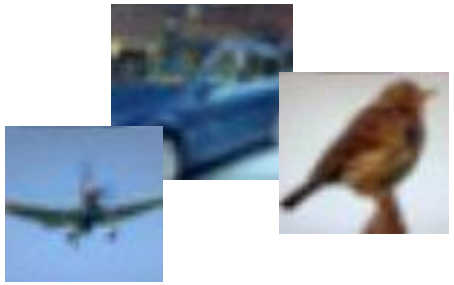
Генеративные модели

Джаин Никита, 172

Генеративные модели



Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Задача: научиться строить распределение $p_{model}(x)$ максимально похожее на $p_{data}(x)$.

Генеративные модели

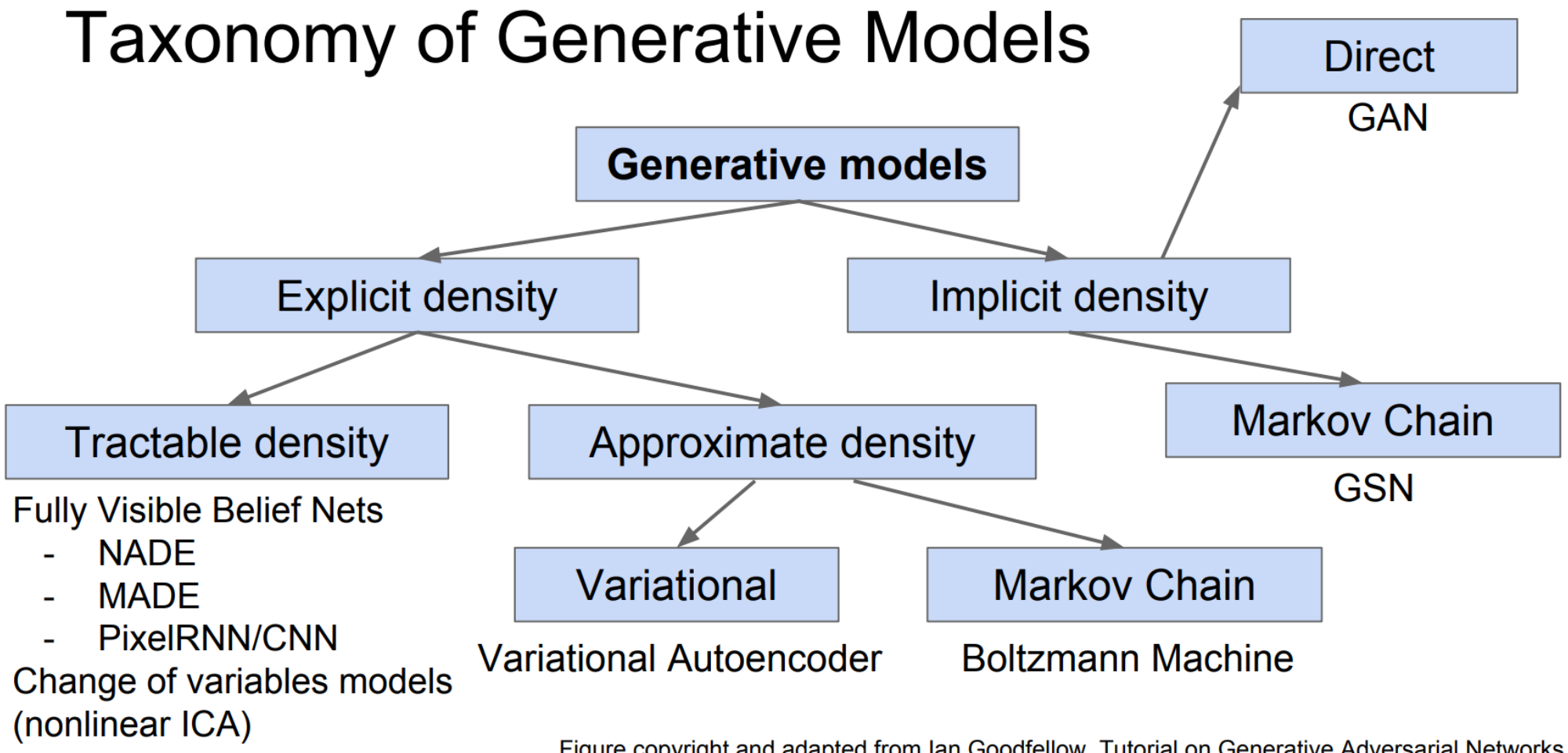


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN и PixelCNN

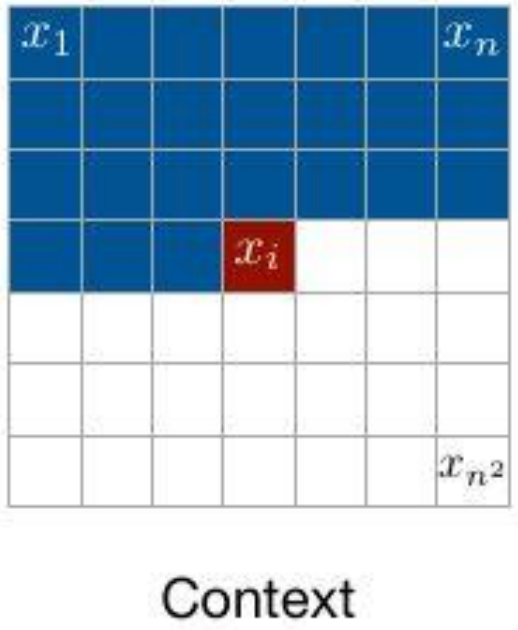
