DEPARTMENT OF COMPUTING

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

VANs and GANs

Lecture notes on the Variational Encoder Networks [2] and the Generative Adversarial Networks [1]

Author: Anton Zhitomirsky

Contents

1 Introduction

1.1 Supervised Learning

Supervised learning is a process which attempts to learn patterns amongst data to predict a label y given an imput x, i.e. learn a function to map $x \to y$. For this to work, supervised learning uses labelled data inputs (x, y) to train its parameters.

$$\text{Data: } (x_1, y_1), \ldots, (x_N, y_N) \sim p_{data}(x, y) \tag{1.1.1}$$

1.2 Unsupervised Learning

However, most of the data we have available isn't well-formed pairs of labelled data. It just an anonymous data point. Thus, the task is now to inferr a function that describes the hidden structure of unlabelled data.

$$\text{Data: } x_1, \ldots, x_N \sim p_{data}(x) \quad \text{no supervision signal} \tag{1.2.1}$$

1.2.1 Probability Distribution/Density Estimation

Assume the data is sampled form an underlying data distribution (Equation 1.2.1) with a given probability density $p_{data}(x)$. The goal is to learn a distribution or **probabilistic model** where θ is the collection of learnable parameters.

$$p_\theta(x) \approx p_{data}(x), \quad \text{with data } x_1, \ldots, x_N \tag{1.2.2}$$

1.3 Generative Latent Variable Models

Design $p_\theta(x)$ as a generative latent variable model (LVM). The idea is to describe the sampling of the oservation x as a generative process where we first sample the latent variable z and then generate x condition on z.

$$z \approx p_\theta(z), \quad x \approx p_\theta(x|z) \tag{1.3.1}$$

$$\Rightarrow p_\theta(x) = \int p_\theta(x|z) p_\theta(z) dz \tag{1.3.2}$$

z : latent variable (unobserved) $\quad x$: observed variable

Examples:

- – z: digit label, writing style, ...
  – x: hand-written digit
- – z: scene, viewing angle, lighting condition, ...
  – x: tet photo image
- – z: semantic, sentiment meaning, ...
  – x: generated text

1.4 Dimensionality Reduction

The goal is to represent the observed data points with lower dimesnional features. Given high-dimensional raw data, it is often sparse, perhaps lying ona low-dimensional mainfold.

1.4.1 Principal Componenet Analysis (PCA)

Given input data, PCA find the principal components to explain the variability in data as orthogonal directions that capture most of the variance in the data. The principal components are generated by magnitude, i.e. direction of gratest variability followed by the next orhogonal (uncorrelated) direction of greatest variability and so on.

In practice, we look at the first k principal components and then project the data points to the subspace span by those key princiapl componenets. Dimensionality reduction is achieved by projecting the data on the top $K < d$ princiapl components ($x \in \mathbb{R}^d$).

1.4.2 Probabilistic PCA

We can generalice PCA to a probabilistic version of it, which is still a Latent Variable Model as before.

$$p(z) = \mathcal{N}(z; 0; I), \quad z \in \mathbb{R}^K, \quad K < d \tag{1.4.1}$$

$$p_\theta(x|z) = \mathcal{N}(x; Wz, \theta^2 I), \quad x \in \mathbb{R}^d \tag{1.4.2}$$

$$\text{Parameters to optimize} : \theta = W \in \mathbb{R}^{d \times K}$$

Probabilistic PCA again assumes the observed data x is generated conditionally on a latent variable z. Here, the prior distribution of x is a standard Gaussian, and the conditional generation process is linear.

By training the probPCA with maximum likelihood, one can show that the W matrix contains the otp K princiapl components, which are the top K eigenvectors of the data covariance matrix.

1.4.3 Auto-encoders

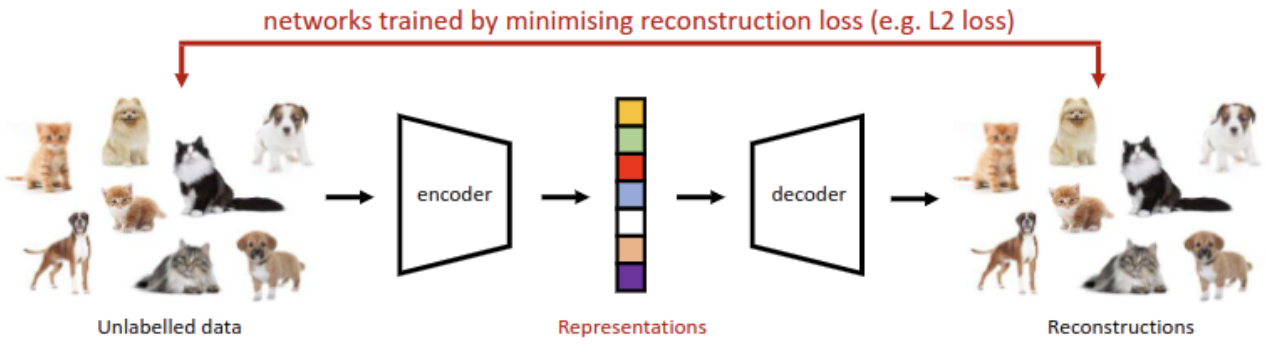


Figure 1: Encoder network extracts data representations (often with lower dimensionality) and Decoder network to reconstruct data given the representations

The dimensionality is lower than the input becuase othewrise the networks can simply learn identity mappings to copy froward whatever is seen.

1.5 Clustering

Clustering discovers group structures in unlabelled datapoints. It does so by grouping datapoints into several clusters, where datapoints in the same cluster are similar, and otherwise dissimilar.

1.5.1 Gaussian mixture model (GMM)

$$p_\theta(z) = Categorical(\pi), \tag{1.5.1}$$

$$\pi = (\pi_1, \dots, \pi_k), \pi_i = p_\theta(z = 1), \quad \sum_{i=1}^{K} \pi_i = 1 \tag{1.5.2}$$

$$p_\theta(x|z) = \mathcal{N}(x; \mu_z; \Sigma_z) \tag{1.5.3}$$

$z \in \{1, \dots K\}$: index of the Gaussian component

μ_z : mean of the i^{th} Gaussian component if $z = i$

Σ_z : Covariance matrix of the i^{th} Gaussian component if $z = i$

Clustering can be done by fititng a GMM model to the data.

We can optimize the Gaussian component parmaeters and the categorical distribution on z by maximum likelihood.

1.6 Representation learning

Both dimensionality reduction and clustering can be viewed as representation learning. The hope is for these models useful for downstream tasks.

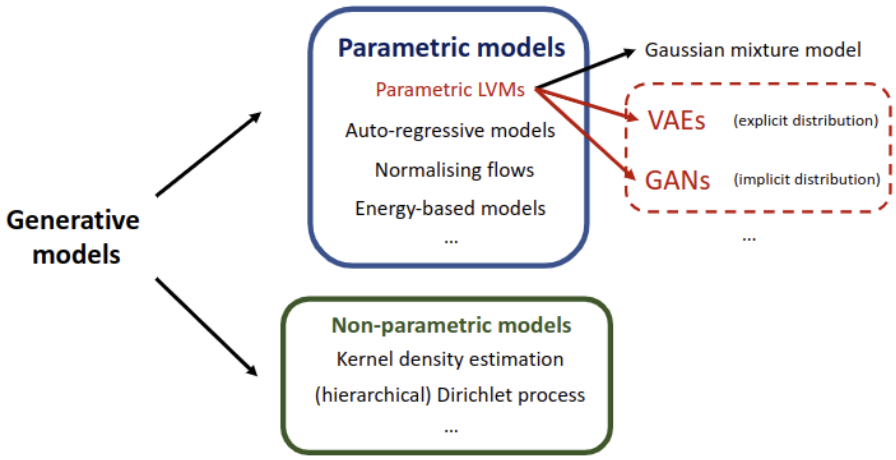


Figure 2

2 Variational AutoEncoder basics

We first begin by discussing the information that is available to us:

$$\underbrace{p(\theta|x)}_{\text{posterior}} = \frac{\overbrace{p(x|\theta)}^{\text{likelihood}} \cdot \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(x)}_{\text{evidence}}} \tag{2.0.1}$$

The goal is to approximate the $p_\theta(x)$ function that is learnt to closely mimic the goal $p_{data}(x)$ probability distirbution. To achieve this, we first need to consider a criterai to measure the closesness of two probability distirbutions. Then we can optimize it to make the model distribution close to the data distribution.

2.1 Divergence minimisation

Given a set of probability distributions $\mathcal{P}$ on a random variable X, a divergence is defined as a function $\mathrm{D}[\cdot||\cdot] : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ such that $\mathrm{D}[P||Q] \geq 0$ for all $P, Q \in \mathcal{P}$, and $\mathrm{D}[P||Q] = 0$ iff. $P = Q$.

The definition of divergence is much weaker than that for a *distance* such as the ℓ_2-norm, since it does not need to satisfy either symmetry in arguments or the triangle inequality. There exist many available divergences to use, some of them will be introduced throughout this course.

In this course we assume the probability distributions/measures in $\mathcal{P}$ are dominated by the Lebesgue measure of the underlying Eucledean space, so that we can work with *probability density functions* (PDFs)

$$p(\boldsymbol{x}) = \frac{dP}{d\boldsymbol{x}}, \forall P \in \mathcal{P}. \tag{1}$$

In the following we will also write $\mathcal{P}$ as the set of PDFs w.l.o.g., and use the terms probability distribution and probability density functions interchangeably (unless specifically mentioned).

We can then find the best parameter settings θ^* that corresponds to the probablistic model which minimises the model distribution's divergence to the data distribution.

$$\theta^* = \arg\min D[p_{data}(x)||p_\theta(x)] \tag{2.1.1}$$

2.2 Kullback-Leibler (KL) divergence

Kullback-Leibler divergence [Kullback and Leibler, 1951; Kullback, 1959], or *KL divergence*, is arguably one of the most widely used divergence measures in machine learning, statistics, and information theory.

Definition 1. *(Kullback-Leibler Divergence) The Kullback-Leibler (KL) divergence on $\mathcal{P}$ is defined as a function $\mathrm{KL}[\cdot||\cdot] : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ with the following form*

$$\mathrm{KL}[p||q] = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}, \quad p, q \in \mathcal{P}, \tag{3}$$

where log *is the natural logarithm (to base e).*

Equivalently, $\mathrm{KL}[p||q] = \mathbb{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]$

One can easily check that indeed the above definition is a valid divergence: define $f(x) = -\log x$ (which is convex) and $g(\boldsymbol{x}) = q(\boldsymbol{x})/p(\boldsymbol{x})$, we have

$$\begin{aligned}\mathrm{KL}[p||q] &= \mathbb{E}_{p(\boldsymbol{x})}[-\log g(\boldsymbol{x})] \\ &\geq -\log \mathbb{E}_{p(\boldsymbol{x})}[g(\boldsymbol{x})] \qquad \text{(Jensen's inequality)} \\ &= -\log \int p(\boldsymbol{x}) \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{x} = -\log 1 = 0,\end{aligned}$$

and the equality holds iff. $p(\boldsymbol{x}) = q(\boldsymbol{x})$.[1] This means one can minimise the KL divergence in order to fit a distribution to a target one. Also notice that the KL divergence is asymmetric, i.e. $\mathrm{KL}[p||q] \neq \mathrm{KL}[q||p]$ in general.

Note, that here $-\log x \equiv \log \frac{1}{x}$
We also use Jensen's inequality (Appendix A.1.2)
[1] techincally speaking $p(\boldsymbol{x}) = q(\boldsymbol{x})$ almost everywhere.
And since probability densities sum to one then we have $-\log 1 = 0$.

