Algorithmical and Statistical Modelling

Lecture Notes

Graduate Lecture

Jacobs University Bremen
Fall 2012
Herbert Jaeger

Course Number 320551

Created Aug 31, 2012 by copying the 2010 lecture notes
Sept 24, 2012: added explanatory paragraph in intro to sampling, some small edits in 4.1
Oct 2, 2012: some minor notation bugs edited in Gibbs sampler section
Oct 8, 2012: updated URLs in section 4.8.
Oct 11, 2012: improvements in the Mau et al evolutionary tree example
Nov 15, 2012: significant corrections in the simulated annealing section, pointed out by Melanie Flöter.

1. Introduction

1.1 The big picture

Before the time of computers, a scientific or engineering *model* of some piece of reality was either

- mathematically rigorous but small (one or a few equations, for instance an equation of an electric oscillator), -- OR it was
- informal and text-based but large (thesauri in the wide sense, e.g. Linné's systematic catalogue of species).

In the time of computers, the concept and the potential of a model has undergone a revolution. It is now possible to capture complex, large chunks of reality AND put them under rigorous control of mathematics and algorithms.

Here are some examples of chunks of reality at your keyboard command:

1. Weather simulations, airflow simulations for airplane design.
2. Simulations of mechanical systems (crashing cars, shaking skyscrapers)
3. Simulations of folding proteins and binding atoms
4. Geographic information systems (GIS's)
5. Genome and protein databases
6. Customer databases, a company's SAP database, Jacobs's campus web files
7. Expert systems for airport traffic control, legal advice, power plant fault management and thousands of other complex everyday systems
8. Manufacturing line control, supervision and management systems
9. Chess computers
10. 3-d voxel models of MRT scans
11. A robot's navigation model of its environment
12. Neural network models that (try to) predict stock markets
13. Autopilots, car steering assistance systems, engine control systems
14. Body and motion models for movie animations and computer games
15. Speech recognition systems, handwritten text recognition systems
16. jpeg, mpeg, midi files (and doc files, too! as models of a text's visual appearance)
17. C programs (are models of algorithms!)

... and tens of thousands more, including, of course, the largest model of all, the Web.

The mathematical / algorithmical challenges presented in these examples are manifold. Here is an ad hoc list of some of the ubiquituous technical issues in modelling:

1. Dealing with large-sized datasets (ask Prof. Baumann who deals with terabyte datasets)
2. Integrating models from diverse formats and sources (numerical vs. symbolic, written in different programming languges, written by different people at different times or generated automatically, expressed in different mathematical formalisms ...)

3. Speeding up computations (fast low-level vs. slow high-level programming languages, model simplification, parallelization, finding fast algorithms, resorting to stochastic algorithms ...)
4. Economizing memory
5. Handling uncertainty (noisy data, probabilistic models, missing data...)
6. Negotiating precision -- high precision is costly (anytime algorithms, ...)
7. Creating models automatically from data (= machine learning) or in interaction with human experts (= knowledge acquisition, user interfaces)
8. Displaying / visualizing / audiolizing / haptifying models (scientific visualization, virtual reality, remote presence)
9. Interconnecting databases and models (high-bandwidth networking, synchronization, data abstraction formats like XML, ...)
10. Detecting model errors and assessing model limitations
11. et cetera

Models are used for many types of purposes:

(2.1) Simulation
(2.2) Prediction and ...
(2.3) ... Retrodiction (yes, sometimes it is hard to know the past, e.g. in earth climate modelling or rebuilding the evolutionary tree)
(2.4) Analysis ("where am I?" asks the robot and queries a *particle filter* model...), i.e. deriving implications from a model – the classical use of rigorous scientific models
(2.5) Documentation, classification, finding ordering systems (the classical use of encyclopaedic scientific models)
(2.6) Control, signal processing (e.g., image restauration, engine control, data transmission)
(2.7) Classification and decision making
(2.8) Functional design (cars, buildings, pharmaceuticals...)
(2.9) Aesthetical design (buildings, cars, perfumes... and movies and music)
(2.10) Fault detection
(2.11) "New insight", "understanding"
(2.12) et cetera

All in all, the computer and the world reflections inside it have provided us an entirely new of way of doing science and engineering – and of living our everyday life (or is that not so? are we the same humans as before? in what sense have we changed?)

It is clear that a single person or a single lecture can capture only a tiny fraction of this grand world of modeling. What will you see in this lecture?

1.2 The lecture picture

Many of the themes around modelling are covered by courses that you can take at Jacobs, for instance:

1. Simulating large-scale physical systems with PDEs (amply covered in the lectures you find in Computational Science)
2. Machine learning techniques – (come to my ML class if you like to learn to teach machines to learn)
3. Database topics (Peter Baumann is expecting you)

4. Artificial Intelligence and robotics (Andreas Birk and Andreas Nüchter will tell you how robots model the world around them, including other robots and you, and how you should model robots; Michael Kohlhase awaits you to treat you with luxurious logics for languages and maths modelling)
5. Biosequence and protein modelling (Jacobs has bioinformatics!)
6. Statistics and statistical software packages (in the other school, Adi Wilhelm would be glad to greet you in his extremely useful courses on statistics)

So you see, (almost) all the Jacobs world next to you does nothing but modelling! and furthermore you see that there are many ways to approach modelling themes, some more mathematical, some more application-driven, some more CS oriented.

But now, what will be the purpose of *this* course?

I have two answers. The first is that you will learn quite a number of practically useful techniques not covered elsewhere at Jacobs – so this is really going to be an "enabling" course.

The other answer is the one that really guided me in assembling the materials for this course. I want to make you aware of the inescapability of the *reality gap* in all modelling attempts, and to pave the way for what I think is the most powerful compromise *bridging technique* that we possess.

What is the reality gap? If you wish to capture a chunk of reality in a model, your solution MUST BE IMPERFECT because of the following reasons:

- Clearly you can't model all of reality in a single model. So you must *isolate*, identify, carve out some part or aspect of the world – necessarily severing in your model the ties the original chunk has with the rest of the world.
- You must choose one *level of description* out of many that would be needed for a total understanding. It is quite a different thing to describe the brain in terms of chemical reactions vs. connected neurons vs. the thoughts that the brain thinks. Some of the most notable and modern results in science lie in methods that describe *self-organization* mechanisms to connect (just) two adjacent levels of description (e.g. Manfred Eigen's hypercycle model of the origins of life – a Nobel prize, or Ilya Prigogines models of oscillating chemical reactions – another Nobel prize, or Heinrich Haken's *Synergetics* model of laser oscillations – almost a Nobel prize).
- You can't measure all the quantities that you would need to determine the starting conditions for a (dynamical) model. Empirical data are always *incomplete*, and models made from them are at best interpolations.
- The very idea of a *measurable* hides within it a brutal act of ignorance – namely of not measuring (even not conceptualizing) all the things that you do *not* measure. A side note: apparently the sensor signals that our brain gets cannot be well characterized as reflecting measurable quantities. For instance, a single receptor cell that we have in a finger joint may respond to vibration *or* temperature *or* pressure *or* joint angle – depending on temporal context and co-variables.
- Our models are *finite* but world chunks – arguably – are not. After we have built a program or formula or database, then that is *all* we have – but reality chunks can always be investigated further. This problem is very acutely known in Artificial Intelligence, where it is called the *frame problem*; it is one of the toughest obstacles to building artificial "intelligent" systems.

- Human-made models are co-determined by *fashions* and historical (even political) context. You can only think about reality in a way that your peers will acknowledge – if you try to progress beyond the view of the day, you are either a genius (but even Einstein's discoveries were well embedded in his day's thinking) or you will be quickly expelled from "the scientific community".
- Models are prone to be *inconsistent*, -- which is a polite way to say they are impossible. This is hard to detect, and if detected, to repair (AI has a subdiscipline devoted to making the best out of inconsistent knowledge, called *nonmonotonic logics*).
- We are very bad, as humans, to understand cyclicity, self-referentialty, or *non-linearity* in general. Thus, even when our models capture such aspects of reality, we have a hard time understanding our own models. Indeed, engineers (typically) prefer to simply avoid nonlinearity, gaining intuitive grasp but missing reality.
- Et cetera.

I think the single most powerful and generally useful theoretical and algorithmical tool we have to come to terms with the reality gap is... shhhh!! -

... statistics.

With statistical methods (in a very wide sense) you may explicitly reflect that there are gaps in your model, but still make the best out of it that you can. You can also enter into your model a model of your own preconceptions, prior assumptions or mere intuitions and reflect – indeed calculate numerically – the extent by which they influence your model. With statistical methods you can state clearly (and be aware of it) where your model is blurry, – but you can also detect the parts where it is razor sharp.

So, I will unfold in this lecture a number of modelling techniques that are all statistical in nature, spanning an arch from the purest possible statistical models – namely, just probability distributions – to intricate computationel methods to deal with large networks of statistically interacting quantities (Bayesian networks). At the time of writing this introduction, the planned layout for this lecture is the following (subject to change):

1. A refresher on probability and statistics, with an emphasis on comparing "classical" with "Bayesian" statistics
2. A glimpse on methods to represent probability distributions mathematically/algorithmically (exponential distributions, mixtures of Gaussians, Parzen windows), illustrating "parametric" and "non-parametric" methods
3. Sampling methods for probability distributions: sampling by transformation, rejection sampling, Markov Chain Monte Carlo sampling, Gibbs sampling, Metropolis algorithm. Application example(s): reconstruction of evolutionary trees from current DNA data.
4. Optimization by simulated annealing, thermodynamic interpretation of Metropolis sampler.
5. Bayesian networks and graphical models (diagnostic reasoning, representation and inference, EM learning algorithm).
6. Fuzzy logic: principles and control applications.

1.3 Supplementary reading

These lecture notes assemble themes from various areas, and I am not aware of a single textbook where you can find all of it together. The lecture notes will point out URLs to online available tutorials and papers at various places. In addition, the following textbooks each contain one or several relevant chapters:

Mitchell, Tom M.: *Machine Learning* (McGraw-Hill, 1997) IRC: Q325.5 .M58. *Chapter 6 gives a concise intro to Bayesian inference and elementary Bayesian networks.*

Duda, R.O., Hart, P.E., Stork, D.G.: *Pattern Classification*, 2nd edition (John Wiley, 2001) IRC: Q327.D83 *Chapter 3 presents foundations of Bayesian vs. maximum likelihood estimation, including intro to hidden Markov models and the EM algorithm – just like chapter 6 from the Mitchell book.*

Durbin, R. et al.: *Biological sequence analysis : probabalistic models of proteins and nucleic acids*. Cambridge, UK : New York : Cambridge University Press, 1998. IRC: QP620 .B576 1998 *Good description of some sampling techniques, interesting applications in biosequence modelling*

Bishop, Christopher M.: Neural Networks for Pattern Recognition. Oxford University Press, 1995. IRC: **QA76.87 .B574 1995** *Chapter 2 amounts to Chapter 3 of these lecture notes (basic methods for parametric and nonparametric density representation and estimation)*

Bishop, Christopher M. Pattern Recognition and Machine Learning. Springer Verlag, New York 2006. Online copy available through IRC. *A thick book on what its title says, with a particularly strong and long chapter on Bayesian networks of various shades.*

Neal, Radford M. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, Dpt. of CS, Univ. Toronto, 1993. ftp://ftp.cs.toronto.edu/pub/radford/review.pdf

S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi (1983) *Optimization by Simulated Annealing.* Science, Vol. 220 Nr 4598, 671-680. *Classical (by now), very readable paper introducing simulated annealing. Includes a beautiful demonstration of simulated annealing in a computer layout optimization task. Here is a local online copy*

Ackley, D.H. and Hinton, G.E. and Sejnowski, T.J.: A Learning Algorithm for Boltzmann Machines. Cognitive Science 9 (1985), 147-169. *Another classic, introducing a neural network model of an associative error-correcting memory, based on the Boltzmann distribution. Here is a local online copy.*

Huang, C., Darwiche, A.: Inference in Belief Networks: A Procedural Guide. Int. J. of Approximate Reasoning 11 (1994), p. 158ff. *this is the tutorial text o which much of the algorithmic outline of the Bayesian Network chapter in my lecture notes is based. Local copy at* http://www.faculty.jacobs-university.de/hjaeger/courses/AlgMod05/ijar95.pdf

2 A refresher on essential probability theory and statistics – classical and Bayesian

The aim of this section is to make you acquainted with a number of notions from probability theory and statistics that constitute a required background for this course.

2.1 A handful of basic concepts

It is possible to become a reasonably good modelling practician without really knowing what probabilities are – you can use equations like the Bayes formula or decision criteria "mechanically" – but it is not possible to become a really creative in this field without this knowledge. Therefore we devote some time to a more rigorous (re-) introduction of the basic concepts of probability theory and statistics.

A fine webpage is http://www.probability.net – you can find there an online tutorial and a dictionary of all important definitions.

We will in some detail consider a simple standard task, namely, estimating the probabilities of symbols from a sample. If the sample is small, this task becomes surprisingly subtle. A typical situation in bioinformatics is the following. Proteins are sequences of amino acids, of which there are 20 different that occur in proteins. They are standardly tagged by 20 capital symbols, as A, G, H, ... all intimately familiar to biologists. Proteins come in families. Some protein in one species has typically close relatives in other species. Related proteins differ in detail but generally can be *aligned*, that is, corresponding sites in the amino acid string can be detected, put side by side, and compared. For instance, consider the following short section of an alignment of 7 amino acids from one family[1]:

```
...GHGK...
   AHGK...
...KHGV...
...THAN...
...WHAE...
...AHAG...
...ALGA...
```

Fig. 2.1: Seven aligned protein sniplets from one protein family (here: of globulines).

A basic task that bioinformatics faces is: for each column in such a *family alignment*, estimate the probability distribution of the amino acids in this column, as you would expect it to be in the total population of all proteins belonging to this family. This task is, on the one hand, important: because such distribution estimates are the basis for deciding whether some newly found protein belongs into the family. On the other hand, this task is apparently rendered difficult by the fact that the sample of aligned proteins used to estimate this distribution is typically quite small – here we have only 7 individuals in the sample. As can be seen in Fig. 2.1, some columns have widely varying entries (e.g. the last column `K K V N E G A`). In contrast, the family of related proteins is huge: in every species one would expect at least one family member; typically many more. So how can one derive "good" estimates for the distribution of symbols in a large population, from very small samples?

[1] Example and some parts of this Section taken from: R. Durbin, S. Eddy, A. Krogh, G. Mitchinson: *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University press 2000

For this task of estimating a probability distribution (and all other such tasks) there are two major types of approaches:

1) The "classical", "frequentist", "objective", where probabilities are defined in terms of limits of frequency counts. This is the kind of probability theory and statistics that has dominated mathematics and statistics in the last centuries and it is the approach taught in most university courses on statistics. In this view, a probability of a symbol in a population is its frequency *in the limit of infinite population size*. The probability *P*(A) of "amino acid A occurs in the population" is objectively defined – at least in ideal theory (assuming the population is infinitely large). The classical approach gives a clear picture of things when one has access to large samples but has difficulties in dealing with small samples.
2) The "Bayesian", "subjective" approach where a probability is defined as a subjective degree in belief that a newly observed symbol would be of some particular kind. Here the probability *P*(A) need not be defined objectively. But this does not mean Bayesian statistics is not a rigorous mathematical field. Bayesian theory is not concernd with what probability *is* but with how rational people should *correctly reason about* probabilities. Bayesian statisticians ask (and answer) questions like: If someone believes some things about some probabilities in some population, what can this person formally deduce from his starting assumptions? The Bayesian approach is better suited than the classical one when it comes to drawing conclusions from small samples. In the words of E.T. Jaynes, a fierce proponent of Bayesian statistics: *"Scientific inference is concerned, necessarily, not with empty assertions of 'objectivity' but with information processing; how to extract the best conclusions possible from the incomplete information available to us."* Because such questions have recently found to be of extreme practical relevance – not the least in bioinformatics and in Artificial Intelligence – Bayesian statistics has seen a surge of interest in the last two decades, and has become very important for practical machine learning techniques.

Until recently there was something like a "war of believers" between the two approaches. The belligerent atmosphere is reflected in the most prominent, original textbook on Bayesian statistics (4 MB) by E.T. Jaynes, called not very modestly "Probability theory: the Logic of Science". Click on http://www.quackgrass.com/roots/0796rts.html for a short intro to Bayesian logic / probability theory. Today the aggressive tone has largely vanished and both approaches to probability are considered as valid, if alternative, perspectives.

Both approaches share the definitions (but not the interpretation) of some elementary concepts, which we will now revise.

Event space, probability space. Symbol: Ω. This concept, which is the fundament of statistics and probability theory, is unfortunately very hard to understand. The reason for this conceptual difficulty is that Ω has a dual nature: (i) as a real-world entity, which cannot be formally specified but needs everyday language to be described, and (ii) as a mathematical object that can be formally specified... Event spaces are, so to speak, the interface between the real world and mathematical (probabilistic) models. If one looks at some event space from the real-world side, one sees a real-world thing, which can be described only with real-world language, i.e. plain English. If one looks at some Ω from the formal side, from maths, one sees a mathematical object (a set, to be more precise – an object of set theory). Confused? you should be...

As a real-world entity, a good way to understand event spaces is to start from scientific paper writing (in the empirical sciences, like physics, biology, experimental psychology etc). Such papers typically describe an experimental setup where certain measurements are taken, or they specify (e.g. in the geosciences or botany) a location on this earth where observations have been made (e.g. a mountain range where rock samples have been taken, or a wildforest area where plant specimen were collected). The essence of empirical science is that other groups than the one who first did the experiment / expedition must be enabled to *reproduce* the findings. The outcomes of reproducing experiments are comparable to the originally reported results only to the extent that the reproducing experimenter reconstructs the original experimental setup (or goes to the same mountains or forests). Only if this similarity is warranted, the data collected by the second experimenter can be assumed to have the same "distribution" (we will soon explain that concept, for the time being you intuition must suffice) as the data sampled in the original investigation. An event space is what is specified (in a scientific paper, for instance) as a setup / context / location / experimental condition with respect to its role as a source of potential data. In papers in physics / chemistry / biosciences, this specification is usually done in a section called "methods". In papers in psychology / medicine /sociology it would typically be the specification of the population of human subjects that were investigated ("As subjects for our study we used undergraduate students from psychology, balanced in gender and with an age between 19 and 23..."). In sum, specifying an event space amounts to specifying particular conditions for collecting data.

As a formal entity, an event space is just seen as a set – and almost always denoted by Ω. In the light of what I said before, the elements ω of this set Ω should be considered as all the *potential acts of measurements* that *could* be made in experiments / expeditions of a certain type. A sometimes used terminology is to speak of ω as *realizations* of Ω. Each such realization is a source of measurement data. For example, if (in the real-world perspective) Ω is "undergraduate students from psychology, balanced in gender and with an age between 19 and 23...", then whenever in some university a particular undergraduate psychology student Mr. A with age between 19 and 23 is chosen from a gender-balanced sample, this Mr. A would be considered an $\omega \in \Omega$. However, mathematicians don't care about this real-world interpretation of the $\omega \in \Omega$. What they care about is that once Ω is fixed, the measurement data that can be obtained from $\omega \in \Omega$ have a well-defined statistical distribution. Mathematicians, then, care about the mathematical apparatus needed to equip (arbitrary) sets with the requisite add-on mathematical structure that enables them to handle such statistical distriubutions of measurement data obtained from $\omega \in \Omega$.

We will now describe this mathematical structure. Unfortunately, it is not simple (and has taken mathematics centuries to develop – only completed by the work of Andrey Kolmogorov in the early 1930's.

Events are subsets $A \subseteq \Omega$. In the simplest case, an event is an *elementary event* $\{\omega\} \subseteq \Omega$, but in general an event is a larger subset. In our example, "proteins belonging to a particular family" would be an event, or "proteins of family *X*, which have amino acid G in position 110".

σ-algebra, σ-field, event field: The set of all events in Ω. Typical symbol: F or $\mathcal{A}$, $\mathcal{B}$, ... (if you have Latex with the AMS package, use \mathfrak font!). We have $F \subseteq \text{Pot}(\Omega)$. If Ω is finite, typically $F = \text{Pot}(\Omega)$. With infinite Ω, F is typically *much* smaller than $\text{Pot}(\Omega)$. Not any subset of $\text{Pot}(\Omega)$ qualifies as a σ-algebra. A σ-algebra must adhere to certain structural axioms. Here is the definition of σ-algebras:

Definition (5.1): $F \subseteq \text{Pot}(\Omega)$ is a σ-algebra if

1) $\Omega \in F$,
2) $A \in F \Rightarrow A^C \in F$ (closure under complement),
3) for every sequence $(A_n)_{n = 1, 2, \ldots}$ in F, the set $\bigcup_{n=1}^{\infty} A_n$ is in F (closure under countable union).

This definition reflects how we would like to be able to reason about events. Condition 1) says that the "all-event" is an event, that is, the event "we observe *some* individual from Ω". Condition 2) requires that if we have some event A, then we can also talk about the event "not A". Finally, condition 3) fixes that if we have a (countable) number of events A_n, then the event "we observe something from one of the A_n" is also a valid event.

σ-algebras are the fundamental concept of probability theory, of measure theory, and the theory of (Lebesgue) integration theory.

A **measureable space** is an event space equipped with some σ-algebra, written (Ω, F).

A **probability measure** P is defined as follows:

Definition (5.2): Let (Ω, F) be a measurable space. A function $P: F \rightarrow [0, 1]$ is a probability measure on (Ω, F), if

(2.1) $P(\Omega) = 1$,

(2.2) for every sequence $(A_n)_{n = 1, 2, \ldots}$ of pairwise disjoint events it holds that

$$P(\bigcup_{n=1}^{\infty} A_n) = \sum_{n=1}^{\infty} P(A_n) \qquad (\sigma\text{-additivity})$$

Remarks:

(2.1) Conditions 1) through 5) are the **Kolmogorov axioms** of probability theory. In classical statistics, these axioms are the *foundation* of probability theory. In Bayesian statistics, these laws are *derived* from other axioms.

(2.2) The "Bayesians" often admit in 5) only finite sequences.

(2.3) The triple (Ω, F, P) is called a **probability space**.

(2.4) Very often it holds that $P(\omega) = 0$ or even $P(A) = 0$ for nonempty A. Then A is a **null set**. For instance, if Ω is the set of all infinite sequences generated by some random number generator, then $P(\pi) = 0$: the chance of obtaining the digit sequence belonging to π is zero.

(2.5) Here are some elementary properties of probability spaces:

i) $A, B \in F \Rightarrow A \cap B \in F, A \cup B \in F, A \setminus B \in F$.

ii) $\varnothing \in F$.

iii) For every sequence $(A_n)_{n = 1, 2, \ldots}$ of events it holds that $\bigcap_{n=1}^{\infty} A_n$ is in F.

iv) $A \subseteq B \Rightarrow P(A) \leq P(B)$.

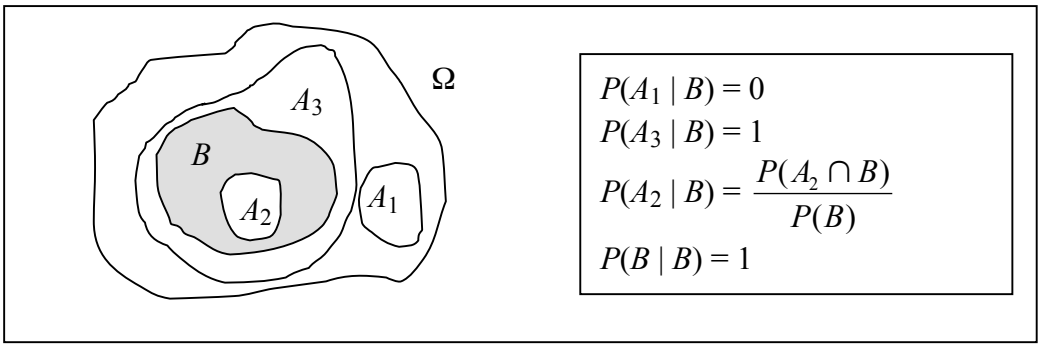
Conditional probability. Let (Ω, F, P) be a probability space, $B \in F$ an event with $P(B) > 0$. The function

(2.1) $P(\cdot \mid B)$: $F \to [0, 1]$

$$A \mapsto \frac{P(A \cap B)}{P(B)}$$

is again a probability measure on (Ω, F), the *conditional probability under hypothesis B*.

Here is a graphical display that explains how to think about conditional probabilities:



Here are some rules for computing with conditional probabilities:

1) $P(A \cap B) =: P(A, B) = P(A \mid B)\, P(B)$

2) Let $\Omega = \bigcup_{i \in I} B_i$ with pairwise disjoint B_i. Then for every $A \in F$,

$$P(A) = \sum_{i \in I} P(B_i) P(A \mid B_i),$$

the *formula of total probability*.

3) Let $\Omega = \bigcup_{i \in I} B_i$ with pairwise disjoint B_i. Then

$$P(B_n \mid A) = \frac{P(B_n) P(A \mid B_n)}{P(A)} = \frac{P(B_n) P(A \mid B_n)}{\sum_{i \in I} P(B_i) P(A \mid B_i)},$$

which is *Bayes formula*.

Baysian statistics *starts* from rules like $P(A \cap B) =: P(A, B) = P(A \mid B)\, P(B)$, which are justified as "rational", "intuitively correct" laws of reasoning about probabilities. (To be more precise, a Baysian statistician would write $P(A \cap B \mid M) =: P(A, B \mid M) = P(A \mid B, M)\, P(B \mid M)$ instead – Baysians *always* include a formal reference M to some prior knowledge about the relevant domain of reality into their formulas, reflecting the fact that all intuitive reasoning

about probabilities must start from some prior assumptions about the world one is reasoning about.)

Further elementary concepts...

Measure space. Typical symbol: $(E, \mathcal{B})$. This concept is closely related to, but fundamentally different from the measurable space (Ω, F). The basic intuition is that the "things" in Ω are "just there as they are there" (a Kantian philosopher might think of the *Ding an und für sich*, the "things as they are for themselves", not being understood or observed by humans). In order to get access to a thing $\omega \in \Omega$, one as to *observe* or *measure* it. The outcome of the observation is an object $a \in E$. In our protein example, ω might be a physical gene coding a protein in a biochemist's sequencing apparatus, and after sequencing the gene, a might be the formal sequence of amino acid symbols corresponding to the protein. Thus E would be created from Ω through the measurement operation "run a gene ω through a sequencer, transform the nucleic acid sequence into an amino acid sequence, and output the sequence of its symbols". With another observation operation one gets another measure space E. Taking up our example, with the measurement operation "run a gene ω through a sequencer, transform the nucleic acid sequence into an amino acid sequence, and output the 110th symbol that you get", one would only observe genes/proteins at the 110th position. The set of possible observation outcomes then would be the set $E = \{\texttt{A}, ..., \texttt{Y}\}$, which has 20 elements. In sum, the measure space builds on a set E that contains *all possible outcomes of observing elements* $\omega \in \Omega$ under a given observation procedure.

It is very important to keep the measure space apart from the underlying probability space with its measurable space (Ω, F). In our last example, the event space Ω of the underlying probability space (Ω, F, P) would have as many elements as there are proteins of the family – potentially (given the open-endedness of evolution) infinitely many. In contrast, E contains just 20 elements.

A measure space is also equipped with a σ-algebra, $\mathcal{B}$. In our last example, since E is finite, we would typically take $\mathcal{B} = \text{Pot}(E)$.

Random variables, typical symbol: X. If you thoroughly understand the concept of a random variable, nothing can happen to you in the remainder of this lecture! Formally, a random variable is a mapping $X: \Omega \rightarrow E$, also written as $X: (\Omega, F, P) \rightarrow (E, \mathcal{B})$. Intuitively, a random variable describes a measurement or observation procedure. To each elementary event $\omega \in \Omega$, a random variable assigns an observation, or measurement outcome, $X(\omega) = x \in E$. In our example, X would assign to every protein the amino acid symbol detected at the 110th position.

Now comes a subtle and powerful idea. A random variable $X: (\Omega, F, P) \rightarrow (E, \mathcal{B})$ "transports" the probability measure P from the underlying probability space (Ω, F, P) into the measure space, creating there another probability measure, the *induced measure* P_X, by the prescription

$$\forall B \in \mathcal{B}: \; P_X(B) = P(X^{-1}(B)) \tag{2.2}$$

In our example, for instance, we would have

$$P_X(\text{"symbol A is observed"}) = P(\text{"all globulines that show A at position 110"})$$

Instead of $P_X(B)$, the notation $P(X \in B)$ is also used.

The measure space $(E, \mathcal{B})$ is typically much smaller and has a simpler structure than the underlying probability space (Ω, F, P). This reflects the loss of information usually incurred by any measurement process!

Distribution. The induced probability measure P_X on the measure space is called the *distribution of the random variable X*.

In real-world modelling tasks, the underlying probability space (Ω, F, P) is usually an object that we cannot directly access or model mathematically (think of how difficult it would be to model "the set of all globulines in all organisms of past, present and future"). However, the much simpler measure spaces with their induced probability measures (distributions) *can* be analysed. Therefore,

the main object of study in probability theory and statistics is distributions.

All we have said so far is just the long, hidden story behind the simple distributions that we are used to work with. It is the story of how axiomatic probability theory tries to come to terms with the concept of probability. The probabilities we mostly speak of are distributions, which are "borrowed" from the underlying (but ignored) probability spaces by virtue of Eqn. (2.2).

A cautionary remark. The distinction between the underlying probability space and distributions is sometimes obscured in introductory textbooks that try to make life (too) easy for the reader. In these books, the distinction between Ω and E is not made. For instance, one may find that the set $\{1, 2, 3, 4, 5, 6\}$ of possible outcomes of throwing a die is called a probability space, and the set of probabilities of these outcomes (all 1/6 for a fair die) are called a probability measure (used interchangeably with distribution). Technically speaking this is admissible: because the events $\omega \in \Omega$ are not formally defined but represent one's pre-mathematical choice of the "piece of reality" one wants to model, one is in principle free to choose *anything* for Ω, including E. But if the distinction is dropped, the intuitive interpretation of random variables as measurement/observation operators is lost; furthermore, some themes of advanced probability theory become impossible to treat (for instance, the question of whether the zigzag trajectories of Brownian motion can be assumed to be continuous).

In finite measure spaces, a distribution is most conveniently expressed by a table or a bar chart giving the values of $P_X(x) = P(X = x)$ for all $x \in E$. In continuous-valued measure spaces, a distribution is often represented by a probability density function (*pdf* is a much-used abbreviation). For instance, the normal distribution density function with mean μ and standard deviation σ is given by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \; e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.3}$$

We use lowercase p to denote pdf's and capital P to denote probability measures or distributions in general. For the event B = "measurement lies between 0.5 and 1.2" (note $B \in \mathcal{B}$) we can use (2.3) to calculate a probability

$$P(X \in B) = \int_{0.5}^{1.2} p(x)dx \,. \tag{2.4}$$

Note that B does not refer to a single measurement "action" – it refers to the class of all individual measurement actions that return a value between 0.5 and 1.2. In probability theory, when talking about distributions, the concept "event" refers not to individual observation actions (those would be the measurements $X(\omega) \in E$), but to classes of measurements, defined by ranges of their outcomes, that is, the concept "event" refers to the $B \in \mathcal{B}$.

When it is clear from the context that one is dealing with distributions (and not with the underlying probability space), often the induced probability measure P_X is simply written as P. In fact, most *"P"* symbols that you will encounter in statistics should actually be interpreted as P_X.

Numerical random variables and expectation. A random variable is *numerical* if the measure space E is numerical, that is, integer-, real-, or complex-valued. The expectation of a numerical random variable X is its "average value" and for integer-valued X is given by

$$E[X] = \sum_{i=-\infty}^{i=\infty} i \; P(X = i), \tag{2.5}$$

and for the case of real-valued variables with pdf p is given by

$$E[X] = \int_{x=-\infty}^{x=\infty} x \; p(x)dx \,. \tag{2.6}$$

Given a numerical random variable X, one obtains a random variable X' that is *normalized to zero mean* by putting $X' = X - E[X]$.

Independent and uncorrelated random variables. Two random variables X, Y: $(\Omega, F, P) \to (E, \mathcal{B})$ are *independent* if for all $A, B \in \mathcal{B}$, it holds that

$$P(X \in A, Y \in B) := P(X^{-1}(A) \cap Y^{-1}(A)) = P(X \in A)\, P(Y \in B). \tag{2.7}$$

X and Y are *uncorrelated* if

$$E[X\,Y] = E[X]\, E[Y], \tag{2.8}$$

or equivalently, if their *covariance*

$$\mathrm{cov}(X, Y) = E[(X - E[X])\,(Y - E[Y])] \tag{2.9}$$

is zero. Only random variables with numerical values can be uncorrelated, but random variables with values in any arbitrary measure space can be independent. Independent numerical random variables are always uncorrelated, but uncorrelated numerical random variables are not necessarily independent. Thus independence is a (much) stronger notion than

uncorrelatedness. Unfortunately, in analytical and computational investigations, independence is also much more difficult to prove or use.

Side remark. A modern and fashionable field within machine learning is *blind source separation*. Given *n* statistically independent signal sources (e.g., speakers in a room) and *n* measurements which each pick up a different mixture of the source signals (e.g. microphones placed at different positions in the room), one can use the fact that the sources are statistically independent to learn from a training sequence of the mixed measurements a filtering device that re-separates the signal mixtures into their independent components. The quality of the separated signals is sometimes astounding. Applications (besides speech processing): picking out a unborn baby's heartbeat from the "noise" signals generated inside a mother; detecting individual signal sources in EEG mixtures of signals. Check http://web.media.mit.edu/~paris/ica.html for pointers to people, papers, labs and striking audio-demos. The results obtained from independence analysis with the modern techniques of blind source separation are often much stronger than results obtained with the more traditional and easier methods of classical linear signal analysis and filtering, which rely merely on uncorrelatedness.

Joint and marginal distributions. Often a probability space (Ω, F, P) is observed/described by several random variables $\mathbf{X} = (X_i)_{i=1,\ldots,n}$ simultaneously. These variables may take values in different measure spaces $(E_i, \mathcal{B}_i)$. Think of this as describing a complex piece of reality in terms of a number of different measurables, observables, concepts. One may glue the individual random variables together in a single *product* random variable $\bigotimes_{i\in\{1,\ldots,n\}} X_i = X_1 \times \cdots \times X_n$ that takes values in the product space $\bigotimes_{i\in\{1,\ldots,n\}} E_i = E_1 \times \cdots \times E_n$, so the values taken by the product random variable is an *n*-tuple of the individual variables' values:

$$(\bigotimes_{i\in\{1,\ldots,n\}} X_i)(\omega) = (X_1(\omega),\ldots,X_n(\omega)) = (x_1,\ldots,x_n)\,. \tag{2.10}$$

The distribution $P_{\bigotimes_{i\in\{1,\ldots,n\}} X_i}$ of this product random variable is called the *joint distribution* of the random variables $(X_i)_{i=1,\ldots,n}$. We also write $P(\bigotimes_{i\in\{1,\ldots,n\}} X_i)$ or simply $P(\mathbf{X})$ for the joint distribution. Notice that the joint distribution is likely to be a very unwieldy beast. To see why, consider the simplest possible case, where all the concerned random variables are binary (you may conceive them as Boolean observations, indicating the presence or absence of an observation). Then the distribution $P(X_i)$ of any individual variable is just a histogram over the values 0 and 1. However, the joint distribution would assign a probability value to each possible combination of the *n* binary observations, which makes this a histogram over 2^n arguments. In naive words, joint distributions are "exponentially more complex" than the individual distributions. If the individual distributions are themselves not as simple as just a binary distribution, it soon becomes practically impossible even to write down some closed formula for characterizing the joint distribution – and the computational and computer-based methods for handling complex distributions that we will learn about in this lecture will be needed.

The joint distribution of a descriptive ensemble $(X_i)_{i=1,\ldots,n}$.comprises the *complete* probabilistic information about the piece of reality that one is modelling by $(X_i)_{i=1,\ldots,n}$. Any specific question that one might ask about this piece of reality can be derived from $P(\bigotimes_{i\in\{1,\ldots,n\}} X_i)$. For instance, one may wish to ignore some of the descriptors and ask for the

distribution of one or a few selected observables only. Such "ignore the rest" distributions are called *marginal distributions*. They can be computed, in principle, by integrating away the others. For instance, if the joint distribution is characterized by an *n*-dimensional pdf *g*, we could recover the pdf g_1 of the marginal distribution of X_1 by

$$g_1(x_1) = \int_{\mathbb{R}^{n-1}} g(x_1,...,x_n)\, dx_2 dx_3 ... dx_n, \tag{2.11}$$

or in the discrete case, where each X_i takes values in $E_i = \{a_1^i,...,a_{m_i}^i\}$, the marginal probabilities of *X1* would be obtained by summing over all combinations of the other variables' values:

$$P(X_1 = a_x^1) = \sum_{a^2 \in E_2} ... \sum_{a^n \in E_n} P(X_1 = a_x^1, X_2 = a^2,..., X_n = a^n). \tag{2.12}$$

Marginal distributions of more than a single variable can be computed by integrating/summing away the remaining ones in a similar way. While thus the marginal distributions can be recovered from the joint distribution, conversely the joint distribution can be constructed from the marginal distributions only if the $(X_i)_{i = 1, ..., n}$.are independent. Then (and only then) it holds that the joint pdf *g* is

$$g(x_1,...,x_n) = g_1(x_1) \cdot ... \cdot g_n(x_n) \tag{2.13}$$

for distributions with pdf's, and

$$P(X_1 = a^1, X_2 = a^2,..., X_n = a^n) = \prod_{i=1,...,n} P(X_i = a^i). \tag{2.14}$$

The joint distribution $P(\bigotimes_{i\in\{1,...,n\}} X_i)$ can be *factorized* into a product of conditional distributions by

$$P(\bigotimes_{i\in\{1,...,n\}} X_i) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_1, X_2)\ldots P(X_n \mid X_1,..., X_{n-1}). \tag{2.15}$$

The proof is a homework exercise.

Samples. The underlying event space Ω contains all *possible* individual measurable events ω (or elementary events). In life's reality, only a small fraction $\omega_i \in \Omega$ (where $i \in I$ and *I* is by life's necessity finite) of all possible measurable events is realized in concrete observations. Such a set $\{ \omega_i \mid i \in I \}$ of actually realized measureable events is called a *sample*. At least, this is the strict definition of the term.

However, we have noted that statisticians prefer not to talk about measurable events ω_i but rather like to think in terms of their distributions. This is reflected in another, related usage of the word *sample*, which also reflects a typical situation in the experimental sciences, namely, that some experiment or measurement is repeated many times in order to obtain an as precise as feasible estimate of some quantity of interest (for instance, by taking the mean over reapeated measurements).

To model this situation the following approach is taken in statistics. The elements ω of the underlying probability space $(\Omega, \mathcal{F}, P)$ are taken to be the *sequences* of repeated experiments – and for mathematical convenience, it is assumed that one such "repeat-experiment-session" ω comprises infinitely many repetitions of the experiment. Next, a sequence $(X_i)_{i \in}$ of random variables is considered, where $X_i(\omega)$ refers to the i-th measurement outcome in the repeat-experiment session ω. This is of course an idealized picture: in practice, an experiment cannot be repeated infinitely many times. What one has in real-life, is the outcomes of n many measurements of one repeat-experiment session ω, that is, the data that one has really available are comprised in a vector $\mathbf{X}(\omega) = (X_1(\omega), ..., X_n(\omega)) = (x_1, ..., x_n) \in E^n$. Such data vectors are then called *samples.*

Although this might appear a bit contrived, it gives a faithful account of how research in the empirical sciences should be carried out: In some Lab A, some quantity of interest is derived with an as great as possible precision (implying repeated measurements) – this is, $(X_1(\omega), ..., X_n(\omega))$ is used by Lab A's statistician to estimate the quantity of interest. Another Lab B may want to contest or improve on Lab A's result. They will also carry out a repeat-experiment session ω', obtain a sample $(X_1(\omega'), ..., X_{n'}(\omega'))$, and infer something about the quantity of interest from *this* sample. Typically, their results will somewhat deviate from Lab A's results. The question, then, is how the conclusions obtained in Lab A from the sample $(X_1(\omega), ..., X_n(\omega))$ can be *compared* with the conclusions obtained in Lab B from the sample $(X_1(\omega'), ..., X_{n'}(\omega'))$ – for instance, if $n' > n$, to what extent are the conclusions drawn by Lab B's statistician more reliable than the findings in Lab A? Such considerations lie at the heart of statistics and the theory of statistical estimation (of quantities of interest from samples).

This strict understanding of a sample as $(X_1(\omega'), ..., X_{n'}(\omega'))$ is not easy to grasp, especially because there is also a "naïve" setup of a probability model where one has only a single random variable X. An example will be helpful to sort these subtle concepts out.

Consider an article in a medical journal where it is stated that patients with a particular form of cancer have, with a probability of 0.1, a particular antibody A in their blood. The most natural probability model would introduce the following items:

"Natural" model:

Ω: set of all patients with this type of cancer (suitably restricted, e.g. all patients *in Germany* who *come to hospital* – depends on the data source used for the journal article)

X: measuring whether a patient carries antibody A. This would typically be effected by a binary indicator X, i.e. the measure space E is $\{0,1\}$ and for a patient ω, $X(\omega) = 1$ iff the patient carries antibody A.

This natural model contrasts with the model that professional statisticians would use. They would set up their probability space and random variables as follows:

"Professional" model:

Ω: set of all sequences of tests for antibody A that would be carried out for one study (the original study of the journal, or some confirmation studies, or hypothetical studies of the same sort that *could* be done). One $\omega \in \Omega$ would be the suite of all such measurements done for one study. (Again, suitable restrictions would apply, e.g. to all such studies in Germany, or studies carried out in a particular year)

X_i: for $i = 1, 2, ..., X_i(\omega)$ is the i-th measurement of the sample for the study ω. Again, a standard choice for the measure space would be the indicator values $\{0,1\}$.

The variables $X_1, X_2, ...$ of the "professional" model would be assumed to be i.i.d., and they would have the same distribution as the RV X from the "natural" model. Both the "naïve" and the "professional" type of model are mathematically correct and conceptually legitimate (and either of the two could be used for answering exam questions…).

The more complicated (and I admit: somewhat less intuitive) "professional" type of model becomes a necessity when it comes to build the theory of statistical estimation – that is, to understand how one can extract an estimate of the distribution P_X from a sample $(X_1(\omega), ..., X_n(\omega)) = (x_1, ..., x_n)$. This is the core task of statistics (classical and Bayesian):

Basic task of statistics. Given a sample $(x_1, ..., x_n)$, find out something about the underlying distribution P_X – typically, give an estimate of P_X.

Parametrized distributions. Concretely, P_X is often to be represented by some (few) parameters. For instance, a normal distribution P_X is characterized by its pdf, which in turn is characterized by its mean μ and its standard deviation σ, that is, by two parameters. In our amino acid example, the distribution P_X of amino acid symbols at location 110 would be represented by 20 probability values of the various possible symbols, that is, by 20 parameters. A common symbol for the set of parameters characterizing a distribution is θ. With parametrized distributions, the basic task of statistics then spells out like this:

Basic task of statistics, formulated as parameter estimation task. Given a sample $(x_1, ..., x_n)$, give an estimate $\hat{\theta}$ of the parameters of the distribution.

Note that there are other, "parameter-free" ways of characterizing a distribution – we will soon meet some.

Estimators. Formally, the task estimating the parameters θ of a distribution from a sample can be expressed in terms of a function T_n which assigns to each sample $(X_1(\omega), ..., X_n(\omega))$ of size n a set $\hat{\theta}$ of parameters. Such functions T_n: $(X_1(\omega), ..., X_n(\omega)) \mapsto \hat{\theta}$ are called *estimators* or *estimation functions*. Note that $T_n(X_1(\omega), ..., X_n(\omega))$ is fully determined by ω, so we might also write $T_n(\omega)$ – that is, estimators are themselves random variables.

The art and science of statistics is to find "good" estimators. The art and science of (much of) algorithmical modelling is to find "good" ways of describing pdf's – an analytic expression being rather the exception than the rule, because one mostly is confronted with high-

dimensional, badly-behaved distributions for which finding an analytic pdf is all but hopeless. (And the art and science of *machine learning* is to find good estimators for a kind of pdf representation that the modellers wish to use, with a little more emphasis than in "ordinary" statistics on efficient algorithms – $T((x_1, ..., x_n))$ must be efficiently computable, that is, one looks for fast learning algorithm).