PixelRNN

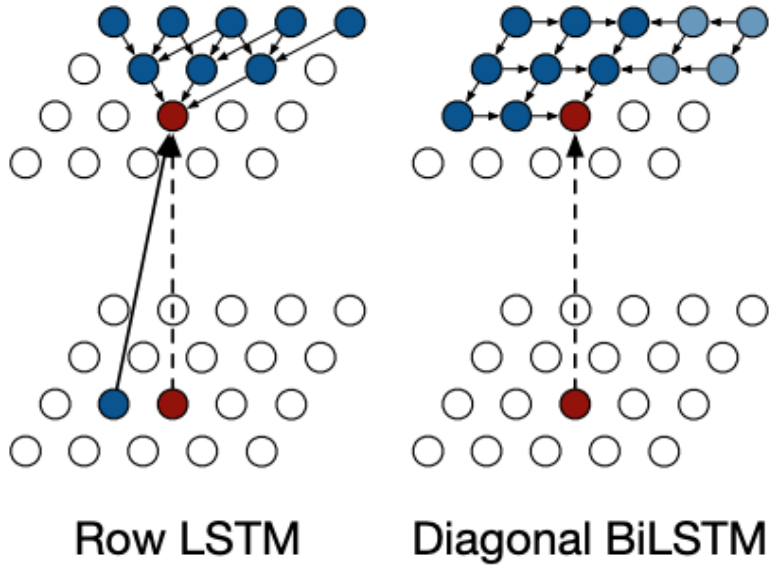
Строим распределение данных изображений, максимизируя функцию правдоподобия по пикселям:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i | x_1, ..., x_{i-1})$$

Чтобы посчитать вероятность каждого пикселя при условии контекста (посчитанных пикселей), используем RNN или LSTM.

Для RGB: $p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$ (2)





