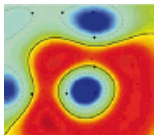
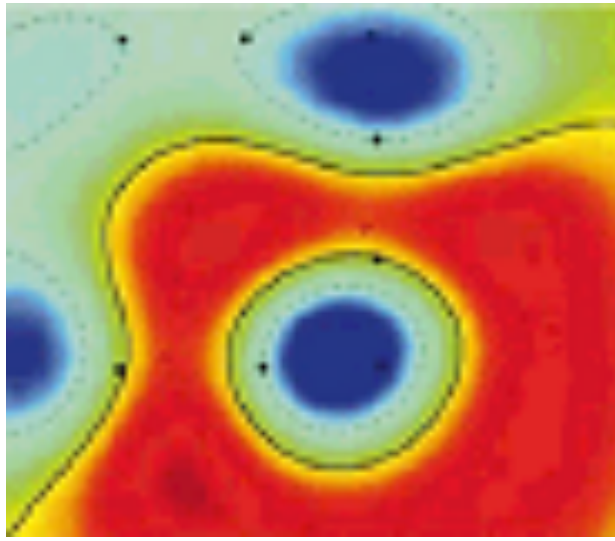


SoSe 2023

Deep Learning 2

Technische Universität Berlin



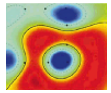
Lecture 3

Generative Models - Part 1

Generative Modelling

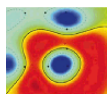
In generative modelling, contrary to discriminative machine learning not a conditional probability density is learned by a joint.
This enables us to:

- Estimate probabilities for a given sample
- Sample new data points from the distribution



Explicit Likelihood Models

- Learning joint probability distributions can be either done via implicit or explicit likelihood models.
- In implicit likelihood models the likelihood of data samples is only used indirectly, while in explicit likelihood models it is modelled directly.
- Autoregressive models, variational autoencoder and normalizing flows are counted as explicit likelihood models and will be explained in this lecture
- Generative Adversarial Networks and diffusion models are implicit likelihood models and will be explained next lecture
- Generally explicit likelihood models tend to generate higher quality data points of more complicated distributions, while implicit likelihood models give direct access to an approximation of the likelihood and often generate more divers examples



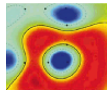
Autoregressive Models

Autoregressive generative models model the probability distribution by making use of the chain rule for probability densities:

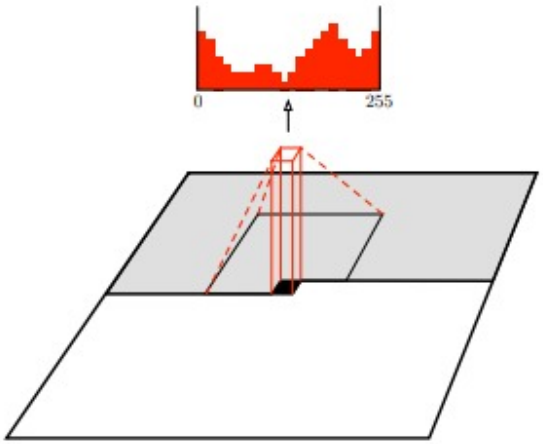
$$P(X_1, X_2, \ldots, X_{n-1}, X_n) = P(X_1) \cdot P(X_2|X_1) \cdot \cdots \cdot P(X_n|X_{n-1}, \ldots, X_2, X_1)$$

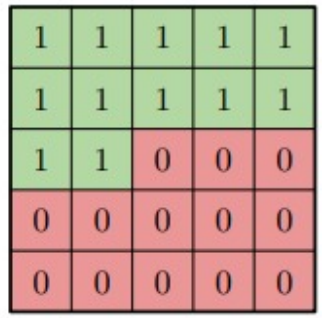
The model is fitted to the data by fitting neural networks to model the conditional densities similar to discriminative machine learning.

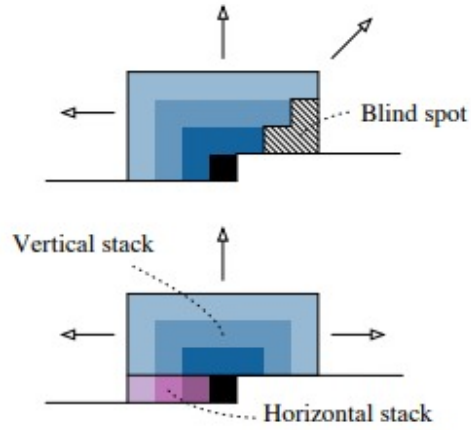
- Very slow both in training and inference
- Good likelihood estimates



PixelCNN (2016) [8]

