ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 13

Генеративно-состязательные нейронные сети (GAN)

Генератор и дискриминатор, минимаксная игра, DCGAN, Conditional GAN и Wasserstein GAN

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан

Учебный год: 2026

> **Место в курсе.** Лекция 11 была посвящена автоэнкодерам и VAE; лекция 12 — Transformer. Теперь рассматривается принципиально иной способ построения генеративных моделей: обучение посредством конкуренции двух сетей. Основополагающая статья — Goodfellow и соавторы (2014) [8]; систематическую основу дают учебники [6, 3, 4, 5].

1. Цель, результаты обучения и карта лекции

Цель лекции — понять, как сформулировать генерацию данных как статистическую задачу и как свести её к состязательной оптимизации двух нейросетей; вывести оптимальный дискриминатор и теоретический критерий равновесия; построить практическую GAN с контролем качества и воспроизводимыми экспериментами.

После изучения студент способен:

1. различать дискриминативные и генеративные модели и описывать распределение данных;
2. объяснять роли генератора G_θ и дискриминатора D_ϕ;
3. выводить и интерпретировать функцию ценности исходной GAN;
4. находить оптимальный дискриминатор при фиксированном генераторе;
5. устанавливать связь оптимальной игры с дивергенцией Йенсена–Шеннона;
6. сравнивать minimax и non-saturating функции потерь;

7. объяснять mode collapse, неустойчивость обучения и насыщение градиентов;
8. проектировать DCGAN и условную GAN, рассчитывать формы тензоров;
9. описывать WGAN, условие липшицевости и gradient penalty;
10. реализовывать обучение на PyTorch и диагностировать результаты.

> **Обозначения** P_{data} — истинное распределение обучающих данных; P_z — заранее заданное распределение скрытого шума; P_g — распределение образцов $G_\theta(z)$; θ — параметры генератора; ϕ — параметры дискриминатора. Все логарифмы в формулах — натуральные.

2. Генеративное моделирование как задача машинного обучения

2.1 От классификации к синтезу объектов

Классификатор оценивает условную вероятность $P(y \mid x)$ или функцию решения $f(x)$: на вход подаётся изображение, выходом является метка. Генеративная модель решает другую задачу: воспроизводить закономерности случайной величины $X \sim P_{\text{data}}$ и порождать новые реализации, статистически похожие на наблюдаемые. Например, на множестве фотографий рукописных цифр мы хотим получить *новую* цифру, а не узнать метку уже имеющейся.

Пусть имеется независимая выборка $x_1, \ldots, x_N \in \mathbb{R}^d$. Конечная выборка не является самим распределением; это ограниченное количество наблюдений. Генеративная модель должна обобщать, а не запоминать массив $\{x_i\}$ [6, 7].

2.2 Явные и неявные модели распределения

Явная модель задаёт выражение для плотности $p_\theta(x)$ либо её аппроксимации. Так устроены многие вероятностные модели; VAE оптимизирует нижнюю вариационную оценку логарифма правдоподобия. **Неявная генеративная модель** задаёт алгоритм семплирования, не обязательно позволяющий вычислить $p_\theta(x)$ точечно. GAN использует именно такой подход.

При $Z \sim P_z$ и $X_g = G_\theta(Z)$ индуцированное распределение равно $P_g = (G_\theta)_{\#}P_z$: для измеримого множества A выполняется

$$P_g(A) = P_z\big(\{z : G_\theta(z) \in A\}\big). \tag{1}$$

Оператор «#» обозначает образ меры при отображении. Это определение не требует существования плотности p_g; однако классический теоретический вывод далее предполагает, что обе сравниваемые меры имеют плотности относительно общей доминирующей меры.

2.3 Сравнение с автоэнкодером и VAE

Обычный автоэнкодер включает $x \to z \to \widehat{x}$ и минимизирует ошибку восстановления. VAE задаёт вероятностный encoder $q_\varphi(z \mid x)$ и decoder $p_\theta(x \mid z)$ и максимизирует ELBO. GAN, напротив, обучает генератор через обратную связь другой сети: дискриминатора. Для базовой GAN обучаемый encoder не нужен. Поэтому нельзя автоматически интерпретировать GAN как метод реконструкции конкретного входного объекта [9, 8].

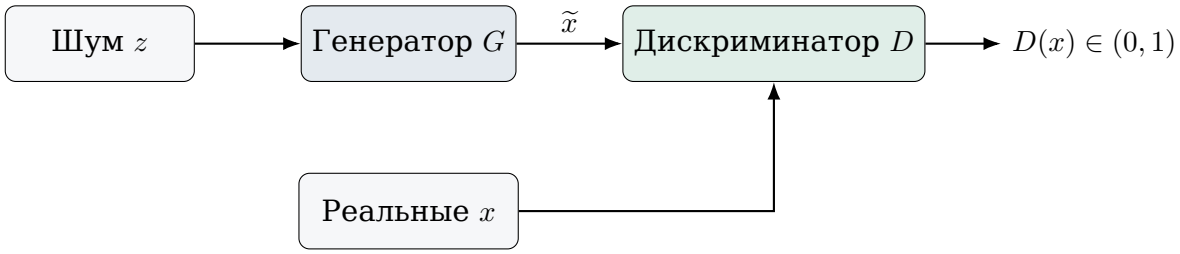
Свойство	AE	VAE	GAN
Главная задача	Реконструкция	Вероятностная генерация	Генерация через состязание
Кодировщик	Обычно да	Да	Не требуется в базовой версии
Главный критерий	Ошибка восстановления	ELBO	Функция ценности двух сетей
Семплировани	Не всегда естественно	Естественно из prior	Естественно из шума

3. Интуиция состязательного обучения

3.1 Два участника и общая цель

В классической GAN работают две нейронные сети. **Генератор** $G_\theta : \mathbb{R}^m \to \mathbb{R}^d$ преобразует случайный вектор z в синтетический объект $\widetilde{x} = G_\theta(z)$. **Дискриминатор** $D_\phi : \mathbb{R}^d \to (0,1)$ возвращает число, интерпретируемое как оценка вероятности класса «реальный», если модель обучена на сбалансированных примерах с соответствующей бинарной целевой функцией [8].

Дискриминатор стремится отличать реальные изображения от синтетических, генератор — вырабатывать объекты, которые дискриминатор считает реальными. Важная оговорка: «обмануть дискриминатор» не гарантирует разнообразие и визуальное качество; несовершенный дискриминатор тоже может давать высокую оценку плохим образцам.



3.2 Почему недостаточно одной сети

Без обученного критерия качества генератор не знает, в каком направлении изменять свои параметры. Бинарный дискриминатор обучается на положительных образцах из P_{data} и отрицательных из P_g; его производная относительно синтетического входа позволяет передать сигнал генератору. Так возникает *дифференцируемая обучающая обратная связь*. Две сети можно обучать по очереди с помощью градиентных методов, но совместная оптимизация не эквивалентна минимизации одной неподвижной функции: цель каждого участника изменяется по мере обучения другого.

