Chapter 8, Part 3: Variational Auto–Encoders

Advanced Topics in Statistical Machine Learning

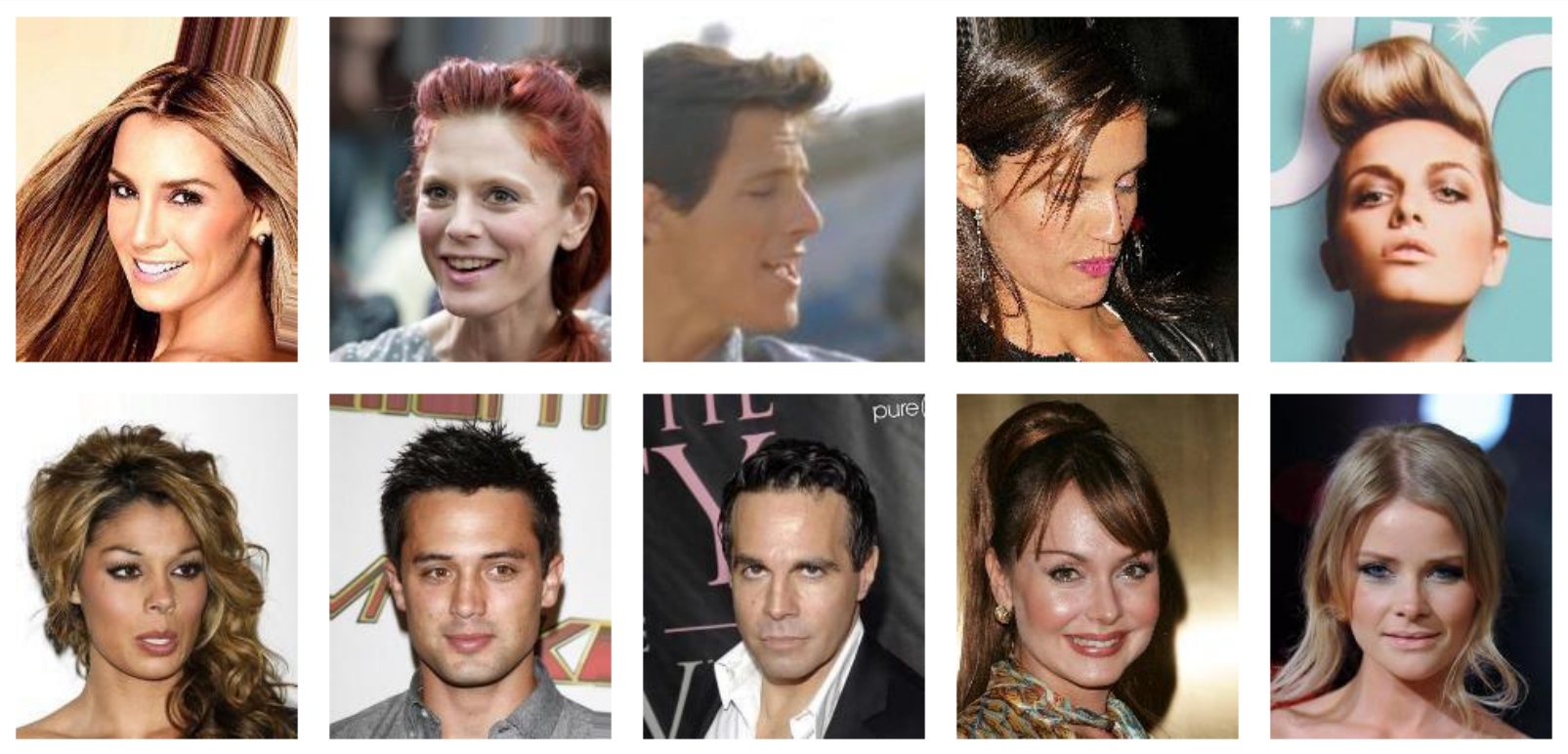
Tom Rainforth

Hilary 2024

rainforth@stats.ox.ac.uk

What If We Can't Manually Construct a Generative Model?

How could we manually construct a generative model for faces?



This would be almost impossible even with a huge about of prior expertise and time to carefully construct the model

[1] G Perarnau et al. "Invertible conditional gans for image editing". In: **arXiv preprint arXiv:1611.06355** (2016).

Learning Factorized Latent Variable Models

Here we instead want to learn a generative model $p_\theta(\mathbf{X}, \mathbf{Z})$ **directly from data**: we will introduce some trainable model parameters θ and then optimize θ to produce the best model

This could be as simple a optimizing the means in a GMM, but we can also consider defining a highly **flexible** model class through θ

As it will typically be reasonable to assume our data is i.i.d. given θ, we will use a factorized latent variable model (LVM) framework

Factorized LVMs have separable posteriors, $p_\theta(z_i|\mathbf{X}) = p_\theta(z_i|x_i)$, and so we can think of the generative model as being defined using only a single arbitrary datapoint, i.e. we can think about our model as being $p_\theta(z)p_\theta(x|z)$ and $\mathbf{X}$ as a set of **examples** that we can use to **learn** θ

Deep Generative Models

A **deep generative model** (DGM) is a (typically factorized) LVM where $x|z$ is based on a deep neural network with parameters θ

- For some models, like VAEs, the network maps to explicit distribution parameters, e.g. $p_\theta(x|z) = \mathcal{N}(x; \mu_\theta(z), \Sigma_\theta(z))$ where μ_θ and Σ_θ are deep neural networks
- For others, like GANs, the mapping $z \mapsto x$ is deterministic, such that all stochasticity is dealt with in z and we have an implicit model $p_\theta(x)$ defined by $p_\theta(z)$ and the map $z \mapsto x$

Typically $p_\theta(z)$ is a simple **fixed** distribution (most commonly $p_\theta(z) = \mathcal{N}(z; 0, I)$) so we will drop the θ subscript for the prior

- Note that z does not need to be factorized: the factorization is over different datapoints, not dimensions

