Lecture Notes

PROBABILISTIC GENERATIVE MODELS

This version: 22nd February 2026

Latest version: github.com/felipe-tobar/Probabilistic-Generative-Models

Felipe Tobar
Department of Mathematics
Imperial College London

f.tobar@imperial.ac.uk
https://www.ma.ic.ac.uk/~ft410/

Preface

This notes are under development for 2026.

Felipe Tobar,
London,
January 2026.

Contents

Week 1

Foundations

1.1 Introduction

A Probabilistic generative model (PGM), or simply, a GM, is a methodology for generating data. In general, the PGM is constructed and adjusted using observations with the aim to synthesise samples with the same statistical properties of the available observations. The *probabilistic* nature of the PGMs studied in this course follows from the fact that the available data will be considered to be realisations of an underlying random variable (RV), e.g., X.

In this sense, the probability distribution of X, denoted $P_X(x)$, as well as its probability density function (pdf) $p_X(x)$ will be central to the study of PGMs. In particular, targeting the pdf is one way of constructing PGMs, in which case the whole PGM paradigm becomes equivalent to the classical statistical modelling approach. However, as we will see in the course, enforcing the sought-after PGM to have an explicit parametric pdf can be rather restrictive.

Throughout the course, we will consider a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, with 3 RVs given by the following measurable maps:

$X : \Omega \to \mathcal{X}$	$Y : \Omega \to \mathcal{Y}$	$Z : \Omega \to \mathcal{Z}$
(observed input)	(observed output)	(latent variable)

Remark 1.1.

Not all three RVs will be present in all our settings. For instance, in classification there is no justification for the latent variable Z (in general), while in clustering, there is no need for X. However, we build the general setup here for formality.

We equip $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ with their Borel σ-algebras $\mathcal{B}(\mathcal{X}), \mathcal{B}(\mathcal{Y}), \mathcal{B}(\mathcal{Z})$, and consider the product measurable space

$$(\mathcal{X} \times \mathcal{Y} \times \mathcal{Z},\ \mathcal{B}(\mathcal{X}) \otimes \mathcal{B}(\mathcal{Y}) \otimes \mathcal{B}(\mathcal{Z})).$$

Then the joint random variable $(X, Y, Z) : \Omega \to \mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ is measurable with respect to $\mathcal{F}$ and the above product σ-algebra.

Furthermore, we will assume that the law of (X, Y, Z) is absolutely continuous with respect to the product base measure on $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$ (e.g. Lebesgue measure when $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \subseteq \mathbb{R}^d$), and hence admits a joint density $p(x, y, z)$. That is, for all $A \in \mathcal{B}(\mathcal{X}), B \in \mathcal{B}(\mathcal{Y}), C \in$

$\mathcal{B}(\mathcal{Z})$,

$$\mathbb{P}\left(X \in A, Y \in B, Z \in C\right) = \int_{A\times B\times C} p(x,y,z)\mathrm{d}x\mathrm{d}y\mathrm{d}z. \tag{1.1}$$

We will also assume that all relevant marginals and conditionals admit densities (with respect to the corresponding base measures), e.g. $p(x,y)$, $p(y|x)$, $p(z|x,y)$, etc.

1.2 Discriminative versus generative

The generative approach aims to characterise the complete generative distribution $p(x,y,z)$, whereas, in some application-specific cases, only the discriminative model, e.g., $p(y|x)$, is needed. Let us examine the following example.

Example 1.1 (Generative and discriminative views of binary classification)**.**
Consider the binary classification problem, where, given an observation $X = x$, one needs to estimate its label Y. A discriminative model would directly parametrise $\mathbb{P}(Y|X = x)$. Since this is a binary classification case, without loss of generality, we can assume $Y \in \{0,1\}$, and model $\mathbb{P}(Y = 1|X = x)$, since $\mathbb{P}(Y = 0|X = x) = 1 - \mathbb{P}(Y = 1|X = x)$. A model for this probability only needs to map $x \in \mathbb{R}^d \to \mathbb{P}(Y = 1|X = x) \in [0,1]$. For instance, a reasonable candidate for this is

$$\mathbb{P}(Y = 1|X = x) = \frac{1}{1 + e^{-\theta^\top x}} \tag{1.2}$$

which is known as the logistic regression.

Conversely, in a generative approach, we aim to model the joint probability $p(Y = y, X = x)$. Modelling this distribution is not easy, however, observe that we can factorise it as

$$p(Y = y, X = x) = p(X = x|Y = y)p(Y = y), \tag{1.3}$$

which yields a pair of much more intuitive distributions to model:

- the class probability $p(Y = y) = (\pi, 1-\pi), \pi \in [0,1]$, and
- the class-conditional probability $p(X = x|Y = y)$, given by a two distributions over $\mathcal{X}$, denoted f_{θ_0} and f_{θ_1}.

Therefore, the classifier is

$$\begin{aligned} p(Y = 1|X = x) &= \frac{p(X = x|Y = 1)p(Y = 1)}{p(X = x)} \\ &= \frac{1}{1 + e^{-\log\left(\frac{\pi}{1-\pi}\frac{f_{\theta_1}(x)}{f_{\theta_0}(x)}\right)}}. \end{aligned} \tag{1.4}$$

Exercise 1.1.
Evaluate eq. (1.4) for $f_{\theta_0} = \mathcal{N}(\mu_0, \Sigma_0)$ and $f_{\theta_1} = \mathcal{N}(\mu_1, \Sigma_1)$. What happens when $\Sigma_0 = \Sigma_1$?

1.3 The pushforward measure

Despite the abundant collection of well-studied statistical models, in some scenarios we can construct a more ad hoc model by applying an appropriate transformation.

Definition 1.1.
Consider a RV $X \in \mathcal{X}$ with measure P_X, and a nonlinear map $T : \mathcal{X} \to \mathcal{X}$. The measure of the transformed RV $T(X)$ is known as the *push forward measure* of P_X through T, and it is denoted by $T_{\#}P_X$

Remark 1.2.
The transformations considered in the course will be such that the pushforward measure has a density. With a slight abuse of notation, we will denote this density as $T_{\#}p_X$.

Example 1.2 (Discrete pushforward)**.**
Let X be a discrete random variable taking values in $\{1, 2, 3\}$ with

$$\mathbb{P}(X = 1) = 0.2, \quad \mathbb{P}(X = 2) = 0.5, \quad \mathbb{P}(X = 3) = 0.3.$$

Define the map $T : \{1, 2, 3\} \to \{a, b\}$ by

$$T(1) = a, \qquad T(2) = b, \qquad T(3) = b.$$

Then the pushforward $T_{\#}\mathbb{P}$ satisfies

$$(T_{\#}\mathbb{P})(\{a\}) = 0.2, \qquad (T_{\#}\mathbb{P})(\{b\}) = 0.8.$$

For an illustration see Fig. 1.1.

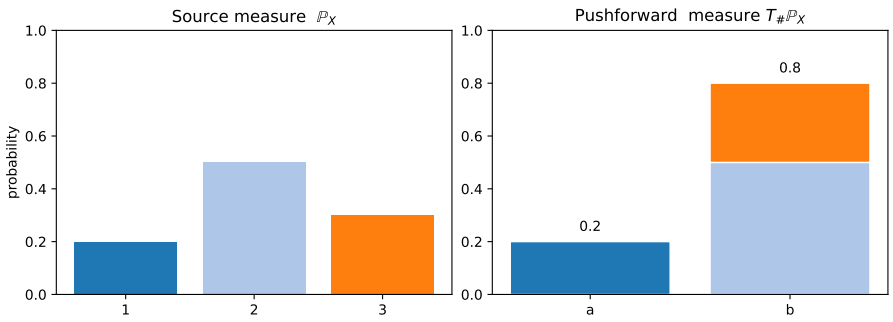

Figure 1.1: Source and pushforward distributions: discrete example.

Example 1.3 (Continuous pushforward)**.**
Let $X \sim \mathcal{N}(0, 1)$ on $\mathbb{R}$ and define $T(x) = x^2$. The pushforward $T_{\#}\mathbb{P}$ is the law of $Y = T(X)$, supported on $\mathbb{R}_+$. Its density is given by

$$p_Y(y) = \frac{1}{\sqrt{2\pi y}} \exp\left(-\frac{y}{2}\right), \qquad y > 0.$$

For an illustration of this case and other related examples, see Fig. 1.2.

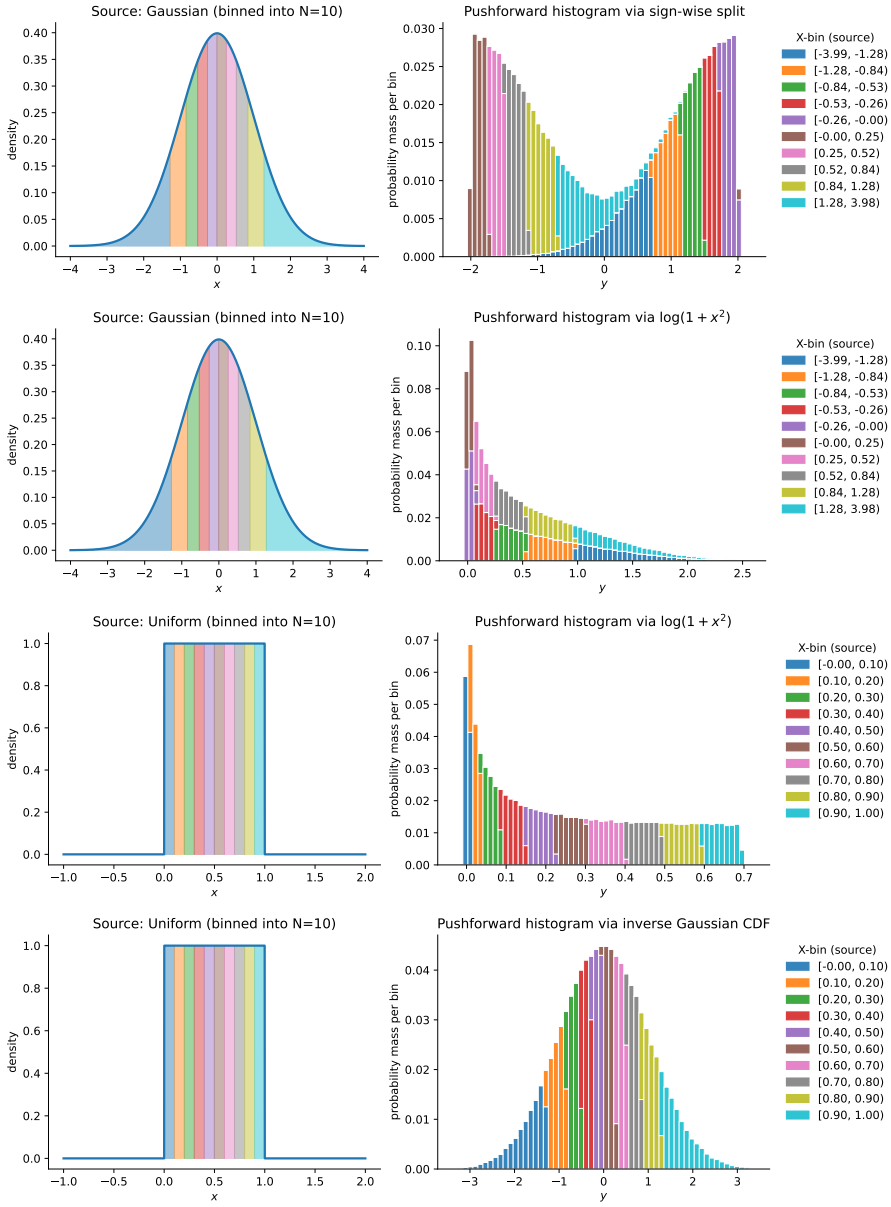

Figure 1.2: Source and target distributions: continuous examples

In general, for arbitrary source distributions and maps it is difficult to compute the target density in closed form, at least in the continuous case. For the specific case of differentiable and invertible maps T, the following theorem given a recipe to compute $p_{T(X)}$

Theorem 1.1 (Change of variable)**.**
Consider two RVs $X, Y \in \mathbb{R}^d$, such that $Y = T(X)$, where $T : \mathbb{R}^d \to \mathbb{R}^d$ is a C^1 diffeomorphism. If X and Y have densities p_X and p_Y respectively, then

$$p_Y(y) = p_X(T^{-1}(y)) \left| \det \nabla_y T^{-1}(y) \right|, \tag{1.5}$$

where $\nabla_y T^{-1}(y)$ is the Jacobian of the inverse map.

Remark 1.3.

Though the above result provides a closed-form expression for the pushforward measure only when T is a C^1 diffeomorphism (continuously differentiable with an the inverse having the same property), we can transform a source RV X into a target RV T with any measurable map. This is because

$$(T_{\#}P_X)(A) \;=\; P_X(T^{-1}(A)), \qquad \forall A \in \mathcal{B}(\mathcal{X}). \tag{1.6}$$

Though in general the pdf of T will not be available in closed form.

1.4 Likelihood-based training

Maximum likelihood (ML) is going to be the canonical methodology for training our PGMs, and, as we will see next, it will recover other forms of training criteria in particular cases.

Consider a PGM for the RV Y, with density $p_\theta(y)$, where $\theta \in \Theta$ denotes the model parameter. Also, consider the realisations of Y given by $y_1, y_2, \ldots, y_N$.

Definition 1.2 (Likelihood function)**.**
The likelihood of the parameter θ is the function $L : \Theta \to \mathbb{R}_+$ given by the probability density function of Y evaluated on the observations. That is,

$$L(\theta) = p_\theta(y_1, y_2, \ldots, y_N). \tag{1.7}$$

NB: We abused notation above stating the joint pdf for the observations.

Remark 1.4.
Very important: the likelihood function is not a probability/density function, as it is a function of the parameter. In particular, it is not true that $\int_\Theta L(\theta)\mathrm{d}\theta$ is one.

Definition 1.3 (Maximum likelihood estimator)**.**
The ML estimator is given by

$$\theta_{ML} = \arg\max L(\theta). \tag{1.8}$$

Remark 1.5.
In general (but, importantly, not always) we will consider i.i.d observations, in which case the likelihood factorises as $L(\theta) = \prod_{n=1}^N p_\theta(y_n)$. Furthermore, when optimising the likelihood we will consider the log-likelihood instead; in the i.i.d. case, this is

$$l(\theta) = \log L(\theta) = \sum_{n=1}^{N} \log p_\theta(y_n). \tag{1.9}$$

Example 1.4 (Gaussian linear regression)**.**
Let us consider the PGM given by

$$Y|x \sim \mathcal{N}(ax, \sigma^2), a, x \in \mathbb{R}, \sigma^2 \in \mathbb{R}_+. \tag{1.10}$$

This is equivalent to $Y = ax + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$. The parameters in this setting are $\theta = (a, \sigma^2)$. Now consider the observations $\{(x_n, y_n)\}_{n=1}^N$.

Since $p(y_n|x_n) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(y_n - ax_n)^2\right)$, we can write the log-likelihood as

$$l(\theta) = \sum_{n=1}^{N} \frac{-1}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2}(y_n - ax_n)^2. \tag{1.11}$$

The optimal (a, σ^2) can be found in closed form using the first order optimality conditions.

Remark 1.6.
Observe that optimising eq. (1.11) recovers the least squares solution.

Example 1.5 (Binary classification)**.**
Consider observations $\{(x_n, y_n)\}_{n=1}^N \subset \mathbb{R}^d \times \{0, 1\}$ from a binary classification setting. Model the classifier as

$$p_\theta(y = 1|x) = \sigma(s(x)), \tag{1.12}$$

where $\sigma(s(x)) = \frac{1}{1+e^{-s(x)}}$, and $s : \mathbb{R}^d \to \mathbb{R}$ is a feature extractor (e.g., $s(x) = a^\top x + b$). Assuming that the observations are i.i.d., we have

$$L(\theta) = \prod_{n=1}^{N} p(y_n|x_n) = \prod_{n=1}^{N} \sigma(s(x_n))^{y_n}(1 - \sigma(s(x_n)))^{1-y_n}, \tag{1.13}$$

and equivalently

$$l(\theta) = \sum_{n=1}^{N} y_n \log \sigma(s(x_n)) + (1 - y_n) \log(1 - \sigma(s(x_n))). \tag{1.14}$$

Does this expression seem familiar? If not, we will find out soon what this is.

Example 1.6 (Clustering)**.**
Consider a set of observations $\{x_n\}_{n=1}^N \in \mathbb{R}^d$ and implement a clustering algorithm. We will assume that there are $K \in \mathbb{N}$ clusters, each specified by a density $p_k, k = 1, \ldots, K$; this means that the probability of the RV X coming from the k-th cluster is $\mathbb{P}(X \in C_k) = \pi_k$, where $\forall k, 0 \leq \pi_k \leq 1$ and $\sum_{k=1}^K \pi_k = 1$.

This is a mixture model, with density $p(x) = \sum_{k=1}^K \pi_k p_k(x)$, and parameters given by the cluster probabilities π_k and the parameters of the densities $p_k = p_{\theta_k}$. The log-likelihood is

$$l(\theta) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k p_k(x_n) \tag{1.15}$$

Note that there are two issues associated to optimising eq. (1.15).

- We do not recover the cluster assignments.
- The problem is ill-posed. E.g., if $p_k = \mathcal{N}(\mu_k, \Sigma_k)$, which is the usual choice, we can set $\mu_k = x_n, \Sigma_k = 0$ which gives $l = \infty$.

We can explicitly model the cluster assignments by introducing a collection of latent random variables $Z_{nk} \in \{0, 1\}, \sum_{k=1}^{K} Z_{nk} = 1$. That is,

$$Z_{nk} = 1 \iff x_n \in C_k, \tag{1.16}$$

where $p(Z_{nk} = 1) = \pi_k$, and more generally $p(z_{nk}) = \prod_{k=1}^{K} \pi_k^{z_{nk}}$.

The model for the observed and latent variable can be expressed as $p(x, z) = p(x \mid z)p(z)$. Since the conditional densities $p(x_n|z_{nk}) = \prod_{k=1}^{K}(p_k(x_n))^{z_{nk}}$, we can express the **complete-data likelihood** by

$$\begin{aligned} l_c(\theta) &= \log \prod_{n=1}^{N} \prod_{k=1}^{K} (\pi_k p_k(x_n))^{z_{nk}} \\ &= \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk}(\log \pi_k + \log p_k(x_n)). \end{aligned} \tag{1.17}$$

Good and bad news: this objective is now theoretically feasible to optimise but impractical since we do not have access to the latent cluster assignments $\{z_{nk}\}_{nk}$.

A workaround to this is to replace the cluster assignments z_{nk}, by their conditional expectations $\gamma_{nk} = \mathbb{E}\,(z_{nk}|x_n, \theta)$. That is,

$$\begin{aligned} \gamma_{nk} &= 1 * \mathbb{P}(z_{nk} = 1|x_n, \theta) + 0 * \mathbb{P}(z_{nk} = 0|x_n, \theta) \\ &= \frac{\pi_k p_k(x_n \mid \theta)}{\sum_{j=1}^{K} \pi_j p_j(x_n \mid \theta)}. \end{aligned} \tag{1.18}$$

Then, we can perform an iterative procedure by: i) optimising $l_c(\theta)$ using γ_{nk} above, and ii) computing γ_{nk} using $\theta^\star = \arg\max \mathbb{E}_{z|x,\theta}[l_c(\theta)]$.

Lastly, even though this latent-variable formulation does not solve the ill-posedness problem of the likelihood, it provides a more controlled optimisation strategy that can be monitored (and regularised) to avoid degenerate optima in practice.

The maximum likelihood estimator (MLE) satisfies several important theoretical properties:

- **Consistency:** Under the assumption that the statistical model is *identifiable*—i.e., different parameter values correspond to different probability distributions—the MLE converges to the true parameter as the number of observations grows. Intuitively, maximising the likelihood asymptotically minimises the Kullback–Leibler divergence between the true distribution and the distribution induced by a candidate parameter.

- **Equivariance:** If $\hat{\theta}_{\text{MLE}}$ is the MLE of θ, then for any transformation g, the MLE of $g(\theta)$ is $g(\hat{\theta}_{\text{MLE}})$. This property allows us to compute MLEs under reparametrisations directly.

- **Asymptotic normality:** For large sample sizes, the MLE is approximately normally distributed around the true parameter with covariance matrix given by the

inverse Fisher information. Formally,

$$\sqrt{n}(\hat{\theta}_{\text{MLE}} - \theta) \xrightarrow{d} \mathcal{N}(0, I(\theta)^{-1}),$$

where $I(\theta)$ is the Fisher information matrix.

- **Asymptotic efficiency:** As a consequence of asymptotic normality, the MLE achieves the Cramér–Rao lower bound for the variance in the limit of large n, making it asymptotically optimal among unbiased estimators.

In practice, these properties justify the widespread use of the MLE: it not only converges to the true parameter under mild assumptions, but also allows for straightforward reparametrisations and provides an estimator with minimal asymptotic variance.

1.5 A brief intro to information theory

NB: This section is based on Chapter 6 of (Murphy, 2022)

Motivation. Let us consider a discrete RV $X \in \{1, 2, \dots, K\}$ with pmf p_X. Observe that $-\log p_X(x)$ represents a measure of *information* gained from obtaining the value a as a sample of X. Now consider a communication channel $A \to B$, where A is transmitting samples of X to B. When B received the samples, its *average information* can be expressed as

$$H(X) = -\sum_{x=1}^{K} p_X(x) \log p_X(x). \tag{1.19}$$

This quantity is known as *entropy* and—in connection with thermodynamics—it represent a measure of disorder or un-predictability of X.

NB: We will use $H(X)$ and $H(p_X)$ interchangeably.
NB: We will usually denote $H(X) = -\mathbb{E}\left(\log p_X\right) = \mathbb{E}\left(\log \frac{1}{p_X}\right)$.

Clearly, $H(X) \geq 0$ with equality achieved for an RV that has always the same outcome with $p_X(x) = 1$. This is the deterministic, predictable, case. To revise further properties, let us recall the following result.

Jensen's Inequality Let $\phi : \mathbb{R} \to \mathbb{R}$ be a *convex* function and let X be a random variable such that $\mathbb{E}[|X|] < \infty$. Then

$$\phi\Big(\mathbb{E}[X]\Big) \;\leq\; \mathbb{E}\Big[\phi(X)\Big]. \tag{1.20}$$

since $\log(\cdots)$ is *concave*, the inequality is reversed and we have:

$$\mathbb{E}[\log X] \;\leq\; \log \mathbb{E}[X]. \tag{1.21}$$

Remark 1.7.

Equality in eq. (1.20) is only achieved when either ϕ is affine, or X is constant almost surely, that is, $\mathbb{P}\left(X = c\right) = 1$. As a consequence, equality in eq. (1.21) is

only achieved when X is constant almost surely (when the argument of the logarithm does not depend on x)

Keep this result in mind, as it will be used throughout the module.

Using Jensen on the definition of the entropy, we have

$$H(X) = \mathbb{E}\left(\log \frac{1}{p}\right) \leq \log \mathbb{E}\left(\frac{1}{p}\right) = \log \sum \frac{1}{p} p = \log K. \tag{1.22}$$

This directly implies that the uniform distribution over $\{1, 2, \ldots, K\}$ has the largest entropy, since

$$H(U_{1:K}) = \mathbb{E}\left(\log \frac{1}{1/K}\right) = \log K. \tag{1.23}$$

Example 1.7 (Bernoulli distribution)**.**
Consider $X \sim p_X(x) = \theta^x (1-\theta)^{1-x}, \theta \in [0, 1]$. The entropy is given by $H(X) = -\theta \log \theta - (1-\theta) \log(1-\theta)$. Figure 1.3 shows this function.

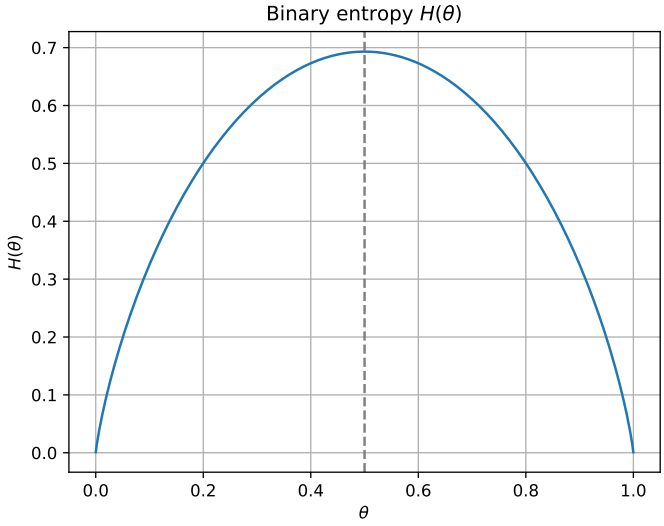

Figure 1.3: Entropy for Bernoulli.

The entropy, in addition to being a measure of disorder, can be understood as the cost of the optimal for of compression. Here, think of a compression strategy using symbols $s_1, s_2, \ldots$ with increasing size (or storage cost). For instance, think of storage using logical gates, meaning that these symbols are (equivalent to)

$$s_1 = 0, s_2 = 1, s_3 = 10, s_4 = 11, \ldots \tag{1.24}$$

where storing more and more symbols becomes increasingly expensive. The compression strategy is then to assign each outcome of X with a symbol s_i. Intuitively, the optimal compression would assign s_i to the i-th most frequent value in $\{1, 2, \ldots., K\}$, meaning that the *cost of storage* precisely grows precisely with $\log \frac{1}{p_X}$.

As a consequence, the average message size is the entropy $H(X)$, and thus can be understood as the cost of this compression strategy.

Now let us go back to our communication channel $A \to B$, where the receiver in B now mistakenly believes that $X \sim q_X$. B's estimated entropy, or averaged information, would be

$$H_{CE}(p_X, q_X) = -\sum_{x=1}^{K} p_X(x) \log q_X(x). \tag{1.25}$$

Following the same rationale as above, this can be interpreted as the cost of compressing the sequence $x_1, x_2, x_3, \ldots$ using q_x.

This quantity is known as the cross-entropy between p_X and q_X. Note that this quantity is not symmetric.

It is relevant to study how $H_{CE}(p_X, q_X)$ and $H(P) = H_{CE}(p_X, p_X)$ relate to one another, in particular if one of them is (always) larger than the other.

Let us see:

$$\begin{aligned} H(p) - H_{CE}(p_X, q_X) &= \sum_x p(x) \log \frac{q(x)}{p(x)} && (1.26)\\ &\overset{\text{Jensen's}}{\leq} \log \sum_x \cancel{p(x)} \frac{q(x)}{\cancel{p(x)}} && (1.27)\\ &= \log 1 = 0. && (1.28) \end{aligned}$$

Therefore, $H(p) \leq H(p, q)$, with equality only achieved when $p = q$, as per Remark 1.7.

Remark 1.8.
Minimising the cross-entropy wrt to one of its arguments is precisely an attempt to match $p = q$.

The use of the entropy/cross-entropy that is going to be more relevant in our case is via its application to continuous RVs. This extension is

$$\begin{aligned} H(p) &= -\int_{\mathcal{X}} p(x) \log p(x) \mathrm{d}x && (1.29)\\ H(p, q) &= -\int_{\mathcal{X}} p(x) \log q(x) \mathrm{d}x && (1.30)\\ & && (1.31) \end{aligned}$$

Remark 1.9.
Unlike its discrete formulation, $H(p)$ can be positive, negative or zero for continuous RVs.

Example 1.8 (Continuous unifom distribution)**.**
Consider a RV $X \sim U_{[a,b]}$, its entropy is

$$H(U_{[a,b]}) = -\int_a^b \frac{1}{b-a} \log \frac{1}{b-a} \mathrm{d}x = \log(b-a).$$

and can be zero (resp. negative) if $b - a = 1$ (resp. $b - a \leq 1$).

The difference between the entropy and crossentropy is of critical relevance in this module

(and life). We will recall a relevant definition first.

Definition 1.4 (Absolute Continuity)**.**
Let $(\Omega, \mathcal{F})$ be a measurable space and let P and Q be probability measures on it. We say that P is *absolutely continuous* with respect to Q, denoted $P \ll Q$, if for every $A \in \mathcal{F}$,

$$Q(A) = 0 \implies P(A) = 0.$$

Remark 1.10.
If $P \ll Q$, and Q admits a density q, P admits a density p satisfying $p(x) = 0$ whenever $q(x) = 0$.

Definition 1.5 (Kullback–Leibler Divergence)**.**
Let P and Q be probability measures on a measurable space $(\Omega, \mathcal{F})$ such that $P \ll Q$. If P and Q admit densities p and q with respect to a common base measure (e.g. Lebesgue measure), then

$$\mathrm{KL}(p \,\|\, q) \;=\; \mathbb{E}_p\!\left[\log \frac{p(X)}{q(X)}\right] \;=\; \int p(x)\, \log \frac{p(x)}{q(x)}\, \mathrm{d}x.$$

Definition 1.6 (Discrete KL Divergence)**.**
For discrete distributions,

$$\mathrm{KL}(p \,\|\, q) \;=\; \sum_x p(x)\, \log \frac{p(x)}{q(x)}.$$

Remark 1.11.
Notice that the KL divergence is always positive:

$$\mathrm{KL}(p \,\|\, q) = H(p, q) - H(p) \geq 0. \tag{1.32}$$

The KL is a *divergence*, i.e., a function that quantifies how far p is form q that is i) always positive, and ii) $\mathrm{KL}(p \,\|\, q) = 0 \iff p = q$ (identify of the indiscernible). However, note that the KL is not a distance, since

- is not symmetric
- does not have triangle inequality.

Critically, the $\mathrm{KL}(p \,\|\, q)$ is only defined when $P \ll Q$.

1.6 KL divergence as a metric to compare p and q

In the continuous case, we are interested in understanding what type of convergence KL gives. Let us consider, other two divergences:

- $L_1\,(p \| q) = \int_{\mathcal{X}} |p(x) - q(x)| \mathrm{d}x$

- $\chi^2(p\|q) = \int_{\mathcal{X}} \frac{|p(x)-q(x)|^2}{q(x)} \mathrm{d}x$.

Example 1.9 (KL versus L_1)**.**
Consider $p(x) = \text{Uniform}(0,1)$ and $q_n(x) = \mathbb{1}x \in [0, 1/n]e^{-n} + \mathbb{1}x \in [1/n, 1]c_n$, with $c_n \geq 0$ so that q integrates 1 ($n > 1$).

Let us first compute c_n explicitly. Since q_n must integrate to one, we require $\int_0^{1/n} e^{-n}\,\mathrm{d}x + \int_{1/n}^1 c_n\,\mathrm{d}x = 1$, which yields

$$\frac{1}{n}e^{-n} + \left(1 - \frac{1}{n}\right)c_n = 1.$$

Solving for c_n, we obtain

$$c_n = \frac{1 - \frac{1}{n}e^{-n}}{1 - \frac{1}{n}} = \frac{n - e^{-n}}{n-1}. \tag{1.33}$$

Note that here, we have

$$L_1(p\|q_n) = \int_0^{1/n} |e^{-n} - 1|\mathrm{d}x + \int_{1/n}^1 |c_n - 1|\mathrm{d}x = \frac{|e^{-n}-1|}{n} + \frac{(n-1)|c_n - 1|}{n} \tag{1.34}$$

$$\mathrm{KL}(p\|q_n) = \int_0^{1/n} -\log e^{-n}\mathrm{d}x + \int_{1/n}^1 -\log c_n \mathrm{d}x = 1 + \frac{-(n-1)}{n}\log c_n. \tag{1.35}$$

Now take $n \to \infty$. From eq. (1.33) we can see that c_n converges to 1. Therefore, $L_1(p\|q) \to 0$. However, note that $\mathrm{KL}(p\|q) \to 1$.

Example 1.10 (KL versus χ^2)**.**
Consider now $p_\epsilon = (1-\epsilon, \epsilon)$ and $q_\epsilon = (1-\epsilon^2, \epsilon^2)$ two Bernoulli distribution with different parameters. Again, we have:

$$\chi^2(p_\epsilon\|q_\epsilon) = \frac{||1 - \epsilon - 1 + \epsilon^2||^2}{1-\epsilon^2} + \frac{||\epsilon - \epsilon^2||^2}{\epsilon^2} = \frac{||\epsilon^2 - \epsilon||^2}{1-\epsilon^2} + ||1-\epsilon||^2 \tag{1.36}$$

$$\mathrm{KL}(p_\epsilon\|q_\epsilon) = (1-\epsilon)\log\frac{1-\epsilon}{1-\epsilon^2} + \epsilon\log\frac{\epsilon}{\epsilon^2} = (1-\epsilon)\log\frac{1}{1+\epsilon} + \epsilon\log\frac{1}{\epsilon}. \tag{1.37}$$

This time, taking $\epsilon \to 0$, we have $\mathrm{KL}(p_\epsilon\|q_\epsilon) \to 0$ (l'Hôpital's rule), but $\chi^2(p_\epsilon\|q_\epsilon) \to 1$

Remark 1.12.
The objective of these examples is to show that under different divergences, one can have different criteria of convergence. In the first case, q_n converges to p under L_1, but not under KL. In the second case, p_ϵ converges to q_ϵ under KL but not under ξ^2. This give a sense of *hierarchy* across divergences, where some are said to induce stronger topologies than others. The stronger the topology, the more demanding the conditions for convergence (or fewer sequences are admitted to converge). In general, we consider KL as one of the stronger divergences (but there are some that are even stronger as we just saw).

Direct versus reverse KL. Since $\mathrm{KL}(p\|q)$ is not symmetric, we are interested in studying the *reverse* divergence $\mathrm{KL}(q\|p)$ and understanding how it relates its *direct*

counterpart.

Since $P \ll Q$ is needed for $\mathrm{KL}\,(p\|q)$, it is required that $P \gg Q$ for $\mathrm{KL}\,(q\|p)$. This gives intuition of $\mathrm{KL}\,(p\|q)$ as a metric assessing how well q approximates p (and not viceversa); this is because if there is a set $A \subset \mathcal{X}$ such that $Q(A) = 0$ and $P(A) > 0$ is strongly penalised, unlike the opposite case.

Let us see a numerical example.

Example 1.11 (Asymmetry of the KL between two Gaussians)**.**
The KL divergence between two Gaussians is

$$\mathrm{KL}\left(\mathcal{N}(\mu_0, \sigma_0^2)\middle\|\mathcal{N}(\mu_1, \sigma_1^2)\right) = \log\frac{\sigma_1}{\sigma_0} + \frac{\sigma_0^2 + (\mu_0 - \mu_1)^2}{2\sigma_1^2} - \frac{1}{2}. \tag{1.38}$$

Let us consider $p = \mathcal{N}(0, 1)$ and $q = \mathcal{N}(0, v^2)$, and evaluate

- $\mathrm{KL}\,(p\|q) = \frac{1}{2}(\log v^2 + v^{-2} - 1)$
- $\mathrm{KL}\,(q\|p) = \frac{1}{2}(-\log v^2 + v^2 - 1)$.

Fig. 1.4 shows these functions, note how the penalisation strength depends on the direction.

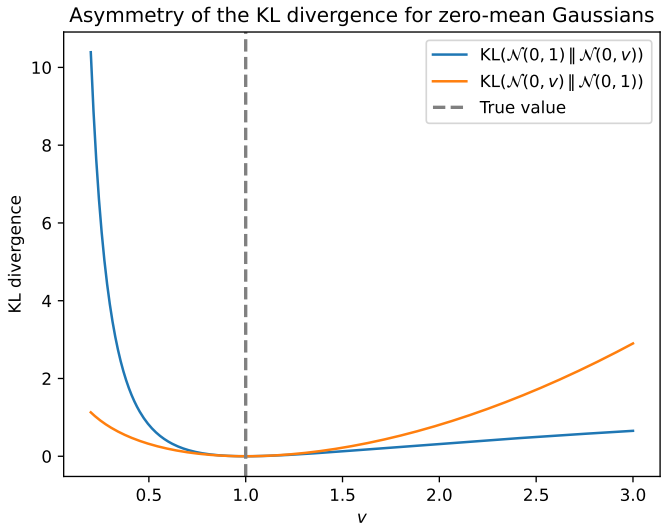

Figure 1.4: Direct and reverse KL for zero mean Gaussians as a function of the variance.

Example 1.12 (KL gradient flow)**.**
Let us now find the approximating q via optimisation for the above example. We can do this via optimisation. Differentiating eq. (1.38) wrt to μ_1 and σ_1, we have

$$\nabla_{\mu_1}\mathrm{KL}\,(\mathcal{N}(\mu_0, \sigma_0)\|\mathcal{N}(\mu_1, \sigma_1)) = \frac{(\mu_1 - \mu_0)}{\sigma_1^2} \tag{1.39}$$

$$\nabla_{\sigma_1}\mathrm{KL}\,(\mathcal{N}(\mu_0, \sigma_0)\|\mathcal{N}(\mu_1, \sigma_1)) = \frac{1}{\sigma_1} - \frac{\sigma_0^2}{\sigma_1^3} = \frac{\sigma_1^2 - \sigma_0^2}{\sigma_1^3} \tag{1.40}$$

Where it is clear the this is minimised for $q = p$. Additionally, we can build the gradient descent rule:

$$\mu_n \to \mu_n - \eta_\mu \frac{(\mu_n - \mu_0)}{\sigma_n^2} \tag{1.41}$$

$$\sigma_n \to \sigma_n - \eta_\sigma \frac{\sigma_n^2 - \sigma_0^2}{\sigma_n^3} \tag{1.42}$$

Figure 1.5 implements these recursions.

