Deep Generative Models

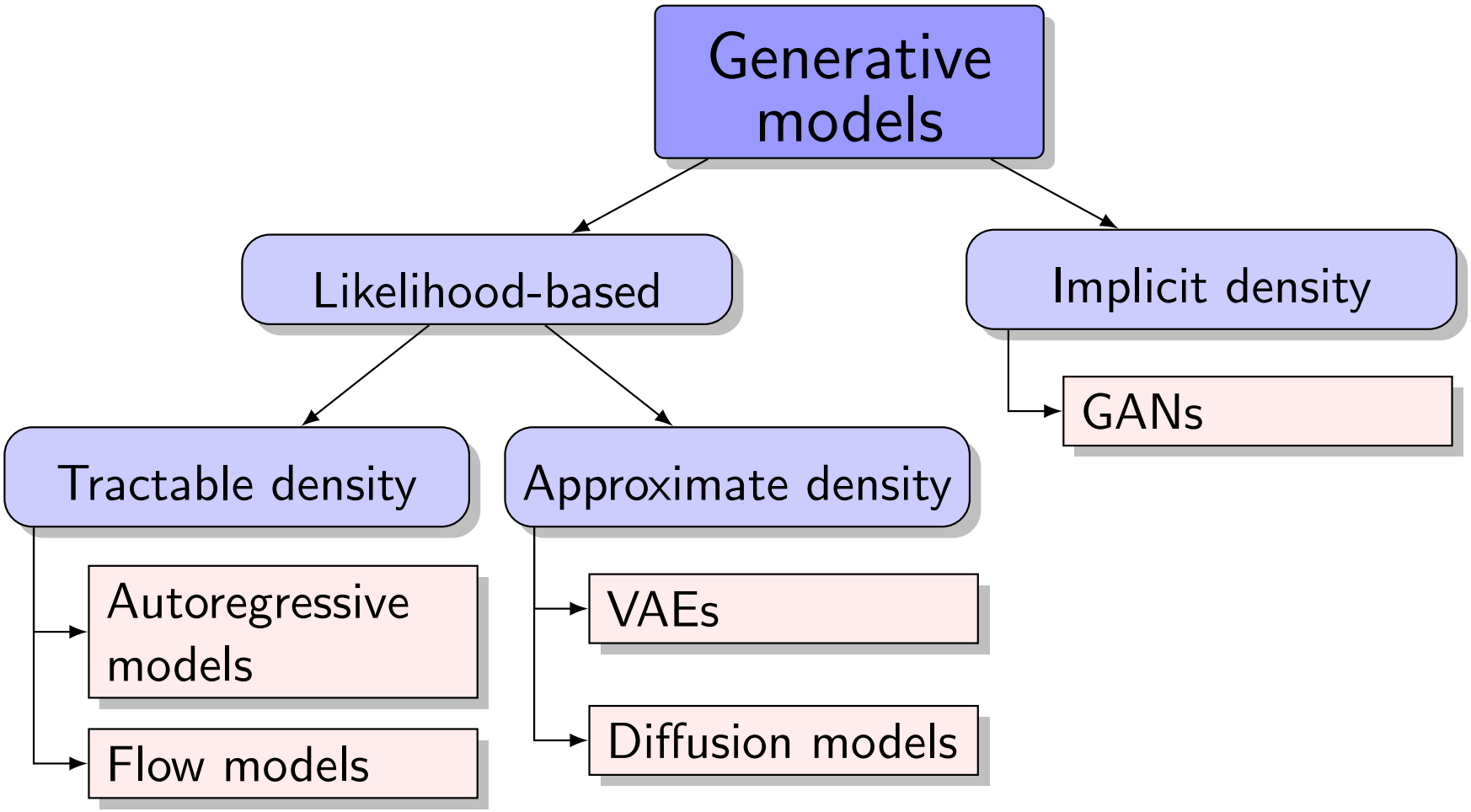
Lecture 1

Roman Isachenko

Moscow Institute of Physics and Technology

Autumn, 2021

Generative models zoo



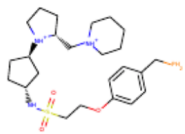
Applications

“ i want to talk to you . ”
“i want to be with you . ”
“i do n’t want to be with you . ”
i do n’t want to be with you .
she did n’t want to be with him .

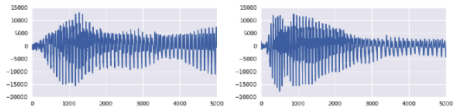
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Text analysis

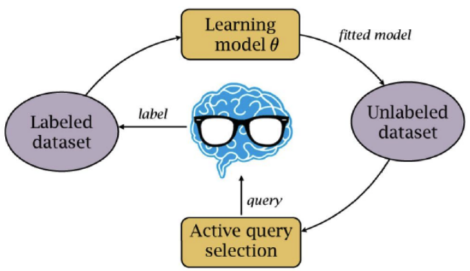
Image analysis



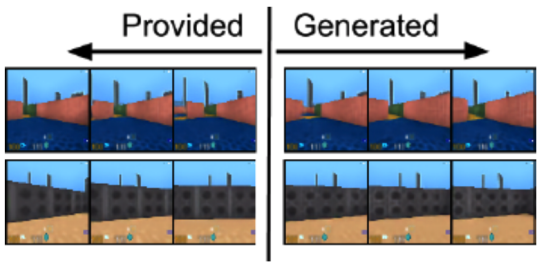
Graph analysis



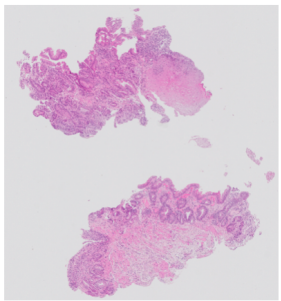
Audio analysis



Active Learning



Reinforcement Learning



Medical data

and more...

