On The Basics Of ... For Dummies

Michiel Baptist

March 5, 2020

Introduction goes here: - Book by me for me - Book of derivations, theory, usefullness - Fill in as I go - ...

Contents

Chapter 1

Definitions, notation and properties

In this section I present an overview of the mathematical notation I employ and some of the more common definitions. Each definition comes with a little intuition, or at least a view into my own way of thinking about the respective definitions. Whenever a definition is mentioned in the text, this chapter is referred to.

1.1 Notation

1.2 Commonly used definitions

1.2.1 Norms

Humans usually like to compare things to each other. Which apple is riper, juicier and generally better looking? Do I procrastinate, do I try to be productive? Often one or several criteria are compared. Color of the apple, size of the apple, and so on. In mathematics, humans have implemented a very similar notion upon objects such as vectors and matrices. This notion is called a *norm*. It's definition is rather simple, but elegant:

Definition 1 (Mathematical norm)**.** Given a vector space V in over a field in $\mathbb{R}$ or $\mathbb{C}$. A norm $\|.\|$ is a function $\|.\| : V \mapsto \mathbb{R}$ such that $\forall v, u \in V$:

1. $\|v\| \geq 0$
2. $\|v + u\| \leq \|v\| + \|u\|$
3. $\|av\| = |a| \, \|v\| \ \forall a \in \mathbb{R}$
4. $\|v\| = 0 \Leftrightarrow v = 0$

Condition 2 is called the triangle inequality. Note that the definition of norm might allow many definitions. Not only the norms such as the euclidean norm (p-norm) for example.

Definition 2 (The $\mathcal{L}^p$ norm or the p-norm)**.** The $\mathcal{L}^p$ norm or p-norm $\|.\|_p$ is defined over n dimensional vector spaces. It is given by:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}} \tag{1.1}$$

These types of norms are most commonly used in statistics and mathematics. The *Euclidean* norm is simply the $\mathcal{L}^2$ norm. The $\mathcal{L}^1$ norm is usually referred to as the taxicab norm or the Manhattan norm. To see why consider taking a taxi to point $(2, 3)$ in the plane and the driver is only allowed to drive horizontally and vertically. The taxicab norm is then simply the total amount of driven distance.

So far we described norms in terms of $1 \times m$ dimensional vectors. The concept of a norm for matrices is somewhat less intuitive. However, some matrix norms are. Take for example the Frobenius norm, it is the natural extension of the $\mathcal{L}^2$ norm for vectors.

Definition 3 (Frobenius norm)**.** The frobenius norm is defined over $m \times n$ matrices $X \in \mathbb{R}^{m \times n}$. It is the natural extension of the 2-norm for vectors. It is defined as:

$$\begin{aligned} \|X\|_F &= \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |x_{ij}|^2} \\ &= \sqrt{trace(X^T X)} \end{aligned} \tag{1.2}$$

To see why the second line in equation 1.2 holds, let's write out $trace(X^T X)$.

$$\begin{aligned} trace(X^T X) &= trace \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_n & — \end{pmatrix} \begin{pmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{pmatrix} \\ &= trace \begin{pmatrix} x_1^T x_1 & & & \\ & x_2^T x_2 & & \\ & & \ddots & \\ & & & x_n^T x_n \end{pmatrix} \\ &= \sum_{i=1}^{n} x_i^T x_i \\ &= \sum_{i=1}^{n} \left(\sum_{j=1}^{m} x_{ij}^2 \right) \end{aligned} \tag{1.3}$$

1.3 Matrix properties

Definition 4 (Idempotent matrix)**.** A matrix A is idempotent iff $AA = A$.

Property 1. Let $X \in \mathbb{R}^{m \times n}$ then

$$XX^T = \sum_{i=1}^{n} x_i x_i^T \tag{1.4}$$

Proof. Depending on your viewpoint of matrix multiplication this property might seem trivial to prove. However, here is one way. Let's begin from the left side of the equation:

$$\begin{aligned}
XX^T &= \begin{pmatrix} — & x_1 & — \\ & \dots & \\ — & x_m & — \end{pmatrix} \begin{pmatrix} | & & | \\ x_1 & & x_m \\ | & & | \end{pmatrix} \\
&= \begin{pmatrix} \|x_1\|^2 & x_1 x_2^T & & \\ x_2 x_1^T & \|x_2\|^2 & & \\ & & \ddots & \\ & & & \|x_m\|^2 \end{pmatrix} \\
&= \begin{pmatrix} \sum_{j=1}^n x_{1j}^2 & \sum_{j=1}^n x_{1j} x_{2j} & & \\ \sum_{j=1}^n x_{1j} x_{2j} & \sum_{j=1}^n x_{2j}^2 & & \\ & & \ddots & \\ & & & \sum_{j=1}^n x_{mj}^2 \end{pmatrix} \\
&= \sum_{j=1}^{n} \begin{pmatrix} x_{1j}^2 & x_{1j} x_{2j} & & \\ x_{1j} x_{2j} & x_{2j}^2 & & \\ & & \ddots & \\ & & & x_{nj}^2 \end{pmatrix} \\
&= \sum_{j=1}^{n} x_j x_j^T
\end{aligned} \tag{1.5}$$

□

1.4 Matrix derivatives

Property 2. Given vector $b \in \mathbb{R}^m$ and symmetric matrix X $in \mathbb{R}^{m \times m}$ then it holds that:

$$\nabla_b b^T X b = 2Xb \tag{1.6}$$

Proof. Let's write out the $b^t Xb$:

$$\begin{aligned} b^T Xb &= \sum_{i=1}^{m} b_i (Xb)_i \\ &= \sum_{i=1}^{m} b_i \big(\sum_{j=1}^{m} x_{ij} b_j \big) \\ &= \sum_{i=1}^{m} \sum_{j=1}^{m} b_i b_j x_{ij} \end{aligned} \tag{1.7}$$

Finding the partial derivative for a given b_k is then straight forward:

$$\begin{aligned} \frac{\partial}{\partial x_k} \sum_{i=1}^{m} \sum_{j=1}^{m} b_i b_j x_{ij} &= 2b_k x_{kk} + \sum_{\substack{j=1 \\ j \neq k}}^{m} b_j x_{kj} + \sum_{\substack{j=1 \\ j \neq k}}^{m} b_j x_{jk} \\ &= 2 \sum_{j=1}^{m} b_j x_{kj} \\ &= 2 x_k b_j \end{aligned} \tag{1.8}$$

The gradient is defined as:

$$\nabla_x f(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_m} f(x) \end{pmatrix} = 2 \begin{pmatrix} x_1 b \\ \vdots \\ x_m b \end{pmatrix} = 2Xb \tag{1.9}$$

□

Property 3. $\nabla_x \|x\|^2 = 2x$

Proof. We find the partial derivative for x_i:

$$\begin{aligned} \frac{\partial}{\partial x_i} \|x\|^2 &= \frac{\partial}{\partial x_i} x^T x \\ &= \frac{\partial}{\partial x_i} \sum_{j=1}^{m} x_j^2 \\ &= 2x_i \end{aligned} \tag{1.10}$$

The gradient is defined as:

$$\nabla_x f(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_m} f(x) \end{pmatrix} = 2 \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} \tag{1.11}$$

□

Property 4. $\nabla_x \|Ax\|^2 = 2A^TAx = 2(Ax)^TA$

Proof. This proof is long and winded. However, it gives you intuition. In the following, a_i is the i'th row of A. $a_{.k}$ is by extension the k'th column of A. Let's begin by taking a partial derivative for x_i:

$$\begin{aligned}
\frac{\partial}{\partial x_k}\|Ax\|^2 &= \frac{\partial}{\partial x_k}\left\langle \begin{pmatrix} a_1x \\ \vdots \\ a_mx \end{pmatrix}, \begin{pmatrix} a_1x \\ \vdots \\ a_mx \end{pmatrix} \right\rangle \\
&= \frac{\partial}{\partial x_k}\sum_{i=1}^{m}(a_ix)^2 \\
&= \frac{\partial}{\partial x_k}\sum_{i=1}^{m}\Big(\sum_{j=1}^{m}a_{ij}x_j\Big)^2 \\
&= 2\sum_{i=1}^{m}\Big(\sum_{j=1}^{m}a_{ij}x_j\Big)\frac{\partial}{\partial x_k}\Big(\sum_{j=1}^{m}a_{ij}x_j\Big) \\
&= 2\sum_{i=1}^{m}\Big(\sum_{j=1}^{m}a_{ij}x_j\Big)a_{ik} \\
&= 2\sum_{i=1}^{m}(a_ix)a_{ik}
\end{aligned} \tag{1.12}$$

In the above equation (a_ix) is the i'th element of Ax. Since i is going from $1 \to m$ we see in the last line that we are multiplying Ax by the k'th column of A. We rewrite as follows:

$$\frac{\partial}{\partial x_i}\|Ax\|^2 = 2(Ax)^TA_{.k} \tag{1.13}$$

The gradient is defined as:

$$\begin{aligned}
\nabla_x f(x) &= \begin{pmatrix} \frac{\partial}{\partial x_1}f(x) \\ \vdots \\ \frac{\partial}{\partial x_m}f(x) \end{pmatrix} = 2\begin{pmatrix} (Ax)^TA_{.1} \\ \vdots \\ (Ax)^TA_{.m} \end{pmatrix} \\
&= 2(Ax)^T\begin{pmatrix} A_{.1} & \dots & A_{.m} \end{pmatrix} \\
&= 2(Ax)^TA
\end{aligned} \tag{1.14}$$

□

1.5 Geometric properties

Property 5. For any $k \neq 1$ and $m > 1$ it holds that:

$$\sum_{i=1}^{m}k^{i-1}(k-1) = k^m - 1 \tag{1.15}$$

Proof. We can decompose the left hand side of the equation:

$$\sum_{i=1}^{m} k^{i-1}(k-1) = \sum_{i=1}^{m} k^i - \sum_{i=1}^{m} k^{i-1} \tag{1.16}$$

We can use the standard proof for geometric series. Assume $\sum_{i=1}^{m} k^i = S$ then it holds that:

$$\begin{aligned} kS &= \underbrace{k^2 + k^3 + \cdots + k^m}_{=S-k} + k^{m+1} \\ &= S - k + k^{m+1} \\ S(k-1) &= k^{m+1} - k \\ S &= \frac{k^{m+1} - k}{k-1} \end{aligned} \tag{1.17}$$

Note that for the right part in the right hand side of equation 1.16:

$$\sum_{i=1}^{m} k^{i-1} = S/k \tag{1.18}$$

We complete the proof by plugging in the results from equation 1.16 and 1.17:

$$\begin{aligned} \frac{k^{m+1} - k}{k-1} - \frac{k^m - 1}{k-1} &= \frac{k^{m+1} - k^m - k + 1}{k-1} \\ &= \frac{(k^m - 1)(k-1)}{k-1} \end{aligned} \tag{1.19}$$

The proof holds for $k \neq 1$. □

Chapter 2

Principle Component Analysis (PCA)

2.1 Matrix factorization as a tool

The core concept of principle component analysis is making use of matrix factorization techniques. These techniques, as the name implies, decompose a matrix of interest $X \in \mathbb{R}^{m \times n}$ into several matrices such that their product equals X. For example, an approximation of the data matrix X could be given by the product of $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$:

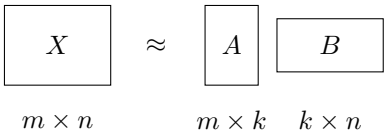


Figure 2.1: Example of factorization.

A whole range of practical problems can be written as a matrix factorization problems. Often we can gain a lot of insights based on decompositions.

In the previous example, the matrix X was a $m \times n$ matrix. It was decomposed into $m \times k$ and $k \times n$ matrices. Naturally, A and B share a dimension such that their product is defined. In matrix factorization problems this is almost always the case. This k can be chosen differently based on the problem you are dealing with. For example, if one wants to decompose a matrix without losing any information at all, one chooses $k = \min(m, n)$. It is not always desirable to decompose a given matrix, without losing information. In most cases one is interested in smaller k values. Be it for reducing dimension, easier understanding of the decomposition, memory and time considerations.

2.2 Frobenius norm as a closeness metric

When using a k value smaller than $\min(m, n)$, we lose information when decomposing matrix $X\mathbb{R}^{m\times n}$. The decomposition then becomes an approximation $X \approx AB$ As we don't want to lose too much information, we need a measure of how *close* two matrices are. One of the ways we can measure this is using the frobenius norm:

$$d(A, B) = \|A - B\|_F \tag{2.1}$$

Assume again, we wish to find a matrix decomposition of X where k is small, so we lose information, the decomposition is an approximation. We would then measure how good this approximation is using the frobenius norm:

$$d(X, AB) = \|X - AB\|_F \tag{2.2}$$

Transforming this into a minimization objective:

$$\min_{A,B} \|X - AB\|_F \tag{2.3}$$

2.3 PCA as dimension reduction

PCA is primarily a technique to *reduce the dimension* of a given data set X. Assume there are n data points of dimension m, i.e. $\mathcal{X} = \{x_i | x_i \in \mathbb{R}^m, i \in \{1 \dots n\}\}$. Conveniently, this can be encapsulated as a matrix $X \in \mathbb{R}^{n\times n}$. We can then decompose the matrix as before into $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$. Assume now that k is significantly smaller $\min(m, n)$. Instead of having $\mathcal{O}(mn)$ values we now only have $\mathcal{O}(k(m + n))$ values. For a large m this is significant. The above was an example of dimension reduction. PCA is an example of such a dimension reduction techniques. Even more, it is one of the most used techniques out there for reducing dimension.