Alternative View of MML for Factorized LVMs

As previously alluded to, a natural framework for model learning is to perform **maximum marginal likelihood** (MML) estimation

$$\theta^* = \arg\max_\theta p_\theta(\mathbf{X}) = \arg\max_\theta \sum_{i=1}^n \log p_\theta(x_i)$$

For factorized LVMs, we can think of MML as an "empirical risk minimization" where the loss is the cross entropy loss $L(\theta, x) = -\log p_\theta(x)$, or equivalently we a minimizing the KL divergence from the **true** data distribution $p_{\text{true}}(x)$ to the model marginal $p_\theta(x)$

$$\begin{aligned}
\arg\max_\theta p_\theta(\mathbf{X}) &= \arg\min_\theta \frac{1}{n} \sum_{i=1}^n -\log p_\theta(x_i) \\
&\xrightarrow{n\to\infty} \arg\min_\theta \mathbb{E}_{p_{\text{true}}(x)} \left[-\log p_\theta(x)\right] \\
&= \arg\min_\theta \mathbb{D}_{\text{KL}}(p_{\text{true}}(x) \,\|\, p_\theta(x))
\end{aligned}$$

Model Learning via Stochastic–Gradient–Based Optimization

Given we are using deep models, we need a **differentiable** objective to optimize which we can produce **mini-batch** estimates for

In many ways, the marginal likelihood seems well–suited for this

$$\nabla_\theta \log p_\theta(\mathbf{X}) = \sum_{i=1}^{n} \nabla_\theta \log p_\theta(x_i)$$

Here our **factorization** is the key to allowing mini-batching of data and thus the use stochastic gradient ascent (SGA) methods:

$$\nabla_\theta \log p_\theta(\mathbf{X}) \approx \frac{n}{|B|} \sum_{i \in B} \nabla_\theta \log p_\theta(x_i)$$

where $B \subset \{1, \ldots, n\}$ is our mini-batch and this sub-sampling of data does not itself introduce any bias into our gradients

Unfortunately, there are still a number of hurdles to overcome

Problem 1: Intractable Loss

The first and most apparent problem is that our "loss function"

$$L(x, \theta) = -\log p_\theta(x) = -\log \left(\mathbb{E}_{p(z)}[p_\theta(x|z)] \right)$$

is itself **intractable** for a given x

Typically, datapoints are not even the inputs to our network(s): our generative model is $p(z)p_\theta(x|z)$ and so we are learning a mapping from the unknown latents z

In principle, we can deal with both by forming a **Monte Carlo estimate** each time we need to evaluate a $p_\theta(x_i)$, namely

$$p_\theta(x_i) \approx \hat{p}_\theta(x_i) = \frac{1}{m} \sum_{j=1}^{m} p_\theta(x_i|\hat{z}_{i,j}) \quad \text{where} \quad \hat{z}_{i,j} \overset{\text{i.i.d.}}{\sim} p(z) \quad (1)$$

but this causes its own issues...

Problem 2: Lack of an Unbiased Gradient Estimator

By Jensen's inequality, $\mathbb{E}[\log \hat{p}_\theta(x_i)] \leq \log \mathbb{E}[\hat{p}_\theta(x_i)] (= \log p_\theta(x_i))$ with equality if and only if the variance of $\hat{p}_\theta(x_i)$ is zero

This means that even though $\hat{p}_\theta(x_i)$ is an unbiased estimate of $p_\theta(x_i)$, the $\log$ induces a **bias** in the estimate of the loss function

Moreover, this bias persists to the estimation of derivatives:

$$\mathbb{E}\left[\nabla_\theta \log \hat{p}_\theta(x_i)\right] = \mathbb{E}\left[\frac{\sum_{j=1}^m \nabla_\theta p_\theta(x_i|\hat{z}_{i,j})}{\sum_{j=1}^m p_\theta(x_i|\hat{z}_{i,j})}\right] \neq \nabla_\theta \log p_\theta(x_i)$$

such that our SGA scheme will not converge to a local optimum of the marginal likelihood

Though correcting this bias is theoretically possible (e.g. Russian Roulette sampling), the resulting schemes are rarely practical

The ELBO as a Proxy for the log Evidence

Given optimizing $\log p_\theta(\mathbf{X})$ directly seems problematic, could we instead work with the ELBO?

$$\begin{aligned}\mathcal{L}(\mathbf{X}, \theta, \phi) &:= \sum_{i=1}^{n} \mathcal{L}(x_i, \theta, \phi_{z_i}) \\ &= \sum_{i=1}^{n} \mathbb{E}_{q_{\phi_{z_i}}(z_i)} \left[\log \frac{p_\theta(x_i, z_i)}{q_{\phi_{z_i}}(z_i)} \right] \leq \log p_\theta(\mathbf{X})\end{aligned}$$

Let's for now presume that ϕ is fixed and to avoid clutter drop the subscript from ϕ_{z_i}. We thus want to solve the optimization

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \mathbb{E}_{q_\phi(z_i)} \left[\log \frac{p_\theta(x_i, z_i)}{q_\phi(z_i)} \right]$$

The ELBO as a Proxy for the log Evidence (2)

This objective has a critical difference to the log evidence: we can directly construct unbiased gradient updates for it and run SGA

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\mathbf{X}, \theta, \phi) &= \sum_{i=1}^{n} \nabla_\theta \, \mathbb{E}_{q_\phi(z_i)} \left[\log \frac{p_\theta(x_i, z_i)}{q_\phi(z_i)} \right] \\
&= \sum_{i=1}^{n} \mathbb{E}_{q_\phi(z_i)} \left[\nabla_\theta \log \frac{p_\theta(x_i, z_i)}{q_\phi(z_i)} \right] \\
&= \sum_{i=1}^{n} \mathbb{E}_{q_\phi(z_i)} \left[\nabla_\theta \log p_\theta(x_i, z_i) \right] \\
&\approx \frac{n}{|B|} \sum_{i \in B} \nabla_\theta \log p_\theta(x_i, \hat{z}_i) =: \widehat{\nabla_\theta \mathcal{L}}(\mathbf{X}, \theta, \phi)
\end{aligned}$$

where each $\hat{z}_i \sim q_\phi(z_i)$ (i.e. we use a single sample estimate of the expectation) and $\mathbb{E}[\widehat{\nabla_\theta \mathcal{L}}(\mathbf{X}, \theta, \phi)] = \nabla_\theta \mathcal{L}(\mathbf{X}, \theta, \phi)$ as required

Will This Give Us Anything Sensible Though?