image credit: https://jmtomczak.github.io/blog/1/1_introduction.html

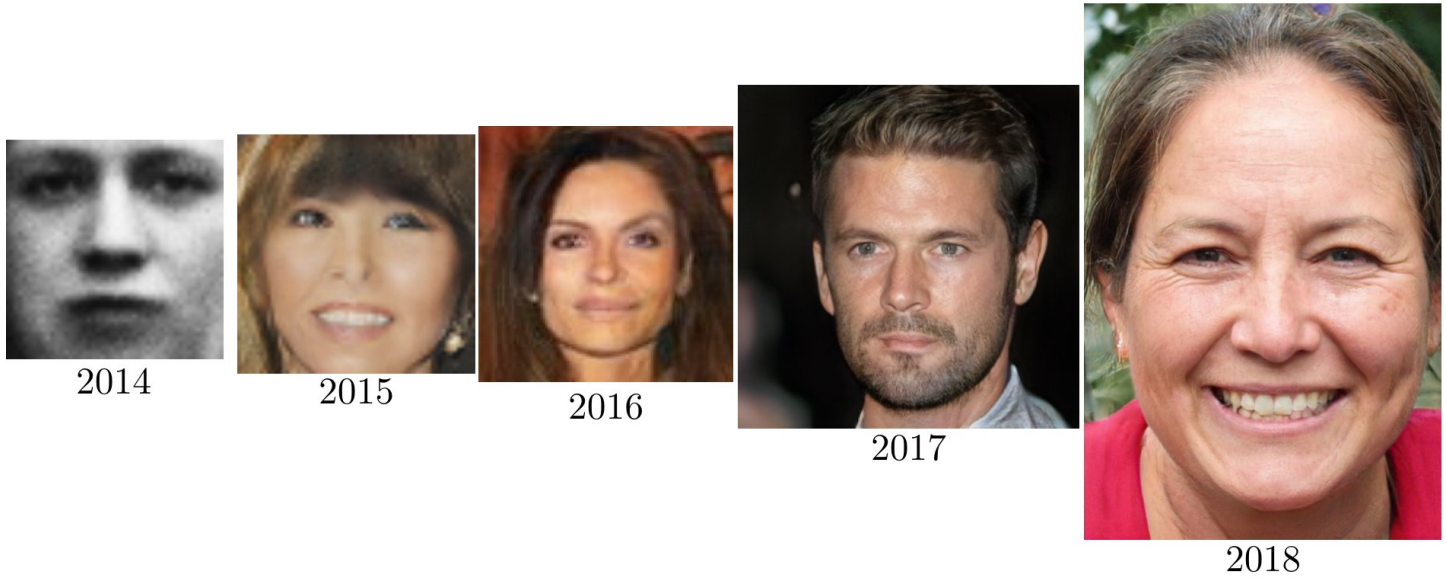
Applications: Image generation (VAE)

Kingma D. P., Welling M. Auto-encoding variational bayes, 2013

Applications: Image generation (DCGAN)

Radford A., Metz L., Chintala S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015

Applications: Face generation (StyleGAN)

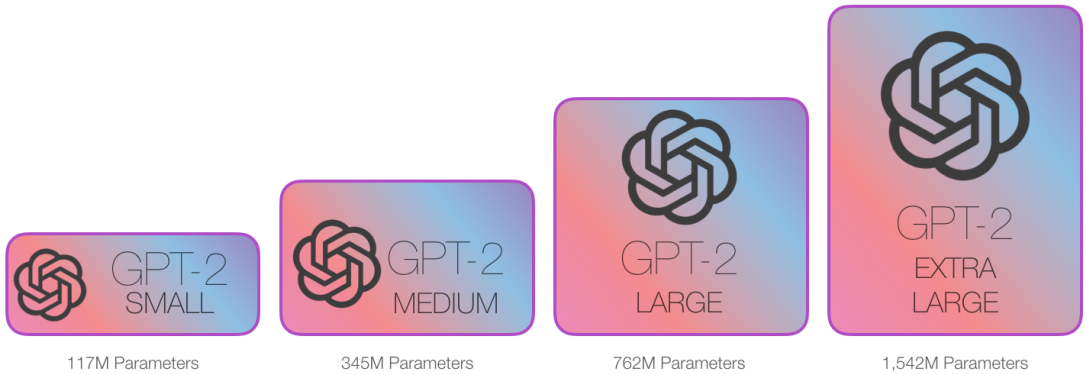


Karras T., Laine S., Aila T. A style-based generator architecture for generative adversarial networks, 2018

Applications: Face generation (VQ-VAE-2)

Razavi A., Oord A., Vinyals O. Generating Diverse High-Fidelity Images with VQ-VAE-2, 2019

