курс «Глубокое обучение»

Обучение без учителя

Александр Дьяконов

11 мая 2020 года

План

Автокодировщики (Auto-encoders)

Denoising Autoencoder

Contractive Autoencoders (CAE)

Sparse Coding

Context Encoders

Глубокие RBM (Deep Boltzmann Machines)

SOM – Самоорганизующиеся карты Кохонена

Сжатие

Оценка плотности

Авторегрессионные модели

Поток (Glow)

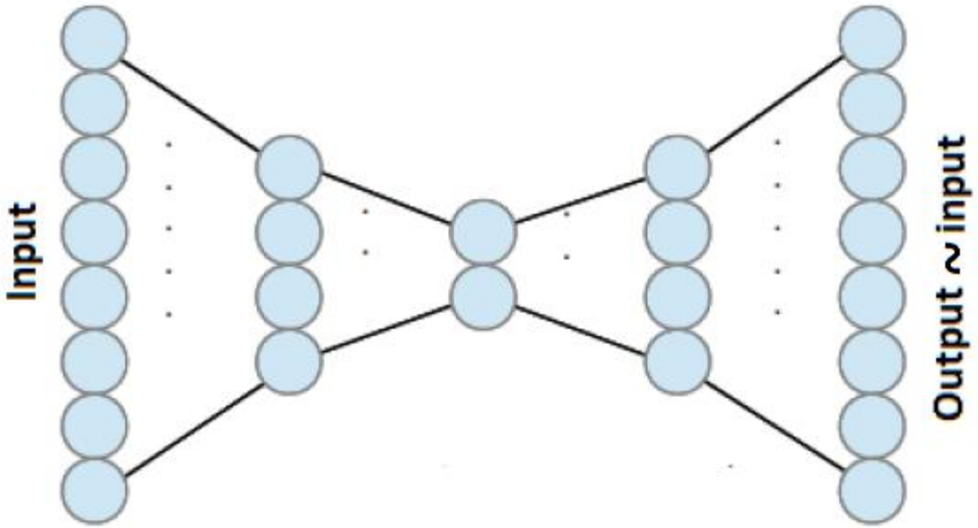
Iterative Attentive Generation

<table>
<tr><th colspan="4">Обучение без учителя
Unsupervised Learning</th></tr>
<tr><th>Невероятностные модели</th><th colspan="3">Вероятностные (генеративные) модели</th></tr>
<tr><td rowspan="3">• Sparse Coding
• Autoencoders
• k-means, …</td><th colspan="2">Плотность в явном виде (explicit density)</th><th>Плотность в неявном виде (implicit density)</th></tr>
<tr><th>Tractable Models</th><th colspan="2">Approximate density</th><td rowspan="3">• GAN
• Momet Matching Networks

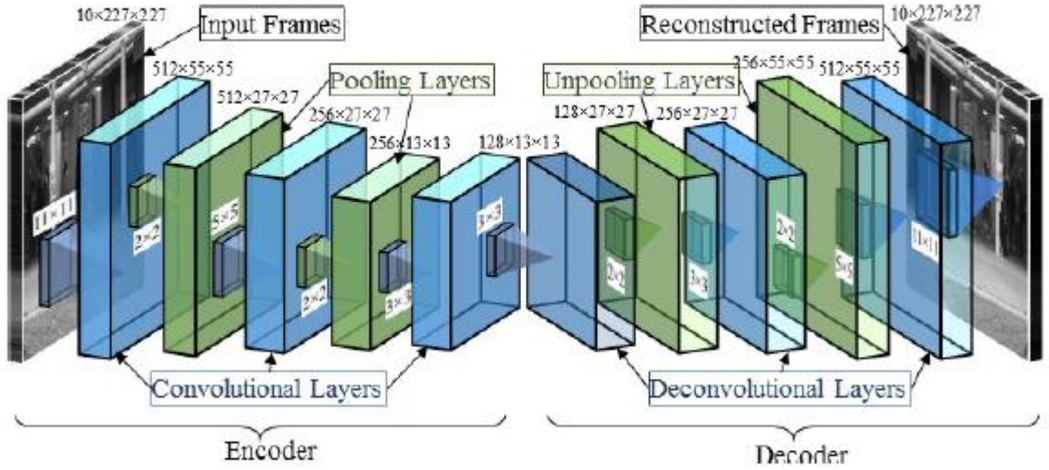
Моделируем процесс сэмплирования, а не плотность</td></tr>
<tr><td>• Fully observed Belief Nets
• NADE
• MADE
• PixelRNN
• nonlinear ICA</td><th>Variational</th><th>Markov Chain</th></tr>
<tr><td></td><td>• VAE</td><td>• Boltzmann Machines
• Helmholtz Machines</td></tr>
</table>

Автокодировщики (Auto-encoders)

Полносвязный с узким горлом (bottleneck)

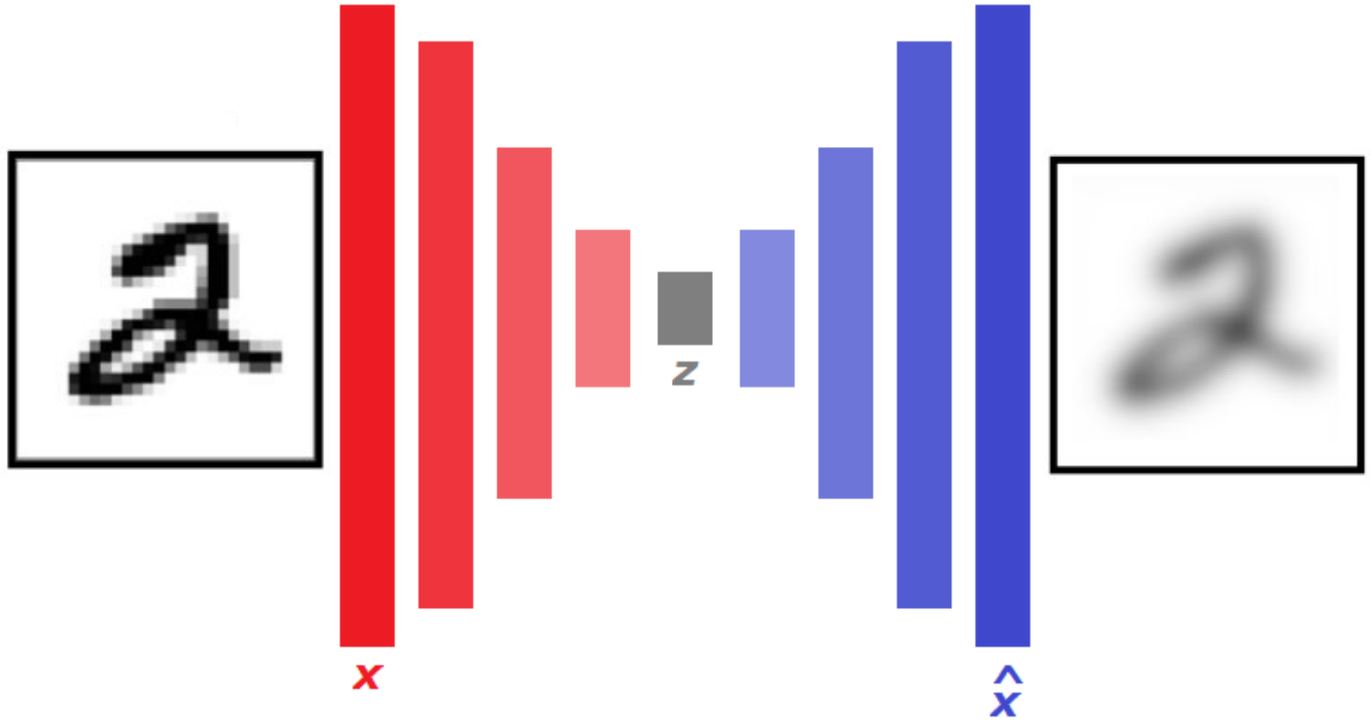


Свёрточный



Знакомая архитектура «кодировщик-декодировщик»

Глубокие автокодировщики – обучение

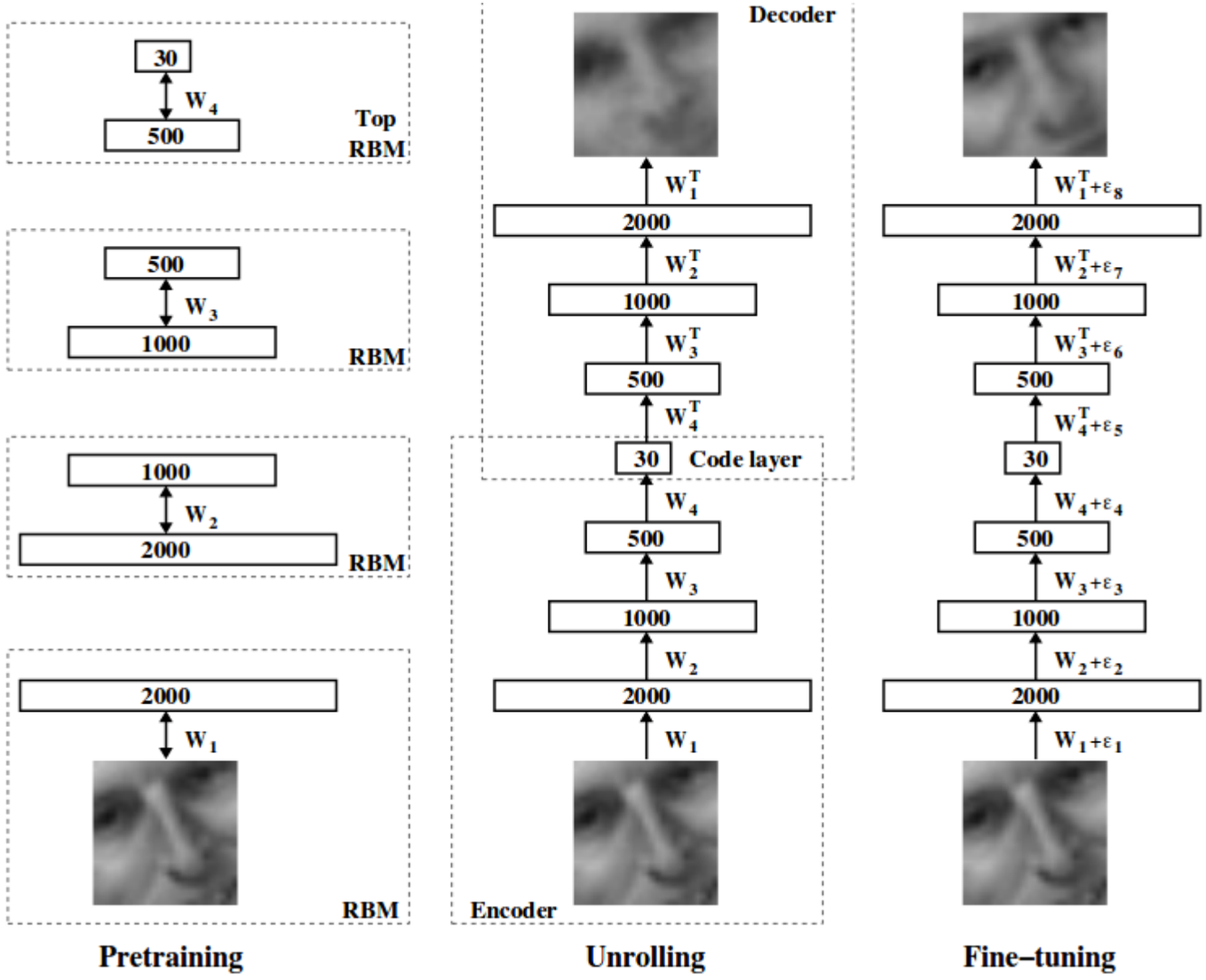


воспроизводим вход

$$\| x - g(f(x)) \| \to \min$$

$$\underbrace{z}_{\hat{x}}$$

Глубокие автокодировщики – обучение с помощью RBM (раньше)

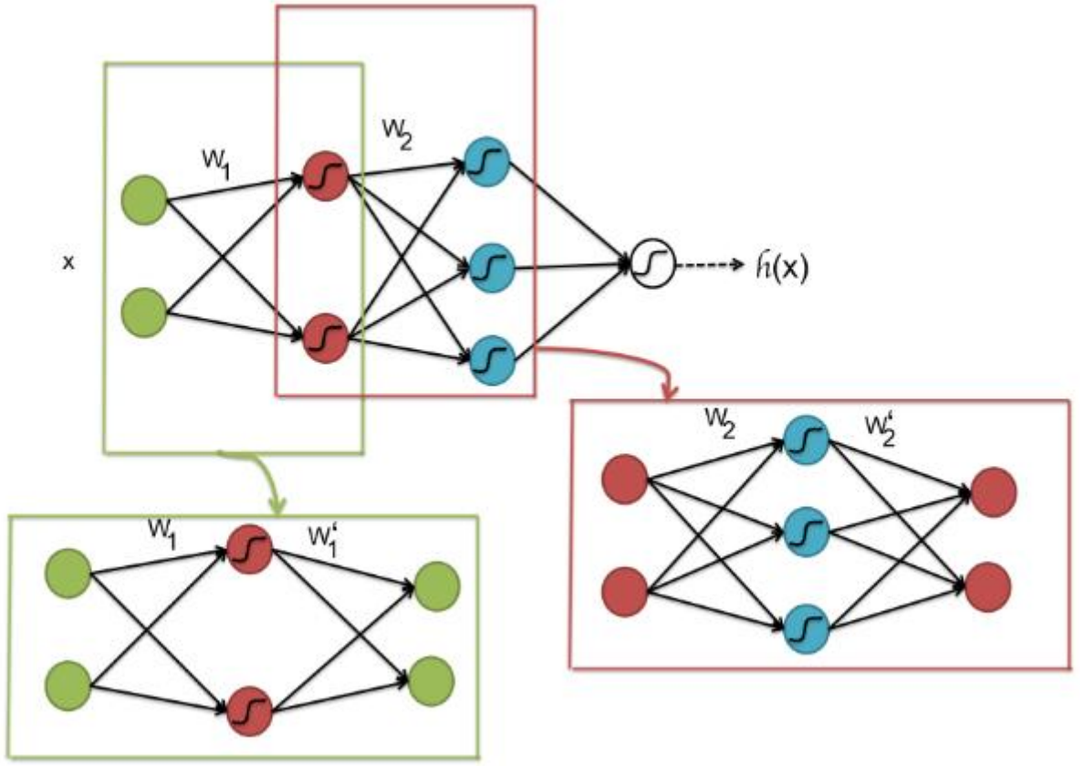


Зачем нужны автокодировщики

- сокращение размерности
- выделение признаков (для других алгоритмов)
- предобучение
- специальные задачи (ниже: устранение шума)

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

Denoising Autoencoder

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

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

Сокращающие автокодировщики – Contractive Autoencoders (CAE)

- идея – сделать менее чувствительными к небольшим изменениям данных
- похоже на «sparse autoencoders» и «denoising autoencoders»

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

штраф – норма Фробениуса Якобиана

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

$h = f(x)$ – **скрытые слои**

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

Sparse Coding

Постановка задачи. Выборка $\{x_i\}_{i=1}^m$, базисы $\{b_i\}_{i=1}^k$, хотим представить в виде разреженной линейной комбинации (sparse linear combination)

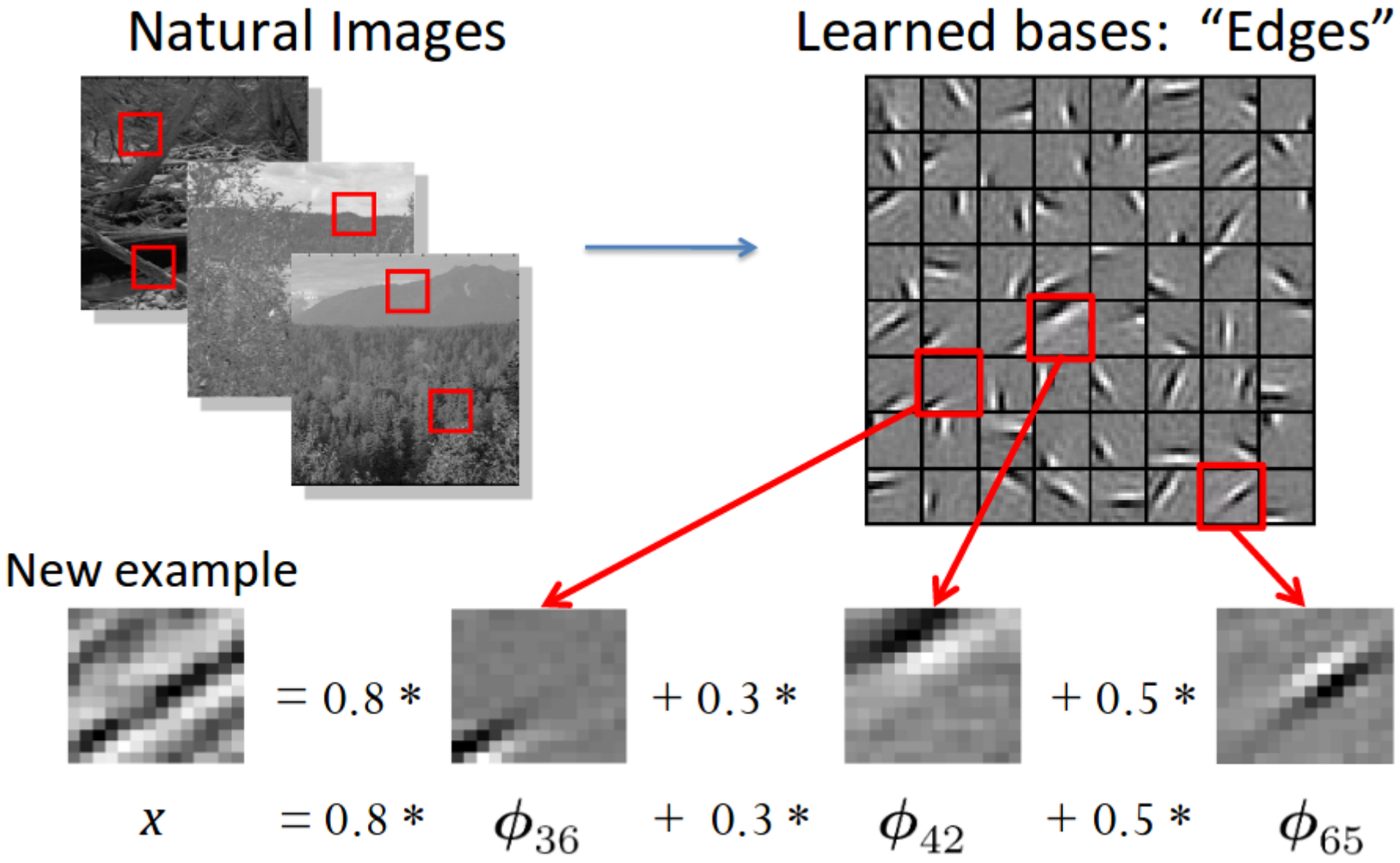
$$x_i \approx \sum_{j=1}^{k} \alpha_{ij} b_j :$$

в основном все α_{ij} нулевые!

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

Alternative optimization – попеременно фиксировать базис и коэффициенты

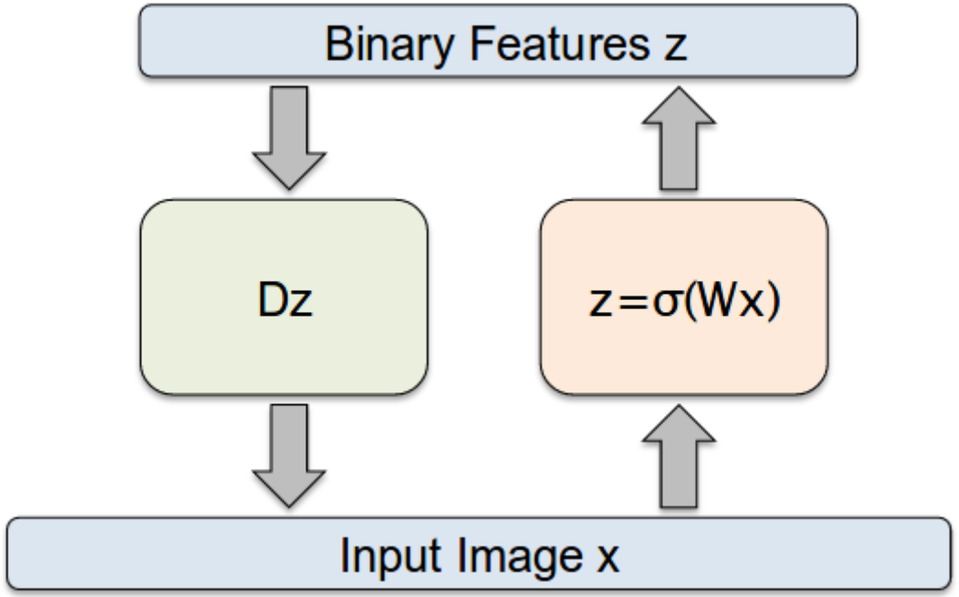
Sparse Coding



Kavukcuoglu, Ranzato, Fergus, LeCun, 2009

Sparse Coding

вписывается в парадигму автокодировщика!