Will this give us anything sensible though? Let's break it down using the other form of the ELBO

$$\mathcal{L}(x, \theta, \phi) = \log p_\theta(x) - \mathbb{D}_{\mathsf{KL}}(q_\phi(z) \,\|\, p_\theta(z|x))$$

We thus see we have two competing terms, optimizing the ELBO wants to:

- Increase the marginal likelihood $\log p_\theta(x)$
- Reduce the divergence between $p_\theta(z|x)$ and $q_\phi(z)$ (remembering we are assuming the later is fixed for now)

As the KL is lower bounded, there is only so much we can gain by reducing it: we should expect at least some improvements in the true log evidence

Variational EM

Nonetheless, if $q_\phi(z)$ is kept fixed, this KL term is going to be restrictive to what we learn: we can only learn models whose posterior is close to $q_\phi(z)$

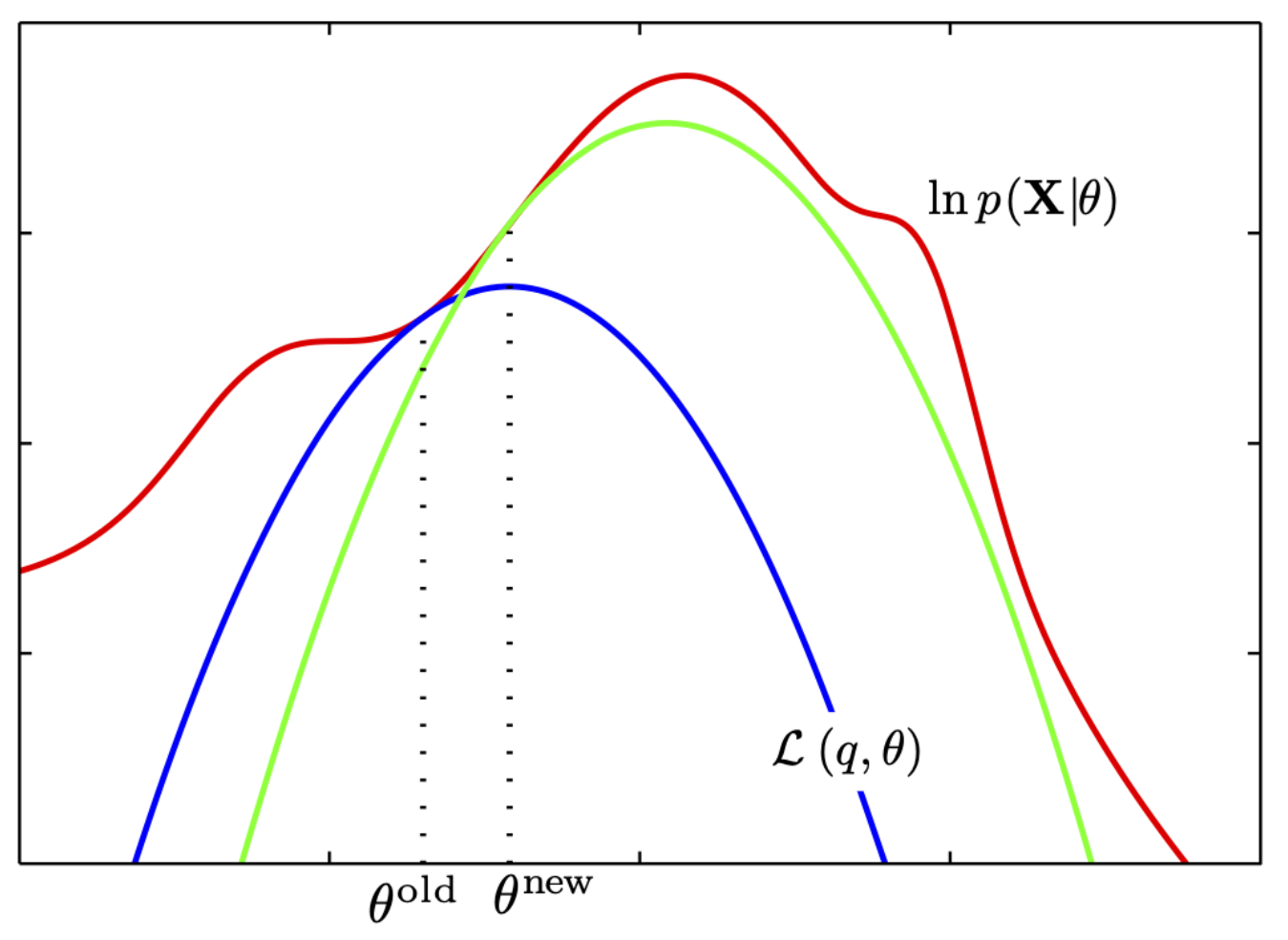
We can alleviate this issue by instead alternating between updating our model and finding a new variational approximation,
i.e. alternate between

1. Update θ using SGA with fixed ϕ
2. Update ϕ by running variational inference with fixed θ

This is known as (stochastic) **variational expectation maximization** (variational EM)

Note that both steps improve the ELBO (modulo the stochasticity our gradient estimates)

Variational EM (2)



Credit: Bishop Section 9.4

Dealing with Large Datasets

The need to alternate between updating θ and ϕ in this approach can be potentially very inefficient

In particular, our variational parameters $\phi = \{\phi_{z_1}, \ldots, \phi_{z_n}\}$ are actually a collection of parameters specific to each each datapoint