Applications: Language modelling



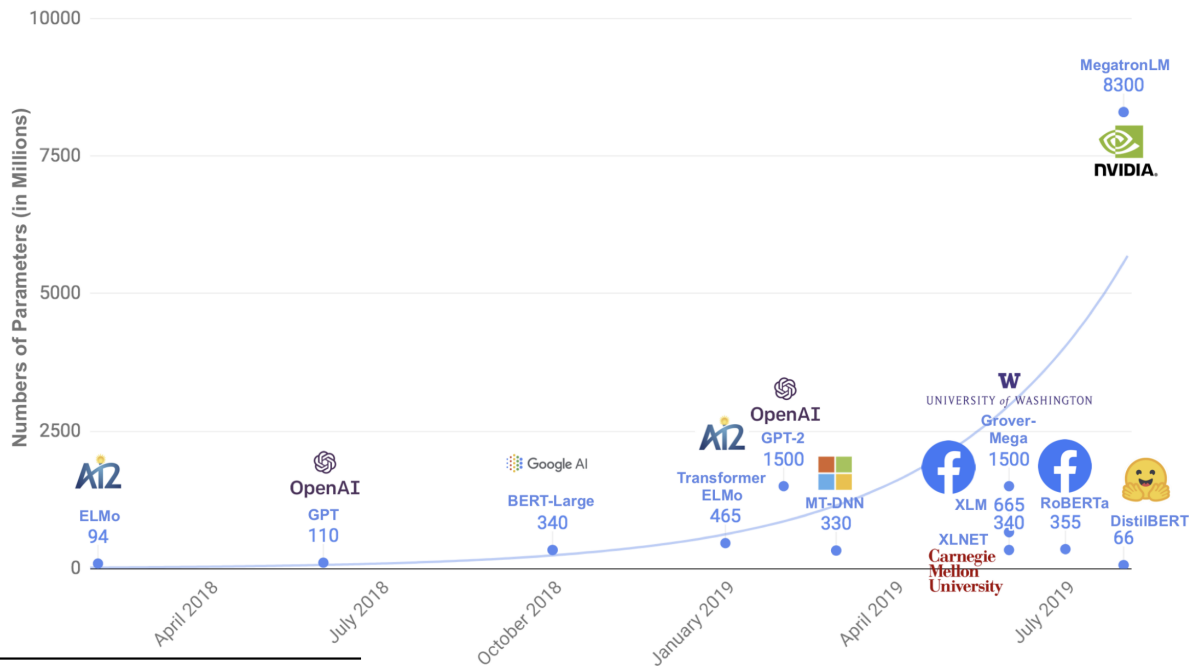
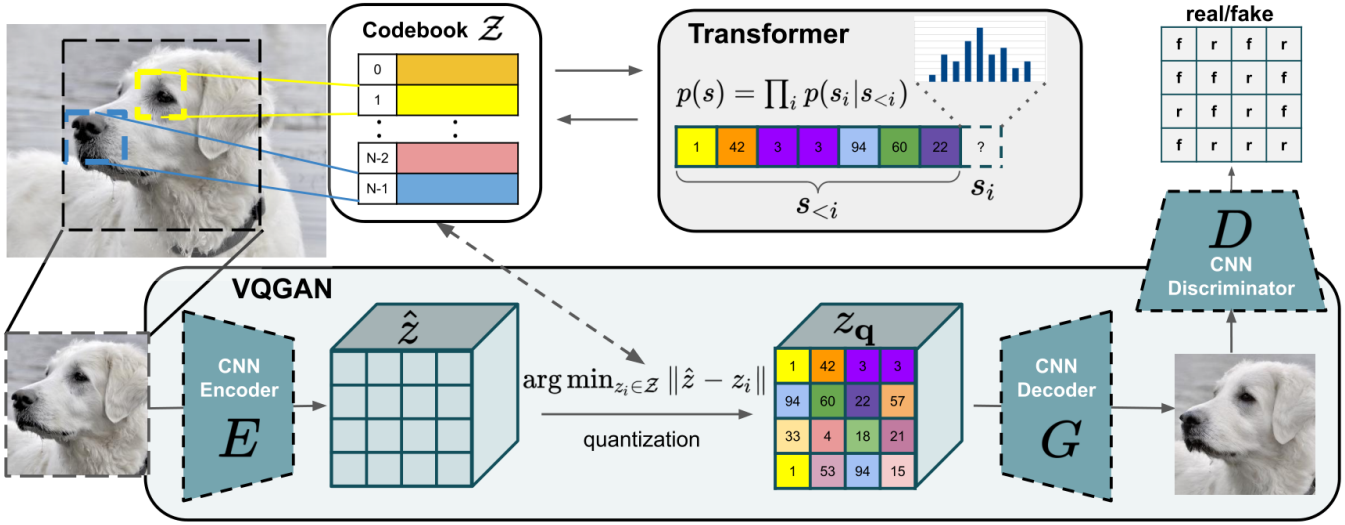


image credit: http://jalammar.github.io/illustrated-gpt2

Sanh V. et al. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 2019.

Applications: Image generation, new era



Esser P., Rombach R., Ommer B. Taming Transformers for High-Resolution Image Synthesis, 2020

Applications: Cross-modal image-text models

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

Edit prompt or view more images↓

TEXT PROMPT an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

Edit prompt or view more images↓

image credit: https://openai.com/blog/dall-e/
Ramesh A. et al. Zero-shot text-to-image generation, 2021

Applications: Image generation

Dhariwal P., Nichol A. Diffusion Models Beat GANs on Image Synthesis, 2021

Problem Statement

We are given i.i.d. samples $\{\mathbf{x}_i\}_{i=1}^n \in X$ (e.g. $X = \mathbb{R}^m$) from unknown distribution $\pi(\mathbf{x})$.

Goal

We would like to learn a distribution $\pi(\mathbf{x})$ for

- evaluating $\pi(\mathbf{x})$ for new samples (how likely to get object $\mathbf{x}$?);
- sampling from $\pi(\mathbf{x})$ (to get new objects $\mathbf{x} \sim \pi(\mathbf{x})$).

