Introduction to generative modeling

Valentin De Bortoli

February 18, 2022

Outline of the course (1/2)

- **Course 1**: Introduction to generative modeling
  - Motivation of generative modeling.
  - Basics on EBMs, normalizing flows, VAEs and GANs.
- **Course 2**: Score-based generative modeling (introduction & practice)
  - Introduction of diffusion models.
  - Connection with ancestral sampling.
  - A variational approach.
  - **Lab session**: MNIST with score-based generative models.
- **Course 3**: Score-based generative modeling (theory & methodology)
  - Stochastic processes and time-reversal.
  - Diffusion models as maximum likelihood models.
  - Some extensions of score-based generative models.
  - **Exercise session**: likelihood computation with score-based generative models.

Outline of the course (2/2)

- **Course 4**: Towards Schrödinger bridges
  - Beyond score-based generative models.
  - The dynamical Schrödinger Bridge problem.
  - Iterative Proportional Fitting.
  - Diffusion Schrödinger Bridge.
  - **Exercise session**: Regularized Optimal Transport and Schrödinger Bridges.

- **Course 5**: Schrödinger Bridges in practice
  - Back to Diffusion Schrödinger Bridges.
  - A network refinement.
  - Links with stochastic control.
  - Likelihood computation with Diffusion Schrödinger Bridges.
  - **Lab session**: MNIST with Diffusion Schrödinger Bridges.

Introduction to generative modeling

Definition

- **Generative modeling**: Given a distribution $\pi \in \mathcal{P}(\mathbb{R}^d)$ how to obtain sample from π?
  - We have access to $\hat{\pi} = (1/N) \sum_{k=1}^{N} \delta_{x^k}$, the **empirical distribution**.
  - $\{x^k\}_{k=1}^{N}$ are samples from π
- A general approach:
  - Start from an **easy-to-sample** distribution $\pi_0 \in \mathcal{P}(\mathbb{R}^p)$ (p can be different from d).
  - Choose a noise distribution π_Z on a space $(Z, \mathcal{Z})$.
  - Define a mapping $g : \mathbb{R}^p \times Z \to \mathbb{R}^d$ such that $g_{\#}(\pi_0, \pi_Z) \approx \pi$.
- In other words:
  - Sample Z from π_Z, sample X_0 from π_0
  - Push with $g(X_0, Z) \to$ approximate sample from π.

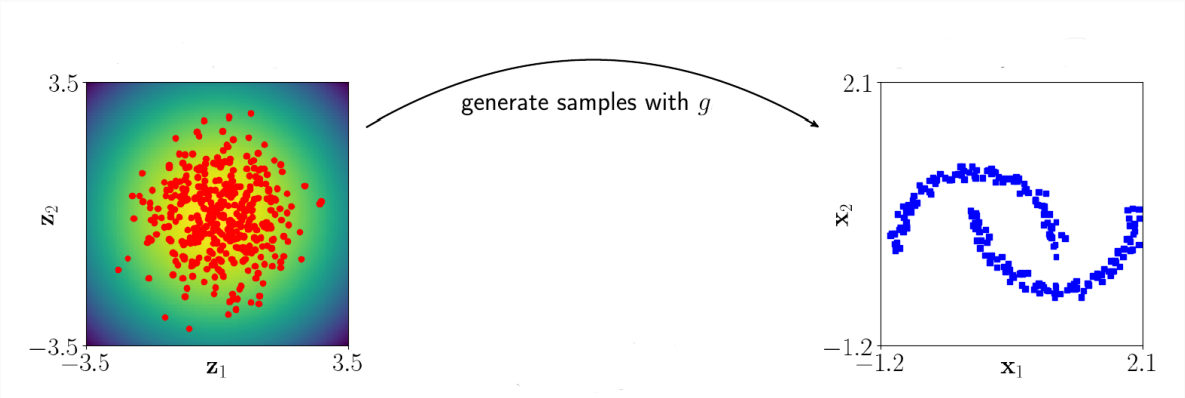

Figure 1: Image adapted from Ruthotto and Haber (2021).

Application (1/3): Data augmentation

- Application in **medical imaging**: Sandfort et al. (2019).
  - **Computerized Tomography (CT)** scans are expensive to generate.
  - Training data: **contrast-enhanced** CT scans.
  - Testing data (real-world data): non-contrast CT scans (**distribution shift**).
- Sandfort et al. (2019) consider a GAN (CycleGAN) to generate non-contrast CT scans from contrast-enhanced CT scans. This is called **data augmentation**.

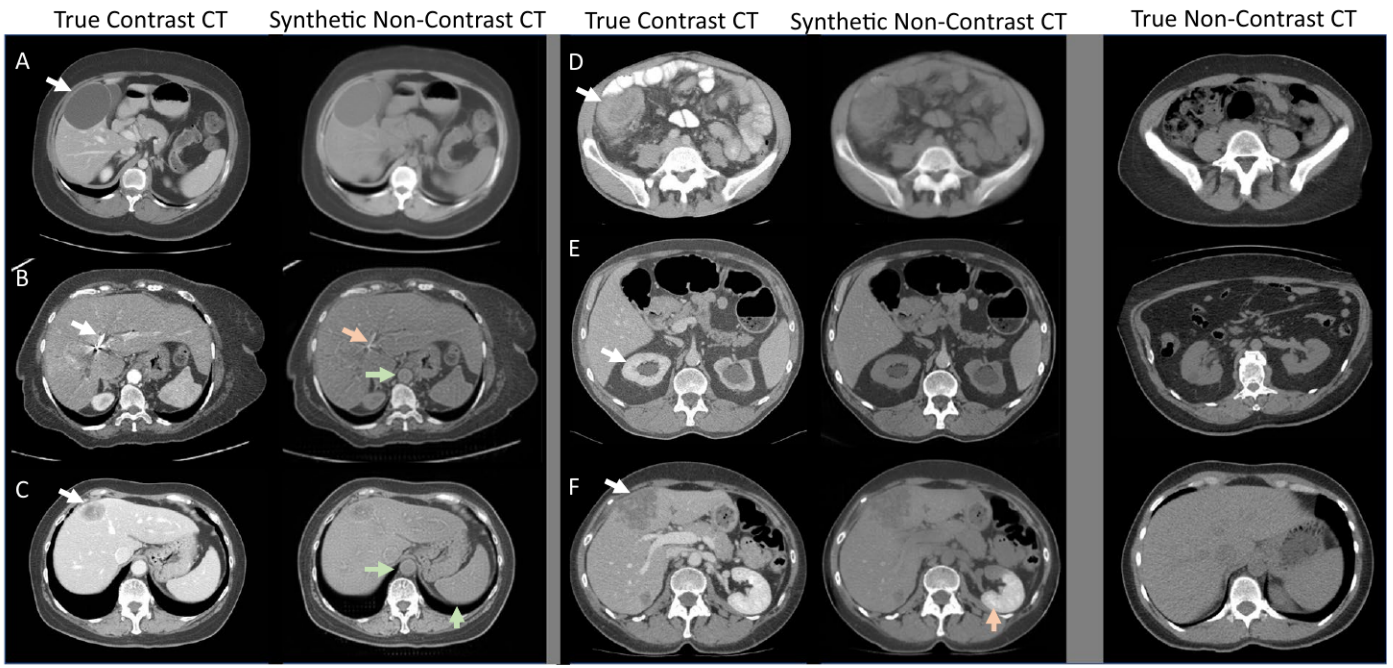

Figure 2: Image extracted from Sandfort et al. (2019).

Application (2/3): Nowcasting

- Application in **meteorology**: Ravuri et al. (2021).
  - Prediction of rain in the next 2 hours: **nowcasting**.
  - Solving physical PDEs: **planet scale** predictions days ahead.
  - Struggle for **high resolution** predictions on short time ranges.
- Access to a lot of high quality data: **conditional GAN**.

Figure 3: Image extracted from Ravuri et al. (2021).

Application (3/3): Protein Folding

- Application in **computational biology**: Senior et al. (2020).
  - **Amino-acid sequence** to **3D structure**.
  - Cryo-Electron Microscopy or crystallography = experimental techniques to determine the shape of the protein.
  - Crystallizing a protein is a real challenge Avanzato et al. (2019).
  - Competition to predict structure: **C**ritical **A**ssessment of protein **S**tructure **P**rediction.
- **Conditional generative modeling**.

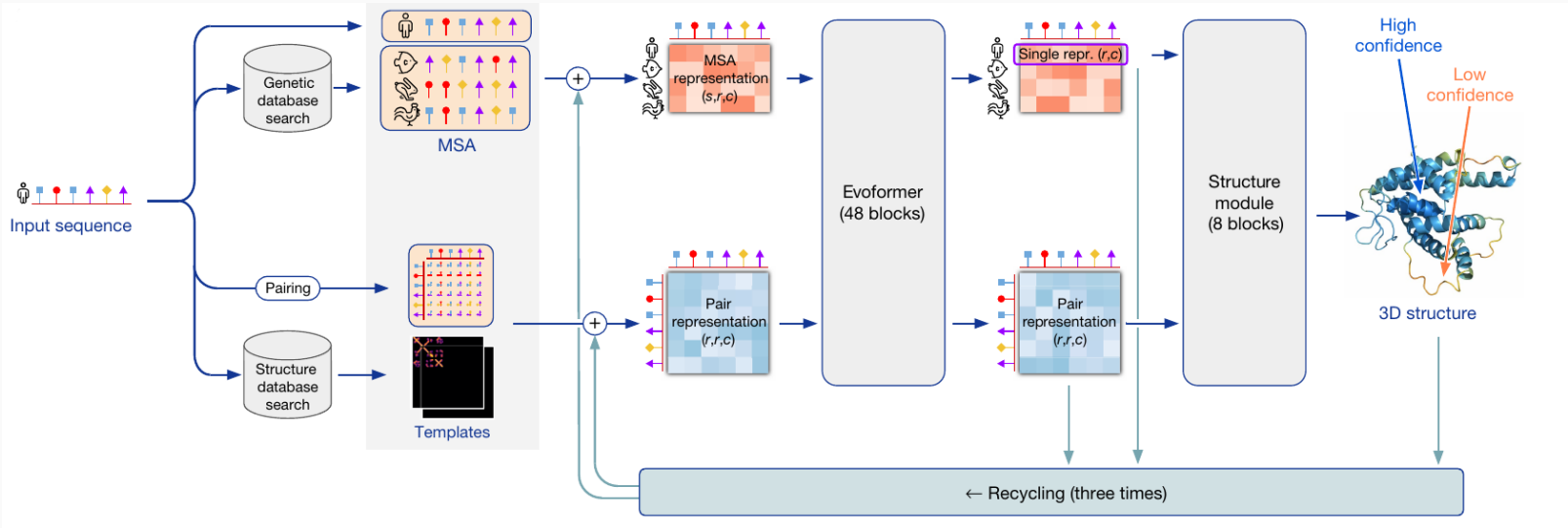

Figure 4: Image extracted from Senior et al. (2020).

Large dataset and training

Example-based synthesis

- Access to only one example.
- Modeling of the density:
  - Maximum entropy.
  - Feature matching and invariance.
- Estimation of parameters and sampling:
  - **S**tochastic **O**ptimization with **U**nadjusted **L**angevin.

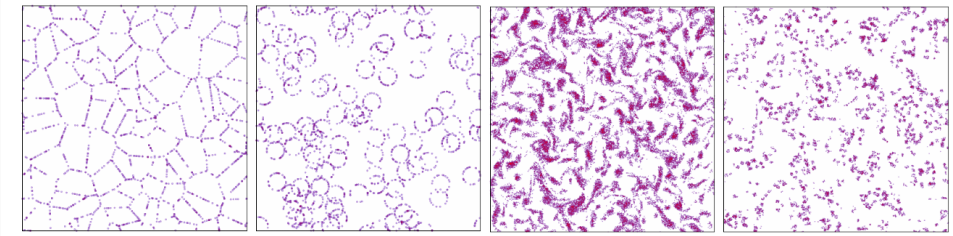

Figure 5: Image extracted from Brochard et al. (2020).

Generative modeling

- Access to many examples.
- Modeling of the density:
  - Energy-based models.
  - Normalizing flows.
  - Variational Autoencoders.
  - Generative Adversarial Networks.
- Training of a neural network.

Figure 6: Image extracted from Dhariwal and Nichol (2021).