While the notion of an estimator typically refers to parametrized distributions, you may also use it in a more loosely fashion for any method that creates a characterization of a distribution from a sample.

One basic task of (the statistical branch of) algorithmical modelling. Given a piece of world (POW) that one wishes to capture in a (statistical) model, find a way to represent its distribution(s) such that

- these distributions are complex enough to capture essential aspects of the POW, and adapted to the particulars of the POW (e.g., a normal distribution would be incapable to express the most interesting aspects of a stock market index)
- these distributions are simple enough to be manipulated with efficient algorithms (aspects of memory demands, "sampling", computing probabilities of events)
- these distributions can be learned from data (the machine learning part of algorithmical modelling)

Unlike in traditional statistics, you are allowed to use a lot of raw computational power, heuristics, and borrow ideas from other fields such as physics, neurobiology, psychology, evolutionary biology, or any other.

2.2 Maximum-likelihood estimators

One of the most common approaches to design estimators is the maximum-likelihood approach. It is conceptually transparent, it is a typical "frequentist" approach, and it works well when the sample size is not too small. We will explain it with our amino acid example.

We use abbreviation *D* ("data") for the sample. The distribution estimation (or learning) task is the following:

Given: a sample *D* of *n* observations of amino acids in some location in *n* representatives of some protein class. In Fig. 2.1, we would have $n = 7$ and for instance $D =$ `H H H H H H L` (in the location shown in the second column in Fig. 2.1) or $D =$ `K K V N E G A` (last column). Another, equivalent way to write *D* is as a count vector $D = (n_1, ..., n_{20})$ where n_i is the number of counts of the *i*-th amino acid symbol in the sample.
Wanted: an estimate $\hat{\theta} = (\hat{\theta}_1,...,\hat{\theta}_{20}) = (\hat{P}_X(\texttt{A}),...,\hat{P}_X(\texttt{V}))$ of the 20 parameters describing the amino acid distribution in some location over all proteins in a family.

Approach: estimate $\hat{\theta} = (\hat{\theta}_1,...,\hat{\theta}_{20})$ such that the $P(D \mid \theta)$ is maximized over all θ, that is, put

(2.16) $T(D) = \theta^{ML} = \text{argmax}_\theta P(D \mid \theta)$.

$P(D \mid \theta)$ is called the *likelihood* of θ given D, and often written as $\mathcal{L}(\theta)$. The notion of likelihood must not be confused with the notion of probability – they are dual concepts. $P(D \mid \theta)$ is the *probability* of D given θ, and it is the *likelihood* of θ given D.

For simple frequency counts as in our example, the ML-estimator θ^{ML} can be analytically shown to be

(2.17) $$\theta^{ML} = (\theta_1^{ML},...,\theta_{20}^{ML}) = (\frac{n_1}{N},...,\frac{n_{20}}{N}),$$

where N is the sample size (here $N = 7$) and n_i is the count number of the i-th amino acid symbol in the sample.

This is beautifully simple and apparently convincing – but very inadequate for small sample sizes. Consider D = `H H H H H H L`. The maximum-likelihood estimator would assign zero probabilities to all amino acids except `H` and `L`. But every geneticist worth his/her salt would expect that in the protein family at large, every other amino acid would also occur in this location in some protein, albeit maybe rarely. But if we really assign zero probabilities to them, we would be forced to exclude every such protein from the family, which is not something we want to happen.

ML estimators of conditional distributions with Gaussian noise. There exists an intimate connection between ML estimators and the "method of least mean square (LMS) errors". We first recapitulate from High School the essentials of the LMS method. It applies in *regression tasks* where one wants to recover a deterministic input-output relationship from noisy observations of the outputs. Assume a situation where a researcher manipulates some experimental setup by subjecting it to inputs $\mathbf{x}_i \in \mathbb{R}^m$, where $i = 1, ..., N$. The researcher obtains scalar measurements y_i as a result. An example would be a psychological experiment where a graphical pattern is flashed on a screen at position $\mathbf{x}_i = (x^1_i, x^2_i)$, and a response time y_i of the subject is measured; an other example would be a medical survey where each $\mathbf{x}_i$ describes a patient by a vector of diagnostic variables, and y_i would be the remaining lifetime after diagnosis. Galileo did do the same thing when he let a heavy object fall from different heights x_i ($m = 1$ in this case) and recorded the falling times y_i (I did not check the history books of modern physics – I just guess that Galileo did something like this). Well, this is just **the** most standard situation in the empirical sciences. Now assume that the researcher knows that there is a law of nature which deterministically establishes a function $f: \mathbb{R}^m \to \mathbb{R}$, that is, on input $\mathbf{x}_i$ the "true" outcome would be $y_i = f(\mathbf{x}_i)$. The researcher even knows the nature of this function – it comes from a family parametrized by parameters $\theta \in \mathbb{R}^d$. So the researcher knows that $f = f(\theta^{true})$ for a particular parameter vector θ^{true}. But, the researcher does not know θ^{true}, and wants to estimate these parameters from his experimental data. For example, Galileo (or later, Newton) might have known that the falling time y is equal to $y = \text{sqrt}(2\, x / g)$, where g is the constant of gravitation, which would be the unknown parameter θ^{true} which he wanted to estimate from his falling experiments.

In such situations, the LMS method is to estimate the sought-after θ^{true} as the parameter vector which minimizes the mean square error of the observations, i.e. to calculate

(2.18) $$\hat{\theta}^{\mathrm{LMS}} = \arg\min_{\theta} \sum_{i=1}^{N} (f(\theta)(\mathbf{x}_i) - y_i)^2.$$

As a justification for the LMS method, you may remember from High School statements like, "we want an estimate that punishes larger deviations from the predicted true outcome more strongly than smaller deviations" – at least, that was how I was taught the LMS principle. However, there is a better and more rigorous justification for the LMS method than this. This runs as follows.

We assume that the measurement process is subject to Gaussian noise, that is, if the effective parameter is θ, then upon input $\mathbf{x}_i$ the observation y_i will be drawn from a Gaussian distribution centered on $f(\theta)(\mathbf{x}_i)$:

(2.19) $$p(y \mid \mathbf{x}_i, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(f(\theta)(\mathbf{x}_i) - y)^2}{2\sigma^2}),$$

where p is the pdf of the distribution of the y_i and σ is the standard deviation of the Gaussian (which we assume is the same for all possible inputs $\mathbf{x}$). Assuming that the observations y_i are independent, and calling the ensemble of all outcomes y_i our data D, then

(2.20) $$p(D \mid \{\mathbf{x}_i\}, \theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(f(\theta)(\mathbf{x}_i) - y_i)^2}{2\sigma^2})$$

is the likelihood of θ, or more conveniently,

(2.21) $$\mathcal{L}(\theta) = \frac{N}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (f(\theta)(\mathbf{x}_i) - y_i)^2$$

its log likelihood. Maximizing the likelihood of θ amounts to finding

(2.22) $$\hat{\theta}^{\mathrm{ML}} = \arg\max_{\theta} -\sum_{i=1}^{N} (f(\theta)(\mathbf{x}_i) - y_i)^2,$$

which is identical to (2.18). We thus find that under an assumption of Gaussian measurement noise, the LMS estimate of the true parameters is the maximum likelihood solution. A note on terminology: When statisticians speak of a "regression problem", they typically refer to exactly this situation, where the parameters of a *regression function* $f(\theta)$ are computed by a LMS calculation, with the tacit understanding that this is also the ML estimate to the extent that a Gaussian measurement noise assumption is valid.

In (very) many cases, one does not know the nature of $f(\theta)$. Then, one often resorts to the least-committing assumption that f is linear, that is, $f(\mathbf{x}) = f(\mathbf{w})(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \mathbf{x}$, where $^{\mathrm{T}}$ is matrix/vector transpose. In this context the parameters θ are typically denoted by $\mathbf{w}$, and called the (linear) *regression weights*. The LMS/ML solution $\hat{\mathbf{w}}^{\mathrm{ML}}$ can then be analytically computed in closed form via the following derivation. First observe that

(2.23) $$\hat{\mathbf{w}}^{\mathrm{ML}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} (\mathbf{w}^{\mathrm{T}} \mathbf{x}_i - y_i)^2 = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} (\mathbf{w}^{\mathrm{T}} \mathbf{x}_i)^2 - 2\mathbf{w}^{\mathrm{T}} \mathbf{x}_i y_i + y_i^2.$$

At the argmin, the gradient w.r.t. **w**

(2.24) $$\nabla_{\mathbf{w}} = \sum_{i=1}^{N} (2\mathbf{w}^{\mathrm{T}} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} - 2\mathbf{x}_i^{\mathrm{T}} y_i) = 2\mathbf{w}^{\mathrm{T}} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^{\mathrm{T}} - 2\sum_{i=1}^{N} \mathbf{x}_i^{\mathrm{T}} y_i = \mathbf{0}^{\mathrm{T}}$$

must be the all-zero row vector. Transposing this equation, and introducing the input data matrix $\Phi = (\mathbf{x}_1 \ldots \mathbf{x}_N)^{\mathrm{T}}$ and an output vector $\mathbf{y} = (y_1 \ldots y_N)^{\mathrm{T}}$, **Error! Reference source not found.** can be written as

(2.25) $$\mathbf{0} = \Phi^{\mathrm{T}} \Phi \mathbf{w} - \Phi^{\mathrm{T}} \mathbf{y},$$

which resolves to **w** as

(2.26) $$\mathbf{w} = (\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}} \mathbf{y}.$$

The matrix $(\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}}$ is known as the (left) *pseudo-inverse* of Φ. It generalizes the usual matrix inverse, which is defined only for full-rank square matrices, to full-column-rank rectangular matrices of size $a \times b$, where $a \geq b$. Indeed, it is obvious to check that $[(\Phi^{\mathrm{T}} \Phi)^{-1} \Phi^{\mathrm{T}}] \Phi = \mathbf{I}_{b \times b}$. Formula **Error! Reference source not found.** indicates one way to compute solutions to linear regression problems: first compute $(\Phi^{\mathrm{T}} \Phi)^{-1}$, then multiply with $\Phi^{\mathrm{T}} \mathbf{y}$. This is fast but prone to numerical instability when $(\Phi^{\mathrm{T}} \Phi)^{-1}$ is not well-conditioned. If you call in Matlab the routine *pinv* (for pseudo-inverse), another algorithm is used which is slower but more stable because it avoids to explicitly compute $(\Phi^{\mathrm{T}} \Phi)^{-1}$.

2.3 The bias-variance dilemma

We have just seen how a maximum-likelihood estimator can yield clearly unsatisfactory results. The problem we stumbled across is known as the *overfitting* problem, or the *bias-variance dilemma*. In fact, it is a *general* problem that *always* raises its ugly head when it comes to statistical model estimation. In a nutshell, the best model of a probability distribution that one can get, given empirical data, is *not* the model yielded by maximum-likelihood methods – because ML methods try to come as close as possible to the empirical distribution represented by the training data; as a result, the model also "models" the purely random fluctuations of the training data. A thorough treatment of the bias-variance dilemma has in the last two decades been started in a modern branch of statistics called *statistical learning theory* – more about this in my Machine Learning lecture. I will here only give a traditional account of the problem, which also explains why it is called "bias-variance" dilemma.

We consider only a special case here, which is enough to demonstrate the concept. Assume that you possess an estimator T_n: $(X_1(\omega), \ldots, X_n(\omega)) \mapsto \hat{\theta}(\omega)$, where $\hat{\theta} \in \mathbb{R}^d$. We ask the question, how much does the estimate $\hat{\theta}(\omega)$ deviate, in the mean square error sense, from the true value θ? That is, we ask for the value of $E[(\hat{\theta} - \theta)^2]$. We can compute this as follows:

$$
\begin{aligned}
E[(\hat{\theta}-\theta)^2] &= E[(\hat{\theta}-E[\hat{\theta}]+E[\hat{\theta}]-\theta)^2] \\
&= E[(\hat{\theta}-E[\hat{\theta}])^2]+E[(E[\hat{\theta}]-\theta)^2]+2E[(\hat{\theta}-E[\hat{\theta}])(E[\hat{\theta}]-\theta)] \\
&= E[(\hat{\theta}-E[\hat{\theta}])^2]+(E[\hat{\theta}]-\theta)^2 .
\end{aligned} \tag{2.27}
$$

The third term in the second line vanishes because $2(E[\hat{\theta}]-\theta)E[(\hat{\theta}-E[\hat{\theta}])] = 2(E[\hat{\theta}]-\theta)(E[\hat{\theta}]-E[E[\hat{\theta}]]) = 2(E[\hat{\theta}]-\theta)\cdot 0 = 0$. Among the remaining two terms, the first term, $E[(\hat{\theta}-E[\hat{\theta}])^2]$, gives the *variance* of the estimates, and the second, $(E[\hat{\theta}]-\theta)^2$, gives the systematic (averaged) squared amount by which the estimates deviate from the correct value. The quantity $E[\hat{\theta}]-\theta$ is the *bias* of the estimator.

Thus we have seen that the error inherent in an estimator can be split into two parts, a variance part that captures how much the estimates scatter around the mean estimate, and a bias part that quantifies how much the mean estimate differs from the correct value. The lessons taught by statistical learning theory is that there is a tension between the two: within a given class of estimators (say, neural networks) one can tune models either towards a low bias error (by data (over-)fitting, using larger networks) or towards a low variance error (by introducing a bias, e.g. small networks), but it is intrinsically impossible to optimize both simultaneously.

For an elementary demonstration of the bias-variance theme, consider a situation where $\theta = (\mu_1, \mu_2)$ is comprised of the two coordinate means of some distribution over $\mathbb{R}^2$. That is, the measure space is $E = \mathbb{R}^2$, and measurement values $X_i(\omega)$ are vectors $\mathbf{x}_i = (x_1, x_2)^T$, where superscript ^{T} denotes transpose. Now consider the following three estimators for $\theta = (\mu_1, \mu_2)$:

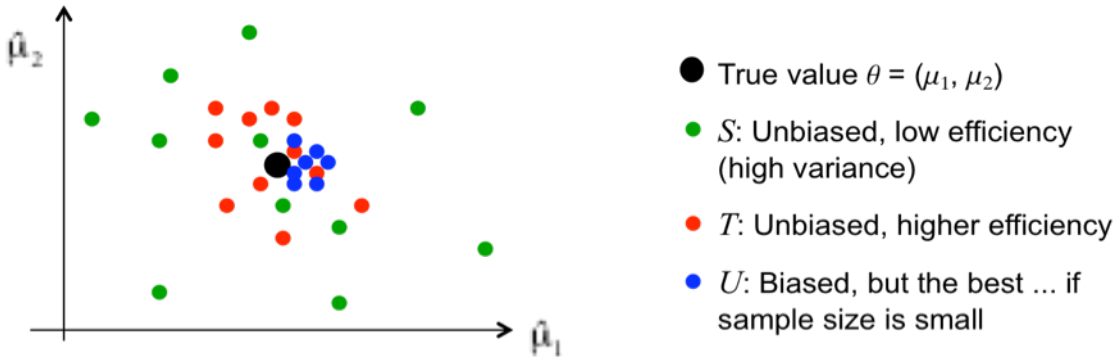
$$S : (\mathbf{x}_1,...,\mathbf{x}_N) \mapsto (\mathbf{x}_1 + \mathbf{x}_2)/2$$

$$T : (\mathbf{x}_1,...,\mathbf{x}_N) \mapsto (\mathbf{x}_1 + \ldots + \mathbf{x}_N)/N$$

$$U : (\mathbf{x}_1,...,\mathbf{x}_N) \mapsto 1/2 \cdot \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} + (\mathbf{x}_1 + \ldots + \mathbf{x}_N)/2N$$

where $(m_1, m_2)^T$ is an informed guess about true (but not precisely known) mean $(\mu_1, \mu_2)^T$. The following figure shows typical outcomes of applying these estimators to samples $(\mathbf{x}_1, ..., \mathbf{x}_N)$. It turns out that the U estimator would fare best in the sense of yielding the lowest expected error, although it is not unbiased – its estimates will be centered not around $(\mu_1, \mu_2)^T$ but around $((m_1, m_2)^T + (\mu_1, \mu_2)^T) / 2$. Intuitively speaking, it is superior because (and if) our guess $(m_1, m_2)^T$ comes close to the true value. This is a simple instance of a general principle in designing estimators: whenever one has *some* prior insight in the nature of the true parameters θ, and one finds a way to insert this knowledge into the estimator, then one may reasonably hope that the resulting estimator is better than another estimator where this prior knowledge has not been inserted. Since this prior knowledge will usually not exactly hit the correct θ, it will however introduce a *bias* into the estimator. In the next subsection 2.4, we will see how one can insert such prior information into an estimator in a principled fashion,

such that prior information in which we only weakly trust has a lower impact than prior information in which we put much trust.



2.4 An estimator with Bayesian priors

In the ML-approach, the problematic zero probability estimates occurred because the estimator *exclusively* used the information given by the sample. The background knowledge that every protein expert has, namely, that every amino acid may (albeit possibly rarely) occur at every position, was ignored. This knowledge is crucial for getting an estimator that really makes sense, and it is the starting point in a Bayesian analysis: start from the assumption ("prior") $P(\theta \mid M)$ about *the distribution of parameters*.

This needs two bits of explanation.

- The first explanation is simple: M does not refer to a measurable event [like the B in the "classical" expression $P(A \mid B)$] but simply is the Bayesian-style notation to make explicit that some background knowledge, or model, M is involved. M need not (and usually cannot) be formalized; it is a pointer to what biologists know *a priori* about distributions of amino acids in families of proteins.
- The second explanation is not so simple. The distribution $P(\theta \mid M)$ is a *hyperdistribution*: it describes how distributions (which are characterized by the various possible settings of θ) are distributed. Syntactically, it is just a distribution of numerical values (namely, the possible values of θ), but semantically, it is a distribution of distributions, because each possible value of θ represents a distribution. In our protein example, the prior wisdom that any amino acid might occur at a given site could be reflected in a choice of $P(\theta \mid M)$ which would assign a relatively high (and nonzero!) pdf value to the distribution parameter θ = (1/20, ..., 1/20).

With a Bayesion prior information $P(\theta \mid M)$, the biologist's background knowledge enters the parameter estimation as follows, through Bayes' formula:

$$P(\theta \mid D, M) = \frac{P(D \mid \theta, M)\, P(\theta \mid M)}{P(D \mid M)} = \frac{P(D \mid \theta)\, P(\theta)}{P(D)} = \frac{P(D \mid \theta)\, P(\theta)}{\int P(D \mid \theta) dP(\theta)}, \tag{2.28}$$

where the three rightmost terms are notational variants. $P(\theta \mid D, M)$ is the *posterior distribution* (of parameters) and $P(\theta \mid M)$ is the prior distribution (of distributions...) or simply the prior.

Notice that $P(\theta \mid D, M)$ is a (hyper)distribution over parameters – but the target of model estimation is some estimate value $\hat{\theta}$ of parameters, not a distribution over candidate values. Therefore, Bayesian model inference must conclude with a final step where from the distribution $P(\theta \mid D, M)$ a specific value $\hat{\theta}$ is obtained. The usual approach here is to take the mean value of θ over this distribution, that is, calculate the *mean posterior estimate*

$$\hat{\theta} = \theta^{\mathrm{PME}} = \int \theta \, P(\theta \mid D, M) \, d\theta \,. \tag{2.29}$$

We will now concretely compute (2.29) step by step for our amino acid distribution problem, where $\theta = (\theta_1, ..., \theta_{20})$.

To start, we remark that with true $\theta = (\theta_1, ..., \theta_{20})$ the sample statistics for D should follow a multinomial distribution, that is, the probability to obtain a sample $D = (n_1, ..., n_{20})$ is

$$P(D \mid \theta) = \frac{N!}{n_1! \cdots n_{20}!} \prod_{i=1}^{20} \theta_i^{n_i} \,. \tag{2.30}$$

Next we try to fix how the prior $P(\theta \mid M)$ should look like. This is a subjective decision! For reasons that will soon become clear we (and most proteinologists) opt for the Dirichlet distribution $P(\theta \mid M) = \mathcal{D}(\theta \mid \alpha)$ with parameters $\alpha = \alpha_1, ..., \alpha_{20}$. We identify $\mathcal{D}(\theta \mid \alpha)$ with its pdf, which is given by

$$\mathcal{D}(\theta \mid \alpha) = \frac{1}{Z(\alpha)} \cdot \prod_{i=1}^{20} \theta_i^{\alpha_i - 1} \cdot \delta\left(\left(\sum_{i=1}^{20} \theta_i \right) - 1 \right). \tag{2.31}$$

Some comments will help to make this formula look less frightening. The factor $1/Z(\alpha)$ is just there to ensure that the integral over $\mathcal{D}(\theta \mid \alpha)$ is one, that is it holds that

$$Z(\alpha) = \int \prod_{i=1}^{20} \theta_i^{\alpha_i - 1} \cdot \delta\left(\left(\sum_{i=1}^{20} \theta_i \right) - 1 \right) \; d(\theta) \,. \tag{2.32}$$

This integral has an explicit solution

$$Z(\alpha) = \frac{\prod_i \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)} \,, \tag{2.33}$$

where Γ is the gamma function. We don't have to understand Γ because it will later cancel out. The δ in (2.31) is the Dirac delta function which is defined by

$$\delta(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} = 0 \\ 0, & \text{if } \mathbf{x} \neq 0 \end{cases} \quad \text{and} \quad \int_{\mathbb{R}^n} \delta(\mathbf{x}) \, d\mathbf{x} = 1. \tag{2.34}$$

(Actually, and more correctly, the Dirac delta function is not a function but a *distribution* in the sense of the *theory of distributions*, a branch of functional analysis. Confusingly, in that theory the notion of a distribution is not the same as the notion of distributions as we know it from probability theory. See Appendix A for a few notes on the theory of distributions.) Equipped with (2.30) and (2.31) we return to (2.28), which we now can calculate as a pdf:

$$
\begin{aligned}
p(\theta \mid D,M) &= \frac{P(D \mid \theta,M)\; p(\theta \mid M)}{P(D \mid M)} \\
&= \frac{1}{P(D \mid M)} \frac{N!}{n_1! \cdots n_{20}!} \prod_{i=1}^{20} \theta_i^{n_i}\; Z^{-1}(\alpha) \cdot \prod_{i=1}^{20} \theta_i^{\alpha_i - 1} \cdot \delta\left((\sum_{i=1}^{20} \theta_i) - 1 \right) \\
&= \frac{1}{P(D \mid M)} \frac{N!}{n_1! \cdots n_{20}!} \frac{Z(D+\alpha)}{Z(\alpha)} \mathcal{D}(\theta \mid D+\alpha)
\end{aligned}
\tag{2.35}
$$

where $D + \alpha = (n_1+\alpha_1, ..., n_{20}+\alpha_{20})$. Because $p(\theta|D,M)$ and $\mathcal{D}(\theta \mid D+\alpha)$ are probability distribution functions, the first three multiplicative terms in the last line of (2.35) must evaluate to unity, whereby we find

$$
p(\theta|D,M) = \mathcal{D}(\theta \mid D+\alpha). \tag{2.36}
$$

Thus we have the posterior distribution of θ. In order to arrive at the posterior mean estimator, we integrate over the posterior distribution (of distributions!):

$$
\begin{aligned}
\theta_i^{\mathrm{PME}} &= \int \theta_i\; \mathcal{D}(\theta \mid D+\alpha)\; d\theta \\
&= Z^{-1}(D+\alpha) \int \theta_i \cdot \prod_{j=1}^{20} \theta_j^{n_j+\alpha_j-1} \cdot \delta\left((\sum_{i=1}^{20} \theta_i) - 1 \right)\; d\theta \\
&= \frac{Z(D+\alpha+e_i)}{Z(D+\alpha)} \\
&= \frac{\prod_j \Gamma(n_j+\alpha_j+e_i)}{\Gamma(\sum_j n_j+\alpha_j+e_i)} \cdot \frac{\Gamma(\sum_j n_j+\alpha_j)}{\prod_j \Gamma(n_j+\alpha_j)} \\
&= \frac{\Gamma(n_j+\alpha_j+e_i)}{\Gamma(n_j+\alpha_j)} \cdot \frac{\Gamma(\sum_i n_i+\alpha_i)}{\Gamma(\sum_j n_j+\alpha_j+e_i)} \\
&= \frac{n_i+\alpha_i}{N+A,}
\end{aligned}
\tag{2.37}
$$

where by $D + \alpha + e_i$ we mean $(n_1+\alpha_1+e_1\ , ..., n_{20}+\alpha_{20}+e_{20})$; $e_i(n) = 1$ if $i = n$, else 0; $A = \alpha_1 + ...+\alpha_{20}$ and in the last step we exploit $\Gamma(x+1) = x\Gamma(x)$.

We see that the posterior mean estimator $\theta_i^{\mathrm{PME}} = \frac{n_i+\alpha_i}{N+A}$ is rather similar to the maximum likelihood estimator $\theta_i^{\mathrm{ML}} = \frac{n_i}{N}$, and we can see how the parameters α_i of the Dirichlet

distribution can intuitively be interpreted as "pseudo-counts". That is, the prior knowledge is entered into the game here by augmenting the empirical counts n_i with extra pseudo-counts α_i. These pseudo-counts reflect the subjective intuitions of the biologist, and there is no rigorous rule of how to set them correctly. There are two limiting cases: if we don't add any pseudo counts, the Bayesian approach reduces the the maximum-likelihood case, that is, only the empirical information enters the estimation. This would drive us to the "far right" side in the bias-variance dilemma, that is, we run danger of overfitting. If we add, on the contrary, very large pseudo-counts, the final "estimate" will just replay the prior information with almost no influence from the empirical information; this would put us to the far left side in the bias-variance-dilemma, that is, we would just get our bias (the Bayesian prior) back. So the Bayesian approach does not solve the bias-variance dilemma; it only makes it transparent and forces the researcher to take his/her stand.

The outcome of (2.37) can be seen in yet another way, which indicates another way of how one may work in one's personal bias into a parameter estimation. Assume that according to your personal insight (before seeing the data) you expect that the parameters $\theta = (\theta_1,...,\theta_{20})$ have true values $\theta^{prior} = (\theta_1{}^{prior},...,\theta_{20}{}^{prior})$. This θ^{prior} does not incorporate any information from D and thus marks the extreme left (bias) end of the bias-variance dilemma. (Note that θ^{prior} is not a proper Bayesian prior – a proper Bayesian prior would be a distribution of distributions θ!). You compute the maximum-likelihood estimator $\theta^{ML} = (\theta_1^{ML},...,\theta_{20}^{ML}) = (\frac{n_1}{N},...,\frac{n_{20}}{N})$ from D. θ^{ML} does not reflect any prior information, fits the data perfectly and thus marks the extreme right end of the bias-variance-range. Now, in order to settle at some compromise between the two extremes, construct "blended" estimators

(2.38) $$\theta^{post} = q\,\theta^{prior} + (1 - q)\,\theta^{ML},$$

where $0 \le q \le 1$. Writing θ^{prior} as $\theta^{prior} = (\frac{\alpha_1}{A},...\frac{\alpha_{20}}{A})$ and putting $q = A / (N + A)$ and $(1 - q) = N / (N + A)$ yields the same result as (2.37). Note however that this procedure of linearly blending the parameters of a "personally expected" distribution with the parameters of a ML distribution does not universally work – not all types of parametrizations θ of distributions allow linear blending.

The biosequence analysis textbook of Durbin et al., from which this example is taken, some thought is given to how one should properly select the pseudo-counts. The proper exploitation of such "soft" knowledge makes all the difference in real-life machine learning problems.

Here is a summary of Bayesian approaches to parameter estimation for parametrized distributions:

1. Carefully choose a prior $P(\theta \mid M)$ which reflects your a-priori expert belief about how distributions θ should be distributed. If you don't know much beforehand, $P(\theta \mid M)$ should be close to uniform; if you have strong preferences for particular θ, make $P(\theta \mid M)$ peak strongly around the preferred values.
2. Make your measurements and think of a proper type of distribution (here: polynomial distribution) to obtain $P(D \mid \theta, M)$.
3. Use Bayes formula to obtain the posterior distribution of distributions, $P(\theta \mid D, M)$.
4. Integrate over $P(\theta \mid D, M)$ to find the final posterior distribution $\hat{\theta}$.

Always do step 1 before making measurements! If your choice of the prior would be influenced by what you empirically observe, the Bayesian approach becomes thoroughly flawed!

2.5 Some general remarks on estimation theory

We have seen that even in a situation as simple as estimating the probabilities of 20 symbols from a sample, several estimators T: $(x_1, ..., x_n) \mapsto \hat{\theta}$ can be considered, which have each their pro's and con's. This situation is typical and has spurred the development of an important subbranch of statistics, *estimation theory*. Estimation theory is concerned in defining general quality criteria for estimators, thus helping to compare the various estimators one might think of in a given situation. The field was pioneered by Sir Ronald Aylmer Fischer in the first half of the 20th century.

The general approach in estimation theory is to investigate the behavior of an estimator T as the number N of observations grows, that is, consider T as a sequence of related estimators T_n: $(x_1, ..., x_n) \mapsto \hat{\theta}$. Note that an estimator T_n is a random variable.

Let θ_0 denote the true distribution. Here are the most important quality criteria for estimators:

1. Unbiasedness. T is *unbiased* if for all n, $E[T_n] = \theta_0$. In our example, θ_i^{ML} was unbiased but θ_i^{PME} was not.

2. Asymptotic unbiasedness. T is *asymptotically unbiased* if $\lim_{n\to\infty} E[T_n] = \theta_0$. θ_i^{PME} is an example.

For the next quality criteria we need to consider a probability space (Ω, F, P) where each $\omega \in \Omega$ is an "experiment" in which we carry out an infinite sequence $x_1, x_2, ... = X_1(\omega), X_2(\omega)...$ of measurements (so X_i is the random variable "carry out the ith measurement within such an experiment"). $T_n(\omega)$ is then $T_n(x_1, ..., x_n)$ for $\omega = x_1, x_2, ...$Then we can define:

3. Strong consistency. T is *strongly consistent* if T_n converges to θ_0 *P-almost-surely*, that is, if

(2.39) $P(\lim_{n\to\infty} T_n(\omega) = \theta_0) = 1$

θ_i^{ML} and θ_i^{PME} are strongly consistent (by the strong law of large numbers). Explanation: The strong law of large numbers isn't actually a law but a property of a sequence $X_1, X_2, \ldots$ of numerical random variables. Such a sequence obeys the strong law of large numbers if

(2.40) $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} (X_i - E(X_i)) = 0$ P-almost surely.

It holds, for instance, if all X_i are integrable, independent, and identically distributed (a fundamental theorem of Kolmogorov). For θ_i^{ML}, we can use the law of large numbers to show(2.39) as follows. For a sample of size N, $\theta_{i,N}^{\mathrm{ML}}(\omega) = \frac{1}{N} \sum_{j=1}^{N} X_j(\omega)$, where $X_j(\omega) = 1$ if the j-th protein sequence in our sample has the i-th amino acid symbol in the location of interest, else $X_j(\omega) = 0$. The X_j are integrabel, independent, and identically distributed, so the strong law applies. The expectation of X_j is θ_i for all j. So we can conclude:

$$\begin{aligned} P(\lim_{N\to\infty} \theta_{i,N}^{ML}(\omega) = \theta_i) &= P(\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} X_j(\omega) = \theta_i) \\ &= P(\lim_{N\to\infty} (\frac{1}{N} \sum_{j=1}^{N} X_j(\omega)) - \theta_i = 0) \\ &= P(\lim_{N\to\infty} (\frac{1}{N} \sum_{j=1}^{N} X_j(\omega) - \theta_i) = 0) \\ &= 1 \end{aligned}$$

where the last equality is justified by the strong law. For θ_i^{PME} a similar argument can be used.

A background note. As we have just seen, the strong law justifies that (and how, namely with probability 1) we may interpret the limit of relative frequency counts, $\lim_{N\to\infty} \frac{1}{N} \sum_{j=1}^{N} X_j(\omega)$, as the probability of a discrete observation outcome. This is, on the one hand, the intuitive foundation of the frequentist approach to probability, but on the other hand, it is also a *derived* result within that theory. Therefore, the law of large numbers (especially the fundamental theorem of Kolmogorov) is a pillar in the frequentist theory of probability.

4. Weak consistency. T is *weakly consistent* if T_n converges to θ_0 *in probability*, that is, if

(2.41) for all $\varepsilon > 0$, $\lim_{n\to\infty} P(\{\omega \in \Omega \mid \| T_n(\omega) - \theta_0 \| > \varepsilon\}) = 0$.

Weak consistency follows from strong consistency, so our two estimators θ_i^{ML} and θ_i^{PME} are weakly consistent, too. Many estimators of great practical significance in machine learning have none of the properties 1. – 4. This is likely to happen if the estimator incorporates a nonlinear optimum finding subroutine, which for instance is the case in most neural network (widely used in pattern recognition) and hidden Markov model (widely used in speech recognition) based estimators.

5. Efficiency. These critiera 1. – 4. are all-or-none, that is, an estimator either has that property or has it not. Another kind of quality criterium asks for the relative *efficiency* of an estimator, that is, how efficiently it makes use of the information contained in a sample. The general idea is that an estimator T_n (which should be unbiased to start with) is efficient if it has small variance $\sigma^2(T_n)$, that is, if the estimates $\hat{\theta}$ returned by T_n are scattered narrowly around θ_0. An unbiased estimator S_n is more efficient than an unbiased estimator T_n, if $\sigma^2(S_n) < \sigma^2(T_n)$.

6. Sufficiency. (Here I roughly follow the book from Duda/Hart/Stork, Section 3.6). Yet another angle on judging the quality of estimators starts from the question whether the choice θ of parameters is appropriate in the first place. For our amino acid example (distribution of discrete symbols in classes) the set of class probabilities makes an obviously adequate set of parameters; when one wants to characterize a normal distribution, one chooses $\theta = (\mu, \sigma)$. But what about cases where one does not have a well-founded intuition about how one should characterize the unknown distribution with a few parameters? Assume that the unknown distribution would rightfully be described by parameters θ, but you don't know which kind of θ. This is a standard situation in practice, where you meet "wild" distributions that cannot be expected to be of any known, simple kind. So you devise of a vector $\mathbf{s}$ of parameters that you can estimate from data instead of θ, hoping that $\mathbf{s}$ contains all the relevant information about the underlying distribution. Such a set $\mathbf{s}$ of parameters that you estimate from data is called *a statistic*. Technically, a statistic is just some (possibly vector-valued) function $\mathbf{s} = \varphi(D)$. A statistic is called *sufficient* if indeed it contains all the relevant information about the underlying distribution, that is, about θ.

Intuitively, one would define $\mathbf{s}$ to be sufficient if

(2.42) $$P(\theta \mid \mathbf{s}, D) = P(\theta \mid \mathbf{s}),$$

that is, if $\mathbf{s}$ extracts from the data D all that is relevant for learning about θ. However, this would imply that θ is taken as a random variable, a perspective not common for "classical" statisticians, who therefore defined sufficiency in another way: a statistic $\mathbf{s}$ is said to be *sufficient* for θ if $P(D \mid \theta, \mathbf{s})$ is independent of θ, that is, $P(D \mid \theta, \mathbf{s}) = P(D \mid \mathbf{s})$. The two ways of defining sufficiency are equivalent. To see this, first assume the classical definition $P(D \mid \theta, \mathbf{s}) = P(D \mid \mathbf{s})$. Use the Bayesian formula to spell out $P(\theta \mid \mathbf{s}, D)$ by

(2.43) $$P(\theta \mid \mathbf{s}, D) = \frac{P(D \mid \mathbf{s}, \theta) P(\theta \mid \mathbf{s})}{P(D \mid \mathbf{s})},$$

where the r.h.s. cancels to $P(\theta \mid \mathbf{s})$ with $P(D \mid \theta, \mathbf{s}) = P(D \mid \mathbf{s})$, yielding the Bayesian-style definition. Conversely, if you assume $P(\theta \mid \mathbf{s}, D) = P(\theta \mid \mathbf{s})$, you get $P(D \mid \theta, \mathbf{s}) = P(D \mid \mathbf{s})$ by a mirrored argument (you need the extra condition $P(\theta \mid \mathbf{s}) \neq 0$.).

A fundamental theorem characterizes sufficient statistics as those statistics $\mathbf{s}$, where $P(D \mid \theta)$ can be factorized into a part that depends only on $\mathbf{s}$ and θ, and another part that depends only on D:

Theorem 2.1 (factorization theorem): A statistics **s** is sufficient for θ if there exist functions g and h such that $P(D \mid \theta) = g(\mathbf{s}, \theta)\, h(D)$.

A proof for the case of discrete distributions can be found in Duda/Hart/Stork (p. 104). The importance of the factorization theorem lies in the fact that if we want to check whether a statistics **s** is sufficient, we can restrict our analysis to the distribution $P(D \mid \theta)$ instead of having to deal with $P(D \mid \theta, \mathbf{s})$. If $D = (x_1, ..., x_n)$, and the individual measurements are statistically independent, $P(D \mid \theta)$ takes the simple form of

$$P(D \mid \theta) = \prod_{k=1}^{n} P(x_k \mid \theta). \tag{2.44}$$

Specifically, the factorization theorem teaches us that a sufficient statistic depends only on the probabilties $P(x_k \mid \theta)$ and not on (inaccessible) assumptions on a prior $P(\theta)$.

Exponential distributions. The factorization theorem and Eq. (2.44) can be applied particularly well if we are dealing with parametric probability distributions from the *exponential* familiy. This family includes most of the standard textbook distributions, for instance the normal, exponential, Poisson, Gamma, Beta, Bernoulli, binomial and multinomial distributions. Exponential distributions are characterized by a pdf of the form

$$p(\mathbf{x} \mid \theta) = \alpha(\mathbf{x}) \exp\left[\mathbf{a}(\theta) + \mathbf{b}(\theta)^{\mathsf{T}}\, \mathbf{c}(\mathbf{x})\right], \tag{2.45}$$

where **a**, **b**, **c** are linear functions.

For exponential distributions, one gets a sufficient statistic by

$$\mathbf{s} = \frac{1}{n} \sum_{k=1}^{n} \mathbf{c}(x_k), \tag{2.46}$$

and the two factorizing functions as

$$g(\mathbf{s}, \theta) = \exp[n(\mathbf{a}(\theta) + \mathbf{b}(\theta)^{T}\, \mathbf{s})], \quad h(D) = \prod_{k=1}^{n} \alpha(x_k). \tag{2.47}$$

A table containing an overview of all these expressions for a dozen or so much-used distributions is shown in Duda/Storck/Hart p. 108-109.

Once one has a sufficient statistic $\mathbf{s} = \varphi(D)$ and the factorization functions, given a sample $\mathbf{x} = (x_1, ..., x_n)$ one can find the maximum likelihood estimator θ^{ML} through

$$
\begin{aligned}
\theta^{ML}(\mathbf{x}) &= \arg\max_{\theta} P(\mathbf{x} \mid \theta) \\
&= \arg\max_{\theta} g(\mathbf{s}, \theta) h(\mathbf{x}) \\
&= \arg\max_{\theta} g(\mathbf{s}, \theta) \\
&= \theta'^{ML}(\mathbf{s}).
\end{aligned}
\tag{2.48}
$$

For the various exponential distributions, the sufficient statistics and the factorization functions *g* are significantly simpler than the original formulae for the pdfs. Because very many distributions of practical relevance are exponential, the tools of sufficient statistics and factorization are of great practical importance.

3 Elementary methods for representing and estimating pdfs

In this section we will learn about a choice of basic approaches to formally specify pdfs (a precondition to run algorithms on them) and to estimate such pdfs from sample data.

In *parametric* methods, a pdf is represented by a closed analytical formula that typically needs only a few parameters to be fully specified. An example is the univariate normal (= Gaussian) distribution which is specified by its mean μ and its standard deviation σ. We will treat the multivariate normal distribution and see how its parameters can be estimated from sample data.

In *nonparametric* methods, no assumption is made concerning a specific analytical form of the pdf. Instead, the sample data themselves determine the shape of the pdf. A simple example is histograms. We will learn about a more involved approach, *Parzen windows.*

The distinction between parametric and nonparametric methods is not clear-cut in statistics. We will learn about a *semi-parametric* method that has features of both, *Gaussian mixtures.* Estimating Gaussian mixture models from data can be effected by a variety of computational methods (none trivial). We will take a look at *gradient-descent* methods and the *expectation-maximization* algorithm. All of these are general-purpose techniques that can be applied in many other situations besides Gaussian mixture estimation.

I lean closely on the treatment of this subject given in Chapter 2 of Bishop (1995), quoting some passages verbatim without stating that in each case.

3.1 Multivariate Gaussians

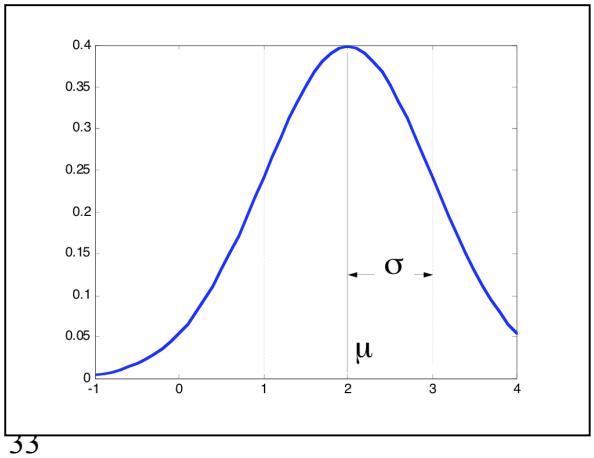
Univariate Gaussians

You know the one-dimensional (univariate) normal distribution $\mathcal{N}(\mu, \sigma)$, which is given by the pdf

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{3.1}$$

is fully specified by its mean μ and standard deviation (square root of variance) σ, has the famous bell-shape with μ being the location of the maximum and $\mu \pm \sigma$ being the locations of the zeros of the second derivative (Figure 3.1).

Figure 3.1 pdf of a normal distribution with mean 2 and standard deviation 1.



Make sure you are aware of the distinction between the Gaussian *distribution*, which is a function from the Borel σ-field $\mathcal{B}$ to [0,1] and denoted by $\mathcal{N}(\mu, \sigma)$, vs. its *pdf*, which is a function from the reals to the non-negative reals and denoted by p (we may also write $p_{\mu, \sigma}$ if we want to point out the underlying distribution parameters). Disturbingly, this distinction between distribution and pdf is not made in some textbooks, specifically, not in Bishop (2006), a source of much confusion.

The normal distribution is so fundamental because by the central limit theorem the mean of independent numerical random variables $X_1, ..., X_n$ converges to a normal distribution. This is however a sloppy formulation, and you might be curious to learn about a rigorous formulation of this foundational tool for many applications in statistics.

Definition a.. Let $(X_i)_{i \in \mathbb{N}}$ be a sequence of independent, real-valued, square integrable random variables with nonzero variances $V(X_i) = E[(X_i - E[X_i])^2]$. Then we say that *the central limit theorem holds* for $(X_i)_{i \in \mathbb{N}}$ if the distributions P_{S_n} of the *standardized sum variables*

$$S_n = \frac{\sum_{j=1}^{i}(X_j - E[X_j])}{\sigma(\sum_{j=1}^{i}(X_j))} \tag{3.2}$$

converges weakly to $\mathcal{N}(0, 1)$.

Explanations.

- A real-valued random variable with pdf p is *square integrable* if its *second moment*, that is the integral $E[X^2] = \int_\Omega X^2(\omega)\, dP = \int_{\mathbb{R}} x^2 P_X(dx) = \int_{\mathbb{R}} x^2 p(x) dx$ is finite. (Notice the notational variants.)
- If $(P_i)_{i \in \mathbb{N}}$ is a sequence of distributions over $\mathbb{R}$, and P a distribution (all over the same measure space $(\mathbb{R}, \mathcal{B})$), then $(P_i)_{i \in \mathbb{N}}$ is said to *converge weakly* to P if

  $$\lim_{i \to \infty} \int f(x) P_i(dx) = \int f(x) P(dx) \tag{3.3}$$

  for all continuous, bounded functions $f: \mathbb{R} \to \mathbb{R}$. Here the σ-field $\mathcal{B}$ is the *Borel* σ-field, which is the most commonly encountered σ-field over $\mathbb{R}$. It is *generated* by all open intervals $(a,b) \subset \mathbb{R}$, that is, $\mathcal{B}$ is the smallest σ-field over $\mathbb{R}$ which contains all these intervals.
- The "central limit theorem" is actually not a theorem but a *property* that some sequence of random variables may or may not have (just like the "law of large numbers" isn't a law but a property, see Section 2.5).