Example 1. As an example we could look at an approximation of a face using PCA.

(from: Turk and Pentland, Eigenfaces for Recognition, 1991)

The dimension of the original data points is the exact amount of pixels in the image. Perhaps the amount of pixels is unmanageable. In this example, the faces were are reconstructed using $k = 4$ images. Thus, instead of considering each data point as a collection of pixels, we reduced the dimension of the data by considering each data point as a combination of 4 base images. We have reduced the dimension of the data to 4. Of course, in order to re-construct the original images, the 4 base images are required.

2.4 Reducing data to a single value

In order to see what PCA does exactly, we will first consider what would happen if we reduced the m dimensional data to a single value. Hopefully, this value will contain the most information as possible. In essence we are looking at what would happen as $k = 1$. In order to reduce each data point to a single value, many approaches exist. Take for example the approach where we simply take the euclidean norm of each point. In this case, we capture some information but we lose a lot of information. To see why consider $(0, 1)$ and $(1, 0)$. They are perpendicular, yet $\|(0, 1)\| = \|(1, 0)\|$.

There are better ways. We could for instance reduce the points by projecting them onto a single line. This is actually the first and only step in PCA. By iterating this exact step we obtain the PCA decomposition. We start by projecting onto a single line.

2.4.1 Projecting a single data point onto a fixed line

Before we project onto it, we need to define exactly what is meant with a line in the $\mathbb{R}^m$ space. One way to define a line is as follows:

Definition 5.

$$a + \mathbb{R}b \equiv \{x \in \mathbb{R}^m | \exists z \in \mathbb{R} \text{ s.t. } x = a + zb\} \tag{2.4}$$

Here, a is sometimes referred to as the offset or intercept and b is sometimes referred to as the direction of the line.

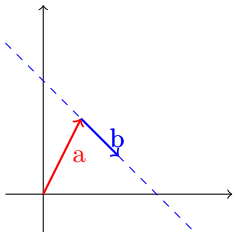


Figure 2.2: Visual of a line

Figure 2.4.1 gives an intuitive idea of what a line in $\mathbb{R}^2$ looks like. Often we want the direction to have unit length. Mathematically this means: $\|b\|_2 = 1$.

Say we want to project a point x onto a single line. There are infinite ways to do this. However, if one wants to keep as much of the information contained in the original point, we need to define an objective function. Ideally, this objective function defines how distant the projected point and the original point are. This concept should sound familiar. Just as in section 2.2, we defined the

Frobenius norm as a metric to measure how distance two matrices are. As the Frobenius norm is the natural extension of the $\mathcal{L}^2$ norm. Let's use the $\mathcal{L}^2$ norm to measure distance between two points.

We know that the projected point is somewhere on the line we are projecting to. Let's call this point $a + bz$, we know our line is fixed. This means a and b are fixed. Thus, the projected point is completely defined by z. The squared euclidean distance between the two points is then defined as:

$$\arg\min_{z \in \mathbb{R}} \|a + bz - x\|^2 \tag{2.5}$$

Figure 2.3 shows an intuitive picture of the **non squared** euclidean distance. There are

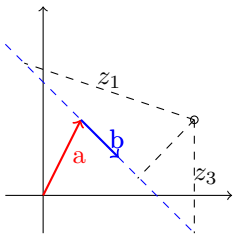


Figure 2.3: Visual of distance to a line for varying z values

It is clear to see that using the squared euclidean distance that we reach the optimal point on the line by projecting the original point orthagonally onto the line. Mathematically we can find this by deriving the squared euclidean distance in equation 2.5 for z.

$$\begin{aligned} \frac{d}{dz} \|a + bz - x\|^2 &= 2(a + bz - x)^T b = 0 \\ &= (a - x)^T b + \underbrace{b^T b}_{\|b\|^2 = 1} z = 0 \\ z &= -(a - x)^T b \end{aligned} \tag{2.6}$$

This confirms our belief that the orthogonal projection does indeed minimize the distance between the original data point and the projected data point. We see this since we are projecting onto the b direction.

To understand exactly what's happening we can have a look at figure 2.4. Here, the green arrow $(x - a)$ is being projected onto b to find the optimal z. Now that we found the z value that minimizes the loss (the projection onto the b direction), let's look at what this means mathematically:

$$x' = a + (x - a)^T b b \tag{2.7}$$

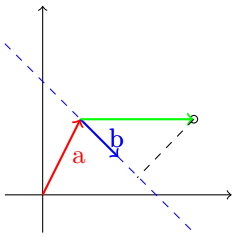


Figure 2.4: Visual of optimal z value as a projection of $(x-a)$ onto the direction.

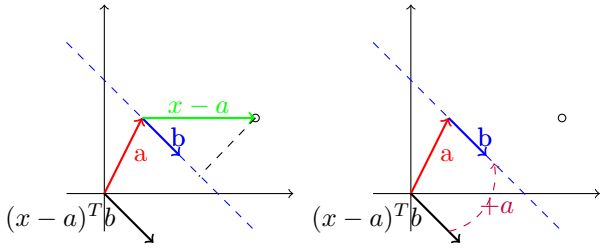


Figure 2.5: Visual of optimal z value plugged back in to find the optimal point again.

To better visualize this result, we can have a look at figure 2.5. Mathematically, the story being told by equation 2.7 is quite logical. We can see 4 steps:

1. Remove the intercept from x: $x - a$
2. Project orthagonally onto b: $(x - a)^T b$
3. Multiply in the b direction: $(x - a)^T bb$
4. Add back the intercept: $(x - a)^T bb + a$

So by minimizing the euclidean distance from a point to a line we find the orthogonal projection. This will be used extensively in the next few sections.

2.4.2 Optimal line through multiple data points

We've seen how to optimally project a single data point to a fixed line, minimizing euclidean distance. We now look at what happens when we have more than a single data point. Additionally, we don't consider the line $a+b\mathbb{R}$ to be fixed. Instead, we turn the exercise upside down. We want to answer the question: *what would be the optimal line through our data points, such that when we project all points orthagonally, we keep as much information as possible?* In line with the previous sections, we measure *information* loss using the euclidean distance.

Mathematically, we wish to answer the following. Consider n data points

$\{x_1, ..., x_n\}$ where $x_i \in \mathbb{R}^m$. How do we find the optimal line considering the mean squared euclidean loss:

$$L(a, b) = \frac{1}{n} \sum_{i=1}^{n} \|a + bz_i - x_i\| \tag{2.8}$$

Additionally we will consider $\|b\| = 1$, b is a unit length vector. This property will become important. From previous the previous section, we already know that the optimal z_i for each individual data point in known to be $z_i = (x_i - a)^T bb$. Due to the fact that one projection of a data point does not interfere with another, we can simply replace z_i in equation 2.8.

Minimizing over a and b and using previous results of equation (2.6) we find:

$$\begin{aligned}
\arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \|a + bz_i - x_i\|^2 &= \arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \left\|a + b(x_i - a)^T b - x_i\right\|^2 \\
&= \arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \left\|bb^T(x_i - a) + (a - x_i)\right\|^2 \\
&= \arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \left\|bb^T(x_i - a) - (x_i - a)\right\|^2 \\
&= \arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \left\|(bb^T - \mathbb{1}_m)(x_i - a)\right\|^2
\end{aligned} \tag{2.9}$$

Note that the $\|\cdot\|$ operation is an even function so 2.9 can be written equivalently as (we can change the sign in the brackets $\|-a\| = \|a\|$):

$$\arg\min_{a,b} \frac{1}{n} \sum_{i=1}^{n} \left\|(\mathbb{1} - bb^T)(x_i - a)\right\|^2 \tag{2.10}$$

Now what exactly does the $(\mathbb{1}_m - bb^T)$ transformation mean? Transforming a vector v out we obtain:

$$(\mathbb{1}_m - bb^T)v = v - b(b^T v) \tag{2.11}$$

Reminding ourselves that b is a unit vector, we can see it is exactly the vector we obtain when taking the difference between an original vector v and the projection of the vector $b(b^T v)$. Unsurprisingly of course, as that is exactly what we started with.

A nice thing about $(\mathbb{1} - bb^T)$ is that it is idempotent. We can show that this is clearly the case:

$$\begin{aligned}
(\mathbb{1} - bb^T)(\mathbb{1} - bb^t) &= \mathbb{1}\mathbb{1} - 2bb^T + b \underbrace{b^T b}_{b^T b = \|b\|} b^T \\
&= \mathbb{1} - bb^T
\end{aligned} \tag{2.12}$$

2.4.3 Optimal value for a

Now let's get back to finding the optimal line for given data. The first order optimality condition states that the gradient with respect to a and b should vanish when optimal. Taking the gradient with respect to a of equation 2.9 we find the following:

$$\begin{aligned}\nabla_a L(a,b) &= \frac{1}{n}\sum_{i=1}^{n}\nabla_a \left\|(\mathbb{1}-bb^T)(x_i-a)\right\|^2 \\ &= \frac{-2}{n}\sum_{i=1}^{n}(\mathbb{1}-bb^T)^T(\mathbb{1}-bb^T)(x_i-a) \\ &= \frac{-2}{n}(\mathbb{1}-bb^T)^T(\mathbb{1}-bb^T)\sum_{i=1}^{n}(x_i-a)\end{aligned} \tag{2.13}$$

In the second line we used the fact that $\nabla_x \|Ax\|^2 = 2AxA^T$, which was proven in the definitions section. We now know that since $(\mathbb{1}-bb^T)$ is clearly symmetric $(A^T = A)$ and we know that the matrix is idempotent from (2.12), thus we can write (2.13) follows:

$$\begin{aligned}\nabla_a L(a,b) = (\mathbb{1}-bb^T)\sum_{i=1}^{n}(x_i-a) &= 0 \\ (\mathbb{1}-bb^T)\sum_{i=1}^{n}(x_i) &= (\mathbb{1}-bb^T)\sum_{i=1}^{n}(a) \\ (\mathbb{1}-bb^T)\frac{1}{n}\sum_{i=1}^{n}(x_i) &= (\mathbb{1}-bb^T)a\end{aligned} \tag{2.14}$$

From (2.14) we can see that the optimality condition is met when $a = \frac{1}{n}\sum_{i=1}^{n} x_i$. This means that the optimal line will always pass through the average of all the data points. Figure 2.6 shows this visually:

Figure 2.6 shows us visually something quite interesting, since we know that the optimal line passes through the average $\bar{x}$ we could try to make the origin itself the average. We could achieve this by subtracting the average from every data point as shown in figure 2.7. This step is very often done as a pre-processing step. It makes derivations easier analytically. When "data is centered", this simply means that the empirical mean $\frac{1}{n}\sum_{i=1}^{n} x_i$ has already been subtracted and that the new mean is the origin.

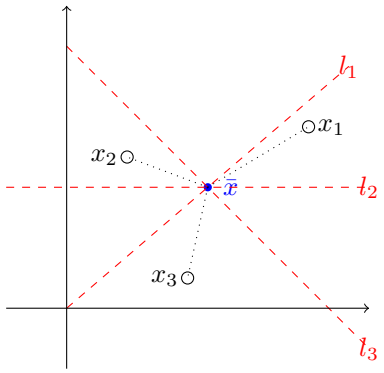


Figure 2.6: Visualisation of lines going through average of data points x_i.

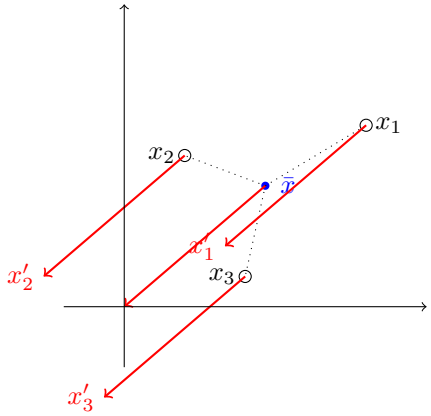


Figure 2.7: Visualisation of substracting mean of points from x_i.