Difference with statistical sampling

- **Generative Modeling**: sampling of target distribution π and we have access to $\hat{\pi} = (1/N) \sum_{k=1}^{N} \delta_{x^k}$, the **empirical distribution**.
- Different setting than **statistical sampling**.
  - Sampling from π with density (w.r.t. the Lebesgue measure on $\mathbb{R}^d$) proportional to $x \mapsto \exp[-U(x)]$.
  - $U : \mathbb{R}^d \to \mathbb{R}$ is called a **potential**.
  - Classical methods: **M**onte **C**arlo **M**arkov **C**hains Roberts et al. (1996); Durmus et al. (2017); Dalalyan (2017).
  - Applications in statistical physics and Bayesian statistics, see e.g. Neal (1992).
- Interaction between **statistical sampling** and **generative modeling**:
  - Modification of GAN losses to design efficient Markov kernels with given invariant measure (Song et al. (2017)).
  - Use of Metropolis-Hastings rejection and discriminator step to improve generative modeling with GAN (Turner et al. (2019))

Outline of the course

- **Goal of the course**:
  - Introduce modern methods of generative modeling.
  - Present their strengths and limitations.
- **Outline of the course**:
  - Energy-based models (EBMs).
  - Variational Autoencoders (VAEs).
  - Normalizing flows.
  - Generative Adversarial Networks (GANs).

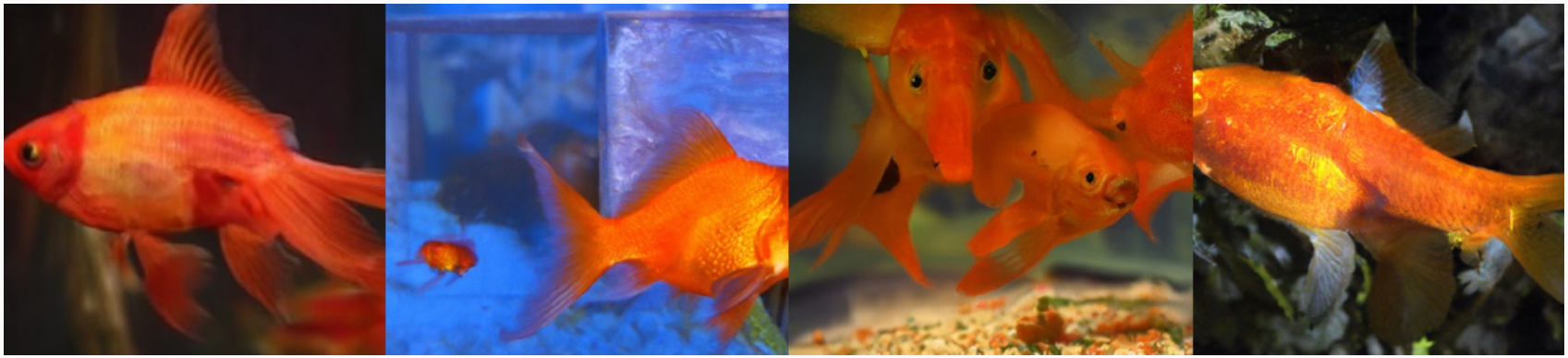

Figure 7: Image extracted from Dhariwal and Nichol (2021).

Energy-based models

Principles of EBMs

- Assume that π (the data distribution) is modelled by a **parametric distribution** π_θ such that for any $x \in \mathbb{R}^d$

$$p_\theta(x) = (\mathrm{d}\pi_\theta/\mathrm{dLeb})(x) = \exp[-f_\theta(x) + L(\theta)] \ ,$$
$$L(\theta) = -\log(\textstyle\int_{\mathbb{R}^d} \exp[-f_\theta(\tilde{x})]\mathrm{d}\tilde{x}) \ .$$

- f_θ is a **neural network** ($\theta \in \Theta$ is a set of parameters).
- Maximizing the **likelihood**

$$\boxed{\hat{\ell}(\theta) = \hat{\pi}[\log(p_\theta)] = -(1/N)\textstyle\sum_{k=1}^N f_\theta(x^k) + L(\theta) \ .}$$

- $\hat{\ell}$ is an **empirical** version of ℓ given by

$$\ell(\theta) = \pi[\log(p_\theta)] = -\mathrm{KL}(\pi|\pi_\theta) + \mathrm{H}(\pi) \ ,$$

  where H is the entropy of π.
- Maximizing the **likelihood** = Minimizing the **Kullback-Leibler** divergence.
- In what follows:
  - **Training EBMs**.
  - **Link with maximum entropy approaches**.

Training EBMs

Training EBMs

- Maximizing the **likelihood**

$$\boxed{\hat{\ell}(\theta) = \hat{\pi}[\log(p_\theta)] = -(1/N)\sum_{k=1}^{N} f_\theta(x^k) + L(\theta)\ .}$$

- Taking the gradient of the **log-partition**

$$\nabla_\theta L(\theta) = \int_{\mathbb{R}}^{d} \nabla_\theta f_\theta(\tilde{x}) \exp[-f_\theta(\tilde{x})] \mathrm{d}\tilde{x} / \int_{\mathbb{R}}^{d} \exp[-f_\theta(\tilde{x})] \mathrm{d}\tilde{x} = \pi_\theta[\nabla_\theta f_\theta]\ .$$

- Taking the gradient of the **empirical likelihood** $\hat{\ell}$

$$\boxed{\nabla_\theta \hat{\ell}(\theta) = -\hat{\pi}[\nabla_\theta f_\theta] + \pi_\theta[\nabla_\theta f_\theta]\ .}$$

- Taking the gradient of ℓ, $\nabla_\theta \hat{\ell}(\theta) = -\pi[\nabla_\theta f_\theta] + \pi_\theta[\nabla_\theta f_\theta]$:
  - $\hat{\ell}$ is the empirical version of ℓ.
  - At **equilibrium** $\theta^\star$, we cannot distinguish the expectation of $\nabla_\theta f_{\theta^\star}$ w.r.t. π and $\pi_{\theta^\star}$.
  - Approximating $\pi_\theta[\nabla_\theta f_\theta]$, requires **statistical sampling**.

MCMC methods for training

- Taking the gradient of the **empirical likelihood** $\hat{\ell}$

$$\boxed{\nabla_\theta \hat{\ell}(\theta) = -\hat{\pi}[\nabla_\theta f_\theta] + \pi_\theta[\nabla_\theta f_\theta] \ .}$$

- The loss $\hat{\ell}$ is called the **contrastive divergence**.
- Computing $\pi_\theta[\nabla_\theta f_\theta]$:
  - **Markov chains** targeting (approximately) π_θ.
  - **U**nadjusted **L**angevin **A**lgorithm

$$\boxed{X_{k+1} = X_k - \gamma \nabla_x f_\theta(X_k) + \sqrt{2\gamma} Z_{k+1} \ ,}$$

  - γ is a stepsize, $\nabla_x f_\theta$ is computed with backpropagation.
- In practice:
  - We add some **regularization** to the contrastive divergence.
  - We consider **short runs** of MCMC.
  - The initialization of the MCMC is important: **warm-start** (**persistent contrastive divergence**, see Tieleman (2008)) or not (see Nijkamp et al. (2019)).
  - Tutorial with Pytorch implementation based on Du and Mordatch (2019).

EBM training algorithm

Algorithm 1 Training of EBM

Input: n_{iter}, K, $\hat{\pi}$, N_{batch}, γ, δ, α, θ_0.
$B \neq \emptyset$.
for $n = 0$ to $n_{\text{iter}} - 1$ **do**
  Sample $X_n^{+,1:N_{\text{batch}}} = \{X_n^{+,k}\}_{k=1}^{N_{\text{batch}}}$ i.i.d. from $\hat{\pi}$.
  if B is not empty **then**
    Sample $X_n^{0,1:N_{\text{batch}}} = \{X_n^{0,k}\}_{k=1}^{N_{\text{batch}}}$ i.i.d. from $(1-\alpha)B + \alpha \mathrm{N}(0, \mathrm{Id})$.
  else
    Sample $X_n^{0,1:N_{\text{batch}}} = \{X_n^{0,k}\}_{k=1}^{N_{\text{batch}}}$ i.i.d. from $\mathrm{N}(0, \mathrm{Id})$.
  end if
  for $k = 0$ to $K - 1$ **do**
    $X_n^{k+1,1:N_{\text{batch}}} = X_n^{k,1:N_{\text{batch}}} + \gamma \nabla_x f_{\theta_n}(X_n^{k,1:N_{\text{batch}}}) + \sqrt{2\gamma} Z_n^{k+1,1:N_{\text{batch}}}$.
  end for
  $X_n^{-,1:N_{\text{batch}}} = X_n^{K,1:N_{\text{batch}}}$.
  $\theta_{n+1} = \theta_n + (\delta / N_{\text{batch}}) \sum_{\ell=1}^{N_{\text{batch}}} \{\nabla_\theta f_{\theta_n}(X_n^{+,\ell}) - \nabla_\theta f_{\theta_n}(X_n^{-,\ell})\}$.
  $B = X_n^{-,1:N_{\text{batch}}}$.
end for

Example-based synthesis

Link with example-based synthesis

- Different density models:
  - In **Energy-Based Models**: $p_\theta(x) = \exp[-f_\theta(x) + L(\theta)]$.
  - In **Maximum Entropy Models**: $p_\theta(x) = \exp[-\langle \theta, f(x) - f(x_0) \rangle + L(\theta)]$.
- Training losses:
  - In **Energy-Based Models**: $\nabla_\theta \hat{\ell}(\theta) = -\hat{\pi}[\nabla_\theta f_\theta] + \pi_\theta[\nabla_\theta f_\theta]$.
  - In **Maximum Entropy Models**: $\nabla_\theta L(\theta) = -\langle \theta, f(x_0) \rangle + \pi_\theta[\nabla_\theta f_\theta]$.
- Some key differences
  - $\hat{\pi}$ is replaced by δ_{x_0}. Only one example to train the model.
  - In EBMs we train a neural network, in Maximum Entropy Models the dependency w.r.t. the parameters is **linear**.
  - More **flexibility** in EBMs but no (trivial) maximum entropy interpretation.
- **Same sampling algorithm**.

Summary of EBMs

- **Advantages**:
  - Model the **potential directly**.
  - Usually allows for model with **less parameters** than VAE, GANs or NFs.
  - Compositionality via Product of Experts Hinton (2002).
- **Problems**:
  - **Training with MCMC is long**. This can be avoided if we replace the Kullback-Leibler objective with a Fisher objective (connection with score-matching Song and Kingma (2021)).
  - **Instabilities** with training Du and Mordatch (2019).
  - Density on $\mathbb{R}^d$. Usually the data is supported on a **low dimensional manifold** Arbel et al. (2020).
- **Links with other methods**:
  - Connection with GANs Che et al. (2020).
  - Connection with VAEs Xiao et al. (2020).
  - Connection with score-matching Song and Kingma (2021); Gao et al. (2020).

Variational Autoencoders

Differences with EBMs

- Basics of **EBMs**:
  - We consider an **exponential model** of the form $p_\theta(x) \propto \exp[-f_\theta(x)]$.
  - We maximize the **log-likelihood** $\ell(\theta) = \pi[\log p_\theta]$.
- In **Variational AutoEncoders** (VAEs):
  - We no longer consider an exponential model (pushforward type model).
  - We consider a **lower-bound** to the log-likelihood.
  - We introduce a **latent space**.

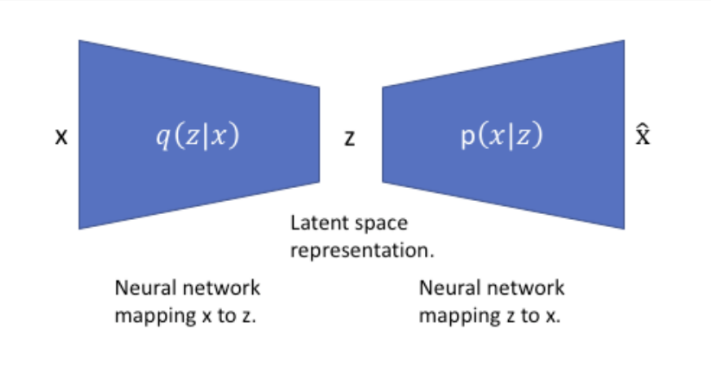

Figure 8: Structure of a VAE.

- In what follows:
  - Introduction of the **ELBO**.
  - **Encoding families** and **reparameterization trick**.
  - **Gaussian case** and **training**.

Introduction of the ELBO

From log-likelihood to ELBO (1/2)

- **Manifold hypothesis**: the data distribution is supported on a space (submanifold) of $\mathbb{R}^d$ with much **lower** dimension than $\mathbb{R}^d$.
  - We define a **joint model** on $\mathbb{R}^d \times \mathbb{R}^p$ and assume that $p_\theta(x) = \int_{\mathbb{R}^p} p(z) p_\theta(x|z) \mathrm{d}x$.
  - The distribution $p_\theta(x|z)$ **decodes** the **latent vector** z.
  - $p(z)$ is called the **prior** distribution (and does not depend on θ).
- A **marginalization** problem.

$$\boxed{\log(p_\theta(x)) = \log(\int_{\mathbb{R}^p} p(z) p_\theta(x|z) \mathrm{d}z) \ .}$$

- **Computing the gradient**.
  - $\nabla_\theta \log(p_\theta(x)) = \int_{\mathbb{R}^p} \nabla_\theta \log(p_\theta(x|z)) p_\theta(z|x) \mathrm{d}z$
  - MCMC techniques targeting the **posterior** $x \mapsto p_\theta(z|x)$.
  - This is similar to Maximum Entropy and Energy-Based Models.