4. Строгая постановка исходной GAN

4.1 Минимаксный функционал

Исходная статья предлагает игру [8]

$$\min_{\theta}\max_{\phi} V(\theta,\phi), \qquad V(\theta,\phi)=\mathbb{E}_{x\sim P_{\text{data}}}[\log D_\phi(x)]+\mathbb{E}_{z\sim P_z}[\log(1-D_\phi(G_\theta(z)))]. \tag{2}$$

Для фиксированного G дискриминатор *максимизирует* логарифмический критерий бинарной классификации; для фиксированного D генератор в оригинальной минимаксной постановке *минимизирует* второе слагаемое. Первое слагаемое не зависит от θ.

Важно о знаках. В стандартных программных библиотеках оптимизатор выполняет минимизацию. Поэтому функция потерь дискриминатора задаётся с минусом:

$$\mathcal{L}_D(\phi;\theta) = -\mathbb{E}_x \log D_\phi(x) - \mathbb{E}_z \log(1-D_\phi(G_\theta(z))), \tag{3}$$

$$\mathcal{L}_G^{\text{MM}}(\theta;\phi) = \mathbb{E}_z \log(1-D_\phi(G_\theta(z))). \tag{4}$$

Знак у второго выражения действительно положительный: минимизировать $\log(1-D)$ означает повышать D на сгенерированных объектах. Однако такая функция может давать очень слабые градиенты на ранних этапах, поэтому на практике обычно применяется другая формула.

4.2 Non-saturating loss

Для генератора часто используют *несатурирующий* вариант [8]:

$$\mathcal{L}_G^{\text{NS}}(\theta;\phi) = -\mathbb{E}_{z\sim P_z} \log D_\phi(G_\theta(z)). \tag{5}$$

Это не буквальная минимизация V из (2), хотя при идеализированном равновесии цели согласуются: синтетическое распределение должно совпадать с истинным. В отличие от минимаксной потери, $\mathcal{L}_G^{\text{NS}}$ даёт большой штраф, если дискриминатор уверенно считает сгенерированное изображение поддельным.

4.3 Оценивание ожиданий на мини-пакете

При независимых $x_i \sim P_{\text{data}}$, $z_i \sim P_z$ используют статистические оценки

$$\widehat{\mathcal{L}}_D = -\frac{1}{B}\sum_{i=1}^{B}\big[\log D(x_i) + \log(1 - D(G(z_i)))\big], \tag{6}$$

$$\widehat{\mathcal{L}}_G^{\text{NS}} = -\frac{1}{B}\sum_{i=1}^{B}\log D(G(z_i)). \tag{7}$$

В вычислениях вместо явных логарифмов вероятностей следует использовать `BCEWithLogitsLoss`, то есть передавать в функцию потерь *логиты* дискриминатора без предварительного `sigmoid`: так избегаются численные потери точности.

5. Оптимальный дискриминатор: вывод без пропусков

5.1 Предположения и преобразование ожиданий

Предположим, что P_{data} и P_g имеют плотности $p_d(x), p_g(x)$ относительно одной меры μ. При фиксированном генераторе

$$V(G,D) = \int \left[p_d(x)\log D(x) + p_g(x)\log(1-D(x))\right] d\mu(x). \tag{8}$$

Поскольку в идеализированной модели значения $D(x)$ можно выбирать независимо для разных x, задача максимизации распадается на поточечные задачи

$$\max_{0\le d\le 1} f_x(d), \qquad f_x(d) = a\log d + b\log(1-d), \quad a = p_d(x),\ b = p_g(x). \tag{9}$$

Если $a+b>0$ и $a,b>0$, то

$$f_x'(d) = \frac{a}{d} - \frac{b}{1-d}, \qquad f_x''(d) = -\frac{a}{d^2} - \frac{b}{(1-d)^2} < 0. \tag{10}$$

Условие $f_x'(d)=0$ даёт $a(1-d)=bd$ и, следовательно,

$$\boxed{D^*(x) = \frac{p_d(x)}{p_d(x)+p_g(x)}}. \tag{11}$$

Строгая отрицательность второй производной доказывает максимум в точке $D^*(x)$. Если $a=0<b$, максимум достигается при $d=0$; если $b=0<a$, при $d=1$ (как предельные значения sigmoid-модели). На множестве $a=b=0$ значение D не влияет на интеграл.

5.2 Интерпретация через апостериорную вероятность

Представим сбалансированную смесь: сначала равновероятно выбираем класс $Y\in\{\text{real},\text{fake}\}$, затем выбираем x из соответствующего распределения. Тогда формула Байеса даёт

$$P(Y=\text{real}\mid X=x) = \frac{\frac{1}{2}p_d(x)}{\frac{1}{2}p_d(x)+\frac{1}{2}p_g(x)} = D^*(x). \tag{12}$$

Поэтому оптимальный дискриминатор равен $1/2$ там, где плотности совпадают. **Это не означает, что каждая сеть, возвращающая $1/2$, является успешной GAN:** значение $1/2$ может также означать необученный либо слишком слабый дискриминатор.

5.3 Пример на дискретном пространстве

Пусть пространство состоит из трёх объектов: a, b, c. Возьмём

$$P_{\text{data}} = (0.6, 0.3, 0.1), \qquad P_g = (0.2, 0.3, 0.5). \tag{13}$$

Тогда согласно (11)

$$D^*(a) = \frac{0.6}{0.8} = 0.75, \quad D^*(b) = \frac{0.3}{0.6} = 0.5, \quad D^*(c) = \frac{0.1}{0.6} = \frac{1}{6}. \tag{14}$$

Дискриминатор чаще относит a к реальному классу, не различает источник b и считает c преимущественно синтетическим. Эти значения следуют *из соотношения вероятностей*, а не из геометрических характеристик объектов.

6. Почему появляется дивергенция Йенсена–Шеннона

6.1 Подстановка оптимального дискриминатора

Подставляем (11) в исходную функцию ценности:

$$C(G) := \max_D V(G, D) = \int p_d \log \frac{p_d}{p_d + p_g}\, d\mu + \int p_g \log \frac{p_g}{p_d + p_g}\, d\mu. \tag{15}$$

Введём среднее распределение $m = (p_d + p_g)/2$. Тогда $p_d + p_g = 2m$, и

$$C(G) = \int p_d \log \frac{p_d}{2m}\, d\mu + \int p_g \log \frac{p_g}{2m}\, d\mu \tag{16}$$

$$= \underbrace{\int p_d \log \frac{p_d}{m}\, d\mu}_{D_{\text{KL}}(P_{\text{data}} \| M)} + \underbrace{\int p_g \log \frac{p_g}{m}\, d\mu}_{D_{\text{KL}}(P_g \| M)} - 2\log 2. \tag{17}$$

Определим

$$D_{\text{JS}}(P\|Q) = \frac{1}{2} D_{\text{KL}}(P\|M) + \frac{1}{2} D_{\text{KL}}(Q\|M), \qquad M = \tfrac{1}{2}(P + Q). \tag{18}$$

Следовательно,

$$\boxed{C(G) = -\log 4 + 2 D_{\text{JS}}(P_{\text{data}} \| P_g).} \tag{19}$$

Это ключевой теоретический результат первоначальной GAN [8, 6].