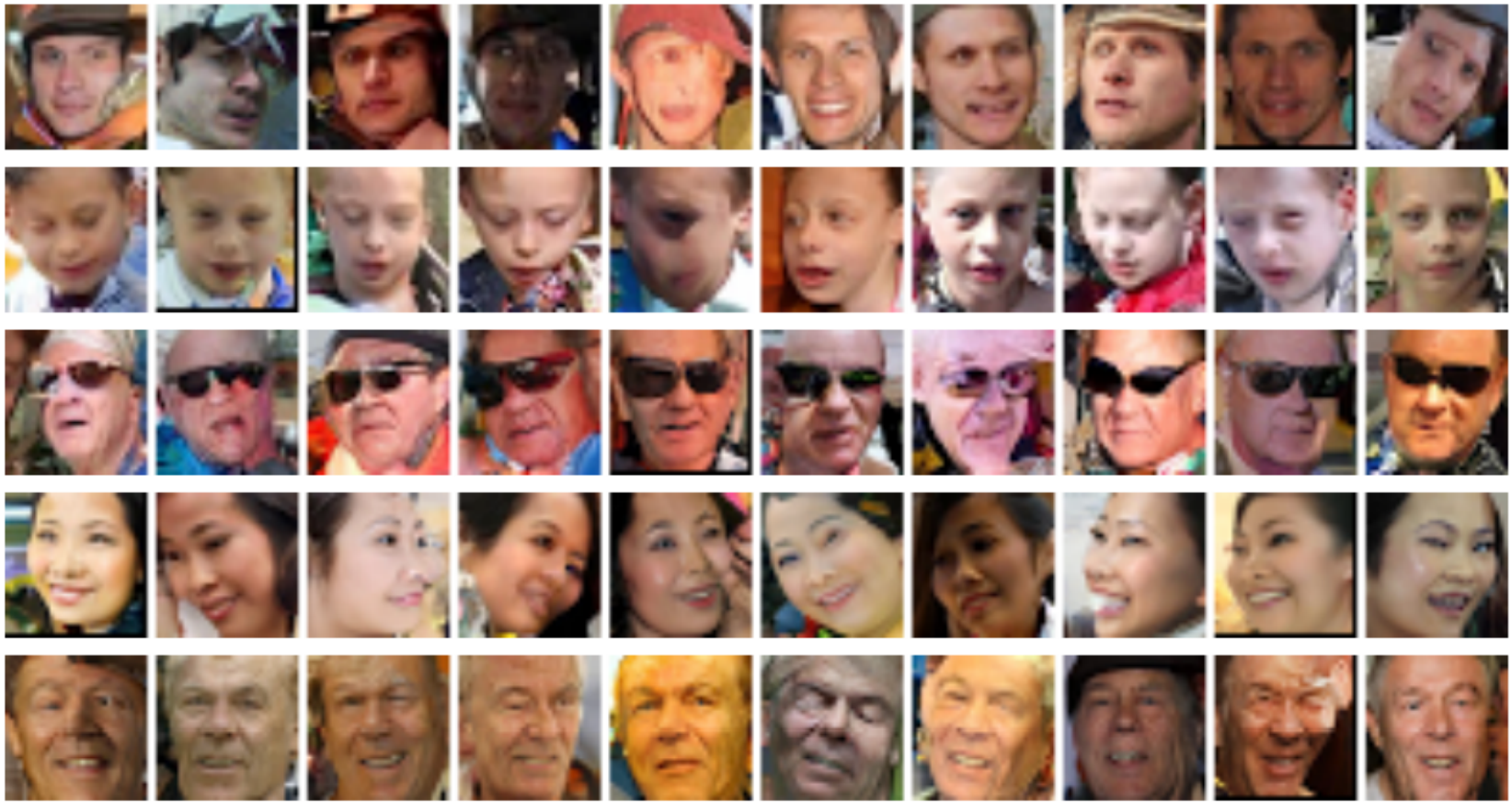




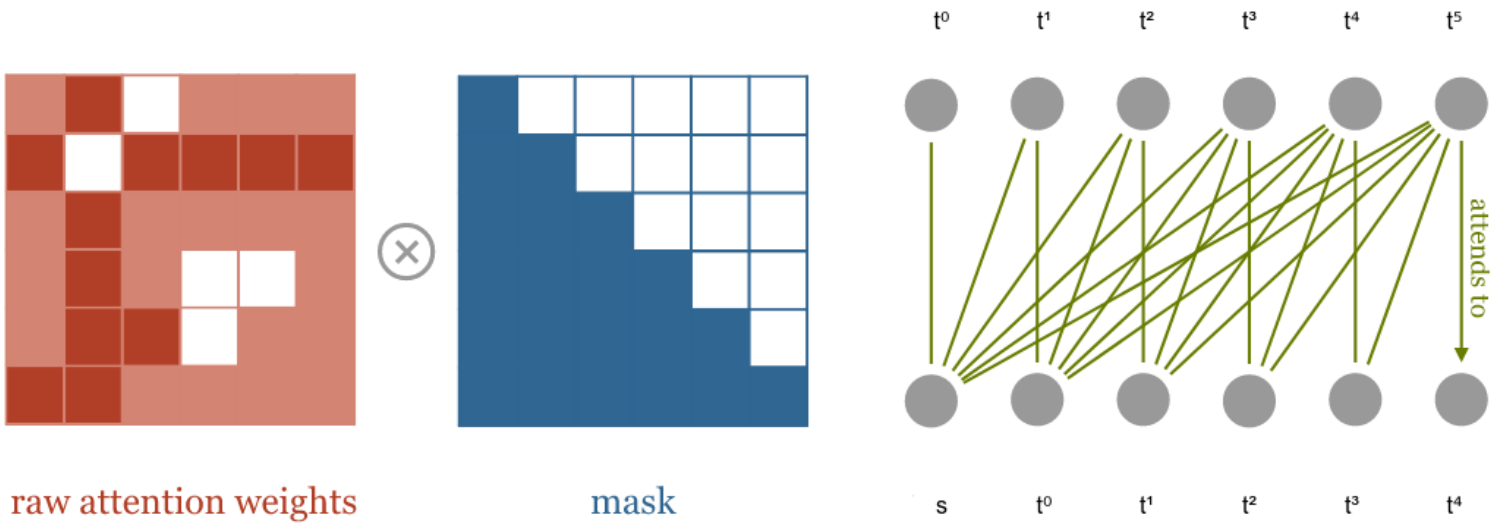


- To generate pixel x_i the model can only condition on the previously generated pixels $x_1, \ldots, x_{i1}$ (Left).
- An example matrix that is used to mask the 5x5 filters to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions (Middle).
- PixelCNNs have a blind spot in the receptive field that can not be used to make predictions (Top Right).
- Two convolutional stacks (blue and purple) allow to capture the whole receptive field (Bottom Right).

Results PixelCNN [8]