Naively, we would have to update this full set of n parameters at each variational inference step, which is very expensive

We can do a bit better by alternating between minibatch updates to θ and ϕ, but now each ϕ_{z_i} is only updated once an epoch, but then the many updates to θ mean each ϕ_{z_i} becomes outdated between updates, such that we need many gradient steps in each variational inference step to avoid a loose ELBO

Amortized Inference

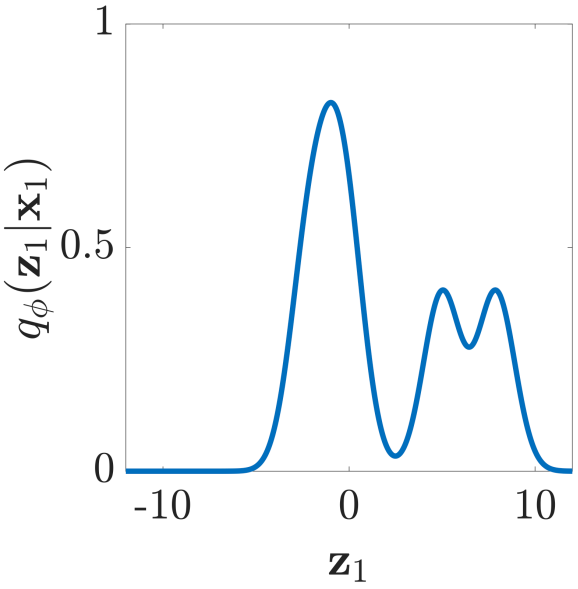
Solution: learning a **mapping** from datapoints to variational approximations instead.

Specifically, we can learn an conditional approximation of the form $q_\phi(z|x)$.

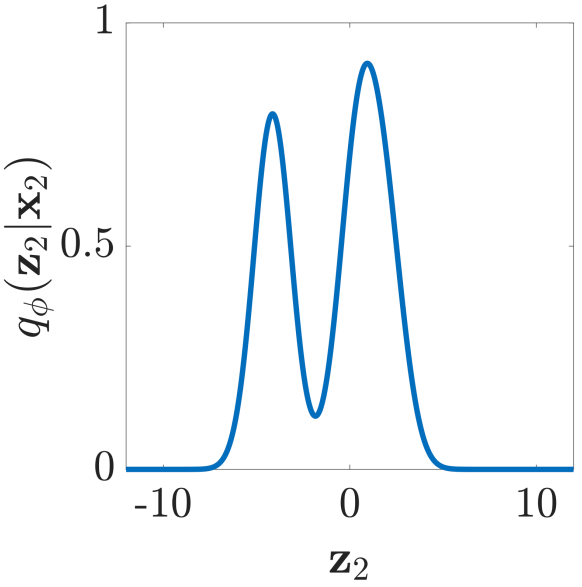
This is known as an **amortized** approximation, and the general process as **amortized inference**

Amortized Inference (2)

$q_\phi(\mathbf{z}|\mathbf{x})$



$q_\phi(\mathbf{z}|\mathbf{x})$



Amortized Inference (3)

- More concretely, in amortized inference we learn a **mapping** from datapoints to our distribution parameters
- This is done by introducing a parameterized **inference network**, typically a deep neural network itself, which ϕ now corresponding to the trainable parameters of that network
- One common choice is to use a Gaussian:

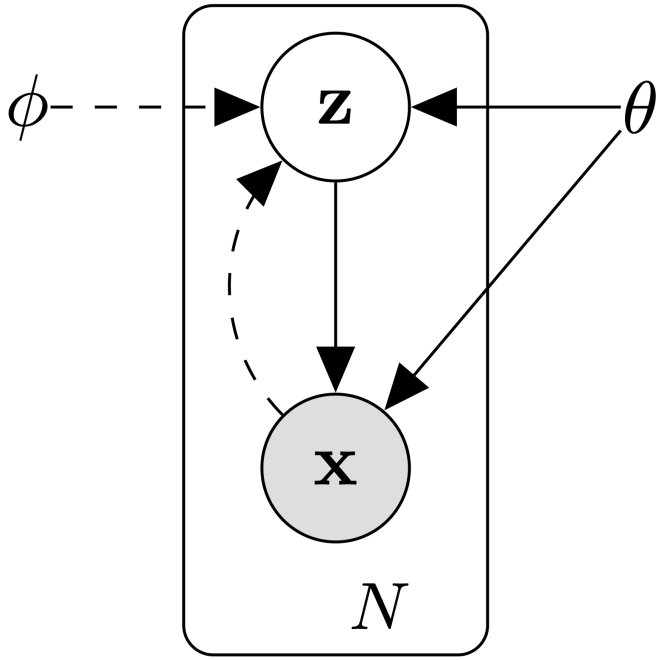
$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \Sigma_\phi(x))$$

  where μ_ϕ and Σ_ϕ are deep neural networks
- Learning a mapping rather than a individual variational approximation allows us to **share information** across datapoints and provide good approximations for points that we have not encountered for a while or even never seen before

Variational Auto–Encoders

$q_\phi(z|x)$

Inference Network



$p_\theta(x|z)$

Generative Network

Set $p(z) = \mathcal{N}(z; 0, I)$ and **simultaneously** maximize

$$\mathcal{L}(\mathbf{X}, \theta, \phi) := \sum_{i=1}^{n} \mathbb{E}_{q_\phi(z_i|x_i)} \left[\log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \right] \tag{2}$$

with respect to **both** θ and ϕ

A Complication: Gradients for the Inference Network

As before, we naturally wish to maximize $\mathcal{L}(\mathbf{X}, \theta, \phi)$ using SGA

However, a complication arises with this when we try and take derivatives with respect to ϕ: these also effect the distribution the expectation is taken with respect to

$$\begin{aligned}
&\nabla_\phi \, \mathbb{E}_{q_\phi(z_i|x_i)} \left[\log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \right] \\
&\qquad = \mathbb{E}_{q_\phi(z_i|x_i)} \left[\nabla_\phi \log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \right] \\
&\qquad\quad + \int \log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \nabla_\phi q_\phi(z_i|x_i) dz_i
\end{aligned}$$