6.2 Следствие: критерий идеального равновесия

Поскольку $D_{\mathrm{KL}}(P\|Q) \geq 0$, также $D_{\mathrm{JS}}(P\|Q) \geq 0$; равенство достигается тогда и только тогда, когда $P = Q$ почти всюду. Поэтому

$$C(G) \geq -\log 4, \qquad C(G) = -\log 4 \iff P_g = P_{\text{data}}. \tag{20}$$

При совпадении распределений $D^*(x) = 1/2$ почти всюду относительно их общей смеси. Здесь принципиально *идеализированное* допущение, что дискриминатор может реализовать оптимальную функцию, генератор обладает достаточной выразительностью и глобальная оптимизация выполнена. Практическое обучение нейронных сетей таких гарантий не обеспечивает.

6.3 Разделённые носители и проблема градиентов

Рассмотрим две точечные меры на прямой $P_{\text{data}} = \delta_0$ и $P_g = \delta_a$ при $a \neq 0$. Носители не пересекаются, и $D_{\mathrm{JS}}(P_{\text{data}}\|P_g) = \log 2$ независимо от расстояния $|a|$. Подставляя в (19), получаем $C(G) = 0$. Идеальный JS-критерий не различает расстояния между несовпадающими точками. Этот пример мотивирует альтернативные метрики, учитывающие геометрию пространства [12].

> **Научная точность** Из постоянства JS на этом частном семействе распределений *не следует*, что любые реальные GAN всегда имеют нулевые градиенты. В практической сети градиенты зависят от параметризации D, оптимизации и выбранной функции генератора. Пример лишь демонстрирует потенциальную проблему несогласованности дивергенции и геометрической близости.

7. Градиенты генератора и проблема насыщения

7.1 Дифференцирование через дискриминатор

Пусть дискриминатор возвращает вероятность $D(G_\theta(z)) = \sigma(a_\phi(G_\theta(z)))$, где a_ϕ — логит. Для одного примера обозначим $s = a_\phi(G_\theta(z))$, $d = \sigma(s)$. Тогда

$$\frac{\partial}{\partial s}\log(1-\sigma(s)) = -\sigma(s) = -d, \qquad \frac{\partial}{\partial s}[-\log\sigma(s)] = \sigma(s) - 1 = d - 1. \tag{21}$$

При $d \approx 0$ модуль производной minimax-loss по логиту мал, а для non-saturating loss приближается к 1. Это объясняет более полезный сигнал на ранних этапах обучения, когда поддельные образцы легко различимы.

Полный градиент по параметрам генератора вычисляется цепным правилом:

$$\nabla_\theta \mathcal{L}_G = \mathbb{E}_z \left[\frac{\partial \ell}{\partial s} \nabla_x a_\phi(x)\big|_{x=G_\theta(z)} J_\theta G_\theta(z) \right]. \tag{22}$$

Здесь $J_\theta G$ — якобиан генератора по параметрам. При обновлении G параметры D остаются фиксированными, но *градиент необходимо распространять через вычисления D к входному изображению*, иначе генератор не получит обучающий сигнал.

7.2 Численный пример для $D(G(z)) = 0.01$

Пусть дискриминатор почти уверен, что синтетическое изображение не является реальным: $d = 0.01$. По логиту дискриминатора minimax-loss даёт -0.01, а non-saturating loss — -0.99. При $d = 0.9$ соответствующие производные равны -0.9 и -0.1. Следовательно, преимущества и недостатки двух целей зависят от текущего состояния обеих моделей.

Пример 1. Сравнение производных на NumPy

```python
import numpy as np
for d in (0.01, 0.5, 0.9):
    grad_minimax = -d
    grad_ns = d - 1
    print(f"D(fake)={d:.2f}, MM={grad_minimax:.2f}, NS={grad_ns:.2f}")
```

Ожидаемый вывод

```
D(fake)=0.01, MM=-0.01, NS=-0.99
D(fake)=0.50, MM=-0.50, NS=-0.50
D(fake)=0.90, MM=-0.90, NS=-0.10
```

8. Алгоритм обучения GAN: от формул к циклу

8.1 Чередующиеся обновления

За одну итерацию обычно выполняются следующие операции:

1. получить мини-пакет реальных объектов $x_1, \dots, x_B$;
2. сгенерировать шум $z_1, \dots, z_B$ и поддельные образцы $G(z_i)$;
3. вычислить $\mathcal{L}_D$, обновить только параметры ϕ;
4. сгенерировать свежий либо повторно использованный шум z_i';
5. вычислить $\mathcal{L}_G^{\mathrm{NS}}$, обновить только параметры θ;
6. сохранить метрики, фиксированную сетку изображений и периодически оценивать качество на независимых примерах.

Число обновлений дискриминатора на обновление генератора может быть различным. Чрезмерно сильный дискриминатор затрудняет оптимизацию генератора; чрезмерно слабый не даёт содержательного сигнала. Универсального соотношения шагов нет.

8.2 Псевдокод

Пример 2. Базовый обучающий цикл в виде псевдокода Python

```
for real in dataloader:
    z = sample_noise(batch_size)
    fake = G(z)
    loss_d = BCE_logits(D(real), ones) + BCE_logits(
        D(fake.detach()), zeros)
    optimizer_d.zero_grad()
    loss_d.backward()
    optimizer_d.step()

    z = sample_noise(batch_size)
    fake = G(z)
    loss_g = BCE_logits(D(fake), ones)
    optimizer_g.zero_grad()
    loss_g.backward()
    optimizer_g.step()
```

`detach()` необходим для шага дискриминатора: иначе ненужные градиенты протекают в генератор. Во время шага генератора `detach()` недопустим: вычислительный граф должен соединять выход D с параметрами G.

8.3 Почему значения loss недостаточны

Потери обеих сетей не обязаны монотонно уменьшаться. При равновесии $D \approx 1/2$ дискриминаторная BCE-сумма для равновзвешенных классов приблизительно равна $2\log 2 \approx 1.386$, а non-saturating loss генератора $\log 2 \approx 0.693$. Однако похожие числа бывают и при плохой сети. Качество оценивают по изображениям, разнообразию, независимым статистическим метрикам и устойчивости по эпохам.

9. Модельная одноразмерная GAN

9.1 Задача и архитектура

Пусть реальные данные $X \sim \mathcal{N}(2, 0.5^2)$, а шум $Z \sim \mathcal{N}(0, 1)$. Генератор $G_\theta(z)$ — небольшая MLP, выдающая число; дискриминатор $D_\phi(x)$ — MLP с одним логитом на выходе. Студенты могут проследить, как эволюционирует гистограмма генератора относительно реального распределения.