A sequence $(X_i)_{i \in \mathbb{N}}$ of random variables (or, equivalently, its associated sequence of distribution $(P_{Xi})_{i \in \mathbb{N}}$) obeys the central limit theorem under rather weak conditions – or in other words, for many such sequences the central limit theorem holds.

A simple, important class of $(X_i)_{i \in \mathbb{N}}$ for which the central limit theorem holds is obtained when the X_i are identically distributed (and, of course, are independent, square integrable and have nonzero variance). Notice that *regardless* of the shape of the distribution of each X_i, the distribution of the normalized sums converge to $\mathcal{N}(0, 1)$!

However, this simple case does not explain the far-reaching, general importance of the central limit theorem (rather, property). In textbooks one often finds statements like, "*if the outcomes of some measurement procedure can be conceived to be the combined effect of many independent causal effects, then the outcomes will be approximately normal distributed*". The "many independent causal effects" that are here referred to are the random variables $(X_i)_{i \in \mathbb{N}}$; they will typically *not* be identically distributed. Still the central limit theorem holds under mild assumptions. Intuitively, all that one has to require is that none of the individual random variables X_i dominates all the others – the effects of any single X_i must asymptotically be "washed out" if an increasing number of other $X_{i'}$ is entered into the sum variable S_n. In mathematical textbooks on probability you may find numerous mathematical conditions which amount to this "washing out". A special case that captures many real-life cases is the condition that the X_i are uniformly bounded, that is, there exists some $b \in \mathbb{R}$ such that $X_i(\omega) < b$ for all i and ω. However, there exist much more general (nontrivial to state) conditions that likewise imply the central limit theorem. For our purposes, the take-home message is,

> if $(X_i)_{i \in \mathbb{N}}$ is a sequence of independent, square integrable, non-zero-variance random variables, none dominating all others – where the individual X_i may each be individually arbitrarily distributed –, then the normalized sums converge to a normal distribution.

Multivariate Gaussians: geometric properties of the pdf

So much for a recap of the univariate Gaussian distribution. We now turn to the multivariate case, which describes distributions in $\mathbb{R}^d$, where $d > 1$, that is, the distribution of a vector-valued random variable $\boldsymbol{X}: \Omega \to \mathbb{R}^d$, $\omega \mapsto (X_1(\omega), ..., X_d(\omega))^\mathsf{T} = (x_1, ..., x_d)^\mathsf{T} = \mathbf{x}$ with *component variables* $X_1, ..., X_d$ (superscript $^\mathsf{T}$ marks vector/matrix transpose). The pdf of a multivariate Gaussian is of the form

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\mathsf{T} \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right), \tag{3.4}$$

where the mean $\boldsymbol{\mu} = E[\mathbf{x}]$ is now a vector, Σ is the $d \times d$ *covariance matrix*

$$\begin{aligned} \Sigma &= (\mathrm{cov}(X_i, X_j))_{i,j=1,...,d} = (E[(X_i - E[X_i])(X_j - E[X_j]])_{i,j=1,...,d} \\ &= E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\mathsf{T}], \end{aligned} \tag{3.5}$$

and $|\Sigma|$ is the determinant of Σ. From (3.5) we see that Σ is a symmetric matrix, and therefore has $d(d+1)/2$ independent components. There are also d independent components in $\boldsymbol{\mu}$, so the pdf p is specified by $d(d+3)/2$ parameters. For illustration, here is a surface plot of a bivariate Gaussian with $\boldsymbol{\mu} = (0.5, 1)^\mathsf{T}$ and $\Sigma = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}$:

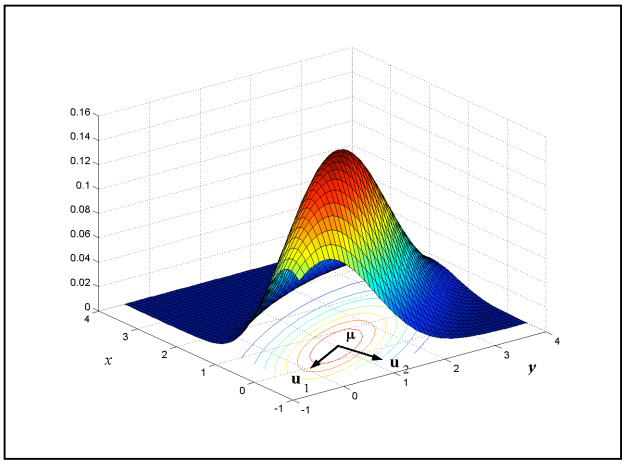


Figure 3.2: pdf of a bivariate Gaussian distribution. $\mathbf{u}_1$ and $\mathbf{u}_2$ are the eigenvectors of Σ. They mark the directions of the principal axes of the contour ellipsoids. The eigenvectors are $\mathbf{u}_1 = (0.52, -0.85)$ and $\mathbf{u}_2 = (-0.85, 0.52)$, with inverse eigenvalues $1/\lambda_1 = 0.38$ and $1/\lambda_2 = 2.62$. Notice that $\text{sqrt}(1/\lambda_1) = 0.62$ and $\text{sqrt}(1/\lambda_2) = 1.62$ – thus the main axes of the contour ellipsiods will have relative extensions 0.62 vs. 1.62.

The geometric shape of a multivariate Gaussian can be best understood when one considers the contour sets $\mathcal{E}_c = \{\mathbf{x} \in \mathbb{R}^d \mid p(\mathbf{x}) = c\}$ of arguments where the pdf has some constant value c. For bivariate Gaussians, the $\mathcal{E}_c$ are ellipses, and in the general case they are d-dimensional hyperellipsoids. The shape of these ellipsoids is determined by properties of Σ. The key to understanding Gaussians is to understand the algebraic properties of Σ, which we will now investigate. Matrices with similar properties as Σ arise frequently in statistics, signal processing and machine learning, therefore our effort has a far greater benefit than "just" understanding Gaussians. I follow a similar treatment of correlation matrices[2] given in Farhang-Boroujeny (1999).

Proposition 3.1. If Σ is nonzero, then it is positive definite, that is, for any nonzero vector $\mathbf{v} \in \mathbb{C}^d$ it holds that $\mathbf{v}^H\Sigma\mathbf{v} > 0$. (Here the superscript ^{H} denotes the conjugate transpose of a vector or matrix).

Proof. We first show that $\mathbf{v}^H\Sigma\mathbf{v}$ is real-valued. This follows from $(\mathbf{v}^H\Sigma\mathbf{v})^* = (\mathbf{v}^H\Sigma\mathbf{v})^H = (\mathbf{v}^H\Sigma^H\mathbf{v}^{HH}) = (\mathbf{v}^H\Sigma\mathbf{v})$, where * denotes conjugates of complex scalars. To show that $\mathbf{v}^H\Sigma\mathbf{v}$ is positive, insert (3.5):

$$\mathbf{v}^H\Sigma\mathbf{v} = \mathbf{v}^H E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T]\mathbf{v} = E[\mathbf{v}^H(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{v}]. \tag{3.6}$$

We note that $\mathbf{v}^H(\mathbf{x}-\boldsymbol{\mu})$ and $(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{v}$ form a pair of complex-conjugate scalars, from which we conclude

$$\mathbf{v}^H\Sigma\mathbf{v} = E[|\,\mathbf{v}^H(\mathbf{x}-\boldsymbol{\mu})\,|^2], \tag{3.7}$$

which is non-negative for any nonzero vector $\mathbf{v}$.

Proposition 3.2. The eigenvalues of Σ are all real and positive. Furthermore, the eigenvectors of Σ can be chosen to be real.

Proof. Consider an eigenvector $\mathbf{u}$ of Σ and its corresponding eigenvalue λ:

[2] Be aware of the distinction between *covariance* matrices $E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T]$ and *correlation* matrices $E[\mathbf{x}\,\mathbf{x}^T]$. Both kinds of matrices share however the algebraic properties presented here.

(3.8) $\Sigma \mathbf{u} = \lambda \mathbf{u}$.

Premultiplying by $\mathbf{u}^H$ and noting that λ is a scalar we get

(3.9) $\mathbf{u}^H \Sigma \mathbf{u} = \lambda \mathbf{u}^H \mathbf{u}$.

The quantity $\mathbf{u}^H \mathbf{u}$ on the rhs. is real and positive since it is the squared length of the vector $\mathbf{u}$. Because Σ is positive definite, $\mathbf{u}^H \Sigma \mathbf{u}$ is positive and real. Therefore, λ is positive and real.

Let $\mathbf{u} = \mathbf{v} + i\,\mathbf{w}$, where $\mathbf{v}$ and $\mathbf{w}$ are real vectors, be an eigenvector of Σ. Then from

(3.10) $\Sigma \mathbf{v} + i\,\Sigma \mathbf{w} = \Sigma (\mathbf{v} + i\,\mathbf{w}) = \lambda\ (\mathbf{v} + i\,\mathbf{w}) = \lambda\ \mathbf{v} + i\,\lambda\,\mathbf{w}$

it follows that $\Sigma \mathbf{v} = \lambda\ \mathbf{v}$, that is, an eigenvector can be chosen to be real.

Because the eigenvectors of Σ can be assumed real, their Hermitians equal their transposes, which we will henceforth use.

Proposition 3.3. If $\mathbf{u}_1$ and $\mathbf{u}_2$ are two eigenvectors of Σ with eigenvalues $\lambda_1 \neq \lambda_2$, (and we can assume that $\mathbf{u}_1$ and $\mathbf{u}_2$ are real) then $\mathbf{u}_1^T \mathbf{u}_2 = 0$, in other words, $\mathbf{u}_1$ and $\mathbf{u}_2$ are orthogonal.

Proof. We have

(3.11) $\Sigma \mathbf{u}_i = \lambda_i \mathbf{u}_i$ for $i = 1, 2$.

Applying the transpose on both sides and noting that the λ_i are scalars and that $\Sigma^T = \Sigma$, we obtain

(3.12) $\mathbf{u}_i^T \Sigma = \lambda_i \mathbf{u}_i^T$ for $i = 1, 2$.

Combining (3.11) and (3.12), we get

(3.13) $(\lambda_1 - \lambda_2)\mathbf{u}_1^T \mathbf{u}_2 = 0$,

which implies our claim because $(\lambda_1 - \lambda_2) \neq 0$.

If there are eigenvectors $\mathbf{u}_i, \ldots, \mathbf{u}_{i+k}$ with equal eigenvalues λ, they are not unique: each linear combination of them yields another eigenvector to the same eigenvalue. However, a little algebra reveals that one may choose them to be orthogonal as well. Therefore, one may find a matrix $\mathbf{U}$ that contains an orthonormal (normalized to norm 1 and orthogonal) set of eigenvectors of Σ in its columns. Let $\mathbf{\Lambda}$ be the matrix

(3.14) $$\mathbf{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & & 0 \\ 0 & \lambda_2 & & 0 \\ & & \ddots & \\ 0 & & & \lambda_d \end{pmatrix}$$

that contains the eigenvalues of the corresponding eigenvectors in $\mathbf{U}$ on its diagonal. Then Σ can be decomposed as

$$\Sigma = \mathbf{U}\,\Lambda\,\mathbf{U}^{\mathsf{T}}, \tag{3.15}$$

which follows from packing together the equations (3.8) into a matrix equation $\Sigma\,\mathbf{U} = \mathbf{U}\,\Lambda$ and observing $\mathbf{U}^{\mathsf{T}} = \mathbf{U}^{-1}$.

The inverse of Σ is

$$\Sigma^{-1} = \mathbf{U}\,\Lambda^{-1}\,\mathbf{U}^{\mathsf{T}}, \tag{3.16}$$

as can be verified by a simple calculation. Thus, the eigenvectors of Σ^{-1} are identical to the eigenvectors of Σ, with the eigenvalues of Σ^{-1} being the inverses $1/\lambda_i$ of the eigenvalues λ_i of Σ. The matrix Σ^{-1} appears in the formula of the multivariate Gaussian in the context of the quadratic form

$$(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}) = \mathbf{v}^{\mathsf{T}}\Sigma^{-1}\mathbf{v}, \tag{3.17}$$

where $\mathbf{v}$ is the centered version of the random variable $\mathbf{x}$. Let $\mathbf{v} = (v_1, ..., v_d)$ and notice that $(v'_1, ..., v'_d) = \mathbf{v}' = \mathbf{U}^{\mathsf{T}}\mathbf{v}$ transforms $\mathbf{v}$ into the coordinate system spanned by the eigenvectors from $\mathbf{U}$. Now consider

$$\mathbf{v}^{\mathsf{T}}\Sigma^{-1}\mathbf{v} = \mathbf{v}^{\mathsf{T}}\mathbf{U}\Lambda^{-1}\mathbf{U}^{\mathsf{T}}\mathbf{v} = \mathbf{v}'^{\mathsf{T}}\,\Lambda^{-1}\mathbf{v}' = \sum_{i=1}^{d}\lambda_i^{-1}v'^2_i. \tag{3.18}$$

We find that the quadratic form $\mathbf{v}^{\mathsf{T}}\Sigma^{-1}\mathbf{v}$ decomposes into a sum of squares of the components of $\mathbf{v}'$, weighted with λ_i^{-1}. Because the components v'_i of $\mathbf{v}'$ are the projections of $\mathbf{v}$ on the eigenvectors in $\mathbf{U}$, we can now interpret the shape of $\mathbf{v}^{\mathsf{T}}\Sigma^{-1}\mathbf{v}$ geometrically as a paraboloid opened upwards, with the main axes lying in the directions of the $\mathbf{u}_i$, and the corresponding curvatures being proportional to the inverse eigenvectors λ_i^{-1}.

Finally, for some constant C let us consider $\lambda_i^{-1}v'^2_i = C$, which is equivalent to $v'_i = \sqrt{C}\sqrt{\lambda_i^{-1}}$. From this we see that the elliptic contour lines shown in Figure 3.2 will extend in the direction of an eigenvector $\mathbf{u}_i$ by amounts proportional to $\sqrt{\lambda_i^{-1}}$.

Multivariate Gaussians: Model estimation

A multivariate Gaussian is characterized by the parameters contained in $\boldsymbol{\mu}$ and Σ, that is, by the expectations $E[\boldsymbol{X}] = (E[X_1] \ldots E[X_d])^{\mathsf{T}}$ and $E[(X_i - E[X_i])(X_j - E[X_j])]_{i,j=1,\ldots,d}$. Our task thus amounts to estimate means and covariances from numerical sample data. These are of course well-studied basic problems in statistics, and we simply rehearse the solution.

Let $D = (\mathbf{x}^1, ..., \mathbf{x}^N)^\mathsf{T} = (x_{ij})_{i=1,...,N;\, j=1,...,d}$ be a sample, written in the customary fashion, that is, each observation is a d-dimensional row in an obsveration matrix D. Then the maximum-likelihood estimate of $(E[X_1] \; ... \; E[X_d])$ is simply the column mean of D:

$$(3.19) \qquad \boldsymbol{\mu}^{\mathrm{ML}} = (E[X_1] \; ... \; E[X_d])^{\mathrm{ML}} = 1/N \; \mathbf{1} \; D,$$

where $\mathbf{1}$ denotes the row vector of N ones. The maximum-likelihood estimate of Σ can be shown to be

$$(3.20) \qquad \Sigma^{\mathrm{ML}} = \frac{1}{N} \sum_{n=1,...N} (\mathbf{x}^n - \hat{\boldsymbol{\mu}})(\mathbf{x}^n - \hat{\boldsymbol{\mu}})^\mathsf{T} = 1/N \; D'^\mathsf{T} D',$$

where $D' = (\mathbf{x}^1 - \hat{\boldsymbol{\mu}}, ..., \mathbf{x}^N - \hat{\boldsymbol{\mu}})^\mathsf{T}$ is the normalized data matrix. However, the maximum-likelihood estimate is not unbiased (it systematically underestimates covariances). One gets an unbiased estimate of Σ via

$$(3.21) \qquad \hat{\Sigma} = 1/(N-1) \; D'^\mathsf{T} D',$$

a result that is proven in all elementary textbooks on statistics.

Further properties of multivariate Gaussians

Uncorrelatedness implies independence. A property that distinguishes Gaussians from generic multivariate distributions is that uncorrelatedness of the component variables $X_1, ..., X_d$ implies their independence. This can be seen as follows. If the variables $X_1, ..., X_d$ are uncorrelated, then Σ^{-1} is a diagonal matrix with the inverse variances $1/\sigma_i^2$ of the variables on the diagonal (easy exercise), and the multivariate Gaussian pdf decomposes as

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{(2\pi)^{d/2} \, |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\mathsf{T} \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right) \\ &= \frac{1}{\prod_{i=1...d} \sqrt{2\pi}\sigma_i} \exp\left(\sum_{i=1...d} -\frac{1}{2} \frac{(x_i - \mu_i)^2}{\sigma_i^2} \right) = \prod_{i=1...d} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{1}{2} \frac{(x_i - \mu_i)^2}{\sigma_i^2} \right) \\ &= \prod_{i=1...d} p(x_i) \end{aligned} \qquad (3.22)$$

into a product of distributions (which are again Gaussian) of the component variables.

Invariance under linear transformations. Linear transformations $\mathbf{A}: \mathbb{R}^d \to \mathbb{R}^h$ transform a d-dimensional random variable $\boldsymbol{X}$, which is normal distributed according to $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, into a h-dimensional random variable $\mathbf{A}\boldsymbol{X}$ which is normal distributed according to $\mathcal{N}(\mathbf{A}\boldsymbol{\mu}, \mathbf{A}\Sigma\mathbf{A}^\mathsf{T})$. As we have seen above, an important special case is obtained for $\mathbf{A} = \mathbf{U}$: the resulting new coordinate variables $x'_i = \mathbf{A}\mathbf{x}[i]$ are uncorrelated and therefore independent. That is, any multivariate Gaussian distribution can be interpreted, up to a linear coordinate transformation, as being generated by d independent, univariate Gaussian distributed random variables.

Invariance under marginal and conditional densities. The marginal densities, obtained from a multivariate Gaussian pdf by integrating away some components, are themselves normal. Similarly, the conditional densities, obtained by setting some of the components to fixed values, are normal. (Make sure you have a clear perception of the distinction between marginal and conditional densities!)

All these pleasant properties taken together with the fact that Gaussians can be understood as limiting distributions of virtually any sum of distributions, it becomes clear why the normal distribution plays such a central role in statistics.

3.2 Mixtures of Gaussians

Clearly, assuming a Gaussian distribution – univariate or multivariate – is not always appropriate. Consider the following univariate distribution, depicting the midterm point distribution of a bygone IUB course in a coarse barchart graphics:

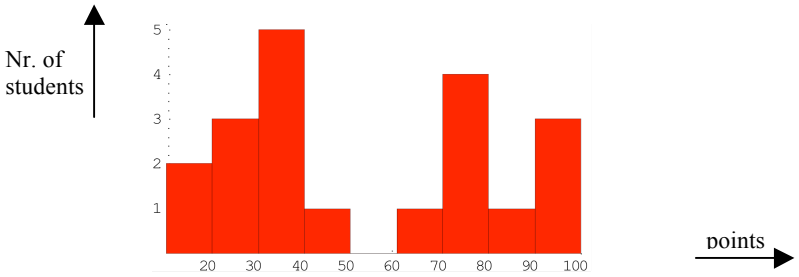


Figure 3.3 An un-normal distribution.

Cases like this – and vastly more complicated ones – are the rule, not the exception, when non-trivial real-life systems are statistically modelled. The question arises, how can such "wild" distributions be approximately represented by a few parameters? One very standard approach is to cover the observation space by a number of weighted Gaussians. Given the apparent "two-bump" appearance of the empirical distribution from Figure 3.3., it seems natural to represent it as a sum of two Gaussians. A pdf made of a *mixture* of two Gaussians for the above example might be

(3.23) $p(x) = 0.6 * p_{30,10}(x) + 0.4 * p_{80,10}(x)$,

where $p_{30,10}$ denotes the pdf of a normal distribution with mean 30 and standard deviation 10. More generally, a *mixture of Gaussians* over $\mathbb{R}^d$ is a distribution that has a pdf of the kind

(3.24) $$p(\mathbf{x}) = \sum_{j=1}^{M} p(\mathbf{x} \mid \theta_j) P(j),$$

where $\theta_j = (\boldsymbol{\mu}_j, \Sigma_j)$ are the parameters of a d-dimensional Gaussian with pdf $p(\mathbf{x} \mid \theta_j)$, and where the *mixture coefficients* $P(j)$ satisfy

(3.25) $$0 \le P(j) \le 1 \quad \text{and} \quad \sum_{j=1}^{M} P(j) = 1.$$

We often prefer to write $p(\mathbf{x}) = \sum_{j=1\ldots M} p(\mathbf{x} \mid j)P(j)$ as a shorthand for (3.24).

Given a mixture model of the kind (3.24), one can construct a generative algorithm that produces samples $\mathbf{x}^i$ from that mixture distribution, in a two-step procedure:

1) Randomly choose one of the M component Gaussians according to the mixture probabilities $P(j)$.
2) If in step 1, the k-th component has been selected, produce a random point from the Gaussian distribution $p(\mathbf{x} \mid \theta_j)$. (How this can be implemented will be explained in Section 4 of these lecture notes).

This two-step routine is an instance of a *sampling algorithm*. Such algorithms produce data points which are distributed according to some given distribution (here: mixture of Gaussians). We will soon see (Section 4) that finding good sampling algorithms can be quite challenging.

This all seems natural and easy enough, but we immediately find ourselves in the midst of truly difficult questions:

1. Given a sample D and assuming we want to model it by a mixture of Gaussians, what is the appropriate number M of mixture components?
2. Given a sample D and assuming we already "know" which M is suitable, how can we find an estimate (maximum likelihood or Bayesian) of the involved parameters of the component Gaussians?

It will turn out that the second question is the easier one, and the first is actually so difficult that we will not present approaches to solving it here[3]. Instead I will only highlight the deeply problematic nature of the first question by a brief discussion.

If you look at Figure 3.3., you might be tempted to say, "well, in this case it seems *obvious* that we should use *two* Gaussians because we clearly see two (noisy) humps." But is this really so? Consider the following two interpretations of the empirical midterm point distribution through mixtures of Gaussians, one with two and and the other with 8 Gaussians:

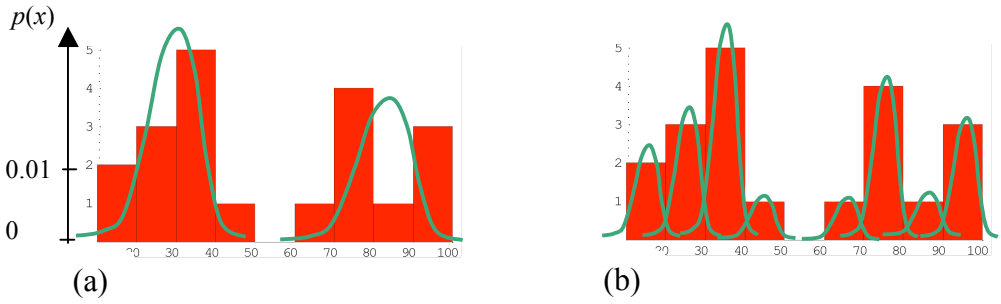


Figure 3.4 Two ways of capturing an empirical distribution by a mixture of Gaussians.

[3] Visit my lecture on Machine Learning for more on this topic – in fact, this difficulty is an instance of *the* fundamental problem of Machine Learning, called the *bias-variance dilemma*, for which only recently a satisfactory mathematical theory has started to be developed (namely, *statistical learing theory*).

Now, version (a) looks "somehow more reasonable" – but it is not quite clear what that means – but then again, if you think about it you will find that the likelihood of a model like (b) is higher than that of (a). So, are maximum-likelihood models not always preferable? The comment you might get from a machine learning expert would go like this: *"Using an increasing number of mixture components, any distribution can be approximated arbitrarily well. However, the models become more and more contrived and complicated, and they model more and more not the true underlying distribution, but the random variations from the sample (overfitting). Instead, you should* bias *the model selection towards simpler models – they are not so prone to overfitting. Read the books by Vapnik, the father of modern statistical learning theory, to understand what is going on here – but frankly, I don't quite understand them completely myself and the theoretical findings don't seem to be too helpful in practice – well if you ask* me*, I go for Bayesian approaches or if I don't have the time for that, I simply prefer simple models..."*

Another fundamental problem of machine learning that has a bearing on finding an appropriate number M of mixture components is known as the *curse of dimensionality*. To appreciate this problem, you should first notice that in very many practical modelling tasks, the sample data vectors $\mathbf{x}_1$ are high-dimensional – in the sense that they are not of length 3 or 5, but of length 50 or more. For instance, customer profile data typically "measure" a customer by several dozens of parameters – age, sex, maritial status, income etc. to start with, followed by many figures describing what products have been bought by the customer in the past, whether s/he paid the bills swiftly, website clicking profile information, etc. (You would probably be amazed to learn about your profile at ebay or amazon!). Or consider the large number of sensors attached to some complex physical, chemical or biochemical experiment, which result in an equally large size of data vectors. Or consider the wealth of information that is gathered in the central surveillance unit of a power plant (or even of a modern car engine).

So, let us consider how an increasing dimension of data vectors affects the estimation of a distribution. Figure 3.5a shows a 3D histogram of a sample of size 50 that was drawn from a 3-dimensional mixture of $M = 3$ similar Gaussians with mixture coefficients $P(1) = P(2) = P(3) = 1/3$, where the means were $\boldsymbol{\mu}_1 = (0, 0, 0)$, $\boldsymbol{\mu}_2 = (0.5, 0, 0)$, $\boldsymbol{\mu}_3 = (1, 0, 0)$, and the covariance matrix was diag(0.15, 0.5, 0.5) for all three components. Figure 3.5b shows a projection of the histogram on the (x, y)-plane, and Figure 3.5c a projection on the x-axis.

The most conspicuous thing you can see in Fig. 3.5a is that you can't see any conspicuous thing – certainly you can't guess from the histogram that it was sampled from a mixture of three Gaussians. Similarly, the 2D projection shows no visually discernible structure of being made from 3 (now 2-dimensional) Gaussians. Only the 1-dimensional projection can visually be interpreted as being composed from three humps.

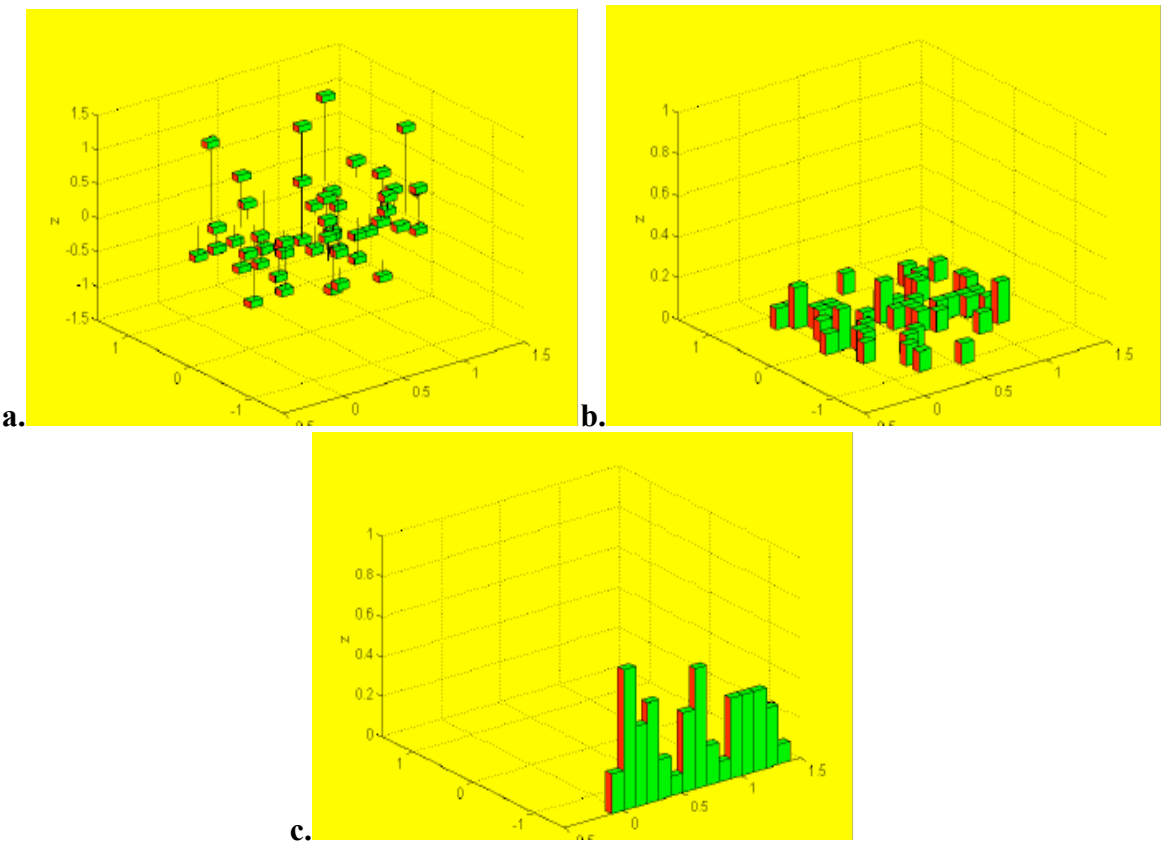


Figure 3.5 a. A sample of size 50 from a mixture of three 3D Gaussians. Each little box represents one data point from the sample. Blue lines indicate position of sample point over $z = 0$ surface. **b.** Projection on (x, y)-plane, **c.** projection on the x-axis. Height of boxes in **a.**, **b.** and **c.** indicate number of sample points that fell into the cubic (square, linear) interval of side length 0.1 of $\mathbb{R}^d$ ($d = 3, 2, 1$; a heigth increment of 0.1 corresponds to one data point).

Roughly speaking, the "curse of dimensionality" refers to the fact that in high-dimensional data spaces, one cannot locally estimate probability densities by "bin counting". In the 1D figure above, you get histogram bars of different heights, indicative of different probability densities along the x-axis. In the 2D histogram, only a few bar locations mark hits of the bins by more than a single data point. Finally, in the 3D histogram, most bin cells are not hit by a data point at all; the cells that mark counts all catch only a single data point.

Why is this? If you wish to estimate densities from a sample in (say) the d-dimensional unit hypercube with a metric resolution (bin side length) of 0.1, then you have to consider 10^d many bins. Now assume that you wish to ascertain a minimal average accuracy of bin frequency estimates by having (on average) 5 hits per bin (as in the 1D example). This implies that you need $5 * 10^d$ data points in your sample. In other words, for a given metric resolution and statistical accuracy, the number of required data points grows exponentially with the data dimension d. Of course, with 70-dimensional customer profile data you can't ever hope to get anywhere close to having 10^{70} customers at your disposal. You feel helpless? Good. Come to my Machine Learning lecture next year to learn about general coping strategies against the curse of dimensionality... If you ask a Machine Learning expert for advice, you will hear something like *"The key to the curse of dimensionality is to* reduce *the dimension of data by a*

kind of preprocessing that is called feature extraction*. Instead of using the raw, high-dimensional data you transform each sample point into a low-dimensional* feature vector *first, and then do your statistics on those. The catch is, of course, to find* good *features that don't lose much information. Please consult the current research literature..."* In this example from Figure 3.5, a good 1-dimensional feature is the *x*-coordinate of the 3-dimensional distribution. However, I knew this beforehand because I knew how the 3-dimensional distribution was specified. This precious knowledge is not available in real-life scenarios – it is the very purpose of modelling and Machine Learning to *find out* about an *unknown* distribution. If I would have chosen the *y*-coordinate as a 1-dimensional feature, it would not have revealed anything about the 3-hump character of the distribution.

This is all that I will remark in this lecture about the fundamental problems of overfitting and the curse of dimensionality, which taken together render the task of finding a good choice for the number *M* of mixture components a formidable challenge. We will simply assume that we already know an appropriate *M*.

We now approach the other question, namely, how can we find an estimate of the involved parameters of the *M* component Gaussians?

We will constrain ourselves to maximum-likelihood estimators and discuss two elementary techniques to find ML-estimates of *M* mixed *d*-dimensional Gaussians, gradient descent methods and the EM algorithm.

Gradient descent methods

A Gaussian mixture distribution is characterized by the means $\boldsymbol{\mu}_j$, the covariance matrices $\boldsymbol{\Sigma}_j$ and the mixture coefficients $P(j)$. We collectively refer to all of these parameters as $\theta = (\theta^1, ..., \theta^k)$. For didactic purposes we will restrict ourselves to a special case of the parameter estimation task, namely, we will assume that each $\boldsymbol{\Sigma}_j$ is equal to $\sigma_j^2 \mathbf{I}_d$, where $\mathbf{I}_d$ is the *d*-dimensional identity matrix[4]. That is, each of the participating Gaussians j is of radially symmetric shape, which is characterized by just one parameter σ_j^2. Using this simplification, the pdf of the *j*-th component Gaussian is given by

$$p(\mathbf{x} \mid j) = \frac{1}{(2\pi\sigma_j^2)^{d/2}} \exp\left(-\frac{\left\| \mathbf{x} - \boldsymbol{\mu}_j \right\|^2}{2\sigma_j^2} \right). \tag{3.26}$$

The model estimation task is specified as follows. We are given a sample $D = (\mathbf{x}^1, ..., \mathbf{x}^N)^T$ and want to find a maximum likelihood estimate θ^{ML} of θ:

[4] A complete treatment can be found in Duda, R. O., Hart, P. E., Stork, D. G., *Pattern Classification (second edition)*. John Wiley and Sons, 2001, Section 10.4. IRC: Q327 .D83 2001

$$\theta^{\mathrm{ML}} = \arg\max_{\theta} \mathcal{L}(\theta) = \arg\max_{\theta} p(D \mid \theta) = \arg\max_{\theta} \prod_{n=1}^{N} \sum_{j=1}^{M} p(\mathbf{x}^n \mid j) P(j)$$

(3.27)

$$= \arg\max_{\theta} \prod_{n=1}^{N} \sum_{j=1}^{M} \frac{1}{(2\pi\sigma_j^2)^{d/2}} \exp\left(-\frac{\left\| \mathbf{x}^n - \boldsymbol{\mu}_j \right\|^2}{2\sigma_j^2} \right) P(j)$$

Maximizing the *likelihood function* $\mathcal{L}(\theta)$ is equivalent to minimizing the negative log likelihood $-\log \mathcal{L}(\theta)$, which can be regarded as an *error function* $E(\theta) = -\log \mathcal{L}(\theta)$. Taking the negative logarithm of probabilities is a trick that is very common in computational statistics, because (i) it avoids numerical underflow problems (notice that the product of probabilities in (3.27) can become extremely small), and (ii) turns the product $\prod_{n=1}^{N}$ in (3.27) into a sum. So we will address the computational task to solve

$$\theta^{\mathrm{ML}} = \arg\min_{\theta} E(\theta)$$

(3.28)

$$= \arg\min_{\theta} -\sum_{n=1}^{N} \log \left(\sum_{j=1}^{M} \frac{1}{(2\pi\sigma_j^2)^{d/2}} \exp\left(-\frac{\left\| \mathbf{x}^n - \boldsymbol{\mu}_j \right\|^2}{2\sigma_j^2} \right) P(j) \right)$$

This is a non-trivial task, because the error function $E(\theta)$ is highly nonlinear and may have (indeed, will typically have) many local minima. You should think of $E(\theta)$ as a rugged "error landscape" over $\mathbb{R}^d$. We wish to find the global minimum of this landscape, or at least a local minimum that is not too different from the global one.

Finding the global minimum of an error landscape function is a central problem in optimization theory and machine learning. In this lecture, we will learn about a generic technique to search for a global optimum in a later section (on *simulated annealing*). Here we will only consider the simpler problem of finding a local minimum. One family of methods to achieve this objective is to perform a *gradient descent* on the error landscape, that is, start at some randomly chosen point θ_0 and then "slide downwards" along the error gradient $\nabla E = (\partial E / \partial \theta^1, \ldots, \partial E / \partial \theta^k)^{\mathsf{T}}$ until one reaches the bottom of the valley, that is, until $\nabla E = 0$. Figure 3.6 illustrates the principle of gradient descent towards a local minimum.

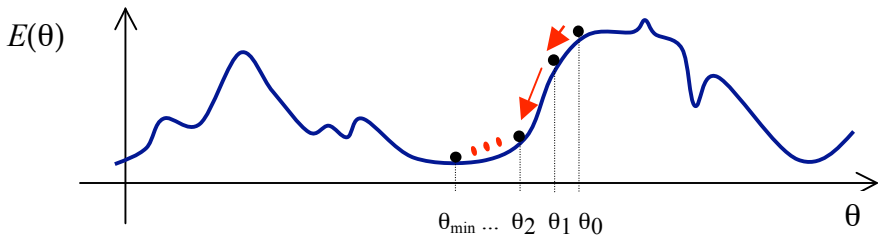


Figure 3.6 Principle of gradient descent for finding a local minimum on the error surface

There are many ways to detail out this basic idea of gradient descent into an iterative algorithm that approaches a local minimum value $\theta_{\min}$ asymptotically through a sequence θ_0, θ_1, θ_2, ... The simplest such algorithms are obtained by an update rule

(3.29) $$\theta_{n+1} = \theta_n - \eta \nabla E\big|_{\theta_n},$$

that is, at each iteration one takes a step along the negative gradient ("straightly downhill"), of a length that is weighted by some predetermined, fixed *learning rate* or *stepsize* η. Finding a good learning rate may be tricky; it shouldn't be too small (then convergence is slow) nor too large (then instability will result through "overshooting" the minimum). There are many refinements of this basic update rule, for instance by adapting the stepsize dynamically or by including curvature information ("second-order gradient methods"). You may learn about such methods in the Machine Learning lecture or by consulting textbooks[5], they are not in the focus of this lecture.

Gradient descent methods are particularly convenient when the gradient ∇E has a closed-form analytical representation that can be easily evaluated numerically. This is the case with mixtures of Gaussians. Differentiation of (3.28) w.r.t. to the means $\boldsymbol{\mu}_j$ gives

(3.30) $$\frac{\partial E}{\partial \boldsymbol{\mu}_j} = \sum_{n=1}^{N} P(j \mid \mathbf{x}^n) \frac{\boldsymbol{\mu}_j - \mathbf{x}^n}{\sigma_j^2},$$

where

(3.31) $$P(j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid j) P(j)}{p(\mathbf{x})}$$

is the probability that a sample point $\mathbf{x}$ was generated by the j-th Gaussian component and $p(\mathbf{x})$ is given by (3.24). For the standard deviation σ_j we obtain

(3.32) $$\frac{\partial E}{\partial \sigma_j} = \sum_{n=1}^{N} P(j \mid \mathbf{x}^n) \left(\frac{d}{\sigma_j} - \frac{\left\| \boldsymbol{\mu}_j - \mathbf{x}^n \right\|^2}{\sigma_j^3} \right).$$

The minimization of the mixing parameters $P(j)$ must carried out observing the constraints (3.25). This could be effected via Lagrange multipliers, but a simpler way is to adopt a frequently used trick, namely, represent $P(j)$ through auxiliary variables γ_j in the *softmax* function:

(3.33) $$P(j) = \frac{\exp(\gamma_j)}{\sum_{k=1,\ldots,M} \exp(\gamma_k)},$$

which monotonically transforms any set of M real numbers into M probabilities that sum to 1 and lie in (0,1). We can now perform an unconstrained minimization of E w.r.t. the γ_j. Observing

(3.34) $$\frac{\partial P(k)}{\partial \gamma_j} = \delta_{jk} P(j) - P(j) P(k),$$

[5] For instance, chapter 7 in Bishop (1999).

which follows from (3.33), and using the chain rule, we find

$$\frac{\partial E}{\partial \gamma_j} = \sum_{k=1}^{M} \frac{\partial E}{\partial P(k)} \frac{\partial P(k)}{\partial \gamma_j} = -\sum_{n=1}^{N} \left(P(j \mid \mathbf{x}^n) - P(j) \right). \tag{3.35}$$

Equipped with all the derivatives (3.30), (3.32) and (3.35), we now can implement an error minimizing algorithm for the basic gradient descent method (3.29) or one of the more refined versions thereof. Notice again that such methods only find the closest-by local minimum where you arrive by going downhill from the starting point θ_0. If you want to chase the global minimum, some (expensive) search methods must be employed (we will meet one of them in this lecture).

The Expectation-Minimization (EM) algorithm

Sliding down the gradient is a natural, but by no means the only method to find a (local) minimum in the error landscape. Another technique is the EM algorithm, which roots in a quite different intuition. The EM algorithm is widely used in computational statistics when it comes to estimate model parameters θ from *incomplete* or *partially observable* data. What "incomplete" means depends on the specific case – for instance, it might mean missing data points in time series measurements, or measurements of physical systems where the available sensors give only a partial account of the system state, of which some variables are *hidden* from observation. In our case it means that we do not know from which of the M Gaussian components a particular sample point $\mathbf{x}^i$ was generated. The "hidden" random variable whose values we ignore is in our case an M-class indicator variable $Y_{\text{component}}$ with the distribution $(P(1), \ldots, P(M))$. Figure 3.7 demonstrates the difference between complete and incomplete data in our situation of mixed Gaussians.

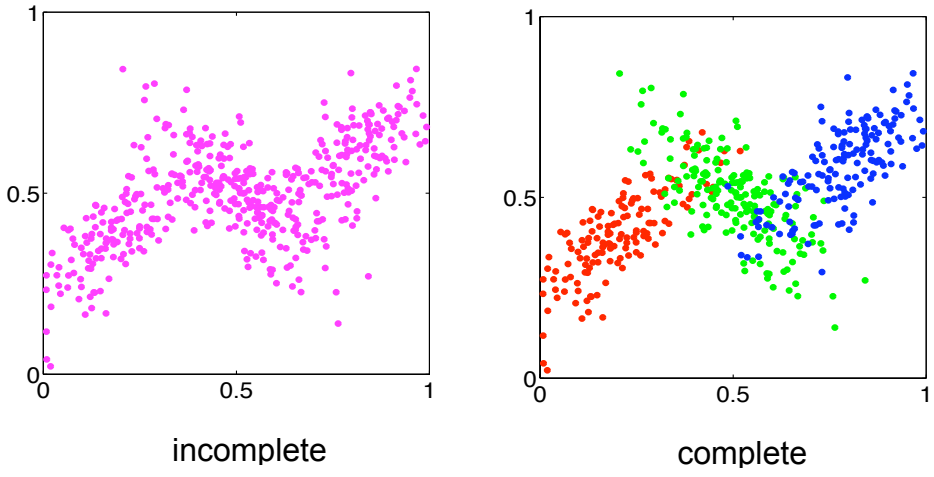


Figure 3.7 A sample from a mixture of three 2-dimensional Gaussians. Left: incomplete data don't include the information which point was "emitted" from which Gaussian. Right: complete data would contain this information. (Figure copied from a presentation of Christopher M. Bishop, http://research.microsoft.com/~cmbishop/downloads/Bishop-ECCV-04-tutorial-B.ppt)

We will not embark on the general theory of EM algorithms in this lecture (that is done in my Machine Learning lectures) but instead only demonstrate its workings, which are intuitive enough, for mixtures of Gaussians. First, notice that at a minimum of the error landscape, the derivatives (3.30), (3.32) and (3.35) become zero. That is, maximum-likelihood solutions for our parameters must satisfy

$$\boldsymbol{\mu}_j^{\mathrm{ML}} = \frac{\sum_{n=1}^{N} P(j \mid \mathbf{x}^n)\mathbf{x}^n}{\sum_{n=1}^{N} P(j \mid \mathbf{x}^n)}, \tag{3.36}$$

$$\sigma_{\mathrm{j}}^{\mathrm{ML}} = \frac{1}{d}\frac{\sum_{n=1}^{N} P(j \mid \mathbf{x}^n)\left\|\boldsymbol{\mu}_j^{\mathrm{ML}} - \mathbf{x}^n\right\|^2}{\sum_{n=1}^{N} P(j \mid \mathbf{x}^n)}, \text{ and} \tag{3.37}$$

$$P^{\mathrm{ML}}(j) = \frac{1}{N}\sum_{n=1}^{N} P(j \mid \mathbf{x}^n), \tag{3.38}$$

which are derived from putting (3.30), (3.32) and (3.35) to zero. Notice that the terms $P(j \mid \mathbf{x}^n) = p(\mathbf{x}^n \mid j)P(j)/p(\mathbf{x}^n)$, which appear abundantly on the rhs of these equations, depend on the distribution $P(j)$ of the "hidden" variable $Y_{\text{component}}$.