2.4.4 Optimal value for b as principal eigenvectors

As figure 2.6 suggests, this of course leaves the question of *How do we choose the direction of the optimal line?* From now, we will assume the data is centered, this means that a in $a + bz$ is assumed to be 0. If we then look at the euclidean loss function as in (2.9), where $a = 0$:

$$\begin{aligned} L(b) &= \frac{1}{n}\sum_{i=1}^{n} \left\| x_i^T bb - x_i \right\|^2 \\ L(b) &= \frac{1}{n}\left(\sum_{i=1}^{n} \left\| x_i^T bb \right\|^2 + \sum_{i=1}^{n} \left\| x_i \right\|^2 - 2\sum_{i=1}^{n} (x_i^T bb)^T x_i \right) \end{aligned} \tag{2.15}$$

Rewriting $\left\|x_i^T bb\right\|^2$ we find:

$$\left\|x_i^T bb\right\|^2 = (x_i^T bb)^T x_i^T bb = (x_i^T b) \underbrace{b^T b}_{=1} b^T x_i = \left\|x_i^T b\right\|^2 \tag{2.16}$$

Further observing that $(x_i^T bb)^T x_i = (x_i^T b) b^T x_i = \left\|x_i^T b\right\|^2$, leaves us with the re-written loss function from equation 2.15:

$$L(b) = \frac{1}{n}\left(-\sum_{i=1}^{n} \left\|x_i^T bb\right\|^2 + \sum_{i=1}^{n} \|x_i\|^2 \right) \tag{2.17}$$

Aditionally since $\|x_i\|^2$ is constant it can be left out from (2.15), when minimizing for b. This results in:

$$L(b) = \frac{-1}{n} \sum_{i=1}^{n} \left\|x_i^T b\right\|^2 \tag{2.18}$$

Now we have simplified what we want to minimize, let's see what exactly is going on. Let's take the mathematical route for now:

$$\begin{aligned} b &\in \arg\min_{b,\|b\|=1} \frac{-1}{n} \sum_{i=1}^{n} \left\|x_i^T b\right\|^2 \\ b &\in \arg\max_{b,\|b\|=1} \frac{1}{n} \sum_{i=1}^{n} (x_i^T b)^T (x_i^T b) \\ b &\in \arg\max_{b,\|b\|=1} b^T \Big(\frac{1}{n} \sum_{i=1}^{n} x_i x_i^T \Big) b \\ b &\in \arg\max_{b,\|b\|=1} b^T \Sigma b \end{aligned} \tag{2.19}$$

In (2.19) we find that $\Sigma = \frac{1}{n}\sum_{i=1}^{n} x_i x_i^T$ is simply the sample covariance matrix[1]! We can now try to minimize this constrained objective. Note that we assumed that $\|b\| = 1$, we can use a Lagrange multiplier to in order to turn the constrained objective into the unconstrained objective:

$$\mathcal{L}(b, \lambda) = b^T \Sigma b + \lambda \|b\|^2 \tag{2.20}$$

Finding the extremal value of (2.20) we derive for b and equal the gradient to 0, as usual:

$$\begin{aligned} \nabla_b \mathcal{L}(b, \lambda) = (2\Sigma b - 2\lambda b) &= 0 \\ \Sigma b &= \lambda b \end{aligned} \tag{2.21}$$

So we have derived that the optimal b is actually an eigenvector of the Σ matrix. The Lagrange multiplier λ is the corresponding eigenvalue. The question still

[1] Note that we already assumed that the data is centered, thus Σ can indeed be considered the sample covariance matrix.

remains, *which eigenvector will maximize $b^T \Sigma b$?*

We can maximize $b^T X b$ when b is the eigenvector which has the highest eigenvalue. To see exactly why this is the case we look again at the Lagrangian in (2.20) and plug in the optimality condition from equation 2.21:

$$\begin{aligned} \mathcal{L}(b, \lambda) &= \lambda \underbrace{b^T b}_{=1} + \lambda \|b\|^2 \\ &= 2\lambda \end{aligned} \tag{2.22}$$

Since we are maximizing, λ should be as large as possible. The eigenvector that has the highest eigenvalue is called the *principal* eigenvector or Σ.

2.4.5 Optimal value of b as projected data variance.

We saw that the optimal direction is actually just the principal eigenvector of the covariance matrix Σ. Let's look at the loss function again from a different perspective. Let's look again at what it means to maximize $b^T \Sigma b$. From (2.19) we know that:

$$b^T \Sigma b = \frac{1}{n} \sum_{i=1}^{n} \left\| x_i^T b \right\|^2 = \frac{1}{n} \sum_{i=1}^{n} z_i^2 = \frac{1}{n} \sum_{i=1}^{n} (z_i - 0)^2 \tag{2.23}$$

Note that this is simply the empirical (biased) variance of the projected data points, since the projected mean is exactly the projection of the origin (centered data) onto the line going through the origin. Thus the projected mean is also 0. We can thus interpret this as:

$$b^T \Sigma b = Var[z] \tag{2.24}$$

So we can either interpret the objective in two ways. One as finding the principal eigenvector of the sample covariance matrix or as the maximization of the projected data. Figure 2.8 below shows this visually: Clearly the purple line has lower projected variance than the green line. Intuitively the worst case line would project all points onto the same point, losing all information (variance). The best case would maximize the the variance. Variance of the projections can then be seen as how much information is retained when projecting.

2.5 Principal Component Analysis(PCA)

We've seen the case of a single line, now we could ask ourselves, *Could we perform another projection?*. Let's take a look at the so called *residual*, which is the original data point minus the projected data point. In essence what the residuals mean is the original signal (data point) when removing all signal in the principal direction.

$$\begin{aligned} r_i := x_i - (a + b z_i) &= x_i - (x_i)^T b b \\ &= (\mathbb{1} - b b^T) x_i \end{aligned} \tag{2.25}$$

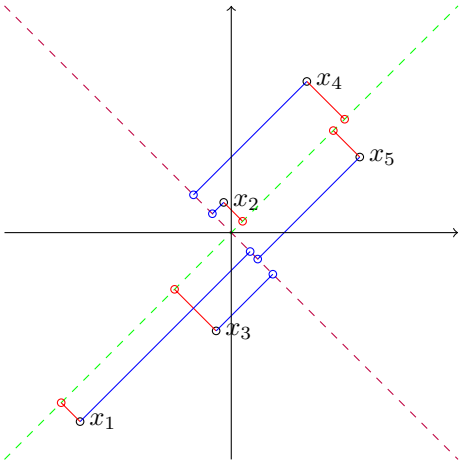


Figure 2.8: Visual of projected variance maximization.

The above result was derived using $x_i^T bb = bx_i^T b = bb^T x_i$. Figure 2.9 shows what the residuals looks like, however since there are only 2 dimensions it's somewhat less clear.

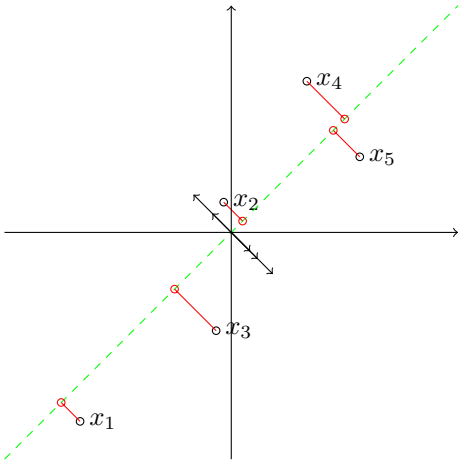


Figure 2.9: Visualisation of residuals.

2.5.1 PCA as an iterative process

What if we project the residuals to an optimal line. We simply treat the residuals as our new data. Note that the analysis to do this is exactly the same case as in previous sections. We also need the fact that the mean of the residuals is again

centered since:

$$\frac{1}{n}\sum_{i=1}^{n} r_i = (\mathbb{1} - bb^T)\frac{1}{n}\sum_{i=1}^{n} x_i = 0 \tag{2.26}$$

This means that the analysis can be done exactly the same. Like in previous section, minimizing $\sum_{i-1}^{n} \left\| r_i^T bb - r_i \right\|^2$ for b will result in having to maximize $b^T(\frac{1}{n}\sum_{i=1}^{n} r_i r_i^T)b = b^T \Sigma_r b$ for b.

We could simply find the principal eigenvector of Σ_r and do this iteratively, but maybe we could exploit a relationship to make it simpler? So what exactly is the relationship between Σ_r and Σ? We analyse this as follows using (2.25), the fact that $(\mathbb{1} - bb^T)$ is idempotent as shown in equation (2.12) and that Σ is symmetric:

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n} r_i r_i^T &= \frac{1}{n}\sum_{i=1}^{n} (\mathbb{1} - bb^T)x_i((\mathbb{1} - bb^T)x_i)^T \\
&= \frac{1}{n}\sum_{i=1}^{n} (\mathbb{1} - bb^T)x_i x_i^T(\mathbb{1} - bb^T)^T \\
&= (\mathbb{1} - bb^T)\Sigma(\mathbb{1} - bb^T)^T \\
&= (\mathbb{1} - bb^T)(\mathbb{1} - bb^T)\Sigma^T \\
&= \Sigma - b\Sigma b^T \\
&= \Sigma - \lambda bb^T
\end{aligned} \tag{2.27}$$

Note that in line 4 in equation 2.27 we used the fact that $(\mathbb{1} - bb^T)$ and Σ are symmetric, which implies $\Sigma(\mathbb{1} - bb^T) = (\mathbb{1} - bb^T)\Sigma$. We can already notice something important, b (the eigenvector of Σ) is now also an eigenvector of $(\Sigma - \lambda bb^T)$:

$$(\Sigma - \lambda bb^T)b = \Sigma b - \lambda b = 0 \tag{2.28}$$

The reverse is obviously true an eigenvector of $(\Sigma - \lambda bb^T)$ is also an eigenvector of Σ, it follows from the previous. This v is the principal eigenvector of $(\Sigma - \lambda bb^T)$ (by construction) and the second principle eigenvector of Σ, by iterating m times we find m pairwise orthogonal eigenvectors. Figure 2.9 shows visually that using the residuals to find an optimal projection will find maximum projected variance when the line is orthogonal to the green line. Another way to see why they are orthogonal is by noting that Σ is positive semi-definite ($\lambda_i > 0$) because it's symmetric so the eigenvectors are orthogonal[2].

2.5.2 PCA as a diagonalization problem

Recall that Σ is symmetric so it may be diagonalized into orthogonal matrices:

$$\Sigma = U\Lambda U^T \tag{2.29}$$

[2]say x and y are eigenvectors with λ_1 and λ_2 as eigenvalues of A then $\lambda_1 < x, y >=< Ax, y >=< x, A^T y >=< x, Ay >=< x, y > \lambda_2$ since $\lambda_1 \neq \lambda_2$ then $< x, y >= 0$.

Where $U = (u_1, u_2, \ldots, u_m)$ and $\Lambda = diag(\lambda_1, \ldots, \lambda_m)$ note that the eigenvectors are orthogonal so that:

$$U^T u_i = \begin{bmatrix} u_1^T \\ \vdots \\ u_m^T \end{bmatrix} u_i = \begin{bmatrix} u_1^T u_i \\ \vdots \\ u_m^T u_i \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ u_i^T u_i = 1 \\ \vdots \\ 0 \end{bmatrix} = e_i \tag{2.30}$$

Using this we can clearly see that:

$$\begin{aligned} U\Lambda U^T &= \begin{bmatrix} u_1 & \ldots & u_m \end{bmatrix} \begin{bmatrix} \lambda_1 & \ldots & 0 \\ \vdots & \ldots & \vdots \\ 0 & \ldots & \lambda_m \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1 u_1 & \ldots & \lambda_m u_m \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \\ &= \Sigma \underbrace{U U^T}_{\mathbb{1}} = \Sigma \end{aligned} \tag{2.31}$$

In order to then reduce the dimension of the data we would simply project our data onto the k principal components:

$$Z = U_k^T X \tag{2.32}$$

2.5.3 Colclusion: PCA

Let's recap what we did to find out what PCA is.

First we investigated that the orthogonal projection is the optimal projection of a data point onto a line considering the euclidean distance as loss measure.

Second we looked at finding the optimal line onto which we could project our data such that we keep as much information as possible. As it turned out, the optimal intercept a is exactly the mean of all the points. The optimal b turned out to be the principal component of the sample covariance matrix. Alternatively, the b direction is exactly the direction which maximizes the projected variance of the points.

Thirdly we looked at iteratively projecting the residuals onto the optimal line. This is exactly what PCA does. Another way to look at it was as a diagonalization problem of $\Sigma = U\Lambda U^T$.

Now where is the *reduction* of the data exactly? Well every point can be projected to d principle eigenvectors of Σ to give us $U^T x_i = (u_1^T x_i \ldots u_d^T x_i) =$

$(z_1 \dots z_d)$ this gives us a reduction to the optimal weights for a linear combination of the lower dimensional basis (the d principal eigenvectors). To reconstruct the data point we take a linear combination of the d principal eigenvectors. This can be done as $(z_1 \dots z_d)U$. Of course when you pick d to be the dimension of the data then there is no reduction but an exact decomposition of the data matrix X. So basically:

- Reduction:
$$Z = U^T X = \begin{pmatrix} — & u_1 & — \\ & \vdots & \\ — & u_d & — \end{pmatrix} \begin{pmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{pmatrix} = \begin{pmatrix} z_{1,1} & \dots & z_{n,1} \\ \vdots & \ddots & \vdots \\ z_{1,d} & \dots & z_{n,d} \end{pmatrix} \tag{2.33}$$
we can clearly see that we reduced the data from $\mathbb{R}^m$ to $\mathbb{R}^d$. Only if $d < m$ is there really a reduction and is there truly an approximation when reconstructing!