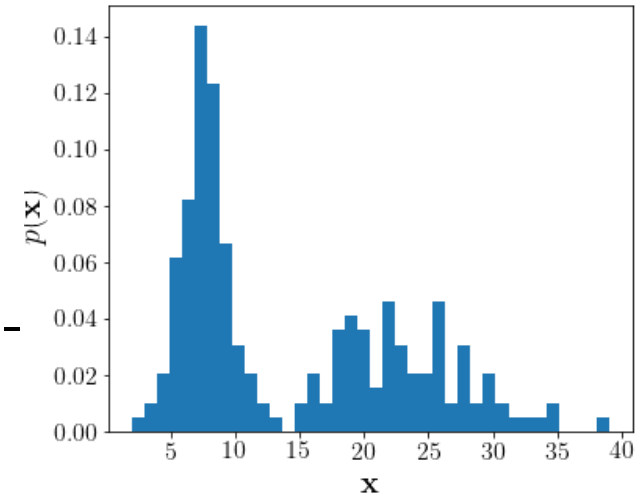
Challenge

Data is complex and high-dimensional. E.g. the dataset of images lies in the space $X \subset \mathbb{R}^{\text{width} \times \text{height} \times \text{channels}}$.

Histogram as a generative model

Let $x \sim \text{Categorical}(\boldsymbol{\pi})$. The histogram is totally defined by

$$\pi_k = \pi(x = k) = \frac{\sum_{i=1}^{k} [x_i = k]}{n}.$$



Problem: curse of dimensionality (number of bins grows exponentially).

MNIST example: 28x28 gray-scaled images, each image is $\mathbf{x} = (x_1, \ldots, x_{784})$, where $x_i = \{0, 1\}$.

$$\pi(\mathbf{x}) = \pi(x_1) \cdot \pi(x_2|x_1) \cdot \cdots \cdot \pi(x_m|x_{m-1}, \ldots, x_1).$$

Hence, the histogram will have $2^{28 \times 28} - 1$ parameters to specify $\pi(\mathbf{x})$.

Question: How many parameters do we need in these cases?

$$\pi(\mathbf{x}) = \pi(x_1) \cdot \pi(x_2) \cdot \cdots \cdot \pi(x_m);$$
$$\pi(\mathbf{x}) = \pi(x_1) \cdot \pi(x_2|x_1) \cdot \cdots \cdot \pi(x_m|x_{m-1}).$$

Tricks

Monte-Carlo estimation

$$\mathbb{E}_{p(\mathbf{x})} f(\mathbf{x}) = \int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \approx \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_i), \quad \text{where } \mathbf{x}_i \sim p(\mathbf{x}).$$

Integral of expected value could be estimated using only samples from the distribution.

Law of the unconscious statistician (LOTUS)

Let X be a random variable and let $Y = g(X)$. Then

$$\mathbb{E}_{p_Y} Y = \mathbb{E}_{p_X} g(X) = \int g(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}.$$

The name LOTUS came from a purported tendency to use the identity without realizing that it must be treated as the result of a rigorously proved theorem, not merely a definition.

Divergences

Fix probabilistic model $p(\mathbf{x}|\boldsymbol{\theta})$ – the set of parameterized distributions.
Instead of searching true $\pi(\mathbf{x})$ over all probability distributions, learn function approximation $p(\mathbf{x}|\boldsymbol{\theta}) \approx \pi(\mathbf{x})$.

What is a divergence?

Let $\mathcal{S}$ be the set of all possible probability distributions. Then $D : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ is a divergence if

- $D(\pi||p) \geq 0$ for all $\pi, p \in \mathcal{S}$;
- $D(\pi||p) = 0$ if and only if $\pi \equiv p$.

General divergence minimization task

$$\min_{\boldsymbol{\theta}} D(\pi||p),$$

where $\pi(\mathbf{x})$ is a true data distribution, $p(\mathbf{x}|\boldsymbol{\theta})$ is a model distribution.

f-divergence family

f-divergence

$$D_f(\pi||p) = \mathbb{E}_{p(\mathbf{x})} f\left(\frac{\pi(\mathbf{x})}{p(\mathbf{x})}\right) = \int p(\mathbf{x}) f\left(\frac{\pi(\mathbf{x})}{p(\mathbf{x})}\right) d\mathbf{x}.$$

Here $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex, lower semicontinuous function satisfying $f(1) = 0$.

Name	$D_f(P\|Q)$	Generator $f(u)$
Kullback-Leibler	$\int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x$	$u \log u$
Reverse KL	$\int q(x) \log \frac{q(x)}{p(x)} \, \mathrm{d}x$	$-\log u$
Pearson χ^2	$\int \frac{(q(x)-p(x))^2}{p(x)} \, \mathrm{d}x$	$(u-1)^2$
Squared Hellinger	$\int \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2 \, \mathrm{d}x$	$(\sqrt{u}-1)^2$
Jensen-Shannon	$\frac{1}{2} \int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \, \mathrm{d}x$	$-(u+1) \log \frac{1+u}{2} + u \log u$
GAN	$\int p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \, \mathrm{d}x - \log(4)$	$u \log u - (u+1) \log(u+1)$

Nowozin S., Cseke B., Tomioka R. f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization, 2016

Forward KL vs Reverse KL

Forward KL

$$KL(\pi||p) = \int \pi(\mathbf{x}) \log \frac{\pi(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})} d\mathbf{x} \to \min_{\boldsymbol{\theta}}$$

Reverse KL

$$KL(p||\pi) = \int p(\mathbf{x}|\boldsymbol{\theta}) \log \frac{p(\mathbf{x}|\boldsymbol{\theta})}{\pi(\mathbf{x})} d\mathbf{x} \to \min_{\boldsymbol{\theta}}$$

What is the difference between these two formulations?

