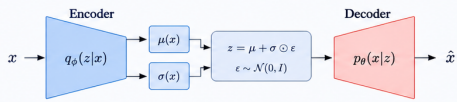
PyTorch: прямой проход генератора

```
G = nn.Sequential(nn.Linear(64, 256), nn.ReLU(),
                  nn.Linear(256, 784), nn.Tanh())
z = torch.randn(32, 64)
fake = G(z) # [32, 784]
```

При использовании `Tanh` реальные изображения следует масштабировать в $[-1, 1]$.

Variational Autoencoder (VAE)

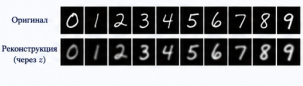
Вероятностная генеративная модель с латентными переменными



Функция потерь (ELBO):

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \text{KL}(q_\phi(z|x) \,||\, p(z))$$

Реконструкция (MNIST)

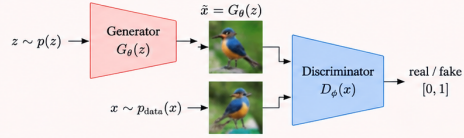


Интерполяция в латентном пространстве



Generative Adversarial Network (GAN)

Состязательная генерация без явного вычисления правдоподобия



Состязательная функция потерь:

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

Сгенерированные примеры (MNIST, DCGAN)



Сгенерированные примеры (CIFAR-10, DCGAN)



Два подхода к генеративному моделированию

VAE: явная вероятностная модель

- Моделирует явное распределение $p_\theta(x)$
- Обучается по вариационной нижней оценке (ELBO)
- Дает аппроксимацию posterior $q_\phi(z|x)$
- Гладкое и структурированное латентное пространство
- Часто более размытые сгенерированные примеры

GAN: неявная модель через состязательное обучение

- Не требует явного вычисления $p_\theta(x)$
- Обучается через состязательную задачу (дискриминатор vs генератор)
- Может генерировать более резкие и реалистичные примеры
- Склонен к mode collapse и нестабильности обучения
- Не предоставляет оценку правдоподобия и явного posterior

GAN: две сети и одна игра

Пусть $G_\theta(z)$ — генератор, $D_\psi(x) \in (0,1)$ — дискриминатор.

$$z \sim p_Z, \quad x \sim p_{\text{data}}, \quad \tilde{x} = G_\theta(z).$$

- D_ψ оценивает, насколько образец похож на реальный.
- G_θ пытается создавать выборки, неотличимые от данных.
- Оптимизация имеет вид *minimax* игры двух моделей.

https://arxiv.org/pdf/1406.2661

Minimax objective

$$\boxed{\min_{\theta} \max_{\psi} V(\theta, \psi)}$$

$$V(\theta, \psi) = \mathbb{E}_{x \sim p_{\text{data}}}[\log D_\psi(x)] + \mathbb{E}_{z \sim p_Z}[\log(1 - D_\psi(G_\theta(z)))].$$

При фиксированном G дискриминатор решает задачу бинарной классификации. При фиксированном D генератор уменьшает способность дискриминатора отличать подделки.

Оптимальный дискриминатор: поточечная задача

При фиксированном распределении генератора p_g:

$$V(D, G) = \int \left[p_{\text{data}}(x) \log D(x) + p_g(x) \log(1 - D(x)) \right] dx.$$

Для $a, b \geq 0$ максимизируем $a \log d + b \log(1 - d)$:

$$\frac{a}{d} - \frac{b}{1 - d} = 0 \quad \Longrightarrow \quad d^* = \frac{a}{a + b}.$$

Следовательно, на точках с $p_{\text{data}}(x) + p_g(x) > 0$:

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

От оптимального D к JS-дивергенции

Положим $m = \frac{1}{2}(p_{\text{data}} + p_g)$. Подставим D_G^*:

$$\begin{aligned} V(D_G^*, G) &= \int p_{\text{data}} \log \frac{p_{\text{data}}}{2m} + p_g \log \frac{p_g}{2m}\, dx \\ &= -\log 4 + D_{\text{KL}}(p_{\text{data}} \| m) + D_{\text{KL}}(p_g \| m) \\ &= \boxed{-\log 4 + 2\,\text{JS}(p_{\text{data}} \| p_g)}. \end{aligned}$$

Глобальный минимум достигается при $p_g = p_{\text{data}}$ (в идеализированной постановке с оптимальным D).

Что означает равновесие GAN?

Если $p_g = p_{\text{data}}$, то

$$D_G^*(x) = \frac{1}{2}, \qquad V(D_G^*, G) = -\log 4.$$

- Оптимальный дискриминатор не способен различать выборки.
- В реальном обучении модели имеют конечную мощность, а оптимизация не обязана достигать равновесия.
- Значение $D(x) \approx \frac{1}{2}$ само по себе не доказывает качество генерации: дискриминатор может быть просто плохо обучен.

PyTorch: discriminator и BCEWithLogits

```
D = nn.Sequential(nn.Linear(784, 256), nn.LeakyReLU(0.2),
                  nn.Linear(256, 1))
real_logits = D(real)
fake_logits = D(fake.detach())
loss_D = F.binary_cross_entropy_with_logits(
  real_logits, torch.ones_like(real_logits)) +
  F.binary_cross_entropy_with_logits(
  fake_logits, torch.zeros_like(fake_logits))
```

D возвращает *logits*; sigmoid встроен в устойчивую численно BCE-функцию.

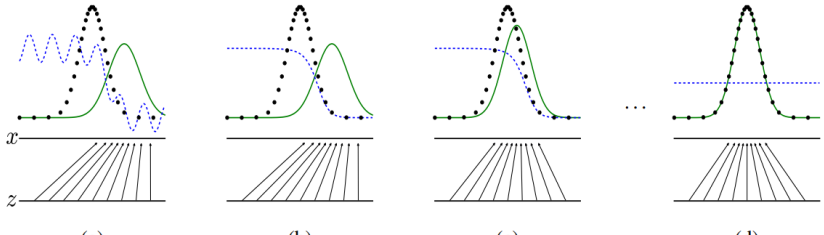


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution 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. $D(\boldsymbol{x}) = \frac{1}{2}$.