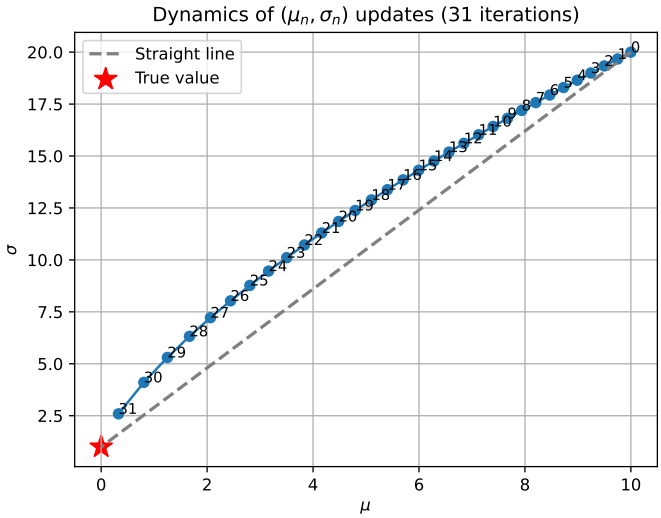

Figure 1.5: KL gradient flow between two Gaussians.

KL and maximum likelihood. Let us now return to the setting of the learning problem. Consider a true model given by p and iid observations $x_1, x_2, \ldots, x_N \sim p(x)$. We could use the KL to look for the best approximator of p with a given family of candidate models $\{q_\theta, \theta \in \Theta\}$. That is,

$$\theta^\star = \arg\min \mathrm{KL}\left(p \| q_\theta\right). \tag{1.43}$$

Though it sounds good, this is unfeasible in practice since p is unknown. However, note that there is a workaround to that. We can write the above expression as

$$\theta^\star = \arg\min \int_{\mathcal{X}} p(x) \log p(x) \mathrm{d}x - \int_{\mathcal{X}} p(x) \log q_\theta(x) \mathrm{d}x \tag{1.44}$$

$$= \arg\max \mathbb{E}_p[\log q_\theta(x)] \tag{1.45}$$

$$\approx \arg\max \sum_{x_i} \log q_\theta(x_i), \tag{1.46}$$

where the last approximation is due to Monte Carlo. This reveals that maximum likelihood is (asymptotically) equivalent to minimising the KL divergence between the model candidate and the (true) empirical distribution.

1.7 Concluding remarks

As we will throughout the module, when designing/choosing a PGM, we will be faced with the following scenarios.

- **Explicit-likelihood models:** These include classical statistical models such as Gaussians, exponentials, Bernoulli, Xi^2, log-normal, but also combinations or transformations that we construct as long as they have an explicit likelihood. These are Gaussian mixtures, piece-wise defined distributions, and amy pushforward model constructed through an invertible transformation so that its density can be calculated via the change of variable theorem. These last models are referred to as normalising flows, as they assume a Gaussian source measure.
- **Implicit models:** As the name suggests, these models are only implicitly defined via a data-generating mechanism, usually involving sampling. For instance, take a RV $Z \in \mathbb{R}^d \sim \mathcal{N}(0, I_d)$ and construct $X = T_\theta(Z)$, where T_θ is collection of neural networks which are sequentially applied to Z with the aim to replicate learnt dynamics that make X *flow* towards the desired distribution. Depending on the parametrisation, these models are known as score-based models, diffusion models, or flow matching.

1.7.1 Suggested exercises

1. **Generative vs discriminative modelling (theory).** Let (X, Y) be random variables with joint distribution $p(x, y)$.
   (a) Define what is meant by a *generative model* and a *discriminative model*.
   (b) Show how a generative model can be used to construct a classifier.
   (c) Discuss one advantage and one limitation of generative modelling relative to discriminative modelling.
2. **Information-theoretic objectives (theory).** Let p be a data-generating distribution and q_θ a parametric model.
   (a) Define the entropy $H(p)$, cross-entropy $H(p, q_\theta)$, and KL divergence $\mathrm{KL}(p\|q_\theta)$.
   (b) Show that maximising the log-likelihood of data sampled from p is equivalent to minimising $\mathrm{KL}(p\|q_\theta)$.
   (c) Explain why minimising $\mathrm{KL}(p\|q_\theta)$ and $\mathrm{KL}(q_\theta\|p)$ lead to qualitatively different approximations.
3. **Maximum likelihood density estimation (coursework).** You are given samples from a one-dimensional distribution.
   (a) Fit a Gaussian model by maximum likelihood.
   (b) Fit a mixture of Gaussians by maximum likelihood.
   (c) Empirically compare the learned models using log-likelihood and visual inspection.

4. **Forward and reverse KL divergence (coursework).** Consider approximating a multimodal target distribution using a unimodal Gaussian.

   (a) Numerically minimise $\mathrm{KL}(p\|q)$ and $\mathrm{KL}(q\|p)$.

   (b) Visualise the resulting solutions.

   (c) Explain the observed behaviour using the geometry of the KL divergence.

Week 2

Expectation Maximisation

NB: This is based on Chapter 9 of (Bishop, 2006).

2.1 Gaussian mixtures

Consider a dataset $\{x_1, x_2, \ldots, x : N\} \subset \mathbb{R}^d$. Our task is to partition this set into $K \in \mathbb{N}$ subsets; we will consider K known for now. Intuitively, each subset of points—referred to as a *cluster*—should share some common or similar patterns; a formal definition of similarity in this case will be ignored until needed.

A natural solution for this segmentation problem is to define K prototypes denoted $\{\mu_1, \mu_2, \ldots, \mu_K\} \subset \mathbb{R}^d$ and determine the assignment of each datapoint x_n to each prototype μ_k, according to a given criterion.

To solve this optimisation problem, we can define a set of binary variables $\{r_{nk}\}_{nk} \subset \{0, 1\}$, where

$$r_{nk} = 1 \iff x_n \text{ is assigned to } \mu_k. \tag{2.1}$$

Then, using the Euclidean distance as similarity criterion, the objective can be written as

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2. \tag{2.2}$$

The solution to the clustering problem obtained via the minimisation of the loss in eq. (2.2) is

$$r_{nk} = \begin{cases} 1, & \text{if } k = \arg\min_j ||x_n - \mu_j||^2, \\ 0, & \text{if not.} \end{cases} \tag{2.3}$$

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}. \tag{2.4}$$

This solution can be calculated by iteratively implementing the above equations, which is known as the k-means algorithm.

Remark 2.1.

Observe that the k-means recursion ensures convergence in a finite number of steps:

this is because eq. (2.3) defines a discrete number of solutions, and (2.4) is the global optima for a given $\{r_{nk}\}_{nk}$.

There are some known drawbacks of k-means, for instance

- Speed: computing the assignment variables has a cost $\mathcal{O}(NK)$.
- It depends on the Euclidean distance that might not be robust to outliers
- It only provide hard assignments, not a degree of *responsibility*.

2.2 The Gaussian mixture model

Let us consider the following PGM:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k), \tag{2.5}$$

where $0 \leq \pi_k \leq 1, \sum_{k=1}^{K} \pi_k = 1$, $\mu_k \in \mathbb{R}^d$ and $\Sigma_k \in \mathbb{R}^{d \times d}$.

This formulation seems to be an improved clustering model wrt K-means, since it—at least—allows for learning the shape (variance) of each cluster and admits the definition of a soft assignment variable.

However, note that the likelihood of this models is ill posed. Denoting the parameters by $\theta = \{\pi_{1:K}, \mu_{1:K}, \Sigma_{1:K}\}$ and the i.i.d. data $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$, the log-likelihood is given by

$$l(\theta) = \log p(\mathbf{x}|\theta) = \log \prod_{n=1}^{N} p(x_n|\theta) = \sum_{n=1}^{N} \log p(x_n|\theta) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k). \tag{2.6}$$

This objective can reach an infinite value if a Gaussian component is assigned to a single datapoint with a vanishing variance. Additionally, for each possible assignment, there are $K!$ different solutions that provide such assignment.

We will derive an equivalent formulation to the PGM above that admits a more interpretable and *stepwise* training procedure. To this end, let us introduce a set of K latent variables $\{z_k\} \subset \{0, 1\}$, $\sum_{k=1}^{K} z_k = 1$. We can write

$$p(x, z) = p(x|z)p(z). \tag{2.7}$$

Also, defining $p(z_k = 1) = \pi_k$, we can express the pmf/pdf:

$$p(z) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{2.8}$$

$$p(x|z) = \prod_{k=1}^{K} \mathcal{N}(\mu_k, \Sigma_k)^{z_k}, \tag{2.9}$$

with the marginal pdf over x as

$$p(x) = \sum_{k=1}^{K} p(z_k)p(x|z_k) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x; \mu_k, \Sigma_k). \tag{2.10}$$

Thus, showing that the formulations are equivalent.

In this formulation, let us define the *responsibilities* of the k-th component to explain the observation x given by

$$\gamma(z_k) \stackrel{\text{def}}{=} p(z_k = 1|x) = \frac{p(x|z_k = 1)p(z_k = 1)}{\sum_{j=1}^{K} p(x|z_j = 1)p(z_j = 1)} \tag{2.11}$$

$$= \frac{\pi_k \mathcal{N}(x; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x; \mu_j, \Sigma_j)}. \tag{2.12}$$

Remark 2.2.
The latent-variable formulation of the GMM allows for direct sampling from that PGM: first sample $z \sim p(z) = \prod_{k=1}^{K}$, and then sample $x \sim p(x|z) = \prod_{k=1}^{K} \mathcal{N}(\mu_k, \Sigma_k)^{z_k}$. This is known as *ancestral sampling*.

2.3 Expectation Maximisation for GMMs

We will introduce a learning approach for PGMs that features a latent variable called Expectation Maximisation (EM). We will first present it in the particular case of the GMM model, and then in the general case.

The first order optimality conditions for the log-likelihood in eq. (2.6) give

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n \tag{2.13}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^\top \tag{2.14}$$

$$\pi_k = \frac{N_k}{N}, \tag{2.15}$$

where we have defined the effective number of samples per component as $N_k = \sum_{n=1}^{N} \gamma(z_{nk})$.

Exercise 2.1.
Derive eqs. (2.13)-(2.15)

Remark 2.3.
Observe how the optimal mean and variance of each component is a weighted average of all the data points, where the weights are proportional to the responsibility (contribution) of that component to generation of the sample. Also, note that eqs. (2.13)-(2.15) can be considered as the soft-assignment version of the K-means solutions, with the additional flexibility of having an learnable expression for the shape of the clusters.

Eqs. (2.13)-(2.15) do not provide a direct closed-form solution, since they depend on the responsibilities $\gamma(z_{nk})$ which are functions of all the parameters. However, they can still be implemented in following the steps:

(E) Compute $\gamma(z_{nk}) = p(z_{nk} = 1|\mathbf{x})$.

(M) Use $\gamma(z_{nk})$ to compute eqs. (2.13)-(2.15).

2.4 An interpretation of EM

Let us now leave the GMM aside. In more general, perhaps abstract, terms, the goal of the EM algorithm is to find maximum likelihood solutions for latent variable models (LVMs) by breaking down the the optimisation problem into a functional approximation of the likelihood, and then the (simpler) optimisation of such approximation.

Recall our notation involving an observed variable x and a latent variable z. In general LVMs, the likelihood can be expressed as

$$p(x|\theta) = \int_{\mathcal{Z}} p(x|\theta, z)p(z|\theta)\mathrm{d}z. \tag{2.16}$$

NB: We treat both the discrete and continuous equivalently.

In general, direct optimisation of eq. (2.16) is difficult. Even calculating the above expression is only possible in limited cases, since mixtures do not mix well with the logarithm. In fact, even for likelihood in the exponential family, the mixture is not longer exponential and thus the application of the logarithm does not remove the exponential as in the single-Gaussian case.

Let us then consider the hypothetical scenario where we have access to the values of z alongside the observed x.

Definition 2.1.
We will refer to $\{\mathbf{x}, \mathbf{z}\}$ as the complete dataset, while $\mathbf{x}$ will be the *observed* or *incomplete* dataset.

The related likelihood to the complete dataset is

$$\log p(\mathbf{x}, \mathbf{z}|\theta) = \log \prod_{n=1}^{N} p(x_n, z_n|\theta) \tag{2.17}$$

$$= \log \prod_{n=1}^{N} p(x_n|z_n, \theta)p(z_n|\theta) \tag{2.18}$$

$$= \sum_{n=1}^{N} \log p(x_n|z_n, \theta) + \log p(z_n|\theta) \tag{2.19}$$

which we will assume is simpler to evaluate and optimise than $\log p(\mathbf{x}|\theta)$.

This, however, is impractical since $\mathbf{z}$ is unknown. An interesting interpretation of this optimisation problem is presented next.

Remark 2.4.

Since z_n is not observed, the complete-data log-likelihood in eq. (2.19) can be interpreted as a random function. Therefore, an alternative optimisation strategy is to estimate its expectation (E step) and then maximise the resulting deterministic expression (M step). At the end of the chapter, we will formally justify why taking the expectation is more than an intuition.

In more detail, this 2-step optimisation procedure results in moving from a candidate solution θ^{old} by first computing $p(\mathbf{z}|\mathbf{x}, \theta^{\text{old}})$ and then the expectation of the complete-data log-likelihood in eq. (2.19) given by

$$Q(\theta, \theta^{\text{old}}) = \sum_{z} \log p(\mathbf{x}, \mathbf{z}|\theta) p(\mathbf{z}|\mathbf{x}, \theta^{\text{old}}), \tag{2.20}$$

to finally reach an updated candidate solution θ^{new}

$$\theta^{\text{new}} = \arg\max_{\theta} Q(\theta, \theta^{\text{old}}). \tag{2.21}$$

Remark 2.5.

This procedure can also be used for maximum a posteriori estimation, in which case $Q(\theta, \theta^{\text{old}}) \to Q(\theta, \theta^{\text{old}}) + \log p(\theta)$ incorporates the prior over the parameter.

Now let us return to the GMM case and feed back these observations. For the GMM, the complete-data log-likelihood is

$$p(\mathbf{x}, \mathbf{z}|\theta) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{nk}}, \tag{2.22}$$

and thus the log-likelihood is

$$l(\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left(\log \pi_k + \log \mathcal{N}(x_n|\mu_k, \Sigma_k) \right), \tag{2.23}$$

which is tractable.

Remark 2.6.

The objective in eq. (2.23) is straightforward to optimise: since only one term in the k-sum is non-zero, the optimal mean and covariances can be computed in the same ways as the single Gaussian case. Furthermore, imposing the first order optimality condition and enforcing the $\sum_{k=1}^{K} \pi_k = 1$ via the Lagrangian, gives $\pi_k = \sum_{n=1}^{N} x_n / N$ directly.

Recall that this optima is a function of $\mathbf{z}$, and thus impossible to calculate directly, so we will calculate its expectation. To this end, we have

$$p(\mathbf{z}|\mathbf{x}, \theta) \propto p(\mathbf{z}, \mathbf{x}|\theta) = \prod_{n=1}^{N} \prod_{k=1}^{K} \underbrace{\pi_k^{z_{nk}} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{nk}}}_{\propto p(z_{nk}|x_n, \theta)}, \tag{2.24}$$

which means that $p(\mathbf{z}|\mathbf{x}, \theta)$ factorises wrt to n and k, and thus all the z_n are independent.

This is reasonable, since the cluster responsibilities over one sample should not affect the rest (due to the i.i.d. assumption).

The expectation of $\mathbf{z}$ can be computed as follows,

$$\begin{aligned}
\mathbb{E}\left(z_{nk}\right) &= 1 \cdot p(z_{nk} = 1|\mathbf{x}, \theta) + 0 \cdot p(z_{nk} = 0|\mathbf{x}, \theta) && (2.25)\\
&= p(z_{nk} = 1|\mathbf{x}, \theta) && (2.26)\\
&= \gamma(z_{nk}) && (2.27)\\
&\stackrel{\text{def}}{=} \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}. && (2.28)
\end{aligned}$$

Note from eq. (2.23) that the objective is linear in z_{nk}, which makes the computation of its expectation straightforward:

$$\mathbf{E}_z \log p(\mathbf{x}, \mathbf{z}|\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left(\log \pi_k + \log \mathcal{N}(x_n|\mu_k, \Sigma_k)\right). \tag{2.29}$$

Exercise 2.2.
Show that GMM recovers K-means. For that, choose $p(x|\mu_k, \Sigma_k) = \mathcal{N}(x|\mu_k, \epsilon I)$ with $\epsilon > 0$ fixed to show that you recover a *soft-assignment* version of K-means. Then, take $\epsilon \to 0$ to recover vanilla K-means.

Exercise 2.3.
See the applications of EM to mixtures of Bernoulli and Bayesian linear regression in (Bishop, 2006).

Example 2.1.
Let us consider an implementation of the GMM training pipeline as described. Assuming $\mathbb{R}^2$ as the sample space, $K = 3$ clusters, $N = 400$ samples, Fig. 2.1 shows two choices for the initial condition. Then, after running the iterative training procedure, Fig. 2.2 shows the learnt clusters alongside the true values. Lastly, Fig. 2.3 shows the evolution of the likelihood per iteration. Note that both initialisations arrived at the same model, but the *good* initialisation was more efficient. The demo is available in the repository.

2.5 EM in its general form

Recall:

$$\underbrace{p(\mathbf{x}|\theta)}_{\text{difficult}} = \sum_{\mathbf{z}} \underbrace{p(\mathbf{x}, \mathbf{z}|\theta)}_{\text{easier}}. \tag{2.30}$$

We are interested in deriving EM as a model-approximation approach. To that end, let us consider a distribution over the latent variable $q(z)$; intuitively, this distribution will approximate $p(z|x, \theta)$. For any choice of q, the following holds:

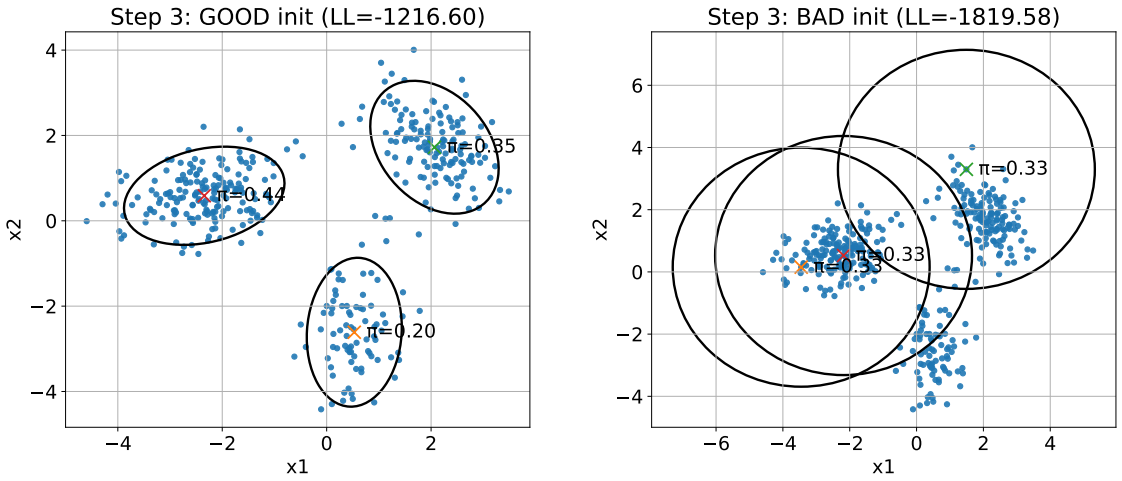

Figure 2.1: Two initialisations for the GMM model

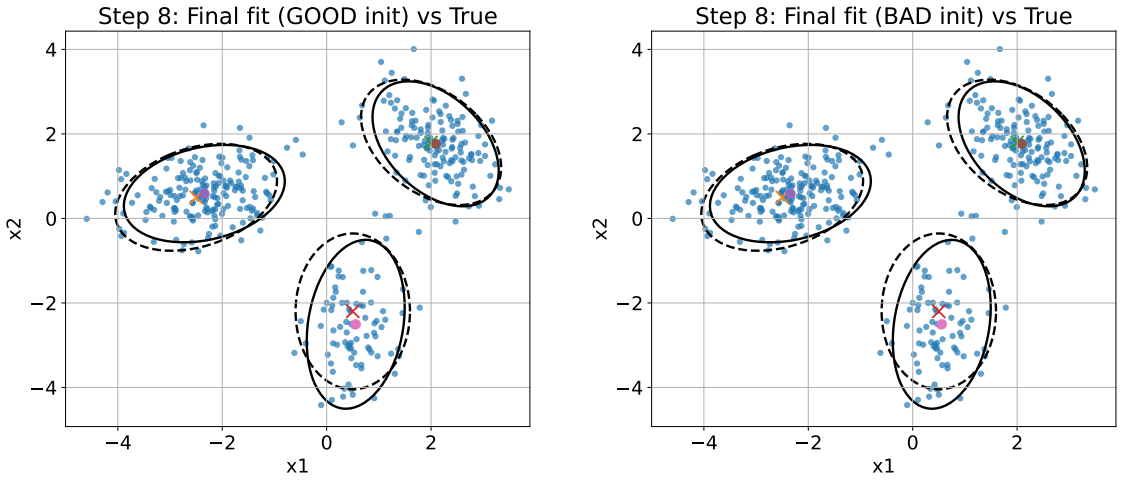

Figure 2.2: Solutions corresponding to the initialisations in Fig. 2.1.

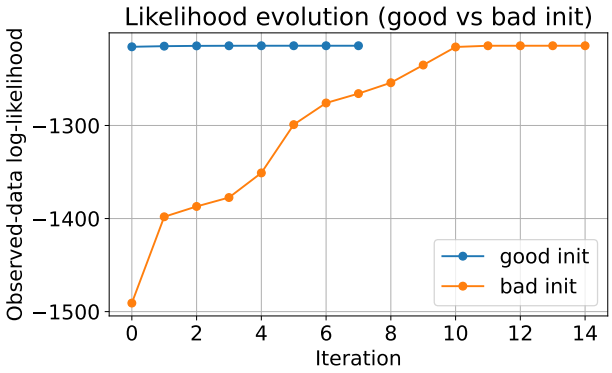

Figure 2.3: Evolution of the log-likelihoodfor both initialisations in Fig. 2.1.

$$
\begin{aligned}
\log p(x|\theta) &= \sum_z q(z) \log p(x|\theta) && \leftarrow \quad \log p(x|\theta) \text{is constant wrt } z \\
&= \sum_z q(z) \log \left(p(x|\theta) \frac{p(z|x,\theta) q(z)}{p(z|x,\theta) q(z)} \right) && \leftarrow \quad \text{multiply by 1} \\
&= \sum_z q(z) \log \left(\frac{p(x,z|\theta)}{q(z)} \cdot \frac{q(z)}{p(z|x,\theta)} \right) && \leftarrow \quad \text{arrange} \\
&= \underbrace{\sum_z q(z) \log \left(\frac{p(x,z|\theta)}{q(z)} \right)}_{\mathcal{L}(q,\theta)} + \underbrace{\sum_z q(z) \log \left(\frac{q(z)}{p(z|x,\theta)} \right)}_{\mathrm{KL}(q(z) \| p(z|x,\theta))}. && \leftarrow \quad \text{split}
\end{aligned}
$$

Remark 2.7.
Recall that the KL is always non-negative, meaning that $\mathcal{L}(q,\theta)$ is a lower bound for $\log p(x|\theta)$.

EM breaks the ML problem into two simpler problems related to the computation of a lower bound and its optimisation. The objective above can be minimised in two stages:

- First, the distribution q is chosen in order to minimise the term $\mathrm{KL}\left(q(z) \| p(z|x,\theta)\right)$, where, the optimal solution in $q(z) = p(z|x,\theta)$.
- Then, using that choice for q, the term $\mathcal{L}(q,\theta)$ is optimised. Notice that this term is an expectation wrt q.

Remark 2.8.
This view formalises the intuition that we presented for the GMM case. The choice of the expectation of the complete log-likelihood is not arbitrary, but follows from finding the optimal approximating distribution in the KL sense.

Monotonicity of EM. Recall the variational decomposition (valid for any distribution q on z)

$$\log p(x|\theta) = \mathcal{L}(q,\theta) + \mathrm{KL}\left(q(z) \| p(z|x,\theta)\right), \tag{2.31}$$

where

$$\mathcal{L}(q,\theta) := \sum_z q(z) \log \frac{p(x,z|\theta)}{q(z)}. \tag{2.32}$$

Since $\mathrm{KL}\left(\cdot\|\cdot\right) \geq 0$, we have $\log p(x|\theta) \geq \mathcal{L}(q,\theta)$ for all (q,θ).

Let θ^t be the current iterate. The E-step sets

$$q^{t+1}(z) := p(z|x,\theta^t), \tag{2.33}$$

for which $\mathrm{KL}\left(q^{t+1}(z) \| p(z \mid x,\theta^t)\right) = 0$. Plugging into (2.31) yields

$$\log p(x|\theta^t) = \mathcal{L}(q^{t+1},\theta^t). \tag{2.34}$$

The M-step then chooses

$$\theta^{t+1} \in \arg\max_{\theta} \ \mathcal{L}(q^{t+1}, \theta), \tag{2.35}$$

hence

$$\mathcal{L}(q^{t+1}, \theta^{t+1}) \geq \mathcal{L}(q^{t+1}, \theta^t) = \log p(x|\theta^t). \tag{2.36}$$

Finally, since $\log p(x|\theta) \geq \mathcal{L}(q^{t+1}, \theta)$ for any θ, we obtain

$$\log p(x|\theta^{t+1}) \ \geq \ \mathcal{L}(q^{t+1}, \theta^{t+1}) \ \geq \ \log p(x|\theta^t). \tag{2.37}$$

Therefore EM produces a non-decreasing sequence of log-likelihood values.

Remark 2.9 (Generalised EM)**.**
The monotonicity argument above only requires that the M-step returns θ^{t+1} such that $\mathcal{L}(q^{t+1}, \theta^{t+1}) \geq \mathcal{L}(q^{t+1}, \theta^t)$. Thus, it is not necessary to fid the exact (local) optimum of $\mathcal{L}$ in the M-step; any update that increases $\mathcal{L}$ yields a *generalised EM* (GEM) procedure with the same monotonicity guarantee. This observation will be key for the variational inference part.

Does EM fix ill-posed maximum likelihood? For several latent-variable models, the maximum likelihood (ML) objective is *ill-posed*, since the log-likelihood can be unbounded. A canonical example is the Gaussian mixture model (GMM): as we saw in class, for any datapoint $x_n \in \mathbb{R}^d$ and any $\epsilon > 0$, consider a mixture component k with

$$\mu_k = x_n, \qquad \Sigma_k = \epsilon I.$$

Then

$$\mathcal{N}(x_n; \mu_k, \Sigma_k) = (2\pi)^{-d/2}|\Sigma_k|^{-1/2} = (2\pi)^{-d/2}\epsilon^{-d/2},$$

so the corresponding contribution to $\log p(x_n|\theta)$ diverges as $\epsilon \to 0$. Consequently, $\sup_\theta \log p(x|\theta) = +\infty$ for unconstrained GMM maximum likelihood.

EM *does not* resolve this ill-posedness, because it is still (attempting to) maximise the same ML objective; it only changes the optimisation route. In fact, the EM updates can actively move toward singular solutions: if a component collapses around a datapoint, the E-step tends to assign it responsibility close to 1 for that datapoint, and the M-step covariance update (a responsibility-weighted empirical covariance) can shrink toward a rank-deficient matrix unless additional constraints are imposed.

Remark 2.10 (How to avoid degeneracy)**.**
To obtain well-posed estimation one typically modifies the objective or the parameter space, e.g.

- **MAP (penalised) EM:** maximise $\log p(x \mid \theta) + \log p(\theta)$ by placing a prior on parameters (e.g. inverse-Wishart prior on Σ_k, Dirichlet prior on mixing weights).
- **Constrained ML:** enforce $\Sigma_k \succeq \sigma^2_{\min} I$ or tie covariances (e.g. $\Sigma_k = \Sigma$).
- **Regularisation / damping:** add a penalty to discourage small determinants, or replace $\Sigma_k \leftarrow \Sigma_k + \epsilon I$ after updates.
- **Alternative components:** heavier-tailed mixtures (e.g. Student-t mixtures) can

reduce single-point capture.

In short: EM guarantees non-decreasing likelihood; when the likelihood is *unbounded above*, monotone ascent is still compatible with divergence toward a singular solution.

Remark 2.11.
EM does not change the (possibly ill-posed) ML objective; it only changes the optimisation procedure. In particular, it replaces direct maximisation of $\log p(x|\theta)$ by alternating easier subproblems (posterior inference and complete-data fitting) and guarantees monotone progress, which often stabilises learning in practice even though degeneracies may still exist in principle.

2.6 Concluding remarks

- **EM as coordinate ascent on a lower bound.** EM alternates between (i) an *inference* step, selecting $q(z) = p(z|x, \theta)$ to tighten a variational lower bound, and (ii) a *learning* step, updating θ to increase that bound:

$$q^{t+1}(z) = p(z|x, \theta^t), \qquad \theta^{t+1} \in \arg\max_{\theta} \mathbf{E}_{q^{t+1}}[\log p(x, z|\theta)].$$

- **Inference vs. learning.** Latent-variable models are expressive, but at the cost of posterior inference. EM addresses this inference need, where the E-step performs posterior computation and the M-step performs parameter fitting.

- **Feasibility of EM.** EM is particularly effective when the complete-data likelihood belongs to a tractable family (often exponential-family), leading to closed-form M-steps. When the exact posterior is intractable, one can replace the E-step by a restricted family q (variational EM).

- **Limitations.** EM is monotone but generally converges to a stationary point, so performance depends on initialization. Moreover, EM does not fix ill-posed ML objectives (e.g. degeneracy in GMMs) without additional constraints or priors.

These ideas generalise beyond mixtures: many modern learning procedures can be understood as optimising tractable surrogates (bounds) of intractable objectives, with EM providing a canonical template.

2.6.1 Suggested exercises

1. **Latent variable models and incomplete data (theory).**

   (a) Define a latent variable model and distinguish between complete and incomplete data.

   (b) Explain why direct maximisation of the marginal likelihood is often intractable.

   (c) Describe how the introduction of latent variables simplifies modelling but complicates inference.

2. **Expectation–Maximisation algorithm (theory).** Consider a latent variable model with parameters θ.
   (a) Derive the EM algorithm starting from the marginal log-likelihood.
   (b) Define the Q-function and explain the role of the E-step and the M-step.
   (c) Prove that each EM iteration does not decrease the log-likelihood.
3. **Gaussian mixture models (coursework).**
   (a) Implement the EM algorithm for Gaussian mixture models.
   (b) Investigate the effect of initialisation on convergence.
   (c) Illustrate the relationship between k-means and GMMs by varying the covariance structure.
4. **Likelihood degeneracy and regularisation (coursework).**
   (a) Demonstrate empirically the likelihood degeneracy of Gaussian mixture models.
   (b) Propose and implement at least one regularisation strategy.
   (c) Analyse how regularisation affects the learned parameters and likelihood.

Week 3

Approximate Inference

NB: This is based on (Andrieu, de Freitas, Doucet & Jordan, 2003), (Blei, Kucukelbir & McAuliffe, 2017), and Chapter 10 of (Bishop, 2006).

3.1 Motivation: intractable posteriors

Bayesian inference in generative models. A probabilistic generative model specifies a joint distribution over observed variables $x \in \mathcal{X}$ and latent variables $z \in \mathcal{Z}$:

$$p_\theta(x, z) = p_\theta(z)\, p_\theta(x \mid z),$$

where θ denotes model parameters.

Given observations x, Bayesian inference aims to compute the posterior distribution

$$p_\theta(z \mid x) = \frac{p_\theta(x, z)}{p_\theta(x)}, \qquad p_\theta(x) = \int p_\theta(x, z)\, dz.$$

The posterior encodes all the uncertainty related to the latent variable z after observing the data, and is the central object of interest in Bayesian modelling.

Remark 3.1.
From now on, we will drop the explicit dependency on the parameter θ. From a Bayesian standpoint, we will consider that the latent variable z encapsulates all (random) unknowns. This includes global variables such as parameters, and local variables such as cluster assignments.

Example 3.1 (Hierarchical Gaussian mixture model)**.**
To illustrate the role of both global and local latent variables in a PGM, we consider the following hierarchical GMM.

Let $x_n \in \mathbb{R}^d$, $n = 1, \dots, N$, denote the observed data. The latent variables are:

- mixture weights $\pi = (\pi_1, \dots, \pi_K) \in \Delta^{K-1}$, $\quad \Delta^{K-1} = \left\{\pi \in \mathbb{R}^K : \pi_k \geq 0 \ \forall k,\ \sum_{k=1}^K \pi_k = 1\right\}.$,
- cluster means $\mu_1, \dots, \mu_K \in \mathbb{R}^d$,
- cluster assignments $z_1, \dots, z_N \in \{1, \dots, K\}$.

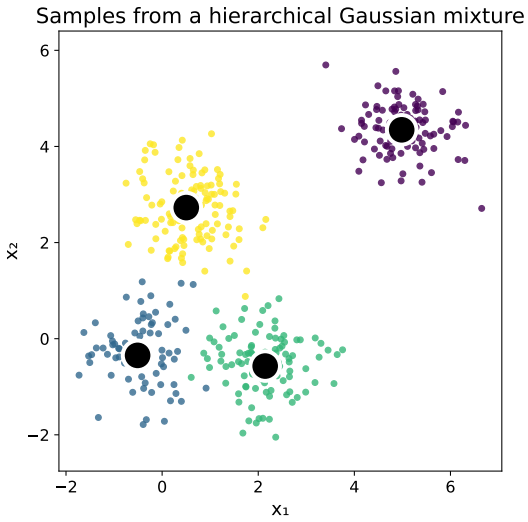

Figure 3.1: Samples from a hierarchical GMM: 2 dimensions, $N = 400$ samples.

The generative process is:

$$\begin{aligned} \pi &\sim \text{Dirichlet}(\alpha) && \\ \mu_k &\sim \mathcal{N}(m_0, \Sigma_0) && k = 1, \ldots, K \\ z_n \mid \pi &\sim \text{Categorical}(\pi) && n = 1, \ldots, N \\ x_n \mid z_n, \{\mu_k\} &\sim \mathcal{N}(\mu_{z_n}, \Sigma) && n = 1, \ldots, N. \end{aligned}$$

The resulting joint distribution factorises as

$$p(x, z, \mu, \pi) = p(\pi) \prod_{k=1}^{K} p(\mu_k) \prod_{n=1}^{N} p(z_n \mid \pi)\, p(x_n \mid z_n, \mu).$$

Given observations $x_{1:N}$, the posterior given by

$$p(z_{1:N}, \mu_{1:K}, \pi \mid x_{1:N}),$$

is analytically intractable due to the hierarchical coupling between the latent variables in the the marginal

$$p(x_{1:N}) = \int p(z_{1:N}, \mu_{1:K}, \pi, x_{1:N}) \mathrm{d}z_{1:N} \mathrm{d}\mu_{1:K} \mathrm{d}\pi.$$

Fig. 3.1 shows data generated by this model using hierarchical sampling ($N = 400, K = 4$), and Fig. 3.2 shows the graphical model representation.

Example 3.2 (Bayesian linear regression)**.**
Let $\{(x_n, y_n)\}_{n=1}^N$ be the observed data, where $\{x_n\}_{n=1}^N \subset \mathbb{R}^d$ are inputs and $\{y_n\}_{n=1}^N \subset \mathbb{R}$ are outputs. The latent variable is the regression weight vector $w \in \mathbb{R}^d$, which is the parameter of the model. Notice that in this case the parameter is the only global latent variable and there are no local latent variables.

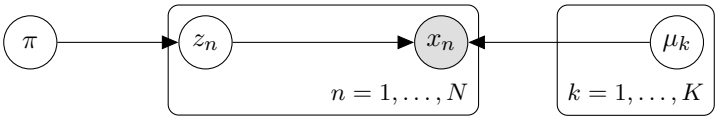

Figure 3.2: Graphical model of a hierarchical Gaussian mixture model. The global mixture weights π and component means $\{\mu_k\}_{k=1}^K$ govern the generation of latent cluster assignments z_n and observations x_n.

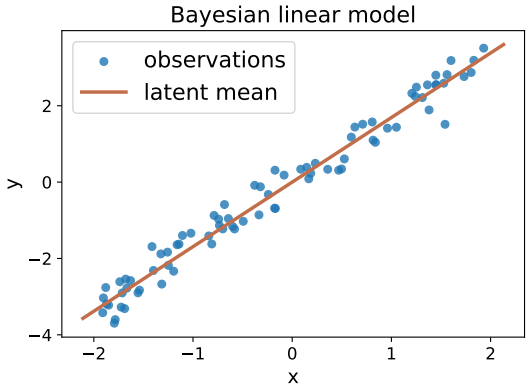

Figure 3.3: Samples from a Bayesian linear model: 1 dimension, $N = 400$ samples.

The generative model is defined by

$$w \sim \mathcal{N}(m_0, \Sigma_0), \qquad y_n \mid w, x_n \sim \mathcal{N}(x_n^\top w, \sigma^2), \quad n = 1, \ldots, N.$$

Denoting the input matrix $X \in \mathbb{R}^{N \times d}$ with rows $x_n^\top$, and the observation vector $y = (y_1, \ldots, y_N)^\top$, the likelihood can be written compactly as

$$y \mid w, X \sim \mathcal{N}(Xw, \sigma^2 I_N),$$

where I_N denotes the $N \times N$ identity matrix. Since the joint distribution factorises as $p(y, w \mid X) = p(w) \prod_{n=1}^N p(y_n \mid w, x_n)$, the posterior distribution over the latent weights can be expressed as

$$p(w \mid X, y) \propto p(w) \prod_{n=1}^{N} p(y_n \mid w, x_n),$$

which admits the closed-form Gaussian solution

$$p(w \mid X, y) = \mathcal{N}(m_N, \Sigma_N),$$

with

$$\Sigma_N^{-1} = \Sigma_0^{-1} + \frac{1}{\sigma^2} X^\top X, \quad m_N = \Sigma_N \left(\Sigma_0^{-1} m_0 + \frac{1}{\sigma^2} X^\top y \right).$$

This model admits exact Bayesian inference is tractable. Fig. 3.3 shows samples from this model alongside the linear function corresponding to the latent weight. Fig. 3.4 shows the graphical model representation.