- Re-construction:
$$\hat{X} = UZ = \begin{bmatrix} \sum_{i=1}^d z_{1,i} u_i & \dots & \sum_{i=1}^d z_{n,i} u_i \end{bmatrix} \tag{2.34}$$
Here I wrote it out explicitly so it's easy to see that the reduction is just keeping track of weights for the linear combination of the d principal eigenvectors.

2.6 Practical PCA: Finding pricipal eigenvector

We've seen a derivation of PCA, one perspective of PCA is iteratively finding principal eigenvectors or certain symmetric matrices (empirical biased covariance matrix of the data or residuals). However we skipped the part where we actually find the principal eigenvector. One practical way of finding the principal eigenvector is the *power method*.

2.6.1 Power method: practical principal eigenvector

Assume a symmetric matrix Σ and we wish to find u_1, the principal eigenvector of Σ. The algorithm goes as follows:

1. Initialize v_0 a random vector. For simplicity we assume that $\langle u_1, v_0 \rangle \neq 0$ and that the principal eigenvalue is unique i.e. $|\lambda_1| > |\lambda_j|$.
2. Iteratively apply the following operation:
$$v_{t+1} = \frac{\Sigma v_t}{\|\Sigma v_t\|} \tag{2.35}$$
3. It holds that:
$$\lim_{t \to \infty} v_t = u_1 \tag{2.36}$$

4. To find the corresponding eignenvalue:

$$\lambda_1 = \lim_{t\to\infty} \frac{\|\Sigma v_t\|}{\|v_t\|} = \frac{\|\Sigma u_1\|}{\|u_1\|} = \frac{\|\lambda_1 u_1\|}{\|u_1\|} \tag{2.37}$$

Why does this algorithm work? We assume Σ (as in last section) is p.s.d. and symmetric (true for XX^T). In this case the eigenvectors are all orthogonal to each other so they form a basis for $\mathbb{R}^m$, this means any $v_0 \in \mathbb{R}^m$ can be written as a linear combination of eigenvectors:

$$v_0 = \sum_{i=1}^{m} a_i u_i \tag{2.38}$$

how does v_t evolve considering this?

$$v_t = \frac{\Sigma^t v_0}{\|\Sigma^t v_0\|} = \frac{\sum_{i=1}^m a_i \Sigma^t u_i}{\|\Sigma^t v_0\|} \tag{2.39}$$

Note equation (2.39) was obtained by recursively finding v_i and using (2.38). Since u_i are eigenvectors by assumption we can re-write it as follows by pulling out the highest eigenvalue:

$$\frac{a_1\lambda_1^t u_1 + \sum_{i=2}^m \frac{a_i}{a_1}(\frac{\lambda_i}{\lambda_1})^t u_i}{\|\Sigma^t v_0\|} \tag{2.40}$$

To understand the limit as $t \to \infty$ we see for the numerator:

$$\lim_{t\to\infty} a_1\lambda_1^t u_1 + \sum_{i=2}^{m} \frac{a_i}{a_1}(\underbrace{\frac{\lambda_i}{\lambda_1}}_{=0})^t u_i = a_1\lambda_1^t u_1 \tag{2.41}$$

The denomenator can be interpreted in the same way (using $\|u_1\| = 1$), cancelling the $a_1\lambda_1^t$ resulting in power iteration method converging to u_1.

2.7 Practical PCA: choosing amount of eigenvectors

How many eigenvectors should we *select* or *keep* and how should we select which ones to keep and not keep. Intuitively from previous section which looked at PCA as a variance maximisation of the projected data, we should keep the k eigenvectors which have the highest eigenvalues. It represents how much information is contained within the eigenvector for specific data.

This leaves the question, how many of the top eivenvectors should we keep? This has to be decided in a more empirical way, for example one could gather all eigenvalues and sort them and plot their value against index as follows:

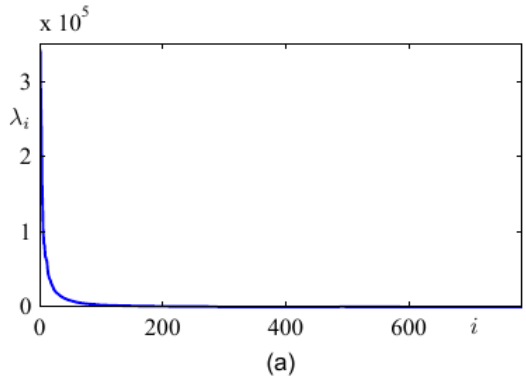


Figure 2.10: Sorted eigenvalues plotted against index

In this case it seems like taking the top 50 eigenvectors to represent a reduced dataset would be a good approximation. An engineering rule would be to cut off at *the knee* where the eigenspectrum starts to plateau rapidly.

Chapter 3

Bayesian Networks

This chapter is partially based on *A Tutorial on Learning With Bayesian Networks* and Christopher M. Bishops classic *Pattern Recognition and Machine Learning*.

In this chapter, we will go over the basics of Bayesian networks. In essence, Bayesian networks are a compact representation over a set of random variables. They are a subset of the so called Probabilistic Graphical Models (PGM) which are more commonly referred to as graphical models. Graphical models express conditional independence between variables using graphs, hence their name. Bayesian networks have become somewhat popular and used in quite a few applications. Many libraries provide easy development using Bayesian networks.

This section is structured as follows. First we delve deeper into *directed acyclic graphs* (DAGs), which is used to encode conditional independence in Bayesian networks. We look into the definition of Bayesian, which turn out to be quite simple yet elegant. A big downside to Bayesian networks, however, is the computational complexity. Thirdly, we have a look at exact inference in Bayesian networks. Then we look at approximate inference.

3.1 Conditional independence as a DAG

Graphical models are based on graphs to encode conditional independence in probability distributions. In the case of Bayesian networks conditional independence is encoded as a *directed acyclic graph*, or more commonly referred to as DAGs. DAGs are defined as follows:

Definition 6 (Directed Acyclic Graph)**.** A graph $G = (V, E)$ is called acyclic when the edges E are directed and the graph contains no directed cycles. A directed cycle is a sequence of connected edges such that at least one vertex is

visited twice.

$$\neg\exists x_1, x_2 \dots x_n \in V : (x_i, x_{i+1}) \in E \text{ and } x_1 = x_n \tag{3.1}$$

As an example figure 3.1 shows two graphs over the vertex set $V = \{A, B, C, D\}$. The left graph is a DAG as there is no vertex from which there is a sequence of directed edges which result in the same vertex. The right graph does contain a directed cycle. For example the path defined by $ACDA$.

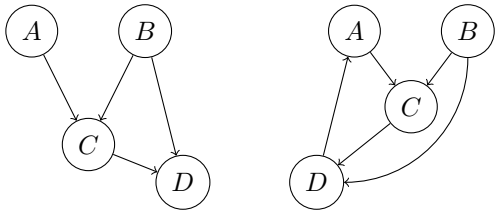


Figure 3.1: Example of a DAG (left) and a graph with a directed cycle (right).

Two important concepts in DAGs is the concept of parents, children and descendants.

Definition 7 (Parents)**.** Given a vertex $v \in V$ in a DAG $G = (V, E)$ the parents of v are all the vertices for which an outgoing edge ends in v:

$$Pa(v) = \{u \in V \mid (u, v) \in E\} \tag{3.2}$$

Definition 8 (Children)**.** Given a vertex $v \in V$ in a DAG $G = (V, E)$ the children of v are all the vertices u in which an outgoing edge of v ends.

$$C(v) = \{u \in V \mid (v, u) \in E\} \tag{3.3}$$

Definition 9 (Descendants)**.** Given a vertex $v \in V$ in a DAG $G = (V, E)$ the descendants of v are all its children and recursively the descendants of all its children.

$$Des(v) = \bigcup_{u \in C(v)} Des(u) \cup C(v) \tag{3.4}$$

Figure 3.2 shows the parents, children and descendants of node v in a DAG.

3.2 Bayesian networks with DAGs

What do these DAGs have to do with Bayesian networks? As mentioned earlier in the context of Bayesian networks they define conditional independence between variables. Let us first define Bayesian networks:

Definition 10 (Bayesian networks)**.** A Bayesian network over variables $\mathbf{X} = \{X_1, \dots X_k\}$ is a graph-probability pair (G, Θ). The graph G is a DAG which

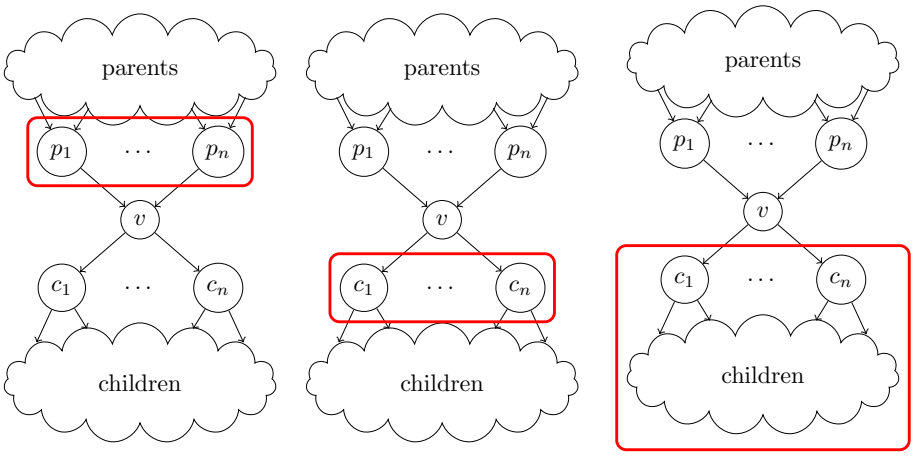


Figure 3.2: Visualization of parents $Pa(v)$, children $C(v)$ and descendants $Des(v)$ of a node v in a DAG.

encodes conditional independence. The vertices V in G are in a one-to-one correspondence with the variables $\mathbf{X}$. We do not make a distinction between the random variable $X_i \in \mathbf{X}$ and the graph node $X_i \in V$. In this definition Θ is a collection of local conditional probabilities for each variable $X_i \in \mathbf{X}$. The set Θ is defined as:

$$\Theta = \{P(X_i|Pa(X_i))|X_i \in \mathbf{X}\} \tag{3.5}$$

Together, G and Θ define a joint probability distribution over the variables $\mathbf{X}$:

$$p(\mathbf{X} = x) = \prod_{i=1}^{k} p(X_i = x_i|Pa(X_i) = x) \tag{3.6}$$

Example 2. This example was taken from Christopher M. Bishop's classic book *pattern recognition and machine learning*. Let's assume we are tasked with fitting a polynomial regression to a data set $\mathbf{x} = \{x_1, \ldots, x_n\}$ where $x_i \in \mathbb{R}^m$ and observations $\mathbf{y} = \{y_1, \ldots, y_n\}$ and wish to predict a given new point x_{n+1}. In the case of Bayesian learning, the model parameters $\mathbf{W}$ and observations $\mathbf{Y}$ will be considered as random variables. Hence, we also assume a prior over $\mathbf{W}$. This prior has one hyper parameter α and is written as $p(\mathbf{W} \mid \alpha)$. Additionally, the noise parameter σ^2 and α are considered known and fixed. The joint distribution over $\mathbf{W}$ and $\mathbf{Y}$ is then:

$$\begin{aligned} p(\mathbf{W}, \mathbf{Y} \mid \mathbf{x}, \alpha, \sigma^2) &= p(\mathbf{W} \mid \alpha, \sigma^2, \mathbf{x}) \prod_{i=1}^{n} p(Y_i \mid \mathbf{W}, \alpha, \sigma^2, \mathbf{x}) \\ &= p(\mathbf{W} \mid \alpha) \prod_{i=1}^{n} p(Y_i \mid \mathbf{W}, \sigma^2, x_i) \end{aligned} \tag{3.7}$$

Note that we sometime leave out the fixed parameters to avoid cluttering the equations too much.

$$\begin{aligned} p(\mathbf{W}, \mathbf{Y} \mid \mathbf{x}, \alpha, \sigma^2) &= p(\mathbf{W}, \mathbf{Y}) \\ &= p(\mathbf{W}) \prod_{i=1}^{n} p(Y_i \mid \mathbf{W}) \end{aligned} \tag{3.8}$$

If we map this to a Bayesian network, we obtain the network depicted in figure 3.3. Note the left figure is the same network, where the blue rectangle is shorthand notation. Everything in the blue box is duplicated N times. A vertex being filled with blue indicates that we observed this random variable. Note that α, σ^2 and x_i are not considered random variables, yet for better understanding they are still depicted.

