

# **Обучение без учителя: автокодировщик, вариационный автокодировщик**

**Александр Дьяконов**

**02 декабря 2021 года**

## План

### **Автокодировщики (Auto-encoders)**

Denoising Autoencoder

Contractive Autoencoders (CAE)

Sparse Coding

Context Encoders

### **Генеративная модель**

**Variational Autoencoders (VAE)**

**Variational Bayesian Inference**

**Reparametrizaton trick**

**Векторная арифметика**

**Conditional VAE (CVAE)**

**Ladder Variational Autoencoders**

**Bidirectional-Inference Variational Autoencoder (BIVA)**

**Vector Quantised-Variational AutoEncoder (VQ-VAE)**

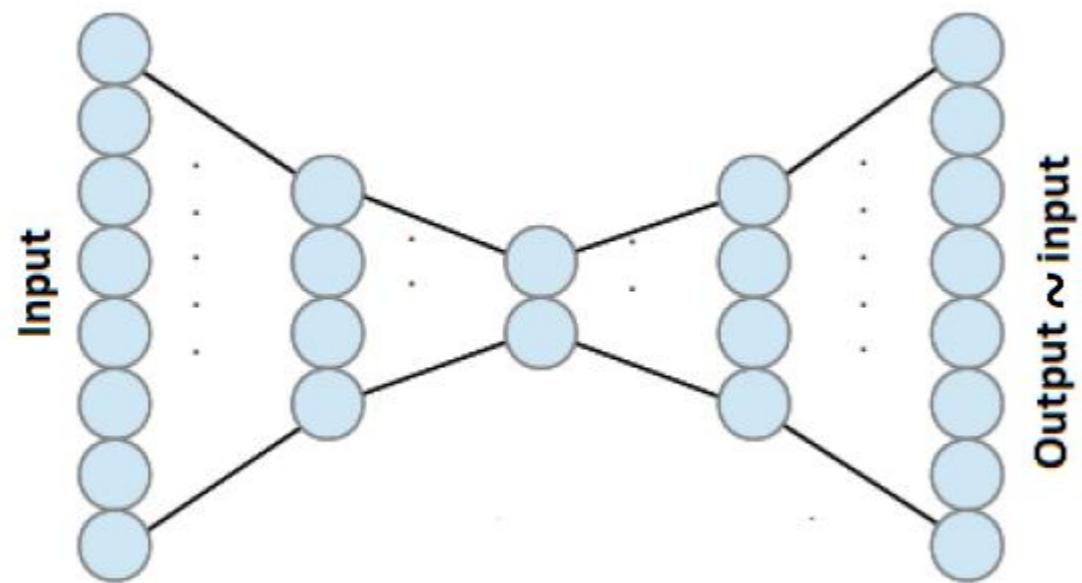
**VQ-VAE-2**

## Обучение без учителя (Unsupervised Learning)

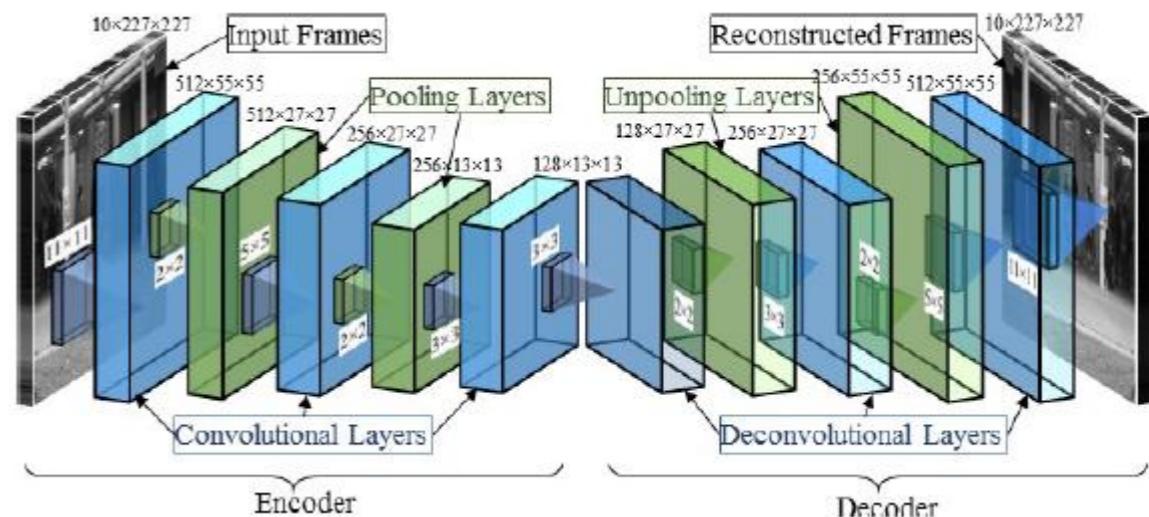
Невероятностные модели	Вероятностные (генеративные) модели		
<ul style="list-style-type: none"><li>• Sparse Coding</li><li>• Autoencoders</li><li>• k-means, ...</li></ul>	Плотность в явном виде (explicit density)		Плотность в неявном виде (implicit density)
	Tractable Models	Approximate density	<ul style="list-style-type: none"><li>• GAN</li><li>• Moment Matching Networks</li></ul> <p>Моделируем процесс сэмплирования, а не плотность</p>
	<ul style="list-style-type: none"><li>• Fully observed Belief Nets</li><li>• NADE</li><li>• MADE</li><li>• PixelRNN</li><li>• nonlinear ICA</li></ul>	Variational  • VAE	

## Автокодировщики (Auto-encoders) / поникающие (undercomplete)

### Полносвязный с узким горлом (bottleneck)



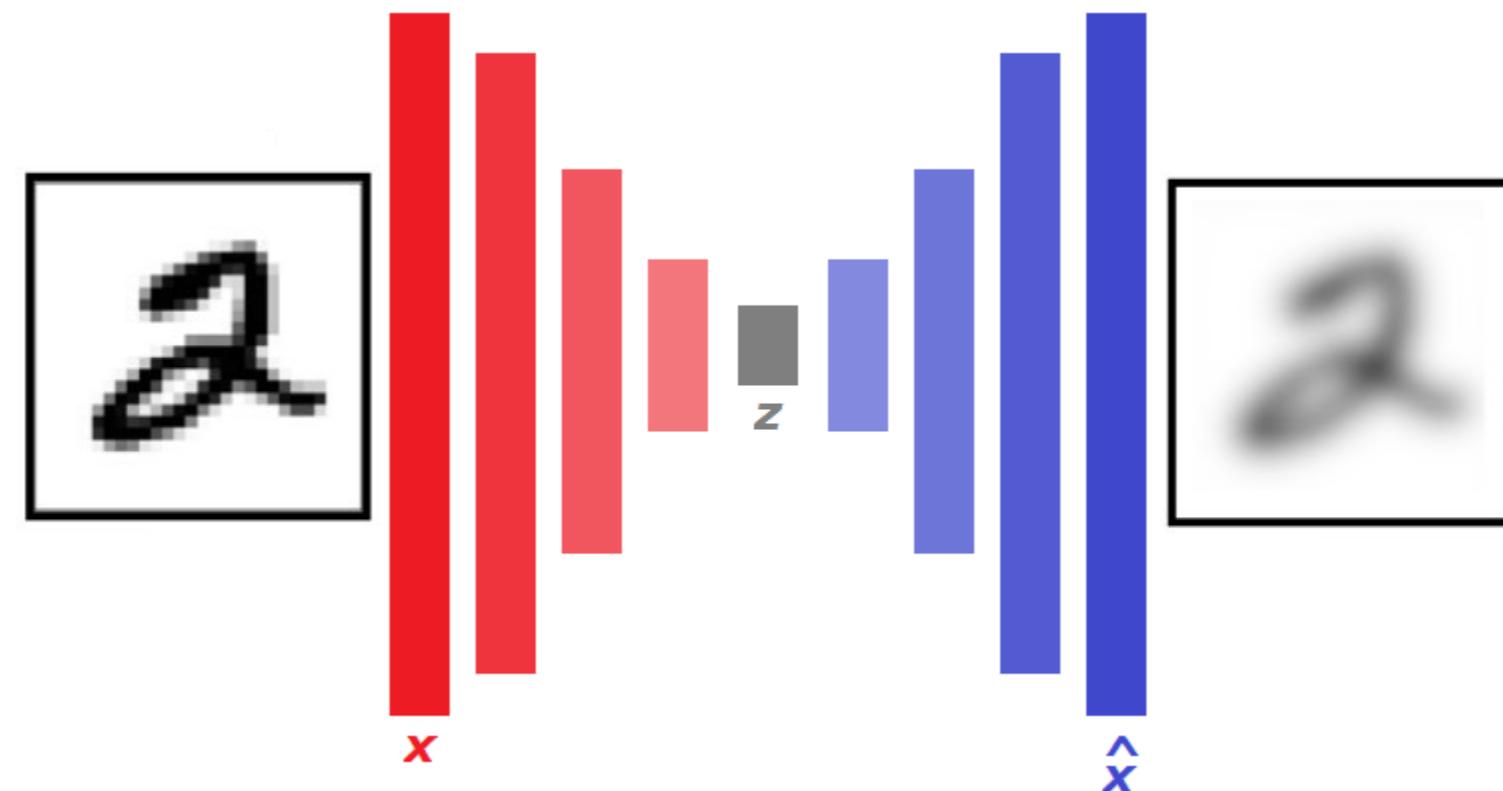
### Свёрточный



**Знакомая архитектура «кодировщик  $p_{\text{enc}}(z | x)$  – декодировщик  $p_{\text{dec}}(x | z)$ »  
encoder-bottleneck-decoder // reconstruction loss**

**сжимаем информацию в латентное пространство,  
пытаемся её восстановить**

## Глубокие автокодировщики – обучение

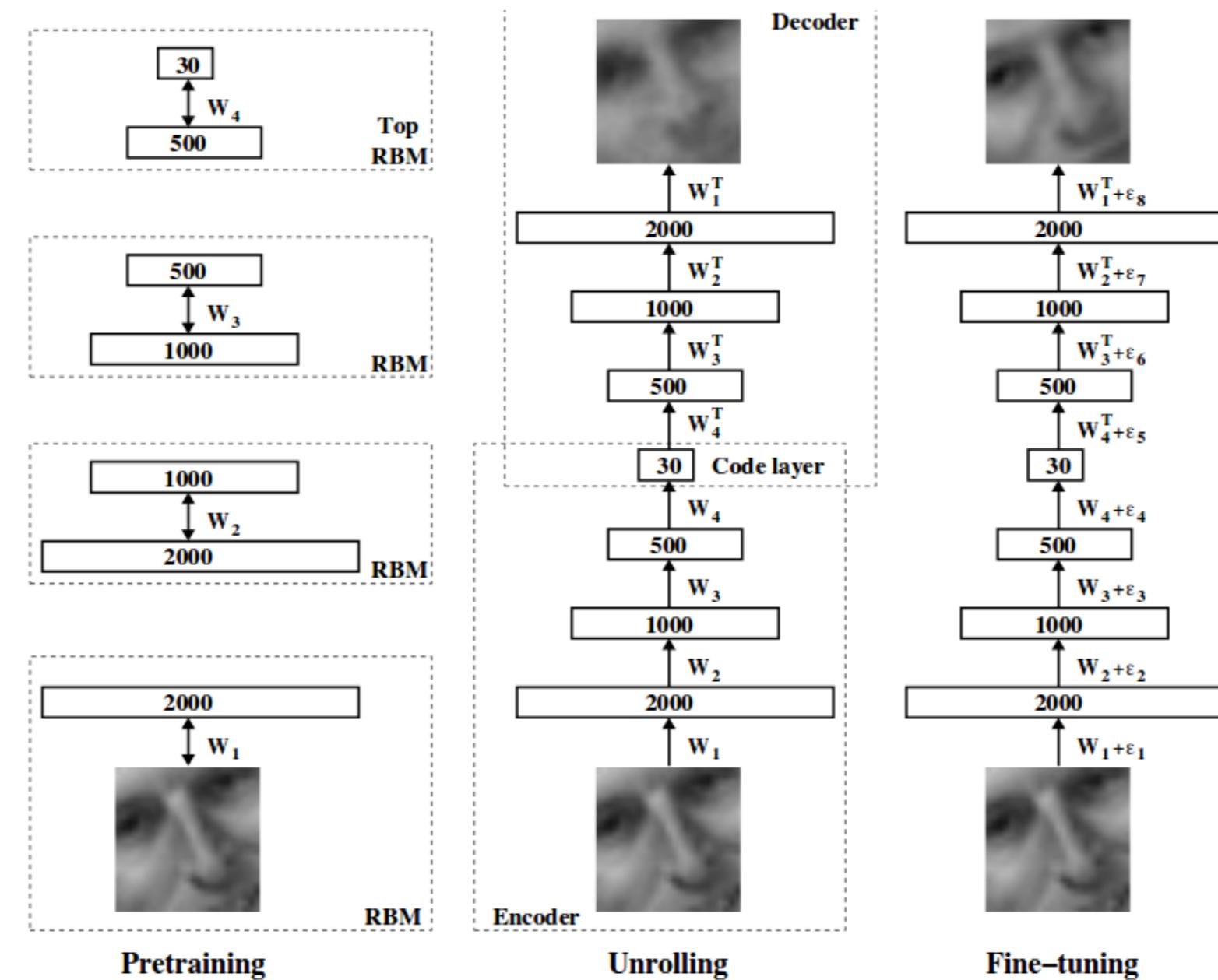


**воспроизводим вход ~ reconstruction error / loss + regularizer**

$$\|x - g(f(x))\| + R(f, g) \rightarrow \min$$

$$\underbrace{\hat{x}}_z$$

## Глубокие автокодировщики – обучение с помощью RBM (раньше)



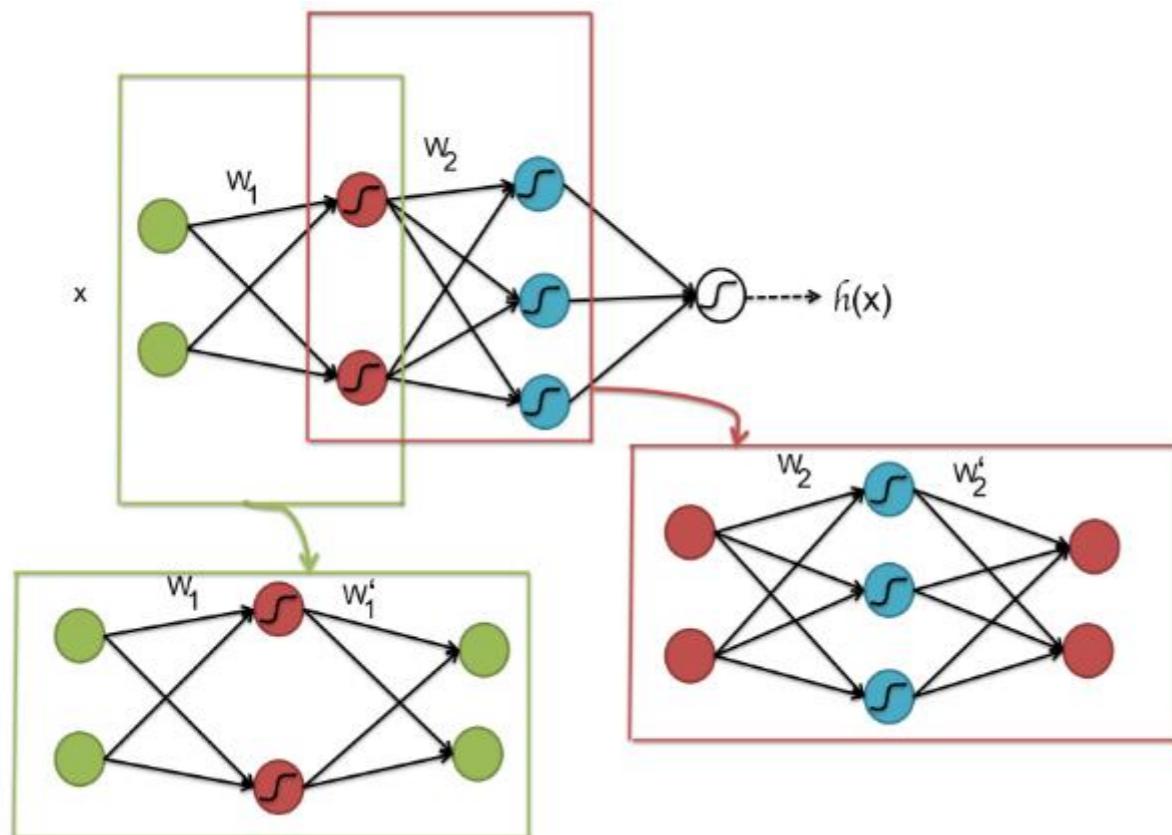
## Зачем нужны автокодировщики

- **сокращение размерности**  
и в новой размерности любая задача
  - **кластеризация**
  - **детектирование аномалий**
  - **поиск**
- **выделение признаков (для других алгоритмов)**
- **предобучение**
- **специальные задачи**
  - **представления специального вида**
  - **устранение шума (Data Denoising)**
  - **image inpainting**

**если нет нелинейности, то ~ PCA**

## Предобучение с помощью автокодировщика (раньше так делали)

послойно обучить автокодировщики



**первый слой должен воспроизводить вход**

**второй слой должен воспроизводить  
первый и т.д.**

**обучить последний слой, используя  
размеченные данные**

**обучить всю сеть, используя размеченные  
данные**

<https://cs.stanford.edu/~quocle/tutorial2.pdf>

## Выученные фильтры автокодировщика

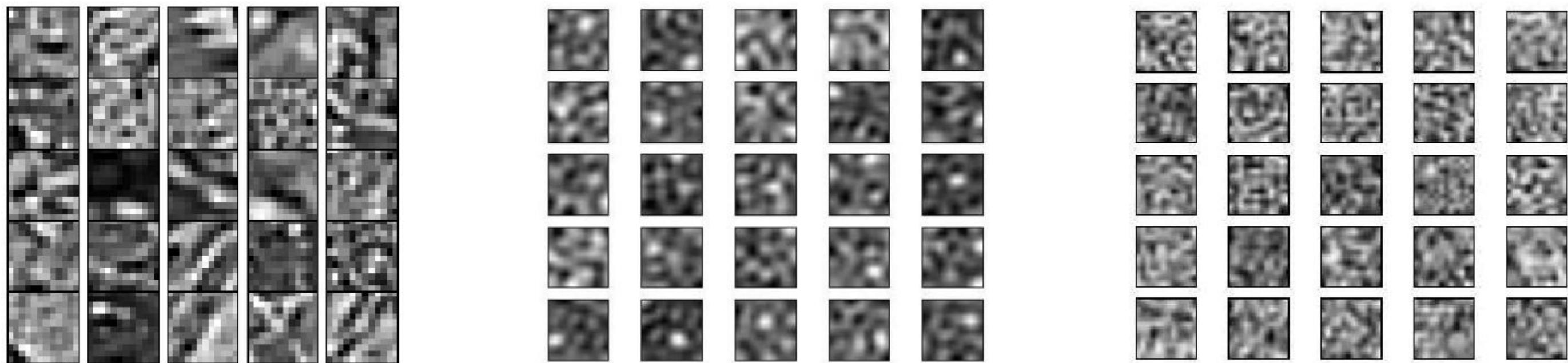


Figure 5: Regular autoencoder trained on natural image patches. *Left*: some of the  $12 \times 12$  image patches used for training. *Middle*: filters learnt by a regular *under-complete* autoencoder (50 hidden units) using tied weights and L2 reconstruction error. *Right*: filters learnt by a regular *over-complete* autoencoder (200 hidden units). The under-complete autoencoder appears to learn rather uninteresting local blob detectors. Filters obtained in the over-complete case have no recognizable structure, looking entirely random.

## Минутка кода: свёрточный автокодировщик

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

class ConvAutoencoder(nn.Module):
    def __init__(self):
        super(ConvAutoencoder, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1) # conv layer (depth from 1 --> 16), 3x3 kernels
        self.conv2 = nn.Conv2d(16, 4, 3, padding=1) # conv layer (depth from 16 --> 4), 3x3 kernels
        self.pool = nn.MaxPool2d(2, 2) # pooling layer to reduce x-y dims by two; kernel and stride of 2
        self.t_conv1 = nn.ConvTranspose2d(4, 16, 2, stride=2) # a kernel of 2 and a stride of 2 will
        self.t_conv2 = nn.ConvTranspose2d(16, 1, 2, stride=2) # increase the spatial dims by 2

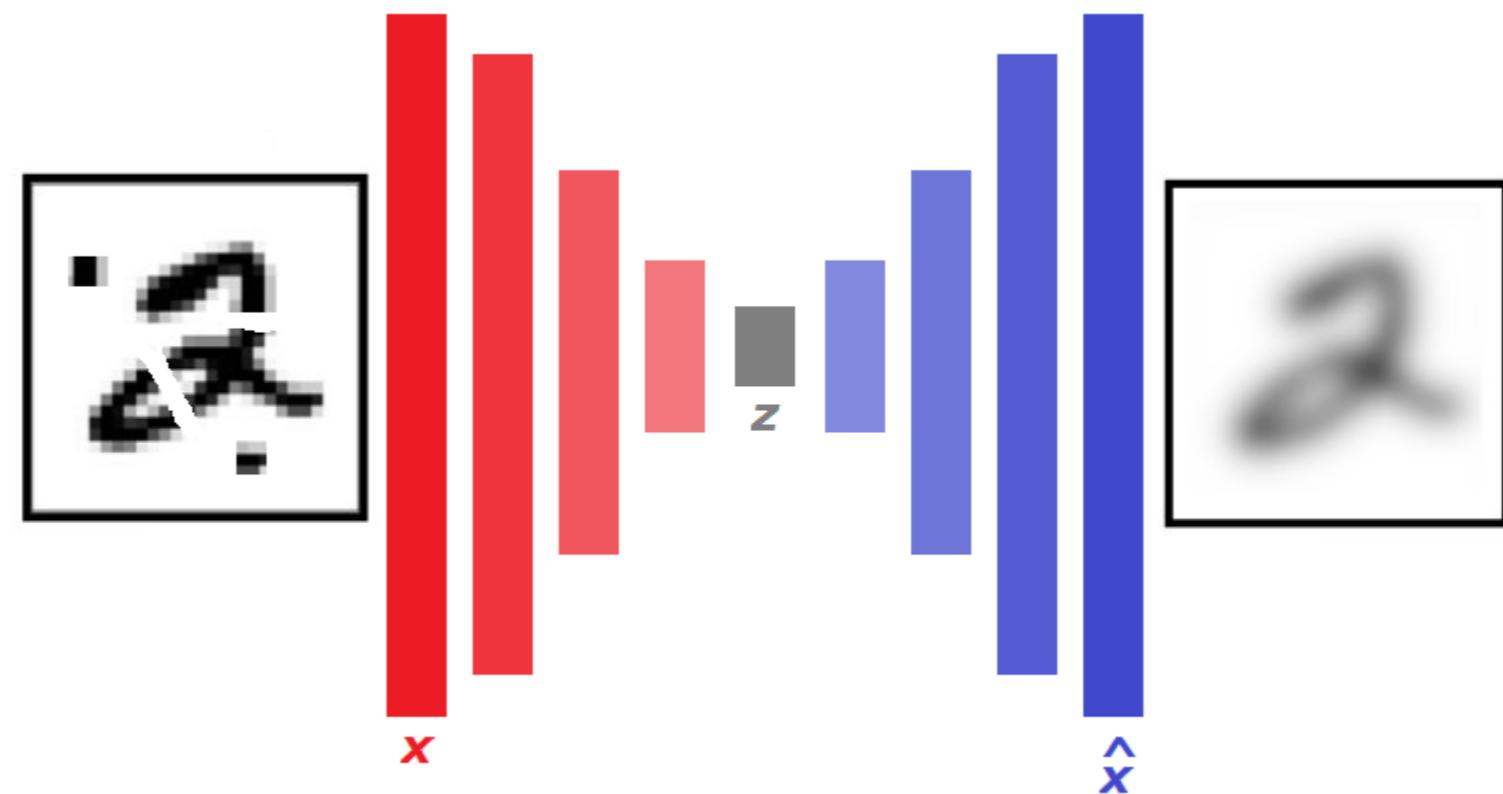
    def forward(self, x):
        ## encoder ##
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x) # compressed representation
        ## decoder ##
        x = F.relu(self.t_conv1(x))
        x = F.sigmoid(self.t_conv2(x)) # output layer (with sigmoid for scaling from 0 to 1)