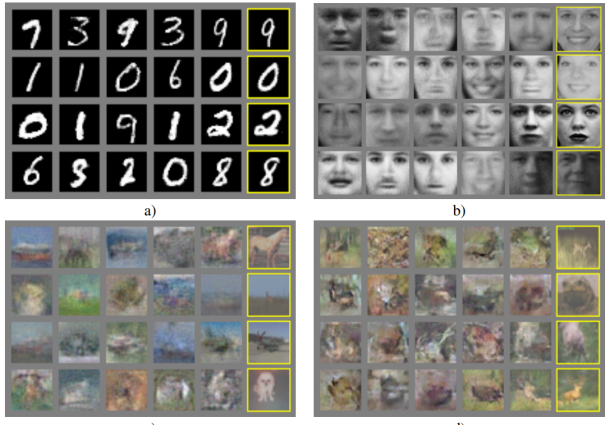
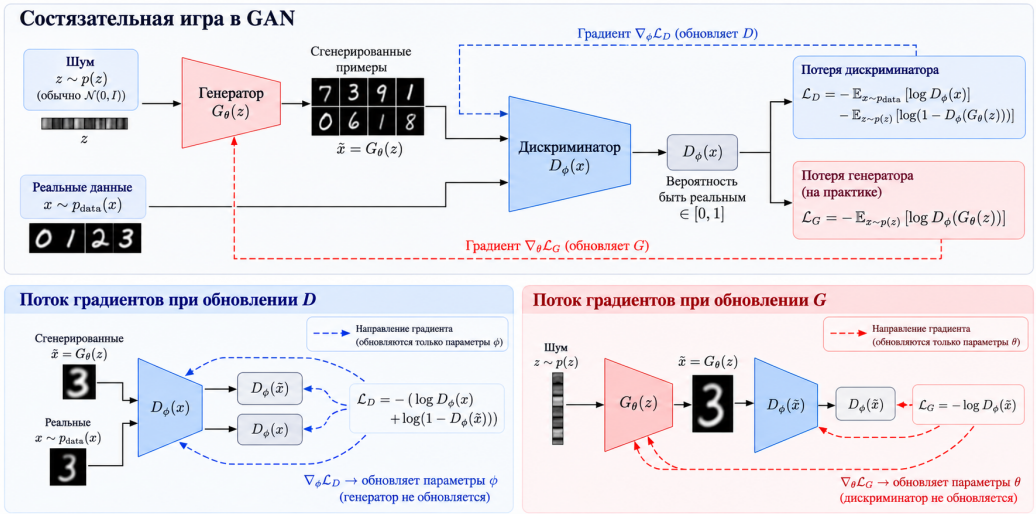


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative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



Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.


Состязательная игра в GAN
Шум
$z \sim p(z)$
(обычно $\mathcal{N}(0, I)$)
z
Генератор
$G_\theta(z)$
Сгенерированные примеры
$\tilde{x} = G_\theta(z)$
Реальные данные
$x \sim p_{\text{data}}(x)$
Дискриминатор
$D_\phi(x)$
$D_\phi(x)$
Вероятность быть реальным
$\in [0, 1]$
Градиент $\nabla_\phi \mathcal{L}_D$ (обновляет D)
Потеря дискриминатора
$\mathcal{L}_D = -\mathbb{E}_{x \sim p_{\text{data}}} [\log D_\phi(x)]$
$-\mathbb{E}_{z \sim p(z)} [\log(1 - D_\phi(G_\theta(z)))]$
Потеря генератора
(на практике)
$\mathcal{L}_G = -\mathbb{E}_{z \sim p(z)} [\log D_\phi(G_\theta(z))]$
Градиент $\nabla_\theta \mathcal{L}_G$ (обновляет G)
Поток градиентов при обновлении D
Направление градиента
(обновляются только параметры ϕ)
Сгенерированные
$\tilde{x} = G_\theta(z)$
Реальные
$x \sim p_{\text{data}}(x)$
$D_\phi(x)$
$D_\phi(\tilde{x})$
$D_\phi(x)$
$\mathcal{L}_D = -(\log D_\phi(x) + \log(1 - D_\phi(\tilde{x})))$
$\nabla_\phi \mathcal{L}_D \rightarrow$ обновляет параметры ϕ
(генератор не обновляется)
Поток градиентов при обновлении G
Направление градиента
(обновляются только параметры θ)
Шум
$z \sim p(z)$
$G_\theta(z)$
$\tilde{x} = G_\theta(z)$
$D_\phi(\tilde{x})$
$D_\phi(\tilde{x})$
$\mathcal{L}_G = -\log D_\phi(\tilde{x})$
$\nabla_\theta \mathcal{L}_G \rightarrow$ обновляет параметры θ
(дискриминатор не обновляется)

Saturating и non-saturating generator loss

Оригинальный minimax loss:

$$\mathcal{L}_G^{\text{sat}} = \mathbb{E}_z \log(1 - D(G(z))).$$

При уверенном дискриминаторе градиенты могут быть слабыми. **Практический non-saturating loss:**

$$\boxed{\mathcal{L}_G^{\text{ns}} = -\mathbb{E}_z \log D(G(z)).}$$

Обе цели поощряют обман дискриминатора, но динамика градиентов различается.

Важно

Связь с JS-дивергенцией выведена для исходной minimax-цели при оптимальном D; её нельзя без оговорок переносить на non-saturating loss.

Задача: Saturating vs Non-saturating Loss

Пусть дискриминатор имеет вид

$$D(G_\theta(z)) = \sigma(a), \qquad \sigma(a) = \frac{1}{1 + e^{-a}},$$

где a — логит дискриминатора.