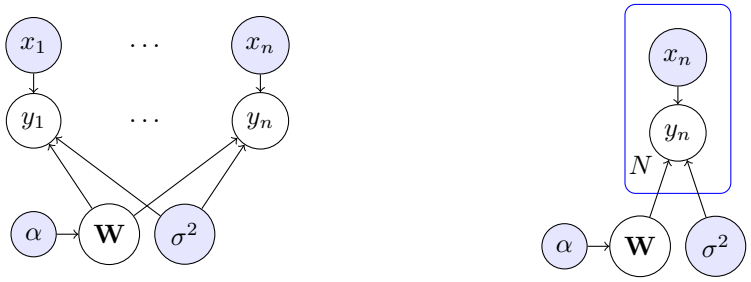


Figure 3.3: Bayesian networks for Bayesian polynomial regression. Left is long, expanded network. Right uses shorthand notation.

From the network we could then marginalize towards $\mathbf{W}$ or towards Y_i:

$$p(\mathbf{W} \mid \mathbf{Y}) = \frac{p(\mathbf{W}) \prod_{i=1}^{m} p(Y_i \mid \mathbf{W})}{p(\mathbf{Y})} \tag{3.9}$$

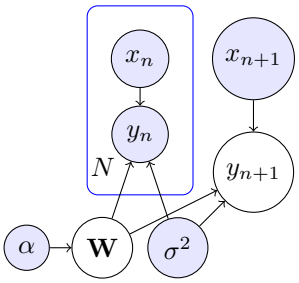
$$p(Y_i \mid \mathbf{W}, \mathbf{Y}^i) = \frac{p(\mathbf{W} \mid \alpha) \prod_{i=1}^{m} p(Y_i \mid \mathbf{W})}{p(\mathbf{W}, \mathbf{Y}^i)} \tag{3.10}$$

This network is not particularly interesting as we already have the observations $\mathbf{y} = y_1, \ldots, y_n$. In order to predict a new data point x_{n+1} we can expand our model as follows:

$$p(Y_{n+1}, \mathbf{W}, \mathbf{Y} \mid x_{n+1}) = p(\mathbf{W}) p(Y_{i+1} \mid \mathbf{W}, x_{n+1}) \prod_{i=1}^{n} p(Y_i \mid \mathbf{W}) \tag{3.11}$$

The Bayesian network corresponding with this model would be an extension of the model in figure 3.3.
To make this example more clear. We could for example assume several conditional probability distributions. Take for example $p(Y_i \mid \mathbf{W})$, in the case of polynomial regression it could for instance signify:

$$p(Y_i \mid \mathbf{W}, x_i) = \mathcal{N}\big(f(\mathbf{W}, x_i), V\big) \tag{3.12}$$



The prior over $\mathbf{W}$ could be chosen as a Gaussian with zero mean and some diagonal variance matrix:

$$p(\mathbf{W} \mid \alpha) = \mathcal{N}\big(\mathbf{0}, \alpha \mathbb{1}\big) \tag{3.13}$$

3.3 Discrete variables in Bayesian networks

We've seen Bayesian networks as defining a joint probability over a set of random variables $\mathbf{X}$. We can apply the Bayesian network principle to a set of discrete variables. In fact, Bayesian networks are usually introduced as a way to compactly define a distribution over discrete variables. The rest of this chapter, we will consider the random variables of a Bayesian network as discrete. However, keep in mind that this doesn't have to be the case.

3.3.1 Full joint probabilities

When dealing with discrete variables, one usually uses multinomial distributions. Given a discrete variable X, which can take on K values from 1 to K with probabilities $\mu = \{\mu_1, \ldots, \mu_K\}$. The probability distribution of X is given by:

$$p(X \mid \mu) = \prod_{i=1}^{K} {\mu_i}^{x_i} \tag{3.14}$$

Where x_i indicates whether X has taken value i. To define this distribution, one would need exactly $K-1$ parameters. We subtract 1 as the condition $\sum \mu_i = 1$ must hold and we lose a degree of freedom. For a single variable this is not yet a problem. However, what happens when we have a join distribution over m variables, where each variable can assume K values? In this case the probability distribution would look as follows:

$$p(X_1, \ldots, X_m) = p(X_1)p(X_2 \mid X1) \ldots p(X_m \mid X_1, \ldots, X_{m-1}) \tag{3.15}$$

For each $p(X_i \mid X_1, \ldots, X_{i-1})$ we need exactly $K^{i-1}(K-1)$ parameters to define it. We derive it mathematically:

$$\sum_{i=1}^{m} K^{i-1}(K-1) = K^m - 1 \tag{3.16}$$

The proof for this equation can be found in the definitions section at the start. In total we obtain exactly $K^m - 1$ parameters needed to fully define the joint distribution. This is of course unmanageable in any real world context. There are several ways one could combat this. One way is by imposing an independence structure in the form of Bayesian networks. Take for example a joint probability distribution over $\mathbf{X} = \{A, B, C, D\}$, each taking values in $\{1, 10\}$. This network would consist of $10^4 - 1 = 9999$ parameters. If we use for example the network in figure 3.4, we would only use $9 + 90 + 900 + 90 = 1089$ parameters.

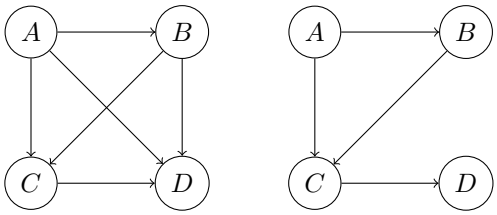


Figure 3.4: Example of fully connected Bayesian network (left) and not fully connected Bayesian network (right).

Using Bayesian networks, we can easily reduce the amount of parameters even further. For example by parameter sharing between connections, referred to as *tying* parameters. Take for example the connections $p(B \mid A)$ and $p(D \mid C)$. If we assume they share parameters we would instead have $9 + 900 + 90 = 999$ parameters. Lastly another simple method of reducing the number of parameters is by using parameterized models for example a logistic regression. As an example, take variable C in figure 3.4. We could define C as:

$$p(C \mid A, B) = \sigma\Big(w_0 + w_1 A + w_2 B\Big) \tag{3.17}$$

3.4 Conditional independence by BNs

An important property in Bayesian networks is conditional independence:

Definition 11. Two variables A and B are conditionally independent given C if and only if:

$$p(A \mid B, C) = p(A \mid C) \tag{3.18}$$

Intuitively, if we have information about C then any information we have about B is irrelevant for the outcome of A. Conditional independence shall be denoted as $A \perp\!\!\!\perp B \mid C$.

The significance of conditional independence will become clear later when looking at inference in Bayesian networks. We can represent $A \perp\!\!\!\perp B \mid C$ in a Bayesian network as depicted in figure 3.5. This is because $p(A, B \mid C) = p(A \mid C)p(B \mid C)$.
Of course, this conditional independence does not mean A and B are independent variables. If C is not given, information about A may very well influence

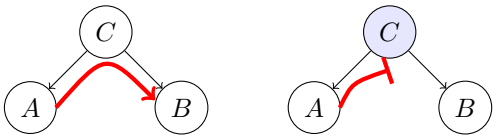


Figure 3.5: Depiction of A is independent of B given C. Left, the information can flow from A to B, because C is unknown. Right the information is blocked, because C is known.

the outcome of B. We notice that information *travels* from A to B if C is unknown. Conversely, information in *blocked* from A to B if C is known.

3.5 D-separation

In this section we have a closer look at the previous observation. How does information *travel* throughout the Bayesian net given specific observations? Which variables are conditionally independent from each other given a certain set of other variables? Is there an easy way to verify these by using the DAG?

To investigate these questions further, we first look at the three building blocks in Bayesian networks. Each of the building blocks are Bayesian networks with exactly three nodes. For each of them we will observe how information travels from one node to another. From there, we can define a notion of conditional independence between groups of variables in more complex Bayesian networks.

3.5.1 Building block: one

This 3 node network has already been discussed. Namely the network in figure 3.5. We've already mentioned that C will *block* information flow from A to B if it is given. We will denote this type of path as $A \leftarrow C \rightarrow B$.

Example 3. Consider $A =$ whether or not you have a runny nose. $B =$ Whether or not you have a fever. $C =$ whether or not you have the flu. Clearly, if you know you have the flu $C = 1$, then whether or not you have a runny nose $A = 1$ is independent from whether or not you have a fever $B = 1$. However, if C is unknown, one might expect to have a higher chance of a runny nose ($A = 1$) if you already have a fever ($B = 1$).

We can verify that, in general $A \not\perp\!\!\!\perp B$:

$$\begin{aligned} p(A, B) &= \sum_c p(A, B \mid c) p(c) \\ &= \sum_c p(A \mid c) p(B \mid c) p(c) \end{aligned} \tag{3.19}$$

This is of course not equal to $p(A)(B)$ in the general case. However, with information about C we do find independence:

$$\begin{aligned} p(A, B \mid c) &= \frac{p(c)p(A \mid c)p(B \mid c)}{p(c)} \\ &= p(A \mid c)p(B \mid c) \end{aligned} \tag{3.20}$$

We conclude that $A \not\perp\!\!\!\perp B$ and $A \perp\!\!\!\perp B \mid C$.

3.5.2 Building block: two

The second building block is shown in figure 3.6. In this case we find a very similar situation as building block one. If C is known, information is *blocked* from A to B. However, information flows freely from A to B if C is unknown. We will denote this type as $A \to C \to B$.

Figure 3.6: The second building block for D-separation. Left, the information can flow from A to B, because C is unknown. Right the information is blocked, because C is known.

Example 4. Consider $A =$ whether you are late for the bus. $C =$ whether you are late for the train and $B =$ whether you are late for your work. Clearly, if you are late for the train ($C = 1$), being late for for the bus ($A = 1$) will not change whether or not you are late for work ($B = 1$). Conversely, if you were already late for the bus ($A = 1$) you will most likely be late for the train as well ($C = 1$). Eventually causing you to be late for your work ($B = 1$). Hence, information flows from A through C to B.

Mathematically, we verify that in the general case, A and B are not independent. I.e. $A \not\perp\!\!\!\perp B$:

$$\begin{aligned} p(A, B) &= \sum_c p(A)p(c \mid A)p(B \mid c) \\ &= p(A) \sum_c p(c \mid A)p(B \mid c) \\ &= p(A)p(B \mid A) \end{aligned} \tag{3.21}$$

In general, $p(B \mid A) \neq p(B)$. In the case C assumes a known value, A and B

are conditionally independent. We verify:

$$\begin{aligned} p(A, B \mid c) &= \frac{p(A, B, c)}{p(c)} \\ &= \frac{p(A)p(c \mid A)p(B \mid c)}{p(c)} \\ &= \frac{p(A)p(c \mid A)}{p(c)} p(B \mid c) \\ &= p(A \mid c)p(B \mid c) \end{aligned} \tag{3.22}$$

We conclude $A \not\perp\!\!\!\perp B$ but $A \perp\!\!\!\perp B \mid C$.

3.5.3 Building block: three

The final building block for d-separation can be seen in figure 3.7. In this three node Bayesian network we observe the opposite. If C is not observed, the information is blocked by C. However, if C is known, information can flow from A to B. This type will be denoted as $A \rightarrow C \leftarrow B$

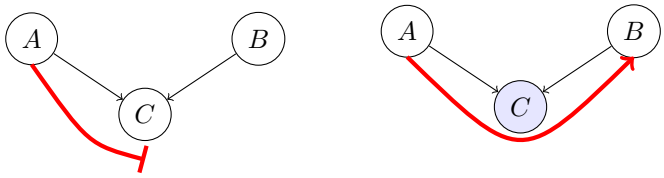


Figure 3.7: The third building block for D-separation. Left, the information is blocked from A to B, because C is unknown. Right the information can flow from A to B, because C is known.

Example 5. Consider A = whether or not your car break fails. B = whether or not your car motor has failed. Finally, consider C = whether or not you had a car accident. If you don't know whether or not you have a car crash (C unknown), then gaining information about your motor ($B = 1$) will not change the probability of your breaks failing ($A = 1$). However, assume now that we know that we know that we had a car crash ($C = 1$). Clearly if we know it wasn't caused by motor failure ($B = 0$), then it was most likely caused by break failure ($A = 1$). Hence, information flows from B to A since we know C.

Let's verify that information gets blocked if C is unknown:

$$\begin{aligned} p(A, B) &= \sum_c p(A)p(B)p(c \mid A, B) \\ &= p(A)p(B) \end{aligned} \tag{3.23}$$

We now show the converse. Given C, information can freely flow from A to B:

$$\begin{aligned} p(A, B \mid c) &= \frac{p(A, B, c)}{p(c)} \\ &= \frac{p(A)p(B)p(c \mid A, B)}{p(c)} \end{aligned} \tag{3.24}$$

In general $A \not\perp\!\!\!\perp B \mid C$.