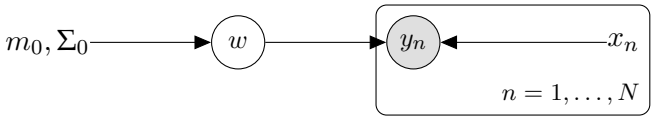

Figure 3.4: Graphical model of Bayesian linear regression. The global weight vector w generates observations y_n given input x_n for $n = 1, \ldots, N$.

Example 3.3 (State space model)**.**

Let us now consider a latent variable model describing the evolution of a dynamical system observed through noisy measurements.

Let $z_t \in \mathbb{R}^m$, $t = 1, \ldots, T$, denote the latent state at time t, and let $x_t \in \mathbb{R}^d$ denote the corresponding observation. The latent states and observations are linked through a transition model and an observation model.

The generative process is defined as follows:

$$\begin{aligned} z_1 &\sim p(z_1) \\ z_t &= f(z_{t-1}) + \epsilon_t, \qquad t = 2, \ldots, T \\ x_t &= h(z_t) + \eta_t, \qquad t = 1, \ldots, T, \end{aligned}$$

where $f : \mathbb{R}^m \to \mathbb{R}^m$ is the (possibly nonlinear) state transition function, $h : \mathbb{R}^m \to \mathbb{R}^d$ is the observation function, and $\epsilon_t \sim p_{\text{transition}}$ and $\eta_t \sim p_{\text{observation}}$ are the process and observation noise sources respectively.

The joint distribution over latent states and observations factorises as

$$p(x_{1:T}, z_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t \mid z_{t-1}) \prod_{t=1}^{T} p(x_t \mid z_t).$$

Given the observations $x_{1:T}$, the posterior $p(z_{1:T} \mid x_{1:T})$ is tractable only when f and h are linear, and both $p_{\text{transition}}$ and $p_{\text{observation}}$ are Gaussian, otherwise, approximate inference methods are required. Fig. 3.5 shows a sample form this model, and Fig. 3.6 the graphical model representation.

Approximate inference replaces the true posterior $p(z \mid x)$ with an approximation, say $q(z)$, that is as close a possible to the true posterior, while at the same time allows for computing expectations and scalable computation. In this sense, there are two dominant paradigms for computing q.

- Monte Carlo methods, which approximate the posterior via samples $z^{(s)} \sim p(z \mid x)$. This estimate is asymptotically exact but often computationally expensive.
- Variational methods, which provide a functional approximation of the posterior by solving an optimization problem:

$$q^\star(z) = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\Big(q(z) \,\|\, p(z \mid x)\Big),$$

where $\mathcal{Q}$ is a tractable family of distributions. Variational approximations are fast

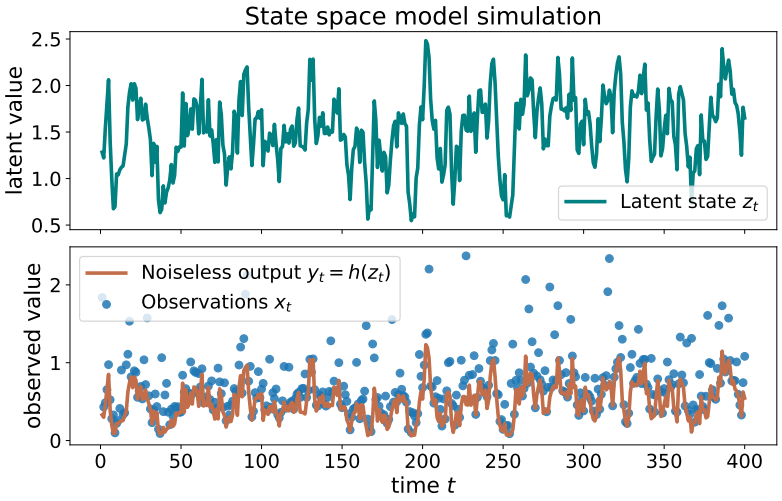

Figure 3.5: Samples from a nonlinear state space model: 1 dimension, $N = 400$ samples.

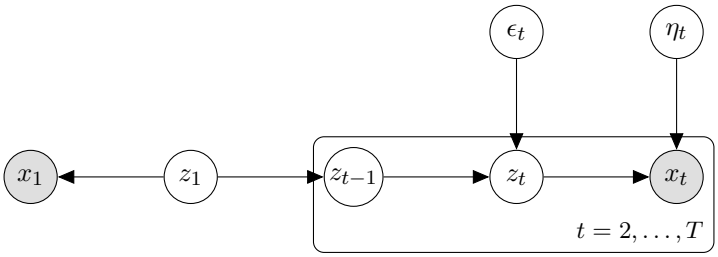

Figure 3.6: Graphical model of a (possibly nonlinear) state space model. Latent states $\{z_t\}_{t=1}^T$ form a Markov chain via the transition dynamics, and each observation x_t depends on the corresponding state z_t through the observation model.

and scalable, but introduce bias.

3.2 Markov chain Monte Carlo

In Bayesian inference, the posterior is usually used to computing expectations of the form

$$\mathbb{E}_{p(z|x)}[f(z)] = \int f(z)\, p(z \mid x)\, dz.$$

When direct evaluation of this integral is infeasible, an alternative is to use a Monte Carlo approximation, that is,

$$\mathbb{E}_{p(z|x)}[f(z)] \approx \frac{1}{S}\sum_{s=1}^{S} f(z^{(s)}), \qquad z^{(s)} \sim p(z \mid x).$$

The law of large numbers guarantees that this estimator converges almost surely as $S \to \infty$. Furthermore, the root-mean-square error of this estimator decreases at a rate $1/\sqrt{S}$

However, the challenge in Bayesian inference is to obtain the samples $\{z^{(s)}\}_{s=1}^S$. This is

because the posterior

$$p(z \mid x) = \frac{p(x, z)}{p(x)},$$

is rarely tractable and expensive to compute numerically in high dimensions due to the integral form of the normalizing constant

$$p(x) = \int p(x, z)\, dz = \int p(x \mid z) p(z)\, dz.$$

As a consequence, direct sampling from $p(z \mid x)$ is not possible.

The rationale behind Markov Chain Monte Carlo (MCMC) methods is to construct a Markov chain whose limiting stationary distribution is the desired posterior. Recall that a Markov chain $\{z^{(t)}\}_{t \in \mathbb{N}}$ is defined by a transition kernel $T(z' \mid z)$, where

$$\mathbb{P}(z^{(t+1)} = z' \mid z^{(t)} = z) = T(z' \mid z).$$

For this Markov chain to have the posterior $p(z \mid x)$ as its limiting distribution, two conditions need to hold. First, the posterior has to be the chain's stationary distribution, that is,

$$\int p(z \mid x)\, T(z' \mid z)\, dz = p(z' \mid x).$$

This means that if the chain starts at $z \sim p(z \mid x)$, then after one step the chain remains in the same distribution. Second, the chain should converge to its stationary distribution. A sufficient (but not necessary) condition for stationarity is the *detailed balance* condition,

$$p(z \mid x)\, T(z' \mid z) = p(z' \mid x)\, T(z \mid z'), \tag{3.1}$$

which implies that $p(z \mid x)$ is a stationary distribution of the Markov chain. The detailed balance condition states that, under the stationary distribution, the probability flow from state z to state z' is exactly balanced by the flow from z' to z. As a result, there is no net movement of probability mass, and the target distribution remains invariant under the Markov chain dynamics.

To ensure convergence to the stationary distribution from an arbitrary initial state, additional regularity conditions are required. In particular, the Markov chain should be *irreducible*, meaning that it is possible to reach any state from any other state with positive probability, and *aperiodic*, meaning that the chain does not get trapped in deterministic cycles. Under these conditions, the distribution of $z^{(t)}$ converges to $p(z \mid x)$ as $t \to \infty$.

Metropolis–Hastings. A construction of a transition kernel satisfying the convergence conditions above can be obtained in two steps. First, a candidate state z' is drawn from an arbitrary proposal distribution $\pi(z' \mid z)$. Second, this proposal is accepted or rejected according to a criterion that depends on how likely z' is under the target distribution $p(z \mid x)$.

Given the current state z, the Metropolis–Hastings (MH) update proceeds as follows:

1. Propose a new state
$$z' \sim \pi(z' \mid z).$$

2. Compute the probability of acceptance
$$\alpha(z, z') = \min\left(1,\ \frac{p(z' \mid x)\,\pi(z \mid z')}{p(z \mid x)\,\pi(z' \mid z)}\right). \tag{3.2}$$

3. Set the new sample as
$$z^{(t+1)} = \begin{cases} z', & \text{with probability } \alpha(z, z'), \\ z, & \text{otherwise.} \end{cases}$$

Remark 3.2.
MH does not require knowledge of the normalising constant $p(x)$. As a consequence, since $p(z \mid x) \propto p(x, z)$, the acceptance probability can be computed using the joint density $p(x, z)$.

Remark 3.3.
MH's transition kernel satisfies the detailed balance condition in eq. (3.1). Therefore, $p(z \mid x)$ is a stationary limiting distribution of the chain built my MH.

Remark 3.4.
A particular instance of MH can be identified by choosing a symmetric proposal, that is, $q(z \mid z') = q(z' \mid z)$. In this case, the acceptance probability reduces to
$$\alpha(z, z') = \min\left(1,\ \frac{p(z' \mid x)}{p(z \mid x)}\right).$$
This is know as the Metropolis method.

In practice, a common choice for the proposal is simply a random walk
$$\pi(z' \mid z) = \mathcal{N}(z, \sigma^2 I),$$
however, though simple to implement, random-walk MH can mix poorly in high-dimensional or highly-correlated posteriors.

Fig. 3.7 shows an implementation of MH to sample from a mixture of 2 Gaussians using a single Gaussian as proposal.

Gibbs sampling. To develop a MH variant that exploits conditional structure in the posterior distribution, the proposal over each coordinate of z can be coupled to previously sampled coordinates. Let us denote $z = (z_1, \ldots, z_m)$, and update each component by sampling from its conditional distribution:
$$z_i \sim p(z_i \mid z_{-i}, x),$$
where z_{-i} denotes all components except z_i.

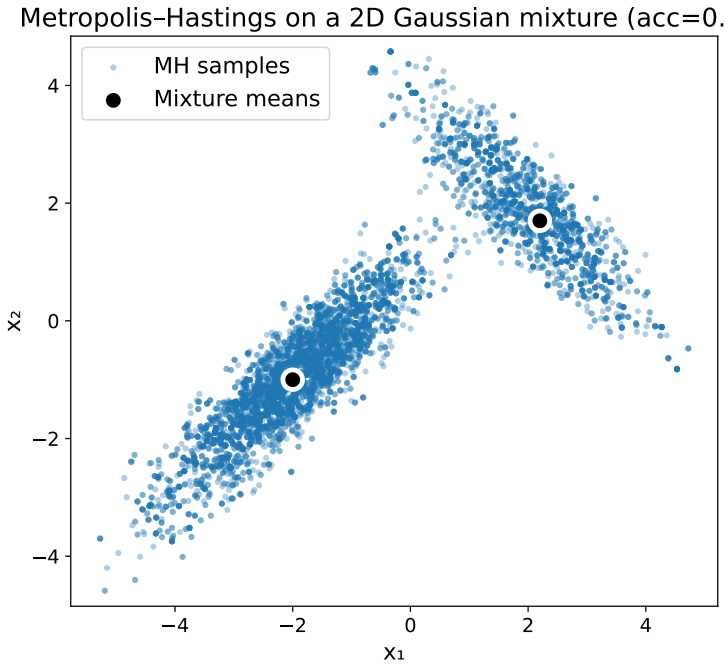

Figure 3.7: Samples from a 2D 2-Gaussian mixture using Metropolis Hastings: $N = 10,000$ samples. Observe the poor performance revealed by the number of accepted samples divided by the number of proposed samples.

A complete Gibbs update consists of sequentially sampling:

$$z_1 \sim p(z_1 \mid z_2, \ldots, z_m, x),$$

$$z_2 \sim p(z_2 \mid z_1, z_3, \ldots, z_m, x),$$

$$\vdots$$

$$z_m \sim p(z_m \mid z_1, \ldots, z_{m-1}, x).$$

To see that Gibbs is a particular instance of MH, let us first denote the target posterior by $r(z) = p(z \mid x)$ for notational simplicity, and consider a fixed index $i \in \{1, \ldots, m\}$. To update the i-th coordinate while keeping z_{-i} fixed, Gibbs samples directly from the conditional posterior

$$z_i' \sim r(z_i \mid z_{-i}),$$

and defines the proposed sample

$$z' = (z_i', z_{-i}).$$

For the complete variable z, this corresponds to a proposal distribution

$$\begin{aligned} \pi(z' \mid z) &= \pi(z_i', z_{-i}' \mid z_i, z_{-i}) \\ &= \pi(z_i' \mid z_i, z_{-i}, z_{-i}')\pi(z_{-i}' \mid z_i, z_{-i}) \\ &= \pi(z_i' \mid z_{-i})\pi(z_{-i}' \mid z_{-i}) \\ &= r(z_i' \mid z_{-i})\, \mathbf{1}\{z_{-i}' = z_{-i}\}. \end{aligned} \tag{3.3}$$

To compute the acceptance probability in eq. (3.2), we can factorise

Using the factorization

$$r(z) = r(z_{-i})\, r(z_i \mid z_{-i}), \qquad r(z') = r(z_{-i})\, r(z'_i \mid z_{-i}),$$

together with the proposal in eq. (3.3) (recall that $z'_{-i} = z_{-i}$)

$$\pi(z' \mid z) = r(z'_i \mid z_{-i}), \qquad \pi(z \mid z') = r(z_i \mid z_{-i}),$$

we obtain

$$\frac{r(z')\,\pi(z \mid z')}{r(z)\,\pi(z' \mid z)} = \frac{r(z_{-i})\, r(z'_i \mid z_{-i})\, r(z_i \mid z_{-i})}{r(z_{-i})\, r(z_i \mid z_{-i})\, r(z'_i \mid z_{-i})} = 1.$$

Therefore, the acceptance probability is

$$\alpha(z, z') = 1,$$

meaning that the proposed move is always accepted.

Remark 3.5.

Gibbs sampling is a special case of MH, in which the proposal distribution is the exact full conditional distribution, and the acceptance probability is identically equal to one. Furthermore, each update (not necessarily the full update) leaves the joint posterior invariant.

Practical considerations. There are relevant implementation practices to be taken into account when using MCMC.

Burn-in period: Though MH is guaranteed to converge to its stationary distribution, the fact that the chain starts from an arbitrary distribution implies that the initial samples will not be representative of the target posterior. Therefore, the initial samples should be discarded as invalid samples from $p(z \mid x)$. When the samples are representative from the target, we say that the chain has *mixed*. Furthermore, assessment of this convergence must be assessed empirically.

Autocorrelation: The samples used for Monte Carlo integration should be i.i.d. samples from the target (posterior) distributions; in fact, when correlated samples are used it is the *effective sample size* that governs the quality of the approximation. Since the samples are generated by a chain with a correlated transition kernel, consecutive samples are not independent (in general correlated) by construction. To alleviate this, the samples from the chain should be *thinned*, that is, to consider a subset of samples, for instance, one every 50 samples.

Fig. 3.8 shows an implementation of Gibbs and MH sampling from a correlated Gaussian. Observe how with 1,000 samples Gibbs fully explores the support of the distribution while MH struggle to propose valid samples.

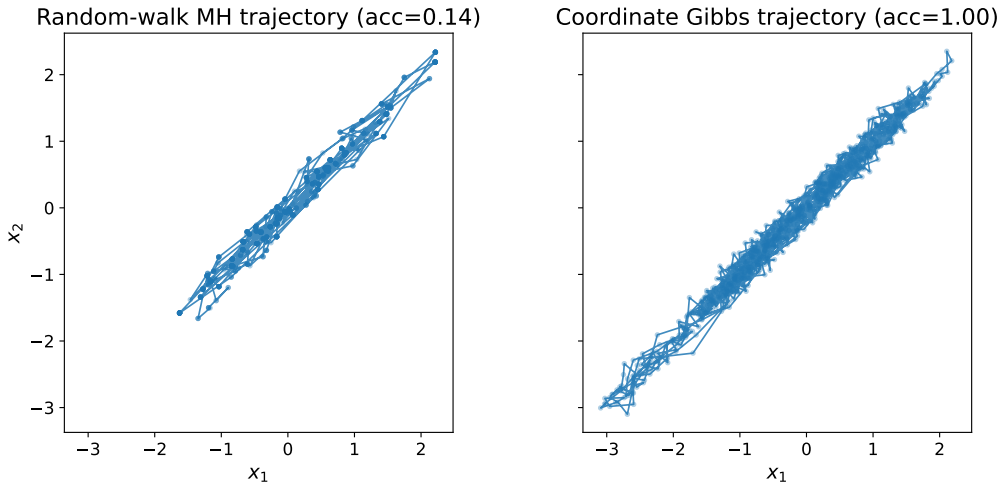

Figure 3.8: Samples from a 2D highly correlated Gaussian using Gibbs (right) and Metropolis Hastings (right): $N = 1,000$ samples.

3.3 The Laplace approximation

Before focusing on variational inference in detail, let us study the following local approximation to the posterior. Consider the mode of the posterior (or one of the modes in the multimodal case), given by $\hat{z} = \arg\max_z \log p(z \mid x)$. Then, a second-order Taylor expansion of the log-posterior around this mode gives (recall that $\nabla \log p(z \mid x)|_{z=\hat{z}} = 0$):

$$\log p(z \mid x) \approx \log p(\hat{z} \mid x) - \tfrac{1}{2}(z-\hat{z})^\top H (z-\hat{z}), \quad H := -\nabla^2 \log p(z \mid x)\Big|_{z=\hat{z}}.$$

The equivalent approximating distribution can be calculated simply by

$$q(z) = \exp\Big(\log p(\hat{z} \mid x) - \tfrac{1}{2}(z-\hat{z})^\top H (z-\hat{z})\Big) \propto \exp\Big(-\tfrac{1}{2}(z-\hat{z})^\top H (z-\hat{z})\Big).$$

Therefore, the approximating distribution is a Gaussian centered in the mode, with a variance given by the curvature of the of the true posterior at the mode: Hence,

$$q(z) = \mathcal{N}\Big(z \mid \hat{z},\ H^{-1}\Big).$$

Remark 3.6.
This approximation only requires the maximiser $\hat{z}$ and the Hessian of $\log p(z \mid x)$, and not the actual value of the mode $\log p(\hat{z} \mid x)$. This is because the normalising constant can be calculated from the curvature when the distribution is Gaussian.

Fig. 3.9 illustrates the Laplace approximation to a 10-DOF χ^2 distribution.

Example 3.4 (Laplace approximation for Bayesian logistic regression)**.**
Consider a one-dimensional Bayesian logistic regression model. Let $\{(x_n, y_n)\}_{n=1}^N$ be observed data with $x_n \in \mathbb{R}$ and $y_n \in \{0, 1\}$. The model is defined as

$$w \sim \mathcal{N}(0, \tau^2), \qquad p(y_n = 1 \mid w, x_n) = \sigma(w x_n),$$

where $\sigma(t) = (1 + e^{-t})^{-1}$ denotes the logistic sigmoid function, and latent variable

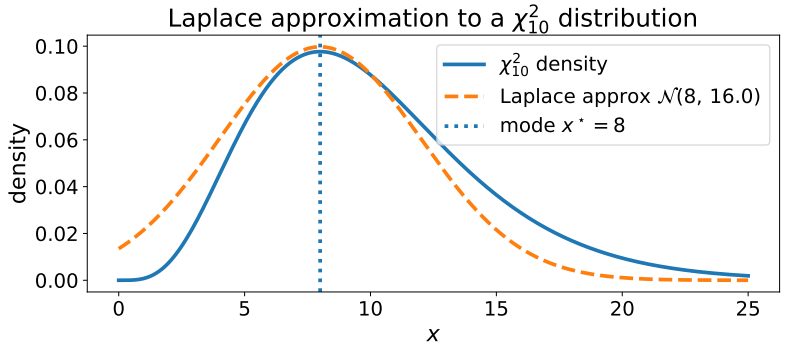

Figure 3.9: Laplace approximation to a 10-DOF χ^2 distribution.

is the regression weight $w \in \mathbb{R}$.

The posterior distribution is (using Bayes theorem)

$$p(w \mid x, y) = \frac{p(y \mid w, x)p(w)}{p(y)} \propto \exp\left(-\frac{w^2}{2\tau^2}\right) \prod_{n=1}^{N} \sigma(wx_n)^{y_n} \Big(1 - \sigma(wx_n)\Big)^{1-y_n}.$$

Using $y \log \sigma(a) + (1-y)\log(1-\sigma(a)) = ya - \log(1+e^a)$, we can write the log-posterior as

$$\log p(w \mid x, y) = -\frac{w^2}{2\tau^2} + \sum_{n=1}^{N} \Big(y_n w x_n - \log(1 + e^{wx_n})\Big) + \text{const.}$$

Its first and second derivatives wrt w are:

$$\nabla \log p(w \mid x, y) = -\frac{w}{\tau^2} + \sum_{n=1}^{N} x_n \Big(y_n - \sigma(wx_n)\Big), \tag{3.4}$$

$$\nabla^2 \log p(w \mid x, y) = -\frac{1}{\tau^2} - \sum_{n=1}^{N} x_n^2 \, \sigma(wx_n)\Big(1 - \sigma(wx_n)\Big). \tag{3.5}$$

Since the first-order optimality condition $\nabla \log p(w \mid x, y) = 0$ from eq. (3.4) cannot be solved explicitly, the maximum can be obtained using the Newton method, that is,

$$w_{\text{new}} = w_{\text{old}} - \frac{\nabla \log p(w \mid x, y)}{\nabla^2 \log p(w \mid x, y)}\bigg|_{w=w_{\text{old}}}$$

This approximation matches the posterior locally around the MAP, but fails to capture global properties such as skewness or heavy tails.

3.4 Variational inference

The evidence lower bound (ELBO). An alternative to estimating the posterior $p(z \mid x)$, is to specify a family of densities over z termed $\mathcal{Q}$, where each $q(\cdot) \in \mathcal{Q}$ is a

candidate approximation to the posterior. Our goal will be then to find the best candidate within $\mathcal{Q}$, according to an optimality criterion given by the KL divergence, that is, we will find

$$q^*(z) = \arg\min_{q \in \mathcal{Q}} \mathrm{KL}\left(q(z) \| p(z \mid x)\right). \tag{3.6}$$

Remark 3.7.
The more general or comprehensive the family $\mathcal{Q}$, the more difficult it is to find the optimal $q^*(z)$.

Observe that solving eq. (3.6) is unfeasible, since it requires $p(z \mid x)$ or equivalently $p(x)$. However, notice that since

$$\mathrm{KL}\left(q(z) \| p(z \mid x)\right) = \int q(z) \log q(z) \mathrm{d}z - \int q(z) \log p(z, x) \mathrm{d}z + \underbrace{\int q(z) \log p(x) \mathrm{d}z}_{\log p(x)},$$

the dependence of the objective on $\log p(x)$ is via a constant which can be ignored when optimising wrt $q(z)$. Therefore, the optimisation problem in eq. (3.6) is equivalent to maximising the evidence lower bound (ELBO) given by

$$\mathrm{ELBO} \stackrel{\mathrm{def}}{=} \int q(z) \log p(z, x) \mathrm{d}z - \int q(z) \log q(z) \mathrm{d}z. \tag{3.7}$$

Let us observe the following decomposition of the ELBO:

$$\begin{aligned} \mathrm{ELBO} &= \int q(z) \log p(z, x) \mathrm{d}z - \int q(z) \log q(z) \mathrm{d}z \\ &= \int q(z) \log p(x \mid z) \mathrm{d}z + \int q(z) \log p(z) \mathrm{d}z - \int q(z) \log q(z) \mathrm{d}z \\ &= \mathbb{E}\left(\log p(x \mid z)\right) - \mathrm{KL}\left(q(z) \| p(z)\right), \end{aligned} \tag{3.8}$$

where all the expectations in this section will be wrt $q(z)$, unless other wise stated.

Maximising the ELBO in eq. (3.8) is a balance between two terms. The first one seeks to assign the mass of q to the values of z that explain the observations x, while the second one ensures that $q(z)$ is close to the prior $p(z)$.

Remark 3.8.
Maximising the ELBO recovers the usual likelihood / prior trade off.

To justify the name of this objective, let us see how it relates to the so called *evidence* $\log p(x)$. Using the expression in eq. (3.7), we have

$$\begin{aligned} \log p(x) - \mathrm{ELBO} &= \int q(z) \log p(x) \mathrm{d}z - \int q(z) \log p(z, x) \mathrm{d}z + \int q(z) \log q(z) \mathrm{d}z \\ &= \int q(z) \log \frac{1}{p(z \mid x)} \mathrm{d}z + \int q(z) \log q(z) \mathrm{d}z \\ &= \mathrm{KL}\left(q(z) \| p(z \mid x)\right). \end{aligned} \tag{3.9}$$

Since the KL divergence is always nonegative, the ELBO is—as its name suggests—a

lower bound of the evidence $\log p(x)$. That is,

$$\log p(x) \geq \text{ELBO}.$$

Furthermore, the gap between these quantities is precisely the discrepancy, in terms of the KL, between the approximate posterior $q(z)$ and the true posterior.

Remark 3.9 (ELBO and parameter identification)**.**
When the latent variable is a parameter, the ELBO can be informally used for model selection, by finding the maximum a posteriori. However, when this is done in practice, it is unknown how far this solution is from the true one.

To do: include an illustration of the biased maxima

Remark 3.10 (Maximising the ELBO and EM)**.**
Observe from eq. (3.7) that the first term in the ELBO is the objective of EM, this is because in EM we have that ELBO $= \log p(x)$, since $q(z) = p(z \mid x)$—see eq. (3.9). This is possible because EM is used in cases where $p(z \mid x)$ can be computed analytically. In VI, however, we do not assume tractable posteriors, but rather we only consider a sufficiently good approximation q within the variational family $\mathcal{Q}$. Therefore, VI can be seen as an extension of EM, used in cases where the posterior is intractable.

The mean field variational family. An explicit family $\mathcal{Q}$ can be chosen based on the trade-off between the family's expressivity and the feasibility of solving the problem in eq. (3.6). We will consider the mean field (MF) family, where the distribution over the latent variable $z = (z_1, z_2, \ldots, z_m) \in \mathbb{R}^m$ factorises as[1]

$$q(z) = \prod_{i=1}^{m} q_i(z_i). \tag{3.10}$$

With the introduction of the MF family, the task of finding an m-dimensional distribution is replaced by finding m 1-dimensional ones.

Remark 3.11 (Generality of the MF family)**.**
The coordinate-wise densities are not specified in explicit form. Instead, appropriate expressions can be chosen for each coordinate according to its characteristics, for example whether the coordinate is continuous or discrete.

Example 3.5 (MF approximation for the hierarchical GMM)**.**
A choice of MF family for the hierarchical Gaussian mixture in Example 3.1 is

$$q(\pi, \mu_{1:K}, z_{1:N}) = \text{Dirichlet}(\pi; \alpha) \prod_{k=1}^{K} \mathcal{N}(\mu_k; m_k, \Sigma_k) \prod_{n=1}^{N} \text{Categorical}(z_n; \phi_n),$$

where $\{\alpha, m_{1:K}, \Sigma_{1:K}, \phi_{1:N}\}$ are the variational parameters.

[1] The name comes from mean-field theory in statistical physics: complex interactions are replaced by an average field acting on each variable. The fact that the optimal q_i is in fact an average, will become clearer in the next section.

Example 3.6 (MF underestimates the marginal variances)**.**
We will illustrate the ability of the MF family to capture marginals, but no dependencies between variables, and show how these captured marginals differ from the true ones. Let us consider $z = [z_1, z_2]^\top \in \mathbb{R}^2$, and the target

$$p(z \mid x) = \mathcal{N}\left(z \,\middle|\, \begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}, \begin{bmatrix}\sigma_{11} & \sigma_{12}\\ \sigma_{21} & \sigma_{22}\end{bmatrix}\right).$$

The optimal MF family for this target is[2] $q(z) = q_1(z_1)q_2(z_2)$, where

$$q_1(z_1) = \mathcal{N}(z_1 \mid m_1, s_1), \quad q_2(z_2) = \mathcal{N}(z_2 \mid m_2, s_2).$$

The optimal variational parameters $\theta = \{m_1, s_1, m_2, s_2\}$ can be found by

$$\theta^* = \arg\min \mathrm{KL}\left(q_1(z_1)q_2(z_2) \| p(z_1, z_2 \mid x)\right),$$

which admits a closed-form since the distributions are Gaussian. Expressing $q(z) = \mathcal{N}(z \mid m, S)$ with

$$m = \begin{bmatrix}m_1\\ m_2\end{bmatrix}, \qquad S = \begin{bmatrix}s_1 & 0\\ 0 & s_2\end{bmatrix},$$

we have

$$\mathrm{KL}\left(q\|p\right) = \frac{1}{2}\Big(\log\frac{\det\Sigma}{\det S} - 2 + \mathrm{tr}(\Sigma^{-1}S) + (m-\mu)^\top\Sigma^{-1}(m-\mu)\Big),$$

where $\mu = (\mu_1, \mu_2)^\top$ and $[\Sigma]_{ij} = \sigma_{ij}$.

Taking derivatives with respect to m_1 yields

$$\frac{\partial}{\partial m_1}\mathrm{KL}\left(q\|p\right) = \Big(\Sigma^{-1}(m-\mu)\Big)_1,$$

which vanishes if and only if $m_1 = \mu_1$. Next, differentiating with respect to s_1 gives

$$\frac{\partial}{\partial s_1}\mathrm{KL}\left(q\|p\right) = \frac{1}{2}\Big((\Sigma^{-1})_{11} - \frac{1}{s_1}\Big),$$

so the optimum satisfies

$$s_1 = \frac{1}{(\Sigma^{-1})_{11}}.$$

By symmetry, we also have $m_2 = \mu_2$ and $s_2 = \frac{1}{(\Sigma^{-1})_{22}}$.

Therefore, the optimal mean-field approximation matches the marginal means but not the marginal variances:

$$q_1(z_1) = \mathcal{N}\Big(z_1 \mid \mu_1,\ (\Sigma^{-1})_{11}^{-1}\Big), \qquad q_2(z_2) = \mathcal{N}\Big(z_2 \mid \mu_2,\ (\Sigma^{-1})_{22}^{-1}\Big).$$

To conclude, let us see how the recovered variances s_1 and s_2 relate to the true ones.

[2] We will not prove this, as it requires results from variational calculus.

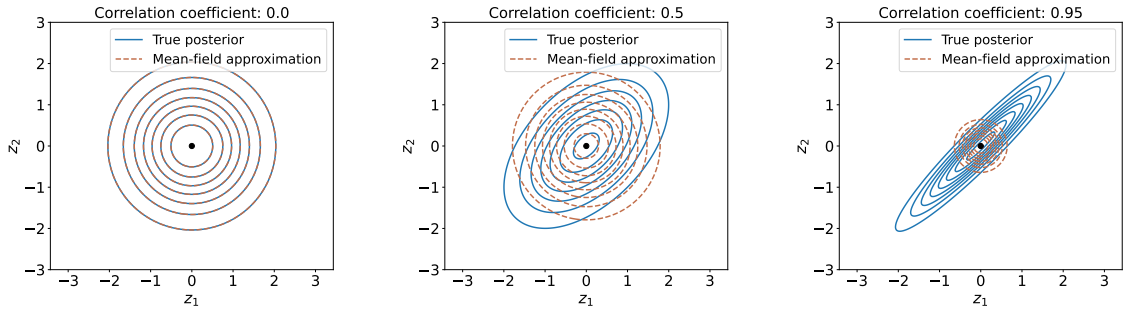

Figure 3.10: Mean-field variational approximation of a 2D Gaussian: uncorrelated and correlated cases.

Since the precision matrix can be calculated explicitly in this case, we have

$$s_1 = \frac{1}{(\Sigma^{-1})_{11}} = \left(\frac{\sigma_{22}}{\sigma_{11}\sigma_{22} - \sigma_{12}^2}\right)^{-1} = \sigma_{11}\frac{\sigma_{22}\sigma_{11} - \sigma_{12}^2}{\sigma_{22}\sigma_{11}} = \sigma_{11}(1-\rho^2),$$

where $\rho = \sigma_{12}/\sqrt{\sigma_{11}\sigma_{22}}$ is the correlation coefficient. Analogously, $s_2 = \sigma_{22}(1-\rho^2)$. Therefore,

- When z_1 and z_2 are uncorrelated ($\rho = 0$), the mean-field variance matches the true marginal variance, that is $s_1 = \sigma_{11}$.
- When $|\rho| > 0$, the mean-field variance is strictly smaller than the marginal variance, revealing the ignored dependencies by the mean-field assumption.
- When $|\rho| \to 1$, z_1 and z_2 are almost deterministically coupled, and we have $s_1, s_2 \to 0$, meaning that case cannot be captured by a mean-field approximation.

  As a consequence, this illustrates that mean-field variational inference systematically underestimates posterior uncertainty in the presence of strong correlations. Fig. 3.10 shows this phenomenon in the three cases indicated above.

Coordiante ascent variational inference (CAVI). Let us consider the *complete conditional* given by

$$p(z_j \mid z_{-j}, x),$$

where the subindex $\cdot_{-j}$ denotes all variables except the j-th one. Under the mean-field family, we cannot choose $q_j(z_j) = p(z_j \mid z_{-j}, x)$ due to the independence assumption. However, let us see that the optimal choice for the MF approximation in the general case has a very similar form.

Consider the update of a single coordinate of the latent variable z_j. Here,

$$\begin{aligned}\text{ELBO} &= \int q(z)\log p(x,z)\mathrm{d}z - \int q(z)\log q(z)\mathrm{d}z \\ &= \int q_j(z_j)\underbrace{\left(\int q_{-j}(z_{-j})\log p(x,z_j,z_{-j})\mathrm{d}z_{-j}\right)}_{\mathbb{E}_{q_{-j}}[\log p(x,z_j,z_{-j})]}\mathrm{d}z_j - \int q_j(z_j)\log q_j(z_j)\mathrm{d}z_j + C(q_{-j}),\end{aligned}$$

where $C(q_{-j})$ denotes terms that only depend on q_{-j}. Also, note that the expectation in

the argument admits the following proportional expression

$$\mathbb{E}_{q_{-j}}\left[\log p(x, z_j, z_{-j})\right] \propto_{q_j} \mathbb{E}_{q_{-j}}\left[\log p(z_j \mid x, z_{-j})\right], \tag{3.11}$$

where the symbol $\propto_{q_j}$ denotes proportionality wrt q_j only. Therefore, we have

$$\begin{aligned}
\text{ELBO} &\propto_{q_j} \int q_j(z_j)\left(\mathbb{E}_{q_{-j}}\left[\log p(z_j \mid x, z_{-j})\right]\right)\mathrm{d}z_j - \int q_j(z_j)\log q_j(z_j)\mathrm{d}z_j \\
&= -\int q_j(z_j)\log\frac{q_j(z_j)}{\exp\mathbb{E}_{q_{-j}}\left[\log p(z_j \mid x, z_{-j})\right]}\mathrm{d}z_j \\
&= \mathrm{KL}\left(q_j(z_j)\middle\|\exp\mathbb{E}_{q_{-j}}\left[\log p(z_j \mid x, z_{-j})\right]\right) + \text{const.}
\end{aligned} \tag{3.12}$$

Therefore, the optimal choice for $q_j(z_j)$ is $\exp\mathbb{E}_{q_{-j}}\left[\log p(z_j \mid x, z_{-j})\right]$.

Remark 3.12 (Mean-field interpretation)**.**
The mean-field family is named by analogy to mean-field models in statistical physics, where the effect that j-th particle receives from the remaining ones is expressed as the mean effect of the field.

Remark 3.13.
This optimal functional form for the MF family is not constrained to a particular expression for the density q_j or for the model p.

Remark 3.14 (CAVI update rule)**.**
Eq. (3.12) provides an update rule for q_j based on the remaining variational factors q_{-j}. Therefore, to compute the complete MF approximation, we first initialise all factors and implement Eq. (3.12) iteratively for $j = 1, \ldots, m$. Based on eq. (3.11), this update rule can be implemented as

$$\log q_j(z_j) \propto_{q_j} \mathbb{E}_{q_{-j}}\left[\log p(x, z)\right]. \tag{3.13}$$

Example 3.7 (CAVI for a two-component Gaussian mixture)**.**
Let $x_1, \ldots, x_N \in \mathbb{R}$ be observations and let $z_n \in \{1, 2\}$ be latent assignments. Consider the model with fixed parameters $(\pi, \mu_1, \mu_2, \sigma^2)$:

$$p(z_n = k) = \pi_k, \qquad p(x_n \mid z_n = k) = \mathcal{N}(x_n \mid \mu_k, \sigma^2), \quad k \in \{1, 2\}.$$

We approximate the posterior with a mean-field family

$$q(z_{1:N}) = \prod_{n=1}^{N} q_n(z_n), \qquad q_n(z_n = k) = \phi_{nk}, \ \sum_{k=1}^{2}\phi_{nk} = 1.$$

From eq. (3.13), CAVI updates each factor by

$$\log q_n^*(z_n) = \mathbb{E}_{q_{-n}}[\log p(x_{1:N}, z_{1:N})] + \text{const.}$$

Since z_n only appears in $\log p(z_n) + \log p(x_n \mid z_n)$, we obtain

$$\phi_{nk} \propto \exp\Big(\log\pi_k + \log\mathcal{N}(x_n \mid \mu_k, \sigma^2)\Big) = \pi_k\,\mathcal{N}(x_n \mid \mu_k, \sigma^2),$$

hence

$$\phi_{nk} = \frac{\pi_k \mathcal{N}(x_n \mid \mu_k, \sigma^2)}{\sum_{j=1}^{2} \pi_j \mathcal{N}(x_n \mid \mu_j, \sigma^2)}.$$

Thus, CAVI recovers the usual mixture responsibilities as a variational posterior.