Рассмотрим две функции потерь генератора:

$$\mathcal{L}_G^{\text{sat}} = \log(1 - \sigma(a)), \qquad \mathcal{L}_G^{\text{ns}} = -\log \sigma(a).$$

Задания

1. Вычислите производные
$$\frac{\partial \mathcal{L}_G^{\text{sat}}}{\partial a}, \qquad \frac{\partial \mathcal{L}_G^{\text{ns}}}{\partial a}.$$
2. Найдите их значения при $D(G_\theta(z)) = 0.01$.
3. Какую функцию потерь лучше использовать, если дискриминатор уверенно распознаёт поддельные изображения? Почему?

PyTorch: шаг генератора

```
fake = G(torch.randn(batch_size, 64, device=device))
fake_logits = D(fake)
loss_G = F.binary_cross_entropy_with_logits(
  fake_logits, torch.ones_like(fake_logits))
opt_G.zero_grad()
loss_G.backward()
opt_G.step()
```

При шаге генератора граф через D должен сохраняться до G, но параметры D на этом шаге не обновляются.

Чередующаяся оптимизация

1. Выбрать батч $x_i \sim p_{\text{data}}$ и шум $z_i \sim p_Z$.
2. Сделать один или несколько шагов максимизации V по ψ.
3. Сэмплировать новый шум и сделать шаг минимизации $\mathcal{L}_G$ по θ.
4. Повторять, отслеживая не только loss, но и качество/разнообразие сэмплов.

$$\psi \leftarrow \psi + \eta_D \widehat{\nabla}_\psi V, \qquad \theta \leftarrow \theta - \eta_G \widehat{\nabla}_\theta \mathcal{L}_G.$$

Одновременное обучение не эквивалентно точному решению внутренней задачи $\max_\psi$.

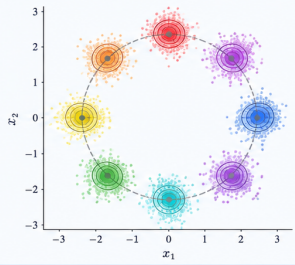
Почему GAN трудно обучать?

- **Mode collapse:** генератор покрывает лишь часть мод распределения.
- **Vanishing gradients:** дискриминатор слишком легко отличает выборки.
- **Осцилляции:** игра может не сходиться при наивном градиентном спуске.
- **Чувствительность:** архитектура, нормализация, learning rate, баланс шагов.

Пример: истинное распределение — смесь восьми гауссиан; генератор выдаёт выборки лишь из двух компонент.

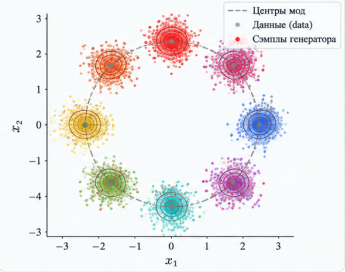
Истинное распределение данных

Смесь 8 гауссиан



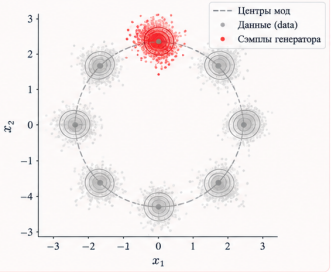
Генератор без mode collapse

Покрывает все моды распределения

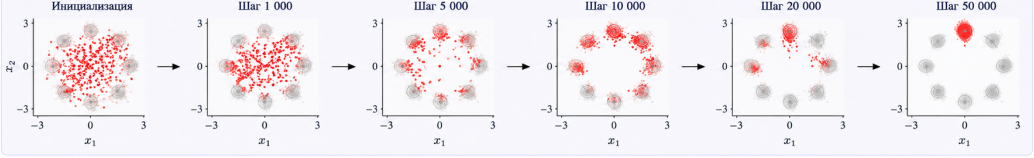


Mode collapse

Генератор покрывает только одну моду



Процесс обучения GAN: пример mode collapse



Почему JS может давать плохой сигнал?

Для распределений с непересекающимися носителями

$$\mathrm{JS}(p_{\mathrm{data}} \| p_g) = \log 2.$$

При перемещении непересекающихся носителей JS может оставаться постоянной.

Следствие

Оптимальный дискриминатор насыщается; градиент по параметрам генератора может оказаться неинформативным. В конечных сетях поведение сложнее, но проблема мотивирует другую метрику.

Wasserstein-1: геометрия расстояния

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|.$$

Канторович–Рубинштейн:

$$\boxed{W_1(P, Q) = \sup_{\|f\|_{\mathrm{Lip}} \le 1} \big(\mathbb{E}_{x\sim P} f(x) - \mathbb{E}_{y\sim Q} f(y)\big).}$$

- f — вещественнозначный *critic*, а не вероятность класса.
- Ограничение Липшица существенно для корректности двойственной формулы.

WGAN: целевая функция

$$\max_{\psi:\, \|f_\psi\|_{\mathrm{Lip}} \le 1} \mathbb{E}_{x \sim p_{\mathrm{data}}} f_\psi(x) - \mathbb{E}_{z \sim p_Z} f_\psi(G_\theta(z)).$$

Генератор минимизирует

$$\boxed{\mathcal{L}_G = -\mathbb{E}_z f_\psi(G_\theta(z)).}$$

- У critic нет sigmoid и BCE.
- В исходном WGAN ограничение Липшица приближали *weight clipping*.
- Clipping часто ухудшает выразительность critic и оптимизацию.

WGAN-GP: gradient penalty

При $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$, $\epsilon \sim U[0,1]$:

$$\boxed{\mathcal{L}_D = \mathbb{E}_{\tilde{x}} f_\psi(\tilde{x}) - \mathbb{E}_x f_\psi(x) + \lambda \mathbb{E}_{\hat{x}}(\|\nabla_{\hat{x}} f_\psi(\hat{x})\|_2 - 1)^2.}$$

- Penalty поощряет единичную норму градиента на интерполяциях.
- Это практическая регуляризация, не точное глобальное обеспечение 1-Lipschitz.
- Часто делают несколько шагов critic на один шаг генератора.