3.5.4 D-separation

So why did we consider these three graphs in the first place? Usually our Bayesian networks do not consist out of three random variables. Instead, networks usually consist of thousands of random variables. D-separation is a useful tool to graphically verify whether two variable sets A and B are conditionally independent given a set of observations C.

Definition 12 (D-separation)**.** Given a Bayesian network (G, Θ) over variables $\mathbf{X}$. Two variable sets $A \subset \mathbf{X}$ and $B \subset \mathbf{X}$ are D-separated given observed variable set $C \subset \mathbf{X}$ iff for each undirected path $P = P_1 \dots P_n$ from set A to set B, the information flow is *blocked*. Either by an observed variable $O \in C$ or by a missing observed variable $O \notin C$. Concretely, for each sub-path $P' = P_i P_{i+1} P_{i+2} \in P$:

- P' is of type $P_i \leftarrow P_{i+1} \rightarrow P_{i+2}$ and $P_{i+1} \in C$ OR
- P' is of type $P_i \rightarrow P_{i+1} \rightarrow P_{i+2}$ and $P_{i+1} \in C$ OR
- P' is of type $P_i \rightarrow P_{i+1} \leftarrow P_{i+2}$ and $(Des(P_{i+1}) \cup \{P_{i+1}\}) \cap C = \emptyset$

Property 6 (D-separation implies conditional independence)**.** Given a Bayesian network (G, Θ) over random variables $\mathbf{X}$. If variable set $A \subset \mathbf{X}$ and $B \subset \mathbf{X}$ are D-separated given $C \subset \mathbf{X}$, then it holds that $A \perp\!\!\!\perp B \mid C$ in the joint distribution.

$$Dsep(A, B \mid C) \implies A \perp\!\!\!\perp B \mid C \tag{3.25}$$

While a proof would justify this property, intuitively it makes a lot of sense. By the property of information flow in the three building blocks (and symmetry of conditional independence) we can see how this property makes sense. As an example, let's look at the Bayesian network in figure 3.8. On the left we find all paths from A to D blocked. Thus we can state $Dsep(A, D \mid \{C, B\})$. However, on the right information can travel through $A \rightarrow C \leftarrow B$, $C \leftarrow B \rightarrow D$. Thus we cannot state $Dsep(A, D \mid C)$.

3.5.5 Markov blanket

Often, one is interested in the conditional probability of a certain variable X in a Bayesian network (G, Θ), given all other variables. As one might expect, we can exploit the D-separation property to avoid redundant computation and

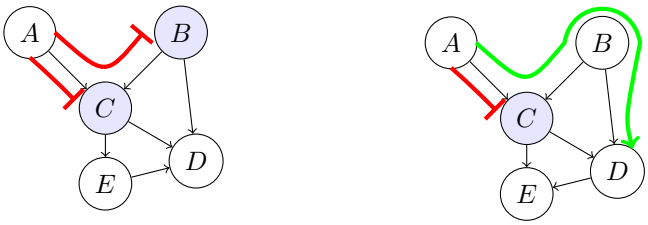


Figure 3.8: Example of D-separation in a Bayesian network. On the left, all paths between A and D are blocked. On the right, there is a path $ACBD$ thus no independence.

reduce complexity. Indeed, if all variables in the network are given except for X we find a Bayesian network as in figure 3.9. We observe through D-separation that information flow is blocked from X to all other nodes except the children, parents and parents of children.

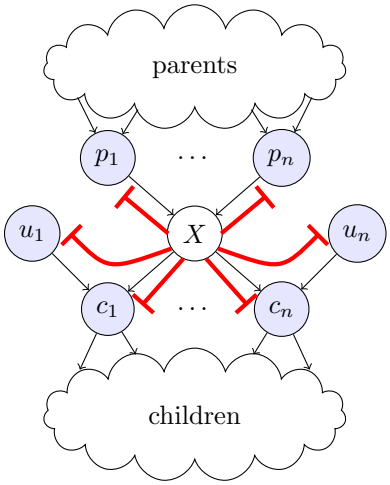


Figure 3.9: Example of Markov blanket. The Markov blanket consists of children, parents and parents of children. Given the Markov blanket, all information flow is blocked.

Definition 13 (Markov blanket)**.** The Markov blanket $B_m(X_i)$ of a node X_i in a Bayesian network (G, Θ) over random variables $\mathbf{X} = \{X_1, \ldots, X_k\}$ is defined as:

$$B_m(X_i) = \left(\bigcup_{u \in C(X_i)} Pa(u) \right) \cup Pa(X_i) \cup C(X_i) \setminus \{X_i\} \tag{3.26}$$

A nice property of the Markov blanket is the following:

Property 7. Given a Bayesian network (G, Θ) over $\mathbf{X} = \{X_1, \ldots, X_k\}$, then the conditional probability distribution of X_i given all other variables $\mathbf{X} \setminus \{X_i\}$ is equal to the conditional distribution of X_i given the Markov blanket.

$$p(X_i \mid X_{j \neq i}) = p(X_i \mid M_b(X_i)) \tag{3.27}$$

Proof. The proof is straight forward:

$$\begin{aligned} p(X_i \mid X_{j \neq i}) &= \frac{p(\mathbf{X})}{p(X_{j \neq i})} \\ &= \frac{\prod_{l=1}^{k} p(X_l \mid Pa(X_l))}{\int_{X_i} \prod_{l=1}^{k} p(X_l \mid Pa(X_l)) dX_i} \end{aligned} \tag{3.28}$$

In the denominator, we can pull out any terms which do not contain X_i. We define the following sets for ease of notation:

$$\begin{aligned} I_b &= C(X_i) \cup \{X_i\} \\ N_b &= V \setminus I_b \end{aligned} \tag{3.29}$$

$$\begin{aligned} S &= \bigcup_{v \in I_b} Pa(v) \\ S' &= \bigcup_{v \in N_b} Pa(v) \end{aligned} \tag{3.30}$$

We can then re-write the previous equation:

$$\begin{aligned} p(X_i \mid X_{j \neq i}) &= \frac{\prod_{l=1}^{k} p(X_l \mid Pa(X_l))}{\int_{X_i} \prod_{l=1}^{k} p(X_l \mid Pa(X_l)) dX_i} \\ &= \frac{\prod_{u \in N_b} p(u \mid Pa(u)) \prod_{v \in I_b} p(v \mid Pa(v))}{\prod_{u \in N_b} p(u \mid Pa(u)) \int_{X_i} \prod_{v \in I_b} p(v \mid Pa(v)) dX_i} \\ &= \frac{p(N_b \mid S' \setminus N_b)}{p(N_b \mid S' \setminus N_b)} \frac{p(I_b \mid S \setminus I_b)}{\int_{X_i} p(I_b \mid S \setminus I_b) dX_i} \\ &= \frac{p(I_b \mid S \setminus I_b)}{\int_{X_i} p(I_b \mid S \setminus I_b) dX_i} \\ &= p(X_i \mid S \setminus I_b \cup I_b \setminus \{X_i\}) \\ &= p(X_i \mid S \cup I_b \setminus \{X_i\}) \end{aligned} \tag{3.31}$$

We are left to prove that $B_m(X_i) = S \setminus \{X_i\}$:

$$\begin{aligned} S \cup I_b \setminus \{X_i\} &= \left(\bigcup_{v \in C(X_i) \cup \{X_i\}} Pa(v) \right) \cup I_b \setminus \{X_i\} \\ &= \left(\bigcup_{v \in C(X_i)} Pa(v) \cup Pa(X_i) \right) \cup C(X_i) \cup \{X_i\} \setminus \{X_i\} \\ &= \left(\bigcup_{u \in C(X_i)} Pa(u) \right) \cup Pa(X_i) \cup C(X_i) \setminus \{X_i\} \end{aligned} \tag{3.32}$$

We conclude that $p(X_i \mid X_{j \neq i}) = p(X_i \mid M_b(X_i))$. □

3.6 Constructing Bayesian networks

Talk about variable ordering, give an example.

3.7 Exact inference in Bayesian networks

We've seen Bayesian networks and argues their usefulness. In the previous sections on conditional independence, D-separation and Markov blankets we've mentioned that these properties are very useful in performing inference. Especially the Markov blanket, which suffices to compute the conditional probability distribution for any variable X_i. This reduces computation time considerably. However, it remains a # P-complete problem. We are after all, summing over an exponential number of configurations.

Several exact inference techniques exists. One of them makes use of dynamic programming and is referred to as variable elimination. Here, one makes smart use of previous computations to avoid redundant computations. Variable elimination is the most straight forward exact inference algorithm.

The second form of exact inference we will discuss in this section is one based on message passing. As we will see, this algorithm is efficient and exact for a subset of Bayesian networks. While the algorithm works for general Bayesian networks, it is not efficient in the general case. This algorithm is referred to as the *sum-product* algorithm or *Belief propagation*. Before we go into sum-product, however, we need to first consider factor graphs. Factor graphs are a graphical representation of a factorization of a function over multiple variables.

3.7.1 Variable elimination

Variable elimination is in a sense just a fancy name for dynamic programming and caching computation in Bayesian networks. Let's look at an example of finding the marginal probability of node E in the Bayesian network in figure 3.8:

$$\begin{aligned} p(E) &= \sum_{a,b,c,d} P(a,b,c,d,E) \\ &= \sum_{a,b,c,d} p(a)p(b)p(c \mid a,b)p(d \mid c,b)p(E \mid c,d) \end{aligned} \tag{3.33}$$

If we were to calculate the probability in this fashion, we would take a sum over $|A||B||C||D| = k^4$ values. Here $|X|$ denotes the amount of values X can assume, i.e. $|\Omega_X|$. We assume the outcome space of each variable to be k. Clearly, in larger networks this would not scale. We can, however, massage the equation a

bit to win in computation time. We demonstrate as follows:

$$\begin{aligned} p(E) &= \sum_{a,b,c,d} p(a)p(b)p(c \mid a,b)p(d \mid c,b)p(E \mid c,d) \\ &= \sum_{b,c,d} p(b)p(d \mid c,b)p(E \mid c,d) \sum_{a} p(a)p(c \mid a,b) \\ &= \sum_{c,d} p(E \mid c,d) \sum_{b} p(b)p(d \mid c,b) \sum_{a} p(a)p(c \mid a,b) \end{aligned} \tag{3.34}$$

You might be wondering why doing this is useful at all. Let's investigate further. The summation over A is a sum over k variables. Next we sum over B, which is again a summation over k variables. Finally, we sum over k^2 values. In total we have summed over $2k + k^2$ values. While it might not seem like much, this is a considerable amount of savings in larger Bayesian networks. In order to understand better what's happening exactly, figure 3.10 $\sum_a p(a)(c \mid a,b)$ assuming that each variable can take values of 0 and 1.

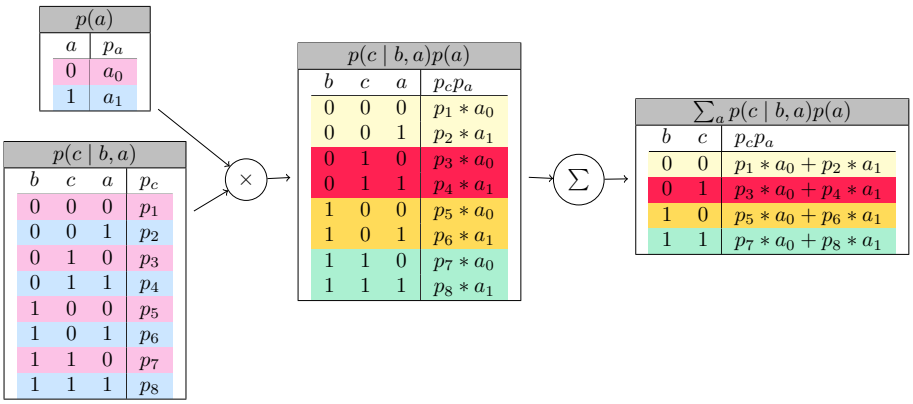


Figure 3.10: Visualization of $\sum_a p(a)p(c \mid b,a)$.

In fact, $\sum_a p(a)p(c \mid b,a)$ is very similar as a local conditional probability. It consists of table, where each possible configuration of b and c are mapped to a value. While the local conditional probabilities sum up to 1, it does not have to be the case for $\sum_a p(a)p(c \mid b,a)$. We will denote $\sum_a p(a)p(c \mid b,a)$ as $f_1(b,c)$ and continue:

$$\begin{aligned} p(E) &= \sum_{c,d} p(E \mid c,d) \underbrace{\sum_{b} p(b)p(d \mid c,b)f_1(b,c)}_{f_2(d,c)} \\ &= \sum_{c,d} p(E \mid c,d) f_2(d,c) \\ &= f_3(E) \end{aligned} \tag{3.35}$$

By recursively pushing the sum into the product we create a computation tree as in figure 3.11.

