SC4/SM8 Advanced Topics in Statistical Machine Learning

Department of Statistics, University of Oxford
`https://canvas.ox.ac.uk/courses/65445`

Lecturer: Tom Rainforth

Notes adapted and extended from previous materials by
Dino Sejdinovic and Yee Whye Teh

Hilary Term 2021

Contents

1 Machine Learning Paradigms

How come a dog is able to catch a frisbee in mid-air? How come a batsman can instinctively predict the flight of a cricket ball, moving at over 100km/h, sufficiently accurately and quickly to hit it when there is not even time to consciously make a prediction? Clearly, neither can be based on a deep explicit knowledge of the laws of physics or some hard-coded model for the movement of objects; we are not even born with the knowledge that unsupported objects will fall down [3]. The only reasonable explanation for these abilities is that the batsmen and the dog have *learned from experience*. We do not have all the knowledge we require to survive from birth, but we are born with the ability to learn and adapt, making observations about the world around us and using these to refine our cognitive models for everything from the laws of physics to social interaction. Classically the scientific method has relied on human interpretation of the world to formulate explicit models to explain our internal intuitions, which we then test through experimentation. However, even as a whole scientific society, our models are often terribly inferior to the subconscious models of animals and children, such as for most tasks revolving around social interaction. This leads one to ask, is there something fundamentally wrong with this hand-crafted modelling approach? Is there another approach that better mimics the way humans themselves learn?

Machine learning is an appealing alternative, and often complementary, approach that focuses on constructing algorithms and systems that can adapt, or learn, from data in order to make predictions that have not been explicitly programmed. This is exciting not only because of the potential it brings to automate and improve a wide array of computational tasks, but because it allows us to design systems capable of going beyond the boundaries of human understanding, reasoning about and making predictions for tasks we cannot solve directly ourselves. As a field, machine learning is very wide ranging, straddling computer science, statistics, engineering, and beyond. It is perhaps most closely related to the field of computational statistics, differing predominantly in its emphasis on prediction rather than understanding. Despite the current hype around the field, most of the core ideas have existed for some time, often under the guise of pattern recognition, artificial intelligence, or computational statistics. Nonetheless, the explosion in the availability of data and in computational processing power in recent years has led to a surge of interest in machine learning by academia and industry alike, particularly in its application to real world problems. This interest alone is enough to forgive the hype, as the spotlight is not only driving the machine learning community itself forward, but helping identify huge numbers of applications where existing techniques can be transferred to fantastic effect. From autonomous vehicles [36], to speech recognition [28], and designing new drugs [12], machine learning is rapidly becoming a crucial component in many technological and scientific advancements.

1.1 Learning From Data

Machine learning is all about learning from *data*. In particular, we typically want to use the data to learn something that will allow us to make *predictions* at unseen datapoints. This emphasis on prediction is what separates machine learning from the field of *computational statistics*, where the aim is typically more to learn about parameters of interest. Inevitably though, the line between these two is often blurry. Like with most fields, the term machine learning does not have a precise infallible definition, it is more a continuum of ideas spanning a wide area of statistics, computer science, engineering, applications, and beyond.

With some notable exceptions that we will discuss later, the starting point for most machine learning algorithms is a *dataset*. Most machine learning methods can be delineated based on the type of dataset they work with, and so we will first introduce some of the most common types of categorization. Note that these are not exhaustive.

1.1.1 Supervised Learning

Supervised learning is arguably the most natural and common machine learning setting. In supervised learning, our aim is to learn a *predictive model* f that takes an input $x \in \mathcal{X}$ and aims to predict its corresponding output $y \in \mathcal{Y}$. Learning f is done by using a *training dataset* $\mathcal{D}$ that is comprised of a set of input–output pairs: $\mathcal{D} = \{x_n, y_n\}_{n=1}^N$. This is sometimes referred to as *labelled data*. The hope is that these example pairs can be used to "teach" f how to accurately make predictions.

The two most common types of supervised learning are **regression** and **classification**. In regression, the outputs we are trying to predict are numerical, such that $\mathcal{Y} \subseteq \mathbb{R}$ (or, in the case of multi-variate regression, $\mathcal{Y} \subseteq \mathbb{R}^d$ for some $d \in \mathbb{N}^+$). Common examples of regression problems include curve fitting and many empirical scientific prediction models; example regression methods include linear regression, Gaussian processes, and deep learning. In classification, the outputs are categorical variables, such that $\mathcal{Y}$ is some discrete (and typically non-ordinal) set of possible output values. Common example classification problems include image classification and medical diagnosis; example classification methods include random forests, support vector machines, and deep learning. Note that many supervised machine learning methods are capable of handling both regression and classification.

1.1.2 Unsupervised Learning

Unlike supervised learning, **unsupervised learning** methods do not have a single unified predictive goal. They are generally categorized by the data having no clear output variable that we are attempting to predict, such that we just have a collection of example datapoints rather than explicit input–output pairs, i.e. $\mathcal{D} = \{x_n\}_{n=1}^N$. This is sometimes referred to as *unlabelled data*.

In general, unsupervised learning methods look to exact some salient features for the dataset, such as underlying structure, patterns, or characteristics. They are also sometimes used to simplify datasets so that they can be more easily interacted with by humans or other algorithms. Common types of unsupervised learning include clustering, feature extraction, density estimation, representation learning, data visualization, data mining, data compression, and some model learning. A host of different methods are used to accomplish these tasks, with approaches that leverage deep learning becoming increasingly prominent.

1.1.3 Semi–Supervised Learning

As the name suggests, **semi–supervised learning** is somewhere in–between supervised and unsupervised learning. It is generally characterized by the presence of a dataset where not all the inputs variables have corresponding output; i.e. only some of a datapoints are *labelled.* In particular, many semi-supervised approaches focus on cases where there is a large amount of unlabelled data available, but only a small amount of labelled data.

The aim of semi–supervised learning can depend on the context. In many, and arguably most, cases, the aim is to use the unlabelled data to assist in learning a predictive model, e.g. through uncovering structure or patterns that aid in training the model or help to better generalize to unseen inputs. However, there are also some cases where the labels are instead used to try and help with more unsupervised–learning–orientated tasks, such as learning features with strong predictive power, or performing representation learning in a manner that emphasizes structure associated with the output labels.

1.1.4 Notable Exceptions

There are also a number of interesting machine learning settings where one does not start with a dataset, but must either gather the data during the learning process, or even simulate our own pseudo data.

Perhaps the most prominent example of this is **reinforcement learning**. In reinforcement learning, one must "learn on the fly:" there is an agent which must attempt an action or series of actions before receiving a reward for those choices. The agent then learns from these rewards to improve its actions over time, with the ultimate aim being to maximize the long-term reward (which can be either cumulative or instantaneous). Reinforcement learning still uses data through its updating based on previous rewards, but it must also learn how to collect that data as well, leading to quite distinct algorithms. It is often used in settings where an agent must interact with the physical world (e.g. self-driving cars) or a simulator (e.g. training AIs for games).

Other examples of machine learning frameworks that do not fit well into any of the aforementioned categories include experimental design, active learning, meta-learning, and collaborative filtering.

1.2 Discriminative Machine Learning

In some machine learning applications, huge quantities of data are available that dwarf the information that can be provided from human expertise. In such situations, the main challenge is in processing and extracting all the desired information from the data to form a useful characterization, typically an artefact providing accurate predictions at previous unseen inputs. Such problems are typically suited to **discriminative machine learning** approaches [10, 58], such as neural networks, support vector machines, and decision tree ensembles.

Discriminative machine learning approaches are predominantly (albeit not exclusively) used for supervised learning tasks. They focus on directly learning a predictive model: given training data $\mathcal{D} = \{x_n, y_n\}_{n=1}^{N}$, they learn a parametrized mapping f_θ from the inputs $x \in \mathcal{X}$ to the outputs $y \in \mathcal{Y}$ that can be used directly to make predictions for new inputs $x \notin \{x_n\}_{n=1}^{N}$. **Training** uses the data $\mathcal{D}$ to estimate optimal values of the parameters θ^*. Prediction at a new input x involves applying the mapping with an estimate of the optimal parameters $\hat{\theta}$ giving an estimate for the output $\hat{y} = f_{\hat{\theta}}(x)$. Some methods may also return additional prediction information instead of just the output itself. For example, in a classification task, we might predict the probability of each class, rather than just the class.

Perhaps the simplest example of discriminative learning is linear regression: one finds the hyperplane that best represents the data and then uses this hyperplane to interpolate or extrapolate to previously unseen points. As a more advanced example, in a neural network one uses training to learn the weights of the network, after which prediction can be done by running the network forwards.

There are many intuitive reasons to take a discriminative machine learning approach. Perhaps the most compelling is the idea that if our objective is prediction, then it is simplest to solve that problem directly, rather than try and solve some more general problem such as learning an underlying generative process [10, 58]. Furthermore, if sufficient data is provided, discriminant approaches can be spectacularly successful in terms of predictive performance. Many discriminative methods are highly flexible and can capture intricate structure in the data that would be hard, or even impossible, to establish manually. Many approaches can also be run with little or no input on behalf of the user, delivering state-of-the-art performance when used "out-of-the-box" with default parameters.

However, this black-box nature is also often their downfall. Discriminative methods sometimes make such weak assumptions about the underlying process that is difficult to impart prior knowledge or domain-specific expertise. This can be disastrous if insufficient data is available, as the data alone is unlikely to possess the required information to make adequate predictions. Even when substantial data is available, there may be significant prior information available that needs to be exploited for effective performance. For example, in time series modelling the sequential nature of the data is critically important information.

The difficultly in incorporating assumptions about the underlying process varies between different discriminative approaches. Much of the success of neural networks (i.e. deep learning) stems from the fact that they still provide a relatively large amount of flexibility to adapt the framework to a particular task, e.g. through the choice of architecture. At the other extreme, random forest approaches provide very little flexibility to adjust the approach to a particular task, but are still often the best performing approaches when we require a fully black-box algorithm that requires no human tuning to the specific task [46].

Not only does the black-box nature of many discriminative methods restrict the level of human input that can be imparted on the system, it often restricts the amount of insight and information that can be extracted *from* the system once trained. The parameters in most discriminative algorithms do not have physical meaning that can be queried by a user, making their operation difficult to interpret and hampering the process of improving the system through manual revision of the algorithm. Furthermore, this typically makes them inappropriate for more statistics orientated tasks, where it is the parameters themselves which are of interest, rather than the ability for the system itself to make predictions. For example, the parameters may have real-world physical interpretations which we wish to learn about.

Most discriminative methods also do not naturally provide realistic uncertainty estimates. Though many methods can produce uncertainty estimates either as a by-product or from a post-processing step, these are typically heuristic based, rather than stemming naturally from a statistically principled estimate of the target uncertainty distribution. A lack of reliable uncertainty estimates can lead to overconfidence and can make certain discriminative methods inappropriate in many scenarios, e.g. for any application where there are safety concerns. It can also reduce the composability of a methods within larger systems, as information is lost when only providing a point estimate.

1.3 Generative Machine Learning

These shortfalls with discriminative machine learning approaches mean that many tasks instead call for a **generative machine learning** approach [6, 20, 40]. Rather than directly learning a predictor, generative methods look to explain the observed data using a *probabilistic model*, sometimes known as a **generative model**. Whereas discriminative approaches aim only to make predictions, generative approaches model how the data is actually generated: they model the joint probability P_{XY} of the inputs X and outputs Y. By comparison, we can think of discriminative approaches as only modeling the outputs given the inputs $Y|X$. For this reason, most techniques for unsupervised learning are based on a generative approach, as here we have no explicit outputs.

Because they construct a joint probability model, generative approaches generally make stronger modeling assumptions about the problem than discriminative approaches. Though this can be problematic when the model assumptions are wrong and is often unnecessary in the limit of large data, it is essential for combining prior information with data and therefore for constructing systems that exploit application-specific expertise. In the eternal words of George Box [8],

All models are wrong, but some are useful.

In a way, this is a self-fulfilling statement: a model for any real phenomena is, by definition, an approximation and so is never exactly correct, no matter how powerful. However, it is still an essential point that is all too often forgotten, particularly by academics trying to convince the world that only their approach is correct. Only in artificial situations can we construct exact models and so we must remember, particularly in generative machine learning, that the first, and often largest, error is in our original mathematical abstraction of the problem. On the other hand, real situations have access to finite and often highly restricted data, so it is equally preposterous to suggest that a method is superior simply due to better asymptotic behavior in the limit of large data, or that if our approach does not work then the solution always just to get more data.[1] As such, the ease of which domain-specific expertise can be included in generative approaches is often essential to achieving effective performance on real-world tasks.

The freedom to choose the form of the generative model is both a blessing and a curse of the generative approach: it allows us to impart our own knowledge about the problem on the model, but we may be forced to make assumptions without proper justification in the interest of tractability, for convenience, in error, or simply because it is challenging to specify a sufficiently general purpose model that can cover all possible cases. Further, even after the form of the generative model has been defined, there are still decisions to be made: do we take a Bayesian or frequentist approach for making predictions? What is the best way to calculate the information required to make predictions? We will go into these questions in more depth later in the course.

As we have shown, generative approaches are inherently probabilistic. This is highly convenient when it comes to calculating uncertainty estimates or gaining insight from our trained model. They are generally more intuitive than discriminative methods, as, in essence, they constitute an explanation for how the data is generated. As such, the parameters tend to have physical interpretation in the generative process and therefore provide not only prediction, but also insight. Generative approaches will not always be preferable, particularly when there is an abundance of data available, but they provide a very powerful framework that is essential in many scenarios.

[1]It should, of course, be noted that the availability of data is typically the biggest bottleneck in machine learning: performance between machine learning approaches is often, if not usually, dominated by variations in the inherently difficulty of the problem, which is itself not usually known up front, rather than differences between approaches.

2 Review of Fundamentals

2.1 Unsupervised Learning Basics

Notational remarks

- We will typically assume that we have collected p variables (features/attributes/dimensions) on n examples (items/observations) which can be represented as an $n \times p$ *data matrix* $\mathbf{X} = (x_{ij})$, where x_{ij} is the observed value of the j-th variable for the i-th example:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1j} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2j} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{ij} & \dots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nj} & \dots & x_{np} \end{bmatrix}. \tag{2.1}$$

- We will denote the *rows* of $\mathbf{X}$ as $x_i \in \mathbb{R}^p$ and treat them as *column vectors*: i.e., the i-th item/example/observation x_i is the transpose of the i-th row of the data matrix $\mathbf{X}$:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{bmatrix} = [x_{i1}, x_{i2}, \dots, x_{ip}]^\top, \quad i = 1, \dots, n. \tag{2.2}$$

- We often assume that $x_1, \dots, x_n$ are *independent and identically distributed (iid)* samples of a *random vector* X over $\mathbb{R}^p$. When referring to the j-th dimension of random vector X, we will write $X^{(j)}$.

The goal of unsupervised learning is to extract key features of the "unlabelled" dataset. While in supervised learning our data items $\{x_i\}_{i=1}^n$ come with an extra piece of information which we are trying to predict, in unsupervised learning we are trying to understand the process which generated data $\{x_i\}_{i=1}^n$ itself.

We will here review two basic unsupervised learning tasks: *dimensionality reduction* and *clustering*. Broadly speaking, **dimensionality reduction** aims to, for each data item $x_i \in \mathbb{R}^p$, find a lower dimensional representation $z_i \in \mathbb{R}^k$ with $k \ll p$ such that the map $x \mapsto z$ preserves certain *interesting statistical properties* in data. **Clustering** on the other hand, partitions the set of n data items into K disjoint groups. We will also review a simple algorithm for each: *Principal Components Analysis (PCA)* for dimensionality reduction and the *K-means* algorithm for clustering.

2.1.1 Dimensionality Reduction with PCA

Principal Components Analysis (PCA) is a dimensionality reduction technique which aims to preserve *variance* in the data. PCA is a *linear* dimensionality reduction technique: it essentially looks for a *new basis* to represent a noisy dataset.

For simplicity, we will assume for PCA that our dataset is **centred**, i.e., that its average is $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = 0$. If not, we can always subtract it from each x_i (this is called **data centering**). Thus, we can write the **sample covariance matrix** S as

$$S = \widehat{\mathrm{Cov}}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^\top = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^\top = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}. \tag{2.3}$$

Note that the matrix S is symmetric and positive semi-definite.

PCA recovers an orthonormal basis $v_1, v_2, \dots, v_p$ in $\mathbb{R}^p$ – vectors v_i are called **principal components** (PC) or **loading vectors** – such that:

- The first principal component (PC) $v_1 \in \mathbb{R}^p$ is (a unit length vector in) the *direction of greatest variance* of data.
- The j-th PC v_j is the *direction orthogonal to* $v_1, v_2, \dots, v_{j-1}$ *of greatest variance*, for $j = 2, \dots, p$.

Given this basis, the k-dimensional representation of data item x_i is the vector of projections of x_i onto the first k PCs:

$$z_i = V_{1:k}^\top x_i = \left[v_1^\top x_i, \dots, v_k^\top x_i\right]^\top \in \mathbb{R}^k,$$

where $V_{1:k} = [v_1, \dots, v_k]$ is a $p \times k$ matrix. This gives us the *transformed data matrix*, also called the **score matrix**

$$\mathbf{Z} = \mathbf{X}V_{1:k} \in \mathbb{R}^{n\times k}. \tag{2.4}$$

Deriving the first principal component

Recall that we model our dataset as an iid sample $\{x_i\}_{i=1}^n$ of a random vector $X = \left[X^{(1)} \dots X^{(p)}\right]^\top$. Projections to PCs define a linear transformation of X given by a random vector $Z = V_{1:k}^\top X$ which is a k-dimensional random vector. Dimensions of Z are called **derived variables**. Consider the first dimension of Z:

$$Z^{(1)} = v_1^\top X = v_{11}X^{(1)} + v_{12}X^{(2)} + \cdots + v_{1p}X^{(p)}. \tag{2.5}$$

The first PC $v_1 = [v_{11}, \dots, v_{1p}]^\top \in \mathbb{R}^p$ is chosen to maximise the sample variance $\widehat{\mathrm{Var}}(Z^{(1)}) = v_1^\top \widehat{\mathrm{Cov}}(X) v_1$, i.e. it is defined as the solution to

$$\max_{v_1} \; v_1^\top S v_1$$
$$\text{subject to: } v_1^\top v_1 = 1.$$

By considering the Lagrangian:

$$\mathcal{L}(v_1, \lambda_1) = v_1^\top S v_1 - \lambda_1 \left(v_1^\top v_1 - 1 \right) \tag{2.6}$$

and the corresponding vector of partial derivatives

$$\frac{\partial \mathcal{L}(v_1, \lambda_1)}{\partial v_1} = 2 S v_1 - 2 \lambda_1 v_1 \tag{2.7}$$

we obtain the eigenvector equation $S v_1 = \lambda_1 v_1$, i.e. v_1 must be an eigenvector of S and the dual variable λ_1 is the corresponding eigenvalue. Since $v_1^\top S v_1 = \lambda_1 v_1^\top v_1 = \lambda_1$, the first PC must be the eigenvector associated with the *largest eigenvalue* of S.

Similarly, we can show that each subsequent PC is given by the eigenvector corresponding to the next largest eigenvalue (problem sheet shows this for the second PC). As a result, we obtain the **eigenvalue decomposition** of S given by

$$S = V \Lambda V^\top \tag{2.8}$$

where Λ is a diagonal matrix with eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0 \tag{2.9}$$

on the diagonal and V is a $p \times p$ orthogonal matrix (i.e. $VV^\top = V^\top V = I$) whose *columns* are the p eigenvectors of S, i.e. the principal components $v_1, \dots, v_p$.

In summary,

- Derived scalar variable (projection to the j-th principal component) $Z^{(j)} = v_j^\top X$ has sample variance λ_j, for $j = 1, \dots, p$.
- Derived variables are *uncorrelated*: $\mathrm{Cov}(Z^{(i)}, Z^{(j)}) \approx v_i^\top S v_j = \lambda_j v_i^\top v_j = 0$, for $i \neq j$.
- The **total sample variance** is given by $\mathrm{Tr}(S) = \sum_{i=1}^{p} S_{ii} = \lambda_1 + \ldots + \lambda_p$, so the **proportion of total variance** explained by the j^{th} PC is $\frac{\lambda_j}{\lambda_1 + \lambda_2 + \ldots + \lambda_p}$

Reconstruction view of PCA

We can map back to the original p-dimensional space using

$$\hat{x}_i = V_{1:k} V_{1:k}^\top x_i. \tag{2.10}$$

This is a **reconstruction** of data item x_i. It can be shown (exercise) that PCA gives the *optimal linear reconstruction* based on a k-dimensional compression.

PCA via the Singular Value Decomposition

PCA can also be understood using the **singular value decomposition** (SVD) of data matrix $\mathbf{X}$. Recall that any real-valued $n \times p$ matrix $\mathbf{X}$ can be written as $\mathbf{X} = UDV^\top$ where

- U is an $n \times n$ orthogonal matrix: $UU^\top = U^\top U = I_n$.
- D is a $n \times p$ matrix with decreasing *non-negative* elements on the diagonal (the singular values of $\mathbf{X}$) and zero off-diagonal elements.
- V is a $p \times p$ orthogonal matrix: $VV^\top = V^\top V = I_p$.

Note that

$$(n-1)S = \mathbf{X}^\top \mathbf{X} = (UDV^\top)^\top (UDV^\top) = VD^\top U^\top UDV^\top = VD^\top DV^\top,$$

using orthogonality of U. The eigenvalues of S are thus the diagonal entries of $\Lambda = \frac{1}{n-1} D^\top D$.

We also have

$$\mathbf{X}\mathbf{X}^\top = (UDV^\top)(UDV^\top)^\top = UDV^\top VD^\top U^\top = UDD^\top U^\top,$$

using orthogonality of V.

The $n \times n$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$ with entries $\mathbf{K}_{ij} = x_i^\top x_j$ is called the **Gram matrix** of dataset $\mathbf{X}$. Note that $\mathbf{K}$ and $(n-1)S = \mathbf{X}^\top\mathbf{X}$ have the same nonzero eigenvalues, equal to the non-zero squared singular values of $\mathbf{X}$ (non-zero entries on the diagonals of $D^\top D$ and $DD^\top$).

If we consider projections to *all principal components*, the transformed data matrix is

$$\mathbf{Z} = \mathbf{X}V = UDV^\top V = UD, \tag{2.11}$$

If $p \leq n$ this means

$$z_i = \left[U_{i1}D_{11}, \ldots, U_{ip}D_{pp}\right]^\top, \tag{2.12}$$

and if $p > n$ only the first n projections are defined (sample covariance will be at most rank n):

$$z_i = \left[U_{i1}D_{11}, \ldots, U_{in}D_{nn}, 0, \ldots, 0\right]^\top. \tag{2.13}$$

Thus, $\mathbf{Z}$ can be obtained from the eigendecomposition of Gram matrix $\mathbf{K}$. When $p \gg n$, eigendecomposition of $\mathbf{K}$ requires much less computation, $O(n^3)$, than the eigendecomposition of the covariance matrix, $O(p^3)$, so is the preferred method for PCA in that case.

2.1.2 Clustering

Clustering is one of the fundamental and ubiquitous tasks in exploratory data analysis – a first intuition about the data is often based on identifying meaningful disjoint groups among the data items. In *partition-based* clustering, which we consider in this note, one divides n data items into K clusters $C_1, \ldots, C_K$ where for all $k, k' \in \{1, \ldots, K\}$,

$$C_k \subset \{1, \ldots, n\}, \qquad C_k \cap C_{k'} = \emptyset \;\; \forall k \neq k', \qquad \bigcup_{k=1}^{K} C_k = \{1, \ldots, n\}.$$

Central to the goals of clustering is the notion of similarity/dissimilarity between data items. There will be many ways to define the notion of similarity, and the choice will depend on the dataset being analyzed and dictated by domain specific knowledge.

Intuitively, clustering aims to group similar items together and to place separate dissimilar items into different groups. However, note that these two objectives in many cases contradict each other (similarity is not a transitive relation, while being in the same cluster is an equivalence relation). One could imagine a long sequence of items such that each next item is very similar to the previous one so that they should all belong to the same cluster – but that would also mean that the endpoints are potentially highly dissimilar. Hence, there are also different clustering techniques which emphasize different aspects of these goals, i.e. whether to keep similar points together or dissimilar points apart.

Lack of Axiomatic Definition

There have been several attempts to construct an axiomatic definition of clustering, but it is surprisingly difficult to put on rigorous footing. Consider the following three basic properties required of a clustering method $\mathcal{F} : (\mathcal{D} = \{x_i\}_{i=1}^n, \rho) \mapsto \{C_1, \ldots, C_K\}$ which takes as an input dataset $\mathcal{D}$ and a dissimilarity function ρ and returns a partition of $\mathcal{D}$:

- **Scale invariance.** For any $\alpha > 0$, $\mathcal{F}(\mathcal{D}, \alpha\rho) = \mathcal{F}(\mathcal{D}, \rho)$, i.e. partition should not depend on units in which dissimilarity is measured.

- **Richness.** For any partition $C = \{C_1, \ldots, C_K\}$ of $\mathcal{D}$, there exists dissimilarity ρ, such that $\mathcal{F}(\mathcal{D}, \rho) = C$, i.e. the outcome is fully controlled by the dissimilarity function.

- **Consistency.** If ρ and ρ' are two dissimilarities such that for all $x_i, x_j \in \mathcal{D}$ the following holds:

$$\begin{aligned} x_i, x_j \text{ belong to the same cluster in } \mathcal{F}(\mathcal{D}, \rho) &\implies \rho'(x_i, x_j) \leq \rho(x_i, x_j) \\ x_i, x_j \text{ belong to different clusters in } \mathcal{F}(\mathcal{D}, \rho) &\implies \rho'(x_i, x_j) \geq \rho(x_i, x_j), \end{aligned}$$

  then $\mathcal{F}(\mathcal{D}, \rho') = \mathcal{F}(\mathcal{D}, \rho)$. In other words, if the items in the same cluster become more similar and the items already separated become less similar, then the clustering should not change.

While all three properties appear natural, and there are algorithms satisfying any two of them, Kleinberg's impossibility theorem [31] states that there exists no clustering method that satisfies all three properties, implying that every clustering method will have some undesirable properties. For further discussion, see Section 22.5 in [51].

We will consider here the simplest widely used clustering method: K-means algorithm and two extensions.

K-means algorithm

K-means is the simplest partition-based clustering algorithm. It uses a preassigned number of clusters and represents each cluster using a **prototype** or **cluster centroid** μ_k.

The idea of K-means is to measure the quality of each cluster C_k using its **within-cluster deviance** from the cluster centroids

$$W(C_k, \mu_k) = \sum_{i \in C_k} \|x_i - \mu_k\|_2^2.$$

The overall quality of the clustering is then given by the total within-cluster deviance:

$$W(\{C_k\}, \{\mu_k\}) = \sum_{k=1}^{K} W(C_k, \mu_k) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|_2^2 = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|_2^2,$$

where $c_i = k$ if and only if $i \in C_k$. This is now the overall objective function used to select both the cluster centroids and the assignment of points to clusters. The joint optimization over both the partition $\{C_k\}$ and centroids $\{\mu_k\}$ is a combinatorial optimization problem and is computationally hard. However, note that

- Given partition $\{C_k\}$, we can easily find the optimal centroids by differentiating W with respect to μ_k:

$$\frac{\partial W}{\partial \mu_k} = 2 \sum_{i \in C_k} (x_i - \mu_k) = 0 \qquad \Rightarrow \mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

- Given prototypes, we can easily find the optimal partition by assigning each data point to the closest cluster prototype:

$$c_i = \operatorname{argmin}_k \|x_i - \mu_k\|_2^2 .$$

Thus one can employ an iterative alternating optimization, which is exactly the K-means algorithm given in Algorithm 2.1.

K-means is a heuristic search algorithm so it can (and often will) get stuck at local optima. The result depends on the starting configurations. Typically one performs a number of runs from different random initial values of centroids, and then chooses the end result with minimum W. Since each step does not increase the objective function and the number of possible partitions is finite, the algorithm will converge to a local optimum. However, note that there could be ties in the cluster assignment, which need to be broken in a systematic fashion.

Algorithm 2.1 K-means algorithm

```
Input: dataset $\mathcal{D} = \{x_i\}_{i=1}^n$, desired number of clusters $K$
Output: partition $\{C_1, \dots, C_K\}$

Randomly initialize $K$ cluster centroids $\mu_1, \dots, \mu_K$.
while the partition has not converged do
    • Cluster assignment: For each $i = 1, \dots, n$, assign each $x_i$ to the cluster with the
      nearest centroid,
          $c_i := \operatorname{argmin}_{k=1,\dots,K} \|x_i - \mu_k\|_2^2$
      Set $C_k := \{i : c_i = k\}$ for each $k$.
    • Move centroids: Set $\mu_1, \dots, \mu_K$ to the averages of the new clusters:
          $\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x_i$
return partition $\{C_1, \dots, C_K\}$
```

K-means++

A simple yet provably effective solution to the problem of initialization of centroids in the K-means algorithm was proposed by [2]. The method starts with sampling a data item from $\mathcal{D} = \{x_i\}_{i=1}^n$ uniformly at random and making it centroid μ_1. We then compute the squared distances $\rho_i^2 = \|x_i - \mu_1\|_2^2$. Centroid μ_2 is then initialized to another data item sampled using the probability mass function $p(i) = \rho_i^2 / \sum_{j=1}^n \rho_j^2$ and the process continues with the probability mass function being updated at each step, i.e. to initialize k-th centroid μ_k, we compute

$$\rho_i^2 = \min\left\{\|x_i - \mu_1\|_2^2, \dots, \|x_i - \mu_{k-1}\|_2^2\right\}. \tag{2.14}$$

Remarkably, this method comes with a precise theoretical guarantee. In particular, [2] show that if partition $\{C_k^{++}\}$ is obtained using K-means++ then

$$\mathbb{E}\left[W\left(\{C_k^{++}\}\right)\right] \leq 8\left(\log K + 2\right) W^*, \tag{2.15}$$

where W^* is the within-cluster deviance of the globally optimal clustering and the expectation is taken over the random sampling used in the initialisation.

DP-means

K-means is intuitive and straightforward to implement, but how do we select the number of clusters K in the first place? Clearly, the objective function is minimized (and equals zero) if we let $K = n$, but this is not a meaningful clustering.

One elegant approach is the *DP-means algorithm* [33] that comes from the interpretation of K-means using small variance asymptotics of the Expectation Maximization (EM) algorithm for mixture modelling (see Chapter 9). We will discuss mixture modelling and EM algorithm later in the course. DP-means starts from a single cluster, i.e. $K = 1$ and modifies the cluster assignment step as follows:

1. Initialize $K = 1$ and $\mu_1 = \frac{1}{n}\sum_{i=1}^{n} x_i$ (the global mean).
2. *DP-means cluster assignment:* For each $i = 1, \dots, n$,
   - if $\min_{k=1,\dots,K} \|x_i - \mu_k\|_2^2 > \lambda$, set $K \leftarrow K + 1$, $c_i \leftarrow K$, $\mu_K \leftarrow x_i$
   - otherwise, set $c_i = \operatorname{argmin}_{k=1,\dots,K} \|x_i - \mu_k\|_2^2$, and update the previous and current cluster centroids that item i belongs to.
3. Repeat Step 2 until it stops changing the cluster assignments for all items.

The tuning parameter λ controls the tradeoff between the traditional K-means objective and the number of clusters, and can be interpreted as a cost associated with each cluster used. Like K-means, DP-means can be shown to terminate after a finite number of iterations (problem sheet).

2.2 Supervised Learning Basics

2.2.1 Empirical Risk Minimisation (ERM)

Loss and Risk

In **supervised learning**, we are trying to learn a function $f : \mathcal{X} \to \mathcal{Y}$ from an input space $\mathcal{X}$ into an output space $\mathcal{Y}$ based on a set of paired examples $(x_1, y_1), \dots (x_n, y_n)$ and a given **loss function** L. It is typically assumed that examples $(x_1, y_1), \dots (x_n, y_n)$ are i.i.d. samples from an *unknown joint probability distribution* $P_{X,Y}$ on $\mathcal{X} \times \mathcal{Y}$. The goal of supervised learning is to find the function f which *minimizes the expectation of the loss* over $P_{X,Y}$, which is called *risk*. In machine learning, if the output space is discrete, this is called *classification*, while *regression* refers to the case the output space is continuous.

Empirical Risk Minimisation (ERM)

A **loss function** is any function

$$L : \mathcal{Y} \times \mathcal{Y} \times \mathcal{X} \to \mathbb{R}^{+}. \tag{2.16}$$

The loss $L(y_i, f(x_i), x_i)$ measure the discrepancy between predicted output values $f(x_i)$ and the true output values y_i at the inputs x_i.

The **risk** of a function $f : \mathcal{X} \to \mathcal{Y}$ is the expected loss:

$$R(f) = \mathbb{E}_{X,Y} L(Y, f(X), X). \tag{2.17}$$

For a given dataset $(x_1, y_1), \dots (x_n, y_n)$, the **empirical risk** of f is given by

$$\hat{R}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i), x_i). \tag{2.18}$$

The **empirical risk minimisation** (ERM) problem aims to find the function with minimum risk:

$$\hat{f} = \text{argmin}_{f \in \mathcal{H}} \hat{R}(f), \tag{2.19}$$

where $\mathcal{H}$ is a given class of functions (called the **hypothesis class**).

Remark 1. The ultimate goal of learning is to minimise the true risk - *not* the empirical risk, which is only an estimate of the true risk. But the true risk of any given function is unknown because the distribution $P_{X,Y}$ is unknown.

Remark 2. Loss functions typically depend on the input x only through $f(x)$, so that with some abuse of notation we often write $L(y, f(x))$ instead of $L(y, f(x), x)$. $L(y, f(x))$ is usually some notion of distance between the true output y and the predicted output $f(x)$.

Examples of hypothesis classes.

Hypothesis classes can be very simple, e.g. for $\mathcal{X} = \mathbb{R}^p$, we can consider all linear functions $f(x) = w^\top x + b$, parametrized by $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$, or we could consider a specific **nonlinear feature expansion** $\varphi : \mathcal{X} \to \mathbb{R}^D$, and a model linear in those features: $f(x) = w^\top \varphi(x) + b$, but nonlinear in the original inputs $\mathcal{X}$, parametrized by $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$. For example, starting with $\mathcal{X} = \mathbb{R}^2$, we can consider $\varphi\left(\begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix}\right) = [x_{i1}, x_{i2}, x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2]^\top$, such that the resulting function can depend on quadratic and interaction terms as well. An important type of hypothesis class we will consider in this course are *Reproducing Kernel Hilbert Spaces (RKHS)*, which are also linear in certain feature expansions but those feature expansions could potentially be infinite-dimensional; see Chapter 4.

Examples of loss functions.

Loss functions come in many different forms. One of the main considerations for selecting loss functions is the type of outputs we are trying to predict, i.e., whether it is real-valued or discrete/categorical. Note that even if outputs are discrete, the function $f(x)$ we are trying to learn is typically real-valued. For example, in binary classification, the common convention is that the two classes are denoted by -1 and $+1$. One associates predictions of these classes with $\text{sign}(f(x))$, whereas the magnitude of $f(x)$ can be thought of as the confidence in those predictions (not necessarily in a probabilistic sense). The loss can penalize misclassification (wrong sign) as well as the overconfident misclassification (wrong sign and large magnitude) and even underconfident correct classification (correct sign but small magnitude). Thus, the loss functions can often be expressed as a function of $yf(x)$.

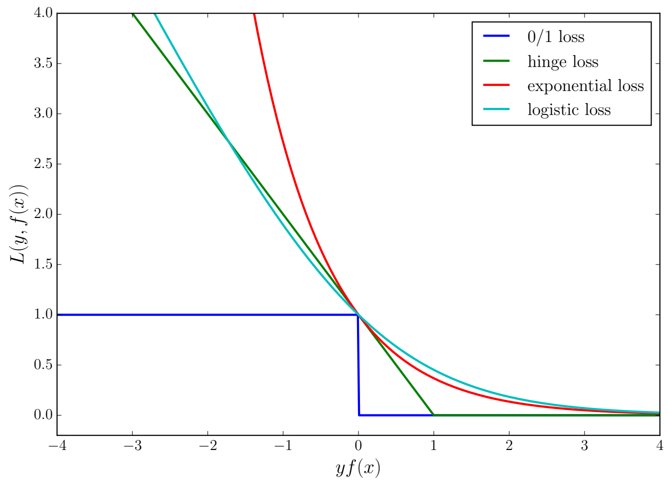


Figure 2.1: Loss functions for binary classification

Below are some loss functions commonly used in binary classification and regression.

- Binary classification:
  - 0/1 *loss*: $L(y, f(x)) = \mathbb{I}\{yf(x) \leq 0\}$,
    (also called **misclassification loss**. The optimal solution is called the **Bayes classifier** and is given by $f(x) = \operatorname{argmax}_{k\in\{0,1\}} \mathbb{P}(Y = k|X = x)$),
  - **hinge loss**: $L(y, f(x)) = (1 - yf(x))_+$
    (used in *support vector machines* - leads to sparse solutions),
  - **exponential loss**: $L(y, f(x)) = e^{-yf(x)}$
    (used in *boosting* algorithms, e.g. Adaboost),
  - **logistic loss**: $L(y, f(x)) = \log\left(1 + e^{-yf(x)}\right)$
    (used in *logistic regression*, and associated with a linear log-odds probabilistic model).
- Regression:
  - **squared loss**: $L(y, f(x)) = (y - f(x))^2$
    (**least squares regression**: optimal f is the conditional mean $\mathbb{E}[Y|X = x]$),
  - **absolute loss**: $L(y, f(x)) = |y - f(x)|$
    (**least absolute deviations regression**, which is less sensitive to outliers: optimal f is the conditional median $\mathrm{med}[Y|X = x]$),

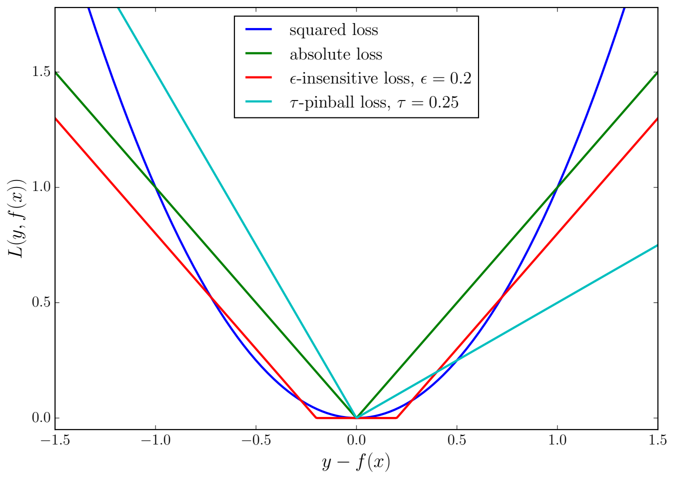


Figure 2.2: Loss functions for regression

– **τ-pinball loss**: $L(y, f(x)) = 2\max\{\tau(y - f(x)), (\tau - 1)(y - f(x))\}$ for $\tau \in (0, 1)$
(**quantile regression**: optimal f is the τ-quantile of $p(y|X = x)$),

– **ϵ-insensitive loss** or **Vapnik loss**: $L(y, f(x)) = \begin{cases} 0, \text{ if } |y - f(x)| \le \epsilon, \\ |y - f(x)| - \epsilon, \text{ otherwise.} \end{cases}$
(**support vector regression**, which leads to sparse solutions).

In binary classification, 0/1 is an idealised version of loss which penalizes misclassification regardless of the magnitude of $f(x)$. However, ERM under 0/1 loss is NP hard[1]. Therefore, we typically use *convex upper bound surrogate losses* (hinge, exponential, logistic[2]). What is the importance of the convexity of loss as a function of $yf(x)$ as shown in Fig. 2.1? Consider the hypothesis class $f(x) = w^\top \varphi(x)$, with $w \in \mathbb{R}^D$ (we ignore the intercept to simplify notation) and assume that $L(y, f(x)) = \rho\,(yf(x))$ for a convex differentiable function ρ. Then the empirical risk and its gradient are given by

$$\hat{R}(w) = \frac{1}{n}\sum_{i=1}^{n} \rho\left(y_i w^\top \varphi(x_i)\right), \quad \frac{\partial \hat{R}}{\partial w} = \frac{1}{n}\sum_{i=1}^{n} \rho'\left(y_i w^\top \varphi(x_i)\right) y_i \varphi(x_i).$$

Furthermore, the Hessian matrix of the empirical risk is given by

$$\frac{\partial^2 \hat{R}}{\partial w \partial w^\top} = \frac{1}{n}\sum_{i=1}^{n} \rho''\left(y_i w^\top \varphi(x_i)\right) \varphi(x_i)\varphi(x_i)^\top, \tag{2.20}$$

[1]It is NP-hard to even approximately minimize the ERM under 0/1 loss - i.e. there is no known polynomial-time algorithm to obtain a solution which is a small constant worse than the optimum.
[2]to make it into an upper bound on 0/1, divide the logistic loss by log(2) - rescaling of the loss does not change the ERM problem

using $y_i^2 = 1$. This Hessian is now a positive semidefinite matrix which can be seen from $\rho''(t) \geq 0 \; \forall t$ and

$$\alpha^\top \frac{\partial^2 \hat{R}}{\partial w \partial w^\top} \alpha \;=\; \frac{1}{n} \sum_{i=1}^{n} \rho'' \left(y_i w^\top \varphi(x_i) \right) \left(\alpha^\top \varphi(x_i) \right)^2 \geq 0.$$

for any $\alpha \in \mathbb{R}^D$. Thus, empirical risk is a convex function of w and thus has a *unique minimum.* Typically, there is no closed form solution for w and iterative optimisation techniques like *gradient ascent* or *Newton-Raphson algorithm* are used.

2.2.2 Regularisation

Recall that we are not ultimately interested in the exact minimizer of the *empirical risk* but in that of the *true risk.* ERM thus risks **overfitting**: when the hypothesis class is complex, one can easily find a function that matches the observed examples exactly but does not *generalise* to other examples drawn from $P_{X,Y}$.

