

# Лекция 14. Variational autoencoders. Diffusion models.

Глубинное обучение

---

Антон Кленицкий

# Recap

---

# Авторегрессионные модели

Упорядочиваем переменные и обрабатываем по очереди, генерируем следующую на основе предыдущих.

$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_1)P(x_2|x_1)\dots P(x_n|x_1, \dots, x_{n-1}) \\ &= \prod_{i=1}^N P(x_i|x_1, \dots, x_{i-1}) \end{aligned}$$

Обучаем модель, которая моделирует это распределение.

$$P(x_1) \rightarrow x_1$$

$$P(x_2|x_1) \rightarrow x_2$$

$$P(x_3|x_1, x_2) \rightarrow x_3$$

...

# PixelCNN

*Van Den Oord A. et al. (2016) Pixel recurrent neural networks.*

- Авторегрессионная модель для генерации изображений
- Предсказываем пиксели как дискретные значения (от 0 до 255) строка за строкой
- Masked convolution

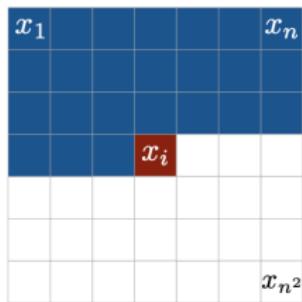


Image credit

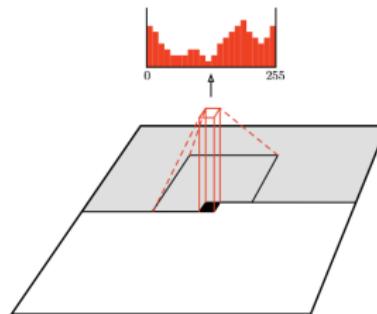


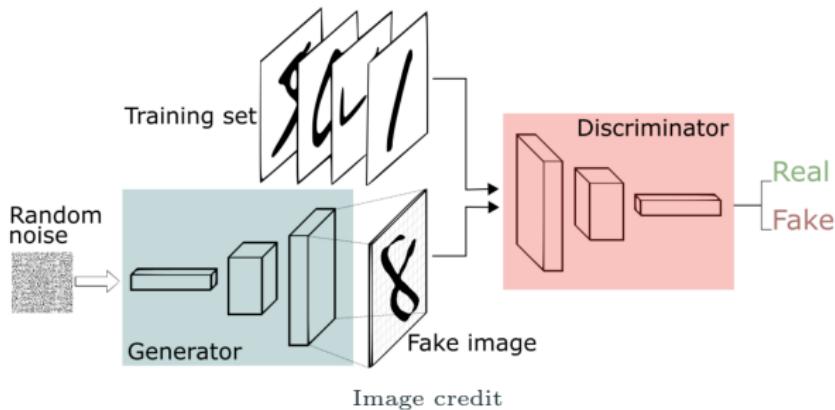
Image credit

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Очевидный минус - долгая генерация

# Generative Adversarial Networks

- Generator ( $G$ ) - сеть, пытающаяся обмануть дискриминатор и синтезировать из шума объекты, не отличимые от реальных
- Discriminator ( $D$ ) - сеть, отличающая настоящие объекты от синтезированных



Функция потерь обучается вместе с данными

# Generative Adversarial Networks

---

Генератор  $G(z) : z \rightarrow x$ ,  $z$  - шум,  $x$  - объекты

Дискриминатор  $D(x) : x \rightarrow [0, 1]$  (бинарная классификация)

Цель дискриминатора - максимизировать величину

$$E_x[\log D(x)] + E_z[\log(1 - D(G(z)))]$$

Цель генератора - минимизировать

$$E_z[\log(1 - D(G(z)))]$$

Получается minimax game

$$\min_G \max_D E_x[\log D(x)] + E_z[\log(1 - D(G(z)))]$$

# Проблемы обучения GAN

---

- Нестабильное обучение

Сходимость зависит от мелких деталей, высокая  
чувствительность к гиперпараметрам

- Vanishing gradients

Если дискриминатор гораздо умнее генератора, то сигмоида  
на выходе дискриминатора насыщается

- Mode collapse

Генератор выдаёт реалистично выглядящие объекты, но они  
покрывают только небольшую часть распределения данных  
 $p(x)$

# Conditional GAN

Условная генерация - дополнительные данные  $y$  являются условием для генератора и дискриминатора.

$$\min_G \max_D E_x[\log D(x|y)] + E_z[\log(1 - D(G(z|y)))]$$

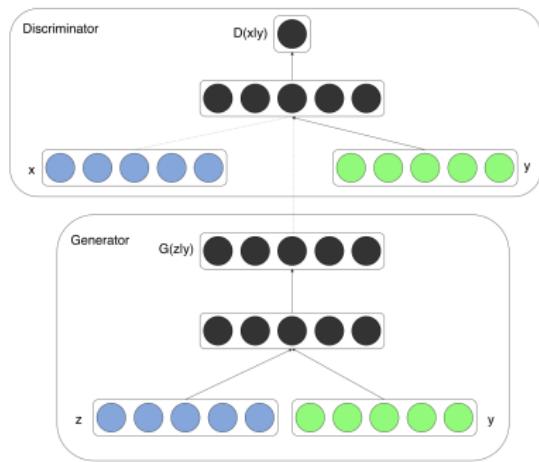
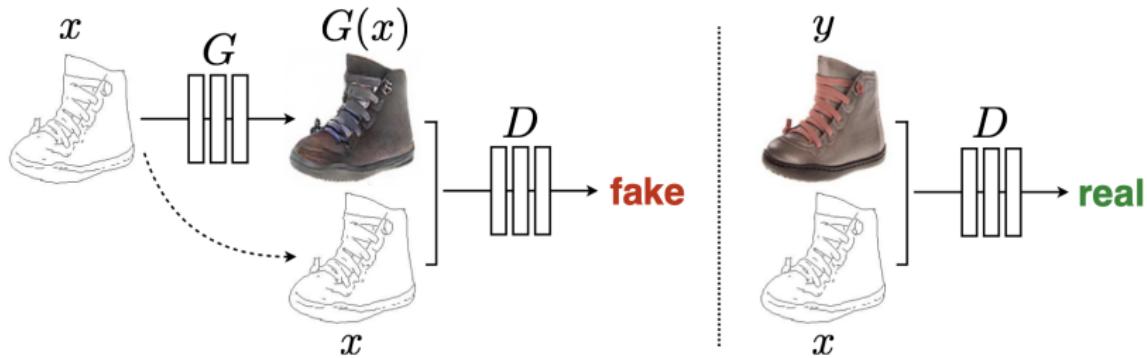


Image credit

$y$  - например, метки классов

*Isola P. et al. (2016) Image-to-image translation with conditional adversarial networks.*



Генератор  $G : \{x, z\} \rightarrow y$

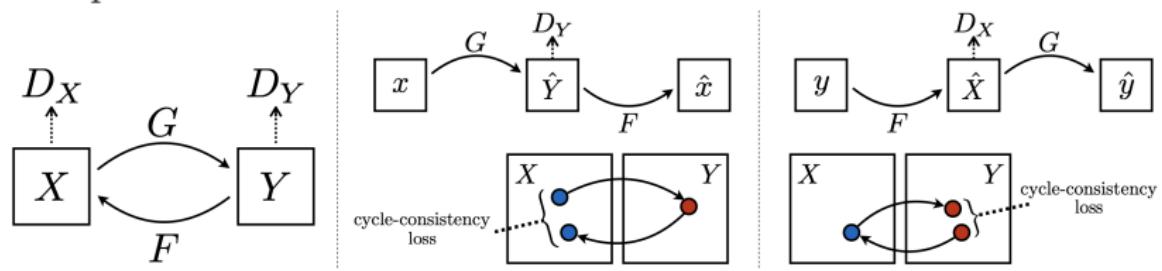
Дискриминатор  $D$  отличает настоящие пары  $\{x, y\}$  от сгенерированных  $\{x, G(x)\}$ .