2.3 Maximum Likelihood Estimation

Given a dataset $\{(\boldsymbol{x}_n)\}_{n=1}^N \sim p_{\text{data}}(\boldsymbol{x})$, we would like to fit to it a generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ with parameter $\boldsymbol{\theta}$. Since the KL divergence can be used to measure the closeness of the model to the underlying data distribution, it makes sense to find the optimal parameters by minimising the KL divergence:

$$\boldsymbol{\theta}^* = \arg\min \mathrm{KL}[p_{\text{data}}(\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{x})]. \tag{4}$$

Expanding the above objective and re-arranging terms, we have

$$\mathrm{KL}[p_{\text{data}}(\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{x})] = \underbrace{\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log p_{\text{data}}(\boldsymbol{x})]}_{\text{constant w.r.t. } \boldsymbol{\theta}} - \underbrace{\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]}_{\text{dependent on } \boldsymbol{\theta}}.$$

This means we can ignore the constant terms w.r.t. $\boldsymbol{\theta}$ and instead work with the following *maximum likelihood* objective:

$$\boldsymbol{\theta}^* = \arg\max \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x})]. \tag{5}$$

The obtained optimal parameters $\boldsymbol{\theta}^*$ is called the *maximum likelihood estimate* (MLE) of the parameters. In practice the data distribution is approximated by the empirical distribution on the dataset $\{\boldsymbol{x}_n\}_{n=1}^N \sim p_{\text{data}}(\boldsymbol{x})$, leading to

$$\boldsymbol{\theta}^* = \arg\max \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n). \tag{6}$$

The equation at (5 or 6) above, requires us to write down the likelihood of θ or, in other words, the marginal distribution of X. This means that we need to calculate the integral over the join distribution over the latent variable z. The conditional distirbution of $p(x|z)$ is often defined by a neural network. In this case, we cannot compute the integral because it would mean computing the network output of z given all possible values of x. This is intractible. Since the marginal distribution is intractible then so is the Maximum Liklihood Estimation.

2.4 Optimising a variational lower-bound

Therefore we opitimize the variational lower-bound of the maximum likelihood objective function instead. We create a lower bound to the log marginal distribution for every input x, and then the expectation of this lower bound will become the lower bound of the maximum likelihood objective.

We are interested in fitting the following latent variable model (LVM) to the data:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}. \tag{7}$$

See Figure 1 (a) for a visualisation of the graphical model. In deep generative modelling context, this LVM is often constructed as (for continuous data)

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{I}), \quad p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; G_{\boldsymbol{\theta}}(\boldsymbol{z}), \sigma^2\mathbf{I}), \tag{8}$$

with $G_{\boldsymbol{\theta}}(\cdot)$ define as a neural network transform that is parameterised by weights $\boldsymbol{\theta}$. For discrete variables $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is usually defined as a categorical distribution with a neural network generator in use accordingly. Now to fit $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ to $p_{\text{data}}(\boldsymbol{x})$ we optimise the MLE objective (5) w.r.t. $\boldsymbol{\theta}$, which involves computing the integral (7). This is intractable as it involves computing the non-linear transformation $G_{\boldsymbol{\theta}}(\boldsymbol{z})$ for every single configuration of $\boldsymbol{z}$ within the support of the Gaussian prior $p(\boldsymbol{z})$, which is the full space $\boldsymbol{z} \in \mathbb{R}^d$.

Variational inference provides a variational lower-bound of $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ as an approximation to it. For any distribution $q(\boldsymbol{z})$ satisfying $q(\boldsymbol{z}) > 0$ whenever $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) > 0$, we have

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= \log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \\
&= \log \int q(\boldsymbol{z}) \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z})} d\boldsymbol{z} \\
&\geq \int q(\boldsymbol{z}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z})} d\boldsymbol{z} \qquad \text{(Jensen's inequality)} \\
&= \mathbb{E}_{q(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}[q(\boldsymbol{z})||p(\boldsymbol{z})] := \mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta}).
\end{aligned} \tag{9}$$

With suitable choice of $q(\boldsymbol{z})$ and tricks that will be introduced later, this varitational lower-bound can be used as a tractable approximation to the marginal log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Here, $\mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta})$ refers to the variational lower bound. This is the quantity that variational inference aims to maximize during the optimization process.

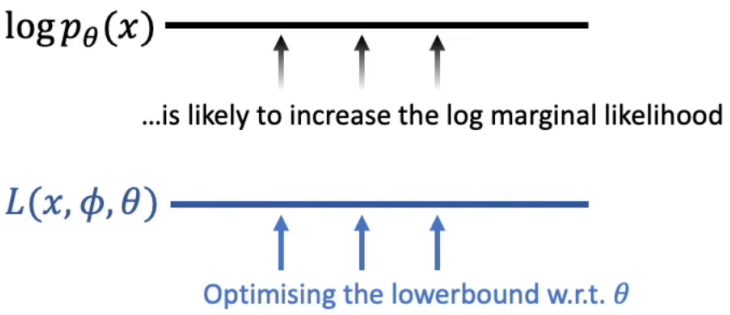


Figure 3: The relationship between the log marginal likelihood and the variational lower bound.

By maximising the lower bound to the log marginal distribution the marginal log liklihood itself is also likely to incrase since it is always greater than or equal to the lower bound.

2.4.1 Alternative Approach

The choice of the $q(\boldsymbol{z})$ distribution is crucial to the quality of the approximation (or the tightness of the lower-bound). To see this, note that

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}, \qquad \text{(Bayes' rule)} \tag{10}$$

$$\begin{aligned}
\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathrm{KL}[q(\boldsymbol{z})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})] &= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{q(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) + \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right] \qquad \text{(Bayes' rule)} \\
&= \mathbb{E}_{q(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}[q(\boldsymbol{z})||p(\boldsymbol{z})] = \mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta}).
\end{aligned} \tag{11}$$

This means the gap (or the approximation error) between the variational lower-bound $\mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta})$ and the marginal log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is the KL divergence $\mathrm{KL}[q(\boldsymbol{z})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})]$. Therefore the lower-bound improves as $q(\boldsymbol{z})$ approaches to the exact posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. It also motivates the optimisation of the variational lower-bound w.r.t. the q distribution to obtain an approxiate posterior: since $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is constant w.r.t. q, maximising $\mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta})$ is equivalent to minimising $\mathrm{KL}[q(\boldsymbol{z})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})]$.

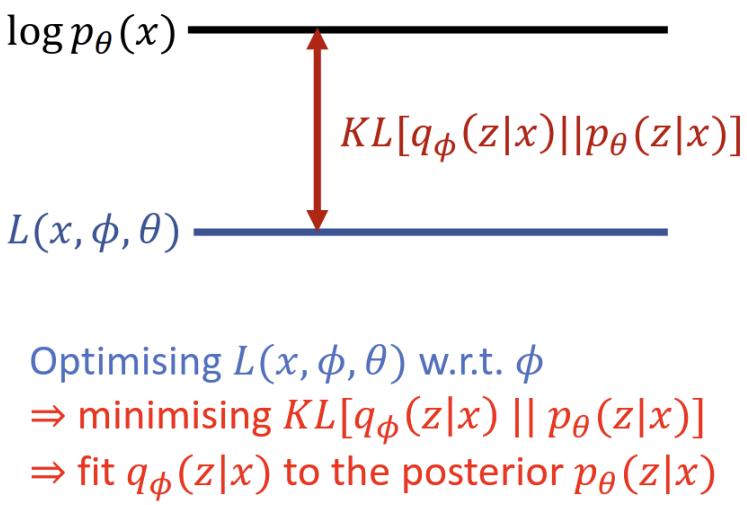


Figure 4: The alternative relationship between the log marginal likelihood and the variational lower bound

This derrivation tells us that the gap between the marignal log likelihood and the variational lower bound is the KL divergence between the Q distribution and the example distribution.

Previously, we said that optimizing the variational lower bound w.r.t to θ, is likely to optimize the marignal log likelihood. However, it doesn't happen always; it depends on the gap.

To reduce the bias of optimizing the varaitional lower bound, we would like it to be as tight as posisble. Therefore we also optimize the variatonal lower bound w.r.t q parameters ϕ. Since the log marginal is a constant w.r.t ϕ it is also equivalent to minimising the KL divergence from q to the exact posterior[1]. This will fit the distribution to the approximation of the example posterior.

2.5 Variational auto-encoders

So far we have the following objectives:

[1]Posterior probability is the probability of the parameters θ given the evidence X, denoted $p(\theta|X)$. This is contrary to likelihood function, which is the probability of the evidence given parameters $p(X|\theta)$

$$\theta^*, \phi^* = \arg\max L(\phi, \theta) \tag{2.5.1}$$

$$L(\phi, \theta) := E_{p_{data}(x)}[E_{p_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)]] \tag{2.5.2}$$

As discussed so far, we wish to fit the generative model (7) to the data by maximum likelihood (5), and variational inference provides a useful approximation $\mathcal{L}(\boldsymbol{x}, q, \boldsymbol{\theta}) \leq \log p_{\boldsymbol{\theta}}(x)$ for a given datum $\boldsymbol{x}$. Since this approximation is required for every datapoint in $\{\boldsymbol{x}_n\}_{n=1}^N$, having N separated q distributions $q_1(\boldsymbol{z}_1), ..., q_N(\boldsymbol{z}_n)$ to pair with $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ can be memory inefficient. However, notice that the exact posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ depends on the input $\boldsymbol{x}$, and the variational lower-bound is tight when $q_n(\boldsymbol{z}_n) \approx p_{\boldsymbol{\theta}}(\boldsymbol{z}_n|\boldsymbol{x}_n)$. This motivates the *variational auto-encoder* (VAE) approach [Kingma and Welling, 2014; Rezende et al., 2014] which defines the q distribution as $q(\boldsymbol{z}) := q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, with the distribution often defined by a neural network:

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), diag(\sigma^2_{\boldsymbol{\phi}}(\boldsymbol{x}))), \quad \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \log \sigma_{\boldsymbol{\phi}}(\boldsymbol{x}) = \mathrm{NN}_{\boldsymbol{\phi}}(\boldsymbol{x}). \tag{12}$$

This allows us to define the VAE optimisation objective:

$$\boldsymbol{\phi}^*, \boldsymbol{\theta}^* = \arg\max \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}), \quad \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathbb{E}_{p_{\mathrm{data}}(\boldsymbol{x})}[\underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]}_{:=\mathcal{L}(\boldsymbol{x}, \boldsymbol{\phi}, \boldsymbol{\theta})}]. \tag{13}$$

Auto-encoder refers to the encode-decode procedure using the q and p distribution respectively. Variational means that both both p and q are trained using the variational lower bound.

In equation 12, similar to the generative model p, we can parameterise q with neural networks; with factorized gaussian distribution, with mean and variance of z parameterised by the neural network transform of x. In the RHS of the equation, we can parameterise the logarithm by the neural networks to ensure that the variance is non-negative.

Since the prior on z is also gaussian we can derrive an analytic form for the KL reguliser. The proof remains in the appendix at Section A.1.3

Given that both $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ are all factorised Gaussian distributions, the KL divergence term in (13) has an analytic form (assuming $\boldsymbol{z} \in \mathbb{R}^d$):

$$\mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})] = \frac{1}{2}\left(||\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x})||_2^2 + ||\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x})||_2^2 - 2\langle \log \boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}), \mathbf{1}\rangle - d\right). \tag{14}$$

2.5.1 Reparameterisation trick

In the VAE optimisation objective: $L(\phi, \theta) := E_{p_{data}(x)}[\underline{E_{p_\phi(z|x)}[\log p_\theta(x|z)]} - KL[q_\phi(z|x)||p(z)]]$ the underlined term is expensive to calculate. This loglikelihood term is intractible, it requires computing the expectation under the q distribution. It is still expensive to pass every z through the generative network.

2.5.2 Monte-carlo estimation

We can therefore use Monte-carlo estimation to help. We can approximate the expected log likelihood return with the log-likelihood return evaluated on a single sample from the q distribution. This way we can differntiate the right-hand side of the expression with respect to θ to obtain the graident for learning θ.

The VAE objective $\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta})$ in (13) is still intractable since the expectation computation $\mathbb{E}_{q_{\boldsymbol{\phi}}}[\cdot]$ requires evaluating neural network transformations for all possible $\boldsymbol{z}$. Monte Carlo (MC) estimation comes into rescue, as we can replace the expectation with MC approximations:

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] \approx \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}), \quad \boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}). \tag{17}$$

By doing so, the gradient of the objective w.r.t. $\boldsymbol{\theta}$ can be estimated as

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\phi}, \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}), \quad \boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}). \tag{18}$$

However, it is unclear how to learn the variational parameter ϕ. We can similarly reparameterise the term

$$E_{q_\phi(z|x)}[\log p_\theta(x|z)] \approx \log p_\theta(x|z), \quad z \sim q_\phi(z|x) \tag{2.5.3}$$

It remains to compute the gradient of the objective w.r.t. $\boldsymbol{\phi}$

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\phi}, \boldsymbol{\theta}) \approx \nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \nabla_{\boldsymbol{\phi}} \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]. \tag{19}$$

While the gradient w.r.t. the KL term tractable (by differentiate eq. (14) w.r.t. $\boldsymbol{\phi}$), MC approximation is still required for the first term in (19).

