Unsupervised Learning

Lecture Notes in Machine Learning

Georgios Pilikos
ggp29@cam.ac.uk

Laboratory for Scientific Computing, Department of Physics
University of Cambridge

Contents

CHAPTER 1

Introduction

The quest for modelling the world goes back centuries in human history. From the movement of planets to the chaotic movements of the stock market, humanity has always endeavor to explain the world using observations. Machine Learning is a field that tackles the problem of turning observations into information and powers our modern society in many different ways (LeCun et al., 2015). From recommendations in our social networks to the automatic diagnosis of diseases, it finds applications in any data-driven discipline. Given observations and assumptions about their generative process, models are formulated and then learned by machine learning algorithms. There are many different methods of learning these models. Three main categories are *supervised learning*, where the observations are labeled (ie value, class) $\{(\mathbf{x}^{(1)}, t^{(1)}), (\mathbf{x}^{(2)}, t^{(2)}), ..., (\mathbf{x}^{(N)}, t^{(N)})\}$, *unsupervised learning*, where the observations are unlabeled (raw data) $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ and *reinforcement learning* where agents take actions while maximizing a reward.

"When we're learning to see, nobody's telling us what the right answers are - we just look. Every so often, your mother says "that's a dog", but that's very little information. You'd be lucky if you got a few bits of information - even one bit per second - that way. The brain's visual system has 10^{14} neural connections. And you only live for 10^9 seconds. So it's no use learning one bit per second. You need more like 10^5 bits per second. And there's only one place you can get that much information: from the input itself." (Gorder, 2006)

-Professor Geoffrey Hinton, University of Toronto and Google

Unsupervised Learning is a sub-field in machine learning that includes models and algorithms that use raw training data and identify patterns/features that can be used to infer values for new data points. That is, models endeavor to learn the underlying generative process of the data using only the inputs. For example, in Figure 1.1, a data set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ of random variables in $\mathcal{R}^2$ is given with the goal of finding structure in the data. This could be a partition of the data set into some number K of groups or an estimate of the distribution of the data points that could be used in many applications. *Clustering* considers the problem of identifying clusters of data points in a multidimensional space. Applications of

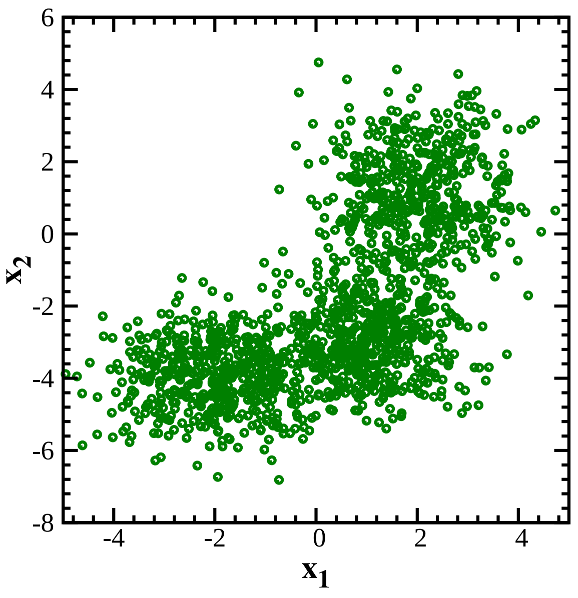


Figure 1.1: Two-dimensional data points

these could be to cluster news articles, medical measurements of biological cells, astrophysical observations that could be used for the identification of new, previously unseen examples in their respective fields.

A closely related problem is that of *density estimation* where the estimation of the probability distribution of multidimensional data points is tackled. Real-world data in multidimensional spaces are usually very complex and can not be represented by simple probability distributions (Gaussian, Dirichlet, Bernoulli etc). Therefore, models of mixtures of probability distributions are used. For example, in Figure 1.1, the data are generated from a mixture of three Gaussian distributions (chapter 2 for Gaussian distribution). The goal of density estimation is to learn the means and covariances of these two-dimensional Gaussian distributions. This is closely related to clustering since in order to solve it, latent (unobserved) variables are used. The introduction of latent variables allows complicated distributions to be formed from simpler components. That is, $z_k^{(i)} \in \{0,1\}$, where $k = 1, ...K$ are latent variables that control assignments of data points to K different distributions (ie drawn from one of the Gaussian distributions) and is very similar to the assignment of data points to one cluster out of K. In fact, mixtures of Gaussians are often used for tackling the clustering problem as discussed in chapter 2. An extension of density estimation is *anomaly detection*, where once the probability distribution of the data is learned, flags of anomalies are raised if a new/unseen data point is obtained that does not seem to be generated by that density. Applications of this could be in credit card fraud detection, power factory failures, detection of failures in unmanned vehicles (drones) and in general in any industry where warning of unexpected events are required automatically and in real-time.

The above models use a latent variable to generate the observations where each observation can only come from one of K partitions (clusters, distributions). That is, they use K hidden binary variables representing on-off encoding for the partition identity which makes

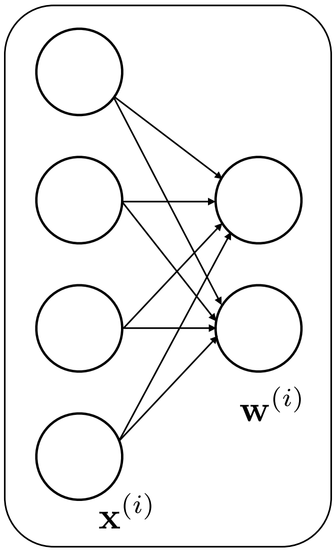


Figure 1.2: Illustration of the use of latent factors to reduce the dimensionality of the input space

their representation power very limited. Complex data might require many latent variables/ factors to describe them. Thus, real-valued latent variables can be used, $\mathbf{w}^{(i)} \in \mathcal{R}^K$. For example, these variables could be drawn from a Gaussian distribution given by:

$$p(\mathbf{w}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{1.1}$$

and a distribution of the inputs $\mathbf{x}^{(i)} \in \mathcal{R}^D$ could be given by:

$$p(\mathbf{x}^{(i)}|\mathbf{w}^{(i)}, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{D}\mathbf{w}^{(i)} + \boldsymbol{\mu}, \boldsymbol{\Psi}), \tag{1.2}$$

where $\boldsymbol{\theta}$ encapsulates all the variables to be learned, $\mathbf{D} \in \mathcal{R}^{D\times K}$ are the connections between the latent variables and the inputs as it can be seen in Figure 1.2. The standard mean and covariance parameters of the multivariate Gaussian distribution are given by $\boldsymbol{\mu}$ and $\boldsymbol{\Psi}$. The above model is called *factor analysis* and can be used to learn latent linear models that depend on factors. The representation power of this model is much stronger and the procedure of obtaining the relevant parameters will be discussed in chapter 3 in detail.

From the above model, it can be seen that high-dimensional data can be represented by lower dimensional factors. High-dimensional data can be projected to lower dimensional manifolds that capture the important features/variations in the data using *dimensionality reduction*. This is extremely useful for numerous reasons. If it is used as a pre-processing step to a regression or classification model, it results in much better predictive accuracy because it allows the model to focus on the important features and ignore the inessentials. Furthermore, it results in much faster computation times since the dimensionality of the problem is much smaller. It can also be used for compression (storing only a portion of the data) and for visualization of high dimensional data into two or three dimensional spaces.

The idea of representing data in lower dimensional manifolds compared to the input space can be used in other concepts, namely for *Representation and Feature Learning* (Bengio, 2013; Bengio et al., 2013). In regression and classification, models use dictionaries of basis functions to transform the input space to another domain which is beneficial for the task and application at hand. For decades, researchers in the field were using their domain expertise to design suitable feature/basis vectors for their specific application. These feature vectors or representations of data would then be used by models to infer information from observations or build classification models. Careful engineering was necessary to identify suitable representations that would allow models to perform their desired task. For example, a human detection system in video would require a representation of an abstract human figure in an image frame which would incorporate our prior knowledge. Using our human intuition, dominant near vertical edges compose a human figure, thus creating feature vectors such as the Histogram of Oriented Gradients (Dalal and Triggs, 2005). It is apparent that the performance of many machine learning techniques heavily depends on the choice of feature vectors (Bengio, 2013) and Feature Learning is a set of techniques that allow machines to learn feature vectors from raw data with great success in many applications (LeCun et al., 2015).

In the Feature Learning literature, there are many routes one can take to solve the problem of learning basis functions. One of the most widely used and effective models is called the Restricted Boltzmann Machine (Hinton, 2002), a probability model for inference. Due to the nature of the problem that searches a huge feature space in high dimensions, exact inference is intractable. Other problems of interest are also solved using probability models with great success such as the Dirichlet Process in clustering and the Beta-Bernoulli Process in feature learning. These also exploit high dimensional spaces and do require some form of approximation to obtain the required probability distribution of quantities of interest. Approximate inference methods on *sampling* have been widely used to numerically obtain a target probability distribution for a model.

The posterior distribution over unobserved model variables might be of direct interest but in general the problem of sampling addresses the need of evaluating expectations in order to make predictions. These expectations are evaluated over posterior distributions of continuous

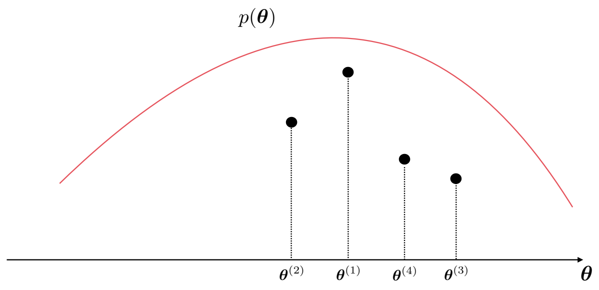

$p(\boldsymbol{\theta})$
$\boldsymbol{\theta}^{(2)}$
$\boldsymbol{\theta}^{(1)}$
$\boldsymbol{\theta}^{(4)}$
$\boldsymbol{\theta}^{(3)}$
$\boldsymbol{\theta}$

Figure 1.3: Approximation of a probability distribution by an ensemble of samples

variables $\boldsymbol{\theta}$ given by:

$$\mathbb{E}[f] = \int f(\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}, \tag{1.3}$$

where $f(\boldsymbol{\theta})$ is some function of the variables (ie likelihood) and $p(\boldsymbol{\theta})$ is their probability distribution (ie posterior). The integral becomes summation in the discrete case. These expectations could be intractable and thus sampling proceeds by drawing independent samples from the distribution $p(\boldsymbol{\theta})$ and approximates the expectations by:

$$\hat{f} = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{\theta}^{(i)}). \tag{1.4}$$

An example is illustrated in Figure 1.3 where a toy probability distribution is approximated by samples. There are of course many problems with sampling since the samples $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^{N}$ might not be independent and thus the effective sample size is much smaller. Models in sampling consider how one can sample as accurate as possible to obtain a sufficient approximation to the target distribution and this will be the subject of chapter 5 in detail.

1.1 Computational Models

Before going into the details of all the above modelling classes, a brief discussion on the basics of modelling is provided since the creation of mathematical models that capture the underlying generative process of observations is the fundamental goal of Machine Learning. A very simple and intuitive example of modelling is fitting a line from observations as can be seen in Figure 1.4. This is an instance of supervised learning, called *linear regression*. In this case, the observations are data points with target values and the goal is to learn a linear model/line that passes through the points. A model for the line is defined by:

$$f(x) = \lambda x + b + \epsilon, \tag{1.5}$$

where λ is the slope and b the intercept to the t-axis which can be encapsulated in $\boldsymbol{\theta} = \{\lambda, b\}$. There is also an assumed noise ϵ in the observations, which is a realistic scenario in the real-world of information. It makes the model robust to noise and controls its complexity. That is, small noise level would mean that the line should pass through all data points exactly (if it was a higher order polynomial model) which would lead to *over-fitting*. A high noise level would oversimplify the model since it would treat the variations in the data as noise and would learn a model that does not pass through the data points. It will thus exhibit the phenomenon of *under-fitting* and would not be able to learn the correct model (error will hide in the noise). The noise level could be learned either by brute force (trying different values) or learned by the observations. However, the main aim is to learn $\boldsymbol{\theta}$.

To learn the parameters, the simplest method is to minimize a cost function that tries to eliminate the error between the generative model $f(x)$ and the observations $\mathbf{t}$. That is:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N}(t^{(i)} - f(x^{(i)}))^2. \tag{1.6}$$

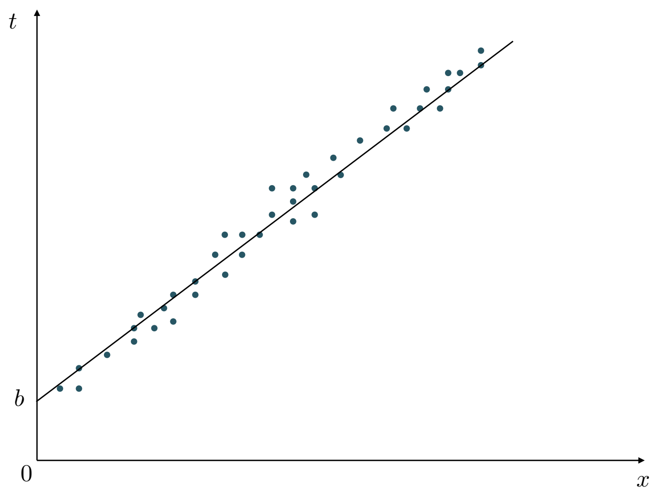


Figure 1.4: Learning a linear model from observations with noise

Then, once these are learned, predictions can be made for new data points by using the learned model with the inferred parameters:

$$f(x^{(*)}) = \hat{\lambda}x^{(*)} + \hat{b}. \tag{1.7}$$

The above model is an example of a *parametric* model. This means that the model has fixed number of parameters. This makes them faster to use and learn but because there are many assumptions about the nature of the data, prior to the learning, it limits their constructive power. For example, in k-means clustering, one must set the number of clusters before the learning commences. An alternative are *non-parametric* models. These models have parameters that grow with the amount of training data used during learning. That is, for example in Dirichlet Processes for clustering, the number of clusters increases as new data points are used (and belong to new/previously unseen clusters). The latter are much more powerful since they are adaptive to new data points and make more sense in the real world where assumptions can not be made safely. However, due to their great flexibility and freedom to explore high-dimensional spaces, they can often be computationally intractable to train. Methods to train models from both categories will be discussed later on.

1.2 Bayesian Non-Parametrics and Inference

Modelling with point-predictions could be a starting avenue for solving problems, however, a probability mass around each inference would be beneficial. *Bayesian Inference* provides a probabilistic framework where all variables are random/uncertain. Then given observations, a posterior distribution of the model's variables is learned. This is based on Bayes' rule (using the sum and product rule of probability theory as defined in (2.5) and (2.6) in section 2.2 later):

$$p(\boldsymbol{\theta}|\mathbf{t}) = \frac{p(\mathbf{t}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathbf{t})}, \tag{1.8}$$

where $p(\boldsymbol{\theta}|\mathbf{t})$ is the posterior distribution, $p(\mathbf{t}|\boldsymbol{\theta})$ is the likelihood model and $p(\boldsymbol{\theta})$ is the prior distribution. Thus, instead of learning one parameter vector $\boldsymbol{\theta}$, a probability distribution over all possible $\boldsymbol{\theta}$ is inferred. Prior knowledge is incorporated through the prior distribution and the likelihood model. For example, one can assume additive Gaussian noise in the observations and define a generative likelihood model as a product of Gaussian distributions for independent data points:

$$p(\mathbf{t}|\boldsymbol{x}, \boldsymbol{\theta}, \sigma^2) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{\left[t^{(i)} - f(\mathbf{x}^{(i)};\boldsymbol{\theta})\right]^2}{2\sigma^2}\right)}. \tag{1.9}$$

Then, the problem of estimating $\boldsymbol{\theta}$ is similar to the least-squares error when minimizing the negative log likelihood which could lead to over-fitting.

To avoid this, a prior distribution over $\boldsymbol{\theta}$ is used. Usually, a flat probability distribution is preferred that does not favor any particular set of configurations and also gives preference to smoother functions. A popular prior is Gaussian which allows easy calculation of the posterior distribution through conjugacy:

$$p(\boldsymbol{\theta}|\alpha) = \prod_{m=0}^{M-1} \sqrt{\frac{\alpha}{2\pi}} e^{-\frac{\alpha}{2}\boldsymbol{\theta}_m^2}. \tag{1.10}$$

Using Bayes' rule:

$$p(\boldsymbol{\theta}|\mathbf{t}, \sigma^2, \alpha) = \frac{p(\mathbf{t}|\boldsymbol{\theta}, \sigma^2)p(\boldsymbol{\theta}|\alpha)}{p(\mathbf{t}|\sigma^2, \alpha)}, \tag{1.11}$$

the posterior distribution of $\boldsymbol{\theta}$ is also Gaussian with $\mathbb{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ due to the use of a Gaussian conjugate prior. The exact formulae of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can be found in (Tipping, 2004). Note that the dependency on $\mathbf{x}$ is dropped for simplicity.

In addition to the definition of the prior on $\boldsymbol{\theta}$, hyper-priors on α and σ^2 are necessary. This is because, we are not directly interested in these hyperparameters and thus, we should average over all possible values. This results in:

$$p(\boldsymbol{\theta}, \sigma^2, \alpha|\mathbf{t}) = \frac{p(\mathbf{t}|\boldsymbol{\theta}, \sigma^2)p(\boldsymbol{\theta}|\alpha)p(\alpha)p(\sigma^2)}{p(\mathbf{t})}. \tag{1.12}$$

Once we obtain the posterior distribution, we use it to obtain the predictive distribution:

$$p(t^{(*)}|\mathbf{t}) = \int p(t^{(*)}|\boldsymbol{\theta}, \sigma^2)p(\boldsymbol{\theta}, \sigma^2, \alpha|\mathbf{t})d\boldsymbol{\theta}d\sigma^2 d\alpha. \tag{1.13}$$

In practice, the above integral is analytically intractable. This is because $p(\mathbf{t})$ in equation (1.12) is a normalizing probability distribution that integrates the numerator. This is an integral in high dimensions and active research in Bayesian Inference focuses on their approximation. The main directions of research are in *variational Bayes* techniques (Beal and Ghahramani, 2003) and *Markov Chain Monte Carlo* (Andrieu et al., 2003). We will discuss the latter in detail in chapter 5. For different applications, one should use different priors, likelihood models and approximation techniques. In the following chapters, we will discuss different models for clustering (Dirichlet Process) and feature learning (Beta-Bernoulli Process) that use this framework under the framework of *Bayesian Non-Parametrics*.

1.3 No free lunch theorem

From this brief introduction to Machine Learning, it can be seen that there are different types of models for different tasks (clustering, dimensionality reduction, feature learning etc). From within these types there are many possibilities for the models by varying the number of parameters, by varying the assumptions on noise, by using different basis functions and non-linearities and by using various learning algorithms to obtain the relevant parameters via training. Some configurations of models work better than others in certain situations and applications.

There is no single best model that works optimally for all types of problems and applications. This is because, we incorporate assumptions and prior knowledge via the models' structure. Part of the machine learning expertise is to choose the best under certain circumstances and we will discuss how to develop many different types of models to solve different problems by finding the best combination of model definitions, number of parameters, basis functions and algorithms.