Maximum likelihood estimation (MLE)

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(\mathbf{x}_i|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(\mathbf{x}_i|\boldsymbol{\theta}).$$

Forward KL vs Reverse KL

Forward KL

$$
\begin{aligned}
KL(\pi||p) &= \int \pi(\mathbf{x}) \log \frac{\pi(\mathbf{x})}{p(\mathbf{x}|\boldsymbol{\theta})} d\mathbf{x} \\
&= \int \pi(\mathbf{x}) \log \pi(\mathbf{x}) d\mathbf{x} - \int \pi(\mathbf{x}) \log p(\mathbf{x}|\boldsymbol{\theta}) d\mathbf{x} \\
&= -\mathbb{E}_{\pi(\mathbf{x})} \log p(\mathbf{x}|\boldsymbol{\theta}) + \text{const} \\
&\approx -\frac{1}{n} \sum_{i=1}^{n} \log p(\mathbf{x}_i|\boldsymbol{\theta}) + \text{const} \rightarrow \min_{\boldsymbol{\theta}} .
\end{aligned}
$$

Maximum likelihood estimation is equivalent to minimization of the Monte-Carlo estimate of forward KL.

Reverse KL

$$
\begin{aligned}
KL(p||\pi) &= \int p(\mathbf{x}|\boldsymbol{\theta}) \log \frac{p(\mathbf{x}|\boldsymbol{\theta})}{\pi(\mathbf{x})} d\mathbf{x} \\
&= \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta})} \left[\log p(\mathbf{x}|\boldsymbol{\theta}) - \log \pi(\mathbf{x}) \right] \rightarrow \min_{\boldsymbol{\theta}}
\end{aligned}
$$

Autoregressive model

MLE problem

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X}|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(\mathbf{x}_i|\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(\mathbf{x}_i|\boldsymbol{\theta}).$$

- We would like to solve the problem using gradient-based optimization.
- We have to efficiently compute $\log p(\mathbf{x}|\boldsymbol{\theta})$ and $\frac{\partial \log p(\mathbf{x}|\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$.

Likelihood as product of conditionals

Let $\mathbf{x} = (x_1, \ldots, x_m)$, $\mathbf{x}_{1:i} = (x_1, \ldots, x_i)$. Then

$$p(\mathbf{x}|\boldsymbol{\theta}) = \prod_{i=1}^{m} p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}); \quad \log p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{i=1}^{m} \log p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}).$$

Example: $p(x_1, x_2, x_3) = p(x_2) \cdot p(x_1|x_2) \cdot p(x_3|x_1, x_2)$.

Autoregressive models

$$\log p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{i=1}^{m} \log p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta})$$

- Sampling is sequential:
  - sample $\hat{x}_1 \sim p(x_1|\boldsymbol{\theta})$;
  - sample $\hat{x}_2 \sim p(x_2|\hat{x}_1, \boldsymbol{\theta})$;
  - ...
  - sample $\hat{x}_m \sim p(x_n|\hat{\mathbf{x}}_{1:m-1}, \boldsymbol{\theta})$;
  - new generated object is $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m)$.
- Each conditional $p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta})$ could be modelled by neural network.
- Modelling all conditional distributions separately is infeasible and we would obtain separate models. To extend to high dimensions we could share parameters $\boldsymbol{\theta}$ across conditionals.

Autoregressive models

For large i the conditional distribution $p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta})$ could be infeasible. Moreover, the history $\mathbf{x}_{1:i-1}$ has non-fixed length.

Markov assumption

$$p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}) = p(x_i|\mathbf{x}_{i-d:i-1}, \boldsymbol{\theta}), \quad d \text{ is a fixed model parameter.}$$

Example

- $d = 2$;
- $x_i \in \{0, 255\}$;
- $\mathbf{h}_i = \text{MLP}_{\boldsymbol{\theta}}(x_{i-1}, x_{i-2})$;
- $\boldsymbol{\pi}_i = \text{softmax}(\mathbf{h}_i)$;
- $p(x_i|x_{i-1}, x_{i-2}, \boldsymbol{\theta}) = \text{Categorical}(\boldsymbol{\pi}_i)$.