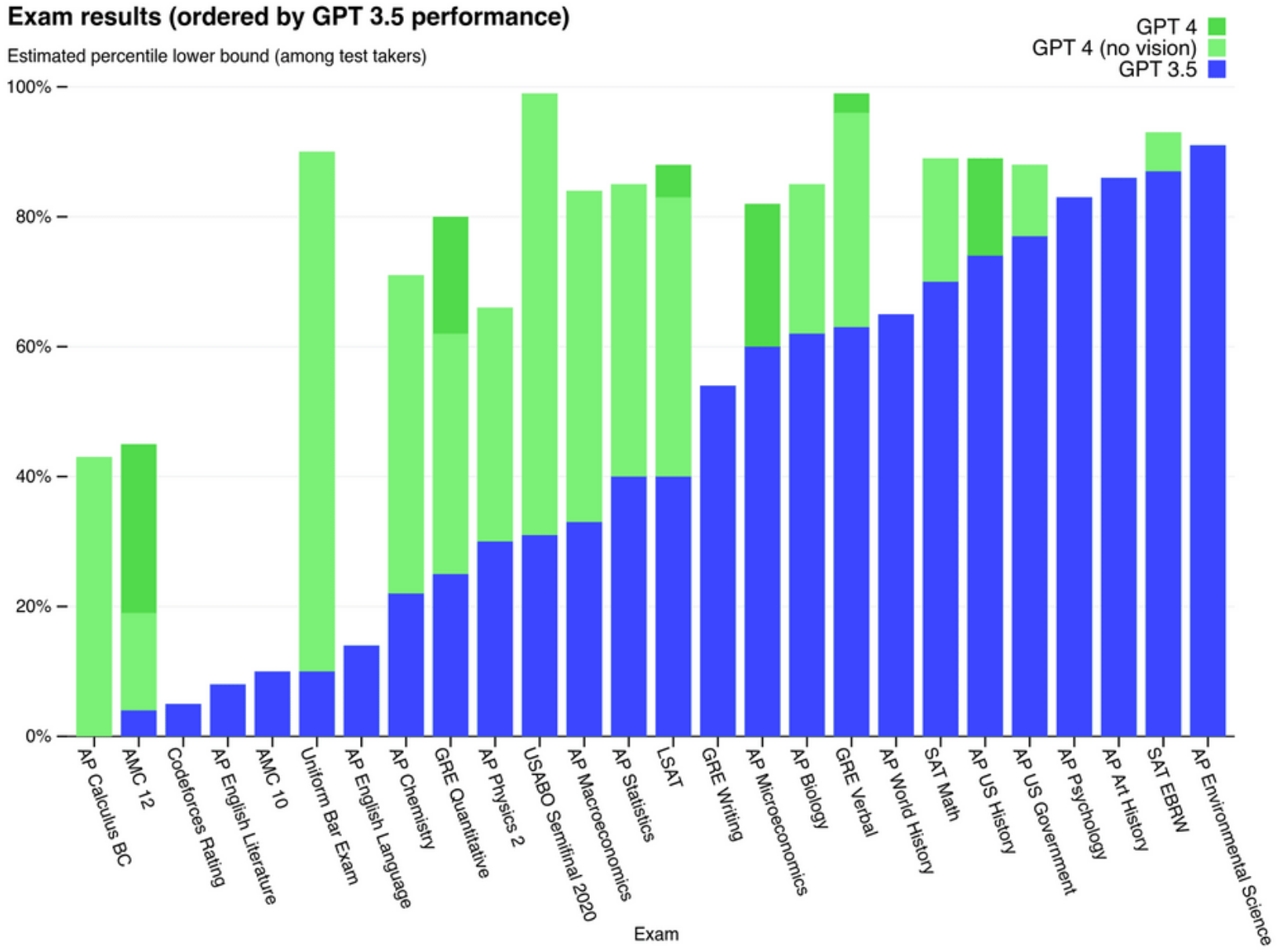


Generative Pre-Trained Transformers GPT 1-4

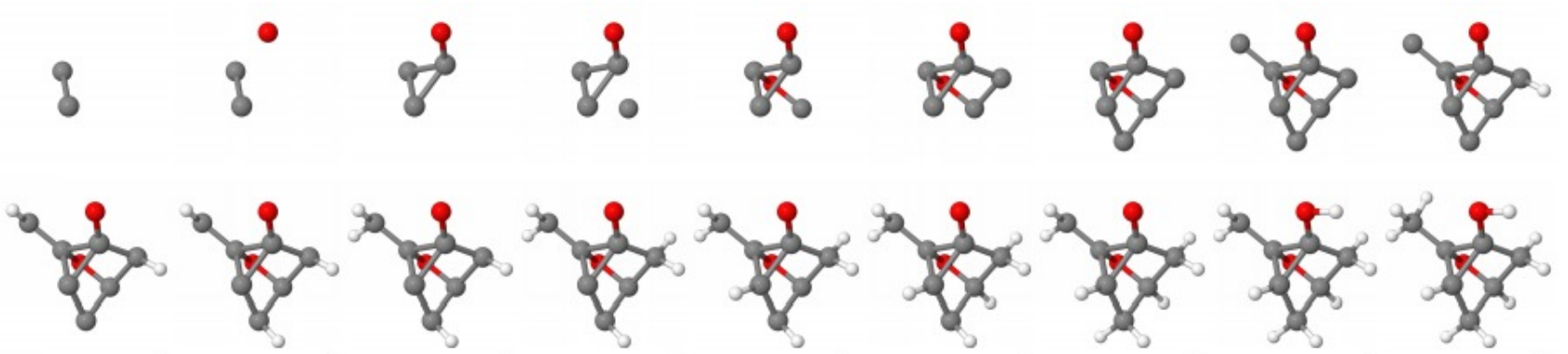


- GPT1-4 use transformer architecture with self-attention
- Self-supervised autoregressive pre-training on large amounts of text data in the internet with pretext task to predict the next token
- Can be trained without rolling out by masking right-to-left only attention mask

Results GPT-4



Application: Molecule Generation [4]



- Generate stable Molecules by attaching one atom at a time.
- Very promising for automated drug discovery and nanotechnology

Bayesian Inference

Another way of generative modelling is Bayesian inference over a latent space Z of the data. Let Z be a latent variable, $P(Z)$ the distribution of it, X is an observation, $P(X)$ the distribution over it and $P(Z|X)$ and $P(X|Z)$ the corresponding conditional distributions. If $P(X|z)$ and $P(z)$ are known, the general problem of Bayesian inference is defined by inferring the probability of a realization z given an observation X. Formally, it is stated as follows:

$$P(z|X) = \frac{P(X|z)P(z)}{P(X)} \tag{1}$$

where P(X) can be expressed as:

$$P(X) = \int_z P(X|z)P(z)dz \tag{2}$$

with the help the law of total probability.

Bayesian Inference

However, even for relatively simple $P(X|z)$, like a multidimensional Gaussian, which is parameterized by a multi-layer Perceptron, this expression becomes intractable. If this is the case, there are two popular approaches to solve the problem.

- One of them, Markov Chain Monte Carlo (MCMC), is to sample from the distributions and approximate the integral.
- The other approach is called Variational Inference (VI). Variational Inference approximates the intractable posterior distribution $P(z|X)$ with a tractable distribution $Q(z|X)$ by minizing the KL divergence between them.

Furthermore, $P(X|z)$ is not known in the common unsupervised learning setting, but there is a trick to solve this problem and VI jointly.

Variational Inference

Minimizing the KL divergence can be rewritten (Exercise!!) to:

$$D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_{z\sim Q(z|X)}\left[\log Q(z|X) - \log P(X|z) - \log P(z)\right] + \log P(X) \tag{3}$$

One still does not know $P(X)$, but it is possible to jointly approximate P(X) and minimize the term $D_{KL}[Q(z|X)||P(z|X)]$ by maximizing the following term, which is also called Evidence Lower Bound (ELBO) and can be expressed as (Exercise!!):

$$\log P(X) - D_{KL}[Q(z|X)\|P(z|X)] = \mathbb{E}_{z\sim Q(z|X)}\left[\log P(X|z)\right] - D_{KL}[Q(z|X)\|P(z)] \tag{4}$$

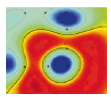
Two things are gained: 1) a tractable approximation $Q(z|X)$ of the posterior distribution $P(z|X)$ and 2) a tractable lower bound approximation of the evidence $P(X)$.

The Original Variational Autoencoder (2013)

Given the findings of the previous section, to actually be able to

- sample from the dataset distribution,
- calculate datapoint posteriors,
- find a latent representation given a dataset.

One needs to define tractable functions $P(X|z)$, $Q(z|X)$ and $P(z)$ which can be optimized by maximizing the ELBO. The prior P(z) can be chosen arbitrarily and thereby allows including prior knowledge about the problem but should be a function from which one can sample easily. The simplest way to model it is with an isotropic gaussian (a multivariate standard normal distribution with a diagonal covariance matrice). From a coding theory perspective, $P(X|z)$ and $Q(z|X)$ can be seen as encoder and decoder.

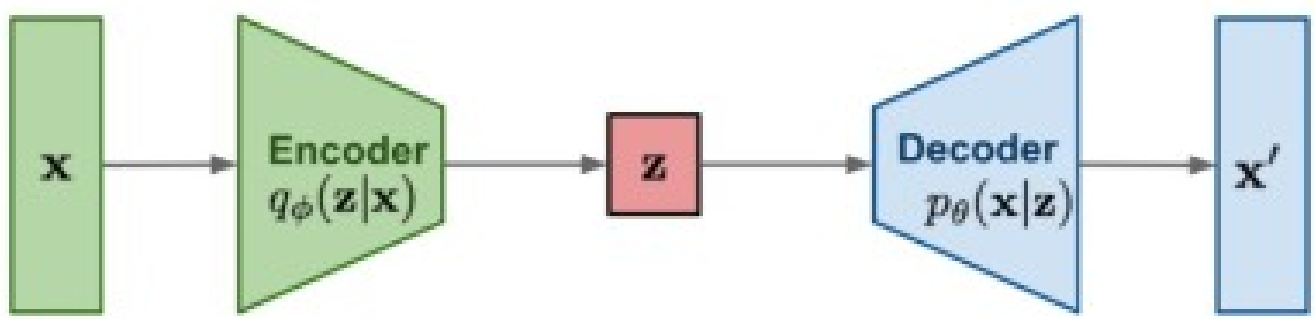


The Original Variational Autoencoder (2013)

By modeling $P(X|z)$ and $Q(z|X)$ as neural networks and reformulating the ELBO to a loss function, the whole architecture can be trained end-to-end. After the training the so developed model, one can

- sample from the dataset distribution by sampling from P(z) and then feed the sampled z through the Q-network,
- calculate datapoint evidence by approximating it with the ELBO
- find a dense latent representation of a data point X by feeding it through the P-network.

Additionally, one can restrict the model only to use very few latent variables and then visualize the resulting manifold.