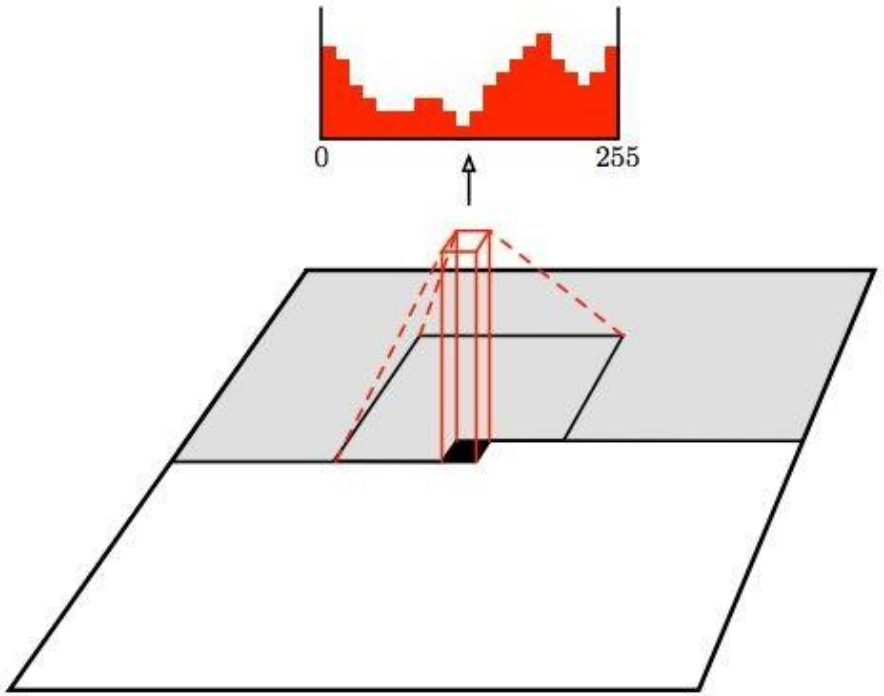
Проблема: последовательная генерация имеет большую вычислительную сложность. (очень медленная генерация!)

PixelCNN

Другой способ посчитать условную вероятность каждого пиксела это использовать CNN.

Для этого будем использовать маски, которые обнуляют все ещё не предсказанные пиксели.

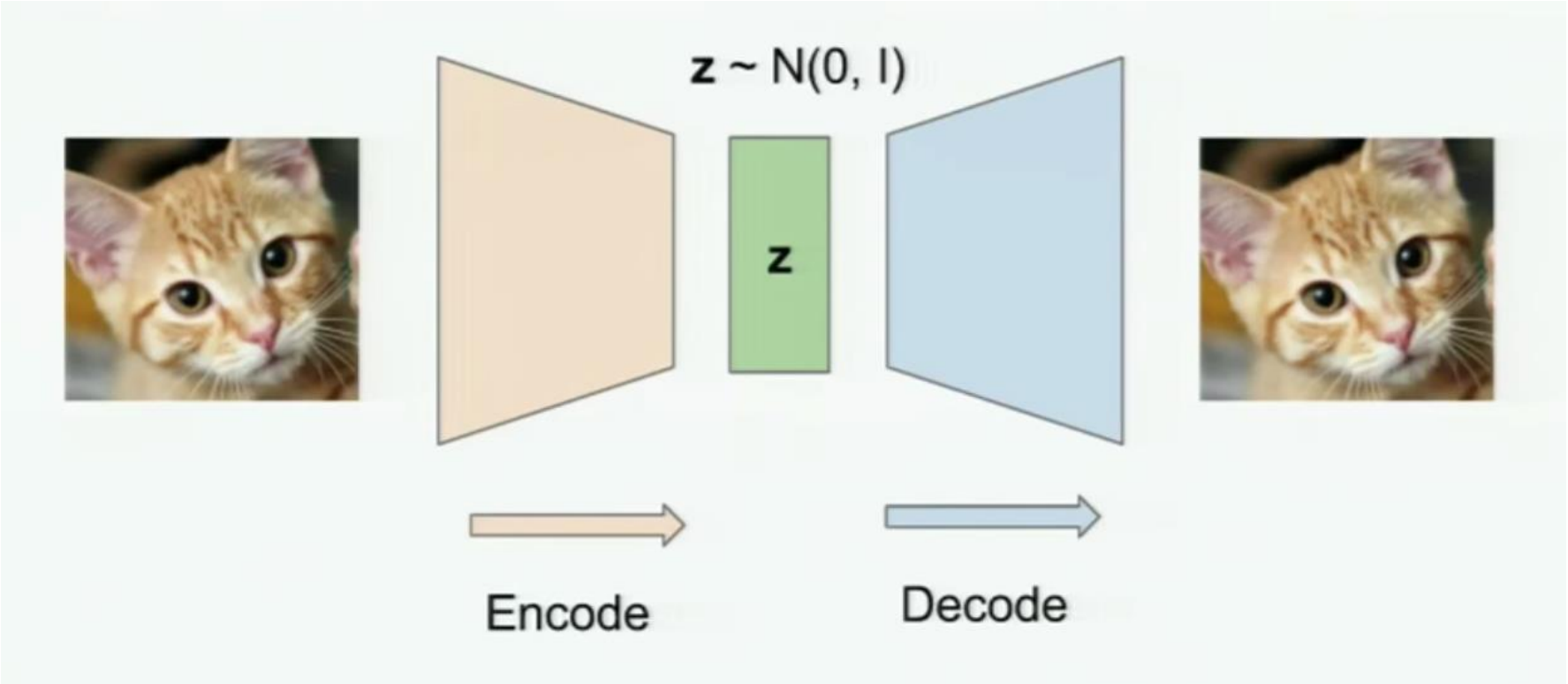
1	1	1	1	1
1	1	1	1	1
1	1	0	0	0
0	0	0	0	0
0	0	0	0	0