From log-likelihood to ELBO (2/2)

- Instead of directly maximizing the **log-likelihood** we are going to consider a **lower-bound**.

$$\begin{aligned}
\log(p_\theta(x)) &= \log(\textstyle\int_{\mathbb{R}^p} p_\theta(x|z)p(z)\mathrm{d}z) \\
&= \log(\textstyle\int_{\mathbb{R}^p} p_\theta(x|z)(p(z)/q(z))q(z)\mathrm{d}z) \\
&\geq \textstyle\int_{\mathbb{R}^p} \log(p_\theta(x|z)p(z)/q(z))q(z)\mathrm{d}z \\
&\geq \textstyle\int_{\mathbb{R}^p} \log(p_\theta(x|z))q(z)\mathrm{d}z - \mathrm{KL}(q|p) \ .
\end{aligned}$$

- Inequality obtained using the concavity of the logarithm.
- This last lower-bound is called the **ELBO** (**E**vidence **L**ower **BO**und) (MacKay (1992)):
  - The first term controls the **reconstruction**.
  - The second term controls how close q is to the **prior**.
- The choice of the **variational distribution** q is crucial

Expectation-Maximization (EM) Algorithm

- Before presenting the **VAE** setting we recall the basics of the **Expectation-Maximization** (EM) algorithm.
- We begin with the same **ELBO**

$$\log(p_\theta(x)) \geq \int_{\mathbb{R}^p} \log(p_\theta(x|z))q(z)\mathrm{d}z - \mathrm{KL}(q|p) \ .$$

- We consider the **following procedure**:
  - Start with $\theta = \theta_0$ and choose $q = p_{\theta_0}(\cdot|x)$.
  - Compute

$$\begin{aligned}\mathcal{L}^0(\theta) &= \int_{\mathbb{R}^p} \log(p_\theta(x|z))p_{\theta_0}(z|x)\mathrm{d}z - \mathrm{KL}(p_{\theta_0}(\cdot|x)|p) \\ &= \int_{\mathbb{R}^p} \log(p_\theta(x,z))p_{\theta_0}(z|x)\mathrm{d}z + \mathrm{H}(p_{\theta_0}(\cdot|x)) \ .\end{aligned}$$

  - $\theta^1 = \arg\max\{\mathcal{L}^0(\theta) \ : \ \theta \in \Theta\}$.
  - Go back to the first step.
- Computing the **expectation might be hard** (useful for mixture of Gaussians models for instance).

Encoding families and reparameterization trick

Encoding variational family

- The **variational distribution** $z \mapsto p_\theta(z|x)$ is **optimal**.

$$\begin{aligned}\log(p_\theta(x)) - \mathcal{L}(\theta) &= \log(p_\theta(x)) - \int_{\mathbb{R}^d} \log(p_\theta(x,z)/q(z))q(z)\mathrm{d}z \\ &= -\int_{\mathbb{R}^d} \log(p_\theta(z|x)/q(z))q(z)\mathrm{d}z = \mathrm{KL}(p_\theta(\cdot|x)|q)\ .\end{aligned}$$

- Hence choosing the **posterior** closes the **variational gap**.
- Unfortunately the posterior can be **hard to compute**.
- In VAEs we consider a **variational family** of distribution $z \mapsto q_\phi(z|x)$ where:
  - ϕ is a parameter of q_ϕ (**parametric** family).
  - q_ϕ transforms a data point into a **latent representation**.
  - The ELBO can be written as follows

$$\boxed{\mathcal{L}(\theta,\phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|z))q_\phi(z|x)\mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p)\ .}$$

Computing the gradient

- The **ELBO** can be written as follows

$$\boxed{\mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p) \ .}$$

- The goal is to **optimize jointly** w.r.t. θ and ϕ.
  - Taking the gradient w.r.t. θ

$$\boxed{\nabla_\theta \mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \nabla_\theta \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z \ .}$$

  - Taking the gradient w.r.t. ϕ

$$\begin{aligned}\nabla_\phi \mathcal{L}(\theta, \phi) = &- \int_{\mathbb{R}^p} \nabla_\phi \log(q_\phi(z|x)) q_\phi(z|x) \mathrm{d}z \\ &+ \int_{\mathbb{R}^d} \log(p_\theta(x, z)/q_\phi(z|x)) \nabla_\phi \log(q_\phi(z|x)) q_\phi(z|x) \mathrm{d}z \ .\end{aligned}$$

  - Using **Monte Carlo** approximations we can approximate these integrals.

- With the **reparameterization trick** (Kingma and Welling (2013)) we will see a simpler way to compute these quantities.

Variational families for sampling

- Back to the original problem: **generative modeling**
  - We want to maximize the **likelihood** (Optional)
  - We want to **sample from the model** (Goal).
- Recall that $p_\theta(x) = \int_{\mathbb{R}^p} p_\theta(x|z)p(z)\mathrm{d}z$.
  - The **prior** distribution must be simple (Gaussian).
  - The **decoding** distribution must be simple (Gaussian).
- Hence, we consider the following parameterizations:
  - $p_\theta(x|z) = \mathrm{N}(m_\theta(z), \Sigma_\theta^{1/2}(z))$.
  - $q_\phi(z|x) = \mathrm{N}(m_\phi(x), \Sigma_\phi^{1/2}(x))$.
- **Sampling** from the model is easy.
  - Sample a Gaussian from $p(z) = \mathrm{N}(0, \mathrm{Id})$.
  - Sample a Gaussian from $p_\theta(x|z) = \mathrm{N}(m_\theta(z), \Sigma_\theta^{1/2}(z))$.

Reparameterization trick

- Recall the **ELBO**

$$\boxed{\mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p) \ .}$$

- The reparameterization trick (Kingma and Welling (2013)) consists into **decoupling the randomness and the parameters**.
  - Sampling from $Z \sim q_\phi(\cdot|x)$ obtained with $Z = g_\phi(x, \varepsilon)$ where $\varepsilon \sim q$.
  - q does not depend on any parameter θ or ϕ.
  - In the **Gaussian** setting $Z = m_\phi(z) + \Sigma_\phi^{1/2}(z)\varepsilon$ with $\varepsilon \sim \mathrm{N}(0, \mathrm{Id})$.
- We can rewrite the ELBO as follows

$$\begin{aligned}
\mathcal{L}(\theta, \phi) &= \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p) \\
&= \int_{\mathbb{R}^p} \log(p_\theta(x|g_\phi(x, \varepsilon))) q(\varepsilon) \mathrm{d}\varepsilon \\
&\quad - \int_{\mathbb{R}^p} \log(q_\phi(g_\phi(x, \varepsilon)|x) / p(g_\phi(x, \varepsilon))) q(\varepsilon) \mathrm{d}\varepsilon \ ,
\end{aligned}$$

  - **Change of variable** $z = g_\varphi(x, \varepsilon)$.
- We can **compute** the terms inside the integral and **differentiate** them w.r.t. θ and ϕ.
- Note that the integrals *do not* depend on the parameters.

Sticking the landing

- **Rewriting** the ELBO:

$$\begin{aligned}\mathcal{L}(\theta,\phi) &= \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x)\mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p) \\ &= \int_{\mathbb{R}^p} \{\log(p_\theta(z,x)) - \log q_\phi(z|x)\} q_\phi(z|x)\mathrm{d}z \\ &= \int_{\mathbb{R}^p} \{\log(p_\theta(g_\phi(x,\varepsilon),x)) - \log q_\phi(g_\phi(x,\varepsilon)|x)\} q(\varepsilon)\mathrm{d}\varepsilon \ .\end{aligned}$$

- Different **estimators of the gradient** of the ELBO Roeder et al. (2017).

$$\begin{aligned}\nabla_\phi \mathcal{L}(\theta,\phi) = \int_{\mathbb{R}^p} \nabla_z \{\log(p_\theta(g_\phi(x,\varepsilon),x)) - \log q_\phi(g_\phi(x,\varepsilon),x)\} \nabla_\phi g_\phi(x,\varepsilon) q(\varepsilon)\mathrm{d}\varepsilon \\ - \int_{\mathbb{R}^p} \nabla_\phi \log q_\phi(g_\phi(x,\varepsilon),x) q(\varepsilon)\mathrm{d}\varepsilon \ .\end{aligned}$$

- Note that $\int_{\mathbb{R}^p} \nabla_\phi \log q_\phi(g_\phi(x,\varepsilon)|x) q(\varepsilon)\mathrm{d}\varepsilon = 0$.
- $\{\varepsilon^k\}_{k=1}^N$ i.i.d. samples from q. Two **unbiased estimators** (and link with **control variates**):
  - **Path derivative** estimator

$$\boxed{\hat{\nabla}_\phi^{\mathrm{PD}} \mathcal{L}(\theta,\phi) = \sum_{k=1}^N \nabla_z \{\log(p_\theta(g_\phi(x,\varepsilon^k),x)) - \log q_\phi(g_\phi(x,\varepsilon^k)|x)\} \nabla_\phi g_\phi(x,\varepsilon^k) \ .}$$

  - **Total derivative** estimator

$$\boxed{\hat{\nabla}_\phi^{\mathrm{TD}} \mathcal{L}(\theta,\phi) = \hat{\nabla}_\phi^{\mathrm{TD}} \mathcal{L}(\theta,\phi) - \sum_{k=1}^N \nabla_\phi \log q_\phi(g_\phi(x,\varepsilon^k)|x) \ .}$$

Gaussian case and training

Interpretation in the Gaussian case

- The **ELBO** is given by

$$\mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|g_\phi(x, \varepsilon)))q(\varepsilon)\mathrm{d}\varepsilon \\ - \int_{\mathbb{R}^p} \log(q_\phi(g_\phi(x, \varepsilon)|x)/p(g_\phi(x, \varepsilon)))q(\varepsilon)\mathrm{d}\varepsilon \ .$$

- Recall that in practice, we restrict ourselves to the **Gaussian case**:
  - $p_\theta(x|z) = \mathrm{N}(m_\theta(z), \Sigma_\theta^{1/2}(z))$.
  - $q_\phi(z|x) = \mathrm{N}(m_\phi(x), \Sigma_\phi^{1/2}(x))$.
- For simplicity assume that $\Sigma_\theta^{1/2} = \Sigma_\phi^{1/2} = \mathrm{Id}$.

$$\mathcal{L}(\theta, \phi) = -(1/2)\int_{\mathbb{R}^p} \|x - m_\theta(m_\phi(x) + \varepsilon)\|^2 q(\varepsilon)\mathrm{d}\varepsilon \\ -(1/2)\int_{\mathbb{R}^p} \|m_\phi(x) + \varepsilon\|^2 q(\varepsilon)\mathrm{d}\varepsilon + C \ .$$

  - C is a constant independent of the parameters .
  - The first term is the **reconstruction loss**.
  - The second term is the **regularization** term.

Influence of loss terms

- **MNIST** reconstruction with VAE (10 digits give 10 classes).
  - Minimizing only the reconstruction loss does not yield meaningful **interpolation** (sampling is hard).
  - Minimizing only the regularization loss does not yield meaningful **encoding**.
- The latent space is **two dimensional** here.

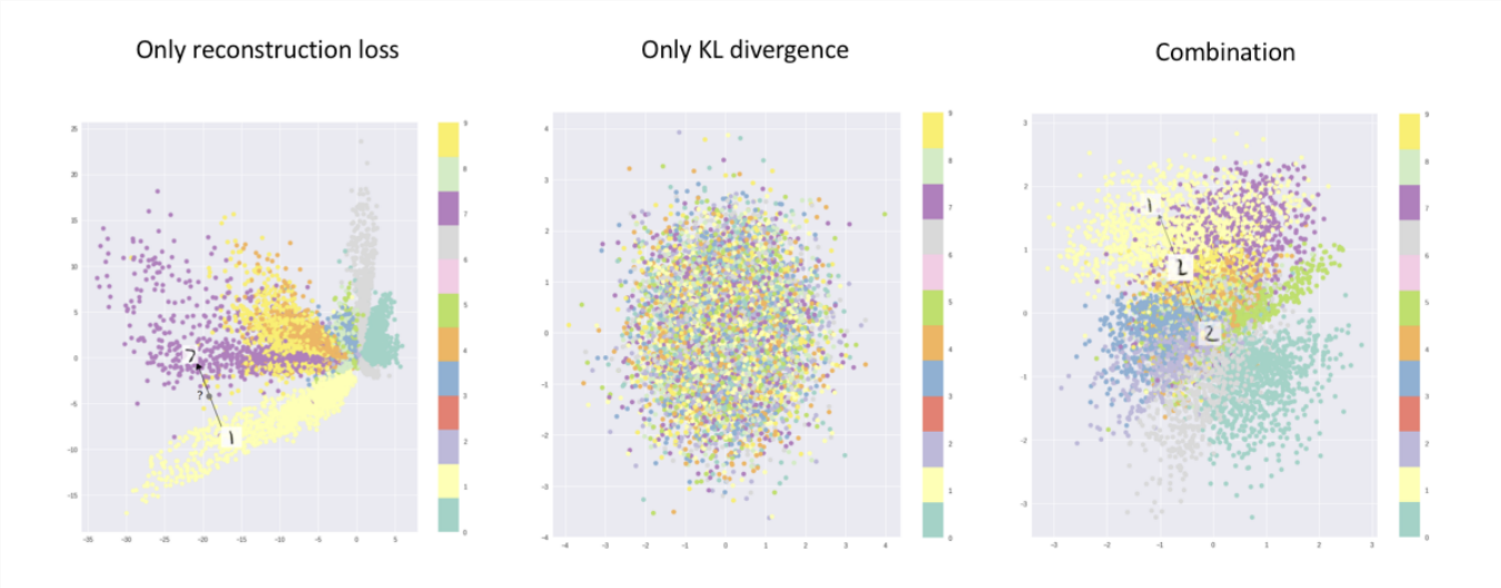

Figure 9: Image extracted from an online tutorial.