Составная функция потерь: GAN loss (отвечает за мелкие детали) + Reconstruction loss (отвечает за общую структуру)

# CycleGAN

Zhu J. Y. et al. (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks.

В отличие от pix2pix не нужны размеченные пары изображений!



Два генератора -  $G : x \rightarrow y$  и  $F : y \rightarrow x$

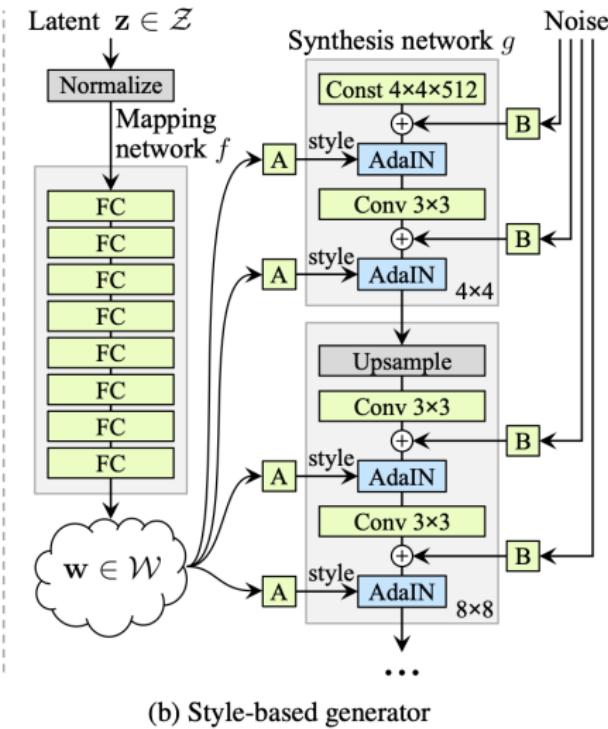
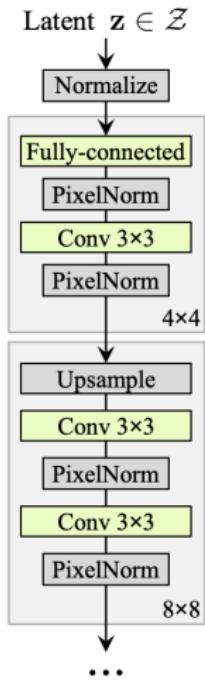
Два дискриминатора -  $D_X$  и  $D_Y$

Дополнительно требуем, чтобы  $F(G(x)) \approx x$  и  $G(F(y)) \approx y$

Функция потерь - GAN loss + Cycle consistency loss

# StyleGAN

Karras T. et al. (2019) A style-based generator architecture for generative adversarial networks.



# Variational Autoencoders

---

# Autoencoders

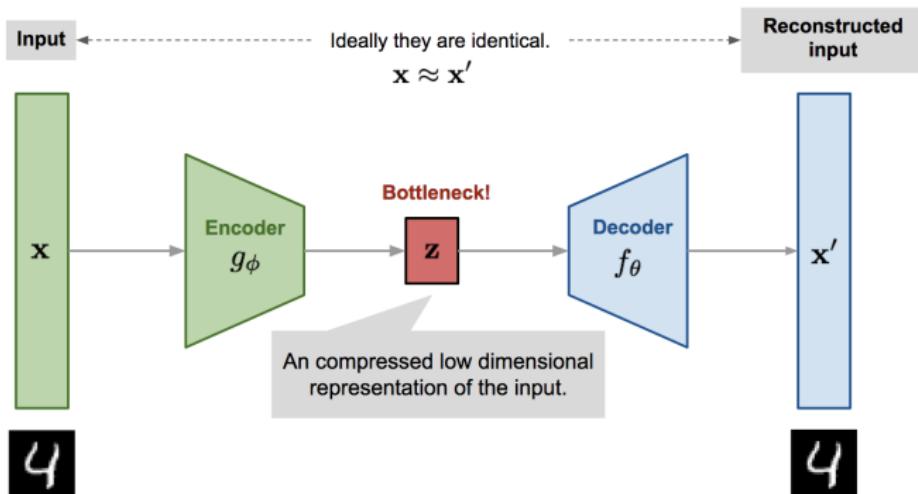


Image credit

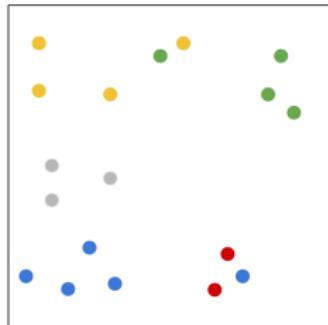
- Denoising autoencoder
- Sparse autoencoder

# Autoencoders

Почему нельзя использовать обычный автоэнкодер как генеративную модель?

- Скрытое пространство имеет сложную структуру
- Если сэмплировать из простого распределения, будем попадать в точки, не соответствующие реальным объектам

Messy Autoencoder Latent Space



Well Distributed VAE Latent Space

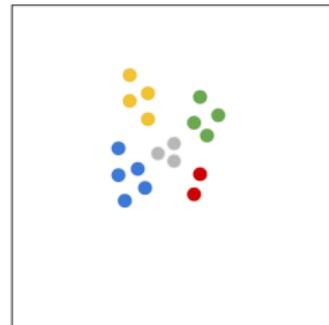
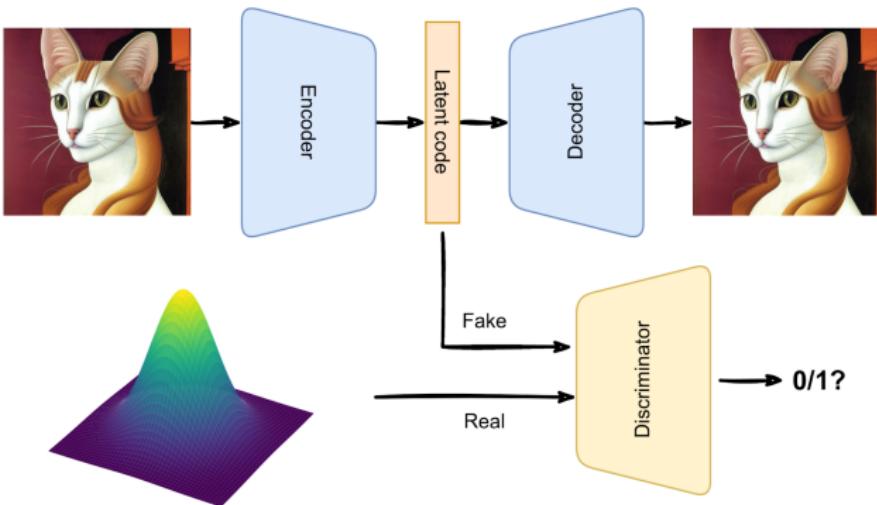


Image credit

# Adversarial autoencoder

*Makhzani A. et al. (2015) Adversarial autoencoders.*

Дискриминатор пытается отличить скрытые представления автоэнкодера от примеров из заданного стандартного распределения



# Variational Autoencoder

Kingma D. P., Welling M. (2013) Auto-encoding variational Bayes.