Пример 3. Полная компактная реализация одномерной GAN

```python
import torch
from torch import nn

torch.manual_seed(42)
device = "cuda" if torch.cuda.is_available() else "cpu"
G = nn.Sequential(nn.Linear(1, 32), nn.ReLU(),
                  nn.Linear(32, 32), nn.ReLU(), nn.Linear(32, 1)).to(device)
D = nn.Sequential(nn.Linear(1, 32), nn.LeakyReLU(0.2),
                  nn.Linear(32, 1)).to(device)  # LOGITS
criterion = nn.BCEWithLogitsLoss()
opt_g = torch.optim.Adam(G.parameters(), lr=0.001)
opt_d = torch.optim.Adam(D.parameters(), lr=0.001)

for step in range(2000):
    real = 2.0 + 0.5 * torch.randn(128, 1, device=device)
    z = torch.randn(128, 1, device=device)
    fake = G(z)
    loss_d = criterion(D(real), torch.ones_like(D(real)))
    loss_d += criterion(D(fake.detach()), torch.zeros_like(D(fake)))
    opt_d.zero_grad(); loss_d.backward(); opt_d.step()

    z = torch.randn(128, 1, device=device)
    fake = G(z)
    logits_fake = D(fake)
    loss_g = criterion(logits_fake, torch.ones_like(logits_fake))
    opt_g.zero_grad(); loss_g.backward(); opt_g.step()

with torch.no_grad():
```

Пример 3 (продолжение)

```
    samples = G(torch.randn(20000, 1, device=device)).cpu().numpy().ravel()
print("Target: mean=2, std=0.5")
print("Generated mean/std:", samples.mean(), samples.std())
```

Ожидаемый результат. Обученный генератор должен получать выборку со средним, близким к 2, и стандартным отклонением, близким к 0.5. Точные числа зависят от оборудования, версии PyTorch, случайности и того, достигнута ли сходимость. Совпадение только первых двух моментов *не доказывает* равенство распределений: полезно дополнительно строить гистограммы, эмпирические функции распределения и сравнивать квантили.

10. Свёрточные GAN для изображений: DCGAN

10.1 От полносвязных слоёв к транспонированным свёрткам

Полносвязный генератор для изображений размера $64 \times 64 \times 3$ быстро становится громоздким. DCGAN использует сверточную архитектуру: генератор преобразует скрытый вектор в изображения с помощью `ConvTranspose2d`, а дискриминатор выполняет пространственную агрегацию с помощью `Conv2d`. Стандартный исторический образец DCGAN обсуждается в [10].

Транспонированная свёртка не равна математически точной обратной свёртке; она реализует линейный оператор, матрица которого транспонирована матрице соответствующей обычной свёртки. Для оси изображения выходная размерность равна

$$H_{\text{out}} = (H_{\text{in}} - 1)s - 2p + d(k-1) + o + 1, \tag{23}$$

где s — stride, p — padding, d — dilation, k — kernel size, o — output padding, причём $0 \le o < s$ для распространённых настроек PyTorch.

10.2 Пошаговый расчёт размеров

Пусть исходная карта имеет размер 4×4, а слои транспонированной свёртки имеют параметры $(k, s, p, d, o) = (4, 2, 1, 1, 0)$. Из (23) следует $H_{\text{out}} = 2H_{\text{in}}$. Таким образом:

$$4 \times 4 \longrightarrow 8 \times 8 \longrightarrow 16 \times 16 \longrightarrow 32 \times 32 \longrightarrow 64 \times 64. \tag{24}$$

При этом размер каналов обычно уменьшается по мере роста пространственного разрешения.

Блок генератора	Вход (C, H, W)	Выход (C, H, W)
Linear + reshape	$(100, 1, 1)$	$(512, 4, 4)$
ConvTranspose 1	$(512, 4, 4)$	$(256, 8, 8)$
ConvTranspose 2	$(256, 8, 8)$	$(128, 16, 16)$
ConvTranspose 3	$(128, 16, 16)$	$(64, 32, 32)$
ConvTranspose 4	$(64, 32, 32)$	$(3, 64, 64)$

10.3 Нормализация изображений и функции активации

Если последний слой генератора — tanh, то его значения лежат в $[-1, 1]$; реальные изображения также должны быть нормированы в $[-1, 1]$. Смешивать диапазоны $[0, 1]$ и $[-1, 1]$ недопустимо: это облегчает задачу дискриминатору по несущественному признаку. В промежуточных слоях генератора часто применяют BatchNorm и ReLU, в дискриминаторе — LeakyReLU. Однако архитектурные рекомендации DCGAN — проверенные эвристики, а не универсальные теоремы [10, 4].

11. Реализация DCGAN на PyTorch: модель

11.1 Генератор и дискриминатор

Следующий пример демонстрирует совместимые тензорные формы для цветных изображений 64×64.

Пример 4. Архитектуры DCGAN для изображений 64 x 64

```python
import torch
from torch import nn

class Generator(nn.Module):
```

Пример 4 (продолжение)

```
    def __init__(self, nz=100, nc=3, ngf=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.ConvTranspose2d(nz, ngf*8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf*8), nn.ReLU(True),
            nn.ConvTranspose2d(ngf*8, ngf*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf*4), nn.ReLU(True),
            nn.ConvTranspose2d(ngf*4, ngf*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf*2), nn.ReLU(True),
            nn.ConvTranspose2d(ngf*2, ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf), nn.ReLU(True),
            nn.ConvTranspose2d(ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh())

    def forward(self, z):
        return self.net(z)

class Discriminator(nn.Module):
    def __init__(self, nc=3, ndf=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(nc, ndf, 4, 2, 1), nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf, ndf*2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf*2), nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf*2, ndf*4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf*4), nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf*4, ndf*8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf*8), nn.LeakyReLU(0.2, True),
            nn.Conv2d(ndf*8, 1, 4, 1, 0))  # logits

    def forward(self, x):
        return self.net(x).flatten(1)

G, D = Generator(), Discriminator()
with torch.no_grad():
    z = torch.randn(8, 100, 1, 1)
    images = G(z)
    logits = D(images)
print(tuple(images.shape), tuple(logits.shape))
```

Ожидаемый вывод

```
(8, 3, 64, 64) (8, 1)
```

11.2 Правильная инициализация

Для DCGAN часто используют нормальную инициализацию свёрточных весов со стандартным отклонением 0.02; для BatchNorm веса задают близкими к 1, смещения — 0 [10].

Пример 5. Инициализация параметров DCGAN

```
def init_dcgan(m):
    name = m.__class__.__name__
    if "Conv" in name:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif "BatchNorm" in name:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.zeros_(m.bias.data)

G.apply(init_dcgan)
D.apply(init_dcgan)
```

12. Подготовка датасета и честный эксперимент

12.1 Практический сценарий

В качестве доступной учебной задачи можно использовать открытый датасет изображений с однородными объектами, например набор фотографий лиц (при соблюдении прав и правил использования), набор изображений одежды либо иной разрешённый набор. Для небольшого вычислительного бюджета достаточно привести изображения к 64×64, использовать пакет 64 и сравнить результаты нескольких эпох. Не следует обещать, что DCGAN за фиксированное число эпох обязательно создаст реалистичные изображения: результат зависит от данных и аппаратных ресурсов.