    return x

model = ConvAutoencoder()
```

[https://github.com/udacity/deep-learning-v2-pytorch/blob/master/autoencoder/convolutional-autoencoder/Convolutional\\_Autoencoder\\_Solution.ipynb](https://github.com/udacity/deep-learning-v2-pytorch/blob/master/autoencoder/convolutional-autoencoder/Convolutional_Autoencoder_Solution.ipynb)

## Denoising Autoencoder



**увеличение выборки с помощью дополнения к ней  
зашумлённых изображений (зашумления разного типа)**

- **больше данных**
- **правильнее формируемые признаки**

$$\| x - g(f(\tilde{x})) \| + R(f, g) \rightarrow \min$$

## Denoising Autoencoder

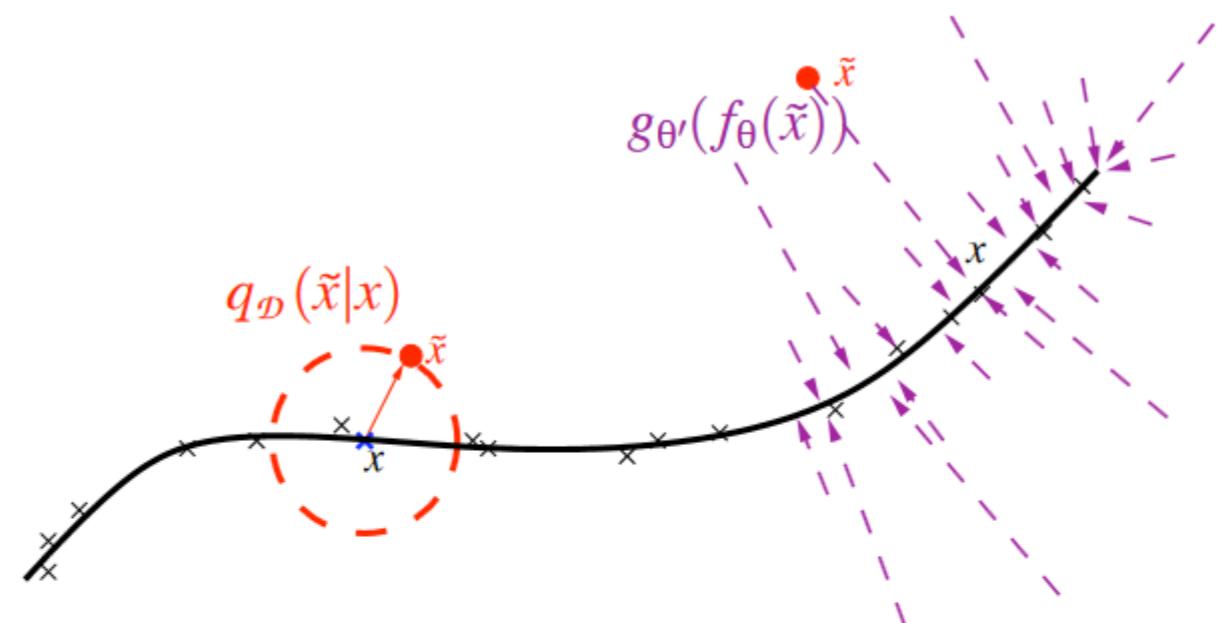


Figure 2: Manifold learning perspective. Suppose training data ( $\times$ ) concentrate near a low-dimensional manifold. Corrupted examples ( $\bullet$ ) obtained by applying corruption process  $q_{\mathcal{D}}(\tilde{X}|X)$  will generally lie farther from the manifold. The model learns with  $p(X|\tilde{X})$  to “project them back” (via autoencoder  $g'_{\theta}(f_{\theta}(\cdot))$ ) onto the manifold. Intermediate representation  $Y = f_{\theta}(X)$  may be interpreted as a coordinate system for points  $X$  on the manifold.

<https://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf>

## Выученные фильтры «шумоподавляющего» автокодировщика

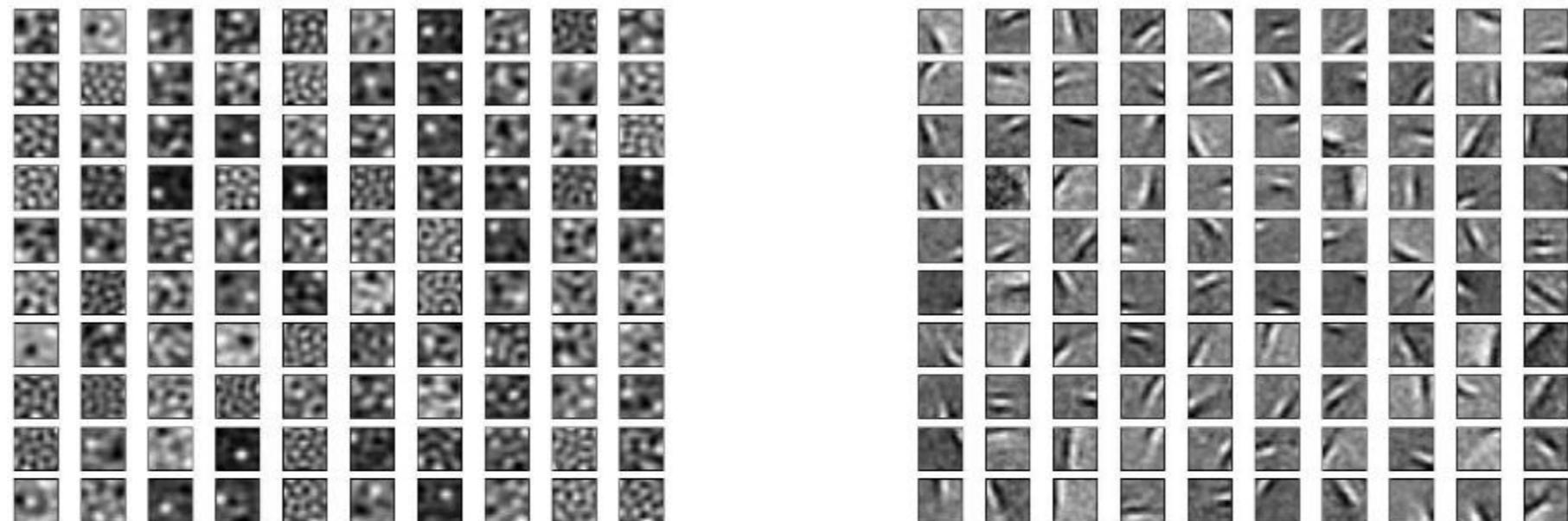


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left*: regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right*: a denoising autoencoder with additive Gaussian noise ( $\sigma = 0.5$ ) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

<https://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf>

## Выученные фильтры «шумоподавляющего» автокодировщика

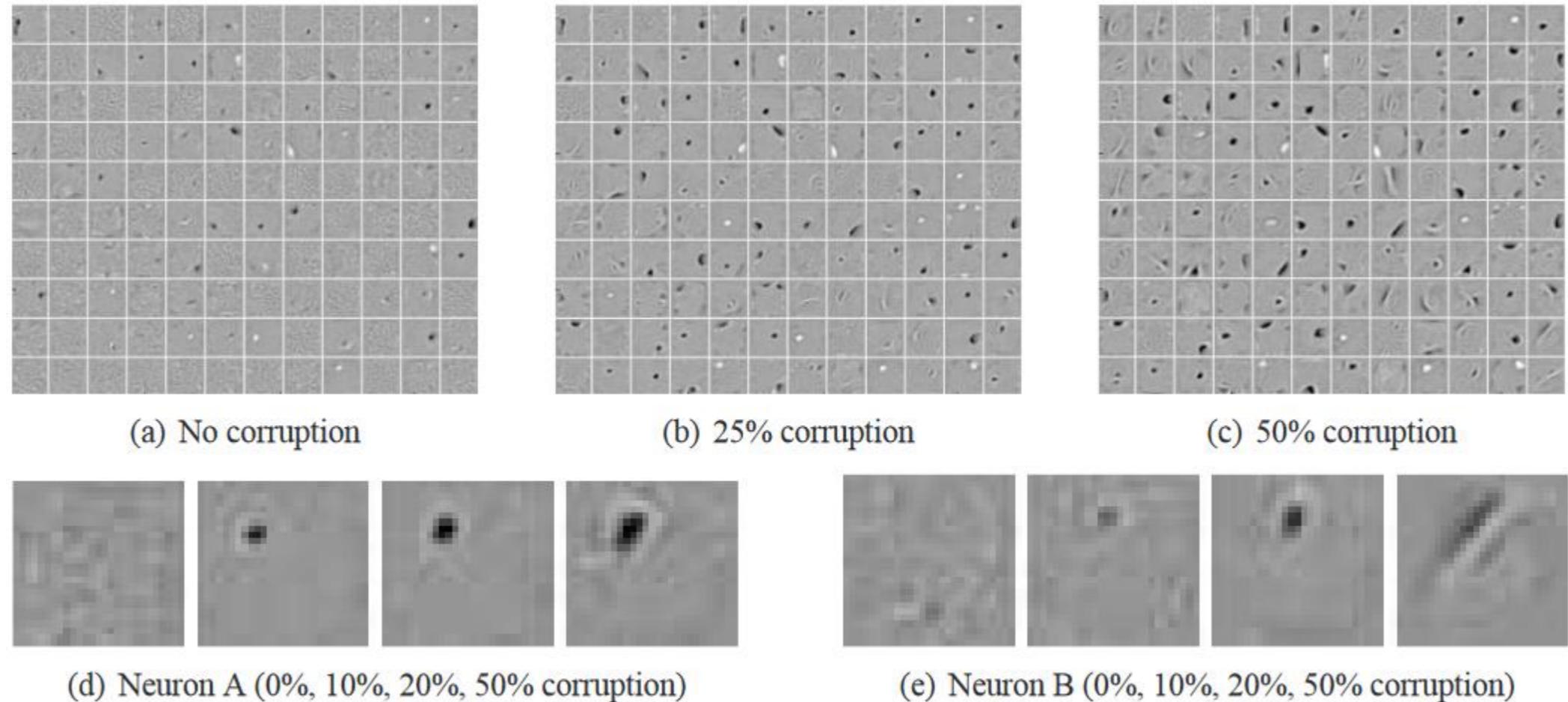


Figure 8: Filters learnt by denoising autoencoder on MNIST digits, using zero-masking noise. (a-c) show some of the filters learnt by denoising autoencoders trained with various corruption levels  $\nu$ . Filters at the same position in the three images are related only by the fact that the autoencoders were started from the same random initialization point in parameter space. (d) and (e) zoom in on the filters obtained for two of the neurons. As can be seen, with no noise, many filters remain similarly uninteresting (undistinctive almost uniform random grey patches). As we increase the noise level, denoising training forces the filters to differentiate more, and capture more distinctive features. Higher noise levels tend to induce less local filters, as expected. One can distinguish different kinds of filters, from local blob detectors, to stroke detectors, and character parts detectors at the higher noise levels.

## Сокращающие автокодировщики – Contractive Autoencoders (CAE)

**идея – сделать представления менее чувствительными  
к небольшим изменениям данных**

⇒ производные по входу должны быть  $\approx 0$

**А «denoising autoencoders» была устойчивость к зашумлению!**

$$\|x - g(f(x))\| + \lambda \|J(x)\|_F^2$$

**штраф – квадрат нормы Фробениуса  $\|\cdot\|_F^2$  Якобиана  $J$**

$$\|J(x)\|_F^2 = \sum_{ij} \left( \frac{\partial f_j(x)}{\partial x_i} \right)^2$$

**производные берутся в скрытом слое  $f(x)$**

**Salah Rifai et al «Contractive Auto-Encoders: Explicit Invariance During Feature Extraction» //  
[https://icml.cc/Conferences/2011/papers/455\\_icmlpaper.pdf](https://icml.cc/Conferences/2011/papers/455_icmlpaper.pdf)**

## Сокращающие автокодировщики – Contractive Autoencoders (CAE)

**Вынуждаем функцию иметь близкую к нулю производную в окрестностях латентных представлений точек**



Figure 1: Regularization forces the auto-encoder to become less sensitive to the input, but minimizing reconstruction error forces it to remain sensitive to variations along the manifold of high density. Hence the representation and reconstruction end up capturing well variations on the manifold while mostly ignoring variations orthogonal to it.

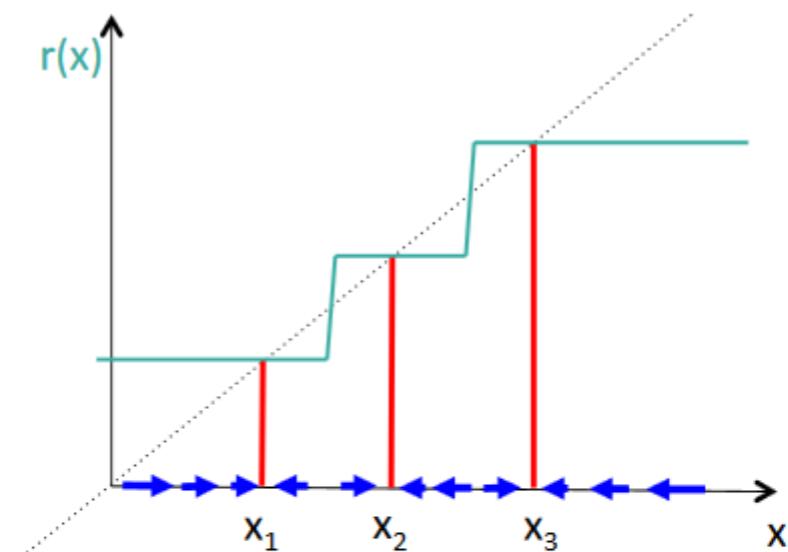


Figure 2: The reconstruction function  $r(x)$  (in turquoise) which would be learned by a high-capacity auto-encoder on a 1-dimensional input, i.e., minimizing reconstruction error *at the training examples  $x_i$*  (with  $r(x_i)$  in red) while trying to be as constant as possible otherwise. The figure is used to exaggerate and illustrate the effect of the regularizer (corresponding to a large  $\sigma^2$  in the loss function  $\mathcal{L}$  later described by (6)). The dotted line is the identity reconstruction (which might be obtained without the regularizer). The blue arrows show the vector field of  $r(x) - x$  pointing towards high density peaks as estimated by the model, and estimating the score (log-density derivative), as shown in this paper.