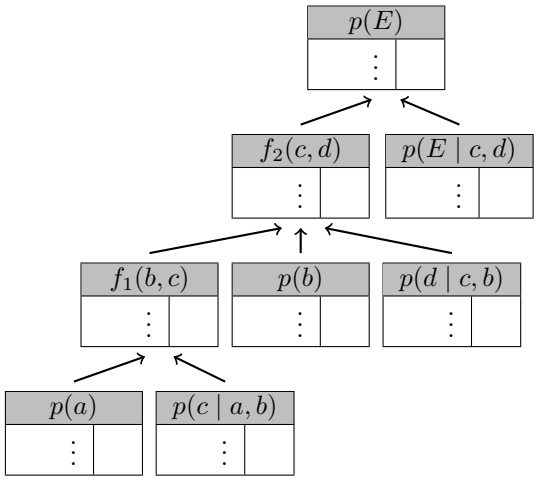


Figure 3.11: The computation tree for variable ordering a, b, c, d.

From equation 3.34 one might observe that this is not the only way one could *push* the sum into the product. In the case of equation 3.34 the order in which we pushed the sum into the product is a, b then c and d. Note that this variable order might not be optimal to reduce the computational time. In fact, the computation tree from figure 3.11 might look wildly different depending on the variable order. A good one might reduce complexity tremendously, a bad ordering might result in no gain.

3.7.2 Factor graphs

In this section, we introduce factor graphs. Factor graphs are very similar to Bayesian networks, in the sense that they are a graphical representation of a factorization of a function. In the case of Bayesian networks, we factorize $p(\mathbf{X})$ using local conditional probabilities $p(X_i \mid Pa(X_i))$. In Bayesian networks, each node in the DAG had exactly one local conditional probability associated with it. In factor graphs, however, we have a set $F = \{f_1, \ldots, f_k\}$ of factors. Each factor f_i is defined over a subset of the variables in the original function.

Definition 14 (Factor graph)**.** A factor graph over $\mathbf{X} = \{X_1, \ldots, X_n\}$ for function $f(\mathbf{X})$ is a set of factors $F = \{f_1, \ldots, f_k\}$, where each factor f_i is defined for a subset of $\mathbf{X}$. For a factor graph it holds that:

$$f(\mathbf{X}) = \prod_{f_i \in F} f_i(\mathbf{X}_i) \tag{3.36}$$

A factor graph is visually represented as a undirected bipartite graph $G = (V, E)$ consisting of two node types as in figure 3.12 on the right. The vertices are

partitioned into factor nodes F and variable nodes $\mathbf{X}$, $V = \mathbf{X} \cup F$. Factor nodes are depicted by a red square and variable nodes depicted by a black circle containing the variable. Variable nodes only have edges to factor nodes and vice versa. A factor node is connected to a variable node if the factor includes this variable:

$$\forall X \in \mathbf{X}, \forall f_i \in F : (X, f_i) \in E \Leftrightarrow X \in \mathbf{X}_i \tag{3.37}$$

As factor graphs are undirected, the edge set E is symmetric. This means that if there is an edge from X to f_i there is also an edge from f_i to X:

$$\forall v, u \in V : (v, u) \in E \Leftrightarrow (u, v) \in E \tag{3.38}$$

In this definition, $\mathbf{X}_i$ denotes the subset of $\mathbf{X}$ over which f_i is defined. Another similarity with Bayesian networks is the encoding of conditional independence. Indeed, conditional independence in Bayesian networks is encoded by a DAG. By the fact that the factors f_i are defined over a subset of $\mathbf{X}$, we implicitly leverage structure of our function $f(\mathbf{X})$.

Example 6. Let us consider the joint probability function defined by the Bayesian network in figure 3.8. Figure 3.12 on the left shows this Bayesian network. We can re-write the Bayesian network over $\mathbf{X} = \{A, B, C, D, E\}$ as a factor graph. Indeed, consider $p(\mathbf{X}) = f(\mathbf{X})$, we factorize:

$$\begin{aligned} p(A, B, C, D, E) &= p(A)p(B)p(C \mid A, B)p(D \mid C, B)p(E \mid C, D) \\ &= f_1(A)f_2(B)f_3(A, B, C)f_4(B, C, D)f_5(C, D, E) \end{aligned} \tag{3.39}$$

How trivial this derivation seems, it shows that any Bayesian network (G, Θ) is already by default a factor graph where $F = \Theta$ for the join distribution $p(\mathbf{X})$. However, not every factor graph is a Bayesian network. Looking at 3.39 we notice that this is not the only way we could transform the Bayesian network into a factor graph. Take for example:

$$\begin{aligned} p(A, B, C, D, E) &= \underbrace{f_1(A)f_2(B)f_3(A, B, C)}_{=f_1(A,B,C)} f_4(B, C, D)f_5(C, D, E) \\ &= f_1(A, B, C)f_4(B, C, D)f_5(C, D, E) \end{aligned} \tag{3.40}$$

This is also a factor graph over $\mathbf{X}$ for $p(\mathbf{X})$. Thus, there are many factor graphs for a given Bayesian network, depending on which factors are taken together. We draw factor graphs in a very similar fashion as Bayesian networks. Figure 3.12 shows the Bayesian network transformed into a factor graph.
The factor graph as seen in figure 3.12 is a bipartite graph. The factors do not have edges between other factors. The same holds for variable nodes. This will play an important role in the sum-product algorithm.

We conclude the section on factor graphs by defining what is meant by the neighbors of a node. In the sum-product algorithm the neighborhood plays an important role.

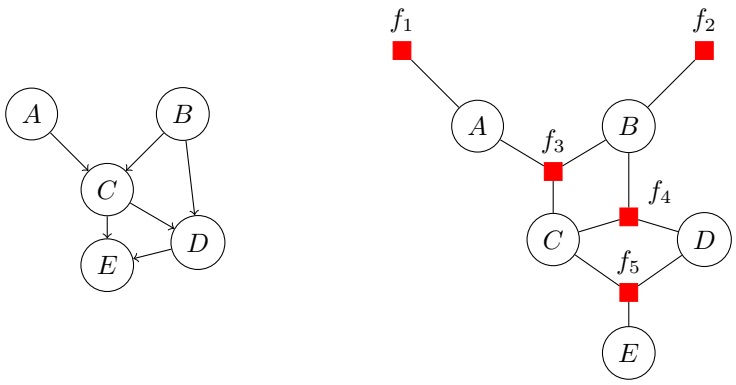


Figure 3.12: Example of a Bayesian network transformed into a factor graph.

Definition 15 (Neighbors in a factor graph)**.** Given a factor graph $G = (V, E)$, the neighbors $N(X)$ of a node $X \in V$ are defined as all the nodes for which there is a connecting edge:

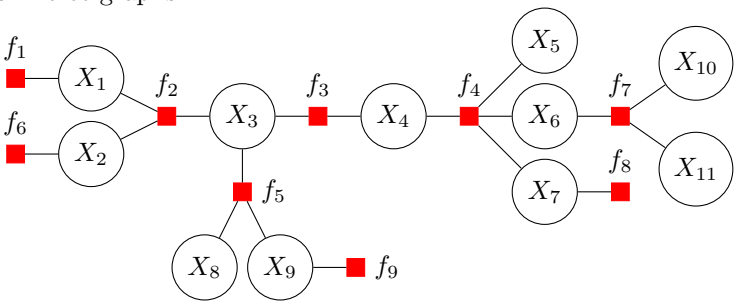
$$N(X) = \{v \in V \mid (v, X) \in E\} \tag{3.41}$$

3.7.3 Sum-product

We will discuss the sum-product algorithm in context of Bayesian networks over discrete variables. However, the algorithm does generalize to continuous variables.

We start our discussion of the sum-product algorithm by applying it to a subset of factor graphs. The types of factor graphs we will consider at first are the tree graphs. These factor graphs do not contain loops. An example of a tree structured factor graph is depicted in figure 3.13. Note this tree graph is not necessarily derived from a Bayesian network. This graph will be the running example throughout this section.

Figure 3.13: The running example of a tree structured factor graph. There are no loops in tree graphs.



Let us consider the problem of finding the marginal probability of a variable $X \in \mathbf{X}$, not considering evidence for now. Mathematically, we want to sum out all the other variables:

$$p(X) = \sum_{\mathbf{X} \setminus X} f(\mathbf{X}) \tag{3.42}$$

We can re-write $f(\mathbf{X})$ as the product of all the factors in the factor graph, by definition of factor graphs. However, let us first partition the factor set F. We partition the factors by considering X as the root node. The partition is then formed by the factor nodes contained in each sub-tree. To be more concrete, consider $S(X) = \{G_1, \ldots, Gm\}$ the set of sub trees $G_i = (V_i \cup F_i, E_i)$ starting at neighbor $f_i \in N(X)$. Figure 3.14 shows what this means for general factor graphs. We then partition the set of factors F into the factor sets F_i contained in each sub-tree G_i.

$$\begin{aligned} \bigcup_{G_i \in S(X)} F_i &= F \\ \bigcap_{G_i \in S(X)} F_i &= \emptyset \end{aligned} \tag{3.43}$$

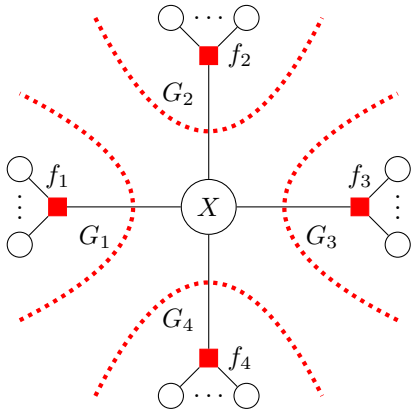


Figure 3.14: Visualization of what is meant by $S(X)$ and F_i. Note that G_i is **not** a factor graph. This would require $X \in F_i \cup V_i$.

Example 7. Take for example X_3 as the root for the factor graph in figure 3.13. We obtain the partitions $F = \{f_1, f_2, f_6\} \cup \{f_5, f_9\} \cup \{f_3, f_4, f_7, f_8\}$.

Using the partition of F, we re-write $f(\mathbf{X})$ from equation 3.42:

$$
\begin{aligned}
f(\mathbf{X}) &= \prod_{f_i \in F} f_i(\mathbf{X}_i) \\
&= \prod_{G_i \in S(X)} \prod_{f_i \in F_i} f_i(\mathbf{X}_i) \\
&= \prod_{G_i \in S(X)} F_i(X, V_i)
\end{aligned}
\tag{3.44}
$$

Where $F_i(X, V_i)$ is defined as the product of all factors in the set F_i. Recall that V_i is the set of vertices of sub graph G_i. We now plug in this result into equation 3.42 and push the sum into the product:

$$
\begin{aligned}
p(\mathbf{X}) &= \sum_{\mathbf{X} \setminus X} \prod_{G_i \in S(X)} F_i(X, V_i) \\
&= \prod_{G_i \in S(X)} \sum_{V_i} F_i(X, V_i) \\
&= \prod_{G_i \in S(X)} \mu_{f_i \to X}(X)
\end{aligned}
\tag{3.45}
$$

Where we define $\mu_{f_i \to X}(X)$ as the sum $\sum_{V_i} F_i(X, V_i)$, they are referred to as *messages* from factor f_i to variable X. The marginal $p(X)$ is then defined as the product of all incoming messages as displayed in figure 3.15. Note, the sum may be pushed into the product because the sub-graphs $S(X)$ have no overlapping vertices.

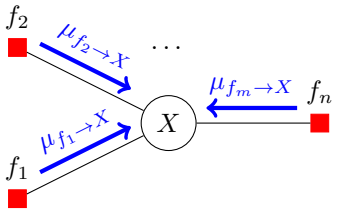


Figure 3.15: Messages passed from factor nodes f_i to a variable node X.

3.8 Approximate inference in Bayesian networks

3.8.1 Sampling methods

Chapter 4

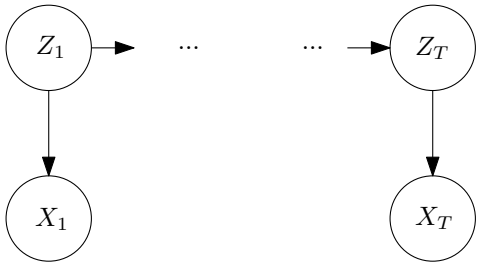
Hidden Markov Models

Talk about HMM use cases, sequential data, text processing, ... a bit outdated maybe.