This second term is not an expectation so we have extra work to produce an appropriate stochastic gradient for it

The Score Function Estimator

One simple way to deal with this is to note that, as $\nabla x = x \nabla \log x$,

$$\begin{aligned}
&\nabla_\phi \, \mathbb{E}_{q_\phi(z_i|x_i)} \left[\log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \right] \\
&\qquad = \underbrace{\mathbb{E}_{q_\phi(z_i|x_i)} \left[-\nabla_\phi \log q_\phi(z_i|x_i) \right]}_{=0} \\
&\qquad\quad + \mathbb{E}_{q_\phi(z_i|x_i)} \left[\log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \nabla_\phi \log q_\phi(z_i|x_i) \right]
\end{aligned}$$

where the first term is zero because $\int \nabla_\phi q_\phi(z_i|x_i) dz_i = 0$

We can now construct unbiased gradient estimates by simple Monte Carlo which is known as the **score function** estimator or the **reinforce** estimator.

Unfortunately, this tends to have very high variance

The Reparameterization Trick

An alternative approach which is lower variance but which cannot be directly applied for discrete latents is to **reparameterize** z

The key idea is to express $q_\phi(z|x)$ in the form $\epsilon \sim q(\epsilon)$, $z = g(\epsilon, x, \phi)$, such that $q(\epsilon)$ is a simple, fixed, distribution and g is a deterministic mapping that ensures z has distribution $q_\phi(z|x)$

We can then express the expectation in terms of ϵ and thus move the gradient inside as follows

$$\nabla_\phi \, \mathbb{E}_{q_\phi(z_i|x_i)} \left[\log \left(\frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)} \right) \right] = \mathbb{E}_{q(\epsilon)} \left[\nabla_\phi \log \left(\frac{p_\theta(x_i, g(\epsilon_i, x_i, \phi))}{q_\phi(g(\epsilon_i, x_i, \phi)|x_i)} \right) \right]$$

Putting it All Together: the VAE Training Algorithm

The VAE training algorithm cycles through the following steps (presuming a reparameterization approach)

1. Sample a mini-batch of datapoints $B \subseteq \{1, \ldots, n\}$
2. Construct a gradient estimate of the ELBO using B and a single sample estimate of the expectation over each z_i

$$\widehat{\nabla_{\theta,\phi}\mathcal{L}}(\mathbf{X}, \theta, \phi) = \frac{n}{|B|} \sum_{i \in B} \nabla_{\theta,\phi} \log \frac{p_\theta(x_i, \hat{z}_i)}{q_\phi(\hat{z}_i | x_i)} \tag{3}$$

   where each $\hat{z}_i = g(\epsilon_i, x_i, \phi)$ and $\epsilon_i \sim q(\epsilon)$
3. Update the parameters using a stochastic gradient maximizer, e.g. SGA with step sizes ρ_t and η_t at iteration t:

$$\theta \leftarrow \theta + \rho_t \widehat{\nabla_\theta \mathcal{L}}(\mathbf{X}, \theta, \phi)$$

$$\phi \leftarrow \phi + \eta_t \widehat{\nabla_\phi \mathcal{L}}(\mathbf{X}, \theta, \phi)$$

4. Repeat until convergence or computational budget reached

Example Samples from Deep Generative Models

Once trained, we can sample using $\hat{z} \sim p(z)$, $\hat{x} \sim p_\theta(x|z = \hat{z})$

NVAE: A Deep Hierarchical Autoencoder. Vahda and Kautz. NeurIPS 2020

Very Deep VAEs Generalize Autoregressive Models. Child. ICLR 2021

The Auto-Encoder View of VAEs

We can also view the VAE as a stochastic auto–encoder where the inference network=encoder and the generative network=decoder