**Guillaume Alain and Yoshua Bengio «What Regularized Auto-Encoders Learn from the DataGenerating Distribution» // <https://arxiv.org/pdf/1211.4246.pdf>**

## Разреженное представление (Sparse Coding)

**Пусть выборка –  $\{x_i\}_{i=1}^m$ ,**

**базисы  $\{b_i\}_{i=1}^k$ ,**

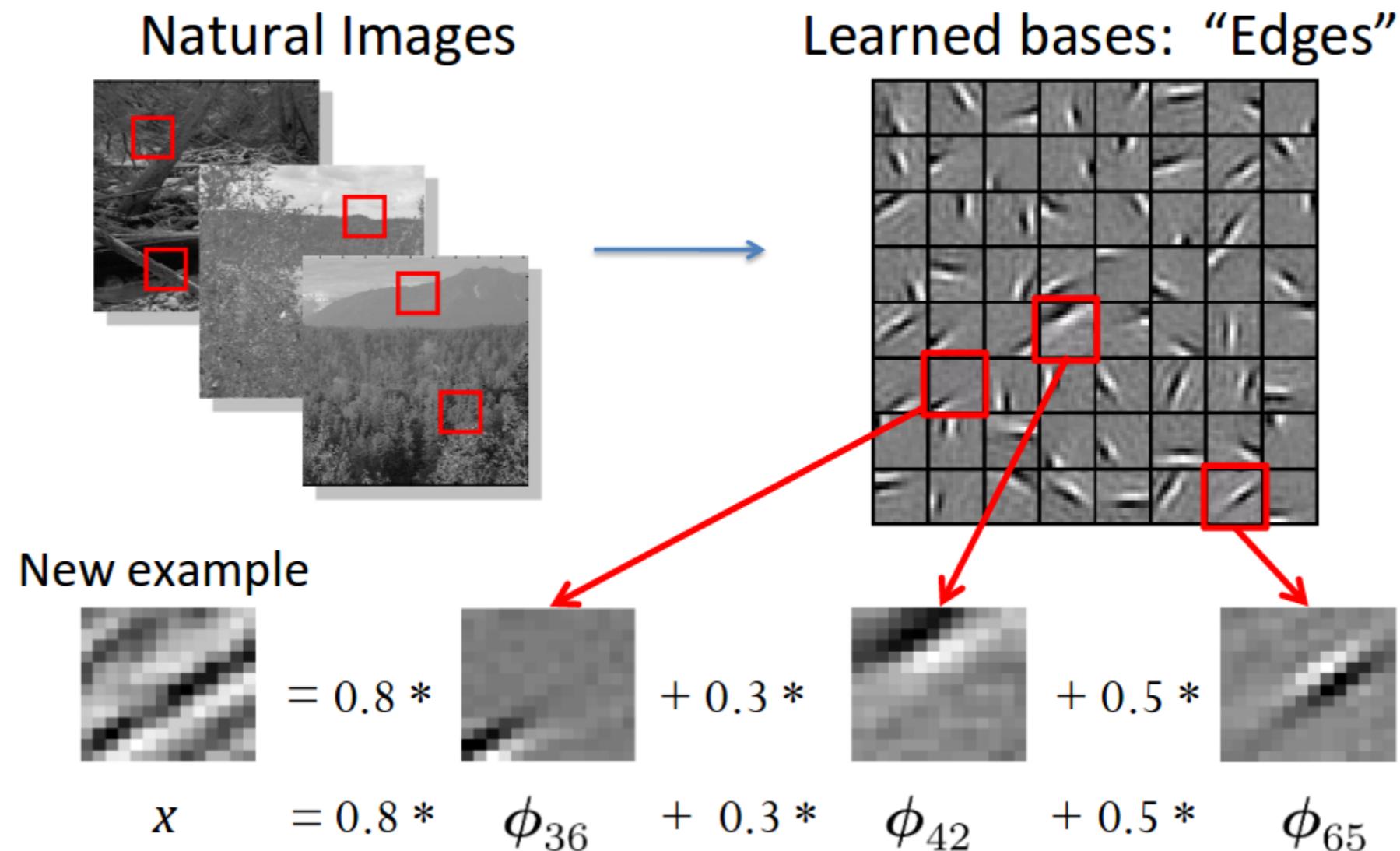
**хотим представить в виде разреженной линейной комбинации  
(sparse linear combination)**

$x_i \approx \sum_{j=1}^k \alpha_{ij} b_j$ : **в основном все  $\alpha_{ij}$  нулевые!**

$$\sum_{i=1}^m \left\| x_i - \sum_{j=1}^k \alpha_{ij} b_j \right\|_2^2 + \lambda \sum_{i=1}^m \sum_{j=1}^k |\alpha_{ij}| \rightarrow \min_{\alpha, b}$$

**Alternative optimization** – попаременно фиксировать базис и коэффициенты

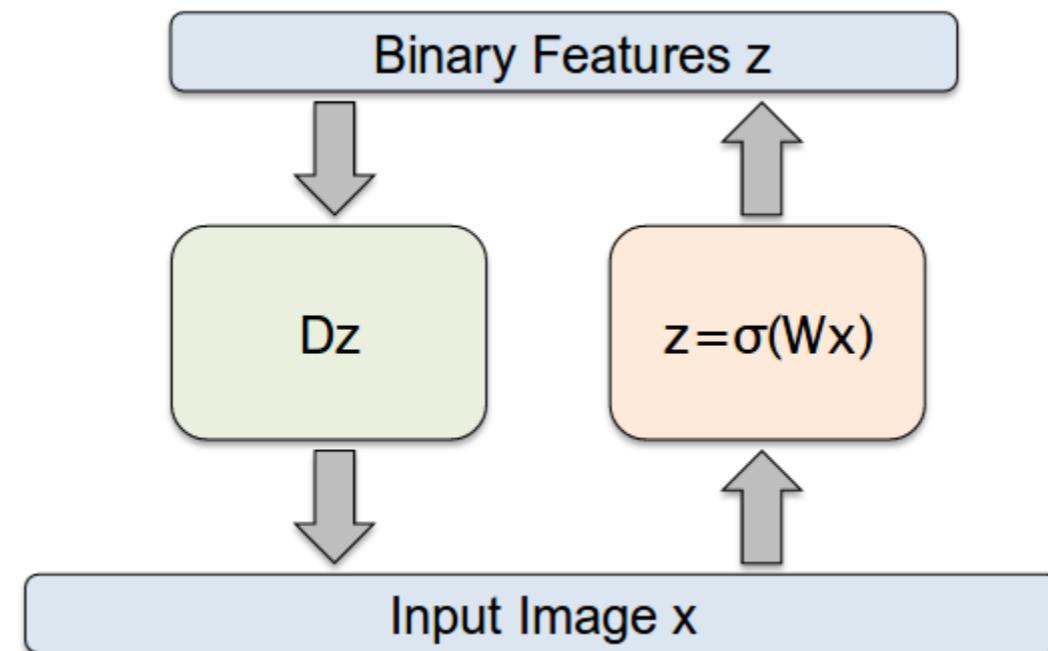
## Разреженное представление (Sparse Coding)



Kavukcuoglu, Ranzato, Fergus, LeCun, 2009

«Глубокое обучение»

## Sparse Coding – вписывается в парадигму автокодировщика

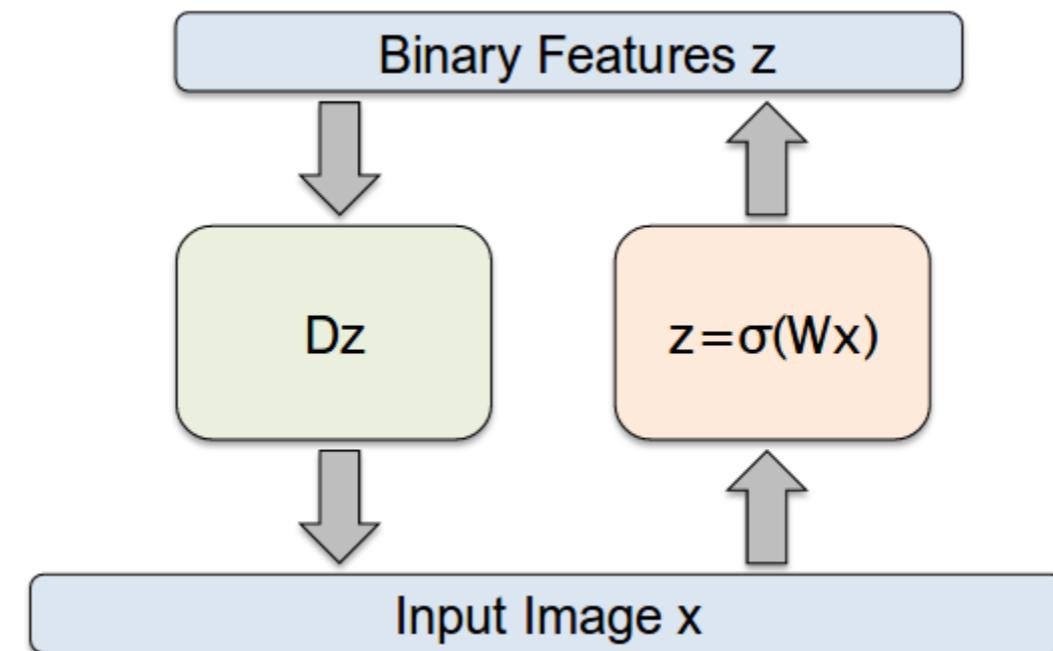


**к скрытых бинарных нейронов**

$$X_j \approx \sum_{t=1}^k D_{jt} \sigma \left( \sum_{i=1}^n W_{ti} X_i \right)$$

**+ L1 регуляризация на значения  $z = \sigma(Wx)$**

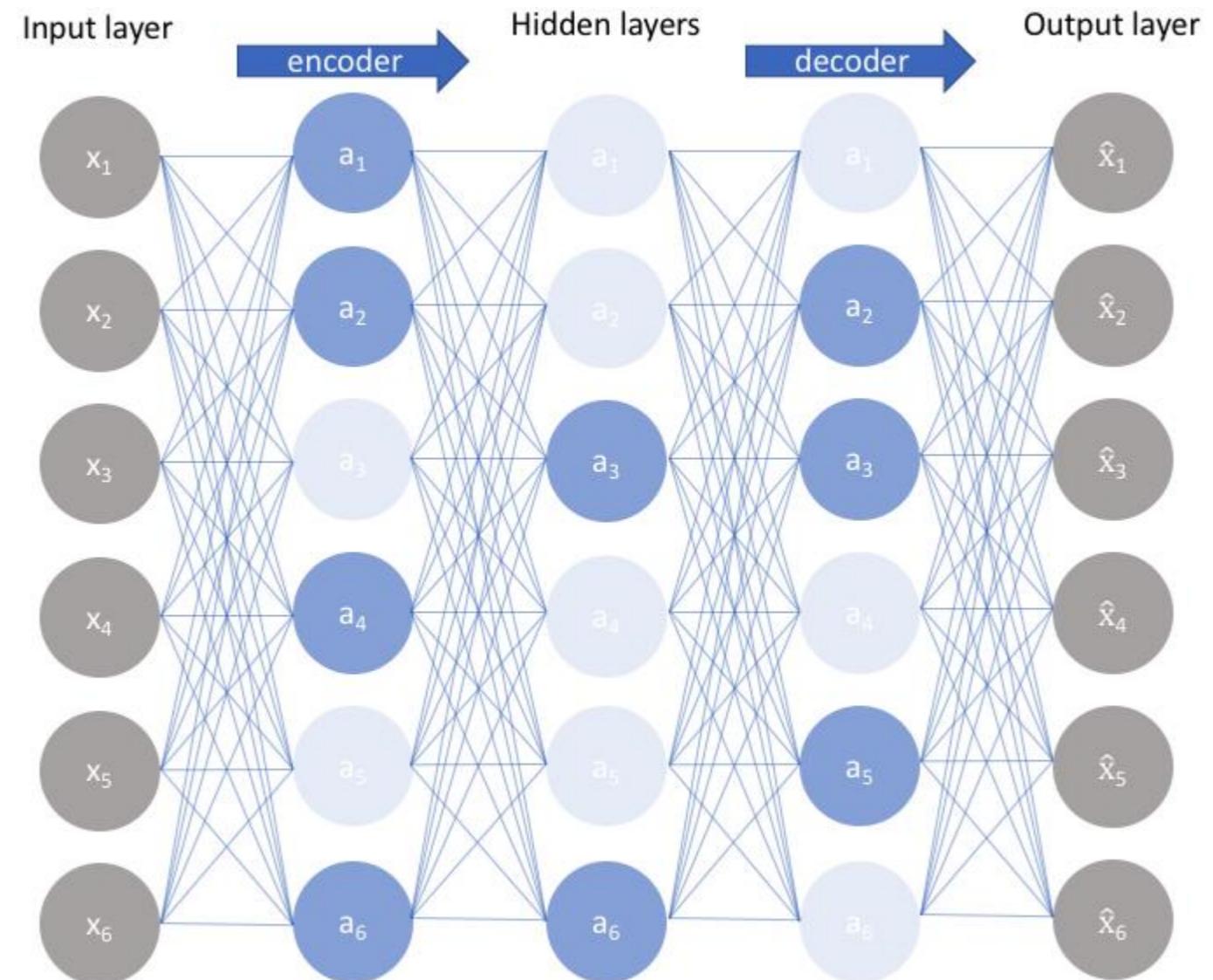
## Sparse Coding – вписывается в парадигму автокодировщика



**к скрытых бинарных нейронов**

$$\| Dz - x \|_2^2 + \lambda \| z \|_1 + \| \sigma(Wx) - z \|_2^2 \rightarrow \min_{D, W, z}$$

## Sparse autoencoders



<https://www.jeremyjordan.me/autoencoders/>

есть ещё вариант с KL-регуляризацией

## Context Encoders

Это формально не автокодировщик...

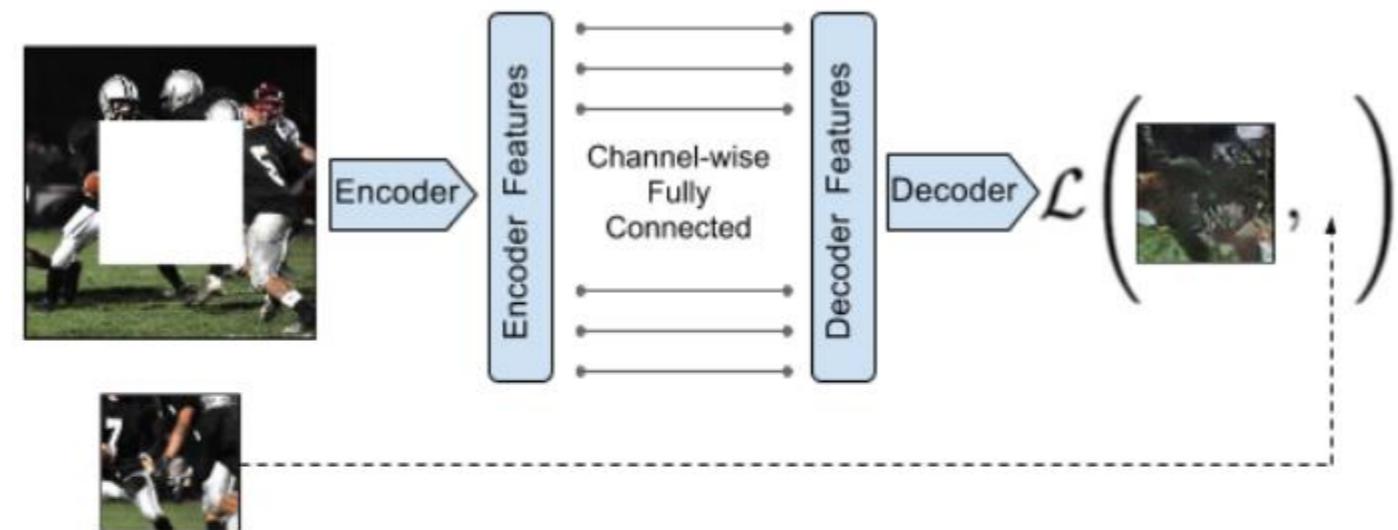


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

<https://people.eecs.berkeley.edu/~pathak/papers/cvpr16.pdf>

## Context Encoders

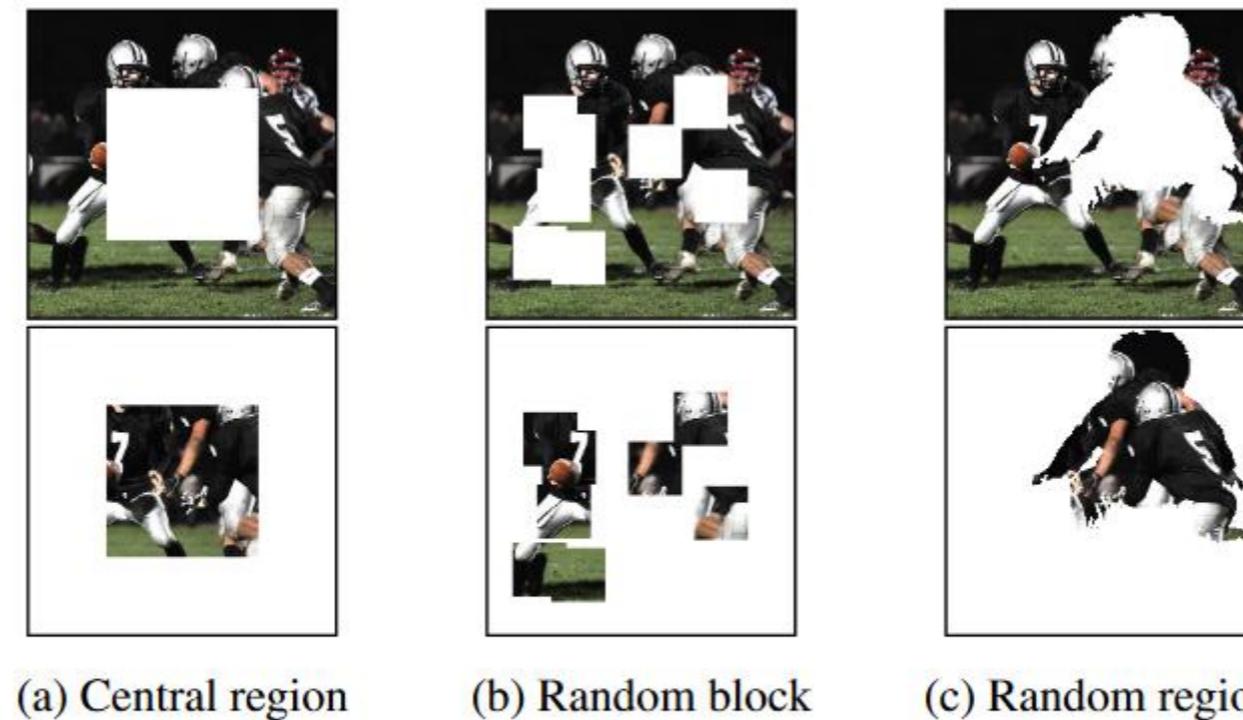
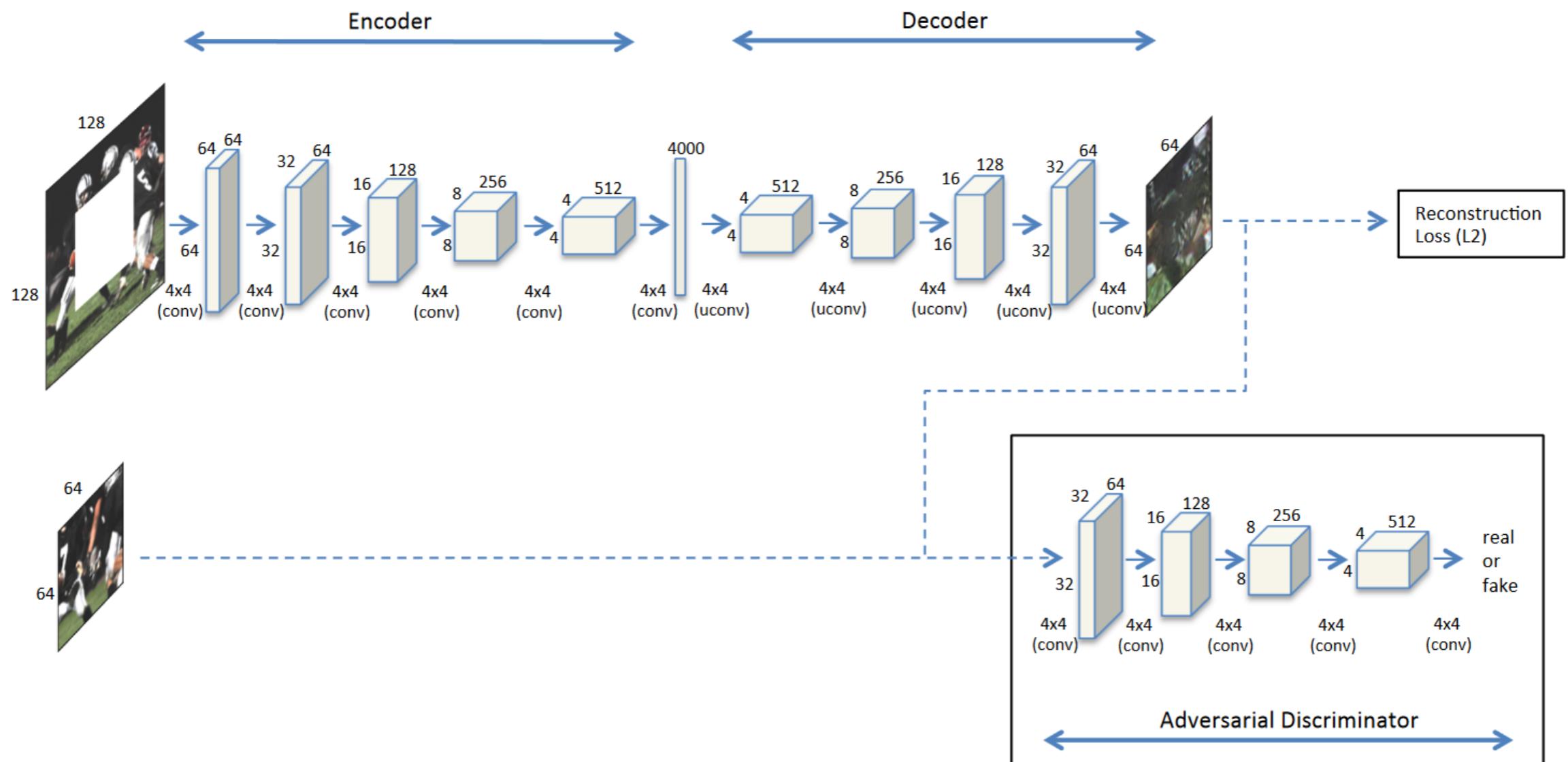


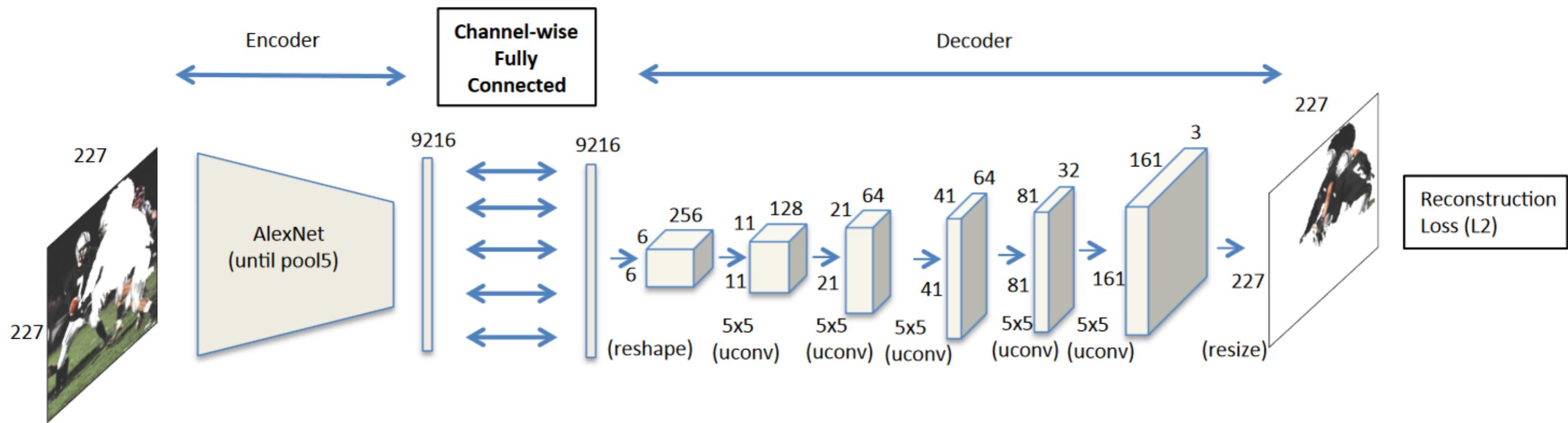
Figure 3: An example of image  $x$  with our different region masks  $\hat{M}$  applied, as described in Section 3.3.

## Context Encoders



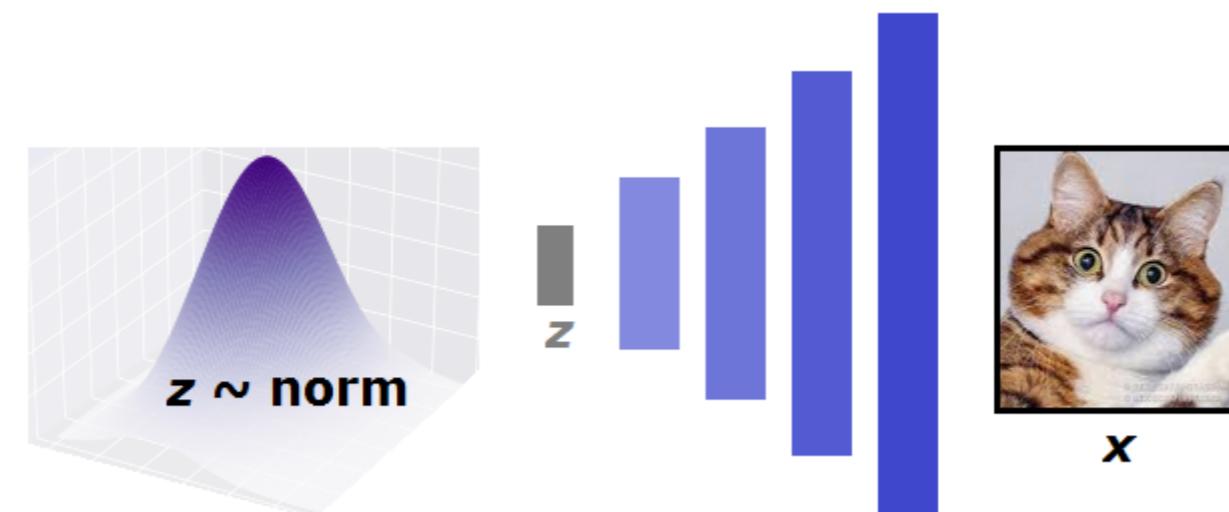
(a) Context encoder trained with joint reconstruction and adversarial loss for semantic inpainting. This illustration is shown for *center region dropout*. Similar architecture holds for arbitrary region dropout as well. See Section 3.2.

## Context Encoders



(b) Context encoder trained with reconstruction loss for feature learning by filling in *arbitrary region dropouts* in the input.

## Генеративная модель



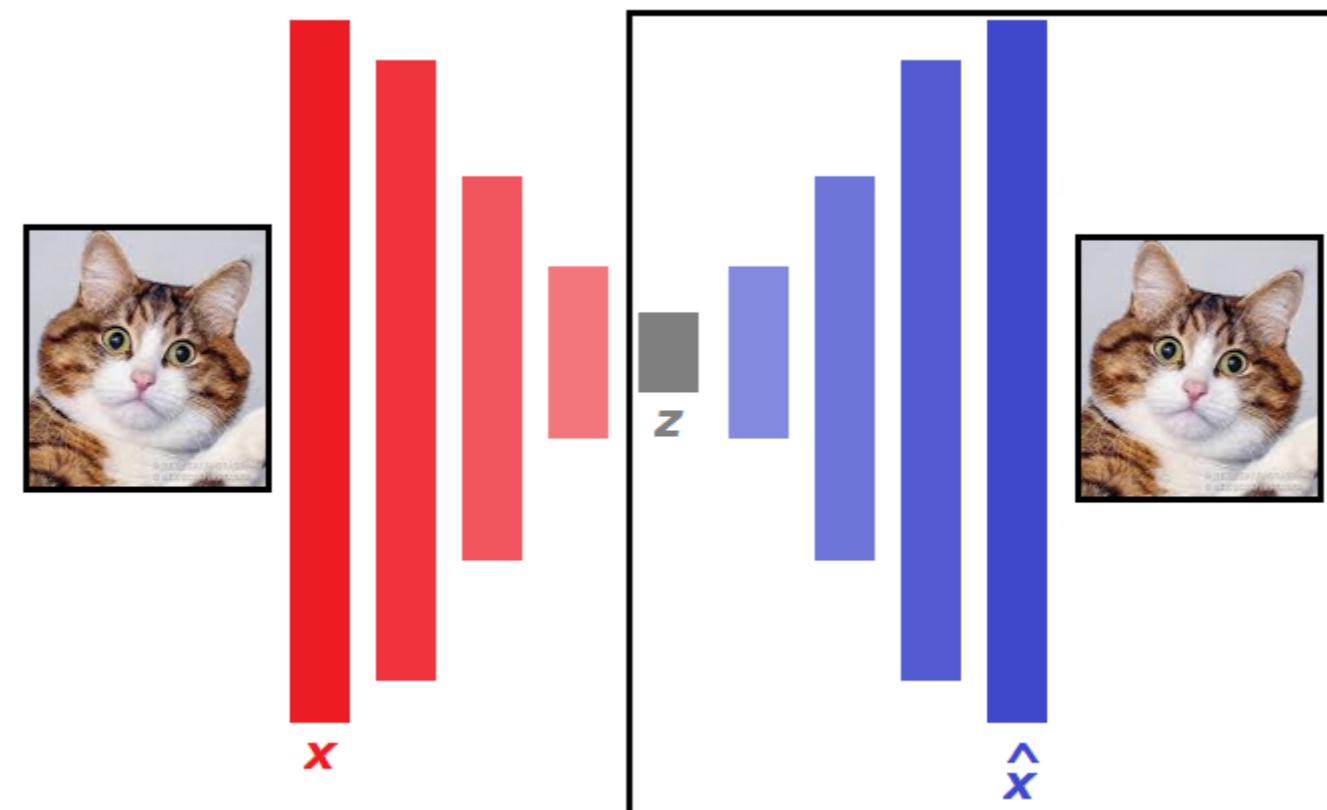
**Проблема – как сделать отображение «шум → что-то разумное»**

дискриминативная постановка понятнее: различаем несколько категорий  
генеративная – порождение!

## Условная генеративная модель (Conditional Generative Model)



## Минусы стандартного автокодировщика



**вряд ли выделенная часть будет генеративной моделью, а хотелось бы...**

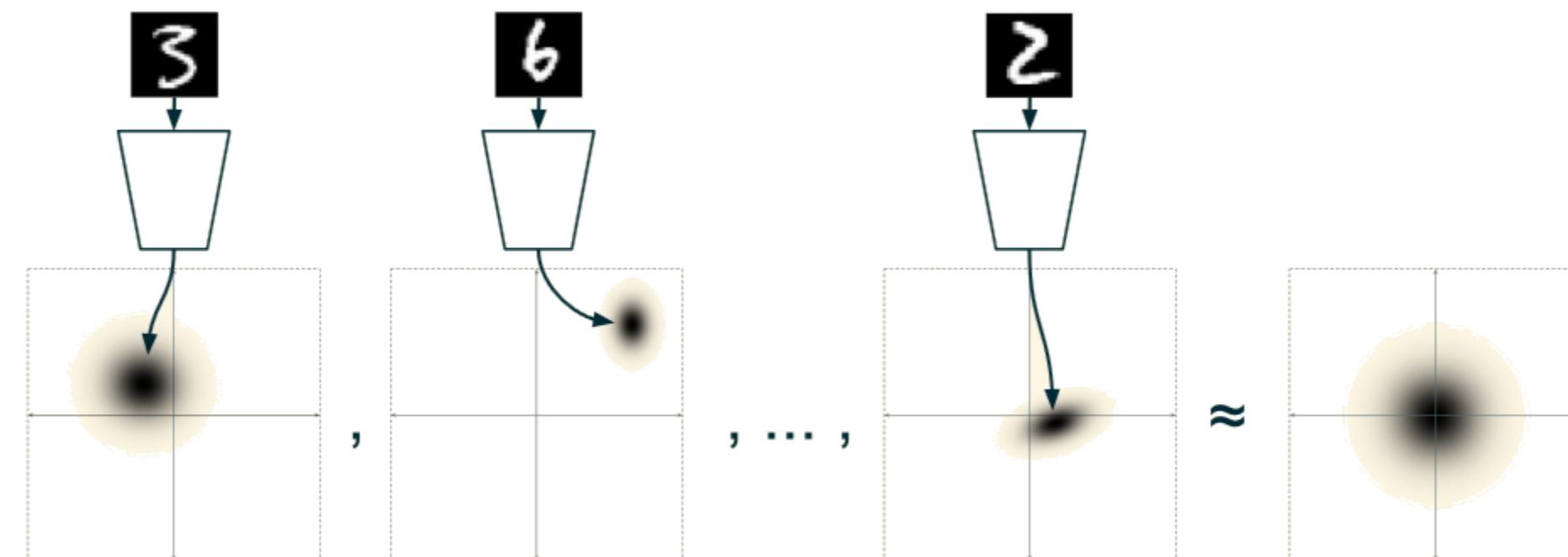
**мы хотим генерировать то, чего не было в обучении**  $\Rightarrow$

- 1) плотность латентного пространства**
- 2) семантическая устойчивость к шумам**

## Variational Autoencoders (VAE)

1) плотность латентного пространства  
**пусть латентное пространство ~ распределение**

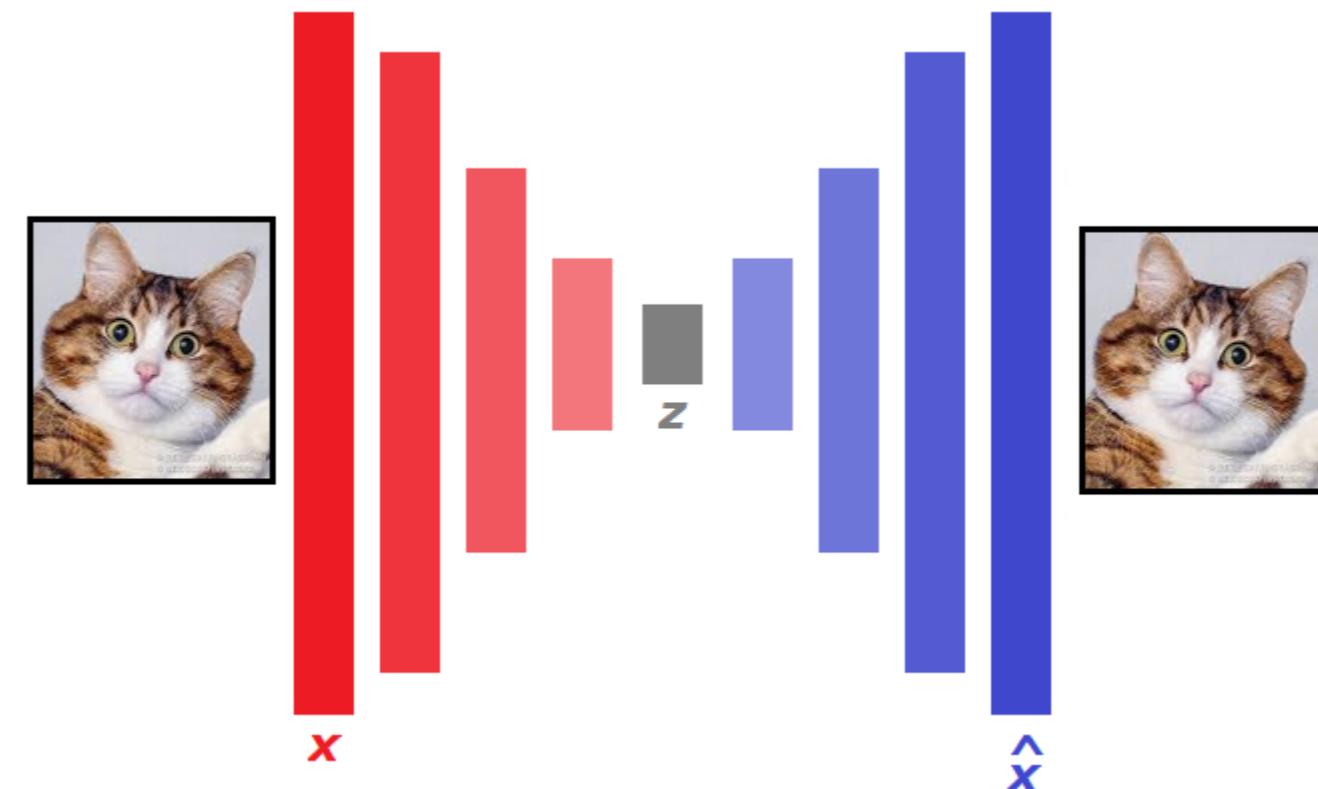
2) семантическая устойчивость к шумам  
**пусть отображение не в вектор, а в распределение (точнее в центр распределения)**  
декодировщику подадим любую точку из этого распределения



<https://ijdykeman.github.io/ml/2016/12/21/cvae.html>

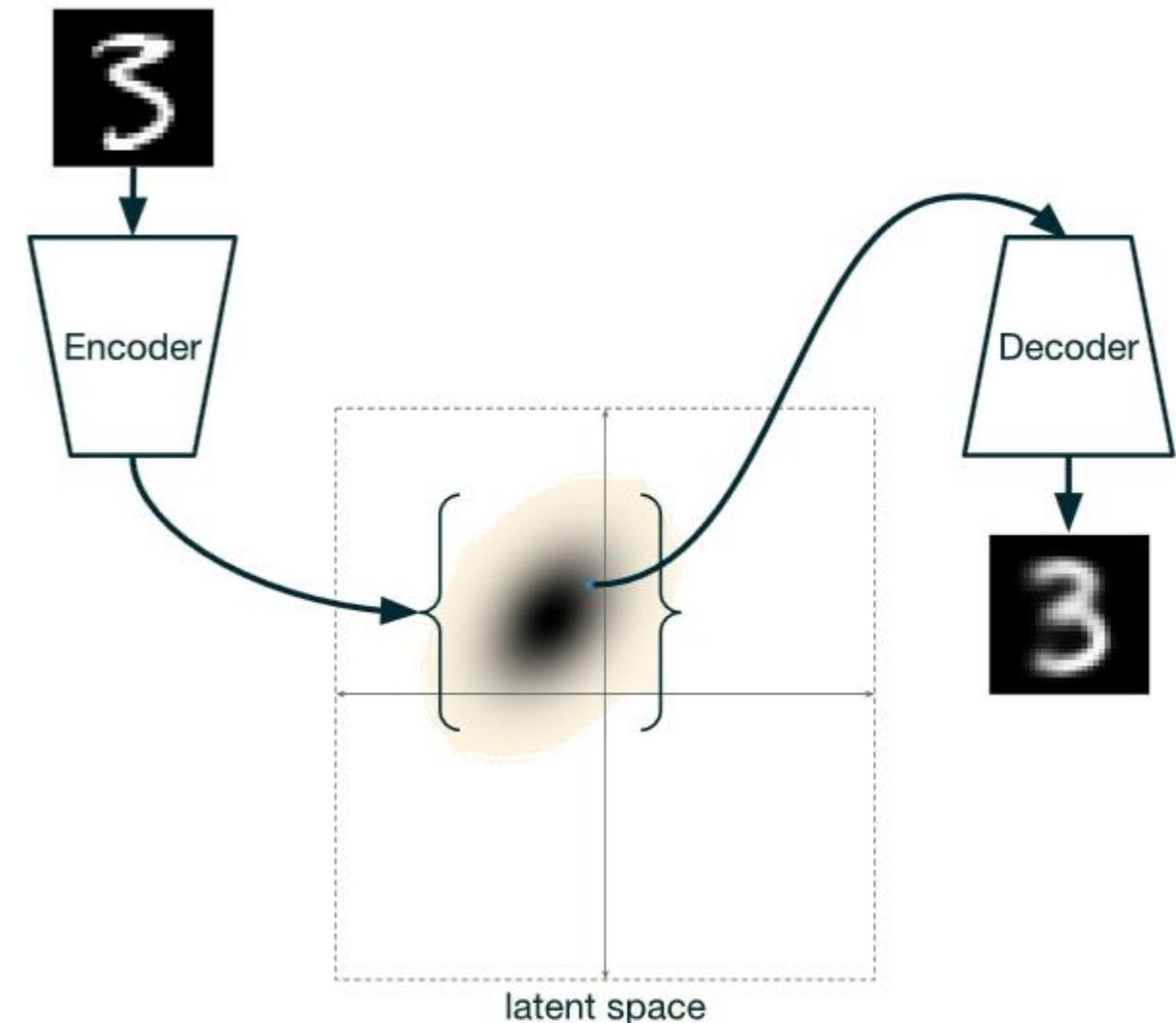
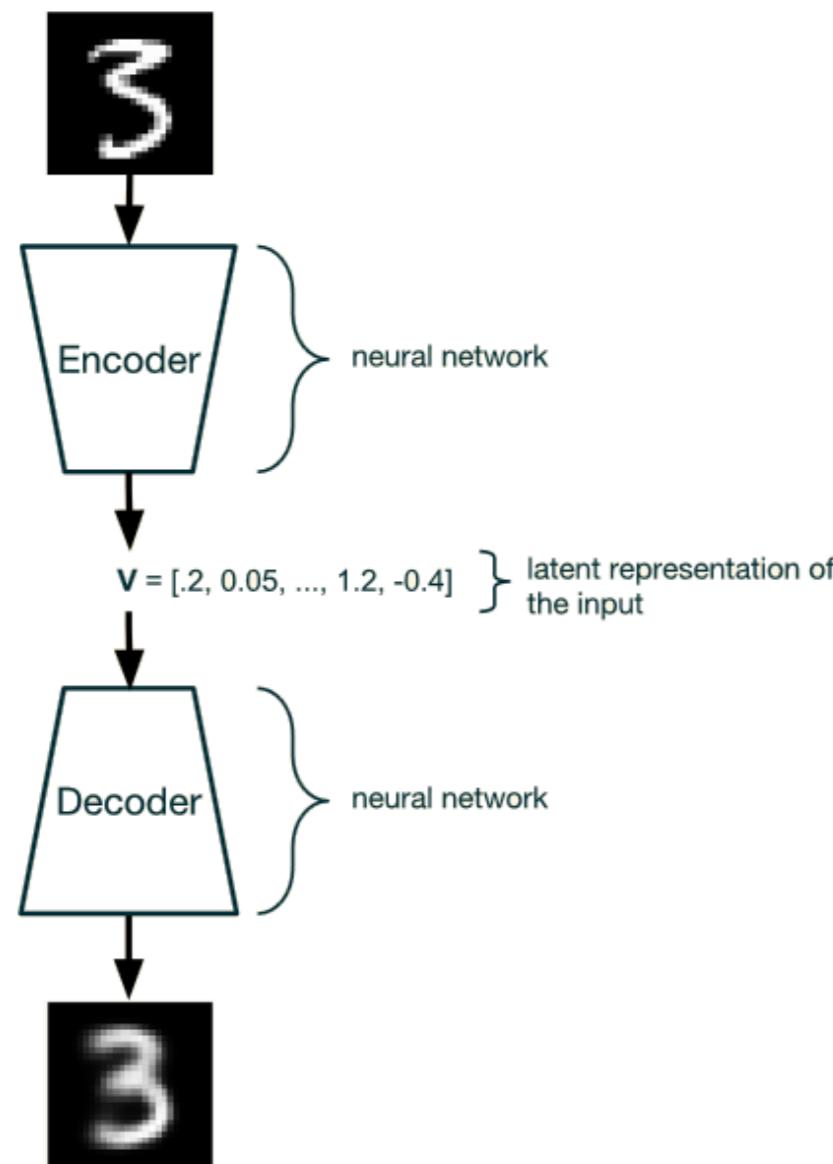
## Variational Autoencoders (VAE)

- 1)  $\Rightarrow$  обучать автокодировщик так, чтобы скрытые переменные  $\sim$  какое-то распределение  
ex:  $z \sim \text{norm}(0, I)$  можно потом сэмплировать отсюда
- 2)  $\Rightarrow$  подавать в декодировщик не выход автокодировщика,  
а что-то из распределения с центром в выходе



т.к. плохо, когда  $z$  кластеризуются... это запоминание данных

## Variational Autoencoders (VAE)



<https://ijdykeman.github.io/ml/2016/12/21/cvae.html>

## Variational Autoencoders (VAE)

**Раньше:**

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

в авторегрессионных моделях

$$L \rightarrow \max_{\theta}$$

**Теперь: через скрытую переменную**

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x | z)dz$$

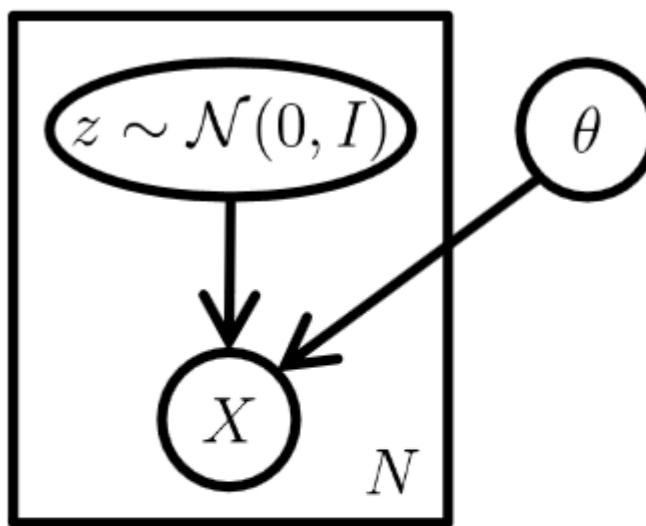


Figure 1: The standard VAE model represented as a graphical model. Note the conspicuous lack of any structure or even an “encoder” pathway: it is possible to sample from the model without any input. Here, the rectangle is “plate notation” meaning that we can sample from  $z$  and  $X$   $N$  times while the model parameters  $\theta$  remain fixed.

<https://arxiv.org/pdf/1606.05908.pdf>