1.4 Organization of notes

In chapter 2, the clustering problem will be addressed and discussions on different models will be provided. Firstly, a deterministic parametric approach will be discussed, then a probabilistic parametric approach and then a probabilistic non-parametric approach with illustrations of their different merits and pitfalls. Then, in chapter 3, the factor analysis problem will be addressed. Dimensionality reduction and source separation problems will also be introduced using models of similar form to that of factor analysis. In chapter 4, representation and feature learning will be discussed and illustrations on how to learn basis functions from data are provided. Different models are discussed ranging from deterministic cost functions with regularizers along with probabilistic models that span very high dimensional spaces. Finally, in chapter 5, approximate inference and sampling is discussed that allows the estimation of high-dimensional integrals in order to obtain the necessary statistics for models in high-dimensions. Further reading and more in depth explanations can be found in the papers that are referenced per chapter along the text. These provide an in-depth explanation and further insights. In addition, the textbooks by (Bishop, 2006), (MacKay, 2002) and (Murphy, 2012) cover the material very well and are highly recommended in order to grasp the concepts in the lecture notes in detail.

CHAPTER 2

Clustering

The need for automatic identification of groups in unlabeled data is increasing with the rapid explosion of information in the world. It is a very useful task that provides a first level of data understanding in a system. Clustering models can be categorized into different types depending on how they model the groups. *Flat clustering* deals with the partition of data points into disjoint sets and *hierarchical clustering* creates a nested tree of partitions with different levels. All the models discussed will be in the flat clustering category. Another distinction is whether the model is *probabilistic* or *deterministic*. That is, it provides a probability/confidence for each assignment of a data point to a cluster if it is probabilistic, known also as *soft clustering* or it provides a hard assignment if it is deterministic, known also as *hard clustering*. In addition, another important distinction is whether the model is *finite* or an *infinite mixture model*. This depends on the definition of the model and controls how it determines the number of clusters. This can be set *a priori* or can grow with the data. Models from both categories will be discussed since the number of clusters is a very strong assumption about the observed data and can have a significant effect in performance.

Applications of clustering models can be seen in many data-driven areas. From genome clustering in biology to market segmentation in finance, clustering can be used to group known behaviors or even detect unusual behavior for *anomaly detection* if a data point behaves differently. An example of clustering is image segmentation where different colors (pixel intensities) are clustered separately with the number of clusters corresponding to the colors used. This is an example of how clustering can be used in image compression where it is possible to represent the image with much fewer colors and still obtain a relatively good representation.

2.1 K-means clustering

A simple yet very popular algorithm for clustering is the K-means clustering. It provides hard clustering and has finite number of parameters where the number of clusters is chosen manually and does not grow with the data. Consider a data set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ of N data points where each $\mathbf{x}^{(i)} \in \mathcal{R}^D$. Each data point could be the raw input or a feature vector with D features. The goal is to partition the data set into K clusters with K fixed *a priori*. A cluster can be viewed as a group of data points very close together in some metric and the distances between other data points in other clusters is much larger.

Each cluster is characterized by its center and thus a variable for this is required per cluster. It is common to define $\boldsymbol{\mu}_k \in \mathcal{R}^D$ with $k = 1, ..., K$ and the goal is to find these along with the cluster assignments for all data points that satisfy a certain metric (ie sum of squared distance between $\boldsymbol{\mu}_k$ and $\mathbf{x}^{(i)}$). For each data point $\mathbf{x}^{(i)}$, a latent variable is introduced that controls its assignment to a cluster. This is defined as a $K-$dimensional binary vector that each corresponding k location is set to 1 if the data point belongs in the k-th cluster and 0 otherwise. That is, $r_k^{(i)} \in \{0, 1\}$ for $k = 1, ..., K$ describing the K clusters and $i = 1, ..., N$ describing for which $\mathbf{x}^{(i)}$ it refers. This is indicated by $r_j^{(i)} = 1$ if $\mathbf{x}^{(i)}$ is in cluster j and 0 otherwise.

In order to solve this problem, the following cost function is formulated:

$$J = \sum_{i=1}^{N} \sum_{k=1}^{K} r_k^{(i)} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_k\|^2, \tag{2.1}$$

and thus by minimizing this, it is possible to obtain the required set of clusters characterized by $\boldsymbol{\mu}_k \ \forall \ k$ and the indicators $r_k^{(i)}$ for all data points. To achieve this, an iterative procedure is followed where firstly, an initial value of the centers is assigned. The procedure begins with the cost function J minimized with respect to the indicator variables keeping the centers fixed. Then, the new values for the indicator variables are considered fixed and the cost function J is minimized with respect to the centers. This procedure is repeated until no changes are observed or after a pre-defined number of iterations.

To formulate the above, first the indicator variables are optimized via:

$$r_k^{(i)} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{if otherwise.} \end{cases} \tag{2.2}$$

with all $\boldsymbol{\mu}_j$ fixed. This corresponds to the assignment of data point $\mathbf{x}^{(i)}$ to the closest cluster center. The optimization of the $\boldsymbol{\mu}_k$ follows a different path. If the $r_k^{(i)}$ are held fixed, J is a quadratic function with respect to $\boldsymbol{\mu}_k$. Thus, by setting the derivative of J to zero:

$$2 \sum_{n=1}^{N} r_k^{(i)} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k) = 0, \tag{2.3}$$

and re-arranging:

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_k^{(i)} \mathbf{x}^{(i)}}{\sum_{i=1}^{N} r_k^{(i)}}. \tag{2.4}$$

The above formulation for the $\boldsymbol{\mu}_k$ has a great intuitive interpretation as the mean of all the data points selected in a particular cluster. The algorithm's steps are summarized in Algorithm 1. The above iterative procedure is repeated until convergence to a solution. The cost function J is guaranteed to decrease (Bishop, 2006) but the solution is not guaranteed to be a global minimum. That is, it could converge into a local minimum. This heavily depends on the initialization of the $\boldsymbol{\mu}_k$ for all clusters which is usually done randomly. In order to obtain better clustering performance, many K-mean clustering instances are initialized randomly with different initial cluster centers. Then, when converged, the instance with the lowest cost function J is chosen.

Algorithm 1 K-means clustering

```
Initialization: $\boldsymbol{\mu}_k$ are initialized $\forall\ k$

1: repeat
2:     for $i = 1$ to $N$ do
3:         $r_k^{(i)} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{if otherwise.} \end{cases}$
4:     for $k = 1$ to $K$ do
5:         $\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_k^{(i)} \mathbf{x}^{(i)}}{\sum_{i=1}^{N} r_k^{(i)}}$
6: until convergence;
7: return ( $\{\boldsymbol{\mu}_k\}_{k=1}^{K}, \{\mathbf{r}^{(i)}\}_{i=1}^{N}$ )
```

In order to illustrate the K-means algorithm, an example is provided in two-dimensions. Given a data set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(1500)}\}$ of 1500 data points where each $\mathbf{x}^{(i)} \in \mathcal{R}^2$, the goal is to find clusters in the data. An illustration of these can be seen in Figure 2.1 where two clusters are reasonable, however, setting the number of clusters is not usually trivial. This can be seen in Figure 1.1 where the number of clusters can not be safely chosen. Later, a technique where the number of clusters are inferred from the data is discussed. Firstly, $K = 2$ is set and the cluster centers are initialized randomly. Figure 2.2a illustrates the initialization with the cluster centers marked with a cross. Then, the algorithm optimizes the latent variables $\mathbf{r}^{(i)}$ for all data points and assigns each data point to the closest cluster center as seen in Figure 2.2b. After that, the cluster means $\boldsymbol{\mu}_k$ are optimized for all clusters and shifted to their new location. The procedure is repeated in Figures 2.2c - 2.2g until no changes occur.

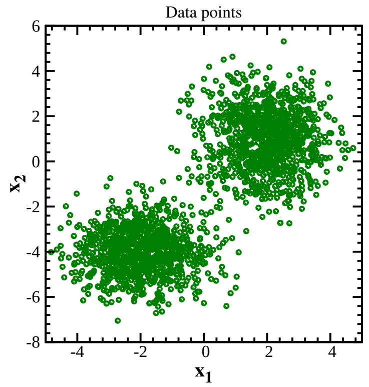
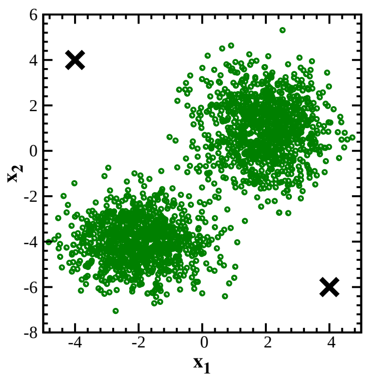
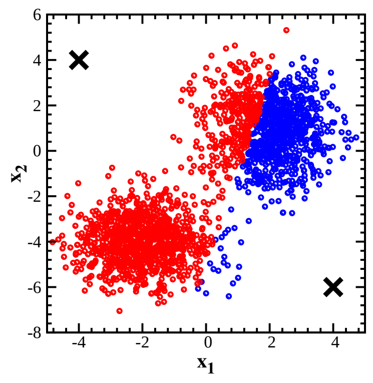


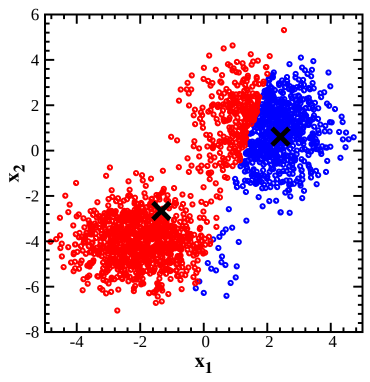
Figure 2.1: Data points in two-dimensions



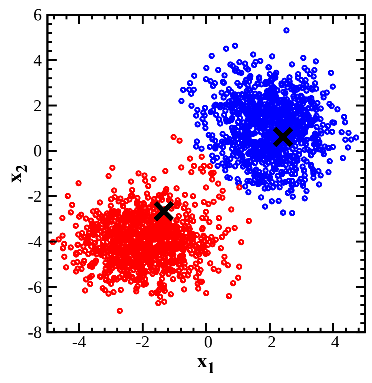
(a) Initialization of clusters



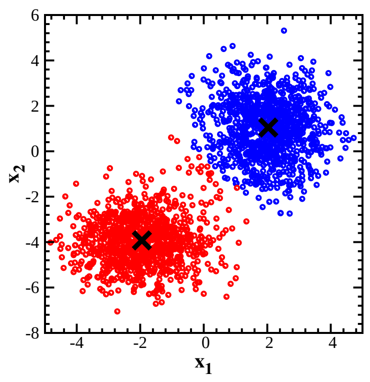
(b) **Iter. 1:** Optimized $r_k^{(i)}\ \forall\ i$



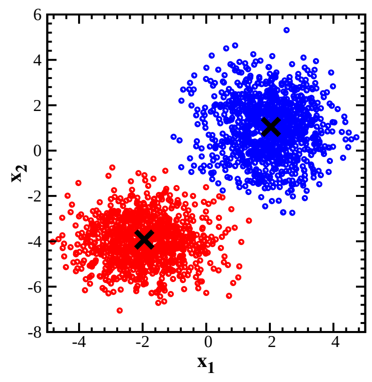
(c) **Iter. 1:** Optimized $\boldsymbol{\mu}_k\ \forall\ k$



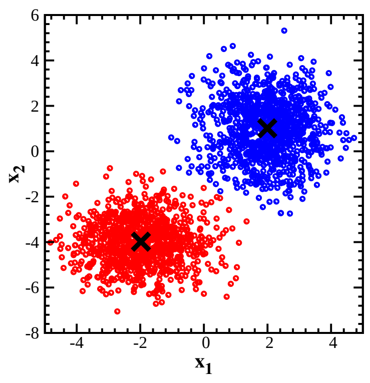
(d) **Iter. 2:** Optimized $r_k^{(i)}\ \forall\ i$



(e) **Iter. 2:** Optimized $\boldsymbol{\mu}_k\ \forall\ k$



(f) **Iter. 3:** Optimized $r_k^{(i)}\ \forall\ i$



(g) **Iter. 3:** Optimized $\boldsymbol{\mu}_k\ \forall\ k$

Figure 2.2: Illustration of the K-means clustering algorithm

2.2 Mixture Models and Gaussians

Complicated probabilistic models with many correlations in their variables are very difficult to train in order to allow the representation of all variations in the data. This is because they require complicated probability distributions and these can not always be expressed explicitly. In order to tackle problems with complicated distributions, mixtures of simpler probability distributions are used. These are called *latent variable models* that usually have fewer parameters as opposed to models that directly represent correlations between the original distribution. In addition, they insert a bottleneck, thus allowing to learn a compressed representation of the data as can be seen in Figure 1.2.

Probabilistic latent variable models are based on the *sum* and *product rule* of probability theory. The sum rule states that a marginal distribution over one random variable (say $\mathbf{x}$) is obtained from the joint distribution of that variable and another (say $\mathbf{z}$) by marginalization of the latter. That is:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}), \tag{2.5}$$

where the sum is performed over all possible values of $\mathbf{z}$. The product rule on the other hand states that the joint distribution between two random variables is obtained by the product of the conditional distribution of one given the other and the marginal distribution of the latter. That is:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}). \tag{2.6}$$

Now, the combination of the above leads to:

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

which is the basis for the creation of *mixture models*. A particular mixture model of interest is the *Gaussian mixture model* which as the name suggests, Gaussian distributions are linearly combined to represent much more complicated probability distributions.

Building more complex probability distributions from simple Gaussian distributions is very useful and Gaussian mixture models can be used in different areas. In particular, the problem of clustering can be solved by these models by modelling each cluster as a separate multivariate Gaussian distribution. Before discussing this in more detail, the Gaussian distribution and the mixture of Gaussians is discussed.

Gaussian Distribution

The Gaussian distribution is a widely used probability distribution for random variables. It is applicable for many models due to the *central limit theorem* which states that under certain conditions, the average of multiple random variables which itself is a random variable has a distribution that tends to a Gaussian in the limit. For a single variable x, the distribution is defined by:

$$\mathcal{N}(x|\mu, \sigma^2) = p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp^{\{-\frac{1}{2\sigma^2}(x-\mu)^2\}}, \tag{2.8}$$

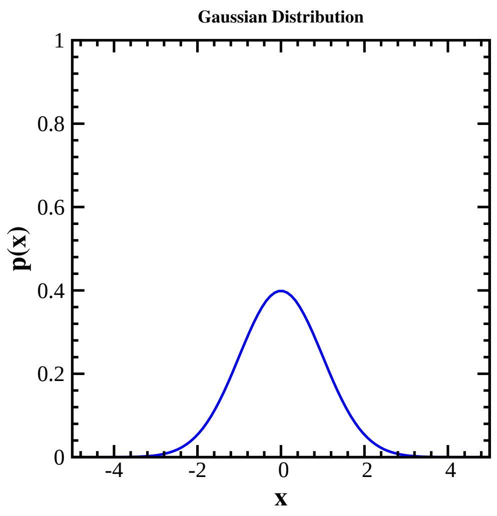


Figure 2.3: Gaussian distribution of a single variable with $\mu = 0$ and $\sigma^2 = 1$

where μ is the mean and σ^2 is the variance of the distribution. An example can be seen in Figure 2.3 with $\mathcal{N}(x|0,1)$. For a vector $\mathbf{x} \in \mathcal{R}^D$, a multivariate Gaussian distribution is defined by:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp^{\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right\}}, \tag{2.9}$$

where $\boldsymbol{\mu} \in \mathcal{R}^D$ is the corresponding mean in D dimensions and $\boldsymbol{\Sigma} \in \mathcal{R}^{D\times D}$ is the covariance matrix with $|.|$ representing the determinant of a matrix. In order to illustrate the significance of Gaussian mixture models, consider a one-dimensional variable x distributed with $p(x)$ in Figure 2.4.

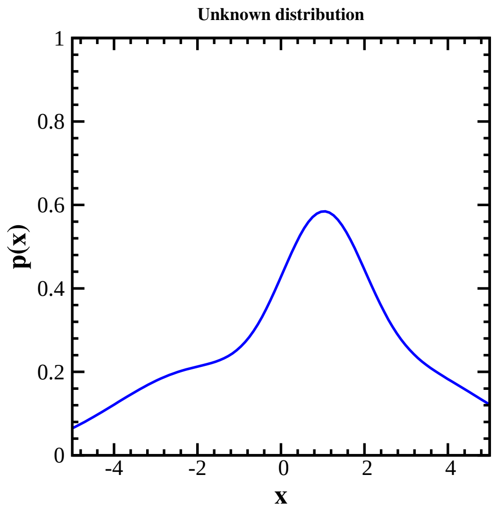


Figure 2.4: Complicated distribution of a single variable x

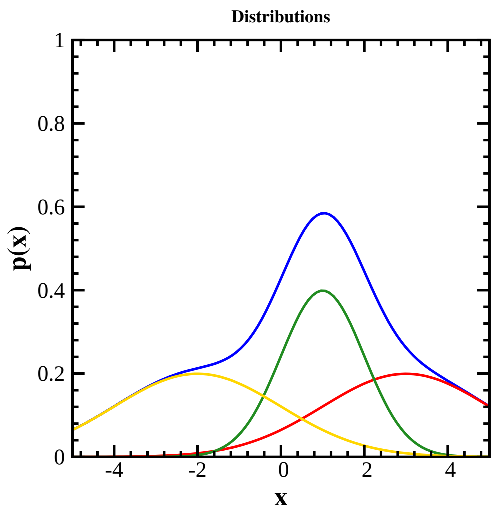


Figure 2.5: Mixture of three Gaussian distributions representing a complicated distribution

This distribution can not be represented explicitly by a single expression. However, if it is possible to use a linear combination of different probability distributions to represent it, it is possible to obtain an equivalent and simpler representation form. In this example, the particular distribution can be represented by three Gaussian distributions with different means and variances as it can be seen in Figure 2.5. In many dimensions, this becomes much more complicated but the same idea holds.

In order to use Gaussian mixture models, the introduction of latent variables is necessary. This is because, it is necessary to identify the contribution of each of the Gaussian distributions (or clusters in clustering) out of all distributions (clusters) in the mixture. The latent variables control the assignment of samples to base distributions but it is beneficial to also obtain a weight indicating the probability of actually coming from that distribution. A graphical model of the joint distribution $p(\mathbf{x}, \mathbf{z})$ of the input and the latent variables can be seen in Figure 2.6.

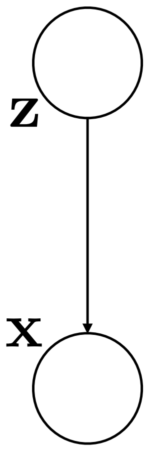


Figure 2.6: Graphical model of the joint distribution $p(\mathbf{x}, \mathbf{z})$

In order to behave as required, $\mathbf{z}$ should have a categorical nature. This is obtained by defining it as a binary random variable having a 1-of-K representation. Thus, $\mathbf{z} \in \{0,1\}^K$ where K is the number of base distributions/clusters in the mixture. Due to the fact that this is a categorical model, $\mathbf{z}$ must satisfy:

$$\sum_{k=1}^{K} z_k = 1, \tag{2.10}$$

where at any given point, there is only one non-zero element in $\mathbf{z}$. This vector of latent variables provides the mechanism to identify one distribution/cluster but it is necessary to have a probability of a sample belonging to any of the distributions/clusters. This is achieved by defining the marginal distribution of $p(\mathbf{z})$ as:

$$p(z_k = 1) = \pi_k, \tag{2.11}$$

Since π_k are probabilities, they must satisfy $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$. To obtain the marginal distribution over the entire latent variable vector $\mathbf{z}$, recall that $\mathbf{z}$ is a 1-of-K vector and thus:

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}. \tag{2.12}$$

This means that only one weight π_k will be active, corresponding to the non zero element in z_k (ie for the k-th distribution/cluster). However, by using $p(z_k = 1)$ for all k, it is possible to obtain the probability for every single distribution/cluster via the Bayes' rule as discussed later on.