Interpolation in the latent space

- Contrary to EBMs the sampling is **explicit**.
- By travelling in the latent space we can **interpolate** in the dataset in a "meaningful" manner.

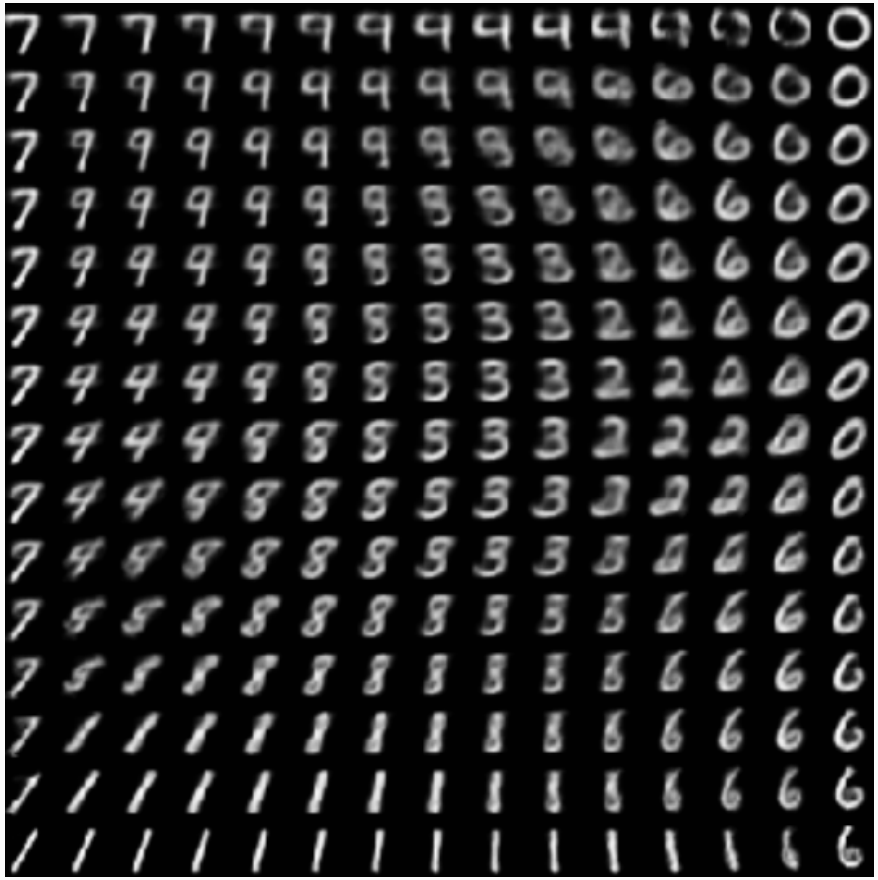

Figure 10: Image extracted from an online tutorial.

VAE training algorithm

Algorithm 2 Training of VAE

Input: n_{iter}, K, $\hat{\pi}$, q, estimator of $\nabla\mathcal{L}$, N_{batch}, δ_θ, δ_ϕ θ_0, ϕ_0.
for $n = 0$ to $n_{\text{iter}} - 1$ **do**
  Sample $X_n^{1:N_{\text{batch}}} = \{X_n^k\}_{k=1}^{N_{\text{batch}}}$ i.i.d. from $\hat{\pi}$.
  $Z_n^{1:N_{\text{batch}}} = \{Z_n^k\}_{k=1}^{N_{\text{batch}}}$ i.i.d. from q.
  Compute estimator of the gradient of the ELBO w.r.t. θ, $\hat{\nabla}_\theta\mathcal{L}(\theta_n, \phi_n)$
  Compute estimator of the gradient of the ELBO w.r.t. ϕ, $\hat{\nabla}_\phi\mathcal{L}(\theta_n, \phi_n)$
  $\theta_{n+1} = \theta_n + \delta_\theta\hat{\nabla}_\theta\mathcal{L}(\theta_n, \phi_n)$
  $\phi_{n+1} = \phi_n + \delta_\phi\hat{\nabla}_\phi\mathcal{L}(\theta_n, \phi_n)$
end for

- Different choices of **estimators** (path derivative, total derivative).
- **Stochastic gradient descent** can be replaced by other algorithms (such as ADAM Kingma and Ba (2014)).

Summary of VAEs

- **Vanilla autoencoders** consist in the reconstruction loss only.
- **Variational autoencoders** are better generative models.
- **Advantages**:
  - VAEs are easy to train with clear estimators of the ELBO.
  - They provide interesting **latent representations**.
- **Problems**:
  - VAEs with Gaussian priors are **not competitive** in generative modeling.
  - The choice of the **latent space dimension** is arbitrary.
  - The choice of the **prior** is arbitrary.
- **Links with other methods**
  - VAE can be combined with normalizing flows Kingma et al. (2016); Vahdat and Kautz (2020).
  - Score-based generative models can be seen as autoencoders Huang et al. (2021); Dieleman (2022); Ho et al. (2020).

Normalizing flows

Principles of normalizing flows

- **Normalizing flows** can be seen as **reparameterization trick**.
- We still aim at maximizing the likelihood $\log(p_\theta(x))$, where p_θ is our model.
  - Model p_θ flexible enough to approximate the **data distribution**.
  - **Sampling** from p_θ must be easy.
- The principles of normalizing flows:
  - Start from a distribution π_0 with density p which is easy to sample.
  - Define $\pi_\theta = (g_\theta)_\# \pi_0$ and its density p_θ.
  - Maximize the **log-likelihood** $\log(p_\theta(x))$.

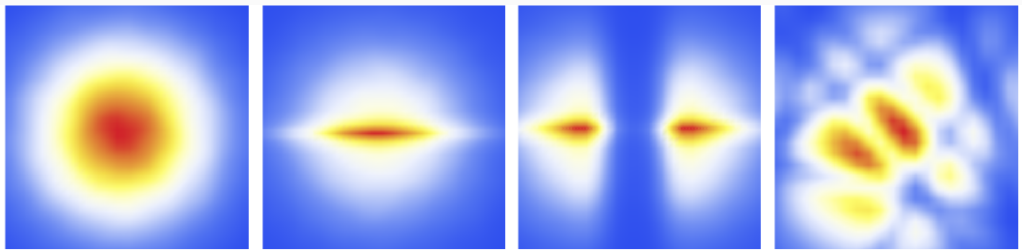

Figure 11: Several transformations of a $\mathrm{N}(0, \mathrm{Id})$ density. Image extracted from Rezende and Mohamed (2015).

- In what follows:
  - First normalizing flows and **GLOW**.
  - **Autoregressive models** and **IAFVAE**.
  - **Continuous Normalizing Flows**.

First normalizing flows and GLOW

Invertible transformations

- The density of $(g_\theta)_\# \pi_0$ is given by a **change of variable**.
  - We assume that g_θ is a **diffeomorphism** (not necessary, one can use the co-area/area formula Caterini et al. (2021)).
  - Using the d-dimensional **change of variable** we have for any $f \in \mathrm{C}_c(\mathbb{R}^d, \mathbb{R})$

$$\mathbb{E}[f(X)] = \mathbb{E}[f(g_\theta(Z))] = \int_{\mathbb{R}^d} f(x) p(g_\theta^{-1}(x)) |\mathcal{J}_\theta(g_\theta^{-1}(x))| \mathrm{d}z \ .$$

  - Hence, maximizing the **log-likelihood** is equivalent to maximizing

$$\boxed{\ell(\theta) = \log(p(g_\theta^{-1}(x))) + \log(|\mathcal{J}_\theta(g_\theta^{-1}(x))|) \ .}$$

- **Composition of transformation**: $g_\theta = g_\theta^0 \circ g_\theta^1 \circ \cdots \circ g_\theta^K$.
- Conditions on the transformations:
  - g_θ and g_θ^{-1} are easy to compute and differentiate.
  - The Jacobian $\mathcal{J}_\theta$ is easy to compute and differentiate.

Different types of flows

- In Rezende and Mohamed (2015) planar and radial flows are presented.
- Two other very efficient flows Dinh et al. (2016, 2014):
  - **Affine coupling layer**.
  - **Invertible 1x1 convolution**.
- How does the **affine coupling layer** work?
  - We split $x \in \mathbb{R}^d$ in $x = (x_0, x_1)$ with $x_0 \in \mathbb{R}^{d_0}$, $x_1 \in \mathbb{R}^{d_1}$.
  - **Forward** transform $g_\theta(x) = (x_0, \exp[s_\theta(x_0)] \odot x_1 + t_\theta(x_0))$.
  - **Reverse** transform $g_\theta^{-1}(x) = (x_0, (x_1 - t_\theta(x_0)) \oslash \exp[s_\theta(x_0)])$.
  - **Log-Jacobian**: $\log(|\mathcal{J}_\theta(x)|) = \sum_{i=1}^{d_1} s_\theta(x_0)_i$.
- How does the **invertible 1x1 convolution** work?
  - Matrix $\mathbf{W}_\theta \in \mathbb{R}^{C \times C}$ (number of channels), $x \in \mathbb{R}^{H \times W \times C}$.
  - **Forward** transform $g_\theta(x)_{i,j} = \mathbf{W}_\theta x_{i,j}$.
  - **Reverse** transform $g_\theta^{-1}(x)_{i,j} = \mathbf{W}_\theta^{-1} x_{i,j}$.
  - **Log-Jacobian** $\log(|\mathcal{J}_\theta(x)|) = H \times W \times \log(|\mathbf{W}_\theta|)$.

Generative Flow (GLOW)

- Results obtained by Kingma and Dhariwal (2018).
- Combining actnorm, invertible convolution and affine coupling layers (multiple times).
- The "actnorm" layer is simply an **affine layer**.
- **High quality results** and **interpolation**.

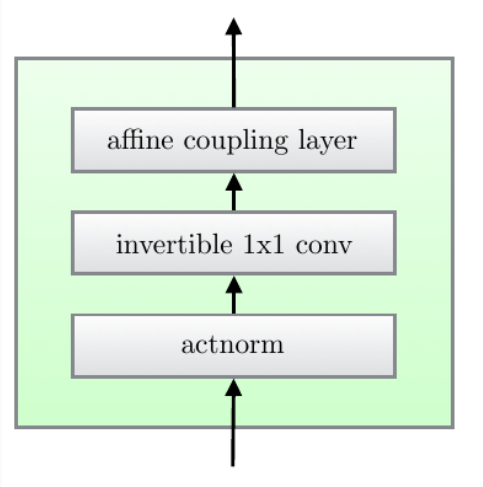

Figure 12: One step of GLOW. Image extracted from Kingma and Dhariwal (2018).

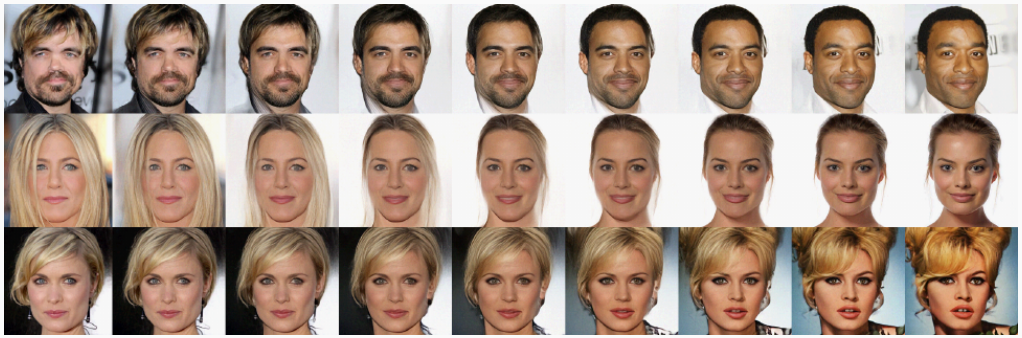

Figure 13: GLOW results. Image extracted from Kingma and Dhariwal (2018).