## Variational Autoencoders (VAE)

$$p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x | z) dz$$

**нельзя напрямую оптимизировать**  $\Rightarrow$  **оптимизируем оценку правдоподобия**

**Напрашивается**

$$p_{\theta}(x) \approx \frac{1}{m} \sum_{t=1}^m p_{\theta}(x | z_t)$$

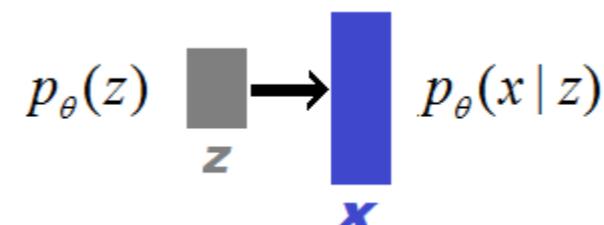
**но нужно много**  $\{z_t\}$ , **для большинства м.б.**  $p_{\theta}(x | z_t) \approx 0$

**практически невычислимо... можно ли генерировать**  $\{z_t\}$ :  $p_{\theta}(x | z_t) >> 0$ ?

**далше с  $p(x | z)$  иметь дело не будем...**

## Variational Autoencoders (VAE)

**«true prior»**



**«true conditional» /  
Likelihood**

**x – изображение, z – скрытая переменная  
семплируем z из  $p_\theta(z)$  и потом получаем  $p_\theta(x | z)$**

$$p_\theta(x) = \int p_\theta(z)p_\theta(x | z)dz$$

**$p_\theta(z) \sim$  пусть простое (нормальное)**

**$p_\theta(x | z) \sim$  сложное, т.к. изображение по вектору – генеративная модель  
вероятностный декодировщик (probabilistic decoder) – пусть нейросеть**

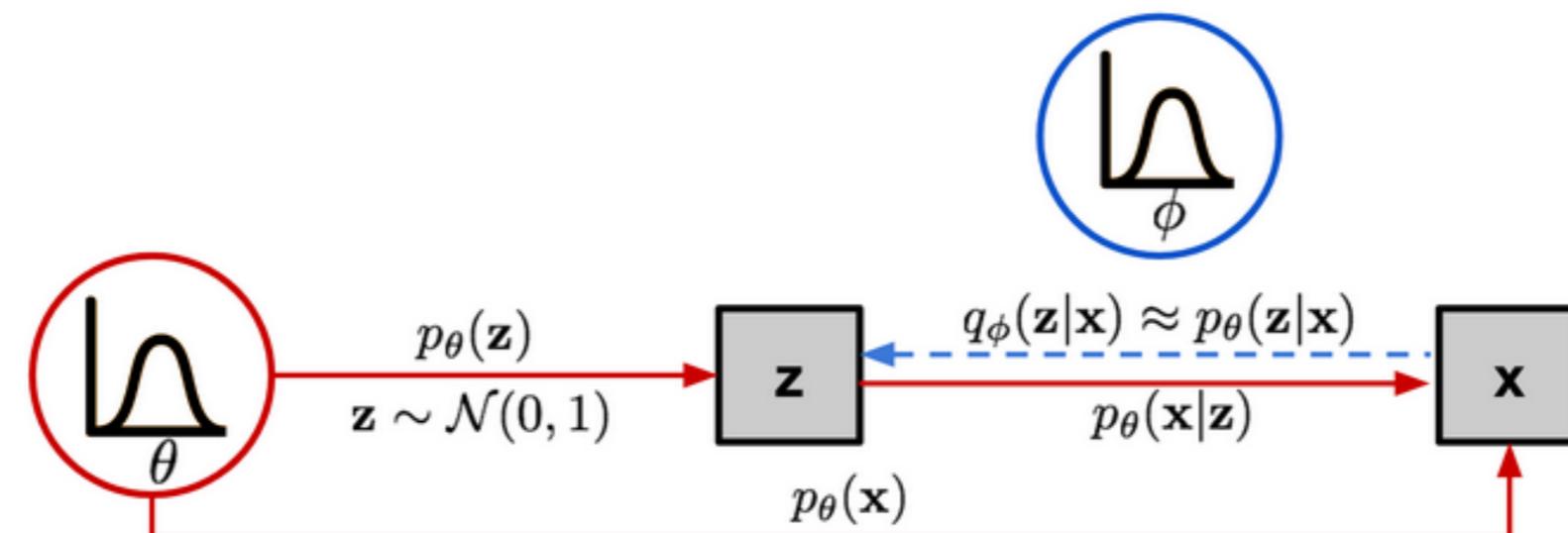
**Но невозможно вычислить  $p_\theta(x | z)$  для всех z (интегрирование невозможно)**

**Kingma D.P., Welling M. «Auto-Encoding Variational Bayes» // <https://arxiv.org/abs/1312.6114>**

## Variational Autoencoders (VAE)

**Решение: кроме декодировщика сделать вероятностный кодировщик (probabilistic encoder)  $q_\phi(z|x)$**   
**это позволит получить оценку на правдоподобие (заодно полезно для других задач как генератор признаков)**

**encoder (recognition/inference) network**  
**decoder (generation) network**



<https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html>

## Variational Bayesian Inference

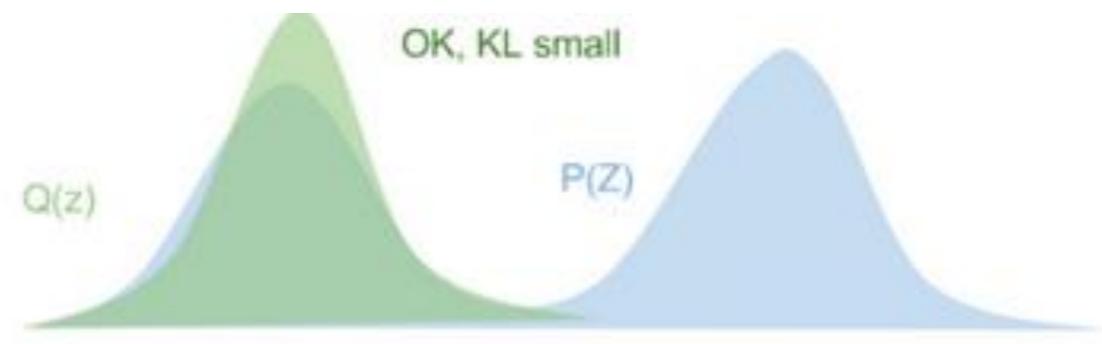
**хотим максимизировать:**

$$\begin{aligned}\log p_\theta(x) &= \int q_\phi(z | x) \log p_\theta(x) dz = \\&= \int q_\phi(z | x) \log \frac{p_\theta(x, z)}{p_\theta(z | x)} dz = \\&= \int q_\phi(z | x) \log \frac{p_\theta(x, z) q_\phi(z | x)}{q_\phi(z | x) p_\theta(z | x)} dz = \\&= \int q_\phi(z | x) \log \left( p_\theta(x | z) \frac{p_\theta(z)}{q_\phi(z | x)} \frac{q_\phi(z | x)}{p_\theta(z | x)} \right) dz = \\&= \int q_\phi(z | x) \log p_\theta(x | z) dz - \int q_\phi(z | x) \log \frac{q_\phi(z | x)}{p_\theta(z)} dz + \int q_\phi(z | x) \log \frac{q_\phi(z | x)}{p_\theta(z | x)} dz\end{aligned}$$

## Variational Bayesian Inference: evidence lower bound (ELBO)

$\int q_\phi(z x) \log p_\theta(x z) dz$	$\sim \mathbf{E}_{z \sim q} \log p_\theta(x z)$	<b>с помощью сэмплирования и кодировщика – это реконструкция данных (Neural Decoder Reconstruction Loss)</b>
$\int q_\phi(z x) \log \frac{q_\phi(z x)}{p_\theta(z)} dz$	$\sim D_{KL}(q_\phi(z x) \  p_\theta(z))$	<b>KL между гауссианой для декодировщика и априорным (можно посчитать) – это близость априорного и апостериорного распределений (регуляризация латентного представления)</b>
$\int q_\phi(z x) \log \frac{q_\phi(z x)}{p_\theta(z x)} dz$	$\sim D_{KL}(q_\phi(z x) \  p_\theta(z x))$	<b>уже говорили о невозможности вычисления, но <math>D_{KL} \geq 0</math> отбросим это слагаемое</b>

$\log p_\theta(x) \geq \text{ELBO}(x, \theta, \phi) = \mathbf{E}_{z \sim q} \log p_\theta(x|z) - D_{KL}(q_\phi(z|x) \| p_\theta(z)) \rightarrow \max$   
**аппроксимация истинного распределения вариационным**

**KL-расхождение****Напоминание...**Forward KL:  $D_{\text{KL}}(P \parallel Q)$ Reversed KL:  $D_{\text{KL}}(Q \parallel P)$ 

**Замечание: нет гарантии, что  $D_{\text{KL}}(q_\phi(z \mid x) \parallel p_\theta(z \mid x))$  будет близка к нулю**

**Ещё важно:**

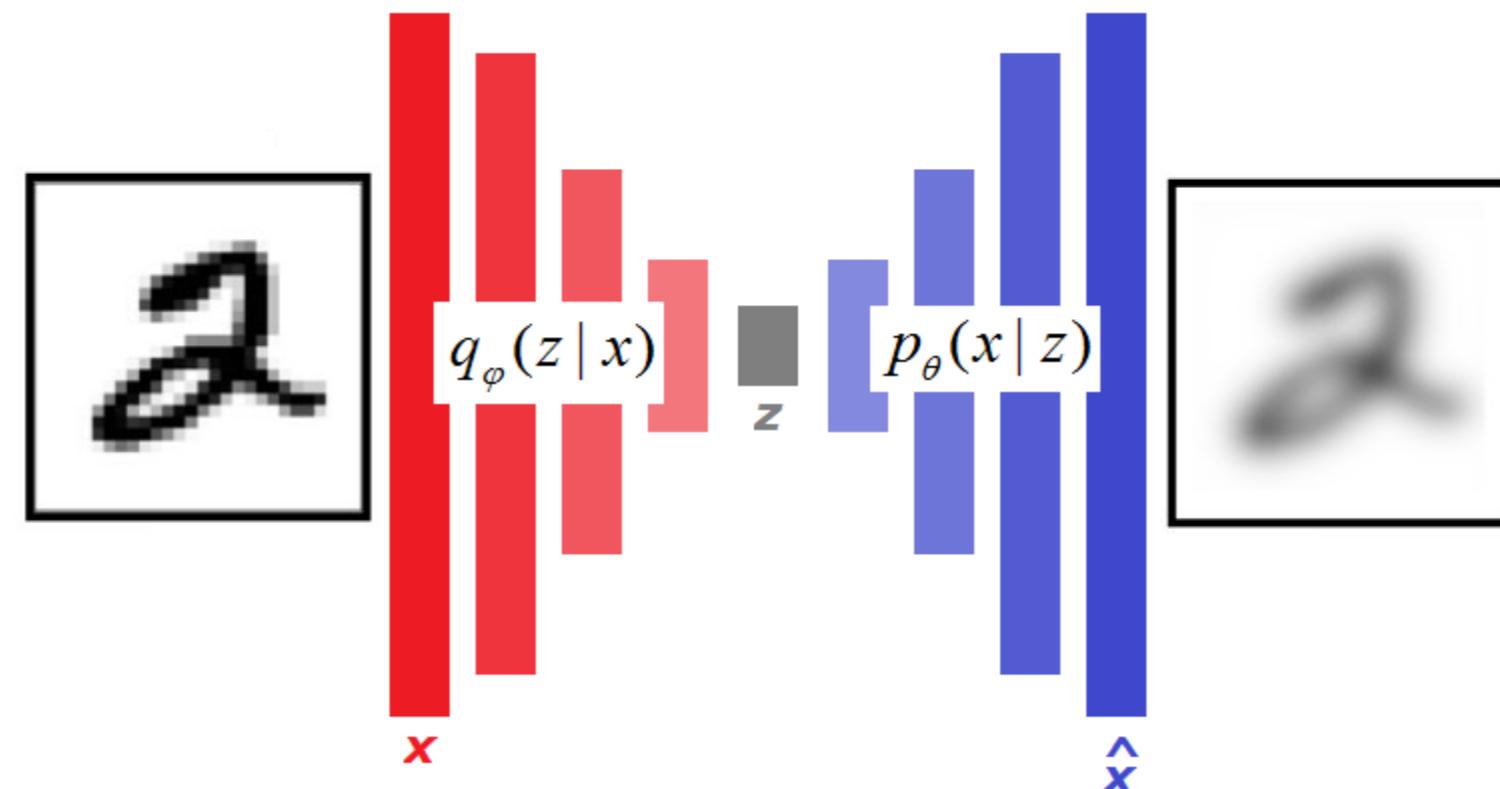
$$D_{\text{KL}}(\text{norm}(\mu_0, \Sigma_0) \parallel \text{norm}(\mu_1, \Sigma_1)) = \frac{1}{2} \left( \text{tr}(\Sigma_1^{-1} \Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1} (\mu_1 - \mu_0) - k + \log \left( \frac{\det \Sigma_1}{\det \Sigma_0} \right) \right)$$

**явная простая формула для нормальных распределений**

<https://blog.evjang.com/2016/08/variational-bayes.html>

## Variational Bayesian Inference

кодировщик - декодировщик



Возьмём в качестве  $q_\phi(z|x)$  (а это моделирует DNN)  
нормальное распределение  $\text{norm}(z | \mu_z(x), \sigma_z(x))$

тут **reparametrization trick** – дальше

## Variational Autoencoders (VAE)

как оптимизировать?

$$\log p_\theta(x) \geq \mathbf{E}_{z \sim q} \log p_\theta(x | z) - D_{KL}(q_\phi(z | x) \| p_\theta(z)) \rightarrow \max$$

**взяли**  $q_\phi(z | x) = \text{norm}(z | \mu_z(x | \varphi), \sigma_z(x | \varphi))$   
**и**  $p_\theta(x) = \text{norm}(x | 0, I)$

**Если считать распределение нормальным:**

$$\mathbf{E}_{z \sim q} \log p_\theta(x | z) \propto \|x - f(z)\|^2$$

выход сети

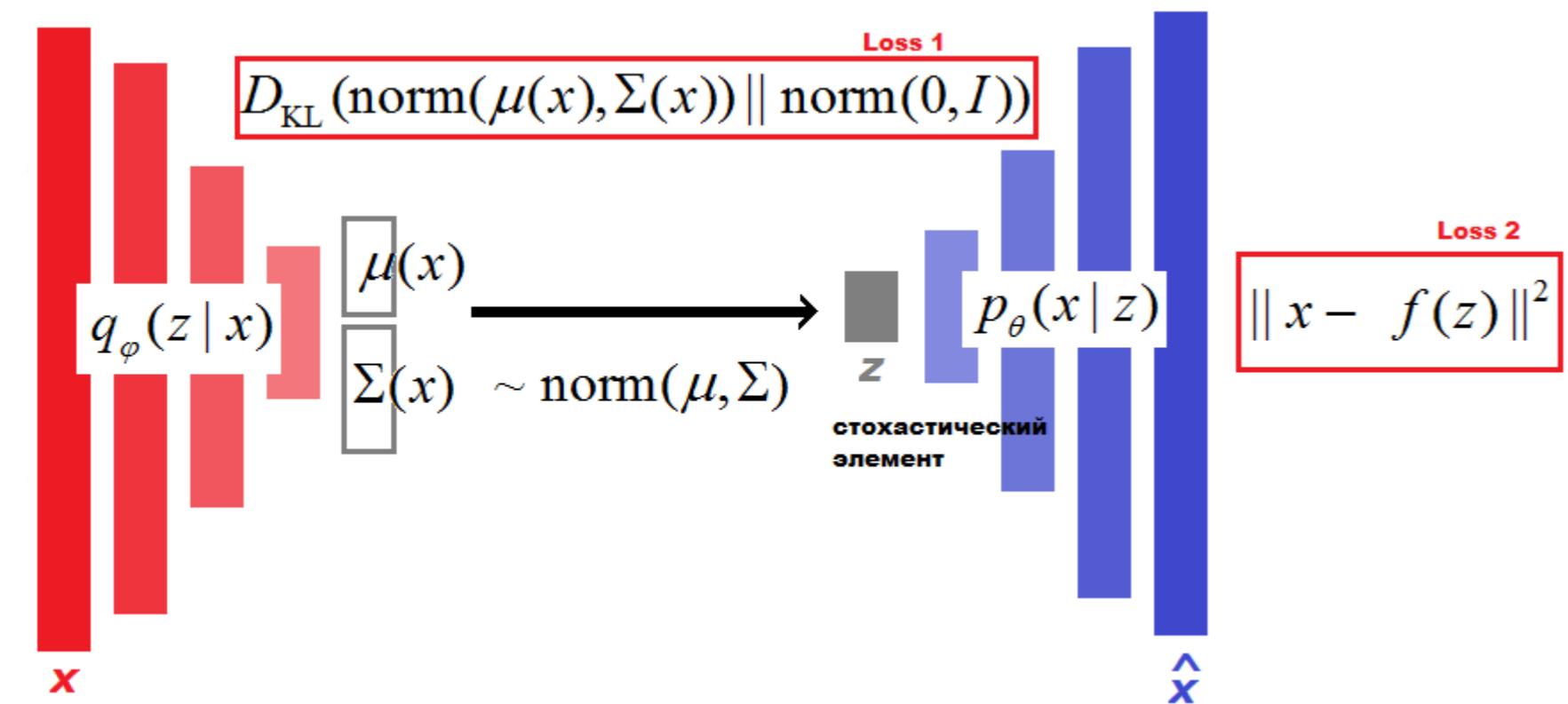
$$D_{KL}(q_\phi(z | x) \| p_\theta(z))$$

**можно эффективно посчитать  
(closed form solution)!**