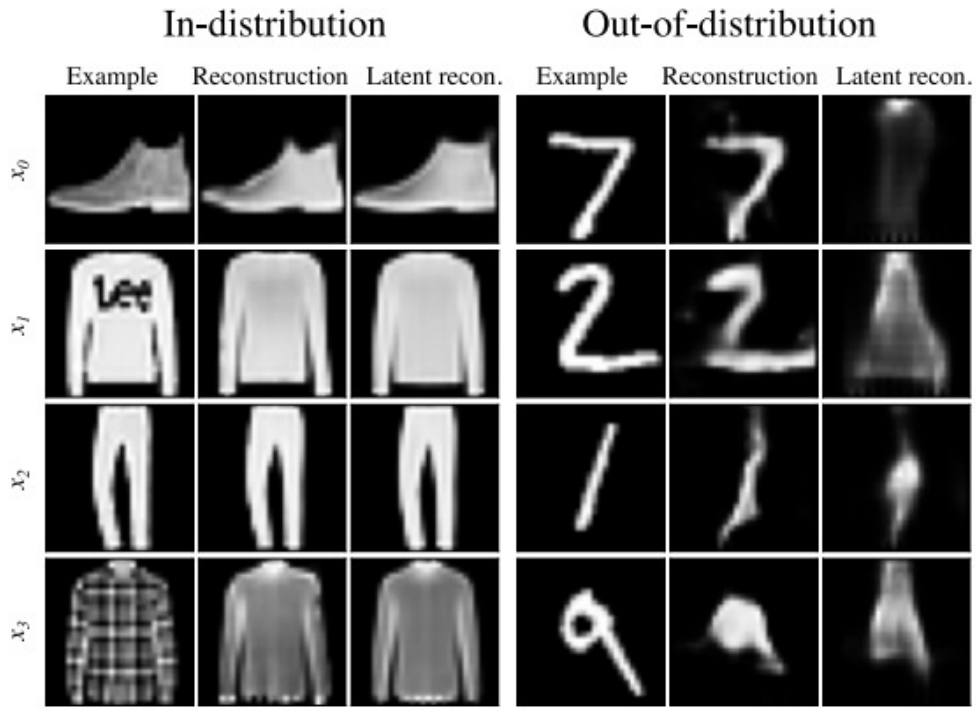


Original Variational Autoencoder (2013) [7]

SOTA: Nouveau Variational Autoencoder (NVAE) (2020) [7]

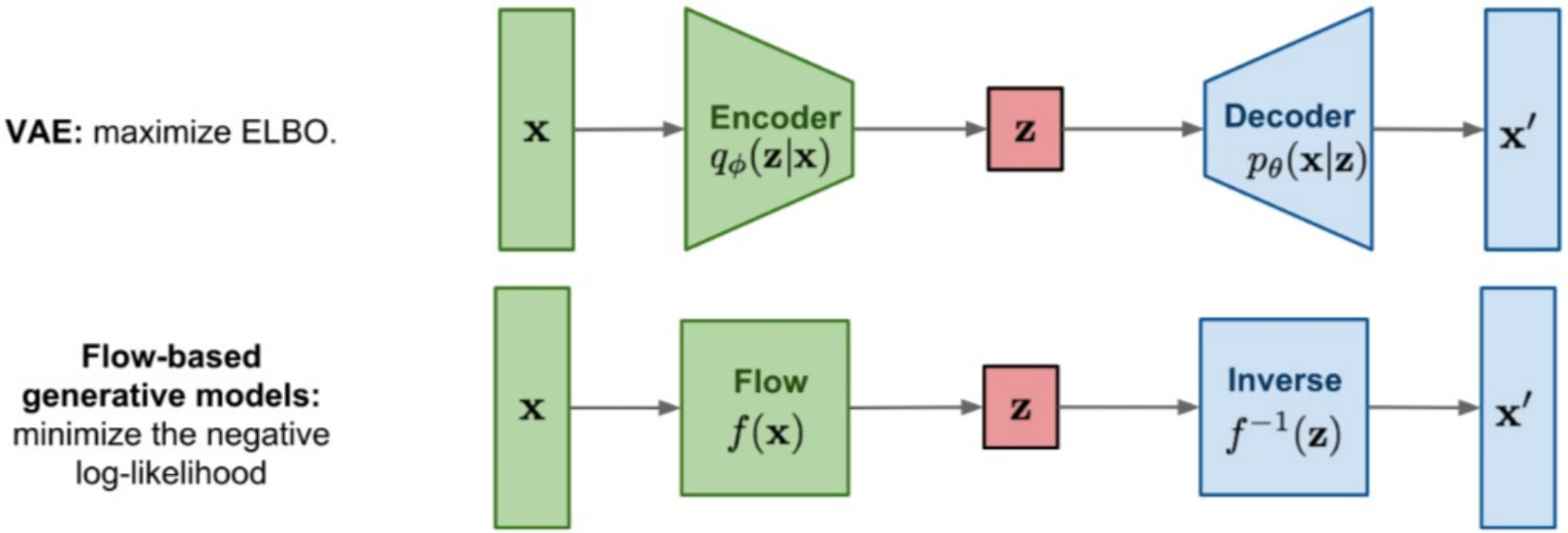
- Uses a hierarchical latent space
- Uses an auto-regressive normalizing flow as prior (it is actually common to mix different approaches for SOTA results)
- Can be effectively used for outlier detection [5] (this is contrary to the goal of generative models often not the case for other models)

Application: Outlier Detection [5]



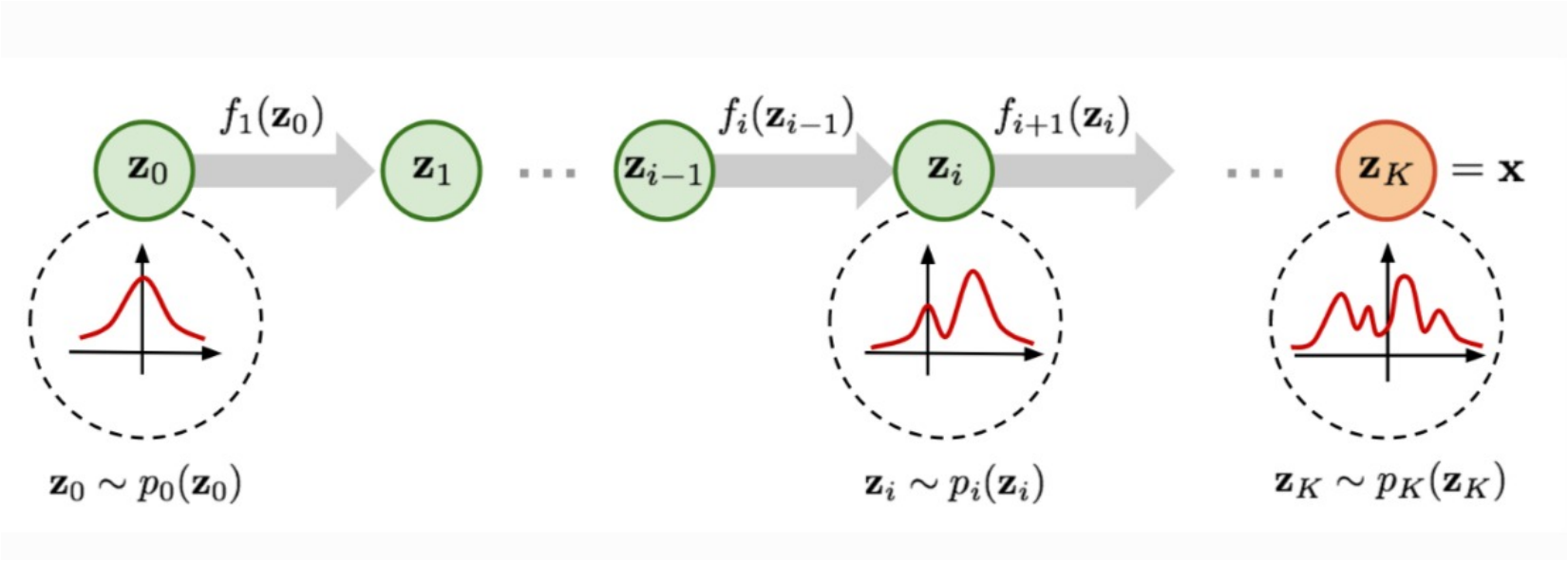
- Reconstruction quality of OOD data is comparable to in-distribution data, resulting in high likelihoods and poor OOD discrimination.
- Samples the k bottom-most latent variables from the conditional prior distribution $p(z >= l|z > l)$ (latent reconstructions) instead of the approximate posterior $q(z > l|z < l)$.
- Thereby, the model reconstructs from the training distribution resulting in lower $p(x|z)$ for OOD data.