Autoregressive models and IAFVAE

A detour by autoregressive models

- Another **generative modeling** approach: **autoregressive models**
  - **M**asked **A**utoencoder for **D**istribution **E**stimation (autoregressive autoencoder), Germain et al. (2015).
  - **PixelRNN** (autoregressive LSTM), Van Oord et al. (2016).
  - Both models are trained by maximizing the **log-likelihood**.

- Both models assume the following **raster-scan** decomposition.

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i | x_{1:i-1}) \ .$$

- **Problems**:
  - As many predicitions as the **dimension**.
  - Can be parallelized for **training but not for sampling**.

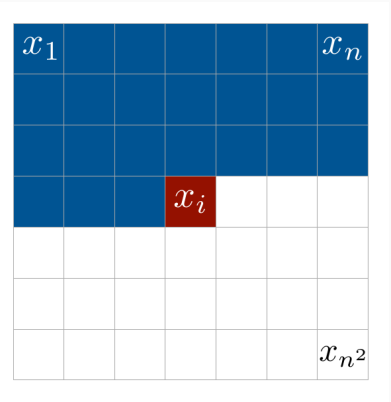

Figure 14: Raster scan order. Image extracted from Van Oord et al. (2016).

The autoregressive layer

- These ideas can however be reapplied to define a **normalizing flow layer**.
- Kingma et al. (2016) introduces the **autoregressive layer**:
  - $x = \{x_i\}_{i=1}^d$
  - $\sigma_\theta^i(x_{1:i-1}) = \text{sigmoid}(s_\theta^i(x_{1:i-1}))$
  - **Forward** transform $g_\theta(x)_i = \sigma_\theta^i(x_{1:i-1})x_i + (1 - \sigma_\theta^i(x_{1:i-1}))t_\theta^i(x_{1:i-1})$.
  - **Reverse** transform $g_\theta^{-1}(x)_i = (x_i - (1 - \sigma_\theta^i(x_{1:i-1}))t_\theta^i(x_{1:i-1}))/\sigma_\theta^i(x_{1:i-1})$.
  - **Log-Jacobian** $\sum_{i=1}^d \log(\sigma_\theta^i(x_{1:i-1}))$.
- The Jacobian is **triangular** (easy computation of the determinant).
- Parameterization with the sigmoid is numerically stable (inspired by LSTM Hochreiter and Schmidhuber (1997)).
- Between each autoregressive layer the ordering is **reversed**.
- More involved autoregressive models in practice:
  - **Masked autoencoders** Germain et al. (2015).
  - **Convolutional** autoregressive models Van Oord et al. (2016).

Inverse Autoregressive Flow VAE

- **Problem**: defining a flow is not enough to obtain flexible generative model.
  - Kingma et al. (2016) uses a VAE and define a **normalizing flow prior**.
  - The model is called **I**nverse **A**utoregressive **F**low VAE.
- The "only" change compared to a classical VAE is the definition of the prior:
  - **Gaussian** assumption in classical VAE.
  - **Normalizing flows** in IAFVAE.
- The training is still done by maximizing the **ELBO**. This is possible because one can compute $\log(q(z|x))$ when parameterized with **normalizing flows**.

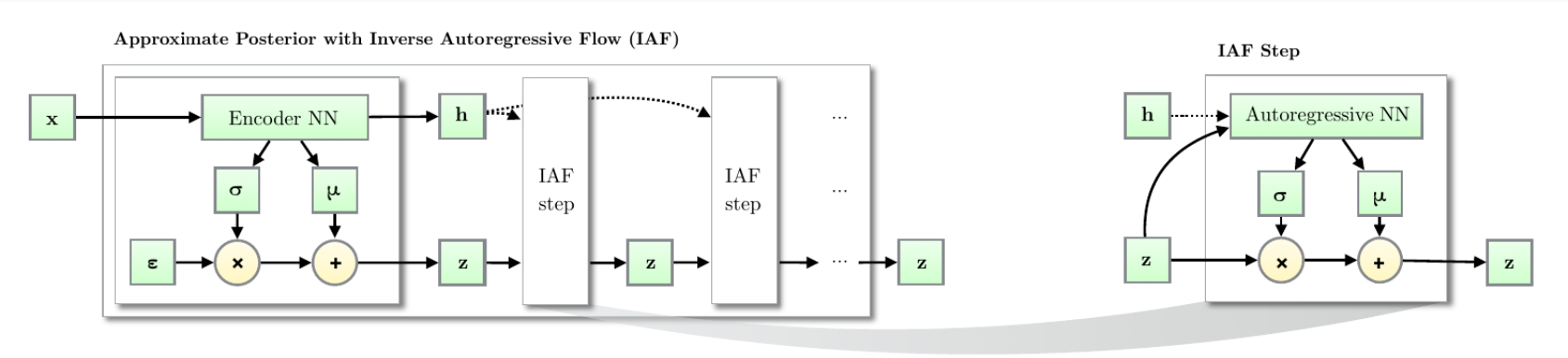

Figure 15: IAF worflow. Image extracted from Kingma et al. (2016).

Rewriting the ELBO

- We recall that the **ELBO** is given by

$$\boxed{\mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p) \ .}$$

- Usually, $p(z)$ is a **Gaussian prior**.
- More complicated prior p_Ψ (**parametric form**).

$$\boxed{\mathcal{L}(\theta, \phi) = \int_{\mathbb{R}^p} \log(p_\theta(x|z)) q_\phi(z|x) \mathrm{d}z - \mathrm{KL}(q_\phi(\cdot|x)|p_\Psi) \ .}$$

- Some interesting cases:
  - **Cascade of Gaussian models** (as in Sønderby et al. (2016)).
  - **Normalizing flow** (as in IAF-VAE Kingma et al. (2016); Chen et al. (2016)).
  - **Diffusion model** (as in Vahdat et al. (2021); Wehenkel and Louppe (2021)).
  - In the case of diffusion model, we need to derive another ELBO.

Nouveau VAE

- Vahdat and Kautz (2020) implements an improved version of IAF-VAE.
  - Introduce new architecture for the neural networks.
  - Obtain competitive results (**VAE + normalizing flow** in latent space).
  - The model is called **N**ouveau **VAE**.
- Note that this idea can be extended to **diffusion models** Wehenkel and Louppe (2021); Vahdat et al. (2021) with state-of-the-art results.

Figure 16: NVAE results. Image extracted from Vahdat and Kautz (2020).

Continuous normalizing flows

Continuous Normalizing Flows

- One **problem** of **normalizing flows**: the set of valid transformation is restricted by the **tractability of the log-Jacobian**.

- Moving from the **discrete** time setting to the **continuous** time setting allows **greater flexibility** (Chen et al. (2018); Grathwohl et al. (2018)).
  - Normalizing flow: $\mathcal{O}(d^3)$ computation.
  - **C**ontinuous **N**ormalizing **F**low (CNF): $\mathcal{O}(d^2)$ computation Chen et al. (2018).
  - CNF with trace estimator: $\mathcal{O}(d)$ computation Grathwohl et al. (2018).

- We introduce a **continuous** evolution:
  - A (stochastic) **dynamics** $\mathrm{d}\mathbf{X}_t = b(t, \mathbf{X}_t)\mathrm{d}t + \sigma(t, \mathbf{X}_t)\mathrm{d}\mathbf{B}_t$ with $\mathbf{X}_0 \sim \pi_0$.
  - $(\mathbf{B}_t)_{t\geq 0}$ is a d-dimensional Brownian motion.
  - $b : \mathbb{R}_+ \times \mathbb{R}^d \to \mathbb{R}^d$, $\sigma : \mathbb{R}_+ \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ smooth, π_0 has density p_0.
  - What is the evolution of $t \mapsto \log(p(\mathbf{X}_t))$?

- This is equivalent to a continuous **change of variable**.

Fokker-Planck derivation (1/3)

- We make the following assumptions (can be relaxed):
  - We assume that for any $t \geq 0$, $\mathcal{L}(\mathbf{X}_t)$ admits a **smooth density** positive p_t w.r.t. the Lebesgue measure such that $(t, x) \mapsto p_t(x) \in \mathrm{C}^\infty(\mathbb{R}_+^\star \times \mathbb{R}^d, \mathbb{R}_+)$.
  - We assume that for any $t \geq 0$, there exists $C_t \geq 0$ such that for any $x \in \mathbb{R}^d$

$$\|b(t, x)\| + \|\sigma(t, x)\| \leq C_t(1 + \|x\|) \ .$$

  - Under the previous assumption, we have for any $t \geq 0$ and $p \in \mathbb{N}$, $\mathbb{E}[\sup_{s \in [0,t]} \|\mathbf{X}_s\|^p] < +\infty$ (**Grönwall lemma**).
- Let $f \in \mathrm{C}_c(\mathbb{R}^d)$ and apply the **Itô formula** to $(f(\mathbf{X}_t))_{t \geq 0}$. For any $s, t \geq 0$

$$\boxed{f(\mathbf{X}_t) - f(\mathbf{X}_s) = \int_s^t \{\langle b(u, \mathbf{X}_u), \nabla f(\mathbf{X}_u)\rangle + (1/2)\langle \Sigma(u, \mathbf{X}_u), \nabla^2 f(\mathbf{X}_u)\rangle\} \mathrm{d}u + \mathbf{M}_t^f - \mathbf{M}_s^f \ .}$$

  - $\Sigma = \sigma\sigma^\top$.
  - The second scalar product is associated with **Frobenius norm**.
  - $(\mathbf{M}_t^f)_{t \geq 0}$ is a square integrable martingale.

Fokker-Planck derivation (2/3)

- Recall the **Itô formula**

$$f(\mathbf{X}_t) - f(\mathbf{X}_s) = \int_s^t \{\langle b(u, \mathbf{X}_u), \nabla f(\mathbf{X}_u)\rangle + (1/2)\langle \Sigma(u, \mathbf{X}_u), \nabla^2 f(\mathbf{X}_u)\rangle\}\mathrm{d}u + \mathbf{M}_t^f - \mathbf{M}_s^f \ .$$

- Taking the expectation.
  - $\mathbb{E}[f(\mathbf{X}_t) - f(\mathbf{X}_s)] = \int_{\mathbb{R}^d} f(x)\{p_t(x) - p_s(x)\}\mathrm{d}x = \int_{\mathbb{R}^d} f(x)(\int_s^t \partial_u p_u(x))\mathrm{d}x$.
  - Using the **divergence theorem** we have

$$\begin{aligned}\mathbb{E}[\langle b(u, \mathbf{X}_u), \nabla f(\mathbf{X}_u)\rangle] &= \int_{\mathbb{R}^d} \langle b(u, x)p_u(x), \nabla f(x)\rangle \mathrm{d}x \\ &= -\int_{\mathbb{R}^d} \mathrm{div}(b(u, \cdot)p_u)(x) f(x)\mathrm{d}x \ .\end{aligned}$$

  - Similarly, $\mathbb{E}[\langle \Sigma(u, \mathbf{X}_u), \nabla^2 f(\mathbf{X}_u)\rangle] = \int_{\mathbb{R}^d} \sum_{i,j=1}^d \partial_{i,j}\{\Sigma_{i,j}(u, \cdot)p_u\} f(x)\mathrm{d}x$.
- Dividing by $(t - s)$, letting $s \to t$ and using the **dominated convergence theorem** we get that for any $f \in \mathrm{C}_c(\mathbb{R}^d)$ and $t > 0$

$$\int_{\mathbb{R}^d} f(x)[-\partial_t p_t(x) - \mathrm{div}(b(t, \cdot)p_t)(x) + (1/2)\sum_{i,j=1}^d \partial_{i,j}\{\Sigma_{i,j}(t, \cdot)p_t\}]\mathrm{d}x = 0 \ .$$

- Choosing an **approximation of the unity** and using the smoothness of b, Σ, p we get that for any $t > 0$ and $x \in \mathbb{R}^d$

$$\partial_t p_t(x) = -\mathrm{div}(b(t, \cdot)p_t)(x) + (1/2)\sum_{i,j=1}^d \partial_{i,j}\{\Sigma_{i,j}(t, \cdot)p_t\}(x) \ .$$

Fokker-Planck derivation (3/3)

- We obtain the **Fokker-Planck** equation

$$\boxed{\partial_t p_t(x) = -\mathrm{div}(b(t,\cdot)p_t)(x) + (1/2)\sum_{i,j=1}^{d} \partial_{i,j}\{\Sigma_{i,j}(t,\cdot)p_t\}(x)\ .}$$

- This equation describes the **evolution of the density**.
- Some special cases:
  - Case $\sigma = 0$ (**deterministic dynamics**)

$$\boxed{\partial_t p_t(x) = -\mathrm{div}(b(t,\cdot)p_t)(x)\ .}$$

  - Case $\sigma = c^{1/2}\,\mathrm{Id}$ $(c > 0)$ (**Langevin dynamics**)

$$\boxed{\partial_t p_t(x) = -\mathrm{div}(b(t,\cdot)p_t)(x) + (c/2)\Delta p_t(x) = -\mathrm{div}(\{b(t,\cdot) - (c/2)\nabla \log p_t\}p_t)(x)\ .}$$

- As a consequence the two following dynamics have the **same** marginal densities.
  - $\mathrm{d}\mathbf{X}_t = b(t,\mathbf{X}_t)\mathrm{d}t + c^{1/2}\mathrm{d}\mathbf{B}_t$
  - $\mathrm{d}\mathbf{X}_t = \{b(t,\mathbf{X}_t) - (c/2)\nabla \log p_t(\mathbf{X}_t)\}\mathrm{d}t.$
  - One is **deterministic**, the other is **stochastic** (we will come back to this).