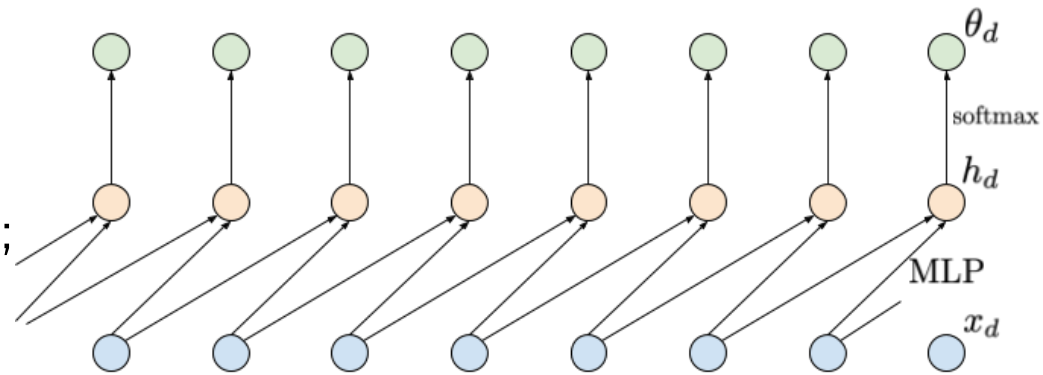
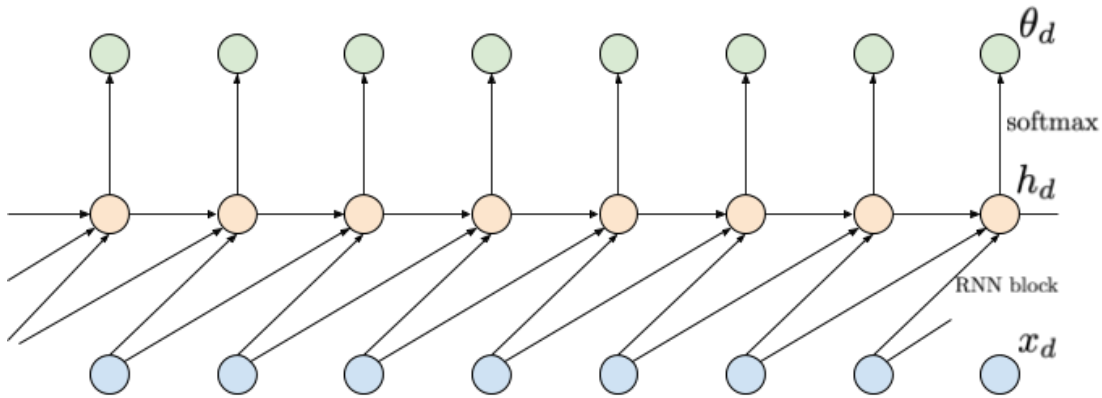


image credit: https://jmtomczak.github.io/blog/2/2_ARM.html

Autoregressive models

- Previous model has **limited** memory d. It is insufficient for many modalities (e.g. for images and text).
- Recurrent NN fixes this problem and potentially could learn long-range dependencies:

$$p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta}) = p(x_i|\mathbf{h}_i, \boldsymbol{\theta}), \quad \mathbf{h}_i = \text{RNN}(\mathbf{x}_{i-1}, \mathbf{h}_{i-1})$$

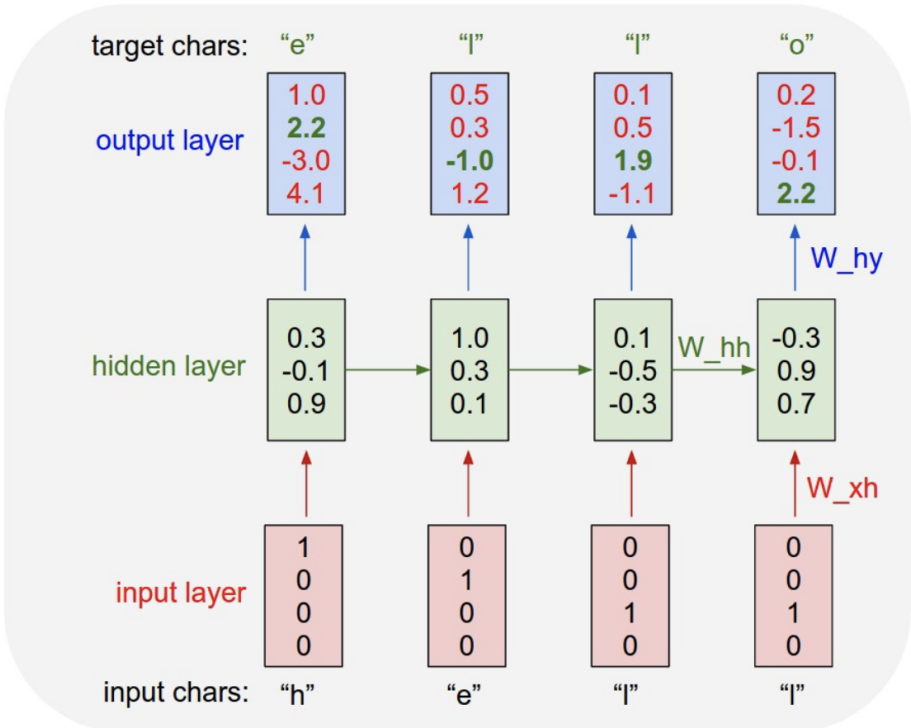


- Sequential computation of all conditionals $p(x_i|\mathbf{x}_{1:i-1}, \boldsymbol{\theta})$, hence, the training is slow.
- RNN suffers from vanishing and exploding gradients.

image credit: https://jmtomczak.github.io/blog/2/2_ARM.html

Char RNN

Model tries to predict the next token (single letter) from previous context.