Another vital component for this model is the conditional distribution of $\mathbf{x}$ given $\mathbf{z}$. If a sample is generated by a distribution k, then the conditional probability distribution is given by:

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{2.13}$$

where $\boldsymbol{\mu}_k$ is the mean of the k-th Gaussian distribution and $\boldsymbol{\Sigma}_k$ its covariance. Consequently, the conditional distribution over all $\mathbf{z}$ is:

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}. \tag{2.14}$$

Using the sum and product rules of probability theory, the marginal distribution of $\mathbf{x}$, which is the model of the samples, is given by:

$$\begin{aligned} p(\mathbf{x}) &= \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) \\ &= \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \end{aligned} \tag{2.15}$$

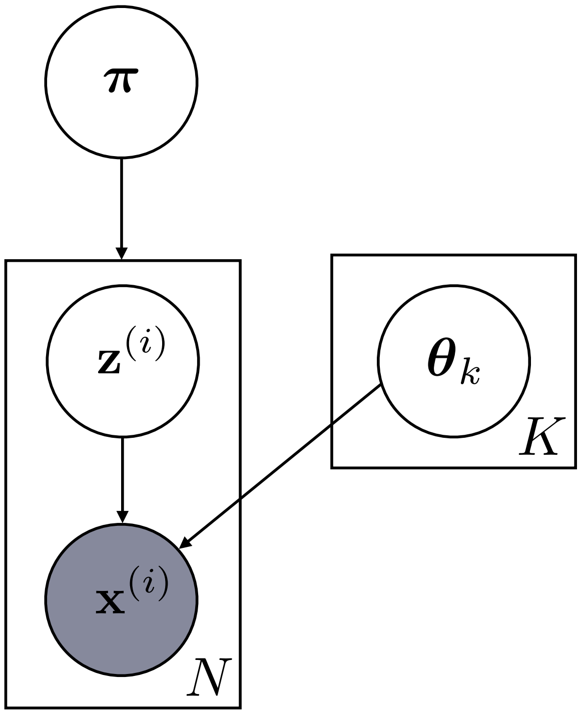


Figure 2.7: Graphical model of the Gaussian mixture model

This model for $\mathbf{x}$ can be generalized for samples that are independent and identically distributed (i.i.d.) $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ and for every sample, there is a corresponding $\mathbf{z}^{(i)}$. The corresponding graphical model is shown in Figure 2.7 where $\boldsymbol{\theta}_k = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$.

Thus, the model is trained using a set of observations for clustering and indicates their distribution/cluster with a certain probability. The probability is obtained for all distribution/clusters in the mixture and is viewed as the *responsibility* that cluster k takes for the explanation of sample $\mathbf{x}$. This is defined as:

$$\begin{aligned}\gamma(z_k) &= p(z_k = 1|\mathbf{x}) \\ &= \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{p(\mathbf{x})} \\ &= \frac{p(z_k = 1)p(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\mathbf{x}|z_j = 1)} \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)},\end{aligned} \tag{2.16}$$

where the Bayes' rule was used to obtain the posterior distribution. This is done for every distribution/cluster in the mixture to obtain the probability that sample $\mathbf{x}$ came from the k-th component.

Everything is now in place with respect to modelling and thus the discussion will continue on how to obtain the necessary statistics for the model given the training data (observations). A very powerful algorithm for training latent variable models is called *Expectation-Maximization* (EM) algorithm and follows a two-step iterative procedure. Given observations

$\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\} \in \mathcal{R}^D$, it is beneficial to compactly represent them in $\mathbf{X} \in \mathcal{R}^{N\times D}$. Then, in order to obtain the relevant statistics $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k\}_{k=1}^K$ from the observations, it is necessary to maximize the likelihood function of the model. This is given by (2.15) but since there are N i.i.d. samples, its product is required:

$$p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{2.17}$$

Taking the log of the above expression for ease of calculation, the log-likelihood is given by:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}. \tag{2.18}$$

In order to obtain the maximum likelihood estimation with the corresponding statistics, the derivatives of the above are required. Firstly, setting the derivative of (2.18) to zero with respect to $\boldsymbol{\mu}_k$:

$$\sum_{i=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k) = 0. \tag{2.19}$$

Multiplying by $\boldsymbol{\Sigma}_k$ and re-arranging:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_k^{(i)} \mathbf{x}^{(i)}, \tag{2.20}$$

where $N_k = \sum_{i=1}^{N} \gamma_k^{(i)}$ translates to the number of points assigned to distribution/cluster k. It can be seen that the mean $\boldsymbol{\mu}_k$ is estimated by a weighted sum of all the samples with weights being their respective responsibilities. By using the above observation, the covariance is given by:

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_k^{(i)} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k)(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k)^T, \tag{2.21}$$

which is a weighted version of the maximum likelihood estimate for a single Gaussian. Finally, the derivative with respect π_k is required. Here, there is an additional constraint to satisfy the definition of probabilities and thus a Lagrange multiplier is added to convert the expression to an unconstrained optimization resulting in:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right), \tag{2.22}$$

where λ is the Lagrange multiplier. Differentiating the above with respect to π_k and setting to zero:

$$\sum_{i=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda = 0. \tag{2.23}$$

After some manipulations and using the constraint on the π_k:

$$\pi_k = \frac{N_k}{N}. \tag{2.24}$$

From the above derivations, it can be seen that the statistics to be estimated depend on the responsibilities $\gamma_k^{(i)}$ which in turn depend on the statistics. Thus, an iterative scheme emerges for finding the solution to this maximization problem of the likelihood. This procedure is the EM algorithm. It comprises of the *Expectation* step where the responsibilities are updated using the current state of the statistics.

Then, the *Maximization* step follows where, using the new responsibilities and the old statistics, new estimates for all three variables are obtained. This procedure is repeated until a convergence criterion on the log-likelihood is satisfied or after a pre-defined number of iterations. The EM algorithm is summarized in Algorithm 2 with a pre-defined number of distributions/clusters k and returns the means and covariances of the mixture along with the responsibilities of each component for each sample in the data set with $\boldsymbol{\gamma}(z_k) \in \mathcal{R}^N$.

The EM algorithm offers many advantages for clustering compared to the K-means algorithm. It is built around a *probabilistic* framework which provides for each sample a confidence for its cluster identity (ie *soft clustering*). However, it takes much longer to run and thus it is very common to initialize the statistics with the output of the K-means algorithm. The initialization also plays a very important rule since the EM algorithm does not guarantee global convergence to a global maxima and thus different initialization could lead to different local maxima. Validation of the results with different initialization and then choosing the configuration with the largest log-likelihood could improve performance.

Algorithm 2 EM algorithm

```
Initialization: $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$ are initialized $\forall\ k$ and $N_k = \sum_{i=1}^{N} \gamma(z_k^{(i)})\ \forall\ k$

 1: repeat
 2:     E step:
 3:     for $i = 1$ to $N$ do
 4:         for $k = 1$ to $K$ do
 5:             $\gamma(z_k^{(i)}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$
        M step:
 6:     for $k = 1$ to $K$ do {
 7:         $\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_k^{(i)}) \mathbf{x}^{(i)}$
 8:         $\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_k^{(i)}) (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k^{\text{new}})(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k^{\text{new}})^T$
 9:          $\pi_k^{\text{new}} = \frac{N_k}{N}$}
10: until convergence;
11: return ( $\{\boldsymbol{\mu}_k\}_{k=1}^{K}, \{\boldsymbol{\Sigma}_k\}_{k=1}^{K}$ , $\{\boldsymbol{\gamma}(z_k)\}_{k=1}^{K}$ )
```

2.3 Dirichlet and the Chinese Restaurant Process

The discussion so far involved parametric clustering models with a pre-defined number of clusters. In some situations however, it might not be possible to determine this number beforehand (ie in Figure 2.8) which could lead to very poor performance. In addition, parametric models are very limited to their capacity to handle the complexity of the world since

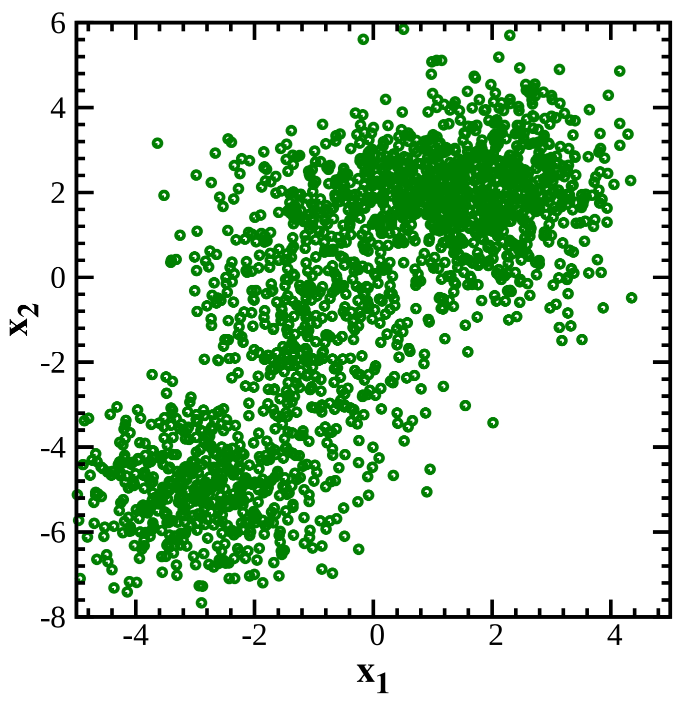


Figure 2.8: Illustration of the difficulty of determining the number of clusters *a priori*

it might be necessary to adapt the behavior as new data points arrive in the process. This means that new data points might not belong to any of the previous clusters in the model and thus the model must be able to create new distinct clusters at run-time.

The *Dirichlet Process* (Neal, 2000) is used in the Bayesian non-parametrics framework for infinite mixture models where the number of parameters/mixture components is unknown and cannot be described by a finite number of parameters. It is a very important building block in Bayesian non-parametrics as a prior distribution with a wide support and tractable inference. The finite mixture model considered in the previous section was defined by:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{2.25}$$

To be truly Bayesian, a prior distribution on the probabilities $\boldsymbol{\pi}$ and on the variables $\boldsymbol{\theta}_k$ describing the Gaussian distributions are required. A Dirichlet prior distribution on $\boldsymbol{\pi}$ is used:

$$Dir(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}. \tag{2.26}$$

For different values of $\boldsymbol{\alpha}$, different probability mass is assigned to different clusters. A prior Gaussian distribution on the means of the clusters could be used, defined by $\boldsymbol{\gamma}$ in the illustration of the model which can be seen in Figure 2.9.

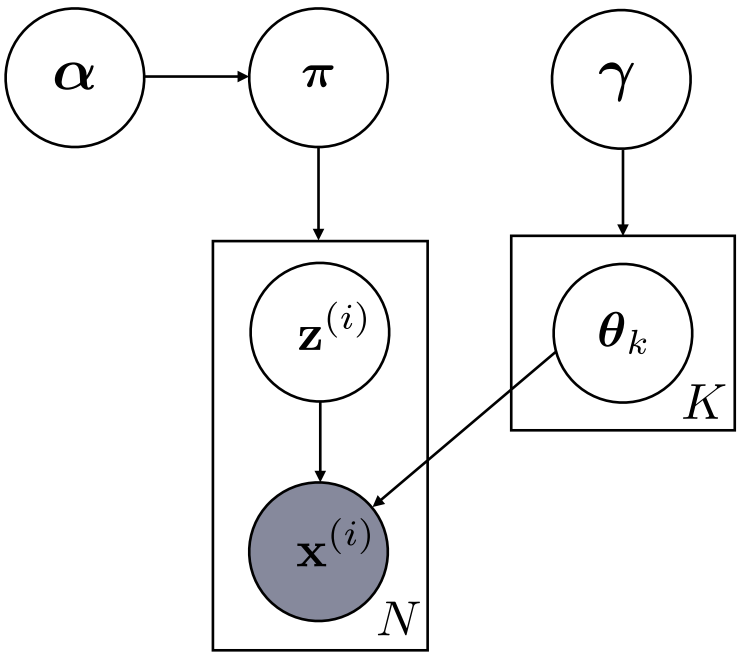


Figure 2.9: Illustration of the graphical model of a mixture model with prior distributions on the variables

An equivalent model form is to define a new distribution G:

$$G(\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \delta_{\boldsymbol{\theta}_k}(\boldsymbol{\theta}), \tag{2.27}$$

where $\boldsymbol{\theta}_k \sim H$ with H a Gaussian distribution. The distribution G is used to obtain a parameter $\bar{\boldsymbol{\theta}}$ which in turn produces $\mathbf{x}$. Thus, for a data point $\mathbf{x}^{(i)}$ and a pre-defined K, firstly K Gaussian means $\boldsymbol{\theta}_k$ are sampled from a Gaussian prior H with each assigned a π_k mixing weight. The weighted sum of the means (delta spikes) is G. From G, a new parameter $\bar{\boldsymbol{\theta}}^{(i)}$ is drawn with probability π_k being equal to $\boldsymbol{\theta}_k$. Then $\bar{\boldsymbol{\theta}}^{(i)}$ is used to obtain $\mathbf{x}^{(i)} \sim \mathcal{N}(\bar{\boldsymbol{\theta}}^{(i)}, \boldsymbol{\Sigma})$. This model can be illustrated in Figure 2.10.

Sampling from the above model will always produce K clusters due to the finite distribution G. Thus, replacing G with a random probability measure, it is possible to obtain variable number of clusters. This random probability measure is the *Dirichlet Process* defined by $G \sim DP(\alpha, H)$ where α is called the *concentration parameter* and H is the *base measure*. A Dirichlet Process is a distribution over probability measures, which means that it puts a probability over a certain partition of the space. Thus, K is not fixed and can vary with certain probability depending on the previous data points already observed. If there are already N samples in the model, this means that there have been N $\bar{\boldsymbol{\theta}}^{(i)} \sim G$ and the new G (posterior distribution after N observations) is:

$$G|\bar{\boldsymbol{\theta}}^{(1)}, ..., \bar{\boldsymbol{\theta}}^{(N)}, \alpha, H \sim DP\left(\alpha + N, \frac{1}{\alpha + N}(\alpha H + \sum_{i=1}^{N} \delta_{\boldsymbol{\theta}_i})\right), \tag{2.28}$$

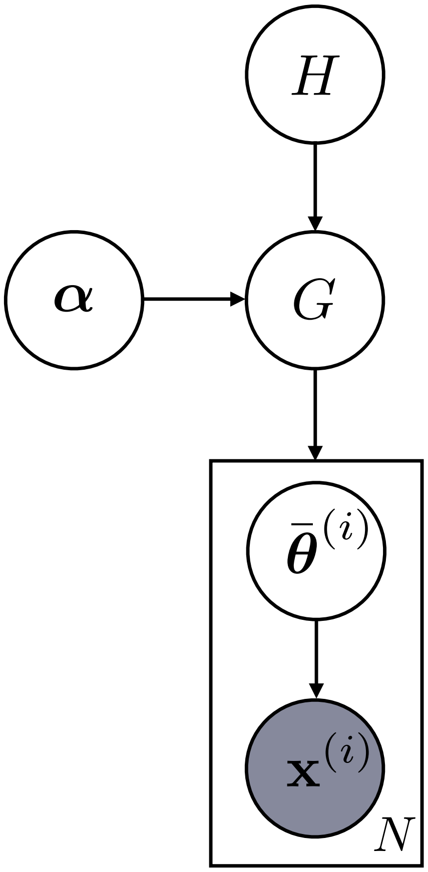


Figure 2.10: Illustration of the graphical model of an alternative view of mixture models

which is also a Dirichlet Process. The predictive distribution of the next $\bar{\boldsymbol{\theta}}^{(N+1)}$ is given by:

$$p(\bar{\boldsymbol{\theta}}^{(N+1)} = \boldsymbol{\theta} | \bar{\boldsymbol{\theta}}^{(1:N)}, \alpha, H) = \frac{1}{\alpha + N} \left(\alpha H(\boldsymbol{\theta}) + \sum_{k=1}^{K} N_k \delta_{\bar{\boldsymbol{\theta}}_k}(\boldsymbol{\theta}) \right), \tag{2.29}$$

where N_k is the number of data points that belong in cluster k. This is known as the *Polya urn* or *Blackwell-MacQueen* sampling scheme. If the viewpoint is the cluster identity, then this is called the *Chinese Restaurant Process* which treats the clusters as tables and the observations as the customers. When a new customer enters the restaurant, he can choose to sit at an existing table with probability N_k (the number of people at that table) or choose to sit at a new table with a diminishing probability $\frac{1}{(\alpha+N)}$. Further information on this model definition, its construction and on how to fit an infinite Dirichlet Mixture Model can be found in Chapter 25 of (Murphy, 2012).

CHAPTER 3

Dimensionality Reduction and Factor Analysis

Models so far have been using latent variables of an on-off encoding scheme. That is, binary variables were used to indicate which cluster or mixture explains a particular observation. This is useful but limits the representation power of modelling. Latent variables with real values could be used to describe the data. These can be used as a building block for a generative model that can model several behaviors. At the same time, many data sets that live in a high-dimensional space could be represented by a subspace with a much lower dimensionality by removing redundant dimensions (that do not offer any changes in variation). Thus, by identifying this subspace and the corresponding latent variables, it could be possible to compress the data and reduce the computational time due to the lower dimensional space. In addition, latent variables can be used to create models that treat the data as a mixture of different sources. This can then help separate different sources that have been mixed without knowing a priori any information about the sources or the weighting of each source in the observed mixed signal. In this chapter, all the above variations of latent variable models with real values will be introduced and different behaviors will be discussed.

3.1 Factor Analysis

Due to the limiting representation power of the binary latent variables and the motivation that real-world data often live in a lower dimensional manifold, alternative latent variable models were introduced. Consider Figure 1.2 where observations $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\} \in \mathcal{R}^D$ are modelled via latent variables $\{\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ..., \mathbf{w}^{(N)}\} \in \mathcal{R}^K$ and $K << D$. A straightforward model for the latent variables is to consider them as draws from a Gaussian distribution defined by:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{I}), \tag{3.1}$$

and then the observations are linear combinations of the latent variables with appropriate weightings incorporated in $\mathbf{D}$ giving:

$$p(\mathbf{x}|\mathbf{w}) = \mathcal{N}(\boldsymbol{\mu} + \mathbf{D}\mathbf{w}, \boldsymbol{\Psi}), \tag{3.2}$$

where $\boldsymbol{\mu} \in \mathcal{R}^D, \mathbf{D} \in \mathcal{R}^{D\times K}$ and $\boldsymbol{\Psi} \in \mathcal{R}^{D\times D}$ is a diagonal matrix. The model uses a diagonal covariance matrix in order to explain the data using the latent variables and not via the correlations in $\boldsymbol{\Psi}$. This model is called *factor analysis* and the goal is to infer the coefficients $\mathbf{w}$ and the weightings incorporated by $\mathbf{D}$.