Black-box variational inference (BBVI). To derive closed-form updates, CAVI relies on model-specific conjugacy thay may not hold in practice. This issue can be addressed with black-box variational inference (BBVI), a Monte Carlo approach to maximise the ELBO that treats the probabilistic model as a black box and thus is free from analytic updates.

Let $q_\lambda(z)$ denote a variational distribution parameterised by λ. As a function of the variational parameters, the ELBO can be written as

$$\mathrm{ELBO}(\lambda) = \mathbb{E}_{q_\lambda(z)}[\log p(x, z) - \log q_\lambda(z)] .$$

Under mild regularity conditions, the gradient of the ELBO can be expressed as

$$\begin{aligned}
\nabla_\lambda \mathrm{ELBO}(\lambda) &= \nabla_\lambda \int q_\lambda(z) \Big(\log p(x, z) - \log q_\lambda(z) \Big) \, dz \\
&= \int \nabla_\lambda q_\lambda(z) \Big(\log p(x, z) - \log q_\lambda(z) \Big) \, dz \qquad \Big(\text{since} \int q_\lambda(z) \nabla_\lambda \log q_\lambda(z) \, dz = 0\Big) \\
&= \int q_\lambda(z) \Big(\log p(x, z) - \log q_\lambda(z) \Big) \nabla_\lambda \log q_\lambda(z) \, dz \\
&= \mathbb{E}_{q_\lambda(z)} \Big[\Big(\log p(x, z) - \log q_\lambda(z) \Big) \nabla_\lambda \log q_\lambda(z) \Big] . \qquad (3.14)
\end{aligned}$$

This expression holds for any choice of variational family and any probabilistic model for which $\log p(x, z)$ can be evaluated pointwise.

In practice, the expectation above can be approximated using Monte Carlo samples $z^{(s)} \sim q_\lambda(z)$, yielding an unbiased estimator of the (stochastic) gradient. Therefore, the variational parameters λ can be optimised via, e.g., stochastic gradient ascent. One caveat of this estimator is that—yet generally applicable—it suffers from high variance and thus slow convergence.

With the appropriate variance reductions techniques, BBVI is applicable to complex, non-conjugate models, with attractive scalability properties. This provides a foundation to combine probabilistic modelling with automatic differentiation at scale, that is appealing for the constructions of modern probabilistic generative models.

3.5 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a probabilistic generative model for collections of discrete data, in particular, for text corpora. This model serves as an illustration for the practical application of variational inference of both global and local latent variables.

Consider a corpus of D documents, where document d is a *bag of words* comprising $w_{d1}, \ldots, w_{dN_d}$ words drawn from a vocabulary of size V. Additionally, assume K latent topics, each represented by a distribution over words.

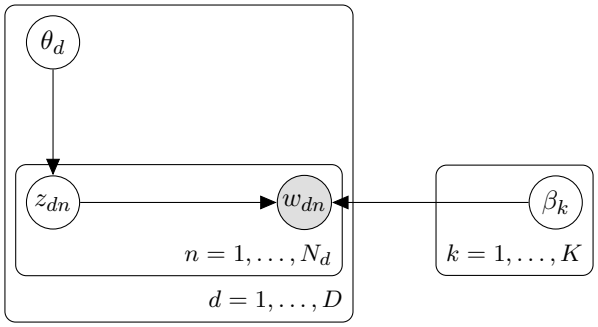

Figure 3.11: Graphical model of Latent Dirichlet Allocation (LDA). Global topic–word distributions β_k generate observed words w_{dn} through latent topic assignments z_{dn}, while document-specific topic proportions θ_d control topic usage within each document.

The generative model is presented in Fig. 3.11 and described as follows:

- For each topic $k = 1, \ldots, K$,

$$\beta_k \sim \text{Dirichlet}(\eta),$$

  where $\beta_k \in \Delta^{V-1}$, is the topic–word distribution.

- For each document $d = 1, \ldots, D$:

  – Draw topic proportions

$$\theta_d \sim \text{Dirichlet}(\alpha).$$

  – For each word $n = 1, \ldots, N_d$, draw its topic assignment and then the word:

$$z_{dn} \sim \text{Categorical}(\theta_d), \qquad w_{dn} \sim \text{Categorical}(\beta_{z_{dn}}).$$

Remark 3.15 (Latent and observed variables in the model)**.**
In the LDA model, the topic–word distributions β_k are the global latent variables, the topic proportions θ_d are document-level local latent variables, and the assignments z_{dn} are word-level local latent variables. Lastly, the words w_{dn} are the observed variables

The joint distribution factorises as

$$p(w, z, \theta, \beta) = \prod_{k=1}^{K} p(\beta_k) \prod_{d=1}^{D} p(\theta_d) \prod_{n=1}^{N_d} p(z_{dn} \mid \theta_d)\, p(w_{dn} \mid z_{dn}, \beta).$$

Given the observed words w, the posterior $p(z, \theta, \beta \mid w)$ is analytically intractable due to the coupling between topic assignments, topic proportions, and topic–word distributions. Therefore, the posterior can be approximated using a mean-field family:

$$q(\beta, \theta, z) = \prod_{k=1}^{K} q(\beta_k) \prod_{d=1}^{D} q(\theta_d) \prod_{d=1}^{D} \prod_{n=1}^{N_d} q(z_{dn}),$$

where

$$q(\beta_k) = \text{Dirichlet}(\lambda_k), \quad q(\theta_d) = \text{Dirichlet}(\gamma_d), \quad q(z_{dn}) = \text{Categorical}(\phi_{dn}).$$

Therefore, using CAVI, each factor is updated by taking the expected log-joint distribution with respect to the remaining variables.

Word-level update. The topic assignment z_{dn} is given by

$$\log q^*(z_{dn} = k) = \mathbb{E}_{q(\theta_d,\beta)}[\log p(z_{dn} = k, w_{dn} \mid \theta_d, \beta)] + \text{const} \tag{3.15}$$

$$= \mathbb{E}_{q(\theta_d)}[\log \theta_{dk}] + \mathbb{E}_{q(\beta_k)}[\log \beta_{k,w_{dn}}] + \text{const} \tag{3.16}$$

$$\Rightarrow \quad \phi_{dnk} \propto \exp\Big(\mathbb{E}_{q(\theta_d)}[\log \theta_{dk}] + \mathbb{E}_{q(\beta_k)}[\log \beta_{k,w_{dn}}]\Big), \qquad k = 1, \ldots, K. \tag{3.17}$$

with normalisation $\sum_{k=1}^{K} \phi_{dnk} = 1$. Using properties of the Dirichlet distribution, these expectations can be calculated in closed form.

Document-level update. The topic proportions are given by

$$\gamma_{dk} = \alpha_k + \sum_{n=1}^{N_d} \phi_{dnk}, \qquad k = 1, \ldots, K,$$

which corresponds to adding the expected topic counts in document d to the prior parameter α.

Global update. Finally, the topic–word distributions is

$$\lambda_{kv} = \eta_v + \sum_{d=1}^{D} \sum_{n=1}^{N_d} \phi_{dnk}\, \mathbf{1}\{w_{dn} = v\}, \qquad k = 1, \ldots, K,\ v = 1, \ldots, V,$$

that is, the prior parameter η plus the expected number of times word v is assigned to topic k across the entire corpus.

LDA models each document (in a group of documents) as a mixture of topics, and VI estimates the (soft) topic assignments for each word, as well as document-level topic mixtures, and global topic–word distributions.

Week 4

Bayesian Nonparametrics

NB: This is based on (Orbanz, 2014, Ch. 1-2) and (Rasmussen & Williams, 2006, Ch. 2).

4.1 Motivation

Recall that, given a collection of observations $x_1, \dots, x_n$, the main objective of the learning problem is to identify the data-generating mechanism. In classical statistics, the first step to achieve this is by defining the set of possible data generators as follows.

Definition 4.1 (Statistical model)**.**
Let $\mathcal{X}$ be a the (observation) sample space. A statistical model is a family of probability distributions

$$\mathcal{P} = \{P_\theta : \theta \in \Theta\}$$

defined on $\mathcal{X}$, where Θ is a parameter space and θ is an unknown parameter indexing the data-generating distribution.

Then, by considering the observations $x_1, \dots, x_n$ as a realisation of a random variable with a distribution in the statistical model, learning reduces to the inverse problem related to identifying the distribution in $\mathcal{P}$ that generated the observations. The canonical procedure to choose this model is maximum likelihood.

Example 4.1 (Density estimation: Gaussian parametric model vs. empirical measure)**.**
Let $x_1, \dots, x_N$ be i.i.d. observations taking values in $\mathcal{X} = \mathbb{R}$.

Consider the statistical model given by

$$\mathcal{P}_{\text{Gauss}} = \Big\{P_{\mu,\sigma^2} : (\mu, \sigma^2) \in \mathbb{R} \times (0, \infty)\Big\}, \qquad p(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big).$$

The optimal (maximum likelihood) parameters for this statistical model are

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x_n, \qquad \widehat{\sigma^2} = \frac{1}{N}\sum_{n=1}^{N}(x_n - \hat{\mu})^2.$$

Alternatively, we can consider an assumption-free estimator of the data-generating

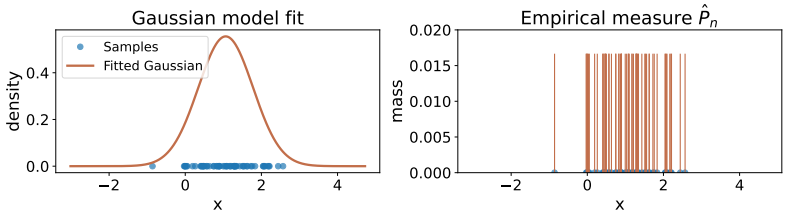

Figure 4.1: Density estimation: Gaussian model versus empirical measure.

distribution given by the empirical measure

$$\hat{P}_N = \frac{1}{N}\sum_{n=1}^{N}\delta_{x_n}, \tag{4.1}$$

where δ_x is the Dirac measure at x. Fig. 4.1 provides an illustration of both estimators.

Note that for the Gaussian statistical model, the parameter space is clearly defined: $(\mu, \sigma^2) \in \Theta = \mathbb{R} \times (0, \infty)$. However, it may not be clear what the parameters are in the second case. From eq. (4.1) notice that the parameters are the locations of the diracs, which are equal to the observations. This implies that the statistical model is the space of all measures with an arbitrary number of diracs in $\mathbb{R}$. This implies that the number of parameters in the second case is infinite.

Remark 4.1.
We will call the statistical model $\mathcal{P}$ **parametric** if the parameter space is finite, and **nonparametric** is parameter space is infinite.

The role of the parameter. We can consider learning the parameter as a form of compression, where the information in the, say N, observations is summarised in the parameter $\theta \in \Theta$—see Fig. 4.2. The parametric representation will then capture the relevant patterns in the data and retain them for subsequent tasks such as performing predictions, discriminative tasks such as classification, and even storage. Critically, when the parameter space is finite (and the dimensionality is lower than the number of datapoints), this compression is *lossy*, meaning that some information in the data is inevitably lost when determining the parameter, and thus the complete observations cannot be fully recovered (or explained) from the parameters alone. Additionally, and from an intuitive perspective, for a fixed number of parameters, an increasing amount of observations will not provide a monotonically-increasing amount of information, as there is only enough capacity in a m-dimensional parametric model.

On the contrary, a non-parametric model (i.e., a statistical model with an infinite-dimensional parameter space) can represent a form of lossless data representation. This means that no information is lost when training the model. In such case, the distinction between the parameter space and the model space is usually blurred, as the parameter becomes the model itself. Though this sounds like a great way to learn models as its provide infinitely-flexible models that do not *saturate* and keep learning as we feed more data to them, their implementation in practice requires careful considerations to deal with computational complexity.

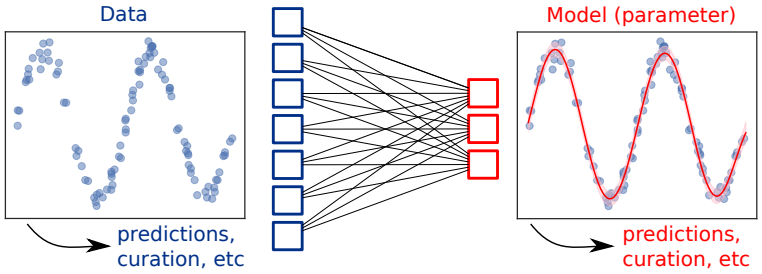

Figure 4.2: Illustration of parameter learning as compression.

Bayesian nonparametric models. Since in Bayesian ML we consider the parameter to be a random variable, we need to define a (prior) distribution over the parameter space Θ. This

Definition 4.2 (Bayesian statistical model)**.**
A Bayesian statistical model describes how data are generated, together with a way to encapsulate prior knowledge of the parameters before observing the data.

Formally, it consists of:

1. a *likelihood* $p(x \mid \theta)$, which specifies how the data x are distributed for a given parameter value θ, and

2. a *prior distribution* $\pi(\theta)$, which represents our beliefs about plausible values of θ before seeing any data.

Together, the likelihood and the prior define a joint density

$$p(x, \theta) = p(x \mid \theta)\, \pi(\theta),$$

which describes both how the data are generated and how the parameters are distributed.

Therefore, under the Bayesian paradigm, learning the model no longer refers to finding the best parameter in the parameter space (and, as a consequence the best model in the statistical model) as in the classical setup outlined above. Instead, we are now interested in finding the posterior distribution over the parameter

$$p(\theta | x_1, \ldots, x_N). \tag{4.2}$$

As a consequence, the parameter remains uncertain given a finite number of observations, but this uncertainty is reduced as we see the observations.

This week we will focus on Bayesian models given by a infinite-dimensional parameter space, where the main challenge will need to define priors in such spaces. Recall that a distribution on an infinite-dimensional space, or equivalently, a collection of random variables indexed by such space is a *stochastic process* with paths in Θ. In particular, we will focus on two such models: The Dirichlet process, which is a prior over probability distributions, and the Gaussian process which is a prior over functions.

Warning: The Dirichlet process is not part of the 2025/26 version of this course. However, a brief (draft) presentation is kept in the lecture notes for completeness; note that this draft is not complete and might have some notational incosisntencies. Interested reader are recommended to see (Orbanz, 2014) for a clear presentation of the Dirichlet process in ML.

4.2 The Dirichlet process

Bayesian mixture model We are very familiar at this stage with the hierarchical GMM, let us consider a different view of this model. Denoting the assignment variable as $z \in \mathbb{N}$, the distribution the observation x generated by a the k-th cluster can be expressed as

$$p_k(x) = p(x \mid z = k), \tag{4.3}$$

where we have not assumed that this distribution is Gaussian. Furthermore, the probability for a given observation to be generated by such cluster, can be denotes as

$$g_k = \mathbb{P}\left(Z = k\right), \tag{4.4}$$

where $\sum_{\in \mathbb{N}} c_k = 1, c_k \geq 0$, $\forall k \in \mathbb{N}$. The resulting model, given by

$$p(x) = \sum_{k \in \mathbb{N}} c_k p(x \mid z = k). \tag{4.5}$$

We will refer to this distribution as a mixture model. Furthermore, when there is only a finite number of probabilities c_k, we will say that this is a finite mixture.

The set of all sequences of the form $\{c_k\}_{k \in \mathbb{N}}$ is called *simplex*, denoted by

$$\Delta \stackrel{\text{def}}{=} \left\{ \{c_k\}_{k \in \mathbb{N}} \middle| \sum_{k \in \mathbb{N}} c_k = 1, c_k \geq 0 \right\}. \tag{4.6}$$

We can also assume that the cluster densities are in a parametric model over $\mathcal{X}$, given by $\{p(\cdot \mid \phi) | \phi \in \Phi\}$. This way, the assignment variable informs the chosen parameters for the conditional probabilites. This yields the following expression for the generative model

$$p(x) = \sum_{k \in \mathbb{N}} c_k p(x \mid \phi). \tag{4.7}$$

This allows us to express the following interpretation: Consider a discrete (atomic) probability measure θ on the parameter space Φ, given by:

$$\theta(\cdot) = \sum_{k \in \mathbb{N}} c_k \delta_{\phi_k}(\cdot). \tag{4.8}$$

This measure assigns probability to atoms in the parameter space and is referred to as

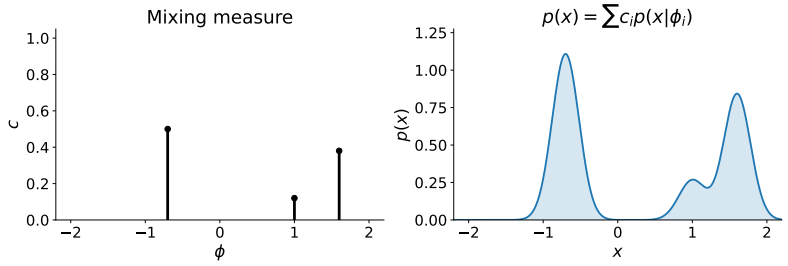

Figure 4.3: Illustration of a mixing measure (left) and the mixing model (right).

the mixing measure—see Fig. 4.3. This is because we can express the PGM as

$$p(x) = \int_{\Phi} p(x \mid \phi)\theta(\phi)\mathrm{d}\phi \tag{4.9}$$

$$= \int_{\Phi} p(x \mid \phi) \sum_{k\in\mathbb{N}} c_k \delta_{\phi_k}(\phi)\mathrm{d}\phi \tag{4.10}$$

$$= \sum_{k\in\mathbb{N}} c_k p(x \mid \phi_k). \tag{4.11}$$

Remark 4.2.
The statistical model used in clustering can be written as the mixture model above. This implies that the parameter of such model is a discrete probability distribution (called the mixing distribution).

Having identified the parameter space of the mixture model, let us now consider a Bayesian mixture model, that is, a random mixing measure

$$\Theta = \sum_{k\in\mathbb{N}} C_k \delta_{\Phi_k}. \tag{4.12}$$

Therefore, the prior of a Bayesian mixture is the distribution of the random mixing law Θ. Constructing this prior is not difficult. First, we can to define a prior for the parameters of the components, that is,

$$\Phi_1, \Phi_2, \ldots, \sim_{\text{i.i.d.}} G, \tag{4.13}$$

and independent of C_k. Then to sample the weights we cannot consider indepenence, that is the collection elements of the sequence $\{c_k\}_{k\in\mathbb{N}}$ needs to add up to one. However, in the finite case, we can simply sample K i.i.d. elements V_k form [0,1], and then define

$$C_k \overset{\text{def}}{=} \frac{V_x}{V_1 + \ldots + V_K}. \tag{4.14}$$

Remark 4.3.
When the RVs V_k follow a gamma distribution, C_k follow a Dirichlet distribution.

Stick-breaking construction. The above construction works well for finite k but very often we will want to define the mixture over infinite components (Why?). For an infinite number of components, the sum of i.i.d. variables $V_1 + V_2 + \ldots$ will diverge almost surely.

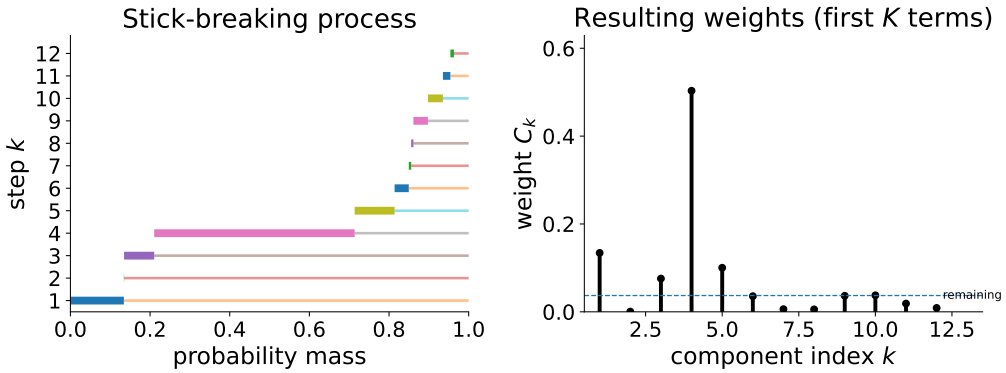

Figure 4.4: Illustration of a the stick breaking process.

Sampling an infinite number of parameters $\Phi_k \sim G$ is not problematic. For the mixing weights C_k, we can consider the stick-breaking construction, which provides a simple way to generate an infinite sequence of non-negative weights that sum to one.

Imagine a stick of length 1 representing total probability mass. The idea is to break this stick an infinite number of times, where the remaining pieces of the stick will be the c_k.

At step k, we:

1. break off a fraction $V_k \in (0, 1)$ of the remaining stick,
2. assign this piece length C_k to component k,
3. keep the rest for future components.

Formally, let $|I_1| = 1$. For $k = 1, 2, \ldots$:

$$V_k \sim H, \qquad C_k = |I_k|\, V_k, \qquad |I_{k+1}| = (1 - V_k)\, |I_k|.$$

The weights $(C_k)_{k \geq 1}$ are non-negative and satisfy

$$\sum_{k=1}^{\infty} C_k = 1 \quad \text{almost surely.}$$

This construction allows us to define mixture models with infinitely many components, while ensuring that the total probability mass remains finite and well defined.

The stick-breaking procedure allows for constructing a mixing measure with infinite components by choosing a specific distribution for the components's parameters G. Choosing H above as a beta distribution we have:

Definition 4.3 (Dirichlet process)**.**
Consider $\alpha > 0$ a concentration parameter, and G a probability measure on Φ. The

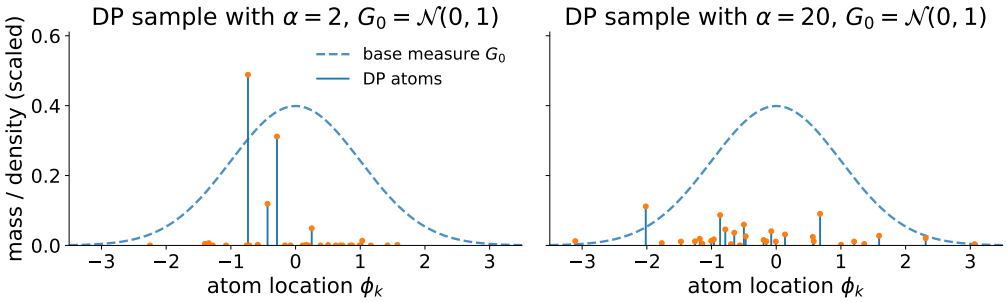

Figure 4.5: Draws from two DPs with a standard Gaussian as base measure, and different concentration parameters.

random discrete probability measure Θ generated by

$$V_1, V_2, \ldots, \sim_{\text{i.i.d.}} \text{Beta}(1, \alpha) \quad \text{and} \quad C_k \stackrel{\text{def}}{=} V_k \prod_{j=1}^{k-1} (1 - V_k) \tag{4.15}$$

$$\Phi_1, \Phi_2, \ldots, \sim_{\text{i.i.d.}} G \tag{4.16}$$

is called a Dirichlet process (DP) with base measure G and concentration α.

Fig. 4.5 shows draws from two DPs with different concentration parameters.

Dirichlet process mixture model. Using a Dirichlet process as a prior on the mixing measure yields an infinite mixture model. Specifically, let

$$\Theta \sim \text{DP}(\alpha, G), \qquad \phi_i \mid \Theta \sim \Theta, \qquad x_i \mid \phi_i \sim p(x \mid \phi_i),$$

for $i = 1, \ldots, N$. Marginally, this defines a mixture model with a countably infinite number of components, where the number of components effectively used by the data grows with N.

Posterior of the Dirichlet process. A key property of the Dirichlet process is conjugacy. If

$$\Theta \sim \text{DP}(\alpha, G) \quad \text{and} \quad \phi_1, \ldots, \phi_N \mid \Theta \sim \Theta,$$

then the posterior distribution of Θ is again a Dirichlet process:

$$\Theta \mid \phi_1, \ldots, \phi_N \;\sim\; \text{DP}\left(\alpha + N,\; \frac{\alpha G + \sum_{n=1}^{N} \delta_{\phi_n}}{\alpha + N}\right).$$

This result shows that Bayesian updating under the Dirichlet process corresponds to combining the prior base measure with the empirical distribution of the observed atoms.

4.3 The Gaussian process

Recall the parametric linear regression model given by

$$y = \phi(x)^\top \mathbf{w} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

where $x \in \mathbb{R}^d$, $\phi : \mathbb{R}^d \to \mathbb{R}^m$ is a fixed feature map, and $\mathbf{w} \in \mathbb{R}^m$ is the unknown parameter.

Let us consider a set of observation inputs $x_1, \ldots, x_N$, and denote the vector representation for the input, features and corresponding outputs:

$$\mathbf{X} = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_N^\top \end{bmatrix} \in \mathbb{R}^{N \times d}, \qquad \mathbf{\Phi} := \begin{bmatrix} \phi(x_1)^\top \\ \phi(x_2)^\top \\ \vdots \\ \phi(x_N)^\top \end{bmatrix} \in \mathbb{R}^{N \times m} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N.$$

Given the Gaussian driving noise ε, the observations $\mathbf{y}$ are distributed according to

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{\Phi w}, \sigma^2 I_N). \tag{4.17}$$

Remark 4.4.
For a fixed map ϕ, the parameter space of this statistical model is $\{(w, \sigma) \in \mathbb{R}^m \times \mathbb{R}_+\}$.

From a Bayesian standpoint, let us consider a standard Gaussian prior on the weights,

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma_{\mathbf{w}}), \tag{4.18}$$

with $\Sigma_{\mathbf{w}}$ a positive definite matrix.

This is a standard latent-variable formulation as the models we have seen earlier. In fact, the marginal law of $\mathbf{y}$, given by

$$p(\mathbf{y} \mid \mathbf{X}) = \int p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}, \tag{4.19}$$

can be calculated analytically: eqs. (4.17) and (4.18) into (4.19) give

$$p(\mathbf{y} \mid \mathbf{X}) = \mathcal{N}\Big(\mathbf{y} \,\Big|\, 0,\ \mathbf{\Phi} \Sigma_{\mathbf{w}} \mathbf{\Phi}^\top + \sigma^2 I_N\Big).$$

Remark 4.5.
Due to the linear-Gaussian formulation of this model, we say that the (latent) parameter can be marginalised out (or integrated out) analytically.

Furthermore, by conjugacy, the posterior distribution of the weights is Gaussian,

$$p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}) = \mathcal{N}(\mathbf{w} \mid \mu, \Sigma),$$

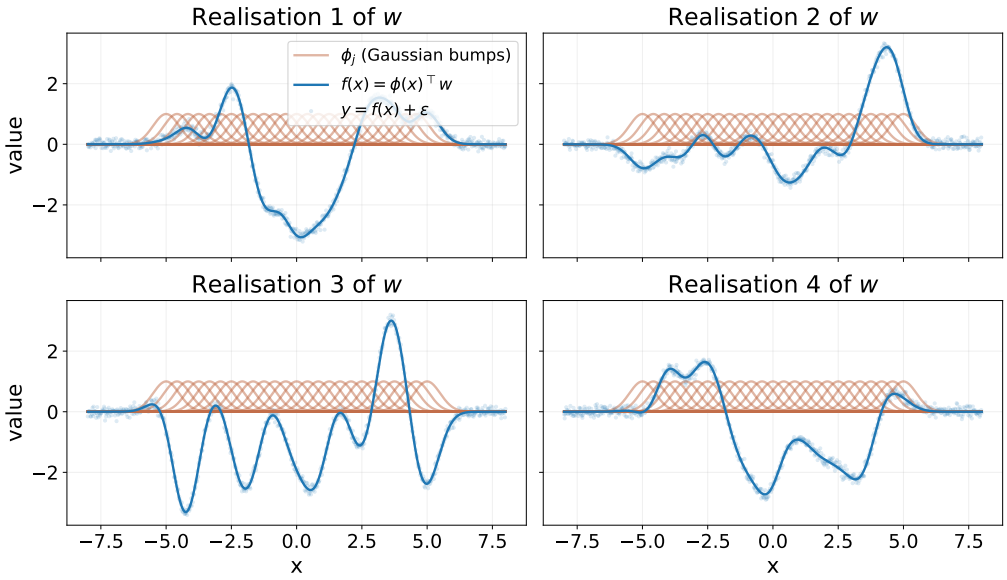

Figure 4.6: 4 realisations for the Bayesian linear model using Gaussian features.

with

$$\Sigma = \left(\Sigma_{\mathbf{w}}^{-1} + \frac{1}{\sigma^2}\Phi^\top\Phi\right)^{-1}, \qquad \mu = \Sigma\left(\frac{1}{\sigma^2}\Phi^\top\mathbf{y}\right).$$

Remark 4.6.

Observe that the posterior mean of $\mathbf{w}$ satisfies

$$\mu \stackrel{\text{def}}{=} \mathbb{E}[\mathbf{w} \mid \mathbf{y}, \mathbf{X}] = \arg\min_{\mathbf{w}} \left\{ \frac{1}{2\sigma^2}\|\mathbf{y} - \Phi\mathbf{w}\|_2^2 + \frac{1}{2}\mathbf{w}^\top\Sigma_{\mathbf{w}}^{-1}\mathbf{w} \right\},$$

meaning that $\mathbb{E}[\mathbf{w} \mid \mathbf{y}, \mathbf{X}]$ coincides with the ridge regression estimator (aka L_2-regularised least squares). In particular, when using an uninformative prior $p(\mathbf{w})$, the solution reduces to the ordinary least squares estimator.

Example 4.2.

Consider a *localised* version of this model where $x \in \mathbb{R}$, and the features ϕ are Gaussian bumps, i.e., $\phi_i = \exp\left(-\gamma(x - c_i)^2\right)$, with γ and c_i fixed hyperparameters. Fig. 4.6 shows four realisations of the model to illustrate its flexibility.

Observe that the posterior predictive distribution for the latent function value $f_* = \phi(x_*)^\top\mathbf{w}$, at a new test input x_* satisfies,

$$p(f_* \mid x_*, \mathbf{X}, \mathbf{y}) = \int p(f_* \mid x_*, \mathbf{w})\, p(\mathbf{w} \mid \mathbf{X}, \mathbf{y})\, \mathrm{d}\mathbf{w}.$$

To calculate this expression, recall that

$$f_* \mid x_*, \mathbf{w} = \phi(x_*)^\top\mathbf{w}, \qquad \mathbf{w} \mid \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\mu, \Sigma),$$

which allows to calculate the integral in closed form, yielding a Gaussian predictive distribution,

$$p(f_* \mid x_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\Big(f_* \,\Big|\, m_*,\ v_*\Big),$$

with the mean and variance given by

$$m_* = \phi(x_*)^\top \mu = \phi_*^\top \Sigma_{\mathbf{w}} \Phi^\top K^{-1} \mathbf{y}, \tag{4.20}$$

$$v_* = \phi(x_*)^\top \Sigma\, \phi(x_*) = \phi_*^\top \Sigma_{\mathbf{w}} \phi_* \; - \; \phi_*^\top \Sigma_{\mathbf{w}} \Phi^\top K^{-1} \Phi \Sigma_{\mathbf{w}} \phi_*, \tag{4.21}$$

where we have denoted $\phi_* \coloneqq \phi(x_*) \in \mathbb{R}^m$ and

$$K \;\coloneqq\; \Phi \Sigma_{\mathbf{w}} \Phi^\top + \sigma^2 I_N \in \mathbb{R}^{N \times N}.$$

4.4 Construction of the GP

Definition 4.4 (Gaussian process)**.**

A *Gaussian process* (GP) is a stochastic process $\{f(x) : x \in \mathcal{X}\}$ such that for any finite set of inputs $x_1, \ldots, x_N \in \mathcal{X}$, the random vector

$$\Big[f(x_1), \ldots, f(x_N)\Big] \in \mathbb{R}^N$$

is multivariate Gaussian. A GP is fully specified by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a covariance (kernel) function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and denoted by

$$f \sim \mathcal{GP}\left(m(\cdot), k(\cdot,\cdot)\right).$$

This means that for any $x_1, \ldots, x_N$,

$$\Big[f(x_1), \ldots, f(x_N)\Big] \sim \mathcal{N}\Big(\Big[m(x_1), \ldots, m(x_N)\Big],\; \Big[k(x_i, x_j)\Big]_{i,j=1}^{N}\Big).$$

The model above, defined by $f(x) \coloneqq \phi(x)^\top \mathbf{w}$ with the prior $\mathbf{w} \sim \mathcal{N}(0, \Sigma_{\mathbf{w}})$, is a GP with

$$m(x) = 0, \qquad k(x, x') = \phi(x)^\top \Sigma_{\mathbf{w}}\, \phi(x').$$

Remark 4.7 (Zero-mean GPs)**.**

We will consider zero-mean GPs only, since the feature map ϕ can be extended with a constant coordinate, that is $\tilde{\phi}^\top = [c, \phi^\top], c \in \mathbb{R}$, which produces a constant term carrying the first coordinate of the parameter $w_1 c$. Therefore, this term can be considered as a trainable mean.

Furthermore, considering the observation noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, we have that y is also a GP:

$$y(\cdot) \sim \mathcal{GP}\left(0, k(x, x') + \sigma^2 \mathbf{1}\{x = x'\}\right).$$

For an input-output observation dataset $\{\mathbf{X}, \mathbf{y}\}$, define $K = \Phi \Sigma_{\mathbf{w}} \Phi^\top \in \mathbb{R}^{N \times N}$, i.e.

$$K_{ij} = k(x_i, x_j) = \phi(x_i)^\top \Sigma_{\mathbf{w}}\, \phi(x_j), \qquad i, j \in \{1, \ldots, N\}.$$

Then the (GP) likelihood admits the explicit density

$$p(\mathbf{y} \mid \mathbf{X}) = \frac{1}{(2\pi)^{N/2}\,|K + \sigma^2 I_N|^{1/2}} \exp\!\left(-\frac{1}{2}\,\mathbf{y}^\top (K + \sigma^2 I_N)^{-1}\mathbf{y}\right). \tag{4.22}$$

Remark 4.8 (Inducing similarties through ϕ)**.**
The covariance of the GP reveals a *similarity-based structure* controlled by inner products between the chosen feature map ϕ (rotated by $\Sigma_{\mathbf{w}}$). Therefore, different notions of similarity can be induced by appropriate choices of ϕ. For instance, taking $\Sigma_{\mathbf{w}} = I$ and $\phi(x) = x/\|x\|$ yields a covariance that is maximised when the inputs are colinear. This motivates the use of high-dimensional (and possibly trainable) feature maps ϕ to extract representations that are relevant for the task.

Remark 4.9 (Cubic computational cost)**.**
Training and inference under the GP model, i.e. evaluating the marginal likelihood and making predictions, requires inversion of the $N \times N$ covariance matrix $K + \sigma^2 I_N$. As a consequence, the computational complexity scales as $\mathcal{O}(N^3)$ and depends on the number of data points N, but not explicitly on the feature dimension m.

Remark 4.10 (Evidence for the kernel trick)**.**
Although the model may be expressed in terms of feature maps (which we would like to be high-dimensional) all interactions between data points occur exclusively through inner products $\phi(x_i)^\top \Sigma_{\mathbf{w}} \phi(x_j)$. Hence, individual features never appear alone but only through their pairwise similarities. This observation calls for exploiting the kernel trick to represent the model.

Remark 4.11 (Avoiding degeneracy in GPs)**.**
Considering m basis functions (the dimension of ϕ) and $N > m$ observations, the covariance matrix $K = \Phi \Sigma_{\mathbf{w}} \Phi^\top$ is rank-deficient. In this case, the Gaussian likelihood is degenerate in the noiseless case, and the GP does not have a density. While the addition of observation noise $\sigma^2 I_N$ ensures invertibility, a well-posed GP that admits a large number of observations N without degeneracy requires feature maps ϕ of (effectively) infinite dimension. This motivates infinite-dimensional feature representations.

To construct a GP with an infinite-dimensional feature map and use it in practice, the explicit treatment of such a map should be bypassed altogether. This is feasible, since the likelihood of the GP, defined in eq. (4.22), only depends on the map via the covariance entries $k(x_i, x_j) = \phi(x_i)^\top \Sigma_{\mathbf{w}}\, \phi(x_j)$.

Remark 4.12 (The kernel trick)**.**
Specifying a GP directly by choosing a kernel function k, rather than an explicit feature map ϕ, requires choosing k to be positive definite. By *Mercer's theorem*, any continuous, symmetric, positive definite kernel admits a (possibly infinite-dimensional) feature representation

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}},$$

for some Hilbert space $\mathcal{H}$. Thus, working directly with positive-definite kernels im-

plicitly defines a corresponding feature map without ever constructing it explicitly.

Example 4.3 (Polynomial feature map)**.**
Let $x \in \mathbb{R}^d$ and consider the polynomial kernel of degree p,

$$k(x, x') = (x^\top x' + c)^p, \qquad c \geq 0. \tag{4.23}$$

This kernel corresponds to a finite-dimensional feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ whose components consist of all monomials in the entries of x of total degree up to p. For clarity, consider the case $d = 2$ and $p = 2$, with $x = (x_1, x_2)^\top$. One valid choice of feature map is

$$\phi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \\ \sqrt{2c}\, x_1 \\ \sqrt{2c}\, x_2 \\ c \end{bmatrix}. \tag{4.24}$$

Then the inner product expands as

$$\langle \phi(x), \phi(x') \rangle = (x_1 x_1' + x_2 x_2')^2 + 2c(x_1 x_1' + x_2 x_2') + c^2 = (x^\top x' + c)^2 = k(x, x').$$

As a consequence, implementing a GP with the kernel in eq. (4.23) is equivalent to implementing a (Gaussian) Bayesian linear regression model with the feature in eq. (4.24).