Variational Autoencoders

VAE

Идея: мы хотим максимизировать правдоподобие $p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$



Введём ограничение на **z**, т.что распределение **z** нормальное (~ N(0, I)).

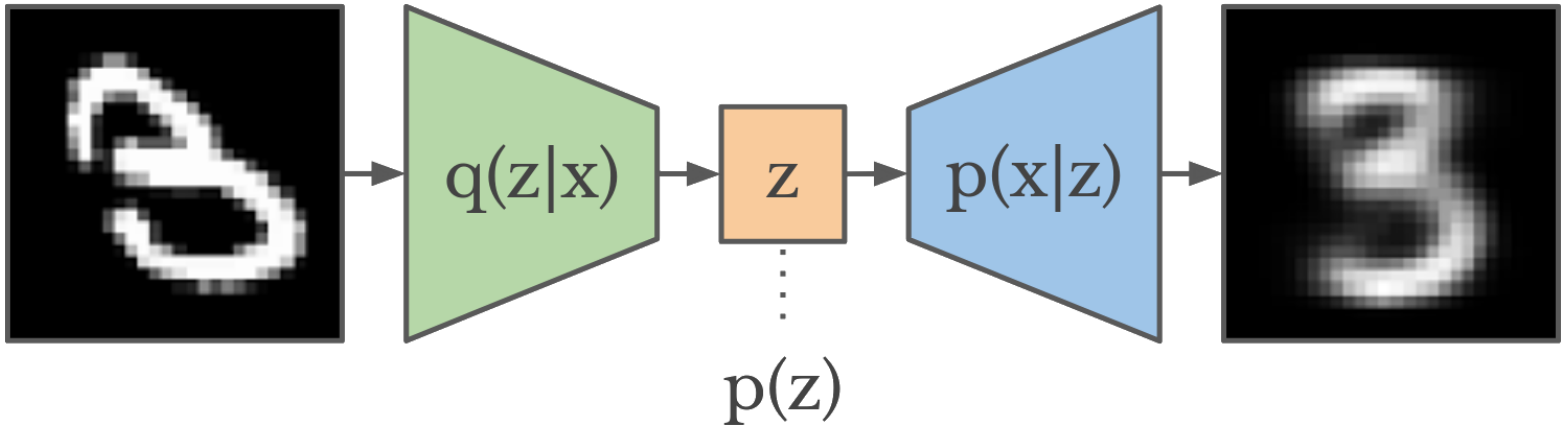
VAE

Мы хотим найти $P(z)$ и $P(x|z)$.