Normalizing Flows



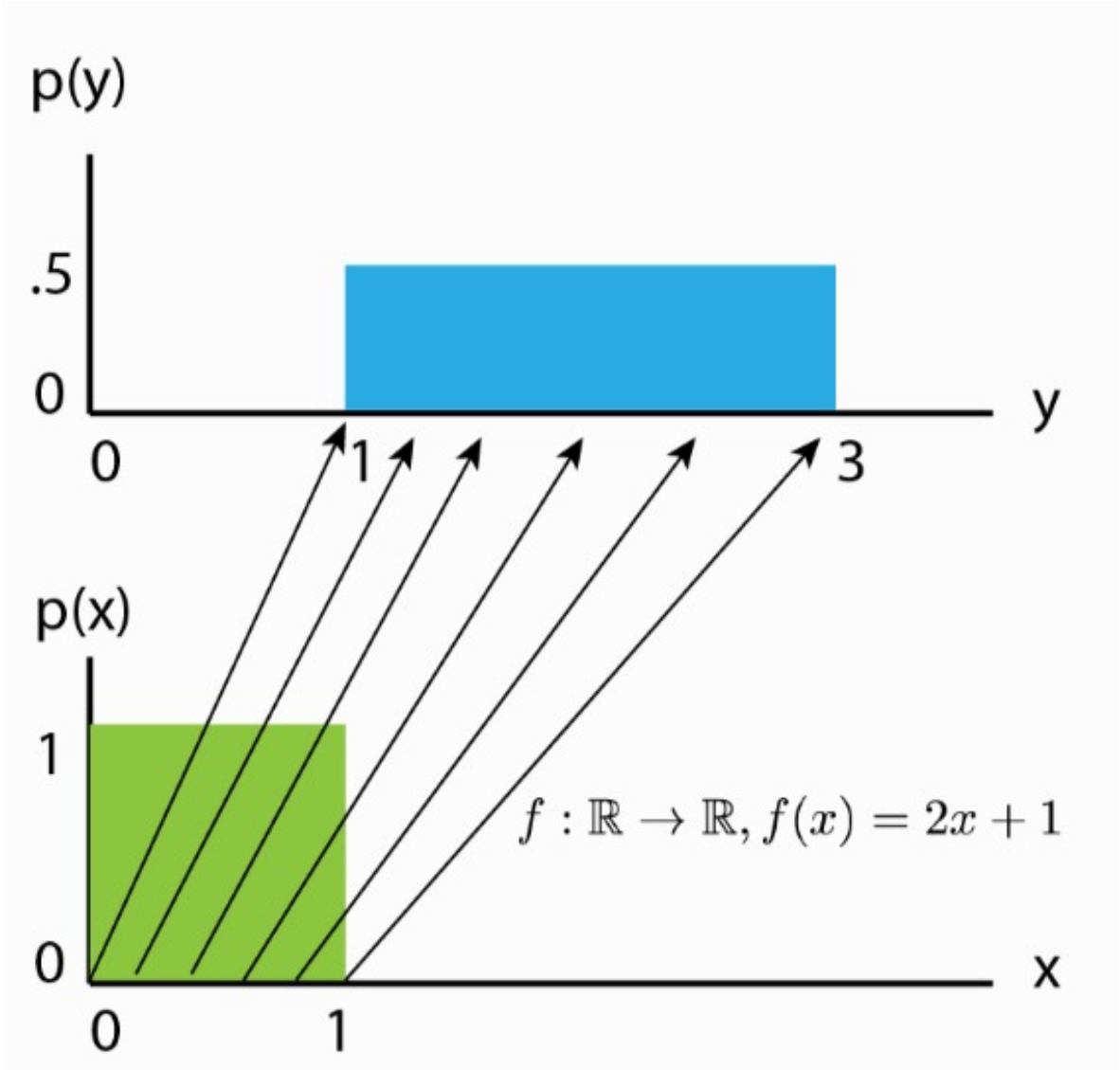
- Very similar to VAE, where encoder and decoder are inverse of each other.
- Can solve similar tasks like likelihood estimation and sampling from distribution.
- Since encoder and decoder are perfectly invertible no reconstruction loss has to be minimized.

The Inverse Model



- Transforms simple tractable probability distribution step by step into more complex distribution.
- Each transformation must be invertible and differentiable, but can be a learned neural network.

Rule for change of Variables



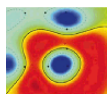
Rule for change of Variables

$$\int p_x(x)dx = \int p_z(z)dz = 1 \qquad \text{(by definition of a probability distribution)}$$

$$\Leftrightarrow p_x(x) = p_z(z)\left|\frac{dz}{dx}\right| = p_z(f(x))\left|\frac{df(x)}{dx}\right|$$

Hence, in order to determine the probability of x, we only need to determine its probability in latent space, and get the derivate of f. Note that this is for a univariate distribution, and f is required to be invertible and smooth. For a multivariate case, the derivative becomes a Jacobian of which we need to take the determinant. As we usually use the log-likelihood as objective, we write the multivariate term with logarithms below:

$$\log p_x(\mathbf{x}) = \log p_z(f(\mathbf{x})) + \log \left|\det \frac{df(\mathbf{x})}{d\mathbf{x}}\right|$$

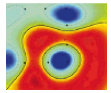


Coupling Layers

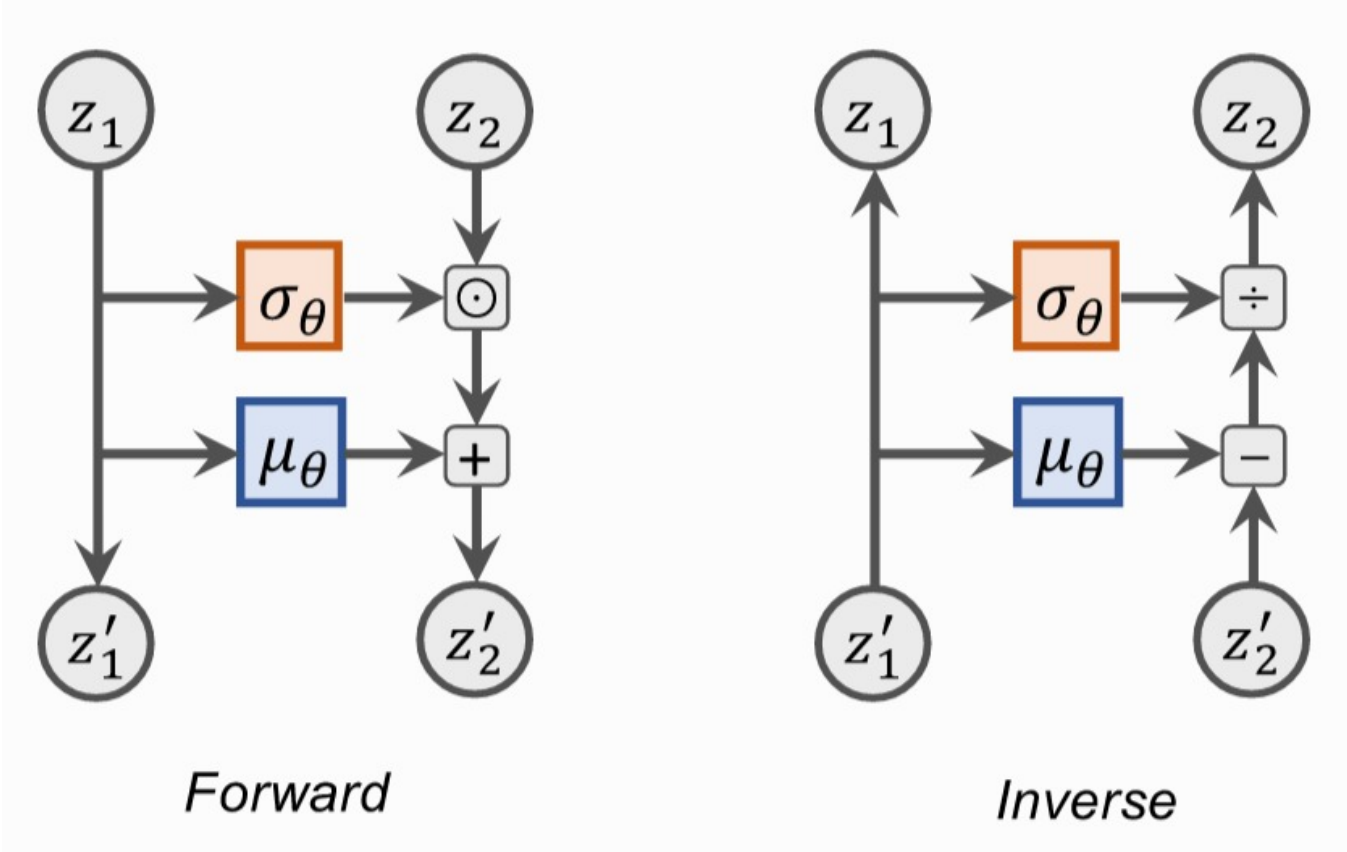
Two design criteria:

- Must be invertible.
- Log-determinant Jacobian must be tractable.

Conventional layers like FCs, CNNs etc. are only invertible in very special cases and the LDJ calculation is usually in $O(n^3)$.

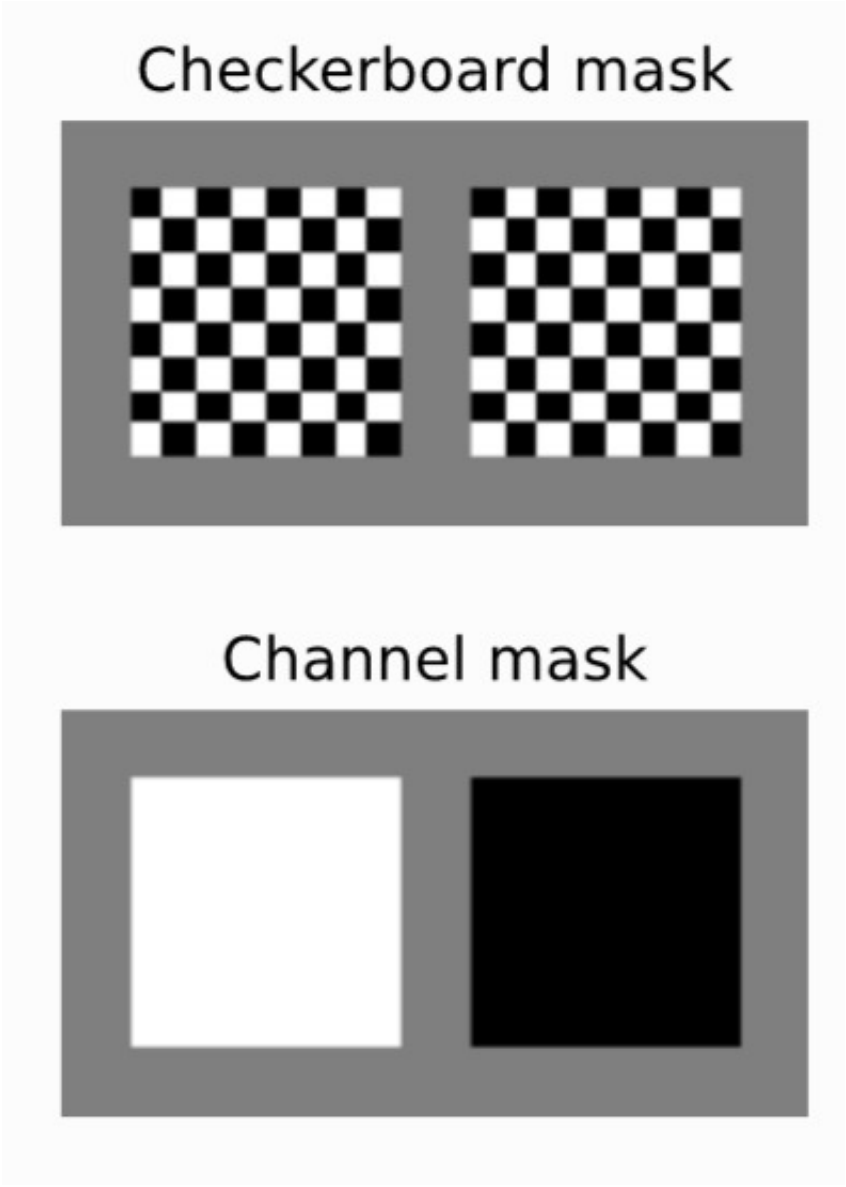


s-t-layers [1]

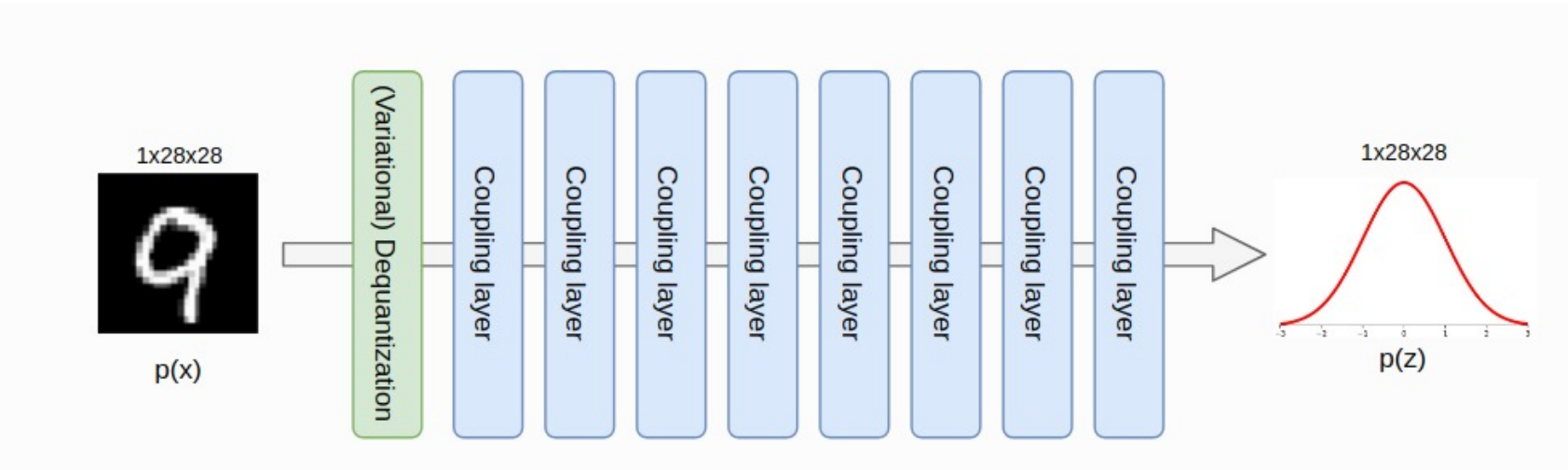


- σ and μ are learned neural networks.
- The LDJ of s-t-layers is the sum of the logs of the scaling factors σ.
- Question: How to split z into z_1 and z_2? Masking!
- Could also be done with autoregressive structure

Masking



Training of Normalizing Flows



Nice (2014) [1]

RealNVP (2017) [2]

Orthogonal 1x1 Convolutions [6]

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:

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

The cost of computing or differentiating $\det(\mathbf{W})$ is $\mathcal{O}(c^3)$, which is often comparable to the cost computing $\texttt{conv2D}(\mathbf{h}; \mathbf{W})$ 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:

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

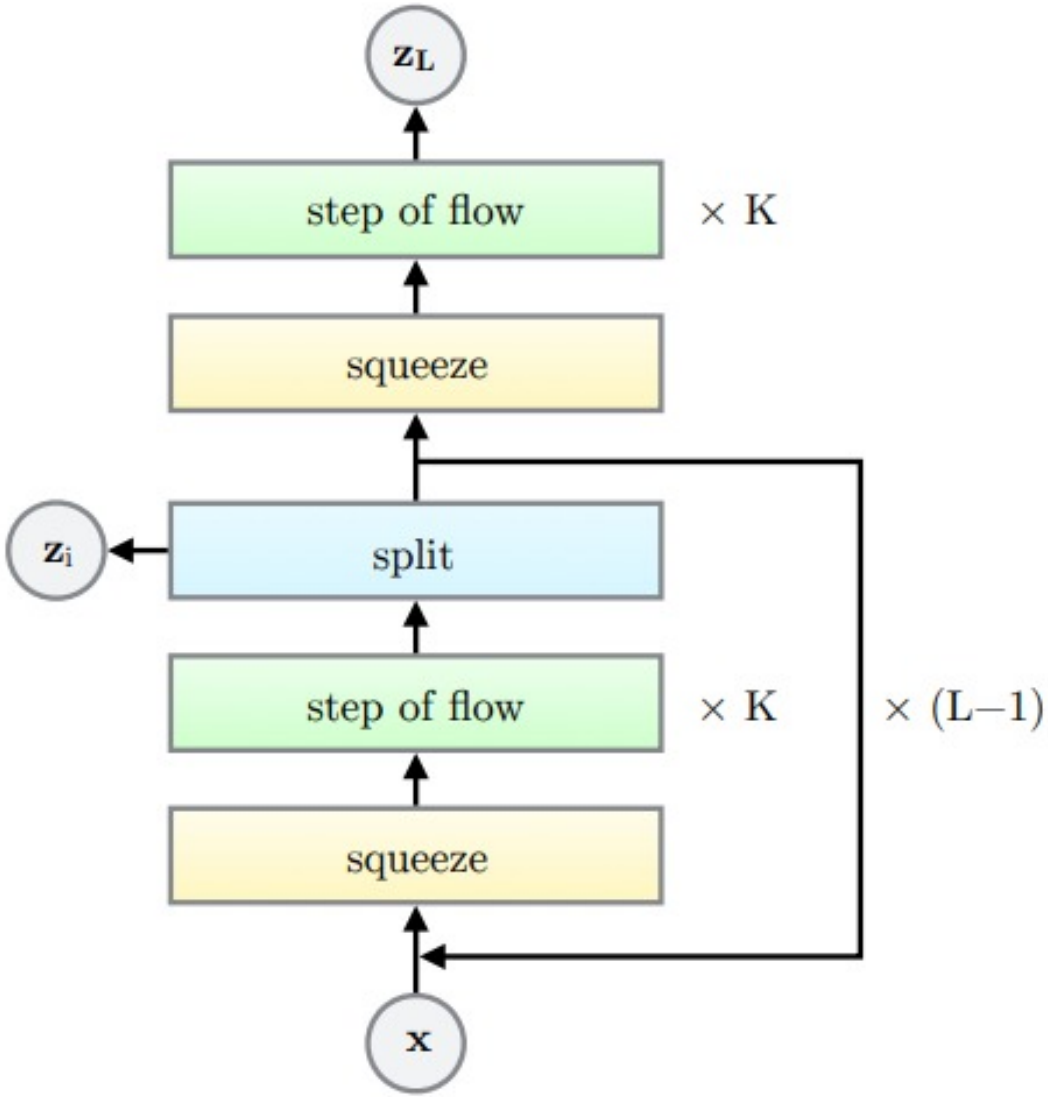
where $\mathbf{P}$ is a permutation matrix, $\mathbf{L}$ is a lower triangular matrix with ones on the diagonal, $\mathbf{U}$ is an upper triangular matrix with zeros on the diagonal, and $\mathbf{s}$ is a vector. The log-determinant is then simply:

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

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).

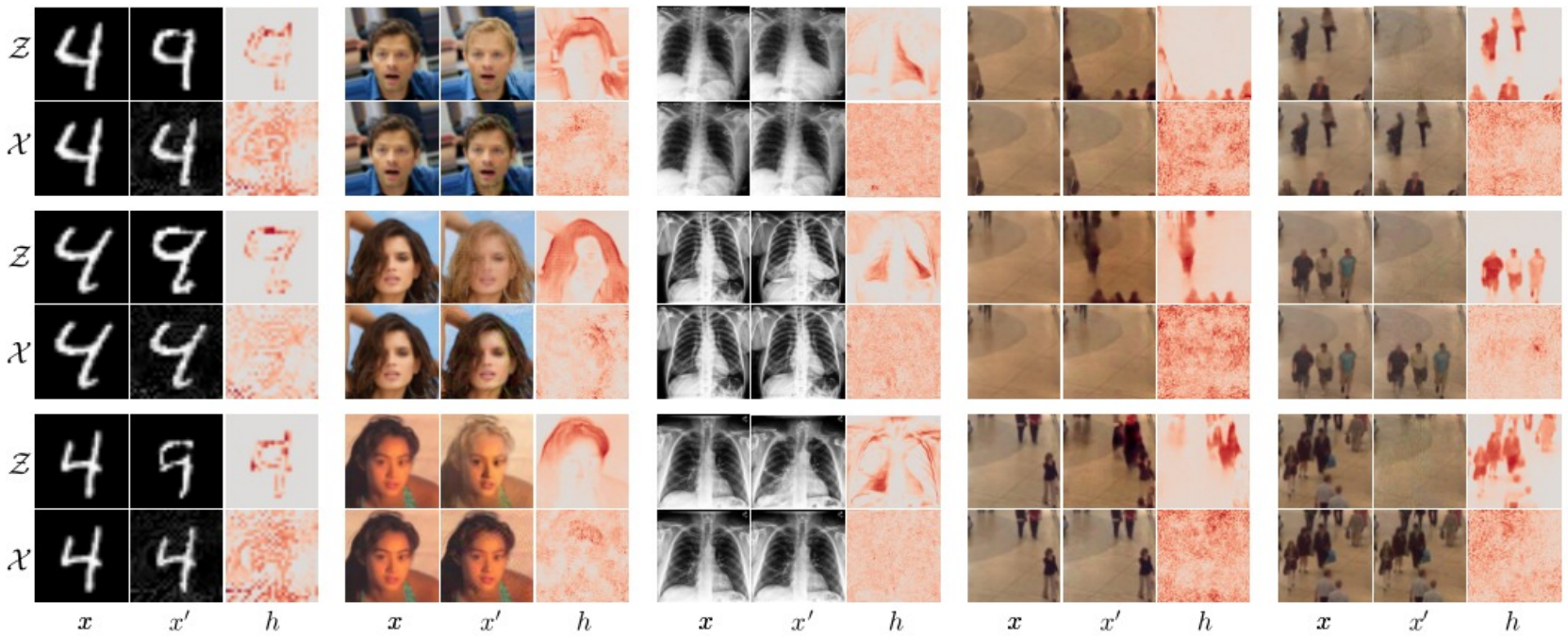
Hierarchical Normalizing Flows [6]



Sota: Glow (2018) [6]

