Лекция 7. VAE. Вариационные автоэнкодеры

Бабич Павел Васильевич

ИММиКН

2026

Autoencoder: сжатие и реконструкция

Пусть $x \in \mathbb{R}^D$, $z \in \mathbb{R}^d$. Детерминированный автоэнкодер задаётся отображениями

$$z = f_\phi(x), \qquad \hat{x} = g_\theta(z), \qquad (\theta^*, \phi^*) \in \arg\min_{\theta,\phi} \frac{1}{N} \sum_{i=1}^{N} \ell\big(x_i, g_\theta(f_\phi(x_i))\big).$$

Например, при $\ell(x, \hat{x}) = \|x - \hat{x}\|_2^2$ минимизируется ошибка реконструкции.

- **Undercomplete AE:** $d < D$ — информационное «бутылочное горлышко».
- **Overcomplete AE:** $d \geq D$ — без ограничений возможна тривиальная копия входа.
- Линейный AE с MSE при подходящих условиях восстанавливает главное подпространство PCA.

Ограничение

У обычного AE нет заданного prior $p(z)$: случайный $z \sim \mathcal{N}(0, I)$ не обязан декодироваться в правдоподобный объект.

PyTorch: обычный автоэнкодер

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

encoder = nn.Sequential(nn.Linear(784, 128), nn.ReLU(),
                        nn.Linear(128, 16))
decoder = nn.Sequential(nn.Linear(16, 128), nn.ReLU(),
                        nn.Linear(128, 784), nn.Sigmoid())

x = torch.rand(32, 784)        # example batch in [0, 1]
z = encoder(x)
x_hat = decoder(z)
loss = F.mse_loss(x_hat, x)
loss.backward()               # gradients for both networks
```

Для реального обучения нужны optimizer и цикл по батчам. Вектор z здесь детерминированный.

Denoising autoencoder: восстановление из шума

Пусть $\tilde{x} \sim c(\tilde{x} \mid x)$ — зашумлённое наблюдение, например

$$\tilde{x} = x + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2 I).$$

Обучаем восстановление *чистого* объекта из повреждённого:

$$\boxed{\mathcal{J}_{\mathrm{DAE}}(\theta, \phi) = \mathbb{E}_{x \sim p_{\mathrm{data}}} \mathbb{E}_{\tilde{x} \sim c(\cdot \mid x)} \big[\ell\big(x, g_\theta(f_\phi(\tilde{x}))\big) \big].}$$

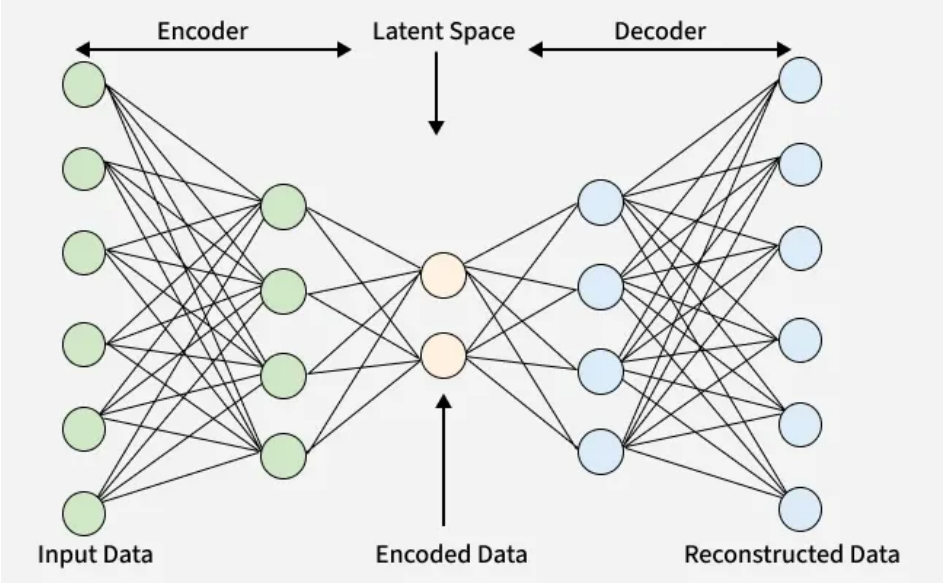
- Шум мешает выучить тождественное отображение и поощряет устойчивые признаки.
- Варианты: гауссов шум, masking noise, случайное зануление пикселей.
- **Важно:** DAE сам по себе не задаёт нормированную плотность $p_\theta(x)$.

PyTorch: denoising autoencoder