Example 4.4 (Gaussian feature map)**.**
Consider the squared exponential (Gaussian) kernel

$$k(x, x') = \sigma^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2} \right).$$

This kernel admits an explicit infinite-dimensional feature representation. By Bochner's theorem, any continuous stationary positive definite kernel can be written as the Fourier transform of a non-negative measure. In particular, the Gaussian kernel can be expressed as

$$k(x, x') = \sigma^2 \int_{\mathbb{R}^d} e^{i\omega^\top (x - x')}\, p(\omega)\, d\omega, \qquad p(\omega) = \mathcal{N}\left(0, \ell^{-2} I_d\right).$$

This yields the (infinite-dimensional) feature map

$$\phi(x) : \omega \mapsto \sigma\, e^{i\omega^\top x}, \qquad \phi(x) \in L^2(p),$$

for which

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{L^2(p)}.$$

Thus, the RBF kernel corresponds to a linear model in an infinite-dimensional Hilbert space.

4.5 Implementing a GP

Choosing the kernel.

Remark 4.13 (Kernel as a similarity measure)**.**
Recall the kernel defines the inner product between the feature maps, as a consequence, its choice will be made on the basis of assessing similarity.

Remark 4.14 (Stationary kernels)**.**
If the kernel is a function of the difference of its arguments, that is, $k(x, x') = k(x - x')$, we will say that that the kernel is stationary. It is worth noting that stationary kernels make the covariance between points invariant under translations in the input space. Since the kernel models similarity between points, in the case of stationary kernels, the closer two points are, the more similar they are considered.

The usual choice for a GP kernel is the *Squared Exponential* (SE) kernel, presented in Example 4.4, given by

$$k_{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right). \tag{4.25}$$

Under the SE kernel, the correlation between two points decays smoothly and long-range correlations are difficult to model due to the tail behaviour of the Gaussian function. The parameters of the SE are the *lengthscale* $\ell > 0$, which controls the memory or correlation length of the process, and the marginal variance σ^2, which controls the amplitude (or power) of the trajectories.

The *Rational Quadratic* (RQ) kernel, given by

$$k_{RQ}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha}, \tag{4.26}$$

is obtained by adding infinite SE kernels with different *lengthscales*[1], where α controls the spread of the considered lengthscales, and ℓ is a sort of reference lengthscale. Due to its additive structure, the RQ kernel is able to model short- and long-range correlations.

The *Periodic* kernel, given by

$$K_P(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell^2}\right), \tag{4.27}$$

allows for modelling periodic functions, where the parameter p controls the period of the functions. An extension of this kernel is the *locally-periodic* kernel

$$K_{LP}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \exp\left(-\frac{2\sin^2(\pi|x - x'|/p)}{\ell^2}\right), \tag{4.28}$$

obtained by multiplying a periodic kernel by an SE kernel, which allows for local periodic behaviour.

[1] Denoting $r = x - x'$, $k_{RQ}(r) = \int k_{SE}(r|\tau)\tau^{\alpha-1}\exp(-\alpha\tau/\beta)\mathrm{d}\tau$, where $\tau = \ell^{-2}$ and $\beta^{-1} = \ell^2$.

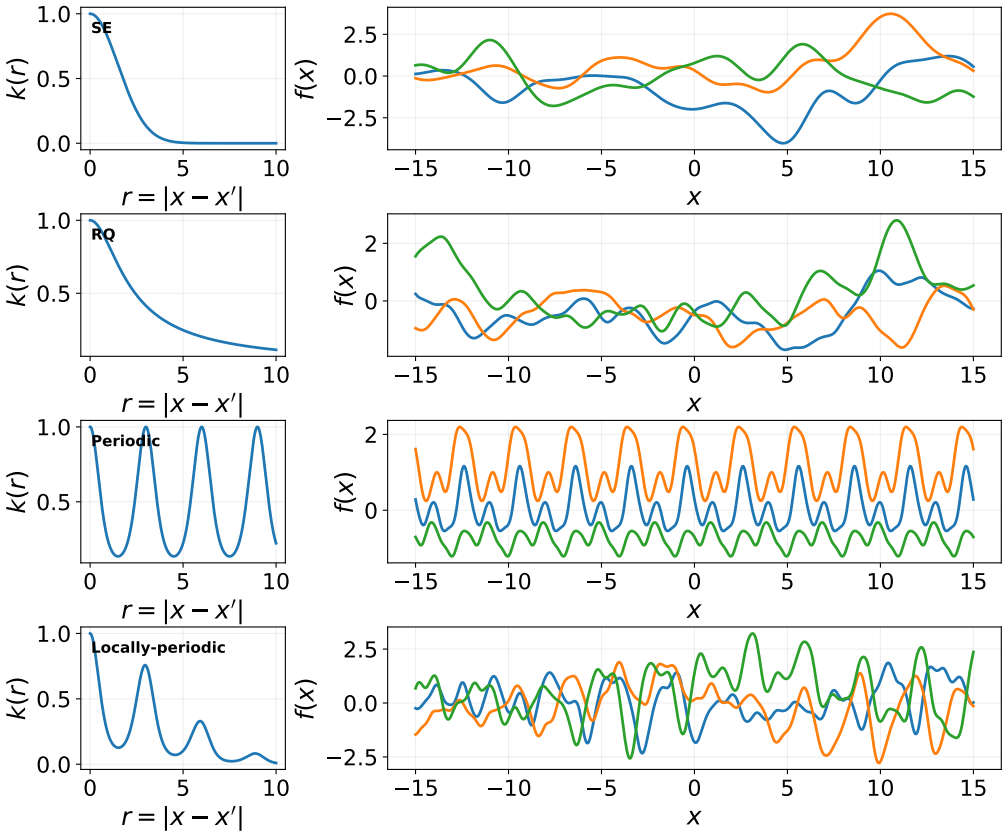

Figure 4.7: Covariance kernels (left) and GP samples (right). From top to bottom: Square exponential, rational quadratic, periodic, and locally-periodic kernels.

Fig. 4.7 shows these four kernels and samples drawn from them.

Remark 4.15.

When designing a Gaussian process and choosing a covariance function, one is not restricted to a fixed set of known kernels. Kernels can be combined to construct richer covariance structures that better capture the properties of the underlying process. In particular, the sum and the product of valid kernels are themselves valid kernels, and the exponential of a kernel, $\exp(k_1(\cdot,\cdot))$ with k_1 a valid kernel, also defines a valid covariance function.

Sampling from a Gaussian process. Since a GP is a generative model for infinite-dimensional functions, sampling from it might seem impossible at first. However, recall that a GP defines a joint Gaussian distribution over the function values *at any finite collection of input points*. See Fig. 4.8 for a graphical model representation.

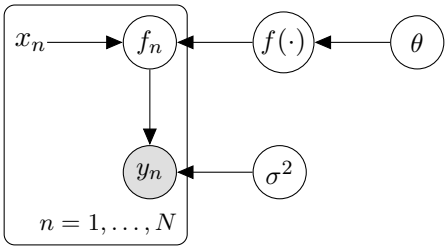

Figure 4.8: Graphical model of a GP model over a finite collection of inputs. The latent function f is drawn from a GP prior and evaluated at inputs x_n, producing latent values f_n, from which noisy observations y_n are generated.

Therefore, to sample a function from a GP, one first selects a finite set of input locations $x_1, x_2, \ldots, x_N$, computes the corresponding mean vector and covariance matrix, and then draws a sample from the resulting multivariate normal distribution. Each draw corresponds to one possible realisation of the random function evaluated at the specified locations. See Algorithm 1, which uses the Cholesky decomposition.

Algorithm 1 Sampling from a Gaussian Process

Require: Mean function $m(\cdot)$, kernel $k(\cdot, \cdot)$, inputs $x_1, \ldots, x_N$
Ensure: Sample $\mathbf{f} = [f(x_1), \ldots, f(x_N)]^\top$
1: Compute $\mathbf{m} = [m(x_1), \ldots, m(x_N)]^\top$
2: Compute $K \in \mathbb{R}^{N \times N}$ with $K_{ij} = k(x_i, x_j)$
3: Draw $\mathbf{z} \sim \mathcal{N}(0, I_N)$
4: Compute L such that $K = LL^\top$ (e.g. Cholesky)
5: Return $\mathbf{f} = \mathbf{m} + L\mathbf{z}$

Training a GP. Training a GP involves learning the kernel hyperparameters θ (and the noise variance σ^2) from data. Thanks to the GP's marginalisation property, the (infinite-dimensional) latent function f can be integrated out analytically, yielding the marginal likelihood

$$p(\mathbf{y} \mid \mathbf{X}, \theta, \sigma^2) = \mathcal{N}\Big(\mathbf{y} \,\Big|\, 0,\ K_\theta + \sigma^2 I_N\Big),$$

where $(K_\theta)_{ij} = k_\theta(x_i, x_j)$. Training is then formulated as the optimisation problem

$$\max_{\theta, \sigma^2} \log p(\mathbf{y} \mid \mathbf{X}, \theta, \sigma^2) = -\frac{1}{2}\mathbf{y}^\top (K_\theta + \sigma^2 I_N)^{-1}\mathbf{y} - \frac{1}{2}\log |K_\theta + \sigma^2 I_N| - \frac{N}{2}\log(2\pi),$$

which balances data fit and model complexity, and can be solved using gradient-based methods. Algorithm 2 presents the GP's training procedure.

Algorithm 2 Training a Gaussian Process via Marginal Likelihood

Require: Data $\{(x_n, y_n)\}_{n=1}^N$, kernel family $k_\theta(\cdot,\cdot)$, initial θ, σ^2
Ensure: Trained hyperparameters $\theta^\star, \sigma^{2\star}$
1: Form $\mathbf{X} = [x_1, \dots, x_N]$ and $\mathbf{y} = [y_1, \dots, y_N]^\top$
2: **repeat**
3: Compute the kernel matrix $K_\theta \in \mathbb{R}^{N\times N}$ with $(K_\theta)_{ij} = k_\theta(x_i, x_j)$
4: Set $C \leftarrow K_\theta + \sigma^2 I_N$
5: Evaluate the (log) marginal likelihood

$$\ell(\theta, \sigma^2) \leftarrow \log p(\mathbf{y} \mid \mathbf{X}, \theta, \sigma^2) = -\frac{1}{2}\mathbf{y}^\top C^{-1}\mathbf{y} - \frac{1}{2}\log|C| - \frac{N}{2}\log(2\pi)$$

6: Compute gradients $\nabla_\theta \ell(\theta, \sigma^2)$ and $\partial_{\sigma^2}\ell(\theta, \sigma^2)$
7: Update θ, σ^2 using a gradient-based optimiser (e.g. gradient ascent)
8: **until** convergence
9: **return** $\theta^\star \leftarrow \theta,\ \sigma^{2\star} \leftarrow \sigma^2$

Posterior computation. After training, the GP posterior predictive is also tractable thanks to the marginalisation property: The conditional distribution of the latent function value f_* at location x_*, conditional to the training set $\mathbf{X}, \mathbf{y}$ is also Gaussian

$$p(f_* \mid x_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}\Big(f_* \,\Big|\, m_*,\ v_*\Big),$$

with mean and variance given by

$$m_* = k_*^\top (K + \sigma^2 I_N)^{-1}\mathbf{y}, \tag{4.29}$$

$$v_* = k(x_*, x_*) - k_*^\top (K + \sigma^2 I_N)^{-1} k_*, \tag{4.30}$$

where $k_* = [k(x_1, x_*), \dots, k(x_N, x_*)]^\top$.

Exploring the posterior mean and variance in eqs. (4.29)-(4.30), we can draw the following observations. First, the posterior mean is a linear combination of the observations, with the weights given by the covariance between training and new samples. Second, the posterior variance features two terms: the first is the prior variance and the second one is a negative semidefinite term that's proportional to the covariance between the training and new points.

Algorithm 3 presents a general procedure for prediction using GPs.

Algorithm 3 Posterior Prediction in Gaussian Process Regression

Require: Training data $\{(x_n, y_n)\}_{n=1}^N$, kernel $k(\cdot,\cdot)$, noise variance σ^2, test inputs $x_1^*, \ldots, x_M^*$

Ensure: Posterior predictive mean $\mathbf{m}_* \in \mathbb{R}^M$ and covariance $\Sigma_* \in \mathbb{R}^{M\times M}$ for $\mathbf{f}_* = [f(x_1^*), \ldots, f(x_M^*)]^\top$

1: Form $\mathbf{y} = [y_1, \ldots, y_N]^\top$
2: Compute $K \in \mathbb{R}^{N\times N}$ with $K_{ij} = k(x_i, x_j)$
3: Compute $K_* \in \mathbb{R}^{N\times M}$ with $(K_*)_{i\ell} = k(x_i, x_\ell^*)$
4: Compute $K_{**} \in \mathbb{R}^{M\times M}$ with $(K_{**})_{\ell r} = k(x_\ell^*, x_r^*)$
5: Set $C \leftarrow K + \sigma^2 I_N$ and compute its Cholesky factorisation $C = LL^\top$
6: Solve $L\mathbf{v} = \mathbf{y}$ and $L^\top \alpha = \mathbf{v}$ (so $\alpha = C^{-1}\mathbf{y}$)
7: Compute posterior mean $\mathbf{m}_* \leftarrow K_*^\top \alpha$
8: Solve $LW = K_*$ for W (so $W = L^{-1}K_*$)
9: Compute posterior covariance $\Sigma_* \leftarrow K_{**} - W^\top W$
10: **return** $\mathbf{m}_*, \Sigma_*$

Plotting a GP. In general, GPs are plotted as follows: The mean is presented as a thick solid line, the ($\pm$ 2 std. dev.) credible interval is represented by a shaded area, samples from the GP are plotted as thin lines, and observations—when present—are plotted as disconnected dots. Fig. 4.9 shows the plots for a full GP implementation example, including: samples from the prior, observed data, and the GP posterior with untrained and trained hyperparameters. The code for these figures is available in `https://github.com/felipe-tobar/GP-lite`.

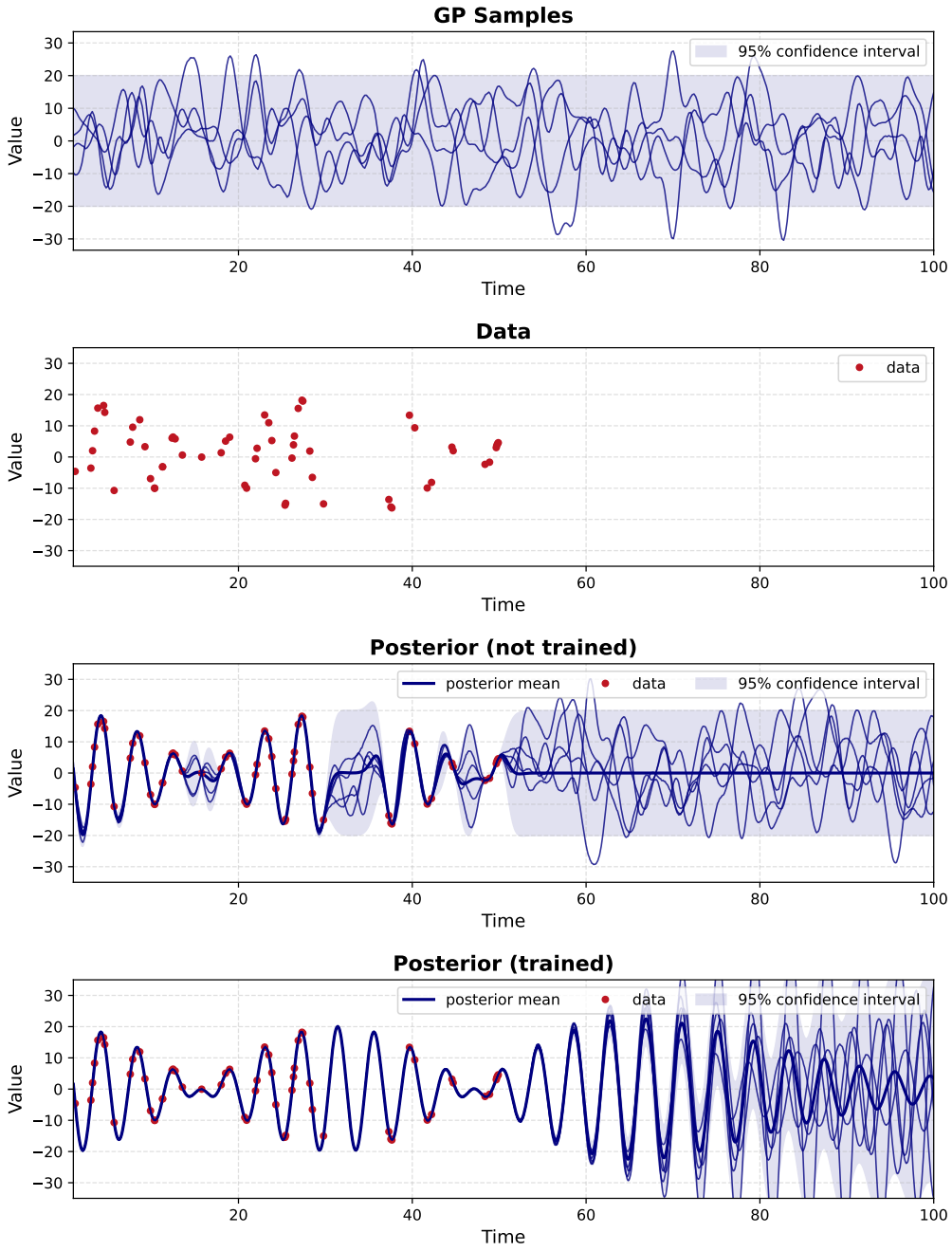

Figure 4.9: Plots from the full GP pipeline. From top to bottom: prior, data, posterior predictive (untrained) and posterior predictive (trained).

Week 5

Optimal Transport

NB: This is based on (Peyré & Cuturi, 2019, Ch. 1-2) and (Kingma & Welling, 2013).

5.1 Motivation

In many applications within generative modelling, we need to compare probability distributions in a way that respects the geometry of the sample space. Classical discrepancies such as ℓ_p distances between histograms or f-divergences (e.g. KL, JS) are often referred to as *vertical distances*, as they compare distributions in a pointwise fashion, ignoring the notion of distance between locations in the sample space.

- **Drawbacks of vertical distances.** Vertical distances become uninformative when the supports of the distributions do not overlap: two distributions with disjoint supports can be arbitrarily far apart or even incomparable, regardless of how close their supports are in space.
- **Sense of proximity and convergence.** When learning generative models, we expect distributions supported on nearby regions to be considered close. For instance, if $x_n \to x$ in $\mathbb{R}^d$, the Dirac measures δ_{x_n} should converge to δ_x. This notion of convergence is not captured by most classical divergences.
- **Lifting the base distance.** Optimal transport provides a principled way to lift a ground distance $d(\cdot, \cdot)$ on the sample space to a distance between probability distributions by explicitly accounting for the cost of transporting mass across space. This leads to geometrically meaningful distances on spaces of measures, known as Wasserstein distances.

5.2 The Monge problem

Let us consider the assignment problem, that is, how to allocate a group of items from a set of source locations to a set of target locations.

Example 5.1 (The vineyard problem)**.**
Consider a wine production company, where grapes produced from different vineyards must be transported to processing plants. An optimal assignment in this case must consider the distance between each vineyard and processing plant, as well

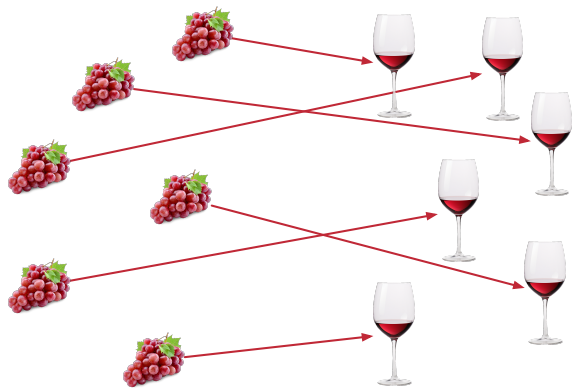

Figure 5.1: Illustration of the vineyard problem.

as their production and processing capacities respectively, to achieve the minimum transport cost. See Fig. 5.1 for an illustration.

We will consider both discrete and continuous distributions on the items to be assigned (or *transported*); we recall them as follows.

Definition 5.1 (Discrete measure with finite support)**.**
Let $\mathcal{X}$ be a set. A measure μ is *discrete with finite support* if there exist points $x_1, \dots, x_n \in \mathcal{X}$ and weights $w_1, \dots, w_n \geq 0$ such that, for any subset $A \subseteq \mathcal{X}$,

$$\mu(A) = \sum_{i=1}^{n} w_i \, \mathbf{1}_{\{x_i \in A\}}.$$

Equivalently,

$$\mu = \sum_{i=1}^{n} w_i \, \delta_{x_i}.$$

The measure μ is a probability measure if $\sum_{i=1}^{n} w_i = 1$.

Definition 5.2 (General measure with density)**.**
Let $\mathcal{X} \subseteq \mathbb{R}^d$. A measure μ is said to *admit a density* if there exists a nonnegative function $\rho : \mathcal{X} \to [0, \infty)$ such that, for any subset $A \subseteq \mathcal{X}$,

$$\mu(A) = \int_A \rho(x) \, dx.$$

The measure μ is a probability measure if

$$\int_{\mathcal{X}} \rho(x) \, dx = 1.$$

Let us consider two discrete distributions μ and ν, given by

$$\mu = \sum_{i=1}^{N} \mu_i \delta_{x_i}, \quad \text{and} \quad \nu = \sum_{j=1}^{M} \nu_j \delta_{y_j}. \tag{5.1}$$

The discrete Monge formulation finds a map, denoted T, that associates each source point x_i with a single point target y_j in order to allocate the mass from μ to ν. Note

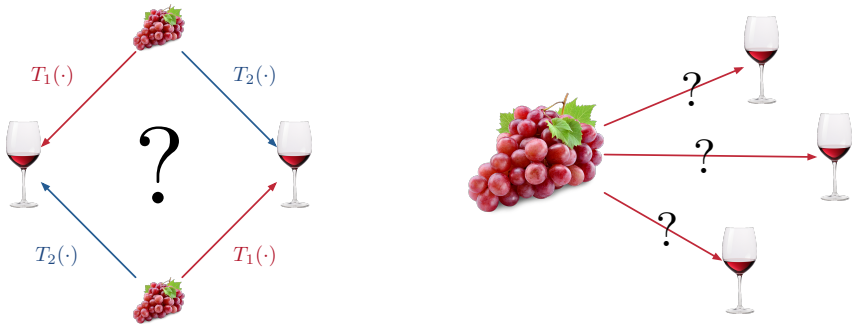

Figure 5.2: Monge's map may be non-unique (left) or may not even exist (right) in particular cases.

that more than one source point can be allocated to the same target, meaning that the map is surjective (onto the set with $\nu_j > 0$) yet not necessarily injective. The map $T : \{x_1, x_2, \dots, x_N\} \to \{y_1, y_2, \dots, y_M\}$, must verify

$$\nu_j = \sum_{i:T(x_i)=y_j} \mu_i, \quad \forall j = 1, \dots, M, \tag{5.2}$$

meaning that ν is the *push-forward measure* of μ through T.

Among the possible maps fulfilling eq. (5.2), Monge's formulation minimises a given transportation cost defined over the source-target pairs. Denoting the transport cost by $c : \{x_1, x_2, \dots, x_N\} \times \{y_1, y_2, \dots, y_M\} \to \mathbb{R}_+$, the optimal map is

$$T^* = \underset{T_{\#}\mu=\nu}{\arg\min} \sum_{i=1}^{N} c(x_i, T(x_i)). \tag{5.3}$$

Example 5.2 (The assignment problem)**.**
When $N = M$ and all masses are equal, that is, $\mu_i = \nu_j = 1/N, \forall i, j$, the mass conservation constraint implies that T is bijective.

Remark 5.1 (Existence and uniqueness)**.**
Monge maps may not exist in the general case. In particular, if $M > N$, a Monge map cannot exist since mass cannot be split. Furthermore, depending on the cost function, multiple maps can achieve the same optimal cost. See Fig. 5.2 for an illustration.

The Monge problem can also be formulated for general (continuous and/or discrete) measures. Let us consider two measures μ and ν supported on spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively, and a cost function $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$. In this case, the Monge map is given by

$$T^* = \underset{T_{\#}\mu=\nu}{\arg\min} \int_{\mathcal{X}} c(x, T(x)) \mathrm{d}\mu(x). \tag{5.4}$$

See Fig. 5.3 for an illustration.[1]

[1] Infinite thanks to Elsa Cazelles (IRIT, CNRS) for kindly sharing these beautiful `tikz` figures.

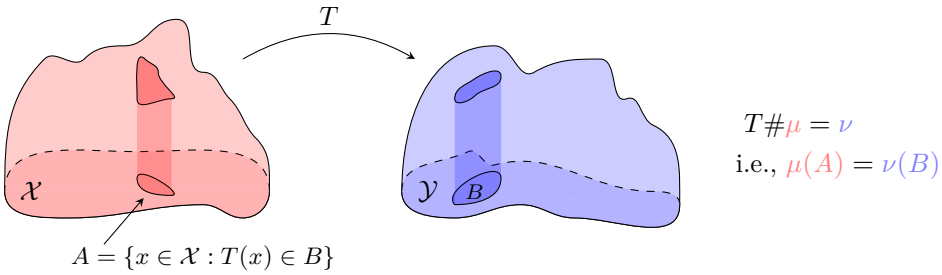

Figure 5.3: Illustration of the Monge map for continuous measures, adapted from (Thorpe, 2018).

We will be particularly interested when the measures involved are probability measures associated to random variables X and Y, this is because we want to rely on the theory of mass transport to construct generative models. However, note that in Monge's standard formulation this is very limiting, since we cannot deal with general cases beyond the histograms with the same number of atoms and uniform weights. In particular, modelling the association between the supports of μ and ν by the map T, does not allow for mass splitting.

5.3 Kantorovich relaxation

To allow for the mass of μ at a given location to be split and then transported to different locations to match ν, Kantorovich's formulation relaxes the deterministic nature of Monge's transport, considering a probabilistic transport instead.

We now return to the discrete measure setting with measures μ and ν. Rather than assuming the existence of a transport map, we will consider a transport plan (or coupling) $P \in \mathbb{R}^{N \times M}$, where $P_{i,j}$ describes the amount of mass that goes from x_i to y_j. The condition of the admissible maps fulfilling the pushforward condition, turns into the following condition for the set of admissible plans:

$$\Pi(\mu, \nu) \stackrel{\text{def}}{=} \{P \in \mathbb{R}_+^{N \times M}, \text{s.t.}, P\mathbb{1}_M = \mu \quad \text{and} \quad P^\top \mathbb{1}_N = \nu\}, \tag{5.5}$$

where $\mathbb{1}_M \in \mathbb{R}^M$ denotes a column vector of ones of dimension M. Note that, unlike the map formulation, the set of couplings always symmetric, i.e., $P \in \Pi(\mu, \nu) \iff P^\top \in \Pi(\nu, \mu)$.

Remark 5.2.

The set $\Pi(\mu, \nu)$ is defined by $N + M$ equality constraints and this is a convex polytope, that is, the convex hull of a finite set of matrices. The vertices of this polytope will be of particular interest.

Then, considering a cost matrix $C_{ij} = c(x_i, y_j)$, Kantorovich's optimal transport cost between discrete measures can be defined as

$$L(\mu, \nu) \stackrel{\text{def}}{=} \min_{P \in \Pi(\mu, \nu)} \langle C, P \rangle = \sum_{i=1}^{N} \sum_{j=1}^{M} C_{ij} P_{ij}. \tag{5.6}$$

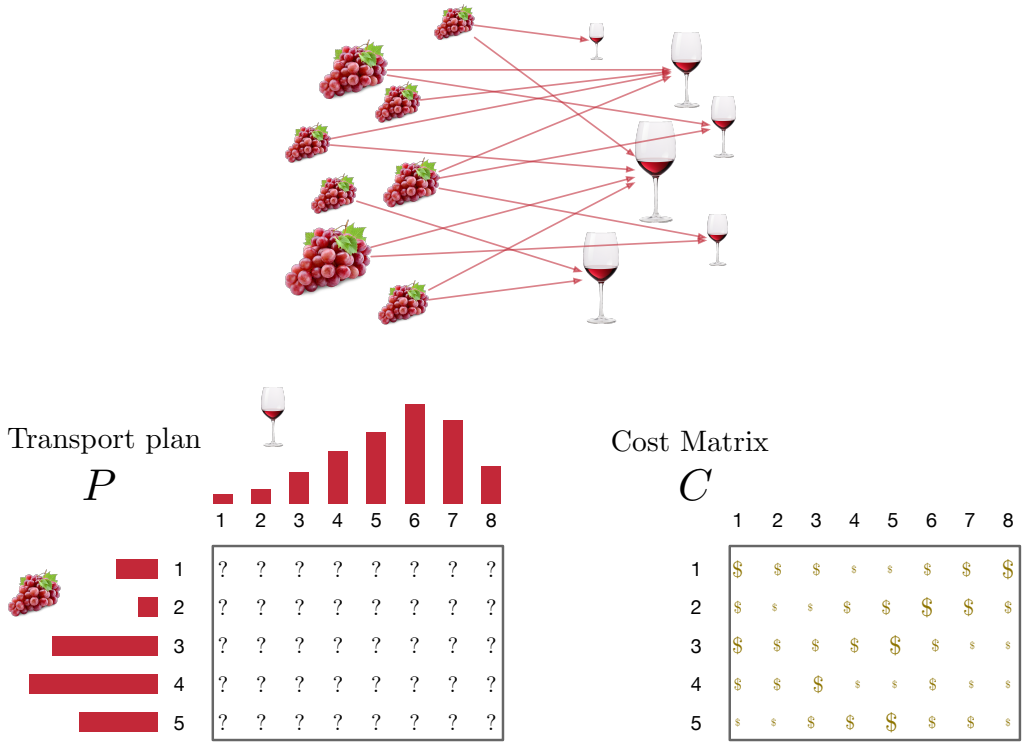

Figure 5.4: Illustration of Kantorovich's formulation: mass splitting (top) and transport plan with cost matrix (bottom).

Example 5.3 (Vineyards and wineries)**.**

To develop an intuition of Kantorovich's problem suppose an operator manages N vineyards and M wine processing plants. Each vineyard, indexed by i, produces μ_i units of grapes during the harvest season. These grapes must be transported in their entirety to the processing plants, where each plant j requires exactly ν_j units of grapes in order to operate at full capacity.

To transport grapes from vineyard i to processing plant j, the operator relies on a logistics provider that charges a cost $C_{i,j}$ per unit of grapes transported along that route. The pricing scheme is linear and uniform: shipping a units of grapes from vineyard i to plant j incurs a total cost of $a\,C_{i,j}$.

The goal of the operator is to determine how much grape mass should be sent from each vineyard to each processing plant so as to satisfy all supply and demand constraints while minimising the total transportation cost.

See Fig. 5.4 for an illustration.

Remark 5.3.

Observe that $\Pi(\mu, \nu)$ always has at least one element, given by $\mu \otimes \nu$, known as the independent coupling. Therefore, Kantorovich's formulation always has a solution.

Remark 5.4 (Equivalence between Monge and Kantorovich)**.**

Whenever the Monge problem admits a solution, the Kantorovich formulation is equivalent to it. More precisely, if there exists an optimal transport map $T : \mathcal{X} \to \mathcal{Y}$ pushing μ onto ν, then the associated transport plan $\pi_T \in \Pi(\mu, \nu)$, defined by

$$\pi_T(x, y) = \mu(x)\, \mathbf{1}_{\{y=T(x)\}},$$

is an optimal solution of the Kantorovich problem and both formulations attain the same optimal cost. In this case, the optimal Kantorovich plan is supported on the graph of the map T.

Conversely, when no optimal transport map exists, the Monge problem is ill-posed, while Kantorovich's relaxation remains well-defined and admits at least one solution, possibly involving mass splitting.

To extend Kantorovich's formulation to general (continuous and/or discrete) measures, let μ and ν be two measures supported on spaces $\mathcal{X}$ and $\mathcal{Y}$, respectively, and let $c : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ be a cost function. A *transport plan* is now a measure π on the product space $\mathcal{X} \times \mathcal{Y}$ whose marginals coincide with μ and ν, that is,

$$\int_{\mathcal{Y}} \pi(x, y)\, dy = \mu(x), \qquad \int_{\mathcal{X}} \pi(x, y)\, dx = \nu(y).$$

The Kantorovich optimal transport cost is then defined as

$$L(\mu, \nu) \stackrel{\text{def}}{=} \min_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\, d\pi(x, y), \tag{5.7}$$

where $\Pi(\mu, \nu)$ denotes the set of all admissible couplings between μ and ν. Unlike the Monge formulation, this problem always admits at least one solution under mild assumptions on c.

Theorem 5.1 (Brenier)**.**

Let μ and ν be probability measures on $\mathbb{R}^d$ with finite second moments. Assume that μ admits a density with respect to the Lebesgue measure, and consider the quadratic cost

$$c(x, y) = \frac{1}{2}\|x - y\|^2.$$

Then the Kantorovich problem between μ and ν admits a unique optimal solution. Moreover, this optimal transport plan is induced by a map $T : \mathbb{R}^d \to \mathbb{R}^d$ of the form

$$T(x) = \nabla\varphi(x),$$

where $\varphi : \mathbb{R}^d \to \mathbb{R}$ is a convex function. In particular, T is the unique optimal solution of the Monge problem.

Corollary 1 (Brenier's theorem in one dimension)**.**

Let μ and ν be probability measures on $\mathbb{R}$, and assume that μ admits a density. Consider the quadratic cost $c(x, y) = (x - y)^2$. Then the unique optimal transport map from μ to ν is given by the *monotone rearrangement*

$$T(x) = F_\nu^{-1}(F_\mu(x)),$$

where $F_\mu(x) = \mu((-\infty, x])$ and $F_\nu^{-1}(t) = \inf\{y \in \mathbb{R} : F_\nu(y) \geq t\}$ denote the cumulative distribution function of μ and the quantile function of ν, respectively. The corresponding Kantorovich optimal plan is supported on the graph of T.

5.4 Calculating OT

Particular case: Gaussian measures. Let $\mu = \mathcal{N}(m_\mu, \Sigma_\mu)$ and $\nu = \mathcal{N}(m_\nu, \Sigma_\nu)$ be Gaussian distributions on $\mathbb{R}^d$ with positive definite covariance matrices, and consider the quadratic cost $c(x, y) = \|x - y\|_2^2$. Since these measures admit densities, the Monge problem admits a unique solution. In this case, the optimal transport map has a closed-form expression and is affine, given by

$$T(x) = m_\nu + A(x - m_\mu), \qquad A = \Sigma_\mu^{-1/2} \left(\Sigma_\mu^{1/2} \Sigma_\nu \Sigma_\mu^{1/2}\right)^{1/2} \Sigma_\mu^{-1/2}.$$

The corresponding optimal transport cost, also known as the squared 2-Wasserstein distance between μ and ν, decomposes into a term measuring the distance between the means and a term measuring the discrepancy between the covariances:

$$W_2^2(\mu, \nu) = \|m_\mu - m_\nu\|_2^2 + \mathrm{Tr}\left(\Sigma_\mu + \Sigma_\nu - 2\left(\Sigma_\mu^{1/2} \Sigma_\nu \Sigma_\mu^{1/2}\right)^{1/2}\right).$$

In the one-dimensional case, this simplifies considerably: the optimal transport map reduces to a simple rescaling and translation, $T(x) = m_\nu + \frac{\sigma_\nu}{\sigma_\mu}(x - m_\mu)$, and the transport cost admits a closed-form expression in terms of the means and variances.

Particular case: 1-dimensional measures. Following from Brenier theorem, a relevant feature of optimal transport is that, in one dimension, the problem admits a closed-form solution. Let μ and ν be probability measures on $\mathbb{R}$, e.g., as illustrated in Fig. 5.5, and assume the L_p distance as the associated ground cost. The optimal transport map is given by the *monotone rearrangement*, which matches equal quantiles of the two distributions.

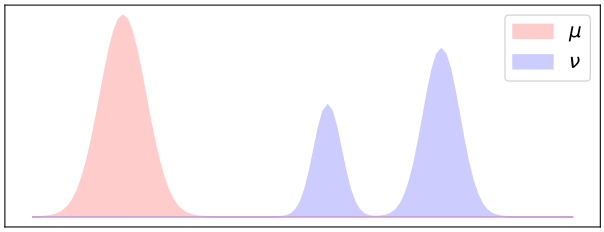

Figure 5.5: Two one-dimensional probability distributions. Optimal transport matches points with equal cumulative mass.

Denote by F_μ the cumulative distribution function of μ, and by $F_\mu^- : [0, 1] \to \mathbb{R}$ its inverse (quantile function), defined as

$$F_\mu^-(t) = \inf\{x \in \mathbb{R} : F_\mu(x) \geq t\}, \qquad t \in [0, 1].$$

Then, for any $p \geq 1$, the optimal transport cost between μ and ν admits the explicit expression

$$\mathrm{OT}^p(\mu,\nu) = \int_0^1 \left(F_\mu^-(t) - F_\nu^-(t)\right)^p dt = \|F_\mu^- - F_\nu^-\|^p_{L^p([0,1])}.$$

This representation, illustrated in Fig. 5.6, shows that one-dimensional optimal transport reduces to measuring the L^p distance between quantile functions.

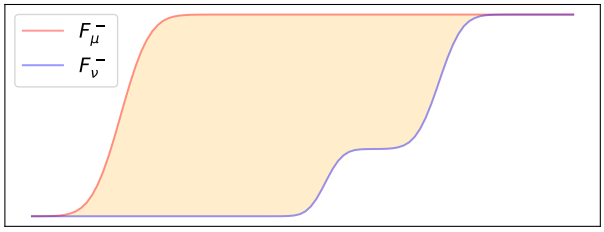

Figure 5.6: Quantile functions of the two distributions. The optimal transport cost corresponds to the L^p distance between them.

The same monotone matching principle applies to discrete one-dimensional measures. In this case, optimal transport is obtained by sorting the atoms of μ and ν and matching them in increasing order, as illustrated in Fig. 5.7.