An equivalent form of the above model is given by:

$$\begin{aligned}\mathbf{w} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}),\\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Psi}),\\ \mathbf{x} &= \boldsymbol{\mu} + \mathbf{D}\mathbf{w} + \boldsymbol{\epsilon}.\end{aligned} \tag{3.3}$$

The joint probability distribution that describes the model is given by:

$$p(\mathbf{x}, \mathbf{w}) = \mathcal{N}(\boldsymbol{\mu}_{wx}, \boldsymbol{\Sigma}), \tag{3.4}$$

where $\boldsymbol{\mu}_{wx}$ is the joint mean of the model. By using the fact that the expectation $\mathbb{E}[\mathbf{w}] = 0$:

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \mathbb{E}[\boldsymbol{\mu} + \mathbf{D}\mathbf{w} + \boldsymbol{\epsilon}]\\ &= \boldsymbol{\mu} + \mathbf{D}\mathbb{E}[\mathbf{w}] + \mathbb{E}[\boldsymbol{\epsilon}]\\ &= \boldsymbol{\mu}.\end{aligned} \tag{3.5}$$

Therefore,

$$\boldsymbol{\mu}_{wx} = \begin{bmatrix}\mathbf{0}\\ \boldsymbol{\mu}\end{bmatrix}. \tag{3.6}$$

Following the same line of thought, the covariance $\boldsymbol{\Sigma}$ is required. This is split into four different blocks, namely $\boldsymbol{\Sigma}_{ww}, \boldsymbol{\Sigma}_{wx}, \boldsymbol{\Sigma}_{xw}$ and $\boldsymbol{\Sigma}_{xx}$. Firstly, from the definition of $\mathbf{w}$, the covariance $\boldsymbol{\Sigma}_{ww} = \boldsymbol{I}$. Then,

$$\begin{aligned}\boldsymbol{\Sigma}_{wx} &= \mathbb{E}[(\mathbf{w} - \mathbb{E}[\mathbf{w}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T]\\ &= \mathbb{E}[(\mathbf{w})(\boldsymbol{\mu} + \mathbf{D}\mathbf{w} + \boldsymbol{\epsilon} - \boldsymbol{\mu})^T]\\ &= \mathbb{E}[\mathbf{w}\mathbf{w}^T]\mathbf{D}^T + \mathbb{E}[\mathbf{w}\boldsymbol{\epsilon}^T]\\ &= \mathbf{D}^T + \mathbb{E}[\mathbf{w}]\mathbb{E}[\boldsymbol{\epsilon}^T]\\ &= \mathbf{D}^T.\end{aligned} \tag{3.7}$$

where the assumption that $\mathbf{w}$ and $\boldsymbol{\epsilon}$ are independent was used. For $\boldsymbol{\Sigma}_{xw}$, the transpose of $\boldsymbol{\Sigma}_{wx}$ is used. Finally:

$$\begin{aligned}\boldsymbol{\Sigma}_{xx} &= \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T]\\ &= \mathbb{E}[(\boldsymbol{\mu} + \mathbf{D}\mathbf{w} + \boldsymbol{\epsilon} - \boldsymbol{\mu})(\boldsymbol{\mu} + \mathbf{D}\mathbf{w} + \boldsymbol{\epsilon} - \boldsymbol{\mu})^T]\\ &= \mathbb{E}[\mathbf{D}\mathbf{w}\mathbf{w}^T\mathbf{D}^T + \boldsymbol{\epsilon}\mathbf{w}^T\mathbf{D}^T + \mathbf{D}\mathbf{w}\boldsymbol{\epsilon}^T + \boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]\\ &= \mathbf{D}\mathbb{E}[\mathbf{w}\mathbf{w}^T]\mathbf{D}^T + \mathbb{E}[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T]\\ &= \mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi}.\end{aligned} \tag{3.8}$$

All the above components can be summarized in the joint probability distribution of the model by:

$$p(\mathbf{x}, \mathbf{w}) \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{0} \\ \boldsymbol{\mu}\end{bmatrix}, \begin{bmatrix}\boldsymbol{I} & \mathbf{D}^T \\ \mathbf{D} & \mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi}\end{bmatrix}\right). \tag{3.9}$$

The marginal distribution of the model with respect to $\mathbf{x}$ is thus given by:

$$p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi}), \tag{3.10}$$

and the aim is to obtain $\mathbf{D}$ and $\boldsymbol{\Psi}$ given observations $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$. In order to achieve this, the log-likelihood of the model needs to be maximized by fitting the observations. The log-likelihood is given by:

$$\mathcal{L}(\boldsymbol{\mu}, \mathbf{D}, \boldsymbol{\Psi}) = \ln \prod_{i=1}^{N} \frac{1}{(2\pi)^{D/2}|\mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T(\mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi})^{-1}(\mathbf{x}^{(i)} - \boldsymbol{\mu})\right), \tag{3.11}$$

and it is necessary to maximize this with respect to the parameters of the factor analysis model. This can not be achieved in closed form and thus the EM algorithm is used but this time, tailored for this latent variable model (Ghahramani and Hinton, 1997).

The formulas for the conditionals of Gaussian distributions are used:

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2) \tag{3.12}$$

$$\boldsymbol{\Sigma}_{1|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21} \tag{3.13}$$

where $\boldsymbol{\mu}_{1|2}$ is the mean of the conditional distribution of $\mathbf{x}_1$ given $\mathbf{x}_2$. These are two random variable used for illustration of the formulas. $\boldsymbol{\Sigma}_{1|2}$ is the corresponding conditional covariance matrix. Two definitions are necessary for the EM algorithm that use the formulas for the conditional of Gaussians. Firstly:

$$\mathbb{E}[\mathbf{w}|\mathbf{x}^{(i)}] = \boldsymbol{\gamma}\mathbf{x}^{(i)}, \tag{3.14}$$

where $\boldsymbol{\gamma} = \mathbf{D}^T(\mathbf{D}\mathbf{D}^T + \boldsymbol{\Psi})^{-1}$ and $p(\mathbf{w}|\mathbf{x}^{(i)})$ is obtained using the joint distribution defined before ($\boldsymbol{\mu} = 0$ for simplicity and without loss of generality). Another quantity of interest is:

$$\mathbb{E}[\mathbf{w}\mathbf{w}^T|\mathbf{x}^{(i)}] = \boldsymbol{I} - \boldsymbol{\gamma}\mathbf{D} + \boldsymbol{\gamma}\mathbf{x}^{(i)}\mathbf{x}^{(i)^T}\boldsymbol{\gamma}^T. \tag{3.15}$$

where the fact that $\mathrm{Cov}\,(x) = \mathbb{E}[xx^T] - \mathbb{E}[x]\mathbb{E}[x^T]$ was used. These two expectations are used in the EM algorithm for the factor analysis model. The steps of the algorithm can be seen in Algorithm 3. Further information about its derivations and theory can be found in (Ghahramani and Hinton, 1997).

Algorithm 3 EM algorithm for Factor Analysis

Initialization: $\mathbf{\Psi}, \mathbf{D}$ are initialized.

```
1: repeat
2:     E step:
3:     for $i = 1$ to $N$ do
4:         Compute $\mathbb{E}[\mathbf{w}|\mathbf{x}^{(i)}]$ and $\mathbb{E}[\mathbf{w}\mathbf{w}^T|\mathbf{x}^{(i)}]$
5:     M step:
6:         $\mathbf{D}^{\text{new}} = \left(\sum_{i=1}^N \mathbf{x}^{(i)}\mathbb{E}[\mathbf{w}|\mathbf{x}^{(i)}]^T\right)\left(\sum_{l=1}^N \mathbb{E}[\mathbf{w}\mathbf{w}^T|\mathbf{x}^{(l)}]\right)^{-1}$
7:         $\mathbf{\Psi}^{\text{new}} = \frac{1}{N}\text{diag}\left\{\sum_{i=1}^N \mathbf{x}^{(i)}\mathbf{x}^{(i)T} - \mathbf{D}^{\text{new}}\mathbb{E}[\mathbf{w}|\mathbf{x}^{(i)}]\mathbf{x}^{(i)T}\right\}$
8: until convergence;
9: return $(\mathbf{\Psi}, \mathbf{D})$
```

3.2 Principal Component Analysis

As discussed, many data sets have the property that their observations $\mathbf{x}^{(i)}$ lie close to a manifold of a lower dimensional space that contains all or most of the variations in the data. This property can be used for many applications where high dimensional data can be projected onto lower dimensional subspaces. These subspaces are often called the principal components and a framework that provides the solution to this is called the *Principal Component Analysis*. This framework is used for various purposes as a visualization tool of high dimensional data sets, as a compression tool by only storing the important information of the data and as a pre-processing tool that allows faster computation due to the lower dimensionality of the problem. It can also be used in anomaly detection where the algorithm learns the subspace of the data set. If a new data point is used such that when projected onto the subspace does not lie close to the rest of the data set, then an anomaly is flagged.

There are many ways to derive the PCA formulation. In these lecture notes, the maximum variance formulation will be followed. That is, the algorithm seeks an orthogonal projection of the data onto a lower dimensional space such that the *variance of the projected data is maximized*. The degrees of variation in the data are identified which in essence contain the most important information in the data set. Given observations $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\} \in \mathcal{R}^D$ contained in $\mathbf{X} \in \mathcal{R}^{N\times D}$, the goal is to find a matrix $\mathbf{D} \in \mathcal{R}^{D\times K}$ that gives rise to a lower dimensional data set $\mathbf{W} \in \mathcal{R}^{N\times K}$ such that:

$$\mathbf{W} = \mathbf{X}\mathbf{D}, \tag{3.16}$$

without loss of the essential information in the data set. For simplicity, consider the case when $D = 2$ and $K = 1$. Given two-dimensional data points $\mathbf{x}^{(i)}$ with two features $x_1^{(i)}$ and $x_2^{(i)}$, the goal is to find a vector $\mathbf{d}$ that maximizes the variance of the orthogonal projection of $\mathbf{x}^{(i)}$ onto that vector. An illustration of the setup can be seen in Figure 3.1.

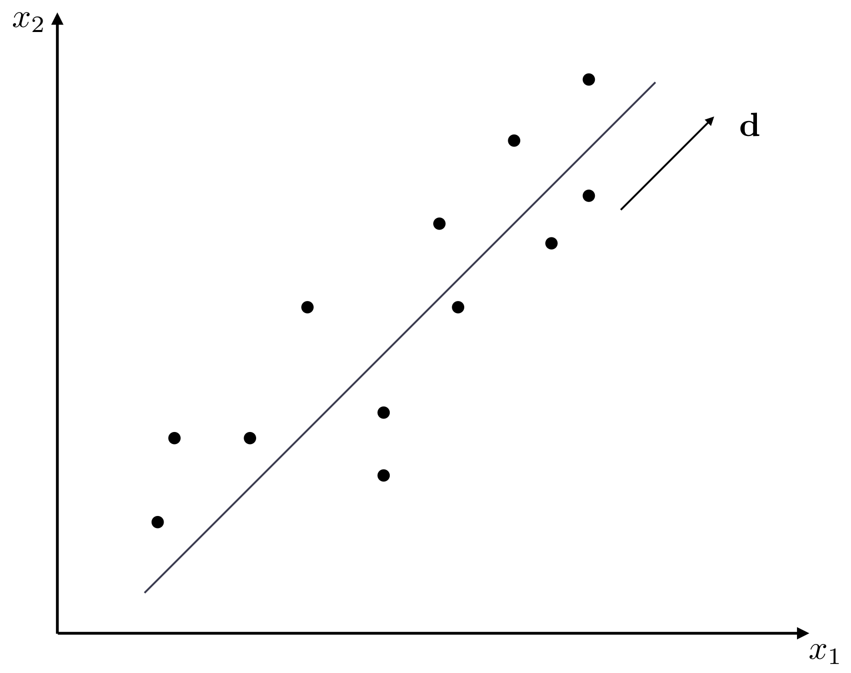


Figure 3.1: 2D data points and a desired projection where their variance is maximized

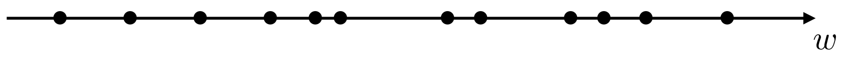


Figure 3.2: 1D data points after projection onto $\mathbf{d}$

The direction of the one-dimensional manifold ($K = 1$) is defined by the D-dimensional vector $\mathbf{d} \in \mathcal{R}^2, D = 2$ and for convenience it is defined as a unit vector ($\mathbf{d}^T\mathbf{d} = 1$). By using this vector, all observations are projected onto a one-dimensional space, a scalar defined by:

$$w^{(i)} = \mathbf{d}^T\mathbf{x}^{(i)} \;\; \forall i. \tag{3.17}$$

An illustration of this can be seen in Figure 3.2. In order to obtain the required $\mathbf{d}$ the projected variance needs to be maximized. Thus, the procedure starts by first calculating the mean of the projected data, $\mathbf{d}^T\boldsymbol{\mu}$ where $\boldsymbol{\mu}$ is the mean of the original data points defined as:

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)}. \tag{3.18}$$

Then, the variance of the projected data is given by:

$$\begin{aligned}\mathrm{Var}[\mathbf{d}^T\mathbf{x}] &= \mathbb{E}[(\mathbf{d}^T\mathbf{x} - \mathbb{E}[\mathbf{d}^T\mathbf{x}])^2] \\ &= \mathbb{E}[(\mathbf{d}^T\mathbf{x} - \mathbf{d}^T\boldsymbol{\mu})^2] \\ &= \frac{1}{N}\sum_{i=1}^{N}(\mathbf{d}^T\mathbf{x}^{(i)} - \mathbf{d}^T\boldsymbol{\mu})^2 \\ &= \mathbf{d}^T\mathbf{C}\mathbf{d}. \end{aligned} \tag{3.19}$$

where $\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T$.

Thus, it is necessary to maximize the variance of the projected data given by (3.19) with respect to $\mathbf{d}$. However, a constraint on its maximization is also present, that $\mathbf{d}$ is a unit length vector. By using a Lagrange multiplier, the following unconstrained optimization problem occurs:

$$\mathbf{d}^T\mathbf{C}\mathbf{d} + \lambda(1 - \mathbf{d}^T\mathbf{d}), \tag{3.20}$$

where λ is a Lagrange multiplier. Differentiating with respect to $\mathbf{d}$, setting to zero and re-arranging:

$$\mathbf{C}\mathbf{d} = \lambda\mathbf{d}. \tag{3.21}$$

This result shows that the maximum variance of the projected data is attained when $\mathbf{d}$ is an eigenvector of the data covariance matrix. By left-multiplying by $\mathbf{d}^T$:

$$\mathbf{d}^T\mathbf{C}\mathbf{d} = \lambda, \tag{3.22}$$

which shows that the maximum variance is obtained with the largest eigenvalue and eigenvector of the data covariance matrix. The eigenvector $\mathbf{d}$ is the first principal component and in this case, it is the only principal component. For a K-dimensional manifold and a D-dimensional input space, the optimal projection are the K eigenvectors of the data covariance matrix that correspond to the K largest eigenvalues.

The eigenvectors give us a new basis to represent our data and it is guaranteed that there will be eigenvectors since the covariance matrix is symmetric. There are many ways one can follow to obtain the eigenvectors of the data covariance matrix. A practical and convenient method is to use the *Singular Value Decomposition*. Suppose that $\mathbf{X} \in \mathcal{R}^{N \times D}$, there exists a factorization of that matrix such that:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.23}$$

where $\mathbf{U} \in \mathcal{R}^{N \times N}$ whose columns are orthonormal, $\mathbf{\Sigma} \in \mathcal{R}^{N \times D}$ and $\mathbf{V}^T \in \mathcal{R}^{D \times D}$ whose rows and columns are orthonormal. The diagonal entries of $\mathbf{\Sigma}$ are the singular values of $\mathbf{X}$ and a connection to the eigenvalues is shown where:

$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}(\mathbf{\Sigma}^T\mathbf{\Sigma})\mathbf{V}^T. \tag{3.24}$$

Right-multiply by $\mathbf{V}$:

$$(\mathbf{X}^T\mathbf{X})\mathbf{V} = \mathbf{V}(\mathbf{\Sigma}^T\mathbf{\Sigma}), \tag{3.25}$$

which shows that the eigenvectors of $\mathbf{X}^T\mathbf{X}$ are equal to $\mathbf{V}$ and the eigenvalues are equal to the squared singular values of $\mathbf{X}$. Thus, by estimating the SVD of $\mathbf{X}$, it is possible to obtain the necessary eigenvectors and eigenvalues of $\mathbf{C}$ and subsequently obtain the principal components of the data set $\mathbf{X}$ (the data set is usually centered prior to PCA). By using more eigenvectors, the representation of the data is more accurate. Thus, there is a trade-off between compression, computational speed and accuracy which needs to be measured given the application.

3.3 Independent Component Analysis

So far the models assumed that the data live in a subspace where all the variations are captured. In this section, a different viewpoint of the latent variable models will be discussed which assumes that the data were generated by different sources and the observations are assumed to be their linear combinations without knowing *a priori* what each source contributes. Consider the *cocktail party problem* where you are in a room full of people and everyone is talking. You are listening to a combination of all these conversations and your goal is to understand what everyone is saying. Thus, from a linear combination of conversations, it is necessary to separate their sources without knowing beforehand what each person was saying. This is also called *blind source separation.* There are many applications of this model other than the acoustic signal processing industry. EEG and MEG signals contain mixed signals that require separations, financial data are highly convoluted and in general data from the real world are highly complex and likely to be generated by many sources.