The idea of the EM algorithm is to solve (3.36), (3.37) and (3.38) with an iterated "ping-pong" procedure, where estimates of the hidden variable distribution lead to updates of estimates of the observable variables, and vice versa, as follows.

1) Start with an initial guess $\theta_0 = (\boldsymbol{\mu}_1^0,...,\boldsymbol{\mu}_M^0,\sigma_1^0,...,\sigma_M^0,P^0(1),...,P^0(M))$ of parameters.
2) Assume that after the k-th iteration you have a parameter estimate $\theta_k = (\boldsymbol{\mu}_1^k,...,\boldsymbol{\mu}_M^k,\sigma_1^k,...,\sigma_M^k,P^k(1),...,P^k(M))$.
   a) Use (3.31) to derive estimates $P^k(j \mid \mathbf{x}^n)$ for the terms $P(j \mid \mathbf{x}^n)$ in the rhs's of (3.36), (3.37) and (3.38). This is, use the current assumed parameters θ_k and the data to obtain the *expected* probability values $P(j \mid \mathbf{x}^n)$ of the hidden class indicator variable $Y_{\text{component}}$ at the sample points $\mathbf{x}^n$.
   b) Insert these $P(j \mid \mathbf{x}^n)$ into the equations (3.36), (3.37) and (3.38), along with replacing the $\boldsymbol{\mu}_j^{\mathrm{ML}}$ in those equations by the $\boldsymbol{\mu}_j^k$. Compute the left sides of the equations to obtain a new estimate of the model parameters $(\boldsymbol{\mu}_1^{k+1},...,\boldsymbol{\mu}_M^{k+1},\sigma_1^{k+1},...,\sigma_M^{k+1},P^{k+1}(1),...,P^{k+1}(M))$. These new parameters *maximize* the likelihood of the data, given the currently assumed sample values $P(j \mid \mathbf{x}^n)$ of the distribution of the hidden variable.
3) Iterate until some stopping criterium is reached (for instance, a predefined number of iterations is done, or the changes between iterations fall below some threshold).

The step 2a is called the *E*-step (from *E*xpectation), and step 2b the *M*-step (from *M*aximization). The theory of EM guarantees that the likelihoods of parameters θ_k grows monotonically with k. Notice that the EM algorithm aims at a maximum-likelihood solution, just as the gradient descent method.

Here is a little EM case study (Figures copied from a presentation of Christopher M. Bishop, http://research.microsoft.com/~cmbishop/downloads/Bishop-ECCV-04-tutorial-B.ppt.) The sample data come from observations of the Old Faithful geysir in the Yellowstone National Park (Figure 3.8).

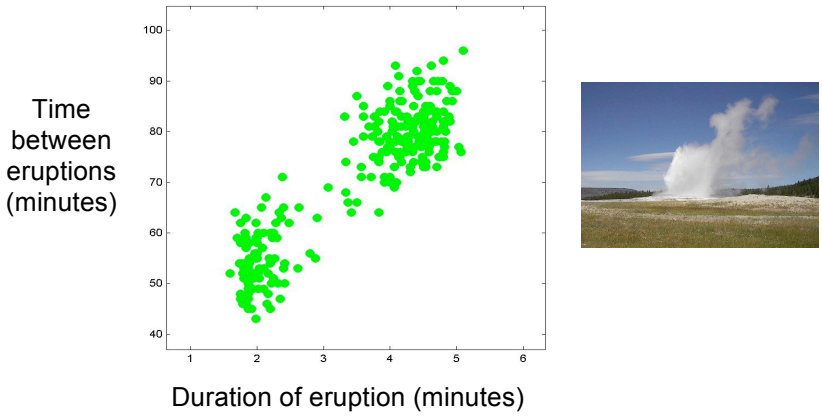


Figure 3.8 A two-dimensional dataset.

Figure 3.9 shows 20 EM iterations, with the first panel showing the initial guess. Color codes the estimated probabilities $P(j \mid \mathbf{x}^n)$.

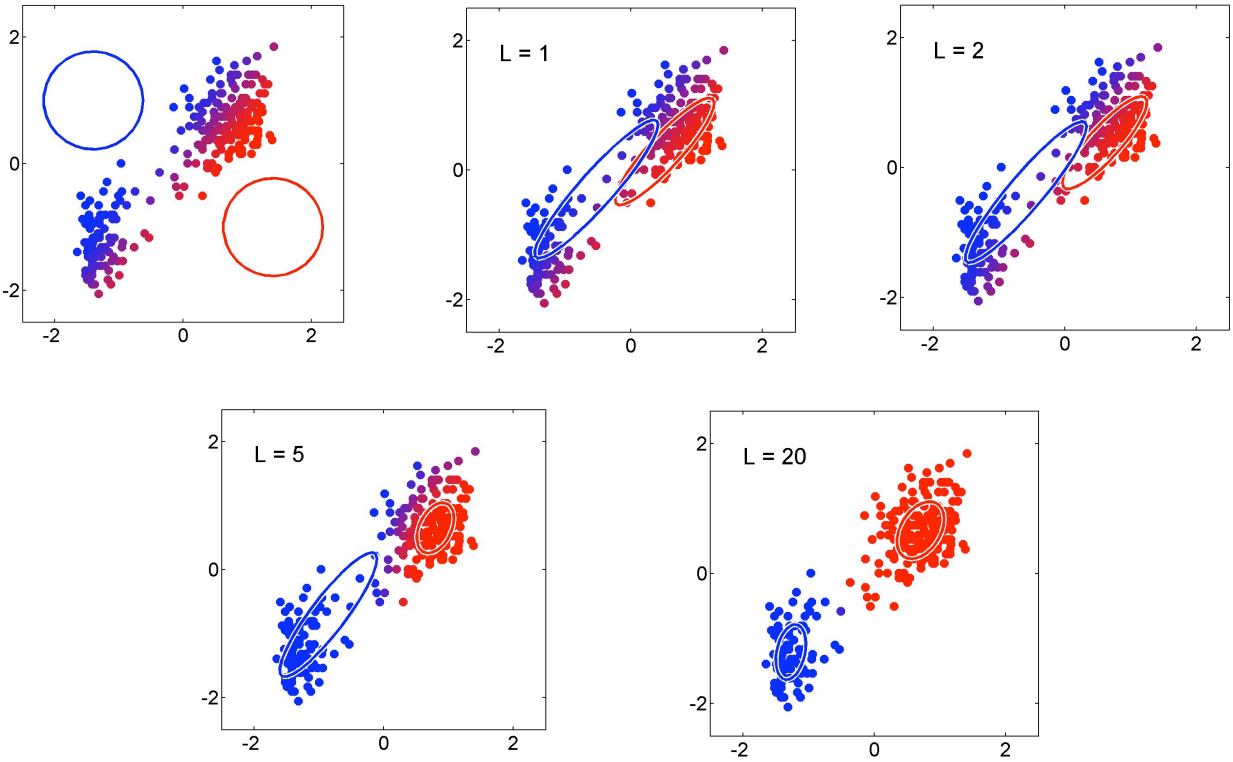


Figure 3.9 The EM algorithm at work on the Old Faithful dataset.

There is no general rule when the EM algorithm should be preferred over gradient descent methods in parametric model estimation tasks. Sometimes (although not in our example) no analytical form of the gradient is available or too expensive to calculate. Then EM is the only

way to go (we will see that this is the case for Bayesian networks, see later in this lecture). The EM algorithm is inherently stable (cannot enter oscillations or worse), which gradient descent methods are not. EM may converge slower than (advanced versions of) gradient descent methods. Finally, there exist computationally cheap *incremental* variants of gradient descent methods (for instance, *stochastic gradient descent* – visit the Machine Learning lecture...) which work well in online situations when data points $\mathbf{x}^n$ arrive in a stream. By contrast, EM is not immediately suited for incremental processing[6].

A general remark: both gradient descent and EM algorithms work generally very well on low-dimensional data sets, but may encounter severe problems (bad local minima, instability [in the case of gradient descent], long time to convergence) when applied to high-dimensional tasks.

Another general remark: the EM algorithm is actually not an algorithm, but a design scheme to construct ML-estimation algorithms. How the *E*- and *M*-step are to be calculated depends on the task at hand, and finding exact or approximate solutions to the expectation or maximization subtasks can be challenging. In spite of these difficulties, the EM "algorithm" is one of the main workhorses in computational statistics, and its introduction in a super-highly cited paper[7] by Dempster and Laird in 1977 marks a turning point in computational statistics.

3.3 Parzen windows

Parzen windows, a simple representative of the larger class of *kernel-based* representations, provide an alternative to mixture models for representing probability distributions. These representations are *non-parametric* because no prior assumptions concerning the shape of the distribution are needed. To introduce Parzen windows, consider the sample of 5 real-valued points shown in Figure 3.10. Centered at each sample point we place a unit square area on the *x*-axis. Weighing them each by 1/5 and summing them gives an intuitively plausible representation of a pdf.

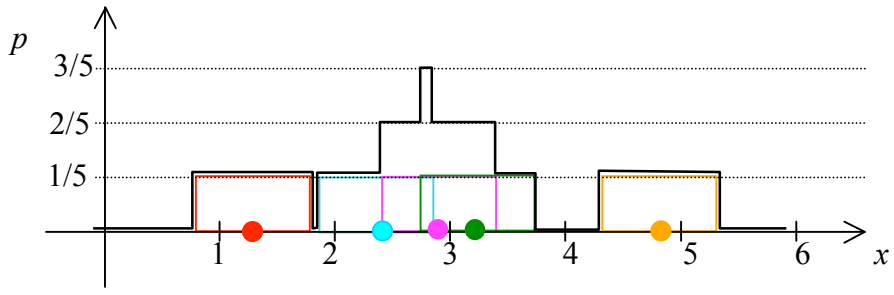


Figure 3.10 Rectangular Parzen window representation of a distribution given by a sample of 5 real numbers. The sample points are marked by colored circles. Each data point carries a square "Parzen window", that is, a rectangular pdf centered on the point. Weighted by 1/5 (colored rectangles) and summed (solid staircase line) they give a pdf.

---

[6] For a nontrivial extension of EM to incremental model update see: Neal, R. M. & Hinton, G. E. (1998), *A view of the EM algorithm that justifies incremental, sparse, and other variants*, in M. I. Jordan, ed., Learning in Graphical Models, Kluwer Academic Publishers.

[7] Dempster, A.P. and Laird, N.M. and Rubin, D.B. (1977), Maximum likelihood from incomplete data via the EM-algorithm. Journal of the Royal Statistical Society 39, 1-38

To make this example more formal and general, consider *d*-dimensional data points $\mathbf{x}^n$. Instead of a unit-length square, we wish to make these points the centers of *d*-dimensional hypercubes of side length *h*. We need a function *H* that indicates which points around $\mathbf{x}^n$ fall into the hypercube centered at $\mathbf{x}^n$. To this end we introduce a *kernel function*, also known as *Parzen window*,

$$H : \mathbb{R}^d \to \mathbb{R}$$

$$\mathbf{u} \mapsto \begin{cases} 1 & \text{if } |u_i| < 1/2 \quad \text{for } i = 1,...,d \\ 0 & \text{else} \end{cases} \tag{3.39}$$

so that *H* is the indicator function of a unit hypercube centered at the origin. Using *H*, we get the *d*-dimensional analog (for a sample of size *N* and hypercubes with sidelength *h*) of the "staircase" pdf in Figure 3.10 by

$$p^X(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^d}H\left(\frac{\mathbf{x}-\mathbf{x}^n}{h}\right), \tag{3.40}$$

observing that the volume of such a cube is h^d. The superscript $\boldsymbol{X}$ in $p^X(\mathbf{x})$ indicates that the pdf depends on the sample $\boldsymbol{X}$. Such a representation is somewhat reminiscent of a histogram, except that we combine the pdf from as many "bars" (hypercubes) as we have data points.

Clearly, given some sample $\{\mathbf{x}^n\}$, we do not really believe that such a rugged staircase reflects the true probability distribution the sample was drawn from. We would rather prefer a smoother version. This can be easily accomodated if we use other, smoother kernel functions. A standard choice is to use multivariate Gaussians with diagonal covariance matrix and uniform standard deviations $\sigma = h$ for *H*. This would turn (3.40) into

$$p^X(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{d/2}}\exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}^n\right\|^2}{2h^2}\right). \tag{3.41}$$

It is clear that any nonnegative kernel function *H* which integrates to unity can be used in equation (3.40) such that the resulting $p^X(\mathbf{x})$ will be a pdf.

The scaling factor *h* determines the width of the Parzen window and thereby the amount of smoothing. Figure 3.11 illustrates the effect of varying *h*.

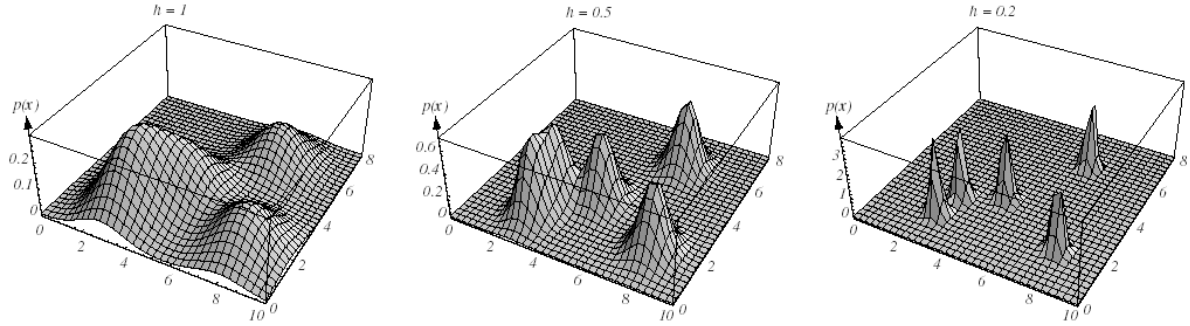


Figure 3.11: The effect of choosing different widths h in representing a 5-point, 2-dimensional sample by Gaussian windows. (Taken from the online set of figures of the book by Duda, Hart & Stork, ftp://ftp.wiley.com/public/sci_tech_med/pattern/)

A valuable insight into the smoothing nature of kernel functions can be obtained by computing the expectation of the estimated density $p^X(\mathbf{x})$. Using (3.40) we get

$$E[p^X(\mathbf{x})] = \frac{1}{N}\sum_{n=1}^{N} E_{\mathbf{x}^n}\left[\frac{1}{h^d}H\left(\frac{\mathbf{x}-\mathbf{x}^n}{h}\right)\right] = E_{\mathbf{x}'}\left[\frac{1}{h^d}H\left(\frac{\mathbf{x}-\mathbf{x}'}{h}\right)\right] \tag{3.42}$$
$$= \int_{\Re^d} \frac{1}{h^d}H\left(\frac{\mathbf{x}-\mathbf{x}'}{h}\right)p(\mathbf{x}')d\mathbf{x}'$$

where $E_{\mathbf{x}}$ denotes the expectation taken over varying $\mathbf{x}$ and $p(\mathbf{x}')$ is the true underlying pdf. We see from this equation that the expected pdf of a Parzen window model is the convolution of the kernel function with the true pdf.

Comments

1) Parzen window representations of pdfs are "non-parametric" in the sense that the shape of such a pdf is determined by a sample (plus, of course, by the shape of the kernel function, which however mainly serves a smoothing purpose). This fact also can render Parzen window representations computationally expensive, because if the sample size is large, a large number of data points have to be stored (and accessed if the pdf is going to be used).
2) The basic Parzen windowing scheme, as introduced here, can be refined in many ways. A natural way to improve on it is to use different widths h for different sample points $\mathbf{x}^n$. One then makes h narrow in regions of the sample set which are densely populated by sample points, and wide in regions that are only thinly covered by sample points. One way of doing that (which I invented while I was writing this – there are many ways to go) would be to (i) choose a reasonably small integer K; (ii) for each sample point $\mathbf{x}^n$ determine its K nearest neighbors $\mathbf{x}_1$, ..., $\mathbf{x}_K$; (iii) compute the mean squared distance d of $\mathbf{x}^n$ from these neighbors, (iv) set h proportional to this d for this sample point $\mathbf{x}^n$
3) As the Figure 3.11 demonstrates, the width h has a strong effect on the Parzen pdf. If h is too large, the smoothing becomes too strong, and information contained in the sample is smoothed away; in contrast, when h is too small, all that we see in the resulting Parzen pdf is the individual data points – the pdf then models not a distribution, but just the sample. Finding a good intermediate value for h in this tradeoff game is not trivial. In fact, this is another instance of the bias-variance dilemma that we already met when we discussed the problem of choosing an appropriate number M of Gaussians in mixture models (cf. Figure

3.4). In the Machine Learning lecture next year, you can learn about general strategies for coping with this problem. Here I mention one very popular (simple but potentially expensive) approach, *cross validation*. It goes like follows:

A simple cross validation scheme for finding a good value of *h*

a) Split the sample $\boldsymbol{X}$ into two parts $\boldsymbol{Y}$ and $\boldsymbol{Z}$.
b) Choose some h and compute $p^{Y}(\mathbf{x} \mid h)$, that is, the Parzen density w.r.t. width h that you get from using only the subsample $\boldsymbol{Y}$.
c) Compute the product density of the subsample $\boldsymbol{Z}$ under this model (or better, the log probability to avoid underflow problems)

$$\log p(\mathbf{Z} \mid h) = \sum_{\mathbf{z} \in \mathbf{Z}} \log p^{\mathbf{Y}}(\mathbf{z} \mid h). \tag{3.43}$$

d) Do this systematically for different h, and pick the h that maximizes (3.43).
e) Once you have a good h, use this for a Parzen model obtained from the complete data set $\boldsymbol{X}$.

The idea behind such cross-validation techniques (of which there are more refined versions) is to pick h such that the obtained model *generalizes* well to "new" data (here, the "new" data $\boldsymbol{Z}$ are artifically obtained by separating them from the original sample $\boldsymbol{X}$).

4) The Parzen-window based distribution (3.41) will easily lead to numerical underflow problems, especially if h is set to small values. A partial solution is to use log probabilities instead, i.e. use

$$\log(p^{X}(\mathbf{x})) = \log\left(\frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{d/2}}\exp\left(-\frac{\left\|\mathbf{x}-\mathbf{x}^{n}\right\|^2}{2h^2}\right)\right) \tag{3.44}$$

instead. Still this will not usually solve your underflow problems, because the exp-sum-terms within the log still underflow. Here is a trick to circumvent this problem, known as the "log-sum-exp" trick. Exploit the following:

$$\begin{aligned}\log\big(\exp(-A)+\exp(-B)\big) &= \log\big(\exp(-A+C)\exp(-C)+\exp(-B+C)\exp(-C)\big)\\ &= -C+\log\big(\exp(-A+C)+\exp(-B+C)\big)\end{aligned} \tag{3.45}$$

where you use $C = \max(A, B)$.

Parzen windowing, unlike mixtures of Gaussians, does not require that some parameters are estimated – the "modelling" or "learning" task is trivial. In my personal view, Parzen window estimates of densities – as compared to the more sophisticated mixture models – are "quick and dirty" methods that one might wish to use when the demands on statistical refinement are low and sample sizes are not too large. However, if one encounters very "wild" distributions where there are no reasons to assume that some simple mixture model comes close to reality, Parzen windowing can be well justified on the grounds that this technique introduces almost no bias into the model.

4.1 Sampling: why it is so important

"Sampling from a pdf" will be the topic for the next lectures. That is, given a pdf for a distribution of some random variable X, artificially "draw" a sample $(x_1, ..., x_N)$ from that distribution. Why should we care?

The universal, and at the same time elementary, type of model of a complex piece of reality is the joint distribution of all the observables that we employ. This joint distribution (and all the conditional distributions that arise from it) will often be an immensely high-dimensional monster. Often we will possess, as the result of a modelling work, a representation of the pdf of this distribution. The pdf may be available to us, in the one extreme, as an analytic function that is characterized by a few parameters – this will be the exception rather than the rule. In the other extreme, the pdf will be represented in a non-parametric way by some generic blackbox representation that is capable to characterize complicated, high-dimensional functions. In any case, *having* the model is rarely enough, we want to be *doing* something with it – prediction, classification, etc.

It turns out that a pdf is something that is not very well suited for *doing* something with. Imagine you have a pdf that describes a 1000-dimensional, numerical observation space (not uncommon at all). How would you do something as simple as just computing the expectation of this distribution? Do you think Matlab or Mathematica could do that? That is, how could a 1000-dimensional function be numerically integrated? I don't think there is a way... Whether the pdf is an analytical formula or a blackbox model with 1,000,000 parameters – if it is sufficiently complex – that is, if it is really interesting – then, in a sense the *only* thing you can "do" with it is to *evaluate* it at some point; after all a pdf is a function.

This should leave you with a taste of dissatisfaction. I have argued the pdf is an excellent kind of model of some piece of reality – but this piece of reality is reflected in the *entire* pdf, however, the *entire* pdf is a mathematical object or data structure that one cannot readily manipulate – all that one can typically do is evaluate it pointwise. This is why *sampling from a pdf* is such an important and recurring task in modelling: Given a pdf, artificially create a sample from it, that is, a set of artificially re-created "observations", which is a concrete instantiation of the model which one then may easily manipulate / analyze / measure / transform – in short, *do* many things with.

There is a second, equally important reason why sampling is so utterly useful. Namely, often one will be in a situation where one has modeled a stochastic system by a function π (analytical or by some blackbox modeling magic) which is nonnegative and has a bounded integral – that is, a function that looks like a pdf *up to a normalization factor*. To turn this function π into a valid pdf, one would have to divide it by its integral value $\int \pi$. However, evaluating an integral over a sufficiently high-dimensional function is typically infeasible, so the route to a valid pdf is barred. The good news is that many sampling methods do not require π to be normalized. Thus, one can operate with π as if it were a proper pdf in all kinds of ways and for all kinds of purposes which only require sampling. It turns out that this is enough for most application types, e.g. simulation, prediction, classification.

An intrinsic drawback of sampling-based usages of pdfs (or proto-pdfs π) is the computational load required by generating (large numbers of) sample points. With the advent of modern

computers however this is becomes an increasingly less painful issue. Many research directions in modern physics have only become possible by the option to simulate complex systems by sampling; it is fair to say that "sampling + computers" has become a key to large branches of physics research. But also in practical engineering and signal processing, sampling has become indispensable. My favourite example is robot navigation. It is today *the* standard approach in autonomous robot navigation to let a robot estimate its position by sampling from a pdf (rather, from a proto-pdf π) which models the distribution of the current position given previous sensor input and robot motions. Try Google or Youtube for *robot particle filter*!

Overview. We will introduce in this section some of the most widely used sampling algorithms – besides the elementary transformations from a uniform distribution, these are *rejection sampling*, the *Gibbs sampler*, and the *Metropolis algorithm*. The latter two are examples of a greater family of sampling strategies collectively known as *Markov Chain Monte Carlo* (MCMC) methods. In addition, I will also present a technique that is closely related to sampling, although its purpose is optimization: *simulated annealing*. We will illustrate the usefulness of sampling techniques by a modelling study taken from the literature where sampling is pivotal: the determination of the evolutionary tree of descendence from DNA data of living species.

My presentation of the material owes much (almost all) to two sources. For the elementary sampling techniques, I heavily lean on the Durbin et al book, *Biological Sequence Analysis*. For the MCMC material I also consulted the extensive survey of these techniques in

- Neal, Radford M. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, Dpt. of CS, Univ. Toronto, 1993. www.cs.toronto.edu/~radford/ftp/review.pdf

4.2 Sampling: definition

Intuitively, "sampling" means to artificially simulate the process of randomly taking measurements from a distribution given by a pdf. A "sampler" is an algorithm for doing this. Of samplers there are many and I have not found a universally agreed definition (http://mathworld.wolfram.com/search/ does not even have an entry). Here is my own definition:

Definition 4.1. Let P_X be a distribution on a measure space $(E, \mathcal{B})$. A sequence $X_1, X_2, \ldots$ of random variables is a *sampler for* P_X, if for all $A \in \mathcal{B}$ it holds that

$$P_X(A) = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} 1_A \circ X_i \qquad P\text{-almost surely}^8, \tag{4.1}$$

where $1_A(x) = \begin{cases} 1, \text{ if } x \in A \\ 0, \text{ else} \end{cases}$ is the *indicator function* for A.

[8] P-almost surely (syn.: "with probability 1", "almost surely", "w.p.1.") means that the equality holds for all $\omega \in \Omega$ except possibly on a null set.

A sample (as introduced in Section 2) is a much more restrictive notion than a sampler. Specifically, a sample results from independent random variables X_1, ..., X_n that all have the same distribution as X, the random variable whose distribution P_X the sample is drawn from. In contrast, the random variables X_1, X_2, ... of a sampler for P_X need not individually have the same distribution as X, and they need not be independent. For instance, let P_X be the uniform distribution on the binary event space $E = \{0,1\}$, that is, $P(X = 1) = P(X = 0) = 1/2$. A sample would be the values $X_1(\omega)$, ..., $X_n(\omega)$ of some random variables X_1, ..., X_n, each of which would have to satisfy $P(X_i = 1) = P(X_i = 0) = 1/2$. In contrast, a sampler could be, for instance, a sequence X_1, X_2, ... such that $P(X_{2i} = 1) = 1$ and $P(X_{2i+1} = 1) = 0$, that is, the sampler would deterministically create an alternating sequence of 0's and 1's – which in the long run would approximate the target distribution of 0's and 1's arbitrarily well. The individual sample point generators X_n of MCMC samplers are, as we will see, generally not distributed identically to the source distribution from which they sample.

4.3 Sampling by transformation from the uniform distribution

There is one sampler that you know very well and have often used: the *random* function that comes by different names in different programming languages (and is internally implemented in many different ways). In Matlab it is called `rand`, which draws a random double-precision number from the continuous, uniform distribution over the interval [0,1]; in C it's also called `rand` but here generates a random integer from the discrete uniform distribution between 0 and some maximal value. At any rate, a pseudo-random number generator of almost uniformly distributed numbers over some interval is offered by every programming language that I know, including Microsoft Word... And that's about it; the only sampler you can directly call in most programming environments is just that, a uniform sampler. (By the way, it is by no means easy to program a good pseudo-random number generator – in fact, designing such generators is an active field of research. If you are interested – the practical guide to using pseudorandom number generators[9] by David Jones is fun to read and very illuminating – thank you Corneliu for pointing this out!)

Assume you have a sampler Z_i for the uniform distribution on measure range [0,1], but you want to sample from another distribution P_X on a measure space $E = \mathbb{R}$, which has a pdf $g(x)$. Then you can use the uniform sampler indirectly to sample from P_X by a coordinate transformation, as follows.

First, compute the cumulative density function[10] $\phi: \mathbb{R} \to [0,1]$, which is defined by

$$\phi(y) = \int_{-\infty}^{y} g(u)\, du\,, \tag{4.2}$$

[9] David Jones (2010), Good Practice in (Pseudo) Random Number Generation for Bioinformatics Applications. A practical guide document, online at http://www.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf , local copy at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/GoodPracticeRNG.pdf

[10] Also known as "distribution function" or "probability distribution function".

and its inverse $\phi^{-1}(x)$ [this may be tricky or impossible to do analytically – then, numerical approximations must be called]. Then obtain a sampler X_i from the sampler Z_i by

$$X_i = \phi^{-1}(Z_i). \tag{4.3}$$

This is a sampler for P_X, which can be seen as follows**.** Let $A = [a,b]$ be an interval on the real line. We have to show that $P_X(A) = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} 1_A \circ X_i = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} 1_A \circ \phi^{-1}(Z_i)$ for any such A. Since the Z_i are i.i.d., $Z_i = Z$, it is clearly sufficient to show that $P_X(A) = E[1_A \circ \phi^{-1}(Z)]$. This can be deduced from $E[1_A \circ \phi^{-1}(Z)] = P(\phi^{-1}(Z) \in A) = P(Z \in \phi(a)) = P(Z \in [\phi(a), \phi(b)]) = \phi(b) - \phi(a) = \int_a^b g(u)\,du = P_X(A)$.

Graphically, the situation looks like this:

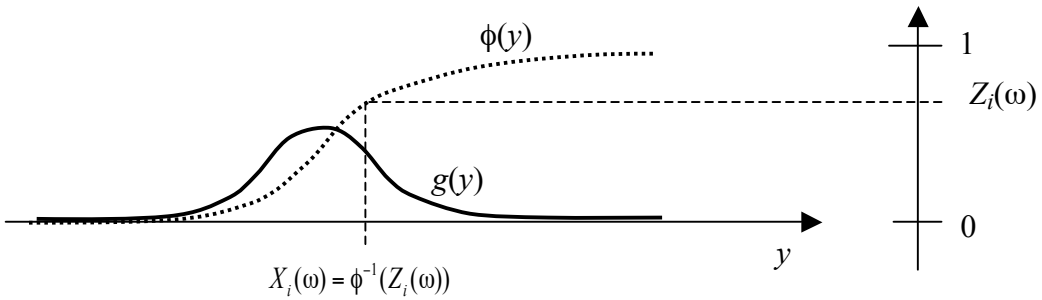


Figure 4.1. Sampling by transformation from a uniform sampler

Out of mathematical curiosity, I explored a little bit how one can sample from the normal distribution $\mathcal{N}(0,1)$. Its inverse cumulative density function is given by

$$\phi^{-1}(Z) = \sqrt{2}\,\mathrm{erf}^{-1}(2Z-1), \tag{4.4}$$

where erf^{-1} is a special function (like exp or sin) which is studied by mathematicians in its own right, called the *inverse error function*. It has the power series

$$\mathrm{erf}^{-1}(Z) = \sum_{k=1}^{\infty} \frac{c_k}{2k+1}\left(\frac{\sqrt{\pi}}{2}Z\right)^{2k+1}, \tag{4.5}$$

where the coefficients can be computed by the recursion $c_0 = 1$ and

$$c_k = \sum_{m=0}^{k-1} \frac{c_m c_{k-1-m}}{(m+1)(2m+1)}. \tag{4.6}$$

This series converges slowly (my source, I must confess, is Wikipedia). It seems that practical implementations for sampling from the normal distribution use *special sampling algorithms*

that exploit the mathematical structure of $\mathcal{N}(0,1)$. One algorithm (see http://en.wikipedia.org/wiki/Normal_distribution#Generating_values_for_normal_random_variables) which I found very elegant is the *Box-Muller algorithm*, which produces a $\mathcal{N}(0,1)$-distributed RV C from *two* independent uniform-[0,1]-distributed variables A and B:

$$C = \sqrt{-2\ln A}\cdot\cos(2\pi B). \tag{4.7}$$

The computational cost is to compute a logarithm and a cosine. An even (much) faster, but more involved (and very tricky) method is the Ziggurat algorithm (check Wikipedia for Ziggurat_algorithm for an article written with tender care). All in all, there exist wonderful things under the sun.

Sampling by coordinate transformation can be generalized to higher-dimensional distributions. Here is the case of a 2-dimensional pdf $g(y_1, y_2)$, from which the general pattern should become clear:

First, define the cumulative density function in the first dimension as the cumulative density function of the marginal distribution $\int_{-\infty}^{\infty} g(y_1,v)dv$ of the first coordinate y_1:

$$\phi_1(y_1) = \int_{-\infty}^{y_1}\left(\int_{-\infty}^{\infty} g(u,v)dv\right)du \tag{4.8}$$

and the conditional cumulative density function of y_2 given y_1:

$$\phi_2(y_2 \mid y_1) = \frac{\int_{-\infty}^{y_2} g(y_1,v)dv}{\int_{-\infty}^{\infty} g(y_1,v)dv} \tag{4.9}$$

Then, for sampling one value (y_1, y_2), sample two values x_1 and x_2 from the uniform sampler Z_i, and transform

$$y_1 = \phi_1^{-1}(x_1), \quad y_2 = \phi_2^{-1}(x_2 \mid y_1). \tag{4.10}$$

A widely used method for drawing a random vector $\mathbf{x}$ from the N-dimensional multivariate normal distribution with mean vector μ and covariance matrix Σ (required to be symmetric and positive-definite) works as follows:

1. Compute the Cholesky decomposition (matrix square root) of Σ, that is, find the unique lower triangular matrix $\mathbf{A}$ such that $\mathbf{A}\,\mathbf{A}^{\mathsf{T}} = \Sigma$.
2. Let $\mathbf{z} = (z_1, ..., z_N)^{\mathsf{T}}$ be a vector whose components are N independent standard normal variates (which can be generated, for example, by using the Box-Muller transform).
3. Let $\mathbf{x}$ be $\mu + \mathbf{A}\,\mathbf{z}$.

4.4 Rejection sampling

Sampling by transformation from the uniform distribution can be difficult or impossible if the pdf g one wishes to sample from has no simple-to-compute cumulative density function, or this has no simple-to-compute inverse. If this happens, it is sometimes possible to sample from a simpler distribution f that is related to the target pdf g and for which sampling by transformation works, and by a simple trick end up with a sampler for the pdf of interest, g. To understand this *rejection sampling* (also known as *importance sampling*) we first need to generalize a little bit the notion of a pdf:

Definition 4.2. A *proto-pdf* on a numerical measure space $(\mathbb{R}^n, \mathcal{B})$ is any nonnegative function $f_0: \mathbb{R}^n \to \mathbb{R}$ with a finite integral $\int_{\mathbb{R}^n} f(x)dx$.

If one divides a proto-pdf f_0 by its integral, one obtains a pdf f. Now assume that $g(y)$ is a pdf over $E = \mathbb{R}$ from which you want to sample, but you can't construct the requisite ϕ^{-1} that you would need for sampling by transformation. However, you find a proto-pdf $f_0 \geq g$ for whose associated pdf f you succeed in finding a cumulative density function and its inverse. Thus, you know how to sample from f, that is, you have a sampler Y_j for f. With that, you can construct a sampler X_i for g as follows. In order to assign a value to X_i,

1. Obtain $\tilde{y}_i = Y_j(\omega)$.
2. Accept $\tilde{y}_i$ as the value of $Z_i(\omega)$ with a probability of $g(\tilde{y}_i) / f_0(\tilde{y}_i)$, and with the complementary probability $1 - g(\tilde{y}_i) / f_0(\tilde{y}_i)$, reject and return to 1 with $j = j + 1$.

Rejection sampling becomes immediately clear if you imagine that "sampling from a pdf" as "piling up the pdf by many grains of sand that you let fall at the various positions y of E with a frequency proportional to $g(y)$". Then, looking at the following Figure where f_0 and g are plotted. Think of the f_0 curve as a sand dune that you get when sampling for f in this sand metaphor. Now, when you drop a grain of sand for modelling the f_0 "sand curve", imagine you paint it red with a probability of $g(\tilde{y}_i) / f_0(\tilde{y}_i)$ before you let it fall down. I think this saves you a mathematical proof...

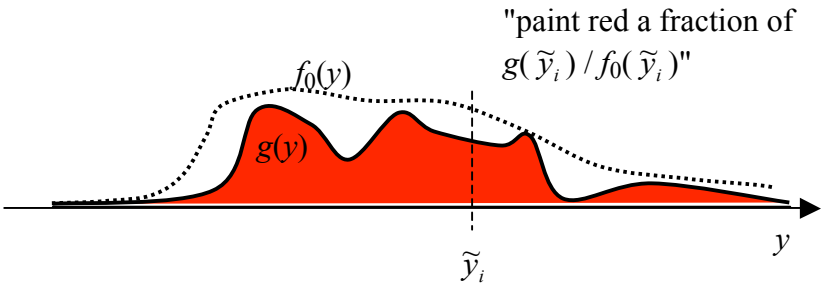


Figure 4.2 The principle of rejection sampling

The computational efficiency of rejection sampling clearly depends on how close f_0 is to g. If the ratio g / f_0 is on average small, there will be many rejections which slow down the algorithm.

4.5 MCMC sampling techniques: general principles

The basic idea of Markov chain Monte Carlo sampling is to construct a random walk $x_1, x_2, ...$ through E which in the long run visits the locations of E with a frequency that is proportional to g. Here are three examples of how such a random walk could be established that visits all places of the unit interval [0,1] according to the uniform distribution:

1. Define X_i = `Rand`. Then all the X_i are i.i.d. random variables, each of which is distributed uniformly over [0,1].
2. Define $X_0 = 0$, X_{i+1} = mod (X_i + sqrt(1/2), 1). The resulting sequence $x_1, x_2,...$ is deterministic!
3. Define $X_0 = 0$, X_{i+1} = mod (X_i + 1/1000 * (`Rand` – 0.5), 1). The resulting random walk roughly looks like a Brownian motion that will slowly and erratically "creep" alongside the unit interval.

These are all MCMC samplers, but they clearly are of different quality – especially the last one is quite awkward because it would need many, many 1000's of iterations until the unit interval would be halfway smoothly covered.

We need some basics of Markov chain theory.

Definition 4.3. A *Markov chain* is a sequence of random variables $X_1, X_2, ...$ (all taking values in a measure space $(E, \mathcal{B})$), such that X_{n+1} depends only on X_n and is conditionally independent on all earlier values:

$$P(X_{n+1} \mid X_1,...,X_n) = P(X_{n+1} \mid X_n). \tag{4.11}$$

Intuitively, a Markov chain is a sequence of measurements/observations, where the next observation only depends on the current one; earlier observations are "forgotten". Markov chains are examples of *stochastic processes*; they are the stochastic processes without memory.

A Markov chain on $(E, \mathcal{B})$ is fully characterized by the *initial distribution* P_{X_1} and the conditional *transition distributions*

$$P^n(X_{n+1} \mid X_n = y) \quad \text{for all } y \in E, \tag{4.12}$$

for which we also write

$$T_n(x \mid y). \tag{4.13}$$

A more common name for $T_n(x \mid y)$ is a *transition kernel.* If $T_n(x \mid y) = T_{n'}(x \mid y)$ for all *n*, *n*', the Markov chain's transition law does not itself depend on time; we then call the Markov chain *homogeneous.* With homogeneous Markov chains, the index *n* can then be dropped from $T_n(x \mid y)$.

A special case that you might already be familiar with is obtained when *E* is finite, say, $E = \{s_1, ..., s_k\}$. Then the transition probabilities $T_{ij} = P(X_{n+1} = s_j \mid X_n = s_i)$ can be collected in a $k \times k$ *Markov transition matrix M* where $M(i,j) = T_{ij} = P(X_{n+1} = s_j \mid X_n = s_i)$, which in its *i*-th row carries the probability vector of the probabilities by which the process will transit from state s_i to the states indexing the columns.

If a Markov chain with finite state set $E = \{s_1, ..., s_k\}$ and Markov transition matrix *M* is executed *m* times, the transition probabilities to transit from state s_i to state s_j can be found in the *m*-step transition matrix M^m:

(4.14) $$P(X_{n+m} = s_j \mid X_n = s_i) = M^m(i, j).$$

We now consider the measure space $E = \mathbb{R}^n$, and assume that all distributions of interest are specified by pdf's. We consider a homogeneous Markov chain. Its transition kernel $T(\mathbf{x} \mid \mathbf{y})$ is represented by the pdf of $P(X_{n+1} \mid X_n = \mathbf{y}) = P_{X_{n+1}|X_n=\mathbf{y}}$. The distributions of X_1, X_2 etc., have the following pdfs:

(4.15) $$g^1: \quad \text{given (the initial distribution)}$$
$$g^{n+1}(\mathbf{x}) = \int_{\Re^n} g^n(\mathbf{y}) T(\mathbf{x} \mid \mathbf{y}) d\mathbf{y}$$

Here is the mathematical core concept for MCMC:

Definition 4.4. Let *g* be the pdf of any distribution *P* on $E = \mathbb{R}^n$, and let $T(\mathbf{x} \mid \mathbf{y})$ be the transition kernel of a homogeneous Markov chain with values in *E*. Then *P* is an *invariant distribution of* $T(\mathbf{x} \mid \mathbf{y})$, if

(4.16) $$g(\mathbf{x}) = \int_{\mathbb{R}^n} T(\mathbf{x} \mid \mathbf{y}) g(\mathbf{y}) d\mathbf{y}.$$

Except for "pathological" cases, a transition kernel has generically at least one invariant distribution.

Furthermore, it is often the case that there exists exactly one invariant distribution *g* of $T(\mathbf{x} \mid \mathbf{y})$, and the sequence of distributions g^i converges to *g* from any initial distribution. We will call the transition kernel $T(\mathbf{x} \mid \mathbf{y})$ *ergodic*[11] if it has this property. The invariant distribution *g* of an ergodic Markov chain is also called its *asymptotic* distribution or its *stationary* distribution or its *equilibrium* distribution.

[11] Warning: there exist several closely related but not fully equivalent definitions of ergodicity in the literature!

The general strategy of MCMC is to construct, for a given distribution with pdf g, some ergodic Markov chain $X_1, X_2, \ldots$ which has g as its invariant distribution. Ergodicity is important to ensure that asymptotically the distributions of X_n converge to the target distribution g regardless of the starting state. The three introductory examples suggest that there will be many possibilities to create such MCMC samplers, but they also warn us that some of them might be quite inefficient.

Quoting from the Neals survey, "The amount of computational effort required to produce a good Monte Carlo estimate using the states generated by a Markov chain will depend on three factors: first, the amount of computation required to simulate each transition; second, the time for the chain to converge to the equilibrium distribution, which gives the number of states that must be discarded from the beginning of the chain; third, the number of transitions needed to move from one state drawn from the equilibrium distribution to another state that is almost independent, which determines the number of states taken from the chain at equilibrium that are needed to produce an estimate of a given accuracy. The latter two factors are related..."

If we start from a pdf g for a distribution P and want to create an MCMC sampler $X_1, X_2, \ldots$ for it, we have to make sure that the Markov chain $X_1, X_2, \ldots$ has P as its invariant distribution. This can be achieved by ensuring that the Markov chain has the property of *detailed balance* w.r.t. P. Detailed balance connects $X_1, X_2, \ldots$ to g in a strong way. It says that if we pick some state $x \in E$ with the probability given by P and multiply it with the transition probability $T(\mathbf{y} \mid \mathbf{x})$ – that is, we consider the kind of "P-T-mixed" probability density of transiting from $\mathbf{x}$ to $\mathbf{y}$ – then this is the same as the reverse transiting probability density from $\mathbf{y}$ to $\mathbf{x}$:

Definition 4.5. Let P be a distribution on $\mathbb{R}^n$ with pdf g and $T(\mathbf{x} \mid \mathbf{y})$ the transition kernel for a homogeneous Markov chain on E. Then $T(\mathbf{x} \mid \mathbf{y})$ has the *detailed balance property* w.r.t. g if

$$\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n\colon\ T(\mathbf{y} \mid \mathbf{x})\ g(\mathbf{x}) = T(\mathbf{x} \mid \mathbf{y})\ g(\mathbf{y}). \tag{4.17}$$

If $T(\mathbf{x} \mid \mathbf{y})$ has the *detailed balance* property w.r.t. g, then g is an invariant distribution of $T(\mathbf{x} \mid \mathbf{y})$, because

$$\int_{\Re^n} g(\mathbf{y}) T(\mathbf{x} \mid \mathbf{y})\, d\mathbf{y} = \int_{\Re^n} g(\mathbf{x}) T(\mathbf{y} \mid \mathbf{x})\, d\mathbf{y} = g(\mathbf{x}) \int_{\Re^n} T(\mathbf{y} \mid \mathbf{x})\, d\mathbf{y} = g(\mathbf{x}). \tag{4.18}$$

Apart from this formal argument, there is also an intuitive argument why detailed balance implies invariance: if $T(\mathbf{x} \mid \mathbf{y})$ has the detailed balance property w.r.t. g, then "applying" T to g (in the sense of computing $\int_{\Re^n} g(\mathbf{y}) T(\mathbf{x} \mid \mathbf{y})\, d\mathbf{y}$) will not change g, because the "probability mass transport" from some point $\mathbf{x}$ to another point $\mathbf{y}$ will be equal to that in the reverse direction.