4.1 notation

- $\mathcal{Z} = \{Z_1, ..., Z_T\}$: A random vector representing the categories, e.g. "verb".
- $\mathcal{X} = \{X_1, ..., X_T\}$: A random vector representing the words, e.g. "work".
- $\mathbf{X} = \{x_1, ..., x_T\}$: A **non-random** vector of **observations**, e.g a full scentence, note that $\mathbf{X} \in O^T$.
- $\mathbf{Z} = \{z_1, ..., z_T\}$: A **non-random** vector of **categories**, e.g a full scentence, note that $\mathbf{Z} \in Q^T$.
- $Q = \{q_1, ..., q_K\}$: The collection of values Z_i can take. Define $|Q| = K$.
- $O = \{o_1, ..., o_L\}$: The collection of words X_i can take i.e. the corpus. Define $|O| = L$.
- $A_{i \to j} = P(Z_t = q_j | Z_{t-1} = q_i)$: transition probability of going from category i to category j. e.g. "Verb" $\to$ "Noun". Independent of t since it's a markov chain.
- $B_{i \to j} = P(X_t = o_j | Z_t = q_i)$: emission probability of emitting word o_j if you're in category i. e.g. "Verb" $\to$ "work". Independent of t since it's a markov chain.
- θ: collection of parameters $\theta = \{A_{ij,B_{ij}}, \pi\} \forall i, j$ here π is the distribution of Z_1. i.e. Which "Adverb" is more likely as the category for the first word than "Verb".

Any hidden markov model can be represented as follows:



4.2 Maximum likelihood for HMM

4.2.1 General principle of maximum likelihood

Given data $\mathcal{X}$ that you assume comes from a probabilistic model $D \sim P(X|\theta)$. Maximum likelihood is a general principle in which you try to find parameters θ such that the data is "most likely" to occur. This means we find:

$$\hat{\theta} = \arg\max_{\theta} P(D|\theta) \tag{4.1}$$

However it is ofter easier to optimize the the log(.) of the likelihood function. Since log(.) is monotonically increasing minimizing this is by defninition identical, although often easier computationally.

$$\hat{\theta} = \arg\max_{\theta} P(D|\theta) = \arg\max_{\theta} \log(P(D|\theta)) \tag{4.2}$$

I will denote the log liklihood as $L(D|\theta)$.

Example: Intuitively if we flip a coin 100 times and find 20 heads(=0) and 80(=1) tails. The parameter of this Bernouilli variable will more likely be higher than $p = 0.5$.

4.2.2 MLE for HMM

Likelihood of HMM

Likelihood of a Hidden Markov Model is found by "pluggin in" bayesian network:

$$P(\mathcal{Z}, \mathcal{X}|\theta) = \prod_{t=0}^{T-1} \underbrace{P(Z_{t+1}|Z_t)}_{transition} \underbrace{P(X_{t+1}|Z_{t+1})}_{emission} \tag{4.3}$$

Where $P(Z_1|Z_0) = P(Z_1)$, is the prior of the bayes net. For hidden markov

models we have addictional constraints on the parameters namely:

$$\begin{aligned}\sum_{j=1}^{K} p(Z_{t+1} = q_j | Z_t = q_i) = \sum_{j=1}^{K} A_{i\to j} = 1 \quad \forall i \\ \sum_{j=1}^{L} p(X_t = o_j | Z_t = q_i) = \sum_{j=1}^{L} B_{i\to j} = 1 \quad \forall i\end{aligned} \tag{4.4}$$

Meaning that for every category in our set Q the total probability of going to any state is 1 and the total probability of seeing any word should be 1.

Small comment

For HMM we can try to apply the maximum likelihood estimate. Since the likelihood for a HMM is given in terms of the observations $\mathbf{X} = \{x_1, ..., x_T\}$ and the known categories $\mathbf{Z} = \{z_1, ..., z_T\}$, in order to perform maximum likelihood we need both of these. This would for example mean that someone would categorise every word in a book before estimating the parameters, which some might consider impossible. However for now we assume this is no problem and continue.

Actual calculation

Almost all ML methods we will be maximizing the log-likelihood instead, note that in the following we no longer assume no data but instead assume observations $\mathbf{X}$ and categories for these $\mathbf{Z}$:

$$\begin{aligned}\hat{\theta} =& \arg\max_{A,B,\pi} \log(\prod_{t=0}^{T-1} P(Z_{t+1} = z_{t+1} | Z_t = z_t) P(X_{t+1} = x_{t+1} | Z_{t+1} = z_{t+1})) \\ =& \arg\max_{A,B,\pi} \sum_{t=0}^{T-1} log(P(Z_{t+1} = z_{t+1} | Z_t = z_t)) + \sum_{t=0}^{T-1} \log(P(X_{t+1} = x_{t+1} | Z_{t+1} = z_{t+1})) \\ =& \arg\max_{A,B,\pi} \sum_{t=0}^{T-1} log(A_{t\to t+1}) + \sum_{t=0}^{T-1} \log(B_{t\to t})\end{aligned} \tag{4.5}$$

Note the notation it quite confusing at this point, however $A_{t\to t+1}$ is the probability of going from **observed** category at step t in the chain to what we **observed** in step $t+1$. Same goes for $B_{t\to t}$.

So what's next? We know the parameters are optimal if the gradient w.r.t. the parameters is 0, i.e. $\nabla L(\mathbf{X}, \mathbf{Z}|\theta) = 0$. However if we do this we overlooked a key component which are the extra constraints we placed on the parameters. A quick look at whet we maximize shows us that setting all parameters to ∞

is optimal, however clearly doesn't result in a proper transition or emission distribution. To include the constraints we use lagrange multipliers[1]:

$$\mathcal{L}(\theta,\lambda,\alpha) = \sum_{t=0}^{T-1} log(A_{t\to t+1}) + \sum_{t=0}^{T-1} \log(B_{t\to t}) + \sum_{i}^{K} \lambda_i (1 - \sum_{j}^{K} A_{i\to j}) + \sum_{i}^{K} \alpha_i (1 - \sum_{j}^{L} B_{i\to j}) \tag{4.6}$$

The above function is called the lagrangian. Now all we have to do is optimize this. Since the general lagrange theory says that for optimal solution $\hat{\theta}$ of the constrained problem, there exists $\hat{\lambda}$ and $\hat{\alpha}$ such that $(\hat{\theta},\hat{\lambda},\hat{\alpha})$ is a stationary point of $\mathcal{L}(\theta,\lambda,\alpha)$. In short, we can maximize $\mathcal{L}(\theta,\lambda,\alpha)$ and find λ and α instead of doing constrained optimization. First we find the the optimal transtion parameters, the emission parameters are identical by symmetry of the lagrangian.

$$\begin{aligned}\frac{\partial}{\partial A_{f\to s}}\mathcal{L}(\theta,\lambda,\alpha) =& \frac{\partial}{\partial A_{f\to s}} \left(\sum_{t=0}^{T-1} log(A_{t\to t+1}) \right) + \sum_{t=0}^{T-1} 0 + \frac{\partial}{\partial A_{f\to s}} \left(\sum_{i}^{K} \lambda_i (1 - \sum_{j}^{K} A_{i\to j}) \right) + \sum_{i=0}^{K} 0 \\ &= \left(\sum_{t=0}^{T-1} \frac{\mathbb{1}[x = A_{f\to s}](A_{t\to t+1})}{A_{f\to s}} \right) - \lambda_f \end{aligned} \tag{4.7}$$

To derive the lagrangian we see that the components which do not contain $A_{f\to s}$ derive to 0. In the first sum only the observations of $A_{f\to s}$, i.e. when we observe going from category f to s., will derive to $\frac{\partial}{\partial A_{fs}} \log A_{f\to s} = \frac{1}{A_{f\to s}}$. Hence the indicator function $\mathbb{1}[x = A_{f\to s}](A_{t\to t+1})$, it is 0 when the input $A_{t\to t+1}$ is not $A_{f\to s}$. Now setting the derivative to zero to find optimality:

$$\begin{aligned} \left(\sum_{t=0}^{T-1} \frac{\mathbb{1}[x = A_{f\to s}](A_{t\to t+1})}{A_{f\to s}} \right) - \lambda_f &= 0 \\ \frac{count(f\to s)}{A_{f\to s}} &= \lambda_f \\ A_{f\to s} &= \frac{count(f\to s)}{\lambda_f} \end{aligned} \tag{4.8}$$

However now we need to find what λ_f is. We do this by enforcing the transition constraint, that is from any category f we must go to any other state

[1] Look at wikipedia for a quick guide, a really usefull trick to know in any field that uses any form of optimization: `https://en.wikipedia.org/wiki/Lagrange_multiplier`

s. In formula we want $\sum_{j=1}^{K} A_{i\to j} = 1 \quad \forall i$:

$$\begin{aligned} \sum_{j=1}^{K} A_{i\to j} &= 1 \\ \sum_{j=1}^{K} \frac{count(f \to j)}{\lambda_f} &= 1 \\ \sum_{j=1}^{K} count(f \to j) &= \lambda_f \end{aligned} \tag{4.9}$$

We find λ_f to be the total amount of time a transition starting in f is observed. We find the estimate of A_{fs} simply as the fraction:

$$\hat{A}_{f\to s} = \frac{count(f \to s)}{\sum_{j=1}^{K} count(f \to j)} \tag{4.10}$$

The fraction of times you observe a transition from f to s compared to the total amount of times a transition from f to any state is observed.

Note by looking at the lagrangian we find symmetry and the maximum likelihood estimate of B_{fo} can be derived in exactly the same way as the maximum likelihood for A_{fs}, resulting in:

$$\hat{B}_{fo} = \frac{count(f \to o)}{\sum_{o=1}^{L} count(f \to o)} \tag{4.11}$$

Again the notation doesn't work out very nice, however it explains the situation. Again the maximum likelihood estimate for observing word o given category f is the fraction of times we observe o in category f compared to the amount of times we observe category f. Note a special case is A_{0j} which is then simply the fraction of times we observe category j compared to the rest of the categories.

4.3 EM for HMM

Expectation maximization is a general and widely used algorithm in machine learning to deal with either missing data or unobserved/latent variables to maximize the parameters of a model. It does this by estimating the likelihood iteratively and adapting the parameters.

4.3.1 EM in general

This part is based on the wikipedia article on EM[2]. Note that I use the same notation of the wikipedia article, $\mathbf{Z}$ does not denote categories in a HMM nor

[2] `https://en.wikipedia.org/wiki/Expectation_maximization_algorithm`

does $\mathbf{X}$ represent observations. In fact in the following we don't even assume that our model is a HMM. Say we have a statistical model that generates $\mathbf{X}$ observed data and $\mathbf{Z}$ unobserved data. This model is parameterized by θ, then the likelihood of this observed/unobserved data is defined as:

$$\begin{aligned} L(\theta; \mathbf{X}) &= \sum_z L(\theta; \mathbf{X}, z) \\ &= \int_Z L(\theta; \mathbf{X}, z) dz \end{aligned} \tag{4.12}$$

We could try to maximize this quantity over θ, and this would work perfectly fine. However, this often proves intractable since we have to sum/integrate over Z.

Since this is intractable, in practice the EM algorithm is used instead. The EM algorithm consists of the following 2 steps:

$$Q(\theta|\theta^{(t)}) = \mathbb{E}_{\mathbf{Z}|\mathbf{X},\theta^{(t)}}[\log L(\theta; \mathbf{X}, \mathbf{Z})] \tag{4.13}$$

Which is the expectation over $\mathbf{Z}$ given observed data $\mathbf{X}$ and previous $\theta^{(t)}$ of the log-likelihood of observed and unobserved data. This is called the **Expectation** step.

$$\theta^{(t+1)} = \arg\max_\theta Q(\theta|\theta^{(t)}) \tag{4.14}$$

The update for $\theta^{(t)}$ is the maximization of Q. This is called the **Maximization** step.

4.3.2 EM for HMM

We can apply EM for HMM, in previous section I tried to emphasise the fact that $\mathbf{Z}$ just denoted variables we don't know or are latent and not specifically the categories of a HMM. However in the case of HMM $\mathcal{Z}$ actually are the latent variables. Aswell as $\mathcal{X}$ being the observed variables.

Expectation in HMM

Plugging in the definition using the log liklihood of HMM and the definition of expectation for discrete variables we find:

$$\begin{aligned}\mathbb{E}_{\mathbf{Z}|\mathbf{X},\theta^{(t)}}[\log L(\theta;\mathbf{X},\mathbf{Z})] &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\theta^{(t)}) \left(\sum_{t=0}^{T-1} log(A_{t,t+1}) + \sum_{t=0}^{T-1} \log(B_{tt}) \right) \\ &= \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\theta(t)) log(A_{0,1}) + ... + \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\theta(t)) log(A_{T-1,T}) \\ &\quad + \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\theta(t)) log(B_{11})\end{aligned} \tag{4.15}$$

Note thet we are summing over $\mathbf{Z}$, this is the vector of categories in each timestep. More formally we can actually write:

$$\sum_{\mathbf{Z}} = \sum_{z_1} \sum_{z_2} ... \sum_{z_T} \tag{4.16}$$

In order to make life easier we look at what happens to a single term of the form $log(A_{t,t+1})$ by taking the sum inside, more concretely what I mean is the following:

$$\sum_{z_1} \sum_{z_2} ... \sum_{z_T} p(z_1, ..., z_T|\mathbf{X},\theta^{(t)}) log(A_{i,i+1}) \tag{4.17}$$