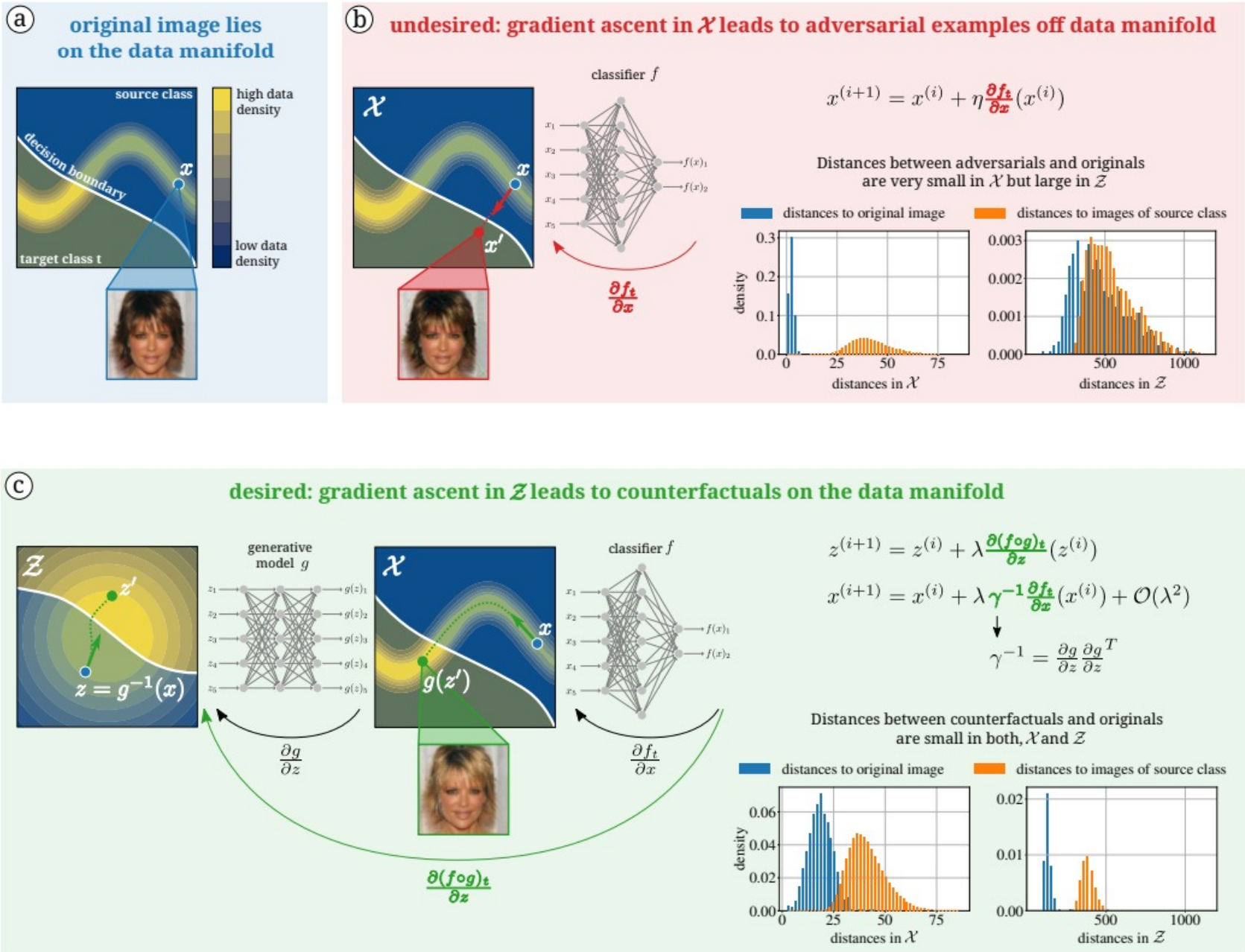
- Very big models necessary for high resolution.
- Training can be unstable especially for very big models.
- Very nice latent space
- Fast high quality image generation while inference

Application: Diffeomorphic Counterfactuals [3]



- Generative Models can be used to generate counterfactual explanations
- One can see, that using a generative model and performing counterfactual search in Z-space is necessary in order to not suffer from a adversarial attack as in the X-space.

Application: Diffeomorphic Counterfactuals [3]



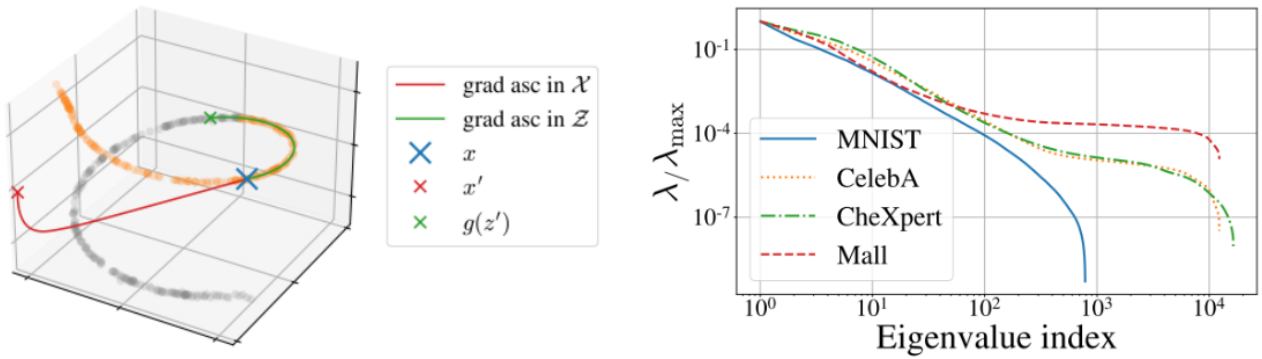
Application: Diffeomorphic Counterfactuals [3]

Algorithm 1 Generating counterfactuals

Require: $x, f, g, g^{-1}, t, \Lambda, \lambda, N$

$z \leftarrow g^{-1}(x)$
for i in range(N) **do**
    $\nabla_z \leftarrow \frac{\partial (f \circ g)_t}{\partial z}$
    $z \leftarrow$ optimizer.step(λ, ∇_z)
    if $f(g(z))_t > \Lambda$ **then**
        return $g(z)$
    end if
end for
return None

Application: Diffeomorphic Counterfactuals [3]



- The update step can be reformulated to $g\left(z^{(i+1)}\right) = g\left(z^{(i)}\right) + \lambda U \Sigma^2 U^T \frac{\partial f_t}{\partial x} + \mathcal{O}\left(\lambda^2\right)$ with Σ^2 being the Eigenvalues of the Jacobians of g shown in the figure (Exercise!!)
- Directions that point out of the manifold have very low eigenvalues, only the few directions along the manifold have high eigenvalues
- Consequently, the search stays on the manifold, since the velocity out of it is very low

References

L. Dinh, D. Krueger, and Y. Bengio.
Nice: Non-linear independent components estimation.
arXiv preprint arXiv:1410.8516, 2014.

L. Dinh, J. Sohl-Dickstein, and S. Bengio.
Density estimation using real nvp.
arXiv preprint arXiv:1605.08803, 2016.

A.-K. Dombrowski, J. E. Gerken, K.-R. Müller, and P. Kessel.
Diffeomorphic counterfactuals with generative models.
arXiv preprint arXiv:2206.05075, 2022.

N. W. Gebauer, M. Gastegger, and K. T. Schütt.
Generating equilibrium molecules with deep neural networks.
arXiv preprint arXiv:1810.11347, 2018.

J. D. Havtorn, J. Frellsen, S. Hauberg, and L. Maaløe.
Hierarchical vaes know what they don't know.
In *International Conference on Machine Learning*, pages 4117–4128. PMLR, 2021.

D. P. Kingma and P. Dhariwal.
Glow: Generative flow with invertible 1x1 convolutions.
Advances in neural information processing systems, 31, 2018.

D. P. Kingma and M. Welling.
Auto-encoding variational bayes.
arXiv preprint arXiv:1312.6114, 2013.

A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al.
Conditional image generation with pixelcnn decoders.
Advances in neural information processing systems, 29, 2016.