- На выходе энкодера распределение вероятностей в скрытом пространстве
- Накладываем ограничение, чтобы распределение в скрытом пространстве было похоже на заданное

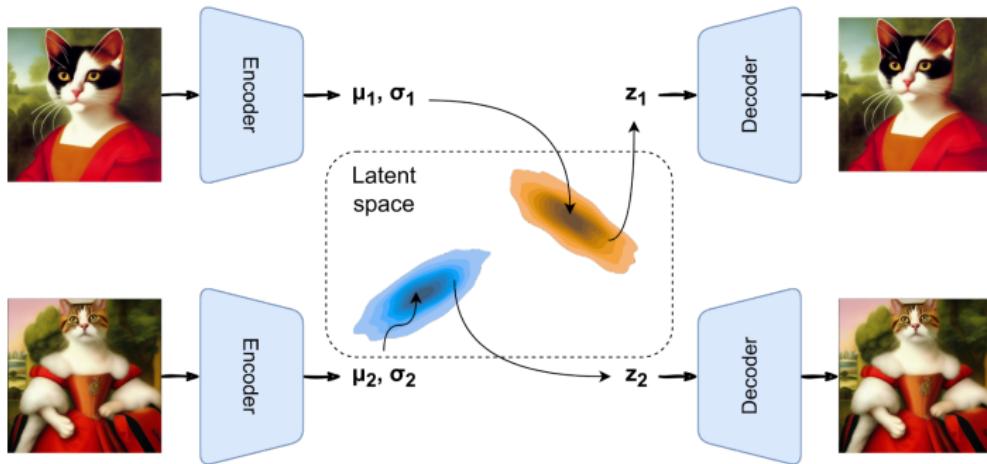


Image credit

# Variational Autoencoder

Latent variable model

$x$  - наблюдаемые данные

$z$  - скрытые переменные

$$p(x, z) = p(x|z)p(z)$$

Априорное распределение

$$p(z) = N(z|0, I)$$

Декодер

$$p(x|z) = N(x|\mu(z), \sigma^2 I)$$

$\mu(z)$  оценивается нейросетью

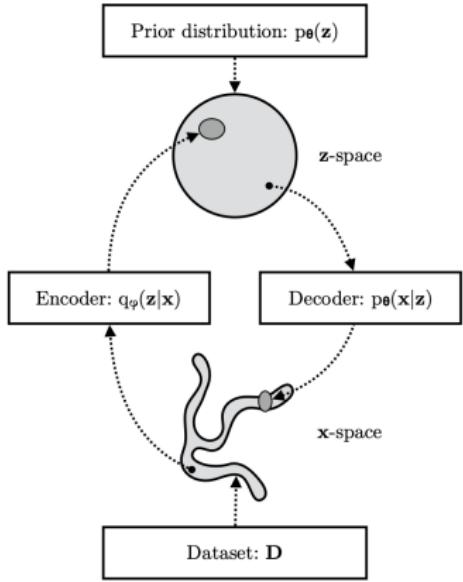


Image credit

# Variational Autoencoder

Хотим посчитать апостериорное распределение

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}$$

Но его нельзя посчитать точно, поэтому аппроксимируем его некоторым распределением  $q(z|x)$   
Это и есть наш энкодер:

$$q(z|x) = N(z|\mu(x), \Sigma(x))$$

$\mu(x), \Sigma(x)$  оцениваются нейросетью

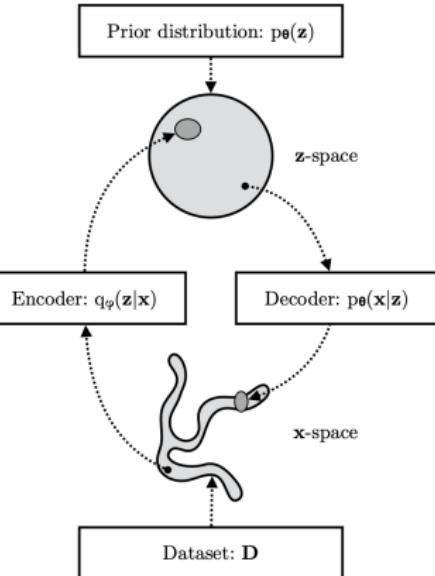


Image credit

# Variational Autoencoder

Функция потерь - reconstruction loss + regularization loss

$$L = -E_{q(z|x)}[\log p(x|z)] + KL(q(z|x)||p(z))$$

$$KL(q(z|x)||p(z)) = \int q(z|x) \log \frac{p(z)}{q(z|x)} dz$$

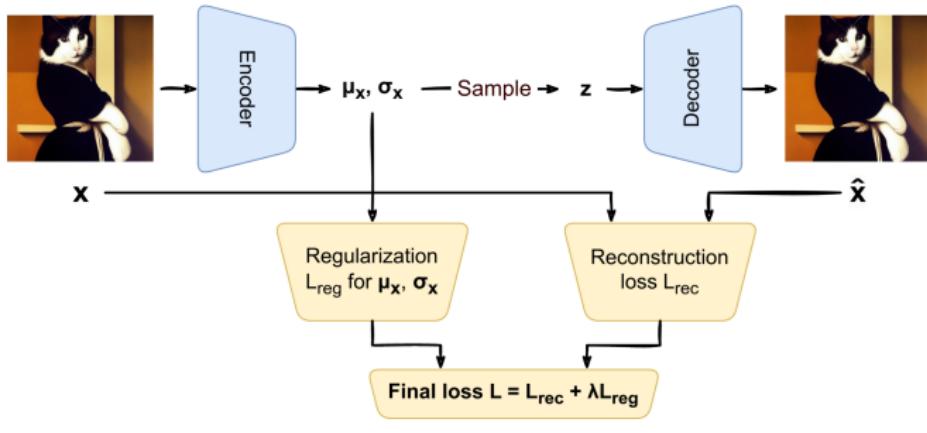
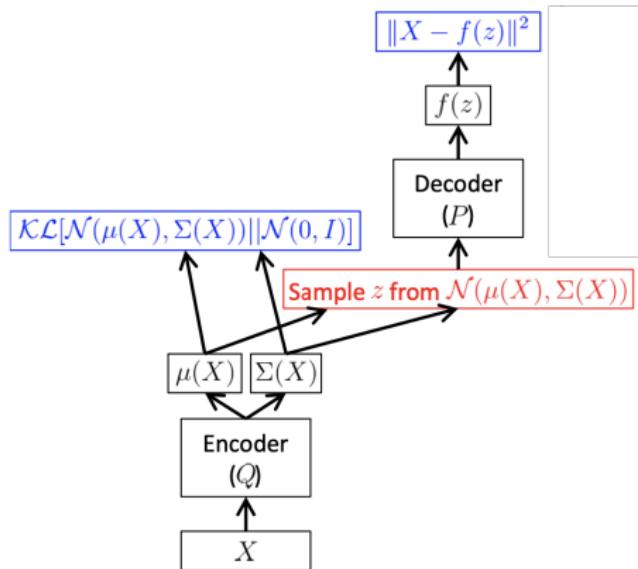


Image credit

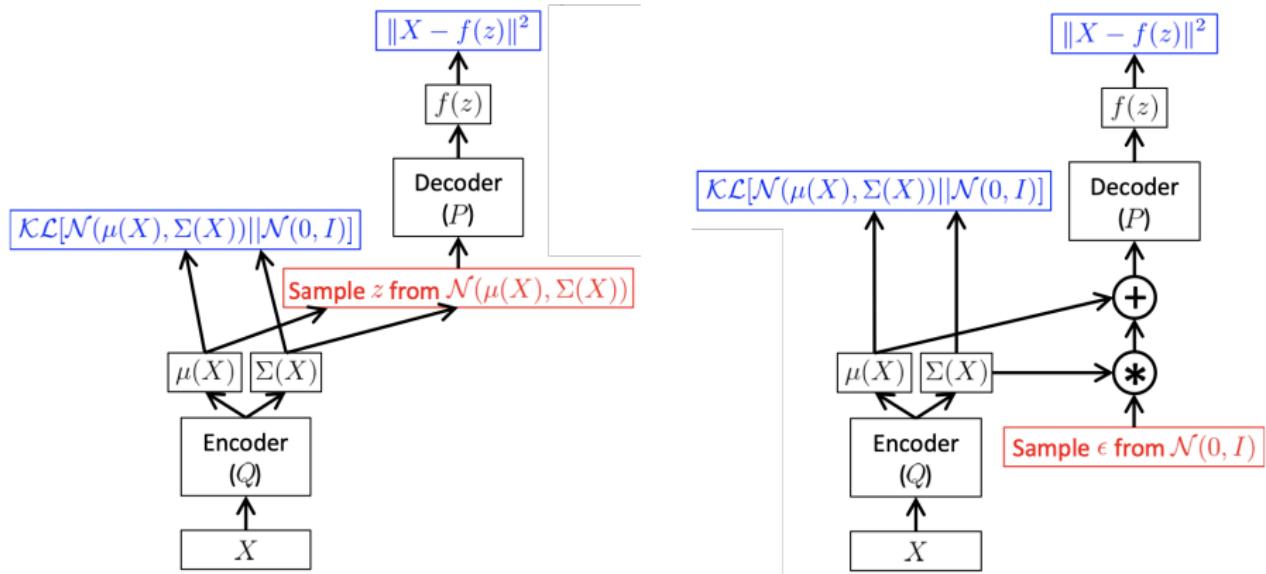
# Reparametrization trick

Как пропускать градиент через сэмплирование?