PyTorch: gradient penalty

```
eps = torch.rand(B, 1, device=real.device)
inter = (eps * real + (1 - eps) * fake.detach())
inter.requires_grad_(True)
scores = critic(inter)
grad = torch.autograd.grad(
  scores.sum(), inter, create_graph=True)[0]
gp = (grad.flatten(1).norm(2, dim=1) - 1).square().mean()
loss_D = critic(fake.detach()).mean()
  - critic(real).mean() + 10.0 * gp
```

Предполагается *real*, *fake* $\in \mathbb{R}^{B \times D}$; для изображений ϵ имеет форму $(B, 1, 1, 1)$.

Другие практики стабилизации

- **Spectral normalization:** $W \leftarrow W/\sigma_{\max}(W)$ для контроля нормы линейных слоёв.
- **Two-time-scale update rule (TTUR):** разные скорости обучения G и D.
- **Data augmentation:** повышает устойчивость при малых датасетах.
- **Архитектурные ограничения:** сверточные слои, нормализация, skip connections.

Осторожно: не существует универсального набора гиперпараметров, гарантирующего сходимость.

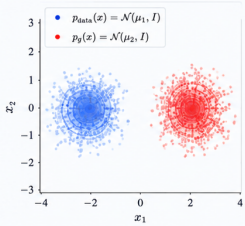
PyTorch: spectral normalization

```
from torch.nn.utils.parametrizations import spectral_norm
critic = nn.Sequential(
  spectral_norm(nn.Linear(784, 256)),
  nn.LeakyReLU(0.2),
  spectral_norm(nn.Linear(256, 1))
)
```

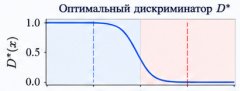
Это альтернатива регуляризации discriminator; spectral normalization не тождественна WGAN-GP.

Jensen–Shannon divergence (GAN)

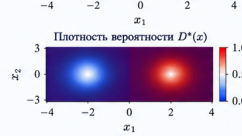
Два разнесённых гауссовских распределения



Оптимальный дискриминатор D^*



Плотность вероятности $D^*(x)$

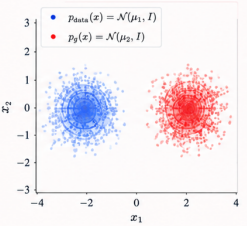


При отсутствии пересечения распределений Jensen-Shannon divergence достигает максимального значения:

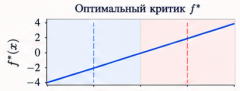
$$JS(p_{\text{data}} \| p_g) = \log 2$$

Wasserstein distance (WGAN)

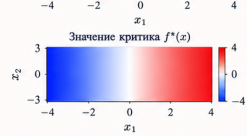
Два разнесённых гауссовских распределения



Оптимальный критик f^*



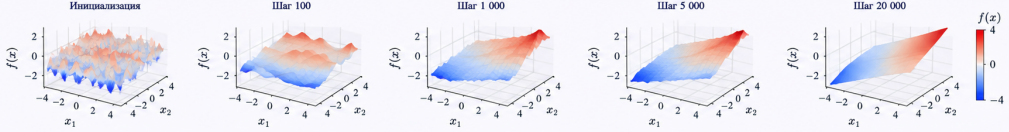
Значение критика $f^*(x)$



Wasserstein distance растёт линейно с расстоянием между распределениями:

$$W(p_{\text{data}}, p_g) \approx \|\mu_1 - \mu_2\|_2$$

Динамика обучения критика (WGAN)



DCGAN: сверточная индуктивная структура

- Генератор: $z \to$ пространственная карта признаков $\to$ upsampling/transposed convolutions.
- Дискриминатор: свёртки со stride для уменьшения пространственного разрешения.
- Нелинейности: ReLU в G, LeakyReLU в D; на выходе G часто Tanh.
- BatchNorm применяют с учётом конкретной архитектуры и функции потерь.

Цель: использовать локальную структуру изображений вместо полносвязной модели.

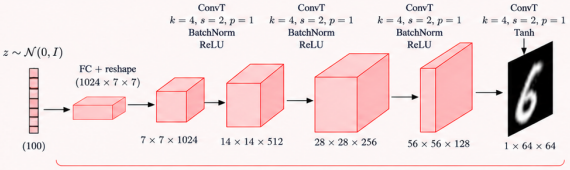
PyTorch: фрагмент DCGAN generator

```
G = nn.Sequential(
  nn.ConvTranspose2d(128, 256, 4, 1, 0, bias=False),
  nn.BatchNorm2d(256), nn.ReLU(True),
  nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
  nn.BatchNorm2d(128), nn.ReLU(True),
  nn.ConvTranspose2d(128, 3, 4, 2, 1, bias=False),
  nn.Tanh())
images = G(torch.randn(16, 128, 1, 1))
```

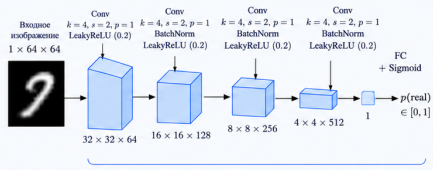
Выход: $(16, 3, 16, 16)$; для 32×32 или 64×64 добавьте upsampling-блоки.