The idea behind **regularisation** is to limit the flexibility of the hypothesis class in order to prevent overfitting. For example, for the hypothesis space $\mathcal{H} = \{f_\theta : \theta \in \mathbb{R}^p\}$ consisting of functions parameterised by a vector θ, this can be achieved by adding a term which *penalises large values for the parameters* θ to the ERM criterion:

$$\min_\theta \hat{R}(f_\theta) + \lambda \|\theta\|_\rho^\rho = \min_\theta \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda \|\theta\|_\rho^\rho$$

where $\rho \geq 1$, and $\|\theta\|_\rho = (\sum_{j=1}^{p} |\theta_j|^\rho)^{1/\rho}$ is the L_ρ norm of θ (also of interest when $\rho \in [0, 1)$, but this is no longer a norm). These methods are also known as **shrinkage** methods since their effect is to shrink parameter estimates towards 0. Note that we have an additional **tuning parameter** (or **hyperparameter**) λ which controls the amount of regularisation, and, as a result, also controls the complexity of the model.

The most common forms of regularisation include **ridge regression** / **Tikhonov regularization**: $\rho = 2$, **LASSO** penalty: $\rho = 1$, and **elastic net** regularization with a mixed L_1/L_2 penalty:

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_\theta(x_i)) + \lambda \left[(1-\alpha)\|\theta\|_2^2 + \alpha \|\theta\|_1 \right].$$

In some hypothesis classes, it is possible to directly penalise some notion of *smoothness* of the function f we are trying to learn, e.g. for $\mathcal{X} = \mathbb{R}$, the regularisation term can consist of the **Sobolev norm**

$$\|f\|_{W^1}^2 = \int_{-\infty}^{+\infty} f(x)^2 dx + \int_{-\infty}^{+\infty} f'(x)^2 dx, \tag{2.21}$$

which penalises functions with large derivative values.

2.2.3 Examples of ERM

Regularised Least Squares / Ridge Regression

This corresponds to the squared loss $L(y, f(x)) = (y - f(x))^2$. For linear functions $f(x) = w^\top x + b$, we have

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^\top x_i - b)^2 + \frac{\lambda}{n} \|w\|_2^2. \tag{2.22}$$

Note the rescaling of the regularisation term and that the bias term b is not included in the regularisation. This is important as otherwise the predictions would depend on the origin for the response variables y (i.e. adding a constant c to each target would result in different predictions from simply shifting the original predictions by c). Fortunately, when using centred inputs, i.e., $\sum_{i=1}^{n} x_i = 0$, b can be estimated by $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$, so we can also assume that the responses are centred and remove the intercept from the model. We obtain the problem

$$\min_{w} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2. \tag{2.23}$$

Differentiating and setting to zero gives the closed form solution

$$w = \left(\mathbf{X}^\top \mathbf{X} + \lambda I\right)^{-1} \mathbf{X}^\top \mathbf{y}. \tag{2.24}$$

Logistic Regression

Despite the name, *logistic regression is a method for classification.* It uses the logistic loss $L(y, f(x)) = \log\left(1 + e^{-yf(x)}\right)$. Hence, again for a linear classifier $f(x) = w^\top x + b$,

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(w^\top x_i + b)}\right) + \frac{\lambda}{n} \|w\|_2^2. \tag{2.25}$$

Logistic regression can also be associated to a probabilistic model. Namely, assume that the function of interest $f(x) = w^\top x + b$ models the log-odds ratio:

$$\log \frac{p(y_i = +1 | w, b, x_i)}{p(y_i = -1 | w, b, x_i)} = w^\top x_i + b. \tag{2.26}$$

Then the conditional distribution of $Y|X$ is given by

$$p(y_i = +1 | w, b, x_i) = \frac{1}{1 + e^{-(w^\top x_i + b)}} = \sigma(w^\top x_i + b), \tag{2.27}$$

$$p(y_i = -1 | w, b, x_i) = \frac{1}{1 + e^{w^\top x_i + b}} = \sigma(-w^\top x_i - b), \tag{2.28}$$

where we denoted by $\sigma(t) = 1/(1 + e^{-t})$ the **logistic function** (aka **sigmoid function**) which maps the real line to the $(0, 1)$ interval. Note that the logistic function satisfies

$\sigma(-t) = 1-\sigma(t)$. Thus, we can write (2.27) and (2.28) as $p(y_i|w,b,x_i) = \sigma(y_i(w^\top x_i+b))$ and the conditional log-likelihood of the outputs given the inputs is

$$\log p(\mathbf{y}|w,b,\mathbf{X}) = \log \prod_{i=1}^{n} \sigma(y_i(w^\top x_i+b)) = -\sum_{i=1}^{n} \log\left(1+e^{-y_i(w^\top x_i+b)}\right).$$

Thus finding the parameters w and b that maximise the conditional log-likelihood is equivalent to minimising the empirical risk corresponding to the logistic loss, which is the negative log-likelihood of the linear log-odds model. Moreover, the regularisation term can be interpreted as a normal prior on w in *Bayesian logistic regression.* Again, there is no closed form solution for logistic regression, but the objective is convex and differentiable and the numerical optimisation via gradient ascent or Newton-Raphson algorithm can be used.

The connection between maximisation of the log-likelihood and minimisation of the empirical risk extends beyond logistic regression. Indeed, in the context of classification, whenever $p(y_i|x_i,\theta)$ is a log-concave function of $y_i f_\theta(x_i)$, we can define a convex loss $\rho(yf_\theta(x)) = -\log p(y_i|x_i,\theta)$. But the converse is not true, e.g. hinge loss used in the SVMs below does not correspond to a negative log-likelihood in any probabilistic model (unless additional artificial classes are introduced).

Support Vector Machines

Support Vector Machines (SVMs) for classification use hinge loss, $L(y,f(x)) = \max\{0, 1-yf(x)\}$. Thus, for a linear classifier $f(x) = w^\top x + b$, we obtain

$$\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1-y_i(w^\top x_i+b)\} + \frac{\lambda}{n}\|w\|_2^2. \tag{2.29}$$

This does not have a closed form solution and requires numerical optimisation.

Eq. (2.29) is not how you would typically see an SVM written in the literature, though. In the next chapter, we will make a deep dive into SVMs, introducing it from the perspective of *maximum margin classification.*

3 Support Vector Machines

These notes are a revised version of lecture notes from the UCL course “Reproducing Kernel Hilbert Spaces in Machine Learning” [21], reproduced here by courtesy of Arthur Gretton.

3.1 Duality in Convex Optimization

We will need some basic results from **duality** in **convex optimization** in order to study support vector machines, one of the fundamental techniques for classification. This review covers the material from [9, Sections 5.1-5.5].

3.1.1 The Lagrangian

Consider a **constrained optimization** problem of an objective function $f_0 : \mathbb{R}^n \to \mathbb{R}$, with m inequality and r equality constraints:

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0 & i = 1, \ldots, m \\ & h_j(x) = 0 & j = 1, \ldots r. \end{array} \tag{3.1}$$

We denote $\mathcal{D} := \bigcap_{i=0}^{m} \mathrm{dom} f_i \cap \bigcap_{j=1}^{r} \mathrm{dom} h_j$, and require the domain $\mathcal{D} \subseteq \mathbb{R}^n$ where the objective function f_0 and the constraint functions $f_1, \ldots, f_m, h_1, \ldots, h_r$ are all defined to be nonempty. We will refer to (3.1) as the **primal problem** and denote by $p^* = f_0(x^*)$ its optimal value. Any point $\tilde{x} \in \mathcal{D}$ for which constraints are satisfied , i.e. $f_i(\tilde{x}) \leq 0$, $h_j(\tilde{x}) = 0$, is called a **primal feasible** point.

The **Lagrangian** $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^r \to \mathbb{R}$ associated with problem (3.1) is given by

$$L(x, \lambda, \nu) := f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{r} \nu_j h_j(x).$$

The vectors $\lambda \in \mathbb{R}^m$ and $\nu \in \mathbb{R}^r$ are called **Lagrange multipliers** or **dual variables**. The **Lagrange dual function** (or just “dual function”) is written

$$g(\lambda, \nu) = \inf_{x \in \mathcal{D}} L(x, \lambda, \nu).$$

The domain of g, dom$g$, is the set of values (λ, μ) for which the Lagrangian is bounded from below, i.e. $g > -\infty$. The dual function is a pointwise infimum of **affine**[1] functions of (λ, ν), hence it is concave in (λ, ν) [9, p. 83]. A **dual feasible** pair (λ, ν) is a pair for which $\lambda \succeq 0$ and $(\lambda, \nu) \in \mathrm{dom} g$.

[1]A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine if it takes the form $f(x) = Ax + b$.

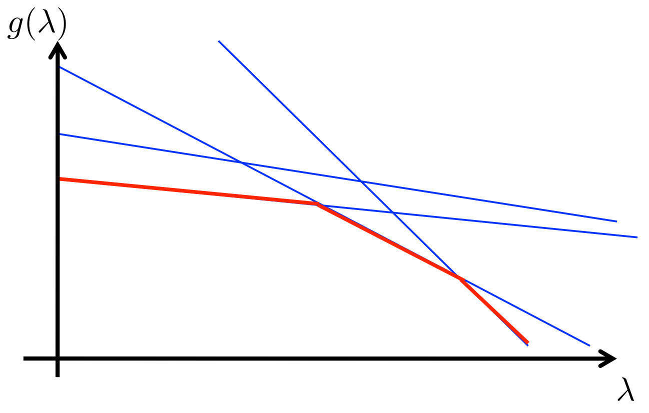


Figure 3.1: Example: Lagrangian with one inequality constraint, $L(x,\lambda) = f_0(x) + \lambda f_1(x)$, where x here can take one of four values for ease of illustration. The infimum of the resulting set of four affine functions is concave in λ.

Proposition 3. *When $\lambda \succeq 0$, then for all ν we have*

$$g(\lambda,\nu) \leq p^*. \tag{3.2}$$

Proof. Assume $\tilde{x} \in \mathcal{D}$ is feasible, i.e. $f_i(\tilde{x}) \leq 0$, $h_j(\tilde{x}) = 0$, and assume $\lambda \succeq 0$. Then

$$\sum_{i=1}^{m} \lambda_i f_i(\tilde{x}) + \sum_{j=1}^{r} \nu_j h_j(\tilde{x}) \leq 0$$

and so

$$\begin{aligned} g(\lambda,\nu) &:= \inf_{x\in\mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{r} \nu_j h_j(x) \right) \\ &\leq f_0(\tilde{x}) + \sum_{i=1}^{m} \lambda_i f_i(\tilde{x}) + \sum_{j=1}^{r} \nu_j h_j(\tilde{x}) \\ &\leq f_0(\tilde{x}). \end{aligned}$$

This holds for every feasible $\tilde{x}$, and thus also for x^*, hence (3.2) holds. □

The Lagrangian can be interpreted as a lower bound on the original optimization problem. Ideally we would write the problem (3.1) as the unconstrained problem

$$\text{minimize } f_0(x) + \sum_{i=1}^{m} I_{-}\left(f_i(x)\right) + \sum_{j=1}^{r} I_0\left(h_j(x)\right),$$

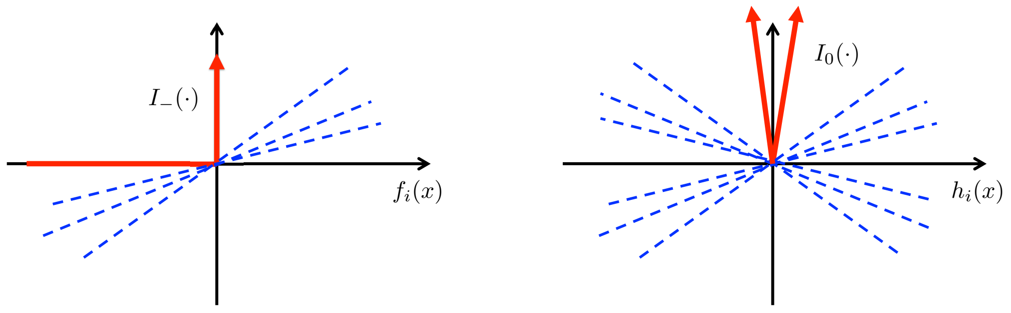


Figure 3.2: Linear lower bounds on indicator functions. Blue functions represent linear lower bounds for different slopes λ and ν, for the inequality and equality constraints, respectively.

where

$$I_-(u) = \begin{cases} 0 & u \leq 0 \\ \infty & u > 0, \end{cases} \qquad I_0(u) = \begin{cases} 0 & u = 0 \\ \infty & u \neq 0, \end{cases}$$

i.e. giving an infinite penalty when any constraint is violated. Instead of these infinite penalty constraints (which are hard to optimize), we replace the constraints with a set of soft linear constraints, as shown in Fig. 3.2. It is now clear why λ must be positive for the inequality constraint: a negative λ would not yield a lower bound. Note also that as well as being penalized for $f_i > 0$, the linear lower bounds reward us for achieving $f_i < 0$. This is illustrated in Fig. 3.3.

3.1.2 The dual problem

The dual problem attempts to find the best lower bound $g(\lambda, \nu)$ on the optimal solution p^* of (3.1). This results in the **dual problem**

$$\begin{array}{ll} \text{maximize} & g(\lambda, \nu) \\ \text{subject to} & \lambda \succeq 0. \end{array} \tag{3.3}$$

Denote by (λ^*, ν^*) the arguments optimizing (3.3) and by d^* the optimal value of the dual problem. Note that (3.3) is always a convex optimization problem, since the function being maximized is concave and the constraint set is convex. The property of **weak duality** is immediate:

$$d^* \leq p^*.$$

The difference $p^* - d^*$ is called the **optimal duality gap**. If the duality gap is zero, then **strong duality** holds:

$$d^* = p^*.$$

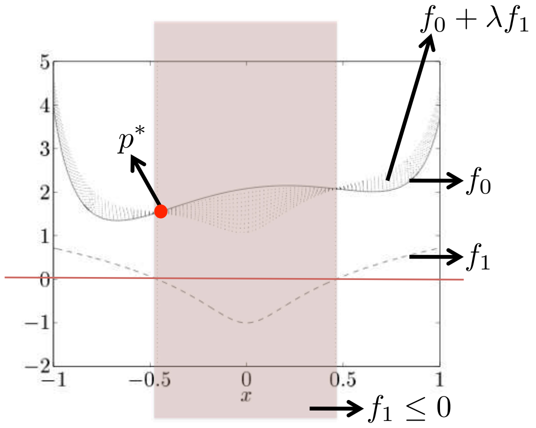


Figure 3.3: Illustration of the Lagrangian on a simple problem with one inequality constraint (from [9, Fig. 5.1]).

Conditions under which strong duality holds are called **constraint qualifications**. As an important case: strong duality holds if the primal problem is convex,[2] i.e. of the form

$$\begin{array}{rll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 0 & i = 1, \ldots, n \\ & Ax = b & \end{array} \tag{3.4}$$

for convex $f_0, \ldots, f_m$, *and* if **Slater's condition** holds: there exists some *strictly* feasible point[3] $\tilde{x} \in \operatorname{relint}(\mathcal{D})$ such that

$$f_i(\tilde{x}) < 0 \quad i = 1, \ldots, m \quad A\tilde{x} = b.$$

A weaker version of Slater's condition is sufficient for strong convexity when some of the constraint functions $f_1, \ldots, f_k$ are affine (note the inequality constraints are no longer strict):

$$f_i(\tilde{x}) \leq 0 \quad i = 1, \ldots, k \quad f_i(\tilde{x}) < 0 \quad i = k+1, \ldots, m \quad A\tilde{x} = b.$$

A proof of this result is given in [9, Section 5.3.2].

[2]Strong duality can also hold for non-convex problems: see e.g. [9, p. 229].

[3]We denote by $\operatorname{relint}(\mathcal{D})$ the relative interior of the set $\mathcal{D}$. This looks like the interior of the set, but is non-empty even when the set is a subspace of a larger space. See [9, Section 2.1.3] for the formal defintion.

3.1.3 A saddlepoint/game characterization of weak and strong duality

In this section, we ignore equality constraints for ease of discussion. We write the solution to the primal problem as an optimization

$$\begin{aligned}\sup_{\lambda\succeq 0} L(x,\lambda) &= \sup_{\lambda\succeq 0}\left(f_0(x)+\sum_{i=1}^m \lambda_i f_i(x)\right)\\ &= \begin{cases} f_0(x) & \text{if } f_i(x)\le 0,\ i=1,\ldots,m\\ \infty & \text{otherwise.}\end{cases}\end{aligned}$$

In other words, we recover the primal problem when the inequality constraint holds, and get infinity otherwise. We can therefore write

$$p^* = \inf_x \sup_{\lambda\succeq 0} L(x,\lambda).$$

We already know

$$d^* = \sup_{\lambda\succeq 0} \inf_x L(x,\lambda).$$

Weak duality therefore corresponds to the **max-min inequality**:

$$\sup_{\lambda\succeq 0}\inf_x L(x,\lambda) \le \inf_x \sup_{\lambda\succeq 0} L(x,\lambda). \tag{3.5}$$

which holds for general functions, and not just $L(x,\lambda)$. Strong duality occurs at a saddlepoint, and the inequality becomes an equality.

There is also a game interpretation: $L(x,\lambda)$ is a sum that must be paid by the person adjusting x to the person adjusting λ. On the right hand side of (3.5), player x plays first. Knowing that player 2 (λ) will maximize their return, player 1 (x) chooses their setting to give player 2 the worst possible options over all λ. The max-min inequality says that whoever plays second has the advantage.

3.1.4 Optimality conditions

If the primal is equal to the dual, we can make some interesting observations about the duality constraints. Denote by x^* the optimum solution of the original problem (the minimum of f_0 under its constraints), and by (λ^*,ν^*) the solutions to the dual. Then

$$\begin{aligned}f_0(x^*) &= g(\lambda^*,\nu^*)\\ &\underset{(a)}{=} \inf_{x\in\mathcal{D}}\left(f_0(x)+\sum_{i=1}^m \lambda_i^* f_i(x)+\sum_{i=1}^r \nu_i^* h_i(x)\right)\\ &\underset{(b)}{\le} f_0(x^*)+\sum_{i=1}^m \lambda_i^* f_i(x^*)+\sum_{i=1}^r \nu_i^* h_i(x^*)\\ &\le f_0(x^*),\end{aligned}$$

where in (a) we use the definition of g, in (b) we use that $\inf_{x \in \mathcal{D}}$ of the expression in the parentheses is necessarily no greater than its value at x^*, and the last line we use that at (x^*, λ^*, ν^*) we have $\lambda^* \succeq 0$, $f_i(x^*) \le 0$, and $h_i(x^*) = 0$. From this chain of reasoning, it follows that

$$\sum_{i=1}^{m} \lambda_i^* f_i(x^*) = 0, \tag{3.6}$$

which is the condition of **complementary slackness**. This means

$$\begin{array}{rcl} \lambda_i^* > 0 & \Longrightarrow & f_i(x^*) = 0, \\ f_i(x^*) < 0 & \Longrightarrow & \lambda_i^* = 0. \end{array}$$

Consider now the case where the functions f_i, h_i are differentiable, and the duality gap is zero. Since x^* minimizes $L(x, \lambda^*, \nu^*)$, the derivative at x^* should be zero,

$$\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{i=1}^{r} \nu_i^* \nabla h_i(x^*) = 0.$$

We now gather the various conditions for optimality we have discussed. The **KKT conditions** for the primal and dual variables (x, λ, ν) are

$$\begin{array}{rcl} f_i(x) & \le & 0,\ i = 1, \ldots, m, \\ h_i(x) & = & 0,\ i = 1, \ldots, r, \\ \lambda_i & \ge & 0,\ i = 1, \ldots, m, \\ \lambda_i f_i(x) & = & 0,\ i = 1, \ldots, m, \\ \nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{r} \nu_i \nabla h_i(x) & = & 0. \end{array}$$

If a convex optimization problem with differentiable objective and constraint functions satisfies Slater's conditions, then the KKT conditions are necessary and sufficient for global optimality.

3.2 Support vector classification

3.2.1 The linearly separable case

We first consider the problem of classifying two clouds of points, where there exists a hyperplane which linearly separates one cloud from the other without error. This is illustrated in Figure 3.4 for a 2-dimensional classification problem. As can be seen, there are infinitely many possible hyperplanes that solve this problem: the question is then: which one to choose? The principle behind support vector machines is that we choose the one which has the largest **margin**, which is defined as twice the *smallest* distance from each class to the separating hyperplane (see Figure 3.4).

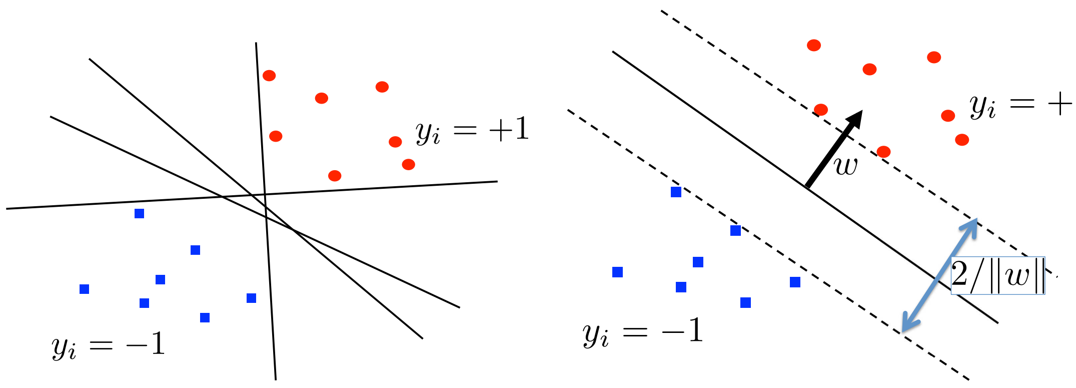


Figure 3.4: The linearly separable case. There are many linear separating hyperplanes, but only one maximum margin separating hyperplane.

This problem can be expressed as follows:[4]

$$\max_{w,b} (\text{margin}) = \max_{w,b} \left(\frac{2}{\|w\|} \right) \tag{3.8}$$

subject to

$$\begin{cases} \min \left(w^\top x_i + b \right) = 1 & i : y_i = +1, \\ \max \left(w^\top x_i + b \right) = -1 & i : y_i = -1. \end{cases} \tag{3.9}$$

The resulting classifier is

$$y = \operatorname{sign}(w^\top x + b),$$

where sign takes value $+1$ for a positive argument, and -1 for a negative argument (its value at zero is not important, since for non-pathological cases we will not need to evaluate it there). We can rewrite to obtain

$$\max_{w,b} \frac{1}{\|w\|} \quad \text{or} \quad \min_{w,b} \|w\|^2$$

subject to

$$y_i(w^\top x_i + b) \geq 1. \tag{3.10}$$

[4]It's easy to see why the equation below is the margin (the distance between the positive and negative classes): consider two points exactly opposite each other and located on the margins, such that $(x_i - x_j) = \beta w$ for some scalar β (where we recall w is orthogonal to the decision boundary, hence aligned with $x_i - x_j$). Then the distance between them (which is the width of the margin) is

$$\|x_i - x_j\| = (x_i - x_j)^\top \frac{(x_i - x_j)}{\|x_i - x_j\|} = (x_i - x_j)^\top \frac{w}{\|w\|} \tag{3.7}$$

Subtracting the two equations in the constraints (3.9) from each other, we get

$$w^\top (x_i - x_j) = 2.$$

Substituting this into (3.7) proves the result.

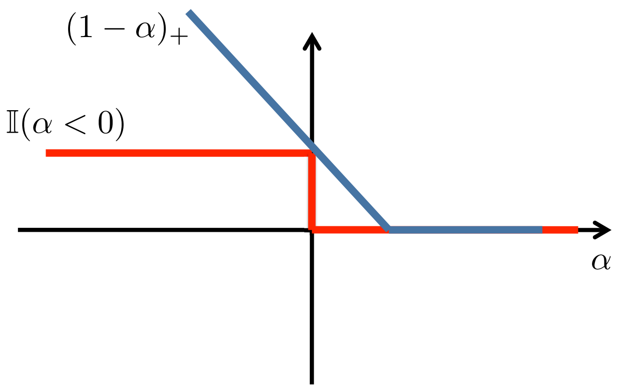


Figure 3.5: The hinge loss is an upper bound on the $0/1$ loss for misclassifications. Note it that is not an upper bound on the $0/1$ loss for the margin errors were we are instead considering $\mathbb{I}(\alpha < 1)$.

3.2.2 When no linear separator exists (or we want a larger margin)

If the classes are not linearly separable, we may wish to allow a certain number of errors in the classifier (points within the margin, or even on the wrong side of the decision boudary). We therefore want to trade off such "margin errors" vs maximising the margin. One natural choice would be to optimise

$$\min_{w,b}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\mathbb{I}\left\{y_i\left(w^\top x_i + b\right) < 1\right\}\right),$$

where C controls the tradeoff between maximum margin and loss (the factor of $1/2$ is to simplify the algebra later, and is not important: we can adjust C accordingly). This is a combinatorial optimization problem, which would be very expensive to solve. Instead, we replace the $0/1$ loss with the hinge loss (see Figure 3.5)

$$h(\alpha) = (1-\alpha)_+ = \begin{cases} 1-\alpha & 1-\alpha > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Obviously other choices are possible as well, but we will find that the hinge loss provides a number of favorable properties.

Substituting in the hinge loss, we get

$$\min_{w,b}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} h\left(y_i\left(w^\top x_i + b\right)\right)\right),$$

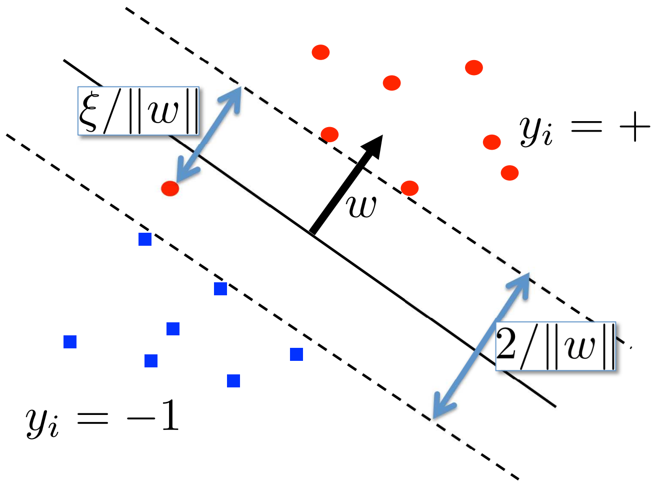


Figure 3.6: The nonseparable case. Note the red point which is a distance $\xi/\|w\|$ from the margin.

or equivalently the constrained problem

$$\min_{w,b,\xi}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i\right) \tag{3.11}$$

subject to[5]

$$\xi_i \geq 0 \qquad y_i\left(w^\top x_i + b\right) \geq 1 - \xi_i$$

(compare with (3.10)). See Figure 3.6. This formulation is known as the C-SVM.

Now let's write the Lagrangian for this problem, and solve it.

$$L(w,b,\xi,\alpha,\lambda) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i\left(1 - y_i\left(w^\top x_i + b\right) - \xi_i\right) + \sum_{i=1}^{n}\lambda_i(-\xi_i) \tag{3.12}$$

with dual variable constraints

$$\alpha_i \geq 0, \qquad \lambda_i \geq 0.$$

We minimize wrt the primal variables w, b, and ξ.

Derivative wrt w:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i = 0 \qquad w = \sum_{i=1}^{n}\alpha_i y_i x_i. \tag{3.13}$$

Derivative wrt b:

$$\frac{\partial L}{\partial b} = \sum_i y_i\alpha_i = 0. \tag{3.14}$$

[5]To see this, we can write it as $\xi_i \geq 1 - y_i\left(w^\top x_i + b\right)$. Thus either $\xi_i = 0$, and $y_i\left(w^\top x_i + b\right) \geq 1$ as before, or $\xi_i > 0$, in which case to minimize (3.11), we'd use the smallest possible ξ_i satisfying the inequality, and we'd have $\xi_i = 1 - y_i\left(w^\top x_i + b\right)$.

Derivative wrt ξ_i:

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \lambda_i = 0 \qquad \alpha_i = C - \lambda_i. \tag{3.15}$$

We can replace the final constraint by noting $\lambda_i \geq 0$, hence

$$\alpha_i \leq C.$$

Before writing the dual, we look at what these conditions imply about the scalars α_i that define the solution (3.13) due to complementary slackness.

Non-margin SVs: $\alpha_i = C > 0$:

1. We immediately have $1 - \xi_i = y_i \left(w^\top x_i + b \right)$.
2. Also, from condition $\alpha_i = C - \lambda_i$, we have $\lambda_i = 0$, hence $\xi_i \geq 0$.

Margin SVs: $0 < \alpha_i < C$:

1. We again have $1 - \xi_i = y_i \left(w^\top x_i + b \right)$
2. This time, from $\alpha_i = C - \lambda_i$, we have $\lambda_i > 0$, hence $\xi_i = 0$.

Non-SVs: $\alpha_i = 0$

1. This time we have: $y_i \left(w^\top x_i + b \right) \geq 1 - \xi_i$
2. From $\alpha_i = C - \lambda_i$, we have $\lambda_i > 0$, hence $\xi_i = 0$.

This means that the solution is *sparse*: all the points which are not either on the margin, or "margin errors", contribute nothing to the solution. In other words, only those points on the decision boundary, or which are margin errors, contribute. Furthermore, the influence of the non-margin SVs is bounded, since their weight cannot exceed C: thus, severe outliers will not overwhelm the solution.

We now write the dual function, by substituting equations (3.13), (3.14), and (3.15) into (3.12), to get

$$\begin{aligned}
g(\alpha, \lambda) &= \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i + \sum_{i=1}^n \alpha_i \left(1 - y_i \left(w^\top x_i + b \right) - \xi_i \right) + \sum_{i=1}^n \lambda_i(-\xi_i) \\
&= \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j + C\sum_{i=1}^m \xi_i - \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j - b\underbrace{\sum_{i=1}^m \alpha_i y_i}_{0} \\
&\qquad + \sum_{i=1}^m \alpha_i - \sum_{i=1}^m \alpha_i \xi_i - \sum_{i=1}^m (C - \alpha_i)\xi_i \\
&= \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j.
\end{aligned}$$

Thus, our goal is to maximize the dual,

$$g(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^\top x_j,$$

subject to the constraints

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} y_i \alpha_i = 0.$$

So far we have defined the solution for w, but not for the offset b. This is simple to compute: for the margin SVs, i.e., those x_i for which $0 < \alpha_i < C$, we have $1 = y_i \left(w^\top x_i + b\right)$. Thus, we can obtain b from any of these, or take an average for greater numerical stability.

3.2.3 The ν-SVM

It can be hard to interpret C. Therefore we modify the formulation to get a more intuitive parameter. Solve

$$\min_{w,\rho,\xi} \left(\frac{1}{2} \|w\|^2 - \nu\rho + \frac{1}{n} \sum_{i=1}^{n} \xi_i \right)$$

subject to

$$\begin{aligned} \rho &\geq 0 \\ \xi_i &\geq 0 \\ y_i \left(w^\top x_i + b \right) &\geq \rho - \xi_i, \end{aligned}$$

where we see that we are now also maximizing ρ. Moreover, we see that whereas for the C-SVM a point on the margin gave $w^\top x_i + b = \pm 1$, we now have that a point on the margin produces $w^\top x_i + b = \pm\rho$. Thus the margin width is $2\rho/\|w\|$, rather than just $2/\|w\|$. Larger values of ρ thus correspond to larger margins and we are directly trying to maximize the size of the margin through the new $-\nu\rho$ term. Our new hyperparameter, ν, now directly controls the strength of this pressure to maximize the margin width, with larger values of ν encouraging larger margins (typically at the expense of more margin errors). Note that the constraint $\rho \geq 0$ comes from the fact that the margin width must be non–negative, though in principle it can be zero. The advantage of this formulation is that ν will permit a number of useful interpretations as we now show (see also the examples sheet).

Dropping b for simplicity, the Lagrangian is

$$\frac{1}{2}\|w\|_{\mathcal{H}}^2 + \frac{1}{n} \sum_{i=1}^{n} \xi_i - \nu\rho + \sum_{i=1}^{n} \alpha_i \left(\rho - y_i(w^\top x_i + b) - \xi_i \right) + \sum_{i=1}^{n} \beta_i(-\xi_i) + \gamma(-\rho)$$

for $\alpha_i \geq 0$, $\beta_i \geq 0$, and $\gamma \geq 0$. Differentiating wrt each of the primal variables w, b, ξ, ρ, and setting to zero, we get

$$\begin{aligned} w &= \sum_{i=1}^n \alpha_i y_i x_i \\ \sum_{i=1}^n \alpha_i y_i &= 0, \\ \alpha_i + \beta_i &= \frac{1}{n} \qquad (3.16) \\ \nu &= \sum_{i=1}^n \alpha_i - \gamma \qquad (3.17) \end{aligned}$$

From $\beta_i \geq 0$, equation (3.16) implies

$$0 \leq \alpha_i \leq n^{-1}.$$

From $\gamma \geq 0$ and (3.17), we get

$$\nu \leq \sum_{i=1}^n \alpha_i.$$

We typically have $\rho > 0$ at the global solution (i.e. non-zero margin) and hence $\gamma = 0$, and (3.17) becomes

$$\sum_{i=1}^n \alpha_i = \nu. \qquad (3.18)$$

Complementary slackness conditions now lead to a very convenient interpretation of parameter ν. In particular, if we denote by $N(\alpha)$ the set of non-margin support vectors, i.e. margin errors, and by $M(\alpha)$ the set of margin support vectors, then (problem sheet):

$$\frac{|N(\alpha)|}{n} \leq \nu \leq \frac{|N(\alpha)| + |M(\alpha)|}{n}.$$

Thus ν corresponds to an upper bound on the proportion of margin errors and a lower bound on the proportion of the overall number of support vectors - tuning ν is hence much more interpretable than tuning C.

Substituting into the Lagrangian, we can also obtain the dual formulation of ν-SVM, i.e.

$$\begin{aligned} &\frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j + \frac{1}{n}\sum_{i=1}^n \xi_i - \rho\nu - \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j + \sum_{i=1}^n \alpha_i \rho - \sum_{i=1}^n \alpha_i \xi_i \\ &\qquad - \sum_{i=1}^n \left(\frac{1}{n} - \alpha_i\right)\xi_i - \rho\left(\sum_{i=1}^n \alpha_i - \nu\right) \\ &= -\frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^\top x_j \end{aligned}$$

Thus, we must maximize

$$g(\alpha) = -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^\top x_j,$$

subject to

$$\sum_{i=1}^{n}\alpha_i \geq \nu \qquad 0 \leq \alpha_i \leq \frac{1}{n}.$$

4 Kernel Methods

4.1 Feature Maps and Feature Spaces

Kernel methods are a versatile algorithmic framework which allows construction of nonlinear machine learning algorithms (for a variety of both supervised and unsupervised learning tasks: clustering, dimensionality reduction, classification, regression) by employing linear tools in a nonlinearly transformed feature space. Let us first recall the definition of an abstract inner product, which is central to kernel methods.

Definition 4. Let $\mathcal{H}$ be a vector space over $\mathbb{R}$. A function $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H} \times \mathcal{H} \to \mathbb{R}$ is said to be an **inner product** on $\mathcal{H}$ if

1. $\langle \alpha_1 f_1 + \alpha_2 f_2, g \rangle_{\mathcal{H}} = \alpha_1 \langle f_1, g \rangle_{\mathcal{H}} + \alpha_2 \langle f_2, g \rangle_{\mathcal{H}}$ (*linear*),
2. $\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$ (*symmetric*),
3. $\langle f, f \rangle_{\mathcal{H}} \geq 0$,
4. $\langle f, f \rangle_{\mathcal{H}} = 0$ if and only if $f = 0$.

We can define a **norm** using the inner product as $\|f\|_{\mathcal{H}} := \sqrt{\langle f, f \rangle_{\mathcal{H}}}$. A **Hilbert space** is a vector space on which an inner product is defined, along with an additional technical condition.[1] We are now ready to define the notion of a *kernel*.

Definition 5. Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **kernel** if there exists a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that $\forall x, x' \in \mathcal{X}$,

$$k(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} .$$

We will call the Hilbert space associated with kernel k a **feature space** and the map φ will be called a **feature map**. Note that we imposed almost no conditions on $\mathcal{X}$: in particular, we do not require there to be an inner product defined on the elements of $\mathcal{X}$. The case of text documents is an instructive example: one cannot take an inner product between two books, but can take an inner product between vector-valued features of the text in those books.

Clearly, a single kernel can correspond to multiple pairs of underlying feature maps and feature spaces. For a simple example, consider $\mathcal{X} := \mathbb{R}^p$:

$$\varphi_1(x) = x \qquad \text{and} \qquad \varphi_2(x) = \left[\frac{x_1}{\sqrt{2}}, \cdots, \frac{x_p}{\sqrt{2}}, \frac{x_1}{\sqrt{2}}, \cdots, \frac{x_p}{\sqrt{2}} \right]^\top .$$

[1]Specifically, a Hilbert space must be *complete*, i.e. it must contain the limits of all Cauchy sequences with respect to the norm defined by its inner product.

Both φ_1 and φ_2 are valid feature maps (with feature spaces $\mathcal{H}_1 = \mathbb{R}^p$ and $\mathcal{H}_2 = \mathbb{R}^{2p}$) of kernel $k(x, x') = x^\top x'$.

It turns out that all kernel functions (defined as inner products between some features) are *positive definite*.

Definition 6. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is **positive definite** if $\forall n \geq 1$, $\forall (a_1, \dots a_n) \in \mathbb{R}^n$, $\forall (x_1, \dots, x_n) \in \mathcal{X}^n$,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) \geq 0.$$

The function $k(\cdot, \cdot)$ is **strictly positive definite** if for mutually distinct x_i, the equality holds only when all the a_i are zero.[2]

Every inner product is a positive definite function, and so is every inner product between feature maps.

Lemma 7. *Let $\mathcal{H}$ be any Hilbert space, $\mathcal{X}$ a non-empty set, and $\varphi : \mathcal{X} \to \mathcal{H}$. Then $k(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$ is a positive definite function.*

Proof.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(x_i, x_j) = \sum_{i=1}^{n} \sum_{j=1}^{n} \langle a_i \varphi(x_i), a_j \varphi(x_j) \rangle_{\mathcal{H}} = \left\| \sum_{i=1}^{n} a_i \varphi(x_i) \right\|_{\mathcal{H}}^2 \geq 0.$$

□

4.2 Reproducing Kernel Hilbert Spaces

We have introduced the notation of feature spaces, and kernels on these feature spaces. What's more, we've determined that these kernels are positive definite. In this section, we use these kernels to define *functions* on $\mathcal{X}$. The space of such functions is known as a **reproducing kernel Hilbert space** (RKHS) defined as follows. We note that this definition is rather strange and will likely be quite difficult to appreciate how a Hilbert space might satisfy the required conditions; for now the important thing is to simply appreciate that RKHSs and reproducing kernels exist.

Definition 8. Let $\mathcal{H}$ be a *Hilbert space of functions* $f \colon \mathcal{X} \to \mathbb{R}$ defined on a non-empty set $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a **reproducing kernel** of $\mathcal{H}$ if it satisfies

- $\forall x \in \mathcal{X}, \quad k_x := k(\cdot, x) \in \mathcal{H}$,
- $\forall x \in \mathcal{X}, \forall f \in \mathcal{H}, \quad \langle f, k(\cdot, x) \rangle_{\mathcal{H}} = f(x)$ (*the reproducing property*).

If $\mathcal{H}$ has a reproducing kernel, it is called a reproducing kernel Hilbert space (RKHS).

[2]The corresponding terminology used for matrices is "positive semi-definite" vs "positive definite".

Note that a reproducing kernel is also a conventional kernel with feature map $\varphi\colon x \mapsto k(\cdot, x)$, known as the **canonical feature map**. We can show this by following the reproducing property backwards with $f = k(\cdot, x')$:

$$k(x, x') = f(x) = \langle f, k(\cdot, x) \rangle_{\mathcal{H}} = \langle k(\cdot, x'), k(\cdot, x) \rangle_{\mathcal{H}} = \langle \varphi(x'), \varphi(x) \rangle_{\mathcal{H}},$$

such that we see that $k(x, x')$ can be expressed as an inner product between features with a feature space $\mathcal{H}$. Note that these features are not specified explicitly in a vector form, but rather as functions on $\mathcal{X}$.

Perhaps the most mind–bending part of kernels methods is now that it turns out that every positive definite function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is also a *reproducing* kernel with a *unique* corresponding RKHS! That is, not only does k being a positive definite function imply that it is also a kernel that can be written as an inner product (such that Lemma 7 effectively also holds in reverse), it can specifically be written as an inner product in a RKHS such that the reproducing property holds. This is know as the **Moore-Aronszajn theorem** [5]. Though a full formal proof of this theorem is beyond the scope of this course,[3] we can still gain substantial insight into kernels, and in particular RKHSs, by deriving the RKHS for an arbitrary positive definite k as follows.

4.2.1 Deriving the RKHS for an Arbitrary Positive Definite Function

Our starting point is simply to define $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to be an arbitrary symmetric positive definite function (note that we are not a priori assuming that this is a valid kernel). In what follows, we will show that k is a reproducing kernel and derive its corresponding RKHS.

Consider functions of the form $f = \sum_{i=1}^{r} \alpha_i k(\cdot, x_i)$ and define $\mathcal{H}_0$ to be the set of all such possible functions, that is

$$\mathcal{H}_0 := \left\{ \sum_{i=1}^{r} \alpha_i k(\cdot, x_i) \right\}_{r \in \mathbb{N},\, \alpha_i \in \mathbb{R},\, x_i \in \mathcal{X}} \tag{4.1}$$

such that $\mathcal{H}_0$ represents the vector space span $\{k(\cdot, x) : x \in \mathcal{X}\}$. Note here that r is unbounded so that f may be formed of an infinite sum. Note also that we trivially have $k(\cdot, x) \in \mathcal{H}_0$ and so the first condition of our RKHS definition will be trivially satisfied provided that $\mathcal{H}_0$ is a subset of the RKHS we form.