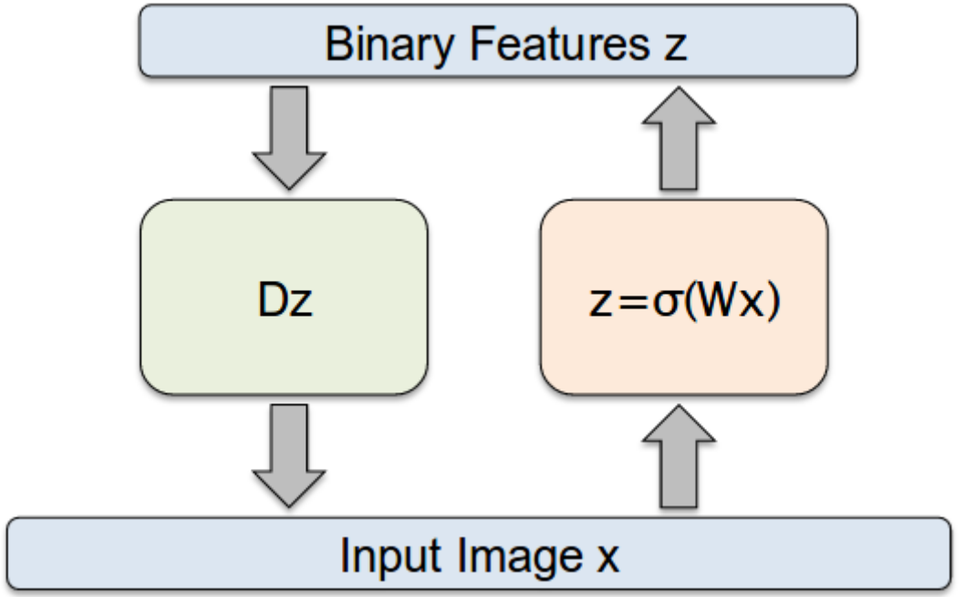
k скрытых бинарных нейронов

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

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

Sparse Coding

вписывается в парадигму автокодировщика!



k скрытых бинарных нейронов

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

Context Encoders

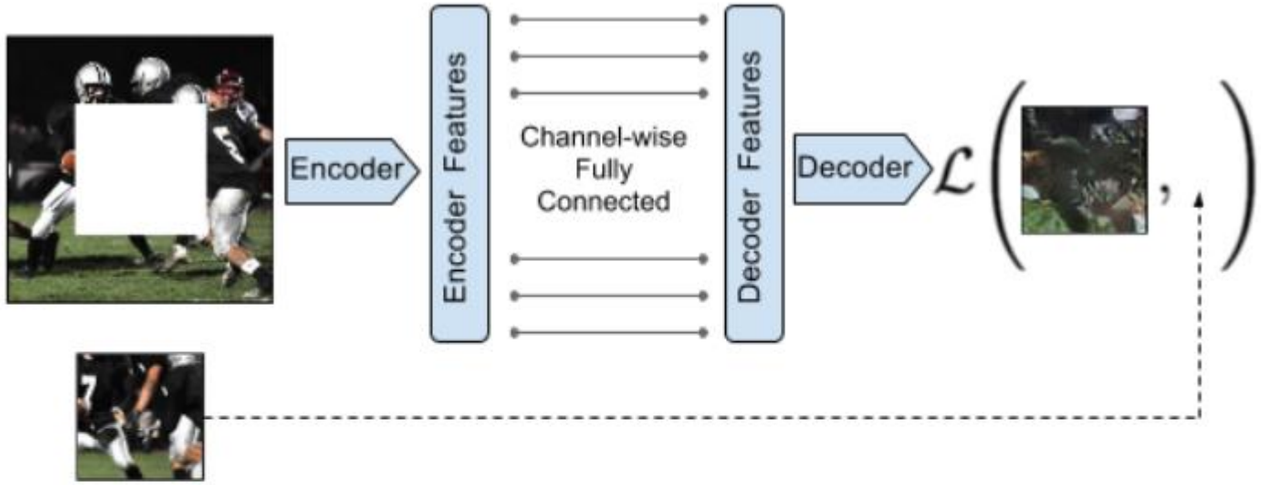
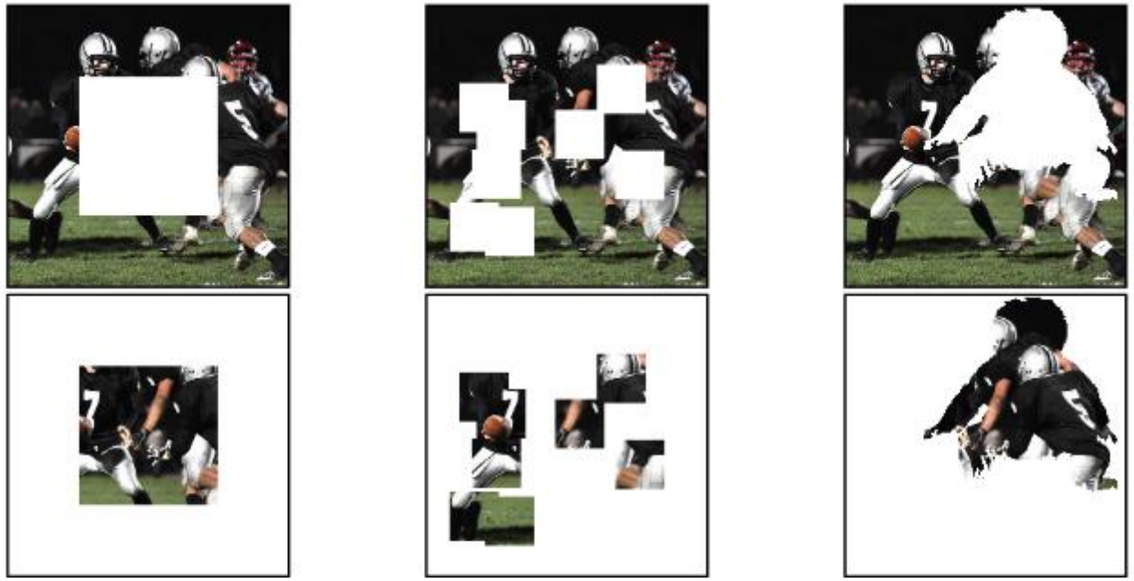


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

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

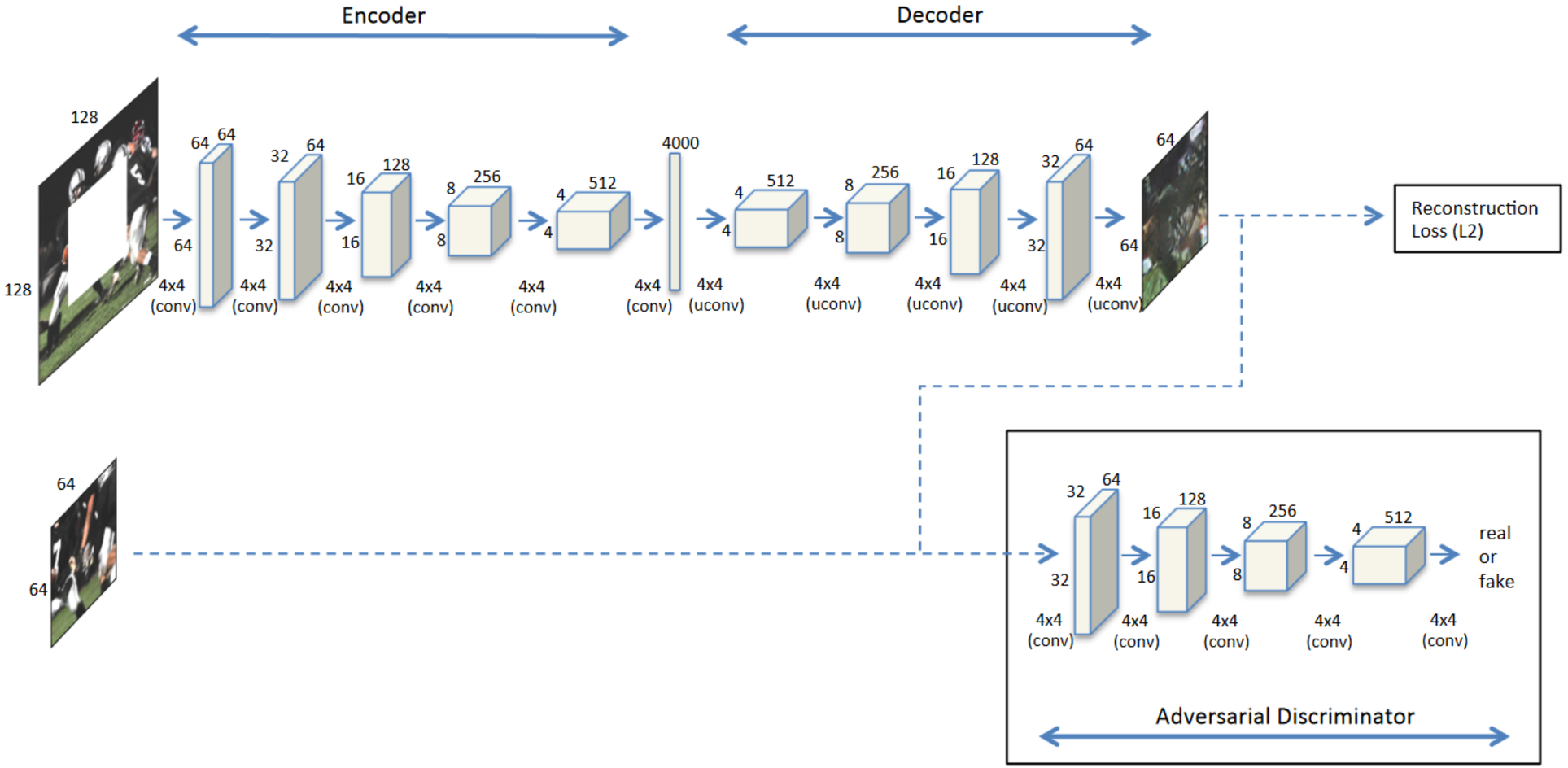
Context Encoders



(a) Central region (b) Random block (c) Random region

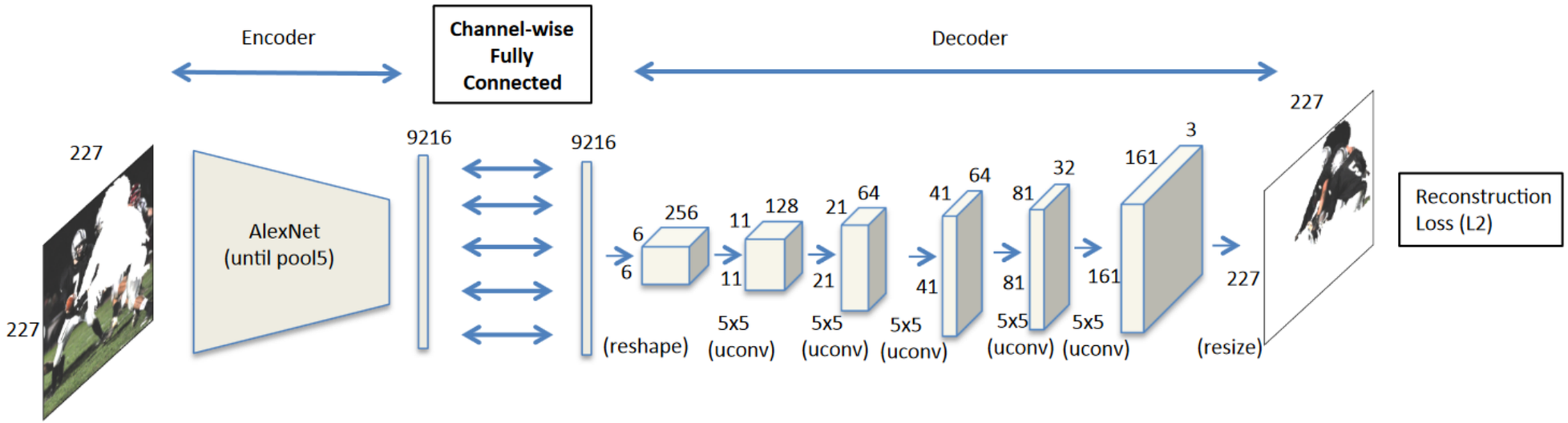
Figure 3: An example of image x with our different region masks $\hat{M}$ applied, as described in Section 3.3.

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.

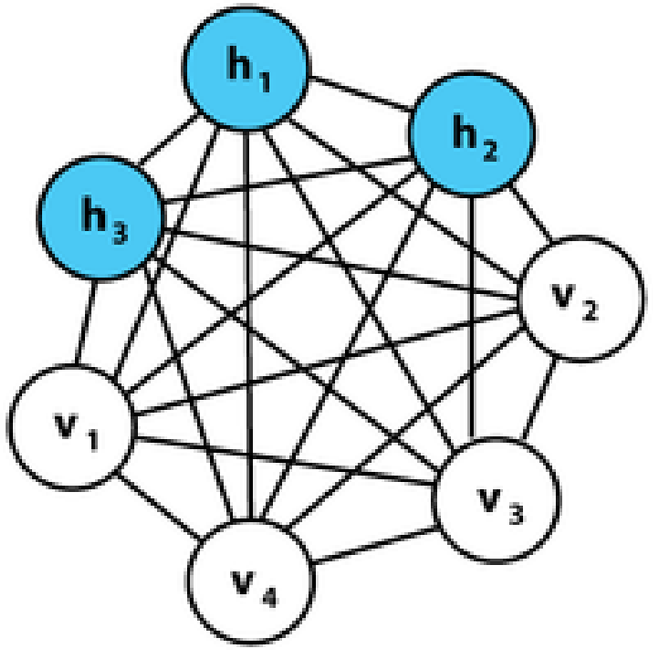
Context Encoders



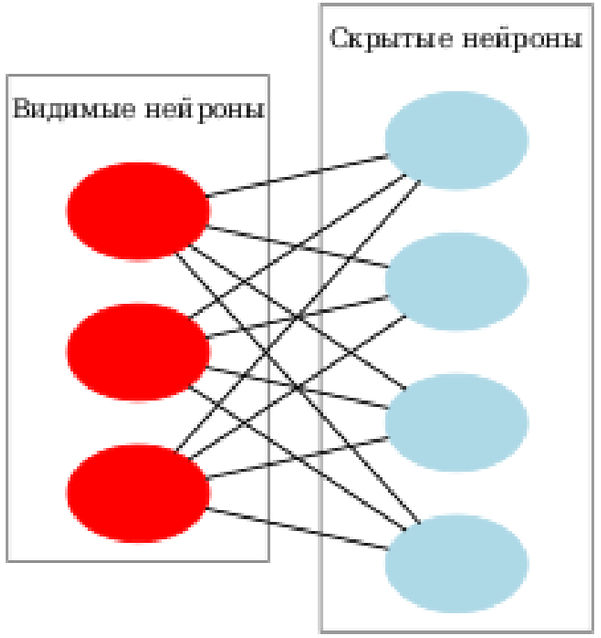
(b) Context encoder trained with reconstruction loss for feature learning by filling in *arbitrary region dropouts* in the input.

Машины Больцмана

Машина Больцмана
Boltzmann machine



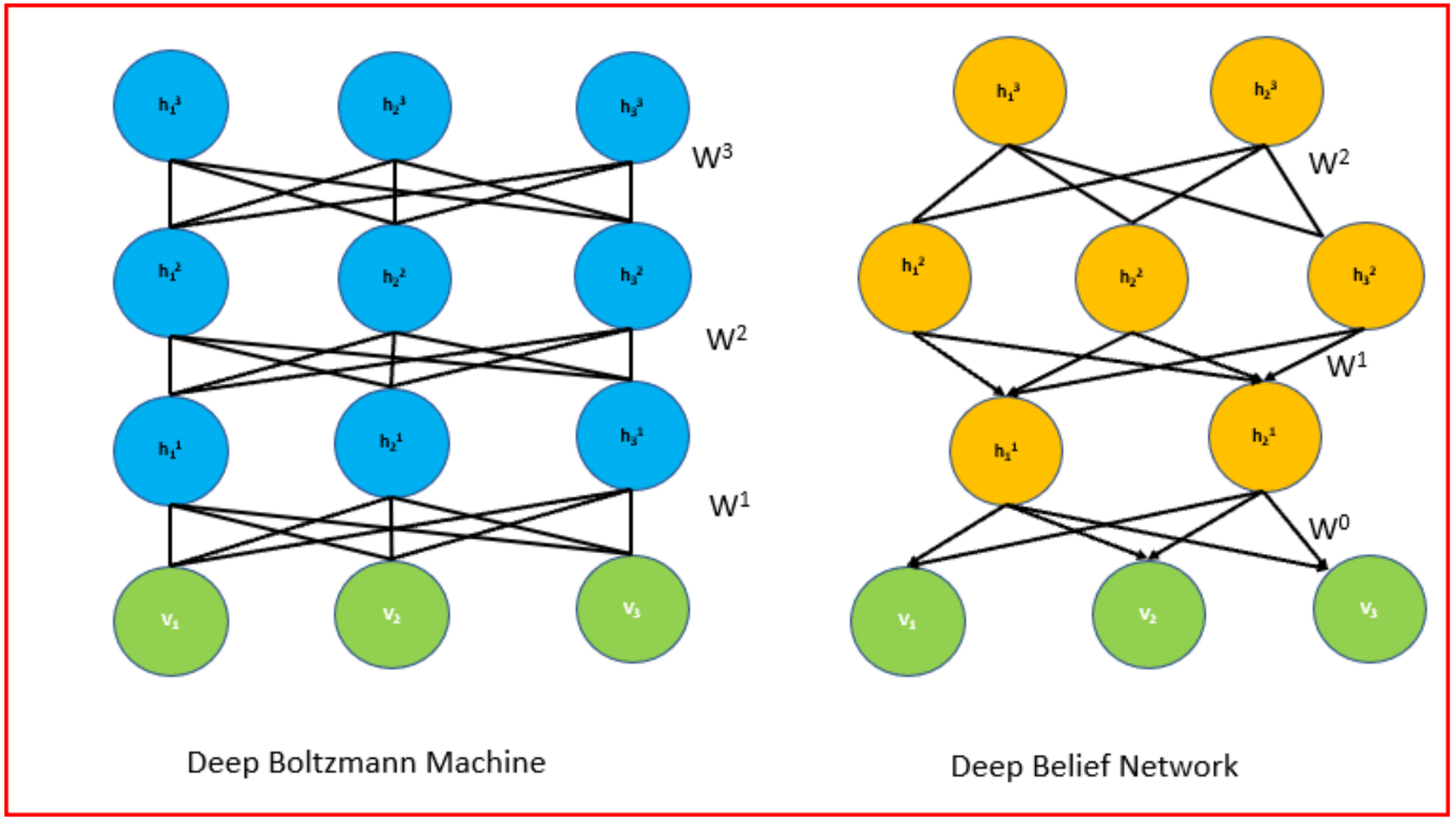
Ограниченная машина Больцмана
Restricted Boltzmann machine (RBM)



связи только со скрытыми нейронами

https://en.wikipedia.org/wiki/Boltzmann_machine

Машины Больцмана

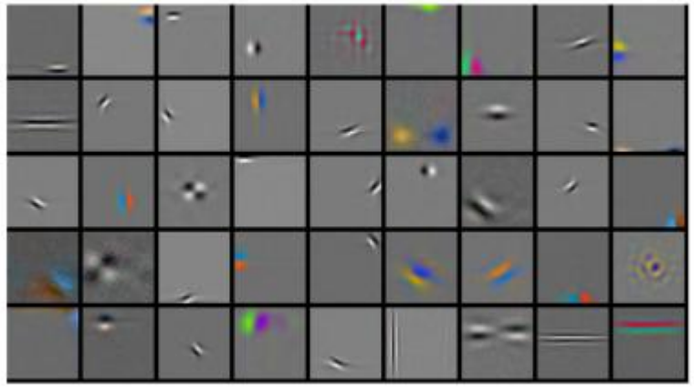


https://medium.com/datadriveninvestor/deep-learning-deep-boltzmann-machine-dbm-e3253bb95d0f