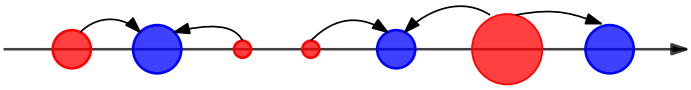

Figure 5.7: Optimal transport between discrete one-dimensional measures: atoms are matched according to their cumulative mass.

General case through linear programming. Kantorovich's problem is defined by the linear programme in eqs. (5.6) or (5.7). Recall that the feasible set of the discrete optimal transport problem is the convex and compact transport polytope $\Pi(\mu,\nu)$, defined by the linear equality and inequality constraints. Since the objective function $\langle C, P\rangle$ is linear in P, the optimal plan is found at an extreme point (vertex) of the polytope. As a consequence, OT plans (in the discrete setting) are typically sparse. See Fig. 5.8 for an illustration.

Remark 5.5 (Intuition into the simplex method)**.**

The simplex method is a classical algorithm for solving linear programs, usually used to solve Kantorovich's OT formulation. It operates by moving along the edges of the feasible polytope from one vertex to another, at each step decreasing the value of the objective function, until no adjacent vertex yields further improvement. Since the optimum of a linear program is attained at a vertex of the feasible set, the simplex method terminates at an optimal solution after visiting only a (typically small) subset of all vertices.

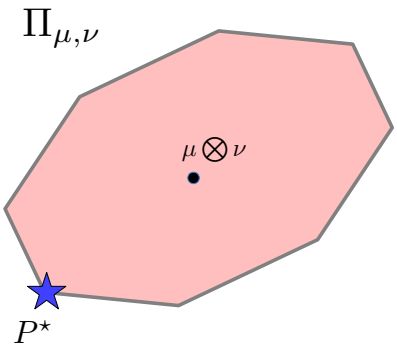

Figure 5.8: Illustration of the transport polytope and the attained optimal plan.

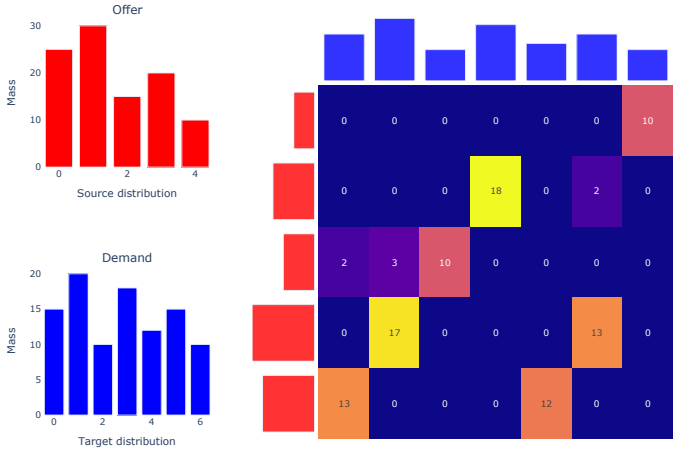

Figure 5.9: Computation of optimal plan (discrete): Source and target discrete distributions (left) and the optimal plan (right) for a given cost matrix. Observe how the plan is sparse, meaning that the mass at each source location is split into only a few target locations.

Algorithm 4 Discrete optimal transport via a linear program (no toolbox)

Require: Weights $\mu \in \mathbb{R}_+^n$, $\nu \in \mathbb{R}_+^m$ with $\sum_i \mu_i = \sum_j \nu_j$; cost matrix $C \in \mathbb{R}^{n \times m}$

Ensure: Optimal transport plan $P^\star \in \mathbb{R}_+^{n \times m}$ and cost $OT(\mu, \nu)$

1: **Define optimisation variables:** $P \in \mathbb{R}^{n \times m}$

2: **Form constraints (transport polytope):**

$$P \geq 0, \qquad P\mathbf{1}_m = \mu, \qquad P^\top \mathbf{1}_n = \nu$$

3: **Define objective:**

$$\min_P \langle C, P \rangle \stackrel{\text{def}}{=} \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} P_{ij}$$

4: **Solve the linear program** with any generic LP solver (simplex / interior-point)

5: **Return** $P^\star$ (an optimal feasible plan) and $OT(\mu, \nu) = \langle C, P^\star \rangle$

Algorithm 4 presents the procedure to solve the OT problem. For illustration, Figs. 5.9 and 5.10 show implementations of this algorithm.

Remark 5.6 (Drawbacks of the linear program)**.**

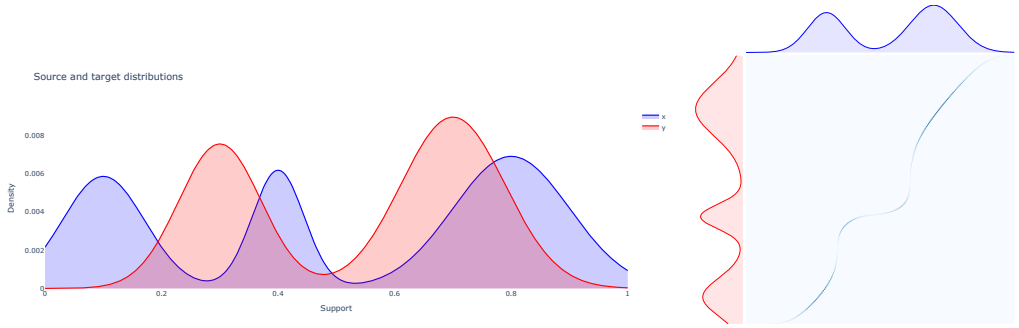

Figure 5.10: Computation of optimal plan between Gaussian mixtures using the L_2 cost (discrete, but more dense): Source and target discrete distributions (left) and the optimal plan (right) for a given cost matrix. Observe how the plan seems extremely sparse with the plans almost completely supported on the plot of a function. Though this a discrete example, since these mixtures of Gaussians are absolutely continuous, the OT cost is unique and equal to Monge's solution (Brenier thm), which can seen from the plot. Furthermore, also in line with Brenier thm, note that the transport is monotonic, meaning transported masses *do not cross paths*.

Kantorovich's formulation allows for a well-posed optimisation problem with attractive uniqueness and existence properties, and numerically stable solutions. However, there are two main drawback of addressing the computation of the OT via linear programming: first, the computational cost becomes prohibitive when the number of atoms grow beyond $\approx 10^4$. Second, solutions are sparse, quasi-deterministic, matrices given at the vertices of $\Pi(\mu, \nu)$.

Entropy-regularised OT (Sinkhorn). To address the computational limitations of linear programming approaches, a popular strategy is to add an entropic regularisation term to the Kantorovich problem, leading to fast iterative algorithms. This approach was popularised in machine learning by (Cuturi, 2013).

For $\alpha \in \mathbb{R}_+$, define

$$\Pi_\alpha(\mu, \nu) = \{\pi \in \Pi(\mu, \nu) : \mathrm{KL}\left(\pi \| \mu \otimes \nu\right) \leq \alpha\}, \tag{5.8}$$

and consider the constrained OT problem wrt the cost C:

$$\pi^\star_\alpha = \underset{\pi \in \Pi_\alpha(\mu,\nu)}{\arg\min} \langle C, \pi \rangle. \tag{5.9}$$

Fig. 5.11 shows an pictorial representation of this formulation.

Given that the optimisation variable in this setting is π, we can write

$$\begin{aligned} \mathrm{KL}\left(\pi \| \mu \otimes \nu\right) &= \sum_{i=1}^{N} \sum_{j=1}^{M} \pi_{i,j} \log \frac{\pi_{i,j}}{\mu_i \nu_j} \\ &\propto_\pi \sum_{i=1}^{N} \sum_{j=1}^{M} \pi_{i,j} \log \pi_{i,j} \\ &= -H(\pi), \end{aligned} \tag{5.10}$$

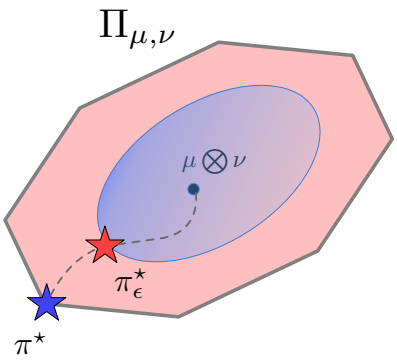

Figure 5.11: Illustration of the transport polytope $\Pi(\mu,\nu)$, the defined subset $\Pi_\alpha(\mu,\nu)$ and their respective optimal plans.

where $\propto_\pi$ denotes equality up to additive terms independent of π (since the marginals of π are fixed to be μ and ν).

As a consequence, the optimal plan in eq. (5.9) can be rewritten using a Lagrangian formulation as

$$\pi^\star_\alpha = \underset{\pi\in\Pi(\mu,\nu)}{\arg\min} \langle C,\pi\rangle - \frac{1}{\lambda}H(\pi), \tag{5.11}$$

where there is one, and only one, $\alpha \in \mathbb{R}_+$ for each $\lambda \in \mathbb{R}_+$.

The Lagrangian is given by

$$\mathcal{L}(\pi,f,g) = \sum_{i=1}^{n}\sum_{j=1}^{m}\left(\frac{1}{\lambda}\pi_{ij}\log\pi_{ij} + \pi_{ij}c_{ij}\right) + f^\top(\pi\mathbf{1}_m - \mu) + g^\top(\pi^\top\mathbf{1}_n - \nu),$$

and thus $\partial\mathcal{L}/\partial\pi_{ij} = 0$ yields

$$\pi_{ij} = e^{-1-\lambda f_i}\, e^{-\lambda c_{ij}}\, e^{-\lambda g_j},$$

meaning that the solution has the form

$$\pi^*_\lambda = \mathrm{diag}(u)\, K\, \mathrm{diag}(v),$$

where $K_{ij} = \exp(-\lambda C_{ij})$ (entrywise).

Finding u and v is not easy. However, (Sinkhorn, 1964) states that since the entries of K are positive, there exist (essentially unique) scaling vectors u, v such that $\mathrm{diag}(u)K\,\mathrm{diag}(v) \in \Pi(\mu,\nu)$. Therefore, u, v can be updated so that $\mathrm{diag}(u)K\,\mathrm{diag}(v)$ satisfies the marginal constraints, that is (Sinkhorn iterations),

$$u \leftarrow \mu./(Kv), \qquad v \leftarrow \nu./(K^\top u). \tag{5.12}$$

Fig. 5.12 shows an illustration of optimal transport plans found via linear programming and Sinkhorn.

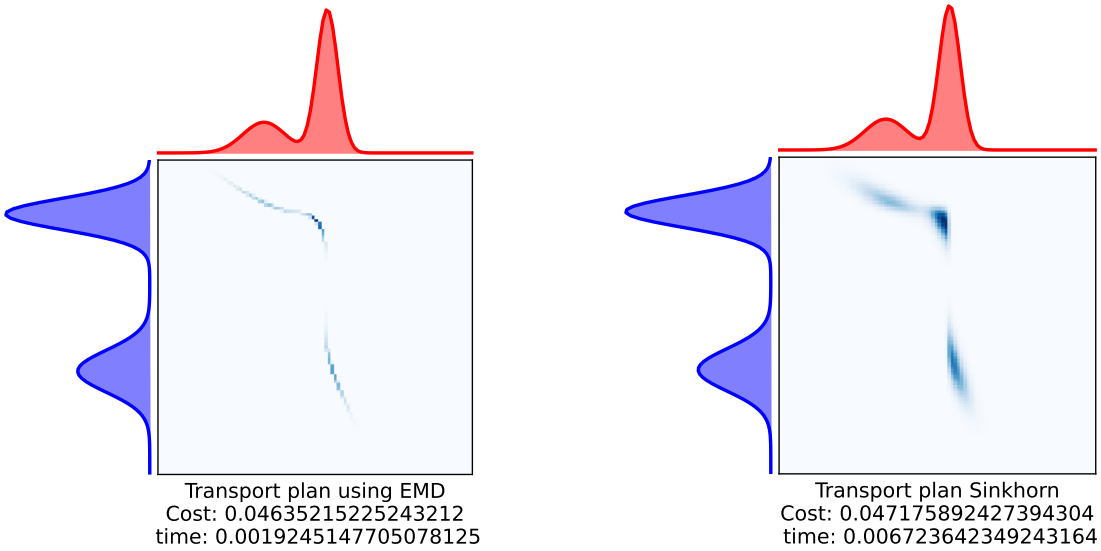

Figure 5.12: Optimal transport plans between 2-component Gaussian mixtures: linear programming (left) and Sinkhorn iterations (right).

5.5 Metric properties

The critical feature of OT theory in the context of generative modelling is the OT problem defines a distance between probability distributions. For this, it is necessary that the cost function takes the form $c(x, y) = d(x, y)^p$, where $d(\cdot, \cdot)$ is a distance defined on the support of the distributions involved, and $p \in \mathbb{N}$. Critically, OT is said to *lift* the distance d from the sample space to the space of distributions, thus translating the geometric properties from the sample to the distribution space.

Definition 5.3 (Wasserstein distance)**.**
Let $(\mathcal{X}, d)$ be a metric space and let μ and ν be probability measures on $\mathcal{X}$ with finite p-th moments. The *p-Wasserstein distance* between μ and ν is defined as

$$W_p(\mu, \nu) \stackrel{\text{def}}{=} \left(\inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} d(x, y)^p \, d\pi(x, y) \right)^{1/p}.$$

Remark 5.7 (Metric properties of the Wasserstein distance)**.**
For $p \geq 1$, the p-Wasserstein distance W_p defines a true metric on the space of probability measures with finite p-th moments: it is nonnegative, symmetric, vanishes if and only if the two measures coincide, and satisfies the triangle inequality. The first three properties follow directly from the definition, while the triangle inequality relies on the ability to *glue* optimal transport plans. Intuitively, transporting mass from μ to ν and then from ν to λ cannot be cheaper than transporting it directly from μ to λ: since one optimal plan defines the lowest-cost transport procedure, forcing the transport to visit any intermediate measure can only increase the overall cost.

Remark 5.8 (Choice of the ground metric)**.**

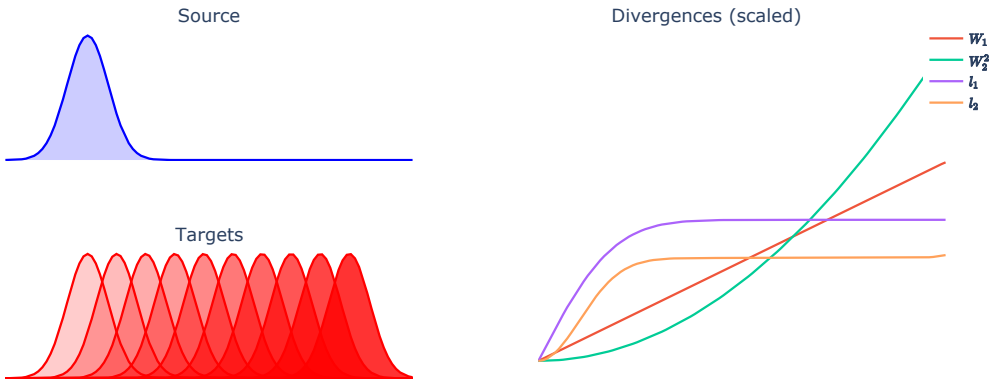

Figure 5.13: Comparison between W_p and l_p, $p \in \{1, 2\}$, for two unimodal distributions. Notice how the l_p distances become uninformative as the support between the source and target distributions becomes too far from each other; conversely, the W_p distances inherit the geometric property of the ground distance and keep an account of such separation.

When $\mathcal{X} \subseteq \mathbb{R}^d$, we will mostly consider $d(x, y) = \|x - y\|_2$, the Euclidean distance. This choice leads to a particularly rich geometric theory and allows for explicit solutions and efficient algorithms in a number of cases, such as Gaussian measures and one-dimensional distributions as seen above.

Remark 5.9 (Discrete measures)**.**
When $\mu = \sum_{i=1}^{N} \mu_i \delta_{x_i}$ and $\nu = \sum_{j=1}^{M} \nu_j \delta_{y_j}$ are discrete probability measures, the p-Wasserstein distance reduces to a finite dimensional optimization problem:

$$W_p^p(\mu, \nu) = \min_{P \in \Pi(\mu,\nu)} \sum_{i=1}^{N} \sum_{j=1}^{M} d(x_i, y_j)^p \, P_{ij},$$

where $\Pi(\mu, \nu)$ denotes the transport polytope. Recall that, in this case, computing W_p amounts to solving a linear program.

Fig. 5.13 shows a comparison between p-Wasserstein and l_p distances for distributions as their support becomes far from one another.

To do: Add the missing axis labels in Fig. 5.13.

Remark 5.10 (Suitability of W_p for learning)**.**
Thus far, we have considered the Wasserstein distance as a metric on spaces of probability measures. In learning applications, however, we are typically interested in spaces of *generative models*, that is, families of probability distributions $(P_\theta)_{\theta \in \Theta}$ indexed by parameters θ. In this setting, learning requires not only a notion of distance, but also a meaningful notion of convergence.

Let μ_{data} denote the true data distribution. The goal of learning is to find parameters θ such that P_θ converges to μ_{data}, or equivalently, such that $D(\mu_{\text{data}}, P_\theta) \to 0$ for a reasonable divergence D. A key advantage of Wasserstein distances is that they metrize a notion of convergence that is sensitive to the geometry of the underlying space. For instance, if $x_i \to x_0$ in $\mathbb{R}^d$, then $W_p(\delta_{x_i}, \delta_{x_0}) \to 0$, whereas many classical divergences fail to capture this intuitive convergence. As a result, the Wasserstein

distance provides a natural framework both for constructing sequences of generative models that converge to a target distribution and assess that convergence, that is, to design a learning algorithm.

Geodesics in the Wasserstein space.

Definition 5.4 (Wasserstein space)**.**

Let $(\mathcal{X}, d)$ be a metric space and let $p \geq 1$. Denote by $\mathcal{P}(\mathcal{X})$ the set of all probability measures on $\mathcal{X}$. The *p-Wasserstein space* is defined as

$$\mathcal{W}_p(\mathcal{X}) = \left\{ \mu \in \mathcal{P}(\mathcal{X}) \;\middle|\; \int_{\mathcal{X}} d(x, x_0)^p \, d\mu(x) < \infty \text{ for some (hence any) } x_0 \in \mathcal{X} \right\},$$

and is equipped with the p-Wasserstein distance W_p.

Remark 5.11 (Intuition)**.**

The Wasserstein space $\mathcal{W}_p(\mathcal{X})$ consists of all probability measures whose mass does not escape to infinity too quickly, so that transporting mass over distance according to the cost d^p has finite total cost.

In geometry, a geodesic generalises the notion of a straight line: it is the shortest path between two points in a given space. In the Euclidean space equipped with the ℓ_2 distance, geodesics correspond to linear interpolations between points, and the space is said to be geodesic. A similar notion exists in the Wasserstein space. While a naive interpolation between distributions, given by $(1-t)\mu_1 + t\mu_2$, corresponds to a *vertical* interpolation of densities—see Fig. 5.14, it does not generally represent the shortest path in Wasserstein space.

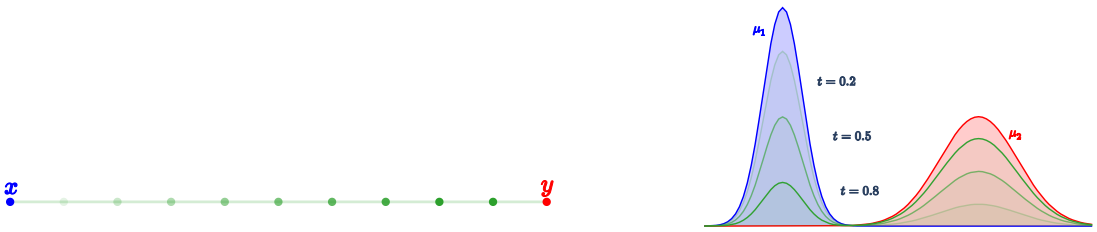

Figure 5.14: Left: geodesics as straight lines in Euclidean space. Right: linear interpolation between one-dimensional distributions.

The Wasserstein space $\mathcal{W}_p$ is itself a geodesic space. When an optimal transport map T exists between two measures μ_1 and μ_2, a Wasserstein geodesic connecting them is given by

$$\forall t \in [0,1], \qquad \mu^{1\to 2}(t) = \Big((1-t)\mathrm{Id} + tT\Big)_{\#}\mu_1.$$

This construction represents a *horizontal* interpolation of mass, where probability is gradually displaced along optimal transport paths. As a result, Wasserstein geodesics provide intuitive and physically meaningful interpolations between distributions, as illustrated in Fig. 5.15.

Wasserstein barycenters. As direct consequence of the Geodesic property of the Wasserstein space, optimal transport also provides a natural notion of averaging multiple

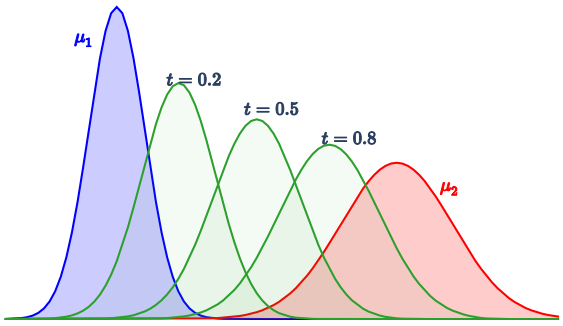

Figure 5.15: A Wasserstein geodesic between two one-dimensional distributions, obtained by transporting mass along optimal paths.

probability measures. Given a collection of distributions $(\mu_i)_{i=1}^s$ and weights $\lambda_i > 0$ such that $\sum_{i=1}^s \lambda_i = 1$, the *Wasserstein barycenter* is defined as

$$\bar{\mu} = \arg\min_{\mu} \sum_{i=1}^{s} \lambda_i \, W_p^p(\mu, \mu_i).$$

This definition generalizes Euclidean averaging to the space of probability measures, taking into account the geometry induced by optimal transport. For discrete measures, one may restrict the optimization to measures with fixed support, fixed weights, or both, leading to tractable computational formulations.

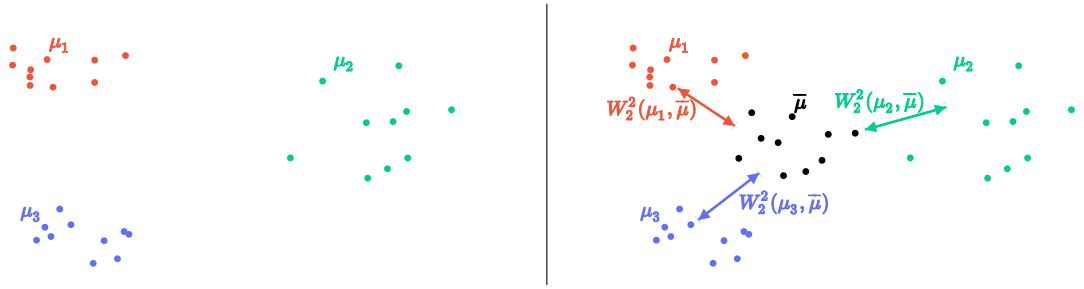

Figure 5.16: Illustration of Wasserstein barycenters: averaging multiple distributions while respecting the geometry of the underlying space.

Wasserstein barycenters differ fundamentally from Euclidean averages, especially in applications such as image processing. While Euclidean averaging often leads to blurred or unrealistic results, Wasserstein barycenters preserve geometric structure by transporting mass coherently across samples. This contrast is illustrated in Fig. 5.17, where averaging images in Wasserstein space produces sharper and more meaningful averages.

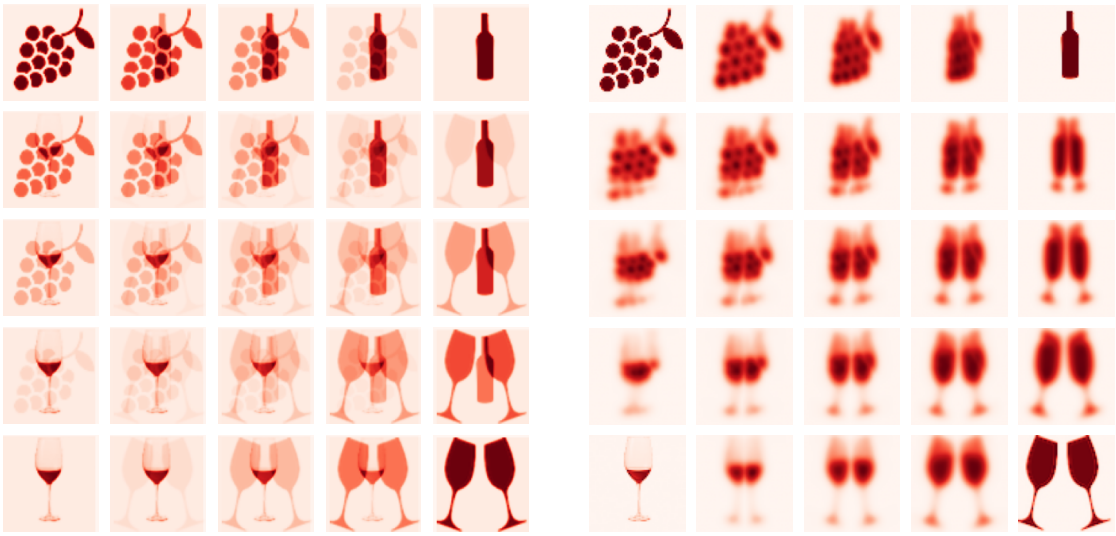

Figure 5.17: Left: Euclidean averaging of images. Right: Wasserstein barycenter, which preserves geometric features.

Exercises - Optimal Transport

Exercise 5.1 (Monge existence/non-existence)**.**
Give two concrete examples with $N = 3$, $M = 3$, and different weight vectors μ, ν with strictly positive coordinates such that i) a Monge map T satisfying $T_{\#}\mu = \nu$ exists, and ii) a Monge map T satisfying $T_{\#}\mu = \nu$ cannot exist. Explain briefly why the problem in the second case is the impossibility of mass splitting.

Solution 5.1 (Monge existence/non-existence)**.**
Let $\mu = (\mu_1, \mu_2, \mu_3)$ on $\{x_1, x_2, x_3\}$ and $\nu = (\nu_1, \nu_2, \nu_3)$ on $\{y_1, y_2, y_3\}$, all coordinates strictly positive.

i) Existence. Take

$$\mu = (1/2,\ 1/3,\ 1/6), \qquad \nu = (1/3,\ 1/6,\ 1/2).$$

Define the map

$$T(x_1) = y_3, \qquad T(x_2) = y_1, \qquad T(x_3) = y_2.$$

Then $T_{\#}\mu = \nu$, since each target mass equals exactly one source mass.

ii) Non-existence. Take

$$\mu = (1/2,\ 1/4,\ 1/4), \qquad \nu = (1/3,\ 1/3,\ 1/3).$$

For any map $T : \{x_1, x_2, x_3\} \to \{y_1, y_2, y_3\}$, we have

$$(T_{\#}\mu)(y_j) = \sum_{i:\, T(x_i) = y_j} \mu_i,$$

so target mass value should be of the form $\sum_{a \in A} a$, where A is a subset of

$$\mathcal{A} = \{1/2, 1/4, 1/4\}.$$

Equivalently, the target values are in the set

$$\{0,\ 1/4,\ 1/2,\ 3/4,\ 1\}.$$

Since $1/3$ does not belong to this set, no such T can satisfy $T_{\#}\mu = \nu$.

The non-existence of a map in the second case is that a Monge map cannot *split* mass: each source atom must be sent entirely to a single target point, so target masses must be sums of whole source masses rather than arbitrary fractions.

Exercise 5.2 (Monge non-uniqueness)**.**
Construct a discrete example with $N = M = 2$ and a cost matrix C with strictly-positive entries such that there are at least two distinct Monge maps achieving the same optimal cost.

Solution 5.2 (Monge non-uniqueness)**.**

Let $\mu = \nu = (1/2,\, 1/2)$ on $\{x_1, x_2\}$ and $\{y_1, y_2\}$, and take the strictly-positive cost matrix

$$C = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Then any Monge map has total cost

$$\frac{1}{2}\, C_{1,T(x_1)} + \frac{1}{2}\, C_{2,T(x_2)} = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1.$$

In particular, the two distinct Monge maps

$$T_1(x_1) = y_1, \quad T_1(x_2) = y_2, \qquad \text{and} \qquad T_2(x_1) = y_2, \quad T_2(x_2) = y_1$$

both satisfy $T_{\#}\mu = \nu$ and achieve the same (optimal) cost.

Exercise 5.3 (A coupling always exists)**.**
Let $\mu \in \mathbb{R}_+^N$ and $\nu \in \mathbb{R}_+^M$ with $\sum_i \mu_i = \sum_j \nu_j = 1$. Show that $P = \mu\nu^\top$ belongs to $\Pi(\mu, \nu)$. Interpret this plan as an *independent coupling* and explain how mass is transported under this plan.

Solution 5.3 (A coupling always exists)**.**
Let $P = \mu\nu^\top$, i.e. $P_{ij} = \mu_i \nu_j \geq 0$. Then the row sums are

$$\sum_{j=1}^{M} P_{ij} = \sum_{j=1}^{M} \mu_i \nu_j = \mu_i \sum_{j=1}^{M} \nu_j = \mu_i,$$

and the column sums are

$$\sum_{i=1}^{N} P_{ij} = \sum_{i=1}^{N} \mu_i \nu_j = \nu_j \sum_{i=1}^{N} \mu_i = \nu_j.$$

Hence $P\mathbf{1}_M = \mu$ and $P^\top \mathbf{1}_N = \nu$, so $P \in \Pi(\mu, \nu)$.

This plan is called the *independent coupling* because it corresponds to drawing (X, Y) with $X \sim \mu$ and $Y \sim \nu$ independently, i.e. $\mathbb{P}(X = x_i, Y = y_j) = \mu_i \nu_j$. Under this plan, each source mass μ_i is split proportionally across all targets: from x_i (where there is a μ_i amount of total mass) an amount $\mu_i \nu_j$ is sent to y_j for every j.

Exercise 5.4 (Solve a 2×2 OT problem by hand)**.**
Let $N = M = 2$, $\mu = (\mu_1, \mu_2)$, $\nu = (\nu_1, \nu_2)$, where $\mu_1 + \mu_2 = \nu_1 + \nu_2 = 1$, and

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}.$$

Solve the Kantorovich problem: find an optimal plan $P^\star \in \Pi(\mu, \nu)$ and the optimal cost.

Solution 5.4 (Solve a 2×2 OT problem by hand)**.**

Any $P \in \Pi(\mu,\nu)$ is determined by $p := P_{11}$:

$$P(p) = \begin{pmatrix} p & \mu_1 - p \\ \nu_1 - p & \mu_2 - \nu_1 + p \end{pmatrix}, \qquad p \in \Big[\max\{0, \nu_1 - \mu_2\},\ \min\{\mu_1, \nu_1\}\Big].$$

The cost is affine in p:

$$\begin{aligned} \langle C, P(p)\rangle &= c_{11}p + c_{12}(\mu_1 - p) + c_{21}(\nu_1 - p) + c_{22}(\mu_2 - \nu_1 + p) \\ &= \text{const } + \ p\,\Delta, \end{aligned}$$

where

$$\Delta := c_{11} - c_{12} - c_{21} + c_{22}.$$

Hence the optimum is attained at an extreme of the feasible region:

$$p^\star = \begin{cases} \max\{0, \nu_1 - \mu_2\}, & \Delta > 0, \\ \min\{\mu_1, \nu_1\}, & \Delta < 0, \\ \text{any feasible } p, & \Delta = 0, \end{cases} \qquad P^\star = P(p^\star), \qquad \text{opt. cost } = \langle C, P(p^\star)\rangle.$$

Exercise 5.5 (1D OT between a two-point distribution and a uniform law)**.**
Let $\mu = \frac{1}{2}\delta_0 + \frac{1}{2}\delta_1$ and $\nu = \mathrm{Unif}([0,1])$ on $\mathbb{R}$. Using the quantile representation, compute $W_1(\mu,\nu)$, i.e., assume a cost $c(x,y) = |x-y|$, and describe what the optimal plan does (compute the integrals explicitly).

Solution 5.5 (1D OT between a two-point distribution and a uniform law)**.**
In one dimension,

$$W_1(\mu,\nu) = \int_0^1 \left|F_\mu^{-1}(t) - F_\nu^{-1}(t)\right| dt.$$

Here $F_\nu^{-1}(t) = t$ for $t \in [0,1]$, and

$$F_\mu^{-1}(t) = \begin{cases} 0, & t \in [0, 1/2], \\ 1, & t \in (1/2, 1]. \end{cases}$$

Therefore

$$\begin{aligned} W_1(\mu,\nu) &= \int_0^{1/2} |0 - t|\, dt + \int_{1/2}^1 |1 - t|\, dt \\ &= \int_0^{1/2} t\, dt + \int_{1/2}^1 (1-t)\, dt = \frac{1}{8} + \frac{1}{8} = \frac{1}{4}. \end{aligned}$$

The optimal coupling is the monotone (quantile) coupling: it sends the lower half of the uniform mass $t \in [0, 1/2]$ to 0 and the upper half $t \in (1/2, 1]$ to 1, i.e. it splits ν into two halves and matches them with the two atoms of μ.
Proposed: Repeat the exercise for W_2.

Exercise 5.6 (Wasserstein geodesic between two Gaussians in one dimension)**.**
Let $\mu_0 = \mathcal{N}(m_0, \sigma_0^2)$ and $\mu_1 = \mathcal{N}(m_1, \sigma_1^2)$ on $\mathbb{R}$. Using the fact that the optimal

map is affine for any $p \geq 1$, show that the displacement interpolation $\mu_t = ((1-t)\mathrm{Id} + tT)_{\#}\mu_0$ is Gaussian, and identify its mean and variance as functions of t.

Solution 5.6 (Wasserstein geodesic between two Gaussians in one dimension)**.**
In 1D the optimal map from $\mu_0 = \mathcal{N}(m_0, \sigma_0^2)$ to $\mu_1 = \mathcal{N}(m_1, \sigma_1^2)$ is affine:

$$T(x) = m_1 + \frac{\sigma_1}{\sigma_0}(x - m_0).$$

Define the interpolation map

$$S_t(x) := (1-t)x + tT(x) = \left((1-t) + t\frac{\sigma_1}{\sigma_0}\right)x + t\left(m_1 - \frac{\sigma_1}{\sigma_0}m_0\right), \qquad t \in [0,1].$$

Since S_t is affine and $X \sim \mathcal{N}(m_0, \sigma_0^2)$ implies $S_t(X)$ is Gaussian, we have

$$\mu_t = (S_t)_{\#}\mu_0 = \mathcal{N}(m_t, \sigma_t^2),$$

with mean and variance

$$m_t = (1-t)m_0 + tm_1, \qquad \sigma_t = (1-t)\sigma_0 + t\sigma_1, \qquad \sigma_t^2 = \Big((1-t)\sigma_0 + t\sigma_1\Big)^2.$$

Exercise 5.7 (Gaussian OT in one dimension)**.**
Let $\mu = \mathcal{N}(m_\mu, \sigma_\mu^2)$ and $\nu = \mathcal{N}(m_\nu, \sigma_\nu^2)$.

1. Derive the optimal transport map $T(x) = m_\nu + \frac{\sigma_\nu}{\sigma_\mu}(x - m_\mu)$.

2. Compute $W_2^2(\mu, \nu)$ and simplify the expression.

Solution 5.7 (Gaussian OT in one dimension)**.**
1) In one dimension (for any p), the optimal transport map is the monotone rearrangement

$$T = F_\nu^{-1} \circ F_\mu.$$

For $\mu = \mathcal{N}(m_\mu, \sigma_\mu^2)$,

$$F_\mu(x) = \Phi\left(\frac{x - m_\mu}{\sigma_\mu}\right),$$

where

$$\Phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-s^2/2}\, ds$$

denotes the cumulative distribution function of the standard normal distribution. Similarly, for $\nu = \mathcal{N}(m_\nu, \sigma_\nu^2)$,

$$F_\nu^{-1}(u) = m_\nu + \sigma_\nu\, \Phi^{-1}(u).$$

Hence

$$T(x) = F_\nu^{-1}(F_\mu(x)) = m_\nu + \sigma_\nu\, \Phi^{-1}\left(\Phi\left(\frac{x - m_\mu}{\sigma_\mu}\right)\right) = m_\nu + \frac{\sigma_\nu}{\sigma_\mu}(x - m_\mu).$$

2) Let $X \sim \mu$ and set $Y = T(X)$, so $Y \sim \nu$ and

$$W_2^2(\mu,\nu) = \mathbb{E}\Big[(X-Y)^2\Big] = \mathbb{E}\Big[(X-T(X))^2\Big].$$

Write $X = m_\mu + \sigma_\mu Z$ with $Z \sim \mathcal{N}(0,1)$. Then $T(X) = m_\nu + \sigma_\nu Z$, so

$$X - T(X) = (m_\mu - m_\nu) + (\sigma_\mu - \sigma_\nu)Z,$$

and using $\mathbb{E}[Z] = 0$, $\mathbb{E}[Z^2] = 1$,

$$W_2^2(\mu,\nu) = (m_\mu - m_\nu)^2 + (\sigma_\mu - \sigma_\nu)^2.$$

Proposed: How does the solution change for other non-Gaussian location-scale distributions (in 1d)?

Exercise 5.8 (1D Wasserstein barycenter of two distributions)**.**
Let $\mu_1 = \mathrm{Unif}([0,1])$ and $\mu_2 = \mathrm{Unif}([2,3])$ on $\mathbb{R}$, and let $\lambda \in (0,1)$. Compute explicitly the 2-Wasserstein barycenter

$$\bar{\mu} = \arg\min_{\mu} \ \lambda W_2^2(\mu,\mu_1) + (1-\lambda) W_2^2(\mu,\mu_2).$$

Identify the support of $\bar{\mu}$ and describe it geometrically.