Архитектура DCGAN

Генератор G (ConvTranspose)



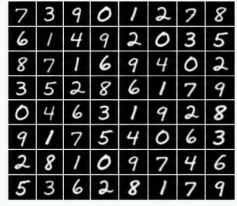
Дискриминатор D (Conv)



Динамика генерации при обучении DCGAN (MNIST)



Пример сгенерированных изображений (64×64)



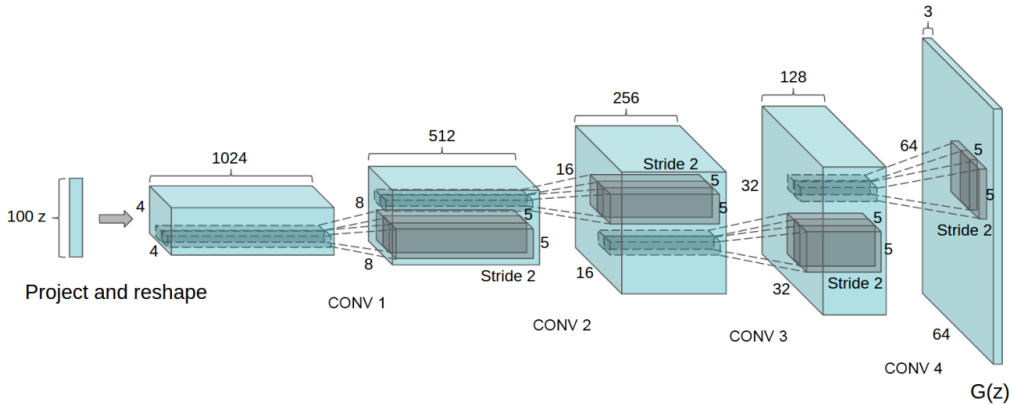
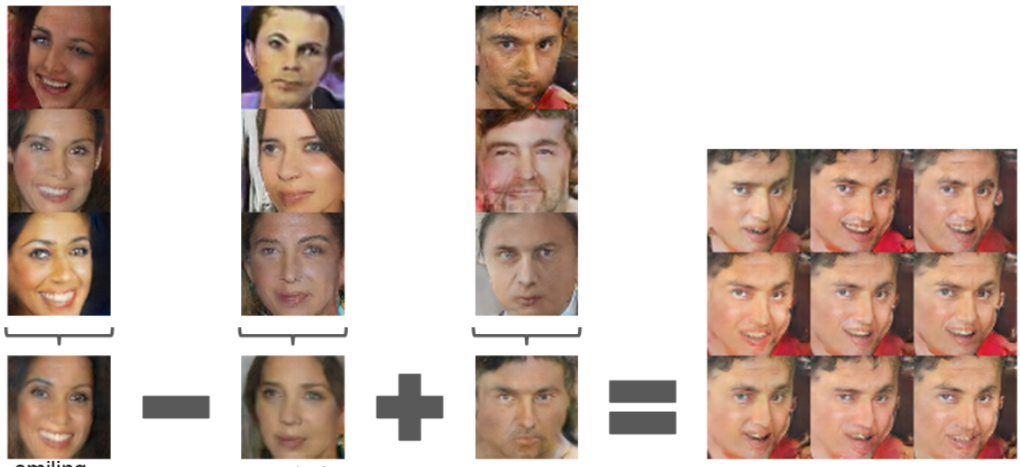
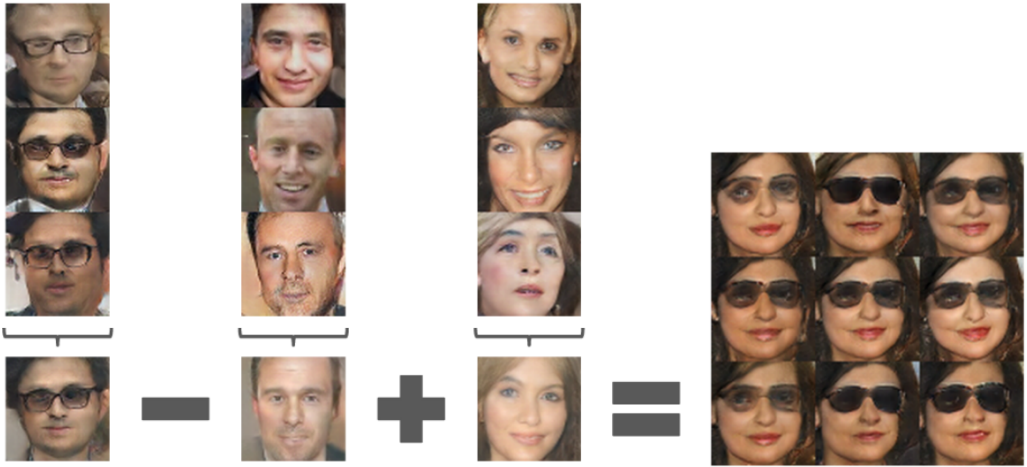


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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


smiling woman
neutral woman
neutral man
smiling man


man
with glasses
man
without glasses
woman
without glasses
woman with glasses

Conditional GAN: моделируем $p(x \mid y)$

Добавляем условие y (метка класса, текстовый embedding, изображение):

$$\tilde{x} = G_\theta(z, y), \qquad D_\psi(x, y) \in (0, 1).$$

$$\min_G \max_D \; \mathbb{E}_{(x,y)\sim p_{\text{data}}} \log D(x, y) + \mathbb{E}_{y\sim p(y),\, z\sim p_Z} \log(1 - D(G(z, y), y)).$$

Пример: $y = 7$ — генерация изображений цифры 7; можно менять z при фиксированном y.

PyTorch: условие как embedding

```
emb = nn.Embedding(num_classes, 16)
y = torch.randint(num_classes, (32,))
z = torch.randn(32, 64)
G_input = torch.cat([z, emb(y)], dim=1)
fake = generator(G_input)
```

Дискриминатор тоже должен получать y; например, через конкатенацию с признаками или projection discriminator.

Image-to-image translation: Pix2Pix

Для *парных* данных (x, y): вход x и целевое изображение y.