Next, we define a functional $h\colon (\mathcal{X} \mapsto \mathbb{R}) \times (\mathcal{X} \mapsto \mathbb{R}) \mapsto \mathbb{R}$ that acts on pairs of $k(\cdot, x)$:

$$h\left(k(\cdot, x), k(\cdot, x')\right) := k(x, x').$$

This is a slightly unusual functional, but on close examination we see that it is unambiguous and well–defined, even if it is somewhat unclear how we would actually perform the required calculations in practice. Namely, because $k(\cdot, x)$ uniquely defines $k(x, x')$ for all possible x' and $k(\cdot, x')$ uniquely defines $k(x, x')$ for all possible x, there is only

[3]See `http://www.stats.ox.ac.uk/~sejdinov/teaching/atml14/Theory_2014.pdf` if you would like the whole shebang.

one possible $k(x, x')$ that is compatible with both $k(\cdot, x)$ and $k(\cdot, x')$, from which we can conclude that h is a well–defined functional.

Using h, we will now define an inner product for $\mathcal{H}_0$, $\langle \cdot, \cdot \rangle_{\mathcal{H}} : \mathcal{H}_0 \times \mathcal{H}_0 \to \mathbb{R}$. Here it is essential to realize that we are constructing/defining this inner product ourselves: it is **not** already implied by (4.1) and we are making deliberate choices to ensure it gives us the properties we want (namely that the reproducing property will hold). In particular, our use of h (which was again our own construction) is not dictated by the space defined in (4.1), but by deliberate choice to induce desired properties in our inner product. As such, there are (at least in principle) alternative inner products we could have defined instead, but these would have implied different Hilbert spaces, remembering that a Hilbert space definition is based on *both* its vector space and its inner product.

To construct this definition of $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, we consider an arbitrary pair $f, g \in \mathcal{H}_0$, where without loss of generality we can denote $f(\cdot) = \sum_{i=1}^{r} \alpha_i k(\cdot, x_i)$ and $g(\cdot) = \sum_{j=1}^{s} \beta_j k(\cdot, x'_j)$, and then define

$$\begin{aligned} \langle f, g \rangle_{\mathcal{H}} :=& \sum_{i=1}^{r} \sum_{j=1}^{s} \alpha_i \beta_j h\left(k(\cdot, x_i), k\left(\cdot, x'_j\right)\right) \qquad (4.2) \\ =& \sum_{i=1}^{r} \sum_{j=1}^{s} \alpha_i \beta_j k\left(x_i, x'_j\right), \end{aligned}$$

where the second line simply involves substituting in the definition of h. Given this definition, we can define our RKHS, $\mathcal{H}$, by completing $\mathcal{H}_0$ with respect to the norm defined by our inner product, $\|\cdot\|_{\mathcal{H}}$. Informally, this means that we "plug all the gaps" in $\mathcal{H}_0$ to produce a complete metric space; e.g. a necessary step in completing the set of rational numbers is to plug the gaps formed by the irrational numbers. More formally, it is done by adding to $\mathcal{H}_0$ the limits of all Cauchy sequences on $\mathcal{H}_0$ with respect to $\|\cdot\|_{\mathcal{H}}$.

Now we have defined $\mathcal{H}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, we need to check the latter is indeed a valid inner product. This is an exercise of proving each of the four conditions of Definition 4. To prove the first condition (linearity), we first note

$$\begin{aligned} af_1 + bf_2 &= a \sum_{i=1}^{r_1} \alpha_{i,1} k(\cdot, x_{i,1}) + b \sum_{i=1}^{r_2} \alpha_{i,2} k(\cdot, x_{i,2}) \\ &= \sum_{i=1}^{r_1+r_2} \alpha_i k(\cdot, x_i) \quad \text{where} \begin{cases} \alpha_i = \mathbb{I}(i \le r_1) a \alpha_{i,1} + \mathbb{I}(i > r_1) b \alpha_{i-r_1,2} \\ x_i = \mathbb{I}(i \le r_1) x_{i,1} + \mathbb{I}(i > r_1) x_{i-r_1,2} \end{cases} \end{aligned}$$

Therefore,

$$\begin{aligned} \langle af_1 + bf_2, g \rangle_{\mathcal{H}} &= \sum_{i=1}^{r_1+r_2} \sum_{j=1}^{s} \alpha_i \beta_j k(x_i, x'_j) \\ &= \sum_{j=1}^{s} \beta_j \left(a \sum_{i=1}^{r_1} \alpha_{i,1} k(x_{i,1}, x'_j) + b \sum_{i=1}^{r_2} \alpha_{i,2} k(x_{i,2}, x'_j) \right) \end{aligned}$$

$$= a \sum_{i=1}^{r_1} \sum_{j=1}^{s} \alpha_{i,1} \beta_j k\left(x_{i,1}, x'_j\right) + b \sum_{i=1}^{r_2} \sum_{j=1}^{s} \alpha_{i,2} \beta_j k\left(x_{i,2}, x'_j\right)$$
$$= a \langle f_1, g \rangle_{\mathcal{H}} + b \langle f_2, g \rangle_{\mathcal{H}}$$

as required and we see that the conditional holds regardless of k.

The symmetry condition, that $\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$ similarly follows trivially for all k from the symmetric nature of our definitions for $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and h.

The third condition, $\langle f, f \rangle_{\mathcal{H}} \geq 0$, turns out to be equivalent to our initial assumption that k is positive definite, as

$$\langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{r} \sum_{j=1}^{r} \alpha_i \alpha_j k\left(x_i, x_j\right)$$

$\geq 0 \; \forall f$ if and only if k is positive definite (remembering that f is defined through the α_i and x_i so this holding for all f is equivalent to this holding for all α_i and x_i).

The final condition, that $\langle f, f \rangle_{\mathcal{H}} = 0$ if and only if $f = 0$, requires us to first consider what the statement $f = 0$ actually means. Importantly, this statement is *not* equivalent to saying that $\alpha_i = 0 \; \forall i$. Instead, we might naturally say that $f = 0 \Leftrightarrow f(x) = 0 \; \forall x \in \mathcal{X}$, i.e. that $f = 0$ means the function evaluates to zero for all possible inputs. From this definition we can immediately show that $f = 0$ implies $\langle f, f \rangle_{\mathcal{H}} = 0$ by noting that

$$\|f\|^2_{\mathcal{H}_k} = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^{r} \alpha_i \sum_{j=1}^{r} \alpha_j k(x_i, x_j) = \sum_{i=1}^{r} \alpha_i f(x_i)$$

$= 0$ if $f(x) = 0, \; \forall x$. Proving that $\langle f, f \rangle_{\mathcal{H}} = 0$ implies $f = 0$ is a little trickier[4] and is covered in the examples sheet, but ultimately does also turn out to be true.

Putting these together, we see that our initial assumption that k was positive definite will guarantee that $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is a valid inner product. We have thus derived a valid Hilbert space. Moreover, we have that $k(\cdot, x) \in \mathcal{H}$ and

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

such that we have proven that k is a valid kernel with the canonical feature map $\varphi \colon x \mapsto k(\cdot, x)$.

We can now prove that k is a reproducing kernel and the derived $\mathcal{H}$ is an RKHS by showing that the reproducing property must hold given our earlier definitions. This is straightforwardly done as follows

$$\langle f, k(\cdot, x) \rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^{r} \alpha_i k(\cdot, x_i), k(\cdot, x) \right\rangle_{\mathcal{H}}$$
$$= \sum_{i=1}^{r} \alpha_i \langle k(\cdot, x_i), k(\cdot, x) \rangle_{\mathcal{H}}$$

[4]Note that we cannot just prove this with the Cauchy–Schwarz inequality (or similar) in the manner one might naturally try to do, as this requires $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ to be a valid inner product in the first place.

$$= \sum_{i=1}^{r} \alpha_i k(x, x_i)$$
$$= f(x) \quad \text{by definition.}$$

Taking stock, we have shown that any positive definite k is a reproducing kernel with a corresponding RKHS whose feature maps correspond to $\varphi(x) = k(\cdot, x)$. We have also shown that reproducing kernels are kernels in the sense of being expressible as inner products between features, and that kernels expressed as inner products between features are positive definite. We have thus come full circle and can conclude that all three notions—i.e. (1) a reproducing kernel, (2) a kernel as an inner product between features, and (3) a positive definite function—are *equivalent.*

We can further straightforwardly prove that any RKHS has a unique reproducing kernel by contradiction. Namely, assume that $\mathcal{H}$ has two distinct reproducing kernels k_1 and k_2. We then have, by linearity and the reproducing property, $\forall f \in \mathcal{H}, x \in \mathcal{X}$:

$$\langle f, k_1(\cdot, x) - k_2(\cdot, x)\rangle_{\mathcal{H}} = \langle f, k_1(\cdot, x)\rangle_{\mathcal{H}} - \langle f, k_2(\cdot, x)\rangle_{\mathcal{H}} = f(x) - f(x) = 0.$$

In particular, this holds for $f = k_1(\cdot, x) - k_2(\cdot, x)$, which yields $\|k_1(\cdot, x) - k_2(\cdot, x)\|_{\mathcal{H}}^2 = 0$, $\forall x \in \mathcal{X}$, which implies $k_1 = k_2$ and we have our desired contradiction.

The inverse result also holds: any reproducing kernel (and thus any positive definite function) has a unique RKHS. This is a little trickier to formalize and prove, but essentially follows from the fact that we could not have constructed the RKHS in a different way and still ensured the reproducing property holds. We will thus henceforth denote the RKHS of kernel k by $\mathcal{H}_k$. For example, for the linear kernel $k(x, x') = x^\top x'$ considered earlier, many possible feature representations exist, but the canonical feature representation that associates to each x the function $k(\cdot, x)\colon x' \mapsto x^\top x'$ is unique and is what determines the structure of its RKHS. In particular, the linear kernel $k(x, x') = x^\top x'$ corresponds to the RKHS $\mathcal{H}_k$ which is the space of all linear functions $f(x) = w^\top x$.

4.3 Operations with Kernels

Kernels can be combined and modified to get new kernels. For example,

Lemma 9. *[Sums of kernels are kernels] Given $\alpha > 0$ and k, k_1 and k_2 all kernels on $\mathcal{X}$, then αk and $k_1 + k_2$ are kernels on $\mathcal{X}$.*

To prove the above, just check *positive definiteness.* Note that a difference between two kernels need not be a kernel: if $k_1(x, x) - k_2(x, x) < 0$, then condition 3 of inner product definition 4 may be violated.

Lemma 10. *[Mappings between spaces] Let $\mathcal{X}$ and $\widetilde{\mathcal{X}}$ be non-empty sets, and define a map $A : \mathcal{X} \to \widetilde{\mathcal{X}}$. Define the kernel k on $\widetilde{\mathcal{X}}$. Then $k(A(x), A(x'))$ is a kernel on $\mathcal{X}$.*

The proof of this is again straight forward: if k is a kernel then $k(A(x), A(x')) = \langle \varphi(A(x)), \varphi(A(x'))\rangle_{\mathcal{H}}$ which is a kernel with features $\varphi(A(x))$. This result is important when we want to define kernels on inputs that do not live in the reals (i.e. $\mathcal{X} \nsubseteq \mathbb{R}^p$): we can project our inputs into the space of reals and then apply a standard kernel in the projected space.

Lemma 11. *[Products of kernels are kernels] Given k on $\mathcal{X}$ and l on $\mathcal{Y}$, then*

$$\kappa\left((x,y),(x',y')\right) = k\left(x,x'\right) l\left(y,y'\right)$$

is a kernel on $\mathcal{X} \times \mathcal{Y}$. Moreover, if $\mathcal{X} = \mathcal{Y}$, then

$$\kappa\left(x,x'\right) = k\left(x,x'\right) l\left(x,x'\right)$$

is a kernel on $\mathcal{X}$.

Here the general proof would require some technical details about Hilbert space tensor products, but the main idea can be understood with some simple linear algebra. We consider the case where $\mathcal{H}$ corresponding to k is $\mathbb{R}^M$, and $\mathcal{G}$ corresponding to l is $\mathbb{R}^N$. Write $k\left(x,x'\right) = \varphi(x)^\top \varphi(x')$ and $l\left(y,y'\right) = \psi(y)^\top \psi(y')$. We will use that a notion of inner product between matrices $A \in \mathbb{R}^{M\times N}$ and $B \in \mathbb{R}^{M\times N}$ is given by

$$\langle A, B\rangle = \operatorname{trace}(A^\top B). \tag{4.3}$$

Then

$$\begin{aligned} k\left(x,x'\right) l\left(y,y'\right) &= \varphi(x)^\top \varphi(x')\psi(y')^\top \psi(y) \\ &= \operatorname{tr}(\psi(y)\varphi(x)^\top \varphi(x')\psi(y')^\top) \\ &= \left\langle \varphi(x)\psi(y)^\top, \varphi(x')\psi(y')^\top \right\rangle, \end{aligned}$$

thus we can define features $A(x,y) = \varphi(x)\psi(y)^\top$ of the product kernel.

The sum and product rules allow us to define a huge variety of kernels.

Lemma 12. ***[polynomial kernel]*** *Let $x, x' \in \mathbb{R}^p$ for $p \geq 1$, and let $m \geq 1$ be an integer and $c \geq 0$. Then*

$$k(x,x') := \left(\langle x, x'\rangle + c\right)^m$$

is a valid kernel.

To prove: expand out this expression into a sum (with non-negative scalars) of kernels $\langle x, x'\rangle$ raised to integer powers. These individual terms are valid kernels by the product rule.

Can we extend this combination of sum and product rule to sums with infinitely many terms? Consider for example the exponential function applied to an inner product $k(x,x') = \exp\left(\langle x, x'\rangle\right)$. Since addition and multiplication preserve positive definiteness and since all the coefficients in the Taylor series expansion of the exponential function are nonnegative, $k_m(x,x') = \sum_{r=1}^{m} \frac{\langle x,x'\rangle^r}{r!}$ is a valid kernel $\forall m \in \mathbb{N}$. Fix some $\{\alpha_i\}$ and $\{x_i\}$. Then $A_m = \sum_{i,j} \alpha_i \alpha_j k_m(x_i, x_j) \geq 0$ $\forall m$ since k_m is positive definite. But $A_m \to \sum_{i,j} \alpha_i \alpha_j \exp\left(\langle x_i, x_j\rangle\right)$ as $m \to \infty$, so $\sum_{i,j} \alpha_i \alpha_j \exp\left(\langle x_i, x_j\rangle\right) \geq 0$ as well. Thus,

$\exp(\langle x, x' \rangle)$ is also a valid kernel (it is called **exponential kernel**). We may combine all the results above (*exercise*) to show that the following kernel, in practice widely used and known under various names: **Gaussian kernel**, **RBF kernel**, **squared exponential kernel** or **exponentiated quadratic kernel**, is valid on $\mathbb{R}^p$:

$$k(x, x') := \exp\left(-\frac{1}{2\gamma^2}\|x - x'\|^2\right).$$

The RKHS of this kernel is infinite-dimensional. Moreover, if the domain $\mathcal{X}$ is a compact subset of $\mathbb{R}^p$, its RKHS is dense in the space of all bounded continuous functions with respect to the uniform norm. Despite that, since all functions in its RKHS are infinitely differentiable, Gaussian kernel is often considered to be excessively smooth. A less smooth alternative is the **Matérn kernel**, given by

$$k(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}}{\gamma}\|x - x'\|\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{\gamma}\|x - x'\|\right), \quad \nu > 0, \gamma > 0,$$

where K_ν is the modified Bessel function of the second kind of order ν. The Matérn kernels corresponding to the values $\nu = s + \frac{1}{2}$ for non-negative integers s take a simpler form, in particular:

- $\nu = 1/2$: $k(x, x') = \exp\left(-\frac{1}{\gamma}\|x - x'\|\right)$,
- $\nu = 3/2$: $k(x, x') = \left(1 + \frac{\sqrt{3}}{\gamma}\|x - x'\|\right)\exp\left(-\frac{\sqrt{3}}{\gamma}\|x - x'\|\right)$,
- $\nu = 5/2$: $k(x, x') = \left(1 + \frac{\sqrt{5}}{\gamma}\|x - x'\| + \frac{5}{3\gamma^2}\|x - x'\|^2\right)\exp\left(-\frac{\sqrt{5}}{\gamma}\|x - x'\|\right)$.

For $\nu = s + \frac{1}{2}$, its RKHS consists of s times differentiable functions with square integrable derivatives of order up to $s + 1$. Moreover, the RKHS norms directly penalize the derivatives of f, e.g. for $\nu = 3/2$ and in one dimension, it can be shown that

$$\|f\|^2_{\mathcal{H}_k} \propto \int f''(x)^2 dx + \frac{6}{\gamma^2}\int f'(x)^2 dx + \frac{9}{\gamma^4}\int f(x)^2 dx.$$

As $\nu \to \infty$, Matérn kernel converges to the Gaussian RBF, i.e. $k(x, x') = \exp\left(-\frac{1}{2\gamma^2}\|x - x'\|^2\right)$.

Another popular choice is the **rational quadratic kernel** which arises as a scale mixture of Gaussian kernels. In particular, consider Gaussian RBF parametrisation $k_\theta(x, x') = \exp\left(-\theta\|x - x'\|^2\right)$ and a Gamma density placed on θ, i.e.

$$p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{\alpha-1}\exp(-\beta\theta)$$

with shape α and rate β.

We then define

$$\begin{aligned}\kappa(x,x') &= \int_0^\infty k_\theta(x,x')p(\theta)d\theta \\ &= \frac{\beta^\alpha}{\Gamma(\alpha)}\int_0^\infty \exp\left(-\theta\left(\left\|x-x'\right\|^2+\beta\right)\right)\theta^{\alpha-1}d\theta \\ &= \frac{\beta^\alpha}{\Gamma(\alpha)}\frac{\Gamma(\alpha)}{\left(\|x-x'\|^2+\beta\right)^\alpha} \\ &= \left(1+\frac{\|x-x'\|^2}{\beta}\right)^{-\alpha}.\end{aligned}$$

Rational quadratic RKHSs contain functions which vary smoothly across multiple length-scales. If we write $\beta = 2\alpha\gamma^2$ and let $\alpha \to \infty$ we again recover Gaussian RBF, i.e. $\exp\left(-\frac{1}{2\gamma^2}\|x-x'\|^2\right)$.

4.4 Representer Theorem

Now that we have defined an RKHS and have a feel for different reproducing kernels, we can consider a RKHS as a hypothesis class for empirical risk minimisation (ERM). A typical and general setup would be that we are looking for the function f^* in the RKHS $\mathcal{H}_k$ which solves the regularised ERM problem

$$\operatorname{argmin}_{f\in\mathcal{H}_k} \hat{R}(f) + \Omega\left(\|f\|^2_{\mathcal{H}_k}\right),$$

for empirical risk $\hat{R}(f) = \frac{1}{n}\sum_{i=1}^n L(y_i, f(x_i), x_i)$, a loss function $L\colon \mathcal{Y}\times\mathcal{Y}\times\mathcal{X}\to\mathbb{R}_+$ and any non-decreasing function Ω.

Theorem 13. *There is always a solution to*

$$\operatorname{argmin}_{f\in\mathcal{H}_k}\ \hat{R}(f) + \Omega\left(\|f\|^2_{\mathcal{H}_k}\right) \tag{4.4}$$

that takes the form

$$f^* = \sum_{i=1}^n \alpha_i k(\cdot, x_i), \qquad \alpha_i \in \mathcal{R} \tag{4.5}$$

where x_i are our input datapoints. If Ω is strictly increasing, all solutions have this form.

Proof. Let f be any minimiser of (4.4). Denote by f_s the projection of f onto the subspace

$$\operatorname{span}\{k(\cdot,x_i):\ i=1,\ldots,n\}$$

such that

$$f = f_s + f_\perp,$$

where $f_s = \sum_{i=1}^n \alpha_i k(\cdot, x_i)$ and $f_\perp$ is orthogonal to the subspace span $\{k(\cdot, x_i) : i = 1, \ldots, n\}$. Since

$$\|f\|_{\mathcal{H}_k}^2 = \|f_s\|_{\mathcal{H}_k}^2 + \|f_\perp\|_{\mathcal{H}_k}^2 \geq \|f_s\|_{\mathcal{H}_k}^2 ,$$

we have

$$\Omega\left(\|f\|_{\mathcal{H}_k}^2\right) \geq \Omega\left(\|f_s\|_{\mathcal{H}_k}^2\right).$$

On the other hand, the individual terms $f(x_i)$ in the loss are given by

$$f(x_i) = \langle f, k(\cdot, x_i)\rangle_{\mathcal{H}_k} = \langle f_s + f_\perp, k(\cdot, x_i)\rangle_{\mathcal{H}_k} = \langle f_s, k(\cdot, x_i)\rangle_{\mathcal{H}_k} = f_s(x_i),$$

so

$$L(y_i, f(x_i), x_i) = L(y_i, f_s(x_i), x_i) \quad \forall i = 1, \ldots, n.$$

and thus empirical risks must be the same: $\hat{R}(f) = \hat{R}(f_s)$. Thus f_s is also a minimiser of (4.4) and if Ω is strictly increasing, it must be that $f_\perp = 0$. □

We see that the key parts of the theorem are the fact that the empirical risk only depends on the components of f lying in the subspace spanned by the canonical features and that the regulariser $\Omega(\cdot)$ is minimised when $f = f_s$ (adding additional orthogonal components to the function makes it more complex but does not change the empirical risk). Moreover, if Ω is strictly increasing, then $\|f_\perp\|_{\mathcal{H}_k} = 0$ is required at the minimum.

4.5 Kernel SVM

We can straightforwardly define a maximum margin classifier, i.e. a Support Vector Machine (SVM) in the RKHS. We write the original hinge loss formulation of the regularized empirical risk minimization (ignoring the offset b here for simplicity[5]):

$$\min_{w \in \mathcal{H}} \left(\frac{1}{2}\|w\|_{\mathcal{H}}^2 + C \sum_{i=1}^n \left(1 - y_i \langle w, k(x_i, \cdot)\rangle_{\mathcal{H}}\right)_+ \right)$$

for the RKHS $\mathcal{H}$ with kernel $k(x, x')$. This **kernel SVM** satisfies the assumption of the representer theorem, so we are looking for the solutions of the form

$$w = \sum_{i=1}^n \beta_i k(x_i, \cdot). \tag{4.6}$$

In this case, maximizing the margin in the RKHS is equivalent to minimizing $\|w\|_{\mathcal{H}}^2$: as we have seen, for many RKHSs (e.g. the RKHS corresponding to a Gaussian kernel), this corresponds to enforcing smoothness of the learned functions.

Substituting (4.6) and introducing the ξ_i variables as before, we get

$$\min_{\beta, \xi} \left(\frac{1}{2}\beta^\top K \beta + C \sum_{i=1}^n \xi_i \right) \tag{4.7}$$

[5]Note that it suffices to add a constant feature or equivalently use the kernel $k(x, x') + c$ for constant c.

$$\text{subject to } \xi_i \geq 0 \qquad y_i \sum_{j=1}^{n} \beta_j k(x_i, x_j) \geq 1 - \xi_i$$

where the Gram matrix K has i, jth entry $K_{ij} = k(x_i, x_j)$. Thus, the primal variables w are replaced with coefficients β. Note that the problem remains convex since matrix K is positive definite. With an easy calculation (left for exercise), we can verify that the dual takes the form

$$g(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j),$$

subject to the constraints

$$0 \leq \alpha_i \leq C,$$

and the decision function takes the form

$$w = \sum_{i=1}^{n} y_i \alpha_i k(x_i, \cdot).$$

This is analogous to the original dual SVM, with inner products replaced with the kernel k.

4.6 Kernel PCA

Kernel PCA is a popular nonlinear dimensionality reduction technique [49]. Assume we have a dataset $\{x_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^p$. Consider an explicit feature transformation $x \mapsto \varphi(x) \in \mathcal{H}$, and assume that we are interested in performing PCA in the feature space $\mathcal{H}$. Assume that the features $\{\varphi(x_i)\}_{i=1}^n$ are centred. Assume for the moment that the feature space is finite-dimensional, i.e. $\mathcal{H} = \mathbb{R}^M$. Then the $M \times M$ sample covariance matrix in the feature space is given by

$$\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^{n} \varphi(x_i) \varphi(x_i)^\top = \frac{1}{n-1} \Phi^\top \Phi,$$

where $\Phi \in \mathbb{R}^{n \times M}$ is the feature representation of the data. To perform PCA, recall that we are interested in solving the eigenvalue problem $\mathbf{S} v_m = \lambda_m v_m$, $m = 1, \ldots, M$, and we need the top $k \ll \min\{n, M\}$ eigenvectors v_m, $m = 1, \ldots, k$, to construct the PC projections $z_i^{(m)} = v_m^\top \varphi(x_i)$. A property analogous to the representer theorem holds here: whenever $\lambda_m > 0$, the eigenvectors lie in the linear span of feature vectors span $\{\varphi(x_i) : i = 1, \ldots, n\}$, i.e.

$$v_m = \sum_{i=1}^{n} a_{mi} \varphi(x_i) \tag{4.8}$$

for some scalars a_{mi}. To see this, note that

$$\lambda_m v_m = \mathbf{S} v_m = \frac{1}{n-1} \sum_{i=1}^{n} \varphi(x_i) \left(\varphi(x_i)^\top v_m \right)$$

and since $\lambda_m > 0$, it suffices to take $a_{mi} = \frac{1}{\lambda_m(n-1)} \left(\varphi(x_i)^\top v_m \right)$ and clearly v_m has form (4.8). Thus eigenvectors can also be recovered in the feature space. Consider now the $n \times n$ kernel matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k(x_i, x_j) = \varphi(x_i)^\top \varphi(x_j)$. By substituting $v_m = \sum_{i=1}^{n} a_{mi} \varphi(x_i)$ back into the eigenvalue problem, we have:

$$\mathbf{S} v_m = \frac{1}{n-1} \sum_{i=1}^{n} \varphi(x_i) \sum_{\ell=1}^{n} a_{m\ell} k(x_i, x_\ell) = \lambda_m \sum_{i=1}^{n} a_{mi} \varphi(x_i).$$

To express the above in terms of the kernel matrix, we project both sides onto $\varphi(x_j)$, for each $j = 1, \ldots, n$. This gives

$$\frac{1}{n-1} \sum_{i=1}^{n} k(x_j, x_i) \sum_{\ell=1}^{n} a_{m\ell} k(x_i, x_\ell) = \lambda_m \sum_{i=1}^{n} a_{mi} k(x_j, x_i), \quad j = 1, \ldots, n,$$

which in matrix notation can be written as

$$\mathbf{K}^2 a_m = \lambda_m (n-1) \mathbf{K} a_m.$$

Assuming that $\mathbf{K}$ is invertible, a_m vectors can be found as the eigenvectors of the kernel matrix $\mathbf{K}$ with corresponding eigenvalues given by $\lambda_m(n-1)$.

But if we simply perform the eigendecomposition of $\mathbf{K}$, we will obtain n-dimensional eigenvectors of unit norm, and we are after the M-dimensional eigenvectors v_m of $\mathbf{S}$ which have unit norm. We see that $1 = v_m^\top v_m = a_m^\top \mathbf{K} a_m = \lambda_m(n-1) a_m^\top a_m$. Thus, if u_m denotes the m-th eigenvector of $\mathbf{K}$ with unit norm, to ensure that v_m has unit norm, we need to rescale $a_m = u_m / \sqrt{\lambda_m(n-1)}$. Now, we have an implicit representation of eigenvectors in terms of their dual coefficients. The PC projections are

$$z_i^{(m)} = v_m^\top \varphi\left(x_i\right) = \left(\sum_{j=1}^{n} a_{mj} \varphi(x_j) \right)^\top \varphi(x_i) = \sum_{j=1}^{n} a_{mj} k(x_j, x_i),$$

or equivalently, the m-th dimension of the PC projections is given by

$$\mathbf{z}^{(m)} \;=\; \mathbf{K} a_m = \lambda_m (n-1) a_m = \sqrt{\lambda_m(n-1)} u_m. \tag{4.9}$$

We have seen this before! Note that PC projections can be discovered from the SVD $\Phi = UDV^\top$ as either $\mathbf{Z} = \Phi V$ or $\mathbf{Z} = UD$. The latter expression is exactly (4.9), since u_m are the eigenvectors of kernel matrix $\mathbf{K}$ (i.e. the left singular vectors of the feature matrix Φ) and $D_{mm} = \sqrt{\lambda_m(n-1)}$. But note that the eigendecomposition of $\mathbf{K}$ and these projections do not require explicit feature transformations - thus, all the computation is happening in the dual representation and $\varphi(x_i)$ need not be computed, only the kernel matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k(x_i, x_j)$. The kernel formalism also allows us to compute the projection $v_m^\top \varphi(\tilde{x})$ of a new (previously unseen) data vector $\tilde{x} \in \mathbb{R}^p$ to the m-th kernel principal component using

$$\left(\sum_{i=1}^n a_{mi}\varphi(x_i)\right)^\top \varphi(\tilde{x}) = \sum_{i=1}^n a_{mi}k(x_i, \tilde{x}) = a_m^\top \mathbf{k}_{\tilde{x}},$$

where $\mathbf{k}_{\tilde{x}} = [k(x_1, \tilde{x}), \ldots, k(x_n, \tilde{x})]^\top$, so again no explicit feature transformations are needed.

Recall that the above all assumes that the features are **centred**, i.e. that $\frac{1}{n}\sum_{i=1}^n \varphi(x_i) = 0$, but if we are just given a kernel function $k(x, x')$, there is no reason to believe that the features would be centred. Fortunately, it is straightforward to transform *any* kernel matrix into a centred form as shown in the examples sheet.

4.7 Representation of probabilities in RKHS

We have seen that kernel methods effectively work on implicit representations of individual data points, via the canonical feature map $\varphi\colon x \mapsto k(\cdot, x)$, such that every data point is represented as a point in the RKHS $\mathcal{H}_k$. One can similarly represent probability distributions P in the RKHSs by considering the **kernel mean embedding**

$$P \mapsto \mu_k(P) = \mathbb{E}_{X\sim P} k(\cdot, X) \in \mathcal{H}_k.$$

This is a potentially infinite-dimensional representation of P akin to a characteristic function of a probability distribution.

The kernel mean embedding converts functions $f \in \mathcal{H}_k$ to their mean with respect to P when we take the inner product, that is:

$$\langle \mu_k(P), f\rangle_{\mathcal{H}_k} = \langle \mathbb{E}_{X\sim P} k(\cdot, X), f\rangle_{\mathcal{H}_k} = \mathbb{E}_{X\sim P}\langle k(\cdot, X), f\rangle_{\mathcal{H}_k} = \mathbb{E}_{X\sim P} f(X), \quad \forall f \in \mathcal{H}_k.$$

It thus exists whenever $f \mapsto \mathbb{E}_{X\sim P} f(X)$ is a bounded functional. Note that this is always true if the kernel function itself is bounded, i.e. $k(x, x') \leq M < \infty\ \forall x, x'$. Namely, by Cauchy-Schwarz

$$\mathbb{E}_{X\sim P} f(X) = \mathbb{E}_{X\sim P}\langle f, k(\cdot, X)\rangle \leq \|f\|_{\mathcal{H}_k} \mathbb{E}_{X\sim P}\|k(\cdot, X)\|_{\mathcal{H}_k} \leq \sqrt{M}\|f\|_{\mathcal{H}_k}$$

Of particular note is the case where f is itself a kernel mean embedding: inner products between kernel mean embeddings can be computed as

$$\langle \mu_k(P), \mu_k(Q)\rangle_{\mathcal{H}_k} = \mathbb{E}_{X\sim P}\mathbb{E}_{Y\sim Q} k(X, Y).$$

4.7.1 Maximum mean discrepancy

We can easily estimate the (squared) distances between probability measures induced by this RKHS representation since they correspond to simple expectations. Such distances

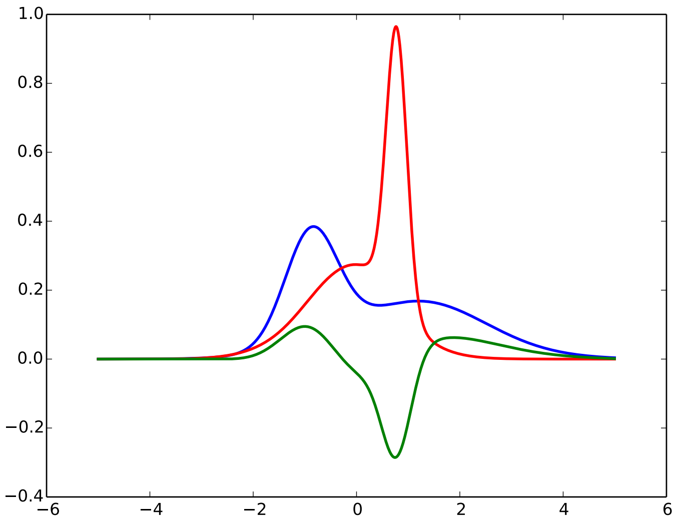


Figure 4.1: Witness function for a difference between two univariate densities

are called the **maximum mean discrepancy** (MMD):

$$\begin{aligned}\mathrm{MMD}_k^2(P,Q) &= \|\mu_k(P)-\mu_k(Q)\|^2_{\mathcal{H}_k} \qquad (4.10)\\ &= \mathbb{E}_{X,X'\overset{iid}{\sim}P}k(X,X')+\mathbb{E}_{Y,Y'\overset{iid}{\sim}Q}k(Y,Y')-2\mathbb{E}_{X\sim P,Y\sim Q}k(X,Y),\end{aligned}$$

where X and X' are independent random samples from P, and similarly Y and Y' are independent samples from Q.

The name MMD comes from the following interpretation: it can also be written as the largest discrepancy between expectations of the unit norm RKHS functions with respect to two distributions (problem sheet):

$$\mathrm{MMD}_k(P,Q) = \sup_{f\in\mathcal{H}_k:\,\|f\|_{\mathcal{H}_k}\le 1} \left|\mathbb{E}_{X\sim P}f(X)-\mathbb{E}_{Y\sim Q}f(Y)\right|.$$

As a consequence, the function f where the supremum is attained, which can be shown to be proportional to the difference between embeddings: $\mu_k(P)-\mu_k(Q)$ is known as the **witness function** for the difference between distributions P and Q. An example of such a witness function is shown in green in Fig. 4.1, where P and Q correspond to distributions on the real line whose densities are drawn in blue and red, respectively. We can see that the witness function is large in amplitude where the difference between two densities is large and it can thus be used to discover regions in the space where two distributions disagree.

For a large class of kernels, including Gaussian, Matern family and rational quadratic, MMD is a proper metric on probability distributions, in the sense that $MMD_k(P,Q)=0$ implies $P=Q$. Such kernels are called **characteristic**. MMD is a popular probability metric, used for nonparametric hypothesis testing [22] and in various machine learning applications, e.g. training deep generative models [18]. Given two samples $\{x_i\}_{i=1}^{n_x}\sim P$

and $\{y_i\}_{i=1}^{n_y} \sim Q$, a simple unbiased estimator of the squared MMD in (4.10) is given by

$$\begin{aligned}\widehat{\mathrm{MMD}}_k^2\left(P,Q\right) = {} & \frac{1}{n_x\left(n_x-1\right)}\sum_{i\neq j} k\left(x_i,x_j\right) + \frac{1}{n_y\left(n_y-1\right)}\sum_{i\neq j} k\left(y_i,y_j\right) \\ & - \frac{2}{n_x n_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} k\left(x_i,y_j\right),\end{aligned} \tag{4.11}$$

which can be interpreted as the difference between within-sample average similarity (self-similarity excluded) and the between-sample average similarity.

4.7.2 Hilbert-Schmidt independence criterion

Another use of kernel embeddings is in measuring dependence between random variables taking values in some generic domains (e.g. random vectors, strings, or graphs). Recall that for any kernels $k_{\mathcal{X}}$ and $k_{\mathcal{Y}}$ on the respective domains $\mathcal{X}$ and $\mathcal{Y}$, we can define $k = k_{\mathcal{X}} \otimes k_{\mathcal{Y}}$, given by

$$k\left((x,y),\left(x',y'\right)\right) = k_{\mathcal{X}}(x,x')k_{\mathcal{Y}}(y,y') \tag{4.12}$$

which is a valid kernel on the product domain $\mathcal{X}\times\mathcal{Y}$ by Lemma 11. The tensor notation signifies that the canonical feature map of k is $(x,y)\mapsto k_{\mathcal{X}}(\cdot,x)\otimes k_{\mathcal{Y}}(\cdot,y)$. Here the feature of pair (x,y), $\varphi(x,y) = k_{\mathcal{X}}(\cdot,x)\otimes k_{\mathcal{Y}}(\cdot,y)$ is understood as a function on $\mathcal{X}\times\mathcal{Y}$, i.e. $(\varphi(x,y))(x',y') = k_{\mathcal{X}}(x',x)k_{\mathcal{Y}}(y',y)$. We are now ready to define an RKHS-based measure of dependence between random variables X and Y.

Definition 14. Let X and Y be random variables on domains $\mathcal{X}$ and $\mathcal{Y}$ (non-empty topological spaces). Let $k_{\mathcal{X}}$ and $k_{\mathcal{Y}}$ be kernels on $\mathcal{X}$ and $\mathcal{Y}$ respectively. The **Hilbert-Schmidt independence criterion** (HSIC) $\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}(X,Y)$ of X and Y is the squared MMD between the joint measure P_{XY} and the product of marginals P_XP_Y, computed with the product kernel $k = k_{\mathcal{X}}\otimes k_{\mathcal{Y}}$, i.e.,

$$\begin{aligned}\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}(X,Y) &= \left\|\mu_k\left(P_{XY}\right) - \mu_k\left(P_XP_Y\right)\right\|^2_{\mathcal{H}_k} \\ &= \left\|\mathbb{E}_{XY}[k_{\mathcal{X}}(.,X)\otimes k_{\mathcal{Y}}(.,Y)] - \mathbb{E}_X k_{\mathcal{X}}(.,X)\otimes\mathbb{E}_Y k_{\mathcal{Y}}(.,Y)\right\|^2_{\mathcal{H}_k}.\end{aligned}$$

A sufficient condition for the HSIC to be well defined is that both kernels $k_{\mathcal{X}}$ and $k_{\mathcal{Y}}$ are bounded. The larger it is, the more dependent the variables X and Y are. It is straightforward to see that $\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}(X,Y) = 0$ if X and Y are independent of one another, while for many common choices of kernel the inverse result also holds (i.e. $\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}(X,Y) = 0$ implies independence).

We can garner some intuition about the HSIC by noting that it is analogous to the squared Frobenius norm (i.e. the sum of the squares of all the individual elements) of the cross–covariance matrix between (finite feature mappings of) X and Y. That is, its finite dimensional equivalent would be to a) map X and Y to new feature vectors $\varphi_{\mathcal{X}}(X)$ and $\varphi_{\mathcal{Y}}(Y)$, b) calculate the pairwise covariances between each element of $\varphi_{\mathcal{X}}(X)$ and each element of $\varphi_{\mathcal{X}}(X)$, and c) take sum of the squares of all of these pairwise covariances.

Estimating the HSIC can be achieved via the following process (the derivation of this follows later):

1. Draw m independent samples $\{(x_i, y_i)\}_{i=1}^m$ from the joint distribution P_{XY};
2. Construct the kernel matrices $K_{ij} = k_{\mathcal{X}}(x_i, x_j)$ and $L_{ij} = k_{\mathcal{Y}}(y_i, y_j)$;
3. Centre the kernel matrices by computing $\widetilde{K} = HKH$ and $\widetilde{L} = HLH$ where $H = I - \frac{1}{m}1$ (i.e. $H_{ij} = \mathbb{I}(i = j) - \frac{1}{m}$);
4. Construct the estimate (noting $\widetilde{K}$ and $\widetilde{L}$ are symmetric)

$$\widehat{\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}}(X, Y) = \frac{1}{m^2}\operatorname{tr}\left(\widetilde{K}\widetilde{L}\right) = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\widetilde{K}_{ij}\widetilde{L}_{ij}. \tag{4.13}$$

Full Details on HSIC

The name of HSIC comes from the operator view of the RKHS $\mathcal{H}_{k_{\mathcal{X}}\otimes k_{\mathcal{Y}}}$. Namely, by repeated use of the reproducing property, it can be verified (see the 2020 SC4 exam) that the difference between embeddings $\mu_k(P_{XY}) - \mu_k(P_X P_Y)$ can be identified with the **cross-covariance operator** $C_{XY} : \mathcal{H}_{k_{\mathcal{Y}}} \to \mathcal{H}_{k_{\mathcal{X}}}$ for which

$$\langle f, C_{XY} g\rangle_{\mathcal{H}_{k_{\mathcal{X}}}} = \operatorname{Cov}\left[f(X)g(Y)\right], \quad \forall f \in \mathcal{H}_{k_{\mathcal{X}}}, g \in \mathcal{H}_{k_{\mathcal{Y}}}.$$

Note that this is analogous to the finite-dimensional property $f^\top C_{XY} g = \operatorname{Cov}\left[f^\top X, g^\top Y\right]$, where X and Y are random vectors and C_{XY} is their cross-covariance matrix, i.e. $[C_{XY}]_{ij} = \operatorname{Cov}\left[X^{(i)}, Y^{(j)}\right]$. HSIC is then simply the squared Hilbert-Schmidt norm $\|C_{XY}\|_{HS}^2$ of this operator.