# Reparametrization trick

Как пропускать градиент через сэмплирование?



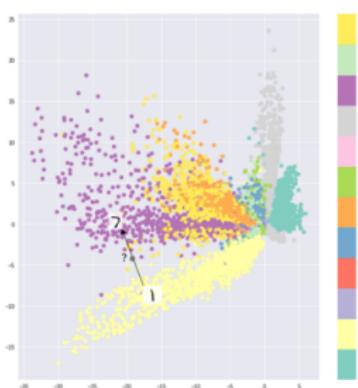
$$z \sim N(z|\mu, \sigma^2 I) \longrightarrow z = \mu + \sigma \odot \varepsilon, \quad \varepsilon \sim N(\varepsilon|0, I)$$

Image credit

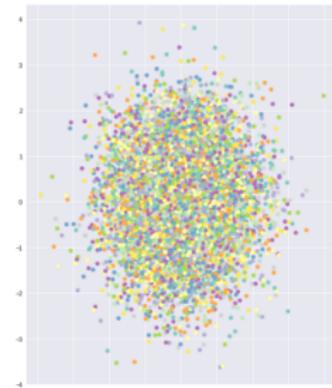
# Latent space

Скрытое пространство, которое выучивает модель

Only reconstruction loss



Only KL divergence



Combination

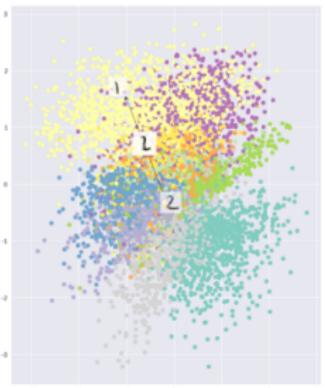


Image credit

# Resynthesis

Можно изменять свойства изображения

- Кодируем изображение с помощью энкодера
- Сдвигаем скрытое представление в нужном направлении
- Декодируем с помощью декодера

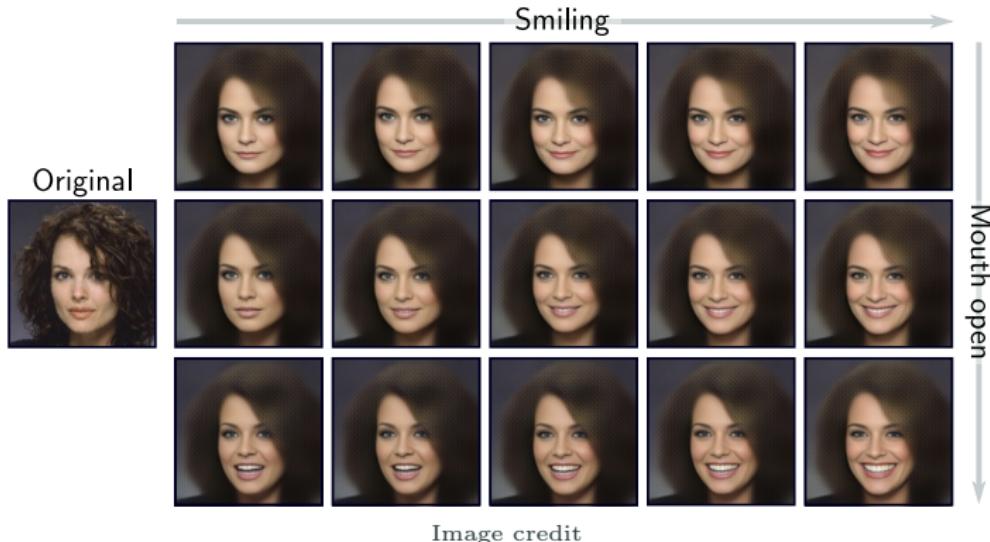


Image credit

## Discrete latent spaces

---

# Discrete latent space

Скрытое представление состоит из набора дискретных кодов

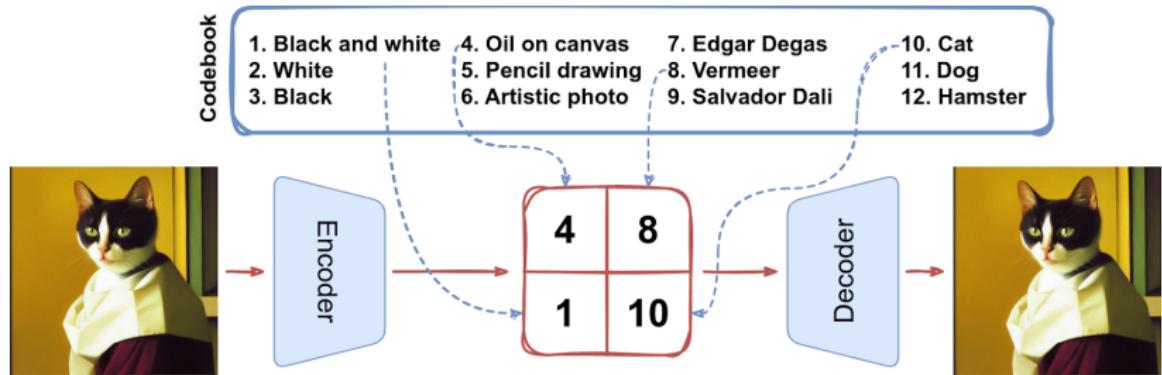
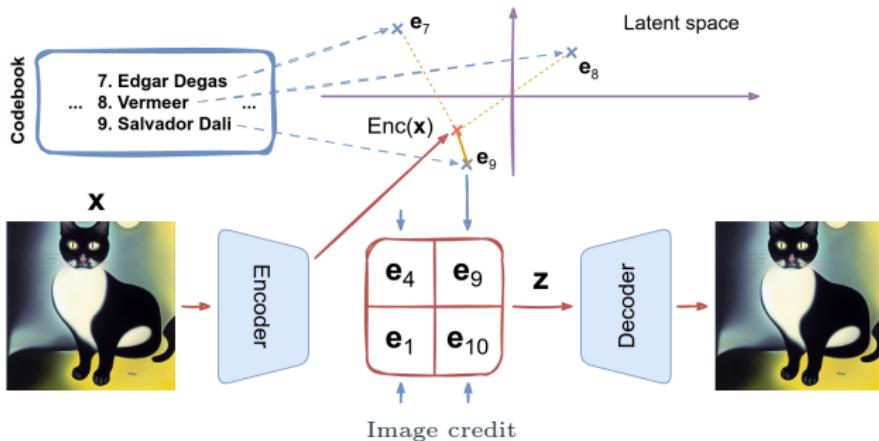