Solution 5.8 (1D Wasserstein barycenter of two distributions)**.**
In 1D, the W_2-barycenter has quantile function given by the convex combination of quantiles:

$$F_{\bar{\mu}}^{-1}(t) = \lambda F_{\mu_1}^{-1}(t) + (1-\lambda) F_{\mu_2}^{-1}(t), \qquad t \in (0,1).$$

Since $F_{\mu_1}^{-1}(t) = t$ and $F_{\mu_2}^{-1}(t) = 2 + t$, we have

$$F_{\bar{\mu}}^{-1}(t) = \lambda t + (1-\lambda)(2+t) = t + 2(1-\lambda).$$

Therefore $\bar{\mu} = \mathrm{Unif}([2(1-\lambda),\, 1+2(1-\lambda)]) = \mathrm{Unif}([2-2\lambda,\, 3-2\lambda])$.

The barycenter is supported on the interval $[2-2\lambda,\, 3-2\lambda]$, i.e. the unit-length interval obtained by translating $[2,3]$ left by 2λ (equivalently, translating $[0,1]$ right by $2(1-\lambda)$). Geometrically, the barycenter slides linearly between the two intervals.

Exercise 5.9 (Barycenter vs Euclidean averaging)**.**
Let $\mu_1 = \delta_0$ and $\mu_2 = \delta_1$ on $\mathbb{R}$.

1. Compute the Euclidean average $\frac{1}{2}(\mu_1 + \mu_2)$.
2. Compute the Wasserstein barycenter with equal weights, i.e., $\lambda = 1/2$.

Compare and interpret the two results.

Solution 5.9 (Barycenter vs Euclidean averaging)**.**
1) The Euclidean (mixture) average is

$$\tfrac{1}{2}(\mu_1 + \mu_2) = \tfrac{1}{2}\delta_0 + \tfrac{1}{2}\delta_1.$$

2) The W_2-barycenter in 1D with equal weights has quantile

$$F^{-1}_{\bar{\mu}}(t) = \tfrac{1}{2}F^{-1}_{\mu_1}(t) + \tfrac{1}{2}F^{-1}_{\mu_2}(t).$$

Since $F^{-1}_{\delta_0}(t) = 0$ and $F^{-1}_{\delta_1}(t) = 1$ for all $t \in (0,1)$, we get

$$F^{-1}_{\bar{\mu}}(t) = \tfrac{1}{2},$$

hence $\bar{\mu} = \delta_{1/2}$.

The Euclidean average is a *mixture* with mass split between 0 and 1, whereas the Wasserstein barycenter *moves* the mass to the midpoint, producing a single Dirac at $1/2$.

Proposed: Repeat the exercise for $\mu_0 = \mathcal{N}(0, \sigma^2)$ and $\mu_1 = \mathcal{N}(1, \sigma^2)$.

Week 6

Deep Latent Variable Models

NB: This is based on (Goodfellow, Bengio & Courville, 2016), (Goodfellow et al., 2014), (Kingma & Welling, 2013), and (Kingma & Welling, 2019).

Motivation and road map. Deep neural networks provide powerful, scalable parametrisations for highly expressive functions. Furthermore, when embedded within probabilistic models, they allow us to move beyond restrictive parametric assumptions while retaining a principled inference procedure. In particular, we will see how to implement the inference ideas developed in the previous chapters, in particular latent variable modelling and variational inference (VI), within a flexible function-approximation framework. Also, the models in this chapter will serve as a critical components in ambitious modelling paradigms introduced in subsequent chapters.

We begin with a concise overview of neural network architectures, including convolutional, auto-encoding and recurrent (and recursive) networks. Then, we will review modern, state-of-the-art, architectures underlying the generative adversarial networks. More than reviewing the architectures in detail, our objective will be to provide a clear understanding of their representational capabilities, modelling assumptions, and training strategies.

The chapter will conclude with the variational autoencoder (VAE), which implements the full variational inference rationale within a deep learning setting. VAEs are latent-variable models using neural networks to parameterise both the generative model and the variational posterior. This provides a scalable approach to practical approximate Bayesian inference in highly expressive models, and sets the foundation for the more flexible probabilistic constructions.

6.1 A brief introduction to neural networks

A neural network (NN) is a function-approximation model originally inspired by biological neurons. As the theory has developed, however, the analogy with biology has become mostly a metaphor. Being general-interest approximators, NNs have been used in ML for both classification, regression, and more recently for synthetic generation. In all cases, the objective is to approximate an unknown target function f^* by learning a parametric mapping

$$y = f(x; \theta),$$

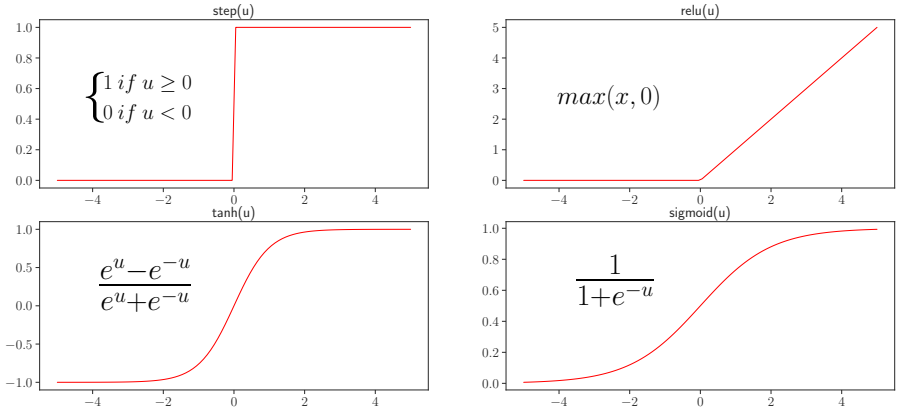

Figure 6.1: Examples of common activation functions.

where the parameters θ are estimated from data. NNs implement a compositional structure, where information processing occurs in stages: processing layers defined by collections of neurons provide increasingly-complex representations that are fed into the next layers. Empirically, this *deep* construction allows NNs to operate directly on raw inputs, learning expressive representations automatically, rather than relying on hand-crafted features.

The neuron and activation functions. The elementary processing unit within a NN is referred to as neuron. Given an input $x = [x_n]_{d=1}^D \in \mathbb{R}^D$, the neuron computes a weighted sum

$$u = \sum_{i=1}^{n} w_d x_d + b,$$

where $W = [w_d]_{d=1}^D \in \mathbb{R}^D$ are the weights and $b \in \mathbb{R}$ is the bias. Then, the output of the neuron is computed by applying an activation function f to the weighted sum, that is,

$$h = f(u).$$

A number of different activation functions ensure the universal approximation properties of NNs, and their choice is mainly guided by their affinity to the optimisation strategy, which is given by their monotonicity and differentiability. Fig. 6.1 shows four examples of popular activations functions.

Neurons receiving the same input can be combined into a *layer*, which then outputs the stacked collection of neuron outputs into the following layer, thus constituting a multi-layer feedforward NN, where information flows in a single direction. Here, notice that the role of the nonlinear activation function is crucial, the compositions of purely-affine layers would collapse to a single linear transformation without much modelling expressivity.

We adopt the following notation for the feedforward l-layer NN: for $k \in \{1, \dots, l\}$, the output of the k-th layer is

$$h^{(k)} = f^{(k)}\left(h^{(k-1)} W^{(k)} + b^{(k)}\right), \qquad h^{(0)} = x, \tag{6.1}$$

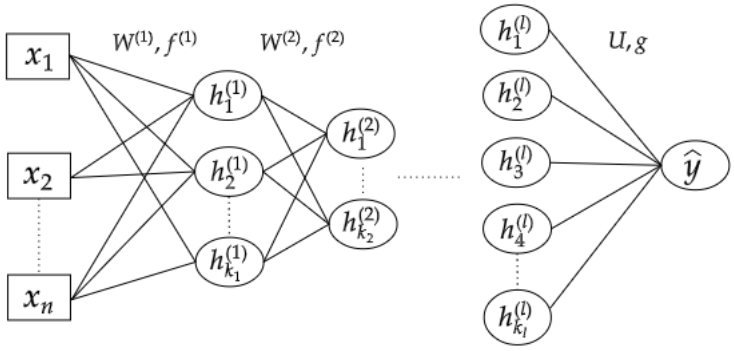

Figure 6.2: A network of depth l - taken from (Goodfellow et al., 2016).

and the network's final layer output is given by

$$\hat{y} = g\Big(h^{(l)}U + c\Big), \tag{6.2}$$

where g denotes the activation function applied at the output layer, defining the output unit. See Fig. 6.2 for an illustration of an l-hidden-layer feedforward NN.

Remark 6.1 (Universal approximation property and depth)**.**
While a single, sufficiently-wide, hidden layer can approximate a broad class of target functions, this may require an impractically large number of units and can lead to poor generalisation. In practice, it is often preferable to increase *depth* (i.e., the number of layers) rather than width, with comparatively fewer units. This motivates the study of *deep* architectures.

Remark 6.2 (Parameters and hyperparameters)**.**
We will refer to the parameters of the NN as the quantities that are trainable by means of continuous (e.g., gradient-based) optimisation, that is, the weights and the biases. Conversely, the modelling choices that are taken using heuristics or via cross-validation, such as the number of layers and activation functions, will be referred to as hyperparameters.

Cost functions and output units. NNs can be formulated as probabilistic models with a parameter θ defining the conditional distribution

$$p(y \mid x; \theta),$$

thus allowing for parameter estimation via maximum likelihood, or equivalently, by minimising the negative log-likelihood. Using a set of observations defining the empirical measure $\hat{p}_{\text{data}}$, the cost function can be expressed as:

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{\text{data}}}\Big[\log p_{\text{model}}(y \mid x)\Big]. \tag{6.3}$$

As a consequence, the choice of output unit, which serves as the link between the learnt representations and the space of the discriminative variable y, also determines the form of the likelihood and thus the cost function. In practice, the output unit in discriminative

modelling is usually set to one of the following choices:

1. **Linear output:**
$$\hat{y} = h^{(l)}U + c.$$
Used for regression. If $p(y \mid x) = \mathcal{N}(y \mid \hat{y}, I)$, maximising the log-likelihood is equivalent to minimising the mean squared error.

2. **Sigmoid output:**
$$\hat{y} = \text{sig}\Big(h^{(l)}U + c\Big).$$
Used for binary classification. The model defines a Bernoulli distribution for $y \mid x$, with the output representing $P(y = 1 \mid x)$.

3. **Softmax output:**
$$\hat{y} = \text{softmax}\Big(h^{(l)}U + c\Big),$$
which generalises the sigmoid function to the multi-class setting.

Training a neural network. Given a dataset $\{(x_n, y_n)\}_{n=1}^N$, the network defines a parametric mapping
$$\hat{y}_n = f(x_n; \theta),$$
and training amounts to minimising an objective of the form
$$J(\theta) = -\frac{1}{N}\sum_{n=1}^{N} \log p(y_n \mid x_n; \theta).$$
Meaning that $\theta^* = \arg\min_\theta J(\theta)$.

In a feedforward network, training is implemented by means of stochastic gradient descent (SGD), where the gradient is computed in two stages. First, the information flows from the input x to the prediction $\hat{y}$ in a process referred to as **forward propagation**, where the output, the cost function $J(X, \theta)$, and critically, all the neuron activations are computed. See Algorithm 5.

Algorithm 5 Forward Propagation

Require: Network depth l
Require: Weights $W^{(k)}$ and biases $b^{(k)}$, for $k = 1, \ldots, l$
Require: Output parameters U, c and output function g
Require: Input x and target y
1: $h^{(0)} \leftarrow x$
2: **for** $k = 1, \ldots, l$ **do**
3: $\quad u^{(k)} \leftarrow h^{(k-1)}W^{(k)} + b^{(k)}$
4: $\quad h^{(k)} \leftarrow f^{(k)}(u^{(k)})$
5: **end for**
6: $\hat{y} \leftarrow g(h^{(l)}U + c)$
7: $J \leftarrow L(\hat{y}, y)$

The second stage is called **backpropagation**, where information now flows from the predicted output back to the input, thus propagating the prediction error through the

NN in order to find the contribution of each trainable parameter in that error. The gradient of the loss wrt all parameters is computed efficiently via the chain rule, using the stored activations from the forward pass and the gradients of the subsequent layer. See Algorithm 6.

Algorithm 6 Backward Propagation

Require: Learning rate λ

Perform forward propagation and store intermediate values
Compute output error signal
Compute gradients with respect to U and c
Update $U \leftarrow U - \lambda \frac{\partial J}{\partial U}$
Update $c \leftarrow c - \lambda \frac{\partial J}{\partial c}$
for $k = l, \ldots, 1$ **do**
    Propagate error to layer k
    Compute $\frac{\partial J}{\partial W^{(k)}}$ and $\frac{\partial J}{\partial b^{(k)}}$
    Update $W^{(k)} \leftarrow W^{(k)} - \lambda \frac{\partial J}{\partial W^{(k)}}$
    Update $b^{(k)} \leftarrow b^{(k)} - \lambda \frac{\partial J}{\partial b^{(k)}}$
end for

In practice, training is performed using mini-batches of data in order to avoid local minima and keep computational/memory overhead at bay, while exploiting GPU parallelism. A full pass through the training set is referred to as an **epoch**. Increasing the number of epochs typically reduces training error, although excessive training may hinder generalisation due to overfitting due to the fact that NNs in practice are heavily overparametrised.

Remark 6.3 (Optimisation algorithms)**.**
SGD's baseline performance is improved through momentum-based updates, which smooth noisy gradients, and adaptive learning-rate methods such as AdaGrad, RMSProp, and Adam. The latter combines momentum with per-parameter adaptive step sizes based on exponentially weighted averages of first and second moments of the gradients, and has become the de facto standard to train NNs.

Remark 6.4 (Regularisation)**.**
Generalisation is practically achieved by training large (deep, overparametrised) architectures with appropriate regularisation. A standard approach is ℓ_2 regularisation (weight decay),

$$\tilde{J}(\theta) = J(\theta) + \frac{\alpha}{2}\|\theta\|_2^2,$$

which penalises large weights and is equivalent to shrinking parameters at each gradient step. Other widely used techniques include **dropout**, which randomly masks units during training to approximate model averaging, data augmentation and noise injection, and **early stopping**, where training is halted once validation performance deteriorates.

Classic architectures. Particular ways of connecting the neurons (and layers) in a NN can be chosen with the objective to capture relevant patterns depending of the nature of the data and the problem at hand.

Convolutional neural networks (CNNs) can be considered as a restriction of fully-connected feedforward NNs, designed for inputs with a spatial structure, such as 1-D signals and 2-D images. A CNN layer applies learnable local convolutional filters and is characterised by the following three key properties:

- **Sparse connectivity:** each output depends only on a local receptive convolution kernel.
- **Parameter sharing:** the same kernel is reused across spatial locations, reducing the number of parameters.
- **Equivariance:** translating the input produces a corresponding translation of the feature map.

To do: Include a diagram of a CNN

Recurrent neural networks (RNNs) are specialised for sequential data $(x^{(1)}, \ldots, x^{(\tau)})$. They maintain a hidden state that is updated iteratively, sharing parameters across time:

$$h^{(t)} = f\left(h^{(t-1)}, x^{(t)}; \theta\right). \tag{6.4}$$

This provides a natural mechanism for modelling temporal dependencies and variable-length inputs. Conceptually, the network can be *unrolled* across time, yielding a deep computation graph in which the same parameters appear at each step.

Training is performed via backpropagation through time (BPTT). In practice, vanilla RNNs can struggle with long-range dependencies due to vanishing and exploding gradients; gated architectures such as LSTMs and GRUs mitigate this by learning when to retain or forget information.

To do: Include a diagram of an RNN - consider moving this to the Transformer chapter.

Example 6.1 (Convolutional vs fully-connected net on MNIST)**.**
To illustrate the ability of the convolutional network to incorporate appropriate inductive biases for images, we will implement this network alongside a fully connected (feedforward) network on the classification of the MNIST dataset. For a fair comparison, both networks will feature the roughly the same number of parameters ($\approx$ 1.2M parameters). Fig. 6.3 shows the training and test losses, observe how the feedforward network achieved a lower training error and a saturated test error, thus suggesting overfitting.

6.2 Neural networks for generative modelling

6.2.1 Autoencoders

The central idea of this architecture is to learn an identity map $\mathrm{NN} : \mathbb{R}^d \to \mathbb{R}^d$ given by a multi-layer network with one of the hidden layers working as a bottleneck. Choosing

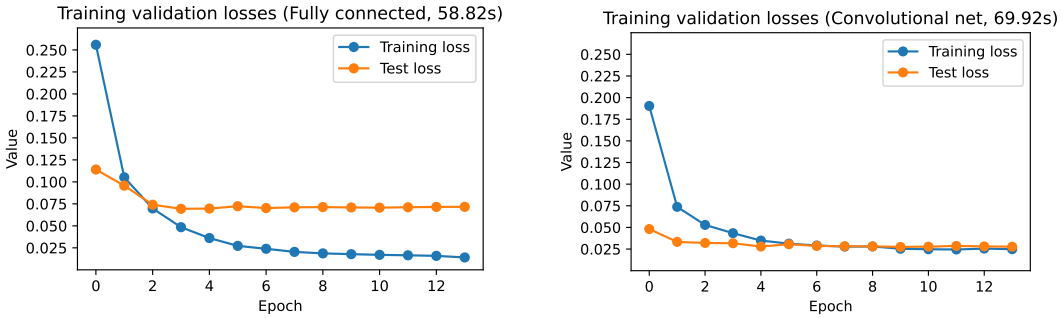

Figure 6.3: CNN vs FC: training dynamics

$m \ll d$ neurons in the bottleneck layer forces data reconstruction from an m-dimensional code.

AEs consist of two networks: an **encoder**, which maps the input $x \in \mathbb{R}^d$ to a latent representation

$$z = f_\theta(x) \in \mathbb{R}^m,$$

and a **decoder**, which reconstructs the input from this representation,

$$r = g_\phi(z) \in \mathbb{R}^d.$$

Training seeks to minimise a reconstruction loss,

$$J(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} L\Big(x^{(i)}, g_\phi(f_\theta(x^{(i)}))\Big),$$

typically mean squared error for continuous data or cross-entropy for binary data.

Remark 6.5 (PCA as a linear autoencoder)**.**
When the networks $f_\theta(\cdot)$ and $g_\phi(\cdot)$ are linear maps and the mean squared error loss is considered, the AE recovers the same subspace as principal component analysis (PCA). With non-linear encoders and decoders, autoencoders can learn richer, non-linear representations.

Building on their (non-linear) dimensionality reduction capability, autoencoders are widely used for *representation learning*. The latent code z provides a compact feature representation that can be used as input to subsequent models for supervised or unsupervised tasks. In natural language processing, closely related ideas give rise to *embeddings*, where discrete objects (e.g. words) are mapped into continuous vector spaces amenable to statistical modelling.

Remark 6.6 (AEs effectiveness to learn representations)**.**
The learned latent space of an AE often acquires meaningful geometric structure, where inputs that are similar in the original space tend to map to nearby points in latent space. This induces a smoother, lower-dimensional representation of the data distribution, which facilitates visualisation and improved efficiency in similarity-based, interpretable, models.

Remark 6.7 (AEs for generative modelling)**.**

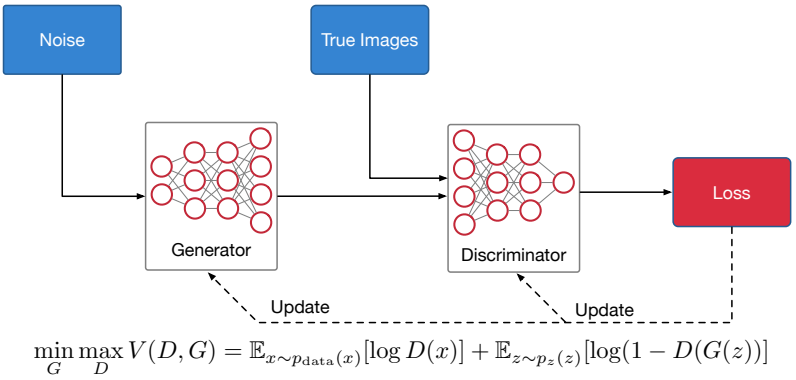

Figure 6.4: Diagram of the GAN architecture

In theory, AEs could be used for generative modelling by learning the distribution of the code z. This should be less challenging than modelling the data distribution $p(x)$ due to the reduced dimensionality of the code. However, since the code is not constrained to have any particular (tractable) distribution, learning and sampling form the code still poses considerable challenges. We will see that this issue can be resolved by imposing a tractable prior on the code, that is the underlying idea behind the variational AEs.

6.2.2 Generative adversarial networks (GANs)

A GAN learns to generate data by training two models in competition: a generator $G(z;\theta^{(g)})$ mapping noise z to synthetic samples, and a discriminator $D(x;\theta^{(d)})$, which outputs the probability that x is a real (and not generated) datapoint.

The standard GAN objective is a minimax game:[1]

$$\min_G \max_D V(G,D),$$

where

$$V(G,D) = \mathbb{E}_{x\sim p_{\text{data}}}\Big[\log D(x)\Big] + \mathbb{E}_{z\sim p(z)}\Big[\log(1 - D(G(z)))\Big]. \tag{6.5}$$

From the point of view of the discriminator, the first term corresponds to correct classifications of true samples and the second one to correct classification of fake samples. From the point of view of the generator (which only appears in the second term), the reward corresponds to fooling the discriminator into classify fake samples as real ones. At equilibrium, it is expected that the generated distribution matches the data distribution and the discriminator cannot distinguish them; as it will be seen through the following results. See Fig. 6.4 for a diagram.

Proposition 6.1 (Optimal discriminator (Goodfellow et al., 2014))**.**

For a fixed generator G inducing a distribution p_g on $\mathcal{X}$ via $x = G(z)$, $z \sim p(z)$,

[1] This formulation corresponds to a two-player *zero-sum game* in classical game theory: one player seeks to minimise the payoff while the other seeks to maximise it.

the objective in eq. (6.5) is maximised pointwise by

$$D^*(x) \;=\; \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Proof. Expressing the second term on eq. (6.5) as an expectation wrt the generated sample, $V(G, D)$ can be written as a single integral:

$$V(G, D) = \int \Big(p_{\text{data}}(x)\, \log D(x) \;+\; p_g(x)\, \log(1 - D(x)) \Big)\, dx.$$

Assuming $D(\cdot)$ is general enough, this problem can be maximised pointwise. Defining $d := D(x) \in (0, 1)$, the integrand can be expressed for a fixed x as

$$f_x(d) := p_{\text{data}}(x)\, \log d + p_g(x)\, \log(1 - d), \qquad d \in (0, 1).$$

Differentiating wrt d and setting to zero gives

$$f_x'(d) = \frac{p_{\text{data}}(x)}{d} - \frac{p_g(x)}{1 - d} = 0 \quad \Longrightarrow \quad d^* = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

Furthermore, note that $f_x(\cdot)$ is a strictly concave function of d since

$$f_x''(d) = -\frac{p_{\text{data}}(x)}{d^2} - \frac{p_g(x)}{(1 - d)^2} < 0.$$

Therefore, d^* is the global maximiser and the optimal discriminator (for fixed G) is

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)},$$

for all x such that $p_{\text{data}}(x) + p_g(x) > 0$. ■

Definition 6.1 (Jensen–Shannon divergence)**.**

Let P and Q be probability distributions that are absolutely continuous with respect to a common measure, with densities p and q. Define the mixture distribution

$$m(x) := \frac{1}{2}\Big(p(x) + q(x)\Big).$$

The Jensen–Shannon (JS) divergence between P and Q is

$$\text{JS}(P\|Q) := \frac{1}{2}\,\text{KL}(P\|M) + \frac{1}{2}\,\text{KL}(Q\|M),$$

where M is the distribution with density m, and

$$\text{KL}(P\|M) = \int p(x) \log \frac{p(x)}{m(x)}\, dx.$$

Remark 6.8 (Jensen–Shannon properties and interpretation)**.**

The JS divergence induces a metric $\sqrt{\text{JS}(P\|Q)}$, and:

- $\mathrm{JS}(P\|Q) \geq 0$,
- $\mathrm{JS}(P\|Q) = 0$ iff $P = Q$ (a.e.),
- $\mathrm{JS}(P\|Q) \leq \log 2$.

The JS divergence is an extension of the KL divergence which, is i) symmetric, ii) finite even when P and Q have disjoint supports, and iii) measures how far each distribution is from the midpoint mixture $M = \frac{1}{2}(P + Q)$.

Theorem 6.1 (Global optimum of the GAN game (Goodfellow et al., 2014))**.**
Define the training problem for the generator after the optimal discriminator has been found

$$C(G) := \max_D V(G, D).$$

Then

$$C(G) = -\log 4 \; + \; 2\,\mathrm{JS}\Big(p_{\text{data}} \,\|\, p_g\Big),$$

with global minimum attained if and only if $p_g = p_{\text{data}}$. Furthermore, at the optimal G^*

$$C(G^*) = -\log 4 \quad \text{and} \quad D^*(x) = \tfrac{1}{2} \text{ for all } x \text{ (on the support)}.$$

Proof. Using the optimal discriminator from Proposition 6.1, we have

$$\begin{aligned} C(G) &= V(G, D^*) \\ &= \mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}\right] \; + \; \mathbb{E}_{x\sim p_g}\left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)}\right]. \end{aligned}$$

Define $m(x) := \frac{1}{2}\Big(p_{\text{data}}(x) + p_g(x)\Big)$ and use the definition of the JS divergence to write

$$\begin{aligned} C(G) &= \mathbb{E}_{p_{\text{data}}}\left[\log \frac{p_{\text{data}}}{m}\right] + \mathbb{E}_{p_g}\left[\log \frac{p_g}{m}\right] - 2\log 2 \\ &= \mathrm{KL}\Big(p_{\text{data}} \| m\Big) + \mathrm{KL}\Big(p_g \| m\Big) - \log 4 \\ &= 2\,\mathrm{JS}\Big(p_{\text{data}} \| p_g\Big) - \log 4. \end{aligned}$$

Since $\mathrm{JS}(\cdot\|\cdot) \geq 0$ with equality if and only if the two distributions are equal, the global minimum of $C(G)$ is attained if and only if $p_g = p_{\text{data}}$, and at that point

$$C(G^*) = -\log 4.$$

Lastly, using G^* in Proposition 6.1, we have $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{data}}(x)} = \frac{1}{2}$ for all x on the support. ■

Remark 6.9 (Practical convergence of GAN training)**.**
The results above assume i) an infinite model capacity, and ii) exact optimisation of the discriminator at each step. However, this does *not* imply that gradient-based training of finite neural networks is guaranteed to converge. The GAN objective defines a saddle-point problem, where simultaneous gradient updated may fail to

converge. Though (Goodfellow et al., 2014) proves existence of the global optimum, convergence is mainly controlled by the training algorithm.

Training in practice. In practice, the discriminator is not fully optimised at each step. Instead, GANs are trained by alternating stochastic gradient updates: for several steps, the discriminator is updated based on the objective in eq. (6.5), while the generator is trained wrt the surrogate minimisation objective

$$-\mathbb{E}_{z\sim p(z)}[\log D(G(z))].$$

Therefore, training proceeds by alternating gradient ascent in the discriminator and gradient descent in the generator. Importantly, the discriminator is *not* trained to full convergence between generator updates, but rather only trained "near" optimality by performing a small number (often 1–5) of updates per generator step.

Example 6.2 (DCGAN on MNIST and celeba)**.**

We implemented a Deep Convolutional GAN (DCGAN) originally proposed by (Radford, Metz & Chintala, 2016). DCGAN implements CNNs tweaks such as strided convolutions, batch normalisation, no fully connected layers,and specific activations that significantly stabilised GAN training for image generation. Our implementation results on MNIST and celeba datasets, are shown in Figs. 6.5 and 6.6 respectively.

Notice how the generator and discriminator errors are noisy but in average they converge. Also note that the the image generation in both cases provide images of a lower quality than the reference datasets, but recall that these examples are designed to run on a laptop and thus the architecture and dataset is somewhat constrained.

6.3 Autoencoding variational Bayes

We will now revisit the latent-variable formulation assumed throughout the module. Consider a dataset $\mathbf{x} = \{x_1, \ldots, x_N\}$ of i.i.d. samples generated by an unknown generative model with unknown density $p_\theta(x)$. Furthermore, assume that the data generation involves a latent variable z, meaning that the generation process consists of sampling $z_n \sim p_\theta(z)$, and then $x_n \sim p_\theta(x \mid z_n)$.

We will assume standard regularity conditions, such as differentiability of all the involved densities wrt their arguments and the parameters θ; also note that θ contains the parameters of all the densities: the prior $p_\theta(z)$ and the likelihood $p_\theta(x \mid z)$. Furthermore, unlike previous sections in the module, we do not make any structural assumptions about the model, in particular, we do not assume the posterior $p_\theta(z \mid x)$ is tractable (as in EM) or that the involved densities are conjugate such as in the mixture models we have seen before.

Using this very general setup, our goal is threefold:

1. to obtain an efficient maximum likelihood estimator for θ;

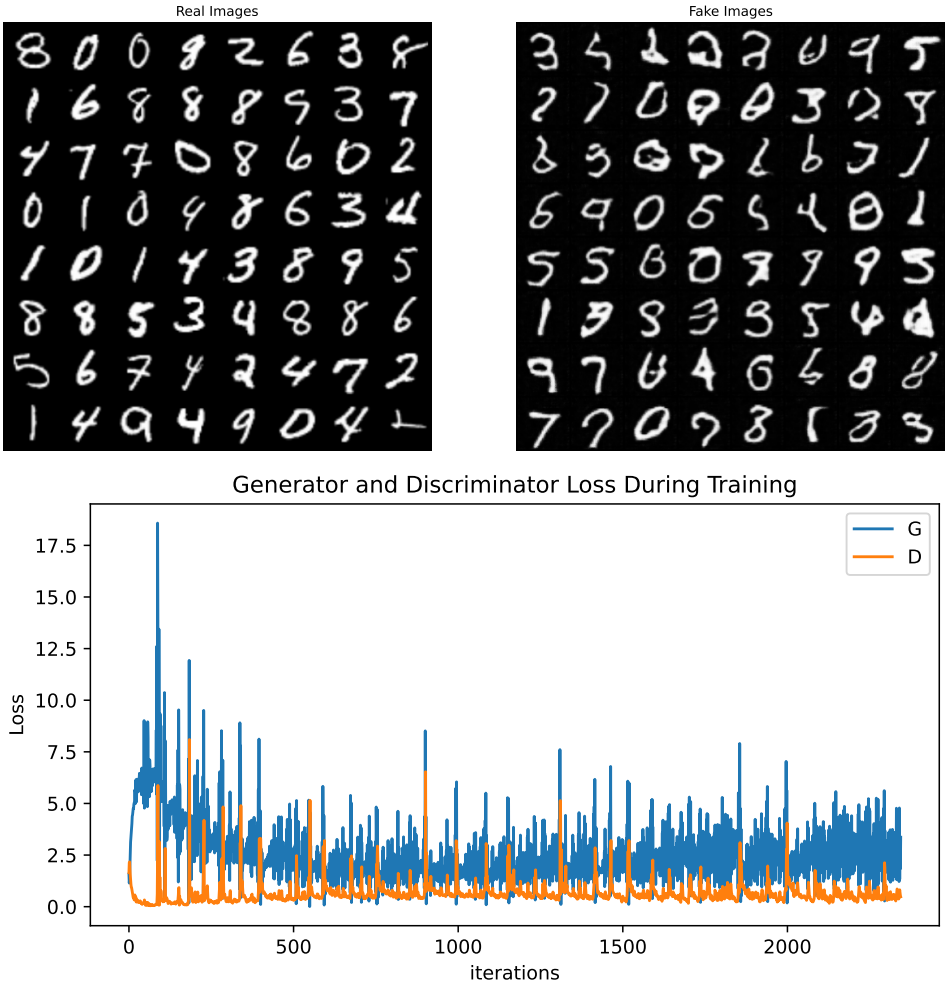

Figure 6.5: GAN on MNIST

2. to construct an efficient approximation of the posterior distribution $p_\theta(z \mid x)$;
3. to develop a methodology to directly synthesise new data distributed according to $p_\theta(x)$.

These objectives are non-trivial. In particular, we must design methods that are robust to potential intractabilities arising from specific choices of the model densities. Moreover, the models of interest are typically complex and high-dimensional, and can only be trained effectively in the large-data regime.

We will follow the standard VI approach seen in the previous chapters, that is, we will consider the variational approximation of the posterior $p_\theta(z \mid x)$ denoted $q_\phi(z \mid x)$, which will be learned by optimising the ELBO.

For a single datapoint [2] x_n, the ELBO is given by

$$\mathrm{ELBO}(\theta, \phi, x_n) = -\mathrm{KL}\left(q_\phi(z \mid x_n) \| p_\theta(z)\right) + \mathbb{E}_{q_\phi(z|x_n)}\left[\log p_\theta(x_n \mid z)\right]. \tag{6.6}$$

[2] If $p_\theta(x_{1:N}, z_{1:N}) = \prod_{n=1}^N p_\theta(x_n, z_n)$ and $q_\phi(z_{1:N} \mid x_{1:N}) = \prod_{n=1}^N q_\phi(z_n \mid x_n)$, then, just like the evidence, the ELBO is additive and thus factorises, i.e., $\mathrm{ELBO}(\theta, \phi, x) = \sum_{n=1}^N \mathrm{ELBO}(\theta, \phi, x_n)$

Figure 6.6: GAN on Celeba

Remark 6.10 (Generality of the formulation)**.**
Notice that the general setting defined so far allows for flexible and general modelling choices, since we have not assumed any form for the model's likelihood $p_\theta(x_n \mid z)$, prior $p_\theta(z)$, and variational approximation $q_\phi(z \mid x_n)$. We are therefore free to choose expressive models for these densities optimised wrt the ELBO in eq. (6.6).

Remark 6.11 (Autoencoder architecture)**.**
Observe that this optimisation procedure implies an autoencoding architecture: for a given choice of the prior $p_\theta(z)$, optimising the ELBO gives i) an encoder model $q_\phi(z \mid x_n)$ that maps the data to the latent space, and then ii) a decoder $p_\theta(x_n \mid z)$ that maps the latent code to the ambient space. The model will then be specified by adopting flexible parametrisations of these densities using neural networks.

For completeness, we define the following objects.

Definition 6.2.
We will refer to $q_\phi(z \mid x_n)$ and $p_\theta(x_n \mid z)$ as the (probabilistic) encoder and decoder respectively.

Sample approximation of the ELBO. We are now interested in maximising the ELBO, and we will approach that problem using gradient-based optimisation. Since the ELBO—including the KL term in eq. (6.6)—can be expressed as the expectation of $f : \mathcal{Z} \to \mathbb{R}$ given by

$$f(z) = -\log \frac{q_\phi(z \mid x_n)}{p_\theta(z)} + \log p_\theta(x_n \mid z) \tag{6.7}$$

wrt $q_\phi(z \mid x_n)$, recall that its gradients can be approximated using Monte Carlo samples as

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x_n)}\left[f(z)\right] = \mathbb{E}_{q_\phi(z|x_n)}\left[f(z)\nabla_\phi \log q_\phi(z \mid x_n)\right] \simeq \frac{1}{L}\sum_{l=1}^{L} f(z^{(l)})\nabla_\phi \log q_\phi(z^{(l)} \mid x_n), \tag{6.8}$$

where $z^{(l)} \sim q_\phi(z \mid x_n)$. However, it has been shown that this estimator exhibits a large variance and is thus unsuitable for our setting.

To introduce a practical estimator of the ELBO and its gradients, we adopt two main considerations that will allow us to calculate eq. (6.6). For the first term (the KL divergence), recall that the general setting defined so far allows for flexible models for both the prior over latent $p_\theta(z)$ and also the likelihood $p_\theta(x \mid z)$. Since the latent variable in this formulation is not tied to have any particular physical interpretation, this generality might be seen as a redundant modelling choice: considering a simple model for the prior $p_\theta(z)$ should not be a limitation to design an expressive model $p_\theta(x)$, since the required flexibility can be represented through the likelihood $p_\theta(x \mid z)$. As a consequence, we will assume we are free to choose a simple—and thus interpretable, easy to sample—prior $p_\theta(z)$.

Given such a choice for the prior of the latent variable, the choice of the variational approximation $q_\phi(z \mid x_n)$ can also be done with tractability of the KL term in mind. Note that $q_\phi(z \mid x_n)$ is not a component of the generative model, but rather of the approximation.

Remark 6.12 (Tractable KL term)**.**

The consideration above suggests that, in some cases, we can choose both $p_\theta(z)$ and $q_\phi(z \mid x_n)$ such that the KL between them is tractable. Note that this does not restrict the modelling generality of the decoder model, which can be chosen to have a tractable form, e.g., Gaussian or Bernoulli distributions, with parameters controlled by a neural network.

A second consideration is to perform a Monte Carlo approximation of the loss in eq. (6.6) bypassing direct sampling from the decoder $z^{(l)} \sim q_\phi(z \mid x_n)$ as stated above, using the reparametrisation trick. That is, we will assume that the sample $z^{(l)}$ can be expressed as a deterministic transformation of an auxiliary variable $\epsilon \in \mathbb{R}$, $\sim p(\epsilon)$ through the function

$$g_\phi : \mathbb{R} \times \mathcal{X} \to \mathcal{Z} \tag{6.9}$$

$$(x, \epsilon) \mapsto z = g_\phi(x, \epsilon). \tag{6.10}$$

With this change of variables, the expectation under the variational distribution can be expressed as

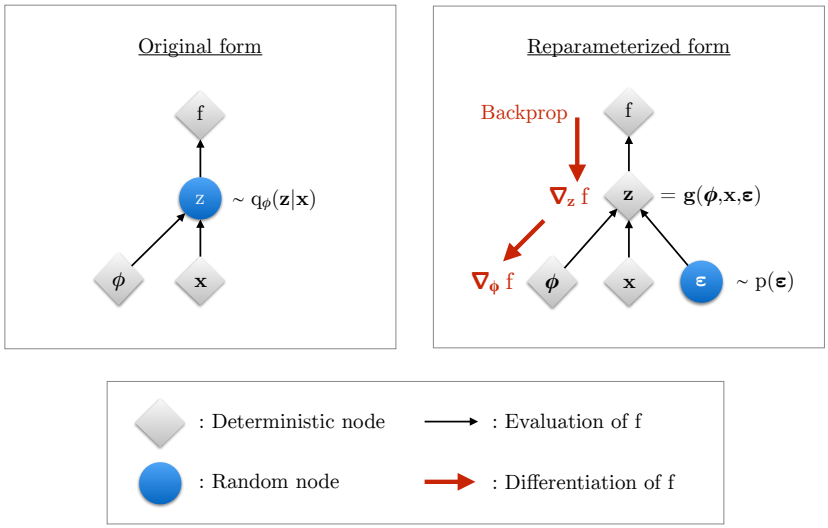

Figure 6.7: Illustration of the reparametrisation trick. Taken from (Kingma & Welling, 2019)

$$\mathbb{E}_{q_\phi(z|x_n)}\left[f(z)\right] = \mathbb{E}_{p(\epsilon)}\left[f(z)\right], \tag{6.11}$$

where $z = g_\phi(x, \epsilon)$.