Использование RBM

4 million **unlabelled** images

Learned features (out of 10,000)



Reuters dataset:
804,414 **unlabeled**
newswire stories
Bag-of-Words

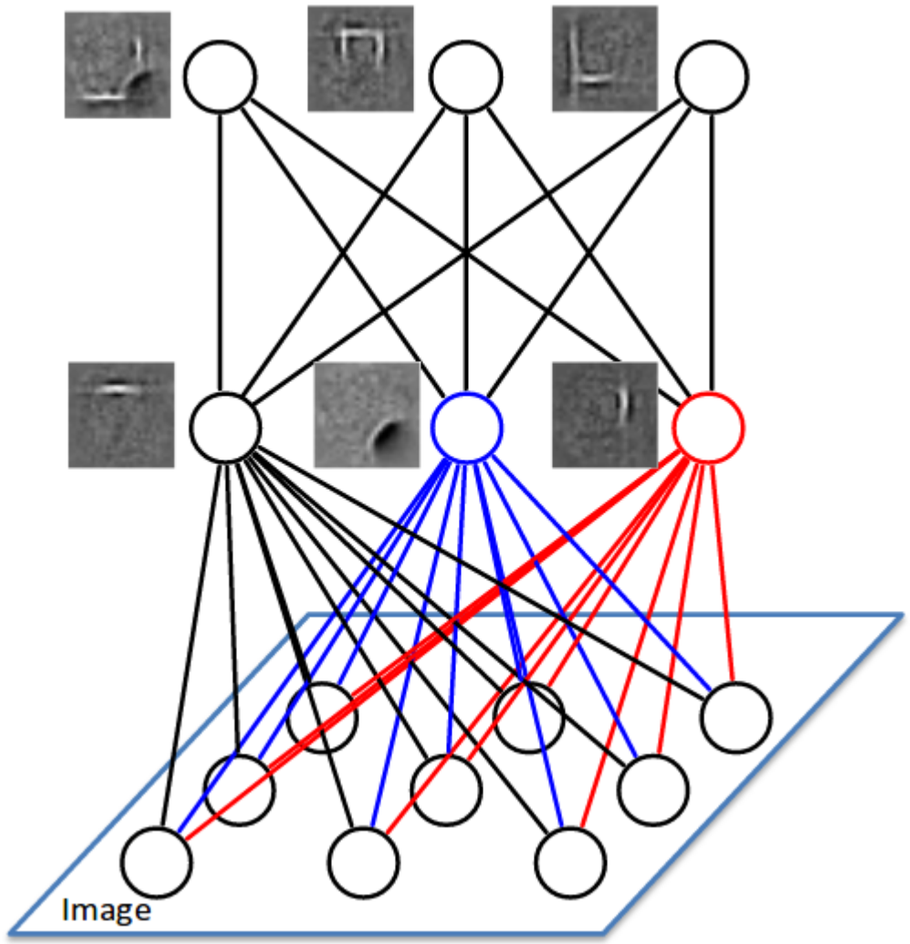
Learned features: ``topics''

russian	clinton	computer	trade	stock
russia	house	system	country	wall
moscow	president	product	import	street
yeltsin	bill	software	world	point
soviet	congress	develop	economy	dow

Энергия RBM

$$P_\theta(v,h) = \frac{1}{Z(\theta)} \exp\left(\sum_{ij} W_{ij} v_i h_j + \sum_i b_i v_i + \sum_j a_j h_j \right)$$

Глубокие RBM (Deep Boltzmann Machines)



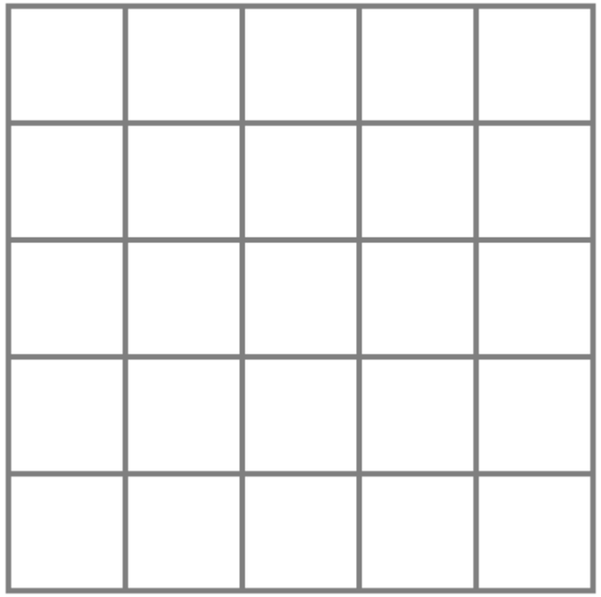
Многоуровневое пространство скрытых переменных (разные степени детализации)

Salakhutdinov, Hinton, 2009

SOM – Самоорганизующиеся карты Кохонена

Хотим отобразить пространство в регулярную структуру (например, узлы решётки решётки)

Решётка задаёт соседство узлов (топологию)



Самоорганизующиеся карты Кохонена

Алгоритм обучения

1. Подаём объект x

2. Находим самый близкий узел

$$(a,b) = \arg\max_{(i,j)} \text{sim}(x, w_{ij})$$

соревнование ~ нейрон-победитель, пример $\text{sim}(x, w_{ij}) = -\| x - w_{ij} \|$

3. Коррекция весов

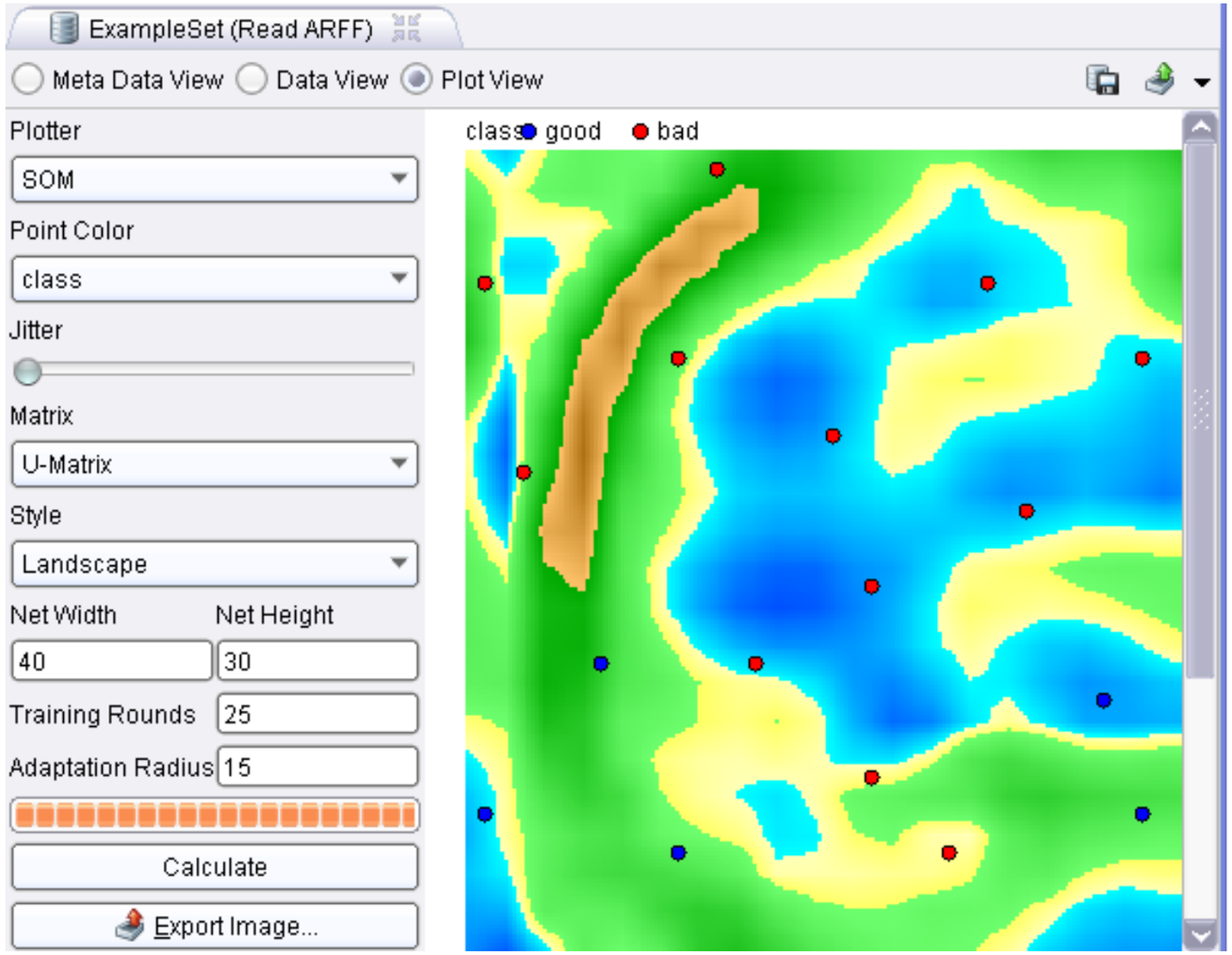
$$w_{ij} \leftarrow w_{ij} + \eta K((a,b),(i,j))x$$

веса нейронов около победителя корректируются больше...

~ кооперация, пример $K((a,b),(i,j)) = \exp(-|a-i| + |b-j|)$

$$(a,b) = \arg\max_{(i,j)} \text{sim}(x, w_{ij})$$

Самоорганизующиеся карты Кохонена



Построение карты Кохонена в программе RapidMiner

Сжатие

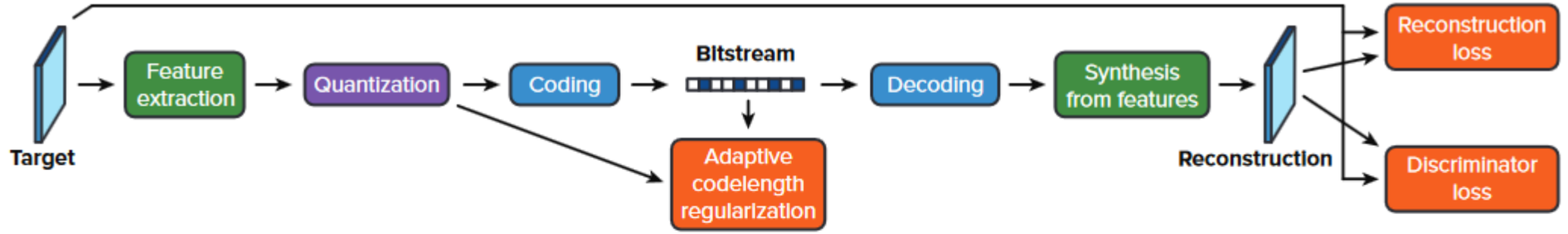


Figure 2. Our overall model architecture. The feature extractor, described in Section 3.1, discovers structure and reduces redundancy via the pyramidal decomposition and interscale alignment modules. The lossless coding scheme, described in Section 3.2, further compresses the quantized tensor via bitplane decomposition and adaptive arithmetic coding. The adaptive codelength regularization then modulates the expected code length to a prescribed target bitrate. Distortions between the target and its reconstruction are penalized by the reconstruction loss. The discriminator loss, described in Section 4, encourages visually pleasing reconstructions by penalizing discrepancies between their distributions and the targets'.

тоже кодировщик-декодировщик

тут немного «мудрённый»

тут используется GAN – будет дальше

Oren Rippel «Real-Time Adaptive Image Compression» https://arxiv.org/pdf/1705.05823.pdf

JPEG 0.0826 BPP (7.5% bigger) | **JPEG 2000** 0.0778 BPP | **WebP** 0.0945 BPP (23% bigger) | **Ours** 0.0768 BPP

JPEG 0.111 BPP (10% bigger) | **JPEG 2000** 0.102 BPP | **WebP** 0.168 BPP (66% bigger) | **Ours** 0.101 BPP

Figure 1. Examples of reconstructions by different codecs for very low bits per pixel (BPP) values. The uncompressed size is 24 BPP, so the examples represent compression by around 250 times. We reduce the bitrates of other codecs by their header lengths for fair comparison. For each codec, we search over bitrates and present the reconstruction for the smallest BPP above ours. WebP and JPEG were not able to produce reconstructions for such low BPP: the reconstructions presented are for the smallest bitrate they offer. More examples can be found in the appendix.

Compression - Lossy

JPEG

JPEG2000

WaveOne

[Rippel & Bourdev, 2017]

Генеративная модель

Данные из некоторого распределения

Сгенерировать новые данные ~ это распределение

Цель: **оценивание распределения данных высокой(!) размерности (изображение, аудио, видео, текст)**

- понять структуру данных
- найти зависимости между переменными
- генерировать новые данные с теми же свойствами
- генерация новых признаков без учителя

Генеративные модели

подходы:

1) максимизация правдоподобия

$$\prod_{x \in \text{train}} p_{\text{model}}(x;\theta) \to \max$$

2) сделать распределение похожим на данные

$$D_{\text{KL}}(p_{\text{data}} \,\|\, p_{\text{model}}) = \int p_{\text{data}}(x) \ln \frac{p_{\text{model}}(x)}{p_{\text{data}}(x)} \partial x \to \min$$

картинка – из точек плотность

Генеративные модели: создание «сырых данных»

Понимание семантики (сжатие, аномалии, …)

Проблема оценки плотности

Как оценить плотность в сложном пространстве

Пример: изображение размера 128×128×3

- **проблема проклятия размерности**
- **быстро вычислять плотность**
- **быстро сэмплировать**
- **проблема обобщения**

Решения для оценки плотности

1. Непараметрические – Гистограммы

2. Параметрические – MLE

$$\sum_{i=1}^{m} -\log p_\theta(x_i) \to \min$$

это эквивалентно

$$\mathrm{KL}(p_{\text{data}} \,\|\, p_\theta) = E_{x\sim p_{\text{data}}}[-\log p_\theta(x_i)] - H(p_{\text{data}}) \to \min$$

здесь p_{data} – эмпирическое распределение

решаем SGD

нужно эффективно вычислять производные от логарифма плотности

Решения для оценки плотности: параметрические

2.1. Авторегрессионные модели

MADE, PixelRNN/CNN, Gated PixelCNN, PixelSNAIL SPN

простые, но не распараллеливаемые

2.2. Flows

NICE, Autoregressive Flows, RealNVP, Glow, Flow++

2.3. Латентные переменные (Latent Variable Models)

Wake Sleep, VAE. IWAE, IAF-VAE

там точное правдоподобие, тут аппроксимация

(нижняя оценка логарифма правдоподобия)

это позже

Авторегрессионные модели

$$\log p_\theta(x) = \sum_{i=1}^{n} \log p_\theta(x_{[i]} \mid x_{[1:i-1]})$$

в байесовских сетях – только по родителям

$$\log p_\theta(x) = \sum_{i=1}^{n} \log p_\theta(x_{[i]} \mid \text{parents}(x_{[i]}))$$

упростили ситуацию – O(n) – параметров

можно сделать эти параметры разделяемыми:

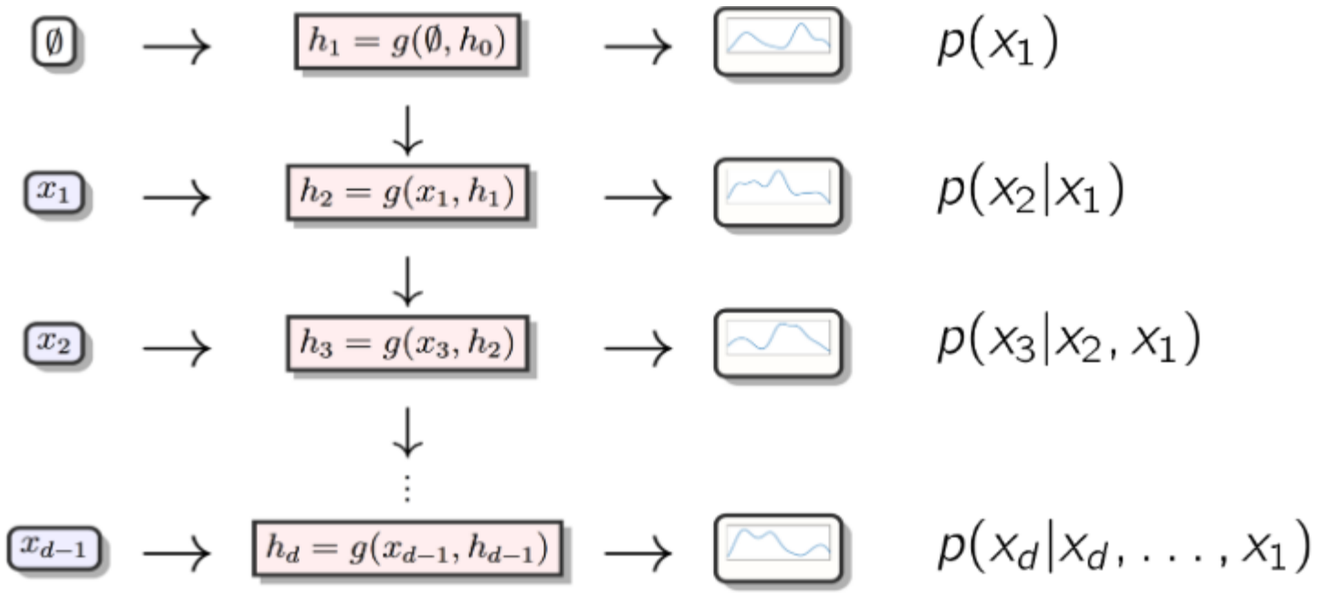
- **RNN**
- **Masking**

VBN (fully visible belief networks)

Авторегрессионные модели

$$\log p_\theta(x) = \sum_{i=1}^{n} \log p_\theta(x_{[i]} \mid x_{[1:i-1]}) \approx \sum_{i=1}^{n} \log p_\theta(x_{[i]} \mid g(x_{[1:i-1]}))$$

– тут может быть глубокая НС



Masked AutoEncoder for Distribution Estimation (MADE)

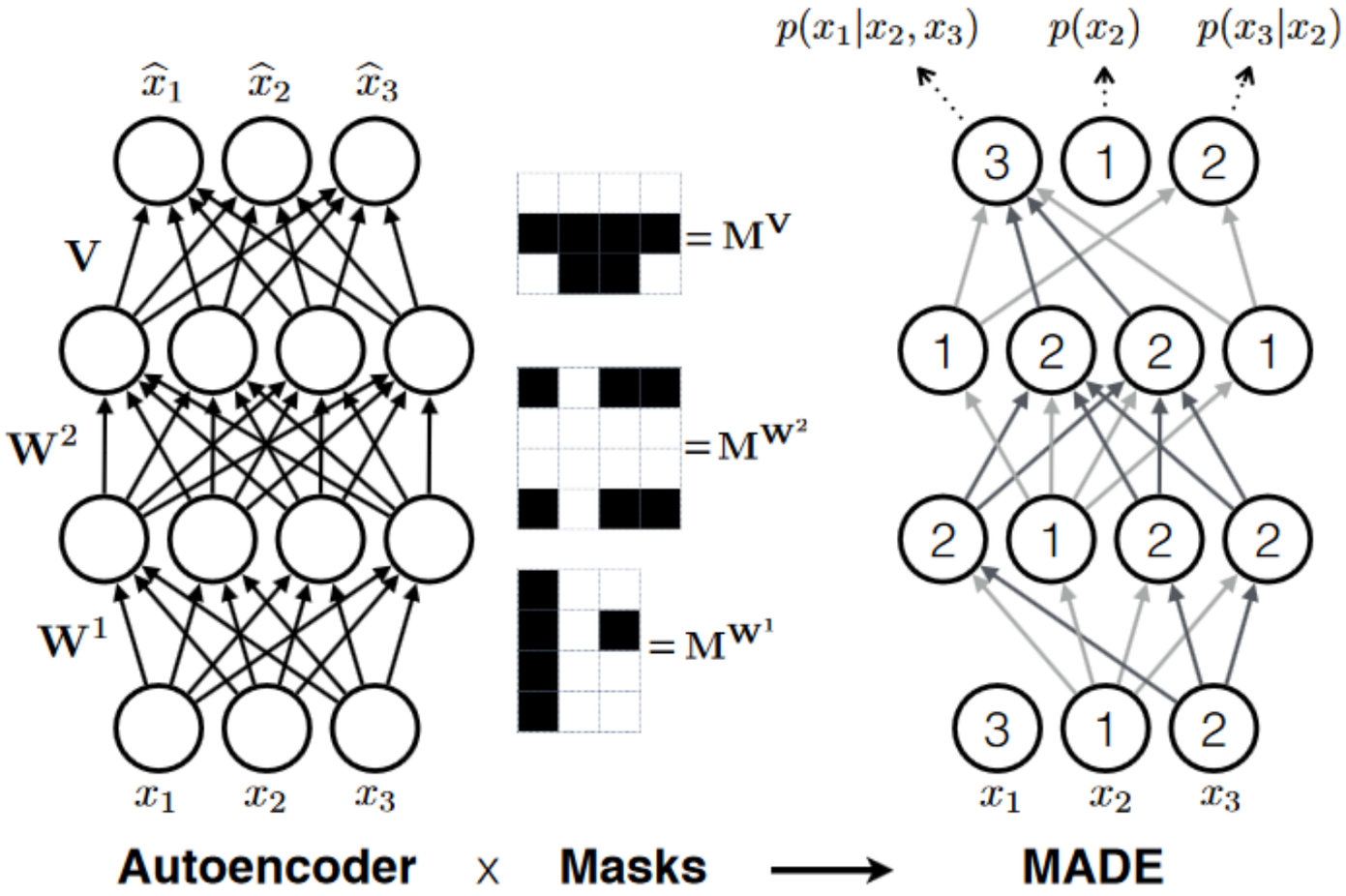


Figure 1. **Left: Conventional three hidden layer autoencoder.** Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard autoencoder cannot predict or sample new data. **Right: MADE.** The network has the same structure as the autoencoder, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks (M^{W^1}, M^{W^2}, M^V). In this example, the ordering of the input is changed from 1,2,3 to 3,1,2. This change is explained in section 4.2, but is not necessary for understanding the basic principle. The numbers in the hidden units indicate the maximum number of inputs on which the k^{th} unit of layer l depends. The masks are constructed based on these numbers (see Equations 12 and 13). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(\mathbf{x}) = p(x_2)\, p(x_3|x_2)\, p(x_1|x_2, x_3)$. Connections in light gray correspond to paths that depend only on 1 input, while the dark gray connections depend on 2 inputs.