Image credit

# VQ-VAE

Van Den Oord A. et al. (2017) Neural discrete representation learning

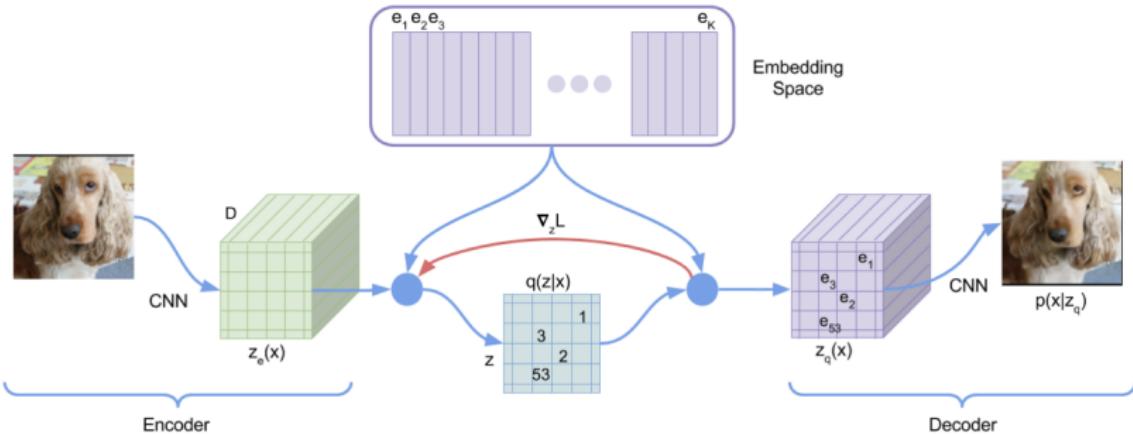
- Codebook содержит дискретные коды с соответствующими им векторами  $e_1, e_2, \dots, e_k$
- Энкодер выдает набор векторов  $z_e(x)$  в том же пространстве
- Каждый вектор  $z_e(x)$  заменяется на ближайший к нему вектор  $e$  из codebook для подачи в декодер



# VQ-VAE

$$q(z = k|x) = \begin{cases} 1, & k = \operatorname{argmin}_j \|z_e(x) - e_j\|_2 \\ 0, & \text{otherwise} \end{cases}$$

$$z_q(x) = e_k, \quad k = \operatorname{argmin}_j \|z_e(x) - e_j\|_2$$



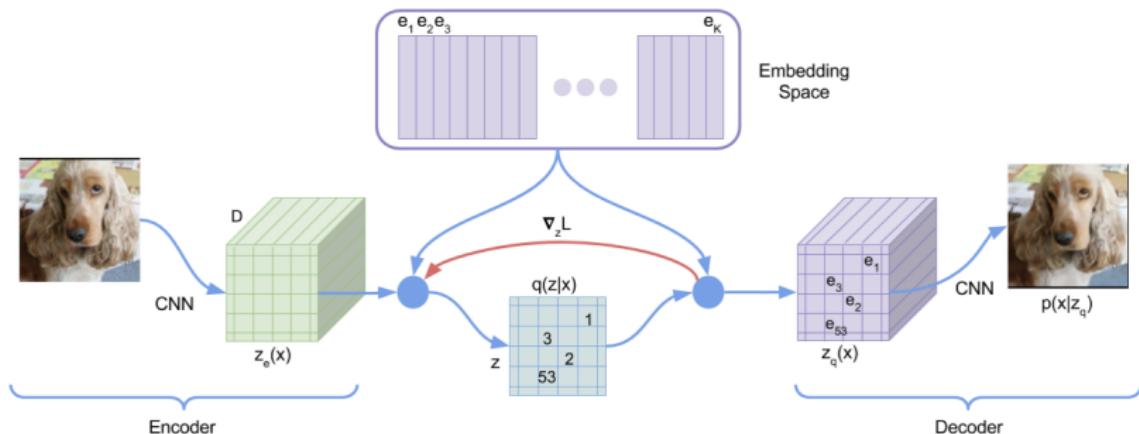
# VQ-VAE

Градиент  $z_q(x)$  из декодера копируется в  $z_e(x)$  в энкодер

Функция потерь

$$L = -\log p(x|z_q(x)) + \|sg[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - sg[e]\|_2^2$$

$sg$  - stop gradient



# dVAE

Ramesh A. et al. (2021) Zero-shot text-to-image generation.

discrete VAE

- Энкодер «по-честному» выдает распределение для дискретных кодов
- Сэмплируем из него для подачи на вход декодера

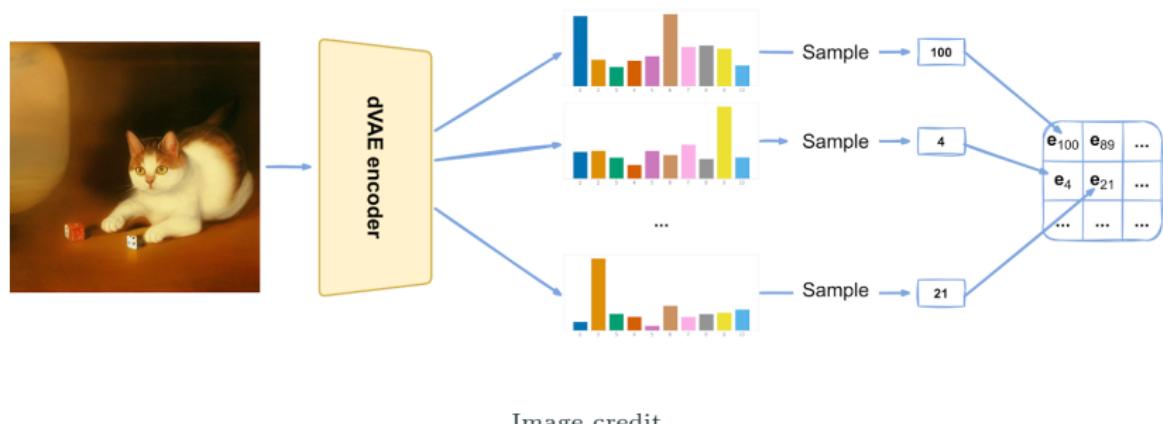


Image credit

## Gumbel-Max trick

---

Как и в обычном VAE, чтобы пропустить градиент через сэмплирование, нужен reparametrization trick

Gumbel distribution

$$p(g_i) = e^{-(g_i + e^{-g_i})}$$

С помощью него можно сэмплировать из категориального распределения с вероятностями  $\pi_i$ :

$$z = \operatorname{argmax}_i (g_i + \log \pi_i) \tag{1}$$

Но через argmax по-прежнему нельзя пропустить градиент

## Gumbel-Softmax trick

---

“Relaxation”: заменяем argmax на softmax!

Получаем веса

$$y_i = \text{softmax} \left( \frac{g_i + \log \pi_i}{\tau} \right)$$

и скрытый вектор равен взвешенной сумме кодов

$$z_q(x) = \sum_{j=1}^k y_j e_j$$

- При  $\tau \rightarrow 0$  получаем дискретное распределение с вероятностями  $\pi_i$
- Можно постепенно понижать температуру в процессе обучения

# Gumbel-Softmax trick

Latent Index	Prob
0	0.2
1	0.3
2	0.4
3	0.1

3. soft-sample codebook vectors from the Gumbel Softmax distribution

$y_0$	0.03	<del>✗</del>	[0.01, -2.3, 5.6, 0.04, -0.1, 8.92, 3.24, ...]
$y_1$	0.01	<del>✗</del>	[5.4, 0.65, 0.2, 4.6, 8.9, -2.43, 0.07, ...]
$y_2$	0.95	<del>✗</del>	[9.78, 0.67, -3.4, 0.2, -1.0, 7.2, 13.8, ...]
$y_3$	0.02	<del>✗</del>	[2.45, -8.9, 0.3, 2.04, -0.89, 19.1, 0.3, ...]

4. feed the resulting vectors to a decoder network to reconstruct the image

Latent Index	Prob
0	0.4
1	0.2
2	0.15
3	0.25

$y_0$	0.89	<del>✗</del>	[0.01, -2.3, 5.6, 0.04, -0.1, 8.92, 3.24, ...]
$y_1$	0.01	<del>✗</del>	[5.4, 0.65, 0.2, 4.6, 8.9, -2.43, 0.07, ...]
$y_2$	0.05	<del>✗</del>	[9.78, 0.67, -3.4, 0.2, -1.0, 7.2, 13.8, ...]
$y_3$	0.05	<del>✗</del>	[2.45, -8.9, 0.3, 2.04, -0.89, 19.1, 0.3, ...]