```
# Reuse encoder and decoder from the previous slide.
clean = torch.rand(32, 784)
noise_std = 0.3
noisy = (clean + noise_std * torch.randn_like(clean))\
        .clamp(0.0, 1.0)

reconstruction = decoder(encoder(noisy))
loss_dae = F.mse_loss(reconstruction, clean)
loss_dae.backward()
```

Цель — восстановить `clean`, а не `noisy`. Для последовательных шагов обучения нужно обнулять градиенты optimizer перед `backward()`. **Переход к VAE:** как сделать латентное пространство не только полезным для реконструкции, но и пригодным для вероятностной генерации?


Encoder
Latent Space
Decoder
Input Data
Encoded Data
Reconstructed Data

Задача генеративного моделирования

Наблюдаем $\mathcal{D} = \{x_i\}_{i=1}^N$, где $x_i \sim p_{\text{data}}(x)$.

Цель максимального правдоподобия

$$\theta^* \in \arg\max_\theta \sum_{i=1}^N \log p_\theta(x_i).$$

Желаемые операции: генерация $x \sim p_\theta$; оценка $\log p_\theta(x)$; представление z.

Ключевое различие: уметь сэмплировать x не означает уметь вычислять $p_\theta(x)$.

Модель с латентными переменными

$$z \sim p(z), \qquad x \mid z \sim p_\theta(x \mid z), \qquad p_\theta(x, z) = p(z)p_\theta(x \mid z).$$

Маргинальное распределение:

$$\boxed{p_\theta(x) = \int_{\mathbb{R}^d} p_\theta(x \mid z)p(z)\, dz.}$$

Обычно $p(z) = \mathcal{N}(0, I)$; параметры условного распределения задаёт нейросеть.

Вычислительная проблема

Интеграл по z обычно не вычисляется аналитически, даже если $p_\theta(x \mid z)$ известно.

PyTorch: prior и генеративный процесс

```
import torch
from torch import nn

latent_dim, data_dim = 2, 2
decoder = nn.Sequential(
    nn.Linear(latent_dim, 64), nn.Tanh(),
    nn.Linear(64, data_dim)
)
z = torch.randn(128, latent_dim)     # z ~ N(0, I)
mu_x = decoder(z)                    # E[x | z]
x = mu_x + 0.1 * torch.randn_like(mu_x)
```

В примере $p_\theta(x \mid z) = \mathcal{N}(f_\theta(z), 0.01I)$. Получать сэмплы просто; вычислять маргинальную плотность $p_\theta(x)$ — нет.

Апостериорное распределение и трудность вывода

После наблюдения x нас интересует скрытая причина z:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p(z)}{p_\theta(x)}.$$

- Числитель доступен для вычисления при заданном (x, z).
- Знаменатель $p_\theta(x) = \int p_\theta(x \mid z)p(z)\,dz$ трудновычислим.
- Следовательно, точный posterior обычно недоступен.

Идея вариационного вывода

Вводим вычислимое семейство $q_\phi(z \mid x)$ и приближаем $p_\theta(z \mid x)$.

Amortized inference: encoder как распределение

$$q_\phi(z \mid x) = \mathcal{N}\big(\mu_\phi(x), \mathrm{diag}(\sigma^2_\phi(x))\big).$$

Нейросеть выдаёт $\mu_\phi(x) \in \mathbb{R}^d$ и $\ell_\phi(x) = \log \sigma^2_\phi(x) \in \mathbb{R}^d$.

- **Encoder:** $x \mapsto (\mu_\phi(x), \ell_\phi(x))$.
- **Decoder:** $z \mapsto$ параметры $p_\theta(x \mid z)$.
- **Amortization:** общие параметры ϕ используются для всех x_i.

Важно: $q_\phi(z \mid x)$ — распределение, а не единственный детерминированный код.

Почему именно диагональная гауссиана?

1. $q_\phi(z \mid x)$ легко параметризуется и сэмплируется.
2. При $p(z) = \mathcal{N}(0, I)$ величина $D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z))$ аналитична.
3. Доступен pathwise gradient estimator (репараметризация).

$$D_{\mathrm{KL}}\left(q_\phi(z \mid x) \parallel p(z)\right) = -\frac{1}{2} \sum_{i=1}^{d} \left(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right)$$

Цена упрощения: диагональная гауссиана может плохо приближать многомодальный posterior.

PyTorch: параметры approximate posterior