Evolution of the log-density

- In CNF we consider a **deterministic dynamics** $\mathrm{d}\mathbf{X}_t = b(t, \mathbf{X}_t)\mathrm{d}t$.
- **Fokker-Planck** equation: for any $t > 0$ and $x \in \mathbb{R}^d$,
  $\partial_t p_t(x) = -\mathrm{div}(b(t, \cdot)p_t)$.
- Differentiating the logarithm: for any $t > 0$ and $x \in \mathbb{R}^d$

$$\partial_t \log(p_t)(x) = -\langle b(t, x), \nabla \log p_t(x)\rangle - \mathrm{div}(b(t, x)) \ .$$

- Differentiating the **evolution** logarithm: for any $t > 0$

$$\begin{aligned}\partial_t \log p_t(\mathbf{X}_t) &= -\langle b(t, \mathbf{X}_t), \nabla \log p_t(\mathbf{X}_t)\rangle - \mathrm{div}(b(t, \mathbf{X}_t)) + \langle b(t, \mathbf{X}_t), \nabla \log p_t(\mathbf{X}_t)\rangle \\ &= -\mathrm{div}(b(t, \mathbf{X}_t)) \ .\end{aligned}$$

- As a result we have for any $t \geq 0$

$$\boxed{\log(p_t(\mathbf{X}_t)) - \log(p_0(\mathbf{X}_0)) = -\int_0^t \mathrm{div}(b(s, \mathbf{X}_s))\mathrm{d}s \ .}$$

  - The evolution of $\partial_t \log p_t(\mathbf{X}_t)$ is an **ODE**.
  - In the **discrete** case we compute a **log-Jacobian** ($\mathcal{O}(d^3)$).
  - In the **continuous** case we compute a **divergence** ($\mathcal{O}(d^2)$).

Back to the log-likelihood

- In practice the **drift** b depends on a parameter θ (neural network b_θ).
- The model is given by p_T (easy-to-sample density p_0 and $T > 0$).
- We want to optimize the **log-likelihood**

$$\boxed{\mathbb{E}[\log(p_T(\mathbf{X}_T))] = \mathbb{E}[\log(p_0(\mathbf{X}_0)) - \int_0^T \operatorname{div}(b_\theta(s, \mathbf{X}_s))\mathrm{d}s] \ .}$$

  - p_0 is often chosen to be **Gaussian**.
  - $\mathbf{X}_0$ is initialized with $\mathbf{Y}_T$ where $\mathbf{Y}_0 \sim \pi$ (the data distribution) and $\mathrm{d}\mathbf{Y}_t = -b_\theta(T - t, \mathbf{Y}_t)\mathrm{d}t$.
  - As a result $\mathcal{L}(\mathbf{X}_T) = \mathcal{L}(\mathbf{Y}_0) = \pi$ and $\mathbb{E}[\log(p_T(\mathbf{X}_T))]$ is the **log-likelihood** of the model.
- How to **train** the model?
  - How to **backpropagate** through an ODE?
  - Methods from **control theory**.
  - Efficient computation.
  - Review on Neural ODEs Kidger (2022) (discretize-then-optimize or optimize-then-discretize).

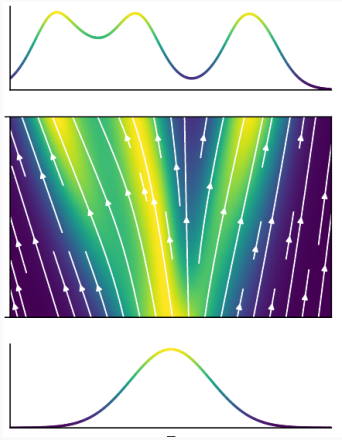

Figure 17: Evolution of the density. Image extracted from Grathwohl et al. (2018).

Basics of optimal control (1/2)

- We recall the method used in Chen et al. (2018) which itself is derived from Pontryagin (1987).
- Assume we want to **minimize** $\mathcal{J}(\theta) = \Phi(z_T) + \int_0^T h(\theta, t, z_t)\mathrm{d}t$ under the condition that $\mathrm{d}z_t = f(\theta, t, z_t)\mathrm{d}t$ and $z_0 = z_{\text{init}} \in \mathbb{R}^d$ is fixed.
  - We assume enough regularity/boundedness on h and f.
  - Everything is **deterministic** here.
- We introduce the **Lagrangian**

$$\boxed{L(\theta, z, \lambda) = \Phi(z_T) + \int_0^T \{h(\theta, t, z_t) + \lambda_t(\dot{z}_t - f(\theta, t, z_t))\}\mathrm{d}t \ .}$$

- Note that $\sup_{z,\lambda} L(\theta, \cdot) = \mathcal{J}(\theta)$.
- Hence, to minimize $\mathcal{J}$ we can consider the following **iterative scheme**:
  - Start with $\theta^0 \in \Theta$.
  - Find z^0, λ^0 such that $\sup_{z,\lambda} L(\theta^0, \cdot) = L(z^0, \lambda^0, \theta^0)$.
  - Let $\theta^1 = \theta^1 - \gamma \nabla_\theta L(z^0, \lambda^0, \theta^0)$.
  - Go back to the first step with $\theta^0 \leftarrow \theta^1$.

Basics of optimal control (2/2)

- We need to find $z^\star, \lambda^\star$ such that $\sup_{z,\lambda} L(\theta, \cdot) = L(z^\star, \lambda^\star, \theta)$.
- Recall that the **Lagrangian** is given by

$$\begin{aligned} L(\theta, z, \lambda) &= \Phi(z_T) + \int_0^T \{h(\theta, t, z_t) + \lambda_t(\dot{z}_t - f(\theta, t, z_t))\}\mathrm{d}t \\ &= \Phi(z_T) + \int_0^T \mathcal{L}_t(\theta, t, \lambda_t, x_t, \dot{x}_t)\mathrm{d}t \ . \end{aligned}$$

- The optimality is given by **Euler-Lagrange** conditions ($u = z$ or λ)

$$\boxed{\partial_{u_t}\mathcal{L}_t(\theta, t, \lambda_t, x_t, \dot{x}_t) - \partial_t\partial_{\dot{u}_t}\mathcal{L}_t(\theta, t, \lambda_t, x_t, \dot{x}_t) = 0 \ .}$$

  - $u = \lambda$ gives $\mathrm{d}z_t^\star = f(\theta, t, z_t^\star)\mathrm{d}t$.
  - $u = z$ gives $\mathrm{d}\lambda_t^\star = -\partial_{z_t} h(\theta, t, z_t^\star)\mathrm{d}t + \lambda_t^\star \partial_{z_t} f(\theta, t, z_t^\star)$.
  - λ has **terminal condition** $\lambda_T^\star = \partial_{z_T}\Phi(z_T^\star)$.
- The last equation is the **adjoint state** evolution equation.
- Computing the **gradient** w.r.t. θ

$$\boxed{\nabla_\theta L(z^\star, \lambda^\star, \theta) = \int_0^T \nabla_\theta h(\theta, t, z_t^\star) - \lambda_t^\star \nabla_\theta f(\theta, t, z_t^\star) \ .}$$

  - We can solve an ODE to compute the gradient.
  - **Continuous** equivalent to the **backpropagation**.

Back to the training of normalizing flows

- Recall that in **optimal control** we minimize $\mathcal{J}(\theta) = \Phi(z_T) + \int_0^T h(\theta, t, z_t)\mathrm{d}t$ with $\mathrm{d}z_t = f(\theta, t, z_t)\mathrm{d}t$ and $z_0 = z_{\text{init}} \in \mathbb{R}^d$.
- In **CNF**, we want to optimize the **log-likelihood**

$$\begin{aligned}\mathbb{E}[\log(p_T(\mathbf{X}_T))] &= \mathbb{E}[\log(p_0(\mathbf{X}_0)) - \int_0^T \mathrm{div}(b_\theta)(s, \mathbf{X}_s)\mathrm{d}s] \\ &= \mathbb{E}[\log(p_0(\mathbf{Y}_T)) - \int_0^T \mathrm{div}(b_\theta)(T-s, \mathbf{Y}_s)\mathrm{d}s] \ .\end{aligned}$$

- For a given sample $\mathbf{Y}_0$, we can define:
  - $z_t = \mathbf{Y}_t$ with $z_{\text{init}} = \mathbf{Y}_0$
  - $f(\theta, t, z) = -b_\theta(T-t, z)$.
  - $h(\theta, t, z) = -\mathrm{div}(b_\theta)(T-t, z)$.
- Therefore, we can apply the previous **optimization scheme** with an **amortization** w.r.t. $\mathbf{Y}_0$.
  - Amortization means that at each optimization step we sample $\mathbf{Y}_0 \sim \pi$.

The CNF method

- We optimize the **log-likelihood**

$$\boxed{\mathbb{E}[\log(p_T(\mathbf{X}_T))] = \mathbb{E}[\log(p_0(\mathbf{X}_0)) - \int_0^T \operatorname{div}(b_\theta(s, \mathbf{X}_s))\mathrm{d}s] \ .}$$

- To do so we use the **amortized adjoint method**.
- The loss is compared with the one obtained in the **discrete-time** setting.

$$\boxed{\mathbb{E}[\log(p(\mathbf{X}_T))] = \mathbb{E}[\log(p_0(g_\theta^{-1}(\mathbf{X}_T))) + \log(|\mathcal{J}_\theta(g_\theta^{-1}(\mathbf{X}_T))|)] \ .}$$

  - Usually $g_\theta = g_\theta^1 \circ \cdots \circ g_\theta^L$.
  - In continuous-time the equivalent of the **autoregressive layer** is **volume preserving** (the divergence term is zero). We say that the flow is **Hamiltonian**.

- Evaluation log-Jacobian $\mathcal{O}(d^3)$.
- Evaluation divergence $\mathcal{O}(d^2)$.
- Hutchinson estimator $\mathcal{O}(d)$
- The last method is called **F**ree-**F**orm **J**acobian **O**f **R**eversible **D**ynamics.

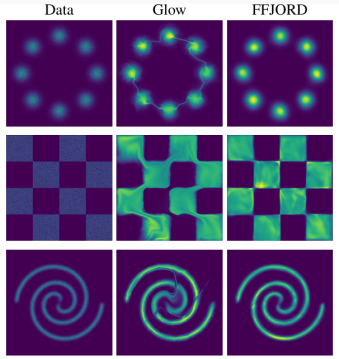

Figure 18: Comparing Glow and FFJORD. Image extracted from Grathwohl et al. (2018).

Summary of Normalizing Flows

- **Advantages**:
  - Normalizing Flows are **flexible**.
- **Problems**:
  - There is no **latent** representation.
  - Vanilla normalizing flows are **not competitive** in generative modeling.
  - The class of flows is restricted in the **discrete-time** setting.
  - The training can be complicated in the **continuous-time** setting.
- **Links with other methods**
  - VAE can be combined with normalizing flows Kingma et al. (2016); Vahdat and Kautz (2020).
  - Score-based generative models can be seen as normalizing flows Song et al. (2021).

Generative Adversarial Networks

Principles of Vanilla GAN

- In **G**enerative **A**dversarial **N**etwork models we *do not* optimize the **log-likelihood** or a lower-bound on the log-likelihood.
- Instead we rely on a **minimax** game.
- We train two **competing** network.
  - A **generative** network which synthesizes data (fake data).
  - A **discriminative** network which tells which data is fake or real.
- This is still related to a **divergence** on probability measures.