Пример 6. Преобразования изображений для DCGAN

```
from torchvision import transforms

transform = transforms.Compose([
    transforms.Resize(64),
    transforms.CenterCrop(64),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
# Для изображений RGB выход transform имеет форму (3, 64, 64)
# и диапазон значений примерно [-1, 1].
```

> Для одноканальных данных применяют $nc = 1$ и нормализацию с одним средним и одним стандартным отклонением. Набор данных не следует произвольно дублировать по train/test и затем сообщать метрики качества как независимые.

12.2 Фиксированный шум для мониторинга

Чтобы сравнивать качество по эпохам, зафиксируйте набор z^{fixed} и показывайте изображения $G_{\theta_e}(z^{\text{fixed}})$ после каждой эпохи e. Тогда изменения отражают эволюцию *одних и тех же* скрытых кодов. Помимо этого полезно сохранять отдельные случайные образцы для оценки разнообразия.

Пример 7. Шум, сохранение генераций и режим eval

```python
import torchvision.utils as vutils
fixed_noise = torch.randn(64, 100, 1, 1, device=device)
G.eval()
with torch.no_grad():
    preview = G(fixed_noise).cpu()
    vutils.save_image(preview, "gan_epoch_preview.png",
                      nrow=8, normalize=True, value_range=(-1, 1))
G.train()
```

13. Полный учебный шаг обучения DCGAN

13.1 Логиты, метки и блокировка ненужных градиентов

Для корректного запуска ниже предполагаем: G и D определены, перенесены на один `device`, `real` имеет форму $(B, 3, 64, 64)$ и значения $[-1, 1]$. Пример иллюстрирует единичную итерацию и легко помещается внутрь цикла по `DataLoader`.

Пример 8. Один полный шаг обучения генератора и дискриминатора

```python
import torch
from torch import nn

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
G, D = Generator().to(device), Discriminator().to(device)
optG = torch.optim.Adam(G.parameters(), lr=2e-4, betas=(0.5, 0.999))
optD = torch.optim.Adam(D.parameters(), lr=2e-4, betas=(0.5, 0.999))
bce = nn.BCEWithLogitsLoss()

# real = next(iter(loader))[0].to(device)
real = torch.rand(16, 3, 64, 64, device=device) * 2 - 1  # тест форм
B = real.size(0)

# Шаг D
noise = torch.randn(B, 100, 1, 1, device=device)
fake = G(noise)
logits_real = D(real)
```

Пример 8 (продолжение)

```
logits_fake = D(fake.detach())
lossD = bce(logits_real, torch.ones_like(logits_real))
lossD += bce(logits_fake, torch.zeros_like(logits_fake))
optD.zero_grad(set_to_none=True)
lossD.backward()
optD.step()

# Шаг G
noise = torch.randn(B, 100, 1, 1, device=device)
fake = G(noise)
logits_fake_for_G = D(fake)
lossG = bce(logits_fake_for_G, torch.ones_like(logits_fake_for_G))
optG.zero_grad(set_to_none=True)
lossG.backward()
optG.step()
print("One training step executed", tuple(fake.shape))
```

Предупреждение: строка `torch.rand(...)` создаёт случайные пиксели и приведена *только для теста совместимости формы и градиентов*; полноценное обучение обязательно должно использовать реальные изображения из датасета. На небольших пакетах статистики BatchNorm могут существенно колебаться. Для воспроизводимости сохраните версии пакетов, seed, конфигурацию модели и применённые преобразования.

14. Проблемы GAN: mode collapse и динамика игры

14.1 Что такое mode collapse

Распределение данных может иметь несколько мод — областей высокой вероятности. Если генератор воспроизводит лишь малую часть этих областей, то возникает *mode collapse*: множество различных входов z порождают одинаковые или очень похожие объекты.

Представим данные из двух групп: красные и синие изображения, каждая с вероятностью $1/2$. Если генератор выпускает только красные изображения, его образцы могут быть локально правдоподобны, но он неправильно описывает распределение. Поэтому визуальная реалистичность отдельных объектов не равна покрытию всего пространства данных.

14.2 Наблюдаемые признаки коллапса

- В сетке из разных z повторяется одно и то же лицо, цифра или композиция.
- Между эпохами генератор резко переключается между несколькими типами объектов.
- Метрики разнообразия резко ухудшаются при внешне хороших изображениях.
- Средние статистики распределения не отражают многомодальность.

14.3 Практические меры

Популярные меры: уменьшить мощность дискриминатора; изменить баланс шагов; подобрать learning rate; применить спектральную нормализацию; использовать условные метки; выбрать модифицированную функцию потерь (например, WGAN-GP). Ни одна мера не служит универсальным доказательством устранения mode collapse [12, 13, 14].

15. Условные генеративно-состязательные сети (Conditional GAN)

15.1 Генерация при заданном условии

В обычной GAN объект определяется случайным кодом z. В cGAN мы дополнительно передаём условие y: метку класса, характеристику объекта или другое наблюдение. Генератор принимает (z, y), дискриминатор оценивает согласованность (x, y) [11].

$$\min_G \max_D V(G,D) = \mathbb{E}_{(x,y)\sim P_{\text{data}}} \log D(x,y) + \mathbb{E}_{y\sim P_y, z\sim P_z} \log\big(1 - D(G(z,y),y)\big). \quad (25)$$

Условие y при синтетическом семплировании нужно получать из корректного распределения классов и согласованно подавать *обеим* сетям. Если в эксперименте реальная выборка несбалансирована, выбор распределения P_y при генерации влияет на распределение результата.

15.2 Схемы подачи меток

1. **One-hot:** метка преобразуется в вектор длины K и объединяется с шумом z.
2. **Embedding:** метка отображается в обучаемый вектор небольшой размерности.
3. **Пространственный канал:** embedding преобразуется в карту и конкатенируется с изображением.
4. **Projection discriminator:** соответствие классу учитывается скалярным произведением признаков и embedding.

Для первой учебной реализации достаточно one-hot и MLP.