Let $\mathbf{x}^{(i)} \in \mathcal{R}^D$ be the observed signal at a receiver at time $t = i$. The observed signal is temporal, that is, it varies with time. However, in this model, each time instance is considered independent of the next and thus, each sample $\mathbf{x}^{(i)}$ is modeled separately (temporal correlation exists but it is not modeled in this context). Furthermore, let $\mathbf{w}^{(i)} \in \mathcal{R}^K$ be the vector of source signals at time i. As discussed, the observed signal is a linear combination of the source signals given by:

$$\mathbf{x}^{(i)} = \mathbf{D}\mathbf{w}^{(i)} + \boldsymbol{\epsilon}^{(i)}, \tag{3.26}$$

where $\mathbf{D} \in \mathcal{R}^{D \times K}$ is called the mixing matrix and $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Psi})$. The goal is to infer the source signals $p(\mathbf{w}^{(i)}|\mathbf{x}^{(i)}, \boldsymbol{\theta})$ from the observed signals with $\boldsymbol{\theta}$ the parameters that describe the sources. The above model is very similar to factor analysis as it models the latent variables. However, in this context, a different prior is used for $p(\mathbf{w}^{(i)})$. This prior is any non-Gaussian distribution given by:

$$p(\mathbf{w}^{(i)}) = \prod_{j=1}^{K} p_j(w_j^{(i)}), \tag{3.27}$$

where j corresponds to the j-th source signal. This model is called the *Independent Component Analysis* (Hyvrinen and Oja, 2000) and the goal is to estimate the parameters $\mathbf{w}$ and the mixing matrix $\mathbf{D}$. For simplicity, assume that $\mathbf{D}$ is square and that there is no noise in the model. The observations are centered (zero-mean) and whitened. This means that $\mathbb{E}[\mathbf{x}\mathbf{x}^T] = \boldsymbol{I}$. The covariance of $\mathbf{x}$ is given by:

$$\begin{aligned} cov[\mathbf{x}] &= \mathbb{E}[\mathbf{x}\mathbf{x}^T] \\ &= \mathbb{E}[\mathbf{D}\mathbf{w}\mathbf{w}^T\mathbf{D}^T] \\ &= \mathbf{D}\mathbb{E}[\mathbf{w}\mathbf{w}^T]\mathbf{D}^T \\ &= \mathbf{D}\mathbf{D}^T = \boldsymbol{I}, \end{aligned} \tag{3.28}$$

which means that $\mathbf{D}$ must be orthogonal. A classic rule for transforming random variables and their densities (Hyvrinen and Oja, 2000) states that for any random vector $\mathbf{x}$ with density $p_x(\mathbf{x})$ and for any matrix $\mathbf{D}$, the density of $\mathbf{y} = \mathbf{D}\mathbf{x}$ is given by:

$$\begin{aligned} p_y(\mathbf{y}) &= p_x(\mathbf{x})|\det\left(\frac{d\mathbf{x}}{d\mathbf{y}}\right)| \\ &= p_x(\mathbf{x})|\det\left(\mathbf{D}^{-1}\right)| \end{aligned} \tag{3.29}$$

This is used to obtain:

$$\begin{aligned} p_x(\mathbf{x}) &= p_x(\mathbf{D}\mathbf{w}^{(i)}) \\ &= p_w(\mathbf{w}^{(i)})|\det(\mathbf{D}^{-1})| \\ &= p_w(\mathbf{V}\mathbf{x}^{(i)})|\det(\mathbf{V})|, \end{aligned} \tag{3.30}$$

where $\mathbf{V} = \mathbf{D}^{-1}$ and recall that $\mathbf{w} = \mathbf{D}^{-1}\mathbf{x}$. Assuming N iid samples of $\mathbf{x}^{(i)}$, the log-likelihood is given by:

$$\begin{aligned} \mathcal{L} &= \sum_{i=1}^{N} \log\ p_x(\mathbf{D}\mathbf{w}^{(i)}) \\ &= \sum_{i=1}^{N} \log\ p_w(\mathbf{V}\mathbf{x}^{(i)})|\det(\mathbf{V})| \\ &= N\ \log\ |\det(\mathbf{V})| + \sum_{i=1}^{N}\sum_{j=1}^{K} \log\ p_j(\mathbf{v}_j^T\mathbf{x}^{(i)}), \end{aligned} \tag{3.31}$$

where $\mathbf{v}_j$ is the j-th row of $\mathbf{V}$. The orthogonality condition on $\mathbf{V}$ simplifies the log-likelihood in (3.31) due to the fact that the determinant of an orthogonal matrix is always a constant.

The determinant part then is removed due to the fact that the derivative of a constant is zero. Thus, the log-likelihood simplifies to:

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{j=1}^{K} \log\ p_j(\mathbf{v}_j^T \mathbf{x}^{(i)}), \tag{3.32}$$

Once the above is formulated, a gradient-ascent algorithm is easily derived to obtain the solution to the optimization problem at hand. There are also many alternatives to gradient ascent which can be found in (Hyvrinen and Oja, 2000).

A question arises as to what is an appropriate source signal prior. Non-Gaussian priors are a requirement for the algorithm to work. Consider the case where the source variables $\mathbf{w}$ are uniformly distributed as can be seen in Figure 3.3. Their linear combination (ie the observations) would have the effect of moving into the space given a linear transformation as can be seen in Figure 3.4. Now, the observations are whitened (transformed such that they have unit covariance) and the goal is to find a matrix/rotation in the space to come back to the sources. However, if the prior on the sources is Gaussian, then this implies that the observations are also Gaussian (central limit theorem). Since the observations are centered and whitened, their distribution is symmetric as can be seen in Figure 3.5. Thus, it is not possible to find a rotation/transformation that produces the sources and this illustrates that non-Gaussianity is essential.

There are many non-gaussians to choose from. Firstly, *super-Gaussian distributions* could be used that have a large spike at the mean with heavy tails. These *leptokurtic* distributions are super-Gaussians if the kurtosis of the distribution is greater than 0. The kurtosis is defined by:

$$\text{kurt}(\text{w}) = \frac{\mu_4}{\sigma^4} - 3, \tag{3.33}$$

where σ is the standard deviation and $\mu_k = \mathbb{E}[(w - \mathbb{E}[w])^k]$. An example is the Laplacian distribution.

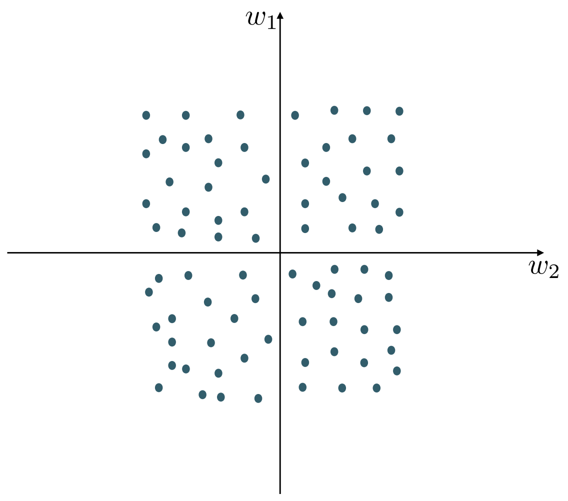


Figure 3.3: Uniformly distributed source variables

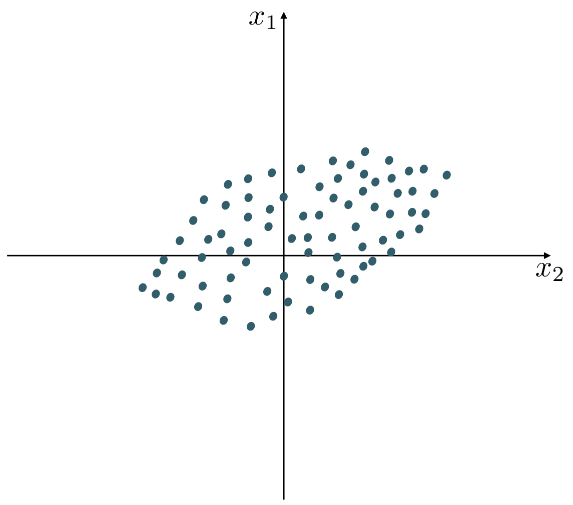


Figure 3.4: The observations are a linear transformation of the sources.

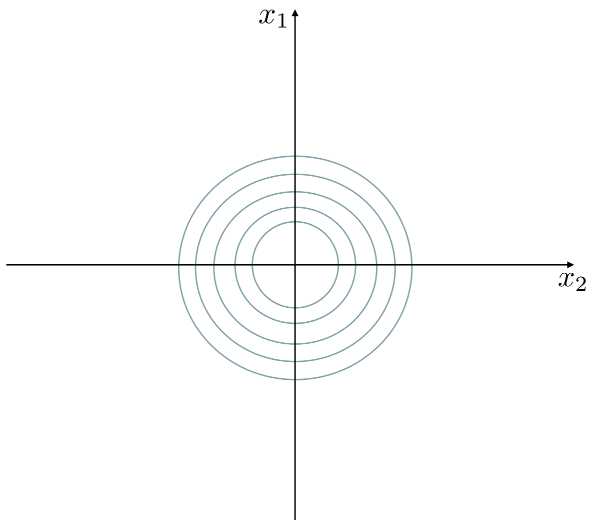


Figure 3.5: Symmetric Gaussian distribution that does not provide any information about the transformation matrix

Second type of distribution suitable for a prior are the *sub-Gaussian distributions* which have negative kurtosis (ie uniform distribution). Lastly, *skewed distributions* allow the existence of asymmetry that makes them non-Gaussian and can be used for this purpose. An example of this is the gamma distribution.

CHAPTER 4

Representation and Feature Learning

The standard pipeline for the majority of machine learning algorithms is to transform the input into a particular feature space where the data have certain structure. Traditionally, this feature space was hand-engineered by experts in the particular field of research such as the HoG (Dalal and Triggs, 2005) for human detection discussed in chapter 1. The procedure of hand-engineering the feature space is very tedious and the features learned might not be optimal or reusable. Unsupervised Feature Learning is an emerging framework which allows the learning of features from training data, suited for the application at hand. There are many algorithms that are used to learn features from data and different models of obtaining solutions. Both deterministic and probabilistic models will be discussed along with parametric and non-parametric models of feature learning.

4.1 Multi-Layer Neural Networks

Multi-layer neural networks are an extension of the logistic regression and plain one-layer neural networks. An example of a multi-layer neural network can be seen in Figure 4.1 with two hidden layers. In the example of human detection in images, the input units could be image pixels and the output units could be the classes of objects (human, tree, cat etc). These networks are essentially a composition of non-linear functions and the goal is to train the model for a particular task through supervision (ie presenting labels given images and then adjust the parameters). The training procedure involves the minimization of a cost function which compares the output of the network with the provided label/value. A simple cost function is defined by:

$$J(\mathbf{a}^{(3)}, \mathbf{y}) = \frac{1}{2} \sum_{l} (a_l^{(3)} - y_l)^2, \tag{4.1}$$

where $\mathbf{a}^{(3)}$ is the output of the network (in the case of the model in Figure 4.1) and $\mathbf{y}$ are the true values that the model is trained on.

At each layer, the total input $\mathbf{w}$ is calculated as a linear combination of the outputs of the units in the layer below where N_0, N_1 and N_2 are the number of units in the respective layer in Figure 4.1. The weights in the linear combination are defined as $\mathbf{d}$ and essentially are connections between layers. They are adjustable and vary depending on the goals of the network. These weights/connections constitute the learned features/basis functions. They usually vary from layer to layer but they can also be constrained to be the same (ie in auto-encoders). Once the total input $\mathbf{w}$ to a layer is calculated, it is passed through a non-linear function $f(\mathbf{w})$ to get the output of that particular layer. The output is defined as $\mathbf{a}$ and is called the activation. Typical non-linear functions $f(\mathbf{w})$ are the rectified linear unit, the hyperbolic tangent and the logistic function.

To train multi-layer neural networks, two stages are required. First, a forward pass through the network is done that calculates all the activations at each layer as seen in Figure 4.1. Then, a backwards pass is done as can be seen in Figure 4.2. At each hidden layer, the error derivative with respect to the output of each unit is calculated. This is a linear combination of the error derivatives with respect to the total inputs to the units in the layers above. To obtain the error derivative with respect to the inputs of the layer, the chain rule is used where essentially the error derivative with respect to the output is multiplied by the gradient of $f(\mathbf{w})$. This is done for all layers and for all units in each layer. The procedure is called *backpropagation* because the error of the network from the output layer is propagated back through the network. At the output layer, the error derivative with respect to the output is the derivative of the cost function $J(\mathbf{a}, \mathbf{y})$ with respect to the activation. This results in a subtraction between the prediction of the network with the true labeled value (in the case of the least-squares cost function in Figure 4.2). To obtain the derivatives with respect to the basis function/features $\mathbf{d}$, the chain rule is used once more. That is:

$$\frac{\partial J}{\partial d_{jk}^{(1)}} = \frac{\partial J}{\partial w_k^{(2)}} \frac{\partial w_k^{(2)}}{\partial d_{jk}^{(1)}} \tag{4.2}$$

Once all the gradients are obtained, the minimization of $J(\mathbf{a}^{(3)}, \mathbf{y})$ is usually done via stochastic gradient descent. That is, the parameters are updated every time using only one input towards the direction of the gradient of the cost function given by:

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \mathbf{y}^{(i)}), \tag{4.3}$$

where $\boldsymbol{\theta}$ are all the adjustable parameters of the model to be learned and α is the learning rate to be chosen a priori. The stochastic gradient descent could be altered to use batches of data summed together before moving onto the direction of the gradient.

For the task of *feature learning*, the target $\mathbf{y}^{(i)}$ of the network is the input itself. That is, given training data $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$, the goal of the network is to reconstruct the input as accurately as possible. Thus, the training data are only the inputs and thus the target is $\mathbf{y}^{(i)} = \mathbf{x}^{(i)}$ for each data point in the training set. This is the key idea of unsupervised feature learning and different levels of representations can be learned that capture different levels of variations in the data. In this chapter, different models of representation learning will be discussed with one layer. These can then be generalized to multiple layers by stacking them together and training them layer-wise.

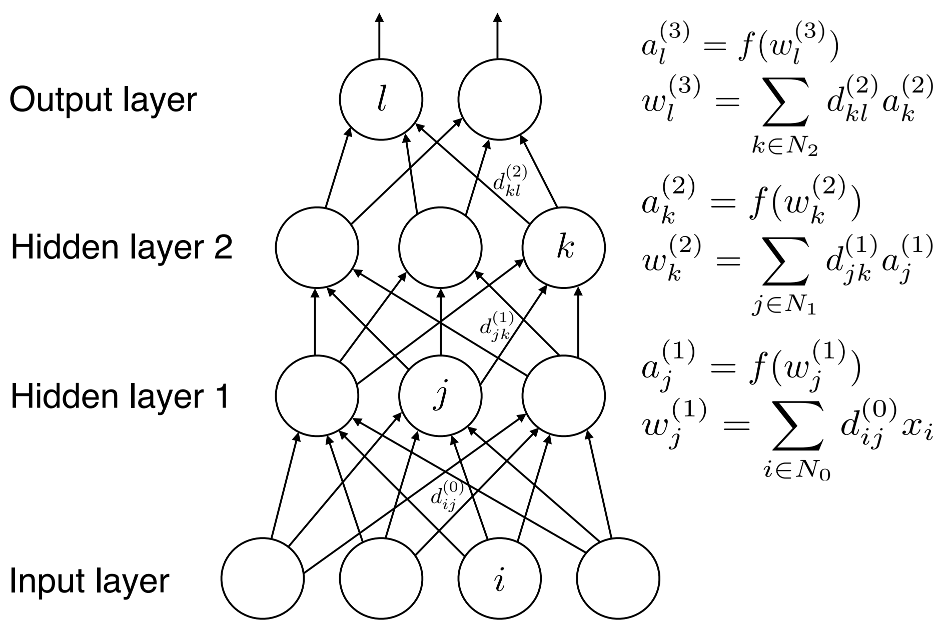

Output layer
Hidden layer 2
Hidden layer 1
Input layer
l
k
j
i
$d_{kl}^{(2)}$
$d_{jk}^{(1)}$
$d_{ij}^{(0)}$
$a_l^{(3)} = f(w_l^{(3)})$
$w_l^{(3)} = \sum_{k \in N_2} d_{kl}^{(2)} a_k^{(2)}$
$a_k^{(2)} = f(w_k^{(2)})$
$w_k^{(2)} = \sum_{j \in N_1} d_{jk}^{(1)} a_j^{(1)}$
$a_j^{(1)} = f(w_j^{(1)})$
$w_j^{(1)} = \sum_{i \in N_0} d_{ij}^{(0)} x_i$

Figure 4.1: Forward pass in a Multi-Layer Neural Network

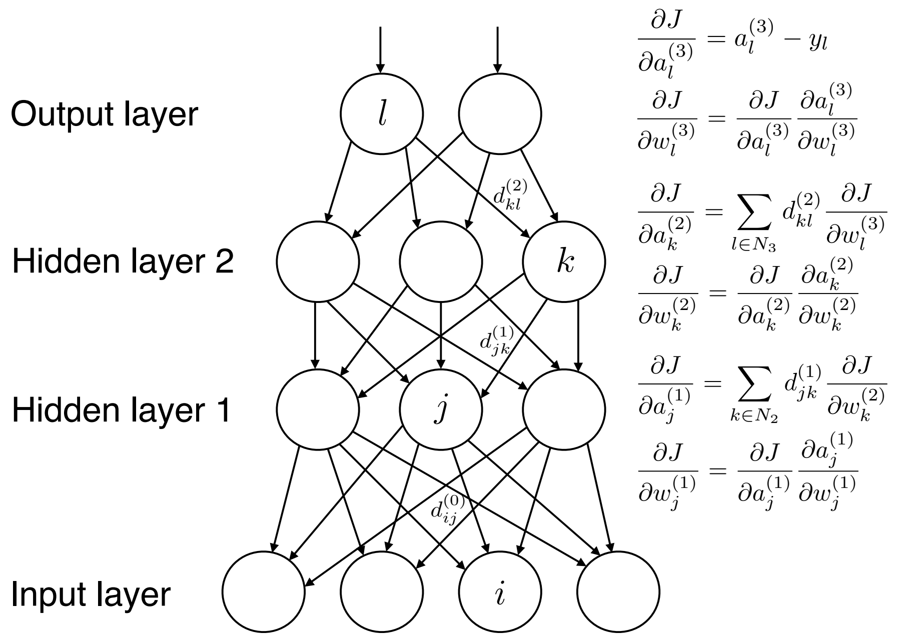

Output layer
Hidden layer 2
Hidden layer 1
Input layer
l
k
j
i
$d_{kl}^{(2)}$
$d_{jk}^{(1)}$
$d_{ij}^{(0)}$
$\frac{\partial J}{\partial a_l^{(3)}} = a_l^{(3)} - y_l$
$\frac{\partial J}{\partial w_l^{(3)}} = \frac{\partial J}{\partial a_l^{(3)}} \frac{\partial a_l^{(3)}}{\partial w_l^{(3)}}$
$\frac{\partial J}{\partial a_k^{(2)}} = \sum_{l \in N_3} d_{kl}^{(2)} \frac{\partial J}{\partial w_l^{(3)}}$
$\frac{\partial J}{\partial w_k^{(2)}} = \frac{\partial J}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial w_k^{(2)}}$
$\frac{\partial J}{\partial a_j^{(1)}} = \sum_{k \in N_2} d_{jk}^{(1)} \frac{\partial J}{\partial w_k^{(2)}}$
$\frac{\partial J}{\partial w_j^{(1)}} = \frac{\partial J}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_j^{(1)}}$

Figure 4.2: Backpropagation through the network to obtain desired derivatives

4.2 Auto-encoders

In this framework, an encoder is explicitly defined as a parameterized function f_θ which will provide the code/activation $\mathbf{a} \in \mathcal{R}^K$ from input $\mathbf{x} \in \mathcal{R}^D$. For each example in the training set $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$, the encoder is given by $\mathbf{a}^{(i)} = f_\theta(\mathbf{x}^{(i)})$. On the other hand, a decoder is explicitly defined as a parameterized function g_θ that maps the feature space back to the input space, thus producing the reconstruction signal $\mathbf{r} = g_\theta(\mathbf{a})$. Note that the encoder and decoder take the input data as input to their function, as opposed to the functions in the multi-layer neural network which use the linear combination of the inputs directly ($\mathbf{w}^{(l)}$) for each layer. The behavior is the same since the linear combination is now performed as an argument to the encoder/decoder function for notation purposes only. The most common encoder and decoder functions for binary data are of the following form (Bengio et al., 2013):

$$f_\theta(\mathbf{x}) = s_f(\mathbf{b} + \mathbf{D}\mathbf{x}), \tag{4.4}$$

$$g_\theta(\mathbf{a}) = s_g(\mathbf{c} + \mathbf{D}^T\mathbf{a}), \tag{4.5}$$

where $\mathbf{b}$ and $\mathbf{c}$ are the biases of the encoder and decoder respectively, $\mathbf{D}^T \in \mathcal{R}^{D\times K}$ are the weights connecting the respective spaces and s_f and s_g are typically element-wise sigmoid functions:

$$s(t) = \frac{1}{1+e^{-t}}. \tag{4.6}$$

Other non-linear functions could also be used (hyperbolic tangents, rectified functions etc). For real-valued data, $g_\theta(\mathbf{a})$ is usually only the linear combination of the weights with the connections. This is because, the data do not require any bound as opposed to binary data that need to be between 0 and 1. Non-linearities, in the case of real data, are not required and thus the linear combination is the only step in the decoder.