```python
import torch
from torch import nn

class Encoder(nn.Module):
    def __init__(self, data_dim=784, latent_dim=16):
        super().__init__()
        self.net = nn.Sequential(nn.Linear(data_dim, 256), nn.ReLU())
        self.mu = nn.Linear(256, latent_dim)
        self.logvar = nn.Linear(256, latent_dim)

    def forward(self, x):
        h = self.net(x.flatten(1))
        return self.mu(h), self.logvar(h)
```

Размеры выходов: $\mu, \log \sigma^2 \in \mathbb{R}^{B \times d}$.

ELBO: вывод через неравенство Йенсена (I)

Для любого $q_\phi(z \mid x)$ с подходящим носителем:

$$\begin{aligned}\log p_\theta(x) &= \log \int q_\phi(z \mid x) \frac{p_\theta(x, z)}{q_\phi(z \mid x)}\, dz \\ &= \log \mathbb{E}_{q_\phi(z|x)} \left[\frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right].\end{aligned}$$

Так как log вогнут, неравенство Йенсена даёт

$$\log \mathbb{E}_q[W] \geq \mathbb{E}_q[\log W].$$

ELBO: вывод через неравенство Йенсена (II)

Следовательно,

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right]$$
$$= \mathbb{E}_q[\log p_\theta(x \mid z)] + \mathbb{E}_q\left[\log \frac{p(z)}{q_\phi(z \mid x)}\right].$$

Evidence Lower Bound

$$\boxed{\mathcal{L}_{\mathrm{ELBO}}(x;\theta,\phi) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x)\|p(z)).}$$

Второй вывод: точный разрыв между ELBO и likelihood

$$
\begin{aligned}
D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z \mid x)) &= \mathbb{E}_q \left[\log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)} \right] \\
&= \mathbb{E}_q[\log q_\phi(z \mid x) - \log p_\theta(x, z)] + \log p_\theta(x) \\
&= \log p_\theta(x) - \mathcal{L}_{\mathrm{ELBO}}(x; \theta, \phi).
\end{aligned}
$$

$$
\boxed{\log p_\theta(x) = \mathcal{L}_{\mathrm{ELBO}}(x; \theta, \phi) + D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z \mid x)).}
$$

Так как $D_{\mathrm{KL}} \geq 0$, ELBO действительно является нижней оценкой.

Когда оценка точна?

$$\log p_\theta(x) - \mathcal{L}_{\mathrm{ELBO}}(x;\theta,\phi) = D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z \mid x)).$$

- Равенство $\mathcal{L}_{\mathrm{ELBO}} = \log p_\theta(x)$ тогда и только тогда, когда $q_\phi(z \mid x) = p_\theta(z \mid x)$ почти всюду.
- Оптимизация по ϕ уменьшает variational gap для фиксированного θ.
- Оптимизация по θ изменяет саму генеративную модель.

Тонкость

Максимизация ELBO не тождественна точной MLE при ограниченном вариационном семействе.

Две составляющие ELBO

$$\mathcal{L}_{\mathrm{ELBO}} = \underbrace{\mathbb{E}_q[\log p_\theta(x \mid z)]}_{\text{качество объяснения } x} - \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z))}_{\text{штраф за отклонение от prior}} .$$

- Reconstruction term стимулирует передачу полезной информации через z.
- KL term ограничивает отличие posterior от prior.
- Оптимизируем $\max_{\theta,\phi} \frac{1}{N} \sum_i \mathcal{L}_{\mathrm{ELBO}}(x_i; \theta, \phi)$.

Вопрос аудитории: что произойдёт при слишком сильном KL-штрафе?

PyTorch: Monte Carlo оценка reconstruction term

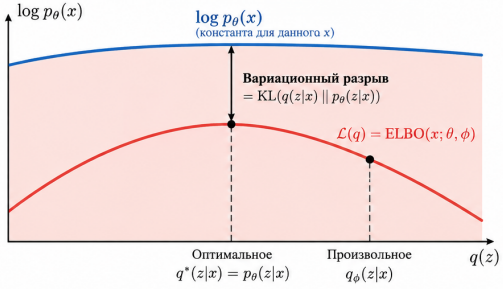
Для $z^{(k)} \sim q_\phi(z \mid x)$:

$$\mathbb{E}_q[\log p_\theta(x \mid z)] \approx \frac{1}{K} \sum_{k=1}^{K} \log p_\theta(x \mid z^{(k)}).$$