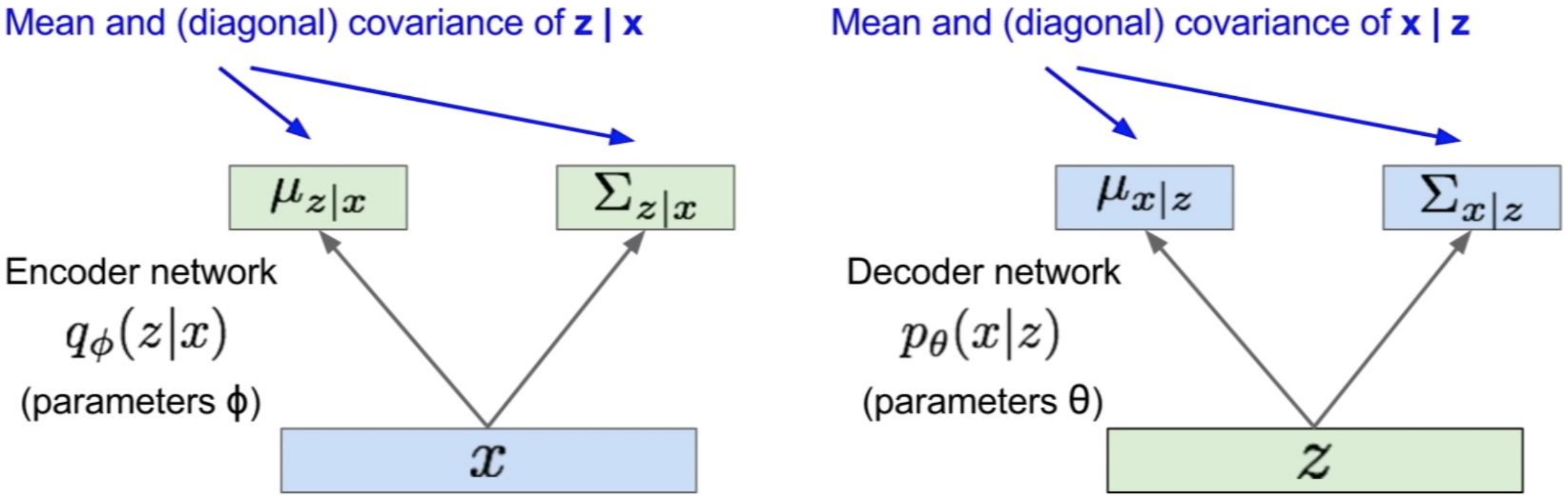
Идея : использовать постериорное распределение $P(z|x)$, из которого мы будем извлекать сэмплы в z. А с помощью z мы сможем найти $P(x|z)$.

Проблема: мы не можем напрямую посчитать $P(z|x)$ (т.к. зависит от $P(x)$).

Решение: оценить $P(z|x)$ с помощью $Q(z|x)$, которое будет приближать настоящее апостериорное распределение.



VAE


Mean and (diagonal) covariance of z | x
$\mu_{z|x}$
$\Sigma_{z|x}$
Encoder network
$q_\phi(z|x)$
(parameters ϕ)
x
Mean and (diagonal) covariance of x | z
$\mu_{x|z}$
$\Sigma_{x|z}$
Decoder network
$p_\theta(x|z)$
(parameters θ)
z

VAE

Распишем логарифм правдоподобия:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