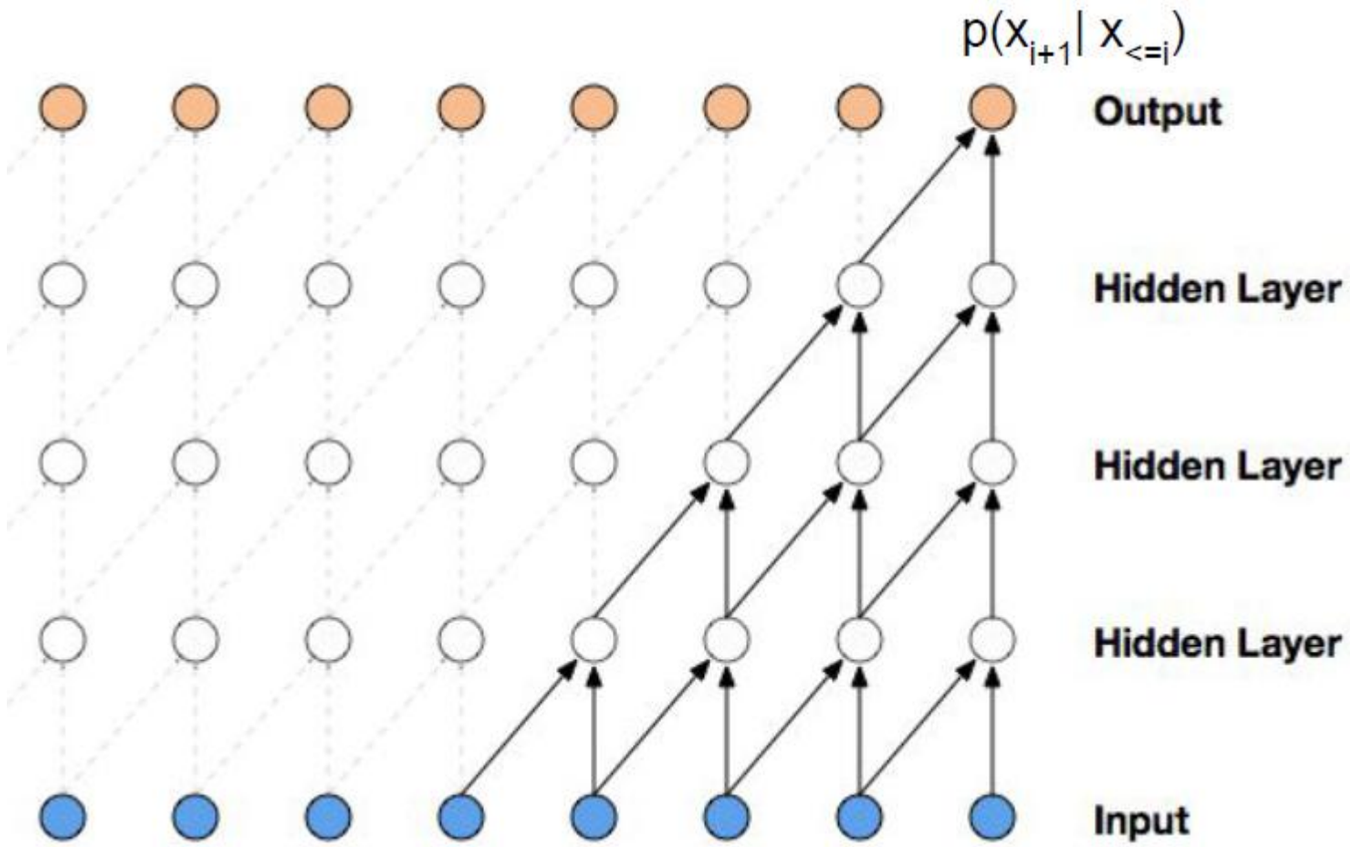
Germain, Mathieu, et al. «Made: Masked autoencoder for distribution estimation» International Conference on Machine Learning. 2015. https://arxiv.org/pdf/1502.03509.pdf

Masked Autoencoder for Distribution Estimation (MADE)

Для однослойного:

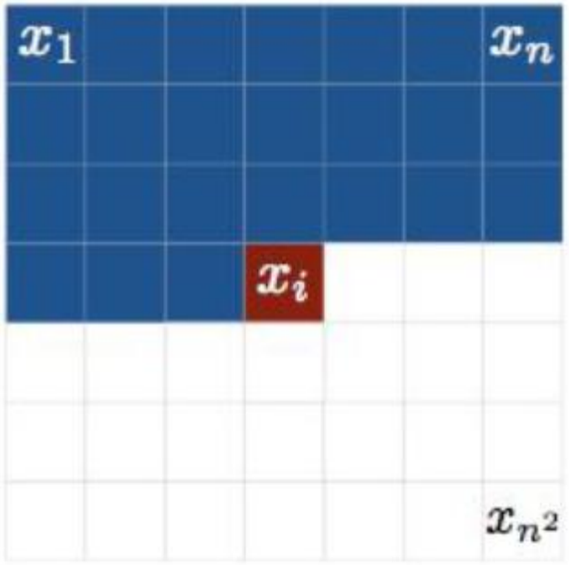
$$\begin{aligned} \mathbf{h}(\mathbf{x}) &= \mathbf{g}(\mathbf{b} + (\mathbf{W} \odot \mathbf{M}^{\mathbf{W}})\mathbf{x}) \\ \hat{\mathbf{x}} &= \text{sigm}(\mathbf{c} + (\mathbf{V} \odot \mathbf{M}^{\mathbf{V}})\mathbf{h}(\mathbf{x})) \end{aligned}$$

Masked Temporal (1D) Convolution



Masked Spatial (2D) Convolution

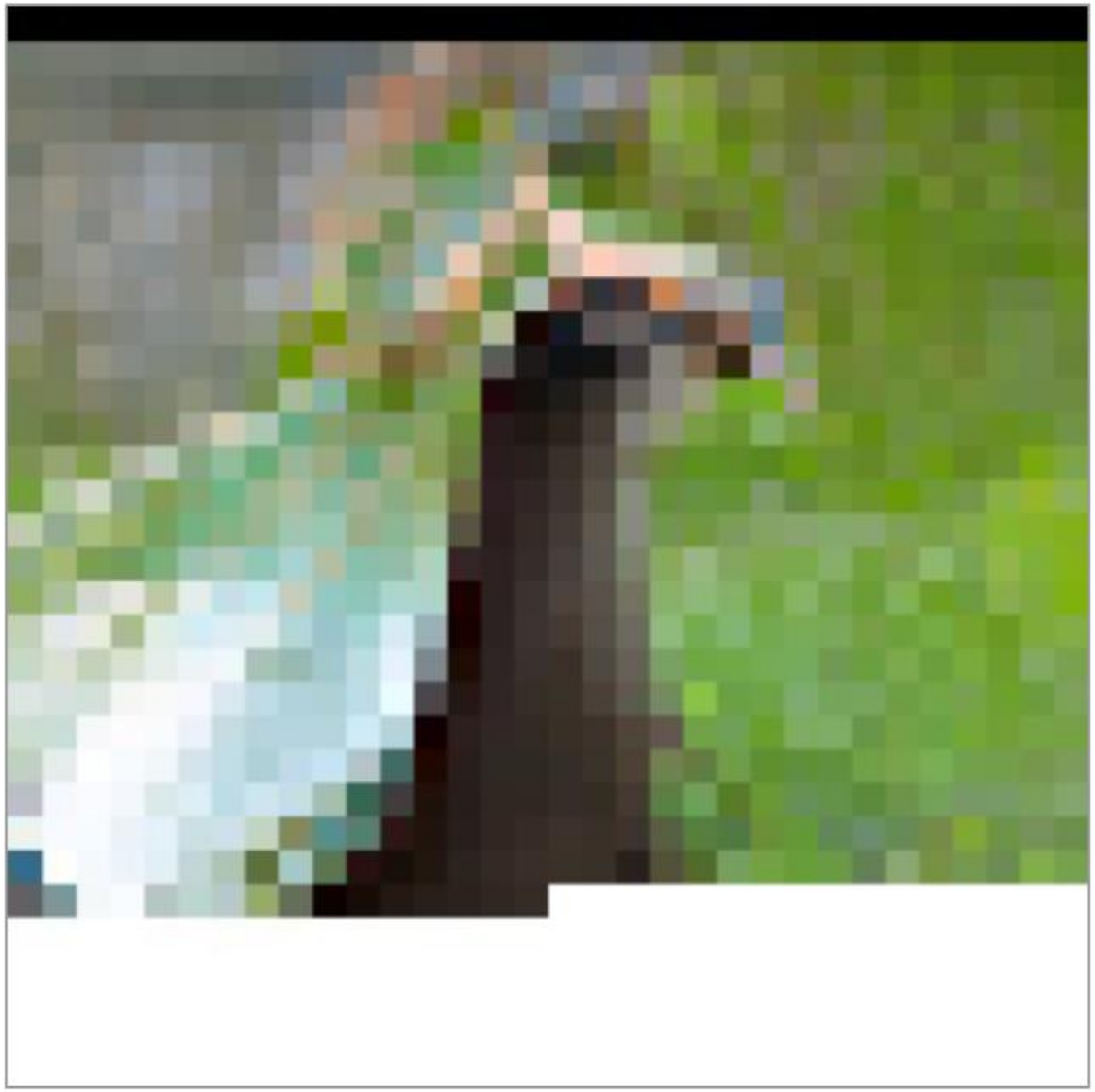
Идея пиксельных сетей

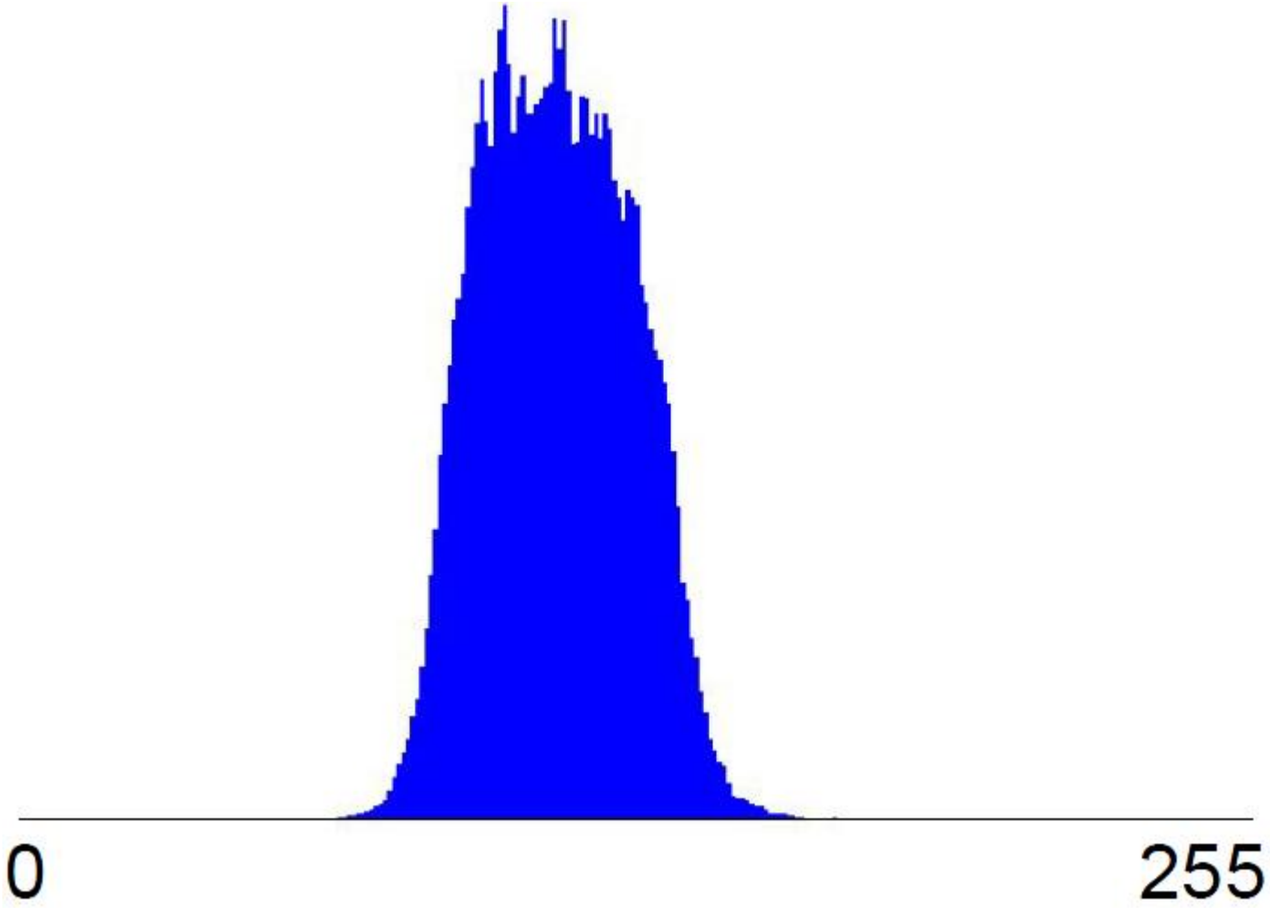


Можно брать свёртки или рекуррентности
+ маскирование (чтобы честная генерация изображения)

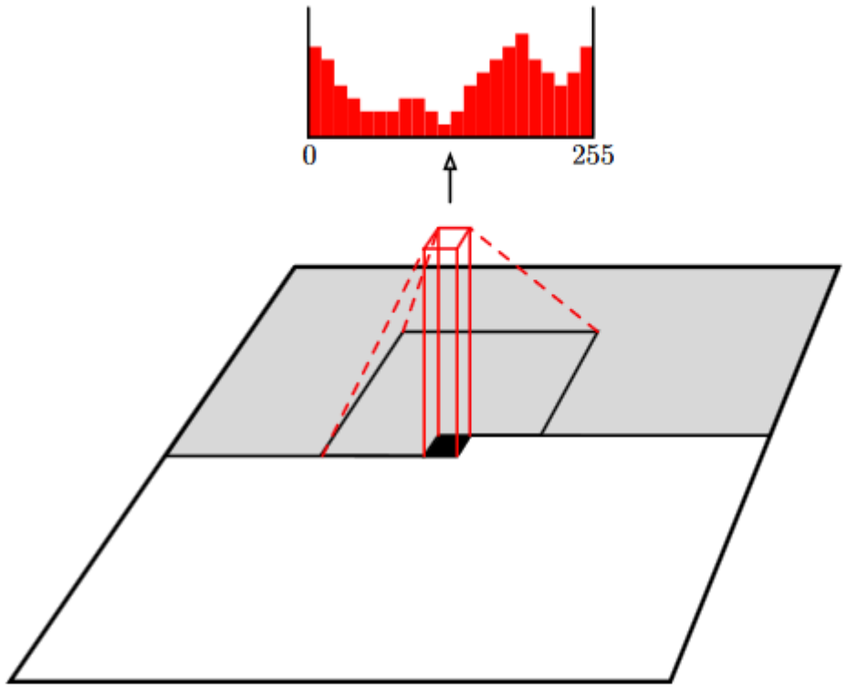
Низкая скорость генерации… но есть улучшения

Softmax-сэмплирование





PixelCNN



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

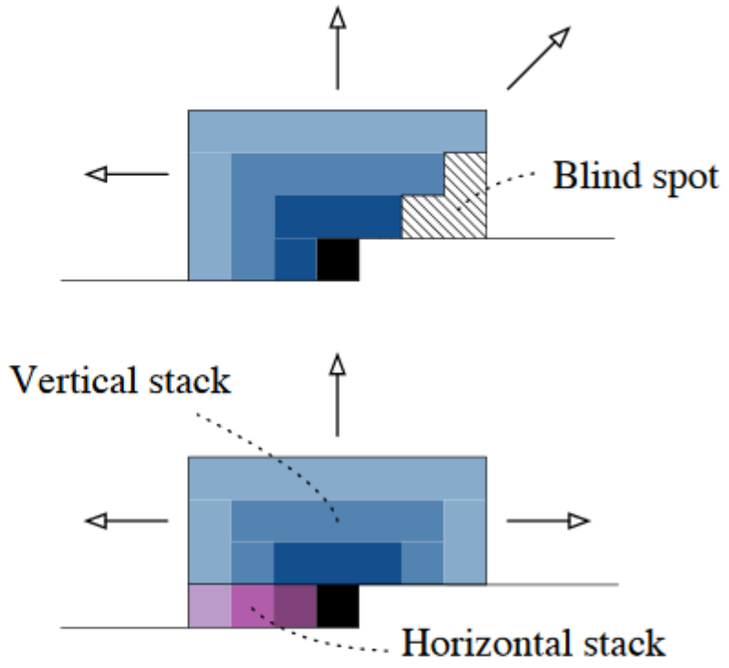


Figure 1: **Left**: A visualization of the PixelCNN that maps a neighborhood of pixels to prediction for the next pixel. To generate pixel x_i the model can only condition on the previously generated pixels $x_1, \ldots x_{i-1}$. **Middle**: an example matrix that is used to mask the 5x5 filters to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. **Right**: Top: PixelCNNs have a *blind spot* in the receptive field that can not be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field.