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

Добавляют попиксельную реконструкцию:

$$\mathcal{L}_{L_1}(G) = \mathbb{E}_{x,y,z} \|y - G(x, z)\|_1.$$

$$\boxed{\min_G \max_D \; \mathcal{L}_{\mathrm{cGAN}} + \lambda \mathcal{L}_{L_1}.}$$

Типичная реализация: U-Net generator и PatchGAN discriminator.

PyTorch: Pix2Pix objective

```
pred_fake = D(torch.cat([source, fake], dim=1))
adv = F.binary_cross_entropy_with_logits(
  pred_fake, torch.ones_like(pred_fake))
rec = F.l1_loss(fake, target)
loss_G = adv + 100.0 * rec
```

$fake = G(source)$; PatchGAN возвращает карту логитов, а не одно число. Показан шаг G, без шага D.

Pix2Pix: условная генерация по парам изображений

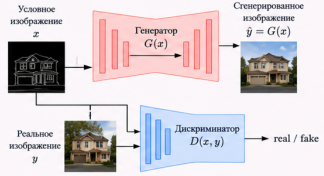
Пример применения условного GAN для задач трансляции изображений

Постановка задачи

Даны пары изображений $(x, y) \sim p_{\text{data}}(x, y)$, где x — условное изображение (например, edge map или семантическая сегментация), y — целевое изображение (например, реалистичное фото).

Мы хотим обучить генератор $G(x)$, который по x синтезирует $\hat{y} = G(x)$ такое, чтобы $\hat{y}$ было неотличимо от реального y.

Архитектура Pix2Pix (cGAN)



Функция потерь

1. Состязательная потеря (условный GAN)

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

2. Потеря реконструкции (L1)

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y}\left[\|y - G(x)\|_1\right]$$

Полная функция потерь

$$\mathcal{L}(G, D) = \mathcal{L}_{GAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

Пример 1: Edge map → Photo (фасады зданий)



Пример 2: Segmentation → Street scene (карты семантической сегментации)



CycleGAN: задача без парных изображений

Даны два домена X и Y, но *нет* соответствующих пар (x, y).

$$G : X \to Y, \qquad F : Y \to X,$$

$$D_Y \text{ отличает } y \text{ от } G(x), \qquad D_X \text{ отличает } x \text{ от } F(y).$$

Одних adversarial loss недостаточно: модель может менять содержание изображения при сохранении реалистичного стиля.

Идея

Добавить требование циклического восстановления: $F(G(x)) \approx x$ и $G(F(y)) \approx y$.

https://arxiv.org/pdf/1703.10593

Cycle consistency loss

$$\boxed{\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_X} \|F(G(x)) - x\|_1 + \mathbb{E}_{y \sim p_Y} \|G(F(y)) - y\|_1.}$$

- Не требует парных наблюдений.
- Ограничивает произвольное изменение структуры при переводе домена.
- Не гарантирует семантическую корректность и может сохранять нежелательную информацию.

Полная цель CycleGAN

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ + \lambda_{\text{cyc}}\mathcal{L}_{\text{cyc}}(G, F).\end{aligned}$$

Иногда добавляют *identity loss*:

$$\mathcal{L}_{\text{id}} = \mathbb{E}_{y\sim p_Y}\|G(y) - y\|_1 + \mathbb{E}_{x\sim p_X}\|F(x) - x\|_1,$$

когда архитектуры G, F допускают входы обоих доменов. На практике часто применяют least-squares adversarial loss (LSGAN) вместо логарифмического.

PyTorch: цикл и identity loss

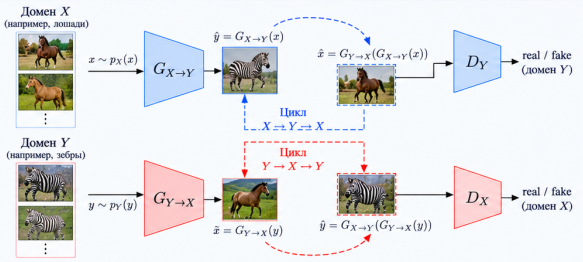
```
fake_y = G(x)
fake_x = F(y)
cycle_x = F(fake_y)
cycle_y = G(fake_x)
loss_cyc = F_nn.l1_loss(cycle_x, x)
  + F_nn.l1_loss(cycle_y, y)
loss_G = adv_G + adv_F + 10.0 * loss_cyc
```

Здесь `F_nn = torch.nn.functional`; переменная `F` обозначает обратный генератор. Adversarial terms считаются отдельно.

CycleGAN: трансляция изображений без парных примеров

Два генератора и циклическая согласованность позволяют обучаться на непарных данных

Архитектура и идея CycleGAN



Функции потерь CycleGAN

1. Состязательные потери (для обоих направлений)

$$\mathcal{L}_{GAN}(G_{X \to Y}, D_Y) = \mathbb{E}_{y \sim p_Y(y)}\left[\log D_Y(y)\right] + \mathbb{E}_{x \sim p_X(x)}\left[\log(1 - D_Y(G_{X \to Y}(x)))\right]$$

$$\mathcal{L}_{GAN}(G_{Y \to X}, D_X) = \mathbb{E}_{x \sim p_X(x)}\left[\log D_X(x)\right] + \mathbb{E}_{y \sim p_Y(y)}\left[\log(1 - D_X(G_{Y \to X}(y)))\right]$$

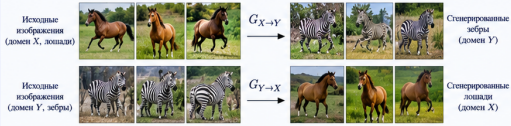
2. Циклическая согласованность

$$\mathcal{L}_{cyc}(G_{X \to Y}, G_{Y \to X}) = \mathbb{E}_{x \sim p_X(x)}\left[\|G_{Y \to X}(G_{X \to Y}(x)) - x\|_1\right] + \mathbb{E}_{y \sim p_Y(y)}\left[\|G_{X \to Y}(G_{Y \to X}(y)) - y\|_1\right]$$