2.5.3 Reparameterisation trick

The MC approximation to $\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})]$ is further assisted by the reparameterisation trick [Kingma and Welling, 2014; Rezende et al., 2014]. Note that the sampling procedure of a Gaussian variable is the following:

$$\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \quad \Leftrightarrow \quad \boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}} + \boldsymbol{\sigma}_{\boldsymbol{\phi}} \odot \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}), \tag{20}$$

[2]Please note that the reparameterisation trick is not the only method to enable MC estimation of VAE gradients w.r.t. ϕ even when we use Gaussian q distributions. *If interested, see e.g., Section 2.2.3 of this note.

with $\odot$ denoting element-wise product. Writing $\pi(\boldsymbol{\epsilon}) := \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \mathbf{I})$ and $T_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) := \boldsymbol{\mu}_{\boldsymbol{\phi}} + \boldsymbol{\sigma}_{\boldsymbol{\phi}} \odot \boldsymbol{\epsilon}$, we have, by LOTUS,

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] = \mathbb{E}_{\pi(\boldsymbol{\epsilon})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|T_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}))], \tag{21}$$

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] = \mathbb{E}_{\pi(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|T_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}))] = \mathbb{E}_{\pi(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} \boldsymbol{z} \nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})|_{\boldsymbol{z}=T_{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{\epsilon})}]. \tag{22}$$

Then with MC estimation:

$$\mathbb{E}_{\pi(\boldsymbol{\epsilon})}[\nabla_{\boldsymbol{\phi}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|T_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}))] \approx \nabla_{\boldsymbol{\phi}} \boldsymbol{z} \nabla_{\boldsymbol{z}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})|_{\boldsymbol{z}=T_{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{\epsilon})}, \ \ \boldsymbol{\epsilon} \sim \pi(\boldsymbol{\epsilon}). \tag{23}$$

Combined with eq. (18) and mini-batch training, one can compute an MC estimation of the VAE objective (13) as

$$\begin{aligned} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}) \approx \frac{1}{M} \sum_{m=1}^{M} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_m|T_{\boldsymbol{\phi}}(\boldsymbol{x}_m, \boldsymbol{\epsilon}_m)) - \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}_m|\boldsymbol{x}_m)||p(\boldsymbol{z}_m)], \\ \boldsymbol{x}_1, ..., \boldsymbol{x}_m \sim \{\boldsymbol{x}_n\}^M, \ \ \boldsymbol{\epsilon}_1, ..., \boldsymbol{\epsilon}_M \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \end{aligned} \tag{24}$$

and apply e.g. automatic differentiation to obtain the (MC estimation of) gradient of the VAE objective w.r.t. parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

2.5.4 Conclusion

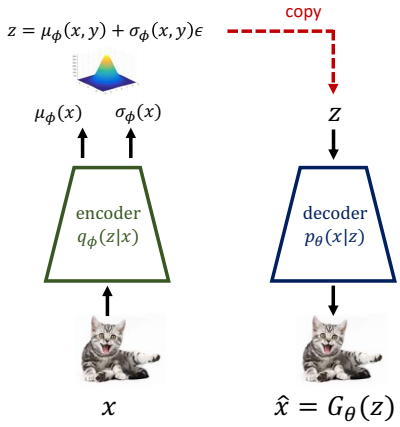
- Combining all the ingredients together:

$$\theta^*, \phi^* = argmax\ L(\phi, \theta)$$

$$L(\phi, \theta) := E_{p_{data}(x)}\{ -\underbrace{E_{N(\epsilon;0,I)}\left[\frac{1}{2\sigma^2} \parallel G_\theta\left(T_\phi(x, \epsilon)\right) - x \parallel_2^2\right]}_{\text{stochastic auto-encoder}} \underbrace{-KL[q_\phi(z|x) \parallel p(z)]}_{\text{KL regularizer}} \}$$

Reconstruction loss

KL regularizer
to make q closer to the prior and prevent $\sigma_\phi(x) \to 0$



- The first term in the variational lower bound corresponds to the reconstruction error of the auto-encoding procedure with a noteble difference, that the encoder injects a noise variable ϵ into the encoding z.
- On the figure, we see that there will be a deterministic auto-encoder if the variance of the Q distribution is 0
- This variance collapse is prevented by the extra KL regularisation term that is not the usual auto-encoder objective.
- The auto-encoder will be stochastic after learning.
- The idea of the reguliser, is to make the Q distribution closer to the prior. So after learning, the stochastic encoding of the observed data will have a high probability on the prior.

2.6 Generating data from the VAE

Once trained sample new images from the model with

$$z \sim p(z), \quad x \sim p(x|z) \tag{2.6.1}$$

Often we define z as a multivariate latent variable. The hope is after learning, different dimensions of z will encode different information of the observed data. Therefore, by varying values in different dimensions in the latent variable this **disentangles** representation.

2.6.1 Pseudo-code

- Initialise θ, ϕ, learning rates γ, choose total iteration T for SGD
- For $t = 1, \dots, T$
  - $x_1, \dots, x_M \sim p_{data}(x)$

  # encoder: performing (approximate) posterior inference
  - Compute $\mu_\phi(x_m), \sigma_\phi(x_m)$ for $m = 1, \dots, M$
  - $z_m = \mu_\phi(x_m) + \sigma_\phi(x_m) \odot \epsilon_m, \ \epsilon_m \sim N(0, I)$ # reparam. trick

  # Decoder: reconstructing data
  - $\hat{x}_m = G_\theta(z_m)$ for $m = 1, \dots, M$

  # update neural network parameters
  - $L = \frac{1}{M}\sum_{m=1}^{M}[-\frac{1}{2\sigma^2}\|x_m - \hat{x}_m\|_2^2 - KL[q_\phi(z_m|x_m) \,||\, p(z_m)]]$
  - $(\theta, \phi) \leftarrow (\theta, \phi) + \gamma \nabla_{(\theta,\phi)} L$

can use the analytic KL form or estimated by Monte Carlo

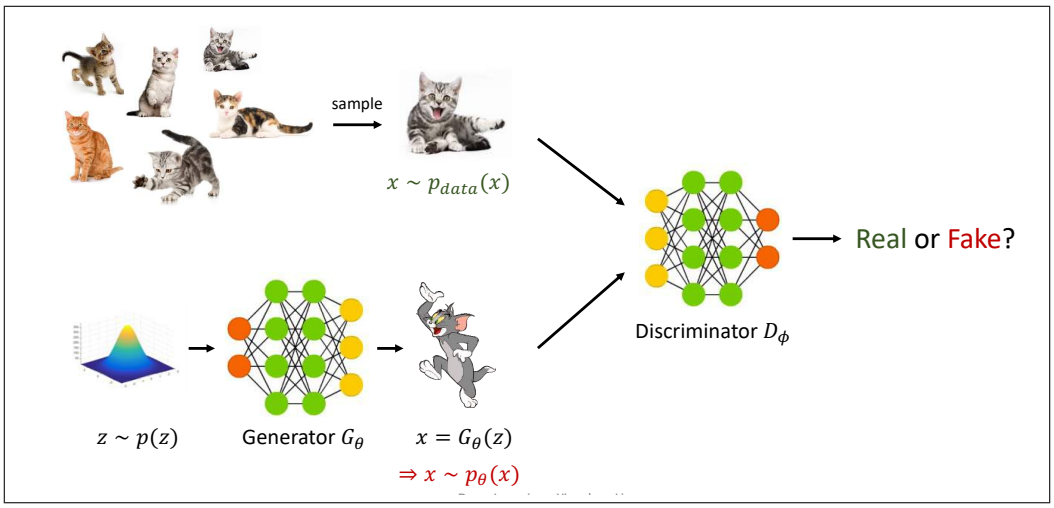
A practical trick: KL annealing

3 GAN

Similarly, we are trying to approximate a model $p_\theta(x) \approx p_{data}(x)$. We achieve this by tring to minimise the divergence between the model and the data, by selecting parameters in $\boldsymbol{\theta}$ which minimise the divergence:

$$\theta^* = \arg_\theta \min D[p_{data}(x)||p_\theta(x)] \tag{3.0.1}$$

3.1 Architecture



3.1.1 Two-player game objective

The generative adversarial network (GAN) approach [Goodfellow et al., 2014] constructs a binary classification task to assist the learning of the generative model distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ to fit the data distribution $p_{\text{data}}(\boldsymbol{x})$. This is done by labelling all the datapoints sampled from the data distribution as "real" data and those sampled from the model as "fake" data. In other words, a joint distribution $\tilde{p}(\boldsymbol{x}, y)$ is constructed as follows for the binary classification task:

$$\tilde{p}(\boldsymbol{x}, y) = \tilde{p}(\boldsymbol{x}|y)\tilde{p}(y), \quad \tilde{p}(y) = \text{Bern}(0.5), \quad \tilde{p}(\boldsymbol{x}|y) = \begin{cases} p_{\text{data}}(\boldsymbol{x}), & y = 1 \\ p_{\boldsymbol{\theta}}(\boldsymbol{x}), & y = 0 \end{cases}. \tag{3}$$

Fitting a binary classifier ("discriminator") with $p_{\boldsymbol{\phi}}(y = 1|\boldsymbol{x}) = D_{\boldsymbol{\phi}}(\boldsymbol{x})$ to $\tilde{p}(y|\boldsymbol{x})$ can be done by maximising the maximum likelihood objective (see eq. (2)):

$$\boldsymbol{\phi}^*(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}), \quad \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) := \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})] + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))]. \tag{4}$$

We have also that $D_\phi(x) := P(x\ is\ real) \wedge 1 - D_\phi(x) = P(x\ is\ fake)$

With a fixed θ: training D_ϕ as the classifier of the binary classification task with maximum likelihood (i.e. negative cross entropy):

$$y = 1\ if\ x \sim p_{data}(x), \quad else \quad y = 0\ if\ x \sim p_\theta(x) \tag{3.1.1}$$

With fixed ϕ: training G_θ to minimize the log-probability of $x \sum p_\theta(x)$ being classified as "fake data" by D_ϕ.

Notice the dependence of the objective (4) on the generative model parameter $\boldsymbol{\theta}$, since the "data distribution" $\tilde{p}(\boldsymbol{x}, y)$ of the binary classification task depends on $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Then the training of the generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ aims at fooling the discriminator, by *minimising* the log probability of making the right decisions:

$$\boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))]. \tag{5}$$

3.2 Solving the two-player game objective

In summary, the two-player game objective for training the GAN generator and discriminator is

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}). \tag{6}$$

Importantly, the terms in the objective related to $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))]$ which in practice is approximated by Monte Carlo:

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x}))] \approx \log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x})), \quad \boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}). \tag{7}$$

Therefore the evaluation of the objective does not require computation of the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, instead one can directly define the sampling process of $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which also defines the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ in an *implicit* way:

$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \Leftrightarrow \quad \boldsymbol{z} \sim p(\boldsymbol{z}), \quad \boldsymbol{x} = G_{\boldsymbol{\theta}}(\boldsymbol{z}), \tag{8}$$

and often we set $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathrm{I})$.