Lhasa Apso (dog)

Lawn mower

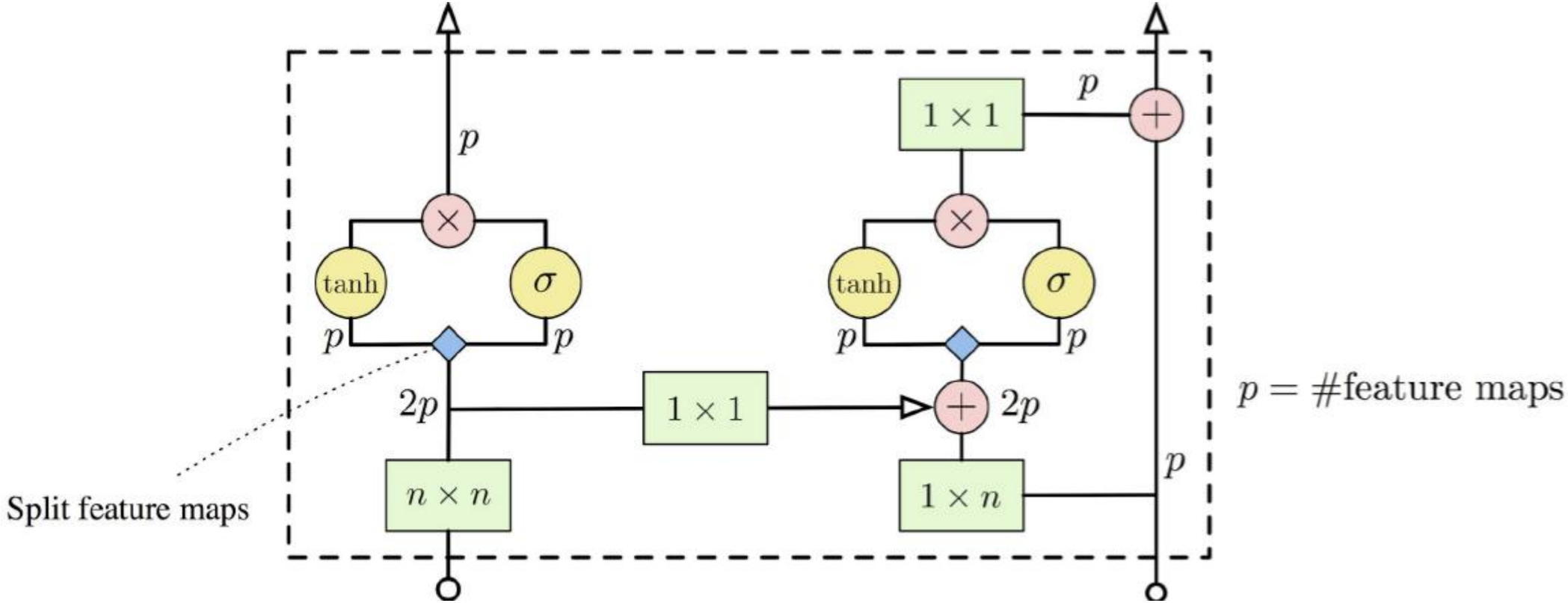
Brown bear

Robin (bird)

Figure 8: Class-Conditional 32 × 32 samples from the Conditional Pixel CNN.

Gated PixelCNN

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x}) \odot \sigma(W_{k,g} * \mathbf{x})$$



Van den Oord, Aaron, et al. «Conditional image generation with pixelcnn decoders." Advances in Neural Information Processing Systems. 2016. https://arxiv.org/pdf/1606.05328.pdf

PixelCNN++

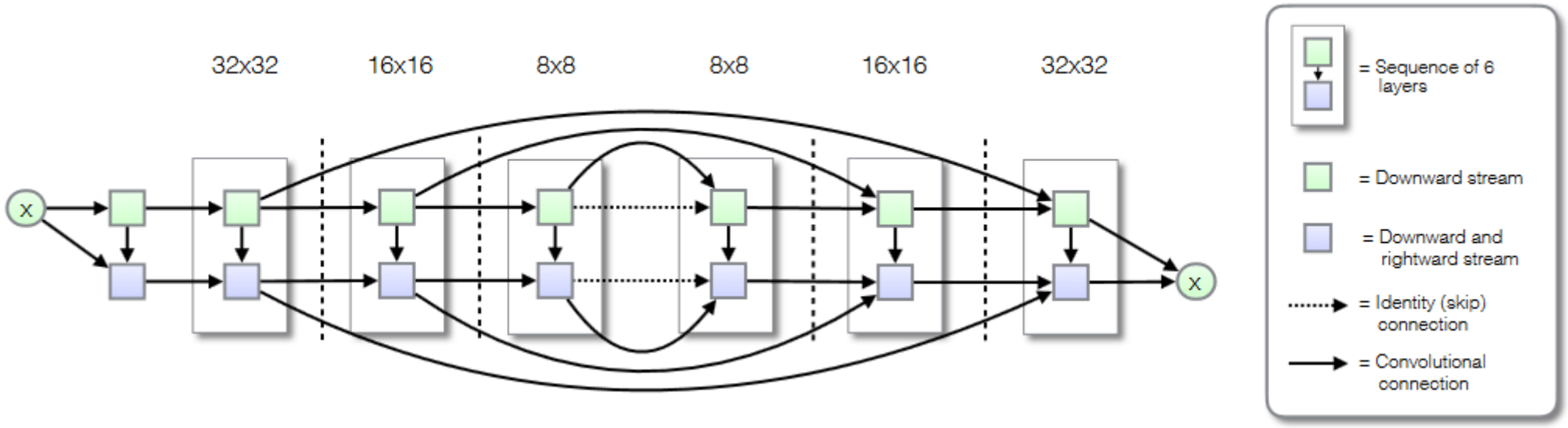


Figure 2: Like van den Oord et al. (2016c), our model follows a two-stream (downward, and downward+rightward) convolutional architecture with residual connections; however, there are two significant differences in connectivity. First, our architecture incorporates downsampling and upsampling, such that the inner parts of the network operate over larger spatial scale, increasing computational efficiency. Second, we employ long-range skip-connections, such that each k-th layer provides a direct input to the $(K-k)$-th layer, where K is the total number of layers in the network. The network is grouped into sequences of six layers, where most sequences are separated by downsampling or upsampling.

Salimans, Tim, et al. «Pixelcnn++: Improving the pixel cnn with discretized logistic mixture likelihood and other modifications.» https://arxiv.org/pdf/1701.05517.pdf

PixelSNAIL

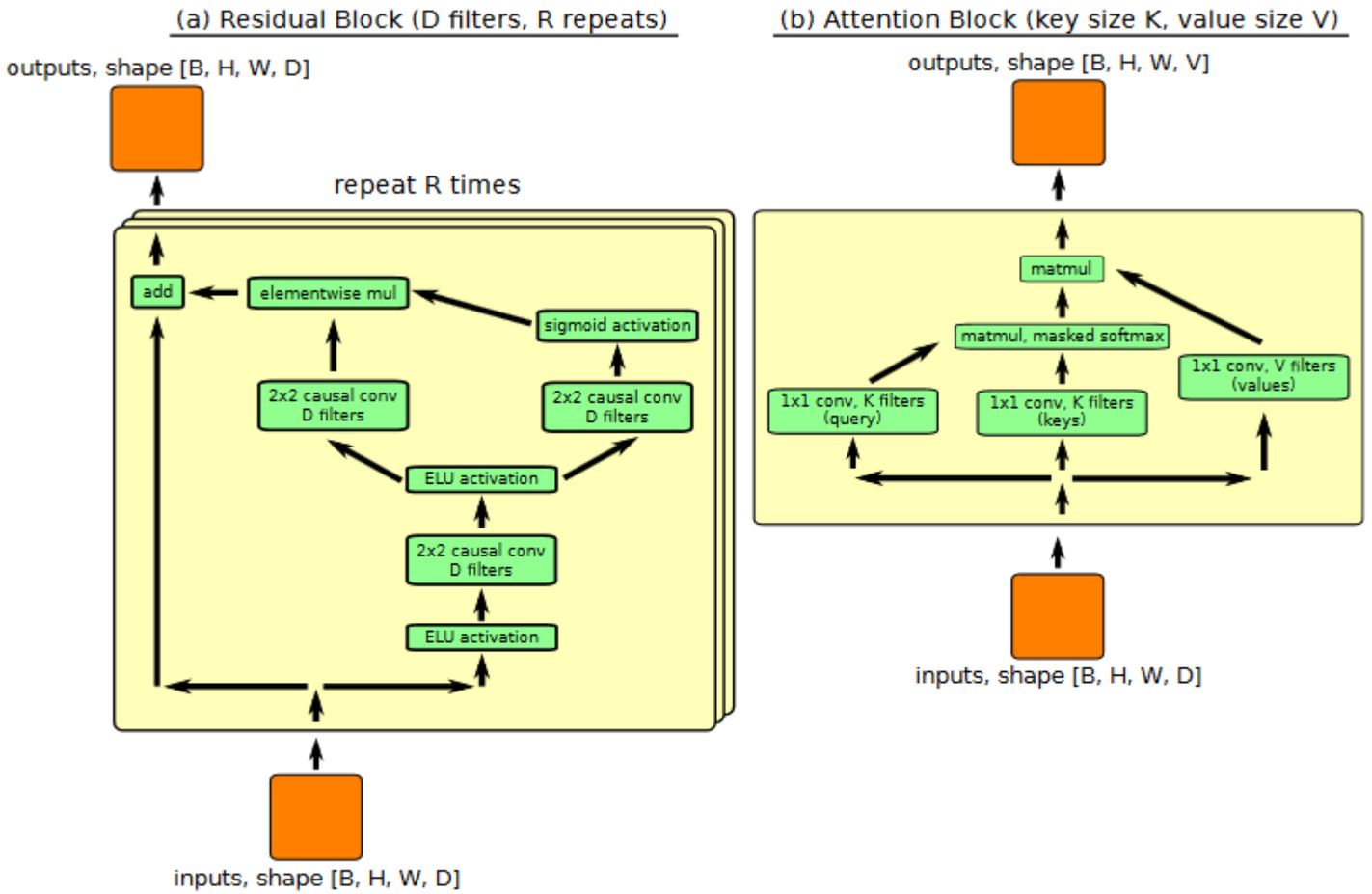


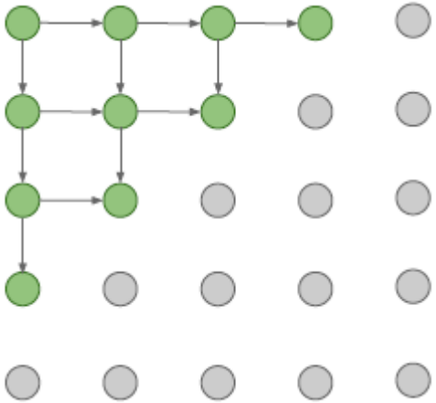
Figure 1: The modular components that make up PixelSNAIL: (a) a residual block, and (b) an attention block. For both datasets, we used residual blocks with 256 filters and 4 repeats, and attention blocks with key size 16 and value size 128.

Chen, Xi, et al. «Pixelsnail: An improved autoregressive generative model»

https://arxiv.org/pdf/1712.09763.pdf

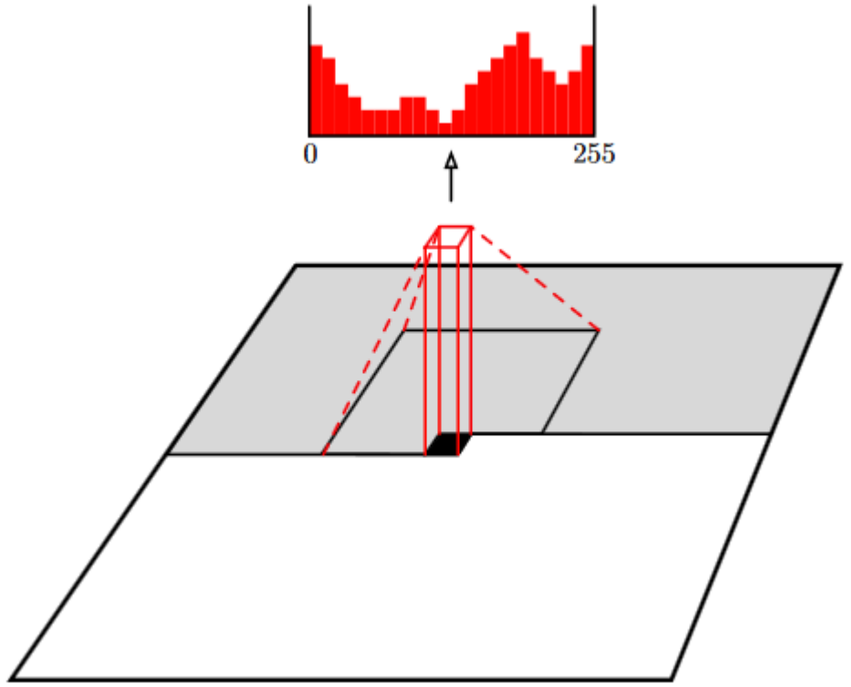
PixelRNN

Генерация изображения из угла (каждый пиксель зависит от предыдущих)



PixelCNN

Генерация изображения из угла (каждый пиксель ~ CNN от предыдущего региона)



PixelRNN

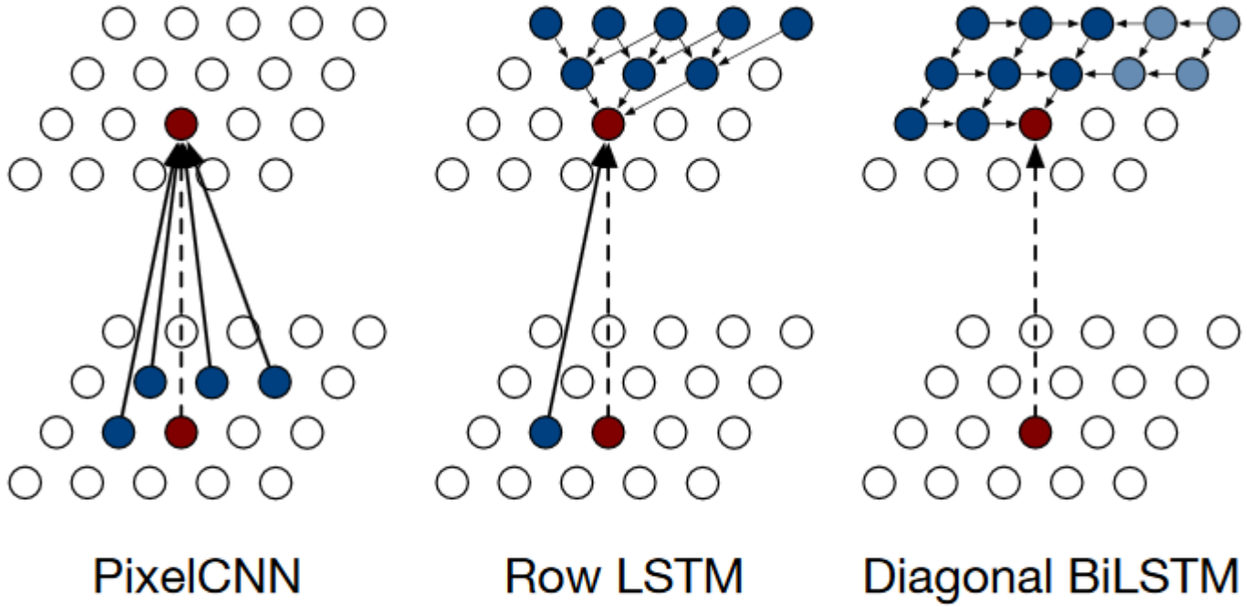


Figure 4. Visualization of the input-to-state and state-to-state mappings for the three proposed architectures.

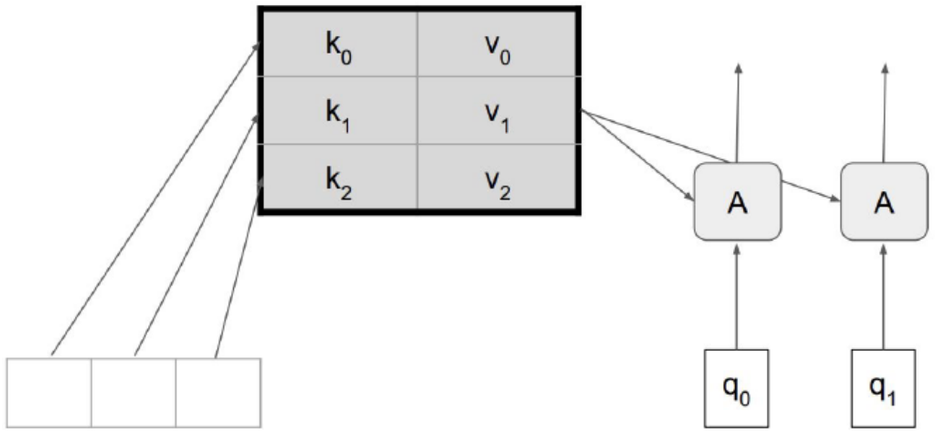
PixelCNN	PixelRNN – Row LSTM	PixelRNN – Diagonal BiLSTM
Full dependency field	Triangular receptive field	Full dependency field
Fastest	Slow	Slowest
Worst log-likelihood	-	Best log-likelihood

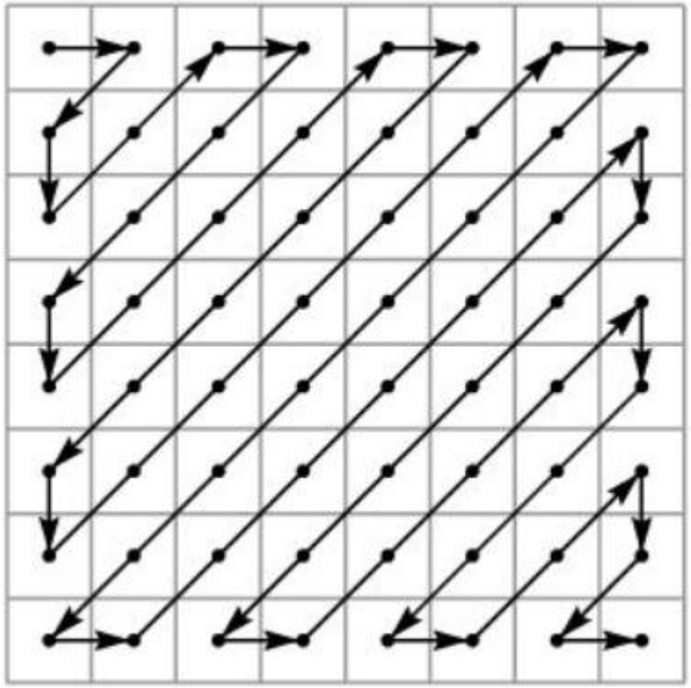
Aaron van den Oord Pixel et al «Recurrent Neural Networks» https://arxiv.org/pdf/1601.06759.pdf

Masked Attention

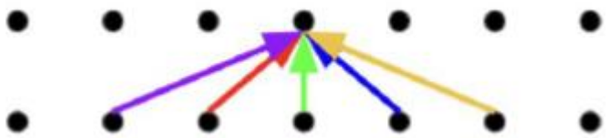
Dot-Product Attention

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i} \text{ - masked}(k_i, q) * 10^{10}}{\sum_j e^{q \cdot k_j} \text{ - masked}(k_i, q) * 10^{10}} v_i$$

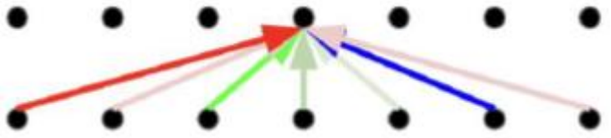




можем легко организовать любой авторегрессионный порядок!