To obtain an estimator of the HSIC, we first express it in terms of the expectations of kernels. Starting from the definition, and expanding the Hilbert space norm into inner products:

$$\begin{aligned}
\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}(X, Y) &= \|\mathbb{E}_{XY}\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right) \\
&\quad - \mathbb{E}_X\left(k(\cdot, X)\right) \otimes \mathbb{E}_Y\left(k(\cdot, Y)\right)\|_{\mathcal{H}_k}^2 \\
&= \langle \mathbb{E}_{XY}\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right), \mathbb{E}_{XY}\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right)\rangle_{\mathcal{H}_k} \\
&\quad + \langle \mathbb{E}_X\left(\mathbb{E}_Y\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right)\right), \mathbb{E}_X\left(\mathbb{E}_Y\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right)\right)\rangle_{\mathcal{H}_k} \\
&\quad - 2\langle \mathbb{E}_{XY}\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right), \mathbb{E}_X\left(\mathbb{E}_Y\left(k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y)\right)\right)\rangle_{\mathcal{H}_k} \\
&= \mathbb{E}_{XY}\left(\mathbb{E}_{X'Y'}\left(\langle k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y), k_{\mathcal{X}}(\cdot, X') \otimes k_{\mathcal{Y}}(\cdot, Y')\rangle_{\mathcal{H}_k}\right)\right) \\
&\quad + \mathbb{E}_{XX'}\left(\mathbb{E}_{YY'}\left(\langle k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y), k_{\mathcal{X}}(\cdot, X') \otimes k_{\mathcal{Y}}(\cdot, Y')\rangle_{\mathcal{H}_k}\right)\right) \\
&\quad - 2\mathbb{E}_{XY}\left(\mathbb{E}_{X'}\left(\mathbb{E}_{Y'}\left(\langle k_{\mathcal{X}}(\cdot, X) \otimes k_{\mathcal{Y}}(\cdot, Y), k_{\mathcal{X}}(\cdot, X') \otimes k_{\mathcal{Y}}(\cdot, Y')\rangle_{\mathcal{H}_k}\right)\right)\right) \\
&= \mathbb{E}_{XY}\left(\mathbb{E}_{X'Y'}\left(k_{\mathcal{X}}(X, X')\, k_{\mathcal{Y}}(Y, Y')\right)\right) \\
&\quad + \mathbb{E}_{XX'}\left(k_{\mathcal{X}}(X, X')\right)\mathbb{E}_{YY'}\left(k_{\mathcal{Y}}(Y, Y')\right) \qquad (4.14) \\
&\quad - 2\mathbb{E}_{XY}\left(\mathbb{E}_{X'}\left(k_{\mathcal{X}}(X, X')\right)\mathbb{E}_{Y''}\left(k_{\mathcal{Y}}(Y, Y'')\right)\right)
\end{aligned}$$

Here the first expectation is taken over two independent copies (X,Y), $(X',Y') \sim P_{XY}$, the second over two independent $X,X' \sim P_X$ and two independent $Y,Y' \sim P_Y$ and the third over a pair $(X,Y) \sim P_{XY}$ sampled from the joint and an independent pair $X' \sim P_X$, $Y'' \sim P_Y$. Now, given a sample $Z = \{z_i\}_{i=1}^m = \{(x_i,y_i)\}_{i=1}^m$, where each $(x_i,y_i) \in \mathcal{X} \times \mathcal{Y}$, we can derive an estimator of the HSIC by estimating each of the three terms in the expansion.

Denote for convenience $k_{ij} = k_{\mathcal{X}}(x_i,x_j)$ and $l_{ij} = k_{\mathcal{Y}}(y_i,y_j)$ for $i,j \in \{1,2,...,m\}$ and define the kernel matrices $K = (k_{ij})_{i,j=1}^m$ and $L = (l_{ij})_{i,j=1}^m$ (recall that they are symmetric and positive-definite). Following, we estimate:

$$\widehat{\text{first term}} = \frac{1}{m^2}\sum_{i=1}^m\sum_{j=1}^m k_{ij}l_{ij} = \frac{1}{m^2}\operatorname{tr}(KL)$$

$$\begin{aligned}\widehat{\text{second term}} &= \frac{1}{m^4}\left(\sum_{i=1}^m\sum_{j=1}^m k_{ij}\right)\left(\sum_{i=1}^m\sum_{j=1}^m l_{ij}\right)\\ &= \frac{1}{m^4}\left(1_m^T K 1_m\right)\left(1_m^T L 1_m\right)\end{aligned}$$

$$\begin{aligned}\widehat{\text{third term}} &= \frac{1}{m^3}\sum_{i=1}^m\sum_{j=1}^m\sum_{q=1}^m k_{ij}l_{iq} = \frac{1}{m^3}1_m^T K L 1_m\\ &= \frac{1}{m^3}1_m^T L K 1_m\end{aligned}$$

Here 1_m is the vector with m entries equal to 1. Therefore an estimator for the HSIC can be written as:

$$\begin{aligned}\widehat{\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}}(X,Y) &= \frac{1}{m^2}\left(\operatorname{tr}(KL) - \frac{2}{m}1_m^T KL1_m + \frac{1}{m^2}\left(1_m^T K1_m\right)\left(1_m^T L1_m\right)\right)\\ &= \frac{1}{m^2}\left(\operatorname{tr}(KL) - \frac{1}{m}\operatorname{tr}\left(1_m1_m^T KL\right) - \frac{1}{m}\operatorname{tr}\left(K1_m1_m^T L\right)\right.\\ &\quad\left.+\frac{1}{m^2}\operatorname{tr}\left(1_m1_m^T K1_m1_m^T L\right)\right)\\ &= \frac{1}{m^2}\operatorname{tr}\left(\left(I - \frac{1}{m}1_m1_m^T\right)K\left(I - \frac{1}{m}1_m1_m^T\right)L\right)\\ &= \frac{1}{m^2}\operatorname{tr}(KHLH).\end{aligned}$$

Here we used that $\operatorname{tr}(AB) = \operatorname{tr}(BA)$, $\operatorname{tr}(A) = \operatorname{tr}\left(A^T\right)$ and that any real number is equal to its own trace. We also defined

$$H := I - \frac{1}{m}1_m1_m^T$$

which is the **centering matrix**. Namely, if A is any $m \times m$-matrix, AH centers the rows of A and HA centers the columns of A. Note also that H is symmetric and idempotent, i.e. $H^2 = H$. Hence, $\operatorname{tr}((HKH)(HLH)) = \operatorname{tr}(H(KHLH)) = \operatorname{tr}(KHLHH) = \operatorname{tr}(KHLH)$.

Recall that the kernel is an inner product between features of the inputs and that inner products are bilinear. Therefore, the matrices $\widetilde{K} = HKH$ and $\widetilde{L} = HLH$ are the kernel matrices for the variables centered in feature space. We therefore arrive at the expression for the estimator:

$$\widehat{\Xi_{k_{\mathcal{X}},k_{\mathcal{Y}}}}(X,Y) = \frac{1}{m^2}\operatorname{tr}\left(\widetilde{K}\widetilde{L}\right), \tag{4.15}$$

which has an intuitive explanation of how it measures the dependence between X and Y. Namely, the function $(A,B) \to \operatorname{tr}\left(A^T B\right)$ is an inner product on the vector space of real $m \times m$ matrices. Therefore, our estimate measures the similarity between the (centered) kernel matrices, which in turn measure the "similarity patterns" between the individual observations. If there is some dependence between the X and Y, we also expect that the kernel matrices will have a similar structure and hence the inner product between them (and hence our HSIC estimator in (4.15)), will be larger.

5 Bayesian Machine Learning

5.1 The Bayesian Paradigm

At its core, the Bayesian paradigm is simple, intuitive, and compelling: for any task involving learning from data, we start with some prior knowledge and then update that knowledge to incorporate information from the data. This process is known as **Bayesian inference**. To give an example, consider the process of interviewing candidates for a job. Before we interview each candidate, we have some intuitions about how successful they will be in the advertised role, e.g. from their application form. During the interview we receive more information about their potential competency and we combine this with our existing intuitions to get an updated belief of how successful they will be.

To be more precise, imagine we are trying to reason about some variables or parameters θ. We can encode our initial belief as probabilities for different possible instances of θ, this is known as a **prior** $p(\theta)$.[1] Given observed data $\mathcal{D}$, we can characterize how likely different values of θ are to have given rise to that data using a **likelihood function** $p(\mathcal{D}|\theta)$. These can then be combined to give a **posterior**, $p(\theta|\mathcal{D})$, that represents our updated belief about θ once the information from the data has been incorporated by using **Bayes' rule**:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int_\Theta p(\mathcal{D}|\theta)p(\theta)d\theta} = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}. \tag{5.1}$$

Here the denominator, $p(\mathcal{D})$, is a normalization constant known as the **marginal likelihood** or **model evidence** and is necessary to ensure $p(\theta|\mathcal{D})$ is a valid probability distribution (or probability density for continuous problems). One can, therefore, think of Bayes' rule in the even simpler form of the posterior being proportional to the prior times the likelihood:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta). \tag{5.2}$$

For such a fundamental theorem, Bayes' rule has a remarkably simple derivation, following directly from the product rule of probability.

Another way of thinking about the Bayesian paradigm is in terms of constructing a generative model corresponding to the joint distribution on parameters and possible data $p(\theta, \mathcal{D})$, then **conditioning** that model by fixing $\mathcal{D}$ to the data we actually observe.

[1] To avoid getting bogged down with tangential technicalities, we will be deliberately a little sloppy about distinguishing between probabilities and probability densities when dealing with Bayesian contexts.

This provides an updated distribution on the parameters $p(\theta|\mathcal{D})$ that incorporates the information from the data.

A key feature of Bayes' rule is that it can be used in a self-similar fashion where the posterior from one task becomes the prior when the model is updated with more data, i.e.

$$p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\theta|\mathcal{D}_1)}{p(\mathcal{D}_2|\mathcal{D}_1)} = \frac{p(\mathcal{D}_2|\theta, \mathcal{D}_1)p(\mathcal{D}_1|\theta)p(\theta)}{p(\mathcal{D}_2|\mathcal{D}_1)p(\mathcal{D}_1)}. \tag{5.3}$$

As a consequence, there is something quintessentially human about the Bayesian paradigm: we learn from our experiences by updating our beliefs after making observations.

All the questions about model parameters θ can be addressed based on the posterior. We can, for example, consider

- *Posterior mode*: $\widehat{\theta}^{\text{MAP}} = \arg\max_{\theta\in\Theta} p(\theta|\mathcal{D})$ (maximum a posteriori).
- *Posterior mean*: $\widehat{\theta}^{\text{mean}} = \mathbb{E}\left[\theta|\mathcal{D}\right]$.
- *Posterior variance*: $\text{Var}[\theta|\mathcal{D}]$.
- *Posterior expectations of functions of parameters*: $\mathbb{E}\left[g\left(\theta\right)|\mathcal{D}\right]$ for some $g : \Theta \to \mathbb{R}^s$.

5.1.1 Example: Predicting the Outcome of a Weighted Coin

To give a concrete example of a Bayesian analysis, consider estimating the probability of getting a heads from a weighted coin. Let's call this weighting $\theta \in [0, 1]$ such that the probability of getting a heads (H) when flipping the coin is $p(y = H|\theta) = \theta$ where y is the outcome of the flip. This will be our likelihood function, corresponding to a **Bernoulli** distribution, noting that the probability of getting a tails (T) is $p(y = T|\theta) = 1 - \theta$. Before seeing the coin being flipped we have some prior belief about its weighting. We can, therefore, define a prior $p(\theta)$, for which we will take the **Beta** distribution

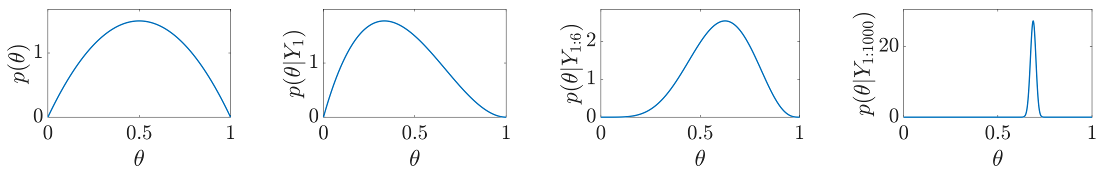
$$p(\theta) = \text{Beta}\left(\theta; \alpha, \beta\right) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \tag{5.4}$$

where $\Gamma(\cdot)$ is the gamma function and we will set $\alpha = \beta = 2$. A plot for this prior is shown in Figure 5.1a where we see that under our prior then it is more probable that θ is close to 0.5 than the extremes 0 and 1.

We now flip the coin and get a tails (T). We can calculate the posterior using Bayes' rule

$$p(\theta|y_1 = T) = \frac{p(\theta)p(y_1 = T|\theta)}{\int p(\theta)p(y_1 = T|\theta)d\theta} = \frac{\theta(1-\theta)^2}{\int \theta(1-\theta)^2 d\theta} = \text{Beta}\left(\theta; 2, 3\right). \tag{5.5}$$

Here we have used the fact that a Beta prior is **conjugate** to a Bernoulli likelihood to give an analytic solution. Conjugacy means that the prior-likelihood combination gives a posterior that is of the same form as the prior distribution. More generally, for a prior of $\text{Beta}(\theta; \alpha, \beta)$ then the posterior will be $\text{Beta}(\theta; \alpha + 1, \beta)$ if we observe a heads and



(a) Prior (b) Posterior 1 flip (c) Posterior 6 flips (d) Posterior 1000 flips

Figure 5.1: Prior and posteriors for coin flip example after different numbers of observations.

Beta$(\theta; \alpha, \beta+1)$ if we observe a tails. Figure 5.1b shows that our posterior incorporates the information from the prior and the observed data. For example, our observation means that it becomes more probable that $\theta < 0.5$. The posterior also reflects the fact that we are still uncertain about the value of θ, it is not simply the empirical average of our observations which would give $\theta = 0$.

If we now flip the coin again, our previous posterior (5.5) becomes our prior and we can incorporate the new observations in the same way. Through our previous conjugacy result, then if we observe n_H heads and n_T tails and our prior is Beta$(\theta; \alpha, \beta)$ then our posterior is Beta$(\theta; \alpha + n_H, \beta + n_T)$. If our sequence of new observations is $HTHHH$ then our new posterior is

$$p(\theta|y_1, \dots, y_6) = \frac{p(y_2, \dots, y_6|\theta)p(\theta|y_1)}{\int p(y_2, \dots, y_6|\theta)p(\theta|y_1)d\theta} = \text{Beta}\left(\theta; 6, 4\right), \tag{5.6}$$

which is shown in Figure 5.1c. We see now that our belief for the probability of heads has shifted higher and that the uncertainty has reduced because of the increased number of observations. After seeing a total of 1000 observations as shown in Figure 5.1d, we find that the posterior has predominantly collapsed down to a small range of θ.

We can extend this example to categorical observations, i.e. $y_i \in \{1, \dots, K\}$, by modelling them with the probability mass function $\pi = (\pi_1, \dots, \pi_K)$ and using a categorical likelihood

$$p(\mathcal{D}|\pi) = \prod_{i=1}^{n} \pi_{y_i} = \prod_{k=1}^{K} \pi_k^{n_k}$$

with $n_k = \sum_{i=1}^{n} \mathbb{I}(y_i = k)$ and $\pi_k > 0$, $\sum_{k=1}^{K} \pi_k = 1$. The conjugate prior on π is the **Dirichlet distribution** $\text{Dir}(\alpha_1, \dots, \alpha_K)$ with parameters $\alpha_k > 0$, and density

$$p(\pi) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}$$

on the probability simplex $\{\pi : \pi_k > 0, \sum_{k=1}^{K} \pi_k = 1\}$. Since

$$p\left(\pi|\mathcal{D}\right) \quad \propto \quad \prod_{k=1}^{K} \pi_k^{n_k + \alpha_k - 1},$$

the posterior is also Dirichlet $\mathrm{Dir}(\alpha_1+n_1,\dots,\alpha_K+n_K)$. The posterior mean is given by

$$\widehat{\pi}_k^{\text{mean}} = \frac{\alpha_k + n_k}{\sum_{j=1}^K \alpha_j + n_j}.$$

5.2 Using the Posterior

In some cases, the posterior is all we care about. For example, in many statistical applications θ is some physical parameter of interest and our core aim is to learn about it. Often though, the posterior will be a stepping stone to some ultimate task of interest.

One common task is making decisions; the Bayesian paradigm is rooted in decision theory. It is effectively the language of **epistemic uncertainty**—that is uncertainty originating from lack of information. As such, we can use it as a basis for making decisions in the presence of incomplete information. As we will show later, in a Bayesian decision framework we first calculate the posterior and then use this to make a decision by choosing the decision which has the lowest expected loss under this posterior.

Another common task, particularly in the context of Bayesian machine learning, is to use the posterior to make predictions for unseen data. For this, we use the so-called **posterior predictive distribution**. Denoting the new data as $\mathcal{D}^*$, this is calculated by first introducing a predictive model for new data given θ, $p(\mathcal{D}^*|\theta)$,[2] then taking the expectation of this over possible parameter values as dictated by posterior as follows

$$p(\mathcal{D}^*|\mathcal{D}) = \mathbb{E}_{p(\theta|\mathcal{D})}[p(\mathcal{D}^*|\theta)] = \int p(\mathcal{D}^*|\theta)p(\theta|\mathcal{D})d\theta. \tag{5.7}$$

Though the exact form of $p(\mathcal{D}^*|\theta)$ can vary depending on the context, it is equivalent to a likelihood term for the new data: if we were to observe $\mathcal{D}^*$ rather than predicting it, this is exactly the likelihood term we would use to update our posterior as per (5.3). Note an important assumption this setup is making: that all the information we want to use for predicting from our model is encapsulated in θ. This can be problematic if θ is no sufficiently rich to fully encapsulate the data.

In case of supervised learning, we are interested typically only in the conditional distribution of outputs given inputs, and not in the marginal distribution over inputs. We can account for this by using a conditional model $p(y|x,\theta)$ and then taking the following posterior predictive distribution

$$p(y|x,\mathcal{D}) = \mathbb{E}_{p(\theta|\mathcal{D})}[p(y|x,\theta)] = \int p(y|x,\theta)p(\theta|\mathcal{D})d\theta.$$

Returning to our coin flipping example, we see that we can make predictions using the

[2]In principle, one can include the data directly in this predictive model, i.e. instead define $p(\mathcal{D}^*|\theta,\mathcal{D})$, but this is rarely done in practice as it would imply that our model is failing to encapsulate aspects of the data.

posterior predictive distribution as follows

$$\begin{aligned} p(y_{N+1} = H|y_{1:N}) &= \int p(y_{N+1} = H|\theta)p(\theta|y_{1:N})d\theta \\ &= \int \theta \text{ Beta}(\theta; \alpha + n_H, \beta + n_T)d\theta \\ &= \frac{\alpha + n_H}{\alpha + n_H + \beta + n_T} \end{aligned}$$

where we have used the known result for the mean of the Beta distribution. The role of the parameters α and β in our prior now become apparent – they take on the role of pseudo-observations. Our prediction is in line with the empirical average from seeing $\alpha + n_H$ heads and $\beta + n_T$ tails. The larger $\alpha + \beta$ is then the strong our prior compared to the observations, while we can skew towards heads or tails being more likely by changing the relative values of α and β.

5.2.1 Example: Bayesian treatment of naïve Bayes classifer.

Consider a K-class classification problem with binary input vectors, i.e. $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, $x_i \in \{0,1\}^p$ and $y_i \in \{1, \ldots, K\}$. A ***naïve Bayes*** classifier uses the following model:

$$p(y_i = k|\theta) = \pi_k, \quad p(x_i|y_i = k, \theta) = \prod_{j=1}^p \phi_{kj}^{x_i^{(j)}} (1 - \phi_{kj})^{1 - x_i^{(j)}},$$

i.e. it assumes that given the class labels, individual dimensions in input vectors are *independent*. The parameters of the model are collated into $\theta = ((\pi_k), (\phi_{kj}))$. It is often used in text classification where data items correspond to documents and $x_i^{(j)}$ indicates whether word j from a list of p words has appeared in document i. Class labels correspond to e.g. topics of the documents. Despite the name, naïve Bayes is often treated in a frequentist way, i.e. using maximum likelihood estimation of parameters. If we set $n_k = \sum_{i=1}^n \mathbb{I}\{y_i = k\}$, $n_{kj} = \sum_{i=1}^n \mathbb{I}(y_i = k, x_i^{(j)} = 1)$, the MLE can be written as

$$\hat{\pi}_k = \frac{n_k}{n}, \qquad \hat{\phi}_{kj} = \frac{\sum_{i:y_i=k} x_i^{(j)}}{n_k} = \frac{n_{kj}}{n_k}.$$

But the MLEs can be problematic in some cases. For example, if the ℓ-th word did not appear in any documents labelled as class k ($n_{kl} = 0$), then $\hat{\phi}_{k\ell} = 0$. But if we then wish to compute the predictive probability once for a new document $\tilde{x}$ which contains ℓ-th word, we have:

$$p(\tilde{y} = k|\tilde{x} \text{ with } \ell\text{-th entry equal to } 1, \hat{\theta}) \propto \hat{\pi}_k \prod_{j=1}^p \left(\hat{\phi}_{kj}\right)^{\tilde{x}^{(j)}} \left(1 - \hat{\phi}_{kj}\right)^{1 - \tilde{x}^{(j)}} = 0,$$

since $\hat{\phi}_{k\ell} = 0$. This means that we will never attribute a new document containing word ℓ to class k (regardless of what other words in it may be!). Moreover, probability of a document under all classes can be 0 by the same reasoning.

Let us consider a Bayesian approach to the same model. We can write the likelihood as

$$\begin{aligned} p(\mathcal{D}|\theta) = \prod_{i=1}^{n} p(x_i, y_i|\theta) &= \prod_{i=1}^{n}\prod_{k=1}^{K}\left(\pi_k \prod_{j=1}^{p} \phi_{kj}^{x_i^{(j)}} (1-\phi_{kj})^{1-x_i^{(j)}}\right)^{\mathbb{I}(y_i=k)} \\ &= \prod_{k=1}^{K} \pi_k^{n_k} \prod_{j=1}^{p} \phi_{kj}^{n_{kj}} (1-\phi_{kj})^{n_k - n_{kj}}. \end{aligned}$$

For a conjugate prior, we can use $\mathrm{Dir}((\alpha_k)_{k=1}^K)$ for π, and $\mathrm{Beta}(a, b)$ for ϕ_{kj} independently. Now, because the likelihood factorises, the posterior distribution over π and (ϕ_{kj}) also factorises, and posterior for π is $\mathrm{Dir}((\alpha_k+n_k)_{k=1}^K)$, and for ϕ_{kj} is $\mathrm{Beta}(a+n_{kj}, b+n_k-n_{kj})$. If we want to predict a label $\tilde{y}$ for a new document $\tilde{x}$, we obtain

$$p(\tilde{x}, \tilde{y} = k|\mathcal{D}) = p(\tilde{y} = k|\mathcal{D}) p(\tilde{x}|\tilde{y} = k, \mathcal{D})$$

with

$$\begin{aligned} p(\tilde{y} = k|\mathcal{D}) &= \frac{\alpha_k + n_k}{\sum_{l=1}^K \alpha_l + n} \\ p(\tilde{x}^{(j)} = 1|\tilde{y} = k, \mathcal{D}) &= \frac{a + n_{kj}}{a + b + n_k} \end{aligned}$$

and the predicted class is

$$\begin{aligned} p(\tilde{y} = k|\tilde{x}, \mathcal{D}) &= \frac{p(\tilde{y} = k|\mathcal{D}) p(\tilde{x}|\tilde{y} = k, \mathcal{D})}{p(\tilde{x}|\mathcal{D})} \\ &\propto \frac{\alpha_k + n_k}{\sum_{l=1}^K \alpha_l + n} \prod_{j=1}^{p} \left(\frac{a + n_{kj}}{a + b + n_k}\right)^{\tilde{x}^{(j)}} \left(\frac{b + n_k - n_{kj}}{a + b + n_k}\right)^{1-\tilde{x}^{(j)}}. \end{aligned}$$

Compared to the MLE plug-in predictions, pseudocounts help to "regularise" probabilities away from the extreme values.

5.3 Bayesian Decision Theory

Suppose we have a supervised learning setting with dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$. Given a new input x, a system is required to make a prediction or take an action, say $f(x)$, which incurs a loss $L(y, f(x), x)$ if the output is y. Empirical risk minimisation (ERM) directly parameterises and optimises for $f_\theta(x)$ as a function of x to reduce empirical risk. Implicitly, this means that we are treating θ as a deterministic parameter to be optimized, but we avoid having to assume a form for the distribution $Y|X$, relying on samples of this instead by using the dataset. We can also think about $\mathcal{D}$ as a random variable here: we desire to minimize the true risk which is an expectation over $\mathcal{D}$, but by necessity we only have a single sample of it. Consequently, we directly introduce regularization to the ERM to try and account for this and prevent overfitting.

The Bayesian approach to decision theory differs from this approach: it instead treats the model parameters θ as a random variable to be marginalized out, but considers the data $\mathcal{D}$ to be deterministic and absolute. As such, there is no concept of hypothetical alternative data that we *could* have seen, with regularization instead dealt with implicitly by the prior. When making predictions, we use our predictive distribution, $p(y|x,\mathcal{D}) = \mathbb{E}_{p(\theta|\mathcal{D})}[p(y|x,\theta)]$, to model the distribution of possible outputs given a new input. If required to make a decision, action, or point estimate prediction $f(x)$, then we *pointwise* define $f(x)$ to be whatever prediction/action will minimize the expected loss for this x if $Y \sim p(y|x,\mathcal{D})$. That is we choose the prediction that minimizes the **posterior expected loss** as follows

$$\begin{aligned} f(x) :=& \operatorname{argmin}_{\hat{y}} \mathbb{E}_{Y\sim p(y|x,\mathcal{D})}[L(Y,\hat{y},x)] && (5.8)\\ =& \operatorname{argmin}_{\hat{y}} \iint L(y,\hat{y},x)p(y|x,\theta)p(\theta|\mathcal{D})dyd\theta. && (5.9)\end{aligned}$$

Thus, for example, if the loss is squared loss, $|y-\hat{y}|^2$, then we have that the optimal prediction is the predictive mean, $f(x) = \mathbb{E}_{Y\sim p(y|x,\mathcal{D})}[Y]$, and if the loss is the absolute loss $|y-\hat{y}|$, the optimal prediction is the predictive median. Note though that, in general, we wish to avoid making such point estimate predictions until we are forced to: the posterior predictive distribution $p(y|x,\mathcal{D})$ is our primary predictive model in a supervised Bayesian setting and encapsulates our full set of beliefs of possible outcomes; it is thus the preferred entity to return from our algorithm whenever possible.

This Bayesian approach to making point predictions, and more generally decisions, applies equally to unsupervised settings and beyond. If d represents a decision and $L(d,\theta)$ a loss function for making decision d with model parameter θ, then the optimal decision d^* optimises the posterior expected loss:

$$d^* := \operatorname{argmin}_d \mathbb{E}_{\hat{\theta}\sim p(\theta|\mathcal{D})}\left[L(d,\hat{\theta})\right].$$

We can further view this a decision function of the data $d^*(\mathcal{D})$ by applying this optimization on a pointwise basis.

Bayesian decision theory separates out the process of modelling and inference from the process of decision making: our treatment of the model parameters θ has no dependency on the loss; the loss only influences how we make predictions or decisions given our model. As such, computational considerations aside, we can first calculate our posterior without worrying our final predictive/decision making task, safe in the knowledge that our optimum solution for this task will depend on both the data and our model only through this posterior.

Advantages of this framework include that: one can ascertain and critique the model's fit to data prior to making decisions, one can quantify the model's uncertainties, one can propagate all predictive information and uncertainty through the posterior predictive rather than being forced to return point estimates, and one can use the same posterior distribution to derive multiple different decisions or predictions. Disadvantages include: Bayesian inference is typically significantly more computationally expensive and less scalable than ERM, and if approximations are required it is unclear how to calibrate the approximation to the loss at hand. In other words, different approximations emphasise differ aspects of the posterior, and the choice of approximations should reflect the aspects of the posterior that are important for the loss function, see for example [34, 44]. This then breaks the separation between modelling/inference and decision making.

5.4 A Fundamental Assumption

Though not strictly necessary, an assumption made by virtually all Bayesian models is that datapoints are conditionally independent given the parameter values. In other words, if our data is given by $\mathcal{D} = \{x_n\}_{n=1}^N$, then the likelihood factorizes as follows

$$p(\mathcal{D}|\theta) = \prod_{n=1}^{N} p(x_n|\theta). \tag{5.10}$$

There are two main motivations for the assumption, one practical, the other theoretical.

The practical motivation is that if datapoints are not conditionally independent given θ, this indicates our model is ignoring information present in the data. In principle, we want to choose models where our parameters encapsulate as much information that is useful for prediction as possible. If our likelihood model directly uses information from other data, e.g. $p(x_2|\theta, x_1)$, this indicates that there was useful predictive information in x_1 that we are not encapsulating through θ. This in turn indicates we could improve our model by incorporating this information, e.g. by introducing an additional parameter into θ. As such, we can argue that if we are always using the best possible model given our our knowledge of the problem, this model should always satisfy (5.10). Furthermore, writing models in a manner that satisfies (5.10) is often highly convenient when designing models and conducting inference, particularly if the datapoints are identically distributed.

Note here that there is an important distinction between assuming that our likelihood model is correct, and assuming that it encapsulates all the information salient for prediction. For example, presume that θ represent a real–world, but unknown, parameter. It is perfectly possible (ignoring the fact that models are never perfect) that $p(x_n|\theta)$ represents the correct conditional distribution of x_n in the frequentist sense that this it is the distribution created by repeating the experiment with the same value of θ, but that θ also contains very little information about x_n, such that fixing θ has little influence on the produced x_n. Thus it is possible for our model to be well-specified, but also useless. Similarly, it is possible for a model to not be exactly correct, but still be very useful, a somewhat important fact given that all models are inevitably wrong anyway.

The theoretical motivation of assuming (5.10) is based on an important result known as **de Finetti's Theorem** [15]. Though the precise result is beyond the scope of this course, the high-level idea of de Finetti's Theorem is that if we have an infinite sequence of exchangeable random variables (i.e. their probability density is the same if we permute their order), then there exists some parameter which all these variables are *conditionally* independent with respect to. Informally, the upshot of this result in our context is that if the order of data does not matter and our data can be assumed to be a finite sample from an infinitely long sequence (both of which are often the case), then there is exists some set of latent variables (i.e. θ) for which (5.10) holds. Note that this not mean this (5.10) holds for any particular model, it simply shows that a model exists where it does hold. Intuitively, this model is also the most powerful possible model as it ensure θ encapsulates all information relevant to predicting a datapoint that can be provided by other datapoints. This, in turn, provides a motivation for making the assumption.

Though this assumption is so ingrained in Bayesian modeling that its presence is often overlooked (one could even make an argument that this assumption is part of the Bayesian approach itself), it is far from inconsequential. For example, one information–theoretic consequence is that, because all information for prediction must pass through θ, there is a finite amount of information that can be stored in the model if θ is finite dimensional, thereby placing a limit on it predictive power in the limit of large data (see also the Bernstein von-Mises Theorem below). To try and cater for this, some *non-parametric* models are based around using an infinite dimensional θ (we will return to this later).

Unfortunately, it is well beyond the scope of this course to fully explore the consequences of this assumption. Our aim has instead been to simply make you aware that it is an explicit assumption being made by virtually all Bayesian methods, as well as explaining some of the justifications for why. At the end of the day though, it is important to remember that all models are inevitably approximations of the truth for any real–world scenario. As such, it is impossible to avoid making assumptions entirely.

5.5 The Bernstein-Von Mises Theorem [Not Examinable]

One of the important implications of the assumption discussed in the last section is the **Bernstein–von Mises theorem**, which explains the behavior of Bayesian methods in the limit of large data. Assume, for the sake of argument, that (5.10) holds and our likelihood model $p(\mathcal{D}|\theta)$ is correct in the sense that the datapoints x_n are all independent and identically distributed (i.i.d.) according to $p(x_n|\theta^*)$ where θ^* are a (finite) set of "ground truth" parameters and the prior $p(\theta)$ satisfies $p(\theta^*) > 0$. Informally speaking, the Bernstein–von Mises theorem now states that in the limit of large N, the posterior distribution $p(\theta|x_{1:N})$ converges to a normal distribution with mean θ^* and variance of order $O(1/N)$ (i.e. it decreases at a rate $1/N$) [17, 19]. It further transpires that when no such θ^* exists (i.e. our model is misspecified), the convergence is instead to the parameters $\hat{\theta}$ which minimize the Kullback-Leibler (KL) divergence (see Chapter 9) to the true data generating distribution. See for example [30] and the references therein for further details.

This is a hugely important result in Bayesian statistics as it demonstrates that, in the (predominantly hypothetical) scenario that our model assumptions are correct, we converge to the true parameters. Because of this, it is sometimes referred to as the *consistency* of Bayesian methods. It also means that the posterior becomes independent of the prior when we are provided with sufficient data: the likelihood always dominates in parametric models if we provide enough data

The Bernstein–von Mises theorem can be both a blessing and a curse. On the one hand, it ensures we reach the correct conclusion with enough data and a model that is powerful enough to encapsulate the true data distribution. On the other hand, it also means that our uncertainty estimates will collapse to zero given enough data even if our model is misspecified and the answer we are collapsing is not correct: even if θ is a real parameter, it is perfectly possible that $\theta^* \neq \theta$ if our likelihood model is not exactly correct. This is closely linked to the fact that Bayesian models fail to capture the **unknown unknowns**, such that their uncertainty estimates are usually overconfident: they fail to account for the fact that the model itself might not be correct. This can be a serious problem when think about the fact that our model is almost invariably an approximation of the truth.

5.6 Bayesian Model Selection

Consider a situation where we do not have one Bayesian model but several. Each model $\mathcal{M}$ has a set of parameters $\theta_\mathcal{M}$, likelihood $p(\mathcal{D}|\theta_\mathcal{M})$, and prior distribution $p(\theta_\mathcal{M})$. Within each model, the posterior distribution is

$$p(\theta_\mathcal{M}|\mathcal{D},\mathcal{M}) = \frac{p(\mathcal{D}|\theta_\mathcal{M},\mathcal{M})p(\theta_\mathcal{M}|\mathcal{M})}{p(\mathcal{D}|\mathcal{M})}$$

where the normalising constant is the marginal probability of the data under model $\mathcal{M}$ (**Bayesian model evidence**):

$$p(\mathcal{D}|\mathcal{M}) = \int_\Theta p(\mathcal{D}|\theta_\mathcal{M},\mathcal{M})p(\theta_\mathcal{M}|\mathcal{M})d\theta$$

In **Bayesian model selection**, one compares models using their **Bayes factors** $\frac{p(\mathcal{D}|\mathcal{M})}{p(\mathcal{D}|\mathcal{M}')}$.

Considering Bayesian model evidence can be interpreted as a Bayesian version of *Occam's Razor*: of two explanations adequate to explain the same set of observations, the simpler should be preferred. Namely, note that the model evidence $p(\mathcal{D}|\mathcal{M})$ is the probability that a set of randomly selected parameter values (under the prior) inside the model would generate dataset $\mathcal{D}$. In that case, models that are *too simple* are unlikely to generate the observed dataset. On the other hand, models that are *too complex* can generate many possible datasets, so again, they are unlikely to generate that particular dataset at random.

5.7 Approximate Bayesian Inference

In all but the simplest models, the posterior distribution $p(\theta|\mathcal{D})$ is intractable and must be estimated. Doing this requires us to perform **approximate Bayesian inference**. At first, this may seem like a straightforward problem: by Bayes' rule we have that $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$ and so we already know the relative probability of any one value of θ compared to another. In practice, this could hardly be further from the truth. Bayesian inference for the general class of graphical models is, in fact, an NP-hard problem [14].

5.7.1 The Challenge of Bayesian Inference

We can break down Bayesian inference into two key challenges: calculating the normalization constant $p(\mathcal{D})$ and providing a useful characterization of the posterior, for example, a set of approximate samples. Interestingly, each of these constitutes a somewhat distinct problem and many inference methods focus only on solving the latter problem. However, this breakdown will still prove useful in illustrating the intuitions about the difficulties presented by Bayesian inference.

The Normalization Constant

Calculating the normalization constant in Bayesian inference is essentially a problem of integration. Our target, $p(\mathcal{D})$, is the expectation of the likelihood under the prior, hence the name *marginal likelihood*. When $p(\mathcal{D})$ is known, the posterior can be evaluated exactly at any possible input point using Bayes' rule directly. When it is unknown, we lack a scaling in the evaluation of any point and so we have no concept of how relatively significant that point is relative to the distribution as a whole. For example, for a discrete problem then if we know the normalization constant, we can evaluate the exact probability of any particular θ by evaluating that point alone. If we do not know the normalization constant, we do not know if there are other substantially more probable events that we have thus–far missed, which would, in turn, imply that the queried point has a negligible chance of occurring.

To give a more explicit example, consider a model where $\theta \in \{1, 2, 3\}$ with a corresponding uniform prior $P(\theta) = 1/3$ for each θ. Now presume that for some reason we are only able to evaluate the likelihood at $\theta = 1$ and $\theta = 2$, giving $p(\mathcal{D}|\theta = 1) = 1$ and $p(\mathcal{D}|\theta = 2) = 10$ respectively. Depending on the marginal likelihood $p(\mathcal{D})$, the posterior probability of $P(\theta = 2|\mathcal{D})$ will vary wildly. For example, $p(\mathcal{D}) = 4$ gives $P(\theta = 2|\mathcal{D}) = 5/6$, while $p(\mathcal{D}) = 1000$ gives $P(\theta = 2|\mathcal{D}) = 1/100$.

Though this example may seem far-fetched, this lack of knowledge of the marginal likelihood is almost always seen in practice for realistic models, at least those with non-trivial solutions. Typically it is not possible to enumerate all the possible values of θ in a reasonable time and we are left wondering: how much probability mass is left that we have not seen? The problem is even worse in the setting where θ is continuous, for which it is naturally impossible to evaluate all possible values for θ. Knowing the posterior only up to a normalization constant is deceptively unhelpful: we never know how much of the probability mass we have missed and therefore whether the probability (or probability density) where we have looked so far is tiny compared to some other dominant region we are yet to explore. At its heart, the problem of Bayesian inference is a problem of where to concentrate our finite computational resources so that we can effectively characterize the posterior. If $p(\mathcal{D})$ is known, then we generally have some idea of whether we are looking in the right place or whether there are places left to look that we are yet to find.[3]

[3]For continuous problems, it might still be difficult to fully calibrate this even when $p(\mathcal{D})$ is known because we may not know what the effective scaling of the input space is. For example, different problem parameterizations will lead to different posterior densities.

Characterizing the Posterior

If we have the normalization constant, it might seem that we are done; after all, we now have the exact form of the posterior (density) using Bayes' rule. Unfortunately, it tends to be the case, particularly when θ is continuous, that this is insufficient to carry out most tasks that we might want to use our posterior for. In particular, except in special cases, we cannot directly draw samples from the posterior even if we know its exact density. As such, knowing $p(\mathcal{D})$ does not allow us to readily calculate expectations with respect to the posterior or even construct Monte Carlo estimates of such expectations.

In fact, knowing $p(\mathcal{D})$ is often not even necessary for calculating such expectations and most prominent inference methods, such as MCMC, actually sidestep estimating the marginal likelihood itself when forming characterizations of the posterior. The problem that these approaches implicitly effectively focus on is figuring out *where* the posterior density is large, remembering that our finite computational resources mean that we cannot check everywhere. This is a difficult problem as, unlike optimization, inference requires us to characterize *all* regions where the posterior density is significant, not just find a single good solution.

5.7.2 Prominent Classes of Approximate Bayesian Inference Methods

- **Markov chain Monte carlo** (MCMC): construct a Markov chain $(X_1, X_2, \dots)$ whose equilibrium distribution can be shown to be the posterior, and where each Markov kernel $X_{t+1}|X_t$ can be efficiently simulated. Simulate the Markov chain for a large enough number of time steps, and use ergodic averages to approximate posterior statistics of interest.

- **Importance sampling**: introduce a tractable proposal $q(\theta)$ and use this to produce weighted samples from the posterior (see details later).

- **Sequential Monte Carlo** (SMC): construct a finite sequence of importance samplers targeting a sequence of distributions with the last being the posterior distribution. Use resampling and rejuvenation steps to improve the effective sample size of each importance sampler.

- **Approximate Bayesian computation** (ABC): A Monte Carlo approach that is applicable in cases where the likelihood is intractable or unknown, but can be simulated. These methods are sometimes also called **likelihood-free** inference as they do not assume knowledge of the likelihood function.

- **Laplace approximation**: Use a local Gaussian approximation constructed at a maximum a posteriori (MAP) parameter (see details later).

- **Variational inference**: Frame approximation of the posterior as optimising a lower bound on the evidence with respect to a variational distribution (will be covered in depth later in the course).

It is beyond the scope of this course to properly survey this literature (note that there are two other Statistics Part C courses covering it), but we will briefly go over two of the simplest and most important approaches—Laplace approximations and important sampling—as these will be used later. We will also revisit this topic to cover one of the most common modern approach, *variational inference*, later in the course.

5.7.3 Laplace Approximation

One of the techniques for approximation of intractable posterior distributions is the **Laplace approximation** also known as **saddlepoint approximation**. The idea is to simply approximate the posterior distribution $p(\theta|\mathcal{D})$ with a (multivariate) Gaussian distribution. Given the ease of manipulating Gaussians, this is a convenient choice, since the various posterior expectations and predictive distributions can be easier to calculate when we have Gaussian approximate posteriors.