There arise some complications with this definition of detailed balance if P or T place nonzero probability mass on single elements $\mathbf{x} \in \mathbb{R}^n$ or single transitions $\mathbf{x} \to \mathbf{y}$, respectively. These complications can be overcome by using the following alternative definition of detailed balance:

Definition 4.6. (= alternative version of Def. 4.5) Let P be a distribution on $(E, \mathcal{B}) = (\mathbb{R}^n, \mathcal{B})$ with pdf g and $T(\mathbf{x} \mid \mathbf{y})$ the transition kernel for a Markov chain on E. Then $T(\mathbf{x} \mid \mathbf{y})$ has the detailed balance property w.r.t. g if

(4.19) $\forall\, A, B \in \mathcal{B}$: $\int_A \int_B g(\mathbf{x}) T(\mathbf{y} \mid \mathbf{x})\, d\mathbf{y}\, d\mathbf{x} = \int_B \int_A g(\mathbf{y}) T(\mathbf{x} \mid \mathbf{y})\, d\mathbf{x}\, d\mathbf{y}$,

that is, the probability of a transition from somewhere in A to somewhere in B is the same as that of the reverse transition.

There is no general recipe to construct efficient MCMC samplers, but there are some standard approaches that work well in a variety of conditions. We present two of these: the *Gibbs sampler* and the *Metropolis algorithm.* Both rest on constructions of Markov chains that exhibit detailed balance.

4.6 Gibbs sampling

Let g be a pdf on $\mathbb{R}^n$. Let

(4.20) $$g_i = g_i(x_i \mid x_1,...,x_{i-1},x_{i+1},...,x_n) = \frac{g(x_1,...,x_n)}{\int_{\Re} g(x_1,...,x_n)\, dx_i}$$

be the conditional density function of the coordinate x_i given the values on the other coordinates. Let g^1 be an initial distribution on $\mathbb{R}^n$, and let an initial value $\mathbf{x}_1 = (x_1^1,...,x_n^1)$ be chosen randomly according to g^1. We define a Markov chain $X_1, X_2, \ldots$, through transition kernels as follows. The idea is to cycle through the n coordinates and at some time $\nu n + i$ ($0 \le \nu$, $1 \le i \le n$) change the previous sample vector $\mathbf{x}_{\nu n + i - 1} = (x_1^{\nu n+i-1},...,x_n^{\nu n+i-1})$ only in the i-th coordinate, by sampling from g_i. That is, at time $\nu n + i$ we set

(4.21) $$\mathbf{x}_{\nu n + i} = (x_1^{\nu n+i},...,x_n^{\nu n+i}) = (x_1^{\nu n+i-1},...,x_{i-1}^{\nu n+i-1},x_i^{\nu n+i},x_{i+1}^{\nu n+i-1},....,x_n^{\nu n+i-1})$$

to the previous value in all coordinates except at dimension i, where we randomly sample $x_i^{\nu n+i}$ from the distribution $g_i(x_i \mid x_1^{\nu n+i-1},...,x_{i-1}^{\nu n+i-1},x_{i+1}^{\nu n+i-1},....,x_n^{\nu n+i-1})$.

This method is known as the *Gibbs sampler*. It uses n different transition kernels $T_1, ..., T_n$, where T_i is employed at times $\nu n + i$ and updates the i-th coordinate only according to

(4.22) $$\begin{aligned} &T_i\big((y_1,...,y_n) \mid (x_1,...,x_n)\big) = \\ &= g_i(y_i \mid y_1,...,y_{i-1},y_{i+1},...,y_n)\, \delta_i\big((y_1,...,y_n) - (x_1,..,x_n)\big) \end{aligned}$$

where δ_i is a version of the Dirac delta that we know from Eq. (2.34). More specifically, $\delta_i\big((y_1,...,y_n) - (x_1,..,x_n)\big)$ concentrates the probability mass on the set of points $(y_1,..,y_n)$ satisfying $(y_1,...,y_{i-1},y_{i+1},...,y_n) = (x_1,..,x_{i-1},x_{i+1},..,x_n)$. More formally, δ_i is a measure (not a probability measure!) on $\mathbb{R}^n$ with the defining property that for all hypercubes $A \subset \mathbb{R}^n$, it holds that

(4.23)

$$\delta_i(A) = \int_A 1\, d\delta_i = \int_{(A \cap \{(x_1,\ldots,x_n) \in \Re^n \mid x_j = 0 \text{ for } i \neq j\})_{[i]}} 1\, d\mu_1 = \mu_1((A \cap \{(x_1,\ldots,x_n) \in \Re^n \mid x_j = 0 \text{ for } i \neq j\})_{[i]}),$$

where μ_1 is the Lebesgue measure on $\mathbb{R}$ and subscript $[i]$ denotes the projection of a higher-dimensional vector on the i-the component.

The Markov chain $X_1, X_2, \ldots$ is not homogeneous because we cycle through different transition kernels. However, we can condense a sequence of n successive updates into a single update that affects all coordinates by putting

(4.24) $$T = T_n \circ \cdots \circ T_1,$$

which yields a homogeneous Markov chain $Y_1, Y_2, \ldots$ with transition kernel T whose observations are

(4.25)
$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x}_1 = (x_1^1,\ldots,x_n^1), \\ \mathbf{y}_2 &= \mathbf{x}_{n+1} = (x_1^{n+1},\ldots,x_n^{n+1}), \\ \mathbf{y}_3 &= \mathbf{x}_{2n+1} = (x_1^{2n+1},\ldots,x_n^{2n+1}), \ldots \end{aligned}$$

To understand this abstractly, note that transition kernels may be concatenated through

(4.26) $$(T_2 \circ T_1)(\mathbf{z} \mid \mathbf{x}) = \int_{\Re^n} T_2(\mathbf{z} \mid \mathbf{y})\, T_1(\mathbf{y} \mid \mathbf{x})\, d\mathbf{y}$$

-- this is actually nothing but the continuous version of the iterated transition matrix for finite state sets from Eq. (4.14).

It should be intuitively clear that g is an invariant distribution of this homogeneous Markov chain, because clearly each transition kernel T_i leaves g invariant: only the i-th component of the observation vector is affected at all, and it is updated exactly according to the conditional distribution g_i for this component. It even holds that T has detailed balance w.r.t. g (exercise!).

It remains to investigate whether the Markov chain with kernel T is ergodic. This is certainly the case when the carrier of g in $\mathbb{R}^n$ (that is, the subset of $\mathbb{R}^n$ where g takes positive values) is n-dimensionally connected (that is, for any points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $g(\mathbf{x}) > 0$, $g(\mathbf{y}) > 0$ there exists a sequence of n-dimensional balls B_i of positive radius, the first containing $\mathbf{x}$ and the last containing $\mathbf{y}$, such that B_i intersects with B_{i+1} with positive volume). However, if the carrier of g is not n-dimensionally connected, then T may or may not be ergodic, and determining whether it is needs to be done on a case by case basis. For instance, if g is a distribution on $\mathbb{R}^2$ whose carrier lies exclusively in the first and third orthant, T would not be ergodic, because the Gibbs sampler, when started from a point in the third orthant, would be unable to jump into the first orthant. This situation is depicted in Figure 4.3.

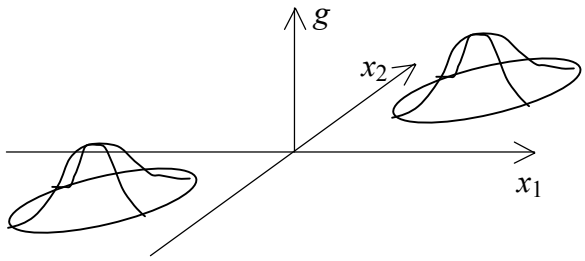


Figure 4.3 A "bipartite" pdf *g* where the Gibbs sampler would fail

The Gibbs sampler is obviously useful only if one can easily sample from the 1-dimensional conditional distributions g_i. Therefore, the Gibbs sampler is mostly employed in cases where these g_i are parametric, analytical distributions, or in cases where E is finite and the g_i thus become simple probability vectors. The Gibbs sampler is attractive for its simplicity. A number of extensions and refinements of the basic idea is presented in the Techreport by R. Neal. The Gibbs sampler is often used in connection with Bayesian networks, which we will treat later in this course.

4.7 The Metropolis algorithm

The Metropolis[12] algorithm is more sophisticated than the Gibbs sampler and works in a much larger number of cases. The drawback is that typically it is computationally more expensive. This algorithm explicitly constructs a Markov chain with detailed balance. Like the Gibbs algorithm, the Metropolis algorithm can be cyclically applied to the n dimensions of the observation vectors in turn ("local" Metropolis algorithm) or it can be applied to all dimensions simultaneously ("global" algorithm). We describe the local version.

Like the Gibbs algorithm, the local Metropolis algorithm updates the sample vector $X_{\nu n+i-1} = \mathbf{x}^{\nu n+i-1} = (x_1^{\nu n+i-1},...,x_n^{\nu n+i-1})$ in the i-th coordinate, yielding $X_{\nu n+i} = (x_1^{\nu n+i},...,x_n^{\nu n+i}) = (x_1^{\nu n+i-1},...,x_{i-1}^{\nu n+i-1},x_i^{\nu n+i},x_{i+1}^{\nu n+i-1},....,x_n^{\nu n+i-1})$. The update $x_i^{\nu n+i-1} \mapsto x_i^{\nu n+i}$ is done in two steps, which together ensure detailed balance w.r.t. the conditional distribution g_i:

Step 1: randomly choose a candidate value $x_i^{\nu n+i}*$ for $x_i^{\nu n+i}$. Usually a *proposal distribution* $S_i(x^* \mid \mathbf{x})$ is used (which may depend on $\mathbf{x}$) which is symmetric in the sense that $S_i(x \mid (x_1,...,x_{i-1},x',x_{i+1},...,x_n)) = S_i(x' \mid (x_1,...,x_{i-1},x,x_{i+1},...,x_n))$ for all $x_1,...,x_{i-1},x_{i+1},...,x_n$.

Step 2: Randomly *accept* or *reject* $x_i^{\nu n+i}*$ as the new value $x_i^{\nu n+i}$, in a fashion that ensures detailed balance. In case of acceptance, $x_i^{\nu n+i}$ is set to $x_i^{\nu n+i}*$; in case of rejection,

[12] Named after the first author of a classical paper: Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller E. (1953) *Equation of state calculations by fast computing machines*, J. of Chemical Physics 21, 1087-1092

$x_i^{\nu n+i}$ is set to $x_i^{\nu n+i-1}$, i.e., the observation vector is not changed at all. The probability for accepting $x_i^{\nu n+i}*$ is determined from an *acceptance distribution* $A_i(x^* \mid \mathbf{x})$.

Step 1 and 2 together yield a transition kernel

$$\begin{aligned} T_i(\mathbf{y} \mid \mathbf{x}) &= T_i((y_1,...,y_n) \mid (x_1,...,x_n)) = \\ &= S_i(y_i \mid \mathbf{x})\, A_i(y_i \mid \mathbf{x})\, \delta((y_1,...,y_{i-1},y_{i+1},...,y_n) - (x_1,..,x_{i-1},x_{i+1},.,x_n)) \\ &\quad + \delta((y_1,...,y_n) - (x_1,...,x_n))\left(1 - \int_{\Re} S_i(y \mid \mathbf{x})\, A_i(y \mid \mathbf{x})\, dy\right) \end{aligned} \tag{4.27}$$

The first term is the probability of proposing a change in component i from x_i to y_i and then accepting the proposed change. The second term integrates up the probability mass that was not used for the first term and concentrates it in $x_i = y_i$, reflecting the rejection cases where the state was not changed.

The proposal and acceptance distribution must be designed to warrant detailed balance, that is, to ensure

$$\forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^n\colon g(\mathbf{x})\; T_i(\mathbf{y} \mid \mathbf{x}) = g(\mathbf{y})\; T_i(\mathbf{x} \mid \mathbf{y})\,. \tag{4.28}$$

There are many ways to solve this problem. For instance, it is not difficult to see (exercise)!) that if $S_i(x^* \mid \mathbf{x})$ is symmetric, using $A_i(x^* \mid \mathbf{x}) = g(x_1,...,x_{i-1},x^*,x_{i+1},...,x_n)/\left(g(x_1,...,x_{i-1},x^*,x_{i+1},...,x_n) + g(x_1,...,x_n)\right)$ yields detailed balance. This is called the *Boltzmann acceptance function*. However, the Metropolis algorithm standardly uses another acceptance distribution, namely the *Metropolis acceptance distribution*

$$A_i(x^* \mid \mathbf{x}) = \min\left(1, g(x_1,...,x_{i-1},x^*,x_{i+1},...,x_n)/g(x_1,...,x_n)\right). \tag{4.29}$$

That is, whenever $g(x_1,...,x_{i-1},x^*,x_{i+1},...,x_n) \geq g(x_1,...,x_n)$, -- that is, the proposed observation has no lower probability than the current one --, accept with certainty; else accept with probability $g(x_1,...,x_{i-1},x^*,x_{i+1},...,x_n)/g(x_1,...,x_n)$.

For symmetric proposal distributions, this strategy implies detailed balance. It is clear that in the rejection case, where the current observation is not changed, that is, $\mathbf{x} = \mathbf{y}$, the detailed balance condition (4.28) trivially holds. In the acceptance case we verify (4.28) as follows:

$$
\begin{aligned}
& g(\mathbf{x})T_i(\mathbf{y}\,|\,\mathbf{x}) = \\
& \quad = g(x_1,...,x_n)\,T_i\big((y_1,...,y_n)\,|\,(x_1,...,x_n)\big) \\
& \quad = g(x_1,...,x_n)\,S_i(y_i\,|\,\mathbf{x})\,A_i(y_i\,|\,\mathbf{x})\,\delta\big((y_1,...,y_{i-1},y_{i+1},...,y_n)-(x_1,...,x_{i-1},x_{i+1},...x_n)\big) \\
& \quad = S_i(y_i\,|\,\mathbf{x})\,\min\big(g(x_1,...,x_n),g(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n)\big)\,\delta(...) \\
(4.30) \qquad & \quad = S_i(x_i\,|\,(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n))\,\min\big(g(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n),g(x_1,...,x_n)\big)\,\delta(...) \\
& \quad = g(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n)\,S_i(x_i\,|\,(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n))\cdot \\
& \qquad \cdot A_i(x_i\,|\,(x_1,...,x_{i-1},y_i,x_{i+1},...,x_n))\;\;\delta(...) \\
& \quad = g(y_1,...,y_n)\,S_i(x_i\,|\,(y_1,...,y_n))\,A_i(x_i\,|\,(y_1,...,y_n))\;\;\delta(...) \\
& \quad = g(\mathbf{y})T_i(\mathbf{x}\,|\,\mathbf{y})
\end{aligned}
$$

Notes:

1. Like the Gibbs sampler, this local Metropolis algorithm can be turned into a global one, then having a homogeneous Markov chain with transition kernel T, by condensing one n-cycle through the component updates into a single observation update, in analogy to (4.25).
2. Detailed balance guarantees that g is an invariant distribution of the homogeneous Markov chain with transition kernel T. Before a Metropolis sampler Y_1, Y_2, ... can be used for sampling, it must in addition be shown that this Markov chain is ergodic. The same caveats that we met with the Gibbs sampler apply.
3. The Metropolis algorithm is more widely applicable than the Gibbs sampler because it obviates the need to sample from a marginal distribution. The price one has to pay is a higher computational cost for the same level of sampling quality, because the rejection events lead to duplicate sample points which obviously leads to an undesirable "granularity" in the sample that has to be washed away by a larger sample size.
4. In statistical physics, the term "Monte Carlo Simulation" is almost synonomous with this Metropolis algorithm (claims Neal – I am no physicist and couldn't check).
5. The quality of Metropolis sampling depends very much on the used proposal distribution. Specifically, its variance should neither be too small (then exploration of new states is confined to a narrow neighborhood of the current state, implying that the Markov chain traverses the distribution very slowly) nor too large (then one will often be propelled far out in the regions of g where it almost vanishes, leading to numerous futile rejection events). A standard choice is a normal distribution centered on the current state (and for a discrete distribution with finite E, a uniform distribution on E).
6. The Metropolis algorithm (and the Gibbs sampler, too) works best if the n state components are statistically maximally independent. Then the state space exploration in each dimension is independent from the exploration in the other dimensions, whereas if the components are statistically coupled, a fast exploration in one dimension is hindered by the fact that moving about in this dimension necessitates a synchronized moving about in the other dimensions, thus larger changes in one dimension have to "wait" for the correlated changes in the other dimensions. Thus if possible one should coordinate-transform the distribution before sampling with the aim to decorrelate the dimensions (which in principle does not imply making them independent but in practice is the best one can do). Standard methods from linear systems theory can be employed for this purpose (introduced in the Machine Learning companion lecture offered in alternating years).
7. We presented the Metropolis algorithm in the special case of sampling from a distribution over (a subset of) $\mathbb{R}^n$. This made the "cylce through dimensions" scheme natural which we

also described for the Gibbs sampler. However, the Metropolis algorithm works also on observation spaces E which do not have the nice vector space structure of $\mathbb{R}^n$; proposal distributions then have to be designed as the individual case requires. The phylogenetic tree example from the next section 4.8 is an example.

4.8 Application example: determining evolutionary trees

This section is a condensed version of a paper by Mau, B., Newton, M.A., Larget, B. (1999), *Bayesian Phylogenetic Inference via Markov Chain Monte Carlo Methods*, Biometrics 55, 1-12, online (preprint ps version) at http://www.genome.wisc.edu/pub/reprints/biometrics.ps; I put a pdf version at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/1748_Mauetal99.pdf. It is a beautiful demonstration of how various modelling techniques are combined to characterize a highly complex pdf (namely, the conditional distribution of all possible evolutionary pasts, given DNA data from living species!), and how the Metropolis algorithm is used to sample from it. This paper was cited 148 times by other papers according to the Science Citation Index, as of October 2007, which is quite a lot. Chapter 8 of the Durbin et al. book gives a more extensive treatment of phylogenetic modelling and briefly discusses the Mau et al. paper.

Data. DNA strings from related l living species, each string of length N, have been aligned without gaps. In the reported paper, $l = 32$ African fish species (cichlids from central African lakes except one cichlid species from America that served as control) were represented by DNA sequences of length 1044. 567 of these 1044 sites were identical across all considered species and thus carried no information about phylogeny. The remaining $N = 477$ sites represented the data D that was entered into the analysis. Reminder: the DNA symbol alphabet is $\Sigma = \{A, C, G, T\}$.

Task. infer from data D the most likely phylogenetic tree, assuming that the considered living species have a common ancestor from which they all descended.

Modelling assumptions. Mutations act on all sites independently. Mutations occur randomly according to a "molecular clock", i.e. a probability distribution of the form

$$P(y \mid x, t, \beta) \tag{4.31}$$

specifing the probability that the symbol $y \in \{A, C, G, T\}$ occurs at a given site where t years earlier the symbol x occurred. β is a set of control parameters specifying further modelling assumptions about the clock mechanism. Mau et al. used a molecular clock model proposed in Hasegawa et al. (1985)[13] which uses two parameters $\beta = (\phi, \kappa)$, the first quantifying an overall rate of mutation, and the second a difference of rates between the more frequent mutations that leave the type of nucleic acid (purine or pyrmidine) unchanged ("transitions") vs. change it ("transversions"). All that we need to know here is that (4.31) can be efficiently computed. Note that β is not assumed to be known beforehand but has to be estimated/optimized in the modelling process, starting from the mere data D.

Representing phylogenetic trees. A phylogenetic tree is a binary tree Ψ. The nodes represent *taxa* (species); leaves are living taxa, internal nodes are extinct taxa, the root node is the

[13] Hasegawa, M., Kishino, H., Yano, T. (1985), *Dating the human-ape splitting by a molecular clock of mitochondrial DNA*. J. of Molecular Evolution 22, 160-174

assumed common ancestor. Mau et al plot their trees bottom-up, root node at the bottom. Vertical distance between nodes metrically represents the evolutionary timespans *t* between nodes. *Clades* are subsets of the leaves that are children of a shared internal node. Figure 3.4 shows a schematic phylogenetic tree and some clades.

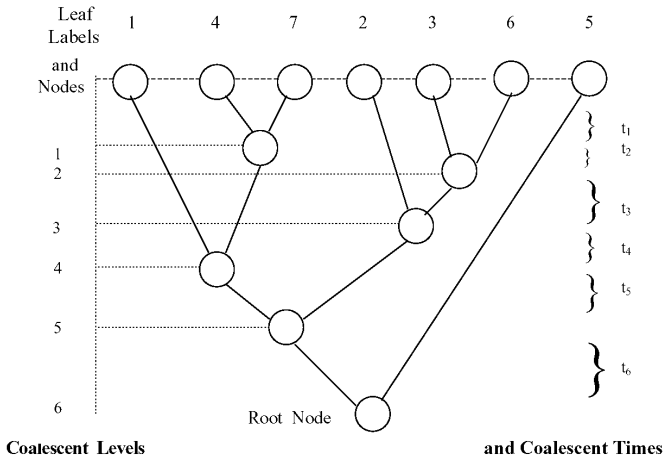


Figure 4.4 An examplary phylogenetic tree (from the Mau et al paper). {4,7}, {1 4 7}, {2, 3, 6} are examples of clades in this tree.

A given evolutionary history can be represented by trees in 2^{n-1} different but equivalent ways (where *n* is the number of living species), through permuting the two child branches of an internal node. For computational purposes a more convenient representation than a tree graph is

(2.1) a specification of a left-to-right order σ of leaves (in Fig. 3.4, σ = (1,4,7,2,3,6,5)), plus
(2.2) a specification of the graph distances *a* between to successive leaves.

The graph distance is the length of the connecting path between the two leaves. In the example tree from Fig. 3.4, the distances between leaves make the distance vector $a = 2(t_1+t_2+ t_3+ t_4, t_1, t_1+t_2+ t_3+ t_4+ t_5, t_1+t_2+ t_3, t_1+t_2, t_1+t_2+ t_3+ t_4+ t_5+ t_6)$. A pair (σ, a) is a compact representation for a phylogenetic tree.

Likelihood of a phylogenetic tree. One subtask that must be solved in order to find the most probable evolutionary tree is to devise a fast method for computing the likelihood of a particular tree Ψ and molecular clock parameters β, given the sequence data:

(4.32) $$L(\Psi,\beta) = P(D \mid \Psi,\beta).$$

Because the mutation processes at different sites are assumed to be independent, $P(D \mid \Psi,\beta)$ splits into a product of single-site probabilities $P(D_i \mid \Psi,\beta)$, where D_i are the symbols found in the sequences from *D* at site *i* – thus D_i is a vector of length *l.* Therefore, we only must find a way to compute

(4.33) $$L_i(\Psi,\beta) = P(D_i \mid \Psi,\beta).$$

We approach this task sideways, assuming first that we know the symbols y_ν at the *i*-th site of the internal nodes ν of Ψ. Let ρ be the root node and π_0 a reasonable assumed distribution of

symbols in ρ (e.g., the global distribution of all symbols in all sites of all sequences in D). Then we get the probability of these hypothetical data $D_i \cup (y_\nu)_{\nu \text{ is internal node}}$ by

$$(4.34) \qquad P(D_i \cup (y_\nu)_{\nu \text{ is internal node}} \mid \Psi,\beta) = \pi_0(y_\rho) \prod_{\substack{\nu \text{ is node of } \Psi \\ \nu \neq \rho}} P(y_\nu \mid y_{par(\nu)}, t_\nu, \beta),$$

where $par(\nu)$ is the parent node of ν and t_ν is the timespan between $par(\nu)$ and ν. From (4.34) we could obtain (4.33) by summing over all possible assignments of symbols to internal nodes, which is clearly infeasible. Fortunately there is a cheap recursive way to obtain (4.33), which works top-down from the leaves, inductively assigning conditional likelihoods $L_\nu(y) = P(D_i \upharpoonright \nu \mid \Psi,\beta, \text{node } \nu = y)$ to nodes ν, where $y \in \Sigma$ and $D_i \upharpoonright \nu$ is the subset of the D_i which are siblings of node ν, as follows:

case 1: ν is a leaf: $$L_\nu(y) = \begin{cases} 1, \textit{if } y = y_\nu \\ 0, \textit{else} \end{cases}$$

case 2: ν is an internal node: $$L_\nu(y) = \left(\sum_{z \in \Sigma} L_\lambda(z) P(z \mid y, t_\lambda, \beta) \right) \left(\sum_{z \in \Sigma} L_\mu(z) P(z \mid y, t_\mu, \beta) \right),$$

where λ, μ are the two children of node ν, t_λ is the timespan from ν to λ, and t_μ is the timespan from ν to μ. Then (4.33) is obtained via

$$(4.35) \qquad L_i(\Psi,\beta) = \sum_{z \in \Sigma} \pi_0(z) L_\rho(z),$$

from which (4.32) is obtained by

$$(4.36) \qquad L(\Psi,\beta) = \prod_{i \text{ is site in } D} L_i(\Psi,\beta).$$

$O(N\,|\Sigma|\,l)$ flops are needed to compute $L(\Psi,\beta)$ – in our example, $N = 477$, $|\Sigma| = 4$, $l = 32$.

The posteriori distribution of trees and mutation parameters. We are actually not interested in the likelihoods $L(\Psi,\beta)$ but rather in the distribution of Ψ,β (a Bayesian hyperdistribution!) given D. Bayes theorem informs us that this desired distribution is proportional to the likelihood times the prior (hyper-)distribution of Ψ,β:

$$(4.37) \qquad P(\Psi,\beta \mid D) \sim P(D \mid \Psi,\beta)\, P(\Psi,\beta) = L(\Psi,\beta)\, P(\Psi,\beta).$$

Lacking a profound theoretical insight, Mau et al. assume for $P(\Psi,\beta)$ a very simple, uniform-like distribution (such uninformedness is perfectly compatible with the Bayesian approach!). Specifically:

1. They bound the total hight of trees by some arbitrary maximum value, that is, all trees Ψ with a greater hight are assigned $P(\Psi,\beta) = 0$.
2. All trees of lesser hight are assigned the same probability. Note that this does *not* imply that all *topologies* are assigned equal prior probabilities. Figure 3.5 shows two topologies, where the one shown in **a.** will get a prior twice as large as the one shown in **b.** Reason: the two internal nodes of the first topology can be shifted up- and

downwards independently, whereas this is not the case in the tree **b.**, thus there are twice as many trees of the topology **a.** than of **b.** *Note that for evolutionary biologists it is the tree topology (which species derives from which species) rather than the tree metrics (how long took what) which is of prime interest*!

3. The mutation parameters β are assigned a prior distribution that is uniform on a range interval chosen generously large to make sure that all biologically halfway plausible possibilities are contained in it.

Figure 4.5. Two tree topologies that get different priors. Topologies are defined by the parenthesis patterns needed to describe a tree. For instance, the tree **a.** would be characterized by a pattern ((x x)(x x)) and the tree **b.** by ((x (x x)) x).

The stage is now set... After all these preparations, we possess a pdf g (up to an unknown scaling factor) of $P(\Psi,\beta \mid D)$, which can be evaluated with a small computational effort for a given Ψ,β. (*Of course*, it would rather be the log of g which would be actually computed, avoiding underflow problems).

What is the structure and dimensionality of the observation space E in which Ψ,β lie? Remember that a tree Ψ can be specified by (σ, a), where σ is a permutation vector of $(1,...,l)$ and a is a numerical vector of length $l - 1$. Noticing that there are $l!$ permutations, a pair (σ, a) reflects a point in a product space $\{1,...,l!\} \times \mathbb{R}^{l-1}$; together with the two real-valued parameters comprised in β brings us to a space $\{1,...,l!\} \times \mathbb{R}^{l+1}$. However, there may be some constraints within the parameters of a that reduce the effective degrees of freedom to a number smaller than $l - 1$. This is hard to analyze (Mau et al. don't do it), and if the DOFs are less than $l - 1$, we would not necessarily find a useful lower-dimensional representation. Be this as it may, all in all the carrier domain of our pdf for $P(\Psi,\beta \mid D)$ has a dimension in the order of l – in our example, 32 –, and a heterogeneous, complicated structure.

The target question that biologists want to get answered: what tree *topology* is the most probable, given DNA sequences D of living species?

The strategy to find out about the answer is brutally simple: sample a large number of trees Ψ_i from g, sort these sampled trees into sets defined by tree topology, then interpret the relative sizes of these sets as probability estimates.

The Metropolis-algorithm at work. Mau et al. use the Metropolis algorithm (in a global version) to sample trees from g. A crucial design task is to find a good proposal distribution $S((\Psi^*,\beta^*) \mid (\Psi,\beta))$. It should lead from any plausible (Ψ,β) [i.d., $g((\Psi,\beta))$ is not very small] to another plausible (Ψ*,β*), which should be however as distinct from (Ψ,β) as possible. The way how Mau et al. go about this task is one of the core contributions of their work.

The authors alternate between updating only β and only Ψ. Updating β to β* is done in a straightforward way: the new *-parameters are randomly drawn from a rectangular distribution centered on the current settings β.

The tricky part is to generate an as different as possible, yet "plausibility-preserving" new tree Ψ* from Ψ. Mau et al. transform Ψ = (σ, *a*) into Ψ* = (σ*, *a**) in two steps:

(2.2) The current tree Ψ is transformed into one of its 2^{n-1} equivalent topological versions by randomly reversing with 0.5 probability every of its internal branches, getting Ψ' = (σ', *a*').

(2.3) In Ψ' the evolutionary inter-species time spans *t* are randomly varied by changing the old values by a random increment drawn from the uniform distribution over [−δ, δ], where δ is a fixed bound (see Fig. 4.6). This gives Ψ* = (σ', *a**).

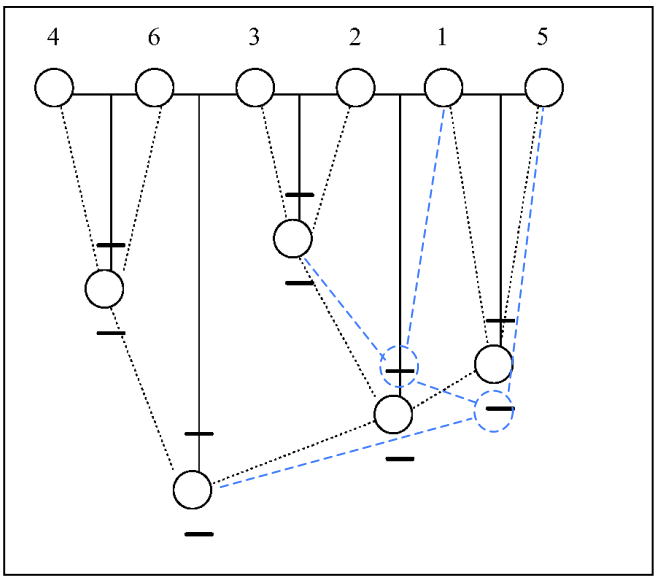


Figure 4.6. Candidate trees, attainable from the current tree, are characterized by intervals of size 2δ, centered at the current internal nodes, that constrain the repositioning of the internal nodes. (Taken from the Mau et al paper). Note that if the two rightmost internal nodes are shifted such that their relative heightes become reversed (dashed blue circles), the topology of the tree would change (dashed blue lines).

Mau et al show that this method yields a symmetric proposal distribution, and that every tree Ψ' can be reached from every other tree Ψ in a bounded number of such transformations – the Markov chain is thus ergodic.

Concretely, the Metropolis algorithm was run for 1,100,000 steps (20 hours CPU time on a 1998 Pentium 200 PC), the first 100,000 steps were discarded to wash out possible distortions resulting from the arbitrary starting tree, the remaining 1,000,000 trees were subsampled by 200 (reflecting the finding that after every 200 steps, trees were empirically independent [zero empirical cross-correlation at 200 step distance]), resulting in a final tree sample of size 5000.

Final findings. The 600 (!) most frequent topologies make up for 90% of the total probability mass. This high variability however is almost entirely due to minute variations within 6 clades (labelled A, ..., F) that are stable across different topologies. Figure 4.7. shows the two most

frequently found topologies resolved at the clade level, with a very strong posterior probability indication for the first of the two.

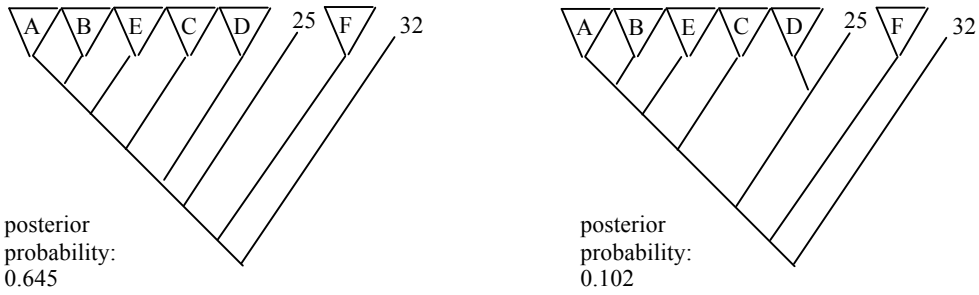


Figure 4.7. The two clade tree structures of highest posterior probability found by Mau et al.

For quality control, this was repeated 10 times – with no change of the final outcome, which makes the authors confident of their work's statistical reliability.

5 Simulated Annealing and Energy-Based Representations of Distributions

"Simulated annealing" (SA) is a general-purpose stochastic optimization method that shares much with the Metropolis algorithm, both historically and technically. Like this algorithm, it was first introduced within the conceptual framework of statistical physics, and again like the Metropolis algorithm, it shows a novel way of how the power of modern computational machinery can be used to answer questions concerning very complex systems – that is, systems with hundreds or even Millions of free parameters, where there is no hope whatsoever of an analytical approach. SA has been introduced in one of these *very* classical landmark papers in 1983[14]. *Very classical landmark paper* means: this paper has been cited more than 6,000 times.

Simulated annealing rests on an insight that every physical system, under certain idealized conditions, gives rise to a statistical distribution of its state energies; and vice versa, every statistical distribution over a measure space leads to a notion of energy defined for the elements of the measure space. This bi-directional translation from thermodynamics to statistics and back has been, and will continue to be, extremely fruitful for both parties, theoretical physics and algorithmical and statistical modelling. With this section I want to provide a first introduction of the basic concepts of this field, and present some important application examples.

5.1 Physical background

Generally, SA is used to search a large space of potential solutions s (called *microstates*) of some combinatorial problem, in order to find a solution that is close to the global minimum w.r.t. some real-valued target function (*objective function*) $E(s) \geq 0$. The two example applications treated in the original paper are representative for the flavour of simulated annealing problems:

1. The first problem concerned the optimization of computer hardware layout, specifically the distribution of functional units to microchips, and the spatial placement of units on a chip. Each possible design is a microstate. Several objective functions are considered in that article, e.g. number of pins of chips (to be minimized) or total wiring length (minimize it, too). Given that there are many thousands of functional units to place and wire, which are moreover heterogeneous, this is clearly a task without an analytical solution (and exhaustive search is clearly also precluded!).
2. The second problem is the classical traveling salesman problem, that is, the problem of finding the shortest itinerary through a map with N cities. This problem is known to be NP-complete (I assume this is known to you – if not, consult Wikipedia, or if you are ashamed of that, any textbook on computational complexity). Here the microstates are the possible traveling routes, and the objective function is the total route length.

To understand the method (and the maths) of SA, we must introduce some concepts from statistical thermodynamics. Consider a vessel containing a liquid, kept at a constant temperature T. Statistical thermodynamics is the branch of physics that explains *macroscopic* measurable quantities from the statistics of *microscopic* states s, also called microstates. For our liquid-in-a-vessel system, macroscopic quantities would be for instance volume,

[14] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi (1983) *Optimization by Simulated Annealing.* Science, Vol. 220 Nr 4598, 671-680. A copy is available from the course homepage.

temperature, pressure – things you can measure with everyday devices. The microstates of our system would be exact specifications of the current positions and velocities of all of the individual liquid molecules in the vessel. It is clear that there are very many (try to imagine how many...) possible microstates! Every microstate has an assigned *energy* $E(s)$. How this energy is defined depends on the physics of the particular model system – for a liquid in a vessel, the energy of a microstate would depend, among other things, on the potential energy of each of the molecules (the higher in the vessel, the greater its potential energy), on their kinetic energies (the faster the molecule, the greater its kinetic energy), but also on inter-molecular forces (the closer together a pair of mutually repellant molecules, the greater the energy contribution from this pairwise force), etc.

Due to molecular motion and maybe quantum-physical effects, the liquid in the vessel is incessantly and randomly changing its microstate. Assuming that the vessel is allowed to exchange heat – that is, energy – with its environment (called a *heat bath*), different microstates may have different energy (this would be prohibited in a perfectly isolated system).

Now one may ask the question, *what is the probability distribution of the microstates*? (This is a crucial question for thermodynamics, because macroscopic states are derived from *averaging* over microstates. For instance, temperature is related to the average velocity of molecules).

A fundamental fact of thermodynamics is the insight that the probability distribution of microstates is fully determined by the energy $E(s)$ of microstates. Specifically, the pdf over microstates is given by the *Boltzmann distribution*

$$(5.1) \qquad p(s) = \frac{1}{Z}\exp(-E(s)/T)$$

where Z is the normalizing factor that ensures that p integrates to unity:

$$(5.2) \qquad Z = \int_S \exp(-E(s)/T)\, ds$$

Note that both p and Z depend on the temperature T, so we sometimes write $p(s,T)$ and $Z(T)$. Z is called the *partition function*.

In order to understand SA, we should get a good intuitive grasp on the Boltzmann distribution, especially how it changes with T. Figure 5.1 shows plots of $p(s)$ for four different temperatures $T = 1, 0.1, 0.01, 0.001$ (red lines). I constructed the energy function $E(s)$ arbitrarily. There are 100 different microstates (points on the x-axis); the energy $E(s)$ is plotted into all panels (dotted blue line). The plots clearly reveal the fundamental two properties of the Boltzmann distribution:

1. As T grows, the distribution becomes more and more uniform. In the limit of $T \to \infty$, $p(s)$ develops into the uniform distribution. Intuitively speaking[15], at high temperatures the overall

[15] The correct intuitions are not really easy to come by in statistical thermodynamics. The treatment given here is extremely simplified, to the extend where it is misleading, so don't take it as an introduction to statistical thermodynamics. For one, the considerations are true only in the limit of an infinitely large heat bath – which implies that the subsystem must have microstates of arbitrarily large energy (so Figure 4.1 is misleading). A

energy of the combined system "vessel plus heat bath" is so abundant that the vessel subsystem gets "kicked around" through its states regardless of their (relatively minor) energy differences.

2. As T decreases toward zero, the Boltzmann distribution concentrates more and more on the low-energy microstates. In the limit of $T \to 0$, the distribution develops into the Dirac δ-peak at the global minimum of E (if there is a single global minimum). The state(s) where E is globally minimal is (are) called the *ground state(s)*.

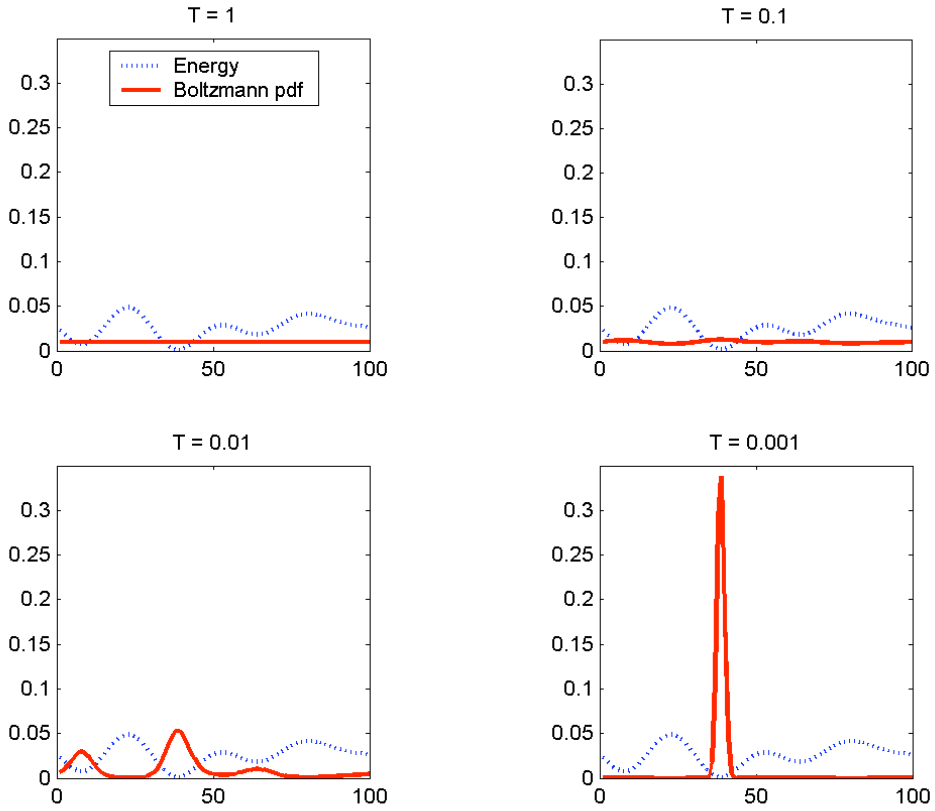


Figure 5.1. The Boltzmann distribution for four different temperatures. For explanation see text.

5.2 From physics to a search algorithm

The fact that when T is lowered, the Boltzmann distribution homes in on the global minimum of the energy function, is the key to SA for finding (almost) globally optimal solutions in combinatorial search tasks. Here are the core ideas:

1. Identify the points of the combinatorial search space with microstates s of an artificial "thermodynamic" system.

---

more correct interpretation of the convergence of the Boltzmann distribution to the uniform distribution with increasing temperature would go like this: "As temperature increases – which means that the total energy of the combined vessel-heatbath-system grows –, the number of microstates of the combined vessel-heat bath system increases, and the ratio of numbers N and N' of microstates where the vessel subsystem has relatively small energies E or E' tends more and more to 1". Second, microstates should be defined in terms of quantum mechanical states.

2. Identify the objective function, of which an (almost) global minimum has to be found, with the energy function $E(s)$.
3. Start with some medium or high temperature T_0 and consider the Boltzmann distribution $p(s, T_0)$, which will be close to uniform (as in the first panel of Fig. 5.1). Start sampling from this distribution (using the Metropolis algorithm would be the classical choice). The sequence of created samples will cover the search space almost uniformly.
4. Now lower the temperature gradually, thereby obtaining a sequence of Boltzmann distributions $p(s, T_0), p(s, T_1), p(s, T_2)$, ... that more and more concentrates on the points in search space (microstates) that have low values of the objective functions. Continue sampling all the time. The sequence of samples should thus concentrate more and more on microstates with low values of the objective function.
5. Continue until $T_n \approx 0$. The hope is that then the cooling process has guided you toward the global minimum and that the samples that you now get are closely scattered around the optimum.

A natural question at this point is, why not start immediately at low temperatures (e.g. in a situation like that shown in the 4th panel of Fig. 5.1), wouldn't that just save the time of "hovering aimlessly" about the search space at high temperatures, and instead directly lead you to the desired minimum, which is well pronounced at low temperatures? The answer is, if one starts at low temperatures, -- or, for that matter, if one cools to rapidly --, one is likely to get "frozen" in a very suboptimal local minimum from which one cannot escape. This becomes intuitively clearer if we re-interpret the Metropolis sampling of $p(s, T_i)$ in physical terms of "jumping around" in the energy landscape $E(s)$ directly.

Recall from (4.29) that the Metropolis acceptance function certainly accepts a jump to a new, proposed sample s^* from a current sample s if $g(s^*) > g(s)$; if $g(s^*) \le g(s)$, the new s^* is accepted with a probability of $P_{\text{accept}} = g(s^*) / g(s)$ [where g was the pdf of the distribution from which one samples]. In the context of SA, $g(s) = p(s, T) = \frac{1}{Z}\exp(-E(s)/T)$. Taking the log of p, one obtains

$$\log(p(s, T)) = \log(\frac{1}{Z}) - E(s)/T \tag{5.3}$$

or equivalently

$$E(s) = -T\log(p(s,T)) + T\log(\frac{1}{Z}), \tag{5.4}$$

which expresses the energy of a microstate s in terms of the temperature and the Boltzmann distribution value at s. We now consider the two cases when (a) the Metropolis algorithm certainly accepts, and (b) when it accepts with probability $P_{\text{accept}} = g(s^*) / g(s)$, and see how we can translate this into energies.