The solution of this minmax optimisation task is at equilibrium of the two player adversarial game

3.2.1 Solution to inner-loop optimisation

Assuming the discriminator network D_ϕ has infinite capacity with fixed θ, we can show that the outermost discriminator is the base classifier, whcih computes the ratio between the data distirbution between and the sum of the data and model distributions.

$$\phi^* = \max_\phi L(\theta, \phi) \quad \text{satisfies} \quad D_{\phi^*}(x) = \frac{p_{data}(x)}{p_{data}(x) + p_\theta(x)} \tag{3.2.1}$$

3.2.2 Equivalence to Jensen-shannon divergence minimisation

In order to justify the two-player game objective (6), in the following we will show that with infinite capacity for both the generator and the discriminator, the global optimum of the generator is $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$. For a fixed generator $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, we compute the gradient of the GAN objective w.r.t. $\boldsymbol{\phi}$:

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \int \left(\frac{p_{\text{data}}(\boldsymbol{x})}{D_{\boldsymbol{\phi}}(\boldsymbol{x})} - \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x})}{1 - D_{\boldsymbol{\phi}}(\boldsymbol{x})} \right) \nabla_{\boldsymbol{\phi}} D_{\boldsymbol{\phi}}(\boldsymbol{x}) d\boldsymbol{x} \tag{9}$$

Given infinite capacity of the discriminator, setting $\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = 0$ results in

$$\frac{p_{\text{data}}(\boldsymbol{x})}{D_{\boldsymbol{\phi}}(\boldsymbol{x})} = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x})}{1 - D_{\boldsymbol{\phi}}(\boldsymbol{x})} \quad \Rightarrow \quad D_{\boldsymbol{\phi}^*(\boldsymbol{\theta})}(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})}. \tag{10}$$

Pluggin in the optimal discriminator to the GAN objective:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta})) &= \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} \left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} \right] + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x})} \right] \\
&= \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})} \left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{\frac{1}{2}(p_{\boldsymbol{\theta}}(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x}))} \right] + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x})}{\frac{1}{2}(p_{\boldsymbol{\theta}}(\boldsymbol{x}) + p_{\text{data}}(\boldsymbol{x}))} \right] - 2 \log 2 \\
&= \underbrace{2(\frac{1}{2} \text{KL} \left[p_{\text{data}}(\boldsymbol{x}) || \frac{1}{2}(p_{\text{data}}(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x})) \right] + \frac{1}{2} \text{KL} \left[p_{\boldsymbol{\theta}}(\boldsymbol{x}) || \frac{1}{2}(p_{\text{data}}(\boldsymbol{x}) + p_{\boldsymbol{\theta}}(\boldsymbol{x})) \right])}_{:= \text{JS}[p_{\text{data}}(\boldsymbol{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{x})]} - 2 \log 2,
\end{aligned} \tag{11}$$

where $\text{JS}[p_{\text{data}}(\boldsymbol{x}) || p_{\boldsymbol{\theta}}(\boldsymbol{x})]$ is the Jensen-Shannon divergence between $p_{\text{data}}(\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Since Jensen-Shannon divergence is a valid divergence measure, this means with infinite capacity for the generator, $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}^*(\boldsymbol{\theta}))$ is minimised iff. $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$.

This shows that at equilibrium, the generative model is identical to the data distribution, whci justifies the GAN objective as a sensible one to be used for generative modelling.

3.3 Algorithm

3.3.1 Double Loop algorithm

We need to come up with a convergence theorem fro equilibrium.

1. **Inner Loop**:

   With fixed θ, optimise ϕ for a few gradient ascent iterations:

$$\max_\phi \mathbb{E}_{p_{data}(x)}[\log D_\phi(x)] + \mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))] \tag{3.3.1}$$

2. **Outer Loop**:
   With fixed ϕ from the inner loop, optimise θ by **One** gradient descent step:

$$\min_{\theta} \mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))] \tag{3.3.2}$$

3. Loop over (1) and (2) until convergence

In practice, the expectations $\mathbb{E}_{p_{data}(x)}[\cdot]$ and $\mathbb{E}_{p_\theta(x)}[\cdot]$ are estimated by minibatches

$$\mathbb{E}_{p_{data}(x)}[\log D_\phi(x)] \approx \frac{1}{M}\sum_{m=1}^{M} \log D_\phi(x_m), \quad x_m \sim p_{data}(x) \tag{3.3.3}$$

$$\mathbb{E}_{p_\theta(x)}[\log(1 - D_\phi(x))] \approx \frac{1}{K}\sum_{k=1}^{K} \log(1 - D_\phi(G_\theta(z_k))), \quad z_k \sim p(z) \tag{3.3.4}$$

3.3.2 Pseudo-code

Practical implementation for solving $\min_\theta \max_\phi E_{p_{data}(x)}[\log D_\phi(x)] + E_{p_\theta(x)}[\log(1 - D_\phi(x))]$ (pseudo code):

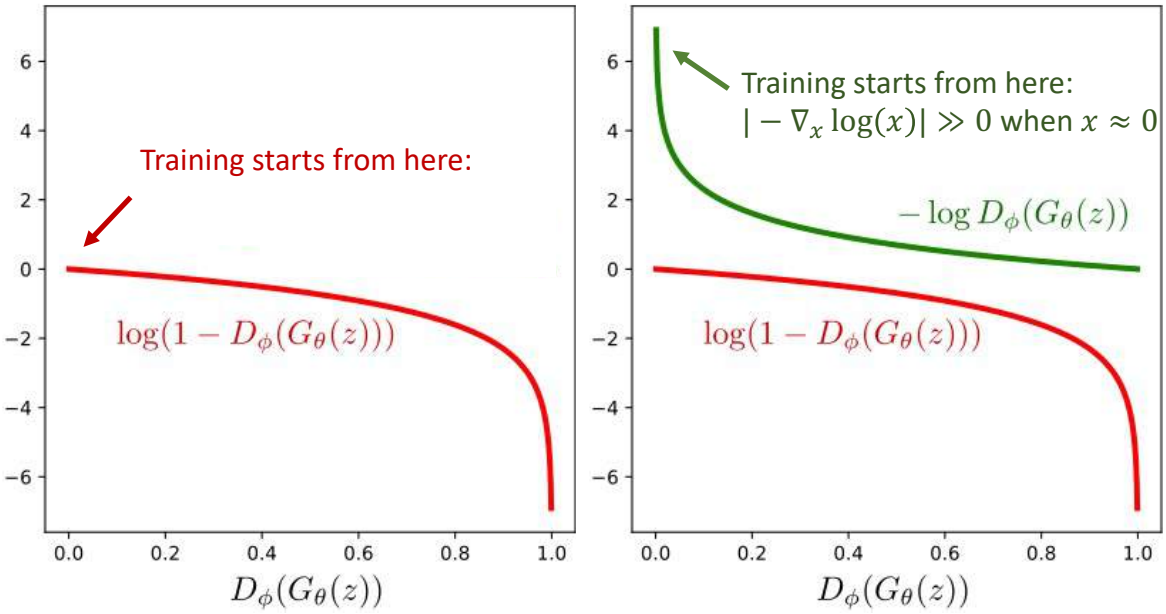
- Initialise θ, ϕ, learning rates γ_D, γ_G, SGD outer-/inner-loop iterations T, K
- For $t = 1, \dots, T$
  - # update discriminator
  - For $i = 1, \dots, K$
    - $z_1, \dots, z_M \sim p(z)$
    - $x_1, \dots, x_M \sim p_{data}(x)$
    - $\phi \leftarrow \phi + \gamma_D \nabla_\phi [\frac{1}{M}\sum_{m=1}^{M} \log D_\phi(x_m) + \frac{1}{M}\sum_{m=1}^{M} \log(1 - D_\phi(G_\theta(z_m)))]$
  - # update generator
  - $z_1, \dots, z_J \sim p(z)$
  - $\tilde{x}_j = G_\theta(z_j), j = 1, \dots, J$
  - $\theta \leftarrow \theta - \gamma_G \nabla_\theta \frac{1}{J}\sum_j^J \log\left(1 - D_\phi(\tilde{x}_j)\right)$

Learning rates γ_D, γ_G & inner-loop iterations K need to be chosen carefully! (otherwise training may be unstable)

This network is pretty unstable because the two networks are playing an adversarial game. The leraning rate, t, and k parameters matter.

3.3.3 Initialisation Problem

Initially, the generator is random, therefore the discirminator can classify with high-accuracy if an image if fake or not: $D_\phi(G_\phi(z)) \approx 0$.



(b) non-saturated loss

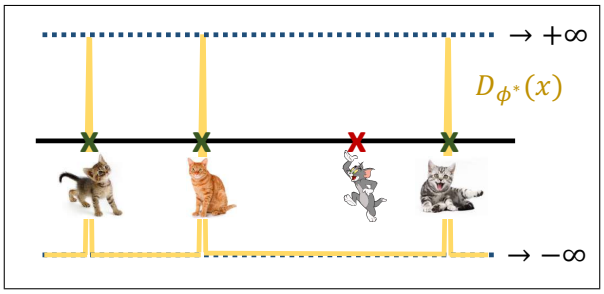
- **(a)**: In the training procedure, the beining of training, the generator starts from the close-to-zero region. And in this region, the gernerator loss is almost flat, therefore it has very little gradient infromtaiton to help with training.
- **(b)**: Instead, maximise hte porbability of the discriminator to make wrong decsiions on fake data that for theis alternative objective, the graident of the lss with respect to X has a very large norm, which provides a strong learningd signal for a generator to imporve the qluaity of the fake images.

3.4 Wasserstein GAN

Discriminator can be used to score the provided inputs. The discriminator should asign high scores to data inputs and low scores to fake inputs.

$$\min_\theta \max_\phi \mathbb{E}_{p_{data}(x)}[D_\phi(x)] - \mathbb{E}_{p_\theta(x)}[D_\phi(x)] \tag{3.4.1}$$

However, this doesn't consider the inifinite capacity of the discriminator network, which should trivially return $D_\phi(x) =_\infty \ if\ x \sim p_{data}(x)\ else\ D_\phi(x) = -\infty$.



This results in a very spiky function, and it is constant around the generated images. This has no use for gradient infromation for generator learning.

To address this issue we need to make sure that we have useful gradient infromation to learn the generator.

3.4.1 Using Wasserstein distnace in GANs

In Wasserstein GANs [Arjovsky et al., 2017], the discriminator is used to parameterise the test function $f := D_{\boldsymbol{\phi}}$, and the Wasserstein distance is used as the loss function for adversarial training:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})], \quad \text{subject to } ||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 \leq 1, \forall \boldsymbol{x} \in \mathbb{R}^d. \tag{24}$$

However, it is impractical to compute the constraint for every $\boldsymbol{x} \in \mathbb{R}^d$. Instead, the point-wise constraint is replaced by the following alternative [Gulrajani et al., 2017]:

$$\mathbb{E}_{\hat{p}(\boldsymbol{x})}[(||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 - 1)^2] = 0, \tag{25}$$

with the auxiliary "interpolation" distribution $\hat{p}(\boldsymbol{x})$ defined by the following generative process:

$$\boldsymbol{x} \sim \hat{p}(\boldsymbol{x}) \quad \Leftrightarrow \quad \boldsymbol{x}_d \sim p_{\text{data}}(\boldsymbol{x}), \boldsymbol{x}_g \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}), \alpha \sim \text{Uniform}([0,1]), \boldsymbol{x} = \alpha \boldsymbol{x}_d + (1-\alpha)\boldsymbol{x}_g. \tag{26}$$

This alternative constraint (25) is justified as follows. Since the original Wasserstein distance objective (24) requires evaluating the discriminator within the supports of $p_{\text{data}}(\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ only, it requires to enforce the $||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 \leq 1$ constraint for $\boldsymbol{x} \in \text{supp}(p_{\text{data}}(\boldsymbol{x})) \cup \text{supp}(p_{\boldsymbol{\theta}}(\boldsymbol{x}))$. Also it can be shown that the optimal discriminator of the objective (24) satisfies $||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 = 1$ for $\boldsymbol{x} \in \text{supp}(p_{\text{data}}(\boldsymbol{x})) \cup \text{supp}(p_{\boldsymbol{\theta}}(\boldsymbol{x}))$. Now the alternative constraint (25) is satisfied iff. $||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 = 1$ for $\boldsymbol{x} \in \text{supp}(\hat{p}(\boldsymbol{x}))$. Given that $\text{supp}(p_{\text{data}}(\boldsymbol{x})) \cup \text{supp}(p_{\boldsymbol{\theta}}(\boldsymbol{x})) \subset \text{supp}(\hat{p}(\boldsymbol{x}))$ by construction, this indicates that the constraint in the Wasserstein distance object (24) is satisfied if the constraint (25) is satisfied. The optimisation of the objective (24) with alternative constraint (25) can be solved by the Lagrange multiplier method, resulting in the WGAN-GP ("Wasserstein GAN with gradient penalty") objective [Gulrajani et al., 2017]:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})] - \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[D_{\boldsymbol{\phi}}(\boldsymbol{x})] + \lambda \mathbb{E}_{\hat{p}(\boldsymbol{x})}[(||\nabla_{\boldsymbol{x}} D_{\boldsymbol{\phi}}(\boldsymbol{x})||_2 - 1)^2]. \tag{27}$$

Here, the discriminator should assign high scores to data inputs and low scores to fake inputs. At the same time, discriminator should be smooth to provide useful graident for learning G_θ.