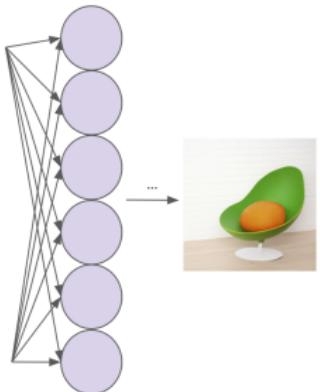
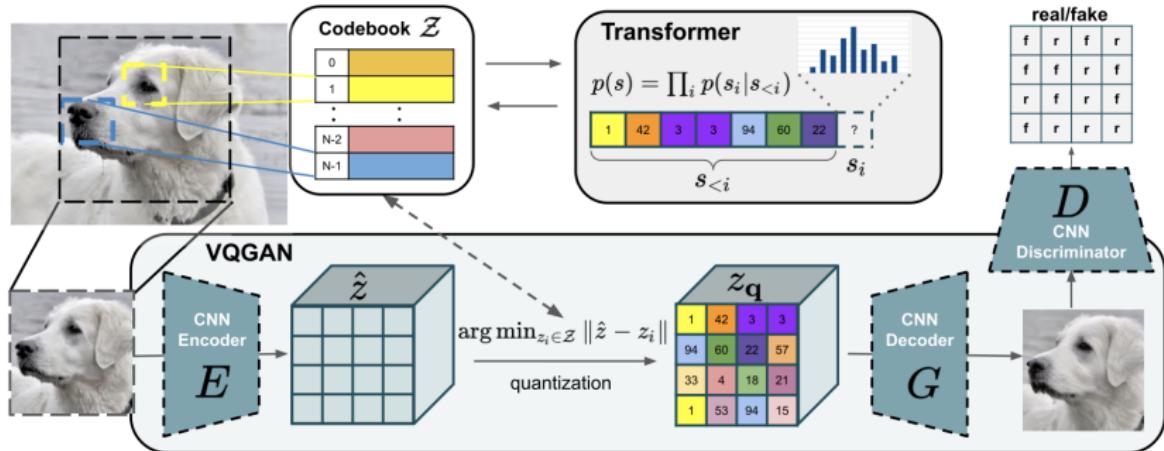


Image credit

# VQ-GAN

Esser P. et al. (2021) Taming transformers for high-resolution image synthesis.

- Perceptual loss - близость признаков, извлеченных предобученной сверточной сетью
- GAN loss с patch-based дискриминатором
- Генерация последовательности кодов с помощью трансформера



text-to-image

---

*Ramesh A. et al. (2021) Zero-shot text-to-image generation.*

Генерация изображений по текстовому описанию



- Обучаем dVAE сопоставлять дискретные коды изображениям
- Обучаем трансформер генерировать последовательность этих кодов, продолжая текстовое описание

# DALL-E

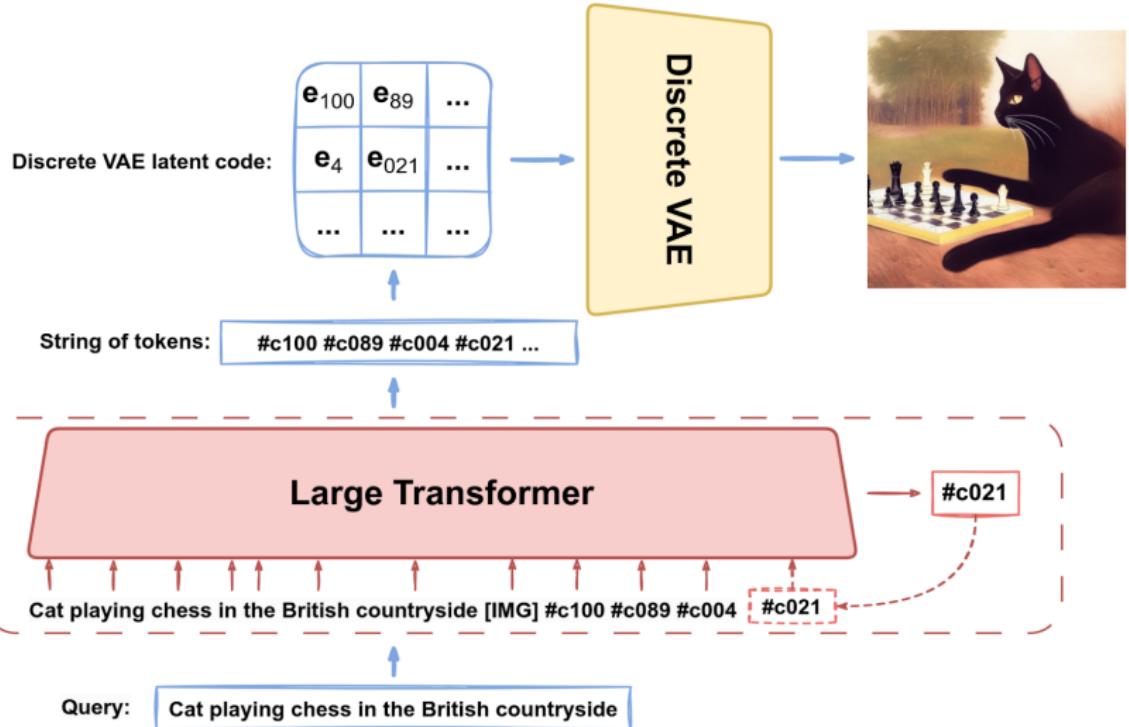


Image credit

- Обучение на 250M парах текст-изображение
- Ранжирование сгенерированных кандидатов с помощью CLIP

## dVAE

- сетка  $32 \times 32$  с дискретными кодами
- 8192 различных кодов

## Transformer Decoder

- 12B параметров
- 256 токенов текста + 1024 токенов изображения ( $32 \times 32$ )

# Diffusion models

---

# Denoising Diffusion Models

*Sohl-Dickstein J. et al. (2015) Deep unsupervised learning using nonequilibrium thermodynamics.*

*Ho J. et al. (2020) Denoising diffusion probabilistic models.*

- Шаг за шагом добавляем шум к изображению, пока оно не превратится в полностью случайный шум
- Модель учится убирать этот шум шаг за шагом, стартуя с полностью случайного шума

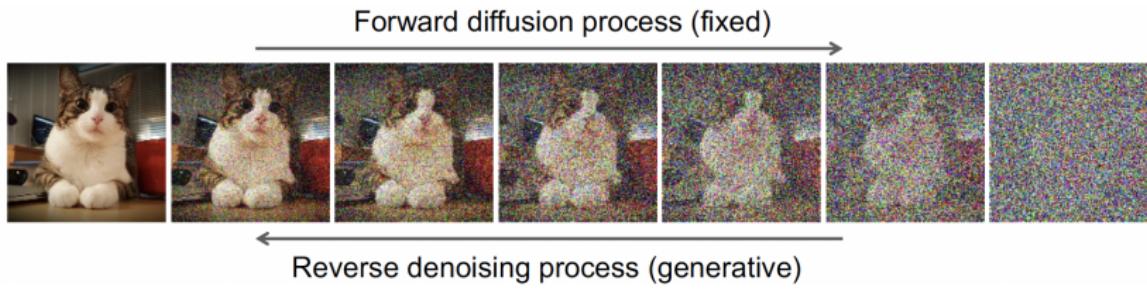
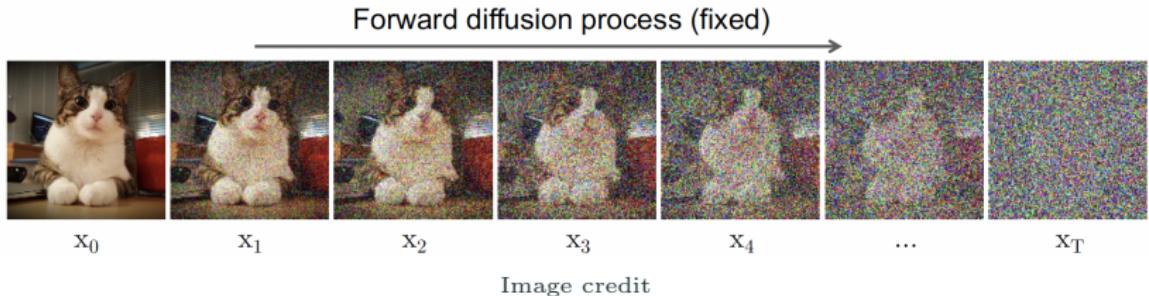