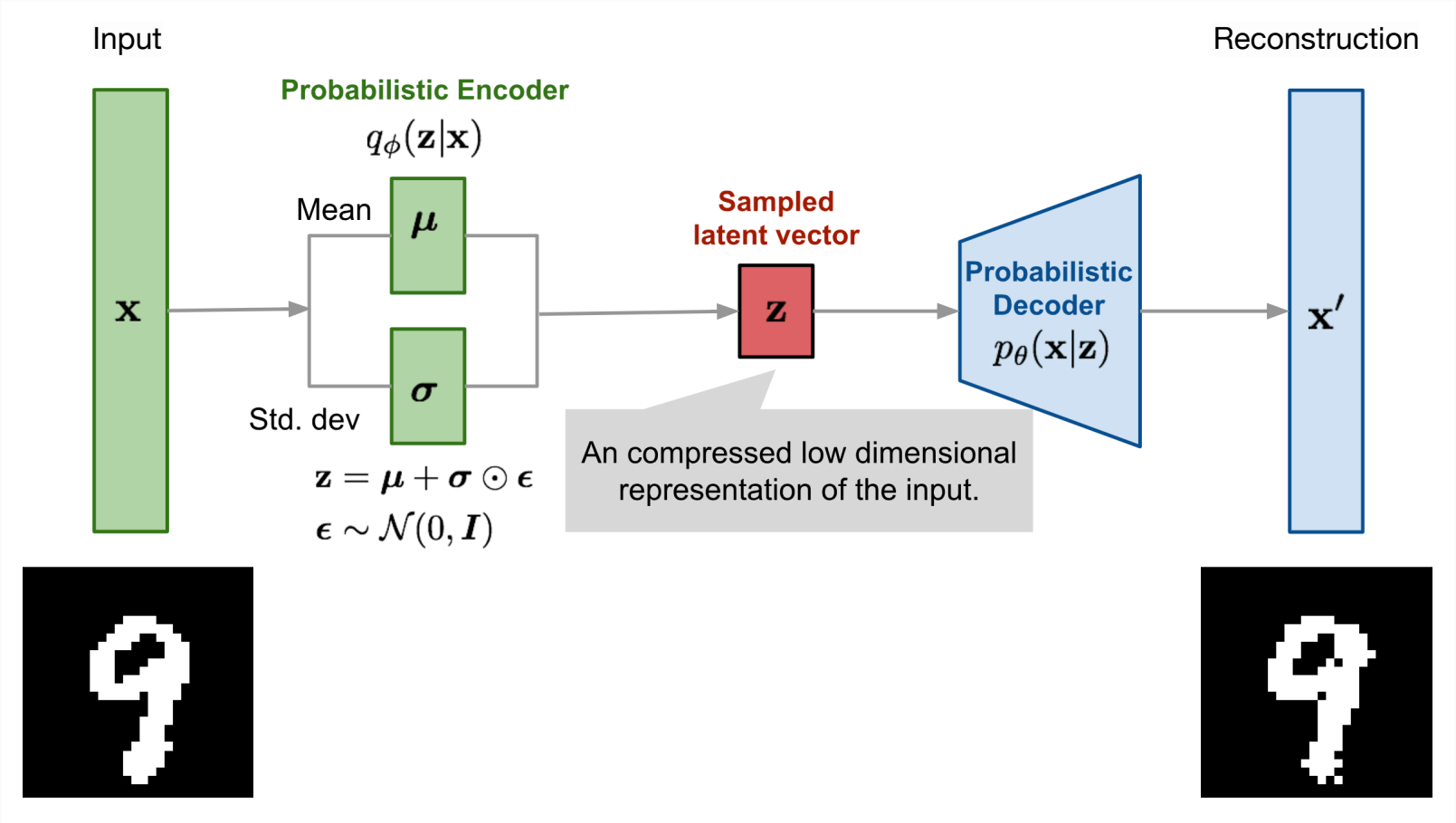
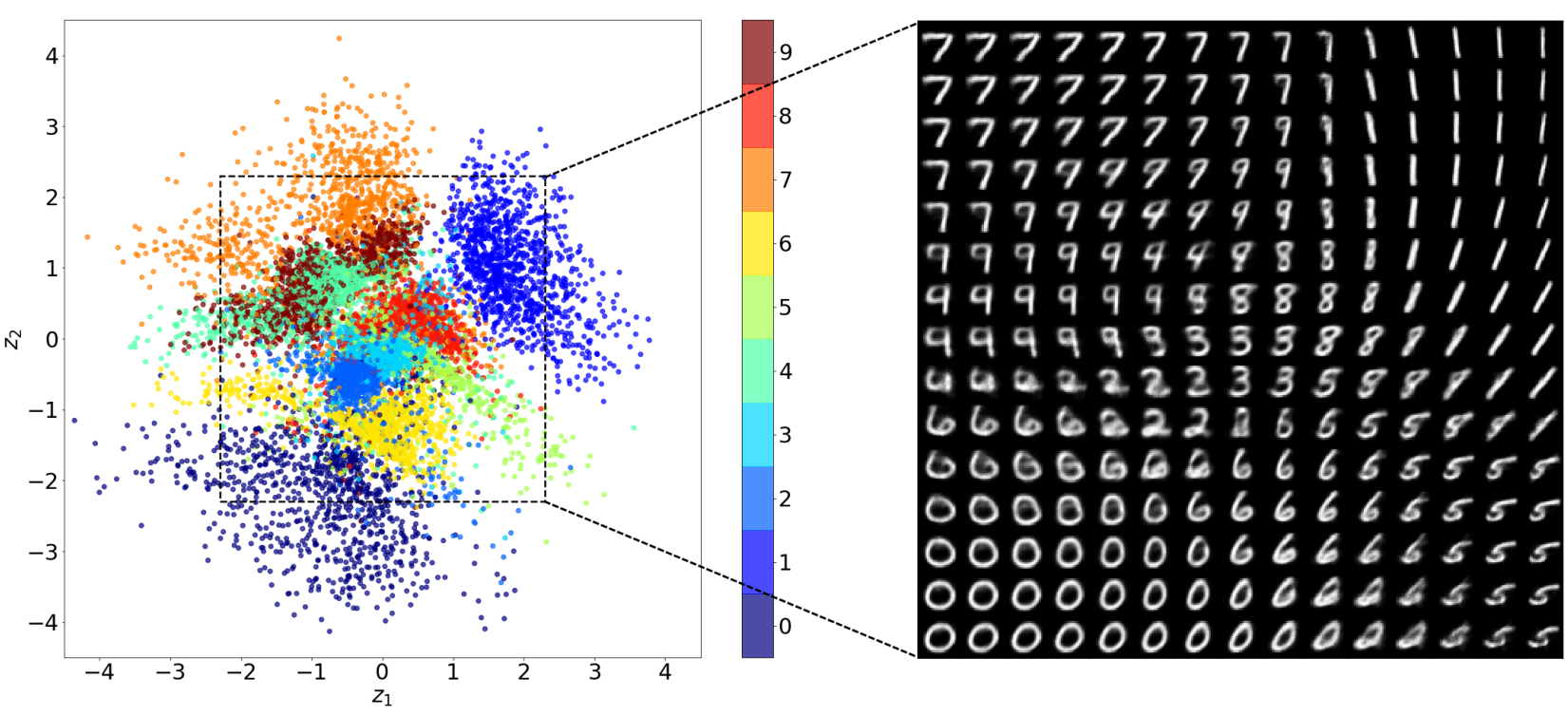


Image adapted from one by Lilian Weng

VAEs for Representation Learning

From this, we see that VAEs can also be view from a **representation learning** perspective: the latents provide a (stochastic) embedding of the data



Credit: `argmax.ai`

Rearranging the ELBO

We can gain insights into this by rearranging the ELBO to

$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - \mathbb{D}_{\mathsf{KL}}(q_\phi(z|x) \,\|\, p(z))$$