- Regularised discriminator can be used to score the provided inputs

$$\min_{\theta} \max_{\phi} E_{p_{data}(x)}[D_\phi(x)] - E_{p_\theta(x)}[D_\phi(x)] \text{ subject to } \|D_\phi(\cdot)\|_L \leq 1$$

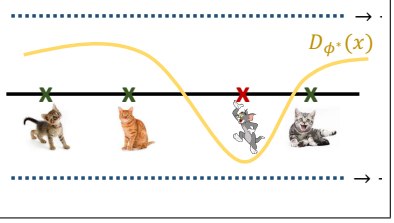
Discriminator should assign high scores to data inputs and low scores to fake inputs
At the same time, discriminator should be smooth to provide useful gradient for learning G_θ

- $\|D_\phi(\cdot)\|_L \leq 1$ is the Lipschitz continuity constraint

$$\|\nabla_x D_\phi(x)\|_2 \leq 1 \text{ for all } x$$

- Equivalent to minimising the Wasserstein distance :

$$W_2(p_{data}(x), p_\theta(x)) := \sup_{\phi:\|D_\phi(\cdot)\|_L \leq 1} E_{p_{data}(x)}[D_\phi(x)] - E_{p_\theta(x)}[D_\phi(x)]$$



To address this problem, we need to make sure that we have useful gradient infromatino to learn the generator. A way to achieve this is to make the discriminator smooth in detail. We firstly, add a contraint that the limshitz-constraint is less than 1, it minimises the wasserstein distance and the supremum is obtained when the discrimintor is one Lipschitz. This creates a stable signal for generating the learning.

- Practical implementation: WGAN-GP

Regulariser to enforce the Lipschitz continuity constraint

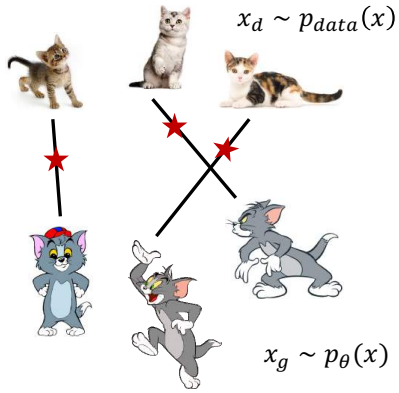
$$\min_{\theta}\max_{\phi} E_{p_{data}(x)}[D_\phi(x)] - E_{p_\theta(x)}[D_\phi(x)] + \lambda E_{\hat{p}(x)}[(\|\nabla_x D_\phi(x)\|_2 - 1)^2]$$

- $\hat{p}(x)$ is defined by the following sampling procedure:

$$x_d \sim p_{data}(x)$$
$$x_g \sim p_\theta(x)$$
$$\alpha \sim Uniform([0,1])$$
$$x = \alpha x_d + (1-\alpha)x_g$$

- Training strategy is similar to the original GAN
  - Double-loop algorithm
  - Minibatch sampling

Arjovsky et al. Wasserstein Generative Adversarial Networks. ICML 2017
Gulrajani et al. Improvedtraining of Wasserstein GANs. NeurIPS 2017



The lipschitz continuity constraint is placed over all possible x inputs, this is intractible. Instead, the WGAN proposed a Lagrange multiplier type of appraoch which minimises the mean suqared error of the difference between the desired and the actual gradient norm

$\hat{p}(x)$ is the auxiliary distribution and is visualised on the rhs figure. First, draw two sets of samples, one for real and fake. Then we run the [UNCLEAR] aginast the real and fake samples and pick in ranodm a point on the line segment connecting the two.

4 Advances and Applications

4.1 Conditional latent variable models

The Goal is to learn a generative model $p_\theta(x|y)$

- x: data to be generated (e.g. an image)
- y: label/info that the generation process is conditioned on (e.g. fur colour)

4.1.1 Idea 1 — Train a set of models for each feature

If $y \in \{1, \ldots, C\}$ we can train a set of models

- $p_\theta(x|y=c) = p_{\theta_c}(x) = \int p_{\theta_c}(x|z)p(z)dz$
- Parameter inefficient: need to train C networks
- Cannot generalise to continuous y

4.1.2 Idea 2 — Make (z, y) as the input of the network

Make both the latent variable z and the conditional variable y as the inputs to the generator network.

- $p_\theta(x|y=c) = p_{\theta_c}(x) = \int p_{\theta_c}(x|z, y=c)p(z)dz$
- Parameter efficeint
- Generalises to continuous y
- Disentangled the learned representation z from the label info y.

4.2 Conditional VAEs

- Training the conditional LVM:

model: $p_\theta(x|y) = \int p_\theta(x|z,y)p(z)dz,$ data: $\{(x_n, y_n)\}_{n=1}^N \sim p_{data}(x,y)$

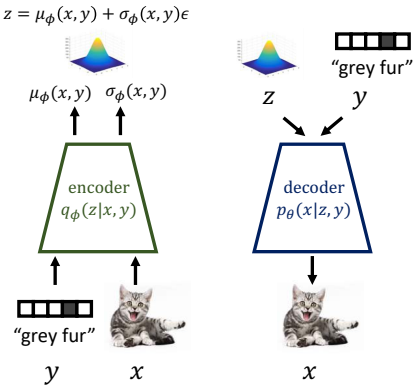
- Maximum Likelihood training (MLE):

$$\max_\theta E_{p_{data}(x,y)}[\log p_\theta(x|y)]$$

- (conditional) variational lower-bound:

$$\log p_\theta(x|y) \geq E_{q_\phi(z|x,y)}[\log p_\theta(x|z,y)] - KL[q_\phi(z|x,y)\|p(z)]$$
$$:= L(x,y,\phi,\theta)$$

$\Rightarrow$ maximise $E_{p_{data}(x,y)}[L(x,y,\phi,\theta)]$ w.r.t. ϕ, θ



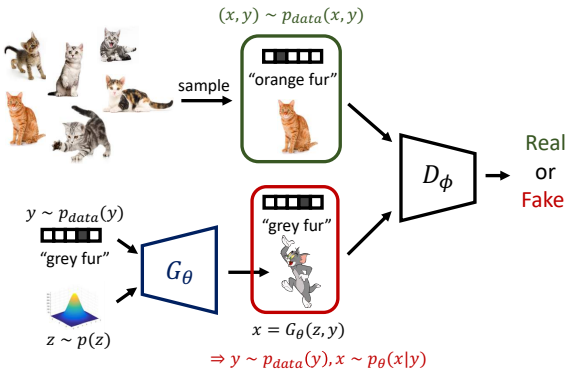
A natural idea is to use maximum likelihood estimation. This, again, is intractible because the conditional distirbution $p(x|y)$ requires integrating out the latent variable z. Therefore, we implore the similar strategy for variational lower-bounds.
This is alsmot identical to the original cases, except that now the encoder needs to take in both X and Y as the inputs to produce the distributional parameters μ and σ. Similarly, the decoder needs Z and Y as input to generate the reconstruction.

- Training the conditional LVM:

model: $p_\theta(x|y) = \int p_\theta(x|z,y)p(z)dz,$ data: $\{(x_n, y_n)\}_{n=1}^N \sim p_{data}(x,y)$

- Adversarial training:
  - Label $(x_n, y_n) \sim p_{data}(x,y)$ as "real"
  - Label $(G_\theta(z,y), y), z \sim p(z)$ as "fake"
  - For fake data, sample label $y \sim p_{data}(y)$

$$\min_\theta \max_\phi \underbrace{E_{p_{data}(x,y)}[\log D_\phi(x,y)]}_{\text{"real"}} + \underbrace{E_{p(z)p_{data}(y)}[\log(1 - D_\phi(G_\theta(z,y),y))]}_{\text{"fake"}}$$

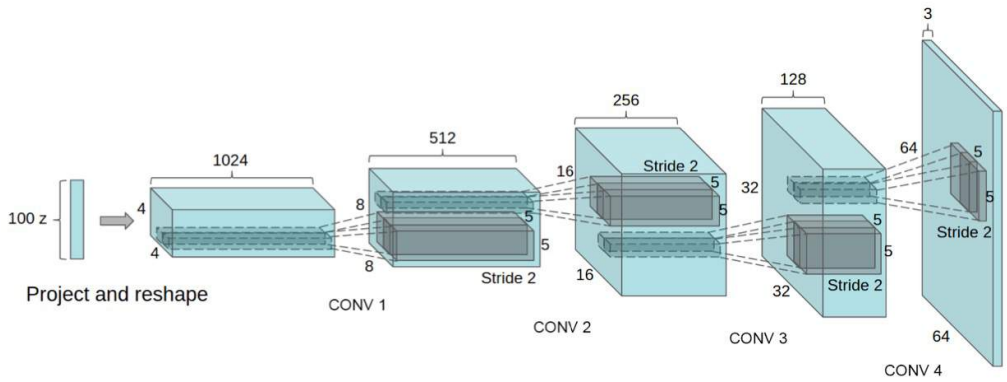


Mirza and Osindero. Conditional Generative Adversarial Networks. arXiv:1411.1784

We can derrive an adverserial training method to train the conditional generative model. In this case we need to modify the task; in the original GAN, we need to distinguish the input images. But now we take in x and label y and assign a label 'real or fake' to this input pair.

4.3 Generative Model Architecture Design

4.3.1 DCGAN

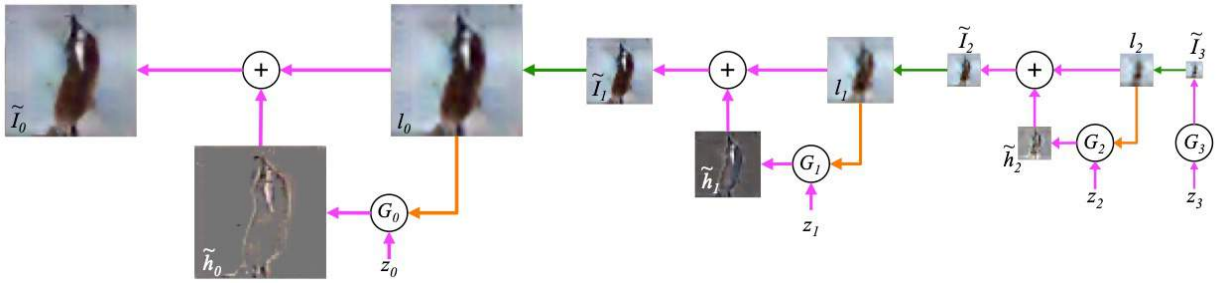


Tricks used in the DCGAN architecture & training:

- Replace pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm.
- Remove fully connected hidden layers for deeper architectures.
- Use LeakyReLU activation in the discriminator for all layers.

First reshape the latent variable vector z into a tensor then apply deconvolutions or transposed convolutions to generate the image. The discriminator follows the similar idea, until the final layer which translates the input into a binary classificaiton logit.

4.3.2 LAPGAN



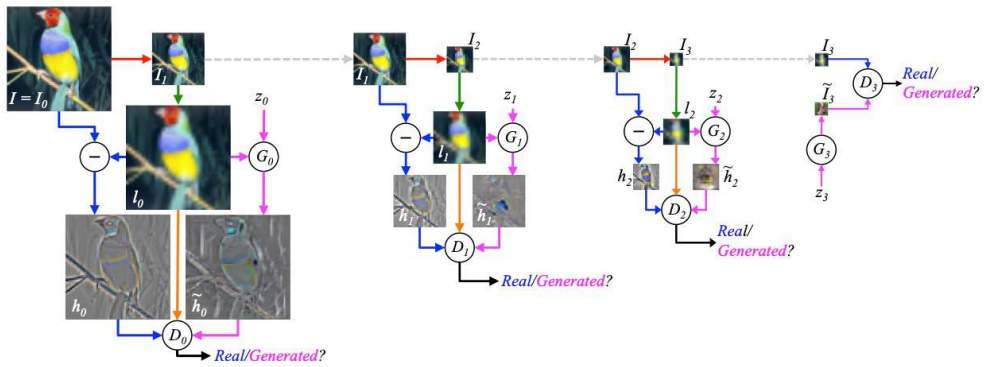
Generator's multi-scale architecture:

- Start generation for low-resolution images: $\tilde{I}_3 = G_3(z_3)$
- Generate higher-resolution images conditioned on the lower-resolution ones:

$$l_i = upscale(\tilde{I}_{i+1}), \qquad \tilde{I}_i = l_i + G_i(z_i, l_i)$$

Construct images in a multi-scale fashion.