```
# x: [B, D]; mu_x: [K, B, D]; sigma_x > 0
import torch
from torch.distributions import Normal

sigma_x = 0.1
log_px_z = Normal(mu_x, sigma_x).log_prob(x[None, ...])
recon_mc = log_px_z.sum(dim=-1).mean(dim=0)  # [B]
```

Это оценка только reconstruction term; KL добавляется отдельно. Для градиентов по encoder потребуется репараметризация.



Ключевое разложение:

$$\log p_\theta(x) = \mathcal{L}(q) + \mathrm{KL}(q(z|x)||p_\theta(z|x))$$

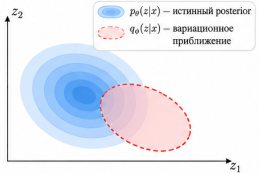
где

$$\mathcal{L}(q) = \underbrace{\mathbb{E}_{q(z|x)}[\log p_\theta(x|z)]}_{\text{качество реконструкции } x} - \underbrace{\mathrm{KL}(q(z|x)||p(z))}_{\text{штраф за отклонение от prior}}.$$

Следствия:

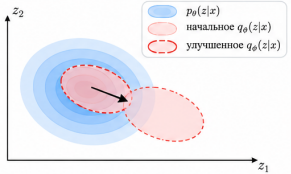
- ELBO — нижняя оценка, так как $\mathrm{KL} \geq 0$.
- Разрыв равен нулю тогда и только тогда, когда $q(z|x) = p_\theta(z|x)$ (почти всюду).
- Оптимизация по ϕ уменьшает разрыв, оптимизация по θ изменяет модель и саму цель.

1. Приближение posterior



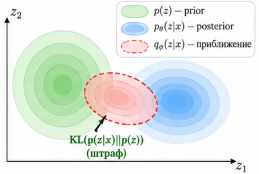
Так как $q_\phi(z|x) \neq p_\theta(z|x)$, возникает вариационный разрыв $\mathrm{KL}(q_\phi(z|x)||p_\theta(z|x) > 0$.

2. Улучшение $q_\phi(z|x)$ (уменьшение разрыва)

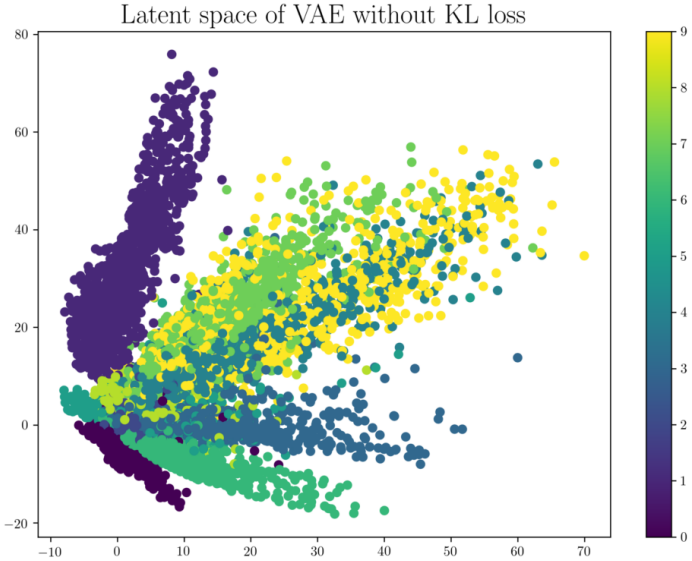


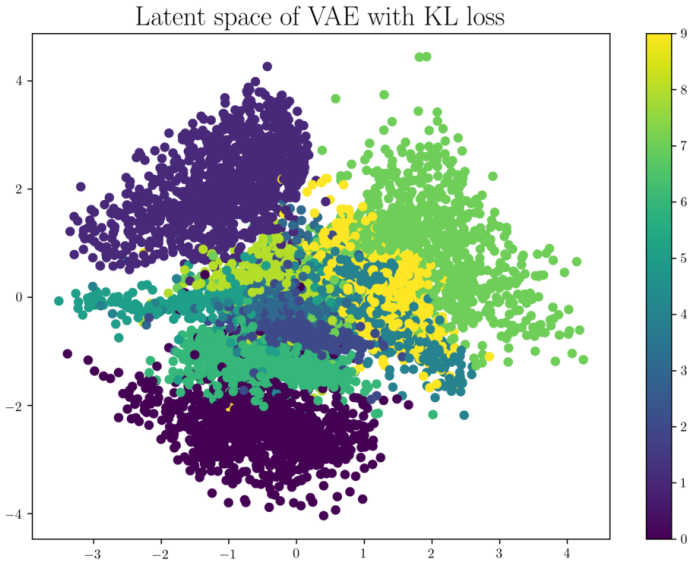
Оптимизация по ϕ делает $q_\phi(z|x)$ ближе к $p_\theta(z|x)$, уменьшая $\mathrm{KL}(q_\phi||p_\theta)$ и увеличивая ELBO.

3. Роль prior $p(z)$



Термин $\mathrm{KL}(q(z|x)||p(z))$ удерживает вариационный posterior рядом с prior, регуляризуя модель и предотвращая переобучение.


Latent space of VAE without KL loss
80
60
40
20
0
−20
−10
0
10
20
30
40
50
60
70
9
8
7
6
5
4
3
2
1
0


Latent space of VAE with KL loss
4
2
0
−2
−4
−3
−2
−1
0
1
2
3
4
9
8
7
6
5
4
3
2
1
0

Likelihood задаёт reconstruction loss