Case (a): if $g(s^*) > g(s)$, that is, $p(s^*, T) > p(s, T)$, then from (5.4) we see that this is equivalent to $E(s^*) < E(s)$. In other words, when the energy of the newly proposed sample / microstate is lower than the current one's, it is accepted with certainty.

Case (b): if $g(s^*) \le g(s)$, that is, the acceptance probability is $P_{\text{accept}} = g(s^*) / g(s) = p(s^*, T) / p(s, T)$, then

$$
\begin{aligned}
\log(P_{\text{accept}}) &= \log(\frac{p(s^*,T)}{p(s,T)}) = \log(p(s^*,T) - \log(p(s,T)) \\
&= \log(\frac{1}{Z}) - E(s^*)/T - \log(\frac{1}{Z}) + E(s)/T \\
&= \frac{1}{T}(E(s) - E(s^*))
\end{aligned}
\tag{5.5}
$$

from which by exponentiation we get

$$
P_{\text{accept}} = \exp((E(s) - E(s^*))/T) = \exp(\Delta E/T)\,, \tag{5.6}
$$

where ΔE is the (negative in sign!) energy "uphill" jump from state s to state s^*.

Summarizing we see that in terms of energy, a new proposed microstate is accepted with certainty if its corresponding energy jump goes "downhill", and if it goes uphill, it is accepted with probability $P_{\text{accept}} = \exp(\Delta E/T)$. That is, the greater the energy increase, the (exponentially) more unlikely is such a step taken; however, this may be compensated by a proportional increase in temperature. In other words, on the average, at higher temperatures we may take higher jumps uphill.

Equipped with this re-interpretation of the Metropolis algorithm in terms of an energy-based acceptance function (incidentally, this is the perspective taken in the original paper from Kirkpatrick et al), we can better understand why slow cooling is important for a final landing in a good local minimum of the energy landscape. We can now intuitively interpret the SA search process as a random jump sequence of a "ball" in the energy landscape $E(s)$, where the temperature determines the ability of the ball to (randomly) climb uphill and in this way overcome "energy barriers". Consider the following energy landscape, with a ball bouncing around at high (left) and low (right) temperature:

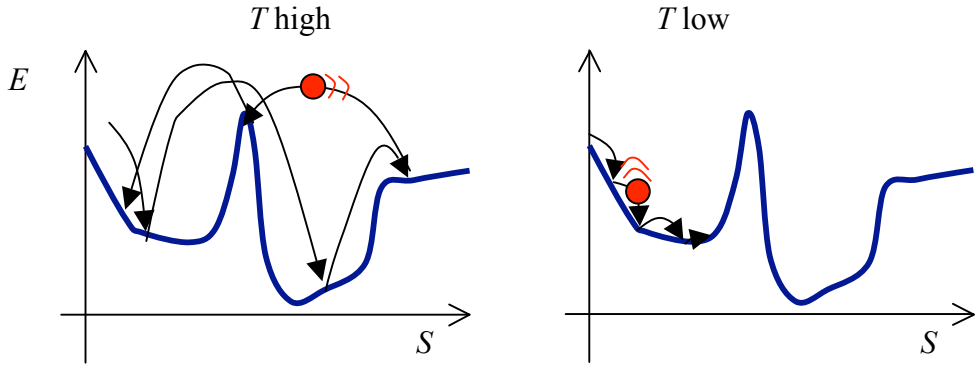


Fig. 5.2: SA seen as an energetic ball game. For explanation see text.

We see that at low temperatures, the search may get trapped in non-optimal regions of the search space that are surrounded by "energy barriers".

Incidentally, this is a danger for the original Metropolis algorithm, too. The pdf $g(s)$ from which one wishes to sample can be converted into an equivalent energy $E(s)$ by putting $T = Z = 1$ and

(5.7) $$E(s) = -\log(g(s)),$$

which – for the sake of checking – gives us a valid Boltzmann probability distribution:

(5.8) $$p(s) = \frac{1}{Z}\exp(-E(s)/T) = \exp(-E(s)) = \exp(\log(g(s))) = g(s).$$

For the case that $g^*(s) < g(s)$, the original Metropolis acceptance probability $P_{\text{accept}} = g^*(s) / g(s)$ can be expressed in terms of energy by

(5.9) $$P_{\text{accept}} = \frac{g^*(s)}{g(s)} = \frac{\exp(-E^*(s))}{\exp(-E(s))} = \exp(E(s) - E(s^*)) = \exp(\Delta E)$$

(where $E(s^*) > E(s)$). Again, the "jumping ball" metaphor makes it clear that the Metropolis algorithm, too, may get trapped in regions surrounded by "energy barriers", that is, (translating back to probabilities) by barriers of low probability.

Returning to the SA, we are now aware that the cooling process is important for the success of running an SA algorithm. A widely used, "quick and dirty" cooling scheme is *exponential cooling*: Put $T_{n+1} = k\, T_n$ for some $k < 1$ (but close to 1). Update T after every single sample. Clearly the size of k is important – a typical way to determine it would be just to experiment.

However, such simple cooling schemes, although widely used, may yield unsatisfactory results. During a SA search run, one may encounter periods where a particularly slow cooling is required, while at other periods, one may cool faster. I will first explain this fact intuitively and then give a mathematical and algorithmical account.

Here are two intuitive approaches to understand the necessity for slower-than-others cooling periods.

The original physical metaphor: simulated annealing. In metallurgy and chemistry, "annealing" refers to a process of carefully cooling a liquid into crystallisation. A general observation is that if a liquid is cooled very quickly, the resulting solid will consist of many fine-granular microcrystals; by contrast, if the cooling is done slowly, large crystals or even a single solitary crystal result. Specifically, the cooling must be slow (heat being withdrawn from the liquid at a low rate) at the solidification temperature, because it is at this temperature that the final crystal structure is determined. You might know from own experience with deep-freezing condiments or producing ice-cream that fast cooling across the solidification temperature produces a substrate structured into many small crystals, whereas slow cooling results in fewer and larger crystals. This is important in both ways in many applications: for instance, when deep-freezing biological specimens (seeds, live cells) it is crucial to "shock-freeze" the material very rapidly, avoiding the growth of larger ice crystals which would destroy the cell membranes; or in producing honey, one way (besides stirring for a long time) to assure that the end-product has a fine crystal-structure is to freeze the freshly harvested, liquid honey (note that it solidifies even above room temperature, albeit slowly). In terms of energy landscape: large crystals corresponds

to microstates of low energy (with solitary crystals corresponding to globally minimal-energetic ground states). Therefore, slow cooling around the critical solidification temperature is a prerequisite for low-energy final products. This correspondence between metallurgy and combinatorial search has given this method its name, simulated annealing.

Naive human problem solving. When solving some magazine puzzle question or a math homework, you will probably have experienced something similar to a phase transition. After an initial thinking phase where you have no clue of how to solve the problem and think of many possible approaches (= a high temperature search phase), the inklings of a solution appear on the horizon (= close to the phase transition), and suddenly! you adopt a particular approach (= beyond the transition) from which it would require quite some effort (= mental re-heating) to escape... If you cooled to quickly (=decided for a solution strategy too early, too quickly) your approach is likely to fail (= lead to a quite suboptimal minimum); if you spent time to consider different solution options (= hover around the critical temperature) and then sloooowly decided, your chances of hitting a good solution strategy are much higher.

There is a physical / mathematical indicator of such *phase transition* temperatures (like at the freezing temperature of water) when it is important to cool very slowly. The *free energy* of a system at temperature T is

(5.10) $$F(T) = E[E(s)] - T\,S(T)$$

where $E[E(s)]$ is the average energy at temperature T,

(5.11) $$E[E(s)] = \int_s E(s)\frac{1}{Z}\exp(-E(s)/T)ds$$

and $S(T)$ is the entropy of the system at temperature T,

(5.12) $$S(T) = -\int_s p(s,T)\log(p(s,T))\ ds\ .$$

Thus the free energy relates the average energy at temperature T with the entropy. Intuitively, the free energy of a system is the "useful" part of its energy, energy that could be exploited at a macroscopic scale (for example, the free energy of a volume of gas would be the energy that one could exploit by expanding the gas in a piston, plus the energy that one might gain from cooling the volume). Phase transitions are defined in physics[16] as discontinuities in the free energy (or one of its derivatives) as temperature (or another macroscopic variable) passes a critical value. For instance, as a volume of water is cooled from some ε value above zero to some ε value below, such that it freezes, one has to extract a certain amount of energy (the melting heat) from that volume – one may in fact exploit this energy; it is part of the free energy of the volume of water. Therefore, the free energy of the volume of water just above zero jumps discontinuously to a lower value as the water is cooled to any T just below freezing.

Similarly, when running SA for a combinatorial search problem, one can in principle compute, at every step n, the free energy F_n of the system, and make the cooling rate depend on the development of the free energy – cool slowly if F_n shows signs of changing rapidly, or in other words, cool in a fashion such F_n decreases smoothly.

[16] http://en.wikipedia.org/wiki/Phase_transition for a very good intro

The free energy can be computed from the partition function Z by observing that

(5.13) $$F(T) = -T \log(Z(T)),$$

which is a homework exercise. So the question is, how can one compute the integral $Z(T) = \int_S \exp(-E(s)/T)\, ds$? The brutal answer is: use sampling![17] for an approximate evaluation of this integral! However, this advice apparently is very expensive: within an SA run (which may have millions of steps!), we repeatedly have to squeeze in complete sampling runs. In the Neal thesis, a number of special-purpose sampling techniques for the partition function are surveyed, none of them cheap and none without problems (Section 6.2, "free energy estimation"). This is an area of active research, because free energy estimation has many other important applications in physics and computational modelling besides steering SA.

Here is a summary of the SA method:

Given: a search space with (very many) microstates s and an objective function $E(s)$, for which a good minimum is sought.

Initialization: Choose a starting temperature T_0 and randomly pick an initial microstate s_0.

Update cycle: Input: T_n and s_n. Output: T_{n+1} and s_{n+1}.

1. Randomly choose a candidate s^*_{n+1} according to some proposal distribution, as known from the Metropolis algorithm (similar considerations apply).
2. Determine T_{n+1}, either by a predefined (e.g., exponential) cooling scheme, or by monitoring the free energy. Note: very often in applications of SA, T is left unchanged in most cycles and is only sometimes re-set to a lower value.
3. Compute $E(s^*_{n+1})$. If this is lower than $E(s_n)$, accept and put $s_{n+1} = s^*_{n+1}$. If $E(s^*_{n+1}) > E(s_n)$, compute $P_{\text{accept}} = \exp((E(s_n) - E(s^*_{n+1}))/T_{n+1})$ and accept s^*_{n+1} with this probability, else reject and put $s_{n+1} = s_n$.

Termination: stop when a predefined (low) stopping temperature is reached, or when the sequence of s_n does not significantly change any more, or when some other suitable termination criterium is met. The last created microstate s_n is the desired result and should have an objective function value close to the global minimum.

[17] See chapter 6.2, "Free Energy Estimation", of the Neal survey for a good intro. Actually, sampling does not lend itself directly to evaluate integrals; rather, it can only easily be used to determine the *ratio* of two integrals. However, it is such ratios that we need for monitoring the development of free energy over time. If the proper value of an integral is sought, only rejection sampling can be used, and the proper value of the integral of the proto-pdf must be known.

5.3 Examples, and a connection to spin glasses

I give a brief review of two of the optimization applications presented in the original article from Kirkpatrick et al: the travelling salesman problem (TSP), and partitioning circuits between microelectronic chips. In addition I provide some background on the Ising model and spin glasses. These are model systems investigated in statistical physics that have a direct bearing on our themes. I rely mostly on the Kirkpatrick article, working in material from the Neal tutorial and the Web. The Kirkpatrick article is very readable and has lots of interesting detail, so my account should be regarded as an appetizer to read the original.

5.3.1 The travelling salesman problem

We start with the TSP. The instance of TSP documented in the original article features 400 cities which were randomly placed in a course structure made from 9 square subblocks on the unit square, as becomes apparent from Fig. 5.3. Intercity distance was Euclidean distance on the plane. The microstates s were circular routes connecting all 400 cities, visiting each one once – formally, these microstates can be regarded as permutations of {2,...,400} (the starting city with index 1 can be fixed w.l.o.g.). The objective function $E(s)$ is the total length of a route. The proposal mechanism, that is, a method to obtain a next microstate s^* from the current s (called a "move" in that article), was to reverse the direction in which a randomly chosen subsection of the tour s was traversed (more on this in the exercise sheet...). Figure 5.3 shows typical tours that were obtained at four temperatures. The effects of cooling become nicely visible, and the microstate sample recorded at the coldest temperature looks intuitively rather optimal – which is about what one can say, because computing the exact global minimum is out of the question here.

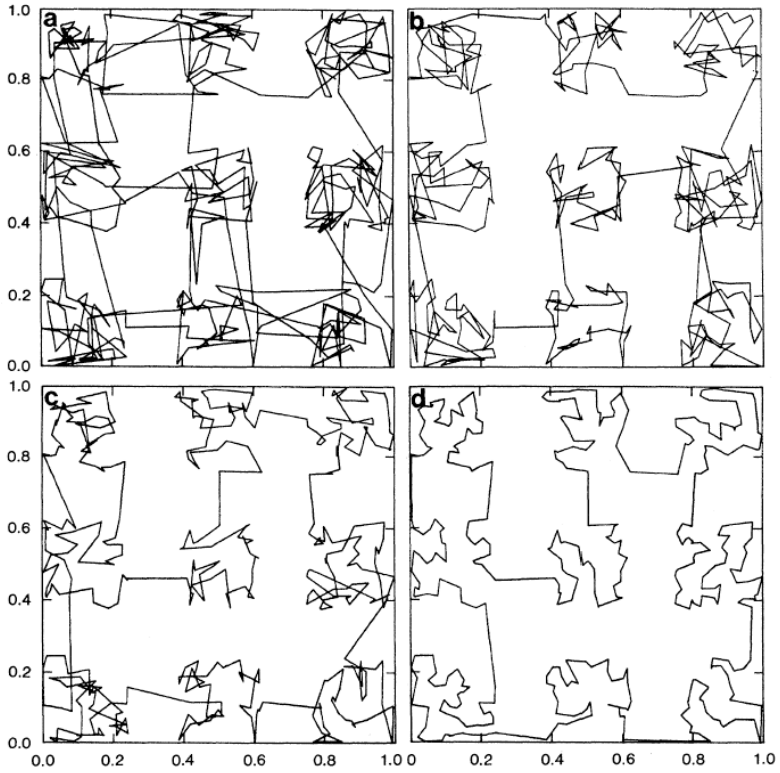


Fig. 5.3. Typical TSP routes obtained by simulated annealing at temperatures T = 1.2 (a), 0.8 (b), 0.4 (c), 0.0 (d). *(Taken from the original SA article by Kirkpatrick et al.)*

5.3.2 Ising systems and spin glasses

Before we turn to the computer layout application, we will briefly introduce a simple thermodynamical model system that has been extensively analysed in statistical physics – the *Ising model* of magnetic solids, and a special case, *spin glasses*. Many lessons learnt from spin glasses have been applied to computational optimization and machine learning, and we will see below that theoretical insights gained from spin glasses can guide the layout of computer hardware!

The (2-dimensional) Ising model is a simplified model of a 2-dimensional solid consisting of atoms arranged on a regular grid. Each atom can be in one of two "spin" states, denoted "up" and "down" (or "+" and "-", or "1" and "0"). You should think of the spin states as orientations of elementary magnets – this is the physical origin of the Ising model. A microstate of an Ising system is an assignment of spins to all atoms; thus an Ising model of $N \times N$ atoms has 2^{N^2} microstates.

Let the atoms be indexed with i, j, and let $\mu_i(s) \in \{-1, +1\}$ be the spin of the i-th atom in a microstate s. In a basic version of the Ising model, the *energy* of a microstate s is given by

$$E(s) = -\sum_{(i,j)\in\mathcal{N}} J\mu_i\mu_j - \sum_i H\mu_i \tag{5.14}$$

where $\mathcal{N}$ is the set of index pairs of directly adjacent atoms (an atom that lies not at the boundary of the $N \times N$ region has four directly adjacent neighbors, left/right/above/below), J is a *coupling strength* constant, and H is the strength of an external magnetic field of "+" orientation. If $J > 0$, states where neighboring atoms have the same orientation are energetically preferred (have lower energy); one then speaks of *ferromagnetic* materials. If $J < 0$, anti-alignment of neighboring atoms is preferred: the case of *anti-ferromagnetic* materials. The second term accounts for a decrease in energy if atoms are aligned with an external magnetic field of "+" orientation.

The overall magnetization of a microstate is defined by

$$M(s) = \sum_i \mu_i \,. \tag{5.15}$$

Ising models have been investigated intensely and in many variations: for different dimensions, different grid topologies, mixtures of ferromagnetic and anti-ferromagnetic effects, etc. Their importance for physics roots in the fact that they are about the simplest models in which fundamental phenomena of statistical thermodynamics arise: phase transitions, percolations, scaling invariances, and many more. Lessons learnt from Ising models have been fruitfully applied in many domains outside physics, for instance image processing (a B/W image can be seen as an Ising model!), machine learning and theoretical neuroscience (the *Boltzmann machine* is a generalization of the Ising model which serves as a model for an associative memory), and many more. We will see below that the Ising model helps to understand problems of computer hardware layout.

One of the first facts discovered (and proven[18]) for the 2D Ising model was a *phase transition* at a *critical temperature*: For large N, when a ferromagnetic Ising system (without external field) is cooled, the mean magnetization (that is, the expectation of $|M(s)|/N$ under the Boltzmann distribution at temperature T) quickly rises from zero to one, with the slope being infinite at the critical temperature $T = 2.269$. Intuitively, at higher temperatures, disordered states with little alignment (and hence no global magnetization) are exceedingly probable, while at lower temperatures, the Boltzmann distribution concentrates on ordered states – at $T = 0$, only the two totally aligned states have nonzero probability.

A nice Java applet for sampling such ferromagnetic Ising states at different temperatures, using the Metropolis algorithm, has been written by Evgeny Demidov. Click at http://www.ibiblio.org/e-notes/Perc/ising.htm and experiment... Figure 5.4 shows snapshots from that simulator, displaying typical Ising states at different temperatures. At low temperatures (left panel), the overwhelming majority of sampled microstates exhibit a dominant orientation (in the example shown, the "+" [= "white"] orientation). At high temperatures (right panel), states are very "noisy" and globally totally disordered. Around the critical temperature (middle), states exhibit *scale invariance*: the spatial patterns formed by the "+" and "-" oriented atoms appear the same at different levels of magnification.

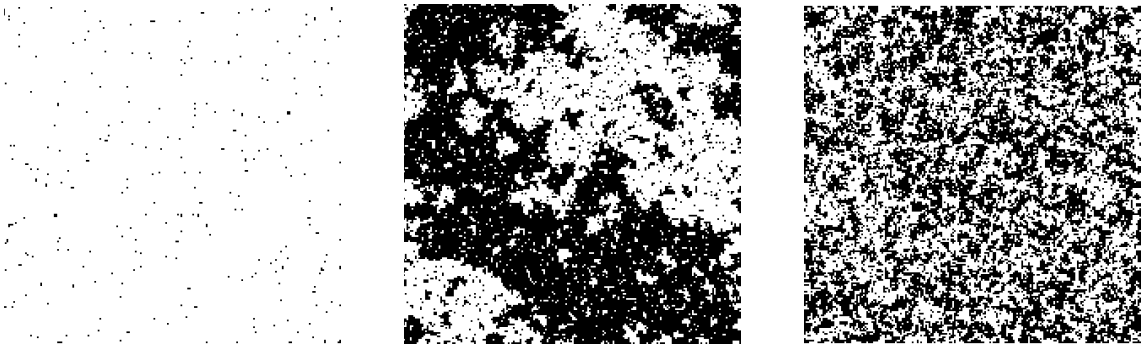


Fig. 5.4. Typical microstates, as sampled with the Metropolis algorithm from a 200 × 200 atom Ising system at temperatures 1.3, 2.269 (critical T), and 3.0 (from left to right). *(Obtained with E. Demidov's online simulator).* For details compare text.

In real magnetic materials, often a mixture of ferromagnetic and anti-ferromagnetic phenomena is observed. In iron, for instance, on a small scale quantum-mechanical rules favour ferromagnetic alignment of neighboring iron atoms. However, on a larger scale (some thousands of atoms), the familiar tendency of magnetic dipoles to anti-align starts to dominate. In iron there are thus two counter-acting tendencies, which could be expressed by splitting in Eqn. (5.14) the first term into two, one with a positive J and one with a negative J – these two terms would of course have to be defined for different spatial scales. An important consequence of these two conflicting tendencies is that there exists no single (or a small number) of ground states in a block of iron – the lowest-energy states all are "compromise" states consisting of local regions with a high degree of internal alignment, but counter-alignment between these regions. Systems where conflicting energy-minimizing mechanisms lead to a large number of compromise ground states are called *frustrated* systems in general, and in the context of Ising models and magnetism they are called *spin glasses* (see

[18] By later Nobel laureate Lars Onsager, http://en.wikipedia.org/wiki/Lars_Onsager for a nice intro see http://en.wikipedia.org/wiki/Ising_model

http://en.wikipedia.org/wiki/Spin_glass, http://en.wikipedia.org/wiki/Ferromagnetism for good intros).

5.3.3 Computer hardware layout: circuit partitioning subtask

According to Kirkpatrick et al., whose paper I follow in this subsection sometimes verbatim, in computer hardware layout one is confronted with a number of subproblems that build and depend on each other, with *partitioning* being the most elementary (the article also treats metrical placement and wiring). The partitioning task, in this paper, is to distribute a set of about 5000 circuits (forming a complete CPU architecture) over two microchips such that (i) the number of pins is low and (ii) the circuits are distributed approximately in equal numbers across the microchips.

Formally, this task can be specified very much in the spirit of Ising models through microstates s, where each microstates s assigns a value $\mu_i(s) \in \{-1, +1\}$ to the i-th circuit (i = 1, ..., 5000), depending on which chip the circuit becomes assigned to. The objective function must reflect the two requirements (i) and (ii). Expanding on the treatment in the original article (it provides no detail), this can be done as follows, always in a true Ising spirit:

For (i), consider a 5000 × 5000 matrix (a_{ij}) with 0-1 entries, a value of $a_{ij} = 1$ indicating that circuits i and j directly exchange a signal (and hence, if placed on different chips, require a pin at each chip). The number of signals that must cross a chip boundary is

$$\sum_{i>j} (a_{ij}/4)(\mu_i - \mu_j)^2 = -\sum_{i>j} (a_{ij}/2)\mu_i\mu_j + \sum_{i>j} (a_{ij}/2).$$

The second sum term is independent of the circuit placement and can be dropped from the energy function, because it does not affect the location of its minima.

For (ii), the objective function should grow with the degree of imbalance of circuits assigned to the two chips. The squared imbalance score $\left(\sum_i \mu_i\right)^2$ is equal to $2\sum_{i>j} \mu_i\mu_j + \sum_i \mu_i^2$.

Again, the second sum term is independent of the placement and can be dropped.

Assembling these two contributions and replacing the constant 2 by a weighting factor λ, one gets an objective function of the form

$$E(s) = \sum_{i>j} \lambda\mu_i\mu_j - \sum_{i>j} a_{ij}\mu_i\mu_j, \tag{5.16}$$

which reveals that the partitioning problem is an instance of a frustrated (spin glass like) system (the first sum term codes a global interaction contribution to the energy, the second one that is "local" with respect to the connection topology induced by the nonzero a_{ij}) This has important consequences for the optimization through simulated annealing (here I cite almost verbatim from Kirkpatrick et al):

1. Because spin glasses have many ground states, there will be many good near-optimal solutions, so a stochastic search like SA should find some.

2. No one of the ground states is significantly better than the others, so it is not very fruitful to search for the absolute optimum.

The SA scheme used by Kirkpatrick et al. involved a proposal scheme that simply flipped the assignment of a randomly chosen circuit. The temperature was lowered with a factor of 0.9 from one temperature to the next lower one. Starting at $T = 10$, at each temperature in the order of 500,000 flips were executed, until a temperature of 0.1 was reached (from which I infer a total runlength of about 20,000,000 updates). Figure 5.5 summarizes the distributions of number of pins obtained at different temperatures. As expected, the average number of pins sampled at decreasing temperatures decreases, as does the variance of that number. At the lowest temperature, the sampling has frozen into apparently a single (or very few) solution(s).

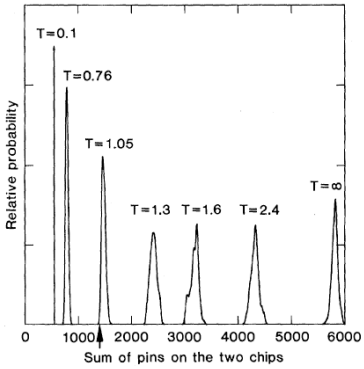


Fig. 5.5: Distributions of total number of pins at various temperatures. The arrow indicates the average number of pins obtained by a greedy search algorithm (Metropolis sampling with $T = 0$). *(From the Kirkpatrick et al article)*

5.4 The Boltzmann Machine

The Boltzmann Machine (BM), introduced in 1985 by Ackley, Hinton and Sejnowski[19], is not actually a machine but an abstract "neural network" model for a representation of complex distributions – or, stated in the terms of cognitive science which is the appropriate background: a model of a contents-addressable, associative memory.

Before going into the technical aspects, I want to explain the background intuitions and the intended use of BMs. A good way to approach BMs is to see them as an abstract model of human memory, including mechanisms for storing, retrieving, associating between stored items, and completing corrupted inputs. Let's consider the question what is the nature of a human's (long-term) memory of the written digit pattern for the number "four" (in the Times bold font: **4**). Numerous and diverse answers to this question have been given in the fields of cognitive science and neuroscience, for instance:

(2.1) The "digit-pattern-4-memory" is a stored *prototype*, that is, we have a conceptual/neural representation of some *ideal*, *prototypical* pattern, -- let's say, something very clean and clear that looks like this:

[19] Ackley, D.H. and Hinton, G.E. and Sejnowski, T.J.: A Learning Algorithm for Boltzmann Machines. Cognitive Science 9 (1985), 147-169. Electronic copy for this course at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/1385_Ackleyetal85.pdf

4

In order to recognize new incoming instances of the digit "four", like this one:

the stored prototype is matched against the new input, which is classified as "four" if the match is close enough. The prototype view is one of the classical models of memory in cognitive science.

(2.2) The "digit-pattern-4-memory" is a set of *processing rules*, which specify how low-level features which are extracted from a visual input can (and must) be combined in order to be recognized as a "four" pattern. This is another classical model, especially in AI and pattern recognition.

(2.3) The "digit-pattern-4-memory" is a huge set of *contextual expectations* (or *anticipations* or *affordances*) which specify in which contexts one should expect the pattern "four" to appear, and when it appears, which further actions or perceptions are likely to occur. This would be the approach of the rather recent and flourishing school of thought of *anticipatory representations*.

This is only a selection among numerous other models of the nature of long-term memory. The most confusing part of this picture is that for each kind of model there is good empirical evidence from psychological or neurophysiological studies.

The BM should be seen on this background of a multitude of models of long-term memory (LTM), because it adds another such model. The fundamental assumption of the BM is that memory is a *generative model of a probability distribution*. Coming back to the pattern "four" example, our memory of this pattern should be seen as a distribution over possible variations of that pattern. A sample from this distribution might look like in figure **5.6**.

Figure 5.6. A sample from a distribution of the pattern "four". (Taken from the widely used MNIST digit benchmark dataset, www.cvl.isy.liu.se/ImageDB/images/external_images/MNIST_digits/index.html)

The BM model of memory is *generative*. In technical terms this means that the BM comes complete with a sampling algorithm, which allows to produce sample items from the memorized distribution. In more intuitive terms one could say that the BM can be run in a mode of "hallucination" or "dreaming" – the term that is mostly used is to say that the BM can *confabulate* pattern samples.

The mathematical format of a BM is closely related to Ising models, and the generating algorithm will be the Metropolis sampler. We now describe the BM in detail.

In its elementary version, a BM describes a probability distribution over N-dimensional binary patterns $\mathbf{v} = (v_1, ..., v_N)^T$, where now $v_i \in \{0, 1\}$ (think of V as a black-and-white image and the v_i as pixels). In line with the neural networks background of BMs, the pixel cells are called "neurons" or "units" (and the entire set of N units can be thought of as a "retina"). Similarly, the values 0 and 1 are called "activations"[20].

To augment the representational capacity of a BM, it is usually equipped with additional *hidden* units $\mathbf{h} = (h_1, ..., h_M)^T$. These are binary units too, i.e. $h_i \in \{0, 1\}$. If we want to refer to both types of units without distinguishing between them, we write $\mathbf{s} = (s_1, ..., s_{N+M})^T \in \{0,1\}^{N+M}$ to denote complete BM states (these are the microstates in the Boltzmann sense).

In a BM, there is an undirected *link* between any two visible and/or hidden units s_i, s_j (in the neural networks metaphor, think of this link as a "synaptic connection" between two neurons.) Each link is characterized by some real number $w_{ij} \in \mathbb{R}$, its *weight*. Special case: setting the weight to zero amounts to cancelling the link altogether. Figure 5.7 gives an impression of a BM set up with two types of visible units, the ones in the retinal "input" layer where geometric patterns are inputted (or confabulated), and a "classification" layer where there is one visible unit for each class of patterns (i.e., there is one unit for the class of all "one" patterns, one for all "two" patterns, etc.). Only a few of the pairwise links between units have been indicated by lines for clarity.

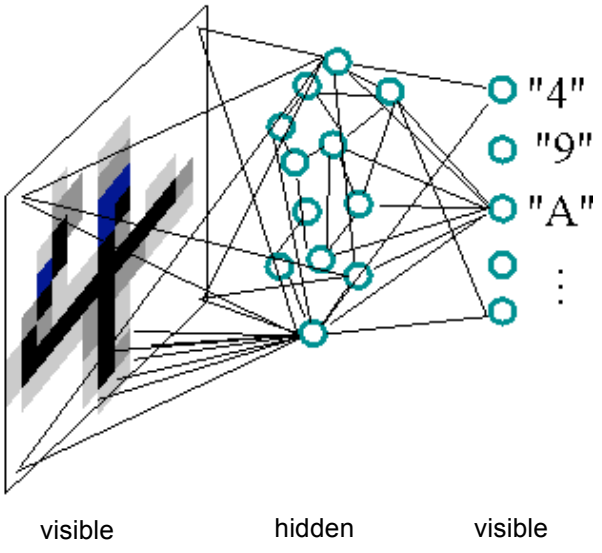


Figure 5.7. Schematic of a BM architecture for memorizing and classifying digits and letter patterns. For explanation see text.

The energy of a microstate in a BM is defined by

[20] I follow the original work of Ackely, Hinton and Sejnowski here. In a number of textbooks one will find a version of the BM which has activations {-1, +1} as in the Ising model. It's a matter of taste.

(5.17) $$E(\mathbf{s}) = -\sum_{i<j} w_{ij} s_i s_j .$$

That is, each particular fixed setting of the weights defines a particular energy landscape over the microstates.

One sometimes adds a third type of unit, a *bias* input unit. There is only one such unit and its activation is unchangeably fixed at a value of 1. If such a bias unit is used, the energy function changes to

(5.18) $$E(\mathbf{s}) = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i,$$

where θ_i is the connection weight between the *i*-th unit and the bias input unit. A bias input is a convenience feature that is often used in neural network architectures; it has several practical and mathematical merits that we do not discuss here. We will work without a bias input in the sequel.

If a temperature *T* is fixed, a BM defines a probability distribution over the visible units, as their marginal distribution of the resulting Boltzmann distribution:

(5.19) $$\begin{aligned} P(\mathbf{s}) &= \frac{1}{Z}\exp(-E(\mathbf{s})/T) \\ P(\mathbf{v}) &= \sum_{\mathbf{s}:(s_1 \ldots s_N)=\mathbf{v}} P(\mathbf{s}) \end{aligned}$$

According to the energy function (5.17), if the *k*-th unit has an activation of zero (i.e. $s_k = 0$), all sum terms in which s_k appears are zero: inactive units contribute nothing to the energy of the current microstate. If s_k jumps from 0 to 1, while all other units stay in their activation states, the energy (5.17) changes by adding the amount

(5.20) $$-\Delta E_k = -\sum_j w_{kj} s_j .$$

Sampling from a BM is done with the Gibbs sampler. The dimensions along which this sampler cycles along are the units s_i. Assume that at some point in the sampling cycle we are visiting the dimension (unit) s_k. In order to carry out the sampling at this dimension, we need to know the conditional distribution of s_k given the current state of the other s_j. Let **s'** be the microstate where $s_k = 1$ and **s** the microstate where $s_k = 0$. Now compute the ratio

(5.21) $$\begin{aligned} \frac{P(s_k = 1 \mid \text{ all other states})}{P(s_k = 0 \mid \text{ all other states})} = \frac{P(\mathbf{s}')}{P(\mathbf{s})} &= \frac{\exp(-E(\mathbf{s}')/T)}{\exp(-E(\mathbf{s})/T)} \\ &= \frac{\exp(-(E(\mathbf{s}) - \Delta E_k)/T)}{\exp(-E(\mathbf{s})/T)} = \exp(\Delta E_k / T). \end{aligned}$$

From $P(s_k = 0 \mid \text{ all other states}) = 1 - P(s_k = 1 \mid \text{ all other states})$ it then follows that

(5.22) $$P_k := P(s_k = 1 \mid \text{all other states}) = \frac{1}{1+\exp(-\Delta E_k / T)}..$$

To create a next sample **s'** from a previous sample with Gibbs sampling, while working at dimension s_k, thus means to set s_k to a value of 1 (regardless of its previous value) with this probability P_k.

A beautiful movie demonstration of a digit-confabulator can be found at Geoffrey Hinton's webpage at http://www.cs.toronto.edu/~hinton/digits.html. This architecture has two more hidden "layers" than sketched out in figure 5.7 but is otherwise similar. The gray level rendering of the retinal visible units in this online movie and in figure 5.7 is obtained by not plotting states (which would be binary B/W), but instead a suitably normalized version of the unit's energy contribution ΔE_k.

This is the first half of the BM story. The second, and more intriguing, half concerns the question how the weights w_{ij} are *learnt* from training data. The scenario is as follows:

- A finite *training set* of samples $D = \{\mathbf{v}^1, \ldots, \mathbf{v}^n\}$ of visible states from an unknown distribution P is all that is available to start with. For example, the MNIST training set (available at Yann LeCun's website http://yann.lecun.com/exdb/mnist/) is a collection of tens of thousands of handwritten digits; figure 5.6 gives a small subset.
- From these training examples, BM weights w_{ij} have to be learnt such that the resulting Boltzmann distribution $\hat{P}$ (at some predetermined fixed temperature T) is as similar as possible to P.
- One common way to measure the similarity of two (discrete) distributions P and $\hat{P}$ is to use their *Kullback-Leibler distance*

(5.23) $$KL(P,\hat{P}) = \sum_{\mathbf{s}} P(\mathbf{s}) \log \frac{P(\mathbf{s})}{\hat{P}(\mathbf{s})}.$$

  The KL distance is not actually a true distance (it is not symmetric). It is always nonnegative and is zero if and only if the distributions P and $\hat{P}$ are identical.
- Not knowing P, only having available the sample set D, one uses instead of the true distribution P the *empirical distribution* P_0 which is given simply by the "histogram" of the training samples. Notice that due to the high dimensionality of pixel image vectors, we should expect to encounter the curse of dimensionality here: the empirical distribution given by databases like MNIST will typically have singleton "counts" at best and should be imagined to be something like is shown in figure 3.5a. I will ask in class why this case is less quite hopeless after all than it first appears… (Hint: the N-dimensional pixel space is populated by training instances in a very uneven way; all samples will be concentrated in a small, low-dimensional submanifold of the N-dimensional hypercube)
- Use gradient descent to minimize $KL(P_0,\hat{P})$.

The beauty of BMs lies in the circumstance that actually the last point is easy: a little non-deep maths[21] reveals that

[21] This derivation can be found in the appendix of the Ackley/Hinton/Sejnowski paper.

(5.24) $$\frac{\partial KL(P_0,\hat{P})}{\partial w_{ij}} = -\frac{1}{T}(p_{ij} - \hat{p}_{ij}),$$

where p_{ij} is the average (over training samples) probability that the units i and j are both active (that is, $s_i = 1$ and $s_j = 1$) when the visible units are set externally to the training samples, and where $\hat{p}_{ij}$ is the probability that these two units are simultaneously active in a "free-running" sampling mode with no external constraints. This yields the following update rule for weights:

(5.25) $$w_{ij}(n+1) = w_{ij}(n) + \varepsilon(p_{ij} - \hat{p}_{ij}),$$

where $w_{ij}(n)$ is the value of the weight w_{ij} at the n-th learning step and ε is a (small) learning rate. Notice that a BM is operated (trained and used) at a fixed temperature, so the learning rule (5.25) does not depend on T. A single weight update step $w_{ij}(n) \rightarrow w_{ij}(n+1)$ involves the following operations:

1. Estimation of p_{ij}: for each training sample $\mathbf{v}^\alpha$, "clamp" (that's a technical term in the neural networks world) the visible units to the values of $\mathbf{v}^\alpha$. While the visible units remain clamped to $\mathbf{v}^\alpha$, run the sampling algorithm until a reasonably representative sample of states of the equilibrium dynamics has been obtained. Use this sample to estimate p_{ij}^α, the probability of units i and j being co-active under clamping by $\mathbf{v}^\alpha$. Do this for all $\alpha = 1, ..., n$. Finally, set p_{ij} to the average of all of these p_{ij}^α. All of this is sometimes called the "wake" phase of the BM learning algorithm.
2. Estimation of $\hat{p}_{ij}$: Similar to one of the substeps of estimating p_{ij}^α in the previous step, only without clamping the visible units. This is sometimes called the "sleep" phase.
3. Weight update: apply (5.25).

It is clear where the catch with this procedure lies: for a single weight update step, one has to run as many complete sampling runs as there are training patterns! At face value, this is simply prohibitive.

In their original paper, Ackley, Hinton and Sejnowski introduce a number of simplifications which allow them to learn some demo examples, even with the slower computers of the early 80'ies. The main simplifications are the following:

1. Instead of (5.25), use $w_{ij}(n+1) = w_{ij}(n) + c\,\mathrm{sign}(p_{ij} - \hat{p}_{ij})$, where c is a constant weight increment and sign is the signum function. This reduces computational load because determining the *sign* of $(p_{ij} - \hat{p}_{ij})$ needs a less accurate sampling than estimating the *size* of this quantity.
2. For each sampling run in steps 1 or 2 above, use a two-phase procedure. In the first phase, start from a higher temperature than the agreed T and carry out a simulated-annealing like cooling from the higher temperature to T. In the second phase, sample at the standard temperature T. The initial annealing phase is intended to prevent that the (randomly generated) starting state is stuck in some narrow, untypical local energy minimum.

Despite these simplifications and tricks (of which the second is dubious if you ask me), the intrinsic computational challenge of a very large number of sampling runs needed to determine weight updates is not dissolved. This is probably a good enough reason to explain that Boltzmann Machines never were used in practical applications. However, the underlying idea is both so simple and so potentially powerful that Geoffrey Hinton and other academic researchers continued to research BM architectures in the decades since. I probably wouldn't have elected BMs as suitable for a lecture nonetheless, if not recently many researchers in machine learning (including myself) have freshly become excited about BMs. This stir was triggered by a Science paper by Hinton and R. R. Salakhutdinov[22], where several developments which started from BMs were combined into a strikingly powerful architecture, now often dubbed *deep belief networks* (DBNs), for learning complex distributions:

- DBNs are layered neural networks, where each layer corresponds to one BM. The hidden units of one such BM make the visible units of the next-higher BM.
- The connectivity of the participating BMs is very much reduced: there are no in-layer connections, only connections between adjacent layers. This type of BM has been called "restricted Boltzmann Machine (RBM)"
- The learning is done in a divide-and-conquer fashion layer by layer.
- Each sampling subroutine is condensed to only two state updates of one layer BM, using a shortcut approximate algorithm called *contrastive divergence*.

These innovations together have brought BMs back on stage with a flourish, under the name of RBMs or DBNs. I can't possibly describe DBNs better (nor more concisely) than Hinton & Salakhutinov did in their Science paper, so the latter is mandatory reading at this point. I insert a TIFF version for convenience; if you want high-resolution pdf, you will have to download the paper from http://www.sciencemag.org/cgi/content/abstract/313/5786/504 .

[22] G. E. Hinton and R. R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks. Science 313 (July 28, 2006), 504-507. Online at http://www.sciencemag.org/cgi/content/abstract/313/5786/504 Due to copyright reasons I cannot make this paper electronically available in a publicly accessible way, but Jacobs members should be able to download it directly from the indicated URL.

materials are identical for all configurations. The blue bars in Fig. 1 summarize the measured SHG signals. For excitation of the *LC* resonance in Fig. 1A (horizontal incident polarization), we find an SHG signal that is 500 times above the noise level. As expected for SHG, this signal closely scales with the square of the incident power (Fig. 2A). The polarization of the SHG emission is nearly vertical (Fig. 2B). The small angle with respect to the vertical is due to deviations from perfect mirror symmetry of the SRRs (see electron micrographs in Fig. 1). Small detuning of the *LC* resonance toward smaller wavelength (i.e., to 1.3-μm wavelength) reduces the SHG signal strength from 100% to 20%. For excitation of the Mie resonance with vertical incident polarization in Fig. 1D, we find a small signal just above the noise level. For excitation of the Mie resonance with horizontal incident polarization in Fig. 1C, a small but significant SHG emission is found, which is again po-

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

to transform the high-dimensional data into a low-dimensional code and a similar "decoder" network to recover the data from the code. Starting with random weights in the two networks, they can be trained together by minimizing the discrepancy between the original data and its reconstruction. The required gradients are easily obtained by using the chain rule to backpropagate error derivatives first through the decoder network and then through the encoder network (*1*). The whole system is called an "autoencoder" and is depicted in Fig. 1.

Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario M5S 3G4, Canada.

*To whom correspondence should be addressed; E-mail: hinton@cs.toronto.edu

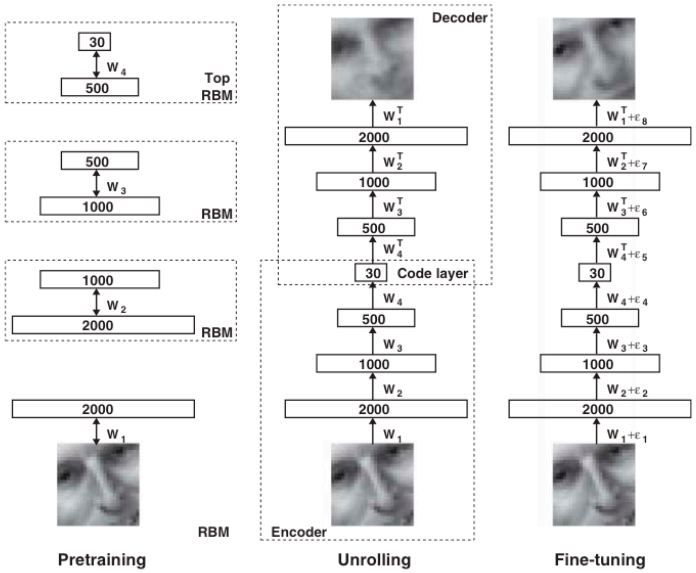


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

It is difficult to optimize the weights in nonlinear autoencoders that have multiple hidden layers (*2*–*4*). With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. If the initial weights are close to a good solution, gradient descent works well, but finding such initial weights requires a very different type of algorithm that learns one layer of features at a time. We introduce this "pretraining" procedure for binary data, generalize it to real-valued data, and show that it works well for a variety of data sets.