Consider for simplicity the case where parameter θ is a scalar and assume that posterior mode $\widehat{\theta}^{\text{MAP}}$ is available. Often, the posterior mode can be found even if the normalising constant $p(\mathcal{D})$ is intractable since it suffices to maximise $p(\theta|\mathcal{D}) \propto p(\theta, \mathcal{D}) = p(\mathcal{D}|\theta)p(\theta)$ using a numerical method. Then, we can use a Taylor expansion of $\log p(\theta|\mathcal{D})$ around the posterior mode $\widehat{\theta}^{\text{MAP}}$:

$$\begin{aligned}\log p(\theta|\mathcal{D}) &= \log p(\widehat{\theta}^{\text{MAP}}|\mathcal{D}) + \left.\frac{\partial \log p(\theta|\mathcal{D})}{\partial \theta}\right|_{\theta=\widehat{\theta}^{\text{MAP}}} \left(\theta - \widehat{\theta}^{\text{MAP}}\right) \\ &\quad + \left.\frac{\partial^2 \log p(\theta|\mathcal{D})}{\partial \theta^2}\right|_{\theta=\widehat{\theta}^{\text{MAP}}} \frac{\left(\theta - \widehat{\theta}^{\text{MAP}}\right)^2}{2} + \mathcal{O}\left(\left(\theta - \widehat{\theta}^{\text{MAP}}\right)^3\right).\end{aligned}$$

By ignoring the third and higher order terms and noticing that the the first derivative at the mode must be zero, we have an approximation:

$$\log p(\theta|\mathcal{D}) \approx \log p(\widehat{\theta}^{\text{MAP}}|\mathcal{D}) - \frac{\tau}{2}\left(\theta - \widehat{\theta}^{\text{MAP}}\right)^2, \tag{5.11}$$

where we write $\tau = -\frac{\partial^2 \log p(\theta|\mathcal{D})}{\partial \theta^2} \geq 0$. But recall that $\log \mathcal{N}\left(\theta|\mu, \sigma^2\right) = \log\left(\left(2\pi\sigma^2\right)^{-1/2}\right) - \frac{1}{2\sigma^2}\left(\theta - \mu\right)^2$, so this second order Taylor approximation has exactly the form of a normal log-density with mean $\mu = \widehat{\theta}^{\text{MAP}}$ and variance $\sigma^2 = \tau^{-1}$ so we can approximate the posterior with $\mathcal{N}\left(\widehat{\theta}^{\text{MAP}}, \left(-\frac{\partial^2 \log p(\theta|\mathcal{D})}{\partial \theta^2}\right)^{-1}\right)$.

This idea easily extends to multivariate densities. In particular, the Laplace approximation of $p(\theta|\mathcal{D})$ is a multivariate Gaussian $\mathcal{N}\left(\widehat{\theta}^{\text{MAP}}, \Sigma\right)$, where *the inverse covariance matrix is given by the negative Hessian of the log-posterior* evaluated at the posterior mode:

$$\Sigma^{-1} = -\left.\frac{\partial^2 \log p(\theta|\mathcal{D})}{\partial \theta \partial \theta^\top}\right|_{\theta=\widehat{\theta}^{\text{MAP}}}.$$

Since $\log p(\theta|\mathcal{D})$ agrees with $\log p(\theta, \mathcal{D})$ up to a constant, they have the same derivatives, so often we work with the *energy function* $J(\theta) = -\log p(\theta, \mathcal{D})$, which is the negative logarithm of the unnormalised posterior. Then we can write

$$\Sigma^{-1} = \left.\frac{\partial^2 J(\theta)}{\partial\theta\partial\theta^\top}\right|_{\theta=\widehat{\theta}^{\text{MAP}}}.$$

5.7.4 Importance Sampling

Importance sampling is an fundamental sampling method that is the cornerstone for many more advanced inference schemes. It relies on drawing samples from a proposal, $\hat{\theta} \sim q(\theta)$, and then assigning an **importance weight** to each sample. These importance weights act like correction factors to account for the fact that we sampled from $q(\theta)$ rather than our target $p(\theta|\mathcal{D})$.

To demonstrate the idea, consider the problem of calculating an expectation $\mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)]$ and assume for now that we can evaluate the density $p(\theta|\mathcal{D})$ exactly, but cannot draw samples from it. Importance sampling rearranges the form of the expectation to generate a different Monte Carlo estimator which we can evaluate directly as follows

$$\begin{aligned}\mathbb{E}_{p(\theta|\mathcal{D})}[f(\theta)] &= \int f(\theta)p(\theta|\mathcal{D})d\theta = \int f(\theta)\frac{p(\theta|\mathcal{D})}{q(\theta)}q(\theta)d\theta \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\frac{p(\hat{\theta}_n|\mathcal{D})}{q(\hat{\theta}_n)}f(\hat{\theta}_n) \quad \text{where} \quad \hat{\theta}_n \sim q(\theta) \qquad (5.12)\end{aligned}$$

and $p(\hat{\theta}_n|\mathcal{D})/q(\hat{\theta}_n) =: w_n$ is known as an importance weight. The key trick we have applied is to multiply the integrand by $q(\theta)/q(\theta)$, which equals 1 for all points where $q(\theta) \neq 0$. Thus if $q(\theta) \neq 0$ for all θ for which $p(\theta|\mathcal{D}) \neq 0$ (to avoid infinite importance weights), this has no effect on the expectation. However, we can view the new formulation as being the expectation of $f(\theta)p(\theta|\mathcal{D})/q(\theta)$ under the distribution $q(\theta)$ (assuming that this has the same reference measure as the posterior). We can now construct a Monte Carlo estimator for this new formulation by choosing $q(\theta)$ to be a distribution we sample from. Note the critical feature here that we only need to use $f(\theta)$ *lazily*, that is we can construct an empirical measure $p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_{n=1}^{N} w_n \delta_{\hat{\theta}_n}(\theta)$ using only the model, then later apply this to estimate the expectation of one or more f.

Importance sampling has a number of desirable properties as an inference method. In particular, it is both *unbiased* and *consistent* (i.e. as $N \to \infty$ the estimate converges to the true expectation) subject to some mild assumptions (most notably, that the proposal has heavier tails than the posterior). The former property means that its efficiency is entirely dependent on the variance of the estimator, which can straightforwardly be shown to be equal to $\text{Var}_{q(\theta)}[p(\theta|\mathcal{D})f(\theta)/q(\theta)]$, such that the optimal proposal is $q(\theta) \propto p(\theta|\mathcal{D})|f(\theta)|$, producing zero variance (i.e. exact) estimates if the sign of $f(\theta)$ if constant. In practice, we often do not know f upfront (or care about multiple different f) such that we instead look to minimize the variance of the weights themselves. This is achieved when

$q(\theta) = p(\theta|\mathcal{D})$, for which all the weights will be exactly 1 and we have exact posterior samples.

In practice, of course, we do not generally have access to a normalized version of the posterior density and instead only the unnormalized density $p(\theta, \mathcal{D}) = p(\theta|\mathcal{D})p(\mathcal{D})$. Here we can still generate importance sampling estimates by **self–normalizing** the importance weights.

The key idea for **self–normalized importance sampling** (SNIS) is that the weights provide an unbiased and consistent estimator of the marginal likelihood

$$\mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} w_n\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[w_n] = \mathbb{E}_{q(\hat{\theta}_1)}\left[\frac{p(\hat{\theta}_1, \mathcal{D})}{q(\hat{\theta}_1)}\right] = p(\mathcal{D}).$$

Now as $\mathbb{E}_{q(\theta)}\left[\frac{p(\theta,\mathcal{D})}{q(\theta)} f(\theta)\right] = E_{q(\theta)}\left[\frac{p(\theta|\mathcal{D})}{q(\theta)} p(\mathcal{D}) f(\theta)\right] = p(\mathcal{D})\ \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$, we can use our samples to construct Monte Carlo estimators for both $p(\mathcal{D})$ and $p(\mathcal{D})\ \mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$ and use the ratio of our estimates to get an estimate for $E_{p(\theta|\mathcal{D})}\left[f(\theta)\right]$ as follows

$$\mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] \approx \frac{\frac{1}{N}\sum_{n=1}^{N} w_n f(\hat{\theta}_n)}{\frac{1}{N}\sum_{n=1}^{N} w_n} \quad \text{where} \quad \hat{\theta}_n \sim q(\theta), \quad w_n = \frac{p(\hat{\theta}_n, \mathcal{D})}{q(\hat{\theta}_n)}. \tag{5.13}$$

This can alternatively be expressed as $\mathbb{E}_{p(\theta|\mathcal{D})}\left[f(\theta)\right] \approx \sum_{n=1}^{N} \bar{w}_n f(\hat{\theta}_n)$ where $\bar{w}_n = \frac{w_n}{\sum_n w_n}$ are the self–normalized importance weights such that $\sum_{n=1}^{N} \bar{w}_n = 1$.

6 Gaussian Processes

6.1 Different views of regression

Regression with least squares loss $L(y, f(x)) = (y - f(x))^2$ implies that we are fitting the *conditional mean* function $f^*(x) = \mathbb{E}[Y|X = x]$. This loss also corresponds to the probabilistic model where y_i is a noisy version of the underlying function f evaluated at input x_i:

$$y_i|f(x_i) \sim \mathcal{N}(f(x_i), \sigma^2), \quad \text{independently for} \quad i = 1, \dots, n. \tag{6.1}$$

There are different ways to model the class of functions f.

- *Frequentist Parametric* approach: model f as f_θ for some parameter vector θ. Fit θ by ML / ERM with squared loss (**linear regression**).
- *Frequentist Nonparametric* approach: model f as the unknown parameter taking values in an infinite-dimensional space of functions (RKHS). Fit f by *regularized* ML / ERM with squared loss (**kernel ridge regression**)
- *Bayesian Parametric* approach: model f as f_θ for some parameter vector θ. Put a prior on θ and compute a posterior $p(\theta|\mathcal{D})$ (**Bayesian linear regression**).
- *Bayesian Nonparametric* approach: treat f as the random variable taking values in an infinite-dimensional space of functions. Put a prior over functions $f \in \mathcal{F}$, and compute a posterior $p(f|\mathcal{D})$ (**Gaussian Process regression**).

6.2 Gaussian Process Regression

Gaussian processes (GPs) are a widely used class of models that allow us to place a prior distribution directly on the space of functions rather than on parameters in a particular family of functions. This prior can then be converted into a posterior distribution once we have seen some data.

One can think of a Gaussian process as an infinite-dimensional generalisation of a multivariate normal distribution. Namely, given an *index set* $\mathcal{X}$, a collection of random variables $\{A_x\}_{x\in\mathcal{X}}$ is said to be a Gaussian process if and only if for every finite set of indices $x_1, \dots, x_n$, vector $[A_{x_1}, \dots, A_{x_n}]^\top$ has a multivariate normal distribution on $\mathbb{R}^n$. Thus, to any Gaussian process, we can associate a random function $f \colon \mathcal{X} \to \mathbb{R}$ by setting $f(x) = A_x$, for all $x \in \mathcal{X}$. Gaussian process is fully specified by its **mean function** and **covariance function**, i.e.

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)], \\ k(x, x') &= \mathbb{E}\left[(f(x) - m(x))(f(x') - m(x'))\right], \end{aligned}$$

where expectations are taken over f (x and x' are fixed elements in the index set $\mathcal{X}$). This means that for any finite set $x_1, \dots, x_n$, $\mathbf{f} = [f(x_1), \dots, f(x_n)]^\top \in \mathbb{R}^n$ has a distribution $\mathcal{N}(\mathbf{m}, \mathbf{K})$, where $\mathbf{m}_i = m(x_i)$ and $\mathbf{K}$ is the covariance matrix given by $\mathbf{K}_{ij} = k(x_i, x_j)$.

We will typically assume that the mean function $m(x)$ is zero under the Gaussian process prior. If we know before seeing any data that the distribution of the function evaluations should be centered around some other mean, we could easily include that into the model. Equivalently, we could also subtract that known mean from the data and just use the zero mean model. If we are looking at the data to estimate the mean function, then often the zero mean GP suffices – in fact, structural information about mean functions (constant, linear) can be included into the choice of the covariance function (*exercises*). Covariance functions $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ obviously has to be positive definite so they are essentially equivalent to the kernel functions we have seen before. In fact, there is a rich connection between RKHS methods and Gaussian processes, an example of which we will discuss below.

6.2.1 Gaussian Conditioning and Regression Model

The convenience of manipulating multivariate normal distributions carries over to Gaussian processes. Let us review the rules for Gaussian conditioning, which are key to Gaussian process regression.

Gaussian Conditioning. Let $\mathbf{z} \sim \mathcal{N}(\mu, \Sigma)$ be a multivariate normal random vector and let us split its dimensions into two parts, i.e.

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}, \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}. \tag{6.2}$$

Note that $\Sigma_{21} = \Sigma_{12}^\top$ due to symmetry of covariance matrices. Then the conditional density of $\mathbf{z}_2$ given $\mathbf{z}_1$ is also normal and given by

$$p(\mathbf{z}_2|\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_2;\, \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{z}_1 - \mu_1),\, \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}). \tag{6.3}$$

For a given set of inputs $\mathbf{x} = \{x_i\}_{i=1}^n$, we denote the vector of evaluations of f by $\mathbf{f} = [f(x_1), \dots, f(x_n)]^\top \in \mathbb{R}^n$ and the vector of observed outputs by $\mathbf{y} = [y_1, \dots, y_n]^\top \in \mathbb{R}^n$. Note that since we treat f as a random function, $\mathbf{f}$ is a random n-dimensional vector. The Gaussian process regression model, assuming likelihood function in (6.1), is then given by

$$\begin{aligned} \mathbf{f} &\sim \mathcal{N}(0, \mathbf{K}) \\ \mathbf{y}|\mathbf{f} &\sim \mathcal{N}(\mathbf{f}, \sigma^2 I), \end{aligned}$$

where $\mathbf{K}$ is the covariance (kernel) matrix given by $\mathbf{K}_{ij} = k(x_i, x_j)$. But because both the prior and the likelihood are normal this simply means that $\mathbf{f}$ and $\mathbf{y}$ are *jointly normal* with

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\mathbf{K} & \mathbf{K}\\ \mathbf{K} & \mathbf{K}+\sigma^2 I\end{bmatrix}\right). \tag{6.4}$$

For example, to find the cross-covariance between $\mathbf{f}$ and $\mathbf{y}$ note that

$$\mathbb{E}\left[\mathbf{f}\mathbf{y}^\top\right] = \mathbb{E}\left[\mathbf{f}(\mathbf{f}+\sigma\epsilon)^\top\right] = \mathbb{E}\left[\mathbf{f}\mathbf{f}^\top\right] + \sigma\mathbb{E}\left[\mathbf{f}\epsilon^\top\right] = \mathbf{K}, \tag{6.5}$$

where $\epsilon \sim \mathcal{N}(0, I)$ is independent of $\mathbf{f}$. Now, we can simply apply the Gaussian conditioning to find the posterior distribution

$$\mathbf{f}|\mathbf{y} \sim \mathcal{N}(\mathbf{K}(\mathbf{K}+\sigma^2 I)^{-1}\mathbf{y}, \mathbf{K} - \mathbf{K}(\mathbf{K}+\sigma^2 I)^{-1}\mathbf{K}).$$

This gives as the posterior distribution of the evaluations of the unknown function at the set of inputs where we have observed noisy evaluations $\mathbf{y}$.

6.2.2 Posterior Predictive Distribution

But we can continue with this formalism further and construct the *posterior predictive distribution*. Suppose $\mathbf{x}' = \{x'_j\}_{j=1}^m$ is a test set. We can extend our model to include the function values $\mathbf{f}' = [f(x'_1), \dots, f(x'_m)]^\top \in \mathbb{R}^m$ at the test set. The prior can now be extended to include $\mathbf{f}'$ (recall that our prior was on the whole function – not on its values at specific locations!), so that the model reads:

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{f}'\end{bmatrix} |\mathbf{x}, \mathbf{x}' \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{\mathbf{x}\mathbf{x}} & \mathbf{K}_{\mathbf{x}\mathbf{x}'}\\ \mathbf{K}_{\mathbf{x}'\mathbf{x}} & \mathbf{K}_{\mathbf{x}'\mathbf{x}'}\end{bmatrix}\right)$$
$$\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 I)$$

where $(\mathbf{K}_{\mathbf{x}\mathbf{x}})_{ij} = k(x_i, x_j)$, $(\mathbf{K}_{\mathbf{x}'\mathbf{x}'})_{ij} = k(x'_i, x'_j)$, $\mathbf{K}_{\mathbf{x}\mathbf{x}'}$ is an $n \times m$ matrix with (i,j)-th entry $k(x_i, x'_j)$ and $\mathbf{K}_{\mathbf{x}'\mathbf{x}} = \mathbf{K}_{\mathbf{x}\mathbf{x}'}^\top$ We are now making use of the joint normality of $\mathbf{f}'$ and $\mathbf{y}$:

$$\begin{bmatrix}\mathbf{f}'\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}\mathbf{K}_{\mathbf{x}'\mathbf{x}'} & \mathbf{K}_{\mathbf{x}'\mathbf{x}}\\ \mathbf{K}_{\mathbf{x}\mathbf{x}'} & \mathbf{K}_{\mathbf{x}\mathbf{x}}+\sigma^2 I\end{bmatrix}\right) \tag{6.6}$$

and from Gaussian conditioning rules again, we can read off the posterior predictive distribution as

$$\mathbf{f}'|\mathbf{y} \sim \mathcal{N}\left(\mathbf{K}_{\mathbf{x}'\mathbf{x}}(\mathbf{K}_{\mathbf{x}\mathbf{x}}+\sigma^2 I)^{-1}\mathbf{y}, \mathbf{K}_{\mathbf{x}'\mathbf{x}'} - \mathbf{K}_{\mathbf{x}'\mathbf{x}}(\mathbf{K}_{\mathbf{x}\mathbf{x}}+\sigma^2 I)^{-1}\mathbf{K}_{\mathbf{x}\mathbf{x}'}\right). \tag{6.7}$$

Thus, we also have a closed form expression for the posterior distribution of the evaluations of the unknown function at any collection of inputs in $\mathcal{X}$. While this follows directly from the joint normality and Gaussian conditioning rules, it is instructive to notice that we could have arrived at the posterior predictive by integrating $p(\mathbf{f}'|\mathbf{f})$ through the posterior $p(\mathbf{f}|\mathbf{y})$, i.e.

$$p(\mathbf{f}'|\mathbf{y}) = \int p(\mathbf{f}'|\mathbf{f})p(\mathbf{f}|\mathbf{y})d\mathbf{f}. \tag{6.8}$$

This follows from $\int \mathcal{N}(a; Bc, D)\mathcal{N}(c; e, F)dc = \mathcal{N}(a; Be, D+BFB^\top)$ (*exercises*). Namely, even if we no longer have the Gaussian observation model (6.1) and $\mathbf{y}$ and $\mathbf{f}$ are no longer jointly normal, we can still use (6.8) to reason about the posterior predictive distribution.

6.2.3 Gaussian Process Conjugacy

A GP prior is conjugate to a Gaussian likelihood so we can think about generalizing $\mathbf{f}'|\mathbf{y}$ to $f|\mathbf{y}$, yielding a **GP posterior** as follows:

Gaussian Process Posterior. A GP prior $f \sim \mathrm{GP}(0, k_{\text{prior}})$ coupled with a Gaussian likelihood $\mathbf{y}|\mathbf{f} \sim \mathcal{N}(\mathbf{f}, \sigma^2 I)$ leads to a GP posterior as follows

$$\begin{aligned} f|\mathbf{y} &\sim \mathrm{GP}(m_{\text{post}}, k_{\text{post}}) \quad \text{where} \\ m_{\text{post}}(x) &= \mathbf{K}_{\mathbf{x}}(x)^T(\mathbf{K}_{\mathbf{xx}} + \sigma^2 I)^{-1}\mathbf{y} \\ k_{\text{post}}(x, x') &= k_{\text{prior}}(x, x') - \mathbf{K}_{\mathbf{x}}(x)^T(\mathbf{K}_{\mathbf{xx}} + \sigma^2 I)^{-1}\mathbf{K}_{\mathbf{x}}(x') \\ \mathbf{K}_{\mathbf{x}}(x) &= \begin{bmatrix} k_{\text{prior}}(x, x_1) & \dots & k_{\text{prior}}(x, x_N) \end{bmatrix}^T \end{aligned} \tag{6.9}$$

Note that this formulations satisfies the consistency of Bayes' rule under multiple observations, such that we get the same result from sequentially conditioning on observations as from conditioning on all observations at once. This is a little ugly to show algebraically but easily confirmed numerically.

6.2.4 Kernel Ridge Regression vs Gaussian Process Regression

If kernel ridge regression (KRR) uses the same kernel as the covariance function in Gaussian process regression (GPR) and moreover, if the regularisation parameter λ in KRR is the same as the noise variance σ^2 in GPR, KRR estimate of the function coincides with the GPR posterior mean. Indeed, recall that in KRR we are solving empirical risk minimisation

$$\min_{f \in \mathcal{H}_k} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \sigma^2 \|f\|^2_{\mathcal{H}_k},$$

and are fitting a function of the form $f(x) = \sum_{i=1}^n \alpha_i k(\cdot, x_i)$. Closed form solution is given by $\alpha = \left(\mathbf{K}_{\mathbf{xx}} + \sigma^2 I\right)^{-1}\mathbf{y}$. But then if we wish to predict function values at a new set $\mathbf{x}' = \{x'_j\}_{j=1}^m$ of input vectors, we have

$$f(x'_j) = \sum_{i=1}^{n} \alpha_i k\left(x'_j, x_i\right) = \left[k(x'_j, x_1), \dots, k(x'_j, x_n)\right] (\mathbf{K}_{\mathbf{xx}} + \sigma^2 I)^{-1}\mathbf{y},$$

and $\left[k(x'_j, x_1), \dots, k(x'_j, x_n)\right]$ is the j-th row of $\mathbf{K}_{\mathbf{x}'\mathbf{x}}$, so this is the same as the mean in (6.7). Note that GPR also gives predictive variance, a measure of uncertainty, which can be important when making predictions far away from the input data. There are other important differences between the two approaches: KRR is frequentist, while GPR is Bayesian, and thus the hyperparameters are fitted in different ways. KRR typically uses cross-validation and grid search, while GPR, as we discuss next, uses maximum marginal likelihood or a fully Bayesian treatment with hyperparameters integrated out.

6.3 Hyperparameter Selection

Probabilistic model given by Gaussian processes allows principled selection of hyperparameters in the model (parameters of the kernel function and the noise variance in the likelihood (6.1)) using *maximum marginal likelihood.*

Marginal likelihood of the hyperparameter vector $\theta = (\nu, \sigma^2)$ which would generally include kernel parameters ν as well as the standard deviation σ^2 of the noise in the observation model, is given by

$$p(\mathbf{y}|\theta) = \int p(\mathbf{y}|\mathbf{f}, \theta)p(\mathbf{f}|\theta)d\mathbf{f} = \mathcal{N}\left(\mathbf{y}; 0, \mathbf{K}_\nu + \sigma^2 I\right).$$

We will introduce the shorthand $\mathbf{K}_{\theta+} = \mathbf{K}_\nu + \sigma^2 I$. Thus, we can write the marginal log-likelihood as

$$\log p(\mathbf{y}|\theta) = -\frac{1}{2}\log|\mathbf{K}_{\theta+}| - \frac{1}{2}\mathbf{y}^\top \mathbf{K}_{\theta+}^{-1}\mathbf{y} - \frac{n}{2}\log(2\pi). \tag{6.10}$$

In general, marginal log-likelihood is a nonconvex function of the parameter vector θ and it can have multiple maxima - thus we typically resort to numerical optimisation methods, such as gradient ascent. The derivative with respect to θ_i (*exercise*) has the form

$$\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\theta) = -\frac{1}{2}\mathrm{Tr}\left(\mathbf{K}_{\theta+}^{-1}\frac{\partial \mathbf{K}_{\theta+}}{\partial\theta_i}\right) + \frac{1}{2}\mathbf{y}^\top \mathbf{K}_{\theta+}^{-1}\frac{\partial \mathbf{K}_{\theta+}}{\partial\theta_i}\mathbf{K}_{\theta+}^{-1}\mathbf{y}. \tag{6.11}$$

Some common kernel choices in this context involve **automatic relevance determination** (ARD) kernel

$$k(x, x') = \tau^2 \exp\left(-\sum_{j=1}^{p}\frac{(x^{(j)} - x'^{(j)})^2}{\eta_j^2}\right), \tag{6.12}$$

which has a global scale parameter τ as well as one *bandwidth* parameter η_j per covariate dimension j. If in the hyperparameter selection, very large values of η_j are selected, this essentially means that the dimension j is switched off (does not contribute to the kernel function). This is very useful in applications where it is likely that not all dimensions will be relevant.

In addition to maximum marginal likelihood, we can also perform full Bayesian inference for hyperparameters. Namely, we could start with a prior $p(\theta)$ on θ and draw samples from the posterior

$$p(\theta|\mathbf{y}) \quad\propto\quad p(\theta)p(\mathbf{y}|\theta) = p(\theta)\int p(\mathbf{y}|\mathbf{f}, \theta)p(\mathbf{f}|\theta)d\mathbf{f}.$$

This means that we can integrate uncertainty over hyperparameters into predictions as well, and approximate (integral is typically not available in closed form)

$$p(\mathbf{f}'|\mathbf{y}) = \int p(\mathbf{f}'|\mathbf{y}, \theta)p(\theta|\mathbf{y})d\theta.$$

6.4 Gaussian Processes for Classification

In Bayesian classification problems, we are interested in modelling the posterior probabilities of the categorical response variable given a set of training examples and a new input vector. These probabilities must lie in the interval $(0, 1)$ while a Gaussian process models functions that have output on the entire real axis. Thus, it is necessary to adapt Gaussian processes by transforming their outputs using an appropriate nonlinear activation/link function. Consider the binary classification model with classes -1 and $+1$, using the logistic sigmoid:

$$p(y_i = +1|f(x_i)) = s(f(x_i)) = \frac{1}{1 + e^{-f(x_i)}}. \tag{6.13}$$

This non-Gaussian form of the likelihood function, however, renders exact posterior inference intractable and approximate methods are needed. There are a number of approximate schemes that can be used but we will focus here on Laplace approximation. We know that

$$\begin{aligned}\log p(\mathbf{f}|\mathbf{y}) &= \text{const} + \log p(\mathbf{f}) + \log p(\mathbf{y}|\mathbf{f}) \\ &= \text{const} - \frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} + \sum_{i=1}^{n} \log s(y_i f(x_i)).\end{aligned}$$

Thus, we can compute the gradient

$$\frac{\partial \log p(\mathbf{f}|\mathbf{y})}{\partial \mathbf{f}} = -\mathbf{K}^{-1}\mathbf{f} + \mathbf{g}_\mathbf{f}, \tag{6.14}$$

where the gradient of the likelihood is $\mathbf{g}_\mathbf{f} = \frac{\partial \log p(\mathbf{y}|\mathbf{f})}{\partial \mathbf{f}}$ with $[\mathbf{g}_\mathbf{f}]_i = \frac{\partial \log p(\mathbf{y}|\mathbf{f})}{\partial f_i} = s(-y_i f(x_i))y_i$. The Hessian is given by

$$\frac{\partial^2 \log p(\mathbf{f}|\mathbf{y})}{\partial \mathbf{f} \partial \mathbf{f}^\top} = -\mathbf{K}^{-1} - \mathbf{D}_f, \tag{6.15}$$

where $\mathbf{D}_f = -\frac{\partial^2 \log p(\mathbf{y}|\mathbf{f})}{\partial \mathbf{f} \partial \mathbf{f}^\top}$ is the negative Hessian of the log-likelihood, which is an $n \times n$ diagonal matrix[1] with $(\mathbf{D}_f)_{ii} = s(f(x_i))s(-f(x_i))$. The overall Hessian of the log-posterior is negative definite, since $\mathbf{K}^{-1}$ is positive definite and $(\mathbf{D}_f)_{ii} \geq 0$. Thus, there is a unique posterior mode. Note also that $\mathbf{D}_f$ depends on $\mathbf{f} = [f_1, \ldots, f_n]^\top$ but not on the labels $\mathbf{y}$. We can now employ numerical optimisation (gradient ascent or Newton-Raphson method) to find the posterior mode $\hat{\mathbf{f}}^{\text{MAP}}$ and approximate the posterior $p(\mathbf{f}|\mathbf{y})$ with a normal distribution:

$$\tilde{p}(\mathbf{f}|\mathbf{y}) \;=\; \mathcal{N}\left(\mathbf{f} \,\middle|\, \hat{\mathbf{f}}^{\text{MAP}}, \left(\mathbf{K}^{-1} + \mathbf{D}_{\hat{\mathbf{f}}^{\text{MAP}}}\right)^{-1}\right).$$

[1]Note that $\frac{\partial^2 \log p(\mathbf{y}|\mathbf{f})}{\partial \mathbf{f} \partial \mathbf{f}^\top}$ is a diagonal matrix in any GP model regardless of the form of the likelihood function as long as it factorizes across observations, i.e. $p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{n} p(y_i|f_i)$ with $(\mathbf{D}_f)_{ii} = -\frac{\partial^2 \log p(y_i|f_i)}{\partial f_i^2}$. In addition, if $\log p(y|f)$ is concave in f, $(\mathbf{D}_f)_{ii} \geq 0$.

Note that this can be rewritten as

$$\tilde{p}\left(\mathbf{f}|\mathbf{y}\right) \;=\; \mathcal{N}\left(\mathbf{f}\,\middle|\,\hat{\mathbf{f}}^{\mathrm{MAP}}, \mathbf{K}-\mathbf{K}\left(\mathbf{K}+\mathbf{D}^{-1}_{\hat{\mathbf{f}}^{\mathrm{MAP}}}\right)^{-1}\mathbf{K}\right),$$

using the Woodbury identity[2] $\left(\mathbf{K}^{-1}+\mathbf{D}\right)^{-1} = \mathbf{K}-\mathbf{K}\left(\mathbf{K}+\mathbf{D}^{-1}\right)^{-1}\mathbf{K}$ for invertible matrices $\mathbf{K}$ and $\mathbf{D}$.

We can use the Laplace approximation further to construct an approximation of the predictive posterior at a test set $\mathbf{x}' = \{x'_j\}_{j=1}^m$, writing

$$\tilde{p}(\mathbf{f}'|\mathbf{y}) = \int p(\mathbf{f}'|\mathbf{f})\tilde{p}(\mathbf{f}|\mathbf{y})d\mathbf{f}, \tag{6.16}$$

which can now be solved in the closed form since $p(\mathbf{f}'|\mathbf{f})$ is also normal,

$$p\left(\mathbf{f}'|\mathbf{f}\right) = \mathcal{N}\left(\mathbf{f}' \,|\, \mathbf{K}_{\mathbf{x}'\mathbf{x}}\mathbf{K}_{\mathbf{x}\mathbf{x}}^{-1}\mathbf{f}, \mathbf{K}_{\mathbf{x}'\mathbf{x}'}-\mathbf{K}_{\mathbf{x}'\mathbf{x}}\mathbf{K}_{\mathbf{x}\mathbf{x}}^{-1}\mathbf{K}_{\mathbf{x}\mathbf{x}'}\right),$$

giving

$$\tilde{p}\left(\mathbf{f}'|\mathbf{y}\right) = \mathcal{N}\left(\mathbf{f}' \,|\, \mathbf{K}_{\mathbf{x}'\mathbf{x}}\mathbf{K}_{\mathbf{x}\mathbf{x}}^{-1}\hat{\mathbf{f}}^{\mathrm{MAP}}, \mathbf{K}_{\mathbf{x}'\mathbf{x}'}-\mathbf{K}_{\mathbf{x}'\mathbf{x}}\left(\mathbf{K}_{\mathbf{x}\mathbf{x}}+\mathbf{D}^{-1}_{\hat{\mathbf{f}}^{\mathrm{MAP}}}\right)^{-1}\mathbf{K}_{\mathbf{x}\mathbf{x}'}\right). \tag{6.17}$$

6.4.1 Numerically stable implementation

Kernel matrix $\mathbf{K}$ can in practice have eigenvalues close to zero and thus be numerically unstable to invert. Fortunately, the direct inversion of $\mathbf{K}$ can be avoided. Consider, for example, the Newton iteration for finding the MAP given by

$$\begin{aligned}
\mathbf{f}^{new} &= \mathbf{f}-\left(\frac{\partial^2 \log p(\mathbf{f}|\mathbf{y})}{\partial\mathbf{f}\partial\mathbf{f}^\top}\right)^{-1}\frac{\partial \log p(\mathbf{f}|\mathbf{y})}{\partial\mathbf{f}} \\
&= \mathbf{f}+\left(\mathbf{K}^{-1}+\mathbf{D}_f\right)^{-1}\left(\mathbf{g}_\mathbf{f}-\mathbf{K}^{-1}\mathbf{f}\right) \\
&= \left(\mathbf{K}^{-1}+\mathbf{D}_f\right)^{-1}\left(\mathbf{K}^{-1}+\mathbf{D}_f\right)\mathbf{f}+\left(\mathbf{K}^{-1}+\mathbf{D}_f\right)^{-1}\left(\mathbf{g}_\mathbf{f}-\mathbf{K}^{-1}\mathbf{f}\right) \\
&= \left(\mathbf{K}^{-1}+\mathbf{D}_f\right)^{-1}\left(\mathbf{D}_f\mathbf{f}+\mathbf{g}_\mathbf{f}\right) \\
&= \left[\mathbf{K}-\mathbf{K}\left(\mathbf{K}+\mathbf{D}_\mathbf{f}^{-1}\right)^{-1}\mathbf{K}\right]\left(\mathbf{D}_f\mathbf{f}+\mathbf{g}_\mathbf{f}\right),
\end{aligned}$$

where the last step makes use of the Woodbury identity from before. We thus have an expression involving only the inverse of $\mathbf{K}+\mathbf{D}_\mathbf{f}^{-1}$ and not of $\mathbf{K}$, which is typically much better behaved. Further manipulations are also possible if this still proves problematic, see e.g. [47, Section 3.4.3].

[2]Woodbury matrix identity or matrix inversion lemma in its general form is $(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1}$ for matrices A,U,C,V of conformable sizes.

6.5 Large-Scale Kernel Approximations

Gaussian processes and kernel methods require computational cost that scales at least as $O(n^2)$ and often as $O(n^3)$ in the number of observations n (due to the need to compute, store and invert the $n \times n$ kernel matrix $\mathbf{K}$). This is the price we pay for having a non-parametric model, i.e. for performing the computation in terms of the dual coefficients. For large datasets (anything above a few thousand datapoints for GPs) this becomes a prohibitive computational cost and memory requirement. Many methods have been proposed to deal with this issue, here we will overview the basic approaches based on the reduced-rank approximation of $\mathbf{K_{xx}}$ - see Chapter 8 of [47] for an in-depth overview.

6.5.1 Low Rank Matrix Approximations

GP regression and kernel ridge regression both require inversion of the matrix $\mathbf{K_{xx}} + \sigma^2 I$. Let us assume for the moment that $\mathbf{K_{xx}}$ can be approximated by a rank m matrix, with $m \ll n$, i.e. $\mathbf{K_{xx}} \approx QQ^\top$, where Q is an $n \times m$ matrix. Then we can apply the matrix inversion lemma and write

$$\left(QQ^\top + \sigma^2 I\right)^{-1} = \sigma^{-2} I - \sigma^{-2} Q \left(\sigma^2 I + Q^\top Q\right)^{-1} Q^\top, \tag{6.18}$$

such that the inversion of an $n \times n$ matrix has been transformed into an inversion of an $m \times m$ matrix. However, in order to derive the optimal reduced-rank approximation to $\mathbf{K_{xx}}$, we need to perform the eigendecomposition of $\mathbf{K_{xx}}$ which is itself a costly operation, requiring $O(n^3)$ computation. Instead, a classic approach is the **Nyström approximation**:

$$\tilde{\mathbf{K}}_{\mathbf{xx}} = \mathbf{K_{xz}} \mathbf{K_{zz}}^{-1} \mathbf{K_{zx}},$$

where $\{z_j\}_{j=1}^m$ is a collection of a small number of inputs in $\mathcal{X}$ (which could be a subset of the training set, but could also be some auxiliary pseudo-inputs) called **inducing points** or **landmark points** that need to be learned, and we denoted as usual $(\mathbf{K_{zz}})_{ij} = k(z_i, z_j)$, $(\mathbf{K_{xz}})_{ij} = k(x_i, z_j)$ and $\mathbf{K_{zx}} = \mathbf{K_{xz}}^\top$. Now, we can set $Q = \mathbf{K_{xz}} \mathbf{K_{zz}}^{-1/2}$ and apply the formula (6.18). Note that this is equivalent to using a finite-dimensional feature map $\phi : x \mapsto \mathbf{K_{zz}}^{-1/2} \left[k(z_1, x), \ldots, k(z_m, x)\right]^\top$and an *approximate kernel* $\hat{k}(x, x') = \phi(x)^\top \phi(x')$.

6.5.2 Sparse Gaussian Processes

There are a large variety of alternative approximate methods in the GP literature that also make use of similar ideas with inducing points to approximate the GP posterior (see e.g., Chapter 2 of [57] for a recent review) in different ways. These are known as **sparse Gaussian process** approximations.[3]

Many sparse GP approaches operate in a way that avoids implying a finite dimensional feature map. This is important when it comes to uncertainty prediction as it avoids

[3]Note there is no particular distinction

collapsing of the uncertainty estimates away from the data points; they maintain the non–parametric nature of GPs. By contrast, any approximation that implies a finite–dimensional feature map will induce a posterior that collapses everywhere given sufficient training data due to the Bernstein von Mises theorem.

Though it is beyond the scope of the course to cover these sparse GP approximations properly, or indeed the many interpretations they afford, we note that they can be intuitively be thought of as summarizing the data through m inducing points and then using these inducing points to fit the GP. The inducing points take the form of pseudo data, with both their input and output values carefully optimized/inferred to achieve the best possible approximation. As such, they play the role of maximally informative "super" datapoints, with the intuition being that smoothness of the GP causes substantial redundancy in the information provided by the full dataset (i.e. groups of closely spaced datapoints can be summarized by a single carefully placed datapoint).

6.5.3 Random Fourier Features

Another popular method are **random Fourier features** (RFF) [43], used regularly in the context of frequentist kernel methods (e.g. KRR) due to their strong theoretical guarantees and asymptotic performance, but less often for GPs due to not maintaining non–parametric uncertainty estimates. The idea behind them is to use something called **Bochner's Theorem**, which gives a representation of **translation-invariant** or **stationary** kernels on $\mathbb{R}^p$ using a Fourier transform, and then derive more tractable approximations from this representation.

If our kernel takes the form $k(x, x') = \kappa(x - x')$ such that it only depends on the difference $x - x'$, it turns out that we can represent it as

$$k(x, x') = 2\,\kappa(0)\,\mathbb{E}\left[\cos(\omega^\top x + b)\cos(\omega^\top x' + b)\right] \tag{6.19}$$

where $b \sim \text{Uniform}(0, 2\pi)$ and $\omega \in \mathbb{R}^p$ has density given by the normalized Fourier transform of κ, that is[4]

$$p(\omega) \propto \int_{\delta \in \mathbb{R}^p} \kappa(\delta)\exp(-i\omega^T\delta)d\delta = \int_{\delta \in \mathbb{R}^p} \kappa(\delta)\cos(\omega^T\delta)d\delta.$$

Though the proof is non–trivial, the intuition here is that a Fourier transformation of κ always produces a non–negative measure if k is a valid kernel (Bochner's Theorem). Thus by normalizing this measure and considering its inverse mapping back into the original space we can derive the above result after some algebraic manipulations.

For many common kernels, $p(\omega)$ is analytically calculable and can be easily sampled from. For example, if $k(x, x') = A\exp\left(-\frac{1}{2\gamma^2}\|x - x'\|^2\right)$ with constants A and γ (i.e. an RBF kernel), then $p(\omega) = \mathcal{N}(\omega; 0, \gamma^{-2}I)$. In such cases, we can derive an

[4]Note that the imaginary part of $p(\omega)$ is always zero. This can be shown by combining Euler's formula, that is $\exp(-i\omega^T\delta) = \cos(\omega^T\delta) - i\sin(\omega^T\delta)$, with the fact that $\kappa(-\delta) = \kappa(\delta)$ because kernels are symmetric. Thus the imaginary part of the integrand is anti-symmetric and thus integrates to zero over the real line.

unbiased $m-$sample Monte Carlo estimate for $k(x, x')$ by sampling $\hat{\omega}_j \overset{iid}{\sim} p(\omega)$ and $\hat{b}_j \overset{iid}{\sim} \text{Uniform}(0, 2\pi)$ and taking

$$k(x, x') \approx k_m(x, x') := \frac{2\kappa(0)}{m} \sum_{j=1}^{m} \cos(\hat{\omega}_j^\top x + \hat{b}_j) \cos(\hat{\omega}_j^\top x' + \hat{b}_j). \tag{6.20}$$

The significance of this approximation is that it can be represented as an explicit inner product between feature maps $\varphi_m : \mathbb{R}^p \mapsto \mathbb{R}^m$ as follows

$$\begin{aligned} &k_m(x, x') = \varphi_m(x)^\top \varphi_m(x') \\ \text{where} \quad &\varphi_m(x) = \sqrt{\frac{2\kappa(0)}{m}} \left[\cos(\hat{\omega}_1^\top x + \hat{b}_1), \cos(\hat{\omega}_2^\top x + \hat{b}_2), \dots, \cos(\hat{\omega}_m^\top x + \hat{b}_m)\right]^\top . \end{aligned} \tag{6.21}$$

We can then use this to "undo" the kernel trick and revert back to an explicit transformation of our input data. That is, we can calculate the resulting feature representation of the data $\Phi_m \in \mathbb{R}^{n \times m}$ and then apply the "unkernelized" version of our method. For example, in kernel PCA or kernel ridge regression we can work with $\Phi_m^T \Phi_m$ (which is $m \times m$) instead of the gram matrix $\Phi_m \Phi_m^T$ (which is $n \times n$), thereby reducing the cost from $O(n^3)$ to $O(m^2 n)$.

Though it is somewhat ironic and disappointing to have come full circle such that we are now essentially not using a kernel method at all but taking a fixed feature map instead, such approximate kernel methods can still be quite useful in practice. Essentially, we have shown that feature maps of the form (6.21) can provide us behavior akin to full kernel methods and come with strong theoretical motivation in terms of their behavior in the limits of large m. Nonetheless, we must be careful to justify the use of kernel methods in cases where we have a large amount of data, with alternative methods that do not suffer from such issues, such as deep learning approaches, often available.