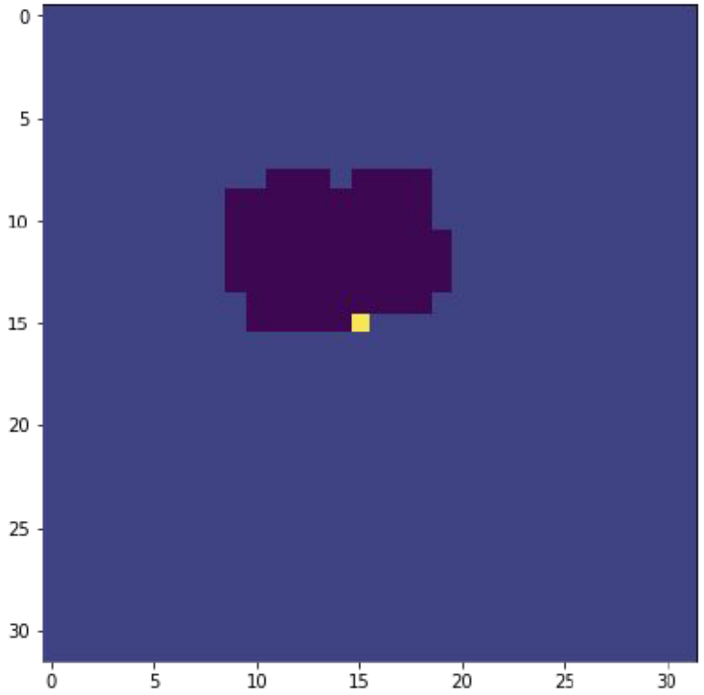


свёртка

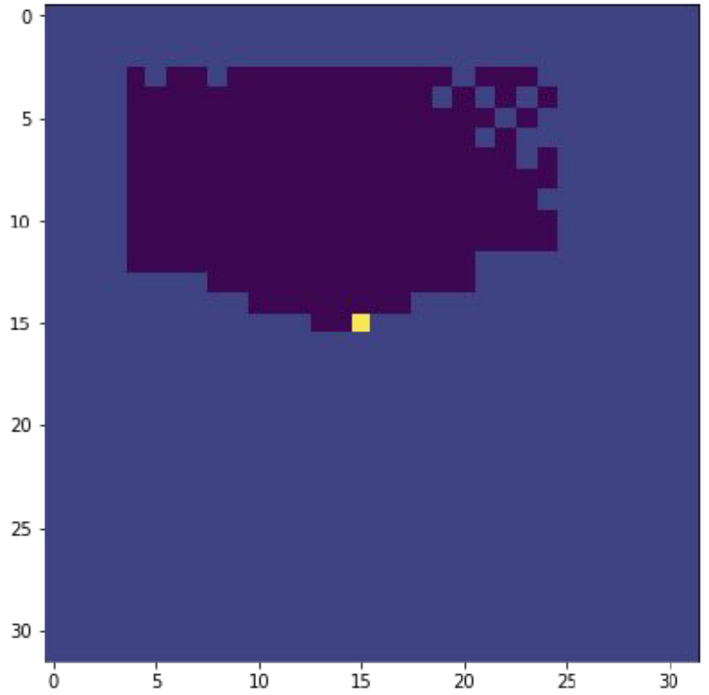


self-attention

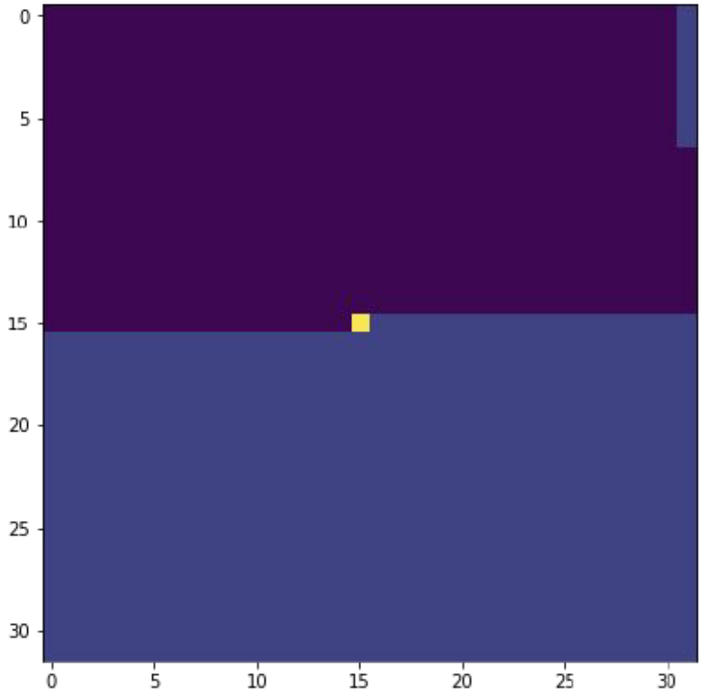
Masked Attention + Convolution



Gated PixelCNN



PixelCNN++

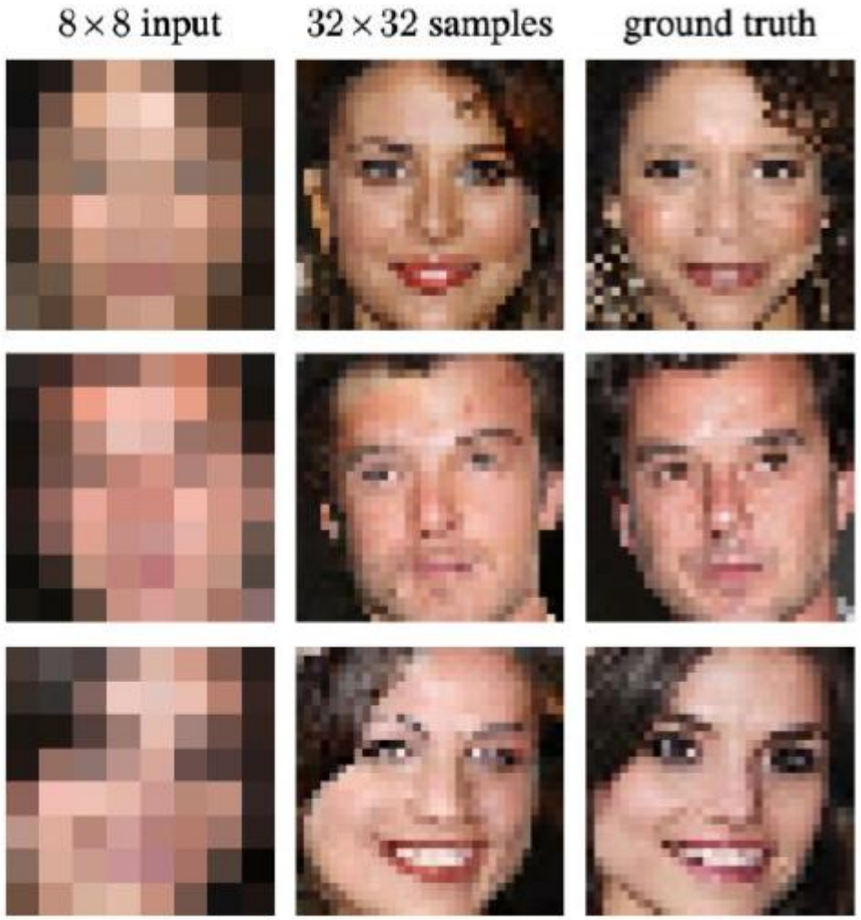


PixelSNAIL

Hierarchical Autoregressive Models with Auxiliary Decoders

De Fauw, Jeffrey, Sander Dieleman, and Karen Simonyan «Hierarchical autoregressive image models with auxiliary decoders» arXiv preprint arXiv:1903.04933 (2019)

Pixel Recursive Super Resolution

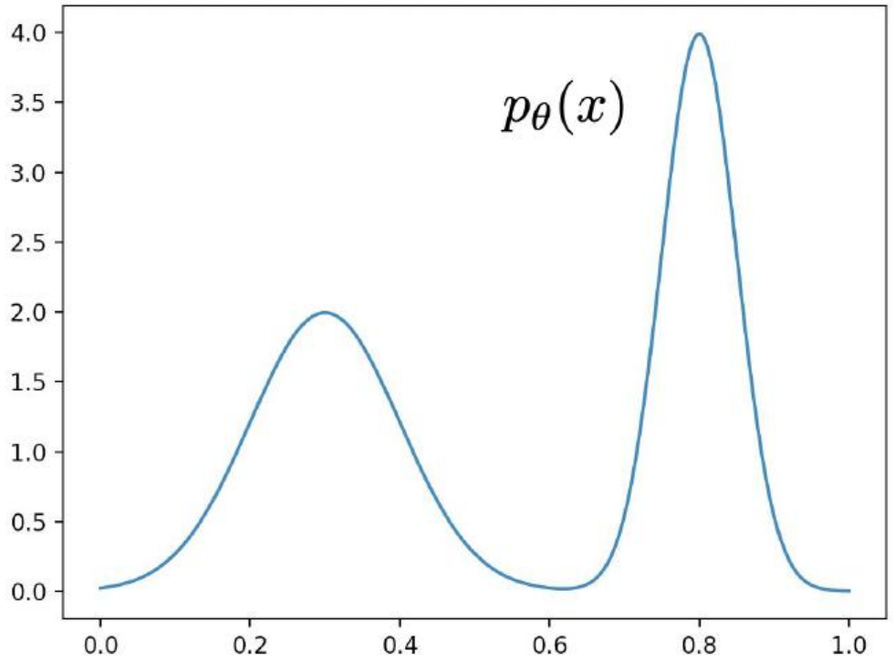


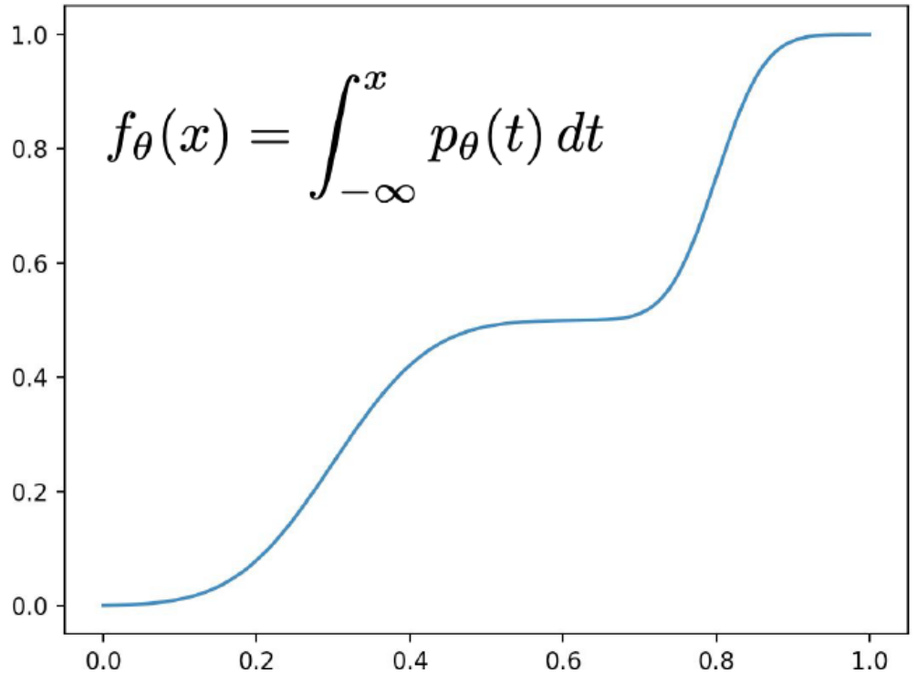
Видео

Dirk Weissenborn, Oscar Tackstrom, Jakob Uszkoreit «Scaling Autoregressive Video Models» arXiv 1906.02634 (2019)

Wilson Yan, Jonatha Ho, Pieter Abbeel. «Natural Image Manipulation for Autoregressive Models using Fisher Scores» arXiv 1912.05015

Потоки (Flows): идея





CDF переводит область определения (распределение) на отрезок [0, 1]

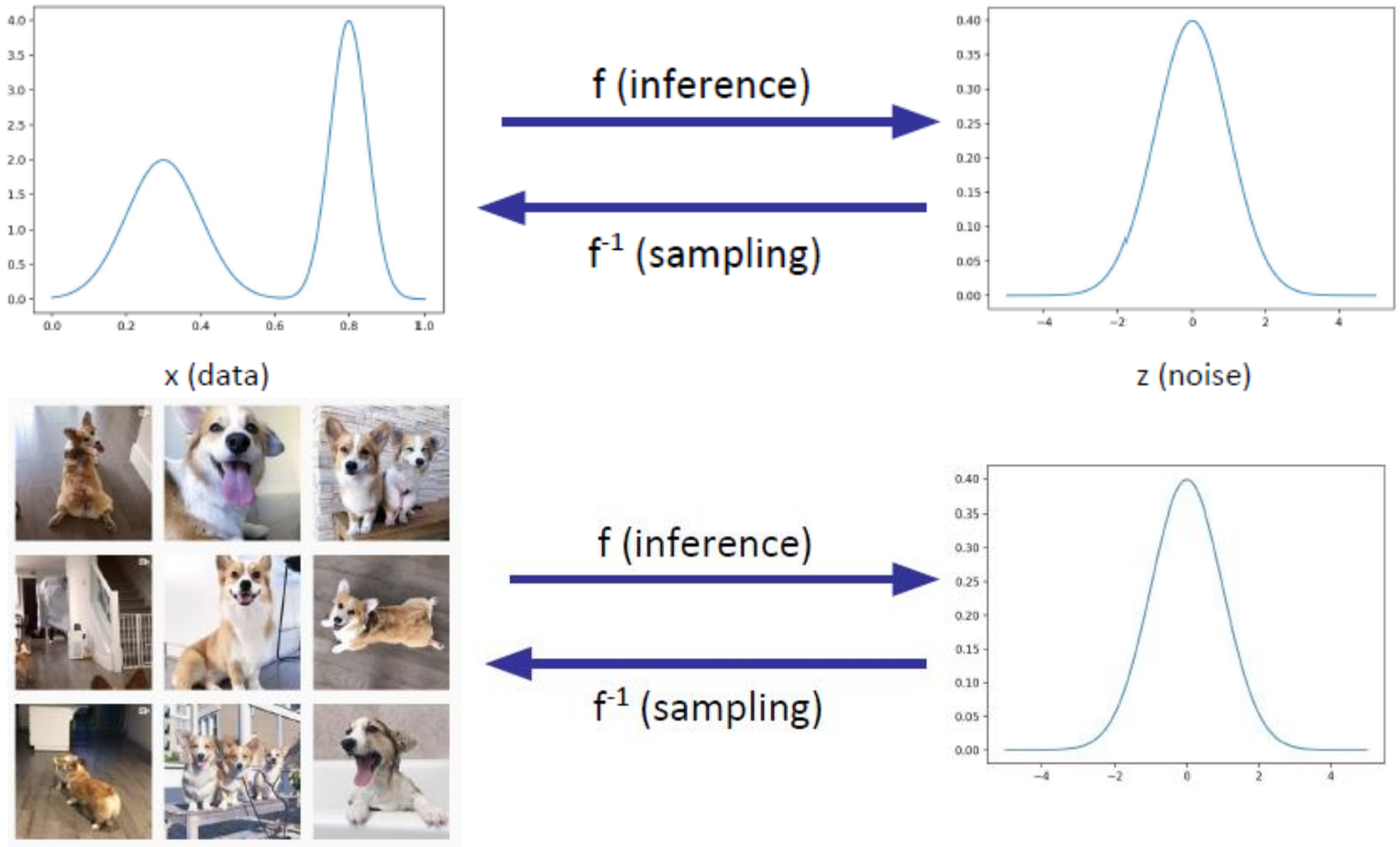
как мы сэмплируем…

$$z = f(x)$$
$$z \sim U[0,1]$$
$$x = f_\theta^{-1}(z)$$

CDF ~ обратимое дифференцируемое отображение data → [0, 1]

вместо плотности можно сразу учить соответствующие отображения

Потоки (Flows)



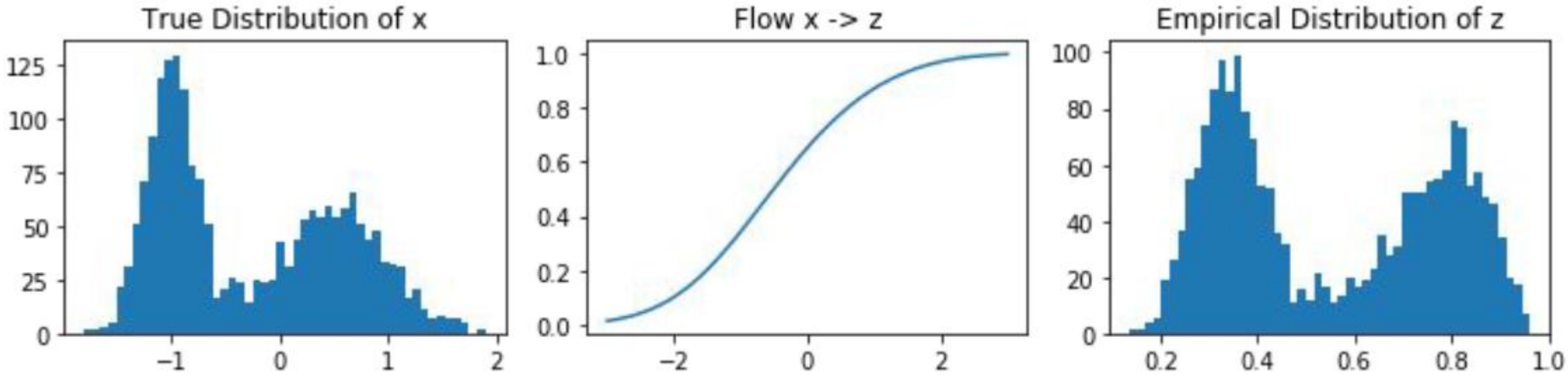
– дифференцируемое обратимое отображение data → шум

переводим распределение данных в базовое (обычно нормальное или равномерное)

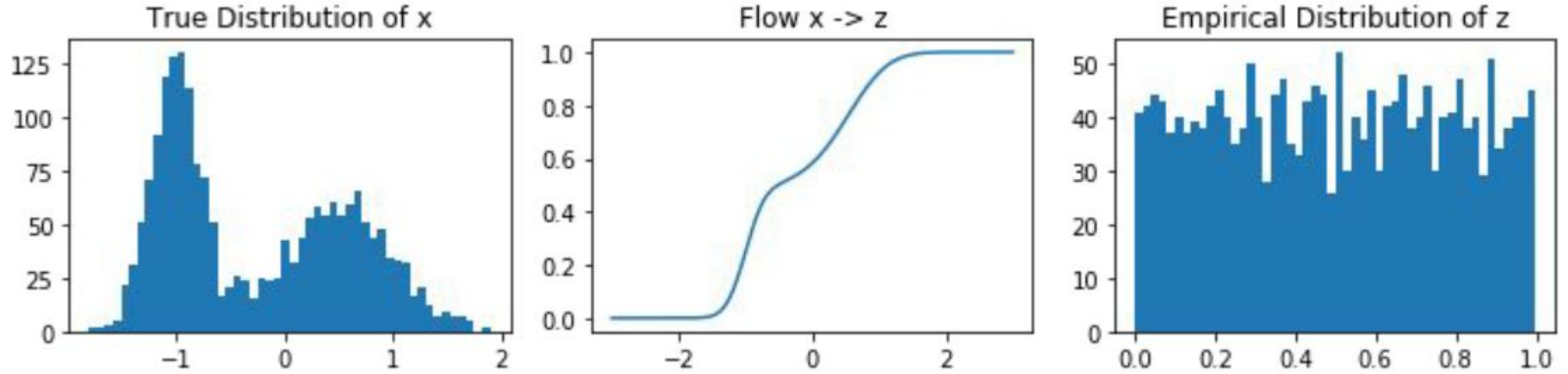
x и z должны быть одной размерности

Потоки (Flows): пример

до обучения



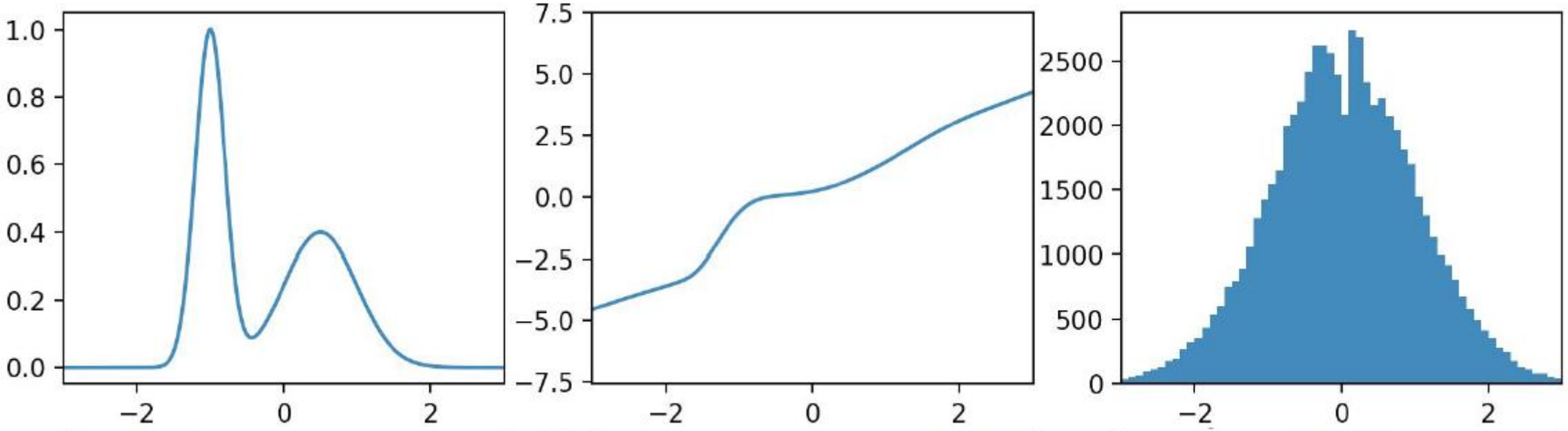
после обучения – равномерное распределение