An ensemble of binary vectors (e.g., images) can be modeled using a two-layer network called a "restricted Boltzmann machine" (RBM) (*5*, *6*) in which stochastic, binary pixels are connected to stochastic, binary feature detectors using symmetrically weighted connections. The pixels correspond to "visible" units of the RBM because their states are observed; the feature detectors correspond to "hidden" units. A joint configuration ($\mathbf{v}$, $\mathbf{h}$) of the visible and hidden units has an energy (*7*) given by

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i\in \text{pixels}} b_i v_i - \sum_{j\in \text{features}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \quad (1)$$

where v_i and h_j are the binary states of pixel i and feature j, b_i and b_j are their biases, and w_{ij} is the weight between them. The network assigns a probability to every possible image via this energy function, as explained in (*8*). The probability of a training image can be raised by

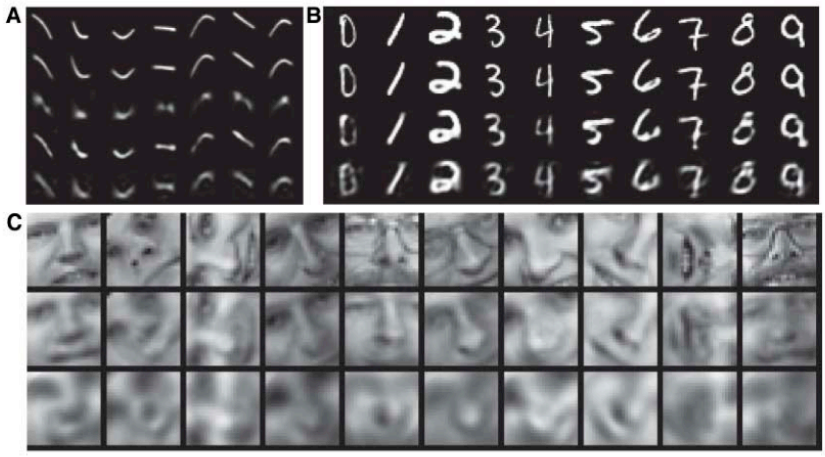


Fig. 2. (**A**) Top to bottom: Random samples of curves from the test data set; reconstructions produced by the six-dimensional deep autoencoder; reconstructions by "logistic PCA" (*8*) using six components; reconstructions by logistic PCA and standard PCA using 18 components. The average squared error per image for the last four rows is 1.44, 7.64, 2.45, 5.90. (**B**) Top to bottom: A random test image from each class; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional logistic PCA and standard PCA. The average squared errors for the last three rows are 3.00, 8.01, and 13.87. (**C**) Top to bottom: Random samples from the test data set; reconstructions by the 30-dimensional autoencoder; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

adjusting the weights and biases to lower the energy of that image and to raise the energy of similar, "confabulated" images that the network would prefer to the real data. Given a training image, the binary state h_j of each feature detector j is set to 1 with probability $\sigma(b_j + \sum_i v_i w_{ij})$, where $\sigma(x)$ is the logistic function $1/[1 + \exp(-x)]$, b_j is the bias of j, v_i is the state of pixel i, and w_{ij} is the weight between i and j. Once binary states have been chosen for the hidden units, a "confabulation" is produced by setting each v_i to 1 with probability $\sigma(b_i + \sum_j h_j w_{ij})$, where b_i is the bias of i. The states of the hidden units are then updated once more so that they represent features of the confabulation. The change in a weight is given by

$$\Delta w_{ij} = \varepsilon\left(\langle v_i h_j\rangle_{\text{data}} - \langle v_i h_j\rangle_{\text{recon}}\right) \quad (2)$$

where ε is a learning rate, $\langle v_i h_j\rangle_{\text{data}}$ is the fraction of times that the pixel i and feature detector j are on together when the feature detectors are being driven by data, and $\langle v_i h_j\rangle_{\text{recon}}$ is the corresponding fraction for confabulations. A simplified version of the same learning rule is used for the biases. The learning works well even though it is not exactly following the gradient of the log probability of the training data (*6*).

A single layer of binary features is not the best way to model the structure in a set of images. After learning one layer of feature detectors, we can treat their activities—when they are being driven by the data—as data for learning a second layer of features. The first layer of feature detectors then become the visible units for learning the next RBM. This layer-by-layer learning can be repeated as many

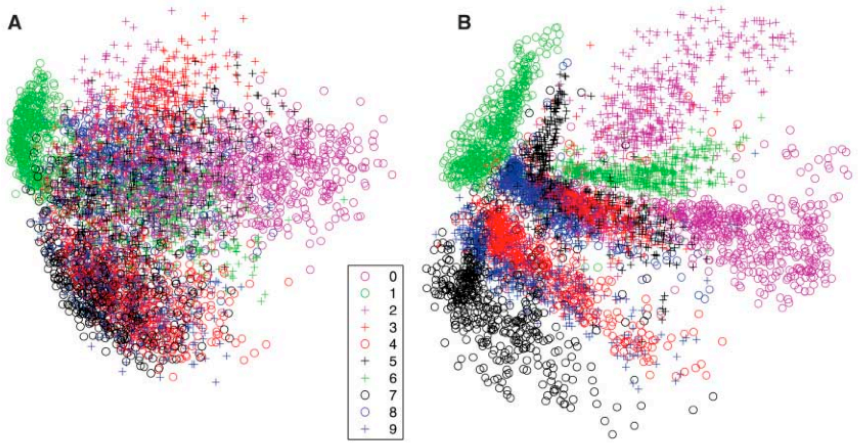


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

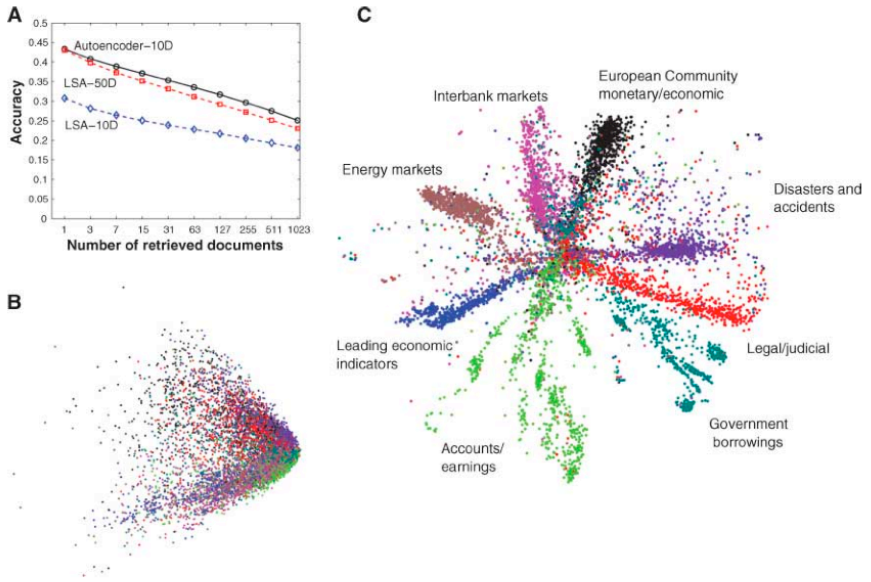


Fig. 4. (**A**) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (**B**) The codes produced by two-dimensional LSA. (**C**) The codes produced by a 2000-500-250-125-2 autoencoder.

times as desired. It can be shown that adding an extra layer always improves a lower bound on the log probability that the model assigns to the training data, provided the number of feature detectors per layer does not decrease and their weights are initialized correctly (*9*). This bound does not apply when the higher layers have fewer feature detectors, but the layer-by-layer learning algorithm is nonetheless a very effective way to pretrain the weights of a deep autoencoder. Each layer of features captures strong, high-order correlations between the activities of units in the layer below. For a wide variety of data sets, this is an efficient way to progressively reveal low-dimensional, nonlinear structure.

After pretraining multiple layers of feature detectors, the model is "unfolded" (Fig. 1) to produce encoder and decoder networks that initially use the same weights. The global fine-tuning stage then replaces stochastic activities by deterministic, real-valued probabilities and uses backpropagation through the whole autoencoder to fine-tune the weights for optimal reconstruction.

For continuous data, the hidden units of the first-level RBM remain binary, but the visible units are replaced by linear units with Gaussian noise (*10*). If this noise has unit variance, the stochastic update rule for the hidden units remains the same and the update rule for visible unit i is to sample from a Gaussian with unit variance and mean $b_i + \sum_j h_j w_{ij}$.

In all our experiments, the visible units of every RBM had real-valued activities, which were in the range [0, 1] for logistic units. While training higher level RBMs, the visible units were set to the activation probabilities of the hidden units in the previous RBM, but the hidden units of every RBM except the top one had stochastic binary values. The hidden units of the top RBM had stochastic real-valued states drawn from a unit variance Gaussian whose mean was determined by the input from that RBM's logistic visible units. This allowed the low-dimensional codes to make good use of continuous variables and facilitated comparisons with PCA. Details of the pretraining and fine-tuning can be found in (*8*).

To demonstrate that our pretraining algorithm allows us to fine-tune deep networks efficiently, we trained a very deep autoencoder on a synthetic data set containing images of "curves" that were generated from three randomly chosen points in two dimensions (*8*). For this data set, the true intrinsic dimensionality is known, and the relationship between the pixel intensities and the six numbers used to generate them is highly nonlinear. The pixel intensities lie between 0 and 1 and are very non-Gaussian, so we used logistic output units in the autoencoder, and the fine-tuning stage of the learning minimized the cross-entropy error $[-\sum_i p_i \log \hat{p}_i - \sum_i (1 - p_i) \log(1 - \hat{p}_i)]$, where p_i is the intensity of pixel i and $\hat{p}_i$ is the intensity of its reconstruction.

The autoencoder consisted of an encoder with layers of size (28×28)-400-200-100-50-25-6 and a symmetric decoder. The six units in the code layer were linear and all the other units were logistic. The network was trained on 20,000 images and tested on 10,000 new images. The autoencoder discovered how to convert each 784-pixel image into six real numbers that allow almost perfect reconstruction (Fig. 2A). PCA gave much worse reconstructions. Without pretraining, the very deep autoencoder always reconstructs the average of the training data, even after prolonged fine-tuning (*8*). Shallower autoencoders with a single hidden layer between the data and the code can learn without pretraining, but pretraining greatly reduces their total training time (*8*). When the number of parameters is the same, deep autoencoders can produce lower reconstruction errors on test data than shallow ones, but this advantage disappears as the number of parameters increases (*8*).

Next, we used a 784-1000-500-250-30 autoencoder to extract codes for all the handwritten digits in the MNIST training set (*11*). The Matlab code that we used for the pretraining and fine-tuning is available in (*8*). Again, all units were logistic except for the 30 linear units in the code layer. After fine-tuning on all 60,000 training images, the autoencoder was tested on 10,000 new images and produced much better reconstructions than did PCA (Fig. 2B). A two-dimensional autoencoder produced a better visualization of the data than did the first two principal components (Fig. 3).

We also used a 625-2000-1000-500-30 autoencoder with linear input units to discover 30-dimensional codes for grayscale image patches that were derived from the Olivetti face data set (*12*). The autoencoder clearly outperformed PCA (Fig. 2C).

When trained on documents, autoencoders produce codes that allow fast retrieval. We represented each of 804,414 newswire stories (*13*) as a vector of document-specific probabilities of the 2000 commonest word stems, and we trained a 2000-500-250-125-10 autoencoder on half of the stories with the use of the multiclass cross-entropy error function $[-\sum_i p_i \log \hat{p}_i]$ for the fine-tuning. The 10 code units were linear and the remaining hidden units were logistic. When the cosine of the angle between two codes was used to measure similarity, the autoencoder clearly outperformed latent semantic analysis (LSA) (*14*), a well-known document retrieval method based on PCA (Fig. 4). Autoencoders (*8*) also outperform local linear embedding, a recent nonlinear dimensionality reduction algorithm (*15*).

Layer-by-layer pretraining can also be used for classification and regression. On a widely used version of the MNIST handwritten digit recognition task, the best reported error rates are 1.6% for randomly initialized backpropagation and 1.4% for support vector machines. After layer-by-layer pretraining in a 784-500-500-2000-10 network, backpropagation using steepest descent and a small learning rate achieves 1.2% (*8*). Pretraining helps generalization because it ensures that most of the information in the weights comes from modeling the images. The very limited information in the labels is used only to slightly adjust the weights found by pretraining.

It has been obvious since the 1980s that backpropagation through deep autoencoders would be very effective for nonlinear dimensionality reduction, provided that computers were fast enough, data sets were big enough, and the initial weights were close enough to a good solution. All three conditions are now satisfied. Unlike nonparametric methods (*15*, *16*), autoencoders give mappings in both directions between the data and code spaces, and they can be applied to very large data sets because both the pretraining and the fine-tuning scale linearly in time and space with the number of training cases.

References and Notes

1. D. C. Plaut, G. E. Hinton, *Comput. Speech Lang.* **2**, 35 (1987).
2. D. DeMers, G. Cottrell, *Advances in Neural Information Processing Systems 5* (Morgan Kaufmann, San Mateo, CA, 1993), pp. 580–587.
3. R. Hecht-Nielsen, *Science* **269**, 1860 (1995).
4. N. Kambhatla, T. Leen, *Neural Comput.* **9**, 1493 (1997).
5. P. Smolensky, *Parallel Distributed Processing: Volume 1: Foundations*, D. E. Rumelhart, J. L. McClelland, Eds. (MIT Press, Cambridge, 1986), pp. 194–281.
6. G. E. Hinton, *Neural Comput.* **14**, 1711 (2002).
7. J. J. Hopfield, *Proc. Natl. Acad. Sci. U.S.A.* **79**, 2554 (1982).
8. See supporting material on *Science* Online.
9. G. E. Hinton, S. Osindero, Y. W. Teh, *Neural Comput.* **18**, 1527 (2006).
10. M. Welling, M. Rosen-Zvi, G. Hinton, *Advances in Neural Information Processing Systems 17* (MIT Press, Cambridge, MA, 2005), pp. 1481–1488.
11. The MNIST data set is available at http://yann.lecun.com/exdb/mnist/index.html.
12. The Olivetti face data set is available at www.cs.toronto.edu/ roweis/data.html.
13. The Reuter Corpus Volume 2 is available at http://trec.nist.gov/data/reuters/reuters.html.
14. S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman, *J. Am. Soc. Inf. Sci.* **41**, 391 (1990).
15. S. T. Roweis, L. K. Saul, *Science* **290**, 2323 (2000).
16. J. A. Tenenbaum, V. J. de Silva, J. C. Langford, *Science* **290**, 2319 (2000).

17. We thank D. Rumelhart, M. Welling, S. Osindero, and S. Roweis for helpful discussions, and the Natural Sciences and Engineering Research Council of Canada for funding. G.E.H. is a fellow of the Canadian Institute for Advanced Research.

Supporting Online Material

www.sciencemag.org/cgi/content/full/313/5786/504/DC1
Materials and Methods
Figs. S1 to S5
Matlab Code

20 March 2006; accepted 1 June 2006
10.1126/science.1127647

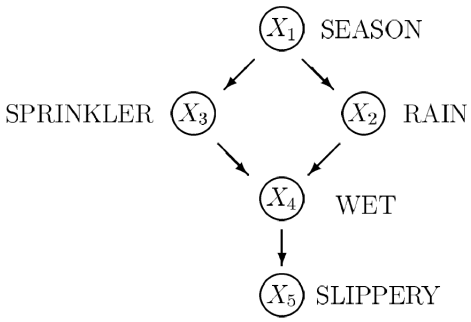
6. Bayesian networks and graphical models

6.1 Introduction[23]

A Bayesian network is "just" an orderly, graph-based way of representing the statistical dependencies between a number of random variables. For a very first impression, we consider a (classical) example. Let X_1, ..., X_5 be five discrete random variables, indicating the following observations:

X_1: indicates the season of the year, has values from {Winter, Spring, Summer, Fall}
X_2: indicates whether it rains, has values {0, 1}
X_3: indicates whether the lawn sprinkler is on, has values {0, 1}
X_4: indicates whether the pavement (close to the lawn) is wet, values from {0,1}
X_5: indicates whether the pavement is slippery, values from {0,1}

There are certain causal influences between some of these random variables. For instance, the season co-determines the probabilities for rain; the sprinkler state co-determines whether the pavement is wet (but one would not say that the wetness of the pavement has a causal influence on the sprinkler state), etc. Such influences can conveniently be expressed by arranging the X_i in a directed, acyclic (! – why?) graph (a DAG), such that each random variable becomes a node, with a link (i, j) denoting that the fact measured by X_i has some causal influence on the fact measured by X_j. Of course there will be some subjective judgement involved in claiming a causal influence between two observables, and denying it for other pairs – such dependency graphs are not objectively "true", they are *designed* to *represent* one's view of a part of the world. Here is the DAG for our example which serves as standard example in many introductions to Bayesian nets:



[23] In this introduction I lean mostly on two BN tutorial texts: **a.** Judea Pearl, Stuart Russell. Bayesian Networks, in M. A. Arbib (Ed.), *Handbook of Brain Theory and Neural Networks*, pp. 157–160, Cambridge, MA: MIT Press, 2003. Online version at http://www.kddresearch.org/Resources/Papers/Intro/pearl-bbn2000.pdf , mirrored at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/1921_PearlRussell03.pdf **b.** K. P. Murphy (2001), An Introduction to Graphical Models. Obtainable at http://www.cs.ubc.ca/~murphyk/Bayes/bayes.html, mirrored at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/2078_Murphy98.pdf

Fig. 6.1: A simple Bayesian network[24].
The intuitive interpretation of arrows in a Bayesian network as "causal influence" is mathematically captured by the following formal, semantic constraint that a DAG must satisfy in order to qualify as a Bayesian network:

Definition 6.1: A directed acyclic graph with nodes $\mathbf{X} = \{X_1, ..., X_n\}$ (random variables, which may take values in different measure spaces) is a *Bayesian network* (BN) for the joint distribution $P(\mathbf{X})$ if every X_i is conditionally independent of its non-descendants in the graph given its parents.

Note that the independence relations expressed in a BN in terms of parents and non-descendants need not be the only independence relations that are actually true in the joint distribution. In our example, for instance, it may be the case that the pavement is *always* slippery (because it was made from polished marble...). Then X_5 would be unconditionally independent from any of the other variables. The complete independence relations between the variables figuring in a BN depend on the particulars of the joint distribution and need not all be represented in the graph structure. A BN is only a partial model of the independence relations that may be present in the concerned variables.

For convenience of notation we often identify the nodes X_i of a BN with their indices *i.* The *descendants* of a node *i* in a DAG *G* are all nodes *j* that can be reached from *i* on a forward path in *G.* For example, in our example network the nodes 4 and 5 are the descendants of node 2. The *parents* of a node *i* in *G* are all the direct graph predecessors of *i*. For instance, the parents of node 4 are the nodes 2 and 3; the node 1 has no parents. In our example, the condition of Def. 6.1 would imply that X_4 is independent of X_1 given X_2 and X_3:

(6.1) $P(X_4 \mid X_1, X_2, X_3) = P(X_4 \mid X_2, X_3)$

In words, if we know whether the sprinkler is on and whether it rains, then we don't gain any additional information about the wetness of the lawn if we additionally know the season.

Let's squeeze in a short note on notation here. By $P(X_4 \mid X_2, X_3)$ we denote the conditional *distribution* of X_4 given X_2 and X_3. For discrete random variables, as in our example, this can be specified by a table:

(6.2)

X_2	X_3	$P(X_4 = 0)$	$P(X_4 = 1)$
0	0	1.0	0.0
0	1	0.1	0.9
1	0	0.1	0.9
1	1	0.01	0.99

For continuous-valued random variables, such conditional distributions cannot in general be specified in a closed form (one would have to specify pdf's for each possible combination of values of the conditioning variables), except in certain special cases, notably Gaussian distributions.

[24] Taken from the Pearl/Russell tutorial.

In contrast, we use lowercase $P(x_4 \mid x_2, x_3)$ as a shorthand for $P(X_4 = x_4 \mid X_2 = x_2, X_3 = x_3)$. This denotes a single probability *number* for particular values x_4, x_2, x_3 of our random variables. – End of the note on notation.

A Bayesian net can be used for reasoning about uncertain causes and consequences in many ways. Here are three kinds of arguments that are frequently made, and for which BNs offer algorithmic support:

- **Prediction.** "*If the sprinkler is on, the pavement is wet with a probability* $P(X_4 = 1 \mid X_3 = 1)$": reasoning from causes to effects, along the arrows of the BN.
- **Abduction.** "*If the pavement is wet, it is more probable that the season is spring than that it is summer, by a factor of y percent*": reasoning from effects to causes, that is, *diagnostic reasoning*, backwards along the network links. [By the way, for turning the reasoning direction around you need Bayes formula, which is what gave Bayesian networks their name].
- **Explaining away, and the rest.** "*If the pavement is wet and we don't know whether the sprinkler is on, and then observe that it is raining, the probability of the sprinkler being on, too, drops low*": reasoning "sideways", of which there are many variants. The common treat in "explaining away" phenomena is that if there are several possible causes $C_1, \ldots, C_k$ for some observed effect E, and we learn that actually cause C_i holds true, then the probabilities drop that the other causes are likewise true in the current situation.

Bayesian networks offer *inference algorithms* to carry out such arguments and compute the correct probabilities, ratios of probabilities, etc. These inference algorithms are not trivial at all, and Bayesian networks have only begun to make their way into applications since efficient inference algorithms had been discovered in the mid-80'ies. We will learn about the historical first of these algorithms in this course; it is still one of the most widely used such algorithms.

To get a first inkling of the complexity of inference in BNs, imagine that the Space Shuttle is being modelled as a BN (which indeed it is), and during the launch countdown a sensor reading *A* wanders towards its red line, [with thousands of sensors, one or two are almost bound to behave like that, typically just because of sensor failure] while another sensor *B* that monitors the same device as *A* with a different measureable shows only a slight tendency above normal. Now try to put yourself into the position of the chief launch responsible engineer, – imagine how *reeeally* eager this person would be to learn about "by which factor is the probability of a launch failure increased if *A* goes toward red line while *B* practically shows no effect" – and to learn about this within a few seconds?? This is a (real) example of a BN used for *decision support* systems. The decision support algorithms must be able to combine the interactions of thousands of random variables quickly and with extreme flexibility. Decision support is always needed when complex systems have to be mastered in moments of crisis – power grids, nuclear power plants, space shuttles are examples; tactical warfare decision-making is another (of which we learn little because military research is rarely documented to the public). BN algorithms and fast computers have become indispensible for decision support because humans are simply quite bad at correctly combining pieces of evidence in uncertain systems. They tend to make gross, systematic prediction errors even in simple diagnostic tasks involving only two observables (which is why in good universities future doctors must take courses in diagnostic reasoning), let alone be capable of dealing with complex systems involving hundreds of observables – and acting under time pressure.

BNs of this flavour – representing complex situations from everyday life or technology – have been systematically explored in Artificial Intelligence since the early 80'ies. The most prominent BN researcher in this field was Judea Pearl – just to drop the name. He developed the first computationally feasible inference algorithms and explored many variants of the basic BN architecture we have sketched. The main types of applications of such BNs are

- **expert systems:** representations of complex pieces of reality, typically for diagnosis, for instance medical diagnosis or car fault diagnosis;
- **decision support systems**[25]**:** helping humans to assess the consequences of actions and choose optimal actions (like in the space shuttle example, but also for economical or political and military decision making);
- **user modelling** for intelligent computer interfaces and Web applications – the (in)famous paperclip guy plaguing Microsoft Word is actually controlled by a BN that learns to represent and predict a user's weaknesses and preferences[26].

In a sense, the most natural (the only?) thing you can do when it comes to handle the interaction between many random variables, is to arrange them in a graph. So it is no surprise that similar formalisms have been invented independently in several fields, comprising besides Artificial Intelligence physics, genetics, statistics, and image processing. However, the most important developments have been seen in Artificial Intelligence and Machine Learning, where feasible inference algorithms for large-scale BNs have first been investigated.

Methods of Bayesian networks have been generalized and ramified in many ways. A good overview is given in the Pearl/Russell tutorial. The generalizations comprise the following.

- Use **undirected graphs** (we'll visit them in this lecture) instead of directed ones, with a different graph-based semantics of statistical independance. Such undirected BNs were actually used in physics before (e.g., the *Ising model* can be seen as an undirected BN), or in image processing (*random Markov field* models of images), or in neural computation (the *Boltzmann machine*).
- Introduce **dynamics**, that is, let BNs evolve over time. This is done by introducing a structural copy of a basic BN for each (discrete) time point, and connecting these copies by temporal arrows that are themselves equipped with a statistical (in)dependence relation. Such models were used since long in speech processing under the name of *hidden Markov models* (HMMs) and in robot control under the name of *partially observable Markov decision processes* (POMDPs).
- Introduce **hidden and "meta" nodes** into BNs, which correspond to unobserved quantities and to model-switching control quantities, respectively. The general theory

[25] See the intro slides from a BN course of Kathryn Laskey, originally retrieved from http://ite.gmu.edu/~klaskey/CompProb/CompProb_Intro.pdf, now at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/KlaskeySlides.zip. She has a professional background in decision support and motivates BNs from that perspective.

[26] See a survey paper of one of Microsoft's head engineers about perspectives of Bayesian user modelling for the future of human computer interaction, E. Horvitz, C. M. Kadie, T. Paek, D. Hovel. Models of Attention in Computing and Communications: From Principles to Applications, *Communications of the ACM* 46(3):52-59, March 2003 http://research.microsoft.com/~horvitz/cacm-attention.pdf, mirrored at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/Howrwitz.cacm-attention.pdf. A popular description of Microsoft's plans to revolutionize user interfaces with Bayesian Network methods can be found in a NY Times article http://partners.nytimes.com/library/tech/00/07/biztech/articles/17lab.html, mirrored at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/NYTimesHorvitzBayesMicrosoft.pdf

of such generalized BNs comprises all the models mentioned so far and some more (notably, the classical statistical techniques of *principal component analysis* (PCA) and *independent component analysis,* ICA) and has been promoted by the neural computation / machine learning superpower[27] Michael I. Jordan in recent years under the name of *graphical models.*

In the algorithmic and mathematical world of BNs there exist two fundamental tasks:

- **Inference:** find algorithms that exploit the graph structure as ingeneously as possible to yield *fast* algorithms for computing all the quantities asked for in prediction, abduction and "sideways reasoning" tasks. This is non-trivial because the exact inference task is NP-hard. The breakthrough of BNs as a practical tool came about when the first feasible inference algorithms were found by Judea Pearl in the early 80'ies. They have been much improved since, and we will learn about one of the modern versions of the core inference algorithm in this lecture.
- **Learning:** find algorithms to construct a BN from observed empirical data (inference algorithms use a BN that is already in place!). All learning algorithms need inference algorithms as a subroutine, so we will naturally first treat the latter. Depending on the available time, we will present a simple learning algorithm (case of all nodes corresponding to observables) or a more complicated, more powerful one (case when there are also "hidden" nodes).

All in all, you are in for an intense experience of advanced, applied algorithmics, beautiful and powerful (and not easy – can't be denied), where the *combination of graph-manipulation algorithms with statistical inference methods* is the key to success.

6.2 Brute-force inference in BNs

A Bayesian network is a probabilistic representation of a piece of reality – that piece which we observe through the random variables that appear in the BN. We have argued earlier in this lecture (Section 2, Eqn. (2.10) ff.) that the joint distribution of all observables of interest is the most comprehensive model of a piece of reality that one may obtain. In fact, any other kind of formal model is "just" some partial view of that most comprehensive model. Now, if one models a fairly complex chunk of reality, one will need many observables, that is, many random variables. One of the saddest facts about life, mathematics, engineering and everything else is that the joint distribution of many random variables is an exponentially complex object, inaccessible in general to an analytical or algorithmical treatment. For instance, the joint distribution of our mere five, very simple random variables would be a histogram over the set

(6.3) $\{\text{Winter, Spring, Summer, Fall}\} \times \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}$,

which has 4 * 2 * 2 * 2 * 2 = 64 elements, so you would need 63 parameters to specify it (one is redundant because all parameters must sum to 1). It is clear that, for real-life modelling tasks involving hundreds or even thousands of random variables, this number explodes. The power of BNs results from the fact that in the presence of independence relations, the joint distribution can be represented, computed, or otherwise manipulated in a much more

[27] Take a look at http://www.cs.berkeley.edu/~jordan/ -- the list of his former students is an awe-inspiring Who is Who of today's machine learning community!

economical way, because only the *local* dependencies between random variables linked by arrows in the BN have to be taken into account. We will investigate brute-force methods for elementary inference in this subsection and see how the graph structure leads to a dramatic reduction in computational complexity (which however still is much too high for practical applications – more refined algorithms will be presented in subsequent subsections).

By the factorization formula (2.15), the joint distribution of our five random variables is

$$(6.4)\qquad \begin{aligned} P(\bigotimes_{i\in\{1,\ldots,5\}} X_i) &= P(\mathbf{X}) = \\ &= P(X_1)\,P(X_2 \mid X_1)\,P(X_3 \mid X_1, X_2)\,P(X_4 \mid X_1, X_2, X_3)\,P(X_5 \mid X_1, X_2, X_3, X_4). \end{aligned}$$

Exploiting the conditional independencies expressed in the BN graph, this reduces to

$$(6.5)\qquad P(\mathbf{X}) = P(X_1)\,P(X_2 \mid X_1)\,P(X_3 \mid X_1)\,P(X_4 \mid X_2, X_3)\,P(X_5 \mid X_4).$$

For representing the factors on the r.h.s. by tables like (6.2), one would need tables of sizes 1×4, 4×2, 4×2, 4×2, and 2×2, respectively. Because the entries per row in each of these tables must sum to 1, one entry per row is redundant, so these tables are specified by 3, 4, 4, 4 and 2 parameters, respectively. All in all, this makes 17 parameters needed to specify $P(\mathbf{X})$, as opposed to the 63 parameters needed for the general product distribution of $X_1, \ldots, X_5$. Generally speaking, the number of parameters required to specify the joint distribution of n discrete random variables with maximally v values each arranged in a BN with a maximum fan-in of k is $O(n\,v^k)$ as opposed to the "raw" number of parameters $O(v^n)$ needed for a naive characterization of the joint distribution. This is a reduction from a space complexity that is exponential in n, to a space complexity that is linear in n! This simple fact has motivated many a researcher to devote his/her life to Bayesian networks (and we can guess that the fan-in in BNs will have an important impact on representational efficiency gains).

Any reasoning on BNs (predictive, abductive, "sidestepping" or others) boils down to evaluate some conditional probability expressions. For instance, the abductive question, "*If the pavement is wet, by which factor y is it more probable that the season is spring than that it is summer*", asks for

$$(6.6)\qquad y = \frac{P(X_1 = spring \mid X_4 = 1)}{P(X_1 = summer \mid X_4 = 1)}.$$

Such ratios of probabilities are very often sought in diagnostic reasoning (consider the questions, "by which factor is it more probable that my symptoms are due to cancer, than that they are due to a harmless cause").

Any conditional probability $P(y_1, \ldots, y_m \mid z_1, \ldots, z_l)$, where the Y_i and Z_j are among the random variables represented in the BN, can be computed from the joint distribution (6.5) of *all* variables in the BN by first transforming $P(y_1, \ldots, y_m \mid z_1, \ldots, z_l)$ into a fraction of two marginal probabilities,

$$(6.7)\qquad P(y_1,..,y_m \mid z_1,..,z_l) = \frac{P(y_1,..,y_m, z_1,..,z_l)}{P(z_1,..,z_l)}$$

and then computing the denominator and the enumerator by marginalization (see Eqns. (2.11) and (2.12)), exploiting the efficient BN representation of the kind exemplified in Eqn. (6.5).

The probability $P(X_1 = spring \mid X_4 = 1)$, for instance, can be computed by

(6.8)

$$
\begin{aligned}
&P(X_1 = spring \mid X_4 = 1) = \\
&\quad = \frac{P(X_1 = spring, X_4 = 1)}{P(X_4 = 1)} \\
&\quad = \frac{\sum_{x_2, x_3, x_5} P(X_1 = spring, x_2, x_3, X_4 = 1, x_5)}{\sum_{x_1, x_2, x_3, x_5} P(x_1, x_2, x_3, X_4 = 1, x_5)} \\
&\quad = \frac{\sum_{x_2, x_3, x_5} P(X_1 = spring) P(x_2 \mid X_1 = spring) P(x_3 \mid X_1 = spring) P(X_4 = 1 \mid x_2, x_3) P(x_5 \mid X_4 = 1)}{\sum_{x_1, x_2, x_3, x_5} P(x_1) P(x_2 \mid x_1) P(x_3 \mid x_1) P(X_4 = 1 \mid x_2, x_3) P(x_5 \mid X_4 = 1)}.
\end{aligned}
$$

This could be done by a brute-force evaluation of the concerned summations. The sum to be taken in the denominator would run over 4×2×2×2 = 32 terms. It is apparent that this approach generally incurs a number of multiplications that is exponential in the size of the BN.

In fact, the task of computing exact inferences in a BN is NP-hard. This can be seen as follows, by reducing the satisfiability problem for Boolean logic[28] to computing inferences in a BN. Let ϕ be a Boolean expression with the Boolean variables $a_1, ..., a_n$. We design a BN with binary random variables X_i where for some node X_K it holds that $P(X_K = 1) > 0$ iff ϕ is satisfiable. First observe that ϕ is constructed from the elementary Boolean expressions $a_1, ..., a_n$ by iterated applications of $\wedge$ and $\neg$, yielding a directed, acyclic graph G of subexpressions (known as a *Boolean circuit*) with initial nodes $a_1, ..., a_n$ and a common maximal node corresponding to ϕ. Label each node with a binary random variable X_i, where the n parent-less nodes corresponding to the elementary expressions $a_1, ..., a_n$ are labelled with $X_1, ..., X_n$, and the unique terminal node corresponding to ϕ by X_K. For each starting node $X_1, ..., X_n$, declare an unconditional probability distribution $P(X_i = 1) = P(X_i = 0) = 1/2$. For each other node X_i, if it corresponds to a subformula $\phi_i = \phi_m \wedge \phi_k$, specify its conditional distribution by $P(X_i = 1 \mid X_m = x_m, X_k = x_k) = 1$ if $x_m = 1$ and $x_k = 1$, $P(X_i = 1 \mid X_m = x_m, X_k = x_k) = 0$, else. If X_i corresponds to a subformula $\phi_i = \neg \phi_m$, set $P(X_i = 1 \mid X_m = 0) = 1$ and $P(X_i = 1 \mid X_m = 1) = 0$. Then prove by induction over the structure of ϕ that for every node X_i, $P(X_i = 1)$ is the fraction of the possible 2^n truth assignments to $a_1, ..., a_n$ which satisfy the Boolean expression corresponding to that node. Hence, $P(X_K = 1) > 0$ iff ϕ is satisfiable.

This argument shows that computing marginals in BNs is in general intractable. In this situation, computer scientists can essentially choose between three options:

[28] See chapters 11 and 12 in my lecture notes on Computability and Complexity for an introduction to NP-hardness, NP-completeness, and the satisfiability problem of Boolean expressions (online at http://minds.jacobs-university.de/sites/default/files/uploads/teaching/lectureNotes/scriptCC.pdf)

1. *Restrict the problem by suitable constraints, earning a tractable problem.* For BNs, this was the first strategy that was used with success: early (now classical) inference algorithms were defined (and were tractable) only for BNs with *tree* graphs.
2. *Use randomized algorithms, i.e. algorithms that invoke random decisions at some point.* Such "guessing" algorithms need not always terminate; if they do, their result is perfectly accurate. The goal is to create randomized algorithms which, if they terminate, they do so quickly, thus being fast almost all of the time and failing (running too long so they need to be abandoned) only rarely. In practice, algorithms that involve some human intervention in the form of "heuristics" can be seen as randomized algorithms. The BN inference "join tree" algorithm which we will study in this lecture contains heuristic elements.
3. *Use approximate algorithms, i.e. algorithms that yield results always and fast, but with an error margin (which should be controllable).* For BN inference, a much-used class of approximate algorithms is based on *sampling*. In order to obtain an estimate of some marginal probability, one samples from the distribution defined by the BN and uses the sample as a basis to estimate the desired marginal. There is an obvious tradeoff between runtime and precision.

There are numerous toolboxes, commercial and free, for building and using BNs. In the practical exercises / miniprojects of this course we use the Matlab-based toolbox written by Kevin Murphy, available at https://code.google.com/p/bnt/.

The fact that BN inference is NP-hard should however not generally discourage you. In practice one can often exploit the structure of the BN to speed up computations in a simple way, without resorting to advanced randomized or approximate algorithms. One evident method works by pulling the sum into the product as far as possible, and evaluate the resulting formula from the inside of bracketing levels. This is called the method of *variable elimination*. An equivalent formula for the sum in the denominator of (6.8), for example, would be

$$(6.9) \qquad \sum_{x_1}\left(P(x_1) \sum_{x_2,x_3}\left(P(x_2 \mid x_1)\, P(x_3 \mid x_1)\, P(X_4 = 1 \mid x_2, x_3) \left(\sum_{x_5} P(x_5 \mid X_4 = 1) \right) \right) \right) .$$

This would need only $2+2\times2+0 = 6$ summations (the sum over x_5 is 1 and need not be explicitly calculated!). However, finding a summation order where this pulling-in leads to the minimal number of summations is again NP-hard, although greedy algorithms for that purpose are claimed to work well in practice.

We will not further pursue the variable elimination method however, but instead we will present a more powerful, more widely used, but also significantly more complex algorithm in the next subsections. To use this algorithm, the BN is first transformed from its *directed* graph representation into another, *undirected* graph representation. Because such undirected graph representations of (in)dependency relationships between RVs are of great interest in their own right, we will devote an entire subsection to them, before we proceed with the join tree inference algorithm for BNs.

6.3 Undirected graphical models

The power of BNs is that (in)dependency relationships between RVs are captured by suitable graph structures, which in turn guide efficient inference algorithms. So far, we have seen how *directed* graphs can be used to this end. But it is also possible to use *undirected* graphs. This leads to *undirected graphical models* (UGMs), which have a markedly different "flavour" than the directed BNs. UGMs originated in statistical physics and image processing, while BNs were first explored in Artificial Intelligence. Depending on the field, UGMs are called by different names[29]:

- **Markov random fields**: the word used in statistics and image processing,
- **Ising models, Boltzmann machines**: widely investigated in statistical physics and machine learning, -- we have already met them but have so far not discussed aspects of independence of RVs and efficient inference algorithms,
- **undirected graphical models**, or **undirected probabilistic independence networks**: terms used in the BN community proper.

We have seen in the preceding subsection that statistical independence can be exploited for efficient inference computations. We will use the following handy notation for statistical, conditional independence between two sets **Y** and **Z** of random variables, given a set **S** of random variables:

Definition 6.2: two sets **Y**, **Z** of random variables are *independent given* **S**, written $\mathbf{Y} \perp \mathbf{Z} \mid \mathbf{S}$, if $P(\mathbf{Y},\mathbf{Z} \mid \mathbf{S}) = P(\mathbf{Y} \mid \mathbf{S})P(\mathbf{Z} \mid \mathbf{S})$.

Remark. It is easy to see that if $\mathbf{Y} \perp \mathbf{Z} \mid \mathbf{S}$, then for any subsets $\mathbf{Y}' \subseteq \mathbf{Y}, \mathbf{Z}' \subseteq \mathbf{Z}$ it holds that $P(\mathbf{Y}', \mathbf{Z}' \mid \mathbf{S}) = P(\mathbf{Y}' \mid \mathbf{S})P(\mathbf{Z}' \mid \mathbf{S})$.

In an UGM, independence relations between sets of random variables are defined in terms of a graph separation property:

Definition 6.3: Let $G = (V, R)$ be an undirected graph with nodes V and links $R \subseteq V \times V$. Let **Y**, **Z**, **S** $\subseteq V$ be disjoint, nonempty subsets. Then **Y** is *separated* from **Z** by **S** if every path from some node in **Y** to any node in **Z** contains a node from **S**. **S** is called a *separator* for **Y** and **Z.**

Here is the definition of undirected graph models, which declares how statistical independence is reflected in the graph's separation properties:

Definition 6.4: An undirected graph with nodes $\mathbf{X} = \{X_1, ..., X_n\}$ is an *undirected graphical model* (UGM) for $P(\mathbf{X})$ if for all **Y**, **Z**, **S** $\subseteq \mathbf{X}$ it holds that if **Y** is separated from **Z** by **S**, then $\mathbf{Y} \perp \mathbf{Z} \mid \mathbf{S}$.

[29] Overview with references in Smyth, P., Heckerman, D., Jordan, M.I.: Probabilistic Independence Networks for Hidden Markov Probability Models. Neural Computation 9(2), 1997, 227-270 Online at http://www.faculty.jacobs-university.de/hjaeger/SharedMaterials/1403_Smythetal97.pdf. A highly readable non-technical overview and comparison of directed and undirected models is given in Smyth, P.: Belief networks, hidden Markov models, and Markov random fields: a unifying view. Pattern Recognition Letters 18 (11-13), 1997, 1261 – 1268. Copy in http://www.faculty.jacobs-university.de/hjaeger/SharedMaterials/2117_Smyth97.pdf

Like with BNs, an UGM need not reflect all the independence relations that govern the joint distribution. An UGM is called a *perfect* model for $P(\mathbf{X})$ if all independence relations of $P(\mathbf{X})$ are reflected in separation properties. A given $P(\mathbf{X})$ need not possess a perfect UGM. Furthermore, an UGM is a *minimal* model for $P(\mathbf{X})$ if the removal of any link would create an independance relation (by separation) not present in $P(\mathbf{X})$. A given $P(\mathbf{X})$ may possess different minimal UGMs. Note finally that a fully connected undirected graph with nodes $\mathbf{X} = \{X_1, ..., X_n\}$ is trivially an UGM for any $P(\mathbf{X})$ because no separators exist.

UGMs are sometimes defined in a slightly different but equivalent way. We formulate this other definition as a proposition:

Proposition 6.1 An undirected graph with nodes $\mathbf{X} = \{X_1, ..., X_n\}$ is an UGM for $P(\mathbf{X})$ iff for all $i = 1, \ldots, n$ it holds that $P(X_i \mid (X_j)_{j \neq i}) = P(X_i \mid (X_j)_{j \in N_i})$, where $N_i \subseteq \{X_1, ..., X_n\}$ is the set of all direct neighbors of X_i in the graph.

In plain English, X_i is independent of all other variables in the system given all graph-neighbor variables of X_i. Proving this proposition is a straightforward exercise.

The characterization of UGMs through the direct graph neighbors reveals how UGMs can be seen as a spatial generalization of Markov processes. Recall that a stochastic process $X_1, X_2, ...$ is defined to possess the Markov property if X_i is independent of all earlier $X_1, ..., X_{i-2}$ given X_{i-1}. The variable X_{i-1} is the direct neighbor of X_i in the "timeline graph" that connects each X_i to X_{i+1}. In a similar way, the spatial direct neighbors N_i of X_i make X_i independent from the other nodes in the graph. This is the reason why UGMs are often called *Markov Random Fields* (MRF) – this is the standard terminology in statistical physics and signal engineering applications, especially in image processing.

We will restrict the remainder of this subsection to MRFs where the participating random variables $\{X_1, ..., X_n\}$ are all nominal, that is, each X_i takes values in a finite set which for simplicity we assume to be $E_i = \{1, 2, \ldots, m_i\}$.

For reasons that will become clear later, Markov Random Fields are usually defined to have one additional property that makes them more special than generic UGMs, namely strict positivity of P. We will adhere to this convention and formally define MRFs as follows:

Definition 6.5: An UGM is a *Markov Random Field* if $P(x_1,...,x_n) > 0$ for all $(x_1,...,x_n) \in \bigotimes_{i=1}^{n} E_i$.

For a first exposure to MRFs, let us see how a 1-dimensional Ising model can be understood as an MRF. For concreteness, consider the Ising model with just 4 atoms whose magnetic spin states are described by $X_1, ..., X_4$, which take values in $\{-1,1\}$. The microstates are the 4-tuples over $\{-1,1\}$. At temperature T, the joint (Boltzmann) distribution over the microstates is given by

$$(6.10)\quad P(X_1, ..., X_4) = \frac{1}{Z}\exp(-\sum_{j=i+1} JX_iX_j / T) = \frac{1}{Z}\frac{1}{\prod_{j=i+1}\exp(JX_iX_j / T)} = \prod_{j=i+1}\frac{1}{\sqrt[3]{Z}\exp(JX_iX_j / T)}.$$

We see that the joint distribution factorizes into three factors, whose values depend on X_1, X_2 and X_2, X_3 and X_3, X_4, respectively. A valid undirected graph representation for this distribution would be obtained by connecting subsequent atoms, as in Fig. 6.2:

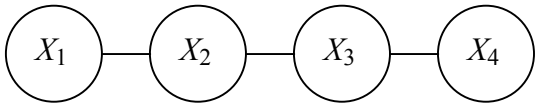


Fig. 6.2: An MRF for a 4-atom Ising model.