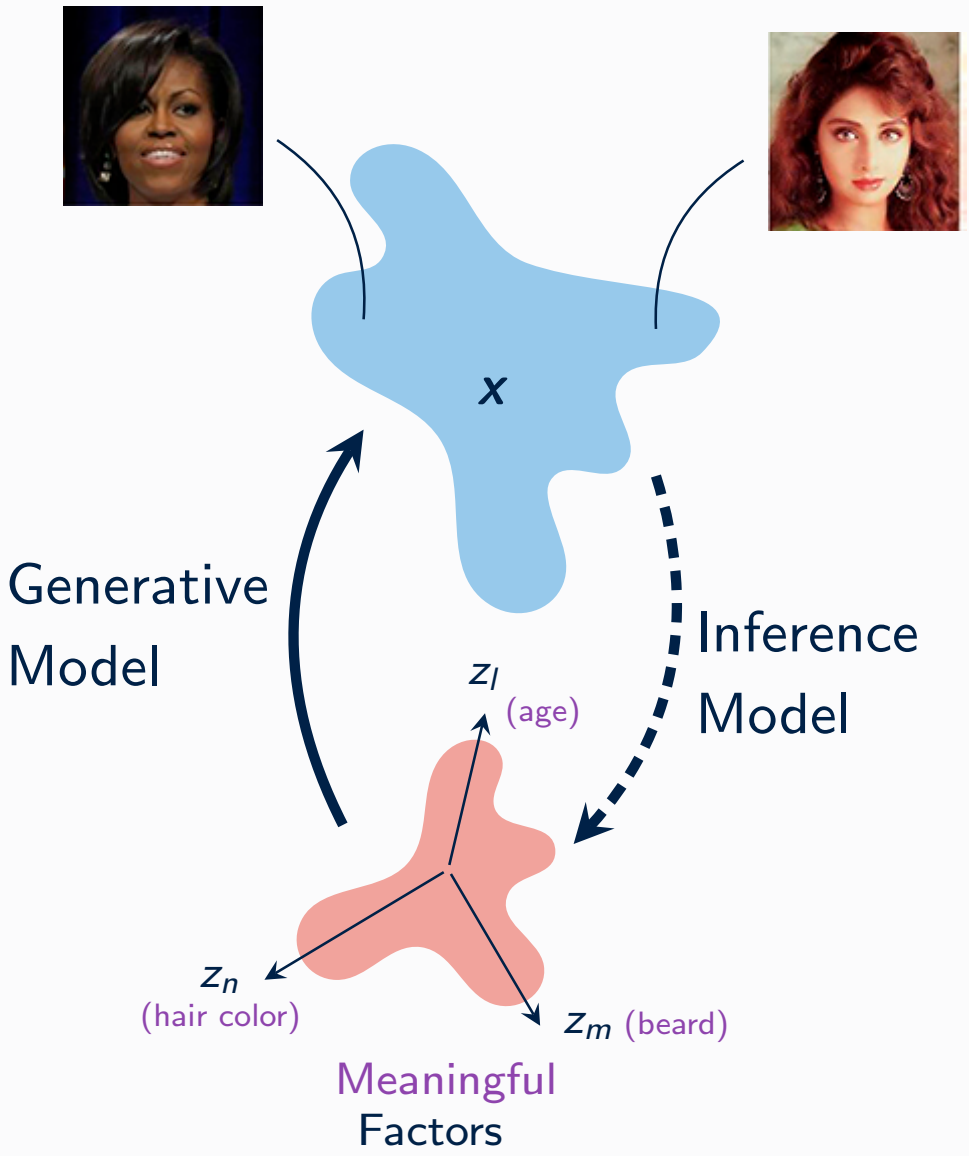
Here the first term is known as the **reconstruction loss** and measures how well the original input is reconstructed during the stochastic encoding–decoding process

On its own, the reconstruction loss encourages as little noise as possible in the encoding–decoding process and reconstructions that match the input as close as possible

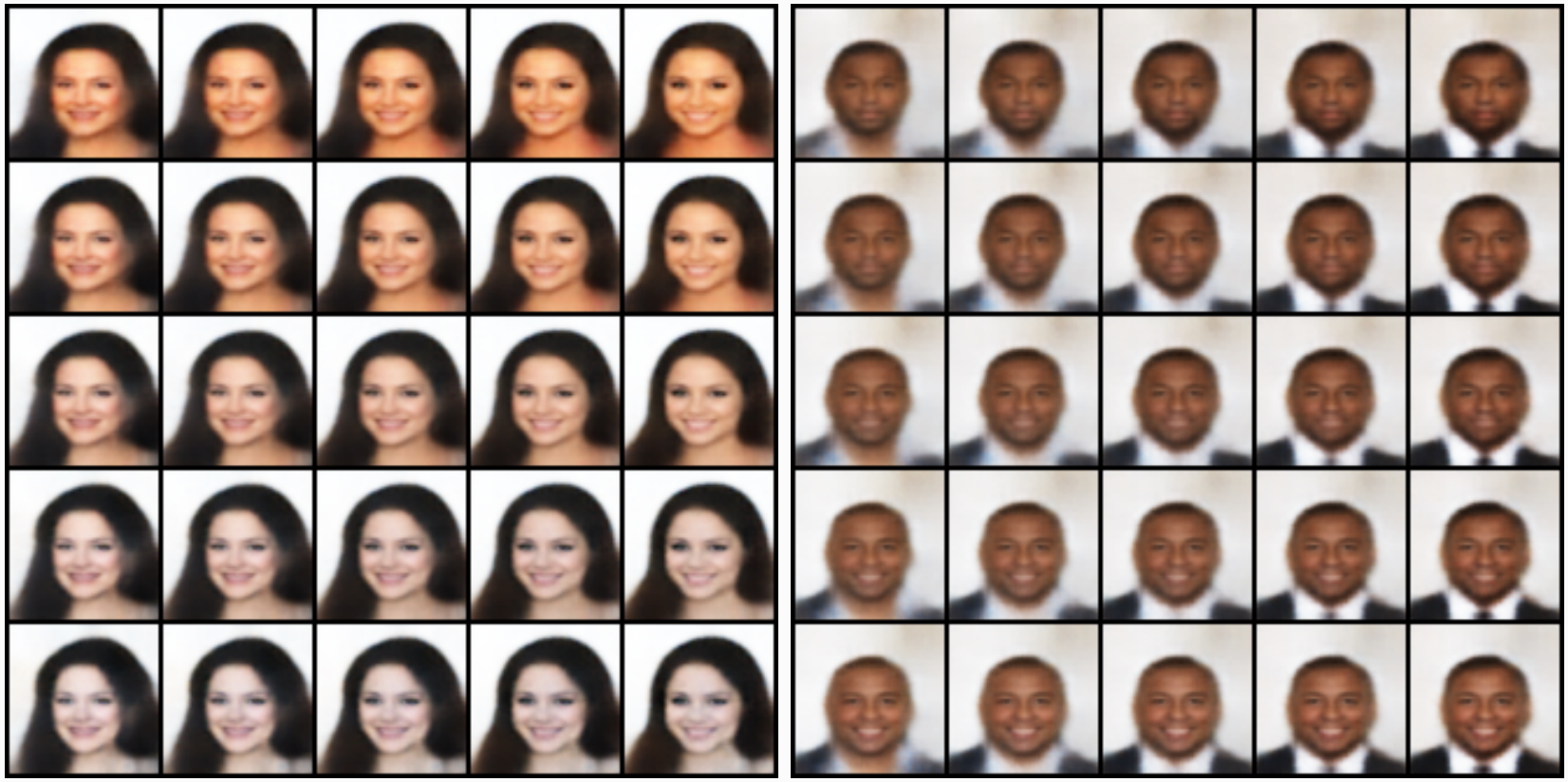
This behavior is unwanted as we simply get lookup table behavior that cannot generalize to other inputs or generate novel examples