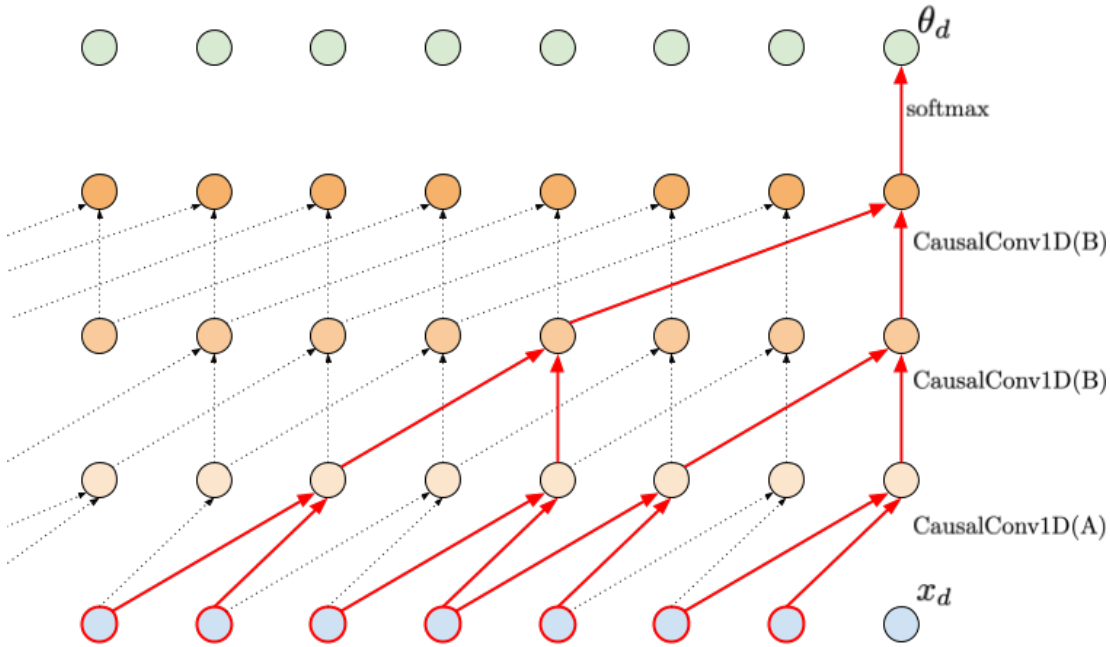



```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

image credit: http://karpathy.github.io/2015/05/21/rnn-effectiveness

Autoregressive models

- Convolutions could be used for autoregressive models, but they have to be **causal**.
- Try to find and understand the difference between Conv A/B.

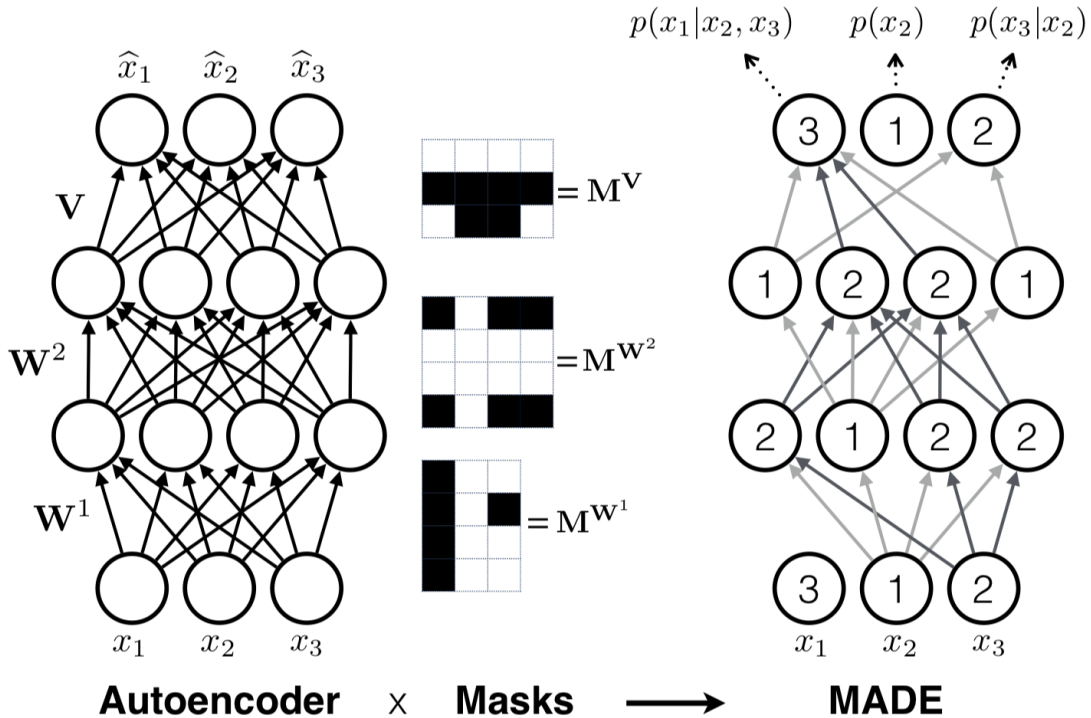


- Could learn long-range dependecies.
- Do not suffer from gradient issues.
- Easy to estimate probability for given input, but hard generation of new samples (the sequential process).

image credit: https://jmtomczak.github.io/blog/2/2_ARM.html

MADE

- Vanila autoencoder is not a generative model. Why?
- Let mask the weight matrices to make the model generative: $\mathbf{W}_M = \mathbf{W} \cdot \mathbf{M}$.



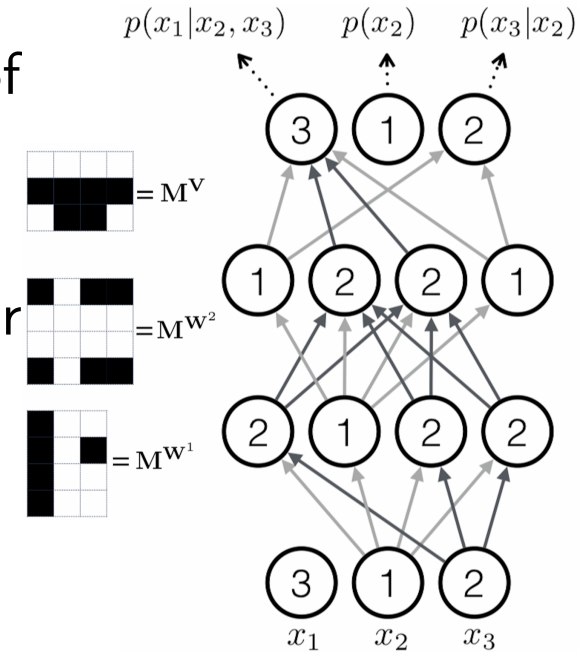
- The question is how to create matrices $\mathbf{M}$ which produce the autoregressive property?

Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015

MADE

Masks generation

- Define the ordering of input elements from 1 to m.
- Assign the random number k from 1 to $m-1$ to each hidden unit. The number gives the maximum number of input units to which the unit can be connected.