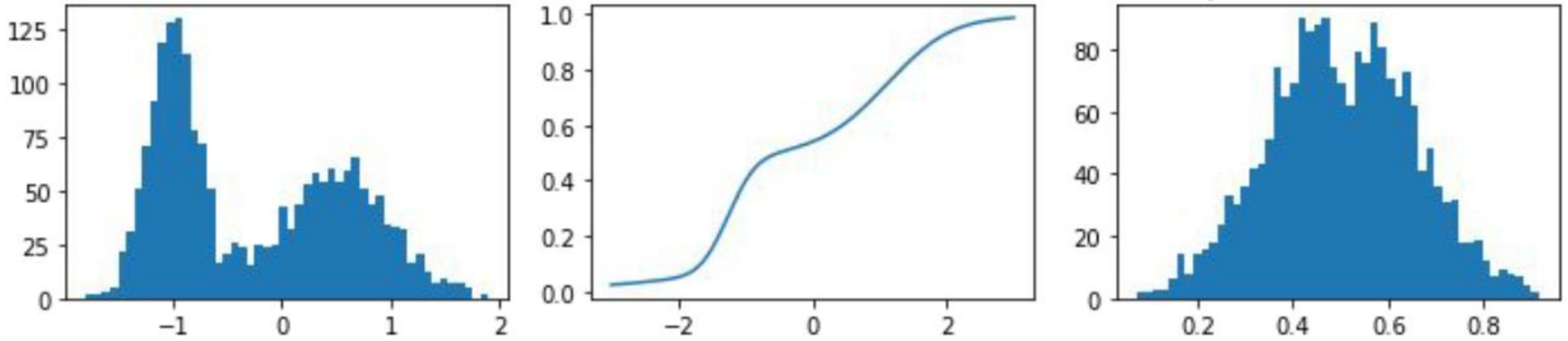
https://sites.google.com/view/berkeley-cs294-158-sp20/home

Потоки (Flows): пример

после обучения – нормальное распределение



после обучения – бета-распределение



Потоки (Flows)

$$p_\theta(x) = p(f_\theta(x))\left|\det\frac{\partial f_\theta(x)}{\partial x}\right|$$

z

ММП

$$\mathbf{E}_x\left[-\log p(f_\theta(x)) - \log\left|\det\frac{\partial f_\theta(x)}{\partial x}\right|\right] \to \min$$

нужно быстро вычислять Якобиан…

Пусть есть композиция (для повышения выразимости):

$$x \to f_1 \to f_2 \to \ldots \to f_k \to z$$

$$\log p_\theta(x) = \log p(f_\theta(x)) + \sum_{i=1}^{k}\log\left|\det\frac{\partial f_i}{\partial f_{i-1}}\right|$$

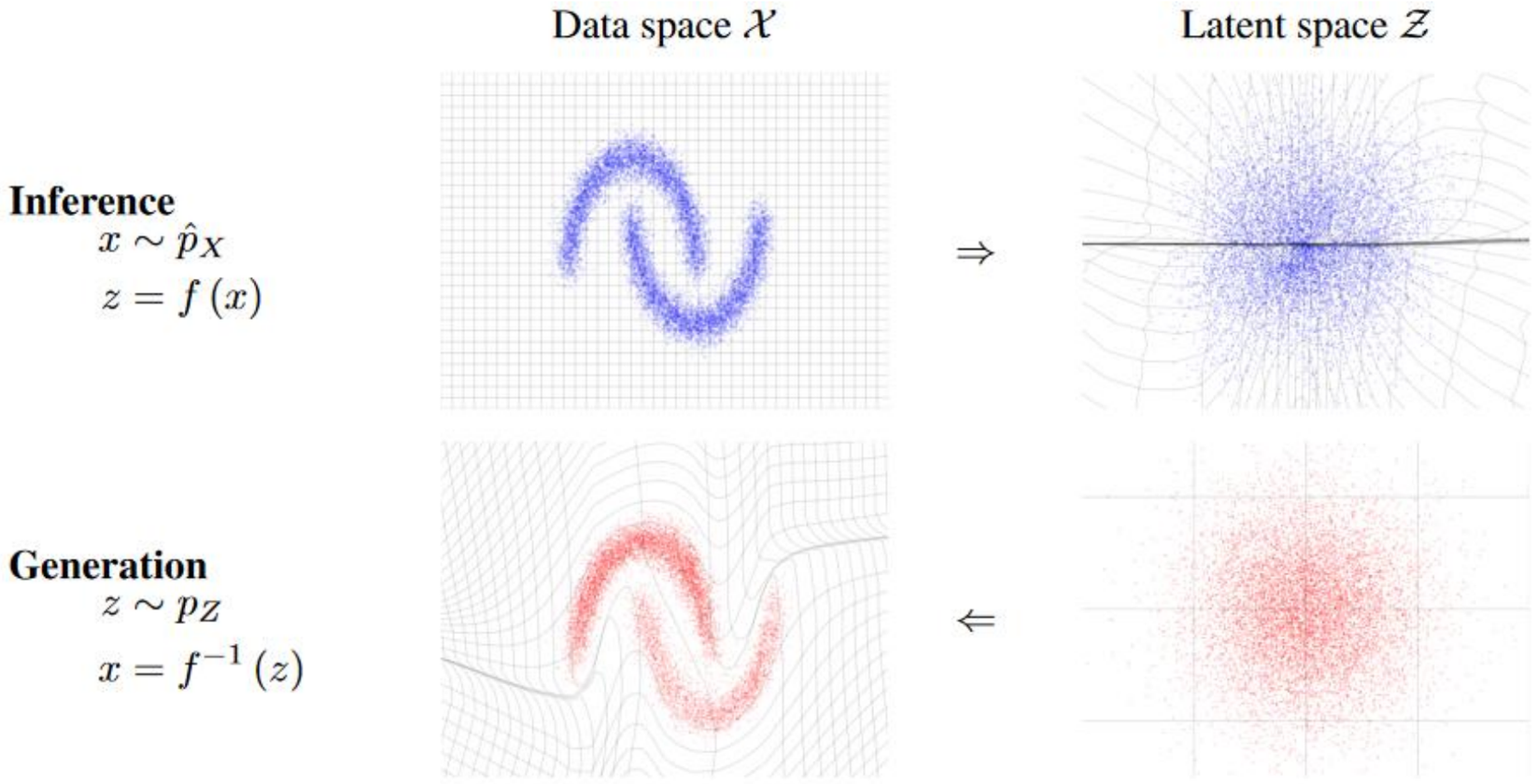


Figure 1: Real NVP learns an invertible, stable, mapping between a data distribution $\hat{p}_X$ and a latent distribution p_Z (typically a Gaussian). Here we show a mapping that has been learned on a toy 2-d dataset. The function $f(x)$ maps samples x from the data distribution in the upper left into approximate samples z from the latent distribution, in the upper right. This corresponds to exact inference of the latent state given the data. The inverse function, $f^{-1}(z)$, maps samples z from the latent distribution in the lower right into approximate samples x from the data distribution in the lower left. This corresponds to exact generation of samples from the model. The transformation of grid lines in $\mathcal{X}$ and $\mathcal{Z}$ space is additionally illustrated for both $f(x)$ and $f^{-1}(z)$.

Потоки (Flows)

вход и выход одной размерности
трансформация обратима
эффективное вычисление Якобиана и его градиента

Dinh et al. «Density estimation using Real NVP» // ICLR 2017 https://arxiv.org/abs/1605.08803

NICE / RealNVP (Non-volume preserving transformations)

разделим вектор на две части

$$x = x_{1:d} = [x_{1:d/2}, x_{(d/2+1):d}]$$

пусть

$$\begin{cases} z_{1:d/2} = x_{1:d/2} \\ z_{(d/2+1):d} = x_{(d/2+1):d} \cdot s_\theta(x_{1:d/2}) + t_\theta(x_{1:d/2}) \end{cases}$$

здесь s_θ, t_θ – произвольные НС

$$\frac{\partial z}{\partial x} = \begin{bmatrix} I & 0 \\ \dfrac{\partial z_{(d/2+1):d}}{\partial x_{1:d/2}} & \operatorname{diag}(s_\theta(x_{1:d/2})) \end{bmatrix}$$

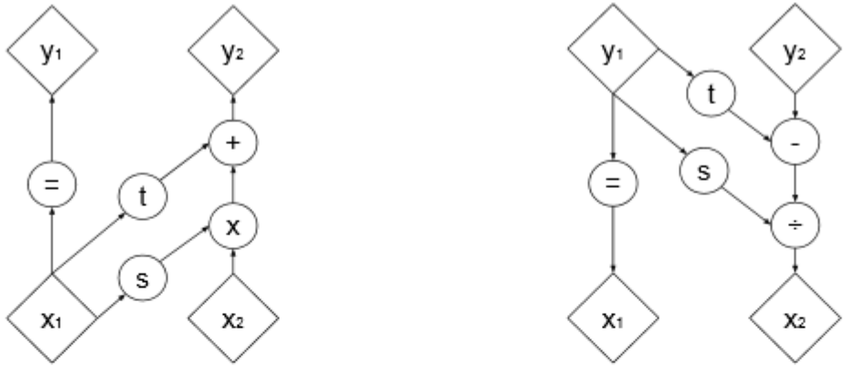
$$\det \frac{\partial z}{\partial x} = \prod_{i=1}^{d} s_\theta(x_{1:d/2})_{[i]}$$

Real NVP: «Coupling layers»

$$\begin{cases} y_{1:d} = x_{1:d/2} \\ y_{(d+1):D} = x_{(d+1):D} \cdot \exp(s(x_{1:d})) + t(x_{1:d}) \end{cases} \Leftrightarrow \begin{cases} x_{1:d/2} = y_{1:d} \\ x_{(d+1):D} = \exp(-s(x_{1:d})) \cdot (y_{(d+1):D} - t(x_{1:d})) \end{cases}$$

почти ничего не изменилось – легко вычисляем Якобиан

обратное преобразование аналогично



(a) Forward propagation (b) Inverse propagation

Figure 2: Computational graphs for forward and inverse propagation. A coupling layer applies a simple invertible transformation consisting of scaling followed by addition of a constant offset to one part $\mathbf{x}_2$ of the input vector conditioned on the remaining part of the input vector $\mathbf{x}_1$. Because of its simple nature, this transformation is both easily invertible and possesses a tractable determinant. However, the conditional nature of this transformation, captured by the functions s and t, significantly increase the flexibility of this otherwise weak function. The forward and inverse propagation operations have identical computational cost.

Real NVP: Masked convolution

разделение переменных может быть сделано с помощью масок:

$$y = b \odot x + (1 - b) \odot \left(x \odot \exp\left(s(b \odot x)\right) + t(b \odot x) \right)$$

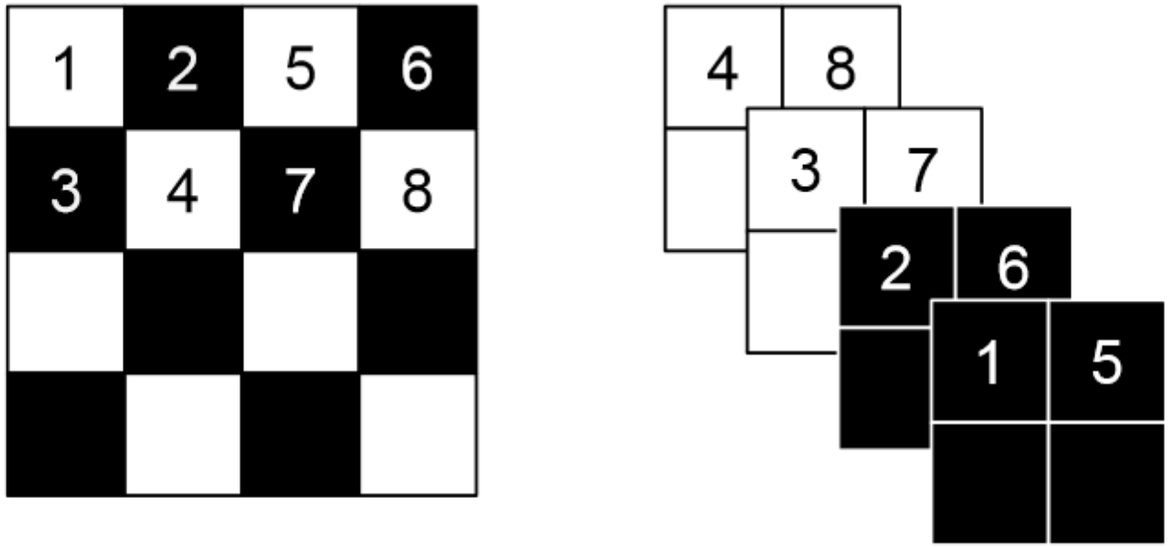
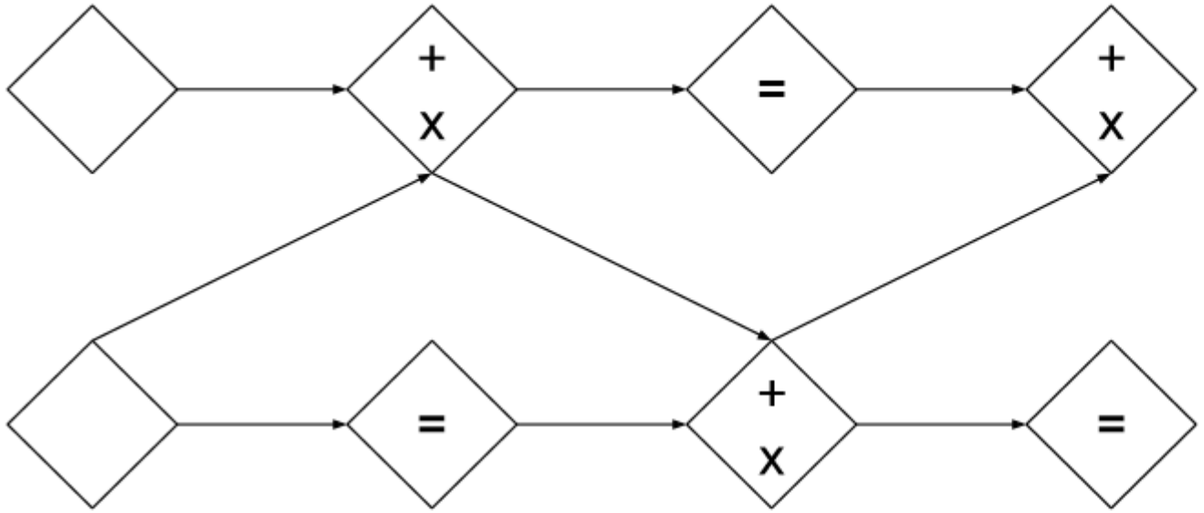
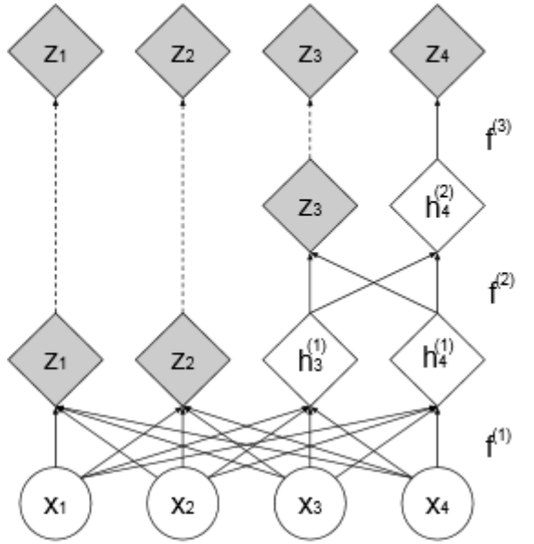


Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation, a checkerboard pattern is used for coupling layers while a channel-wise masking pattern is used afterward.

Real NVP: Combining coupling layers



(a) In this alternating pattern, units which remain identical in one transformation are modified in the next.



(b) Factoring out variables. At each step, half the variables are directly modeled as Gaussians, while the other half undergo further transformation.

Real NVP: примеры

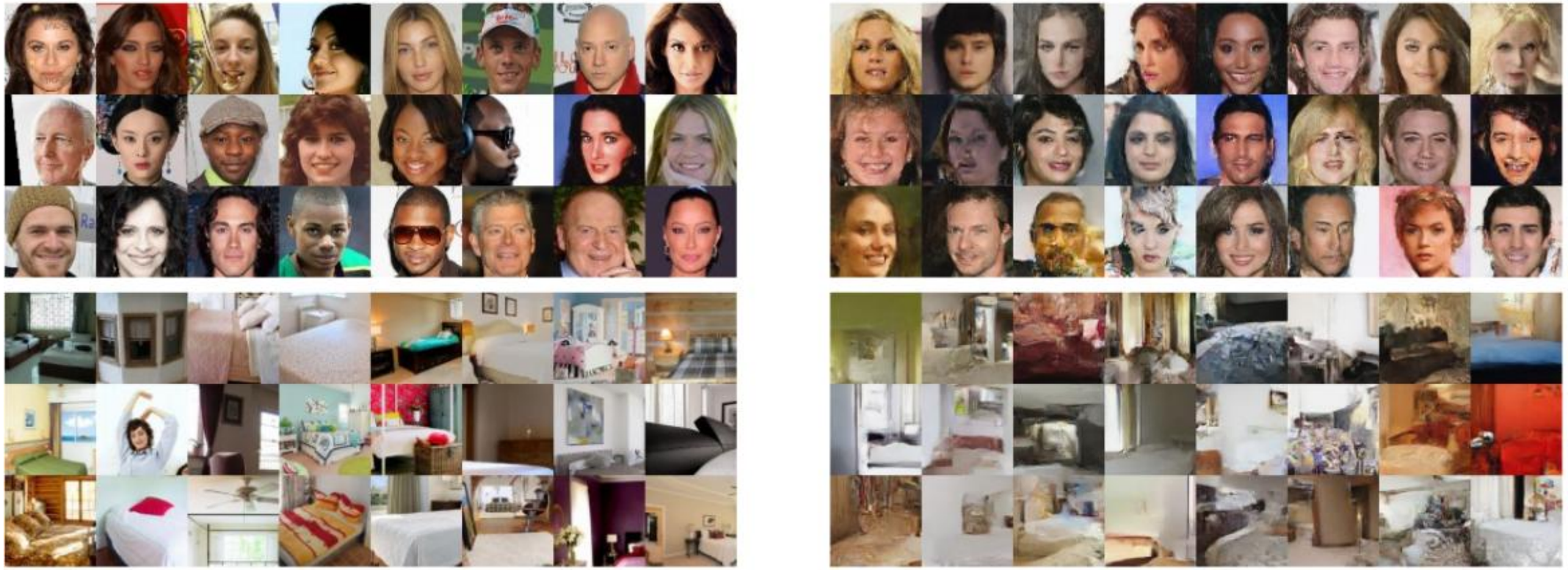


Figure 5: On the left column, examples from the dataset. On the right column, samples from the model trained on the dataset. The datasets shown in this figure are in order: CIFAR-10, Imagenet (32×32), Imagenet (64×64), CelebA, LSUN (bedroom).

Real NVP: примеры

Figure 10: Samples from a model trained on *LSUN* (*church outdoor* category).

Применение с обратимыми свёртками: Glow

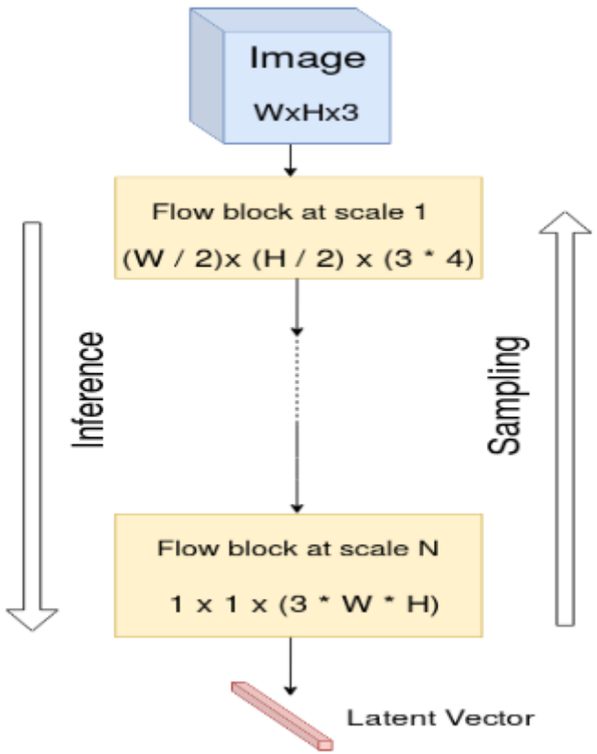
Flow-based generative models (Dinh et al., 2014)
точное значение логарифма правдоподобие
получение латентных переменных
возможность распараллеливания обучения и работы

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Figure 1: Synthetic celebrities sampled from our model; see Section 3 for architecture and method, and Section 5 for more results.