The set of parameters to be estimated are collectively $\boldsymbol{\theta} = \{\mathbf{b}, \mathbf{c}, \mathbf{D}\}$. Training an autoencoder involves the learning of $\boldsymbol{\theta}$ from which $\mathbf{D}^T$ provide the learned basis functions. Training of the encoder and decoder is done simultaneously on the task of reconstructing the original input (Figure 4.3). This is done by minimizing a cost function:

$$J_{AE}(\boldsymbol{\theta}) = \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, g_\theta(f_\theta(\mathbf{x}^{(i)}))), \tag{4.7}$$

where $L(.,.)$ could be the least-squares fit. The derivatives of the network are obtained via back-propagation. As discussed in the case of multi-layer neural networks, a first pass through the network is performed in order to obtain the values of the activations and the different variables of the model. Then, a backwards pass is performed to obtain the derivatives and compared with each input. Note that in the auto-encoder, the weights of the encoder and the decoder are tied (ie one is the transpose of the other). Once all the derivatives are obtained, a variant of the stochastic gradient descent can be used. However, regularization is necessary to avoid learning the identity function. This is achieved with the following regularized auto-encoders.

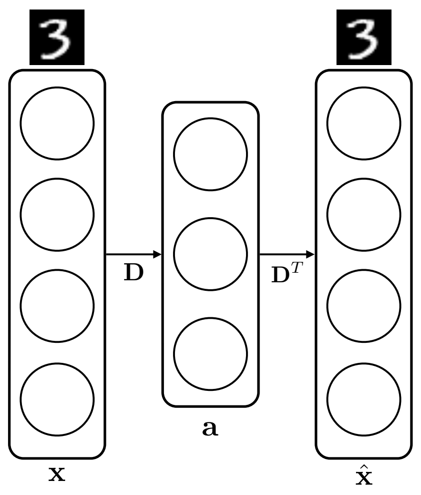


Figure 4.3: Training of an auto-encoder

Denoising Auto-encoders

The Denoising Autoencoder (DA) was proposed (Vincent et al., 2008), where the cost function is altered so that the optimizer minimizes the distance between a corrupted version of the input space and the reconstruction. This is defined as:

$$J_{DAE}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathbb{E}_{q(\mathbf{x}_c|\mathbf{x}^{(i)})}\Big[L(\mathbf{x}_c^{(i)}, g_\theta(f_\theta(\mathbf{x}_c^{(i)})))\Big], \tag{4.8}$$

where $\mathbb{E}_{q(\mathbf{x}_c|\mathbf{x}^{(i)})}[.]$ averages over corrupted examples drawn from the corruption process $q(\mathbf{x}_c|\mathbf{x}^{(i)})$. This corruption process varies but for simplicity consider this as a process that randomly sets a pre-defined proportion of the input to 0. This cost function is again optimized by using stochastic gradient descent and some interesting features are obtained as it can be seen in Figure 4.4a (30% of noise in input).

Contractive Auto-encoders

The Contractive Auto-encoder (CA) was proposed (Rifai et al., 2011) by adding an analytic contractive penalty to the cost function. This is the Frobenius norm of the encoder's Jacobian which translates in penalizing the sensitivity of learned features to variations other than the inputs (Bengio et al., 2013). The cost function becomes:

$$J_{CAE}(\boldsymbol{\theta}) = \sum_{i=1}^{N} L(\mathbf{x}^{(i)}, g_\theta(f_\theta(\mathbf{x}^{(i)}))) + \lambda \|J_f(\mathbf{x}^{(i)})\|_F^2, \tag{4.9}$$

where $J_f(\mathbf{x}^{(i)})$ is the Jacobian of the encoder at a training example, $\|.\|_F$ is the Frobenius norm and λ is a hyperparameter that controls the regularization. The cost function is optimized via the same route obtaining features as shown in Figure 4.4b.

4.3 Restricted Boltzmann Machines

Probabilistic models are defined by a probability function over configurations of inputs and hidden units. Then, they are trained to maximize the likelihood of the model given the training data. Undirected graphical models parametrize the joint probability distribution $p(\mathbf{x}, \mathbf{a})$ through a product of unnormalized non-negative clique potentials (Bengio et al., 2013). A particular form is the Boltzmann distribution with clique potentials constrained to be positive and defined by:

$$p(\mathbf{x}, \mathbf{a}) = \frac{1}{Z_\theta} exp(-\mathcal{E}_\theta(\mathbf{x}, \mathbf{a})), \tag{4.10}$$

where $\mathcal{E}_\theta(\mathbf{x}, \mathbf{a})$ is the energy function that contains the interactions between the clique potentials and Z_θ is the partition function. For a Boltzmann Machine, the energy function is defined by (Bengio et al., 2013):

$$\mathcal{E}_\theta(\mathbf{x}, \mathbf{a}) = -\frac{1}{2}\mathbf{x}^T\mathbf{U}\mathbf{x} - \frac{1}{2}\mathbf{a}^T\mathbf{V}\mathbf{a} - \mathbf{x}^T\mathbf{D}\mathbf{a} - \mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{a}, \tag{4.11}$$

where the set of parameters to be estimated are $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{V}, \mathbf{D}, \mathbf{b}, \mathbf{c}\}$. Each matrix and vector corresponds to the connections between the variables being multiplied with each other. This type of graph results in an intractable problem of inference (Bengio et al., 2013). Therefore, a restricted version of the model was proposed which is tractable.

Restricted Boltzmann Machines (RBMs) restrict the connections in the Boltzmann energy function defined above by not allowing connections between $\mathbf{x}$ and itself and between $\mathbf{a}$ and itself, ie $\mathbf{U} = \mathbf{0}$ and $\mathbf{V} = \mathbf{0}$. With this restriction, the RBM is tractable. In order to learn $\boldsymbol{\theta}$, maximization of the likelihood of the training data is required. Equivalently, the maximization of the log-likelihood giving a cost function:

$$J_{RBM}(\boldsymbol{\theta}) = \sum_{i=1}^{N} log \sum_{\mathbf{a}} P(\mathbf{x}^{(i)}, \mathbf{a}; \boldsymbol{\theta}). \tag{4.12}$$

The gradient of the cost function is given by (Bengio et al., 2013):

$$\frac{\partial J_{RBM}}{\partial \theta_j} = -\sum_{i=1}^{N} \mathbb{E}_{p(\mathbf{a}|\mathbf{x}^{(i)})}\left[\frac{\partial \mathcal{E}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}, \mathbf{a})}{\partial \theta_j}\right] + \sum_{i=1}^{N} \mathbb{E}_{p(\mathbf{a},\mathbf{x})}\left[\frac{\partial \mathcal{E}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{a})}{\partial \theta_j}\right]. \tag{4.13}$$

The second expression in the above formula is intractable and thus sampling is required to approximate it with a finite sum given by:

$$\mathbb{E}_{p(\mathbf{a},\mathbf{x})}\left[\frac{\partial \mathcal{E}_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{a})}{\partial \theta_j}\right] = \frac{1}{L}\sum_{l=1}^{L} \frac{\partial \mathcal{E}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}^{(l)}, \tilde{\mathbf{a}}^{(l)})}{\partial \theta_j} \tag{4.14}$$

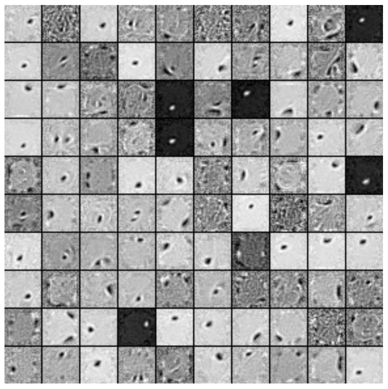
The samples are obtained from:

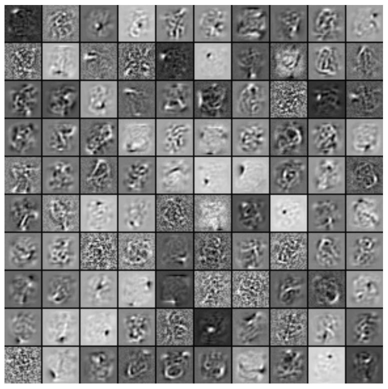
$$\tilde{\mathbf{x}}^{(l)} \sim p(\mathbf{x}|\tilde{\mathbf{a}}^{(l-1)}) \tag{4.15}$$

$$\tilde{\mathbf{a}}^{(l)} \sim p(\mathbf{a}|\tilde{\mathbf{x}}^{(l)}) \tag{4.16}$$

using Gibbs sampling (Hinton, 2002) which will be discussed in chapter 5 in detail. Once an expectation of the gradients is obtained, it can be used in similar fashion as before but in this case using stochastic steepest ascent to find a local optimum for the parameters. Examples of learned basis functions by an RBM can be seen in Figure 4.4c.

In order to illustrate the functionality of the above models, their application on the MNIST dataset[1] is illustrated. This dataset is composed of 28×28 images of 10 classes of hand-written digits. The goal is to train a model that is able to automatically classify unseen images. Thus, one might expect the features learned to be for example edge detectors of particular regions that are unique for each class. The models require large amounts of training data to learn the lower dimensional manifold where the feature space lies. Thus, 60000 training samples from the dataset were used. Variations within the models themselves exist by changing their numerous hyperparameters. A hyperparameter study is required or other more efficient techniques (Snoek et al., 2012) could be used. For the purposes of these lecture notes, three different models are illustrated with 500 hidden units and Figures 4.4a - 4.4c illustrate the basis functions learned by each model using Theano (Bastien et al., 2012). The learned basis functions are very similar and resemble edge detectors in particular regions.


(a) 30% DA basis functions


(b) CA basis functions

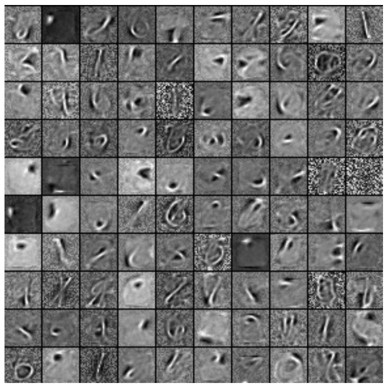

(c) RBM basis functions

Figure 4.4: Fraction of learned basis functions for MNIST

[1]http://yann.lecun.com/exdb/mnist/

4.4 Sparse Coding and the K-SVD

A more challenging feature space to explore is that of natural images and thus we turn our attention to methods that have been introduced with that in mind. Given a set of training signals, the aim is to find the dictionary that leads to the best representation for each member in this set under sparsity constraints. K-SVD (Aharon et al., 2006) is a generalization of the K-means clustering algorithm that was discussed in chapter 2 and alternates between sparse coding and a process of updating the dictionary columns using the SVD discussed in chapter 3. That is, the aim is to find the best dictionary to compose the training data (notation in paper) $\mathbf{X} \in \mathcal{R}^{D\times N} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ in sparse representations:

$$\min_{\mathbf{D},\mathbf{W}} \|\mathbf{X} - \mathbf{D}\mathbf{W}\|_2^2 \quad \text{subject to} \quad \forall i \ \ \|\mathbf{w}^{(i)}\|_0 \leq T_0. \tag{4.17}$$

where $\mathbf{D} \in \mathcal{R}^{D\times K}$ contains the learned basis functions, $\mathbf{W} \in \mathcal{R}^{K\times N}$ contains the coefficients and T_0 is the sparsity of the signal. In order to solve this, the algorithm alternates between $\mathbf{D}$ and $\mathbf{W}$. It initializes $\mathbf{D}$ randomly and uses a pursuit algorithm to compute the representation vectors (coefficients) $\mathbf{w}^{(i)}$ for each training example $\mathbf{x}^{(i)}$ by solving:

$$\min_{\mathbf{w}^{(i)}} \|\mathbf{x}^{(i)} - \mathbf{D}\mathbf{w}^{(i)}\|_2^2 \quad \text{subject to} \quad \|\mathbf{w}^{(i)}\|_0 \leq T_0 \ \forall i. \tag{4.18}$$

Once this is solved, the dictionary columns are updated by computing an overall representation error matrix. An SVD decomposition is performed on that matrix and the updated column is obtained from the singular vectors. This is repeated until a sufficient condition for convergence is reached. An example of learned dictionary basis functions can be seen in Figure 4.5 where each square is a column of $\mathbf{D}$ re-arranged into an 8×8 gray-scale image. If we compare these with the Haar wavelets (Figure 4.6) which are a standard transform for image compression, it confirms that edges in images capture the most significant variations in the signal.

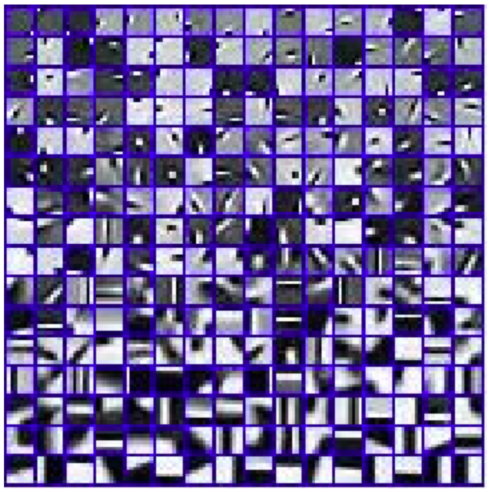


Figure 4.5: Learned basis functions by K-SVD

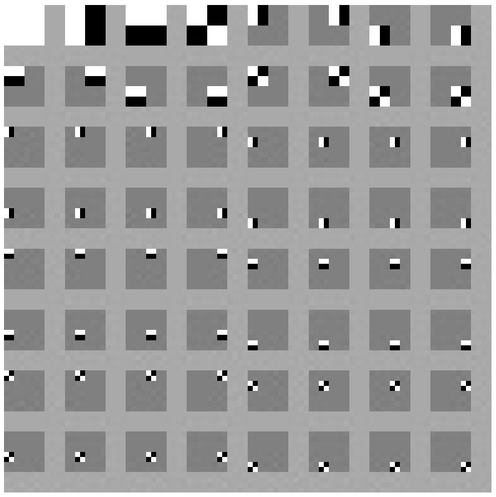


Figure 4.6: Haar wavelet basis functions

4.5 Beta Process Factor Analysis and the Indian Buffet Process

Dictionary learning can be cast as a factor analysis problem, with factors corresponding to the coefficients and the matrix that connects them corresponding to the dictionary of basis functions. Utilizing non-parametric Bayesian methods from the stochastic processes literature, it is possible to learn dictionaries of basis functions and at the same time infer the exact number of dictionary elements. A statistical model is considered that uses Bayesian inference. The model is defined by:

$$\mathbf{x}^{(i)} = \mathbf{D}\mathbf{w}^{(i)} + \boldsymbol{\epsilon}^{(i)}, \tag{4.19}$$

where $\mathbf{x}^{(i)} \in \mathcal{R}^D$ is an image patch (ie 8×8), $\mathbf{D} \in \mathcal{R}^{D\times K}$ is the dictionary of basis functions, $\mathbf{w}^{(i)} \in \mathcal{R}^K$ are the respective sparse coefficients and $\boldsymbol{\epsilon}^{(i)}$ is the assumed noise in the measurements. The goal is to infer simultaneously $\mathbf{D}$ and $\{\mathbf{w}^{(i)}\}_{i=1}^N$ from image patches $\{\mathbf{x}^{(i)}\}_{i=1}^N$ which might be obtained either from one single image or many. Bayesian priors are imposed on $\mathbf{D}$, $\mathbf{w}^{(i)}$ and $\boldsymbol{\epsilon}^{(i)}$ to capture the prior belief of the data at hand. Firstly, the coefficients $\mathbf{w}^{(i)}$ are restricted to be sparse and at the same time, only a very small proportion of columns from $\mathbf{D}$ are selected. Let binary vector $\mathbf{z}^{(i)} \in \{0,1\}^K$ indicate which columns/basis functions are used to represent $\mathbf{x}^{(i)}$ where 1 indicates selection. For $\{\mathbf{x}^{(i)}\}_{i=1}^N$ there is a respective set of binary vectors $\{\mathbf{z}^{(i)}\}_{i=1}^N$. In order to model these binary vectors, a beta-Bernoulli process prior is convenient:

$$\mathbf{z}^{(i)} \sim \prod_{k=1}^{K} \text{Bernoulli}(\pi_k), \tag{4.20}$$

where π_k is the k-th element of $\boldsymbol{\pi}$ defined by:

$$\boldsymbol{\pi} \sim \prod_{k=1}^{K} \text{Beta}(\alpha/K, b(K-1)/K). \tag{4.21}$$

The hierarchical form of the Bayesian model can be written as (Zhou et al., 2012):

$$\mathbf{x}^{(i)} = \mathbf{D}\mathbf{w}^{(i)} + \boldsymbol{\epsilon}^{(i)},$$

$$\mathbf{w}^{(i)} = \mathbf{z}^{(i)} \odot \mathbf{s}^{(i)},$$

where $\mathbf{s}^{(i)}$ is drawn from a Gaussian distribution with a gamma hyper-prior. The above construction along with the beta-Bernoulli Process (BP) is referred to as the BP Factor Analysis (BPFA) model (Zhou et al., 2012). In order to infer the dictionary $\mathbf{D}$, the negative logarithm of the posterior density function is optimized using Gibbs sampling discussed in chapter 5 in detail. Further prior knowledge could be placed in the model, imposing a greater degree of statistical structure. In this context, image patches could be clustered into different categories of texture and colour. Then, image patches of the same cluster must utilize the same set of basis functions. In order to impose such constraint, Dirichlet Processes (Zhou et al., 2012) could be used to incorporate the knowledge of clustering.

Indian Buffet Process

In a latent feature model, each data point $\mathbf{x}^{(i)} \in \mathcal{R}^D$ is represented by a number of latent features ${\mathbf{w}^{(\mathbf{i})}}^T \in \mathcal{R}^K$, contained in a matrix $\mathbf{W} \in \mathcal{R}^{N \times K}$ and each row contains the K feature values for each data point $\mathbf{x}^{(i)}$. This kind of model is specified by a prior on the features $p(\mathbf{W})$ and a likelihood model $p(\mathbf{X}|\mathbf{W})$ to obtain the configurations of the observations through the joint distribution. The focus is to specify the right prior $p(\mathbf{W})$ in order to incorporate the number of features, their values and their probabilities. Matrix $\mathbf{W}$ can be split into $\mathbf{W} = \mathbf{Z} \circ \mathbf{S}$ where $\mathbf{Z}$ is a sparse binary matrix that indicates which features are used by each data point and $\mathbf{S}$ contains the feature values.