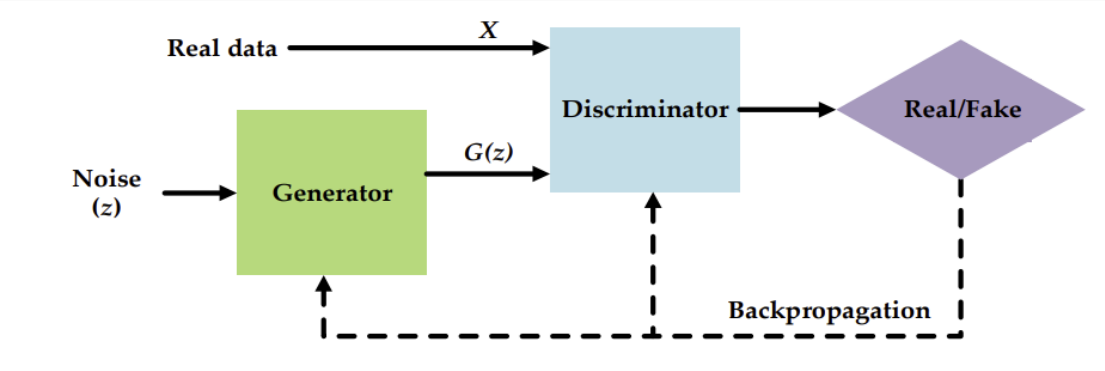

Figure 19: Original GAN model. Image extracted from Feng et al. (2020).

- In what follows:
  - **Vanilla** and **Least-Square GANs**.
  - **IPM** and **WPGAN**.
  - State-of-the-art and a **cautionary tale**.

Vanilla and Least-Square GANs

Loss function and Jensen-Shannon divergence

- We consider a **generator** $g : \mathbb{R}^p \to \mathbb{R}^d$ and a **discriminator** $d : \mathbb{R}^d \to [0,1]$ (networks) which optimize the loss

$$\boxed{\ell(g,d) = -\int_{\mathbb{R}^d} \log(d(x))\mathrm{d}\pi(x) - \int_{\mathbb{R}^p} \log(1-d(g(z)))\mathrm{d}\pi_0(z)\ .}$$

  - π is the data distribution, π_0 is an easy-to-sample distribution.
  - We denote p_g the density of $g_\#\pi_0$ (assuming that it exists) and p the one of π.
  - In practice we **parameterize** the generator and discriminator.
- For a fixed generator, the **optimal discriminator** is given by $d^\star$ such that for any $x \in \mathbb{R}^d$

$$\boxed{d^\star(x) = p(x)/(p(x)+p_g(x))\ .}$$

- Plugging this optimal discriminator into ℓ we get

$$\ell(g,d^\star) = \log(4) - \int_{\mathbb{R}^d} \log(p(x)/p_{\text{mid}}(x))\mathrm{d}p(x) - \int_{\mathbb{R}^d} \log(p_g(x)/p_{\text{mid}}(x))\mathrm{d}p_g(x)\ .$$

  - Hence, $\ell(g,d^\star) = \log(4) - \mathrm{JS}(\pi, g_\#\pi)$, where JS is the **Jensen-Shannon** divergence.
  - The discriminator can be used to improve the quality of samples using a **Metropolis-Hastings** step Turner et al. (2019).

A link with regression

- The **discriminator** tries to classify the data
  - $d(x) = 1$ if the data is from the original dataset.
  - $d(x) = 0$ if the data is from the generated dataset.
- Consider Y a **Bernoulli random variable** and $Y = 1$ with probability $d_\theta(X)$.
- We have $p(Y|x,\theta) = p(Y=1|x,\theta)^Y p(Y=0|x,\theta)^{1-Y}$. Hence, we get that

$$\begin{aligned}
&\int_{\mathbb{R}^d \times \{0,1\}} \log(p(y|x,\theta)) \mathrm{d}\bar{\pi}(x,y) \\
&= \int_{\mathbb{R}^d \times \{0,1\}} \{y \log p(y=1|x,\theta) + (1-y) \log p(y=0|x,\theta)\} \bar{p}(x) \mathrm{d}\bar{\pi}(x,y) \\
&= \int_{\mathbb{R}^d \times \{0,1\}} \{y \log(d_\theta(x)) + (1-y) \log(1-d_\theta(x))\} \mathrm{d}\bar{\pi}(x,y) \ .
\end{aligned}$$

  - $\bar{\pi}$ is the such that $\bar{\pi}_1 = \mathrm{Ber}(1/2)$.
  - $(X,Y) \sim \bar{\pi}$ is such that $X \sim \pi$ (**data distribution**) if $Y = 1$ and $X \sim g_\# \pi_0$ (**generated distribution**) if $Y = 0$.

$$\begin{aligned}
&\int_{\mathbb{R}^d \times \{0,1\}} \log(p(y|x,\theta)) \mathrm{d}\bar{\pi}(x,y) \\
&\qquad = (1/2) \int_{\mathbb{R}^d} \log(d_\theta(x)) \mathrm{d}\pi(x) + (1/2) \int_{\mathbb{R}^p} \log(1 - d_\theta(g(z))) \mathrm{d}\pi_0(z) \ .
\end{aligned}$$

- We recover the **cross-entropy loss**. This is the same as optimizing $\mathrm{KL}(\bar{\pi}|p(|\theta))$ (**maximum likelihood**).

Least-square GAN

- Another flavor of GAN: **L**east **S**quare **GAN** Mao et al. (2017).
- The **discriminator** is a classifier.
  - In **vanilla GAN** we consider the **cross-entropy loss** Goodfellow et al. (2014).
  - In **LSGAN** we consider the **square loss** Mao et al. (2017).
- The (coupled) losses are given by

$$\ell_g(d) = \int_{\mathbb{R}^d} (d(x) - 1)^2 \mathrm{d}\pi(x) + \int_{\mathbb{R}^p} (d(g(z) + 1)^2 \mathrm{d}\pi_0(z) \ ,$$
$$\ell_d(g) = \int_{\mathbb{R}^p} d(g(z)^2 \mathrm{d}\pi_0(z) \ .$$

- We have $\ell_{d^\star}(g) = \chi^2((\pi + g_\# \pi_0)/2 | g_\# \pi_0)$, with χ^2 the **Pearson divergence**.

Figure 20: LSGAN results on LSUN. Image extracted from Mao et al. (2017).

IPM and WPGAN

Integral Probability Metrics

- The concept of **generator/discriminator** can be recovered using **I**ntegral **P**robability **M**etrics.
- An **IPM** is defined by a class of functions $\mathsf{F} \subset \mathcal{F}(\mathbb{R}^d)$ (measurable function from $\mathbb{R}^d$ to $\mathbb{R}$).
- We define $\mathcal{P}_\mathsf{F} = \{\pi \in \mathcal{P}(\mathbb{R}^d) \ : \ \sup_{f \in \mathsf{F}_0} \pi[|f|] < +\infty\}$ (with $f \in \mathsf{F}_0$ if $f(0) = 0$ and $f \in \mathsf{F}$) and d_F such that for any $\pi_1, \pi_2 \in \mathcal{P}_\mathsf{F}$

$$\boxed{d_\mathsf{F}(\pi_1, \pi_2) = \sup\{\pi_1[f] - \pi_2[f] \ : \ f \in \mathsf{F}\} \ .}$$

  - Symmetric and non-negative if $\mathsf{F} = -\mathsf{F}$.
  - Defines a distance on $\mathcal{P}_\mathsf{F}$ if F separates $\mathcal{P}_\mathsf{F}$ in the following sense: for any $\pi_1, \pi_2 \in \mathcal{P}_\mathsf{F}$ there exists $f \in \mathsf{F}$ such that $\pi_1[f] - \pi_2[f] \neq 0$.
  - $f \in \mathsf{F}$ can be seen as a **discriminator** between two probability measures.
- Let F be the set of 1-**Lipschitz** functions.
  - F is separating.
  - The associated **IPM** is called the **Wasserstein distance** of order 1 and is denoted $\mathbf{W}_1$.

Basics on Wasserstein distances

- We have define the **Wasserstein distance of order one** as

$$\boxed{\mathbf{W}_1(\pi_1, \pi_2) = \sup\{\pi_1[f] - \pi_2[f] \ : \ f \in \mathrm{Lip}_1(\mathbb{R}^d)\} \ .}$$

- This is the **dual formulation** of the following definition

$$\boxed{\mathbf{W}_1(\pi_1, \pi_2) = \inf\{\int_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\| \mathrm{d}\Pi(x, y) \ : \ \Pi \in \Lambda(\pi_1, \pi_2)\} \ .}$$

  - $\Lambda(\pi_1, \pi_2)$ is the set of **couplings** between π_1 and π_2
  - For any $\mathrm{A} \in \mathcal{P}(\mathbb{R}^d \times \mathbb{R}^d)$, $\Pi(\mathrm{A} \times \mathbb{R}^d) = \pi_1(\mathrm{A})$ and $\Pi(\mathbb{R}^d \times \mathrm{A}) = \pi_2(\mathrm{A})$.

- By changing $\|x - y\|$ into $\|x - y\|^p$ we can define Wasserstein cost of order p with $p > 0$.

  - This is a distance on $\mathcal{P}_p(\mathbb{R}^d) = \{\pi \in \mathcal{P}(\mathbb{R}^d) \ : \ \int_{\mathbb{R}^d} \|x\|^p \, \mathrm{d}\pi(x) < +\infty\}$ if $p \geq 1$.
  - This distance is stronger than the **weak convergence** (equivalent on **compact sets**).
  - Wasserstein costs are **IPM** only for $p = 1$.

Wasserstein GAN

- Different **divergences** yield different **losses** (vanilla GAN, LSGAN).
- Each **IPM** can be turned into a GAN.

$$\boxed{\inf\{d_{\mathsf{F}}(\pi, g_{\#}\pi_0) \ : \ g \in \mathrm{G}\} = \inf \sup\{\pi[f] - \pi_0[f \circ g] \ : \ f \in \mathsf{F}, g \in \mathrm{G}\}\ ,}$$

  - F is the space of test functions (**discriminators**).
  - F is the space of **generators**.
- Arjovsky et al. (2017) uses the Wasserstein distance of order one.
  - Lipschitz condition is enforced by **clipping of the parameters**.

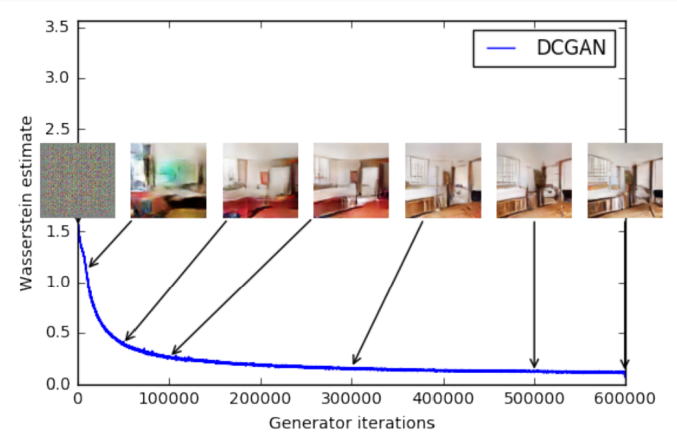

Figure 21: Influence of training on Wasserstein GANs. Image extracted from Arjovsky et al. (2017).

Stability with gradient penalty

- **Gradient clipping** can lead to undesired behavior.
- Gulrajani et al. (2017) proposes to change the loss function of the GAN.

$$\boxed{\ell(f, g) = \pi[f] - \pi_0[f \circ g] + \lambda \pi_0[(\|\nabla f\| \circ g - 1)^2] \ .}$$

  - $\lambda > 0$ is a **regularization parameter**.
  - The last term is a **gradient penalty**.

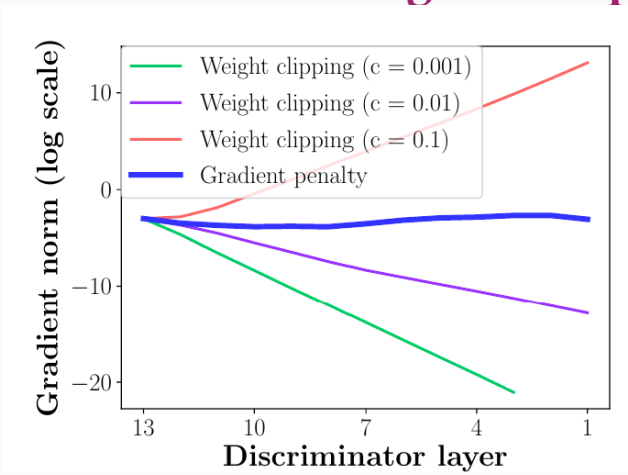

Figure 22: Influence of the regularization. Image extracted from Gulrajani et al. (2017).