7 Bayesian Optimization

Bayesian optimization is a global optimization scheme that requires only that the target function can be evaluated (noisily) at any given point [39, 27, 11, 50]. It does not require derivatives,[1] naturally incorporates noisy evaluations, and is typically highly efficient in the number of function evaluations. It is therefore suited to problems where the target function corresponds to the output of a simulator, estimation scheme, algorithm performance evaluation, or other cases where the target is not known in closed form and is expensive to evaluate. It remains a fast growing area of active research and has been successfully applied to a wide range of problems such as hyperparameter tuning [52], tuning simulators or physical experiments [56], and even perfecting dessert recipes [32]

7.1 Motivation: tuning hyperparameters as optimizing "black-box" functions

Through the course we have considered/will consider several families of machine learning models which have complex inference algorithms and often require tuning of a number of hyperparameters in order to make them work in practice. These could for example be kernel parameters, the number of layers and units per layer in a deep neural network, learning rates, regularization parameters, or batch sizes in stochastic optimizers. Choosing the right hyperparameters can often have massive impact on performance, but performing the required optimization can be extremely difficult and time consuming as the algorithm itself needs to be trained for each tested configuration. The classic approach for doing this is often considered to be "graduate student descent," that is getting somebody to manually tune the parameters for you!

BO provides a mechanism to do this in a more principled and automated way, i.e. without having "human in the loop." Ideally, we would want a fully automated machine learning pipeline where (nearly) optimal model configuration is selected with a small number of algorithm runs.

More broadly, we are interested *globally* maximizing[2] some function $f(x)$ within some domain $\mathcal{X}$,

$$x^* = \operatorname{argmax}_{x \in \mathcal{X}} f(x),$$

in a setting where evaluations of $f(x)$ are very *expensive* and potentially also *noisy*. We will consider f to be a *black–box* function such that we can only make pointwise evaluations for selected inputs x_i, yielding $y_i = f(x_i) + \epsilon_i$, where ϵ_i represents any

[1]Nonetheless, derivative measurements, if available, can still often be exploited to improve the BO procedure. See, for example, [41, Section 4.7].

[2]For minimization, we can simply consider maximizing $-f(x)$ instead.

potential noise in this evaluation. Each such evaluation is expected to incur a substantial (computational) cost, e.g. corresponding to training a large machine learning model or running a physical experiment, such that we need to be very careful in choosing the best possible inputs to evaluate; we will often have a strict budget on the number of evaluations we can afford. We will also generally assume that $\mathcal{X}$ is a bounded domain such that $\mathcal{X} \subset \mathbb{R}$, but note that there are also strategies for running BO in unbounded domains or where inputs are not real valued, see e.g. [45].

7.2 Optimization Through a Surrogate Model

The key idea of BO is to place a **surrogate model** on f, most commonly a GP (we will assume this is the case from here on), that allows us to reason about its values at points we have not directly evaluated. In particular, it allows estimation of the expected value and uncertainty in $f(x)$ for all $x \in \mathcal{X}$. As new evaluations are made, the resultant information is incorporated into the model so that our surrogate becomes more accurate with time, with the aim being to achieve an accurate representation in region around the true optimum. Under the standard assumption of a GP prior and a Gaussian likelihood (i.e. assuming that our noise terms ϵ_i are i.i.d. Gaussian), the information contained by our surrogate is contained within the GP posterior mean, which we will denote $\mu(x)$, and its marginal posterior standard deviation, which we will denote $\sigma(x) := \sqrt{k_{\text{post}}(x, x)}$.

Because BO employs only point-wise function evaluations, there are only two things we need to control (assuming that we have a fixed budget of function evaluations)

1. Where we *evaluate* the function;
2. Where we *predict* the optimum to be given our evaluations.

Both of these can be controlled using our GP surrogate.

Given a set of evaluations $\{x_i, y_i\}_{i=1}^n$, the second of these is straightforward: we use the tested input which has the highest posterior mean under our input, that is x_{i^*} where $i^* = \text{argmax}_{i \in \{1,\dots,n\}} \mu(x_i)$, as our prediction. If our evaluations are noiseless this will coincide with largest of our evaluations (presuming the GP is setup appropriately), but it allows for marginalizing out this noisy in the more general case. There are of course also some variations on this. For example, in some rare cases we might instead allow return of a point that is not actually in our set of evaluations at all by finding and returning $\text{argmax}_{x \in \mathcal{X}} \mu(x)$. Alternatively, we might more conservatively choose the evaluated point with the best lower bound on the function by considering $\mu(x) - \beta\sigma(x)$, for some $\beta > 0$, instead of $\mu(x)$.

The first problem, of where to *evaluate* the function, is the real challenge of optimization. It constitutes a (Bayesian) **sequential experimental design** problem: at each iteration of our optimization algorithm we want to choose to evaluate the point that will result in the most *information* being gained about the location of the optimum.

To encapsulate this, we specify a so-called **acquisition function** $\zeta : \mathcal{X} \to \mathbb{R}$ that encodes the relative utility of evaluating a new point. The optimum of this acquisition function is then the optimal point to evaluate at the next iteration. At a high level,

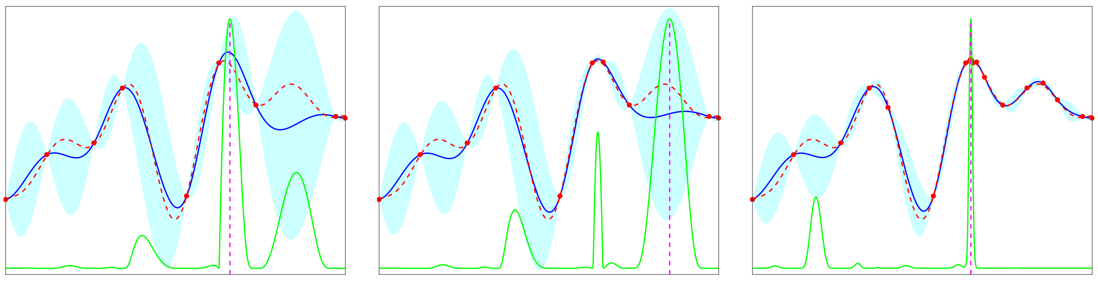

(a) 10 evaluations (b) 11 evaluations (c) 20 evaluations

Figure 7.1: Illustration of using Bayesian optimization to optimize the toy one dimensional problem given in (7.1). Red dots show the function evaluations, the dotted red line shows the true function, the solid blue line is the GP mean, the shaded region is the GP mean ±2 standard deviations, the green line is the acquisition function, and the dotted purple line is the location of the maximum of the acquisition function. Note that the scales for the acquisition functions are different for the different iterations. See main text for further description.

the utility of a point is based on a trade-off between *exploration*, namely the desire to evaluate points where our uncertainty is high to reduce the uncertainty in those regions, and *exploitation*, namely the desire to evaluate points where the expected function value is high so that we get a good characterization of the function in promising regions. For example we want to refine our estimates of promising local optima, while also investigating places that may contain peaks we have missed thus far. We will return to discuss specific acquisition functions in depth in the next section, noting only for now that good acquisition functions should balance this **exploration–exploitation trade–off**.

When direct evaluation of f is expensive, the acquisition function constitutes a cheaper–to–evaluate substitute, which is optimized to ascertain the next point at which the target function should be evaluated in a sequential fashion. By interleaving optimization of the acquisition function, evaluating f at the suggested point, and updating the surrogate (and thus also the acquisition function), BO forms a global optimization algorithm that is typically very efficient in the required number of function evaluations, whilst naturally dealing with noise in the outputs.

Figure 7.1 shows an example of BO being used to maximize the following simple one-dimensional function

$$f(x) = \frac{x}{15} - \frac{x^2}{50} - \frac{\sin x}{x}, \quad -5 \leq x \leq 5 \tag{7.1}$$

with observation noise $\epsilon \sim \mathcal{N}(0, 0.01^2)$. Consider, for example, the acquisition function after 10 function evaluations shown in green in Figure 7.1a. This is reasonably large in the two regions of highest uncertainty, but it is highest in the region near the true global optima where both the expected value and uncertainty are high. At the next iteration (Figure 7.1b), on the other hand, the uncertainty in this region has dropped dramatically and the point with the highest uncertainty now has the largest value for the acquisition function and so this is the point that will be evaluated next. Overall, we see that BO works by interleaving regressing a GP to the evaluations, optimizing the acquisition function to find the next point to evaluate, evaluating the target function at that point, and the updating the GP regression again.

By the time we have made 20 evaluations (Figure 7.1c), we have a very accurate characterization of the true function in the region of its true optimum, and therefore a highly accurate optimization. The characterization is less accurate elsewhere, highlighting the sample efficiency of BO as it shows how computational resources are focused on where they are required: we do not need to the function's exact value at points we are highly confident are not the global optimum.

At first, it might seem strange to replace our original optimization problem with another in the form of optimizing the acquisition function to choose the point to evaluate next. This new optimization problem is itself a global optimization of a typically highly multi-modal function and we need to carry it out at each iteration of the BO algorithm. Indeed if the target function f is cheap to evaluate, taking such an approach would be rather foolish. The key difference is that the surrogate optimization problem can be solved without the need to evaluate the original target function. Therefore, if f is expensive to evaluate, this additional computation can be justified if it keeps the number of required evaluations of f to a minimum.

There are also a number of other features that mean the problem of optimizing the acquisition function is typically much easier than the original target – it can be evaluated exactly, derivatives are often available, and provided an appropriate kernel is used, then it is guaranteed to be smooth. It is also not typically necessary to ensure that the acquisition function is optimized exactly, as evaluating a point that is sub-optimal under the acquisition function simply means that a point which we expected to be marginal less helpful is evaluated at the next iteration; it does not undermine the optimization of $f(x)$ itself in anyway. This can be useful when the time for a BO iteration is comparable to a function evaluation, as the amount of computational effort undertaken by the BO can be tuned to the problem at hand.

7.2.1 Choice of Surrogate

We have thus far assumed that a GP is used for the surrogate. Though this is far from the only possible choice, GPs remain the most common surrogate model used in practice because of a number of beneficial features. For example, they are very powerful regressors that can accurately represent the function from relatively few evaluations, especially for low-dimensional smooth functions. They also naturally produce uncertainty estimates that are typically more accurate than those produced by alternatives which often have to resort to post-processing or heuristics to estimate uncertainty. This also leads to simple and tractable acquisition functions as we will see later. Similarly, their ability to incorporate noisy observations is often helpful. However, because GP inference is only analytic for Gaussian likelihoods, it is often impractical to incorporate non-Gaussian noise. This problem can be particularly manifest when the observations are bounded, such as when the target is a probability and thus always positive. A common way of dealing with this is to optimize a mapping of the original function, for example optimizing the log evidence in Bayesian model learning problems.

When prior information is known about the function, their flexibility in choosing the covariance kernel can also be very helpful, for example allowing known smoothness to be incorporated. However, this can also be a curse as an inappropriate choice of kernel can severely hamper the performance. In particular, it will typically be necessary to do inference over, or at least optimize, the GP hyperparameters. Another drawback to using GPs is poor scaling in the number of iterations. This can be particularly problematic for BO as we will be training a new GP at each iteration and then also need to evaluate it a large number of times as well;[3] the overall cost of running BO algorithm for n iterations will be $O(n^4)$ which can quickly catch up with the cost of the function evaluations themselves. Random forests can be an attractive, far cheaper, alternative in settings where this becomes problematic [26]. As with all kernel methods, the scaling with the input dimensionality is also typically poor, with neural network based approaches sometimes preferable for moderate or high dimensional problems [53].

7.3 Acquisition Functions

We now introduce a number of common acquisition functions when using a GP surrogate. The choice of acquisition function can be critical to the performance of Bayesian optimization and its choice forms a basis for a significant proportion of the Bayesian optimization literature [50]. In particular, some acquisition functions have pathologies which can severely impede the performance of BO. This has led to the development of some very powerful, but computationally intensive and algorithmically complicated, acquisition strategies, such that there is often a speed/simplicity vs per iteration performance trade-off in their selection.

7.3.1 Probability of Improvement

One of the conceptually simplest acquisition functions is the **probability of improvement** (PI), namely the probability that the function value at a point is higher than the current estimated optimum. The probability of improvement has a simple analytic form because the marginal distributions for function evaluations are Gaussian. Specifically, let

$$\mu^+ = \max_{i \in \{1,\dots,n\}} \mu(x_i) \tag{7.2}$$

denote the point with the highest expected value and define

$$\gamma(x) = \frac{\mu(x) - \mu^+ - \xi}{\sigma(x)}, \tag{7.3}$$

[3]Note that we only need to do the critical $O(n^3)$ matrix once, after which prediction at new points can be done in $O(n^2)$.

where $\xi \geq 0$ is a user set parameter that represent a threshold of improvement we wish to pass. The probability of improvement is then defined as the probability of improving the objective function by at least an amount ξ

$$\begin{aligned}\zeta_{\text{PI}}(x) :&= P(f(x) \geq \mu^+ + \xi|\mathcal{D}) \\ &= \int_{\mu^+ + \xi}^{\infty} \mathcal{N}(f(x); \mu(x), \sigma^2(x)) df(x) \\ &= \Phi\left(\gamma\left(x\right)\right)\end{aligned}$$

where $\Phi(\cdot)$ represents the unit normal cumulative distribution function. Note here that we are considering the probability of the true function value being larger than our threshold $\mu^+ + \xi$, not that a noisy output evaluation will be above the threshold.

It is easy to see that setting ξ too small will lead to pathologies, with $\xi \to 0$ potentially causing evaluations only infinitesimally far away from the current best estimate of the optimum. Note also that $\xi \to \infty$ corresponds to pure exploration as this will corresponding to choosing the point with the largest $\sigma(x)$.

The performance of PI further turns out to be extremely sensitive to the value of ξ, with it often needing to be set adaptively over different iterations. It is also easy to see that it fail to encapsulate the *magnitude* of any potential gains, such that it fails to use all the information available. Because of these shortfalls, it is rarely used in practice.

7.3.2 Expected Improvement

A method less sensitive to tuning is the **expected improvement** (EI). It is conceptually similar to PI, but incorporates the idea that large improvements are more beneficial than small improvements. It, therefore, better represents the importance of exploration and is less prone to over-exploitation. It can also be calculated analytically for a GP by making use of the following general result for Gaussian random variables.

Lemma 15. *Let $S \sim \mathcal{N}\left(m, \tau^2\right)$. Denote by Φ the cdf of a standard normal random variable. Then*

$$\mathbb{E}\left[\max\left(0, S\right)\right] = \int_0^{\infty} s\mathcal{N}\left(s; m, \tau^2\right) ds = m\Phi\left(\frac{m}{\tau}\right) + \tau\mathcal{N}\left(\frac{m}{\tau}; 0, 1\right).$$

Now more formally defining the expected improvement, we now have

$$\begin{aligned}\zeta_{\text{EI}}(x) &:= \int_{\mu^+ + \xi}^{\infty} (f(x) - \mu^+ - \xi)\mathcal{N}(f(x); \mu(x), \sigma^2(x)) df(x) \\ &= \int_0^{\infty} s\mathcal{N}(s + \mu^+ + \xi; \mu(x), \sigma^2(x)) ds \\ &= \int_0^{\infty} s\mathcal{N}(s; \mu(x) - \mu^+ - \xi, \sigma^2(x)) ds\end{aligned}$$

and applying Lemma 15

$$
\begin{aligned}
&= \left(\mu(x) - \mu^{+} - \xi\right) \Phi(\gamma(x)) + \sigma(x)\mathcal{N}(\gamma(x); 0, 1) \\
&= \sigma(x)\Bigg(\gamma(x)\Phi(\gamma(x)) + \mathcal{N}(\gamma(x); 0, 1)\Bigg).
\end{aligned}
$$

The parameter ξ plays a similar role as for the PI, but in this case $\xi = 0$ no longer corresponds to pure exploitation and is actually a perfectly valid choice (though sometimes a little prone to insufficient exploration). More generally, it is usually found that EI is not overly sensitive to ξ and that cooling schedules are not usually needed. It is thus typically fixed to a small, but non–zero, value.

It is worth noting here that, though these presented versions are the most common forms used for EI and PI, alternative variants can also be calculated that incorporate uncertainty in the function's value at our best point. That is, instead of considering the improvement over $\mu^{+} + \xi$, they consider improvement over $f(x^{+}) + \xi$ where x^{+} is the input associated with μ^{+} and $f(x^{+})$ is now itself considered a random variable. These variants will be derived in the examples sheet.

7.3.3 Upper Confidence Bounding

Another possible acquisition strategy is to maximize the **upper confidence bound** (UCB) for the function value, namely

$$
\text{UCB}\left(x\right) = \mu\left(x\right) + \beta\sigma\left(x\right) \tag{7.4}
$$

where $\beta \geq 0$ is again a parameter that will control the exploration/exploitation trade-off and is sometimes known as an **optimism boost**. This bound represents the point at which the probability the function is less than this value is $\Phi(\beta)$.

In general, β needs to slightly increase as we take more iterations (though potentially only at a logarithmic rate) to avoid pathologies where certain regions are never evaluated no matter how many iterations we take because they have $\mu\left(x\right) + \beta\sigma\left(x\right) < \mu^{+}$. Note that the same principle is used extensively in the theory of K-armed bandits which deals with a similar setup of the optimization of uncertain objectives over a discrete number of choices (not covered in this course).

7.3.4 Information-Based Policies

At the end of the day, the aim of BO is to find the location of the optimum. The true utility of evaluating a new point is, therefore, in the information it provides about the location of the true maximum x^{*} and the assistance it provides in guiding future evaluations. Though the latter of these is difficult to actively incorporate, the former is something that can be targeted in a principled manner by considering the distribution on the location of the maximum $p(x^{*}|\mathcal{D})$ where $\mathcal{D}$ is the previous evaluation pairs.

The simplest way this can be done is using Thompson sampling, where instead of optimizing an acquisition function directly, one looks to sample the next point to evaluate from $p(x^*|\mathcal{D})$. In the GP setting, this is equivalent to sampling a particular function realization f and then taking x_{n+1} as the optimum of this particular realization. This can prove surprisingly expensive, as sampling a joint realization of a GP at M points is an $O(M^3)$ operation, but it is still occasionally done in practice.

A more complicated, but theoretically more powerful approach, is to try and directly optimize for the point that will, in expectation, most reduce the uncertainty about the location of the maximum. These so-called **entropy search** (ES) methods [23, 24] use as an acquisition function the expected gain in Shannon information about the location of the optimum, that is the reduction in entropy of the location of the maximum. Equivalently, this is conditional mutual information between x^* and the value of the new evaluation y_{n+1}, given the previously observed data $\mathcal{D}$ and point we choose to evaluate x_{n+1}. These methods are special instances of the more general concept of sequential Bayesian experimental design [13].

Unfortunately, this mutual information is not analytically tractable and actually rather challenging to estimate effectively. One must thus resort to approximation schemes to be able to use the approach. Despite these approximations, some of the resulting ES schemes provide excellent empirical performance in terms of the number of function evaluations and form some of the state-of-the-art acquisition functions for BO.

8 Deep Learning

Deep learning is a machine learning approach where we construct a parameterized predictive function using a computational graph of simpler differentiable functional blocks, and then train this on data using some form of gradient–based optimization. Though many of the key ideas stem back to the 1980s and 90s, and sometime even further, the field has seen a meteoric rise over the last decade. This is due to a combination of outstanding empirical successes for large, high–dimensional, datasets (particularly vision and natural language processing); its flexibility to deal with a wide array of problems that few other approaches are appropriate for; and the development of highly advanced software and hardware that the makes developing and deploying models simple, scalable, and efficient. Though the speed of its rise has often lead to over-hyping and sometimes even downright misinformation, it is undeniable that it is an extremely powerful approach that now forms the basis for a significant proportion of both machine learning research and its applications, while being at the centre of many major technological advancements.

Note that this is a chapter where the lectures (and slides) contain a particularly large amount of additional information on top of the content of these notes. In particular, much of the motivation, high–level intuitions, and examples will only be covered in the lectures themselves.

8.1 Why Deep Learning

Consider the empirical risk minimisation framework for linear binary classification:

$$\min_{w \in \mathbb{R}^p} \left(\frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^{n} L(y_i, w^\top x_i) \right)$$

We have seen how we can generalise the linear method to a non-linear one using a kernel. The resulting ERM objective is:

$$\min_{w \in \mathcal{H}_k} \left(\frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^{n} L(y_i, \langle w, k(x_i, \cdot) \rangle_{\mathcal{H}_k}) \right)$$

This implicitly and non-linearly maps inputs x_i into an RKHS $k(x_i, \cdot)$ element. A choice for the kernel corresponds effectively to a choice of features to use, and often domain knowledge is required to make an appropriate choice.

There are a number of problems with this approach:

1. For many problems it may not be obvious what kernel/features to use to achieve good performance. In particular, for high–dimensional problems it may be extremely difficult to manually construct appropriate measures of similarity or meaningful features.

2. Even if a kernel family can be identified, we still need to optimize the hyperparameters which can be both challenging and computationally expensive.

3. Kernel methods can be computationally slow. Using the Representer Theorem means that optimisation is based on the Gram matrix which is n^2 in size, and can require $O(n^3)$ computational cost to optimise. This is infeasible for large n.

From the perspective of ERM, deep learning addresses both the modelling (kernel/feature choice) and computational issues faced by kernel methods. Important ideas are:

1. Instead of "feature engineering," we should do **feature learning** using **end-to-end learning**. That is, as much of the system as possible is learnt simultaneously by minimising the (regularised) empirical risk. This ensures that all parts of the system are optimised for our overall task.

2. Efficient, automated, approaches for calculating the derivatives coupled with simple *stochastic gradient descent* (SGD) optimisation algorithms allows for scalability to significantly larger datasets and model sizes.

In addition, deep learning has a number of important advantage over traditional ML frameworks:

1. Domain knowledge can be reflected in the design choices for the neural architecture and other components of the system (including data pre-processing, regularisation and optimiser) for good generalisation properties. Such inductive biases are further often form much weaker assumptions than fixed generative model approaches (e.g. traditional Bayesian models); deep learning can sometimes form an effective middle ground where we can utilize prior expertise without being overly restrictive.

2. Modularity allows for straightforwardly building complex models, known as *architectures*, from a large range of simple *modules*, along with easy reuse and combination of models.

3. Highly developed software and hardware enables easy and efficient implementation. In particular,

   a) Highly customized differentiable programming frameworks allow for easy construction of models and automated training using SGD approaches.

   b) Acceleration using graphics processing units (GPUs) and other customized hardware, along with distributed computing, provides faster and even more scalable learning.

8.2 Basic Deep Learning

The basic idea of deep learning is to construct functions (**neural networks**) as compositions of parametrized simpler functions arranged as multiple **layers**[1]

$$\begin{aligned} h^0 &= x \\ h^\ell &= f^{(\ell)}(h^{\ell-1}) \quad \text{for } \ell = 1, \dots, m \\ f(x) = h^m &= f^{(m)} \circ f^{(m-1)} \circ \cdots f^{(2)} \circ f^{(1)}(x) \end{aligned}$$

Each layer (say ℓ) can be thought of as producing a **representation** h^ℓ of the input x, with the representation of each subsequent layer being more useful for making the final prediction of the output y. Individual elements (i.e. dimensions) of each h^ℓ are known as **units**, h^0 is known as the **input layer**, h^m is known as the **output layer**, and other h^ℓ are known as **hidden layers**. Typically each h^ℓ is high-dimensional, as we want to learn **rich representations** of the input x that are useful to modelling complex input-output relationships.

Each $f^{(\ell)}$ is a differentiable, and typically simple, function and can have a vector of learnable parameters $\theta^{(\ell)}$ of length p_l. The overall function f has parameters $\theta = [\theta^{(1)}, \dots, \theta^{(m)}]$ of length $p = \sum_{\ell=1}^m p_l$. We often write f_θ to make explicit dependence on the parameters θ. Using a regulariser r, the ERM problem is then,

$$\min_\theta \quad \hat{R}(\theta) \qquad\qquad \hat{R}(\theta) = \lambda r(\theta) + \frac{1}{n}\sum_{i=1}^n L(y_i, f_\theta(x_i)) \tag{8.1}$$

A key assumption of deep learning is that all functions we work with should be differentiable (on sets of non–zero measure), and gradient descent is almost exclusively used for minimising (8.1). Starting with an initial θ_0, one can then take gradient descent steps of the form

$$\theta_t = \theta_{t-1} - \alpha \nabla_\theta \hat{R}(\theta_{t-1}), \tag{8.2}$$

where α is a step size and $\nabla_\theta \hat{R}(\theta_{t-1})$ represents the gradient of R at $\theta = \theta_t$.

Letting $L_i = L(y_i, f_\theta(x_i))$, we straightforwardly have

$$\nabla_\theta \hat{R} = \lambda \nabla_\theta r(\theta) + \frac{1}{n}\sum_{i=1}^n \nabla_\theta L_i.$$

Presuming a sensible choice of regulariser, e.g. $r(\theta) = \|\theta\|_2^2$, the $\nabla_\theta r(\theta)$ term can be straightforwardly calculated. For $\nabla_\theta L_i$, the gradients can be computed by applying the chain rule multiple times. Namely, if we let $\frac{\partial L_i}{\partial \theta^{(\ell)}}$ denote a *row* vector of the gradients

[1]Note that not all deep learning models can be written in this form as we will see later. We will refer to networks that can be written in this form (and which do not share parameters between different $f^{(\ell)}$) as **feed–forward** networks, noting though that usage of this term can vary through the literature (e.g. sometimes people use it specifically to denote the MLP model we introduce later).

for parameters in the ℓ^{th} layer and use the shorthand h_i^ℓ for the value of h^ℓ when the network input is x_i, then using the recursive definition of our network we have

$$\frac{\partial L_i}{\partial \theta^{(\ell)}} = \frac{\partial L_i}{\partial h_i^m} \frac{\partial h_i^m}{\partial h_i^{m-1}} \cdots \frac{\partial h_i^{\ell+1}}{\partial h_i^\ell} \frac{\partial h_i^\ell}{\partial \theta^{(\ell)}} \tag{8.3}$$

where

- $\frac{\partial L_i}{\partial h_i^m}$ is a *row* vector with d_m elements representing the derivative of the loss with respect to our set of output units $h_i^m = f_\theta(x_i)$ (note that even if our output is multi-dimensional, our loss is still a scalar)
- Each $\frac{\partial h_i^k}{\partial h_i^{k-1}}$ is a $d_k \times d_{k-1}$ matrix representing the Jacobian of $f^{(k)}$ with respect to the input h_i^{k-1}
- $\frac{\partial h_i^\ell}{\partial \theta^{(\ell)}}$ is a $d_\ell \times p_\ell$ matrix representing the Jacobian of $f^{(\ell)}$ with respect to its parameters $\theta^{(\ell)}$

We thus have a row vector times a series of matrix multiplications. There are now two possible orderings we might process these calculations in:

$$\frac{\partial L_i}{\partial \theta^{(\ell)}} = \left(\left(\left(\frac{\partial L_i}{\partial h_i^m} \frac{\partial h_i^m}{\partial h_i^{m-1}}\right) \cdots \frac{\partial h_i^{\ell+1}}{\partial h_i^\ell}\right) \frac{\partial h_i^\ell}{\partial \theta^{(\ell)}}\right) \tag{8.4}$$

$$= \left(\frac{\partial L_i}{\partial h_i^m} \left(\frac{\partial h_i^m}{\partial h_i^{m-1}} \cdots \left(\frac{\partial h_i^{\ell+1}}{\partial h_i^\ell} \frac{\partial h_i^\ell}{\partial \theta^{(\ell)}}\right)\right)\right) \tag{8.5}$$

While these two orderings give the same answer, they differ significant in computational cost. Namely we have that the cost of (8.4) is[2]

$$O\left(d_m d_{m-1} + d_{m-1} d_{m-2} + \cdots + d_{\ell+1} d_\ell + d_\ell p_\ell\right) = O\left(d_\ell p_\ell + \sum\nolimits_{k=\ell+1}^m d_k d_{k-1}\right)$$

while for (8.5) the cost is

$$O\left(d_{\ell+1} d_\ell p_\ell + d_{\ell+2} d_{\ell+1} p_\ell + \cdots + d_m d_{m-1} p_\ell + d_m p_\ell\right) = O\left(d_m p_\ell + \sum\nolimits_{k=\ell+1}^m d_k d_{k-1} p_\ell\right).$$

We thus see that (8.5) is roughly p_l times more costly. (8.4) computes gradients of the loss with respect to the variables and parameters in a *backward* fashion, and is called **backpropagation**. (8.5) computes the Jacobians in a forward fashion. The term forward-propagation typically refers to the computation of the function $f(x_i)$ in a forward fashion. The gradients with respect to all parameters can be computed in

[2]In many common cases, e.g. a fully connected layer, $\frac{\partial h_i^\ell}{\partial \theta^{(\ell)}}$ is sparse and so this cost can be further reduced to $O\left(p_\ell + \sum_{k=\ell+1}^m d_k d_{k-1}\right)$.

a single backward propagation phase, by computing $\frac{\partial L_i}{\partial h_i^\ell}$ and $\frac{\partial L_i}{\partial \theta^{(\ell)}}$ iteratively for $\ell = m, m-1, \dots, 1$, with

$$\frac{\partial L_i}{\partial h_i^\ell} = \frac{\partial L_i}{\partial h_i^{\ell+1}} \frac{\partial h_i^{\ell+1}}{\partial h_i^\ell}$$

for $\ell \neq m$.

Note however that back-propagation requires **activations** h_i^ℓ to be stored in the forward pass to be used in the backward pass, so is more memory intensive than computing the Jacobians in the forward pass.

8.3 Computation Graphs and Automatic Differentiation

In the basic example, $f(x)$ is obtained by composing a chain of simpler functions. In general, $f(x)$ can be computed using a **computation graph**. This is a graph, with nodes corresponding either to inputs or operators (ops in short; simpler functions) and directed edges connecting them. We assume that the graph is acyclic. Each input is a multi-dimensional array (often referred to as a **tensor**[3]), and each op is a function $(v_1, \dots, v_n) = \mathsf{op}(u_1, \dots, u_m)$ taking a list of input tensors $u_1, \dots, u_m$ and outputs a list of tensors $v_1, \dots, v_n$. An edge $\mathsf{op}_1 \to \mathsf{op}_2$ connecting two ops means that an output of op_1 is used an input of op_2 (technically we need to specify which of the outputs of op_1 and inputs of op_2, but this is typically suppressed for notational convenience).

The implementation of each op needs to construct two things: op itself, and vjp which computes the backpropagated gradients. Given tensors $\Delta_1, \dots, \Delta_n$ of same shape as $v_1, \dots, v_n$, interpreted as gradients of some objective function with respect to v_j, vjp computes the gradients with respect to the u_i's:

$$\mathsf{vjp}((u_i)_{i=1}^m, (\Delta_j)_{j=1}^n) = \left(\sum_{j=1}^n \Delta_j \cdot \frac{\partial v_j}{\partial u_i} \right)_{i=1}^m$$

where $\cdot$ denotes a dot product across elements of Δ_j and v_j,[4] and the ith entry of the resulting output is the same shape as u_i. vjp stands for **vector-Jacobian product** (in the simplest case Δ is a row vector and we multiply it with the Jacobian $\frac{\partial v}{\partial u}$). In packages that do automatic differentiation, there is also a forward mode differentiation, which implements a Jacobian-vector product:

$$\mathsf{jvp}((u_i)_{i=1}^m, (\delta_i)_{i=1}^m) = \left(\sum_{i=1}^n \frac{\partial v_j}{\partial u_i} \cdot \delta_i \right)_{j=1}^n$$

[3]These are not real tensors used in physics, in that we do not distinguish between covariant and contravariant indices.

[4]In most systems, the first dimension for each of Δ_j, v_j, and u_j corresponds to different datapoints in the batch (for which all calculations are completely separate), such that this dot product sums across all dimensions in the tensors except the first.

Here δ_i is a tensor of the same shape as u_i and interpreted as the gradient of u_i with respect to some scalar.

The op-level computations can now be chained together for computations over the whole graph. The forward pass computes each op once its inputs are all given or computed (in a so-called topological order), and the backward pass computes the gradients using the corresponding VJPs, in reverse order on the graph. For forward mode differentiation, both the op and the JCPs are computed together. However the gradients of each u_i with respect to each element of the input nodes that feed into u_i needs to be computed and kept track of.

As before, the overall computational cost for forward mode differentiation is linear in the dimensionality of the inputs of the graph, while backward mode differentiation has computational cost linear in the dimensionality of the outputs. In machine learning the final output is almost always a single dimension (the objective) so backward mode differentiation is much more predominant.

Note that each op in the computation graph can itself be implemented using its own computation graph, by specifying the inputs and outputs and abstracting away the inner ops. Once a procedure is implemented to create a computation graph, this can then be used to create multiple ops that are used in a larger computation graph. This notion of **modularity** is important in allowing very complex computation graphs to be constructed and used in deep learning. In fact the whole endeavour can be thought of as a new way of engineering complex software, called **differentiable programming**, which has wider implications for computer science as a field and driving much research across many fronts.

8.4 Basic Modules

There are a huge variety of layers/modules that are in use across deep learning. Here we will describe a selection of the more popular ones.

To help with exposition, we define three kinds of objects used to construct a computation graph

op A *node* in a computation graph. Given an input, its output will be a block of (hidden) unit values. We can think of ops as *evaluations* of functions; they have no parameters of their own.

module A *function* that can applied multiple times within the graph. It may be *parameterised*, in which case the parameters are *shared* across uses. Applying a module creates an op.

factory A procedure that *generates* modules. This allows us to produce multiple modules that have separate sets of parameters

For example, consider a factory $\mathsf{Factory}(n, \sigma)$ that generates functions of the form $f : \mathbb{R}^n \to \mathbb{R}^n$, $f(x) = \sigma(Wx)$ where $W \in \mathbb{R}^{n \times n}$ and σ is an elementwise non-linearity. The following generates two such functions, and applies each twice to a vector x:

$$\begin{aligned}
\mathsf{module}_1 &\sim \mathsf{Factory}(n, \sigma) \\
\mathsf{module}_2 &\sim \mathsf{Factory}(n, \sigma) \\
\mathsf{op}_1 &= \mathsf{module}_1(x) \\
\mathsf{op}_2 &= \mathsf{module}_1(\mathsf{op}_1) \\
\mathsf{op}_3 &= \mathsf{module}_2(\mathsf{op}_2) \\
\mathsf{op}_4 &= \mathsf{module}_2(\mathsf{op}_3)
\end{aligned}$$

The two modules are functions of the same form, but with two different sets of parameters, say W_1 and W_2. We use $\sim$ in the above as module_1 and module_2 are different objects, with different parameters that are initially randomly drawn from the same distribution given in $\mathsf{Factory}(n, \sigma)$. The computation graph consists of 4 ops, two sets of parameters, and 1 input vector, and produces:

$$\sigma(W_2\sigma(W_2\sigma(W_1\sigma(W_1 x))))$$

- **linear module**:

  $$\begin{aligned}
  \mathsf{module} &\sim \mathsf{Linear}(m, n) \\
  \mathsf{module}(x) &= Wx + b
  \end{aligned}$$

  where $x \in \mathbb{R}^n$ is the input, and $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are parameters called **weights** and **biases** respectively. It is specified by the input and output dimensions n and m. In some frameworks n is typically not directly provided and inferred from the inputs into f.

- **nonlinearity**. These are simply take an array as input and apply a nonlinear function to each element of the array. They don't have learnable parameters. For example:

  $$\begin{aligned}
  \mathsf{sigmoid}(x) &= 1/(1 + \exp(-x)) \\
  \mathsf{tanh}(x) &= (\exp(x) - \exp(-x))/(\exp(x) + \exp(-x)) \\
  \mathsf{ReLU}(x) &= \max(0, x) \\
  \mathsf{softplus}(x) &= \log(1 + \exp(x)) \\
  \mathsf{swish}(x) &= x\mathsf{sigmoid}(x) \\
  \mathsf{ELU}_\alpha(x) &= \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}
  \end{aligned}$$

  A **softmax** is also a nonlinearity (but not elementwise):

  $$\mathsf{softmax}([x_1, \ldots, x_d]) = \left[\frac{\exp(x_1)}{\sum_{i=1}^d \exp(x_i)}, \ldots, \frac{\exp(x_d)}{\sum_{i=1}^d \exp(x_i)}\right]$$

- **2D convolutional module** (Conv2D). The most common convolutional factory produces 2D convolutional modules that have 3D inputs and outputs. Here a **convolutional** kernel is applied to the first two dimensions of th input and the third input dimension is a number of **input channels** that are summed over. This is then done separately for each **output channel** to produce the output, specifically we have

$$\begin{aligned} \mathsf{module} &\sim \mathsf{Conv2D}(c_{\text{in}}, c_{\text{out}}, d_1, d_2) \\ x' &= \mathsf{module}(x) \\ x'_{i'j'k'} &= \sum_{i=1}^{d_1}\sum_{j=1}^{d_2}\sum_{k=1}^{c_{in}} w_{ijkk'}\, x_{i'+i-1,j'+j-1,k} \quad \forall i', j', k' \end{aligned}$$

  Technically, this is a cross-correlation, not a convolution! The two dimensional array given by $\{w_{ijkk'}\}_{i=1:d_1,j=1:d_2}$ is known as a **filter**, with each input–output channel pair (i.e. each kk') having their own filter. These filters then form the parameters of the module, akin to the weights of a linear module. In fact, a convolutional module can be viewed as special case of a linear module with a sparse weight matrix (along with reshaping of the input and output tensors).

  Additional variations include:

  – There can be a bias for each output channel.

  – There can be padding around edges of input (e.g. by 0s), so that output has same height and width as input.

  – There can be striding, e.g. for 2x3 strides only every 2nd row and 3rd column of x' is computed (and x' is reduced 2x3 times in size).

  A **depthwise spatial convolution** applies a separate convolution kernel to each channel of x separately (i.e. there is a one to one mapping between input and output channels with no summing across channels).

  A 1x1 **convolution** or **pointwise convolution** is one whose kernel has height 1 and width 1. It is equivalently to taking the channels at each (i, j) location as a vector, and multiplying by a parameter matrix of size $C' \times C$. There are also 1D and 3D convolutions.

- **max-pooling**: For each channel, this takes the maximum value over the elements of x that would have participated in a 2D depthwise convolution:

$$\begin{aligned} x' &= \mathsf{MaxPool}_{d_1,d_2}(x) \\ x'_{i'j'k'} &= \max_{i=1}^{d_1}\max_{j=1}^{d_2} x_{i+d_1(i'-1),\, j+d_2(j'-1),\, k'}. \end{aligned}$$

  Max-pooling technically can be considered as a nonlinearity. **Average-pooling** applies an average instead of max.

8.5 Larger Modules

New and more complex modules can be constructed from simpler ones. Popular examples are:

- **Multi-layer perceptrons** (MLP) simply chain together multiple *fully-connected* linear modules (i.e. linear modules with a full weight matrix) and nonlinearities. An MLP is specified by the number of fully-connected layers, the sizes of each layer, and nonlinearity.

$$f \sim \mathsf{MLP}(d_m, d_{m-1}, \dots, d_0, \sigma) \qquad \begin{cases} f^{(\ell)} \sim \mathsf{Linear}(d_\ell, d_{\ell-1}) \quad \text{for } \ell = 1, \dots, m \\ \quad f = f^{(m)} \circ \sigma \circ f^{(m-1)} \circ \cdots \circ \sigma \circ f^{(1)} \end{cases}$$

$$f(x) = W_m \sigma(W_{m-1}\sigma(\cdots W_2\sigma(W_1 x + b_1) + b_2 \cdots) + b_{m-1}) + b_m \tag{8.6}$$

  The outputs of each σ op is a vector, each entry of which is called an **activation**,[5] corresponding to the value of a **hidden unit** of the MLP. The input into the σ op is a vector of the **pre-activations**. One can also, at least in principle, construct an MLP with different **activation functions** σ_ℓ at different layers, but this is rarely done in practice.

- **Recurrent neural networks** (RNNs) are NNs that process inputs and outputs in a sequence. In the simplest setting, we have a sequence of inputs $x_1, x_2, \dots$ and want to predict a corresponding sequence of outputs $y_1, y_2, \dots$. We would like each prediction for y_t to depend on the sequence of inputs so far $x_{1:t}$. RNNs do this by producing a sequence of representations h_t of the sequence history $x_{1:t}$. h_t is a function of $x_{1:t}$ that can be computed recursively from h_{t-1} and x_t. RNNs use two modules, an encoder encode and a decoder decode, and, starting with an initial state h_0, computes as follows:

$$\begin{aligned} \mathsf{encode} &\sim \mathsf{EncoderFactory} \\ \mathsf{decode} &\sim \mathsf{DecoderFactory} \\ h_t &= \mathsf{encode}(h_{t-1}, x_t) \\ \hat{y}_t &= \mathsf{decode}(h_t) \end{aligned}$$

  We can also think of h_t as the system's memory, with the encoder being the RNN's procedure to update its memory based on new information in x_t. In the simplest case both encode and decode are dense layers followed by a nonlinearity (or MLPs).

- **Long short-term memory** (LSTM) is a particular type of RNN. It was designed specifically to address gradient explosion/vanishing problems in standard RNNs (see Section 8.6). In addition to h_t, it adds another hidden state called the cell state C_t. Both h_t and C_t can be thought of as memories of the LSTM, with C_t

[5] There is some inconsistency in how the term "activation" is used in the literature with some, including the previous machine learning module, using it to refer to the inputs of σ rather than that the output. However, the latter is the overriding modern usage, hence our usage.

playing the role of the long term memory, and h_t the short term memory. The LSTM module is defined as follows:

$$\begin{aligned}
\mathsf{forget\_gate}_t &= \mathsf{sigmoid}(W_f[h_{t-1}, x_t] + b_f) \\
\mathsf{insert\_gate}_t &= \mathsf{sigmoid}(W_i[h_{t-1}, x_t] + b_i) \\
C'_t &= \mathsf{tanh}(W_C[h_{t-1}, x_t] + b_C) \\
C_t &= \mathsf{forget\_gate}_t * C_{t-1} + \mathsf{insert\_gate}_t * C'_t \\
\mathsf{output\_gate}_t &= \mathsf{sigmoid}(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= \mathsf{output\_gate}_t * \mathsf{tanh}(C_t)
\end{aligned}$$

where $*$ indicates an element-wise product. The basic operation is as follows: given the current hidden state h_{t-1} and input x_t, the system decides what parts of the cell state C_t to forget via $\mathsf{forget\_gate}_t$, what new information C'_t to insert into it via $\mathsf{insert\_gate}_t$, and what to output to the new state h_t via the $\mathsf{output\_gate}_t$.