Generating the full image in one shot may be hard. It may be easier to generate small scale versions and scale up. It has a set of generators which generate images at different resolutions. the images are then upscaled into higher resolutions, until the point where the blurry image is given in as a conditional input for the next generator to generate a sharper and refined version of it.



LAPGAN's discriminator design:

- Multiple discriminators in use, paired with generators at different resolutions
- At each resolution, check whether the "fine details" generated by G_i matches the real ones:
  - "real" input: $h_i = I_i - l_i, l_i = upscale(I_{i+1}), I_{i+1} = downscale(I_i)$
  - "fake" input: $\tilde{h}_i = G_i(z_i, l_i)$

Training at multiple resolutions

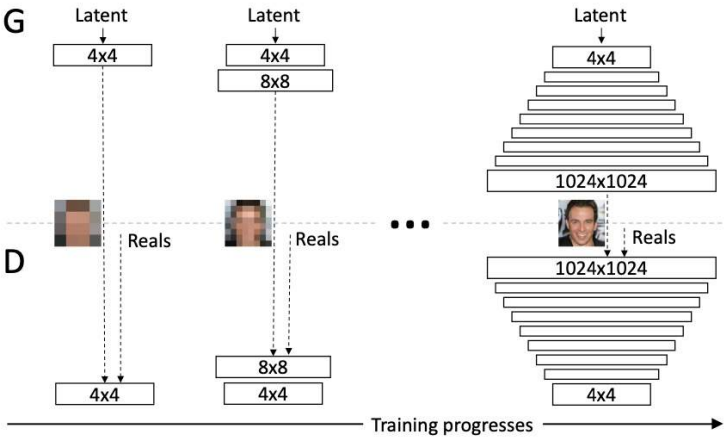
To train multiple generators, introduce multiple discriminators at different scales. The idea is that with higher resoultions on a real image it is striaghtforward to downslae it to generate lower resolution versions. Since upscaling an image is easy, we can produce the difference betwene the original image and the blurred version of it (because the models are incharge of producing refinements).

We repeat this process to train the generator. At the smallest level, we still use the original GAN procedure, however, at this point the iamge resouliton is very low, so the discriminator is less likely to distinguish fake from real — therefore trining may be easier.

4.3.3 Progressive GAN

Progressively building GAN generator and discriminator:

- High-res images downscaled to get training data of low resolutions
- Train a GAN starting from 4x4 images
- Add new layers into generator and discriminator
- Adapt old & new layers by GAN training with 8x8
- Continue with 16x16, 32x32…



trains the generator and discriminator networks in a progressive way. Start with a low resolution and train a gan there. After training at that resolution scale, progress up, and train both old and new with higher resolution image. This procedure is repeated.

4.3.4 StyleGAN

Disentangling different sources of randomness:

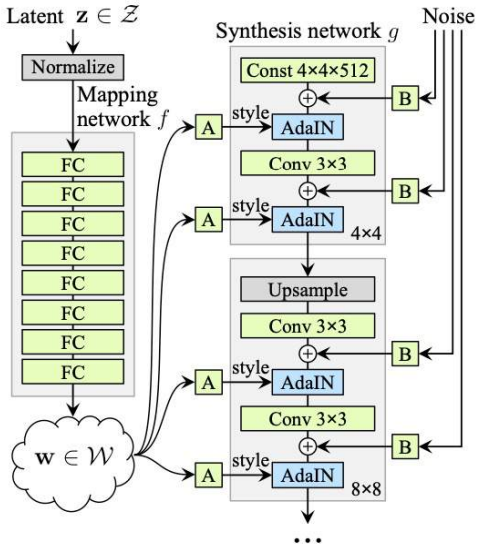
- Latent variable z is transformed to "style" representation w
- This "style" w controls generation at every resolutions
- Fine details generated with noise at different scales

$$y = (y_s, y_b) = A(w)$$

$$x = \text{"upscaled last block output"} + B(\epsilon)$$

$$AdaIN(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

Channel index — normalised feature map for each channel

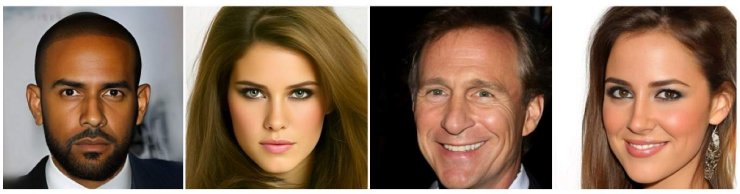


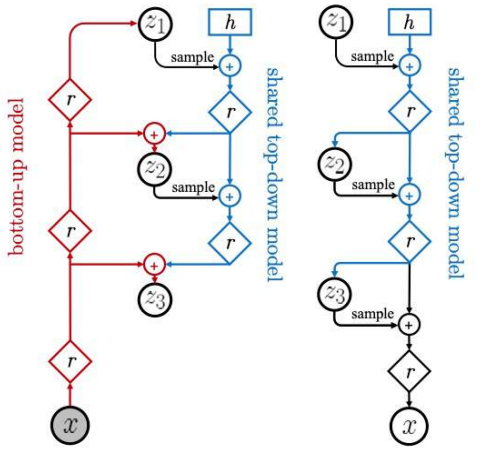
Idea: the latent variable z represents the concept of stypes of the image. The z varibale is transformed into a style representation vector w which is used at each resolution scale of the generator to normalize and shift the feature mass. The noise contributes to a source of randomness in the image.

4.3.5 NVAE — improved VAE image generation

State-of-the-art VAE for image generation (2020):

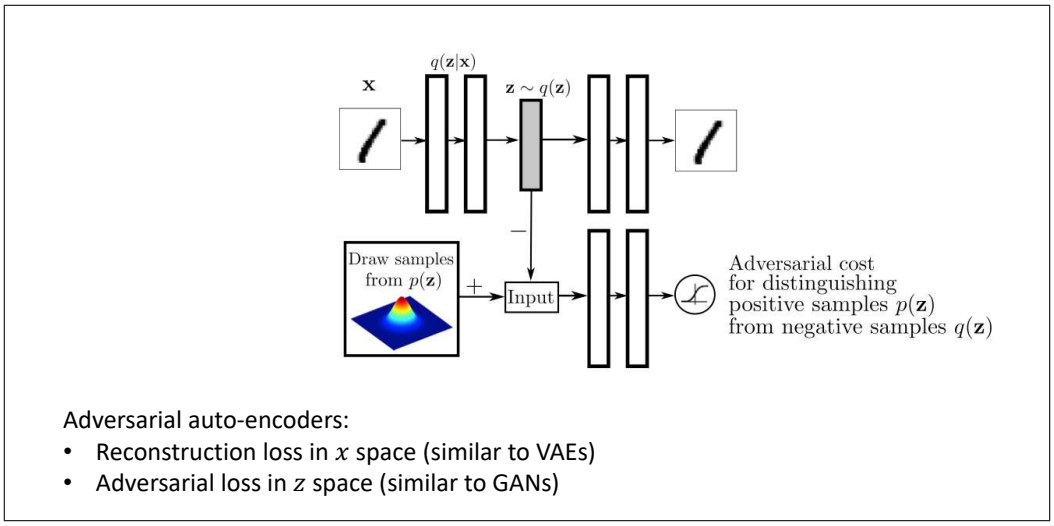
- Hierarchical LVM with multi-scale architecture
- Using residual networks for the r blocks
- BatchNorm in usage
- Improved q distribution design to control the $KL[q(z|x)||p(z)]$ term





The paper provides an imporved q distribution to mitigate the challenge of optimisation of the variational lower bound, which tends to prefer smoother image reconstruction.

4.3.6 Combining VAEs and GANs



There are recent efforts that try to combine VAEs and GANs — achieve best of both worlds. For VAEs it has an autoencoder architecture which allows us to inferr the latent code Z given an input image. The reconstruction error in VAE, is strong signal for generator training.

4.3.7 Summary

- GAN progression:
  - DCGAN – fully convolutional neural networks
  - LAPGAN & Progressive GAN – multi-scale architectures
  - StyleGAN – disentangling sources of randomness
- VAE progression:
  - Hierarchical LVMs
  - Tuning the KL regulariser
  - Deep learning tricks applied
  - Incorporate design ideas from GAN networks

GANs tend to produce sharper images

GAN is often preffered for better visual quality, VAEs preffered for applications that need good likelihood estimates.

4.4 Applications of Generative Models

- Super-resolution
- Image-to-image translation — translations between iamges in two domains X and Y

4.5 Other types of generative models

- Normalising flow — translate a gaussian curve
- Continuous time generative models
- Energy-based models — uses NN to parameterise an energy function - observed data will have low energy, and other points will have high energy.

A Supplied VAE notes

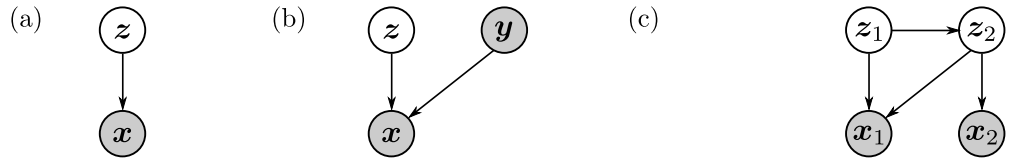
A.1 Prerequisites

A.1.1 Probabilistic graphical models

In machine learning tasks we may define the model as a distribution on a set of random variables with particular dependency structures. Some of the variables might be unobserved as well. Probabilistic graphical models are powerful models that use graphs to describe the dependency structure of the random variables. In particular we consider direct acyclic graphs (DAGs) which are graphs with directed edges and without directed cycles. By assuming Markov properties, DAGs can be used to describe the factorisation structure of the joint distribution. Interested readers are referred to e.g. Chapter 8 of Bishop [2007] for a formal introduction of probabilistic graphical models. For this course we only introduce the principles for reading joint distributions from a DAG (and vice versa). Assuming we are interested in the distribution $p(\boldsymbol{x}_1, ..., \boldsymbol{x}_D)$ for a given DAG with nodes $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_D\}$ and directed edges between them, then the joint distribution is:

$$p(\boldsymbol{x}_1, ..., \boldsymbol{x}_D) = \prod_{i=1}^{D} p(\boldsymbol{x}_i | pa(\boldsymbol{x}_i)), \tag{2}$$

where $pa(\boldsymbol{x}_i) \subset \{\boldsymbol{x}_1, ..., \boldsymbol{x}_D\}$ represents the parent nodes of $\boldsymbol{x}_i$ in the DAG. For a DAG there always exists root node(s) that have no parents (i.e. $pa(\boldsymbol{x}_i) = \emptyset$), and in such case $p(\boldsymbol{x}_i | pa(\boldsymbol{x}_i)) = p(\boldsymbol{x}_i)$. Conversely, given a joint distribution in the form of (2), we can also draw the corresponding DAG by adding arrows from nodes in $pa(\boldsymbol{x}_i)$ to $\boldsymbol{x}_i$. A number of examples are visualised in Figure 1.



$p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$ $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{y})p(\boldsymbol{z})p(\boldsymbol{y})$ $p(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{z}_1, \boldsymbol{z}_2) = p(\boldsymbol{x}_1|\boldsymbol{z}_1, \boldsymbol{z}_2)p(\boldsymbol{x}_2|\boldsymbol{z}_2)p(\boldsymbol{z}_2|\boldsymbol{z}_1)p(\boldsymbol{z}_1)$

Figure 1: Examples of probabilistic graphical models: graphs & the corresponding factorisations of the joint distributions. Shaded nodes represent observed variables and the other nodes represent unobserved/latent variables. Example (a) corresponds to the latent variable model used in VAEs & GANs, and example (b) corresponds to to the latent variable model used in conditional VAEs & GANs where $\mathbf{y}$ represents additional information that the generative model is conditioned on.

A.1.2 Jensen's inequality

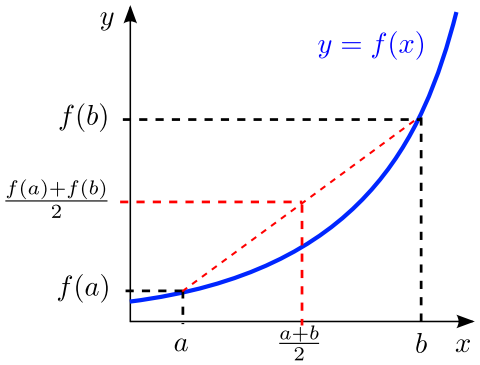
Below we introduce Jensen's inequality as a prerequisite for later discussions on divergences.