Пример 9. Условный генератор и дискриминатор для MNIST

```python
import torch
from torch import nn

class ConditionalG(nn.Module):
    def __init__(self, latent_dim=64, classes=10):
        super().__init__()
        self.embed = nn.Embedding(classes, 16)
        self.net = nn.Sequential(
            nn.Linear(latent_dim + 16, 256), nn.ReLU(),
            nn.Linear(256, 512), nn.ReLU(),
            nn.Linear(512, 28 * 28), nn.Tanh())
    def forward(self, z, y):
        x = torch.cat((z, self.embed(y)), dim=1)
        return self.net(x).view(-1, 1, 28, 28)

class ConditionalD(nn.Module):
    def __init__(self, classes=10):
        super().__init__()
        self.embed = nn.Embedding(classes, 16)
        self.net = nn.Sequential(
            nn.Linear(28 * 28 + 16, 256), nn.LeakyReLU(0.2),
            nn.Linear(256, 1))  # logits
    def forward(self, image, y):
        x = torch.cat((image.flatten(1), self.embed(y)), dim=1)
        return self.net(x)

G_c, D_c = ConditionalG(), ConditionalD()
z = torch.randn(8, 64)
y = torch.tensor([0, 1, 2, 3, 4, 5, 6, 7])
with torch.no_grad():
    images = G_c(z, y)
    logits = D_c(images, y)
print(tuple(images.shape), tuple(logits.shape))
```

Ожидаемый вывод

```
(8, 1, 28, 28) (8, 1)
```

Демонстрация результата: студент выбирает цифру $y = 7$, после чего генератор выдаёт несколько *разных* изображений семёрки при разных z. Если он выдаёт одинаковые изображения, необходимо диагностировать условный mode collapse.

16. Wasserstein GAN: другая геометрия сравнения распределений

16.1 Мотивация: расстояние переноса

Вернёмся к δ_0 и δ_a. *Расстояние Вассерштейна первого порядка* для этих распределений равно

$$W_1(\delta_0, \delta_a) = |a|, \tag{26}$$

то есть отражает удалённость их носителей, в отличие от JS. В общем случае

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big], \tag{27}$$

где $\Pi(P, Q)$ — множество совместных распределений с маргиналами P и Q. Минимум вычисляется по всем допустимым способам «переноса» массы из одного распределения в другое.

При необходимых условиях конечности первых моментов теорема Канторовича–Рубинштейна даёт двойственную форму

$$W_1(P, Q) = \sup_{\|f\|_{\mathrm{Lip}} \le 1} \left(\mathbb{E}_{x\sim P} f(x) - \mathbb{E}_{x\sim Q} f(x)\right). \tag{28}$$

Условие $\|f\|_{\mathrm{Lip}} \le 1$ означает $|f(x) - f(y)| \le \|x - y\|$. Эта формула лежит в основе WGAN [12].

16.2 Критик вместо вероятностного дискриминатора

WGAN использует сеть $f_\phi : \mathbb{R}^d \to \mathbb{R}$, называемую *критиком*. Её выход — *не вероятность* и не должен ограничиваться sigmoid. Состязательная задача записывается как

$$\min_{\theta} \max_{\phi \in \Phi_{\text{Lip}}} \left[\mathbb{E}_{x \sim P_{\text{data}}} f_\phi(x) - \mathbb{E}_{z \sim P_z} f_\phi(G_\theta(z)) \right]. \tag{29}$$

Критик стремится увеличить разницу оценок реальных и синтетических объектов; генератор стремится увеличить среднюю оценку своих объектов. При реализации с оптимизаторами минимизации удобно положить

$$\mathcal{L}_{\text{critic}} = \mathbb{E}_z f_\phi(G(z)) - \mathbb{E}_x f_\phi(x), \tag{30}$$

$$\mathcal{L}_G^{\text{WGAN}} = -\mathbb{E}_z f_\phi(G_\theta(z)). \tag{31}$$

16.3 Почему требуется ограничение липшицевости

Если не ограничить f, то умножение f на произвольно большое число увеличивало бы разность ожиданий без ограничения, и двойственная формула перестала бы вычислять W_1. Исходный WGAN предлагал *weight clipping* как приближение к требованию 1-Lipschitz, но это может чрезмерно ограничивать семейство критиков и затруднять оптимизацию [12, 13].

17. WGAN с градиентным штрафом (WGAN-GP)

17.1 Идея штрафа

В WGAN-GP применяется штраф на норму градиента критика по входу [13]. Определяют

$$\widehat{x} = \varepsilon x + (1-\varepsilon)\widetilde{x}, \qquad \varepsilon \sim U[0,1],\ \widetilde{x} = G(z), \tag{32}$$

и добавляют к функции потерь критика

$$\mathcal{L}_{\text{GP}} = \lambda \mathbb{E}_{\widehat{x}} \left(\|\nabla_{\widehat{x}} f_\phi(\widehat{x})\|_2 - 1 \right)^2 . \tag{33}$$

Финальная минимизируемая критиком функция:

$$\mathcal{L}_C = \mathbb{E}_{\widetilde{x}} f_\phi(\widetilde{x}) - \mathbb{E}_x f_\phi(x) + \mathcal{L}_{\text{GP}}. \tag{34}$$

Такой штраф обеспечивает *мягкое приближение* условия на липшицевость лишь на выбранных интерполированных точках; он сам по себе не доказывает глобальную 1-липшицевость обученного критика.

17.2 Реализация gradient penalty в PyTorch

Пример 10. Функция градиентного штрафа для WGAN-GP

```python
import torch

def gradient_penalty(critic, real, fake, lam=10.0):
    b = real.size(0)
    eps_shape = (b,) + (1,) * (real.ndim - 1)
    eps = torch.rand(eps_shape, device=real.device)
    mixed = (eps * real + (1 - eps) * fake.detach()).requires_grad_(True)
    scores = critic(mixed).view(b)
    gradients = torch.autograd.grad(
        outputs=scores,
        inputs=mixed,
        grad_outputs=torch.ones_like(scores),
        create_graph=True,
        retain_graph=True,
        only_inputs=True
    )[0]
    norm = gradients.flatten(1).norm(2, dim=1)
    return lam * ((norm - 1) ** 2).mean()
```

Здесь функция `critic` должна возвращать *скаляр на объект*. Для WGAN-GP не следует автоматически переносить архитектуру дискриминатора с BatchNorm: из-за зависимости нормализованных выходов от других образцов в пакете производная по конкретному x_i становится неоднозначной для поточечного штрафа; обычно применяют LayerNorm, InstanceNorm при подходящих обстоятельствах или исключают нормализацию критика [13].

18. Сравнение вариантов GAN

Модель	Цель	Особенность	Ограничение
Vanilla GAN	Состязательная BCE	Простая исходная модель	Нестабильная динамика
DCGAN	BCE с CNN	Генерация изображений	Чувствительность к обучению
cGAN	Условное состязание	Управление классом	Возможен коллапс по классу
WGAN	Приближение W_1	Критик вместо вероятности	Требование Lipschitz
WGAN-GP	W_1 + штраф	Градиентная регуляризация критика	Дополнительная стоимость градиентов

19. Оценка качества генеративных моделей

19.1 Визуальное качество и разнообразие

Качественная оценка должна учитывать *реалистичность*, *разнообразие* и *соответствие задаче*. Последние два свойства различны: модель может создавать красивые изображения только одного класса и тем самым плохо покрывать распределение.