**обучаем градиентным спуском НС**

**VAE = AE со специальной регуляризацией (в латентном пространстве)**

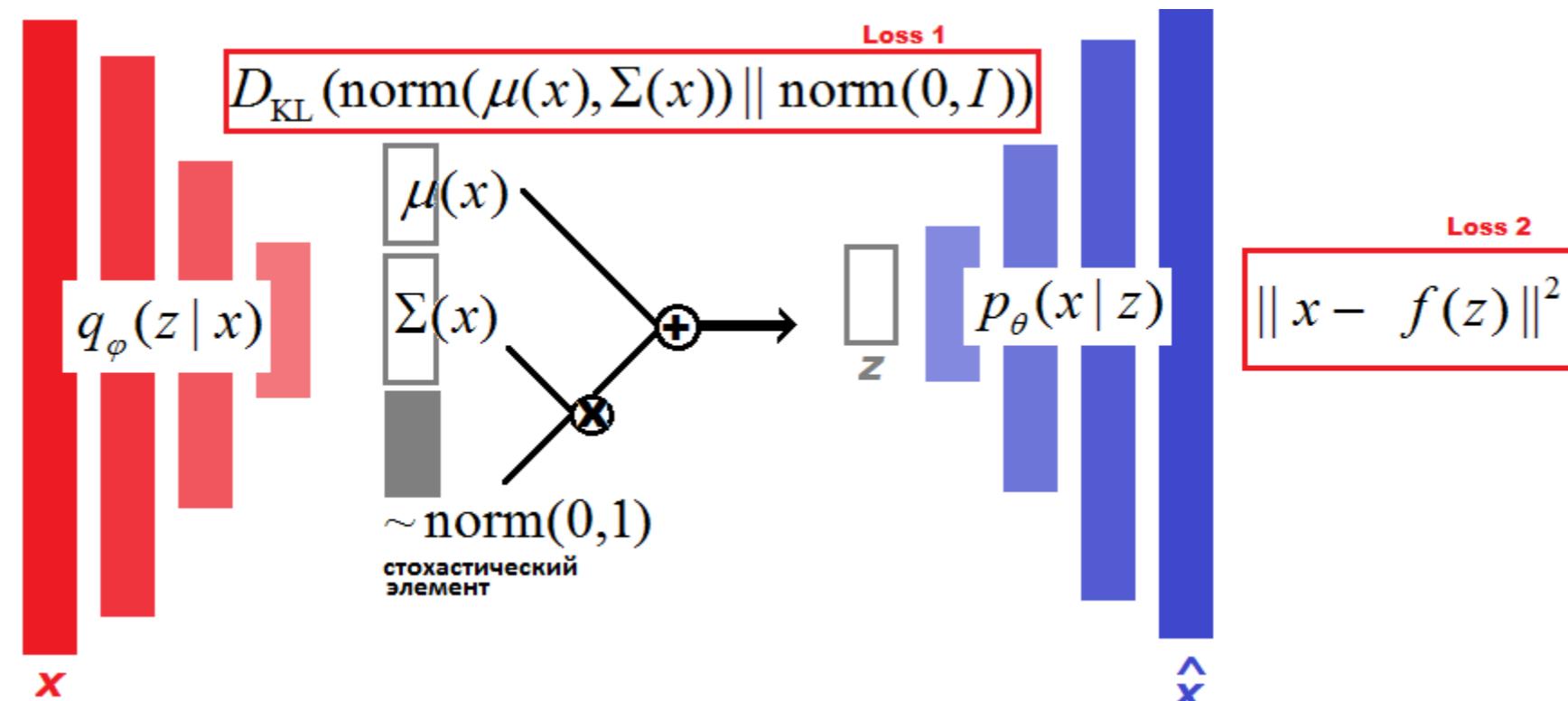
## Variational Autoencoders (VAE)



**Но как тут пропускать градиент?  
у нас полностью  
стохастический элемент!**  
т.к. делаем сэмплирование  
 $z \sim \text{norm}(z | \mu_z(x), \sigma_z(x))$

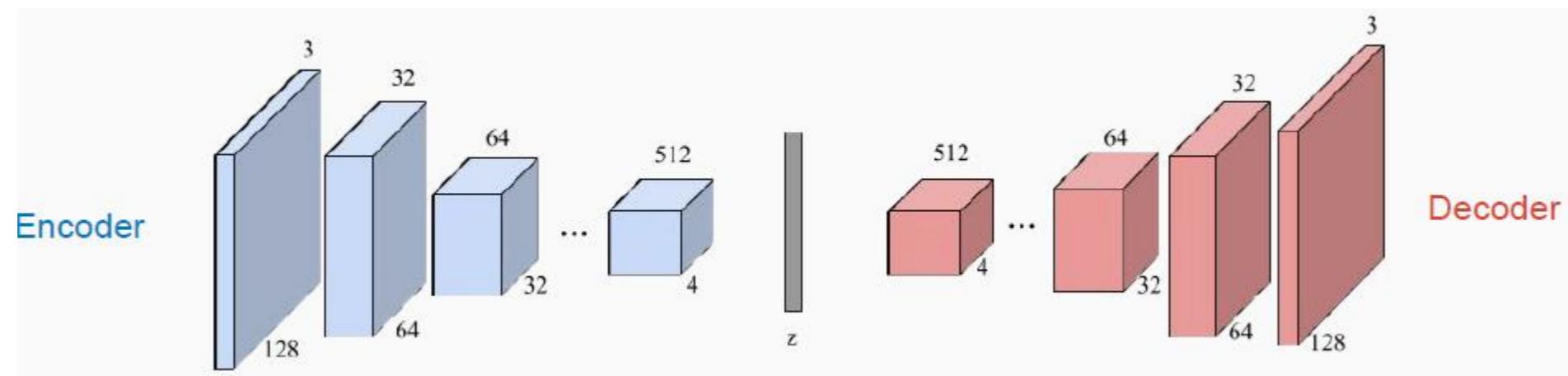
## Reparametrization trick: как делать ВР через распределение?

**DNN-кодировщик выдаёт вектор средних и матрицу ковариации (этого достаточно для моделирования распределения), будем сэмплировать  $\varepsilon$ !**



**градиент будет свободно распространяться через нестохастические элементы**

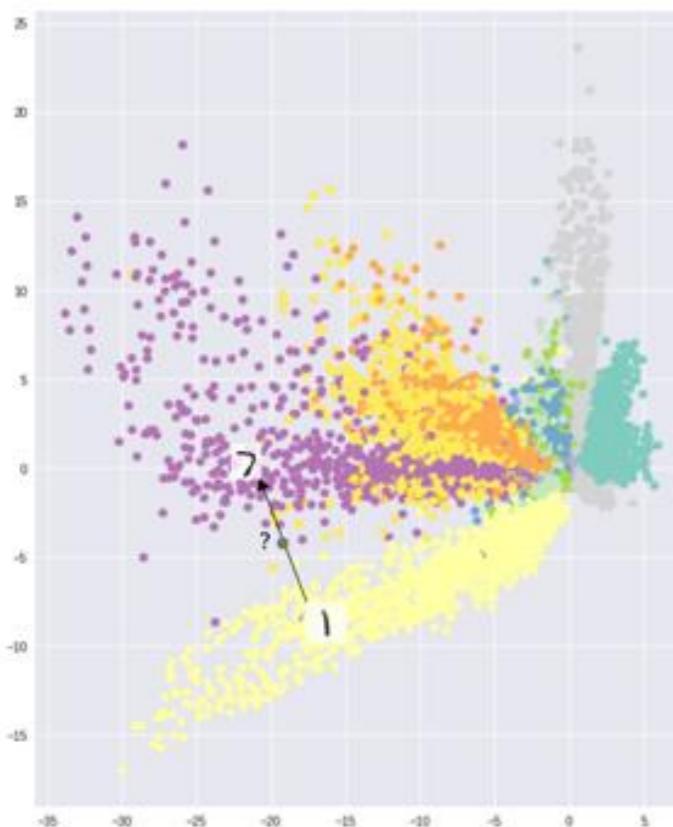
## Common VAE architecture



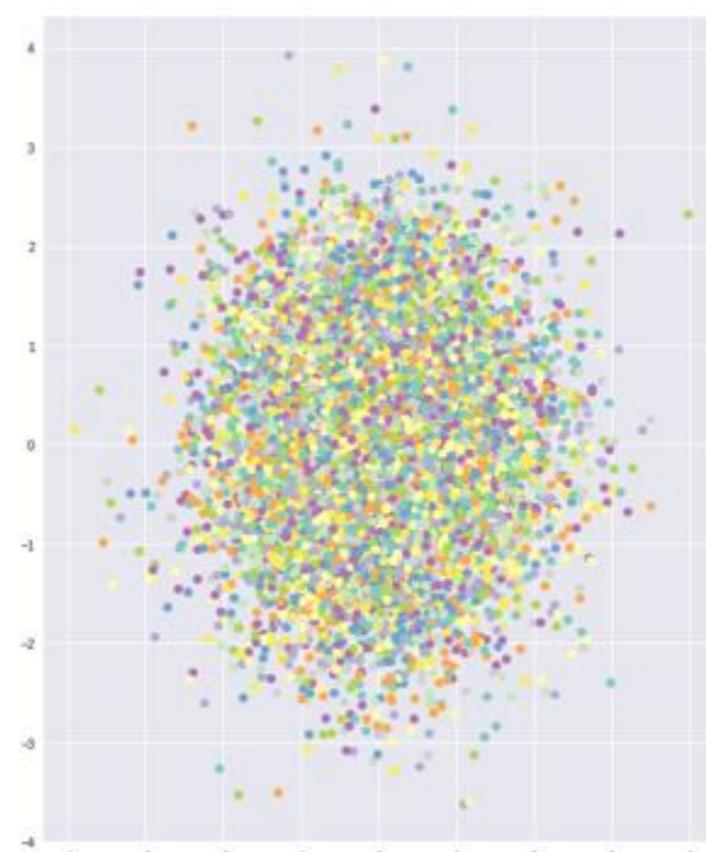
похожа на DCGAN...

## Variational Autoencoders (VAE)

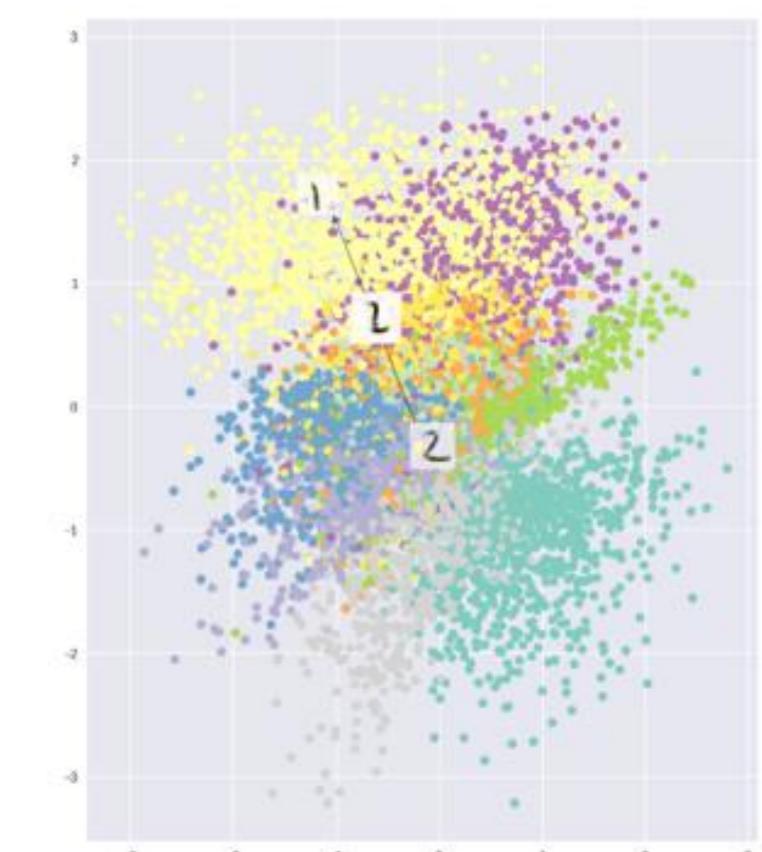
Only reconstruction loss



Only KL divergence

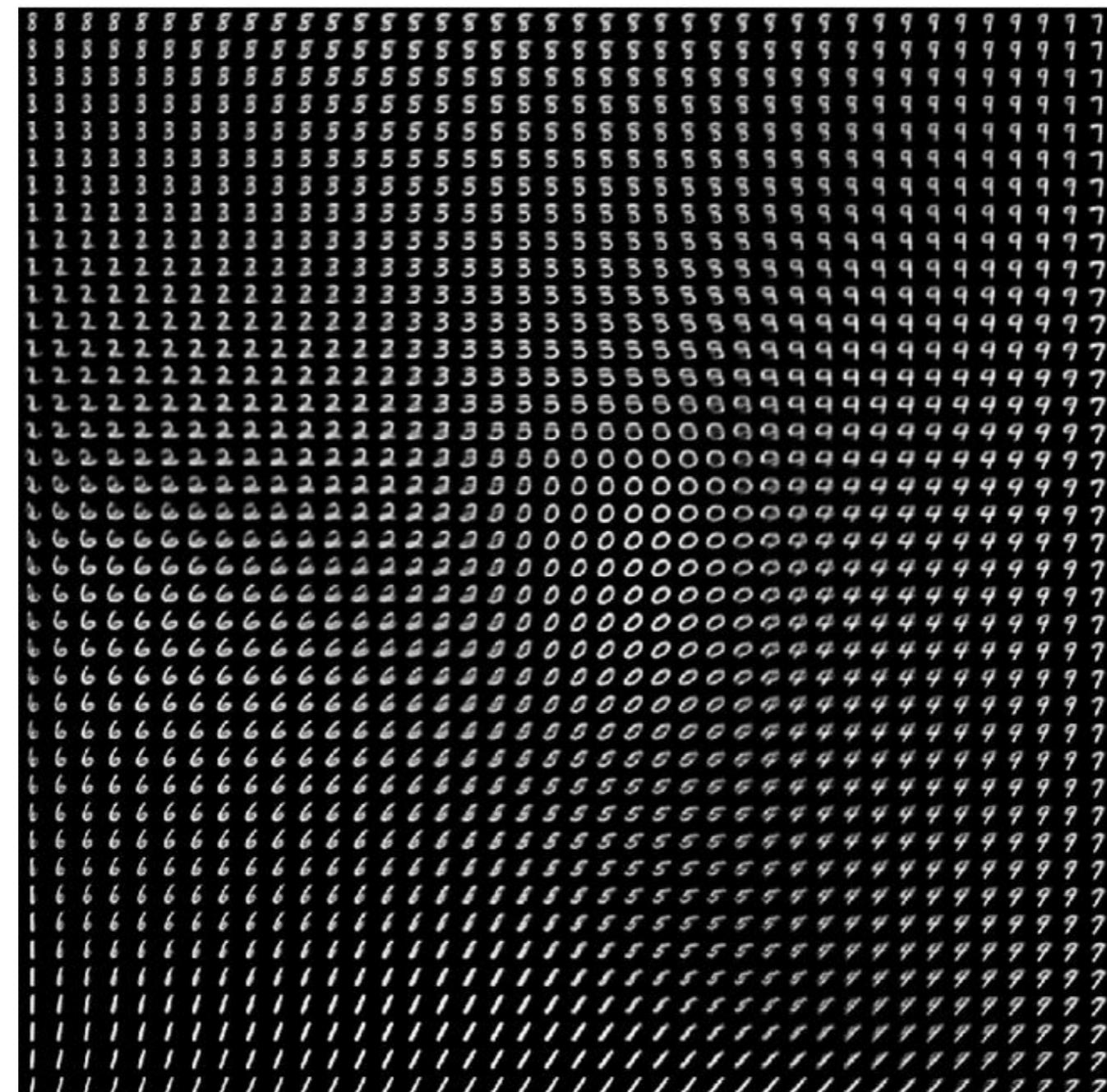


Combination

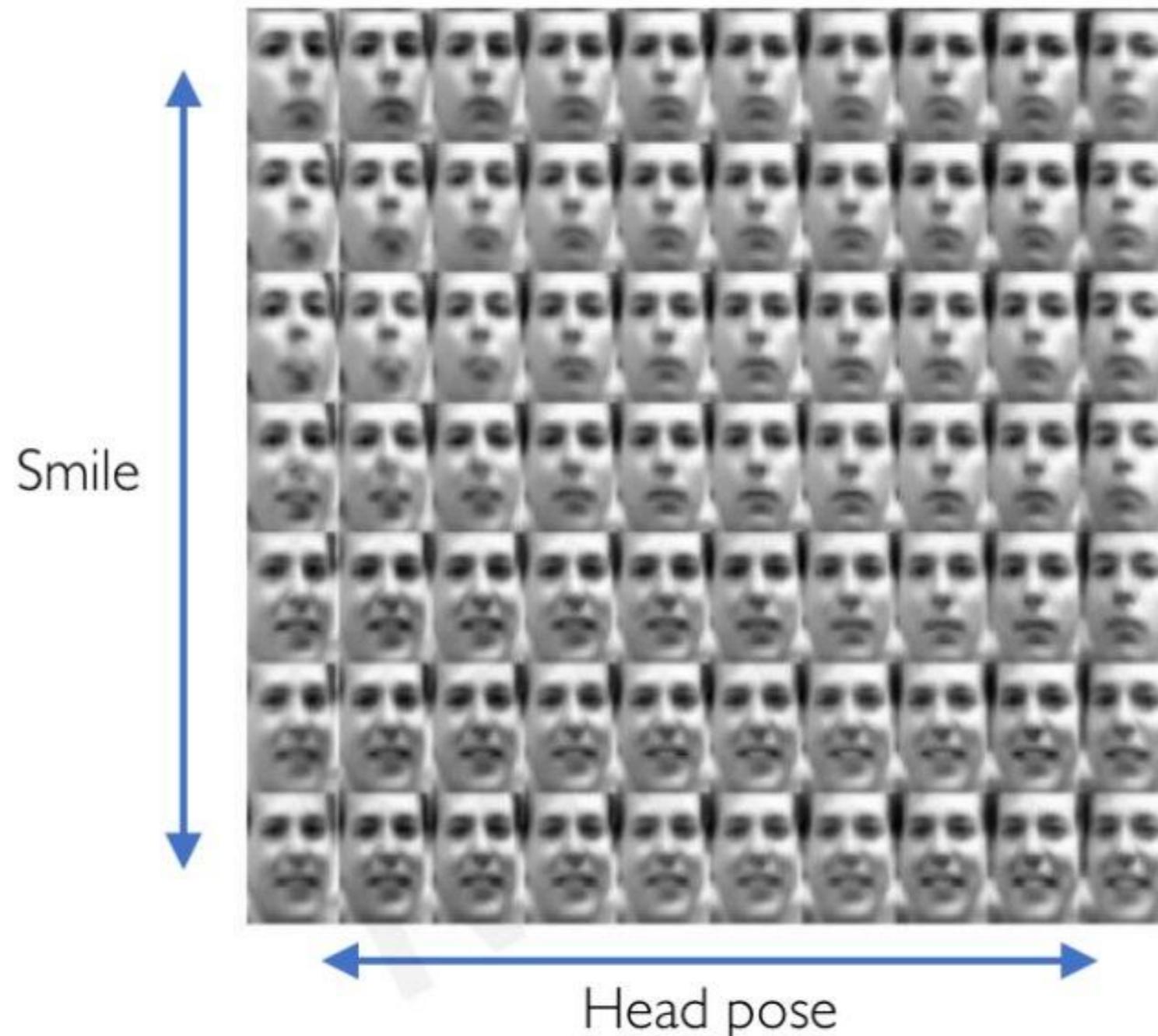


<https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf>

## Variational Autoencoders (VAE): вариация двух координат z



## Могут быть интерпретируемые латентные переменные



## Недостатки VAE

- сложно оценивать плотность
- результаты генерации не очень...
- могут быть проблемы при обучении

В VAE для последовательностей KL-слагаемое уходило в ноль, а другое плохо оптимизировалось. Решение – добавление меняющихся весов:

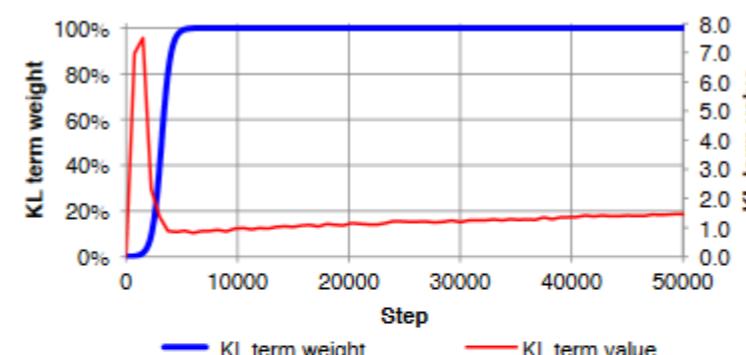


Figure 2: The weight of the KL divergence term of variational lower bound according to a typical sigmoid annealing schedule plotted alongside the (unweighted) value of the KL divergence term for our VAE on the Penn Treebank.

<https://arxiv.org/pdf/1511.06349.pdf>

## Функция ошибки в автокодировщиках – Feature Conceptual Loss

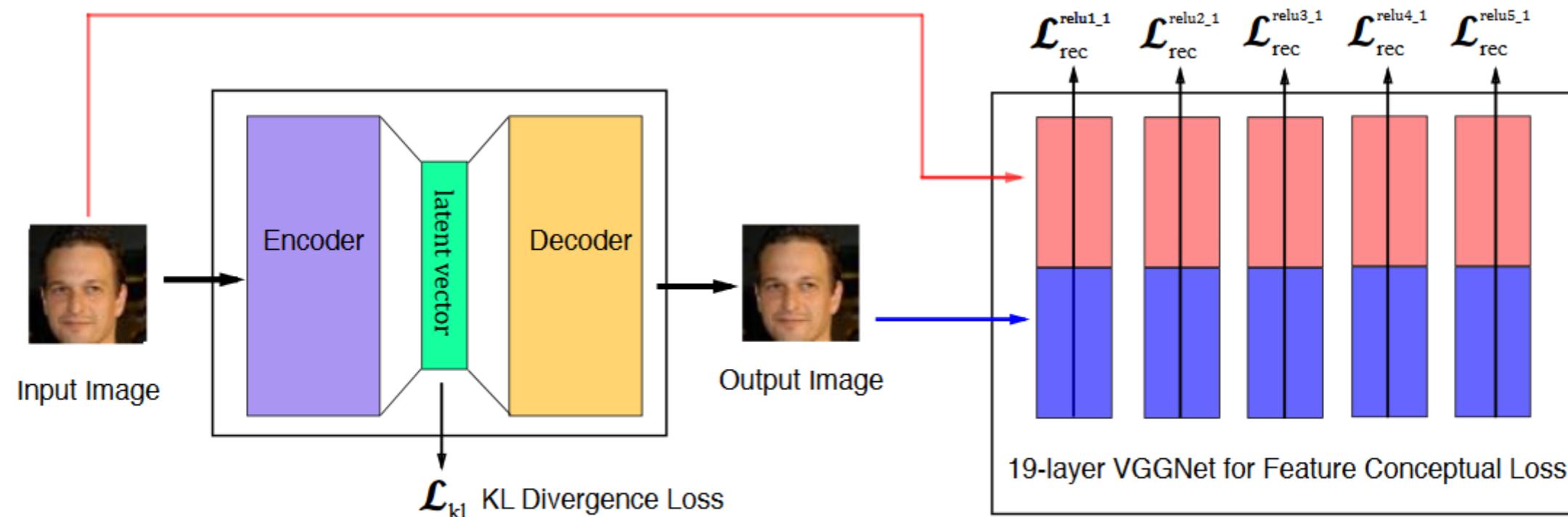


Figure 1. Model overview. The left is a deep CNN-based Variational Autoencoder and the right is a pretrained deep CNN used to compute feature perceptual loss.

**функция ошибки может быть хитрой, например, насколько похожи признаки изображений (вычисляются с помощью предобученной НС)**

Xianxu Hou, Linlin Shen, Ke Sun, Guoping Qiu «Deep Feature Consistent Variational Autoencoder» <https://arxiv.org/abs/1610.00291>

## Векторная арифметика

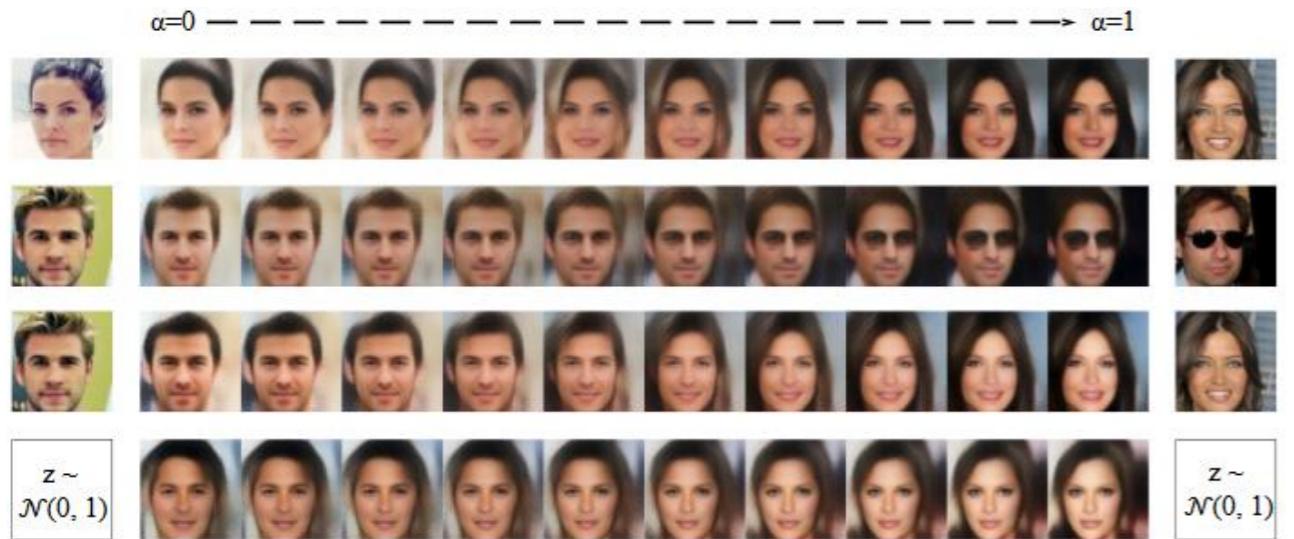


Figure 5. Linear interpolation for latent vector. Each row is the interpolation from left latent vector  $z_{left}$  to right latent vector  $z_{right}$ . e.g.  $(1 - \alpha)z_{left} + \alpha z_{right}$ . The first row is the transition from a non-smiling woman to a smiling woman, the second row is the transition from a man without eyeglass to a man with eyeglass, the third row is the transition from a man to a woman, and the last row is the transition between two fake faces decoded from  $z \sim \mathcal{N}(0, 1)$ .

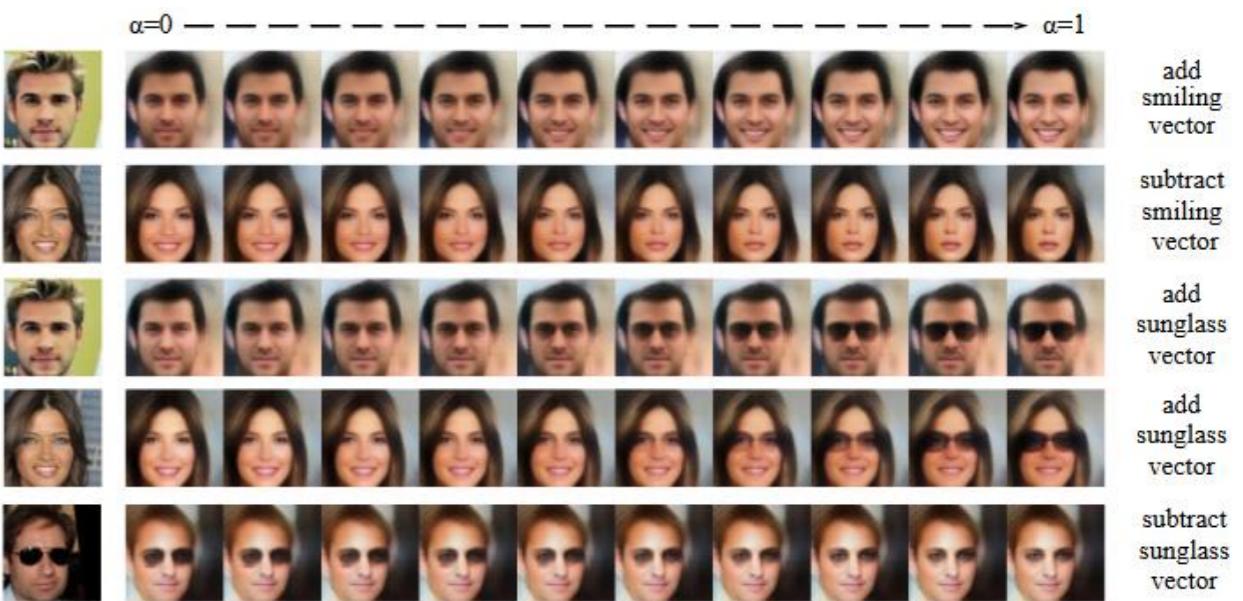


Figure 6. Vector arithmetic for visual attributes. Each row is the generated faces from latent vector  $z_{left}$  by adding or subtracting an attribute-specific vector, i.e.,  $z_{left} + \alpha z_{smiling}$ , where  $\alpha = 0, 0.1, \dots, 1$ . The first row is the transition by adding a smiling vector with a linear factor  $\alpha$  from left to right, the second row is the transition by subtracting a smiling vector, the third and fourth row are the results by adding a eyeglass vector to the latent representation for a man and women, and the last row shows results by subtracting an eyeglass vector.

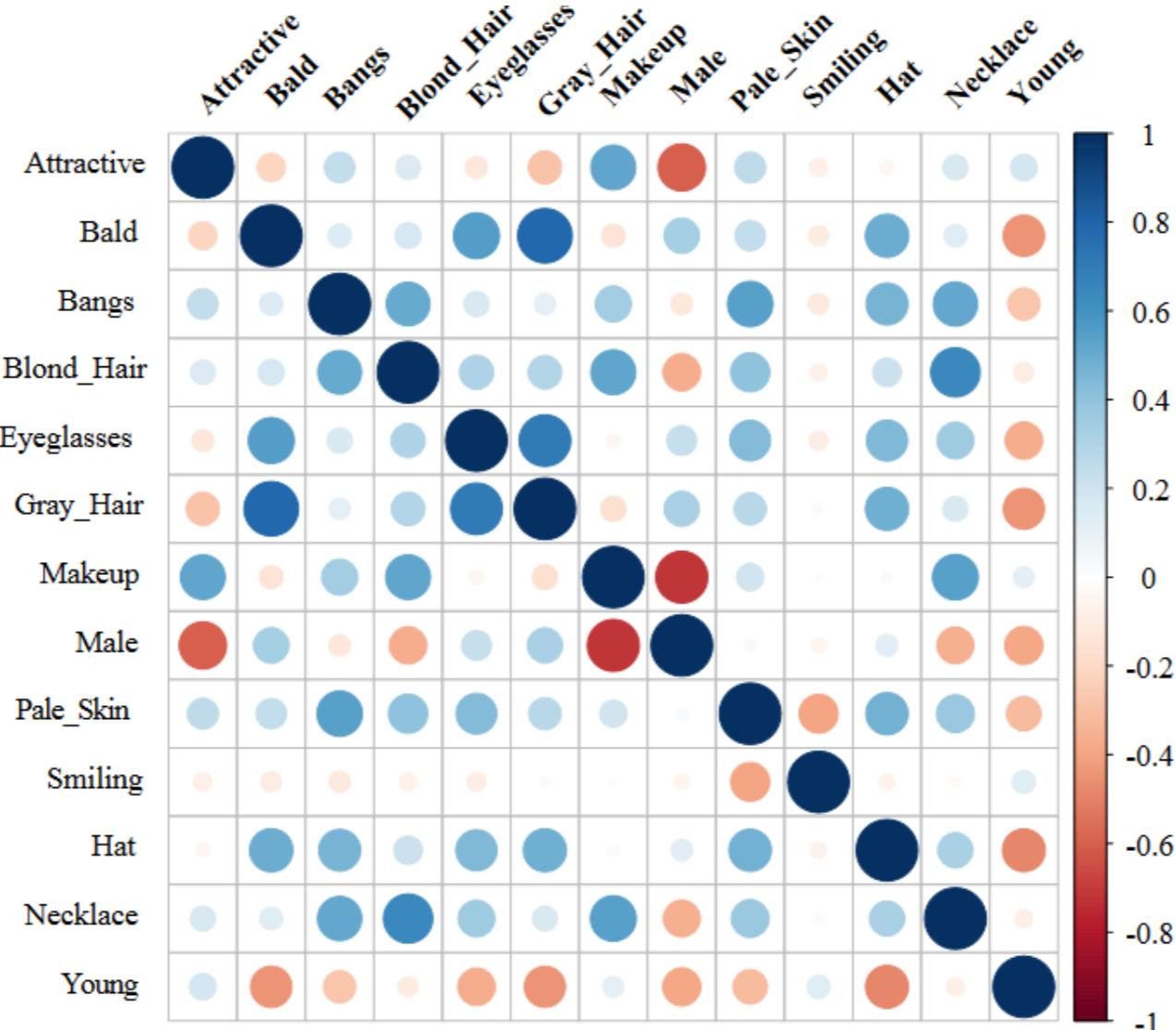
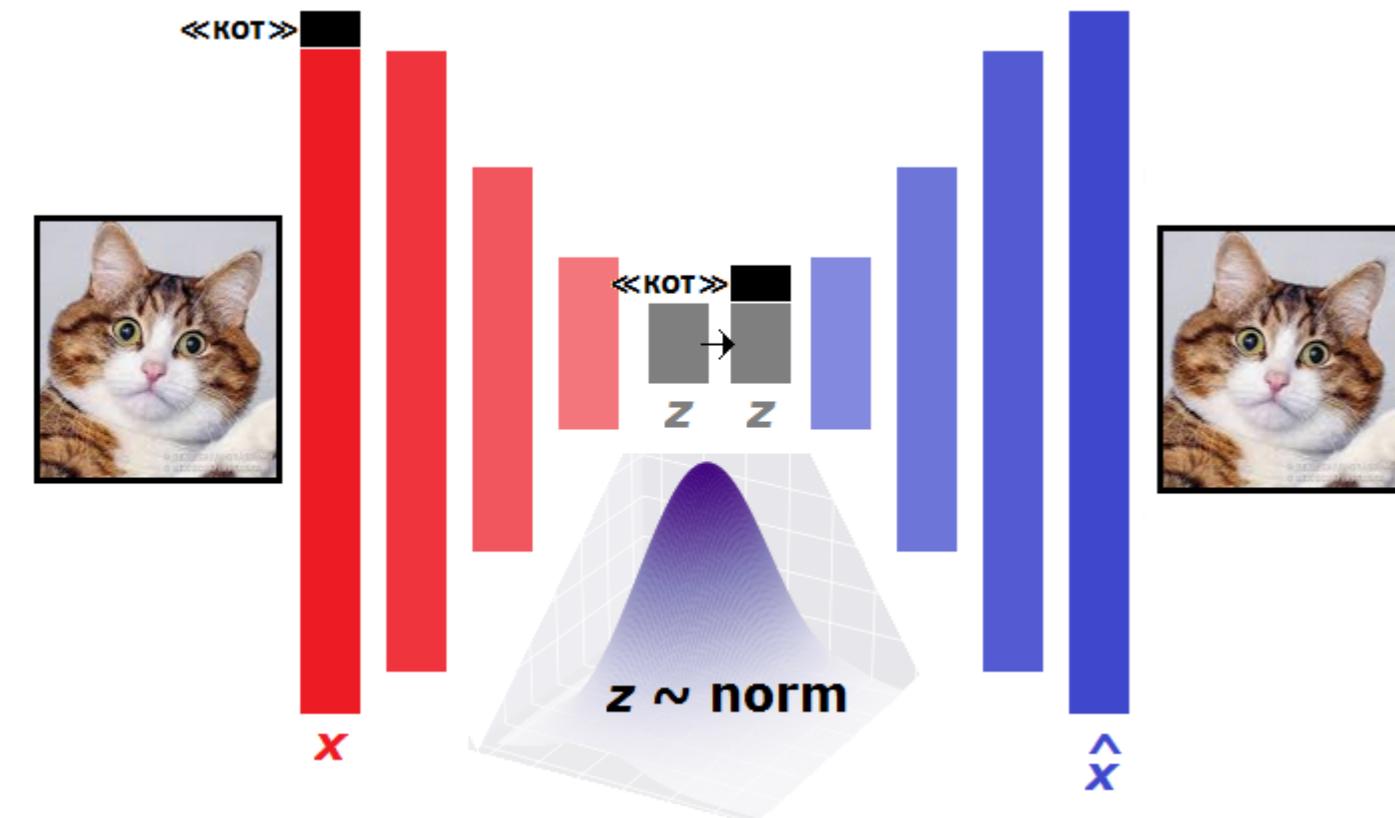


Figure 7. Diagram for the correlation between selected facial attribute-specific vectors. The blue indicates positive correlation, while red represents negative correlation, and the color shade and size of the circle represent the strength of the correlation.

## Conditional VAE (CVAE)



**В обычном VAE научились генерировать объекты из обучения...**

**А если мы хотим сгенерировать конкретный объект?**

Кстати, тогда можно не запоминать некоторую информацию в латентном пространстве

**Если есть какие-нибудь дополнительные метки...**

$$p_{\theta}(x | z) \rightarrow p_{\theta}(x | z, y)$$

Xinchen Yan, Jimei Yang, Kihyuk Sohn, Honglak Lee «Attribute2Image: Conditional Image Generation from Visual Attributes» <https://arxiv.org/abs/1512.00570>

## disCVAE: генерация изображений по описанию

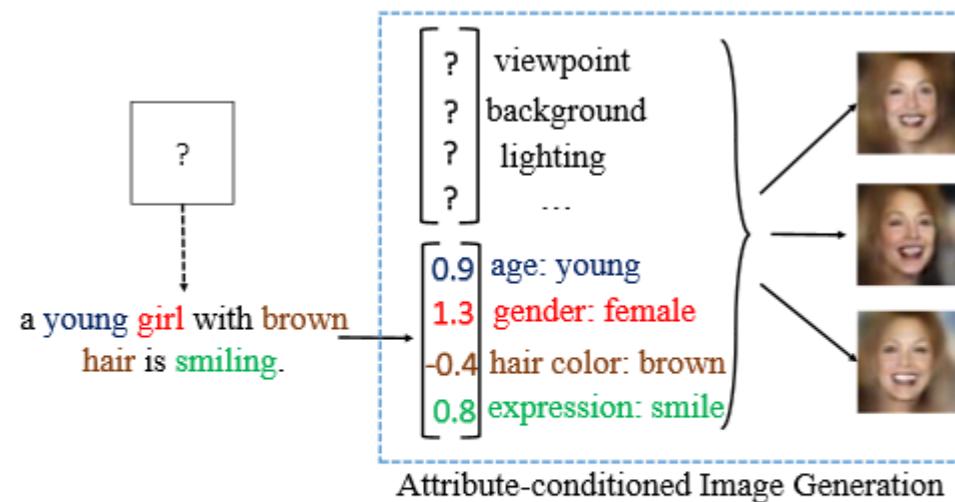


Fig. 1. An example that demonstrates the problem of conditioned image generation from visual attributes. We assume a vector of visual attributes is extracted from a natural language description, and then this attribute vector is combined with learned latent factors to generate diverse image samples.