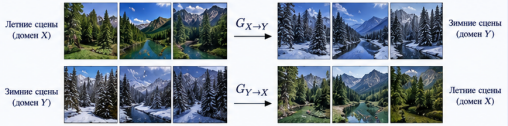
Полная функция потерь

$$\mathcal{L}(G_{X \to Y}, G_{Y \to X}, D_X, D_Y) = \mathcal{L}_{GAN}(G_{X \to Y}, D_Y) + \mathcal{L}_{GAN}(G_{Y \to X}, D_X) + \lambda \mathcal{L}_{cyc}(G_{X \to Y}, G_{Y \to X})$$

Пример 1: Horse ↔ Zebra



Пример 2: Summer ↔ Winter (сезонная трансляция)



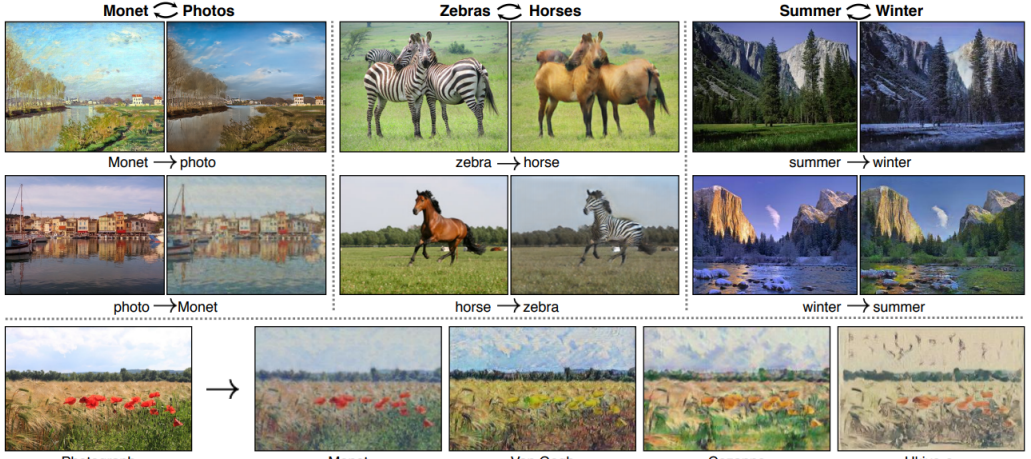


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos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

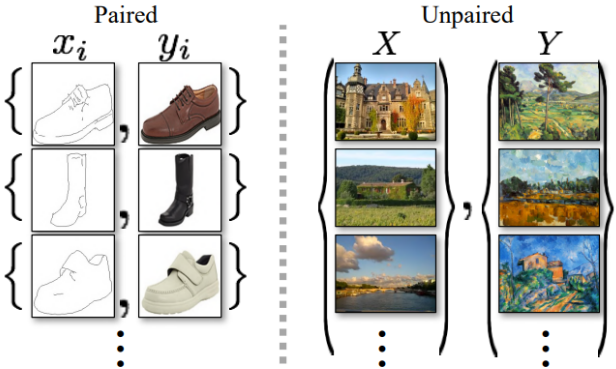


Figure 2: *Paired* training data (left) consists of training examples $\{x_i, y_i\}_{i=1}^N$, where the correspondence between x_i and y_i exists [22]. We instead consider *unpaired* training data (right), consisting of a source set $\{x_i\}_{i=1}^N$ ($x_i \in X$) and a target set $\{y_j\}_{j=1}^M$ ($y_j \in Y$), with no information provided as to which x_i matches which y_j.

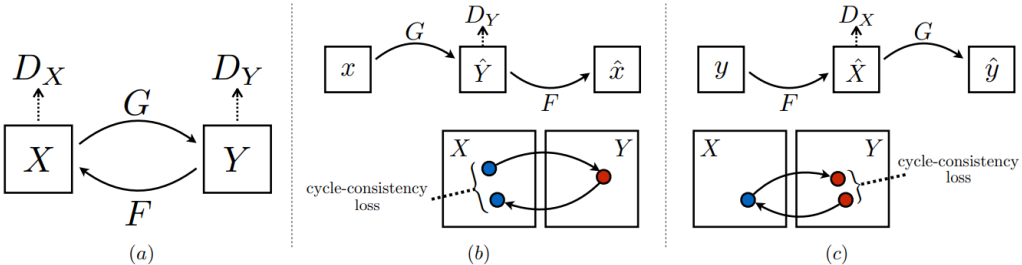


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, 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: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

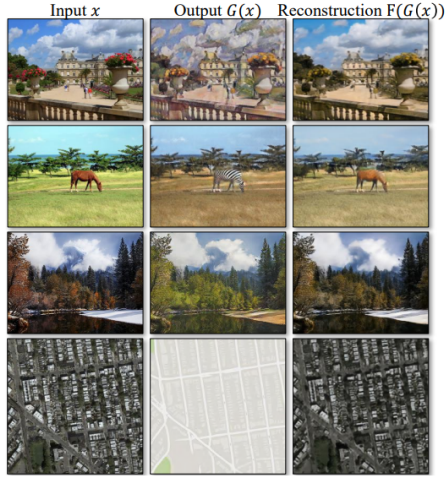


Figure 4: The input images x, output images $G(x)$ and the reconstructed images $F(G(x))$ from various experiments. From top to bottom: photo↔Cezanne, horses↔zebras, winter→summer Yosemite, aerial photos↔Google maps.

StyleGAN: управление стилем и разрешением

- Mapping network: $w = M(z)$ вместо прямой подачи z во все слои.
- Style modulation позволяет влиять на признаки на разных масштабах.
- Добавление шума контролирует стохастические локальные детали.
- Style mixing помогает исследовать, какие уровни отвечают за разные свойства изображения.