Image credit

# Forward diffusion process (encoder)



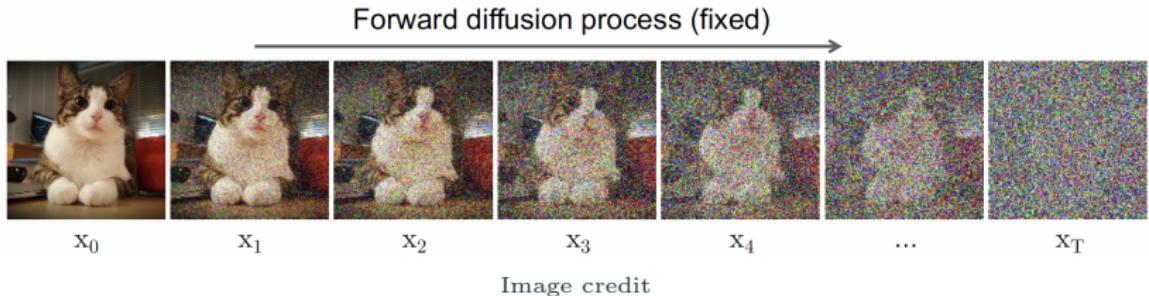
$x_0 \sim q(x_0)$  - реальные данные

$x_1, x_2, \dots, x_T$  - скрытые переменные

$T \sim 1000$

$$q(x_t|x_{t-1}) = N \left( x_t | \sqrt{1 - \beta_t} x_{t-1}, \beta_t I \right)$$

# Forward diffusion process (encoder)



$x_0 \sim q(x_0)$  - реальные данные

$x_1, x_2, \dots, x_T$  - скрытые переменные

$T \sim 1000$

$$q(x_t|x_{t-1}) = N \left( x_t | \sqrt{1 - \beta_t} x_{t-1}, \beta_t I \right)$$

$$x_t = \sqrt{1 - \beta_t} x_{t-1} + \beta_t \varepsilon, \quad \varepsilon \sim N(0, I)$$

## Forward diffusion process (encoder)

---

$$x_t = \sqrt{1 - \beta_t} x_{t-1} + \beta_t \varepsilon, \quad \varepsilon \sim N(0, I)$$

$\beta_t$  - заданные константы, noise schedule

- такое, чтобы  $q(x_T|x_0) \approx N(x_T|0, I)$
- $\beta_1 < \beta_2 < \dots < \beta_T$
- Например, линейное расписание с  $\beta_1 = 10^{-4}, \beta_T = 0.02$

## Forward diffusion process (encoder)

---

$$x_t = \sqrt{1 - \beta_t} x_{t-1} + \beta_t \varepsilon, \quad \varepsilon \sim N(0, I)$$

$\beta_t$  - заданные константы, noise schedule

- такое, чтобы  $q(x_T|x_0) \approx N(x_T|0, I)$
- $\beta_1 < \beta_2 < \dots < \beta_T$
- Например, линейное расписание с  $\beta_1 = 10^{-4}, \beta_T = 0.02$

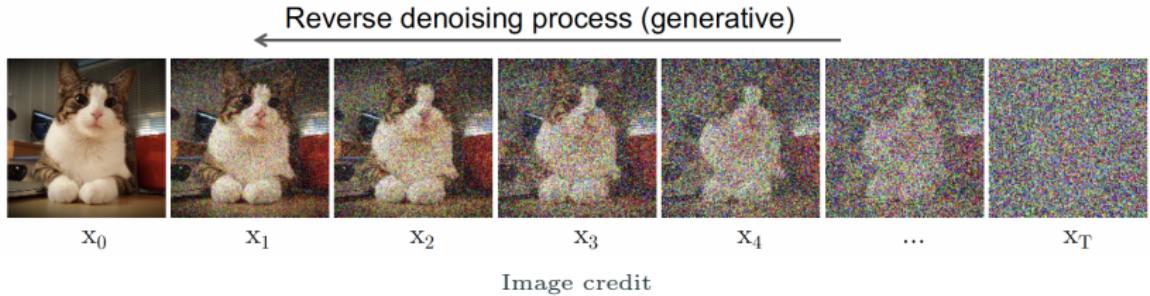
Не нужно считать  $x_t$  итеративно, можем сразу сэмплировать любой шаг (diffusion kernel):

$$q(x_t|x_0) = N(x_t|\sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)I)$$

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \varepsilon$$

$$\bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$$

# Reverse denoising process (decoder)



Хотим найти распределение

$$q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1})q(x_{t-1})}{q(x_t)}$$

Точно посчитать нельзя, аппроксимируем его в виде гауссова распределения и оцениваем параметры нейросетью:

$$p_\theta(x_{t-1}|x_t) = N(x_{t-1}|\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

$$p(x_T) = N(x_T|0, I)$$

## Loss function

---

$$L = -E_q \log p_\theta(x_0|x_1) + \sum_{t=2}^T KL(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t))$$

Reconstruction loss + по одному слагаемому с KL-divergence  
для каждого шага

-> можем брать случайный шаг и считать градиенты только  
для него

$$q(x_{t-1}|x_t, x_0) = N(x_{t-1}|\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I),$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t, \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

## Reparameterization

---

Если положить  $\Sigma_\theta(x_t, t) = \sigma_t^2 I$  ( $\sigma_t^2 = \beta_t$ ), то

$$L_t = KL(q(x_{t-1}|x_t, x_0)||p_\theta(x_{t-1}|x_t)) \sim \|\tilde{\mu}_t(x_t, x_0) - \mu_\theta(x_t, t)\|^2$$

Но  $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\varepsilon$ , поэтому

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{1 - \beta_t}} \left( x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon \right)$$

Так как  $x_t$  известно, вместо среднего  $\mu_\theta(x_t, t)$  модель может предсказывать шум  $\varepsilon_\theta(x_t, t)$ , который надо добавить к  $x_0$ , чтобы получить  $x_t$ .

В итоге

$$L_t \sim \|\varepsilon - \varepsilon_\theta(x_t, t)\|^2$$

С такой функцией потерь результаты получаются лучше.

# Denoising model

- U-net, так как одинаковая размерность на входе и на выходе
- Добавляем информацию о текущем шаге, например, в виде синусоидальных positional embeddings

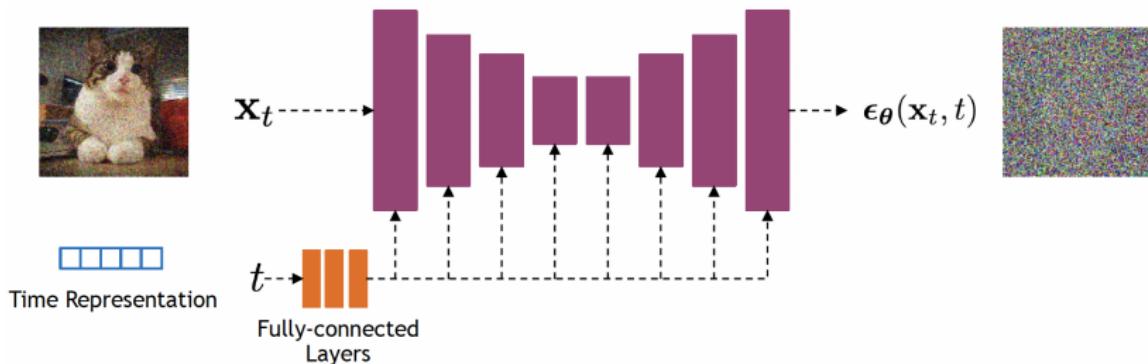


Image credit

# Diffusion summary

---

**Algorithm 1** Training

```
1: repeat
2:    $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 
3:    $t \sim \text{Uniform}(\{1, \dots, T\})$ 
4:    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
5:   Take gradient descent step on
       $\nabla_{\theta} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\theta}(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2$ 
6: until converged
```