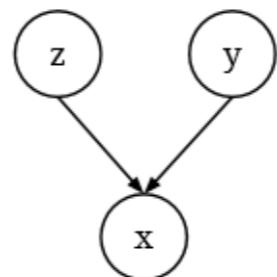
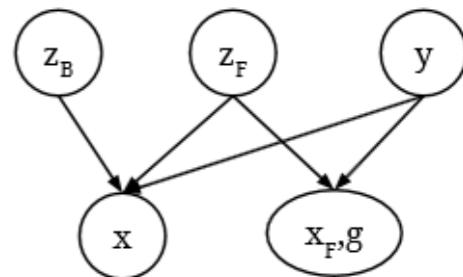
(a) CVAE:  $p_0(x|y, z)$ (b) disCVAE:  $p_0(x, x_F, g|y, z_F, z_B)$ 

Fig. 2. Graphical model representations of attribute-conditioned image generation models (a) without (CVAE) and (b) with (disCVAE) disentangled latent space.

**«disentangled representation»**

**отдельно компонента объекта и фона**

$$z = [z_F, z_B]$$

**обуславливается только объект**

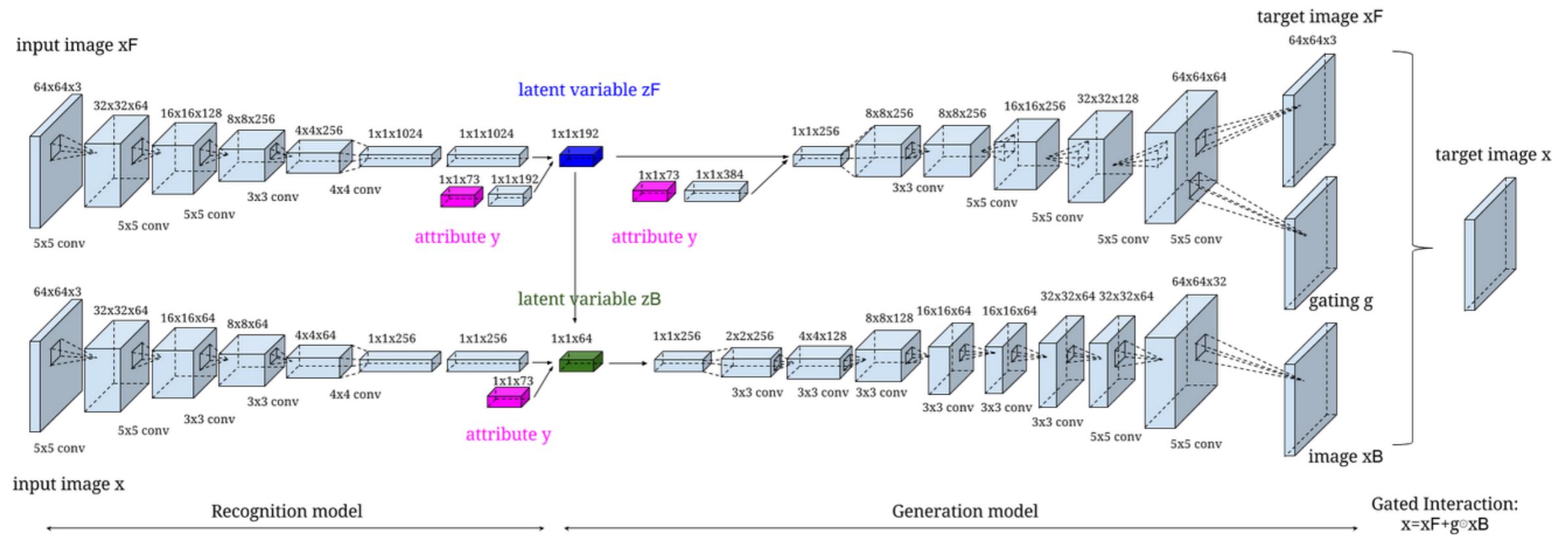
**изображение можно представить как «объект» + «задний план»**

$$x = x_F \odot (1 - g) + x_B \odot g$$

**с практической точки зрения лучше**

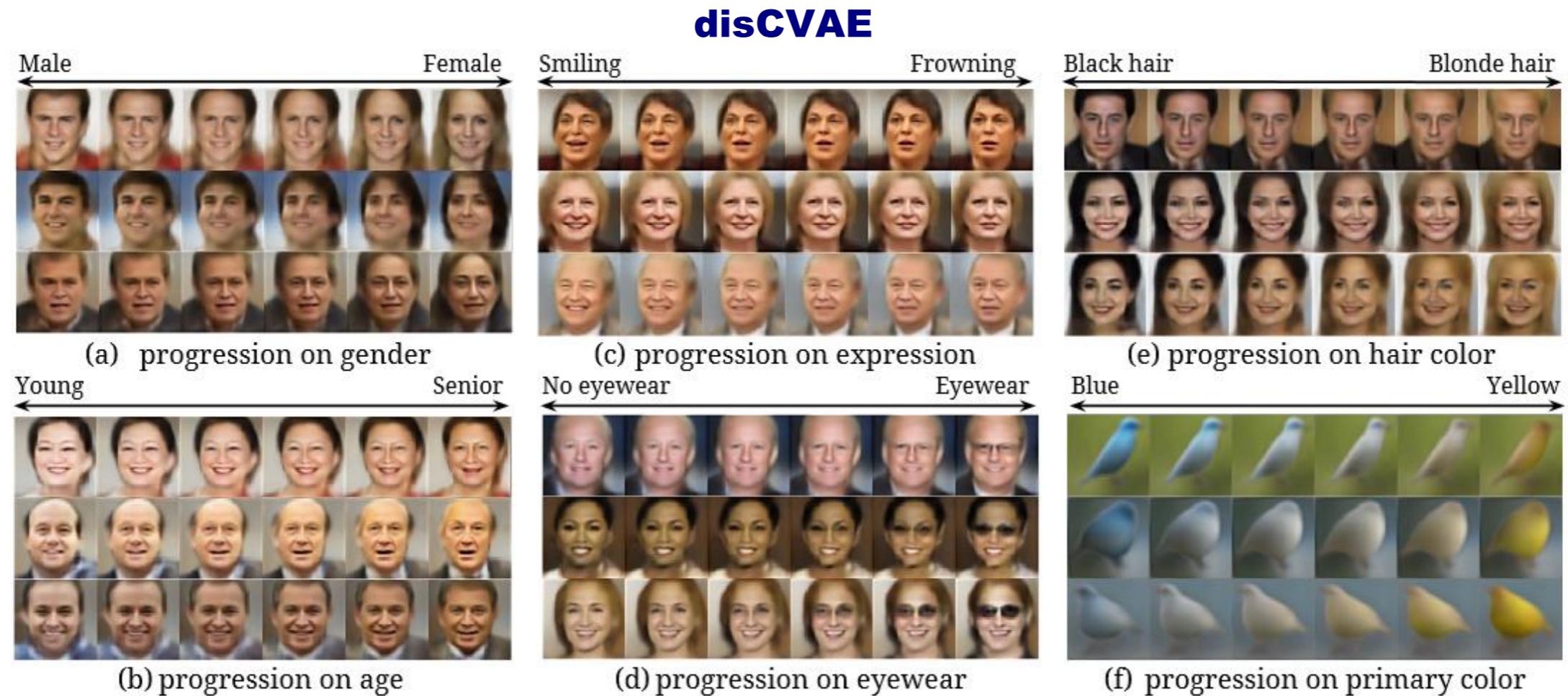
$$x = x_F + x_B \odot g$$

## disCVAE



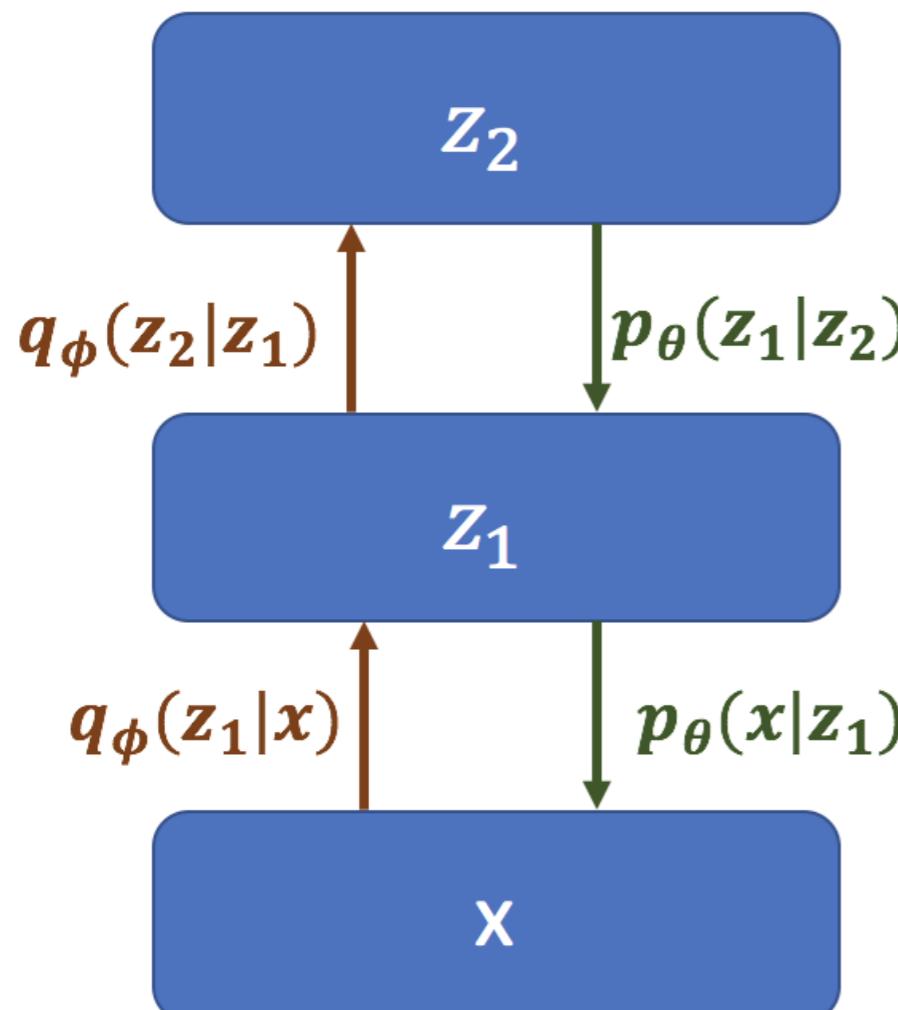
1. Sample foreground and background latent variables  $z_F \sim p(z_F)$ ,  $z_B \sim p(z_B)$ ;
2. Given  $y$  and  $z_F$ , generate foreground layer  $x_F \sim \mathcal{N}(\mu_{\theta_F}(y, z_F), \sigma_0^2 I_{N_x})$  and gating layer  $g \sim Bernoulli(s_{\theta_g}(y, z_F))$ ; here,  $\sigma_0$  is a constant. The background layer (which correspond to  $x_B$ ) is implicitly computed as  $\mu_{\theta_B}(z_B)$ .
3. Synthesize an image  $x \sim \mathcal{N}(\mu_{\theta}(y, z_F, z_B), \sigma_0^2 I_{N_x})$  where  $\mu_{\theta}(y, z_F, z_B) = \mu_{\theta_F}(y, z_F) + s_{\theta_g}(y, z_F) \odot \mu_{\theta_B}(z_B)$ .

изображение  
представлялось в виде  
foreground image (F) +  
маска × background  
image (B)



**Fig. 4.** Attribute-conditioned image progression. The visualization is organized into six attribute groups (e.g., “gender”, “age”, “facial expression”, “eyewear”, “hair color” and “primary color (blue vs. yellow)”). Within each group, the images are generated from  $p_{\theta}(x|y, z)$  with  $z \sim \mathcal{N}(0, I)$  and  $y = [y_{\alpha}, y_{rest}]$ , where  $y_{\alpha} = (1 - \alpha) \cdot y_{min} + \alpha \cdot y_{max}$ . Here,  $y_{min}$  and  $y_{max}$  stands for the minimum and maximum attribute value respectively in the dataset along the corresponding dimension.

## Иерархические VAE (HVAE)



Попытка добавить глубины...  
стекинг VAE

+) лучше оптимизируется Evidence Lower Bound (ELBO)

-) была надежда, что будет «иерархия выучиваемых представлений» нет подтверждений?

<https://ermongroup.github.io/blog/hierarchy/>

## Ladder Variational Autoencoders (вариационный автокодировщик «стремянка»)

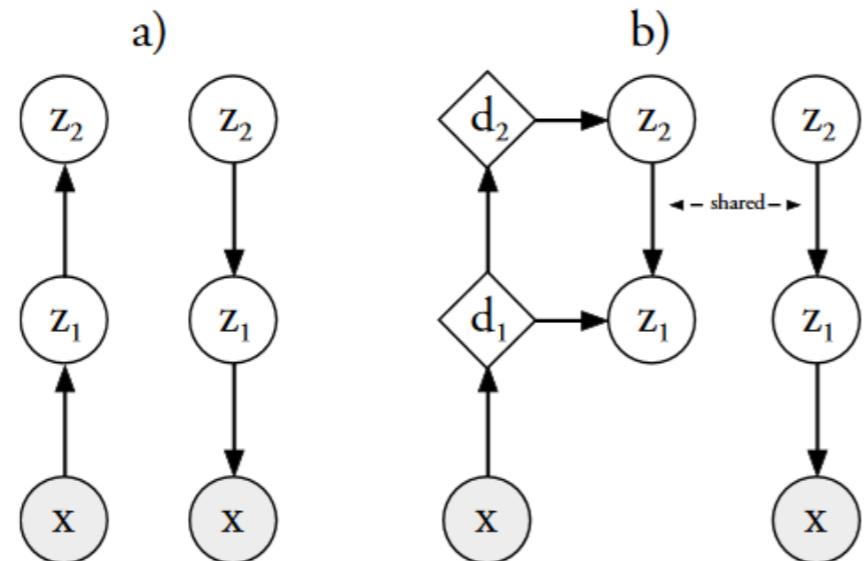


Figure 1: Inference (or encoder/recognition) and generative (or decoder) models for a) VAE and b) LVAE. Circles are stochastic variables and diamonds are deterministic variables.

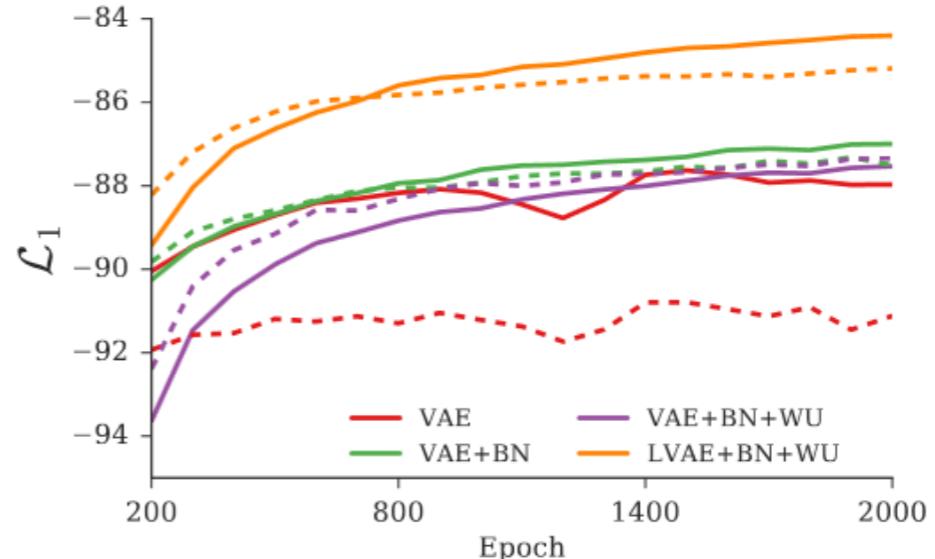


Figure 2: MNIST train (*full lines*) and test (*dashed lines*) set log-likelihood using one importance sample during training. The LVAE improves performance significantly over the regular VAE.

**отличие от VAE:  $d1$  и  $d2$  детерминированные (НС), не шум подаём на вход след. VAE  
латентные переменные разбиваются на слои:**

$$p_{\theta}(\mathbf{z}) = p_{\theta}(\mathbf{z}_L) \prod_{i=1}^{L-1} p_{\theta}(\mathbf{z}_i | \mathbf{z}_{i+1}) \quad (1)$$

$$p_{\theta}(\mathbf{z}_i | \mathbf{z}_{i+1}) = \mathcal{N}(\mathbf{z}_i | \mu_{p,i}(\mathbf{z}_{i+1}), \sigma_{p,i}^2(\mathbf{z}_{i+1})), \quad p_{\theta}(\mathbf{z}_L) = \mathcal{N}(\mathbf{z}_L | \mathbf{0}, \mathbf{I}) \quad (2)$$

$$p_{\theta}(\mathbf{x} | \mathbf{z}_1) = \mathcal{N}(\mathbf{x} | \mu_{p,0}(\mathbf{z}_1), \sigma_{p,0}^2(\mathbf{z}_1)) \text{ or } P_{\theta}(\mathbf{x} | \mathbf{z}_1) = \mathcal{B}(\mathbf{x} | \mu_{p,0}(\mathbf{z}_1)) \quad (3)$$

## Ladder Variational Autoencoders (вариационный автокодировщик «стремянка»)

**warm-up (WU) – регуляризатор с коэффициентом, который растёт от 0 до 1**

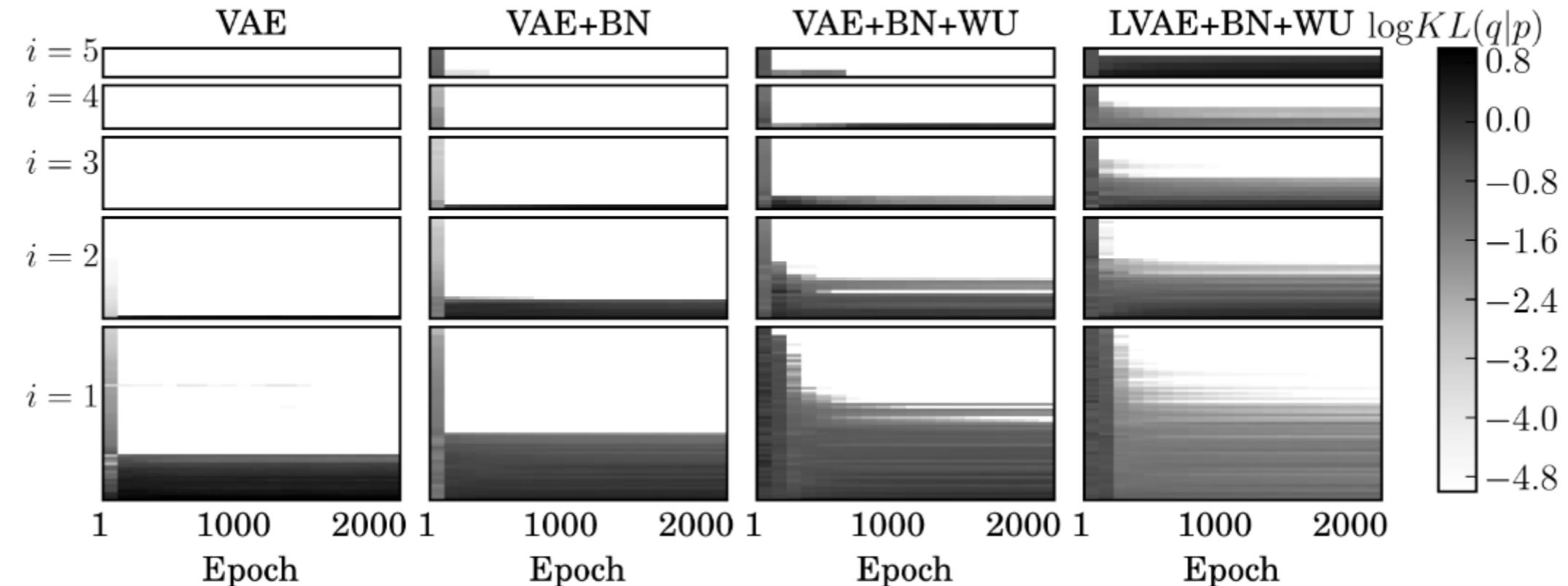


Figure 4:  $\log KL(q|p)$  for each latent unit is shown at different training epochs. Low  $KL$  (white) corresponds to an inactive unit. The units are sorted for visualization. It is clear that vanilla VAE cannot train the higher latent layers, while introducing batch normalization helps. Warm-up creates more active units early in training, some of which are then gradually pruned away during training, resulting in a more distributed final representation. Lastly, we see that the LVAE activates the highest number of units in each layer.

## Ladder Variational Autoencoders (вариационный автокодировщик «стремянка»)

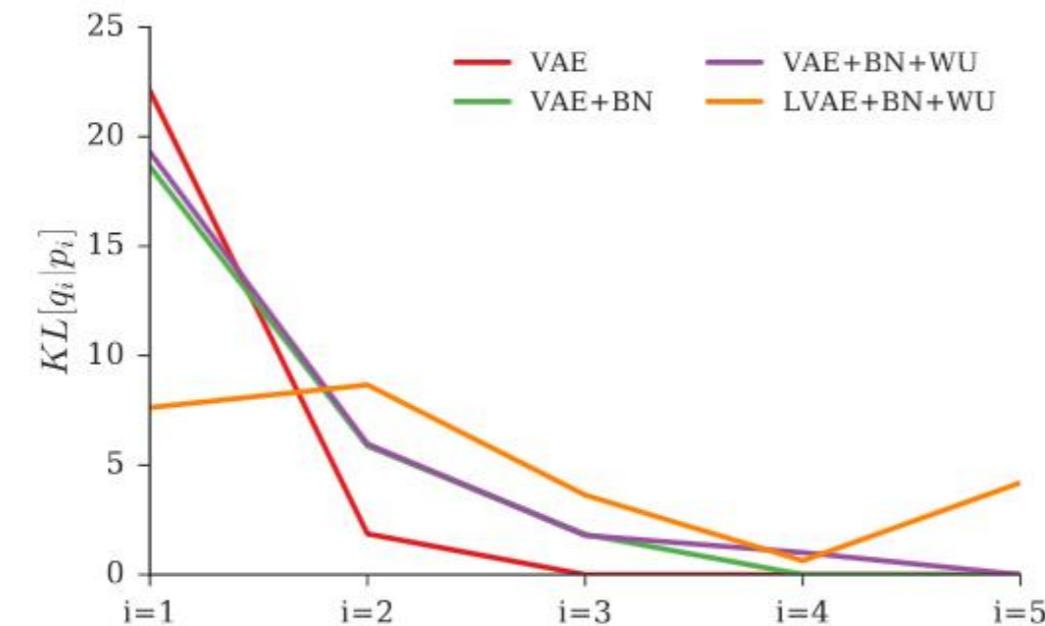


Figure 5: Layer-wise  $KL[q|p]$  divergence going from the lowest to the highest layers. In the VAE models the KL divergence is highest in the lowest layers whereas it is more distributed in the LVAE model

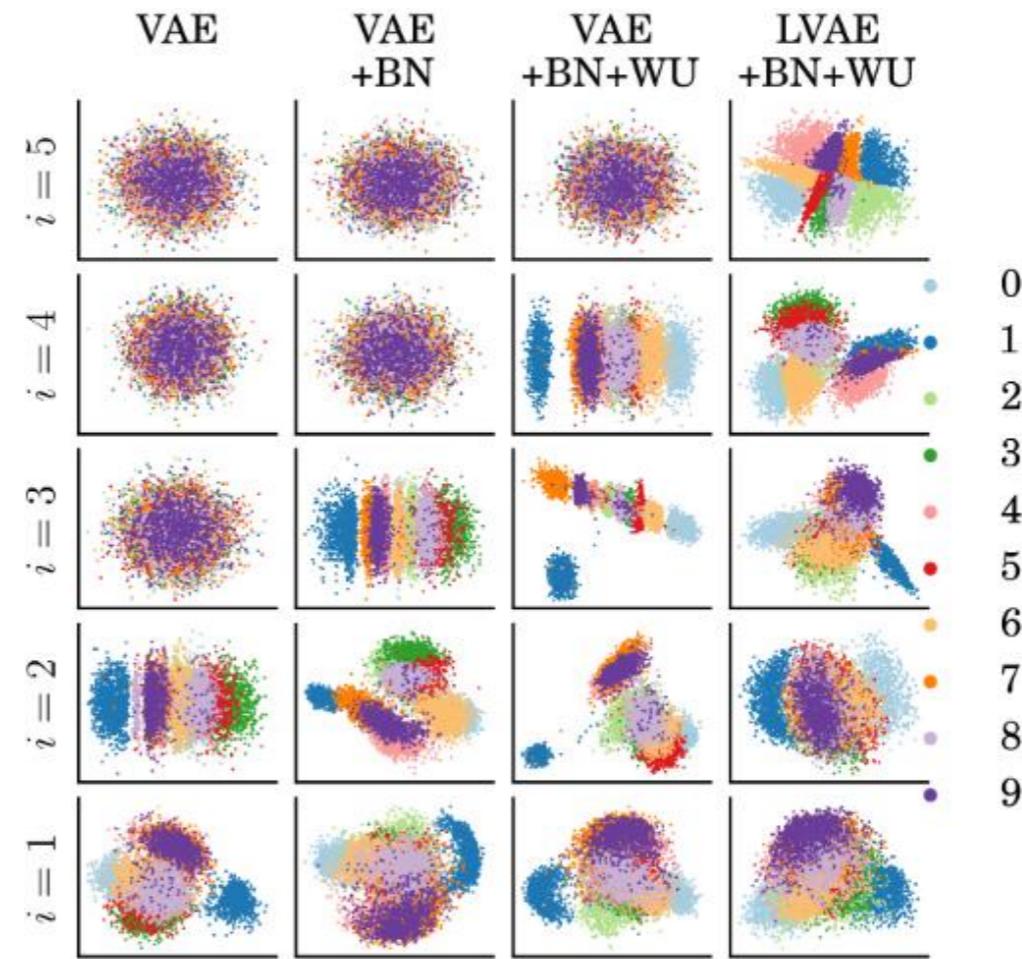