Bernoulli decoder для бинарных пикселей $x_j \in \{0, 1\}$:

$$p_\theta(x \mid z) = \prod_{j=1}^{D} \text{Bernoulli}(x_j; \pi_{\theta,j}(z)).$$

Тогда $-\log p_\theta(x \mid z)$ — binary cross-entropy. **Gaussian decoder** с фиксированной дисперсией:

$$p_\theta(x \mid z) = \mathcal{N}(f_\theta(z), \sigma_x^2 I), \quad -\log p_\theta(x \mid z) = \frac{\|x - f_\theta(z)\|_2^2}{2\sigma_x^2} + C.$$

Вывод: BCE и MSE возникают из вероятностных предположений, а не как произвольные эвристики.

KL для диагональных гауссиан

Если $q = \mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$ и $p = \mathcal{N}(0, I)$, то

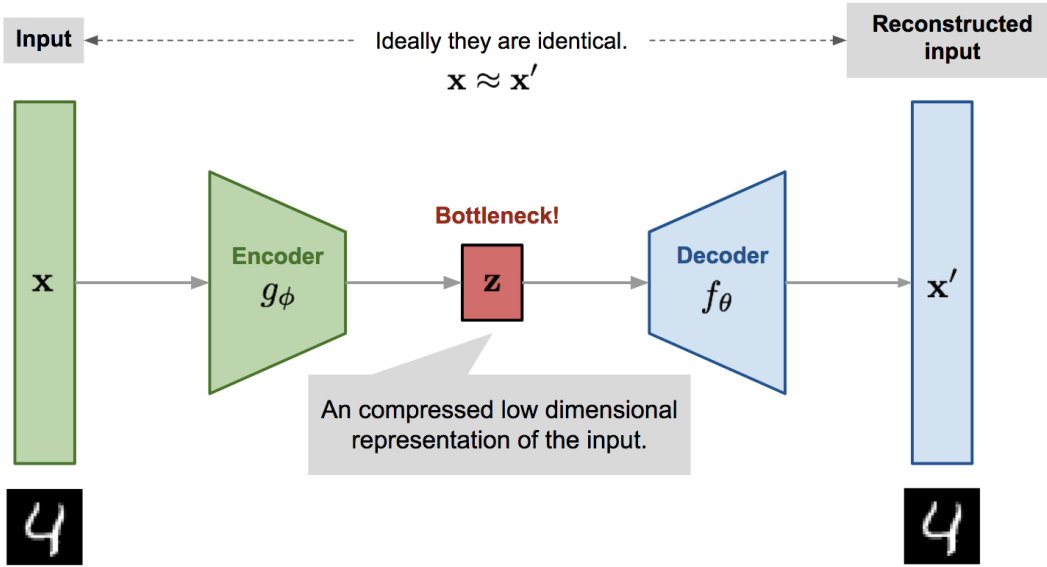
$$\boxed{D_{\mathrm{KL}}(q\|p) = \frac{1}{2}\sum_{j=1}^{d}\left(\mu_j^2 + \sigma_j^2 - 1 - \log\sigma_j^2\right).}$$

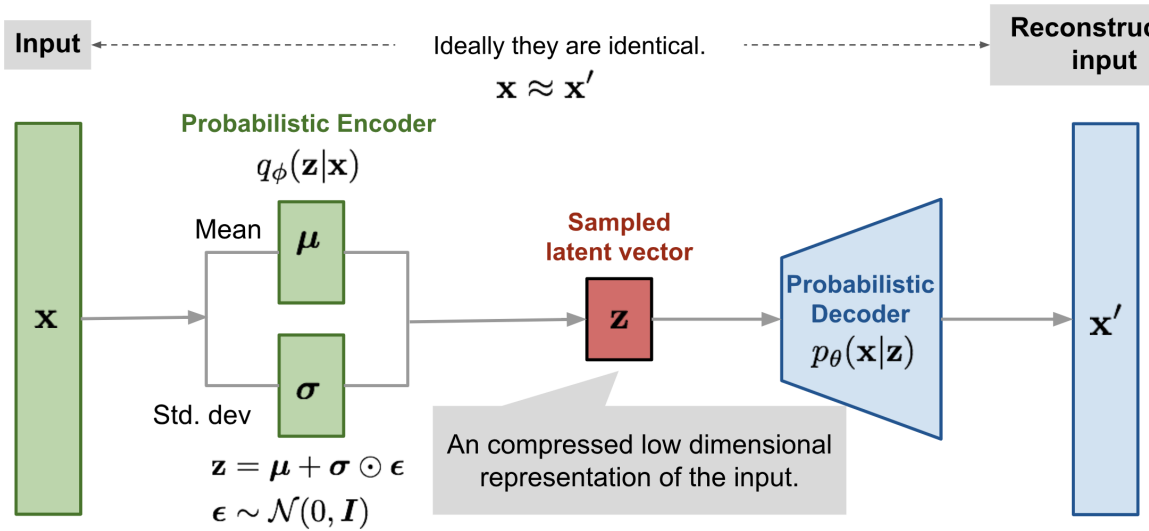
При $\ell_j = \log\sigma_j^2$:

$$D_{\mathrm{KL}}(q\|p) = \frac{1}{2}\sum_{j=1}^{d}\left(\mu_j^2 + e^{\ell_j} - 1 - \ell_j\right).$$

Отрицательный ELBO (минимизируем):

$$\mathcal{J}(\theta, \phi; x) = -\mathbb{E}_q[\log p_\theta(x \mid z)] + D_{\mathrm{KL}}(q\|p).$$


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstruc
input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
Mean
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}$
Std. dev
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
$\mathbf{x}$
Sampled
latent vector
$\mathbf{z}$
Probabilistic
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
An compressed low dimensional
representation of the input.

PyTorch: Bernoulli reconstruction и аналитический KL

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

# logits, x: [B, D]; mu, logvar: [B, d]
recon = F.binary_cross_entropy_with_logits(
    logits, x, reduction="none"
).sum(dim=1)
kl = -0.5 * (1 + logvar - mu.square() - logvar.exp()).sum(1)
loss = (recon + kl).mean()
```