The $\mathbb{D}_{\mathsf{KL}}(q_\phi(z|x) \,\|\, p(z))$ term acts as a regularizer that guards against overfitting and encourages **smoothness** of the latent space

Learning Meaningful Representations



Meaningful Latent Spaces



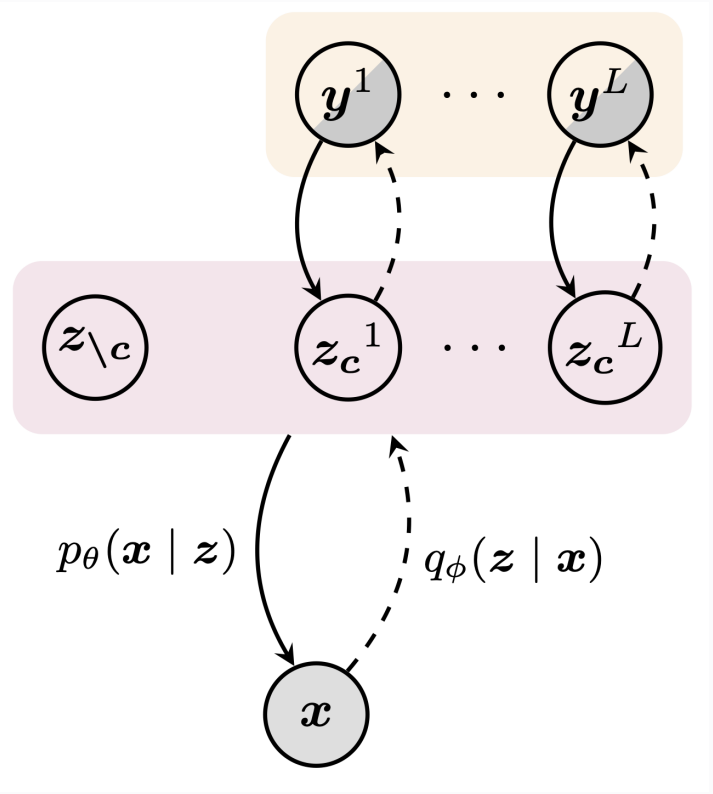
[Left] Varying latents corresponding to age and skin tone. [Right] Varying latents corresponding to smiling and wearing a tie. Source: Joy, Schmon, Torr, Siddharth, and Rainforth. Capturing Label Characteristics in VAEs. ICLR 2021

Semi–Supervised VAEs

One way to induce meaningful latents is to exploit any available labels to capture characteristics

For example, we can simultaneously train classifiers from certain latents to certain labels to force information relevant to those labels to be stored in the corresponding latents

This can be done in a semi–supervised manner, such that we only need a labels for a small number of datapoints



CCVAE model from Joy et al 2021 `https://github.com/thwjoy/revae-demo`

Recap

- **Deep generative models** allow us to learn powerful generative models directly from data
- They work by parametrizing $x|z$ using a deep neural network and then training this with example data
- One way to do this is by optimizing a lower bound on the log evidence (the **ELBO**)
- **Variational auto-encoders** (VAEs) introduce an **amortized inference network** trained simultaneously to the generative model which allows efficient and effective training
- We can also view VAEs as stochastic autoencoders that can be used to learn compressed **representations** of data, which can then, e.g., be used as features in downstream tasks

Further Reading

- Some discussion of alternative bounds to the standard ELBO is given in the notes
- Modern probabilistic programming systems like Pyro allow for easy construction and training of VAEs and their descendants: `https://pyro.ai/examples/vae.html`, `https://www.youtube.com/watch?v=vgFWeEyen6Y&t=1058s`
- Diederik P Kingma, Max Welling, et al. "An introduction to variational autoencoders". In: **Foundations and Trends in Machine Learning** 12.4 (2019), pp. 307–392
- GANs, another common deep generative model approach: `arxiv.org/abs/1406.2661`, `https://reiinakano.com/gan-playground/`, `https://www.youtube.com/watch?v=HGYYEUSm-0Q`
- Diffusion/denoising/score-based models, current state of the art approach: `https://scorebasedgenerativemodeling.github.io/`,

Thanks for following the course!

Links for feedback: Lectures Feedback Form

Part C Classes

MSc Classes