Proposition 1. *(Jensen's inequality) If $f : \mathbb{R} \to \mathbb{R}$ is a convex function, then for any distribution $p(x)$,*

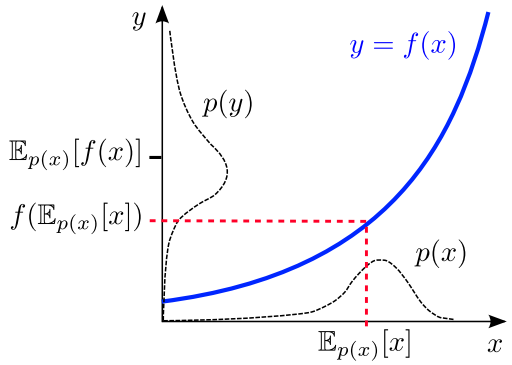
$$\mathbb{E}_{p(x)}[f(x)] \geq f(\mathbb{E}_{p(x)}[x]),$$

with equality holds iff. f is linear or $p(x)$ is a delta measure.

A visual proof is provided in the below figures.



(a) for discrete distributions



(b) for continuous distributions

Jensen's inequality can be generalised to functions formed by compositions of functions. To see this, we first introduce the *law of the unconscious statisticians* (LOTUS) rule:

Proposition 2. *(LOTUS) Given a distribution $p_X(\boldsymbol{x})$ and a function $\mathbf{y} = g(\boldsymbol{x})$ such that $\mathbb{E}_{p_X(\boldsymbol{x})}[g(\boldsymbol{x})] < +\infty$, the random variable $Y = g(X)$ has its distribution $p_Y(\mathbf{y})$ satisfying $\mathbb{E}_{p_Y(\mathbf{y})}[\mathbf{y}] = \mathbb{E}_{p_X(\boldsymbol{x})}[g(\boldsymbol{x})]$.*

Then a generalised version of Jensen's inequality reads as follows.

Proposition 3. *(Generalised Jensen's inequality) If a function $g(\boldsymbol{x})$ maps inputs to scalar outputs in $\mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$ is a convex function, then for any distribution $p_X(\boldsymbol{x})$,*

$$\mathbb{E}_{p_X(\boldsymbol{x})}[f(g(\boldsymbol{x}))] \geq f(\mathbb{E}_{p_X(\boldsymbol{x})}[g(\boldsymbol{x})]),$$

with equality holds iff. f is linear or $p_X(\boldsymbol{x})$ is a delta measure.

Proof.

$$\begin{aligned}
\mathbb{E}_{p_X(\boldsymbol{x})}[f(g(\boldsymbol{x}))] &= \mathbb{E}_{p_Y(y)}[f(y)] && \text{(LOTUS applied to } y = g(\boldsymbol{x})) \\
&\geq f[\mathbb{E}_{p_Y(y)}[y]] && \text{(Jensen's inequality)} \\
&= f(\mathbb{E}_{p_X(\boldsymbol{x})}[g(\boldsymbol{x})]). && \text{(LOTUS applied to } y = g(\boldsymbol{x}))
\end{aligned}$$

□

A.1.3 Analytic KL between factorised Gaussians

To see this, let us assume two factorised distributions $p(\boldsymbol{z}) = \prod_{i=1}^{d} p(z_i)$ and $q(\boldsymbol{z}) = \prod_{i=1}^{d} q(z_i)$. Then the KL divergence from q to p can be written as a sum of KL divergences:

$$\begin{aligned} \mathrm{KL}[q(\boldsymbol{z})||p(\boldsymbol{z})] &= \mathbb{E}_{q(\boldsymbol{z})}\left[\log \frac{\prod_{i=1}^{d} q(z_i)}{\prod_{i=1}^{d} p(z_i)}\right] = \mathbb{E}_{q(\boldsymbol{z})}\left[\sum_{i=1}^{d} \log \frac{q(z_i)}{p(z_i)}\right] \\ &= \sum_{i=1}^{d} \mathbb{E}_{q(z_i)}\left[\log \frac{q(z_i)}{p(z_i)}\right] = \sum_{i=1}^{d} \mathrm{KL}[q(z_i)||p(z_i)]. \end{aligned} \tag{15}$$

Then, assuming each $q(z_i)$ and $p(z_i)$ distributions are Gaussians: $q(z_i) = \mathcal{N}(z_i; \mu_i, \sigma_i^2)$, $p(z_i) = \mathcal{N}(z_i; 0, 1)$, we have the KL divergence as:

$$\begin{aligned} \mathrm{KL}[q(z_i)||p(z_i)] &= \mathbb{E}_{q(z_i)}\left[\log \frac{\frac{1}{\sqrt{2\pi\sigma_i^2}} \exp[-\frac{1}{2\sigma_i^2}(z_i - \mu_i)^2]}{\frac{1}{\sqrt{2\pi}} \exp[-\frac{1}{2} z_i^2]}\right] \\ &= \mathbb{E}_{q(z_i)}\left[-\log \sigma_i - \frac{1}{2\sigma_i^2}(z_i - \mu_i)^2 + \frac{1}{2} z_i^2\right] \\ &= -\log \sigma_i - \frac{1}{2\sigma_i^2}\mathbb{E}_{q(z_i)}\left[(z_i - \mu_i)^2\right] + \frac{1}{2}\mathbb{E}_{q(z_i)}\left[(z_i - \mu_i)^2 - \mu_i^2 + 2\mu_i z_i\right] \\ &= -\log \sigma_i - \frac{1}{2} + \frac{1}{2}[\sigma_i^2 + \mu_i^2]. \end{aligned} \tag{16}$$

Writing $\boldsymbol{\mu_\phi}(\boldsymbol{x}) = [\mu_1, ..., \mu_d]$, $\boldsymbol{\sigma_\phi}(\boldsymbol{x}) = [\sigma_1, ..., \sigma_d]$, we can sum up the KL divergence (16) over $i = 1, ..., d$ and write the resulting $\mathrm{KL}[q(\boldsymbol{z})||p(\boldsymbol{z})]$ as (14). This is done by noticing e.g. $||\boldsymbol{\mu_\phi}(\boldsymbol{x})||_2^2 = \sum_{i=1}^{d} \mu_i^2$ and $\sum_{i=1}^{d} \log \sigma_i = \langle \log \boldsymbol{\sigma_\phi}(\boldsymbol{x}), \mathbf{1} \rangle$.

A.2 Conditional VAE

For conditional generative models, the goal is to generate data (e.g. images) conditioned on additional information. Such additional information can be class labels (which is discrete) or the viewing angle for the image (which is continuous). Mathematically, this corresponds to learning a generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})$ which approximates the data distribution $p_{\text{data}}(\boldsymbol{x}|\mathbf{y})$. Here $\boldsymbol{x}$ is the random variable for data (e.g. images) and $\mathbf{y}$ is the random variable corresponding to the additional information (e.g. label or viewing angle).

For the design of the generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})$, we use a conditional LVM as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})p(\boldsymbol{z})d\boldsymbol{z}, \tag{25}$$

See Figure 1 (b) for a visualisation of the graphical model. Often we set $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\mathbf{0},\mathrm{I})$. If $\boldsymbol{x}$ is continuous, then we can define e.g.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y}) = \mathcal{N}(\boldsymbol{x};G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y}),\sigma^2\mathrm{I}), \tag{26}$$

with $G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y})$ defined by a neural network that takes both $\boldsymbol{z}$ and $\mathbf{y}$ as inputs. Similar to VAEs, learning is done by maximising a variational lower-bound:

$$\boldsymbol{\phi}^*,\boldsymbol{\theta}^* = \arg\max \mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta}),\ \mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta}) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x},\mathbf{y})}[\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})] - \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})||p(\boldsymbol{z})]], \tag{27}$$

$$\mathbb{E}_{p_{\text{data}}(\boldsymbol{x},\mathbf{y})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})] \geq \mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta}). \tag{28}$$

Although in principle the choice of the q distribution is flexible (since the variational lower-bound holds for almost any q distribution satisfying mild conditions, see Section 1.2), using $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})$ and parameterising it with flexible neural networks would return the best posterior approximation. Using Bayes' rule

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})p(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})}, \tag{29}$$

we can show that maximising variational lower-bound w.r.t. q is also equivalent to minimising the KL divergence $\mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})]$:

$$\begin{aligned}
&\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y}) - \left(\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})] - \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})||p(\boldsymbol{z})]\right) \\
=&\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})p(\boldsymbol{z})}\right] \\
=&\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})}\right] = \mathrm{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})].
\end{aligned} \tag{30}$$

Therefore if we were to replace $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\mathbf{y})$ with $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$, then the optimal solution does not return the exact posterior approximation, unless the learned generator is degenerate: $G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y}) = G_{\boldsymbol{\theta}}(\boldsymbol{z})$. In such case the $\mathbf{y}$ information is ignored (i.e. $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$) and the model is no longer a conditional generative model.

A.3 *Practical interpretations & KL annealing

A.3.1 Comparisons with auto-encoders

Looking at the likelihood part of the VAE objective (13), under Gaussian likelihood assumption we have (by using the reparam. trick)

$$\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[-\frac{1}{2\sigma^2}||\boldsymbol{x} - G_{\boldsymbol{\theta}}(T_{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{\epsilon}))||_2^2\right] + \text{const.} \tag{31}$$

On the other hand, an auto-encoder contains a pair of encoder $E_{\boldsymbol{\phi}}(\cdot)$ and decoder $D_{\boldsymbol{\theta}}(\cdot)$ which are trained using e.g. ℓ_2 reconstruction loss:

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \ \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[||\boldsymbol{x} - D_{\boldsymbol{\theta}}(E_{\boldsymbol{\phi}}(\boldsymbol{x}))||_2^2]. \tag{32}$$

Comparing the reconstruction loss of the auto-encoder training objective to (31), we see that VAEs can be viewed from a viewpoint of *stochastic* auto-encoder. Architecture-wise, the main difference is the usage of stochastic encoder $T_{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{\epsilon})$ that injects random noise $\boldsymbol{\epsilon}$ to the encoding of $\boldsymbol{x}$. Training objective-wise, the VAE objective has the extra $\text{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]$ term which regularises the q distribution towards the prior $p(\boldsymbol{z})$. When $p(\boldsymbol{z})$ is non-degenerate (e.g. $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\mathbf{0},\mathbf{I})$) the resulting $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ at optimum is non-degenerate as well, i.e. $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) > \mathbf{0}$.

A.3.2 KL annealing

Practitioners sometimes find that training VAEs with the original variational lower-bound objective (13) leads to under-fitting issues, in such case often the reconstructed images using the model are blurry. A practical strategy to alleviate this is to introduce a "KL annealing" coefficient β and optimise the $\boldsymbol{\theta},\boldsymbol{\phi}$ parameters using the following objective:

$$\boldsymbol{\phi}^*,\boldsymbol{\theta}^* = \arg\max \mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta},\beta), \quad \mathcal{L}(\boldsymbol{\phi},\boldsymbol{\theta},\beta) = \mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] - \beta\text{KL}[q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]}_{:=\mathcal{L}(\boldsymbol{x},\boldsymbol{\phi},\boldsymbol{\theta},\beta)}]. \tag{33}$$

If using $0 < \beta < 1$, this objective introduces less regularisation for the $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to be close to the prior $p(\boldsymbol{z})$. In particular when $\beta = 0$, it results in a stochastic auto-encoder which is trained by the reconstruction loss only. Since stochasticity in $\boldsymbol{z}$ naturally degrades the quality of reconstruction, training with reconstruction loss only will drive $\boldsymbol{\phi}$ towards making $\boldsymbol{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{x}) \to \mathbf{0}$ for any $\boldsymbol{x}$, which also means $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \to \delta(\boldsymbol{z} = \boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}))$. In such case the resulting model is simply an auto-encoder which cannot be used directly as a generative model for new images.

We should also emphasise that for $\beta < 1$, $\mathcal{L}(\boldsymbol{x},\boldsymbol{\phi},\boldsymbol{\theta},\beta)$ is no longer a lower-bound for $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, and the training objective (33) cannot be well justified using (approximate) MLE for learning $p_{\boldsymbol{\theta}}(\boldsymbol{x}) \approx$

$p_{\text{data}}(\boldsymbol{x})$. In fact for small β the learned generative distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$ can be very different from $p_{\text{data}}(\boldsymbol{x})$, again explaining why generation quality can be worse when using such small β values. Therefore β needs to be carefully chosen to achieve the trade of between good reconstruction & good generation. Another strategy is to use different β_t values for different training epochs $t = 0, ..., T$; a recommended recipe is to select increasing values $0 \leq \beta_1 \leq ... \leq \beta_T$.