Суммирование по признакам соответствует log-likelihood одного объекта; усреднение выполняется по батчу.

Проблема дифференцирования ожидания

Нужно оптимизировать

$$\nabla_\phi \mathbb{E}_{z \sim q_\phi(z|x)}[f_\theta(z)].$$

Наивная запись $z \sim q_\phi(z \mid x)$ не задаёт удобного дифференцируемого пути от z к ϕ.

Pathwise estimator

Перепишем $z = g_\phi(x, \varepsilon)$, где распределение ε не зависит от ϕ.

Тогда при стандартных условиях обмена производной и ожидания:

$$\nabla_\phi \mathbb{E}_\varepsilon[f_\theta(g_\phi(x, \varepsilon))] = \mathbb{E}_\varepsilon[\nabla_\phi f_\theta(g_\phi(x, \varepsilon))].$$

Гауссовская репараметризация

$$\boxed{\varepsilon \sim \mathcal{N}(0, I), \qquad z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon.}$$

При фиксированном ε:

$$\frac{\partial z_j}{\partial \mu_j} = 1, \qquad \frac{\partial z_j}{\partial \sigma_j} = \varepsilon_j.$$

- Случайность остаётся в ε.
- По параметрам encoder можно применять обычный backpropagation.
- Часто достаточно одного сэмпла ε на объект в SGD.