19.2 Inception Score

Для генеративных изображений исторически использовали Inception Score (IS) [15]:

$$\mathrm{IS} = \exp\left(\mathbb{E}_{x\sim P_g} D_{\mathrm{KL}}(p(y \mid x) \| p(y))\right), \tag{35}$$

где $p(y \mid x)$ — предсказания заранее обученного классификатора, а $p(y) = \mathbb{E}_{x\sim P_g} p(y \mid x)$. IS поощряет уверенные предсказания отдельным изображениям и разнообразие предсказанных классов, но не сравнивает напрямую распределение с реальным и зависит от классификатора. Поэтому IS не следует использовать как единственный критерий.

19.3 Fréchet Inception Distance

FID сравнивает нормальные аппроксимации эмпирических распределений признаков $\varphi(x)$ реальных и синтетических изображений [16]. Если средние и ковариации равны (μ_r, Σ_r) и (μ_g, Σ_g), то

$$\mathrm{FID} = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2\left(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\right)^{1/2}\right). \tag{36}$$

Формула с симметричным произведением внутри корня численно удобнее для объяснения, чем несимметричная запись с $(\Sigma_r\Sigma_g)^{1/2}$; обе совпадают по следу при соответствующих математических предпосылках. **Меньшее значение FID обычно лучше**, но оценка смещена на конечной выборке и зависит от предварительной обработки, размера выборки и реализации признакового экстрактора.

19.4 Численный пример FID в одном измерении

Для одномерных гауссовых признаков с $\mu_r = 1$, $\sigma_r = 2$, $\mu_g = 3$, $\sigma_g = 1$ получаем

$$\mathrm{FID} = (1-3)^2 + (2-1)^2 = 4 + 1 = 5. \tag{37}$$

Это не FID реальных фотографий: пример лишь поясняет геометрию среднего и стандартного отклонения в формуле.

19.5 Precision и Recall генеративных моделей

Для диагностики покрытия распределения полезно анализировать *precision* генерации (какая доля синтетических объектов выглядит правдоподобно относительно данных) и *recall* (какая доля областей реальных данных покрыта генератором). Конкретные оценки зависят от определения опорных областей в пространстве признаков. Студент должен указывать выбранную методику и не смешивать эти показатели с precision/recall бинарного дискриминатора.

20. Частые ошибки реализации и их исправление

1. **Sigmoid и BCEWithLogitsLoss одновременно.** Приводит к неправильной функции потерь. Решение: возвращать из D логиты.
2. **Отсутствие detach на шаге D.** Расточительно вычисляет градиенты G. Решение: передавать fake.detach().
3. **Наличие detach на шаге G.** Блокирует градиент к G. Решение: использовать выход $G(z)$ без разрыва вычислительного графа.
4. **Неверные метки.** При non-saturating loss генератора меткой синтетического объекта для критерия G служит класс «реальный».
5. **Несогласованная нормализация.** Диапазон реальных изображений должен соответствовать последнему слою G.
6. **Слишком сильный дискриминатор.** Может подавить полезный сигнал; проверяйте баланс, архитектуры и скорости обучения.
7. **Оценка только по loss.** Даже «правильные» средние потери не гарантируют хорошую генерацию.
8. **Отсутствие фиксированного шума.** Затрудняет сравнение эпох по одинаковым латентным кодам.
9. **Оценка FID по малому набору.** Сравнение становится ненадёжным и трудно воспроизводимым.
10. **Путаница WGAN и GAN.** Критик WGAN не выдаёт вероятность и не обучается с BCE.

21. Проектная работа на четыре недели

Тема: «Генерация рукописных цифр с использованием DCGAN или Conditional GAN».

Неделя	Что сделать	Артефакт для отчёта
1	Изучить данные MNIST; реализовать нормализацию, визуализацию, определить входы/выходы.	EDA, параметры датасета, архитектурная схема.
2	Обучить базовую MLP GAN; сохранить фиксированный шум и результат эпох.	Код, графики потерь, сетка генераций.
3	Реализовать DCGAN или cGAN; выполнить 2–3 сравнения гиперпараметров.	Сравнительная таблица, визуальный анализ mode collapse.
4	Создать интерфейс для выбора цифры/генерации; оценить разнообразие, оформить отчёт.	Демо-приложение, модели, отчёт, презентация.

Требования к итоговой сдаче: исполняемый проект Python/PyTorch; инструкции установки; обученные веса либо скрипт для их получения; минимум три настройки обучения; таблица гиперпараметров; сохранённые сетки изображений; отдельная оценка разнообразия; краткая демонстрация нового семплирования в интерфейсе. Если имеется только CPU, допускается упрощённая MLP GAN на MNIST с ясным описанием аппаратных ограничений.

22. Разбор нескольких задач с решениями

22.1 Задача 1. Потеря дискриминатора

Пусть на одном реальном изображении $D(x) = 0.8$, на одном поддельном $D(G(z)) = 0.2$. Суммарная отрицательная логарифмическая потеря равна

$$\mathcal{L}_D = -\log(0.8) - \log(1 - 0.2) = -2\log 0.8 \approx 0.44629. \tag{38}$$

Если используем среднее по двум отдельным объектам, а не сумму двух усреднённых слагаемых, результат составит 0.22314. Поэтому при сравнении экспериментов надо фиксировать конвенцию нормировки loss.

22.2 Задача 2. Производная по логиту

Обозначим $d = \sigma(s)$. Для non-saturating loss $\ell(s) = -\log \sigma(s)$:

$$\ell'(s) = -\frac{\sigma'(s)}{\sigma(s)} = -\frac{\sigma(s)(1-\sigma(s))}{\sigma(s)} = \sigma(s) - 1. \tag{39}$$

Для $d = 0.15$ производная равна -0.85. Если другие множители цепного правила ненулевые, генератор получает существенный градиент.

22.3 Задача 3. Проверка равновесия

Если $P_g = P_{\text{data}}$, то по (11) $D^* = 1/2$, а по (19) $C(G) = -\log 4 \approx -1.38629$. При этом $\mathcal{L}_D = -V = \log 4 \approx 1.38629$, а non-saturating loss равна $\log 2 \approx 0.69315$.

22.4 Задача 4. Размер транспонированной свёртки

Пусть $H_{\text{in}} = 7$, $k = 3$, $s = 2$, $p = 1$, $d = 1$, $o = 1$. По (23)

$$H_{\text{out}} = (7-1)\cdot 2 - 2 + 2 + 1 + 1 = 14. \tag{40}$$

Нередко студенты ошибаются, считая $H_{\text{out}} = 13$: дополнительный параметр output padding изменяет размер выхода, а не фактическое дополнение входа нулями.

22.5 Задача 5. Два точечных распределения

Для $P = \delta_0$, $Q = \delta_5$: $W_1(P, Q) = 5$, $D_{\text{JS}}(P\|Q) = \log 2$. Если приблизить вторую меру к δ_1, Wasserstein уменьшается до 1, а JS не меняется. Этот расчёт демонстрирует, почему расстояние переноса помогает отражать геометрическую близость раздельных распределений.