≥ 0

Decoder reconstruction loss

regularization

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

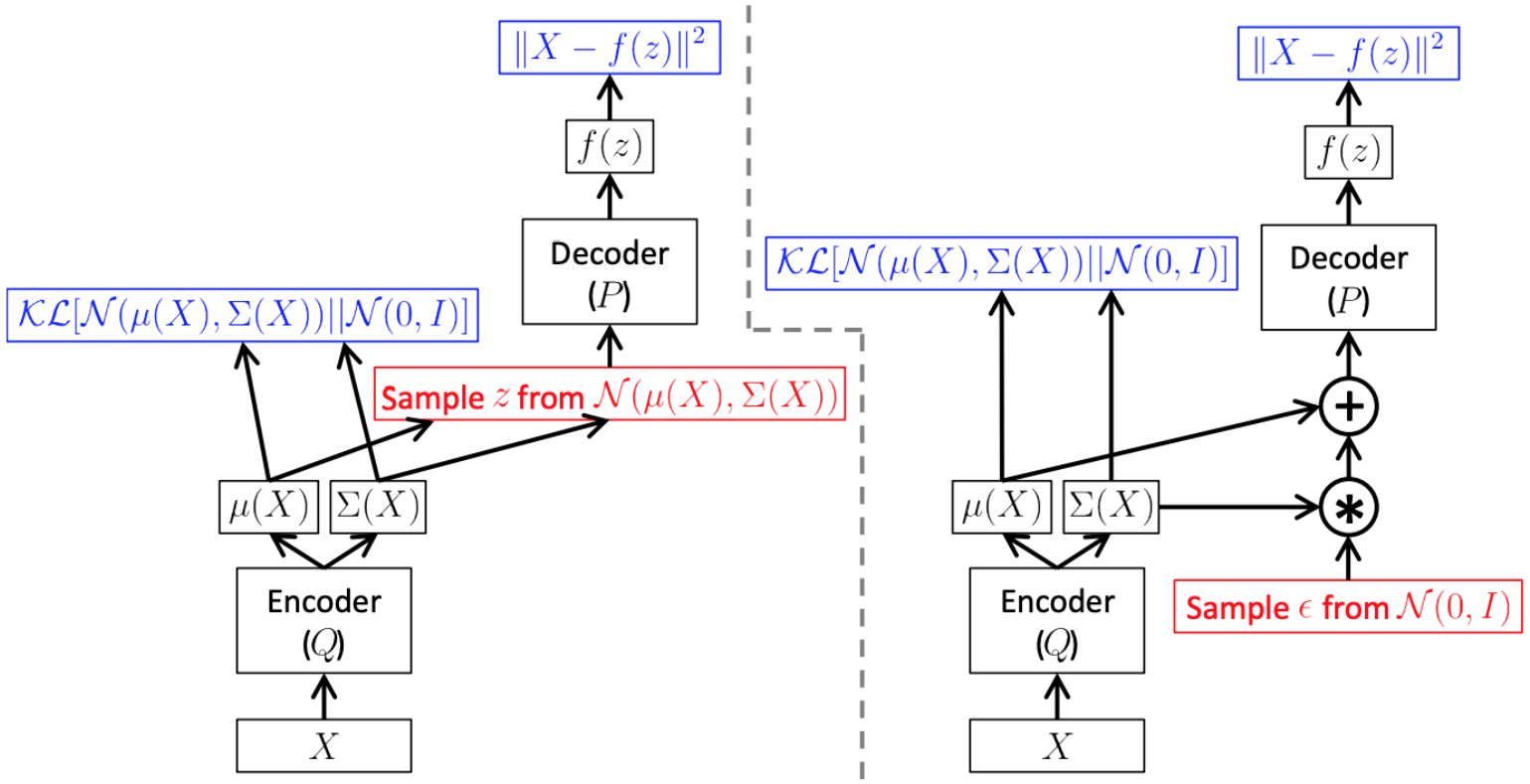
Мы получили нижнюю оценку на правдоподобие, которую мы и будем максимизировать:

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

VAE

Проблема: мы не можем с помощью backpropogation обновить веса в энкодере.



Generative Adversarial Network

GAN

А что если бы мы могли построить модель, которая не нуждается в явном доступе к **p(X)** для генерации?

Задача: научиться трансформировать данные из простых распределений в более сложные.

Генератор:

$\{\tilde{\boldsymbol{x}}_j\} \sim p_{gen,\theta}$ — this distribution is built the following way:

$$\tilde{\boldsymbol{x}}_j = G_\theta(\boldsymbol{z}_j)$$

$$\boldsymbol{z}_j \sim N(\mathbf{0}, \mathbb{I})$$

G_θ — generator neural network

Добавим ещё одну модель $D_\phi(x)$ (дискриминатор), который будет отличать сгенерированные данные от настоящих:

$$\max_\phi \left(\mathbb{E}_{x\sim p(x)}(\log(D_\phi(x)) + \mathbb{E}_{z\sim\mathcal{N}(0;1)}(1 - \log(D_\phi(G_\theta(z))\right)$$

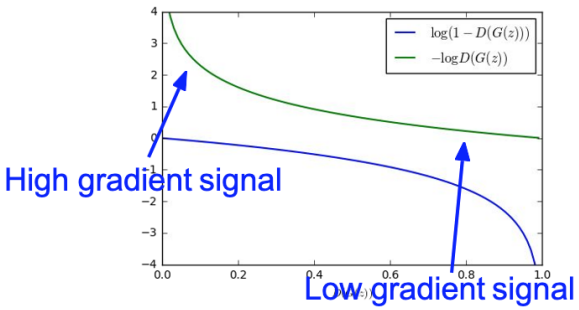
В то же время потребуем от генератора:

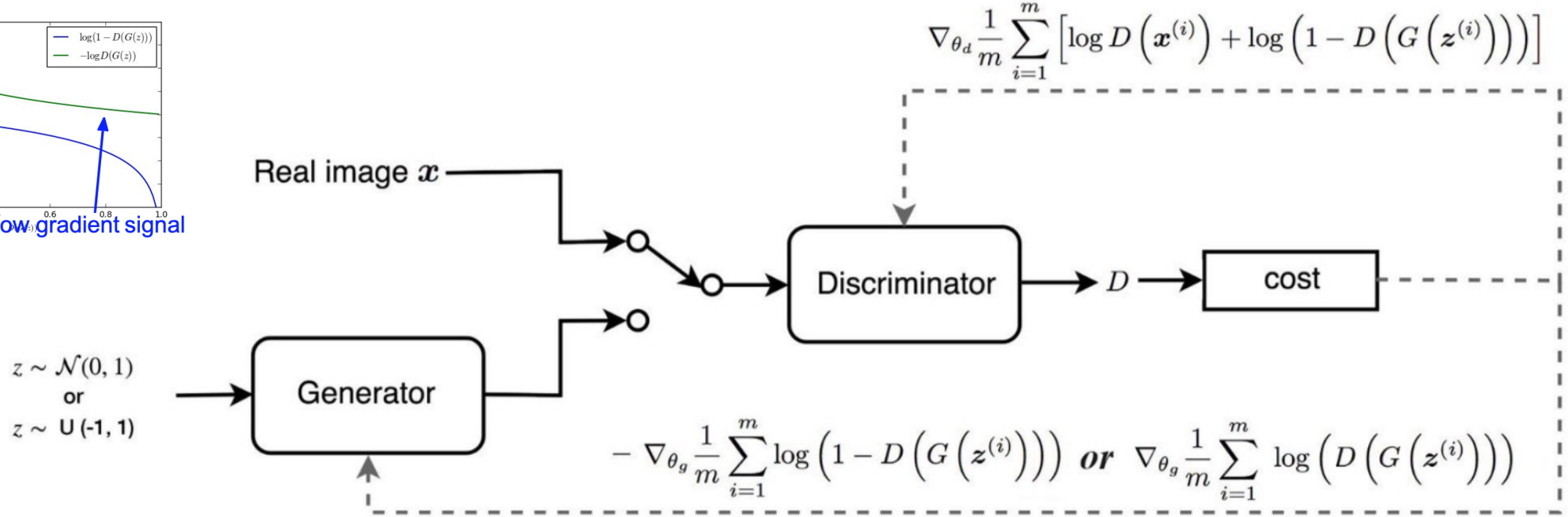
$$\min_\theta \mathbb{E}_{z\sim\mathcal{N}(0;1)}(1 - \log(D_\phi(G_\theta(z))$$

GAN

Получаем следующий loss: $V(\phi, \theta) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} [log D_\phi(\boldsymbol{x})] + \mathbb{E}_{\tilde{\boldsymbol{x}} \sim p_{gen,\theta}} [log\,(1 - D_\phi(\tilde{\boldsymbol{x}}))]$

$$V(\phi, \theta) \rightarrow \min_\theta \max_\phi V(\phi, \theta)$$





GAN

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

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

end for

GAN

При фиксированном $p_{gen,\theta}$, оптимальным значением дискриминатора будет:

$$D_{\phi^*}(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{gen,\theta}(\boldsymbol{x})}$$

Если мы подставим его в наш loss, получим:

$$\min_G V(D^*, G) = 2D_{JS}(p_r \| p_g) - 2\log 2 \quad \text{, где} \quad D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q||\frac{p+q}{2})$$

Получается, что оптимизируя $p_{gen,\theta}$, мы оптимизируем JS divergence

Вопросы

1. Почему PixelCNN работает лучше PixelRNN? В чём заключаются минусы таких подходов?

2. Опишите устройство VAE.

3. Обьяснить смысл лосса GANa.

Список источников

- https://arxiv.org/pdf/1312.6114.pdf
- https://jaan.io/what-is-variational-autoencoder-vae-tutorial/
- https://arxiv.org/pdf/1601.06759.pdf
- https://arxiv.org/pdf/1406.2661.pdf
- https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
- https://towardsdatascience.com/summary-of-pixelrnn-by-google-deepmind-7-min-read-938d9871d6d9