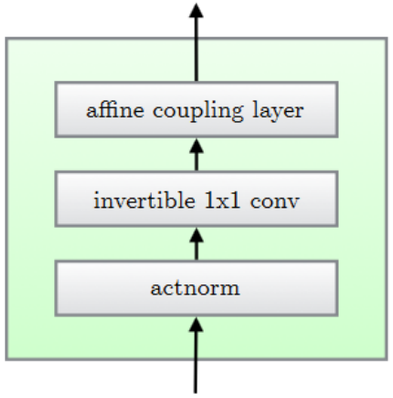
Применение с обратимыми свёртками: Glow

Идея:

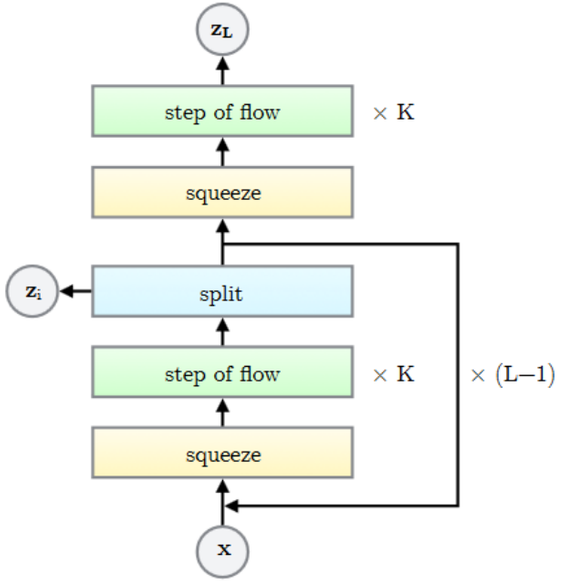


Diederik P. Kingma, Prafulla Dhariwal «Glow: Generative Flow with Invertible 1×1 Convolutions» https://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

Применение с обратимыми свёртками: Glow



(a) One step of our flow.



(b) Multi-scale architecture (Dinh et al., 2016).

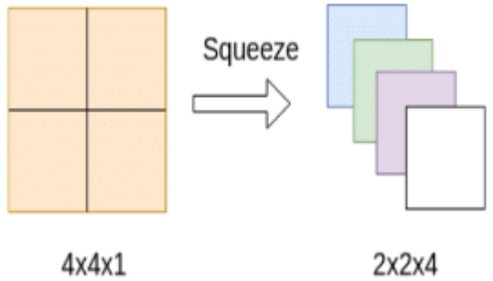
Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\mathtt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

Figure 2: We propose a generative flow where each step (left) consists of an *actnorm* step, followed by an invertible 1×1 convolution, followed by an affine transformation (Dinh et al., 2014). This flow is combined with a multi-scale architecture (right). See Section 3 and Table 1.

Применение с обратимыми свёртками: Glow

Squeeze –



Actnorm (activation normalizaton) – хотим делать batchnorm, но из-за больших изображений батчи по 1 изображению ⇒ сдвиг и нормировку делаем свою для каждого канала и обучаем

Применение с обратимыми свёртками: Glow

Invertible1×1convolution –

(Dinh et al., 2014, 2016) proposed a flow containing the equivalent of a permutation that reverses the ordering of the channels. We propose to replace this fixed permutation with a (learned) invertible 1×1 convolution, where the weight matrix is initialized as a random rotation matrix. Note that a 1×1 convolution with equal number of input and output channels is a generalization of a permutation operation.

The log-determinant of an invertible 1×1 convolution of a $h \times w \times c$ tensor $\mathbf{h}$ with $c \times c$ weight matrix $\mathbf{W}$ is straightforward to compute:

$$\log \left| \det \left(\frac{d\,\mathtt{conv2D}(\mathbf{h}; \mathbf{W})}{d\,\mathbf{h}} \right) \right| = h \cdot w \cdot \log |\det(\mathbf{W})| \tag{9}$$

The cost of computing or differentiating $\det(\mathbf{W})$ is $\mathcal{O}(c^3)$, which is often comparable to the cost computing $\mathtt{conv2D}(\mathbf{h}; \mathbf{W})$ which is $\mathcal{O}(h \cdot w \cdot c^2)$. We initialize the weights $\mathbf{W}$ as a random rotation matrix, having a log-determinant of 0; after one SGD step these values start to diverge from 0.

LU Decomposition. This cost of computing $\det(\mathbf{W})$ can be reduced from $\mathcal{O}(c^3)$ to $\mathcal{O}(c)$ by parameterizing $\mathbf{W}$ directly in its LU decomposition:

$$\mathbf{W} = \mathbf{PL}(\mathbf{U} + \mathrm{diag}(\mathbf{s})) \tag{10}$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal, $\mathbf{U}$ is an upper triangular matrix with zeros on the diagonal, and $\mathbf{s}$ is a vector. The log-determinant is then simply:

$$\log |\det(\mathbf{W})| = \mathtt{sum}(\log |\mathbf{s}|) \tag{11}$$

The difference in computational cost will become significant for large c, although for the networks in our experiments we did not measure a large difference in wallclock computation time.

In this parameterization, we initialize the parameters by first sampling a random rotation matrix $\mathbf{W}$, then computing the corresponding value of $\mathbf{P}$ (which remains fixed) and the corresponding initial values of $\mathbf{L}$ and $\mathbf{U}$ and $\mathbf{s}$ (which are optimized).

Affine Coupling Layers – см. табл

Split (Dinh et al., 2014) – splits h the input tensor into two halves along the channel dimension

Concat –reverse operation: concatenation into a single tensor

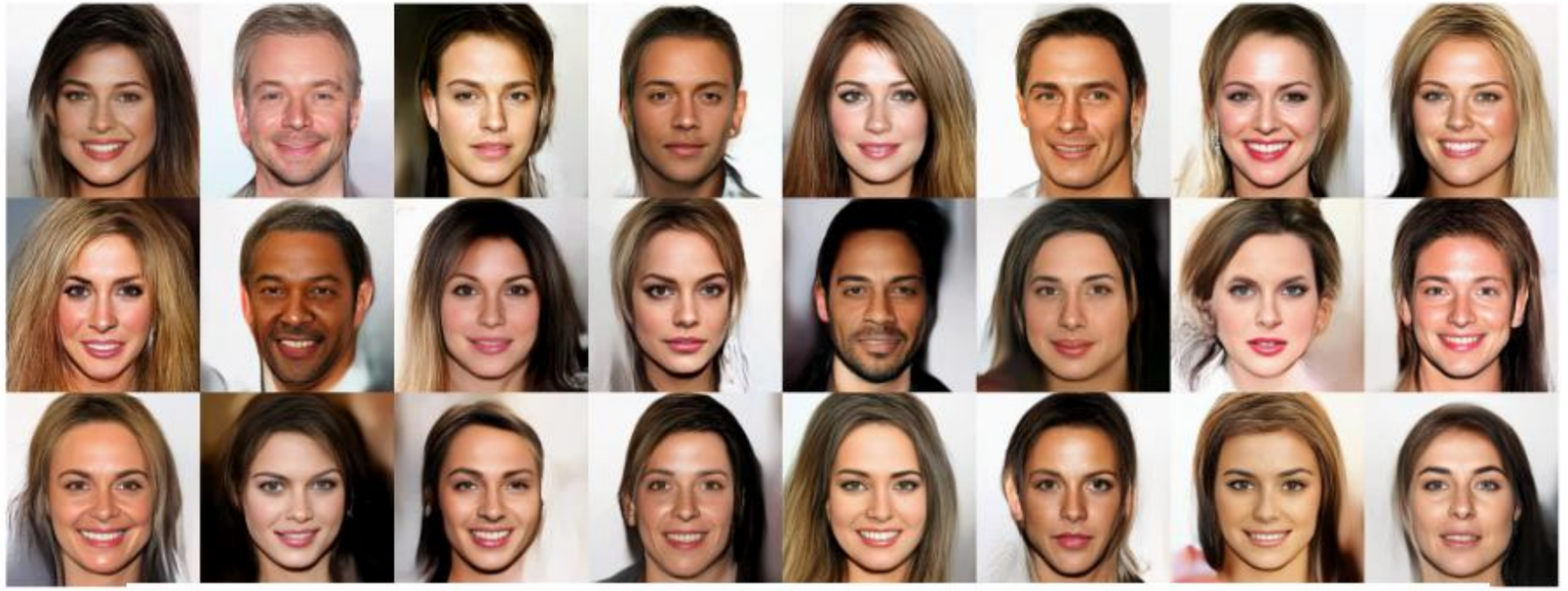


Figure 4: Random samples from the model, with temperature 0.7.

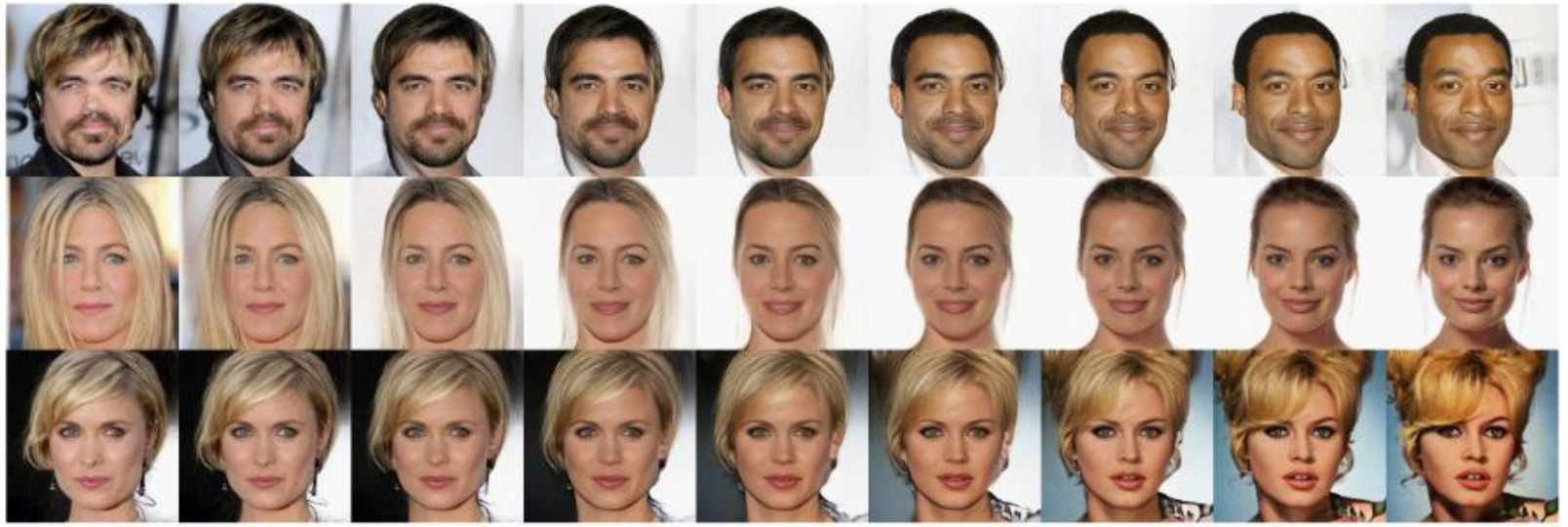


Figure 5: Linear interpolation in latent space between real images.

Continuous time flows (FFJORD)

Allows for unrestricted architectures. Invertibility and fast log probability computation guaranteed

Авторегрессионные потоки (Autoregressive Flows)

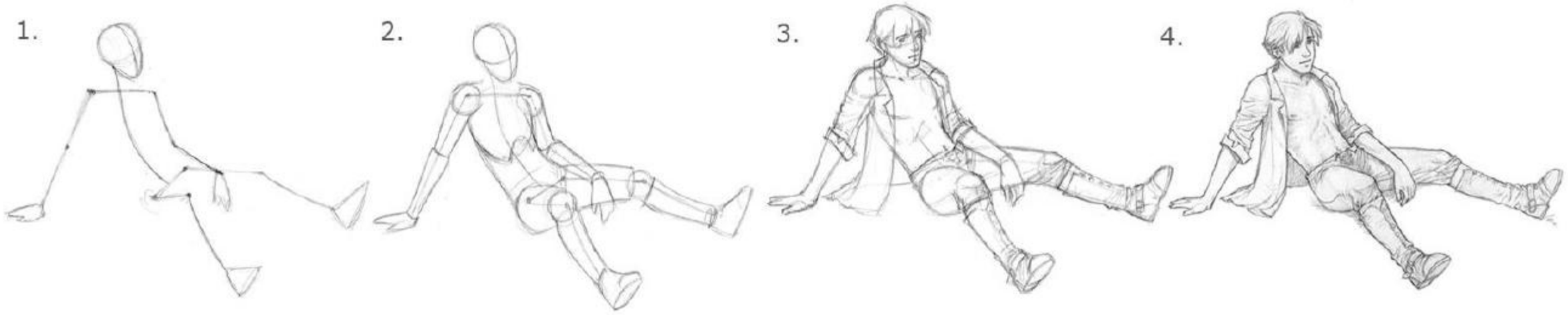
смесь авторегресии и Real NVP

$$x_1 \sim p_\theta(x_1) \qquad x_1 = f_\theta^{-1}(z_1)$$
$$x_2 \sim p_\theta(x_2 \mid x_1) \qquad x_2 = f_\theta^{-1}(z_2; x_1)$$
$$x_3 \sim p_\theta(x_3 \mid x_1, x_2) \qquad x_3 = f_\theta^{-1}(z_3; x_1, x_2)$$

George Papamakarios «Masked Autoregressive Flow for Density Estimation» //
https://arxiv.org/pdf/1705.07057.pdf

Генерация изображений с помощью RNN

Как человек рисует… генерирует изображение



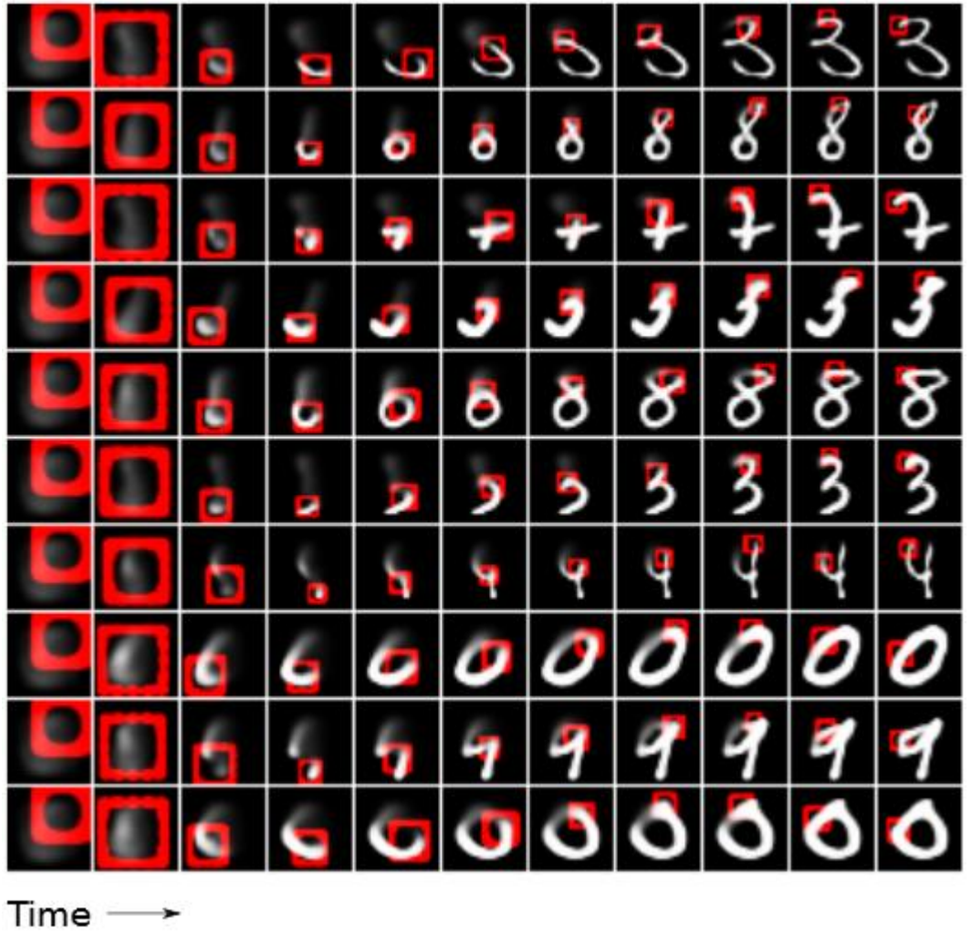
Внимание – сосредотачивается на конкретной части

Итерационность – уточняет рисунок

Отклик – рисует, смотрит на весь рисунок, оценивает

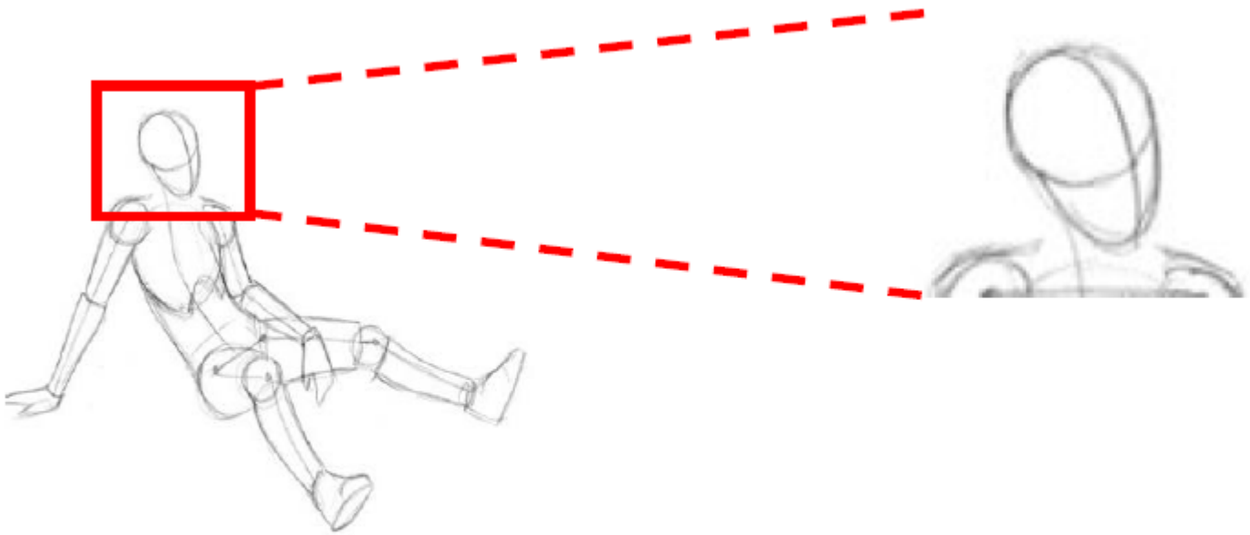
Iterative Attentive Generation

DRAW = Deep Recurrent Attentive Writer



«DRAW: A Recurrent Neural Network For Image Generation»
[Karol Gregor и др. 2015 https://arxiv.org/abs/1502.04623]

Iterative Attentive Generation



Original image
500×500

Glimpse
60×60

много тонкостей…

Итоги – автокодировщик

кодировщик-декодировщик для восстановления входа
понятие «узкое горло»

есть разные виды…

регуляризация в автокодировщиках

- **+ шум (в Denoising Autoencoders)**
- **+ специальное слагаемое к функции ошибки (в Sparse Autoencoders)**
- **+ специальное слагаемое к функции ошибки для стабильности латентного представления (используем Якобиан в Contractive Autoencoders)**

Итог – восстановление плотности

авторегрессионные модели

потоки

авторегрессионные потоки

Отличный курс

CS294-158-SP20 «Deep Unsupervised Learning»

https://sites.google.com/view/berkeley-cs294-158-sp20/home