23. Контрольные вопросы

1. Чем генеративная модель отличается от классификатора?
2. Что обозначают P_{data}, P_z и P_g?
3. Что значит неявное задание распределения через $G(z)$?
4. Какие роли выполняют генератор и дискриминатор?
5. Почему обучение GAN называют состязательным?
6. Запишите оригинальную minimax-функцию GAN.
7. Как записать loss дискриминатора для минимизирующего оптимизатора?
8. Чем non-saturating loss отличается от minimax loss?
9. Почему в PyTorch применяют BCEWithLogitsLoss?
10. При каких допущениях можно вывести оптимальный D^*?
11. Получите $D^*(x)$ из первого порядка условия оптимальности.
12. Почему вторая производная доказывает максимум?
13. Как $D^*(x)$ связан с формулой Байеса?
14. Что означает $D^*(x) = 1/2$?
15. Дайте определение KL-дивергенции.
16. Дайте определение JS-дивергенции.
17. Выведите $C(G) = -\log 4 + 2\,\mathrm{JS}$.
18. При каком распределении генератора достигается глобальный минимум $C(G)$?
19. Почему этот теоретический результат не гарантирует практическую сходимость?
20. Как различаются градиенты MM и NS для $D(G(z)) \approx 0$?
21. Для чего используется detach при обучении дискриминатора?
22. Почему нельзя применять detach на шаге генератора?
23. Почему обе loss-функции могут колебаться во время обучения?
24. Что такое mode collapse?
25. Чем реалистичность изображений отличается от разнообразия?
26. Какие особенности содержит DCGAN?
27. Что такое ConvTranspose2d и как вычислять выходную размерность?
28. Почему важна нормализация данных в диапазоне выхода генератора?
29. Какова роль фиксированного латентного шума при мониторинге?
30. Чем cGAN отличается от обычной GAN?
31. Что подавать на вход условного дискриминатора?
32. Почему генерация одного класса не доказывает успех cGAN?
33. Запишите W_1 через совместные распределения.
34. Сформулируйте двойственную формулу Канторовича–Рубинштейна.

35. Что означает условие 1-Lipschitz?
36. Почему критик WGAN не должен использовать sigmoid на выходе?
37. Чем WGAN отличается от Vanilla GAN по функции потерь?
38. В чём недостатки weight clipping?
39. Как формируется градиентный штраф WGAN-GP?
40. Почему WGAN-GP не гарантирует глобальную липшицевость?
41. Что оценивает Inception Score и чего он не оценивает?
42. Что такое FID и какие статистики используются в формуле?
43. Почему FID зависит от размера выборки и предобработки?
44. Как организовать честное сравнение двух GAN?
45. Какие файлы и графики должен содержать воспроизводимый GAN-проект?

24. Задания для самостоятельной работы

1. Докажите формулу оптимального дискриминатора при конечном дискретном множестве из K объектов.
2. Для $P_d = (0.5, 0.25, 0.25)$ и $P_g = (0.2, 0.2, 0.6)$ вычислите D^* и $C(G)$.
3. Повторите вывод связи GAN с JS, явно указав все предположения.
4. Сравните производные minimax и non-saturating loss на сетке $d = 0.01, 0.1, 0.5, 0.9, 0.99$.
5. Реализуйте 1D GAN для смеси $\frac{1}{2}\mathcal{N}(-2, 0.2^2) + \frac{1}{2}\mathcal{N}(2, 0.2^2)$; проверьте mode collapse.
6. Измените число скрытых слоёв генератора и оцените влияние на покрытие мод.
7. Реализуйте DCGAN для набора изображений размера 64×64; проверьте формы всех промежуточных тензоров.
8. Сравните два learning rate, фиксируя все остальные условия и seed.
9. Реализуйте cGAN на MNIST, визуализируя для каждой метки несколько выходов.
10. Исследуйте влияние количества шагов дискриминатора на качество генерации.
11. Напишите свою функцию gradient penalty и проверьте форму градиентов на тестовой сети.
12. Вычислите вручную FID для одномерных признаковых гауссовых распределений с разными параметрами.
13. Объясните, почему невысокая потеря дискриминатора не обязательно означает высокое качество генератора.
14. Подготовьте таблицу сравнения GAN, DCGAN, cGAN, WGAN и WGAN-GP для выбранного практического проекта.
15. Сделайте приложение Gradio или Streamlit для генерации цифры по выбранной метке и кнопке «Сгенерировать».

25. Итоги и переход к следующей лекции

GAN — это способ неявного моделирования данных посредством конкуренции генератора и дискриминатора. Центральная теорема связывает оптимальный дискриминатор с отношением плотностей, а оптимальную минимаксную ценность — с JS-дивергенцией. Но идеальное равновесие не обеспечивает успешную работу алгоритма в конечной выборке и при ограниченной вычислительной мощности. Для инженерной практики необходимо отдельно контролировать баланс обучения, разнообразие, предобработку изображений и достоверность показателей качества.

В следующей лекции будут рассмотрены генеративные модели следующего поколения, в частности методы, непосредственно моделирующие процесс постепенного удаления шума, и их сопоставление с GAN и VAE.

Библиография

Список литературы

[1] Траск Э. *Грокаем глубокое обучение*. СПб.: Питер. (Русское издание книги: Trask A. *Grokking Deep Learning*. Manning, 2019.)

[2] Рашид Т. *Создаём нейронную сеть*. СПб.: Диалектика. (Оригинал: Rashid T. *Make Your Own Neural Network*. 2016.)

[3] Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed. O'Reilly, 2022.

[4] Chollet F. *Deep Learning with Python*. 2nd ed. Manning, 2021.

[5] Stevens E., Antiga L., Viehmann T. *Deep Learning with PyTorch*. Manning, 2020.

[6] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016. https://www.deeplearningbook.org/

[7] Nielsen M. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/

[8] Goodfellow I. et al. Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 2014. https://arxiv.org/abs/1406.2661

[9] Kingma D. P., Welling M. Auto-Encoding Variational Bayes. *ICLR*, 2014. https://arxiv.org/abs/1312.6114

[10] Radford A., Metz L., Chintala S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *ICLR*, 2016. https://arxiv.org/abs/1511.06434

[11] Mirza M., Osindero S. Conditional Generative Adversarial Nets, 2014. https://arxiv.org/abs/1411.1784

[12] Arjovsky M., Chintala S., Bottou L. Wasserstein GAN. *ICML*, 2017. https://arxiv.org/abs/1701.07875

[13] Gulrajani I. et al. Improved Training of Wasserstein GANs. *NeurIPS*, 2017. https://arxiv.org/abs/1704.00028

[14] Miyato T. et al. Spectral Normalization for Generative Adversarial Networks. *ICLR*, 2018. https://arxiv.org/abs/1802.05957

[15] Salimans T. et al. Improved Techniques for Training GANs. *NeurIPS*, 2016. https://arxiv.org/abs/1606.03498

[16] Heusel M. et al. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *NeurIPS*, 2017. https://arxiv.org/abs/1706.08500