Figure 6: PCA-plots of samples from  $q(z_i|z_{i-1})$  for 5-layer VAE and LVAE models trained on MNIST. Color-coded according to true class label

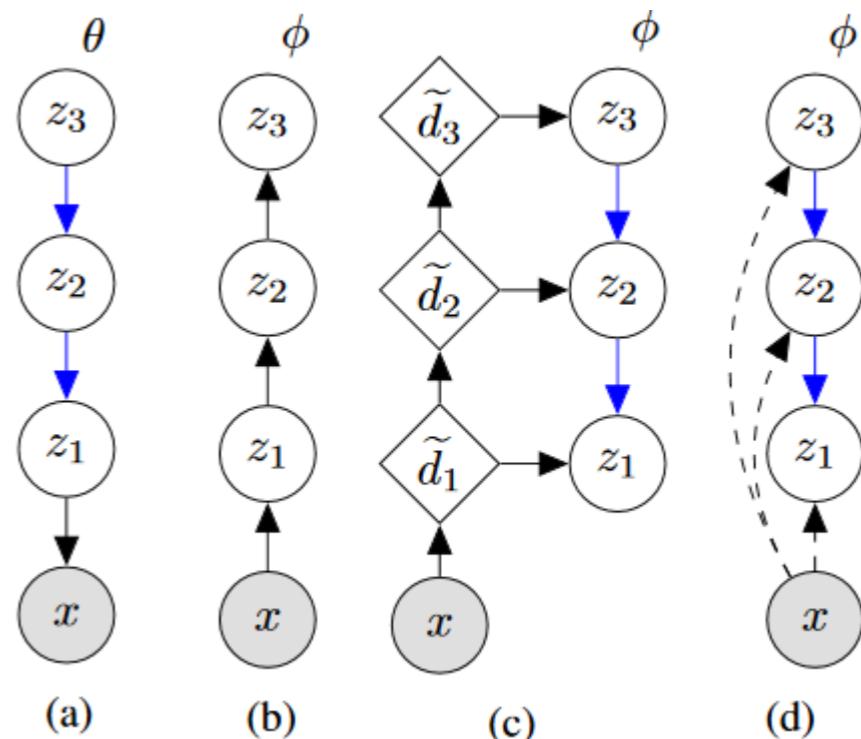
## Bidirectional-Inference Variational Autoencoder (BIVA)

**BIVA = LVAE +**

**1) a deterministic top-down path**

**2) apply a bidirectional inference network.**

Figure 5: (a) Generative model of a VAE/LVAE with  $L = 3$  stochastic variables, (b) VAE inference model, (c) LVAE inference model, and (d) skip connections among stochastic variables in the LVAE where dashed lines denote a skip-connection. Blue arrows indicate that there are shared parameters between the inference and generative model.



**синие стрелки – разделение параметров (одна и та же функция):**

$$p_{\theta}(z_i|z_{i-1}) = \mathcal{N}(z_i|\mu_{p,i}(z_{i-1}), \sigma_{p,i}^2(z_{i-1}))$$

Lars Maaløe «BIVA: A Very Deep Hierarchy of Latent Variables for Generative Modeling» //  
<https://arxiv.org/abs/1902.02102>

## Bidirectional-Inference Variational Autoencoder (BIVA)

**На каждом слое переменные расщепляются на 2 пути: BU (bottom-up) и TD (top-down)**

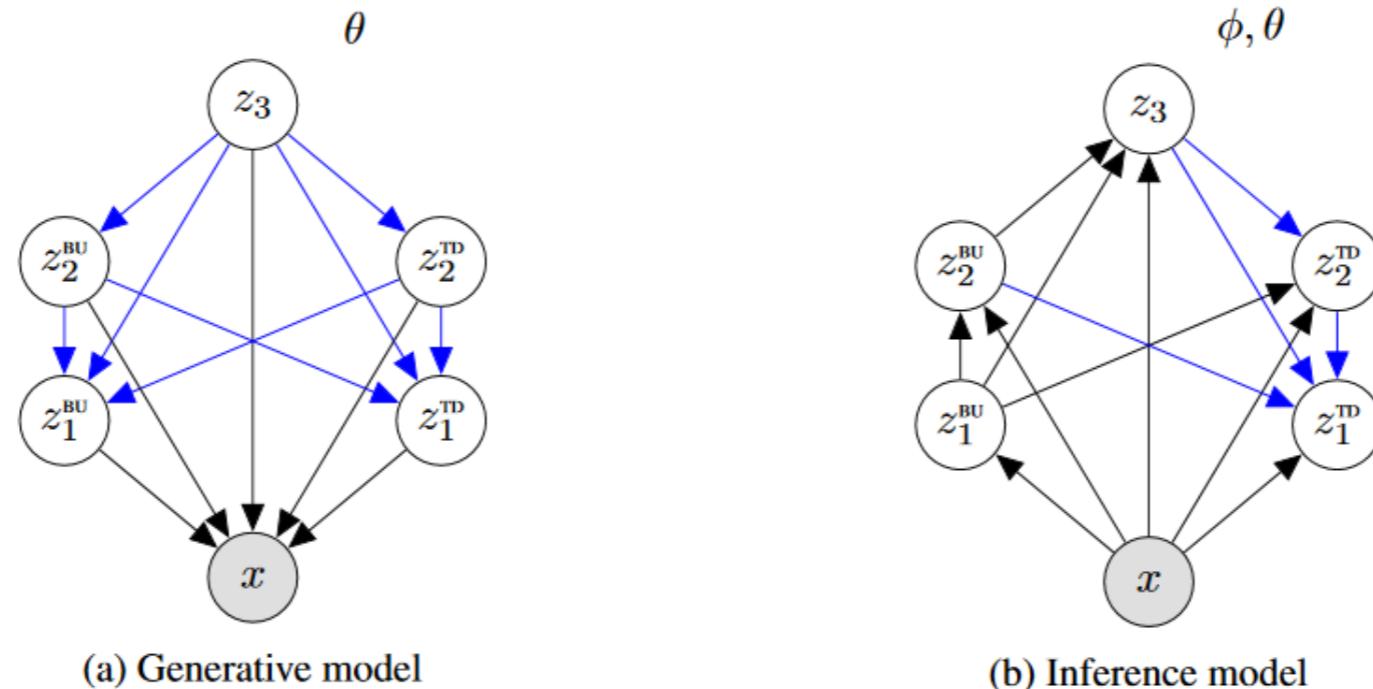


Figure 1: A  $L = 3$  layered BIVA with (a) the generative model and (b) inference model. Blue arrows indicate that the deterministic parameters are shared between the inference and generative models. See Appendix B for a detailed explanation and a graphical model that includes the deterministic variables.

$$p_{\theta}(x, \mathbf{z}) = p_{\theta}(x|\mathbf{z})p_{\theta}(\mathbf{z}_L) \prod_{i=1}^{L-1} p_{\theta}(z_i^{\text{BU}}|z_{>i})p_{\theta}(z_i^{\text{TD}}|z_{>i})$$

$$q_{\phi}(\mathbf{z}|x) = q_{\phi}(z_L|x, z_{<L}^{\text{BU}}) \prod_{i=1}^{L-1} q_{\phi}(z_i^{\text{BU}}|x, z_{<i}^{\text{BU}})q_{\phi,\theta}(z_i^{\text{TD}}|x, z_{<i}^{\text{BU}}, z_{>i}^{\text{BU}}, z_{>i}^{\text{TD}})$$

## Bidirectional-Inference Variational Autoencoder (BIVA)

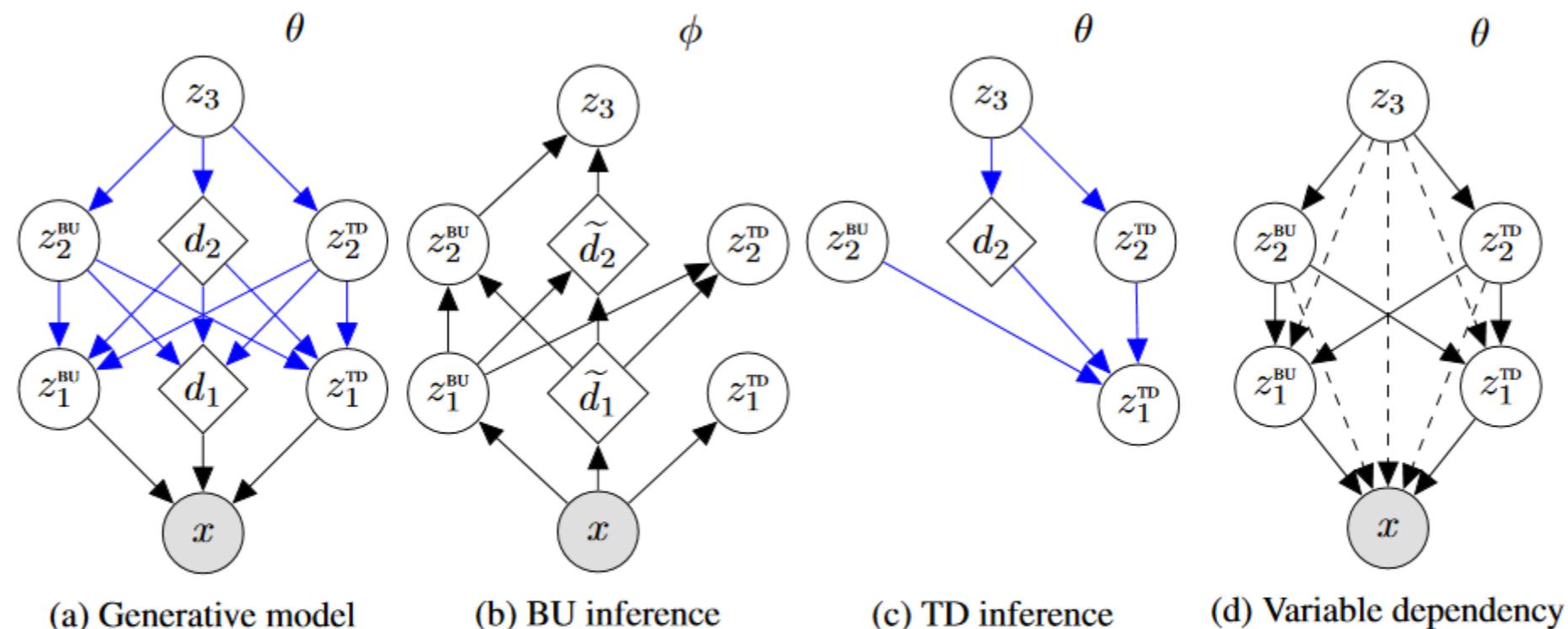


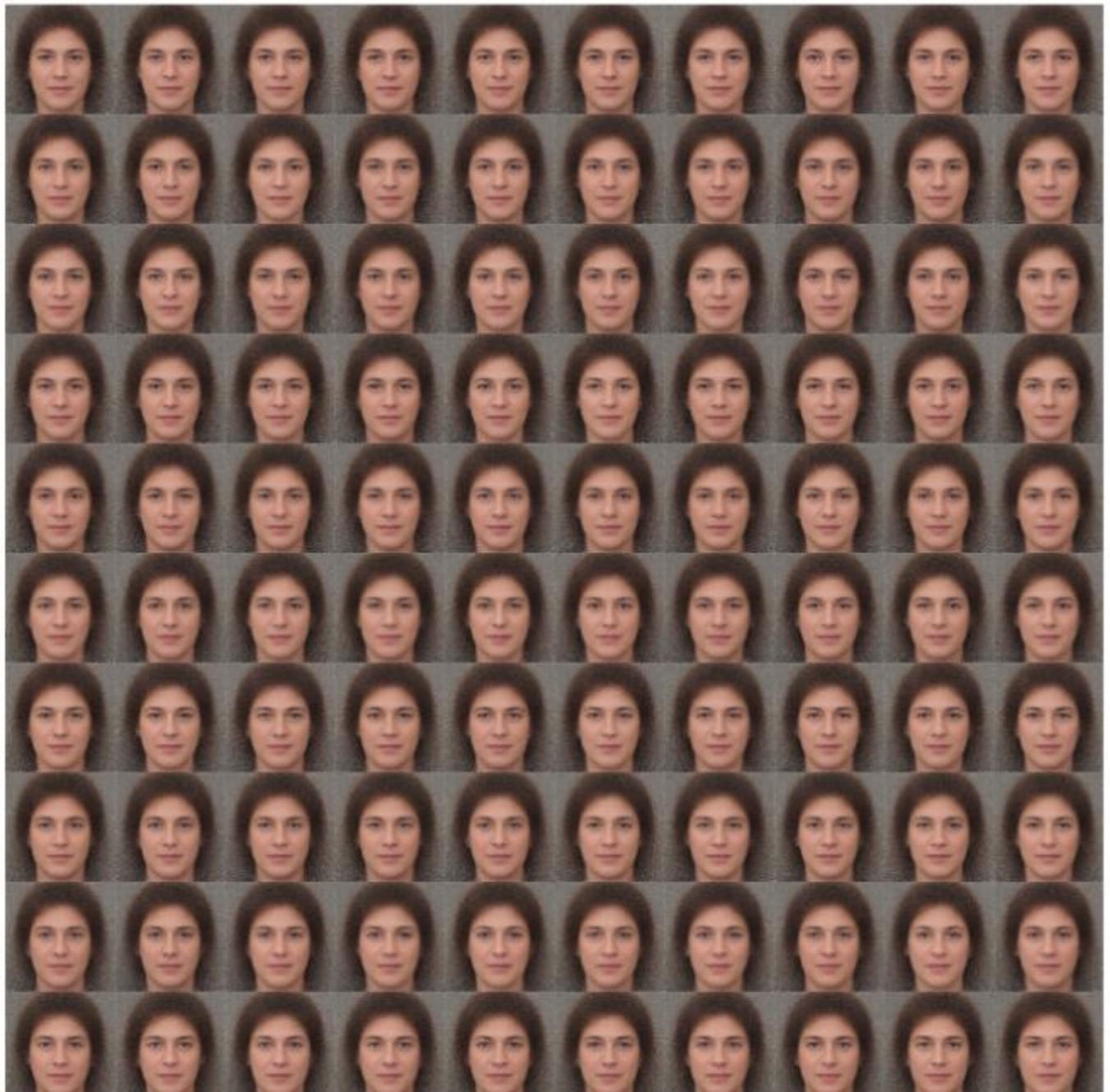
Figure 6: A  $L = 3$  layered BIVA with (a) the generative model, (b) bottom-up (BU) inference path, (c) top-down (TD) inference path, and (d) variable dependency of the generative models where dashed lines denote a skip-connection. Blue arrows indicate that the deterministic parameters are shared within the generative model or between the generative and inference model.

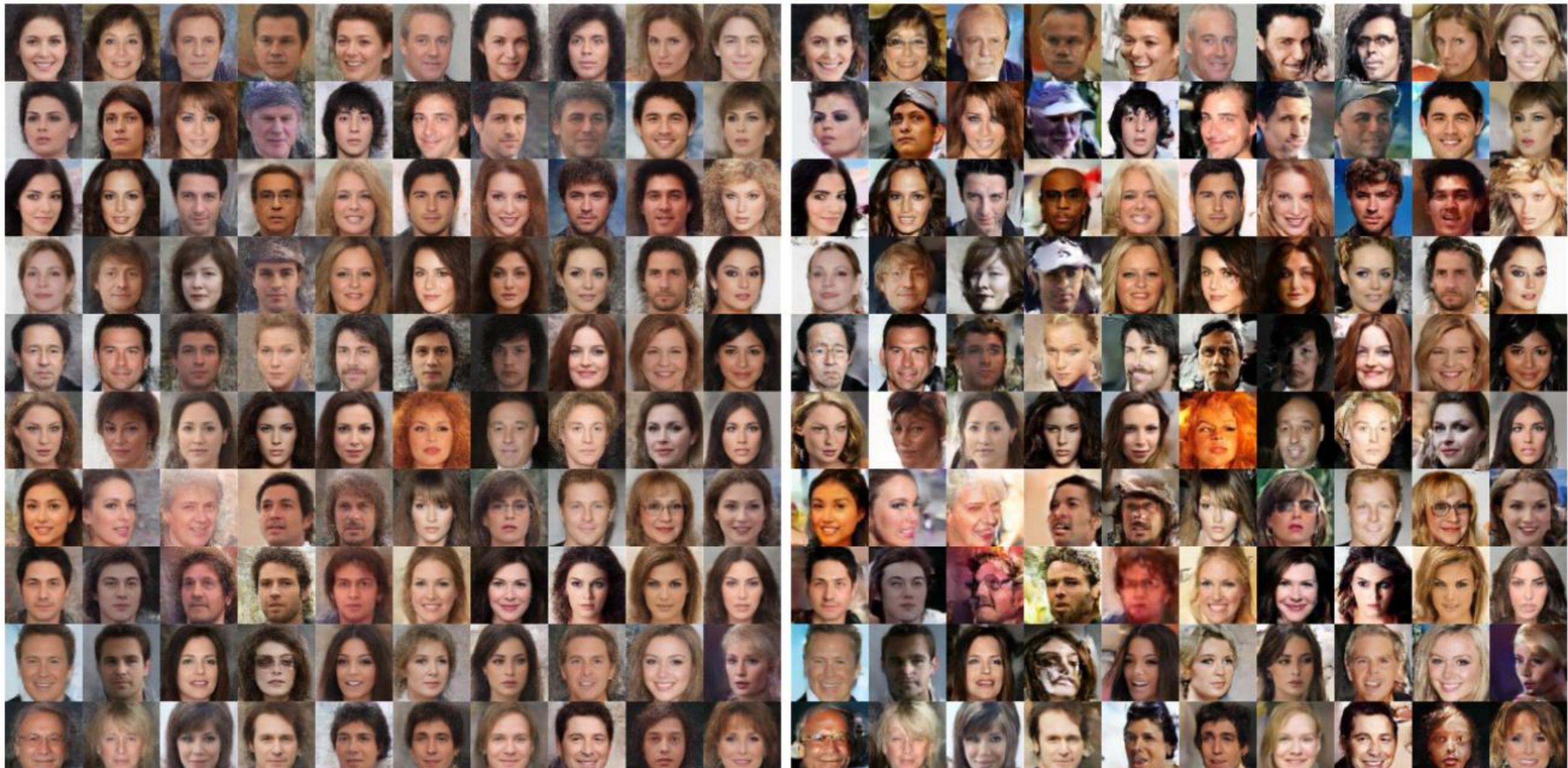
**в генеративной модели много прокидывания связей  
дву направленный вывод (Bidirectional inference)**

## Bidirectional-Inference Variational Autoencoder (BIVA)



Figure 10: 64x64 CelebA samples generated from a BIVA with increasing levels of stochasticity in the model (going from close to the mode to the full distribution). In each column the latent variances are scaled with factors 0.1, 0.3, 0.5, 0.7, 0.9, 1.0. Images in a row look similar because they use the same Gaussian random noise  $\epsilon$  to generate the latent variables. BIVA has  $L = 20$  stochastic latent layers connected by three layer ResNet blocks.

(a)  $\sigma^2 = 0.01$ (b)  $\sigma^2 = 0.1$

(c)  $\sigma^2 = 0.5$ (d)  $\sigma^2 = 1.0$

## Vector Quantised-Variational AutoEncoder (VQ-VAE)

### отличия от стандартного VAE

- выход кодировщика дискретный (дискретное латентное состояние выбирается 1-NN)
- prior обучаем (над латент. пр-ом – авторегрессионное, например с помощью PixelCNN)

Вспомним...

VAE = AE со специальной регуляризацией (в латентном пространстве)  
а тут латентное пространство ещё дискретное + авторегрессионная модель

VQ-VAE: Neural Discrete Representation Learning, A. van den Oord, O. Vinyals and K. Kavukcuoglu,  
NeurIPS 2017, <https://arxiv.org/pdf/1711.00937.pdf>

## Vector Quantised-Variational AutoEncoder (VQ-VAE)

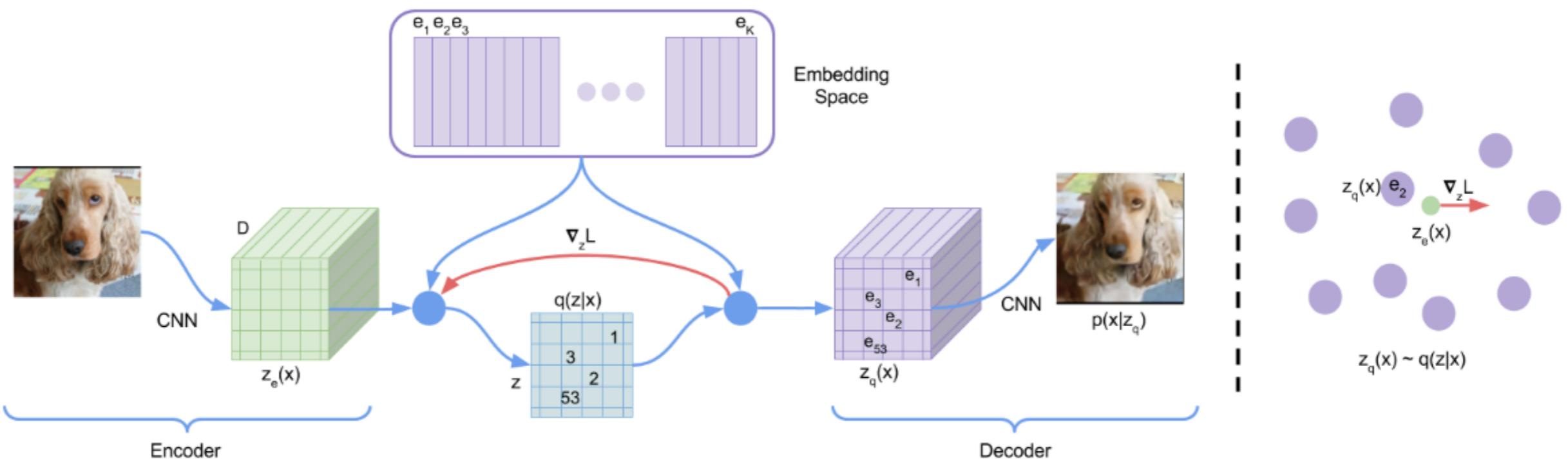


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder  $z(x)$  is mapped to the nearest point  $e_2$ . The gradient  $\nabla_z L$  (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

**для  $w \times h$  векторов в тензоре находим дискретные представления  
градиент перекидывается по красной стрелке**

## Vector Quantised-Variational AutoEncoder (VQ-VAE)

$$q(z = e_k \mid x) = \begin{cases} 1, & k = \arg \min_i \| z_e(x) - e_i \|_2 \\ 0, & \text{иначе,} \end{cases}$$

**оптимизируем:**

$$L = \underbrace{\|\mathbf{x} - D(\mathbf{e}_k)\|_2^2}_{\text{reconstruction loss}} + \underbrace{\|\text{sg}[E(\mathbf{x})] - \mathbf{e}_k\|_2^2}_{\text{VQ loss}} + \underbrace{\beta \|E(\mathbf{x}) - \text{sg}[\mathbf{e}_k]\|_2^2}_{\text{commitment loss}}$$

**sg = stop\_gradient**

**VQ loss – ошибка между представлениями (кодовыми словами) и выходами кодировщика**

**Commitment loss – чтобы не было частой смены кодов**

**в кодовой книге (embed. space) обновляем коды с помощью ЕМА  
(exponential moving average)**