8.6 Optimisation

Optimising neural networks is difficult, and a lot of research effort has been devoted to optimisation algorithms and techniques specially tailored to deep learning settings.

Recall the ERM problem (8.1), reproduced here (with L2 regularization):

$$\min_\theta \quad \hat{R}(\theta) \qquad\qquad \hat{R}(\theta) = \frac{\lambda}{2}\|\theta\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} L(y_i, f_\theta(x_i)) \tag{8.7}$$

where f_θ is a neural network with parameters θ. Automatic differentiation tools allows the gradient $\nabla_\theta \hat{R}(\theta)$ to be computed straightforwardly, so gradient based optimisation algorithms can be employed. However there are still many difficulties, both computationally, and in terms of the complexity of navigating the "loss landscape".

The first and more tractable difficulty is computation. Computing the gradient of (8.1) scales linearly in the dataset size n. Many datasets of interest in deep learning are very large, such that computing $\nabla_\theta \hat{R}$ is impractical even with modern GPUs and TPUs. The solution is to replace the $\nabla_\theta \hat{R}$ with a stochastic estimate, say $\widehat{\nabla_\theta \hat{R}}$, so the **stochastic gradient descent** (SGD) algorithm is:

$$\theta_t = \theta_{t-1} - \alpha_t \widehat{\nabla_\theta \hat{R}}(\theta_{t-1}) \tag{8.8}$$

where α_t is an iteration dependent step size. In expectation, the stochastic gradients need to be moving in the "downhill' direction. An unbiased estimate of the gradient $\nabla_\theta \hat{R}(\theta)$ can be obtained by using a small subsample of the dataset, called a **minibatch**. Let B be a random subset of $\{1, \ldots, n\}$ with $|B| \ll n$. Then:

$$\widehat{\nabla_\theta \hat{R}}(\theta) = \lambda\theta + \frac{1}{|B|}\sum_{i\in B} \frac{dL(y_i, f_\theta(x_i))}{d\theta}$$

Note that the variance of the stochastic gradient scales as $1/|B|$. Typically the dataset is split into $n/|B|$ minibatches, and each is used in turn. Once the whole dataset has been processed once (called an **epoch**), the split into minibatches is randomised for the next epoch, with the processing continuing until out computational budget is exceeded, we are satisfied that the training is sufficiently converged, or some other termination criterion is met.

SGD is very well studied in the optimisation literature and there is a wealth of powerful results, particularly for strongly convex objectives. While some of the intuitions gained are relevant in the deep learning setting, many problems faced in optimising deep learning models are precisely because the objectives are not well-behaved. Unlike linear regression or kernel methods, f_θ for NNs can be a highly complex nonlinear function of θ, and $\hat{R}(\theta)$ is not convex in θ. It can have multi-modality, be ill-conditioned with ridges, valleys and flat plateaus.

As an example of the complexity, consider the gradients for a MLP (8.6), which are given by (8.4). This is given by a product of a sequence of matrices. If the matrices have eigenvalues that are above 1, there could be directions where the product become very large, this is called **exploding gradients**. Conversely, if there are eigenvalues that are small, the gradients can **vanish**. Such exploding/vanishing gradient problems are particularly severe in deep models with many layers since the resulting gradients are products of many matrices. In particular, the chain of encode ops in an RNN can be thought of as producing a very deep network, such that their gradients tend to be particularly badly behaved. Part of the design of the LSTM cell state is as a way to mitigate exploding/vanishing gradients. Each cell state C_t relates to the previous C_{t-1} only via the forget gates, so the resulting gradient is just a product of the forget gates, which take values in $(0, 1)$ and will not explode. They can vanish over long time horizons though, so "long term credit assignment" is still difficult in LSTMs.

8.6.1 SGD Guidance

In practice, the step size or learning rate α_t is one of the most important tuning parameters of a deep learning system. Too small a step size and convergence can become very slow, too large and the optimisation can become unstable or even diverge. A good rule of thumb is find the smallest step size such that SGD becomes unstable, then reduce it by a factor of 2-10. Alternatively, find the largest stable step size.

SGD tends to behave as follows: initially, when the parameters are far away from a good mode, it is easy to estimate the gradient from the minibatch and the stochastic gradient tends to point in the same direction as the gradient, so that SGD tends to get to the vicinity of a mode quickly. Once it is close to a minimum the magnitude of the true gradient decreases, the behaviour of the algorithm starts being dominated by the stochasticity, and SGD will "jump around" the local minimum. Small step sizes suppress the variance of the stochastic gradients, and it is typical to decrease the step size over the course of the optimisation. For example decreasing it by a constant factor, say by half every 10 epochs, or whenever SGD "looks like it got stuck". Schemes for setting the step size are known as **schedulers**.

Early analysis of SGD used step sizes that decrease slowly to guarantee convergence, most notably ensuring that the **Robbins–Monro** conditions are satisfied:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad\qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty.$$

For example, $\alpha_t = a/(b+t)$ for positive constants a and b. However such step size schedules are now not popular anymore. This is for two reasons. Firstly, while such schedules decrease slowly for large t, they decrease rapidly initially so learning can be very slow. Secondly, the slow decrease has the effect of simulated annealing with a slowly decreasing temperature, and the converged minimum tends to be "sharp and deep". There is much evidence that such minima tend to generalise poorly. In fact it has been argued that the stochasticity in SGD has a regularising effect. If the step size is large enough, SGD will be oblivious to such bad minima and converge to wider minima that generalise better. Note that the step size schedule in the previous paragraph decreases exponentially quickly and does not satisfy the requirements here. Other recent schedules like cosine and linear decay also do not satisfy the requirements.

8.6.2 Momentum and ADAM

While the stochasticity of SGD can have a regularising effect, it is still useful to reduce the stochasticity, so that larger learning rates can be used (with effectively the same variance, the convergence can be faster). A popular approach is using **momentum** to average out the stochastic gradients over previous minibatches:

$$\begin{aligned} \Delta_t &= \beta\Delta_{t-1} + (1-\beta)\widehat{\nabla_\theta \hat{R}}(\theta_t), \qquad 0 < \beta < 1 \\ \theta_t &= \theta_{t-1} - \alpha_t \Delta_t \end{aligned}$$

The average is using an exponentially decaying weight over past minibatch gradients. Momentum can also help in narrow valleys, as the direction along the valley tends to get small gradients which can be accumulated by the momentum.

Another issue with standard SGD is that it uses a single step size for all parameters. Because of the nonlinearity of the function f_θ, different parameters may have different scales and may need different step sizes. **ADAM** is a popular optimiser that does this by keeping weighted moving average estimates for both the gradient (Δ_t) and the *squared* gradient (V_t), scaling the step size using the latter (which is related to the curvature of the loss landscape) as follows

$$\begin{aligned} \Delta_t &= \beta_1\Delta_{t-1} + (1-\beta_1)\widehat{\nabla_\theta \hat{R}}(\theta_{t-1}) & \tilde{\Delta}_t &= \frac{\Delta_t}{1-(\beta_1)^t} \\ V_t &= \beta_2 V_{t-1} + (1-\beta_2)\left(\widehat{\nabla_\theta \hat{R}}(\theta_{t-1})\right)^2 & \tilde{V}_t &= \frac{V_t}{1-(\beta_2)^t} \\ \theta_t &= \theta_{t-1} - \frac{\alpha_t}{\sqrt{\tilde{V}_t}+\epsilon}\tilde{\Delta}_t \end{aligned}$$

where the division on the last line is applied element–wise and $0 < \beta_1, \beta_2 < 1$.

Both Δ_0 and V_0 are initialised at 0, and the division by $1 - \beta^t$ ensures that the exponentially decaying weights for averaging sum to 1 (i.e. it accounts for the fact that the earlier steps are averaging over fewer gradients because of the zero initialization). Note that $\tilde{\Delta_t}$ and $\tilde{V_t}$ quickly converge to just Δ_t and V_t respectively as t increases. Typical values for β_1 and β_2 can be surprisingly large, e.g. 0.99 and 0.999 respectively, such that large amounts of historical gradient information is accumulated and the gradients change quite slowly.

8.7 Initialisation and Parameter Scales

The parameters of a neural network are almost exclusively initialised randomly, typically with zero-mean Gaussian distributions. Random initialisation is important to "break symmetry" so that each unit in a layer will learn differently. Variances of the initialisation are chosen so that at initialisation the network does something reasonable.

Consider again the simple MLP (8.6). Ignoring the nonlinearities, the pre-activation going into a unit h_j^ℓ has the form:

$$\tilde{h}_j^\ell = \sum_{k=1}^{d_{\ell-1}} W_{jk}^\ell h_k^{\ell-1} + b_j^\ell$$

It is generally a good idea to control the typical scale of $\tilde{h}_j^\ell$ to be $O(1)$. Many nonlinearities (e.g. $\mathsf{sigmoid}, \mathsf{tanh}, \mathsf{softplus}$) are designed to have their nonlinear regimes around $[-2, 2]$, so if $\tilde{h}_j^\ell$ is much bigger or small than $O(1)$, the nonlinearities will look like linear functions, or worse, constant functions (constant functions will have zero gradient so the unit will not be able to learn due to vanishing gradient problem).

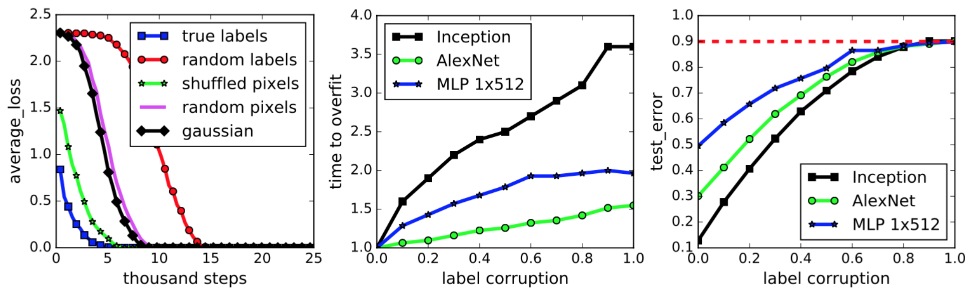
We can control the typical scales of pre-activations layer by layer. Suppose each $h_k^{\ell-1} = O(1)$, and suppose they are close to being independent (which will typically be a reasonable assumption at the point of initialization). Then $\mathrm{Var}(\tilde{h}_j^\ell) = O(d_{\ell-1}\sigma_{\ell w}^2 + \sigma_{\ell b}^2)$ where $\sigma_{\ell w}^2$ is the variance of the weights and $\sigma_{\ell b}^2$ the variance of the biases, for layer ℓ. If we set $\sigma_{\ell w}^2 = O(1/d_{\ell-1})$, we can get $\mathrm{Var}(\tilde{h}_j^\ell) = O(1)$. Using $\sigma_{lw}^2 = 1/d_{\ell-1}$ at initialisation is known as **Xavier initialisation**.

8.8 Regularisation

The most common classical regulariser used in deep learning is L_2 norm regularisation of the empirical risk. This is typically called **weight decay**, as its effect is to decay each parameter by $(1 - \alpha_t \lambda)$ at each iteration.

Other more algorithmic forms of regularisation are also often used in deep learning. For example, we have seen that the stochasticity of SGD can be thought of as a form of regularisation.

Another popular regularisation via randomisation method is called **dropout**, which works as follows: during training, for each data item, and for each unit in the network,



(a) learning curves (b) convergence slowdown (c) generalization error growth

Figure 1: Fitting random labels and random pixels on CIFAR10. (a) shows the training loss of various experiment settings decaying with the training steps. (b) shows the relative convergence time with different label corruption ratio. (c) shows the test error (also the generalization error since training error is 0) under different label corruptions.

Figure 8.1: Figure from [59].

randomly remove it from the network (with probability p, typically $p = 0.5$) by multiplying its activation by 0. The reduced networks are then trained using standard SGD. Dropout can be thought of as a cheap and exponentially large ensemble of neural networks, with each network being constructed by randomly dropping out units of a main network.

Another popular regularisation technique is **early stopping**. This involves monitoring the validation loss during training, and stopping when it starts going up. Controlling the amount of computation is a generic and actively studied way to control the complexity of models, both in deep learning and more generally.

8.9 Other Observations

A well known classical result is that an MLP with a single hidden layer and a single element–wise non–linearity can approximate any well–behaved function arbitrarily well if it has enough hidden units (see examples sheet). More generally, there is common folk wisdom that the hypothesis classes of most modern neural networks are extremely rich. In fact, while technically the loss landscape is highly complex and multi-modal, recent empirical observations show that unregularized modern deep neural networks are effectively always capable of overfitting the training data, even random datasets (see Figure 8.1). This can reduce the impact from the imperfect nature of the optimization process (and indeed show that these imperfections are potentially even important) as it can mean we often hit issues with overfitting earlier than those from limitations in the optimization process itself (though this is certainly not always the case).

In general, using larger deep learning models almost never hurts in practice provided our training process remains stable. In fact, recent empirical and theoretical results show a curious "double descent" style learning curve for large models [4] (see Figure 8.2), where

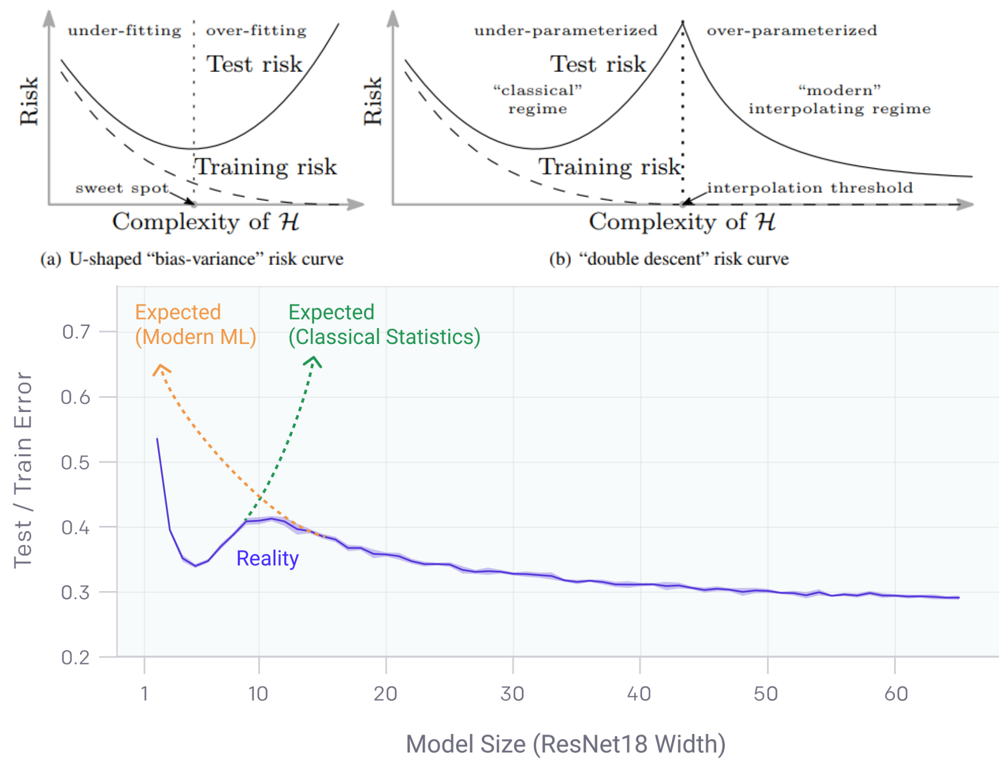


Figure 8.2: Figures from [4] and OpenAI blog on "Deep Double Descent".

the best generalisation performance is obtained for the most overparameterised models. All these results show that classical statistical and learning theory fail to explain the behaviour of modern deep learning models, and these empirical findings have driven much exciting theoretical research in recent years. It is also means that much of deep learning becomes a somewhat empirical science, in the sense that many of the mechanisms underlying its practical success are not currently well explained at a first–principles level, such that we must often be guided most by what seems to work well in practice.

To conclude, we summarize some important points on deep learning

- More data is typically better than fancier models (as long as your model is not under capacity); this is also often true of machine learning more generally. As such it can often be useful to check simple baselines first.
- To ensure that our approach is effective for large datasets, it can be best to start with a class of models that ensures convenient properties for training and scaling (e.g. differentiable computational graphs and deep neural networks) rather than purely prioritizing the low–level fidelity.
- Whole system learning, or end-to-end learning, is typically beneficial: we should

look to optimize all aspects of the machine learning pipeline simultaneously, rather than just individual components.

- There is no clear distinction between model and computation: the only thing that really matters is the final model we learn and the amount computation required to train it, we should thus consider the whole process as a single problem.
- Everything is a regulariser: avoiding direct regularization of parameters (e.g. avoiding using a specific RKHS norm) can help avoid the introduction of unwanted strong inductive biases, with implicit algorithmic approaches to regularization (e.g. early stopping) often providing more flexibility while still preventing overfitting.
- If a deep learning model is properly set up it should be able to overfit when there is insufficient training data. It can thus often be useful when to take a small subset of our data (e.g. a single minibatch) and check that our network and training scheme is able to overfit to it. This can often catch issues in the optimizer, model capacity, or data pre–processing early. Checking a model that has less sensitivity to the exact set up/hyperparameters, such as a random forest,[6] can also be useful to calibrate the level of performance that might be achievable, and thus whether the performance of our network is sensible.

[6]Random forests and their derivatives tend to work very well "out–of–the–box" [46], that is without any tuning to the particular problem. This often makes them a useful test bed and first benchmark before implement more complicated models like deep learning architectures; they are generally very quick to train and require effectively no inputs other than the data, such that they are trivial to use.

9 Latent Variable Models and Variational Approaches

K-means and PCA are two popular unsupervised learning methods. Their goals are for summarising and visualising data. A wider class of methods for unsupervised learning are based on **latent variable models**, where the goal is to model the data generating process using latent variables, with the latent variables representing a summary of the data. For example, in mixture models, the latent variables correspond to assignments of data items to clusters, while in variational auto-encoders, the latent variables are lower dimensional representations of data. Latent variable models are based on a probabilistic generative modeling approach, which provides a rich toolbox of techniques, and a coherent principle to reason about, and model, complex data and to derive new algorithms.

9.1 Latent Variable Models

A **latent variable model** (**LVM**) is a probabilistic generative model where each datapoint x_i has a corresponding **latent variable** z_i. For data $\mathbf{X} = \{x_i\}_{i=1}^n$ and latents $\mathbf{Z} = \{z_i\}_{i=1}^n$, an LVM is given by

$$p_\theta(\mathbf{X}, \mathbf{Z}) = p_\theta(\mathbf{Z}) \prod_{i=1}^n p_\theta(x_i|z_i) \tag{9.1}$$

where θ are global variables that are usually treated deterministically. Occasionally we take a Bayesian approach for θ instead, giving

$$p(\theta, \mathbf{X}, \mathbf{Z}) = p(\theta)p(\mathbf{Z}|\theta) \prod_{i=1}^n p(x_i|z_i, \theta). \tag{9.2}$$

In both cases, the underlying assumption is that each datapoint x_i can be explained by its latent variable z_i in conjunction with the global parameters θ. Each z_i is often lower dimensional, simpler, and/or more interpretable than the corresponding x_i, thereby providing a useful **representation** of the datapoint.

If we assume that our data is i.i.d. given θ, this implies that all the z_i should also be conditionally independent of each other given θ. This is typically a reasonable assumption when there is no natural ordering to the data. Moreover, it can be an extremely powerful assumption to make as it needs a number of important factorizations that we will exploit throughout the chapter. The resulting class of models are known as **factorized** LVMs and have joint distributions

$$p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{i=1}^{n} p_\theta(z_i) p_\theta(x_i | z_i)$$

We will focus mostly on such factorized LVMs.

9.1.1 Clustering and Mixture Modelling

K-means and hierarchical clustering are non-probabilistic algorithms, based on intuitive notions of clustering “similar" instances together and “dissimilar" instances apart. Their goal is not to model the probability of the observed data items. In contrast, a **mixture model** is a probabilistic counterpart which described a full generative process of data, and can produce “soft” assignment of data items to clusters. Mixture models assume that our dataset $\mathbf{X}$ was created by sampling iid from K distinct populations (called **mixture components**). In other words, datapoints come from a mixture of several sources and the model for the data can be viewed as a convex combination of several distinct probability distributions, often modelled with a given parametric family.

Samples in population k can be modelled using a distribution F_k with density $f_k(x; \lambda_k)$, where λ_k is the *model parameter* for the k-th component. For a concrete example, consider a p-dimensional multivariate normal density with unknown mean μ_k and *known diagonal* covariance $\sigma^2 I$ (such that $\lambda_k = \mu_k$),

$$f_k(x; \lambda_k) = |2\pi\sigma^2|^{-\frac{p}{2}} \exp\left(-\frac{1}{2\sigma^2} \|x - \mu_k\|_2^2 \right). \tag{9.3}$$

More generally, a mixture model corresponds to the following generative model, whereby for each data item $i = 1, 2, \ldots, n$, we

(i) first sample the **assignment variable** (independently for each data item i):

$$Z_i \overset{iid}{\sim} \text{Discrete}(\pi_1, \ldots, \pi_K) \qquad \text{i.e., } \mathbb{P}(Z_i = k) = \pi_k$$

where for $k = 1, \ldots, K$, $\pi_k \geq 0$, such that $\sum_{k=1}^{K} \pi_k = 1$, are the **mixture weights** (also known as the **mixing proportions**), which are additional model parameters to be inferred;

(ii) then, given the assignment $Z_i = k$ of the mixture component, $X_i = (X_i^{(1)}, \ldots, X_i^{(p)})^\top$ is sampled (independently) from the corresponding k-th component:

$$X_i | (Z_i = k) \sim f_k(x; \lambda_k).$$

We observe $X_i = x_i$ for each i but do not observe its assignment Z_i (latent variables), and would like to learn the parameters $\theta = (\lambda_1, \ldots, \lambda_K, \pi_1, \ldots, \pi_K)$ as well as infer the latent variables.

Here the log joint density of the model is given by

$$p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{i=1}^{n} \pi_{z_i} f_k(x_i; \lambda_{z_i}) \tag{9.4}$$

from which we can marginalize over the latent variables to give the **marginal distribution**

$$p_\theta(\mathbf{X}) = \sum_{z_1=1}^{K} \cdots \sum_{z_n=1}^{K} \prod_{i=1}^{n} \pi_{z_i} f_k(x_i; \lambda_{z_i}) = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \pi_k f_k(x_i; \lambda_k) \right). \tag{9.5}$$

Once $\mathbf{X}$ is observed, this forms the marginal likelihood of the model, or equivalently the model evidence. The log of this marginal likelihood, sometimes called the **marginal log likelihood**, is quantity that will be of particular interest:

$$\ell(\theta) := \log p_\theta(\mathbf{X}) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k f_k(x_i; \lambda_k).$$

Another important quantity is the **posterior distribution**,

$$p_\theta(\mathbf{Z}|\mathbf{X}) = \frac{p_\theta(\mathbf{X}, \mathbf{Z})}{p_\theta(\mathbf{X})} = \prod_{i=1}^{n} p_\theta(z_i|x_i) \tag{9.6}$$

which factorises into posterior distributions of individual latent variables z_i given the corresponding observations x_i in case of K-means, as our data is assumed iid.

In the context of latent variable models, there are two important processes: **inference**, which is the computation or approximation of the posterior distribution over latent variables, and learning, which is the estimation of the model parameters θ. Typically the learning algorithm is done via maximising the marginal log likelihood (or some approximation thereof), which generally quires some form of inference to be conducted alongside or as a subroutine. However, direct maximisation is often not feasible and the marginal log likelihood will often have many local optima. Fortunately, there is a general approach for both learning and inference based on the formulation of an objective function called the *evidence lower bound* (ELBO), which is a lower bound on the marginal log likelihood. We will find that the ELBO permits both model learning and inference, either in alternating fashion through the *Expectation Maximisation (EM) algorithm*, or simultaneously, via the *variational auto-encoder* framework.

9.2 KL Divergence and Gibbs' Inequality

Before we describe the variational formulation, we will review the notion of **Kullback-Leibler (KL) divergence** or **relative entropy** between probability distributions P and Q.

KL divergence.

- Let P and Q be two absolutely continuous probability distributions on $\mathcal{X} \subseteq \mathbb{R}^d$ with densities p and q respectively. Then the KL divergence *from* P *to* Q is defined as
$$\mathbb{D}_{\mathrm{KL}}\left(P \parallel Q\right) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} dx. \tag{9.7}$$
- Let P and Q be two discrete probability distributions with probability mass functions p and q respectively. Then the KL divergence *from* P *to* Q is defined as
$$\mathbb{D}_{\mathrm{KL}}\left(P \parallel Q\right) = \sum_i p(x_i) \log \frac{p(x_i)}{q(x_i)}. \tag{9.8}$$

In both cases, we can write
$$\mathbb{D}_{\mathrm{KL}}\left(P \parallel Q\right) = \mathbb{E}_p \left[\log \frac{p(X)}{q(X)}\right], \tag{9.9}$$
where $\mathbb{E}_p$ denotes that expectation is taken over p. By convexity of $f(x) = -\log(x)$ and Jensen's inequality (9.11), we have that
$$\mathbb{D}_{\mathrm{KL}}\left(P \parallel Q\right) = \mathbb{E}_p \left[-\log \frac{q(X)}{p(X)}\right] \geq -\log \mathbb{E}_p \frac{q(X)}{p(X)} = 0, \tag{9.10}$$
where in the last step we used that $\int_{\mathcal{X}} q(x)dx = 1$ in continuous case and $\sum_i q(x_i) = 1$ in discrete case.

Jensen's inequality. Let f be a convex function and X be a random variable. Then
$$\mathbb{E}\left[f(X)\right] \geq f\left(\mathbb{E}X\right). \tag{9.11}$$
If f is strictly convex, then equality holds if and only if X is almost surely a constant.

Thus, we conclude that KL-divergence is always non-negative. This consequence of Jensen's inequality is called **Gibbs' inequality**. Moreover, since $f(x) = -\log(x)$ is strictly convex on $x > 0$, the equality holds if and only if $p(x) = q(x)$ almost everywhere, i.e. $P = Q$. Note that in general KL-divergence is *not symmetric*: $\mathbb{D}_{\mathrm{KL}}\left(P \parallel Q\right) \neq \mathbb{D}_{\mathrm{KL}}\left(Q \parallel P\right)$.

9.3 Variational Free Energy and the EM Algorithm

We now break from the structuring of the lectures/slides—where we introduce ELBOs through the eyes of variational inference—to provide an alternative viewpoint through the **expectation maximization algorithm**, more commonly known as simply the **EM algorithm**. If you have not already done so, it is recommend that you go through the lectures/slides before this section, as the viewpoint there will be the main one we consider and the one that is most prevalent in modern research and applications; with the following given as an alternative (somewhat more "old–school") perspective.

The EM algorithm is a general purpose iterative strategy for local maximisation of the marginal likelihood under missing data/hidden variables. The method has been proposed many times for specific models—it was given its name and studied as a general framework by [16].

Let $\mathbf{X}$ be a set of observed variables and let $\mathbf{Z}$ be a set of latent variables. Our probabilistic model is given by $p_\theta(\mathbf{X}, \mathbf{Z})$, where $\mathbf{Z}$ is a set of unknown variable we need to infer through Bayesian inference. We would like to maximise the marginal log-likelihood $\ell(\theta) = \log p_\theta(\mathbf{X}) = \log \int p_\theta(\mathbf{X}, \mathbf{Z}) d\mathbf{Z}$ with respect to θ. However, marginalisation of latent variables typically results in an intractable optimization problem and we need to resort to approximations or find some other way around this intractability.

Now, assume for a moment that we have access to another objective function $\mathcal{L}(\theta, q)$, where $q(\mathbf{Z})$ is a certain distribution on $\mathbf{Z}$, which we are free to choose and will call **variational distribution**. Moreover, assume that $\mathcal{L}$ satisfies

$$\mathcal{L}(\theta, q) \leq \ell(\theta) \text{ for all } \theta,\ q, \tag{9.12}$$

$$\max_q \mathcal{L}(\theta, q) = \ell(\theta), \tag{9.13}$$

i.e. $\mathcal{L}(\theta, q)$ is a *lower bound on the log-likelihood* for any variational distribution q (9.12), which also *matches the log-likelihood* at a particular choice of q (9.13).

Given these two properities, we can construct an alternating maximisation: *coordinate ascent* algorithm as follows:

Coordinate ascent on the lower bound. For $t = 1, 2 \ldots$ until convergence:

$$q^{(t)} := \operatorname{argmax}_q \mathcal{L}(\theta^{(t-1)}, q)$$
$$\theta^{(t)} := \operatorname{argmax}_\theta \mathcal{L}(\theta, q^{(t)}).$$

Theorem 16. *Assuming* (9.12) *and* (9.13)*, coordinate ascent on the lower bound* $\mathcal{L}(\theta, q)$ *does not decrease the log likelihood* $\ell(\theta)$*.*

Proof. $\ell(\theta^{(t-1)}) = \mathcal{L}(\theta^{(t-1)}, q^{(t)}) \leq \mathcal{L}(\theta^{(t)}, q^{(t)}) \leq \mathcal{L}(\theta^{(t)}, q^{(t+1)}) = \ell(\theta^{(t)})$. □

But how can we find such lower bound $\mathcal{L}$? It is given by the so called *variational free energy*, which we define next.

Definition 17. Variational free energy in a latent variable model $p_\theta(\mathbf{X}, \mathbf{Z})$ is defined as

$$\mathcal{L}(\theta, q) = \mathbb{E}_{\mathbf{Z}\sim q}[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q(\mathbf{Z})], \tag{9.14}$$

where q is any probability density/mass function over the latent variables $\mathbf{Z}$.

The variational free energy is equivalent to the concept of the **ELBO** as covered in lectures and to which we will return to later in the notes.

Consider the KL divergence between $q(\mathbf{Z})$ and the true conditional based on our model $p_\theta(\mathbf{Z}|\mathbf{X}) = p_\theta(\mathbf{X}, \mathbf{Z})/p_\theta(\mathbf{X})$ for the observations $\mathbf{X}$ and a fixed parameter vector θ. Since KL is non-negative,

$$\begin{aligned} 0 \le \mathbb{D}_{\mathrm{KL}}\left[q(\mathbf{Z}) \parallel p_\theta(\mathbf{Z}|\mathbf{X})\right] &= \mathbb{E}_{\mathbf{Z}\sim q}\left[\log \frac{q(\mathbf{Z})}{p_\theta(\mathbf{Z}|\mathbf{X})}\right] \\ &= \log p_\theta(\mathbf{X}) + \mathbb{E}_{\mathbf{Z}\sim q}\left[\log \frac{q(\mathbf{Z})}{p_\theta(\mathbf{X}, \mathbf{Z})}\right]. \end{aligned}$$

Thus, we have obtained a lower bound on the marginal log-likelihood which holds true for *any parameter value* θ and *any choice of the variational distribution* q:

$$\ell(\theta) = \log p_\theta(\mathbf{X}) \ge \mathbb{E}_{\mathbf{Z}\sim q}\left[\log \frac{p_\theta(\mathbf{X}, \mathbf{Z})}{q(\mathbf{Z})}\right] = \underbrace{\mathbb{E}_{\mathbf{Z}\sim q}\left[\log p_\theta(\mathbf{X}, \mathbf{Z})\right]}_{\text{energy}} \overbrace{-\mathbb{E}_{\mathbf{Z}\sim q}\left[\log q(\mathbf{Z})\right]}^{\text{entropy}}. \tag{9.15}$$

The right hand side in (9.15) is precisely the variational free energy; we see it decomposes in two terms. The first term is sometimes referred to as the *energy*. If we observed $\mathbf{Z}$, we would just maximise the complete data log-likelihood $\log p_\theta(\mathbf{X}, \mathbf{Z})$, but since $\mathbf{Z}$ is not observed we need to integrate it out, recalling that we are free to choose the distribution this done with respect to q. The second term is the Shannon entropy $H(q) = -\mathbb{E}_q \log q(\mathbf{Z})$ of the variational distribution $q(\mathbf{Z})$, and does not depend on θ (it can be thought of as the complexity penalty on q).

The terms "energy" and "entropy" come from statistical physics. The energy term here should in fact be the *negative* of the average energy of a system in thermal equilibrium, and "free energy" means energy that is accessible or useful (free as opposed to being tied up in entropy so is not accessible). Again the free energy term here should in fact be the *negative* free energy. Physical systems prefer low energy states, while here we are trying to maximise the variational free energy. We keep this inverted usage as it is common in machine learning.

The inequality becomes an equality when KL divergence is zero, i.e. when $q(\mathbf{Z}) = p_\theta(\mathbf{Z}|\mathbf{X})$, which means that the optimal choice of variational distribution q for fixed parameter value θ is *the true conditional of the latent variables given the observations and that* θ.

Thus, we have proved the following lemma:

Lemma 18. *Let $\mathcal{L}$ be the variational free energy in a latent variable model $p_\theta(\mathbf{X}, \mathbf{Z})$. Then (a) $\mathcal{L}(\theta, q) \leq \ell(\theta)$ for all q and for all θ, and (b) for any θ, $\mathcal{L}(\theta, q) = \ell(\theta)$ iff $q(\mathbf{Z}) = p_\theta(\mathbf{Z}|\mathbf{X})$.*

Thus, properties (9.12) and (9.13) are satisfied and we can recast the alternating maximisation of the variational free energy into iterative updates of q (E-step, via the plug-in full conditional of $\mathbf{Z}$ using the current estimate of θ) and the updates of θ (M-step, by maximising the 'energy' for the current estimate of q). Namely, we have the following

EM algorithm. Initialize $\theta^{(0)}$. At time $t \geq 1$:

- E-step: Set $q^{(t)}(\mathbf{Z}) = p_{\theta^{(t-1)}}(\mathbf{Z}|\mathbf{X})$
- M-step: Set $\theta^{(t)} = \arg\max_\theta \mathbb{E}_{\mathbf{Z}\sim q^{(t)}}\left[\log p_\theta(\mathbf{X}, \mathbf{Z})\right]$.

If both the E-step and M-step can be solved exactly, the EM algorithm converges to a local maximum likelihood solution. There are a number of classical models where this will indeed be the case, but as we move to more complicated approaches later in the chapter, we will consider models where neither step can be done analytically, necessitating the use of more general approaches.

9.4 EM Algorithm for Mixtures

Consider again our mixture model from Section 9.1.1 with

$$p_\theta(\mathbf{X}, \mathbf{Z}) = \prod_{i=1}^{n} \pi_{z_i} f_k(x_i; \lambda_{z_i}).$$

Recall that our latent variables $\mathbf{Z}$ are discrete (they correspond to cluster assignments) so q is a probability mass function over $\mathbf{Z} := (z_i)_{i=1}^n$. Using the expression (9.4), we can write the variational free energy as

$$\begin{aligned}
\mathcal{L}(\theta, q) =& \mathbb{E}_q[\log p_\theta(\mathbf{X}, \mathbf{Z}) - \log q(\mathbf{Z})] \\
=& \mathbb{E}_q\left[\left(\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{I}(z_i = k)\left(\log \pi_k + \log f_k(x_i; \lambda_k)\right)\right) - \log q(\mathbf{Z})\right] \\
=& \sum_{\mathbf{Z}} q(\mathbf{Z})\left[\left(\sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{I}(z_i = k)\left(\log \pi_k + \log f_k(x_i; \lambda_k)\right)\right) - \log q(\mathbf{Z})\right] \\
=& \sum_{i=1}^{n}\sum_{k=1}^{K} q(z_i = k)\left(\log \pi_k + \log f_k(x_i; \lambda_k)\right) + H(q).
\end{aligned}$$

We will denote $Q_{ik} = q(z_i = k)$, which is called the **responsibility** of cluster k for data item i.

Now, the E-step simplifies because

$$p_\theta(\mathbf{Z}|\mathbf{X}) = \frac{p_\theta(\mathbf{X},\mathbf{Z})}{p_\theta(\mathbf{X})} = \frac{\prod_{i=1}^n \pi_{z_i} f_k(x_i;\lambda_{z_i})}{\sum_{\mathbf{Z}'}\prod_{i=1}^n \pi_{Z'_i} f_k(x_i;\lambda_{Z'_i})} = \prod_{i=1}^n \frac{\pi_{z_i} f_k(x_i;\lambda_{z_i})}{\sum_k \pi_k f_k(x_i;\lambda_k)}$$
$$= \prod_{i=1}^n p_\theta(z_i|x_i).$$

Thus, for a fixed $\theta^{(t-1)} = (\lambda_1^{(t-1)},\dots,\lambda_K^{(t-1)},\pi_1^{(t-1)},\dots,\pi_K^{(t-1)})$ we can set

$$Q_{ik}^{(t)} = p_{\theta^{(t-1)}}(z_i = k|x_i) = \frac{\pi_k^{(t-1)} f_k(x_i;\lambda_k^{(t-1)})}{\sum_{j=1}^K \pi_j^{(t-1)} f_k(x_i;\lambda_j^{(t-1)})}. \tag{9.16}$$

Now, consider the M-step. For mixing proportions we have a constraint that $\sum_{j=1}^K \pi_j = 1$, so we introduce the Lagrange multiplier and obtain

$$\nabla_{\pi_k}\Big(\mathcal{L}(\theta,q) - \lambda(\textstyle\sum_{j=1}^K \pi_j - 1)\Big) = \displaystyle\sum_{i=1}^n \frac{Q_{ik}}{\pi_k} - \lambda = 0 \quad\Rightarrow\quad \pi_k \propto \sum_{i=1}^n Q_{ik}.$$

Since

$$\sum_{k=1}^K\sum_{i=1}^n Q_{ik} = \sum_{i=1}^n \underbrace{\sum_{k=1}^K Q_{ik}}_{=1} = n,$$

the M-step update for mixing proportions is

$$\pi_k^{(t)} = \frac{\sum_{i=1}^n Q_{ik}^{(t)}}{n}, \tag{9.17}$$

i.e., they are simply given by the total responsibility of each cluster. Note that this update holds regardless of the form of $f_k(\cdot;\lambda_k)$ used for mixture components.

Setting the derivative with respect to λ_k to 0, we obtain

$$\nabla_{\lambda_k}\mathcal{L}(\theta,q) = \sum_{i=1}^n Q_{ik}\nabla_{\lambda_k}\log f_k(x_i;\lambda_k) = 0. \tag{9.18}$$

This equation can be solved quite easily for mixture of normals in (9.3), giving the M-step update

$$\mu_k^{(t)} = \frac{\sum_{i=1}^n Q_{ik}^{(t)} x_i}{\sum_{i=1}^n Q_{ik}^{(t)}}, \tag{9.19}$$

which implies that the k-th cluster mean estimate is simply a weighted average of all the data items, where the weights correspond to the responsibilities of cluster k for these points.

Put together, the EM for normal mixture model with known (fixed) covariance is very similar to the K-means algorithm where cluster assignments are soft, i.e. rather than assigning each data item x_i to a single cluster at each iteration, we carry forward a responsibility vector $(Q_{i1},\dots,Q_{iK})$ giving probabilities of x_i belonging to each cluster.

EM for Normal Mixtures (known covariance) – “Soft K-means"

1. Initialize K cluster means $\mu_1, \dots, \mu_K$ and mixing proportions $\pi_1, \dots, \pi_K$.
2. *Update responsibilites (E-step):* For each $i = 1, \dots, n$, $k = 1, \dots, K$:
$$Q_{ik} = \frac{\pi_k \exp\left(-\frac{1}{2\sigma^2}\|x_i - \mu_k\|_2^2\right)}{\sum_{j=1}^K \pi_j \exp\left(-\frac{1}{2\sigma^2}\|x_i - \mu_j\|_2^2\right)} \tag{9.20}$$
3. *Update parameters (M-step):* Set $\mu_1, \dots, \mu_K$ and $\pi_1, \dots, \pi_K$ and based on the new cluster responsibilities:
$$\pi_k = \frac{\sum_{i=1}^n Q_{ik}}{n}, \qquad \mu_k = \frac{\sum_{i=1}^n Q_{ik} x_i}{\sum_{i=1}^n Q_{ik}}. \tag{9.21}$$
4. Repeat steps 2-3 until convergence.
5. Return the responsibilites $\{Q_{ik}\}$ and parameters $\mu_1, \dots, \mu_K, \pi_1, \dots, \pi_K$.

In some cases, depending on the form of $f_k(\cdot; \lambda_k)$, the M-step for π_k and E-step can both be done exactly, but the M-step update for model parameters of the components cannot. In these cases, we can use *gradient ascent* algorithm *inside the M-step*:
$$\lambda_k^{(r+1)} = \lambda_k^{(r)} + \alpha \sum_{i=1}^n Q_{ik} \nabla_{\lambda_k} \log f_k(x_i; \lambda_k^{(r)}).$$
This leads to a **generalized EM algorithm** for mixture models.

Mixture models can be applied to supervised learning settings as well. In a **mixture of experts**, we assume that a dataset of input/output pairs $\{(x_i, y_i)\}_{i=1}^n$ can be modelled using K experts. Expert k predicts y given x using some supervised model (e.g. a neural network) which outputs a distribution $f_{\theta_k}(y|x)$ with parameters θ_k. Given x, an expert is stochastically chosen using a gating predictor $g_\eta(x)$ which produces a vector of probabilities over experts. The marginal probability of y given x is then:
$$p(y|x) = \sum_{k=1}^K \{g_\eta(x)\}_k f_{\theta_k}(y|x).$$
The parameters of the mixture of experts can be learnt to maximise the log marginal likelihood given the dataset using the generalised EM algorithm.