PyTorch: репараметризация и градиент

```python
import torch

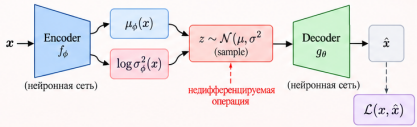
mu = torch.zeros(8, 2, requires_grad=True)
logvar = torch.zeros(8, 2, requires_grad=True)
std = torch.exp(0.5 * logvar)
eps = torch.randn_like(std)
z = mu + std * eps
objective = z.square().mean()
objective.backward()
print(mu.grad.shape, logvar.grad.shape)  # [8, 2], [8, 2]
```

ε не зависит от параметров encoder; граф вычислений проходит через μ и σ.

1. Прямой sampling (не работает для градиентов)

Сэмплируем напрямую: $z \sim \mathcal{N}(\mu, \sigma^2)$

Вычислительный граф (прямой проход)



Обратный проход (градиенты)



Формально

- Сэмплируем: $z \sim \mathcal{N}(\mu_\phi(x), \sigma^2_\phi(x))$.
- Целевая функция:

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

- Градиент по ϕ требует правила производной через распределение, что приводит к score-function градиенту (REINFORCE):

$$\nabla_\phi \mathbb{E}_q[f(z)] = \mathbb{E}_q[f(z)\nabla_\phi \log q_\phi(z \mid x)]$$

Плюсы / Минусы

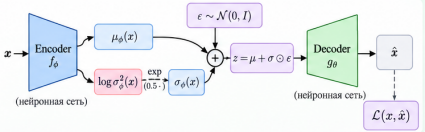
- ⊕ Подходит для любых распределений (не только гауссовских).
- ⊖ Высокая дисперсия градиентов.
- ⊖ Недифференцируемая операция sampling — нельзя использовать обычный backprop.

2. Репараметризация (дифференцируемый путь)

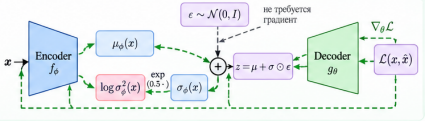
Представляем z как детерминированную функцию параметров и шума:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Вычислительный граф (прямой проход)



Обратный проход (градиенты)



Формально

- Выборка через репараметризацию:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

- Тогда

$$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,I)}[\log p_\theta(x \mid \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon)]$$

- Градиент по ϕ вычисляется обычным backprop через детерминированный граф:

$$\nabla_\phi \mathbb{E}_\epsilon[f(\mu_\phi + \sigma_\phi \odot \epsilon)] \approx \frac{1}{K} \sum_{k=1}^{K} \nabla_\phi f(\mu_\phi + \sigma_\phi \odot \epsilon^{(k)})$$

Плюсы / Минусы

- ⊕ Низкая дисперсия градиентов (pathwise).
- ⊕ Работает с backprop.
- ⊖ Стандарт для VAE.
- ⊖ Требует, чтобы распределение было репараметризуемым (например, Gaussian, переключаемые версии для некоторых семейств).

Прямой проход (вычисление) | Обратный проход (градиент проходит) | Градиент не проходит | Не требуется градиент

Три режима использования обученного VAE

1. **Реконструкция:** $x \to q_\phi(z \mid x) \to p_\theta(x \mid z)$.
2. **Генерация:** $z \sim p(z)$, затем $x \sim p_\theta(x \mid z)$.
3. **Интерполяция:** $z_t = (1-t)z_0 + tz_1$, $t \in [0,1]$.

При визуализации часто показывают среднее $\mathbb{E}[x \mid z]$, а не случайную выборку из $p_\theta(x \mid z)$.

Posterior collapse

Симптом

$$q_\phi(z \mid x) \approx p(z), \qquad D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z)) \approx 0.$$

Decoder может почти не использовать z, особенно если способен хорошо моделировать x без него.

- Диагностика: KL по латентным координатам, активные измерения, качество условной генерации.
- Возможные меры: KL annealing, free bits, ограничение мощности decoder.

Оговорка: малый KL сам по себе не доказывает неудачное обучение; важен контекст задачи.

Откуда берётся размытость реконструкций?