Since the expectation under $p(\epsilon)$ no longer depends on the variational parameters as in eq. (6.8), the gradient and expectation operators can be interchanged, and thus we can produce a simple Monte Carlo estimate of the gradient given by

$$\begin{aligned}\nabla_\phi \mathbb{E}_{q_\phi(z|x_n)}\left[f(z)\right] &= \nabla_\phi \mathbb{E}_{p(\epsilon)}\left[f(z)\right] \\ &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\phi f(z)\right] \\ &\simeq \frac{1}{L}\sum_{l=1}^{L} \nabla_\phi f(z^{(l)}).\end{aligned} \tag{6.12}$$

Remark 6.13.

The introduction of the reparameterisation trick in this context has several key advantages. First, the resulting gradient estimator is unbiased (Kingma & Welling, 2019). Second, it typically has significantly lower variance than estimators that differentiate through the sampling distribution itself, because the randomness is isolated in an auxiliary variable and gradients are taken through a deterministic mapping—see Fig. 6.7. Third, by disentangling the source of randomness from the variational parameters, the objective becomes directly differentiable with respect to ϕ, allowing us to use standard automatic differentiation and backpropagation.

To do: Replace Fig. 6.7 by own diagram

Remark 6.14.

A natural question concerns the choice of the mapping g_ϕ. By construction, g_ϕ must

define a pushforward of a simple base distribution:

$$z = g_\phi(x, \epsilon), \qquad \epsilon \sim p(\epsilon),$$

so that the induced distribution of z matches the variational family $q_\phi(z \mid x)$. In other words, $q_\phi(\cdot \mid x)$ is the pushforward of $p(\epsilon)$ under $g_\phi(x, \cdot)$.

This leaves two modelling choices: the base distribution $p(\epsilon)$ and the transformation g_ϕ. In practice, $p(\epsilon)$ is typically chosen to be simple (e.g. a standard Gaussian), and modelling flexibility is introduced through the parameterised map g_ϕ. For example, if $q_\phi(z \mid x)$ is Gaussian with mean $\mu_\phi(x)$ and diagonal covariance $\Sigma_\phi(x)$, we can do

$$g_\phi(x, \epsilon) = \mu_\phi(x) + \Sigma_\phi(x)^{1/2}\epsilon.$$

Since more expressive variational families can be obtained by choosing richer transformations g_ϕ, in practice, $p(\epsilon)$ can be fixed and then g_ϕ can be designed to be sufficiently expressive while remaining differentiable and computationally tractable.

The variational autoencoder. We are now able to introduce a particular architecture to implement the inference procedure described above. Let us consider a fixed (i.e., parameter-free) prior for the latent variable given by an isotropic multivariate Gaussian

$$p_\theta(z) = \mathcal{N}(z; 0, I). \tag{6.13}$$

Furthermore, we will model the decoder $p_\theta(x \mid z)$ as a multivariate Gaussian (in the case of continuous data) or a Bernoulli (in the case of binary data), both with parameters θ given by a neural network that takes z as input. Though the posterior $p_\theta(z \mid x)$ is intractable, given that the prior on the latent variable is Gaussian, we will choose a Gaussian variational approximation for the posterior, i.e., the encoder, given by

$$q_\phi(z \mid x) = \mathcal{N}(z \mid \mu_\phi(x), \sigma^2_\phi(x) I), \tag{6.14}$$

where the variational parameters $\mu_\phi(x), \sigma^2_\phi(x)$ are also controlled by a neural network taking x as input.

Note that in this case implementing the reparametrisation trick is straightforward due to the Gaussian assumption for the encoder: sampling $\epsilon \sim \mathcal{N}(0, I)$, and setting $z = g_\phi(x, \epsilon) = \mu_\phi(x) + \sigma_\phi(x)\epsilon$ we obtain samples from eq. (6.14). Furthermore, since in this case both the prior $p_\theta(z)$ and the encoder $q_\phi(z \mid x)$ are Gaussian, the KL term in the ELBO can be computed explicitly, meaning that only the *expected reconstruction error* needs to be approximated via Monte Carlo—see Remark 6.12.

The resulting objective for a single datapoint x_n is

$$\mathrm{ELBO}(\theta, \phi, x_n) \simeq \frac{1}{L}\sum_{l=1}^{L} \log p_\theta\Big(x_n \mid z_n^{(l)}\Big) - \frac{1}{2}\sum_{j=1}^{d}\Big(\mu_j(x_n)^2 + \sigma_j(x_n)^2 - \log \sigma_j(x_n)^2 - 1\Big), \tag{6.15}$$

where $z_n^{(l)} = \mu_\phi(x_n) + \sigma_\phi(x_n)\epsilon^{(l)}$ and $\epsilon^{(l)} \sim \mathcal{N}(0, I)$.

Example 6.3 (VAE on the MNIST dataset)**.**

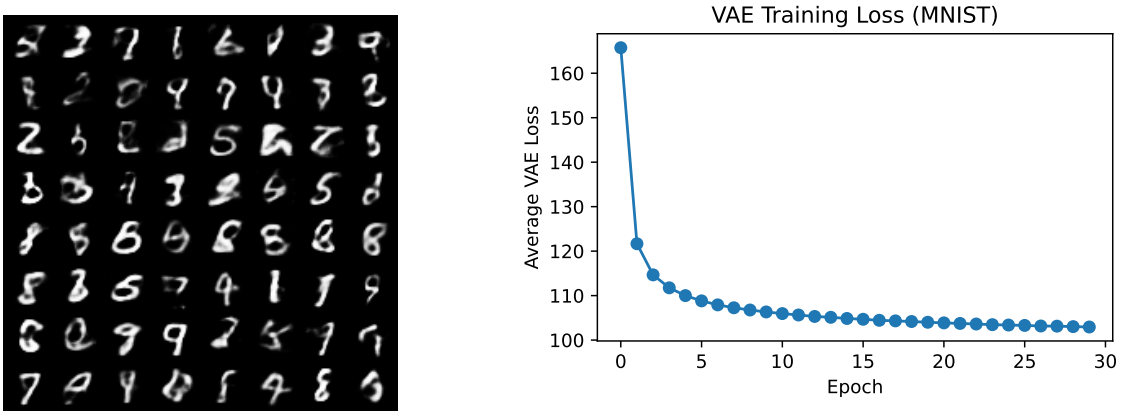

Figure 6.8: VAE on MNIST

We implemented a VAE with the architecture described above on the MNIST dataset. The model was trained over 30 epochs and the results are shown in Fig. 6.8. Observe the monotonic behaviour of the loss and the reasonable synthetic digits.

Week 7

Sequence models

7.1 Probabilistic modelling of discrete sequences

Across several disciplines, it is required to perform *sequence modelling*. Given an ordered collection of discrete symbols

$$x_{1:N} = (x_1, x_2, \ldots, x_N),$$

where each x_n belongs to a finite (or countable) vocabulary $\mathcal{V}$, we aim to construct a probabilistic model for this sequence. Understanding such a structure would enable task such as compression, next-symbol prediction, or in the case of text, translation and summarisation.

With no prior assumption of similarity or proximity in $\mathcal{V}$, the problem has combinatorial complexity. From a probabilistic perspective, the distribution of the sequence can be factorised as

$$p(x_{1:N}) = \prod_{n=1}^{N} p(x_n \mid x_{1:n-1}),$$

where the sequence modelling problem reduces to the next-symbol prediction task.

Observe that this factorisation becomes unfeasible in practice: since the conditioning statement $x_{1:n-1}$ grows with n, the transition probability cannot be coherently modelled with a fixed number of parameters. A standard approach is to assume a k-th order Markovian transition, that is,

$$p(x_n \mid x_{1:n-1}) \approx p(x_n \mid x_{n-k:n-1}).$$

This yields the k-gram factorisation

$$p(x_{1:N}) \approx \prod_{n=1}^{N} p(x_n \mid x_{n-k:n-1}),$$

where the transition kernel is modelled as a look-up table $T \in [0,1]^{|\mathcal{V}|^k \times |\mathcal{V}|}$ with entries trained from empirical counts. While meaningful, this approach has poor scalability properties, where the number of parameters required to specify all conditionals scales on the order of $|\mathcal{V}|^{k+1}$.

As a consequence, and even for moderately-sized vocabularies and context lengths k, these counting-based k-gram models suffer from severe training-samples sparsity and a direct manifestation of the curse of dimensionality.

Remark 7.1.

The fundamental limitation of k-gram models stems from the absence of underlying structure for the elements in $\mathcal{V}$, where the conditioning contexts $x_{n-k:n-1}$ are treated as unrelated atoms. Consequently, any two given contexts correspond to entirely distinct parameters even if they convey similar meanings. Therefore, k-gram models do not exploit any notion of similarity or shared structure among symbols or contexts, where the curse of dimensionality arises from a failure to impose meaningful inductive biases.

Language models. We can introduce a parametric model that identifies related contexts, thus replacing the look-up table of probabilities with a parameterised family

$$p_\theta(x_n \mid x_{1:n-1}),$$

where θ denotes a finite-dimensional set of parameters. Since parametric families operate on spaces with a notion of similarity, we must introduce a representation map (or embedding)

$$e : \mathcal{V} \to \mathbb{R}^d,$$

which assigns a continuous vector representation to each discrete symbol. This is also expected to promote generalisation, where symbols that convey similar meanings are mapped to nearby vectors. By composing the embeddings with the a parametric model, the next-token distribution can be modelled as

$$p_\theta(x_n = v \mid x_{1:n-1}) = \frac{\exp\Big(s_\theta(e(v), e(x_{1:n-1}))\Big)}{\sum_{v' \in \mathcal{V}} \exp\Big(s_\theta(e(v'), e(x_{1:n-1}))\Big)},$$

where s_θ is a learnable scoring function and $e(x_{1:n-1}) = (e(x_1), e(x_2), \ldots, e(x_{n-1}))$.

The parameters θ, including the embedding map e, are estimated by maximising the log-likelihood of observed sequences,

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} \log p_\theta(x_n \mid x_{1:n-1}).$$

In this formulation, a notion of similarity is exploited through the embedding map e and the parameters of s_θ, which is usually implemented by a feedforward network. If similar contexts induce similar embeddings, the model should be able to generalise beyond observed k-tuples. As a consequence, the exponential dependence on $|\mathcal{V}|^k$ is replaced by a parameter count that grows with the embedding dimension and network width.

Remark 7.2 (Distributed respresentations)**.**

One possible choice for the embedding map e is a one-hot encoding, that is, for a

vocabulary of size $|\mathcal{V}|$ we define

$$e(v) = e_i \in \mathbb{R}^{|\mathcal{V}|},$$

where e_i is the i-th canonical basis vector corresponding to token v. However, observe that this representation does not introduce any notion of similarity among tokens. Conversely, using a learnable embedding

$$e : \mathcal{V} \to \mathbb{R}^d, \qquad d \ll |\mathcal{V}|,$$

whose coordinates are trained jointly with the rest of the model, allows tokens with similar functional or semantic roles to acquire similar representations in $\mathbb{R}^d$, thereby enabling generalisation across related symbols.

These are referred to as *distributed representations* (Bengio, Ducharme, Vincent & Jauvin, 2003) because the information associated with a token is not stored in a single dedicated coordinate (as in a one-hot vector), but rather distributed across many components of its embedding vector. As a consequence, each coordinate affects the representation of many different tokens, where meaning is thus encoded in patterns of activations (rather than single coordinates).

Remark 7.3.

Although distributed representations alleviate the combinatorial explosion, this parametric approach remains constrained by a fixed context window of size k. Therefore, all information outside this window is discarded, irrespective of its semantic relevance. Thus, long-range dependencies cannot be captured unless k is increased, which again involves a computational challenge.

Recurrent neural networks (RNNs). To implement flexible, data-dependent, context lengths, RNNs (Elman, 1990) introduce a latent state variable $h_n \in \mathbb{R}^d$ that evolves recursively according to

$$h_n = f_\theta(h_{n-1}, e(x_n)),$$

where f_θ is a learnable nonlinear transformation. The next-token distribution is then defined as

$$p_\theta(x_{n+1} \mid x_{1:n}) = \mathrm{Softmax}\Big(W h_n\Big).$$

Though the hidden state h_n has a fixed size, it represents a summary of the entire past $x_{1:n}$, and the Markov assumption is now placed on the latent state rather than the observed sequence. Conceptually, this construction allows for modelling arbitrary context lengths, where the state is updated at every time step, preserving relevant information indefinitely by implementing a learned compression mechanism.

Despite eliminating the rigidity of fixed context windows, RNNs involve an information bottleneck, where the entire history $x_{1:n}$ must be compressed into a single fixed-dimensional state vector. Long Short-Term Memory (LSTM) networks partially address this issue by introducing a memory cell together with input, forget, and output gates (Hochreiter & Schmidhuber, 1997). These allow to preserve information over long and

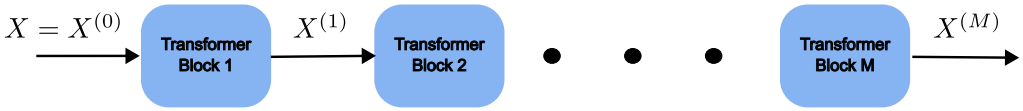

Figure 7.1: Diagram of the transformer blocks

short time horizons, and alleviates vanishing-gradient problems.

Remark 7.4.

The fundamental constraint of RNNs (including LSTMs) is that all past information must ultimately be encoded in a single evolving state vector. Also, due to the sequential recursion, full parallelisation is not possible. These limitations motivate a different perspective, where the model can access expressive per-token representations directly when needed, thus replacing sequential compression with content-based memory (thus leading to attention).

The attention paradigm. Attention was initially introduced as a mechanism augmenting recurrent encoder–decoder models (Bahdanau, Cho & Bengio, 2014), addressing the compression bottleneck by allowing the decoder to access all intermediate encoder states rather than relying solely on a single final hidden vector. Though this significantly improved long-range modelling, once direct access to all positions was made available through the attention mechanism, the temporal recurrence became unnecessary for modelling long-range dependencies. This shift in perspective paved the way to the *Attention Is All You Need* paradigm (Vaswani et al., 2017).

7.2 Overview of the Transformer architecture

Let us consider a sequence of token representations

$$x_1, x_2, \ldots, x_N \in \mathbb{R}^D$$

concatenated as a column-wise matrix

$$X = [x_1, x_2, \ldots, x_N]^\top \in \mathbb{R}^{N \times D}.$$

The representations can be fixed such as a one-hot encoding of symbols, a pre-learned representation such as word2vec (Mikolov, Chen, Corrado & Dean, 2013), or a learnable representation to be trained alongside the rest of the model, and apply to any type of data that can be *tokenised*.

We aim to construct a series of representations of the form $X^{(0)}, X^{(1)}, \ldots, X^{(M)}$, where $X^{(0)} = X$, and the transition from $X^{(m)}$ to $X^{(m+1)}$, $m \in \{0, \ldots, M-1\}$, is modelled by a unit referred here to as a Transformer block, as illustrated in Fig. 7.1. We next describe the structure of this unit.

Attention. For each $X^{(m)}$, we would like to construct a set of enriched representations

$$z_1, z_2, \ldots, z_N,$$

where z_n conveys *contextualized* information of the entire sequence as opposed to the *isolated* representation x_n. For each token, this can be achieved by implementing a mapping of the form

$$z_n = f(x_n, x_1, \ldots, x_N),$$

where the feature map f is the same for all positions, therefore, exploiting a shared-parameter structure, and the output remains in $\mathbb{R}^D$.

This feature mapping can be implemented via an attention mechanism; a particular instance of this is defined as follows. For the token embeddings X, let us define

$$\begin{aligned} \text{Query:} \quad & Q = XW_Q \in \mathbb{R}^{N \times D_K} \\ \text{Key:} \quad & K = XW_K \in \mathbb{R}^{N \times D_K} \\ \text{Value:} \quad & V = XW_V \in \mathbb{R}^{N \times D_V}, \end{aligned}$$

where $W_Q, W_K \in \mathbb{R}^{D \times D_K}$ and $W_V \in \mathbb{R}^{D \times D_V}$ are learnable weight matrices.

The *scaled dot-product self-attention* of X is defined as

$$\mathrm{Att}(X) \stackrel{\text{def}}{=} \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{D_K}}\right) V \in \mathbb{R}^{N \times D_V},$$

giving the contextualised representation by

$$Z = \mathrm{Att}(X).$$

The attention matrix $QK^\top \in \mathbb{R}^{N \times N}$ contains pairwise similarity scores between all tokens. After scaling by $\sqrt{D_K}$ and applying the row-wise softmax, these scores become attention weights, so that each output representation is a convex combination of the value vectors. The role of the scaling by $\sqrt{D_K}$ is to control the magnitude of the dot products as the dimensionality of the query/key embeddings grows.

Intuitively, an implicit underlying assumption to construct the contextual representation is that different tokens should be able to depend on different parts of the sequence. The token at position n may pay attention to the token in position n', while another token may need to focus on another part of the sequence. Furthermore, contextualisation should not be fixed but rather content-dependent. This allows for each token to selectively aggregate information from the rest of the sequence according to the learning strategy (to be defined later). The attention mechanism achieves this by producing, for each token, a learned, data-dependent weighted combination of the input representations $x_1, \ldots, x_N$.

MLP stage. After the attention operation, the representation is linear in the token values and thus may not provide the required modelling capacity. To increase the expressivity of the model, a nonlinear transformation can be applied to the attention output Z. Following the same parameter-sharing concept above, this transformation should be the same for each token. This can be achieved by a position-wise multilayer perceptron (MLP)

$$z_n \mapsto \mathrm{MLP}_\theta(z_n), \forall n = 1, \ldots, N. \tag{7.1}$$

Residual connection. Rather than requiring each Transformer block to construct an entirely new representation from scratch, feature extraction can be facilitated by modelling the updated representation as an additive improvement of the previous one. This can be achieved by incorporating residual connections both in the attention and MLP stages, thus guiding the expressivity of each component to what needs to be refined rather than relearned (He, Zhang, Ren & Sun, 2016).

Additionally, this residual formulation not only encourages incremental (and, arguably, stable) representation learning, but also improves optimisation by preserving a direct path for backpropagating gradients across layers.

Normalisation. Lastly, as the learnt representations will propagate through multiple blocks, their magnitude can progressively vanish/explode and thus result in insensitive/unstable optimisation dynamics. Furthermore, with the aim to preserve the architecture of the attention block across layers, it is required that successive layers operate on well-conditioned inputs of similar magnitude.

This can be ensured by introducing a normalisation step that re-centres and re-scales the output of each component at each position (Ba, Kiros & Hinton, 2016). That is,

$$\bar{x}_{d,n} \stackrel{\text{def}}{=} [\text{LayerNorm}(X)]_{d,n} = \gamma_d \frac{h_{d,n} - \text{mean}\,(x_n)}{\sqrt{\text{var}\,(x_n) + \varepsilon}} + \beta_d, \tag{7.2}$$

where

$$\text{mean}\,(x_n) = \frac{1}{D}\sum_{d=1}^{D} x_{d,n}; \quad \text{var}\,(x_n) = \frac{1}{D}\sum_{k=1}^{D} \Big(x_{d,n} - \text{mean}\,(x_n)\Big)^2,$$

$\gamma_d, \beta_d \in \mathbb{R}$ are learnable scale and shift parameters, and $\varepsilon > 0$ is a small constant introduced for numerical stability.

Temporal (positional) encoding. With the components presented so far, the Transformer block is invariant to permutations of the input tokens. This can be a limitation in some cases, since it only allows to learn relationships among tokens that depend on their content, and not on their order. For instance, consider the sentences *"the dog chased the cat"* and *"the cat chased the dog"*. Though the same tokens appear in both cases, it is clear that the actors have different roles, which leads to different meanings. A permutation-invariant mechanism would be unable to distinguish between the meanings of these sequences.

To encode temporal structure, each token representation can be augmented with a position-dependent feature before entering the Transformer block. One alternative is to simply add a function of the token position to the token representation x_n with

$$\tilde{x}_n = x_n + p(n),$$

where $p(n)$ only depends on the position n. This function can be fixed, such as a vector of sinusoids of different frequencies and phases encoding the token's position as done by (Vaswani et al., 2017), or a free parameter that is learnt with the rest of the model.

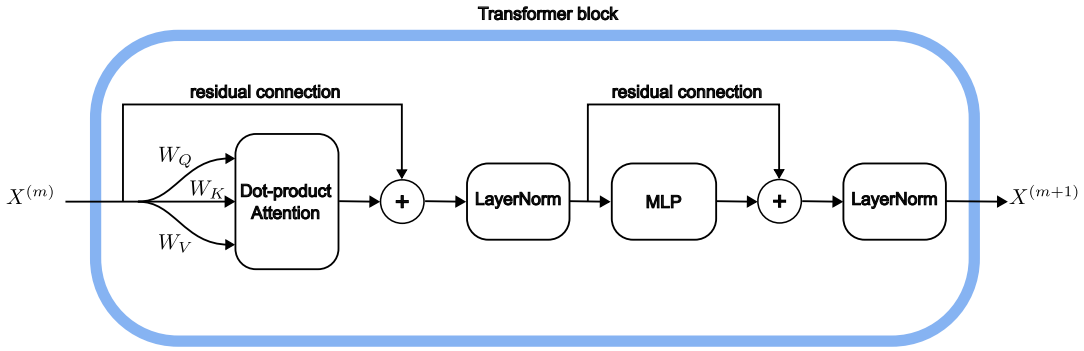

Figure 7.2: Diagram of a single transformer block, identifying attention, residual connections, normalisation and MLP stages.

Remark 7.5 (Architecture within each block)**.**
The attention block operates in two stages: first, attention performs global information exchange across tokens, and the MLP performs local nonlinear feature refinement. After each of these stages a residual connection and normalisation is included both to aid the feature extraction and make training more stable. See Fig. 7.2 for an illustrative diagram.

Remark 7.6 (How are the parameters learnt?)**.**
So far, it may make little sense how all the introduced parameters will be learnt, so that they adopt the role assigned to them. Recall that the Transformer is a processing stage taking part in a larger architecture usually used to address and end-to-end task such as a next-word predictor or an object recognition network. Therefore, all the parameters will be learnt alongside the rest of the architecture to address the given task.

7.3 Attention in more detail

The attention module (general form). Let $Q = \{q_i\}_{i=1}^{N_Q}$ be a set of queries, $K = \{k_j\}_{j=1}^{N_K}$ a set of keys, and $V = \{v_j\}_{j=1}^{N_V}$ a set of values, with $q_i, k_j \in \mathbb{R}^{D_K}$ and $v_j \in \mathbb{R}^{D_V}$.

An attention module is a mapping that produces, for each query q_i, a weighted combination of the values:

$$\mathrm{Att}(q_i, K, V) = \sum_{j=1}^{n_k} \alpha_{ij} v_j,$$

where the attention weights α_{ij} satisfy

$$\alpha_{ij} = \frac{\exp\Big(s(q_i, k_j)\Big)}{\sum_{\ell=1}^{n_k} \exp\Big(s(q_i, k_\ell)\Big)}.$$

Here $s : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a *similarity* function measuring the relevance of key k_j to query q_i. Therefore, attention computes a data-dependent convex combination of values, with weights determined by query–key similarity function $s(\cdot, \cdot)$.

Remark 7.7 (Scaled dot-product self-attention)**.**
The scaled dot-product self-attention mechanism used in transformer architectures is a particular instance of the general attention module defined above, where:

- The queries, keys, and values are all linear projections of the same input sequence $X = [x_1, x_2, \ldots, x_N]^\top \in \mathbb{R}^{N\times D}$ (hence "self"-attention):

$$q_n = W_Q x_n, \quad k_n = W_K x_n, \quad v_n = W_V x_n,$$

  where $W_Q, W_K \in \mathbb{R}^{D\times D_K}$ and $W_V \in \mathbb{R}^{D\times D_V}$.

- The compatibility function is chosen to be the scaled dot product:

$$s(q_i, k_j) = \frac{q_i^\top k_j}{\sqrt{D_K}}.$$

Remark 7.8.
Though the inner product might seem somewhat limiting as a similarity measure, recall that this inner product is taken on a trained high-dimensional embedding space that is subsequently refined through a series of Transformer blocks.

In tasks such as translation, style transfer, conditional generation or sequence-to-sequence problems in general, it is often desirable to understand how one sequence interacts with another sequence. From the core definition of attention, it is natural to ask how the same mechanism can be used to describe how a sequence "attends" to another.

Cross-Attention. To extend the general attention module so that the queries and the key–value pairs come from *different* sources, let $X = \{x_i\}_{i=1}^{N_Q}$ and $Y = \{y_j\}_{j=1}^{N_K}$ be two distinct sequences, e.g., a target sequence and a source sequence in sequence-to-sequence models. Queries are computed from X, while keys and values are computed from Y, i.e.,

$$q_i = W_Q x_i, \quad k_j = W_K y_j, \quad v_j = W_V y_j.$$

The output for each q_i, $i = 1, \ldots, N_Q$, is then

$$\mathrm{Att}(q_i, K, V) = \sum_{j=1}^{N_K} \frac{\exp(s(q_i, k_j))}{\sum_{\ell=1}^{N_K} \exp(s(q_i, k_\ell))}\, v_j.$$

Thus, cross-attention allows elements of one sequence to attend to, and aggregate information from, another sequence. In Transformer architectures, it is typically used in encoder–decoder models, where decoder queries attend to encoder key–value representations.

Kernel perspective. The convex combination of values computed by the attention mechanism can be interpreted as a data-dependent linear estimator, in a remarkable analogy to the predictors arising in Gaussian process regression and kernel methods. In kernel regression, given data $\{(x_j, y_j)\}_{j=1}^{n}$ and a positive definite kernel $k(\cdot,\cdot)$, the

prediction at a query (or test) point x takes the form

$$\hat{f}(x) = \sum_{j=1}^{n} w_j(x)\, y_j,$$

where the weights depend on kernel evaluations between the query and the data,

$$w(x) = K(x, X)\, K(X, X)^{-1}, \qquad K(x, X) = \Big(k(x, x_1), \ldots, k(x, x_n)\Big).$$

Thus, the estimator is linear in the observations, with the coefficients determined by similarities between the query and the support points induced by the chosen kernel.

Attention has the same structural form:

$$\mathrm{Att}(q_i, K, V) = \sum_{j=1}^{n_k} \alpha_{ij} v_j, \qquad \alpha_{ij} = \frac{\exp(s(q_i, k_j))}{\sum_\ell \exp(s(q_i, k_\ell))},$$

where the similarity function $s(q_i, k_j)$ can be related to the kernel, and the weights determine how the values are combined. In kernel methods, the weights are obtained through algebraic operations involving kernel evaluations, or equivalently, linear operations in the corresponding feature space, again, resembling the dot-product attention structure seen above.

The main difference, however, is the use of the softmax normalisation in attention. Recall that the analogous mixing weights in kernel estimators need not be positive nor sum to one. In contrast, the softmax in the attention module enforces

$$\alpha_{ij} \geq 0, \qquad \sum_{j=1}^{n_k} \alpha_{ij} = 1,$$

so that attention produces a convex combination of values. This nonlinear normalisation step aids the propagation of learnt relationships between tokens across layers.

The kernel analogy can also be identified in cross-attention, where the key–value pairs $\{(k_j, v_j)\}$ correspond to the observations of one sequence, and the queries $\{q_i\}$ correspond to the query locations from another sequence. This is resembles the structure of multioutput GPs, where one channel can be estimated from observations of another channel, based on a kernel specifying the covariances across space (channel) and time.

Multi-head attention. The focused similarity measure implemented by attention is often insufficient to capture the different types of relations occurring across sequences. For example, in the sentence *"The animal didn't cross the street because it was too tired,"* the token "it" must attend to "animal" through a semantic relation, while other tokens may attend according to syntactic dependencies (e.g., subject–verb pairs) or positional structure. This phenomenon is also present in continuous time series, where values of the series relate at different scales, thus giving way to, e.g., rough and smooth components; in GP regression, this is accounted for using multiple kernels which are linearly combined. This concept also applies to attention.

Since distinct interaction patterns correspond to different notions of similarity, we can

assume that each of these notions can be modelled by a single attention *head*. Formally, several attention heads can be implemented concurrently, each with its own learned projections. For $h = 1, \dots, H$, define

$$q_i^{(h)} = W_Q^{(h)} x_i, \quad k_j^{(h)} = W_K^{(h)} y_j, \quad v_j^{(h)} = W_V^{(h)} y_j,$$

and compute

$$\text{head}^{(h)}(q_i) = \sum_{j=1}^{n_k} \alpha_{ij}^{(h)} v_j^{(h)}, \qquad \alpha_{ij}^{(h)} = \frac{\exp\Big(s^{(h)}(q_i^{(h)}, k_j^{(h)})\Big)}{\sum_\ell \exp\Big(s^{(h)}(q_i^{(h)}, k_\ell^{(h)})\Big)}.$$

Each head therefore induces its own data-dependent linear estimator based on a distinct learned similarity function. The outputs of the H heads are concatenated and linearly projected:

$$\text{MHA}(q_i) = W_O \,\text{Concat}\Big(\text{head}^{(1)}(q_i), \dots, \text{head}^{(H)}(q_i)\Big),$$

where W_O is a trainable set of weights that mixes the contributions of each attention head.

7.4 Encoder-decoder architectures

Transformers' exceptional modelling abilities have been primarily used in two general-interest tasks: *representation learning* and *conditional generation*. These are implemented by the encoder and the decoder architectures respectively.

Let $X = (x_1, \dots, x_n)$ be an input sequence. The *encoder* computes a contextualised representation

$$H = \text{Enc}(X) = (h_1, \dots, h_n) \qquad h_i \in \mathbb{R}^d,$$

as described above using the composition of multiple transformer blocks. Here, each h_i aggregates information from all others elements $h_{\neq i}$. Intuitively, the encoder processes (or "reads") the full sequence and builds a structured internal description of it.

The *decoder*, by contrast, defines a conditional generative model over an output sequence $Y = (y_1, \dots, y_m)$, factorising the joint distribution autoregressively:

$$p(Y \mid X) = \prod_{t=1}^{m} p\Big(y_t \mid y_{<t}, H\Big), \qquad H = \text{Enc}(X).$$

The decoder implements the transformer blocks to compute a hidden state as

$$z_t = \text{Dec}(y_{<t}, H),$$

using masked self-attention so as to ensure that z_t depends only on previous outputs $y_{<t}$, and (optionally) cross-attention to condition the generation on the encoder representations H.

Remark 7.9.

The encoder computes global, bidirectional representations of an input sequence,

and the decoder defines an autoregressive conditional model that generates outputs sequentially (possibly conditioning on the encoder's output).

Transformer architectures differ in terms of their use of an encoder, a decoder, or both.

Example 7.1 (Encoder-only architecture: BERT)**.**
These architectures are used for representation learning tasks where the goal is to understand an input sequence, such as classification, sentence similarity, or token-level labeling. For example, *BERT* (Bidirectional Encoder Representations from Transformers), uses deep encoder stacks trained with masked language modeling and next-sentence prediction objectives (Devlin, Chang, Lee & Toutanova, 2018).

Example 7.2 (Decoder-only architecture: GPT)**.**
These models are used for autoregressive generation tasks, modelling a sequence left-to-right for text generation, dialogue systems, or code completion. Given a sequence $Y = (y_1, \ldots, y_T)$, the next-token conditional distribution is parameterized by a masked self-attention network via

$$p(y_t \mid y_{<t}) = \mathrm{Softmax}(W z_t).$$

These conditionals are trained by maximising the likelihood of observed sequences under the autoregressive factorization. For example, *GPT* (Generative Pretrained Transformer) implements stacked masked self-attention layers trained with next-token prediction (Radford, Narasimhan, Salimans & Sutskever, 2018).

Example 7.3 (Encoder-decoder architecture (Vaswani et al., 2017))**.**
The original Transformer architecture combines both components into a single encoder–decoder model designed for sequence-to-sequence tasks, such as machine translation, summarisation, or speech-to-text. This encoder–decoder structure, introduced in *"Attention Is All You Need"* (Vaswani et al., 2017), is particularly suited to tasks where one sequence must be transformed into another, such as translation or summarization. It combines full bidirectional understanding of the input with controlled, autoregressive generation of the output. See Fig. 7.3 for a diagram.

Pretraining and fine-tuning. Decoder-only language models are first trained in a large-scale *pretraining* phase using causal, i.e., next-word, prediction in a task-agnostic manner. In the next stage, referred to as *fine-tuning*, the pretrained model is adapted to particular objectives using curated datasets, such as instruction–response pairs or alignment procedures designed to shape behaviour according to human preferences.

Although the training objective remains simple next-token prediction, sufficiently large models capture complex linguistic (and arguably reasoning) patterns. Because many tasks can be formulated directly in natural language, e.g., "Translate this sentence to French" or "Summarise the following paragraph", the modelling power of these architecture renders the design of distinct models for each task unnecessary. Instead, the task itself is specified as part of the input prompt, and the model produces the appropriate output as a continuation of the sequence.

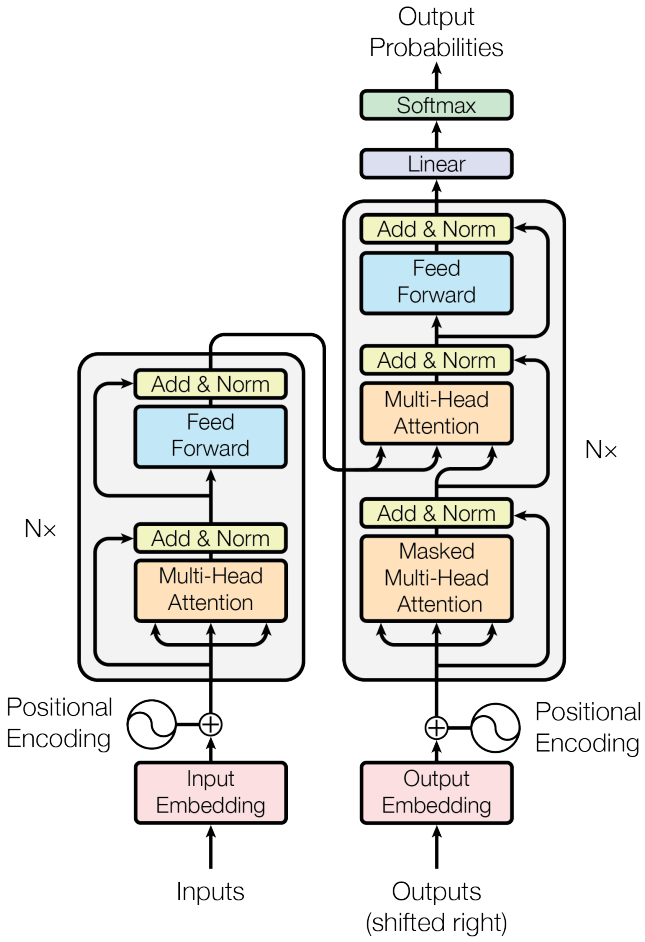

Figure 7.3: Encoder-Decoder architecture in the original transformer. Taken from (Vaswani et al., 2017)

References

Andrieu, C., de Freitas, N., Doucet, A. & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine Learning*, *50*(1–2), 5–43.

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K. & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y., Ducharme, R., Vincent, P. & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, *3*, 1137–1155.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems* (Vol. 26, pp. 2292–2300).

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. Cambridge, MA: MIT Press. Retrieved from `https://www.deeplearningbook.org`

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems*, *27*.

He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of CVPR*.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Kingma, D. P. & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. & Welling, M. (2019). *An introduction to variational autoencoders* (Vol. 12) (No. 4). Now Publishers. doi: 10.1561/2200000056

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Murphy, K. P. (2022). *Probabilistic machine learning: An introduction* (1st ed.). Cambridge, MA, USA: The MIT Press.

Orbanz, P. (2014). *Lecture notes on bayesian nonparametrics.* Lecture notes. Retrieved from `https://www.gatsby.ucl.ac.uk/~porbanz/papers/porbanz_BNP_draft.pdf` (Accessed: 2026-01-29)

Peyré, G. & Cuturi, M. (2019). *Computational optimal transport: With applications to data science*. Now Publishers. Retrieved from `https://www.nowpublishers.com/article/Details/MAL-089`

Radford, A., Metz, L. & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations (ICLR)*.

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian processes for machine learning.* MIT Press. Retrieved from `http://www.gaussianprocess.org/gpml`

Sinkhorn, R. (1964). A relationship between arbitrary positive matrices and doubly stochastic matrices. *The Annals of Mathematical Statistics*, *35*(2), 876–879. doi: 10.1214/aoms/1177703591

Thorpe, M. (2018). *Introduction to optimal transport.* Cambridge, UK. Retrieved from `https://www.damtp.cam.ac.uk/research/cia/files/teaching/Optimal_Transport_Notes.pdf` (Lecture notes, Lent 2018. Current version: Thursday 8th March, 2018)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.