- Connect each hidden unit with number k with the previous layer units which has the number is **less or equal** than k.
- Connect each output unit with number k with the previous layer units which has the number is **less** than k.

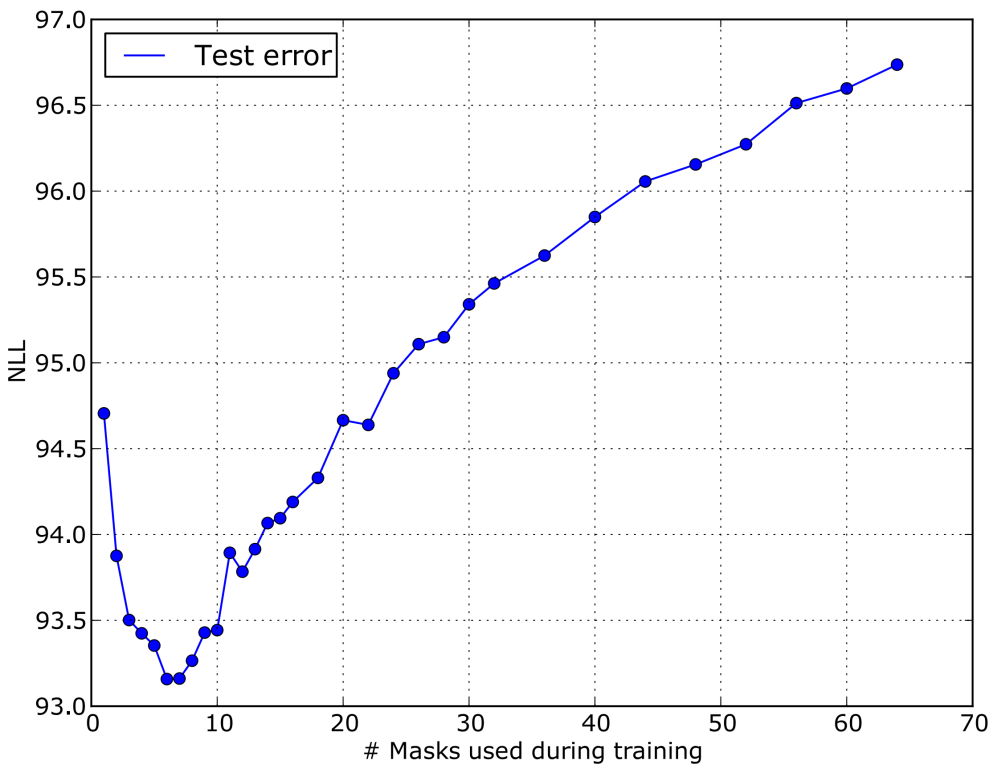


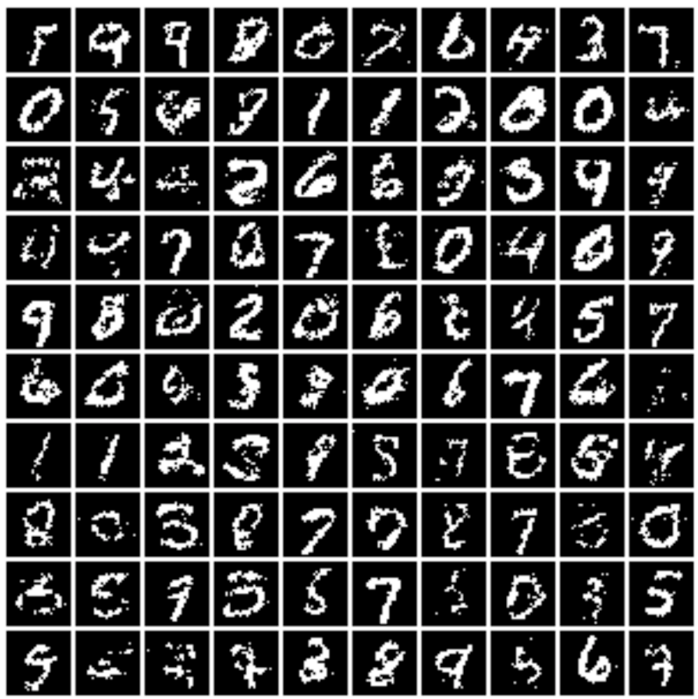
Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015

MADE

Possible variations

- Order agnostic training (missing values in partially observed input vectors can be imputed efficiently);
- Connectivity-agnostic training (cheap ensembling).





Germain M. et al. Made: Masked autoencoder for distribution estimation, 2015

Summary

- We are trying to approximate the distribution of samples for density estimation and generation of new samples.
- To fit model distribution to the real data distribution one could use divergence minimization framework.
- Minimization of forward KL is equivalent to the MLE problem.
- Autoregressive models decompose the distribution to the sequence of conditionals.
- Sampling from autoregressive models is trivial, but sequential
  - sample $x_1 \sim p(x_1)$;
  - sample $x_2 \sim p(x_2|x_1)$;
  - ....
- Density estimation:

$$p(\mathbf{x}) = \prod_{i=1}^{m} p(x_i|\mathbf{x}_{1:i-1}).$$

- Autoregressive models work on both continuous and discrete data.