9.5 Variational Inference

One of the workhorses of Bayesian machine learning are *variational approximations*, which turn posterior inference in intractable Bayesian models into optimization. We have seen that Bayesian model selection proceeds by optimizing (maximizing) the model evidence. While model evidence is almost always intractable, using the same principles as in the EM algorithm (Gibbs inequality), lower bounds may be available which can be optimized instead. The variational approach to Bayesian machine learning is often referred to as **variational Bayes**.

9.5.1 The ELBO

To given contrast to the lectures, assume that we wish to take a Bayesian view on both the latents $\mathbf{Z}$ and the model parameters θ, such that our joint is given by $p(\mathbf{X}, \mathbf{Z}, \theta)$. Now, because we are using a Bayesian model, our treatment of latent variables and model parameters is exactly the same. We can now consider some joint distribution $q(\mathbf{Z}, \theta)$ of latent variables and parameters, called a **variational distribution** (similarly to EM, but note that EM was not allowed to place a distribution over θ). We claim that the quantity

$$\mathcal{L}(q) = \mathbb{E}_q \left[\log p(\mathbf{X}, \mathbf{Z}, \theta)\right] + H(q) \tag{9.22}$$

is a lower bound on log-evidence $\log p(\mathbf{X})$. Namely, we can write

$$\begin{aligned}
\mathcal{L}(q) &= \mathbb{E}_q \left[\log p(\mathbf{X}, \mathbf{Z}, \theta)\right] - \mathbb{E}_q[\log q(\mathbf{Z}, \theta)] \\
&= \log p(\mathbf{X}) + \mathbb{E}_q \left[\log p(\mathbf{Z}, \theta|\mathbf{X})\right] - \mathbb{E}_q[\log q(\mathbf{Z}, \theta)] \\
&= \log p(\mathbf{X}) - \mathrm{KL}\left(q(\mathbf{Z}, \theta)||p(\mathbf{Z}, \theta|\mathbf{X})\right),
\end{aligned}$$

which is by Gibbs inequality maximised (and equal to log-evidence) when the KL is zero, i.e. when $q(\mathbf{Z}, \theta) = p(\mathbf{Z}, \theta|\mathbf{X})$. Thus, for any variational distribution q, $\mathcal{L}(q) \leq \log p(x)$. Expression 9.22 is called the **evidence lower bound** (ELBO).

To reason about all the unknowns in the model, we would simply need to compute the joint posterior $p(\mathbf{Z}, \theta|\mathbf{X})$, but this is almost always intractable. Hence, variational Bayesian inference *approximates* the posterior by starting with a family $\mathcal{Q}$ of tractable variational distributions $q(\mathbf{Z}, \theta)$ (e.g. $q_\phi(\mathbf{Z}, \theta)$ where ϕ are the **variational parameters**), and aims to minimize the divergence $\mathrm{KL}\left(q(\mathbf{Z}, \theta)||p(\mathbf{Z}, \theta|\mathbf{X})\right)$ over $\mathcal{Q}$ or, equivalently, maximise the ELBO, i.e. find the tightest lower bound on the log-evidence.

In a nutshell, variational Bayes projects the (intractable) posterior $p(\mathbf{Z}, \theta|\mathbf{X})$ onto a tractable family $\mathcal{Q}$ with respect to the KL divergence $\mathrm{KL}\left(q(\mathbf{Z}, \theta)||p(\mathbf{Z}, \theta|\mathbf{X})\right)$. Alternative divergences are possible in this context. In particular, since KL is not symmetric, minimization of the "reverse" divergence $\mathrm{KL}\left(p(\mathbf{Z}, \theta|\mathbf{X})||q(\mathbf{Z}, \theta)\right)$ results in a different family of approximate Bayesian methods, known as **expectation propagation** (EP).

9.5.2 Variational EM and Mean-Field Variational Family

Finding a variational approximation requires specifying the variational family $\mathcal{Q}$. The complexity of $\mathcal{Q}$ determines the difficulty of the optimization; it is more difficult to optimize over a large family $\mathcal{Q}$ than over a simpler, smaller one. Consider a family $\mathcal{Q}$ of variational distributions which factorize across the latents and the parameters:

$q(\mathbf{Z},\theta) = q_{\mathbf{Z}}(\mathbf{Z})\, q_\theta(\theta)$. For a fixed q_θ, we can solve for $q_{\mathbf{Z}}$ which maximises ELBO (*exercise):*

$$q_{\mathbf{Z}}(\mathbf{Z}) \propto \exp\left(\int \log p(\mathbf{X},\mathbf{Z},\theta) q_\theta(\theta)\, d\theta\right),$$

and by symmetry, for a fixed $q_{\mathbf{Z}}$, we can solve for q_θ which maximises ELBO:

$$q_\theta(\theta) \propto \exp\left(\int \log p(\mathbf{X},\mathbf{Z},\theta) q_{\mathbf{Z}}(\mathbf{Z})\, d\mathbf{Z}\right).$$

Now, one can formulate an algorithm similar to EM, which alternates between optimising $q_{\mathbf{Z}}$ and q_θ, such that each iteration increases ELBO and thus decreases the KL divergence from the posterior. Hence such an algorithm is sometimes called a **variational Bayesian EM** algorithm, VBEM, or sometimes simply **variational EM**.

We notice the symmetry between $\mathbf{Z}$ and θ. Indeed, the distinction between parameters and latent variables disappears in Bayesian modelling, as all unobserved quantities in the model are treated in the same way and our goal is to approximate their posterior distribution. In the rest of this section, we will drop θ from the notation and treat them as a part of the set of all unobserved quantities $\mathbf{Z}$.

A more general simplification we often make in variational Bayes is to focus on a **mean-field approximation** where the variational distribution fully factorizes

$$q(\mathbf{Z}) = \prod_{j=1}^{m} q_j(z_j),$$

i.e. all latent variables are mutually independent and each latent z_j is governed by its own variational factor q_j. Note that there could be a mix between categorical and continuous latents, each having the appropriate factor q_j. Also, z_j itself need not be a univariate latent—see an example with LDA below. Using the mean-field family implies that we will not be able to capture any posterior correlations between the latent variables z_j and $z_{j'}$ for $j \neq j'$ and that the best we can hope for is a rich representations of the posterior marginals.

An iterative procedure similar to Bayesian EM can now be applied to each individual factor, giving rise to an algorithm called **coordinate ascent variational inference** (CAVI) shown in Algorithm 9.1. CAVI is closely related to Gibbs sampling (a popular class of MCMC algorithms), and is based on iteratively approximating the full conditionals $p(z_j|\mathbf{Z}_{-j},\mathbf{X}) \propto p(\mathbf{X},\mathbf{Z})$, where $\mathbf{Z}_{-j} = [z_1,\dots,z_{j-1},z_{j+1},\dots,z_n]$.

9.5.3 Complete conditionals in the exponential family

When the complete conditionals $p(z_j|\mathbf{Z}_{-j},\mathbf{X})$ belong to an exponential family of distributions, i.e. they are given by

$$p(z_j|\mathbf{Z}_{-j},\mathbf{X}) = h(z_j)\exp\left[z_j^\top \eta_j(\mathbf{Z}_{-j},\mathbf{X}) - A(\eta_j(\mathbf{Z}_{-j},\mathbf{X}))\right],$$

a particularly convenient form of CAVI is available as the updates are available in closed form. Here we assume z_j is already transformed to its appropriate sufficient statistic, $h(\cdot)$

Algorithm 9.1 Coordinate Ascent Variational Inference (CAVI)

```
Input: model $p(\mathbf{X}, \mathbf{Z})$, dataset $\mathbf{X}$
Output: a variational posterior $q(\mathbf{Z})$

while the ELBO has not converged do
  for $j = 1, \ldots, m$
    $q_j(z_j) \propto \exp\left(\mathbb{E}_{\mathbf{Z}_{-j} \sim q}\left[\log p\left(z_j | \mathbf{Z}_{-j}, \mathbf{X}\right)\right]\right)$
return $q\left(\mathbf{Z}\right) = \prod_{j=1}^{m} q_j\left(z_j\right)$
```

is a base measure, $A\left(\cdot\right)$ is the log-normalizer and $\eta_j\left(\mathbf{Z}_{-j}, \mathbf{X}\right)$ are the natural parameters (which depend on the conditioning set). The CAVI update is now

$$\begin{aligned} q_j(z_j) &\propto \exp\left(\mathbb{E}_{\mathbf{Z}_{-j}}\left[\log p\left(z_j | \mathbf{Z}_{-j}, \mathbf{X}\right)\right]\right) \\ &= \exp\left(\log h\left(z_j\right) + z_j^\top \mathbb{E}_{\mathbf{Z}_{-j}}\left[\eta_j\left(\mathbf{Z}_{-j}, \mathbf{X}\right)\right] - \mathbb{E}_{\mathbf{Z}_{-j}}\left[A\left(\eta_j\left(\mathbf{Z}_{-j}, \mathbf{X}\right)\right)\right]\right) \\ &\propto h\left(z_j\right) \exp\left(z_j^\top \mathbb{E}_{\mathbf{Z}_{-j}}\left[\eta_j\left(\mathbf{Z}_{-j}, \mathbf{X}\right)\right]\right) \end{aligned}$$

and thus, the variational factors are in the same exponential family as the complete conditionals with natural parameter being the expected natural parameter of the complete conditional

$$\phi_j = \mathbb{E}_{\mathbf{Z}_{-j}}\left[\eta_j\left(\mathbf{Z}_{-j}, \mathbf{X}\right)\right].$$

This setup describes many models, including Bayesian Gaussian mixtures, Dirichlet process mixtures, matrix factorization, multilevel regression, and latent Dirichlet allocation, giving thus one classical overarching CAVI algorithm with closed-form updates for many classical instances of Variational Bayes. However, such occurrences are less common in modern large–scale variational systems that tend to be based on the (stochastic) gradient based approach discussed in the lectures.

While the distinction between parameters and latent variables disappears in Bayesian modelling, there is still a relevant distinction in terms of where in the model hierarchy these unobserved quantities appear. The impact of such hierarchy is that not all updates in Algorithm 9.1 need to be performed sequentially. Multiple levels of hierarchy are also possible. In particular, for large datasets, we often perform **stochastic variational inference**, wherein updates for variational approximations of global parameters θ are made more regularly than for local latents. This is achieved by using stochastic gradient updates, analogous to those discussed in the lectures where we were looking to optimize θ rather than perform inference over it.

9.5.4 Example: Topic Modelling [Not Examinable]

Topic models[1] are a class of probabilistic models of text that lead to parsimonious representations of hidden thematic structure of a collection of documents. A popular approach

[1]While the LDA model covered here is not directly examinable, many of the discussed ideas will be helpful for more completely understanding variational inference.

to topic modelling is Latent Dirichlet Allocation (LDA) [7].[2]

Latent Dirichlet Allocation

Latent Dirichlet allocation (LDA) captures the intuition that a text document typically exhibits multiple topics and blends them in a particular way. In LDA, each topic is modelled as a probability distribution over words and each document as a mixture of corpus-wide topics (i.e. it can be identified with a distribution over topics). Each observed word in a document is then treated as a draw from the mixture, i.e. it belongs to one of the topics (mixture components). Mixture proportions are thus unique for each document, i.e. they are local latents, but mixture components are shared across the whole collection—they are global parameters.

This setting is also called a **mixed membership model**. Another important example of mixed membership model is the STRUCTURE model in population genetics. There, the DNA of each individual in a population is modelled as a mixture over ancestral populations, with each individual having different proportions of their DNA coming from each ancestral population. This is most clear for individuals of mixed ethnic ancestry, but such approaches are also applicable to, e.g. "native" British, where the the inferred population structure has been used to understand the history and migration patterns of the peoples of the British isles before modern times [37].

Let K be the number of topics and V the size of the vocabulary. LDA posits the following conditionally conjugate model.

1. For each topic in $k = 1, \ldots, K$,
   a) Draw a distribution over V words $\beta_k \sim \mathrm{Dir}_V(\eta)$
2. For each document in $d = 1, \ldots, D$,
   a) Draw a vector of topic proportions[3] $\theta_d \sim \mathrm{Dir}_K(\alpha)$
   b) For each word in $n = 1, \ldots, N_d$,
      i. Draw a topic assignment $z_{dn} \sim \mathrm{Mult}(\theta_d)$, i.e. $p(z_{dn} = k|\theta_d) = \theta_{dk}$
      ii. Draw a word $w_{dn} \sim \mathrm{Mult}(\beta_{z_{dn}})$, i.e. $p(w_{dn} = v|\beta, z) = \beta_{z_{dn}v}$

Given we observe the words in the documents, the goal of LDA is to find the posterior

$$p\left(\beta_{1:K}, \theta_{1:D}, \{z_{d,1:N_d}\}_{d=1}^{D} \,|\, \{w_{d,1:N_d}\}_{d=1}^{D}\right) = p(\text{topics, proportions, assignments} \,|\, \text{words})$$

Note that a corpus of text to be analyzed may consist of millions of documents, thus

[2]Be careful not to confuse Latent Dirichlet Allocation with Linear Discriminant Analysis which shares the acronym—these two models are not related.

[3]Note our usage of "θ" here is as a specific set of parameters rather than all the global parameters of the model. This notation is used for consistency with the standard names used for LDA in the literature.

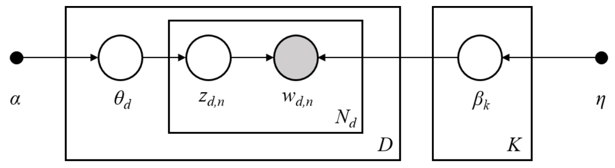


Figure 9.1: Graphical model representation of LDA. Plates represent replication, for example there are D documents each having a topic proportion vector θ_d

having possibly billions of latent variables. We can write the joint distribution as

$$\begin{aligned} p(\beta,\theta,z,w) &= \prod_{k=1}^{K} p(\beta_k;\eta) \prod_{d=1}^{D} \left\{ p(\theta_d;\alpha) \prod_{n=1}^{N_d} p(z_{dn}|\theta_d)\, p(w_{dn}|\beta,z) \right\} \\ &= \frac{1}{B(\eta)^K B(\alpha)^D} \prod_{k=1}^{K}\prod_{v=1}^{V} \beta_{kv}^{\eta_v-1} \prod_{d=1}^{D} \left\{ \prod_{k=1}^{K} \theta_{dk}^{\alpha_k-1} \prod_{n=1}^{N_d} \theta_{d,z_{dn}} \beta_{z_{dn},w_{dn}} \right\}. \qquad (9.23) \end{aligned}$$

The model has the following latents: β (topics), θ (proportions), and z (assignments). Note that there are also hyperparameter vectors $\eta \in \mathbb{R}_+^V$ and $\alpha \in \mathbb{R}_+^K$ in the Dirichlet priors—these are assumed fixed. While β and θ are “global” variables, we will still be performing variational inference over them as well.

Data are the observed words $\{w_{dn}\}$. There will be a slight abuse of notation here: we denote by w_{dn} both the appropriate draw from vocabulary $\{1,\ldots,V\}$—to be used for indexing—and its “one-hot” encoding, i.e. a binary V-vector with $w_{dn}[v]=1$ if $w_{dn}=v$ and zero otherwise. We will write $w_{dn}[\cdot]$ in the case of the latter. Similarly, we denote by z_{dn} both the appropriate topic assignment from $\{1,\ldots,K\}$ and its “one-hot” encoding i.e. a binary K-vector with $z_{dn}[k]=1$ if $z_{dn}=k$ and zero otherwise. We will write $z_{dn}[\cdot]$ in the case of the latter.

We will use a mean-field family of the form

$$q(\beta,\theta,z) \;=\; \prod_{k=1}^{K} q(\beta_k;\zeta_k) \prod_{d=1}^{D} \left\{ q(\theta_d;\gamma_d) \prod_{n=1}^{N_d} q(z_{dn};\phi_{dn}) \right\}.$$

The complete conditionals are proportional to the joint distribution in (9.23):

1. Complete conditional on the topic assignment is a multinomial with

$$p(z_{dn}=k|\theta_d,\beta,w_d) \propto \theta_{dk}\beta_{k,w_{dn}} = \exp\left(\log\theta_{dk} + \log\beta_{k,w_{dn}}\right). \qquad (9.24)$$

   Thus, for the variational approximation we also use a multinomial but with a “free parameter” ϕ_{dn}, where we denote $\phi_{dn}[k] = q(z_{dn}=k)$, i.e. ϕ_{dn} is simply a probability mass function over K topics.

2. Complete conditional on the topic proportions depends only on the assignments and is given by

$$p(\theta_d|z_d) = \mathrm{Dir}_K\left(\theta_d;\alpha + \sum_{n=1}^{N_d} z_{dn}[\cdot]\right). \qquad (9.25)$$

For the variational approximation we also use Dirichlet, with parameter vector $\gamma_d \in \mathbb{R}_+^K$.

3. Complete conditional on the topics is

$$p(\beta_k|z,w) = \text{Dir}_V\left(\beta_k; \eta + \sum_{d=1}^{D}\sum_{n=1}^{N_d} z_{dn}[k]\, w_{dn}[\cdot]\right). \tag{9.26}$$

For the variational approximation we also use Dirichlet, with parameter vector $\zeta_k \in \mathbb{R}_+^V$.

With these full conditionals we can derive the CAVI updates in the LDA model. We will need the following standard result about the Dirichlet distribution given here without proof.

Proposition 19. *If $\pi \sim Dir_L(\alpha)$, then*

$$\mathbb{E}[\log \pi_j] = \psi(\alpha_j) - \psi\left(\sum_{\ell=1}^{L} \alpha_\ell\right),$$

where $\psi(u) = \frac{\Gamma'(u)}{\Gamma(u)} = \int_0^\infty \left(\frac{e^{-t}}{t} - \frac{e^{-ut}}{1-e^{-t}}\right) dt$ is the digamma function.

Now we can obtain the closed-form updates for each set of the latents.

Proposition 20. *CAVI updates in the LDA model are given by*

1. $\phi_{dn}[k] \propto \exp\left(\psi(\gamma_{dk}) + \psi(\zeta_{k,w_{dn}}) - \psi\left(\sum_{v=1}^{V} \zeta_{k,v}\right)\right)$,
2. $\gamma_d = \alpha + \sum_{n=1}^{N_d} \phi_{dn}$,
3. $\zeta_k = \eta + \sum_{d=1}^{D}\sum_{n=1}^{N_d} \phi_{dn}[k]\, w_{dn}[\cdot]$,

where ψ is the digamma function.

Proof. Steps (2) and (3) directly follow from the exponential family properties of Dirichlet distribution. Namely for (2), we note that

$$\log p(\theta_d|z_d) = \text{const} + \sum_{k=1}^{K}\left(\alpha_k + \sum_{n=1}^{N_d} z_{dn}[k] - 1\right)\log \theta_{dk},$$

so that

$$\exp\left(\mathbb{E}_{z_d \sim q}[\log p(\theta_d|z_d)]\right) \propto \prod_{k=1}^{K} \theta_{dk}^{\alpha_k + \sum_{n=1}^{N_d} \phi_{dn}[k] - 1},$$

which is proportional to the Dirichlet distribution with parameter vector $\gamma_d = \alpha + \sum_{n=1}^{N_d} \phi_{dn}$. The same logic applies for (3).

For (1), we make use of Proposition 19 and write

$$\begin{aligned}
\phi_{dn}[k] &\propto \exp\left(\mathbb{E}_{\theta_d,\beta\sim q}\left[\log p\left(z_{dn}=k|\theta_d,\beta,w_d\right)\right]\right) \\
&\propto \exp\left(\mathbb{E}_{\theta_d\sim q}\left[\log\theta_{dk}\right]+\mathbb{E}_{\beta_k\sim q}\left[\log\beta_{k,w_{dn}}\right]\right) \\
&\propto \exp\left(\psi\left(\gamma_{dk}\right)-\psi\left(\sum_{\ell=1}^{K}\gamma_{d\ell}\right)+\psi\left(\zeta_{k,w_{dn}}\right)-\psi\left(\sum_{v=1}^{V}\zeta_{k,v}\right)\right) \\
&\propto \exp\left(\psi\left(\gamma_{dk}\right)+\psi\left(\zeta_{k,w_{dn}}\right)-\psi\left(\sum_{v=1}^{V}\zeta_{k,v}\right)\right),
\end{aligned}$$

as required. □

9.6 Variational Auto–Encoders

So far we have mostly implicitly been thinking in terms of manually constructed generative models where the global parameters θ are small to moderate dimensional, with the model making fairly strong assumptions about the generative process. However, in many scenarios one needs to deal with complex or high–dimensional data, where such manual construction is not feasible and we would rather specify a very general model class and then train the model from the data.

Deep generative models (**DGMs**) are a class of, typically factorized, LVMs that allow us to deal with such settings by using deep neural networks to parameterize the generative process in high flexible manner and then learning the parameters of this network in an end–to–end manner using stochastic gradient techniques.

DGMs provide a mechanism to train generative models in a data–driven manner that imposes far weaker assumptions than traditional approaches like graphical models. This allows them to learn extremely powerful models and operate in settings that would be far beyond the reach of traditional approaches, particularly those with high–dimensional data like images. This leads to a wide range of applications such as synthetic data generation, feature/representation learning, nonlinear dimensionality reduction (compared with, e.g., the linear dimensionality reduction of PCA), manipulation of individual datapoints, construction of robust machine learning systems, data cleaning (i.e. dealing with things like incomplete data, outliers, non–numeric data, and multiple data modalities), interpretable machine learning, metric/manifold learning, and simply data (pre–)processing in general.

There are a wide range of DGMs used in practice, far more than we can realistically cover; we will focus on one particular popular example: **variational auto–encoders** (VAEs) [29, 48].

A VAE is a factorized LVM[4] with two key components: a **generative network**, or **decoder**, $p_\theta(x|z)$, and an **inference network**, or **encoder**, $q_\phi(z|x)$. The parameters of these networks, θ and ϕ, are trained by simultaneously optimizing the ELBO with respect to both.

[4]There are also some extensions that do not make fully factorized assumptions, e.g. [35].

In the standard approach, which we will focus on, the prior, $p(z)$, is fixed to an isotropic Gaussian $p(z) = \mathcal{N}(z; 0, I)$, though there are various extensions that instead employ hierarchical latent spaces [54] or learn the prior itself during training [55]. Note that each of the prior, encoder, and decoder are defined on for a single arbitrary datapoint: because of the factorization assumption the generative model is the same for all datapoints, while there are no *stochastic* global parameters. Thus there are no probabilistic interactions between different datapoints x_i and x_j, or their corresponding latents z_i and z_j. We, therefore, define our generative model as $p(z)p_\theta(x|z)$ and use $\mathbf{X}$ as a set of example datapoints for training θ and ϕ.

While the role of the decoder is self–apparent through its inclusion in the generative model itself, the encoder can be viewed in a number of ways. The viewpoint most in line with the concepts we have introduce thus far is that it is a functional approximation of the posterior $p_\theta(z|x)$.[5] That is, rather than individually learning a variational approximation for each datapoint, we learn a *mapping* from datapoints to a variational approximation for that datapoint. In other words, we learn a conditional distribution on z given x.

This process is known as **amortised inference**. The word amortised comes, via the probabilistic programming literature, from the compilers literature, where amortisation refers to pre-computing functions at compile time so that function evaluations can be performed efficiently at run time. In the fixed model setting, amortised inference uses a function approximator which is learnt at 'training time,' so that inference for new data items at 'test time' (i.e. when dealing with a particular datapoint or dataset depending on context) can be performed efficiently using the function approximator. This is as opposed to iterative inference procedures, e.g. Markov chain Monte Carlo or coordinate ascent variational inference, which can be much more expensive at test time. In the context of VAEs, this amortised approximation is learned simultaneously to the generative model, and allows us to share information across different datapoints, rather than requiring separate inference schemes/variational parameters for each.

Both the decoder and the encoder are former by using a deep neural network to map from their respective inputs to distribution parameters in their corresponding output spaces. For example, a common choice for both is a Gaussian with diagonal covariance for which we have

$$p_\theta(x|z) = \mathcal{N}(x; \mu_\theta(z), \sigma^2_\theta(z)) \tag{9.27}$$

$$q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \sigma^2_\phi(x)) \tag{9.28}$$

[5]It is beyond the scope of this course to properly explore the plethora of subtleties surrounding VAE encoders, and indeed VAEs more generally, with these still being a highly active area of research. However, it is important to be aware of the fact that this viewpoint of the encoder as simply an approximation of the decoder posterior is actually somewhat limited: the way VAEs are trained means that the encoder influences the learning of the decoder just as much as the decoder influences the learning of the encoder. As such, the two are always intertwined and in many applications, such as representation learning, the encoder is actually the primary entity of interest, rather than something introduced simply to aid with the training of the generative model. Moreover, inductive biases in the encoder design, such as the standard choice of a diagonal covariance, can heavily influence the models learned.

where $\mu_\theta(z), \sigma^2_\theta(z)$ are outputs of the decoder deep neural network with parameters θ (where $\sigma^2_\theta(z)$ is a diagonal covariance matrix with zeros off the main diagonal) and analogously $\mu_\phi(x), \sigma^2_\phi(x)$ are the outputs of the encoder network with parameters ϕ. The distributional form of the decoder naturally changes with the type of data used, with, for example, Bernoulli distributions also common. However, it is generally important that the *form* of this distribution remains simple (but with the parameter mappings themselves complex) to ensure that the information about the generations is stored in the latents, rather than the randomness in the generation of x given $\mathbb{E}_{X\sim p_\theta(x|z=z_i)}[X]$. In short, the variability in x for a given z should effectively correspond to noise regardless of the form of the decoder (see e.g. `http://ruishu.io/2017/01/14/one-bit/`).

More complicated forms for the encoder distribution can also be helpful. For example, some approaches augment VAE posterior approximations by transforming drawn samples through mappings (so called *flows*) with additional trainable parameters to achieve richer variational families [42].

9.6.1 Revisiting the ELBO

Training in VAEs is done by simultaneously optimizing the ELBO with respect to both θ and ϕ using a stochastic gradient approach. This is based on exploiting the factorization assumption over datapoints to employ mini–batching. Namely, we have

$$\mathcal{L}(\mathbf{X}, \theta, \phi) := \sum_{i=1}^n \mathcal{L}(x_i, \theta, \phi) = \sum_{i=1}^n \mathbb{E}_{q_\phi(z_i|x_i)}\left[\log \frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)}\right] \leq \log p_\theta(\mathbf{X}) \tag{9.29}$$

where each $\mathcal{L}(x_i, \theta, \phi) \leq \log p_\theta(x_i)$ is the ELBO for a single datapoint and $p_\theta(x) = \mathbb{E}_{p(z)}[p_\theta(x|z)]$. Here we can approximate the sum using a mini-batch B in standard way, that is

$$\nabla_{\theta,\phi}\mathcal{L}(\mathbf{X}, \theta, \phi) \approx \frac{n}{|B|}\sum_{i\in B} \nabla_{\theta,\phi}\mathbb{E}_{q_\phi(z_i|x_i)}\left[\log \frac{p_\theta(x_i, z_i)}{q_\phi(z_i|x_i)}\right].$$

Dealing with the expectation can be dealt with using (typically single sample) Monte Carlo estimates, but requires some care for the gradients of ϕ, as discussed later, due to its presence in the distribution the expectation is taken with respect to. Note that, as opposed to the EM algorithm, which has two separate steps for inference and learning, we will calculate unbiased derivative estimates with respect to θ and ϕ at the same time with the same mini-batch / latent samples, and then update both in the same step.

Consider now different ways in which we can write the ELBO for a single x:

$$\begin{aligned}
\mathcal{L}(x, \theta, \phi) &= \log p_\theta(x) - \mathbb{D}_{\mathrm{KL}}(q_\phi(z|x) \| p_\theta(z|x)) \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x, z)}{q_\phi(z|x)}\right] && (9.30) \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p(z)}{q_\phi(z|x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{D}_{\mathrm{KL}}\left(q_\phi(z|x) \| p(z)\right). && (9.31)
\end{aligned}$$

The final expression is particularly convenient as for a standard choices of a diagonal Gaussian encoder (as per (9.28)) and a standard normal prior $p(z)$, the KL divergence is available in closed form:

$$\begin{aligned}\mathbb{D}_{\mathrm{KL}}\left(q_\phi(z|x)||p(z)\right) &= \frac{1}{2}\left[\mu_\phi(x)^\top \mu_\phi(x) + \operatorname{tr}\left(\Sigma_\phi(x)\right) - \log\det\left(\Sigma_\phi(x)\right) - k\right] \\ &= \frac{1}{2}\sum_{j=1}^{k}\left[\mu^2_{\phi,j}(x) + \sigma^2_{\phi,j}(x) - \log\left(\sigma^2_{\phi,j}(x)\right) - 1\right]. \end{aligned} \tag{9.32}$$

When this assumption does not hold, we can simply use the form of the ELBO in (9.30) instead, noting though that this tends to be slightly higher variance.

The formulation of (9.31) is also insightful from the perspective of the VAE itself. Namely, it shows how we can view a VAE as a stochastic **auto–encoder** as discussed in the lectures. In short, $\mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$ forms a **reconstruction loss** that measures how effectively the original input is preserved when passed through the encoder and then back through the decoder, while $\mathbb{D}_{\mathrm{KL}}\left(q_\phi(z|x)\|p(z)\right)$ acts a regulariser that increases the entropy of the encoding process to enforce smoothness in the space of z. This alternative viewpoint of VAEs is extremely important as it shows how VAEs can used for **representation learning**, wherein z can be viewed as a compressed feature representation of x. This can then be used in downstream prediction tasks, or as mechanism for interpreting or manipulating individual datapoints. See the lectures for more in depth discussion.

9.6.2 The Reparametrization Trick

In order to optimize the ELBO objective over θ and ϕ, it remains to unbiasedly estimate the $\nabla_{\theta,\phi}\mathcal{L}(x_i,\theta,\phi)$ terms.

For θ, this can be straightforwardly achieved by drawing Monte Carlo samples from $q_\phi\left(z|x_i\right)$. Typically only a single such sample is drawn in practice for each datapoint, on the basis that it is almost always better to increase the size of the mini–batch to reduce the variance, rather than drawing additional samples of z for each x_i. We can see this by noting that such an approach is essentially equivalent to drawing $|B|$ samples from the joint $p_{\text{data}}(x)q_\phi(z|x)$ (where $p_{\text{data}}(x)$ is the empirical data distribution).

For ϕ, such an approach would result in a biased estimator and thus cannot be used directly. This is because ϕ affects the distribution the expectation is taken with respect to itself. Thankfully, there is a simple solution to this known as the **reparametrization trick**, or sometimes as the **pathwise estimator**. This involves expressing the distribution $q_\phi(z|x)$ it the form of a base random draw $\epsilon \sim q(\epsilon)$ (typically just $\mathcal{N}(0,I)$) and a deterministic mapping $z = g(\epsilon, x, \phi)$. This then allows us to write $\mathcal{L}(x_i,\theta,\phi)$ as an expectation over ϵ, which is itself independent of ϕ, taking care to note that z itself now depends on ϕ. Namely we have

$$\nabla_\phi \mathbb{E}_{q_\phi(z_i|x_i)}\left[\log\left(\frac{p_\theta(x_i,z_i)}{q_\phi(z_i|x_i)}\right)\right] = \mathbb{E}_{q(\epsilon)}\left[\nabla_\phi \log\left(\frac{p_\theta(x_i, g(\epsilon_i,x_i,\phi))}{q_\phi(g(\epsilon_i,x_i,\phi)|x_i)}\right)\right] \tag{9.33}$$

where we can sample $\epsilon_i \sim \mathcal{N}(0, I)$ and then calculate the required derivatives using automatic differentiation. Critically, the resulting stochastic gradients will be unbiased estimates of the true gradients.

To give a concrete example, in the case of a diagonal Gaussian encoder, a draw $z_i \sim \mathcal{N}\left(z; \mu_\phi(x), \sigma^2_\phi(x)\right)$ can be written as $z_i = \mu_\phi(x) + \sigma_\phi(x)\epsilon_i$, with $\epsilon_i \sim \mathcal{N}(0, I)$, and the multiplication with $\sigma_\phi(x)$ being element–wise.

9.6.3 Score Function Estimator

Unfortunately, the reparameterization trick can only be applied if $q_\phi(z|x)$ is continuous with respect to z as otherwise applying the chain rule to (9.33) requires us to back-propagate our derivatives through z. This essentially means that we cannot use the reparametrization trick whenever the z are discrete.

On the plus side, there is an alternative approach to obtain unbiased estimates of gradients with respect to variational parameters, termed the **score function gradient** or **REINFORCE gradient**, that can still operate in such settings. On the downside, this estimator tends to have much higher variance than the reparametrized gradient and should thus be avoided if possible. Variance reduction techniques, like control variates, can be helpful in cases where the approach is unavoidable.

To derive this estimator, first note the following result, known as the **score identity**,

$$\mathbb{E}_{q_\phi(z)}\left[\nabla_\phi \log q_\phi(z)\right] = \int q_\phi(z)\, \nabla_\phi \log q_\phi(z)\, dz = \int \nabla_\phi q_\phi(z)\, dz = \nabla_\phi \int q_\phi(z)\, dz = 0,$$

which critically still also holds when dz is a counting measure. The score function gradient can now be derived as follows

$$\begin{aligned}\nabla_\phi \mathcal{L} &= \nabla_\phi \int q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} dz \\ &= \int \left(\nabla_\phi q_\phi(z|x)\right) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} dz + \underbrace{\int q_\phi(z|x) \left(\nabla_\phi \log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right) dz}_{=0}\end{aligned}$$

where the second term being zero follows by the score identity and lack of dependence of $p_\theta(x, z)$ on ϕ, and so also noting $\nabla x = x \nabla \log x$, we have

$$= \mathbb{E}_{q_\phi(z|x)}\left[\left(\nabla_\phi \log q_\phi(z|x)\right) \log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right].$$

This is now something where we can directly construct Monte Carlo estimates, calculating the required derivatives through automatic differentiation (note that here z is not a function of ϕ as it was for the reparametrized estimator).

9.6.4 Alternative Bounds

Assume that we have access to some strictly *positive* and *unbiased* estimator $\hat{p}_\theta(x)$ of $p_\theta(x)$, that is $\mathbb{E}[\hat{p}_\theta(x)] = p_\theta(x)$ and $\hat{p}_\theta(x) > 0$. Jensen's inequality then tells us that

$$\mathbb{E}[\log \hat{p}_\theta(x)] \le \log \mathbb{E}[\hat{p}_\theta(x)] = \log p_\theta(x)$$

such that the expectation of the logarithm of our estimator is a lower bound on the evidence. The VAE framework we have described now simply correspond to the special case of $\hat{p}_\theta(x) = p_\theta\left(x, z\right)/q_\phi(z|x)$, which is the importance weight if $q_\phi(z|x)$ is the proposal distribution and the target is the posterior $p_\theta(z|x)$.

One can thus naturally consider other types of estimators. Essentially, any inference method that produces an unbiased marginal likelihood estimator will produce a valid lower bound. Examples of suitable such estimators are importance sampling, annealed importance sampling, and sequential Monte Carlo. For example, the former of these is constructing by independently sampling from the encoder K time, $z_k \overset{iid}{\sim} q_\phi(\cdot|x)$, and then constructing the bound

$$\mathcal{L}_K(x, \theta, \phi) = \mathbb{E}_{\prod_{k=1}^K q_\phi(z_k|x)}\left[\log\left(\frac{1}{K}\sum_{k=1}^{K}\frac{p_\theta\left(x, z_k\right)}{q_\phi\left(z_k|x\right)}\right)\right] \tag{9.34}$$

In general, the use of alternative marginal likelihood estimators like this allow one to achieve *tighter* variational bounds than the standard ELBO, which thus more accurately represent the true model evidence.

Another important variation in the standard ELBO is given by the so-called β–VAE [1, 25]. This tries to directly control the level of regularization in the ELBO by introducing a scaling factor β as follows

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

Higher values of β correspond to larger degrees of regularization and tend to produce smoother latent spaces (i.e. models were small changes in z lead to small changes in $p_\theta(x|z)$), at the expense of a drop in reconstruction quality and typical generation fidelity. It is still a valid lower bound on the evidence for $\beta \geq 1$, though $0 < \beta < 1$ is also sometimes useful in practice if the amount of regularization in the original ELBO is deemed too large. However, the bound no longer becomes tight if $q_\phi(z|x) = p_\theta(z|x)$ for any β other than $\beta = 1$. Numerous interesting results have been shown for the β–VAE, such as the fact that the value of β is intricately linked to the smoothness of the latent space which is learned [38] and also the amount of information lost in the encoding–decoding process, with such an *information bottleneck* allowing one the encoding of only important information [1].

Bibliography

[1] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

[2] David Arthur and Sergei Vassilvitskii. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, 2007.

[3] Renée Baillargeon. The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development*, 1(46-83):1, 2002.

[4] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[5] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics.* Kluwer, 2004.

[6] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[7] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, April 2012.

[8] George EP Box. Robustness in the strategy of scientific model building. *Robustness in statistics*, 1:201–236, 1979.

[9] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[10] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[11] Eric Brochu, Vlad M Cora, and Nando de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv.org, December 2010.

[12] Robert Burbidge, Matthew Trotter, B Buxton, and S Holden. Drug design by machine learning: support vector machines for pharmaceutical data analysis. *Computers & chemistry*, 26(1):5–14, 2001.

[13] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical Science*, pages 273–304, 1995.

[14] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993.

[15] Bruno De Finetti. La prévision: ses lois logiques, ses sources subjectives. In *Annales de l'institut Henri Poincaré*, volume 7, pages 1–68, 1937.

[16] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[17] Joseph L Doob. Application of the theory of Martingales. *Le calcul des probabilites et ses applications*, pages 23–27, 1949.

[18] Gintare Karolina Dziugaite, Daniel M. Roy, and Zoubin Ghahramani. Training generative neural networks via Maximum Mean Discrepancy optimization. In *Proc. Uncertainty in Artificial Intelligence (UAI)*, 2015.

[19] David A Freedman. On the asymptotic behavior of Bayes' estimates in the discrete case. *The Annals of Mathematical Statistics*, pages 1386–1403, 1963.

[20] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 2015.

[21] Arthur Gretton. Lecture notes on Reproducing kernel Hilbert spaces in Machine Learning, University College London, 2017. `http://www.gatsby.ucl.ac.uk/~gretton/coursefiles/rkhscourse.html`.

[22] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

[23] Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *JMLR*, 13(Jun):1809–1837, 2012.

[24] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *NIPS*, pages 918–926, 2014.

[25] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[26] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learn. Intell. Optim.*, pages 507–523. Springer, 2011.

[27] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *J Global Optim*, 13(4):455–492, 1998.

[28] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *The 2nd International Conference on Learning Representations (ICLR)*, 2014.

[30] BJK Kleijn, AW Van der Vaart, et al. The Bernstein-von-Mises theorem under misspecification. *Electronic Journal of Statistics*, 6:354–381, 2012.

[31] Jon M. Kleinberg. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems 15*, pages 463–470. MIT Press, 2003.

[32] Greg Kochanski, Daniel Golovin, John Karro, Benjamin Solnik, Subhodeep Moitra, and D Sculley. Bayesian optimization for a better dessert. In *NIPS, workshop on Bayesian optimization*, 2017.

[33] Brian Kulis and Michael I. Jordan. Revisiting k-means: New Algorithms via Bayesian Nonparametrics. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 513–520, 2012.

[34] Simon Lacoste-Julien, Ferenc Huszár, and Zoubin Ghahramani. Approximate inference for the loss-calibrated bayesian. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 416–424, 2011.

[35] Tuan Anh Le, Maximilian Igl, Tom Rainforth, Tom Jin, and Frank Wood. Auto-encoding sequential monte carlo. In *International Conference on Learning Representations*, 2018.

[36] Stéphanie Lefèvre, Dizan Vasquez, and Christian Laugier. A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1(1):1, 2014.

[37] Stephen Leslie, Bruce Winney, Garrett Hellenthal, Dan Davison, Abdelhamid Boumertit, Tammy Day, Katarzyna Hutnik, Ellen C Royrvik, Barry Cunliffe, Daniel J Lawson, et al. The fine-scale genetic structure of the british population. *Nature*, 519(7543):309–314, 2015.

[38] Emile Mathieu, Tom Rainforth, N Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pages 4402–4412. PMLR, 2019.

[39] J Mockus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[40] Kevin P Murphy. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[41] Michael Osborne. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature.* PhD thesis, PhD thesis, University of Oxford, 2010.

[42] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[43] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2007.

[44] Tom Rainforth, A Goliński, Frank Wood, and Sheheryar Zaidi. Target–aware bayesian inference: how to beat optimal conventional estimators. *Journal of Machine Learning Research*, 21(88), 2020.

[45] Tom Rainforth, Tuan Anh Le, Jan-Willem van de Meent, Michael A Osborne, and Frank Wood. Bayesian optimization for probabilistic programs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 280–288, 2016.

[46] Tom Rainforth and Frank Wood. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*, 2015.

[47] C.E. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[48] Danilo Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[49] B. Schölkopf, A. J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10:1299–1319, 1998.

[50] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[51] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, New York, NY, USA, 2014.

[52] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012.

[53] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable Bayesian optimization using deep neural networks. In *ICML*, 2015.

[54] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 3745–3753, 2016.

[55] Jakub Tomczak and Max Welling. Vae with a vampprior. In *International Conference on Artificial Intelligence and Statistics*, pages 1214–1223. PMLR, 2018.

[56] Tsuyoshi Ueno, Trevor David Rhone, Zhufeng Hou, Teruyasu Mizoguchi, and Koji Tsuda. Combo: an efficient bayesian optimization library for materials science. *Materials discovery*, 4:18–21, 2016.

[57] Mark Van der Wilk. *Sparse Gaussian process approximations and applications.* PhD thesis, University of Cambridge, 2019.

[58] Vladimir Naumovich Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[59] C. Zhang, S. Bengio, M. Herdt, B. Recht, and O Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2017.

Index