Идея: более структурированное управление генерацией, чем в простом DCGAN.

https://arxiv.org/pdf/1812.04948

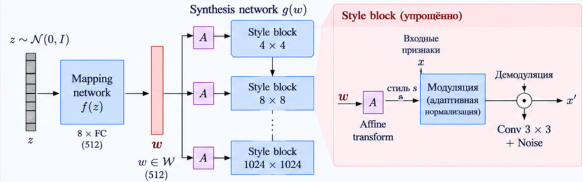
PyTorch: идея style mapping

```
mapping = nn.Sequential(
  nn.Linear(128, 128), nn.LeakyReLU(0.2),
  nn.Linear(128, 128))
z = torch.randn(16, 128)
w = mapping(z)
style = affine(w) # affine maps w to layer controls
```

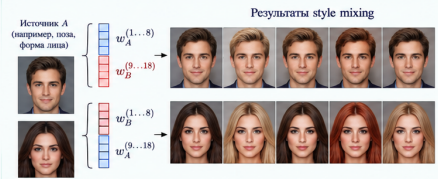
Это иллюстрация *mapping network*, а не полная реализация StyleGAN с modulated convolutions.

StyleGAN: управление генерацией через стили на разных уровнях

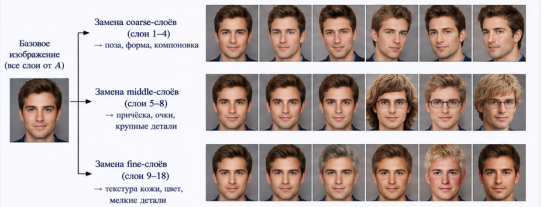
Архитектура StyleGAN (Style-based generator)



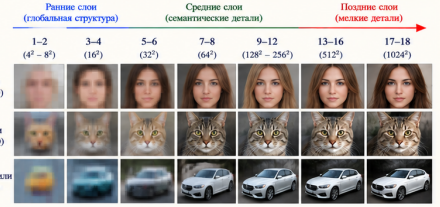
Style mixing: комбинация высокоуровневых и низкоуровневых стилей

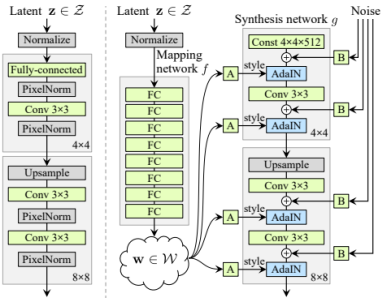


Влияние разных слоёв на генерацию



Интерпретация слоёв: от глобальной структуры к мелким деталям





(a) Traditional (b) Style-based generator

Figure 1. While a traditional generator [30] feeds the latent code though the input layer only, we first map the input to an intermediate latent space $\mathcal{W}$, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the nonlinearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The mapping network f consists of 8 layers and the synthesis network g consists of 18 layers — two for each resolution ($4^2 - 1024^2$). The output of the last layer is converted to RGB using a separate 1×1 convolution, similar to Karras et al. [30]. Our generator has a total of 26.2M trainable parameters, compared to 23.1M in the traditional generator.

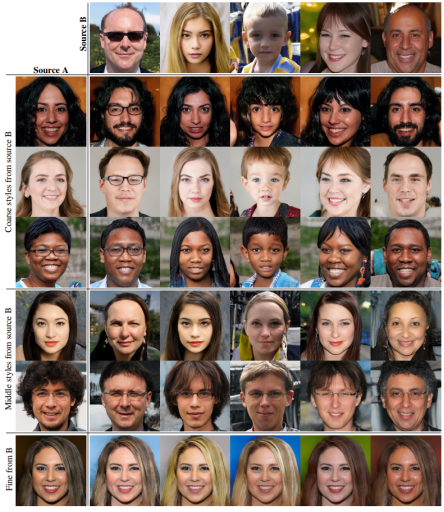


Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions ($16^2 - 32^2$) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles ($64^2 - 1024^2$) from B brings mainly the color scheme and microstructure.

Как сравнивать GAN?

- **FID:** расстояние между гауссовскими аппроксимациями распределений признаков real/fake.
- **Precision/Recall:** отдельно качество и покрытие разнообразия.
- **Inception Score:** уверенность и разнообразие предсказанных классов; не использует реальные данные напрямую.
- **Визуальная диагностика:** сэмплы, mode collapse, артефакты, запоминание.

$$\mathrm{FID} = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\big(\Sigma_r + \Sigma_g - 2(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2})^{1/2}\big).$$

Метрики зависят от feature extractor, препроцессинга и размера выборки.

PyTorch: сбор сэмплов для оценки

```
G.eval()
with torch.no_grad():
  z = torch.randn(256, latent_dim, device=device)
  generated = G(z).cpu()
# Convert [-1, 1] to [0, 1] if needed:
generated = (generated + 1) / 2
```

Для корректного FID требуются одинаковый препроцессинг и достаточно большие наборы реальных и синтетических изображений.

Сравнение: VAE, GAN, Flows

	VAE	**GAN**	**Flows**
Обучение	ELBO	Adversarial	Exact likelihood
Плотность	Нижняя оценка	Обычно недоступна	Вычислима
Сэмплирование	Обычно быстрое	Обычно быстрое	Через прямой flow
Проблемы	Posterior collapse	Mode collapse	Обратимость, Jacobian

Следующая лекция: Normalizing Flows — можно ли совместить выразительный генератор с точным вычислением плотности?

Четыре формулы, которые стоит запомнить

$$\min_G \max_D \quad \mathbb{E}_x \log D(x) + \mathbb{E}_z \log(1 - D(G(z))),$$

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

$$V(D_G^*, G) = -\log 4 + 2\,\mathrm{JS}(p_{\text{data}} \| p_g),$$

$$\mathcal{L}_{\text{cyc}} = \mathbb{E}_x \| F(G(x)) - x \|_1 + \mathbb{E}_y \| G(F(y)) - y \|_1.$$

Вопрос аудитории: почему CycleGAN нужны два генератора, а Pix2Pix достаточно одного?