---

**Algorithm 2** Sampling

```
1:  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t = T, \dots, 1$  do
3:    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else  $\mathbf{z} = \mathbf{0}$ 
4:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
5: end for
6: return  $\mathbf{x}_0$ 
```

---

Data

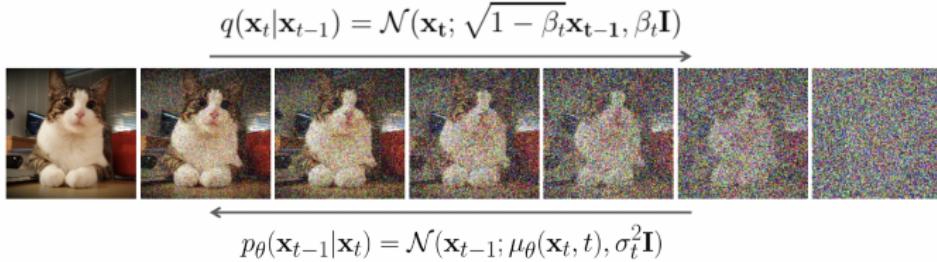
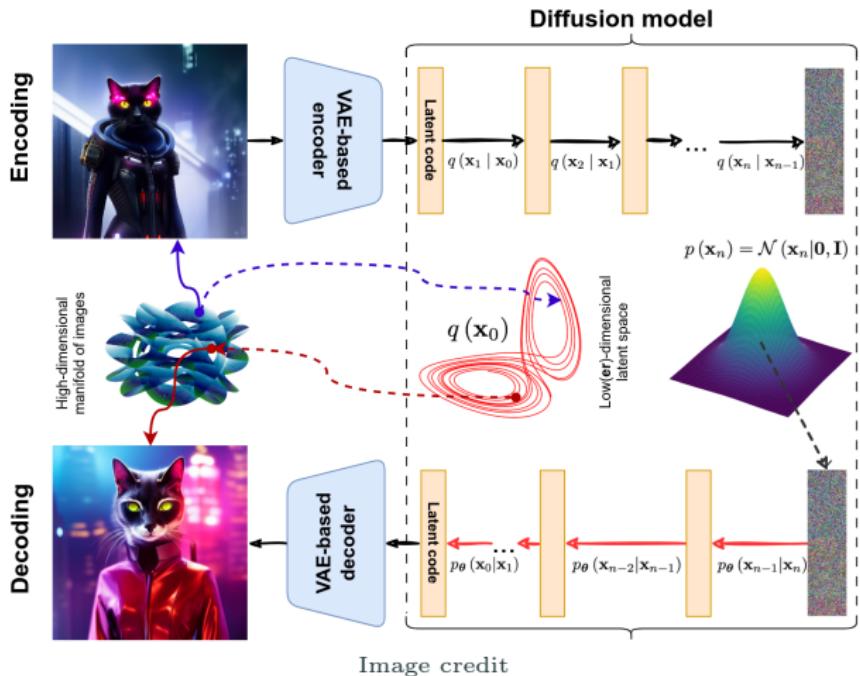


Image credit

# Stable Diffusion

Rombach R. et al. (2022) High-resolution image synthesis with latent diffusion models.



# Stable Diffusion

Rombach R. et al. (2022) High-resolution image synthesis with latent diffusion models.

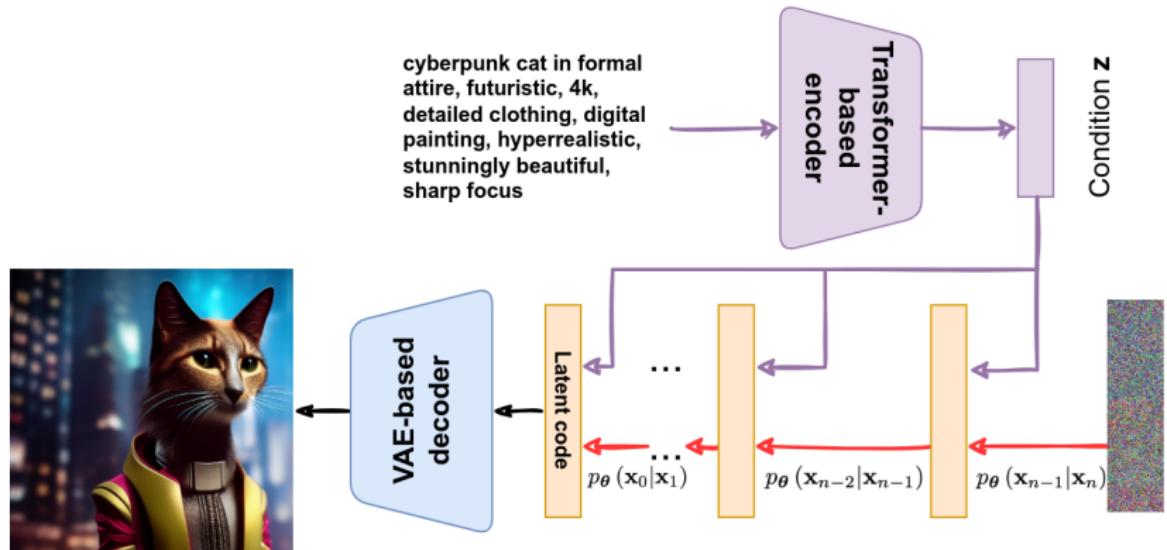
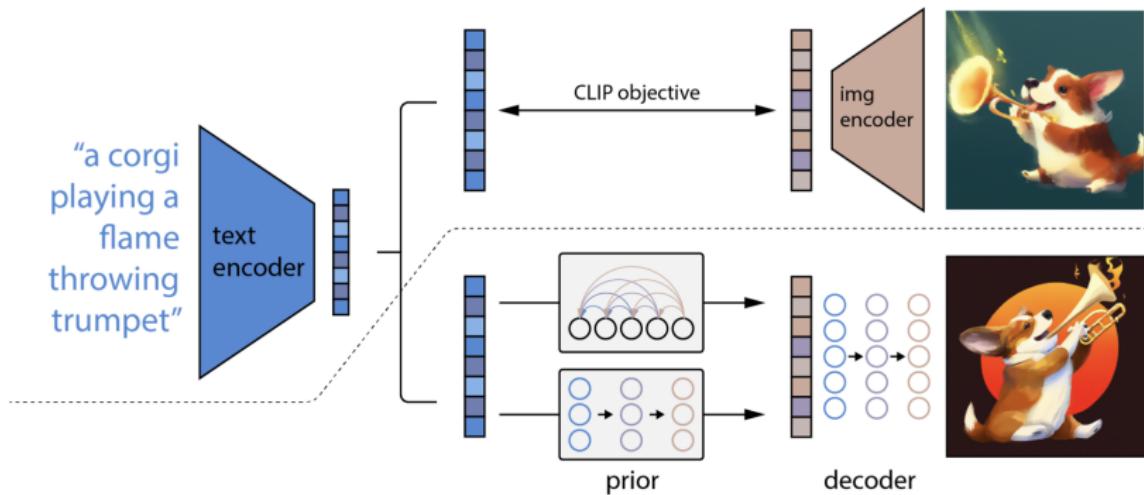


Image credit

# DALL-E 2

Ramesh A. et al. (2022) Hierarchical text-conditional image generation with clip latents.



## На этом все

- Доделываем домашние задания
- Проходим опрос по курсу