Sometimes $\beta > 1$ values are also used but for a different purpose. Although there is no theoretical guarantee, existing research shows that empirically, with factorised prior $p(\boldsymbol{z})$ and $\beta > 1$, one can train a VAE to obtain a *disentangled representation*, so that controlled generation can be achieved by varying different dimensions of the $\boldsymbol{z}$ variable [Higgins et al., 2017].

References

Bishop, C. M. (2007). *Pattern Recognition and Machine Learning*. Springer.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *International Conference on Learning Representations*.

Kullback, S. (1959). *Information theory and statistics*. John Wiley & Sons.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286.

B Supplied GAN notes

B.1 Binary Classification

Given a data distribution $p_{\text{data}}(\boldsymbol{x}, y)$ with $y \in \{0, 1\}$, we would like to fit a binary classifier $p_{\boldsymbol{\phi}}(y|\boldsymbol{x})$ to the conditional distribution $p_{\text{data}}(y|\boldsymbol{x})$. A maximum likelihood estimate of the parameters $\boldsymbol{\phi}$ is obtained by solving the following optimisation task:

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x},y)}[\log p_{\boldsymbol{\phi}}(y|\boldsymbol{x})], \tag{1}$$

Assume the dataset is balanced, i.e. $p_{\text{data}}(y) = \text{Bern}(0.5)$, then the above objective is equivalent to

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{p_{\text{data}}(\boldsymbol{x}|y=1)}[\log p_{\boldsymbol{\phi}}(y = 1|\boldsymbol{x})] + \mathbb{E}_{p_{\text{data}}(\boldsymbol{x}|y=0)}[\log(1 - p_{\boldsymbol{\phi}}(y = 1|\boldsymbol{x}))]. \tag{2}$$

The negation of the above maximum likelihood objective is also known as the cross-entropy loss.

B.2 Generative adversarial networks (GANs)

B.2.1 Alternative loss for the generator

In the original GAN paper [Goodfellow et al., 2014] the authors proposed to optimise an alternative "non-saturated" objective for the generator, given a fixed discriminator:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})]. \tag{12}$$

Compared with the original objective (5) which minimises the log probability of making correct predictions, the alternative objective maximises the log probability of making *wrong* predictions. To see how this approach helps, notice that the discriminator often has near-perfect classification performance at the beginning of GAN training (since at this stage the "fake" data quality is bad). In this case $D_{\boldsymbol{\phi}}(\boldsymbol{x}) \approx 0$ for $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x})$. Also assume the generative model is implicitly defined by $\boldsymbol{z} \sim p(\boldsymbol{z}), \boldsymbol{x} = G_{\boldsymbol{\theta}}(\boldsymbol{z})$. Note that $D_{\boldsymbol{\phi}}(\boldsymbol{x})$ is often defined using sigmoid activation $\text{sigmoid}(t) = (1 + exp[-t])^{-1}$ at the last layer, i.e. $D_{\boldsymbol{\phi}}(\boldsymbol{x}) = \text{sigmoid}(d_{\boldsymbol{\phi}}(\boldsymbol{x}))$ with $d_{\boldsymbol{\phi}}(\boldsymbol{x})$ parameterised by a neural network. This means $D_{\boldsymbol{\phi}}(\boldsymbol{x}) \approx 0$ when $d_{\boldsymbol{\phi}}(\boldsymbol{x}) \to -\infty$ (so at the beginning of GAN training $d_{\boldsymbol{\phi}}(\boldsymbol{x}) \to -\infty$ for $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x})$). Therefore the gradients of the two objectives w.r.t. $\boldsymbol{\theta}$ are

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log(1-D_{\boldsymbol{\phi}}(\boldsymbol{x}))] = -\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{z})}[\log(1+\exp[d_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))])] = -\mathbb{E}_{p(\boldsymbol{z})}[\underbrace{D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))}_{\approx 0} \nabla_{\boldsymbol{\theta}} d_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))], \tag{13}$$

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log D_{\boldsymbol{\phi}}(\boldsymbol{x})] = -\nabla_{\boldsymbol{\theta}}\mathbb{E}_{p(\boldsymbol{z})}[\log(1+\exp[-d_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))])] = \mathbb{E}_{p(\boldsymbol{z})}[\underbrace{(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z})))}_{\approx 1} \nabla_{\boldsymbol{\theta}} d_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}))]. \tag{14}$$

It is clear that the alternative objective addresses the vanishing gradient problem of the original one (5) at the beginning of training, hence the name "non-saturated objective".

Another justification of the alternative objective is provided by deriving the optimal solution of the generator, given the optimal discriminator. Define $f(t) = \log(1 + t^{-1}) - \log 2$, in which $f(t)$ is convex and $f(1) = 0$. Then we can define an f-divergence [Csiszár, 1963; Morimoto, 1963; Ali and Silvey, 1966] as

$$\begin{aligned} \mathrm{D}_f[p_{\boldsymbol{\theta}}(\boldsymbol{x})||p_{\text{data}}(\boldsymbol{x})] &:= \int p_{\boldsymbol{\theta}}(\boldsymbol{x}) f\left(\frac{p_{\text{data}}(\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right) d\boldsymbol{x} \\ &= \int p_{\boldsymbol{\theta}}(\boldsymbol{x}) \log\left(1 + \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})}\right) d\boldsymbol{x} - \log 2 \\ &= -\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\log D_{\boldsymbol{\phi}^*(\boldsymbol{\theta})}(\boldsymbol{x})] - \log 2. \end{aligned} \tag{15}$$

This shows that maximising the alternative "non-saturated objective" is equivalent to minimising an f-divergence between the model and the data distribution. Therefore again with infinite capacity of the generator, the optimal solution of the generative model is $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = p_{\text{data}}(\boldsymbol{x})$.

B.3 Conditional GAN

Similar to conditional VAEs, conditional GAN is a particular parameterisation of $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})$ which approximates the data distribution $p_{\text{data}}(\boldsymbol{x}|\mathbf{y})$. Again $\boldsymbol{x}$ is the random variable for data and $\mathbf{y}$ is the random variable corresponding to the additional information. For the design of the generative model $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})$, we use a conditional LVM as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})p(\boldsymbol{z})d\boldsymbol{z}, \tag{16}$$

and often we set $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z};\mathbf{0},\mathrm{I})$. But different from conditional VAE which explicitly specifies the distribution form of $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y})$, for conditional GAN it is defined implicitly by the following sampling process:

$$\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z},\mathbf{y}) \quad \Leftrightarrow \quad \boldsymbol{z} \sim p(\boldsymbol{z}), \boldsymbol{x} = G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y}), \tag{17}$$

with $G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y})$ defined by a neural network that takes both $\boldsymbol{z}$ and $\mathbf{y}$ as inputs. Similar to GANs, learning is done by optimising an adversarial objective:

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \; \mathbb{E}_{p_{\text{data}}(\boldsymbol{x},\mathbf{y})}[\log D_{\boldsymbol{\phi}}(\boldsymbol{x},\mathbf{y})] + \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})p_{\text{data}}(\mathbf{y})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x},\mathbf{y}))]. \tag{18}$$

In practice the component related to the generator parameters $\boldsymbol{\theta}$ is computed by

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})p_{\text{data}}(\mathbf{y})}[\log(1 - D_{\boldsymbol{\phi}}(\boldsymbol{x},\mathbf{y}))] \approx \log(1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z},\mathbf{y}),\mathbf{y})), \quad \boldsymbol{z} \sim p(\boldsymbol{z}), \mathbf{y} \sim p_{\text{data}}(\mathbf{y}). \tag{19}$$

Using similar techniques, one can derive the optimal discriminator for a fixed generative model with parameter $\boldsymbol{\theta}$:

$$D_{\boldsymbol{\phi}^*(\boldsymbol{\theta})}(\boldsymbol{x},\mathbf{y}) = \frac{p_{\text{data}}(\boldsymbol{x},\mathbf{y})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})p_{\text{data}}(\mathbf{y}) + p_{\text{data}}(\boldsymbol{x},\mathbf{y})}, \tag{20}$$

and with the optimal discriminator, maximising $\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi})$ w.r.t. $\boldsymbol{\theta}$ is equivalent to minimising the Jensen-Shannon divergence $\text{JS}[p_{\text{data}}(\boldsymbol{x},\mathbf{y})||p_{\boldsymbol{\theta}}(\boldsymbol{x}|\mathbf{y})p_{\text{data}}(\mathbf{y})]$.

B.4 *Wasserstein GAN

B.4.1 Wassterstein distance

Wasserstein distance is a key concept developed in optimal transport, which aims at finding the lowest cost approach to transform a distribution to another [Villani, 2008]. The dual form of the Wasserstein distance is defined by taking the optimal *test functions* from $\mathcal{F} = \{f : ||f||_L \leq 1\}$, the set of 1-Lipschitz functions:

$$W_2[p,q] = \sup_{||f||_L \leq 1} \mathbb{E}_p[f(\boldsymbol{x})] - \mathbb{E}_q[f(\boldsymbol{x})]. \tag{21}$$

As a reminder, a function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be l-Lipschitz (denoted as $||f||_L \leq l$) if

$$|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leq l||\boldsymbol{x}_1 - \boldsymbol{x}_2||_2, \quad \forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d. \tag{22}$$

If f is differentiable everywhere, then

$$||f||_L \leq 1 \quad \Leftrightarrow \quad ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \leq 1, \forall \boldsymbol{x} \in \mathbb{R}^d. \tag{23}$$

B.4.2 Integral probability metrics (IPMs)

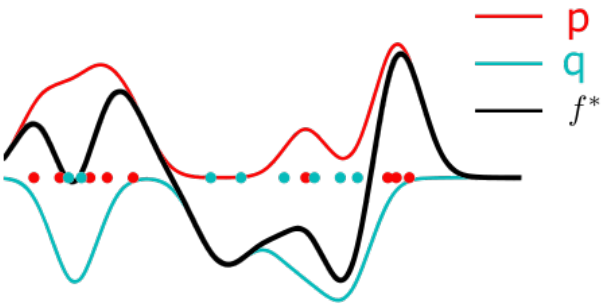
Wasserstein distance is an instance of a family of distance measures between distributions, named integral probability metrics (IPMs).

Definition 1. *(Integral probability metric (IPM)) Given a set of test functions $\mathcal{F}$, consider the following quantity:*

$$\mathrm{D}[p, q] = \sup_{f \in \mathcal{F}} |\mathbb{E}_p[f(\boldsymbol{x})] - \mathbb{E}_q[f(\boldsymbol{x})]|, \tag{28}$$

where $|\cdot|$ denotes a norm in the output space of f. If $\mathcal{F}$ is sufficiently large such that $\mathrm{D}[p, q] = 0$ iff. $p = q$, then $\mathrm{D}[p, q]$ is said to be an integral probability metric defined by the test functions in $\mathcal{F}$.

To provide an intuition of IPMs, consider a strategy of comparing distributions by comparing their *moments*, e.g. mean, variance, kurtosis, etc. Loosely speaking, if two distributions p and q have the same moments for all orders then p and q should be identical.[1] Therefore, to check whether p and q are identical or not, one can find the best moment, or in a broader sense the best test function



f that can distinguish p from q the most, and if such optimal test function still fails to distinguish between p and q, then the two distributions p and q are identical.[2]

The intuition is further visualised in the above figure.[3] We see from the visualisation that the optimal test function f^* takes positive values in the region where $p(\boldsymbol{x}) > q(\boldsymbol{x})$ and vise versa. In other words, the optimal test function tells us more than whether $p = q$ or not; it also provides information on *how* p and q differ from each other. This is a useful property for IPMs for applications in adversarial learning: as f^* describes in detail the difference between p and q, we can optimise the q distribution in a guided way towards approximating the target distribution p. Indeed various versions of IPMs have been used as optimisation objectives in the GAN literature, e.g. see Li et al. [2017]; Mroueh and Sercu [2017]; Mroueh et al. [2018].

References

[1] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[2] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *CoRR* abs/1906.02691 (2019). arXiv: 1906.02691. URL: http://arxiv.org/abs/1906.02691.