By defining priors on $p(\mathbf{S})$ and $p(\mathbf{Z})$, the overall prior is obtained by $p(\mathbf{W}) = p(\mathbf{S})p(\mathbf{Z})$. Thus, the focus is on $p(\mathbf{Z})$ to define a prior over infinite binary matrices. The *Indian Buffet Process* (Griffiths and Ghahramani, 2011) is a stochastic process that defines a probability distribution over sparse binary matrices with a finite number of rows (data points) but infinite number of columns (features). The probability of a matrix $\mathbf{Z}$ under the Indian Buffet Process is defined by:

$$p(\mathbf{Z}) = \frac{\alpha^{K_+}}{\prod_{i=1}^{N} K_1^{(i)}!} \exp^{\{-\alpha H_N\}} \prod_{k=1}^{K_+} \frac{(N - m_k)!(m_k - 1)!}{N!}, \tag{4.22}$$

where m_k is the number of previous data points that used that feature, K_+ is the number of features for which $m_k > 0$, H_N is the N-th harmonic number and $K_1^{(i)}$ indicates the number of new features used by the i-th data point.

The analogy to the Indian buffet is that many Indian restaurants offer buffets with "infinite" number of dishes. N customers (data points) enter the restaurant one after the other and each customer has a choice from infinite number of dishes (features). The first customer takes a serving from each dish and stops after Poisson(α) number of dishes. Then the i-th customer chooses dishes subject to their popularity with probability $\frac{m_k}{i}$ with m_k the number of previous customers that used that dish. At the end of the dishes that have already been used, the customer tries Poisson($\frac{\alpha}{i}$) new dishes.

The above prior on the infinite sparse binary matrices could be used for modelling latent feature models with infinite number of features under the Bayesian non-parametric framework. Inference in these kind of models is challenging but has to be tractable to make the framework practical. Sampling methods have been successful derived and used for this purpose (Griffiths and Ghahramani, 2011) such as a variant of Gibbs sampling. More on sampling can be found in the next chapter.

CHAPTER 5

Sampling

Consider the problem of estimating the average age of all people in the United Kingdom. One would have to visit every single person, record their age and then calculate the average value. This is not a realistic and practical solution. A more tractable solution would be to choose samples of the population randomly from different cities and then average over the samples. This will provide a very good approximation of the true solution much faster and easier. The above toy problem illustrates the idea of *sampling* and the motivation as to why one should sample. In some cases, it might even not be possible to visit every single grid of the problem's domain due to the *curse of dimensionality* and therefore sampling would be the only possible solution. A recent survey has placed the Metropolis algorithm among the top ten algorithms that have influenced the development of science and engineering in the past century (Beichl and Sullivan, 2000). It is under the framework of Markov Chain Monte Carlo (MCMC) algorithms that create chains of sampling. Before starting the discussion on different sampling methods, it is advantageous to mention some problems in the machine learning literature that use the idea of sampling to obtain solutions and specifically in the Bayesian Inference literature. Given some unknown random variables $\mathbf{x}$ and $\mathbf{y}$ the Bayes' rule states that:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{\int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x})d\mathbf{x}}. \tag{5.1}$$

The normalizing factor is typically intractable and in order to obtain the posterior distribution, approximation is required. Another quantity of interest could be the marginal from the joint distribution $p(\mathbf{x}, \mathbf{y})$ given by:

$$p(\mathbf{x}) = \int_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})d\mathbf{y}, \tag{5.2}$$

which also could be intractable. More generally, although the posterior distribution over unobserved variables would be of direct interest, it is usually used for the purpose of evaluating expectations to make predictions. This is given by:

$$\mathbb{E}[f(\mathbf{x})] = \int_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x}|\mathbf{y})d\mathbf{x}, \tag{5.3}$$

where $f(\mathbf{x})$ is a function of $\mathbf{x}$. More generally, given a variable $\mathbf{x}$, its expectation is of interest or the expectation of a function $f(\mathbf{x})$:

$$\mathbb{E}[f(\mathbf{x})] = \int_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x})d\mathbf{x}. \tag{5.4}$$

The idea of sampling methods is to obtain samples drawn independently from the distribution $p(\mathbf{x})$ and approximate the expectation by the sum:

$$\hat{f} = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}^{(i)}). \tag{5.5}$$

Therefore, sampling methods solve the following two problems:

1. Draw samples $\mathbf{x}^{(i)} \sim p(\mathbf{x})$ where $p(\mathbf{x})$ is known upto a proportionality constant.
2. Calculate $\mathbb{E}[f(\mathbf{x})]$ either directly or by using the samples drawn after solving the first problem.

5.1 Inverse CDF Sampling

The simplest sampling method is that of sampling from a simple nonuniform distribution, assuming that a source of drawing samples from a uniform distribution exists. This assumption is valid since almost all scientific computing packages contain a routine that draws random samples from a uniform distribution and is available for use. The goal is to be able to transform the uniformly distributed random draws into a desired nonuniform distribution. Consider a random variable x which is uniformly distributed in the interval $[0, 1]$. A function that transforms x to a desired variable $y = f(x)$ is required in order to make y distributed by a desired nonuniform probability distribution $p(y)$. By changing the variables:

$$p(y) = p(x)\left|\frac{dx}{dy}\right|, \tag{5.6}$$

with $p(x) = \frac{1}{b-a}$ where $b = 1$ and $a = 0$. Therefore, $p(x) = 1$, re-arranging and integrating:

$$x = \int_{-\infty}^{y} p(\hat{y})d\hat{y}. \tag{5.7}$$

Therefore, $x = h(y)$ is the indefinite integral of $p(y)$ and the desired function to transform x into y is given by the inverse of the above, that is, $y = h^{-1}(x)$. This means that in order to transform a uniformly distributed random number x into a random variable y with a desired nonuniform distribution $p(y)$, it is necessary to use a function that is the inverse of the Cumulative Distribution Function (CDF) of $p(y)$.

For example, if the desired distribution is given by:

$$p(y) = \gamma \exp(-\gamma y), \tag{5.8}$$

where $0 \leq y < \infty$, then the CDF is given by:

$$\begin{aligned} h(y) &= \int_0^y \gamma \exp(-\gamma \hat{y}) d\hat{y} \\ &= -\exp(-\gamma y) + 1. \end{aligned} \tag{5.9}$$

Then,

$$y = h^{-1}(x), \tag{5.10}$$

and changing variables and re-arranging results in:

$$y = -\frac{1}{\gamma} \ln(1 - x). \tag{5.11}$$

Thus, by generating random uniformly distributed numbers x and transforming them using the inverse CDF $h^{-1}(x)$ of the desired nonuniform distribution, the result is sampling from the desired $p(y)$. This technique is very simple but yet very effective as it allows to sample any nonuniform distribution from a simple transformation of a uniformly distributed variable. However, the technique relies on the ability to invert and calculate the indefinite integral (CDF) of the required distribution. This is possible for simple nonuniform distribution, however it might not be possible for more complex ones. Therefore, other methods are necessary and are discussed next.

5.2 Rejection Sampling

Rejection sampling is a technique that allows the sampling from a distribution $p(x)$ which is known up to a proportionality constant. That is, it is necessary to sample from $p(x)$ but only $\tilde{p}(x)$ is known where:

$$p(x) = \frac{1}{C} \tilde{p}(x), \tag{5.12}$$

and C is a constant. This is often the case in many problems encountered in Bayesian Inference, for example, in the Bayes' rule where the normalizing constant is unknown but the rest is available. In order to use rejection sampling, it is required to use a simple *proposal distribution* $q(x)$ from which it is possible to draw samples. A constant k is also chosen so that the distribution $kq(x)$ always encloses the distribution $\tilde{p}(x)$ for all x. Firstly, a sample $x^{(i)}$ is drawn from $q(x)$. Then, a random number u drawn from a uniform distribution in the interval $[0, 1]$ is obtained. If u is smaller than the ratio $\frac{\tilde{p}(x^{(i)})}{kq(x^{(i)})}$ then the sample $x^{(i)}$ is accepted and is said to be drawn from the desired distribution $p(x)$. An illustration of the rejection sampling can be seen in Figure 5.1 where the area above the red curve is the rejection region and the area under the red curve is the accepted region. Algorithm 4 summarizes the rejection sampling method.

The values $x^{(i)}$ are sampled from $q(x)$ and are accepted with probability $\frac{\tilde{p}(x)}{kq(x)}$. This means that the acceptance probability is equal to:

$$\begin{aligned} p(\text{acceptance}) &= \int \frac{\tilde{p}(x)}{kq(x)} q(x) dx \\ &= \frac{1}{k} \int \tilde{p}(x) dx, \end{aligned} \tag{5.13}$$

This means that k should be chosen as small as possible to ensure a great degree of acceptance. However, k controls the coverage of the proposed distribution over the desired distribution and the bigger, in this case, the better. Thus, there is a trade-off between coverage and redundancy in sampling which needs to be quantified. Often, it might even not be possible to find the right k to cover the desired distribution and thus other techniques are used.

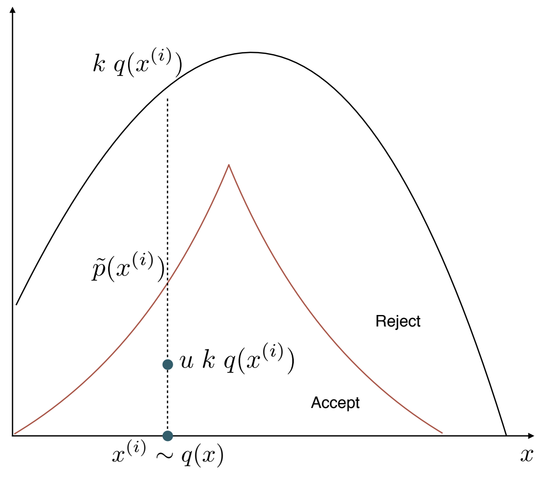


Figure 5.1: Rejection sampling scheme where a sample $x^{(i)}$ is accepted if $u\ k\ q(x^{(i)}) < \tilde{p}(x^{(i)})$.

Algorithm 4 Rejection Sampling

Initialization: $i = 0$ and number of samples N is set

```
1: repeat
2:     $x^{(i)} \sim q(x)$
3:     $u \sim \mathcal{U}_{(0,1)}$
4:     if $\left(u < \frac{\tilde{p}(x^{(i)})}{kq(x^{(i)})}\right)$
5:         accept $x^{(i)}$ and $i = i + 1$
6:     else
7:         reject
8: until $(i == N - 1)$
9: return $(\{x^{(i)}\}_{i=0}^{N-1})$
```

5.3 Importance Sampling

This technique provides a framework for approximating expectations directly. The idea of sampling as stated in (5.5) is that of drawing samples from a distribution $p(\mathbf{x})$ and then take the average over all function values at those samples for any function f. However, it might be impractical to sample from $p(\mathbf{x})$ and rather more efficient to calculate the expression given a particular value. This is often the case and thus *importance sampling* was proposed with this in mind.

A proposal distribution $q(\mathbf{x})$ is again used such that samples from $q(\mathbf{x})$ are easily drawn. The expectation of $f(\mathbf{x})$ is expressed by:

$$\begin{aligned}\mathbb{E}[f] &= \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \\ &= \int f(\mathbf{x})\frac{p(\mathbf{x})}{q(\mathbf{x})}q(\mathbf{x})d\mathbf{x}. \end{aligned} \tag{5.14}$$

N samples are i.i.d. drawn from $q(\mathbf{x})$ and then inserted into the expression:

$$\begin{aligned}\mathbb{E}[f] &\approx \frac{1}{N}\sum_{i=1}^{N}\frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}f(\mathbf{x}^{(i)}) \\ &\approx \frac{1}{N}\sum_{i=1}^{N} w(\mathbf{x}^{(i)})f(\mathbf{x}^{(i)}), \end{aligned} \tag{5.15}$$

where $w(\mathbf{x}^{(i)}) = \frac{p(\mathbf{x}^{(i)})}{q(\mathbf{x}^{(i)})}$ are the importance weights. All the samples are used as opposed to rejection sampling and each is given a weight of importance in the distribution $w(\mathbf{x}^{(i)})$.

If the distribution $p(\mathbf{x})$ is only known upto a normalizing constant, that is $p(\mathbf{x}) = \frac{1}{C}\tilde{p}(\mathbf{x})$, and $\tilde{p}(\mathbf{x})$ can be easily calculated, importance sampling can be used with a proposal distribution $q(\mathbf{x}) = \frac{1}{M}\tilde{q}(\mathbf{x})$. The expectation of $f(\mathbf{x})$ is now given by:

$$\mathbb{E}[f] = \frac{M}{C}\int f(\mathbf{x})\frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})}q(\mathbf{x})d\mathbf{x} \tag{5.16}$$

Sampling $\mathbf{x}^{(i)} \sim q(\mathbf{x})$ and approximating with a finite sum results in:

$$\begin{aligned}\mathbb{E}[f] &\approx \frac{M}{C}\frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{p}(\mathbf{x}^{(i)})}{\tilde{q}(\mathbf{x}^{(i)})}f(\mathbf{x}^{(i)}) \\ &\approx \frac{M}{C}\frac{1}{N}\sum_{i=1}^{N}\tilde{w}(\mathbf{x}^{(i)})f(\mathbf{x}^{(i)}), \end{aligned} \tag{5.17}$$

where $\tilde{w}(\mathbf{x}^{(i)}) = \frac{\tilde{p}(\mathbf{x}^{(i)})}{\tilde{q}(\mathbf{x}^{(i)})}$ are again the weights of each sample in the distribution. The constant ratio can also be calculated using the same samples $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ by:

$$\begin{aligned}\frac{C}{M} &= \frac{1}{M}\int \tilde{p}(\mathbf{x})d\mathbf{x} \\ &= \int \frac{\tilde{p}(\mathbf{x})}{\tilde{q}(\mathbf{x})}q(\mathbf{x})d\mathbf{x}, \end{aligned} \tag{5.18}$$

and then approximating the integral with a finite sum using the samples obtained from $q(\mathbf{x})$:

$$\frac{C}{M} \approx \frac{1}{N}\sum_{i=1}^{N} \tilde{w}(\mathbf{x}^{(i)}). \tag{5.19}$$

Hence, the expectation is obtained using:

$$\mathbb{E}[f] \approx \sum_{i=1}^{N} s(\mathbf{x}^{(i)})f(\mathbf{x}^{(i)}), \tag{5.20}$$

where $s(\mathbf{x}^{(i)}) = \frac{\tilde{w}(\mathbf{x}^{(i)})}{\sum_{i=1}^{N}\tilde{w}(\mathbf{x}^{(i)})}$.

Importance sampling relies on the samples drawn from $q(\mathbf{x})$ and thus this distribution has to be chosen with great consideration. A major concern is that $p(\mathbf{x})f(\mathbf{x})$ varies in space and its probability mass is concentrated in small regions. The importance weights $w(\mathbf{x}^{(i)})$ are then dominated by only a few weights and the rest are near zero which makes the samples associated with them, negligible. Therefore, the useful sample size is much smaller than N and this can be much worse if none of the samples live in regions where $p(\mathbf{x})f(\mathbf{x})$ is large. In order to find a good $q(\mathbf{x})$ that will not have the above problems, the minimization of the variance of the estimator in (5.15) is necessary. This is attained (Andrieu et al., 2003) by:

$$q(\mathbf{x}) = \frac{|f(\mathbf{x})|\,p(\mathbf{x})}{\int |f(\mathbf{x})|\,p(\mathbf{x})d\mathbf{x}}. \tag{5.21}$$

Although this result does not help directly since it is not easy to sample from $|f(\mathbf{x})|\,p(\mathbf{x})$, it illustrates an essential insight. That is, higher sampling efficiency is possible when the important regions $|f(\mathbf{x})|\,p(\mathbf{x})$ are identified and the focus of sampling is towards those areas. The key requirements of $q(\mathbf{x})$ is thus not to be small or zero in areas where $|f(\mathbf{x})|\,p(\mathbf{x})$ is large. This is illustrated in Figure 5.2 where $q(\mathbf{x})$ is large when $|f(\mathbf{x})|\,p(\mathbf{x})$ is significant and smaller when it is less significant.

There are alternative methods one can use to obtain much better proposal distributions $q(\mathbf{x})$. Instead of setting $q(\mathbf{x})$ at the beginning of the sampling procedure, it is possible to adapt the parameters of the proposal distribution as new samples are obtained. This can be done using an online scheme of *adaptive importance sampling* where the derivative of a cost function is obtained and the parameters are updated using a gradient descent or Hessian optimization scheme. In the long run, this can have great increase in sampling efficiency since the sampler learns the important regions and samples more frequently in those areas.

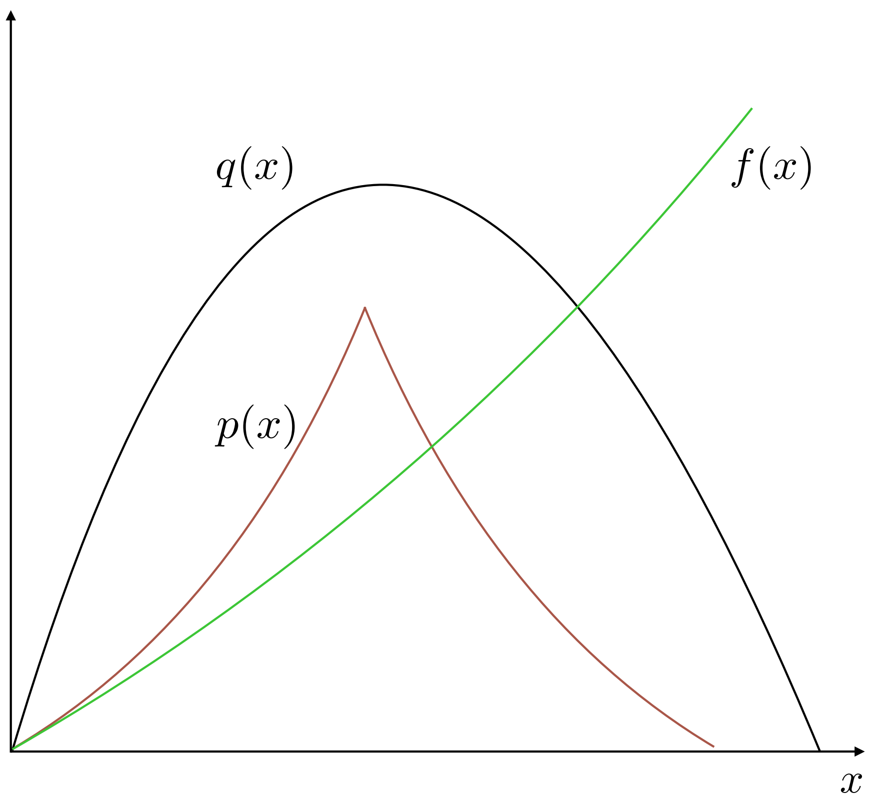


Figure 5.2: Illustration of the importance sampling scheme where $q(\mathbf{x})$ should be chosen to have sufficient mass when both $p(\mathbf{x})$ and $f(\mathbf{x})$ are large

5.4 Markov Chain Monte Carlo

The methods discussed so far are effective in low dimensions however, they struggle when the problem lives in a high dimensional space. Ideas from the above techniques and insights will be useful when discussing more advanced sampling schemes such as the *Markov Chain Monte Carlo* (MCMC). These group of methods scale well with the dimensionality of the problem and are one of the most widely used tools in approximate inference and learning. MCMC are used when it is not possible to draw samples from $p(\mathbf{x})$ but rather only able to evaluate $p(\mathbf{x})$ upto a normalizing constant. Before discussing the sampling scheme, an introduction to Markov Chains will be given to motivate MCMC.