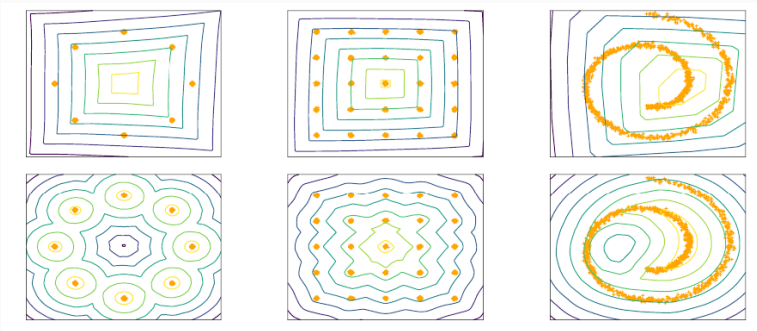

Figure 23: Influence of the gradient penalty. Image extracted from Gulrajani et al. (2017).

State-of-the-art and a cautionary tale

Style GAN

- A first **state-of-the-art** GAN: **Style GAN** Karras et al. (2019).
- Style GAN uses the loss of a **GP-WGAN**.
- Main innovation is the **architecture of the generator**.
- Architecture is used in **style-transfer** Huang and Belongie (2017).

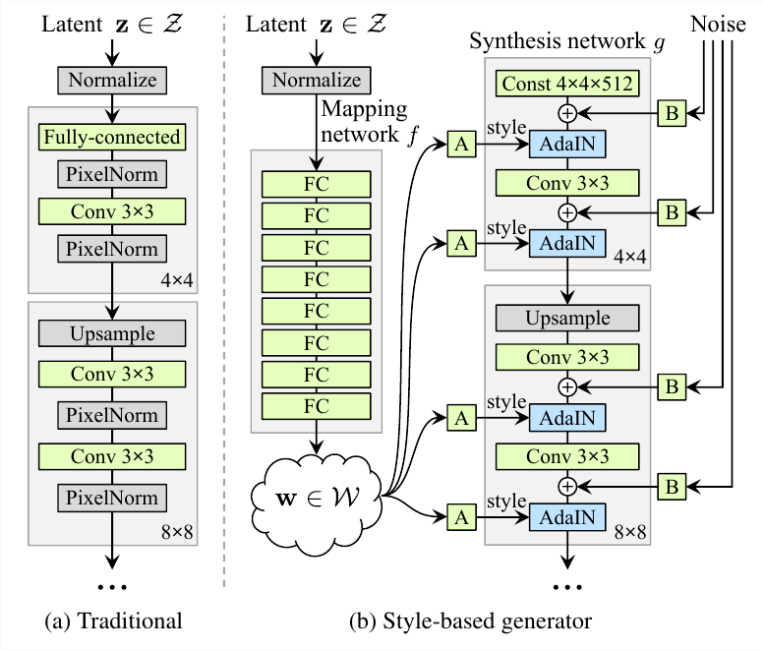

Figure 24: Style GAN architecture and results. Image extracted from Karras et al. (2019).

Big GAN

- Another **state-of-the-art** GAN: **BigGAN** (vanilla GAN).
  - Large models and large batch size improve the results.
  - Introduction of a **truncation trick** to obtain a trade-off between **quality** and **diversity**.
- There are still problems with **training instablities**.
- **Truncation trick**:
  - Train model with **standard Gaussian** in the latent space.
  - Sample with **truncated Gaussian**.
  - This improves the quality of results (but reduces the diversity).

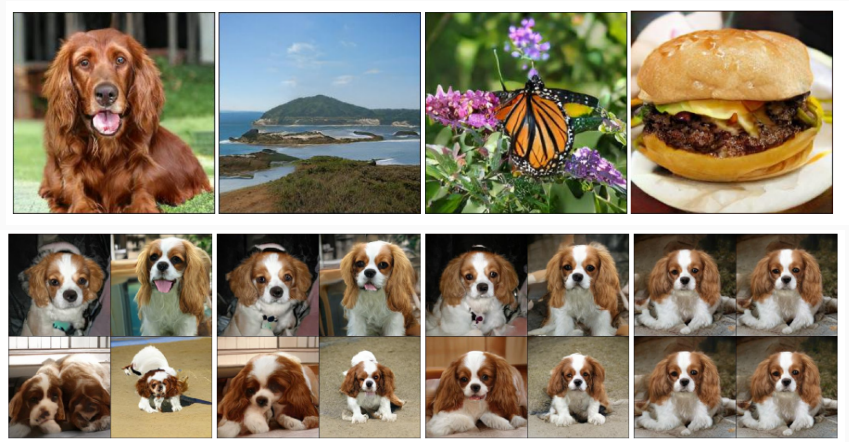

Figure 25: BigGAN results and truncation trick. Image extracted from Brock et al. (2018).

A cautionary tale

- Most of the recent **improvements** come from the **architecture**.
- Vanilla GAN performs as well as other GANs upon **fair comparison** Lucic et al. (2018).
- Wasserstein GANs do not really estimate the **Wasserstein distance** Stanczuk et al. (2021).
  - Lipschitz regularization is always beneficial.
  - Is the Wasserstein distance of order one really what we want to minimize? There is a **conflict with perceptual criterion**.

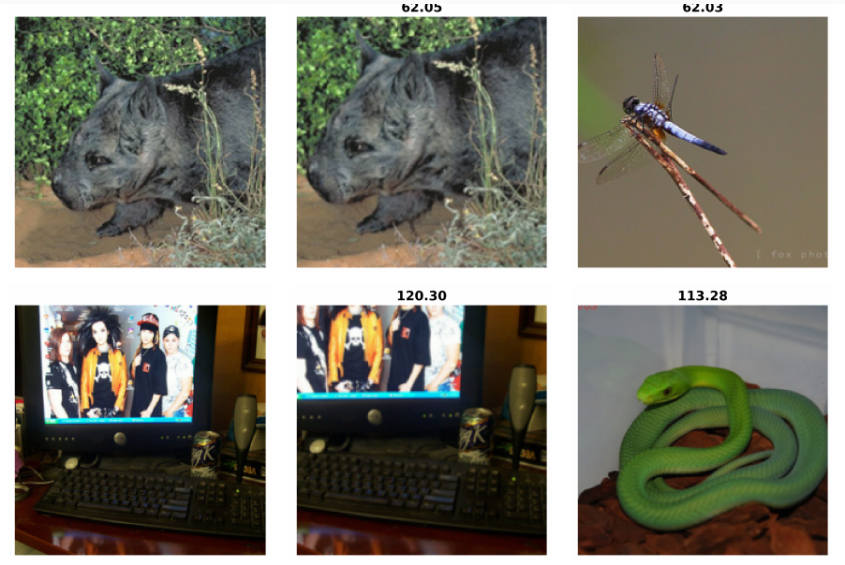

Figure 26: From left to right: original image, modified image, completely different image with lower Euclidean norm. Image extracted from Stanczuk et al. (2021).

Summary of GANs

- **Advantages**:
  - GANs provide **state-of-the-art** results
  - They provide interesting **latent representations**.
  - They allows flexible losses and formulations.
- **Problems**:
  - it is **very hard to train** (collapse during training).
  - Diversity is a problem (**mode collapse**).
  - **Theoretical analysis** is hard Biau et al. (2020).
- **Links with other methods**
  - GANs can be combined with score-based models Xiao et al. (2021).

Conclusion

Conclusion

- Generative modeling has **many different flavors**:
  - Energy-Based Models.
  - Variational AutoEncoders.
  - Normalizing Flows (and Autoregressive models).
  - Generative Adversarial Networks.
- Depending on the application **architecture** matters.
- Until recently GANs were the **state-of-the-art** in terms of visual results.
- In the next sessions: **score-based generative modeling**:
  - New contender with state-of-the-art results.
  - Theoretical analysis is possible.
  - Links with stochastic control and optimal control.

See you all on the 28/02!

References

Michael Arbel, Liang Zhou, and Arthur Gretton. Generalized energy based models. *arXiv preprint arXiv:2003.05033*, 2020.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Victoria A Avanzato, Kasopefoluwa Y Oguntuyo, Marina Escalera-Zamudio, Bernardo Gutierrez, Michael Golden, Sergei L Kosakovsky Pond, Rhys Pryce, Thomas S Walter, Jeffrey Seow, Katie J Doores, et al. A structural basis for antibody-mediated neutralization of nipah virus reveals a site of vulnerability at the fusion glycoprotein apex. *Proceedings of the National Academy of Sciences*, 116(50):25057–25067, 2019.

Gérard Biau, Benoît Cadre, Maxime Sangnier, and Ugo Tanielian. Some theoretical properties of gans. *The Annals of Statistics*, 48(3):1539–1566, 2020.

References ii

Antoine Brochard, Bartłomiej Błaszczyszyn, Stéphane Mallat, and Sixin Zhang. Particle gradient descent model for point process generation. *arXiv preprint arXiv:2010.14928*, 2020.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Anthony L Caterini, Gabriel Loaiza-Ganem, Geoff Pleiss, and John P Cunningham. Rectangular flows for manifold learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling. *Advances in Neural Information Processing Systems*, 33:12275–12287, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

References iii

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Arnak S Dalalyan. Further and stronger analogy between sampling and optimization: Langevin monte carlo and gradient descent. *arXiv preprint arXiv:1704.04752*, 2017.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

Sander Dieleman. Diffusion models are autoencoders, 2022. URL https://benanne.github.io/2022/01/31/diffusion.html.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019.

References iv

Alain Durmus, Gareth O Roberts, Gilles Vilmart, Konstantinos C Zygalakis, et al. Fast langevin based algorithm for mcmc in high dimensions. *The Annals of Applied Probability*, 27(4):2195–2237, 2017.

Jie Feng, Xueliang Feng, Jiantong Chen, Xianghai Cao, Xiangrong Zhang, Licheng Jiao, and Tao Yu. Generative adversarial networks based on collaborative learning and attention mechanism for hyperspectral image classification. *Remote Sensing*, 12(7):1149, 2020.

Ruiqi Gao, Yang Song, Ben Poole, Ying Nian Wu, and Diederik P Kingma. Learning energy-based models by diffusion recovery likelihood. *arXiv preprint arXiv:2012.08125*, 2020.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889. PMLR, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

References v

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Chin-Wei Huang, Jae Hyun Lim, and Aaron C Courville. A variational perspective on diffusion-based generative models and score matching. *Advances in Neural Information Processing Systems*, 34, 2021.

References vi

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE international conference on computer vision*, pages 1501–1510, 2017.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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

References vii

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.

David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

Radford Neal. Bayesian learning via stochastic dynamics. *Advances in neural information processing systems*, 5, 1992.

Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run mcmc toward energy-based model. *Advances in Neural Information Processing Systems*, 32, 2019.

References viii

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

Suman Ravuri, Karel Lenc, Matthew Willson, Dmitry Kangin, Remi Lam, Piotr Mirowski, Megan Fitzsimons, Maria Athanassiadou, Sheleem Kashem, Sam Madge, et al. Skilful precipitation nowcasting using deep generative models of radar. *Nature*, 597(7878):672–677, 2021.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

Gareth O Roberts, Richard L Tweedie, et al. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341–363, 1996.

Geoffrey Roeder, Yuhuai Wu, and David K Duvenaud. Sticking the landing: Simple, lower-variance gradient estimators for variational inference. *Advances in Neural Information Processing Systems*, 30, 2017.

Lars Ruthotto and Eldad Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, 44(2):e202100008, 2021.

References ix

Veit Sandfort, Ke Yan, Perry J Pickhardt, and Ronald M Summers. Data augmentation using generative adversarial networks (cyclegan) to improve generalizability in ct segmentation tasks. *Scientific reports*, 9(1):1–9, 2019.

Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. *Advances in neural information processing systems*, 29, 2016.

Jiaming Song, Shengjia Zhao, and Stefano Ermon. A-nice-mc: Adversarial training for mcmc. *Advances in Neural Information Processing Systems*, 30, 2017.

Yang Song and Diederik P Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.

References x

Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34, 2021.

Jan Stanczuk, Christian Etmann, Lisa Maria Kreusser, and Carola-Bibiane Schönlieb. Wasserstein gans work because they fail (to approximate the wasserstein distance). *arXiv preprint arXiv:2103.01678*, 2021.

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071, 2008.

Ryan Turner, Jane Hung, Eric Frank, Yunus Saatchi, and Jason Yosinski. Metropolis-hastings generative adversarial networks. In *International Conference on Machine Learning*, pages 6345–6353. PMLR, 2019.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020.

Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. *Advances in Neural Information Processing Systems*, 34, 2021.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR, 2016.

Antoine Wehenkel and Gilles Louppe. Diffusion priors in variational autoencoders. *arXiv preprint arXiv:2106.15671*, 2021.

Zhisheng Xiao, Karsten Kreis, Jan Kautz, and Arash Vahdat. Vaebm: A symbiosis between variational autoencoders and energy-based models. *arXiv preprint arXiv:2010.00654*, 2020.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion gans. *arXiv preprint arXiv:2112.07804*, 2021.