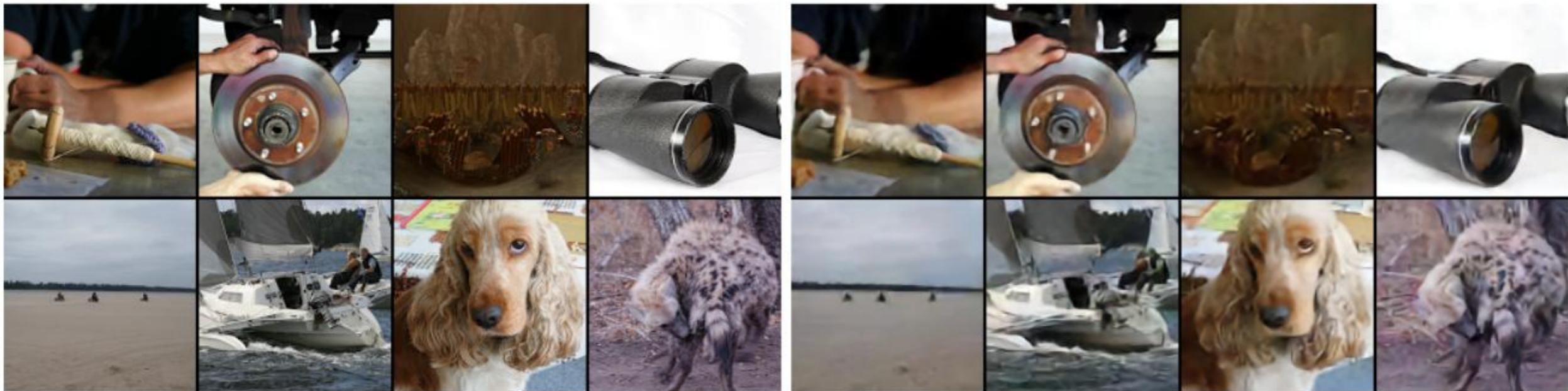


Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

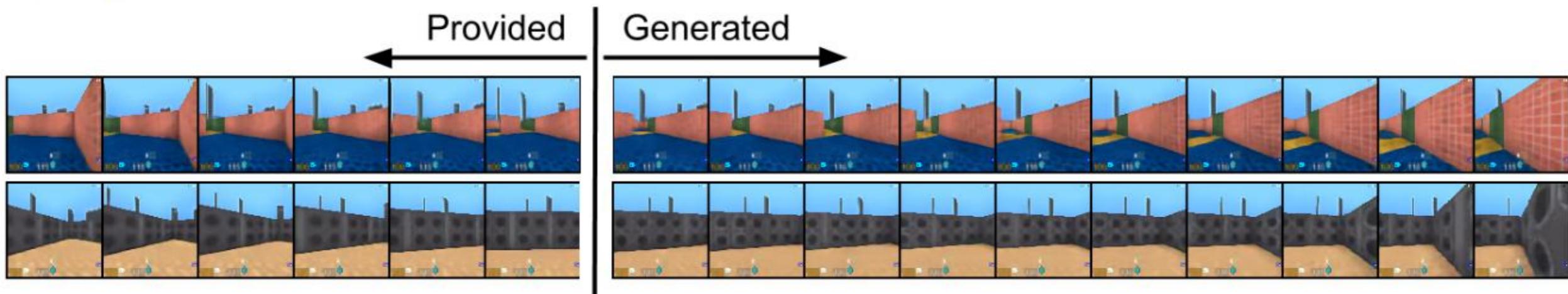


Figure 7: First 6 frames are provided to the model, following frames are generated conditioned on an action. Top: repeated action "move forward", bottom: repeated action "move right".

**VQ-VAE-2**

Figure 1: Class-conditional 256x256 image samples from a two-level model trained on ImageNet.

**кодировщик и декодировщик – простые сети прямого распространения  
VQ-VAE требует авторегрессионную модель в латентном пространстве  
(а не в пространстве пикселей)**

**в отличие от GAN большое разнообразие и нет «mode collapse»**

## VQ-VAE-2

**здесь изображение кодируется на нескольких уровнях (абстракции)  
с помощью кодировщика см. ниже**

**Algorithm 1** VQ-VAE training (stage 1)

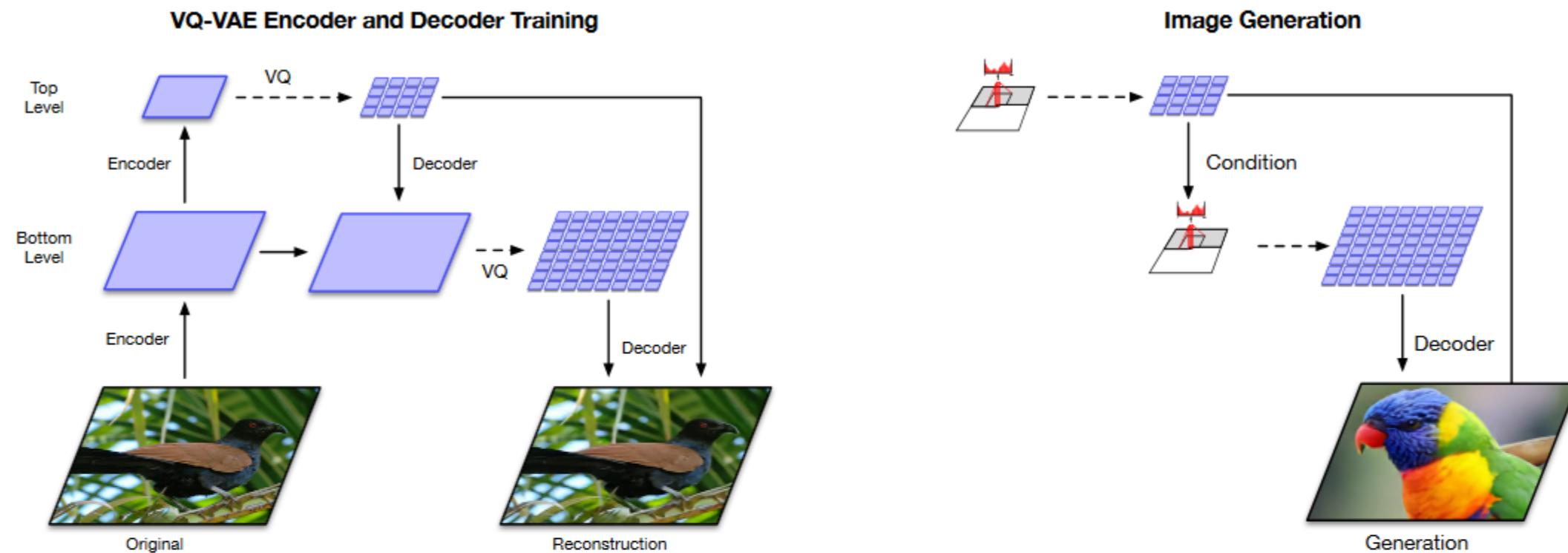
**Require:** Functions  $E_{top}$ ,  $E_{bottom}$ ,  $D$ ,  $\mathbf{x}$   
(batch of training images)

- 1:  $\mathbf{h}_{top} \leftarrow E_{top}(\mathbf{x})$   
▷ quantize with top codebook eq 1
- 2:  $\mathbf{e}_{top} \leftarrow Quantize(\mathbf{h}_{top})$
- 3:  $\mathbf{h}_{bottom} \leftarrow E_{bottom}(\mathbf{x}, \mathbf{e}_{top})$   
▷ quantize with bottom codebook eq 1
- 4:  $\mathbf{e}_{bottom} \leftarrow Quantize(\mathbf{h}_{bottom})$
- 5:  $\hat{\mathbf{x}} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$   
▷ Loss according to eq 2
- 6:  $\theta \leftarrow Update(\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}))$

**Algorithm 2** Prior training (stage 2)

- 1:  $\mathbf{T}_{top}, \mathbf{T}_{bottom} \leftarrow \emptyset$  ▷ training set
- 2: **for**  $\mathbf{x} \in$  training set **do**
- 3:    $\mathbf{e}_{top} \leftarrow Quantize(E_{top}(\mathbf{x}))$
- 4:    $\mathbf{e}_{bottom} \leftarrow Quantize(E_{bottom}(\mathbf{x}, \mathbf{e}_{top}))$
- 5:    $\mathbf{T}_{top} \leftarrow \mathbf{T}_{top} \cup \mathbf{e}_{top}$
- 6:    $\mathbf{T}_{bottom} \leftarrow \mathbf{T}_{bottom} \cup \mathbf{e}_{bottom}$
- 7: **end for**
- 8:  $p_{top} = TrainPixelCNN(\mathbf{T}_{top})$
- 9:  $p_{bottom} = TrainCondPixelCNN(\mathbf{T}_{bottom}, \mathbf{T}_{top})$   
▷ Sampling procedure
- 10: **while** true **do**
- 11:    $\mathbf{e}_{top} \sim p_{top}$
- 12:    $\mathbf{e}_{bottom} \sim p_{bottom}(\mathbf{e}_{top})$
- 13:    $\mathbf{x} \leftarrow D(\mathbf{e}_{top}, \mathbf{e}_{bottom})$
- 14: **end while**

## VQ-VAE-2



(a) Overview of the architecture of our hierarchical VQ-VAE. The encoders and decoders consist of deep neural networks. The input to the model is a  $256 \times 256$  image that is compressed to quantized latent maps of size  $64 \times 64$  and  $32 \times 32$  for the *bottom* and *top* levels, respectively. The decoder reconstructs the image from the two latent maps.

(b) Multi-stage image generation. The top-level PixelCNN prior is conditioned on the class label, the bottom level PixelCNN is conditioned on the class label as well as the first level code. Thanks to the feed-forward decoder, the mapping between latents to pixels is fast. (The example image with a parrot is generated with this model).

Figure 2: VQ-VAE architecture.

**VQ-VAE-2** $h_{\text{top}}$  $h_{\text{top}}, h_{\text{middle}}$  $h_{\text{top}}, h_{\text{middle}}, h_{\text{bottom}}$ 

Original

Figure 3: Reconstructions from a hierarchical VQ-VAE with three latent maps (top, middle, bottom). The rightmost image is the original. Each latent map adds extra detail to the reconstruction. These latent maps are approximately 3072x, 768x, 192x times smaller than the original image (respectively).

**результат учёта разных уровней иерархии кодирования**

**VQ-VAE (Proposed)****BigGAN deep**

Figure 5: Sample diversity comparison for the proposed method and BigGan Deep for Tinca-Tinca (1st ImageNet class) and Ostrich (10th ImageNet class). BigGAN samples were taken with the truncation level 1.0, to yield its maximum diversity. There are several kinds of samples such as top view of the fish or different kinds of poses such as a close up ostrich absent from BigGAN's samples. Please zoom into the pdf version for more details and refer to the Supplementary material for diversity comparison on more classes.

## Adversarial Autoencoder

**Здесь в узком горле будет работать дискриминатор,  
чтобы распределение было похоже на заданное**

**чтобы понять, надо знать, что такое GAN**

## Итог

**VAE – естественная реализация автокодировщика**

**векторная арифметика в латентном пространстве**

**есть специальные латентные пространства и функции ошибок**

**далее сравним с другими решениями**

**Самая лучшая статья по VAE:**

**Carl Doersch «Tutorial on Variational Autoencoders» // <https://arxiv.org/abs/1606.05908>**