A *Markov Chain* is a stochastic process in which future states are independent of past states given the current state. Consider a random variable $\mathbf{x} \in \mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_s\}$ where s is the number of possible values that can take. A draw of $\mathbf{x}^{(i)}$ is a state at iteration i. The next draw is dependent only at the current draw $\mathbf{x}^{(i)}$ and not on any of the past draws resulting in the following Markov property:

$$p(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}, \mathbf{x}^{(i-1)}, ..., \mathbf{x}^{(1)}) = p(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}). \tag{5.22}$$

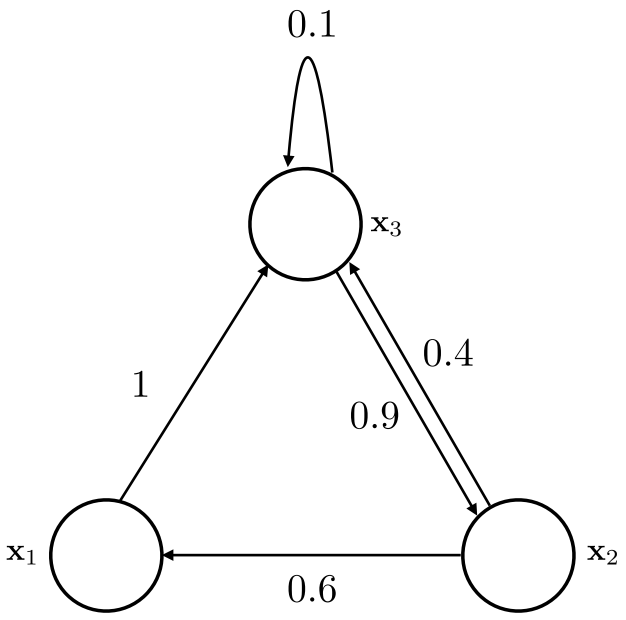


Figure 5.3: Illustration of a transition graph for a variable $\mathbf{x}$ living in $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$

That is, the Markov chain is a set of samples $\mathbf{x}$ that are each slightly dependent on the previous one. The chain moves around the parameter space and remembers only where it was in the previous state. The movement around the space is controlled by a *transition matrix* which describes the probability of moving to some other state based on the current state. This transition matrix is defined by $\mathbf{T} = \mathbf{T}(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})$ and the chain is said to be homogeneous if $\mathbf{T}$ remains invariant for all i. This means that the transition matrix is fixed throughout the chain.

In order to illustrate the great power of Markov chains, an example is illustrated in Figure 5.3. A variable $\mathbf{x} \in \mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ has three different states/possibilities. The edges between each state correspond to the probability of moving from one state to the other. Thus, the transition matrix of the example is given by:

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 \\ 0.6 & 0 & 0.4 \\ 0 & 0.9 & 0.1 \end{bmatrix} \tag{5.23}$$

with $\sum_{\mathbf{x}^{(i+1)}} \mathbf{T}(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) = 1$ for all i (ie each row of the transition matrix sums to 1). The chain is initialized with a starting distribution $\Pi^{(0)} = [0.5, 0.3, 0.2]^T$. Then, at the first iteration a new sample is drawn $\mathbf{x}^{(1)} \sim \Pi^{(1)}$ where $\Pi^{(1)} = \Pi^{(0)}\mathbf{T} = [0.18, 0.18, 0.64]^T$. At iteration 2, $\mathbf{x}^{(2)} \sim \Pi^{(2)}$ where $\Pi^{(2)} = \Pi^{(1)}\mathbf{T}$ and so on. At iteration i, the distribution is $\Pi^{(i)} = \Pi^{(0)}\mathbf{T}^i$ and this converges to a probability distribution $p(\mathbf{x})$, called the *invariant distribution*, for any starting distribution. This stability result of the Markov Chain is the key to MCMC since if the invariant distribution is the target distribution then it is possible to sample from it by initializing the chain from any point in the probability space. This is satisfied however under certain conditions on the transition matrix $\mathbf{T}$. That is, the transition matrix has to have the properties of *irreducibility* and *aperiodicity.* Irreducibility means that for any state of the chain, there is positive probability of visiting all other states and aperiodicity means that the chain is not trapped in cycles. In order to ensure that a particular distribution $p(\mathbf{x})$ is the desired invariant distribution, the following *detailed balance* condition is required:

$$\begin{aligned} p(\mathbf{x}^{(i+1)})\mathbf{T}(\mathbf{x}^{(i)}|\mathbf{x}^{(i+1)}) &= p(\mathbf{x}^{(i)})\mathbf{T}(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) \\ p(\mathbf{x}^{(i+1)}) &= \sum_{\mathbf{x}^{(i)}} p(\mathbf{x}^{(i)})\mathbf{T}(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}), \end{aligned} \tag{5.24}$$

where both sides were summed over $\mathbf{x}^{(i)}$. Markov Chains can then be used to build MCMC samplers since the draws from the Markov Chain correspond to draws from the target distribution if this is also the invariant distribution. This is only possible if the Markov Chain satisfies the above conditions in order to satisfy the *ergodicity property* which allows the dependent samples of the Markov Chain to be used as independent in the Monte Carlo integration. Consider an example of the concepts of irreducibility and aperiodicity in the real-world. Internet search follows links from different web pages which can be viewed as nodes in a transition graph. As web users, the need to avoid cycles (aperiodicity) and able to access all the existing web pages in the world (irreducibility) is essential. This is exactly what information retrieval algorithms use in order to rank the results in web searches. A popular example is the PageRank algorithm that ranks the web pages from their invariant probability distribution.

Another important consideration is to choose a *burn-in* period. This is the procedure where samples from the beginning of the chain are thrown away until a number of samples has passed. This is due to the fact that at the beginning of the chain, the samples are often random (since the initial state of the chain can be anywhere in the probability space) and thus do not correspond to the invariant distribution. As the samples get closer to the invariant distribution, it is said that the Markov Chain has burned in and all the samples are now drawn from the target distribution. It is not clear, however, when the burn-in period starts and thus empirical validation is required. In addition, by only keeping every k-th draw from the chain, it could help to get closer to i.i.d. draws. This is called *thinning* but it is not necessary if the ergodicity property is satisfied. Thus, MCMC are a class of methods that simulate draws that are slightly dependent but can be used to approximate a target distribution. Two very popular methods are discussed below.

Metropolis-Hastings

The Metropolis-Hastings (MH) algorithm is the most popular MCMC sampler and many practical MCMC samplers can be interpreted as special cases of MH. In a similar fashion with the rejection and importance sampling, it requires a proposal distribution $q(\mathbf{x})$ and involves the sampling of a candidate $\mathbf{x}^*$ given the current state. That is, a new sample is drawn from $q(\mathbf{x}^*|\mathbf{x}^{(i)})$ given the current state. This sample however is not guaranteed to be accepted. The Markov chain only moves to $\mathbf{x}^*$ if a condition is satisfied, otherwise it stays at its current state and tries a new sample. A draw from a uniform distribution is obtained in the interval

$[0, 1]$ and then compared with $\mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*)$ which is defined by:

$$\mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*) = \min \left\{ 1, \frac{p(\mathbf{x}^*)q(\mathbf{x}^{(i)}|\mathbf{x}^*)}{p(\mathbf{x}^{(i)})q(\mathbf{x}^*|\mathbf{x}^{(i)})} \right\}. \tag{5.25}$$

This means that a new sample is accepted with probability $\mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*)$ and heavily depends on the choice of the proposal distribution. The transition matrix of the MH algorithm is defined by:

$$\mathbf{T}(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)}) = q(\mathbf{x}^{(i+1)}|\mathbf{x}^{(i)})\mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^{(i+1)}) + \delta_{\mathbf{x}^{(i)}}(\mathbf{x}^{(i+1)})r(\mathbf{x}^{(i)}), \tag{5.26}$$

where $r(\mathbf{x}^{(i)})$ deals with the rejection of the draw and is defined by:

$$r(\mathbf{x}^{(i)}) = \int_X q(\mathbf{x}^*|x^{(i)})(1 - \mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*))d\mathbf{x}^*. \tag{5.27}$$

It can be shown (Andrieu et al., 2003; Bishop, 2006) that the samples generated by the MH algorithm are draws from the target distribution since the transition matrix satisfies the detailed balance condition. The algorithm always allows for rejection and therefore satisfies the aperiodic condition. For irreducibility, the support of the proposed distribution has to include the support of the target distribution. Thus, great care is needed to propose a good $q(\mathbf{x})$. A summary of the algorithm can be seen in Algorithm 5. Note that it is only necessary to know $p(\mathbf{x})$ upto a constant of proportionality since it is only required in the form of ratios and the constants cancel each other out.

Algorithm 5 Metropolis-Hastings Algorithm

Initialization: Initialize $\mathbf{x}^{(0)}$ and number of samples N

```
1: for $i = 0$ to $N - 1$ do
2:     $u \sim \mathcal{U}_{(0,1)}$
3:     $\mathbf{x}^* \sim q(\mathbf{x}^*|\mathbf{x}^{(i)})$
4:     if $\left(u < \mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*) = \min\left\{1, \frac{p(\mathbf{x}^*)q(\mathbf{x}^{(i)}|\mathbf{x}^*)}{p(\mathbf{x}^{(i)})q(\mathbf{x}^*|\mathbf{x}^{(i)})}\right\}\right)$
5:         $\mathbf{x}^{(i+1)} = \mathbf{x}^*$
6:     else
7:         $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)}$
   return $(\{x^{(i)}\}_{i=0}^{N-1})$
```

The specific choice of the proposed distribution can change the performance of the algorithm. It is very common to use a Gaussian proposal distribution $q(\mathbf{x}^*|\mathbf{x}^{(i)}) = \mathcal{N}(\mathbf{x}^{(i)}, \sigma^2)$ which is easy to sample and the results can be easily interpreted. Different choices of the proposal standard deviation lead to very different results. If the standard deviation is too small, the acceptance rate will be high but with poor performance since only a small proportion of probability space will be covered and thus many modes of probability mass could be missed. On the other hand, if the standard deviation is too big, the rejection rate can be very high because of the spaces visited for which the target probability is very small. Thus, the chain will not move and the samples will be highly correlated. If all the modes and large proportion of the probability space is explored along with high acceptance rate, then the MH sampler has good sampling performance with the chain mixing well. There are many variations of proposed distributions for MH but as it can be seen, not a single distribution can be successful for any given application. However, there are techniques which can adapt proposal distributions to the observations at hand. Further information on this can be found in (Andrieu et al., 2003).

Gibbs Sampling

Gibbs Sampling (Geman and Geman, 1984) is a special case of the MH algorithm which is widely used due to its simplicity. Consider $\mathbf{x} \in \mathcal{R}^D$ and that the full conditionals $p(x_j|x_1, x_2, ..., x_{j-1}, x_{j+1}, ..., x_D)$ $\forall j$ are available. Gibbs sampling uses as a proposal distribution for $j = 1, .., D$:

$$q(\mathbf{x}^*|\mathbf{x}^{(i)}) = \begin{cases} p(x_j^*|\mathbf{x}_{-j}^{(i)}) & \text{if } \mathbf{x}_{-j}^* = \mathbf{x}_{-j}^{(i)} \\ 0 & \text{otherwise.} \end{cases} \tag{5.28}$$

where $\mathbf{x}_{-j}^{(i)}$ denotes $x_1^{(i)}, x_2^{(i)}, ..., x_D^{(i)}$ with j omitted. The same applies for $\mathbf{x}_{-j}^*$. The corresponding acceptance probability is given by:

$$\begin{aligned} \mathcal{A}(\mathbf{x}^{(i)}, \mathbf{x}^*) &= \min\left\{1, \frac{p(\mathbf{x}^*)q(\mathbf{x}^{(i)}|\mathbf{x}^*)}{p(\mathbf{x}^{(i)})q(\mathbf{x}^*|\mathbf{x}^{(i)})}\right\} \\ &= \min\left\{1, \frac{p(\mathbf{x}^*)p(x_j^{(i)}|\mathbf{x}_{-j}^*)}{p(\mathbf{x}^{(i)})p(x_j^*|\mathbf{x}_{-j}^{(i)})}\right\} \\ &= \min\left\{1, \frac{p(x_j^*|\mathbf{x}_{-j}^*)p(\mathbf{x}_{-j}^*)p(x_j^{(i)}|\mathbf{x}_{-j}^*)}{p(x_j^{(i)}|\mathbf{x}_{-j}^{(i)})p(\mathbf{x}_{-j}^{(i)})p(x_j^*|\mathbf{x}_{-j}^{(i)})}\right\} \\ &= \min\left\{1, \frac{p(\mathbf{x}_{-j}^*)}{p(\mathbf{x}_{-j}^{(i)})}\right\} \\ &= 1, \end{aligned} \tag{5.29}$$

where the fact that $\mathbf{x}_{-j}^* = \mathbf{x}_{-j}^{(i)}$ is used due to the fact that all these components of $\mathbf{x}$ are fixed/unchanged by that sampling step. In addition, $p(\mathbf{x}^*) = p(x_j^*|\mathbf{x}_{-j}^*)p(\mathbf{x}_{-j}^*)$ is used in order to simplify the expression. This means that the acceptance probability for each proposal is one and thus at every iteration, the full conditionals can be used to draw samples for the Markov Chain without throwing away any samples. The size of the samples needs to be specified a priori and matches the number of iterations of the sampler. It is very important to monitor the evolution of the sampler and make sure that the number of iterations is sufficient.

Consider the distribution $p(\mathbf{x}) = p(x_1, x_2, ..., x_D)$. The Gibbs sampler can be used to sample from the joint distribution if the full conditionals for each parameter x_j are available. This is due to the following derivation which shows that knowledge of the conditional densities can be used to obtain the joint distribution. By using the product and the sum rule of probability theory, it is possible to completely represent the joint distribution with conditional distributions. The following example illustrates this with a joint distribution $p(x_1, x_2)$ given by two random variable x_1 and x_2 :

$$\begin{aligned} p(x_1, x_2) &= p(x_2|x_1)p(x_1) \\ &= \frac{p(x_2|x_1)}{\frac{1}{p(x_1)}} \\ &= \frac{p(x_2|x_1)}{\int \frac{p(x_2)}{p(x_1)} dx_2} \\ &= \frac{p(x_2|x_1)}{\int \frac{\frac{p(x_2)}{p(x_1,x_2)}}{\frac{p(x_1)}{p(x_1,x_2)}} dx_2} \\ &= \frac{p(x_2|x_1)}{\int \frac{p(x_2|x_1)}{p(x_1|x_2)} dx_2}. \end{aligned} \tag{5.30}$$

In general terms, the full conditionals for each parameter can be obtained by writing out the probability distribution expression and then pick blocks of parameters (groups of 1, 2, 4 etc). Drop every term that depends on those parameters and treat it as a constant.

For example, for blocks of 1, consider the probability distribution $p(\mathbf{x}) = p(x_1, x_2, x_3)$. The Gibbs sampler initializes $x_1^{(0)}, x_2^{(0)}$ and $x_3^{(0)}$ and then starts the process of keeping all variables fixed except one at each step. It draws from the conditional distribution of $p(x_1|x_2^{(0)}, x_3^{(0)})$ and sets the value as the new value of $x_1^{(1)}$. Next, it draws from the conditional distribution of $p(x_2|x_1^{(1)}, x_3^{(0)})$ using the new value of x_1 and sets the new value of $x_2^{(1)}$ to the result. Finally, it draws from the conditional distribution of $p(x_3|x_1^{(1)}, x_2^{(1)})$ using the new value of x_1 and x_2 and sets the new value of $x_3^{(1)}$ to the result. This is repeated until sufficient samples are obtained. A summary of the algorithm can be seen in Algorithm 6. Great consideration is required when choosing the block size, the burn-in period or whether to use thinning.

Algorithm 6 Gibbs Sampling

```
Initialization: Initialize $\mathbf{x}^{(0)} \in \mathcal{R}^D$ and number of samples $N$

• for $i = 0$ to $N-1$ do
•     $x_1^{(i+1)} \sim p(x_1|x_2^{(i)}, x_3^{(i)}, ..., x_D^{(i)})$
•     $x_2^{(i+1)} \sim p(x_2|x_1^{(i+1)}, x_3^{(i)}, ..., x_D^{(i)})$
•     ⋮
•     $x_j^{(i+1)} \sim p(x_j|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., x_D^{(i)})$
•     ⋮
•     $x_D^{(i+1)} \sim p(x_D|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{D-1}^{(i+1)})$
  return $(\{\mathbf{x}^{(i)}\}_{i=0}^{N-1})$
```

Bibliography

M. Aharon, M. Elad, and A. Bruckstein. k -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11): 4311–4322, Nov 2006.

C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50(1-2):5–43, 2003. ISSN 0885-6125.

F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

M. J. Beal and Z. Ghahramani. The variational bayesian em algorithm for incomplete data: with application to scoring graphical model structures, 2003.

I. Beichl and F. Sullivan. The metropolis algorithm. *Computing in Science Engineering*, 2 (1):65–69, Jan 2000. ISSN 1521-9615.

Y. Bengio. Deep learning of representations: Looking forward. In *Proceedings of the First International Conference on Statistical Language and Speech Processing*, SLSP'13, pages 1–37, Berlin, Heidelberg, 2013. Springer-Verlag. ISBN 978-3-642-39592-5.

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. ISSN 0162-8828.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, June 2005.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(6):721–741, Nov. 1984. ISSN 0162-8828.

Z. Ghahramani and G. E. Hinton. The em algorithm for mixtures of factor analyzers. Technical report, 1997.

P. Gorder. Neural networks show new promise for machine vision. *Computing in Science Engineering*, 8(6):4–8, Nov 2006. ISSN 1521-9615. doi: 10.1109/MCSE.2006.117.

T. L. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. *J. Mach. Learn. Res.*, 12:1185–1224, July 2011. ISSN 1532-4435.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, Aug. 2002. ISSN 0899-7667.

A. Hyvrinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13:411–430, 2000.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 05 2015.

D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, New York, NY, USA, 2002. ISBN 0521642981.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012. ISBN 0262018020, 9780262018029.

R. M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):pp. 249–265, 2000. ISSN 10618600.

S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. *Proceedings of the Twenty-eight International Conference on Machine Learning, ICML*, 2011.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

M. Tipping. Bayesian inference: An introduction to principles and practice in machine learning. In O. Bousquet, U. von Luxburg, and G. Rtsch, editors, *Advanced Lectures on Machine Learning*, volume 3176 of *Lecture Notes in Computer Science*, pages 41–62. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-23122-6.

P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

M. Zhou, H. Chen, J. Paisley, L. Ren, L. Li, Z. Xing, D. Dunson, G. Sapiro, and L. Carin. Nonparametric bayesian dictionary learning for analysis of noisy and incomplete images. *Image Processing, IEEE Transactions on*, 21(1):130–144, Jan 2012.