При фиксированной дисперсии Gaussian decoder максимизация log-likelihood соответствует MSE.

$$f_\theta(z) \approx \mathbb{E}[x \mid z] \quad \text{(при достаточной выразительности модели)}.$$

Если $p(x \mid z)$ многомодально, условное среднее может выглядеть размытым.

- Это не универсальная теорема о любых VAE.
- Эффект зависит от likelihood, архитектуры и качества латентного представления.

PyTorch: один шаг обучения и KL annealing

```
# encoder(x) -> mu, logvar; decoder(z) -> logits
mu, logvar = encoder(x)
std = torch.exp(0.5 * logvar)
z = mu + std * torch.randn_like(std)
logits = decoder(z)
recon = F.binary_cross_entropy_with_logits(
    logits, x, reduction="none"
).sum(dim=1)
kl = -0.5 * (1 + logvar - mu.square() - logvar.exp()).sum(1)
beta = min(1.0, epoch / 20.0)
loss = (recon + beta * kl).mean()
optimizer.zero_grad(); loss.backward(); optimizer.step()
```

Здесь x имеет форму $[B, D]$ и значения в $[0, 1]$; β растёт от 0 до 1.

β-VAE: управление компромиссом

$$\mathcal{L}_\beta = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] - \beta D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z)).$$

- $\beta = 1$: стандартный ELBO.
- $\beta > 1$: более сильная регуляризация латентного представления.
- $\beta < 1$: меньший штраф за отклонение от prior.

Тонкость: при $\beta \neq 1$ это уже не стандартный ELBO исходной модели. Disentanglement не гарантирован.

Conditional VAE

Пусть y — наблюдаемое условие (например, метка класса).

$$p_\theta(x, z \mid y) = p_\theta(x \mid z, y)p(z \mid y), \qquad q_\phi(z \mid x, y).$$

Тогда

$$\boxed{\mathcal{L}_{\mathrm{CVAE}}(x, y) = \mathbb{E}_{q_\phi(z|x,y)}[\log p_\theta(x \mid z, y)] - D_{\mathrm{KL}}(q_\phi(z \mid x, y) \| p(z \mid y)).}$$

Частный выбор: $p(z \mid y) = \mathcal{N}(0, I)$, независимо от y.

VQ-VAE: дискретные латентные коды (обзор)

Вместо непрерывного z используется ближайший элемент кодовой книги $\{e_k\}_{k=1}^{K}$:

$$k^*(x) = \arg\min_k \|z_e(x) - e_k\|_2^2, \qquad z_q(x) = e_{k^*(x)}.$$

- Дискретные представления допускают отдельное обучение prior над индексами.
- Недифференцируемое квантование требует специальных приёмов обучения.
- VQ-VAE — родственная, но не тождественная гауссовскому VAE конструкция.

PyTorch: β-VAE и условный decoder

```
# beta-VAE: loss components already computed
beta = 4.0
loss_beta = (recon + beta * kl).mean()

# Conditional VAE: concatenate one-hot labels to z
import torch.nn.functional as F
y_onehot = F.one_hot(y, num_classes=10).float()
z_cond = torch.cat([z, y_onehot], dim=-1)
logits_cond = conditional_decoder(z_cond)
```

Для полного CVAE encoder также должен получать (x, y); пример иллюстрирует только условный decoder.

Четыре ключевых равенства

$$p_\theta(x) = \int p_\theta(x \mid z) p(z)\, dz,$$

$$\log p_\theta(x) = \mathcal{L}_{\mathrm{ELBO}}(x) + D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z \mid x)),$$

$$\mathcal{L}_{\mathrm{ELBO}}(x) = \mathbb{E}_q[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x) \| p(z)),$$

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I).$$

PyTorch: генерация после обучения

```
# decoder returns Bernoulli logits for D pixels
with torch.no_grad():
    z = torch.randn(64, latent_dim, device=device)
    logits = decoder(z)
    pixel_prob = torch.sigmoid(logits)
    samples = torch.bernoulli(pixel_prob)
```

Для визуализации часто используют `pixel_prob` вместо случайных бинарных `samples`.

От VAE к GAN и Normalizing Flows

Модель	Обучающий принцип	Плотность $p_\theta(x)$
VAE	нижняя оценка likelihood	обычно трудна
GAN	состязательное обучение	обычно недоступна
Flows	точное maximum likelihood	вычислима

Вопрос к следующей лекции: можно ли обучить генератор $x = G_\theta(z)$, не задавая явное likelihood и не вычисляя ELBO?