To see that this MRF is a model for (6.10), it suffices to take into account that (6.10) factorizes in the general format

(6.11) $P(X_1, ..., X_4) = \varphi_1(X_1, X_2)\, \varphi_2(X_2, X_3)\, \varphi_3(X_3, X_4)$,

with some functions φ_i. The specific form of these functions is not relevant, but notice that they are strictly positive. We have to check that the independence relations prescribed by the UGM are satisfied by $P(X_1, ..., X_4)$. For illustration, we just check that $X_1 \perp X_4 \,|\, \{X_2, X_3\}$:

$$
\begin{aligned}
P(x_1, x_4 \,|\, x_2, x_3) &= \frac{P(x_1, x_4, x_2, x_3)}{P(x_2, x_3)} = \frac{\varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)}{\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)} \\
&= \frac{\varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)}{\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)} \\
&= \frac{\varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)}{(\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4))^2} \\
&\overset{(*)}{=} \frac{\varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\sum_{x_4} \varphi_3(x_3, x_4)}{\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)} \frac{\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)\sum_{x_1} \varphi_1(x_1, x_2)}{\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)} \\
&= \frac{\sum_{x_4} \varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)}{\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)} \frac{\sum_{x_1} \varphi_1(x_1, x_2)\varphi_2(x_2, x_3)\varphi_3(x_3, x_4)}{\varphi_2(x_2, x_3)\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4)} \\
&= \frac{P(x_1, x_2, x_3)}{P(x_2, x_3)} \frac{P(x_4, x_2, x_3)}{P(x_2, x_3)} = P(x_1 \,|\, x_2, x_3) P(x_4 \,|\, x_2, x_3)
\end{aligned}
\qquad (6.12)
$$

where in step (*) we used $\sum_{x_1, x_4} \varphi_1(x_1, x_2)\varphi_3(x_3, x_4) = \left(\sum_{x_1} \varphi_1(x_1, x_2)\right)\left(\sum_{x_4} \varphi_3(x_3, x_4)\right)$.

Generalizing from this finding (without proof) we see that an *n*-dimensional Ising model on a rectangular grid can be represented by a UGM where the grid connections between atoms form the graph structure of the UGM.

The connection between a factorized representation of a joint distribution $P(\mathbf{X})$ as in (6.11) and the graph structure of an associated UGM can be expressed in a very general way. Recall that a *clique* in an undirected graph is a completely connected subset of nodes. A *maximal*

clique is a clique to which no node can be added without losing the clique property. Joint distributions of random variables which can be factorized similar to (6.11) are of general interest and are given a name:

Definition 6.6 Let $\mathbf{X} = \{X_1, ..., X_n\}$ be a set of RVs. A distribution $P(\mathbf{X})$ is a *Gibbs distribution* if the X_i can be are arranged as nodes in an undirected graph G whose cliques $C \subseteq \text{Pot}(\mathbf{X})$ allow to factorize $P(\mathbf{X})$ by

$$P(x_1,...,x_n) = \prod_{c \in C} \varphi_c(\mathbf{x}_c), \tag{6.13}$$

where $\mathbf{x}_c$ is the subset of the values $x_1,...,x_n$ belonging to nodes in clique c and for each clique c, φ_c is a clique function that assigns a real number to each of the possible combinations of values that the random variables occurring in that clique can take.

The Boltzmann distribution for our simple 4-atom Ising system is a Gibbs distribution, as is witnessed by (6.11).

Gibbs distributions and Markov Random Fields are closely connected – in fact, they are identical. We show that (i) every Gibbs distribution defines a MRF (easy), and (ii) conversely, every MRF is a Gibbs distribution (not so easy, famous landmark theorem)[30].

Proposition 6.2 With the notation and assumptions from Def. 6.6, a Gibbs distribution (6.13) is a Markov Random Field.

Proof. Using the characterization of Proposition 6.1, without loss of generality it suffices to show that $P(X_1 = x_1 \mid X_2 = x_2,...,X_n = x_n) = P(X_1 = x_1 \mid X_2 = x_2,...,X_k = x_k)$, where for notational simplicity we assume $N_1 = \{x_2,...,x_k\}$. Writing out the first probability, we get

$$\begin{aligned} &P(X_1 = x_1 \mid X_2 = x_2,...,X_n = x_n) \\ &\quad = \frac{P(X_1 = x_1, X_2 = x_2,...,X_n = x_n)}{P(X_2 = x_2,...,X_n = x_n)} \\ &\quad = \frac{\prod_{c \in C} \varphi_c(x_1, x_2,..., x_n \downarrow c)}{\sum_{e \in E_1} \prod_{c \in C} \varphi_c(e, x_2,..., x_n \downarrow c)}. \end{aligned}$$

The notation $x_1, x_2,..., x_n \downarrow c$ is to be understood as "all of the $x_1,...,x_n$ which lie in c". Let $i > k$, that is, x_i is neither x_1 nor any of its graph neighbors. Let c^* be a clique containing x_i. Then c^* cannot also contain x_1, because if it did, then there would be a graph link between x_1 and x_i, contradicting $i > k$. Then $\varphi_{c^*}(x_1, x_2,..., x_n \downarrow c^*) = \varphi_{c^*}(e, x_2,..., x_n \downarrow c^*)$, thus the c^* terms cancel from the ratio. In other words, the ratio (and thus, $P(X_1 = x_1 \mid X_2 = x_2,...,X_n = x_n)$) depends only on cliques that are identical to or contained in $\{x_1\} \cup N_1$. □

The reverse direction has first been proven by John M. Hammersfield and Peter Clifford in 1971. They did not publish their result because they felt that the positivity constraint on P was

[30] For both (i) and (ii) I follow (sometimes verbatim) David Pollard, a Yale statistician who has provided particularly simple proofs; my source is an online manuscript http://www.stat.yale.edu/~pollard/Courses/251.spring04/Handouts/Hammersley-Clifford.pdf . A copy is also at http://www.faculty.jacobs-university.de/hjaeger/SharedMaterials/Hammersley-Clifford.pdf .

unnatural, and wanted to relax it. Thus it happened that their result was proven independently, and published, by Julian Besag in 1972[31]; in 1974 it was shown[32] that the positivity condition is indeed *necessary*. The theorem is nonetheless known today as the *Hammersley-Clifford theorem*.

Proposition 6.3 (Hammersley-Clifford) Let C be the set of cliques of a MRF for $P(\underset{i=1,\ldots,n}{\otimes} X_i)$, where the X_i are nominal. Then $P(\underset{i=1,\ldots,n}{\otimes} X_i)$ can be factorized by

$$P(x_1,\ldots,x_n) = \prod_{c\in C} \varphi_c(\mathbf{x}_c), \tag{6.14}$$

where $\mathbf{x}_c$ is the subset of the values $x_1,\ldots,x_n$ belonging to nodes in clique c and for each clique c, φ_c is a clique function that assigns a positive real number to each of the possible combinations of values that the random variables occurring in that clique can take.

We do not present a proof here. A relatively simple proof, provided by Dave Pollard, is online at http://www.stat.yale.edu/~pollard/251.spring04/Handouts/Hammersley-Clifford.pdf (a copy is also at http://www.faculty.jacobs-university.de/hjaeger/SharedMaterials/Hammersley-Clifford.pdf).

Note. Because the product of clique functions of subcliques of a maximal clique is a function of the maximal clique, one can restrict the formulation of Definition 6.6 and of the Hammersley-Clifford theorem to maximal cliques. We will denote the set of all maximal cliques in an undirected graph by $\overline{C}$.

In our little Ising example, the maximal cliques in the UGM depicted in Fig. 6.2 are just the sets {1, 2}, {2, 3} and {3, 4}, and the clique function of the clique $\{i, i+1\}$ is $\frac{1}{\sqrt[3]{Z}\exp(JX_iX_j/T)}$.

Due to the strict positivity of the clique functions φ_c, one can write them as exponential functions:

$$\begin{aligned} P(x_1,\ldots,x_n) &= \prod_{c\in\overline{C}} \varphi_c(\mathbf{x}_c) = \prod_{c\in\overline{C}} \exp(-(-\log(\varphi_c(\mathbf{x}_c)))) \\ &=: \prod_{c\in\overline{C}} \exp(-E_c(\mathbf{x}_c)) \\ &= \exp(-\sum_{c\in\overline{C}} E_c(\mathbf{x}_c)) \end{aligned} \tag{6.15}$$

Here we introduced *energy* functions $E_c(\mathbf{x}_c) = -\log(\varphi_c(\mathbf{x}_c))$ on the maximal cliques, thereby establishing a connection between the formalisms of Gibbs distributions and Boltzmann distributions. In a Gibbs distribution, the (Boltzmann) energy (at a unit temperature) is the sum of all maximal clique energies. Due to this connection to statistical physics, the clique functions φ_c are often called *clique potentials* or *(local) potential functions.*

[31] Julian Besag (1972): Nearest-neighbour Systems and the Auto-logistic Model for Binary data. Journal of the Royal Statistical Society, Series B, 34, pp. 75–83.
[32] John Moussouris (1974): Gibbs and Markov Random Systems with Constraints. Journal of Statistical Physics, 10, pp. 11-33

Summing up, we find that UGMs / Gibbs distributions / MRFs admit a *local* characterization of the *global* joint distribution through the clique potentials. The largest sets of RVs that have to be modelled together are the clique sets. Similarly, in BNs based on DAGs, the global joint distribution can be characterized locally through the conditional probability tables that we may store with each node. An important difference between directed BNs and undirected MRFs is that the conditional probability tables of BNs are *interpretable* in terms of probability, whereas potential functions are *arbitrary functions* which generally have no direct interpretation in terms of probabilities.

A major field of applications of MRFs is image processing (denoising, reconstruction). I wrap up this subsection with a simple tutorial example from this field. In the remainder of this subsection I quote almost verbatim from Section 8.3.3. of "the new Bishop book"[33], adapting the notation and adding a few explanations.

(Quasi-quote begins)

We can illustrate the application of undirected graphs using an example of noise removal from a binary image. Although a very simple example, this is typical of more sophisticated applications. Let the observed noisy image be described by an array of binary pixel values $y_i \in \{-1, +1\}$, where the index $i = 1, \ldots, n$ runs over all pixels. We shall suppose that the image is obtained by taking an unknown noise-free image, described by binary pixel values $x_i \in \{-1, +1\}$ and randomly flipping the sign of pixels with some small probability. An example binary image, together with a noise corrupted image obtained by flipping the sign of the pixels with probability 10%, is shown in Figure 6.3. Given the noisy image, our goal is to recover the original noise-free image.

[33] Christoper M. Bishop, Pattern Recognition and Machine Learning. Springer Verlag 2006

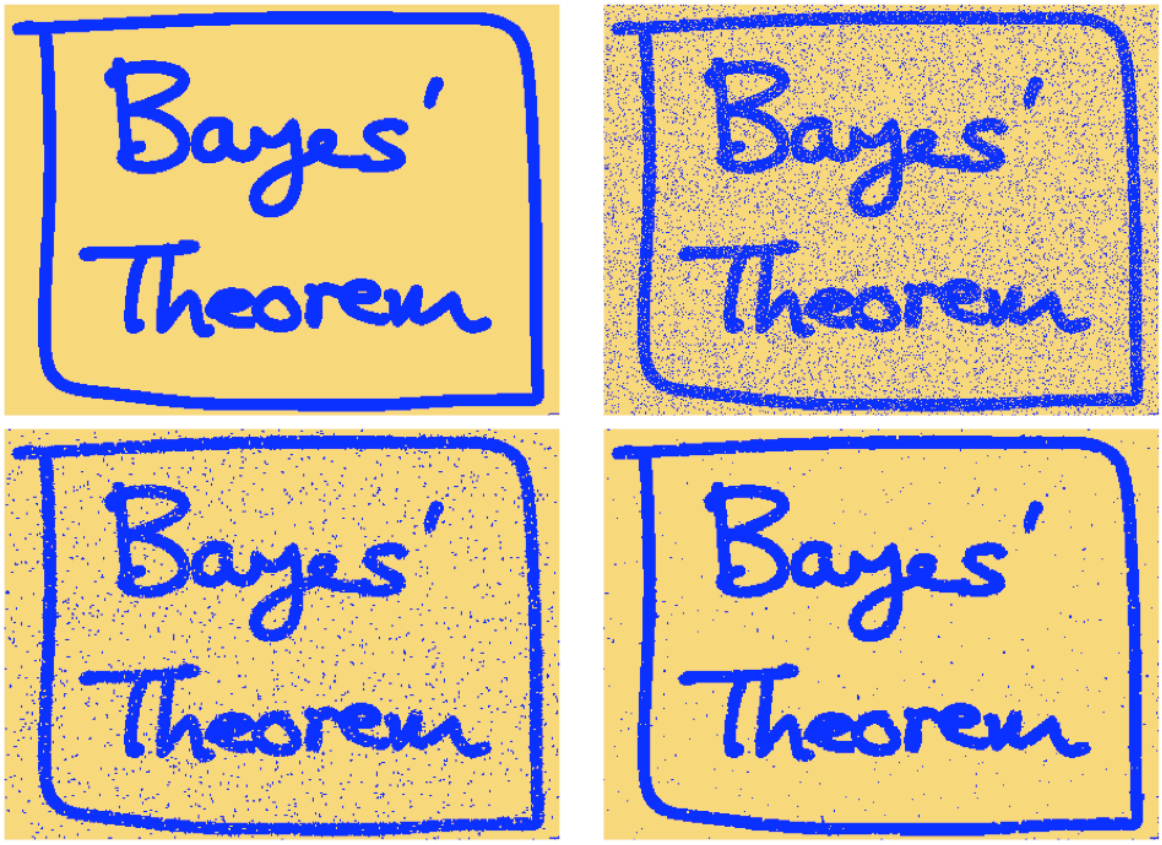


Figure 6.3 Illustration of image de-noising using a Markov random field. The top row shows the original binary image on the left and the corrupted image after randomly changing 10% of the pixels on the right. The bottom row shows the restored images obtained using iterated conditional models (ICM) on the left and using the graph-cut algorithm on the right. ICM produces an image where 96% of the pixels agree with the original image, whereas the corresponding number for graph-cut is 99%.

Because the noise level is small, we know that there will be a strong correlation between x_i and y_i. We also know that neighbouring pixels x_i and x_j in an image are strongly correlated. This prior knowledge can be captured using the Markov random field model whose undirected graph is shown in Figure 6.4. This graph has two types of cliques, each of which contains two variables. The cliques of the form $\{x_i, y_i\}$ have an associated energy function that expresses the correlation between these variables. We choose a very simple energy function for these cliques of the form $-\eta\, x_i\, y_i$ where η is a positive constant. This has the desired effect of giving a lower energy (thus encouraging a higher probability) when x_i and y_i have the same sign and a higher energy when they have the opposite sign.

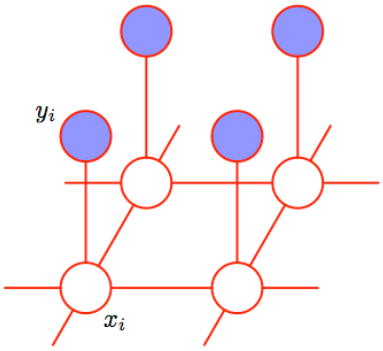


Figure 6.4 An undirected graphical model representing a Markov random field for image de-noising, in which x_i is a binary variable denoting the state of pixel i in the unknown noise-free image, and y_i denotes the corresponding value of pixel i in the observed noisy image.

The remaining cliques comprise pairs of variables $\{x_i, x_j\}$ where i and j are indices of neighbouring pixels. Again, we want the energy to be lower when the pixels have the same sign than when they have the opposite sign, and so we choose an energy given by $-\beta\, x_i\, x_j$ where β is a positive constant.

Because a potential function is an arbitrary, nonnegative function over a maximal clique, we can multiply it by any nonnegative functions of subsets of the clique, or equivalently we can add the corresponding energies. In this example, this allows us to add an extra term $h\, x_i$ for each pixel i in the noise-free image. Such a term has the effect of biasing the model towards pixel values that have one particular sign in preference to the other.

The complete energy function for the model then takes the form

$$E(\mathbf{x},\mathbf{y}) = h\sum_i x_i - \beta\sum_{i,j} x_i x_j - \eta\sum_i x_i y_i \tag{6.16}$$

which defines a joint distribution over x and y given by

$$P(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\exp(-E(\mathbf{x},\mathbf{y})). \tag{6.17}$$

We now fix the elements of **y** to the observed values given by the pixels of the noisy image, which implicitly defines a conditional distribution P(**x** | **y**) over noise-free images. This is an example of the Ising model. For the purposes of image restoration, we wish to find an image **x** having a high probability (ideally the maximum probability). To do this we shall use a simple iterative technique called *iterated conditional modes*, or *ICM*, which is simply an application of coordinate-wise gradient ascent. The idea is first to initialize the variables $\{x_i\}$, which we do by simply setting $x_i = y_i$ for all i. Then we take one node x_j at a time and we evaluate the total energy for the two possible states $x_j = +1$ and $x_j = -1$, keeping all other node variables fixed, and set x_j to whichever state has the lower energy. This will either leave the probability unchanged, if x_j is unchanged, or will increase it. Because only one variable is changed, this is a simple local computation that can be performed efficiently. We then repeat the update for

another site, and so on, until some suitable stopping criterion is satisfied. The nodes may be updated in a systematic way, for instance by repeatedly raster scanning through the image, or by choosing nodes at random.

If we have a sequence of updates in which every site is visited at least once, and in which no changes to the variables are made, then by definition the algorithm will have converged to a local maximum of the probability. This need not, however, correspond to the global maximum.

For the purposes of this simple illustration, we have fixed the parameters to be $\beta = 1.0$, $\eta = 2.1$ and $h = 0$. Note that leaving $h = 0$ simply means that the prior probabilities of the two states of x_i are equal. Starting with the observed noisy image as the initial configuration, we run ICM until convergence, leading to the de-noised image shown in the lower left panel of Figure 6.3. Note that if we set $\beta = 0$, which effectively removes the links between neighbouring pixels, then the global most probable solution is given by $x_i = y_i$ for all i, corresponding to the observed noisy image.

For certain classes of model, including the one given by (6.16), there exist efficient algorithms based on *graph cuts* that are guaranteed to find the global maximum[34]. The lower right panel of Figure 6.3 shows the result of applying a graph-cut algorithm to the de-noising problem.

(*Here the quasi-quote from the Bishop book ends.*)

6.4 The first steps of the join tree inference algorithm for BNs

After this excursion into the land of undirected graphical models, we resume our treatment of BNs based on directed acyclical graphs. We will develop the classical inference algorithm that can compute conditional and absolute probabilities on BNs. This algorithm, often referred to as *join tree algorithm*, is exact (i.e., it computes exact, not approximate values of the desired probabilities), and avoids simultaneous marginalization over all RVs which one wants to ignore.

To run this efficient algorithm, the original DAG must be transformed into an auxiliary graph structure, called a *join tree*[35], by a rather involved process. This has to be done only once for a given BN however; the same join tree can be re-used for all subsequent inference calls. The transformation from a DAG to a join tree runs through 4 stages[36]:

Given: A BN $\mathcal{G}_d$ with nodes $\mathbf{X} = \{X_1, ..., X_n\}$.

---

[34] Here the Bishop book supplies the following references: (1) Greig, D., Porteous, B., Seheult, A.: Exact maximum a-posteriori estimation for binary images. Journal of the Royal Statistical Society, Series B 51(2), 1989, 271-279. (2) Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. On Pattern Analysis and Machine Intelligence 21(11), 2001, 1222-1239. (3) Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Trans. On Pattern Analysis and Machine Intelligence 26(2), 2004, 147-159.

[35] Also called *junction trees* in the literature.

[36] In the description of the algorithms I closely follow: Huang, C., Darwiche, A.: Inference in Belief Networks: A Procedural Guide. Int. J. of Approximate Reasoning 11 (1994), p. 158ff. http://www.faculty.iu-bremen.de/hjaeger/courses/AlgMod05/ijar95.pdf If you want to implement BNs yourself, you should inspect this paper first.

Step 1: Transform $\mathcal{G}_d$ to an *undirected* graph $\mathcal{G}_m$, called the *moral undirected graphical model*, (moral UGM) which is essentially an equivalent way of expressing the independencies expressed in $\mathcal{G}_d$, but using the UGM "graph separation semantics" for expressing conditional independencies.

Step 2: Add some further undirected edges to $\mathcal{G}_m$ which *triangulate* $\mathcal{G}_m$, obtaining $\mathcal{G}_t$. This may destroy some of the original independence relations between $\{X_1, ..., X_n\}$ but will not introduce new ones.

Step 3: Detect all *cliques* $\mathbf{C}_i$ in $\mathcal{G}_t$ (while this is NP-complete for general undirected graphs, it can be done efficiently for triangulated undirected graphs).

Step 4: Build an undirected *join tree* $\mathcal{T}$ with nodes $\mathbf{C}_i$. This is the desired target structure. It represents an elegant factorization of the joint probability $P(\mathbf{X})$ which in turn can be processed with a fast, local inference algorithm known as *message passing*.

We will go through these steps in an informal manner mostly. The purpose of this presentation is not to provide you with a recipe for building executable code – because, first, nobody would today re-build BN tools from scratch because there exist numerous convenient toolboxes. The other reason for not presenting a specific algorithm in detail is that steps 2 and 4 have non-unique solutions, and you need to utilize some special heuristic algorithm of your liking (many are discussed in the literature). The purpose of our treatment here is just to provide you with a navigation guide to gain an overview of the general picture.

6.4.1 The moral UGN

We first turn to the task of transforming a BN for $P(\mathbf{X})$ into an UGM for $P(\mathbf{X})$. This can be done in many ways. The simplest would be to create a fully connected UGM. The art lies however in transforming a BN into an UGM such that as few as possible of the valuable independence relations expressed in the BN get lost. The other extreme would be to turn a BN for $P(\mathbf{X})$ into an UGM for $P(\mathbf{X})$ simply by re-declaring all directed links from the BN as undirected. This works only rarely. For instance, if we would change the BN from Fig. 6.1 into an UGM without further ado, in the new UGM model we would earn the independence relation $X_2 \perp X_3 \mid \{X_1, X_4\}$ not present in the original BN.

The method that we will use adds only a minimal number of links to the BN, creating the following UGM associated with an BN for $P(\mathbf{X})$:

Definition 6.7: From a BN $\mathcal{G}_d$ with nodes $\mathbf{X} = \{X_1, ..., X_n\}$ construct an UGM $\mathcal{G}_m$ in two steps:

(7.1)Create a copy $\mathcal{G}_m^0$ of $\mathcal{G}_d$. Convert all directed edges in $\mathcal{G}_m^0$ to undirected ones, obtaining $\mathcal{G}_m^1$.

(7.2)If a node X_i had two parents X_j and X_k in $\mathcal{G}_d$, add the undirected edge (j, k) to $\mathcal{G}_m^1$, obtaining the *moral* UGM $\mathcal{G}_m$.

This peculiar name comes from the act of "marrying" previously unmarried parents. The moral UGM $\mathcal{G}_m$ of a BN $\mathcal{G}_d$ does not imply any independence relations that were not already implied in the BN (proof omitted here). Figure 6.5 gives an example of moralizing a BN.

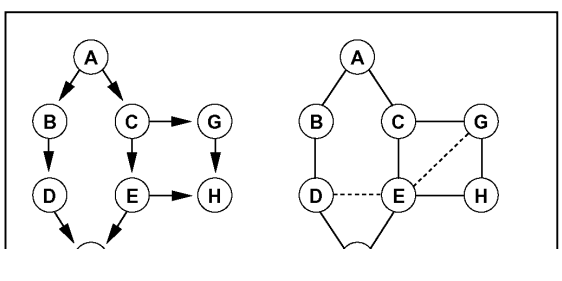


Figure 6.5 A BN and its associated moral UGM (from the Huang & Darwiche paper).

6.4.2 Triangulation

Definition 6.8. An undirected graph is *triangulated* if every cycle of length at least 4 contains two nodes not adjacent in the cycle which are connected by an edge.

Any undirected graph can be triangulated by adding edges. There are in general many ways of doing so. For best efficiency of the inference algorithms, one should aim at triangulating in a way that minimizes the size of cliques of the triangulated graph. This is (again) an NP-complete problem. However there are numerous heuristic algorithms (each one amounting to, say, a master thesis project) which yield close to optimal triangulations in (low) polynomial time. One of them is sketched in the Huang/Darwiche guideline for BN inference design.

After triangulation, we have an UGM $\mathcal{G}_t$. Figure 6.6 shows a possible triangulation of the UGM from Fig. 6.5.

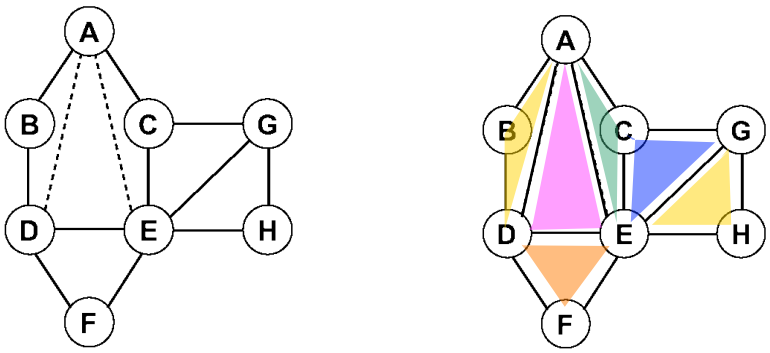


Figure 6.6 Left: A triangulated version $\mathcal{G}_t$ of the moral UGM $\mathcal{G}_m$ from Fig. 6.3 (from the Huang & Darwiche paper). Right: the 6 cliques in $\mathcal{G}_t$.

Triangulation only adds edges. Therefore, no new separation triples **Z**, **Y**, **S** may appear (only some previously found in $\mathcal{G}_m$ might get lost). An inspection of Def. 6.4 reveals that if $\mathcal{G}_m$ was an UGM for $P(\mathbf{X})$, then $\mathcal{G}_t$ will be one, too.

The reason for triangulating is that triangulated graphs can be *decomposed* into junction trees in step 4.

6.4.3 Finding the cliques

Step 3 is to find all cliques $\mathbf{C}_i$ in $\mathcal{G}_t$. For a change, finding all cliques in a triangulated graph is not NP-complete – efficient general-purpose algorithms for finding all cliques in a triangulated graph have been discovered. In the Huang & Darwiche guideline, a special method is instead suggested that determines the cliques already during the triangulation process, with no extra cost involved. In our running example, we get 6 cliques, namely all the "triangles" ABD, ACE, ADE, DEF, CGE, and EGH (our example conveys maybe a wrong impression that we always get cliques of size 3 in triangulated graphs -- in general, one may find cliques of any size greater 1 in such graphs). Figure 6.4 (right) illustrates these.

6.4.4 Building the join tree

This is again a more interesting and involved step. After BNs and UGMs, join trees are our third graph-based representation of independence relations governing a set **X** of random variables. We will discuss join trees in their own right first, and then consider how a join tree can be obtained from the cliques of a triangulated UGM.

Definition 6.9. Let $\mathbf{X} = \{X_1, ..., X_n\}$ be a set of discrete random variables with joint distribution $P(\mathbf{X})$. A *join tree* for $P(\mathbf{X})$ is an undirected tree whose nodes are labelled with subsets $\mathbf{C}_i \subseteq \mathbf{X}$ (called *clusters*) and whose links are labelled with subsets $\mathbf{S}_j \subseteq \mathbf{X}$ (called the *separator sets* or *sepsets*). Furthermore, each cluster $\mathbf{C} = \{Y_1,...,Y_k\}$ is associated with a *belief potential* $\varphi_{\mathbf{C}} : Y_1 \times ... \times Y_k \to \mathbb{R}^{\geq 0}$, and each sepset $\mathbf{S} = \{S_1,...,S_l\}$ with a belief potential $\varphi_{\mathbf{S}} : S_1 \times ... \times S_l \to \mathbb{R}^{\geq 0}$, such that the following conditions are satisfied:

2. The graph has the *running intersection property*, that is, given two nodes labelled with **C** and **C'**, all labels **C*** of nodes on the path between **C** and **C'** must have labels containing $\mathbf{C} \cap \mathbf{C}'$.
3. Each edge between two nodes labelled by **C** and **C'** is labelled by $\mathbf{S} = \mathbf{C} \cap \mathbf{C}'$.
4. For each cluster $\mathbf{C} = \{Y_1,...,Y_k\}$ and neighboring sepset $\mathbf{S} = \{Y_1,...,Y_l\}$, where $l \leq k$ (note that $\mathbf{S} \subseteq \mathbf{C}$), it holds that $\varphi_{\mathbf{C}}$ is *consistent* with $\varphi_{\mathbf{S}}$ in the following sense

(6.18) $$\sum_{y_{l+1},...,y_k} \varphi_{\mathbf{C}}(y_1,...,y_k) = \varphi_{\mathbf{S}}(y_1,...,y_l),$$

that is, $\varphi_{\mathbf{S}}$ can be obtained by marginalization from $\varphi_{\mathbf{C}}$.

5. The joint distribution $P(\mathbf{X})$ is factorized by the belief potentials according to

(6.19) $$P(\mathbf{X}) = \frac{\prod_i \varphi_{\mathbf{C}_i}}{\prod_j \varphi_{\mathbf{S}_j}}.$$

Figure 6.7 shows the graph structure an example of a join tree (which was incidentally derived from our running example $\mathcal{G}_t$).

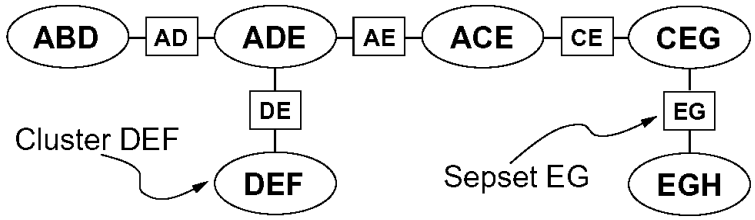


Figure 6.7 A join tree (from the Huang & Darwiche paper)

We defined BNs and UGMs through the statistical independence relations they assert through their graph structure. From these independence relations we could infer factorizations of the joint distribution. In our treatment of join trees we have conversely used a factorization property to define join trees. For completeness, I mention without proof how statistical independence relations are reflected in a join tree:

Proposition 6.4. Let $\mathcal{T}$ be a join tree for $P(\mathbf{X})$ and $\mathbf{Y}, \mathbf{Z}, \mathbf{S} \subseteq \mathbf{X}$. Re-interpret the link labels as nodes (so the join tree from Fig. 6.7 would become an undirected tree with 11 nodes and unlabelled links), obtaining a "homogenized" tree $\mathcal{T}'$. If for all $Y \in \mathbf{Y}$, $Z \in \mathbf{Z}$, the path from any node in $\mathcal{T}'$ containing X to any node containing Y passes through a node labelled with $\mathbf{S}$, then $\mathbf{Y} \perp \mathbf{Z} \,|\, \mathbf{S}$.

An important property of join trees is that the belief potentials are the marginal distributions of their variables:

Proposition 6.5. Let $\mathcal{T}$ be a join tree for $P(\mathbf{X})$, and let $\mathbf{K} = \{X_1, ..., X_k\} \subseteq \mathbf{X} = \{X_1, ..., X_k, X_{k+1}, ..., X_n\}$ be a clique or sepset label set. Then for any value instantiation $x_1, ..., x_k$ of the variables from $\mathbf{K}$, it holds that

(6.20) $$\sum_{x_{k+1},...,x_n} P(x_1,...,x_n) = \varphi_{\mathbf{K}}(x_1,...,x_k),$$

that is, $\varphi_{\mathbf{K}}$ is the marginal distribution of the variables $\mathbf{K}$.

A proof this this proposition amounts to a medium difficult exercise. We will from now on use the shorthand notation

(6.21) $$\sum_{\mathbf{X}\setminus\mathbf{K}} P(\mathbf{X}) = \varphi_{\mathbf{K}}$$

to denote marginalization.

For every triangulated UGM $\mathcal{G}_t$ of $P(\mathbf{X})$ there exist join trees for $P(\mathbf{X})$ whose nodes are the maximal cliques of $\mathcal{G}_t$. We will illustrate this with a simple example before giving the construction algorithm. Consider the UGM from Fig. 6.8 (left).

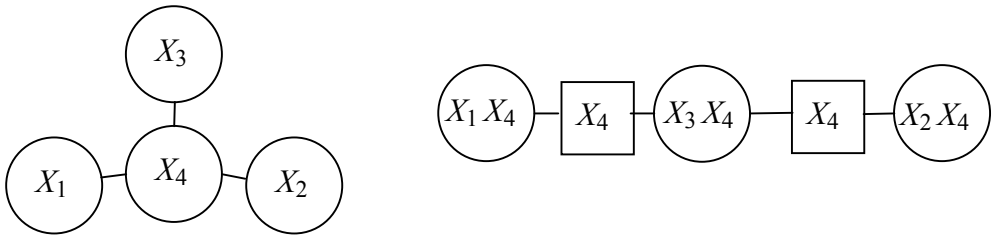


Figure 6.8 A simple UGM (left) and one of its join trees

It should be clear that the joint distribution of $\mathbf{X} = \{X_1, X_2, X_3, X_4\}$ can be written in the following form, exploiting the obvious independence relations asserted by the UGM:

$$
\begin{aligned}
P(x_1,x_2,x_3,x_4) &= P(x_4)P(x_1 \mid x_4)P(x_2 \mid x_4,x_1)P(x_3 \mid x_4,x_1,x_2) \\
&= P(x_4)P(x_1 \mid x_4)P(x_2 \mid x_4)P(x_3 \mid x_4) \\
&= \frac{P(x_4)P(x_1,x_4)P(x_2,x_4)P(x_3,x_4)}{P(x_4)P(x_4)P(x_4)} \\
&= \frac{P(x_1,x_4)P(x_2,x_4)P(x_3,x_4)}{P(x_4)P(x_4)}
\end{aligned}
\tag{6.22}
$$

in which we recognize the factorization (6.19). This example also demonstrates that the join tree obtained from an UGM is not unique (there are obvious variants of the join tree resulting from the symmetry of the UGM) and that a given sepset may appear several times in a join tree.

The general recipe for constructing a join tree from a triangulated UGM for $P(\mathbf{X})$ is simple enough[37]:

1. Begin with an empty set S, and a completely unconnected graph whose nodes are the m maximal cliques $\mathbf{C}_i \subseteq \mathbf{X}$ of the UGM $\mathcal{G}_t$.
2. For each distinct pair of cliques $\mathbf{C}_i$, $\mathbf{C}_j$ create a candidate sepset $\mathbf{S}_{ij} = \mathbf{C}_i \cap \mathbf{C}_j$, and put it into S. S will then contain $m(m-1)/2$ such $\mathbf{S}_{ij}$ (some of which may be the empty set).
3. From S iteratively choose $m-1$ sepsets and use them to create connections in the node graph, such that each newly chosen sepset connects two subgraphs that were previously unconnected.

This general recipe leaves you with much liberty in choosing the sepsets from S. Not all choices will result in a valid join tree. In order to ensure the join tree properties, we choose, at every step from 3., the candidate sepset that has the largest mass (among all those which

[37] I rephrase the recipe given in the Huang/Darwiche paper.

connect two previously unconnected subgraphs). The *mass* of a sepset is the number of variables it contains.

This is not the only possible way of constructing a join tree, and it is still underspecified (there may be several maximal mass sepsets at our disposal in a step from 3.) Huang and Darwiche propose a full specification that heuristically optimizes the join tree w.r.t. the ensuing inference algorithms. Note: if the original BN was not fully connected, some of the sepsets used in the join tree will be empty; we actually get a join *forest* then.

6.5 Computing conditional probabilities on join trees

All this work of transforming a BN into a join tree was done in order to enable efficient inference algorithms for calculating conditional distributions of the kind $P(Y \mid Z_1 = z_1, ..., Z_k = z_k) = P(Y \mid z_1, ..., z_k)$, which we saw in Subsection 6.2. is the basic operation needed for (and sufficient for) all kinds of statistical inferences that one might wish to carry out over $P(\mathbf{X})$. This entire approach was pioneered by Lauritzen an Spiegelhalter in a celebrated paper[38], who in turn built on a diversity of earlier results of a more restricted nature (inference on BNs with a tree structure) due to Pearl and others. In my rendering I again follow closely (sometimes verbatim) the paper by Huang and Darwiche, filling in some gaps.

We will describe how to compute $P(Y \mid z_1, ..., z_k)$ in three subsections:

1. Given the graph of a join tree $\mathcal{T}$ (obtained from a BN with the methods previously described) we show how to construct belief potentials such that Eqns. (6.18) and (6.19) are satisfied, that is, complete the construction of the join tree $\mathcal{T}$.
2. We describe how from $\mathcal{T}$ unconditional distributions $P(Y)$ can be computed.
3. We describe how $P(Y \mid z_1, ..., z_k)$ can be computed.

Notice that in practice one most often wants not only a single probability value $P(y \mid z_1, ..., z_k)$ but the entire distribution $P(Y \mid z_1, ..., z_k)$. For instance, in a medical diagnosis situation the values of Y might be a set of different causes for observed symptoms $z_1, ..., z_k$. A doctor will want to guide his/her treatment decisions on the full spectrum of probabilities for the different possible causes.

6.5.1 Computing belief potentials

Belief potentials accounting for (6.18) and (6.19) are constructed in two steps. First, the potentials are *initialized* in a way such that (6.19) holds. Second, by a sequence of *message passes* that are *globally propagated*, local consistency (6.18) is achieved.

6.5.1.1 Initialization

[38] Landmark article: **Local Computations with Probabilities on Graphical Structures and Their Application to Expert Systems.** S. L. Lauritzen; D. J. Spiegelhalter, *Journal of the Royal Statistical Society. Series B (Methodological)*, Vol. 50, No. 2. (1988), pp. 157-224. Online at http://www.jstor.org/cgi-bin/jstor/printpage/00359246/di993224/99p0299j/0.pdf?backcontext=page&dowhat=Acrobat&config=jstor&userID=d4c92cf9@iu-bremen.de/01cc99333c0050190c661&0.pdf

Initialization works in two steps:

(2.2) For each clique or sepset **K** (we use symbol **K** for cliques **C** or sepsets **S**), set $\varphi_{\mathbf{K}}$ to the unit function:

(6.23) $$\varphi_{\mathbf{K}} \equiv 1 .$$

(2.3) For each variable X_k, do the following. Assign to X_k a clique $\mathbf{C}_{X_k}$ that contains X_k and its parents $\mathbf{\Pi}_{X_k}$ w.r.t. the original directed graph from the starting BN. Note: due to the moralizing, such a clique must exist. Multiply $\varphi_{\mathbf{C}_{X_k}}$ by $P(X_k \mid \mathbf{\Pi}_{X_k})$:

(6.24) $$\varphi_{\mathbf{C}_{X_k}} = \varphi_{\mathbf{C}_{X_k}} P(X_k \mid \mathbf{\Pi}_{X_k}).$$

Make sure that you understand the operation expressed in (6.24) (interpret $P(X_k \mid \mathbf{\Pi}_{X_k})$ as a function of *all* variables contained in $\mathbf{C}_{X_k}$). After this initialization, the conditional distributions $P(X_k \mid \mathbf{\Pi}_{X_k})$ of all variables (and hence the information from the BN) have been multiplied into the clique potentials, and (6.19) is satisfied:

(6.25) $$\frac{\prod_i \varphi_{\mathbf{C}_i}}{\prod_j \varphi_{\mathbf{S}_j}} = \frac{\prod_{k=1,\ldots,n} P(X_k \mid \mathbf{C}_{X_k})}{1} = \frac{\prod_{k=1,\ldots,n} P(X_k \mid \Pi_{X_k})}{1} = P(\mathbf{X}),$$

where i ranges over all cliques, j over all sepsets, and k over all variables.

After having initialized the join tree potentials, we make them locally consistent by propagating the information that has been locally multiplied in across the entire join tree. This is done through a suite of local *message passing* operations, each of which renders one clique/sepset pair consistent. We first describe a single message pass operation and then show how they can be scheduled such that a message pass does not destroy consistency of clique/sepset pairs that have been made consistent in an earlier message passing.

6.5.1.2 A single message pass

Consider two adjacent cliques **C** and **D** with an intervening sepset **S** (as in Figure 6.9), and their associated belief potentials $\varphi_{\mathbf{C}}$, $\varphi_{\mathbf{D}}$, and $\varphi_{\mathbf{S}}$. A *message pass from* **C** *to* **D** occurs in two steps:

1. "Projection": create a copy $\varphi_{\mathbf{S}}^{old}$ of $\varphi_{\mathbf{S}}$ for later use, then recompute $\varphi_{\mathbf{S}}$ by marginalization from **C:**

(6.26) $$\varphi_{\mathbf{S}}^{old} = \varphi_{\mathbf{S}}, \quad \varphi_{\mathbf{S}} = \sum_{\mathbf{C}\setminus\mathbf{S}} \varphi_{\mathbf{C}} .$$

This obviously makes $\varphi_{\mathbf{S}}$ consistent with $\varphi_{\mathbf{C}}$ according to Eqn. (6.18). The joint distribution $P(\mathbf{X})$ becomes changed through this operation by a factor of $\varphi_{\mathbf{S}}^{old} / \varphi_{\mathbf{S}}$ (notice that $\varphi_{\mathbf{S}}$ appears in the *denominator* of (6.19)!)

2. "Absorption": multiply the belief potential of **D** by the inverse of $\varphi_{\mathbf{S}}^{old} / \varphi_{\mathbf{S}}$ in order to restore the joint distribution:

$$\varphi_{\mathbf{D}} = \varphi_{\mathbf{D}} \frac{\varphi_{\mathbf{S}}}{\varphi_{\mathbf{S}}^{old}} . \tag{6.27}$$

A technical detail: if $\varphi_{\mathbf{S}}^{old}(\mathbf{s}) = 0$, then it can been shown that also $\varphi_{\mathbf{S}}(\mathbf{s}) = 0$; in this case set $\varphi_{\mathbf{S}}(\mathbf{s}) / \varphi_{\mathbf{S}}^{old}(\mathbf{s}) = 0/0 = 0$.

After this step, **C** is consistent with **S** in the sense of (6.18). To also make **D** consistent with **S**, a message passing in the reverse direction must be carried out. An obvious condition is that this reverse-direction pass must preserve the consistency of **C** with **S**. This is warrented if a certain order of passes is observed, to which we now turn our attention.

6.5.1.3 Coordinating all message passes

In order to achieve local consistency (6.18) for all neighboring clique-sepset pairs in the join tree, as many message passes as there are such pairs must be executed, one for each pair. In order to avoid that local consistency for a pair **C**, **S** achieved by a message pass from **C** to **D** (where **S** is a sepset between **C** and **D**) is not destroyed by a subsequent message pass in the reverse direction, the global order of these passes is crucial. We will motivate a global propagation scheme by considering some connection **C – S – D** within the tree (where **S** is a sepset between **C** and **D**), as depicted in Figure 6.9:

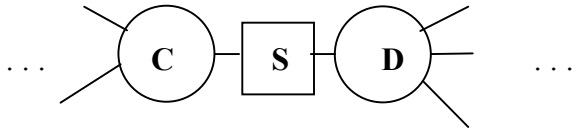


Figure 6.9 A connection **C – S – D** within the tree.

This connection will be hit twice by a message pass, one in each direction. Assume that the first of the two passes went from **C** to **D**. After this pass, we have potentials $\varphi_{\mathbf{C}}^{0}$, $\varphi_{\mathbf{D}}^{0}$, and $\varphi_{\mathbf{S}}^{0}$, and **S** is consistent with **C**:

$$\varphi_{\mathbf{S}}^{0} = \sum_{\mathbf{C} \setminus \mathbf{S}} \varphi_{\mathbf{C}}^{0} . \tag{6.28}$$

At some later time, a message pass sweeps back from **D** to **C**. Before this happens, the potential of **D** might have been affected by some other passes, so it is $\varphi_{\mathbf{D}}^{1}$ when the pass from **D** to **C** occurs. After this pass, we have

(6.29) $$\varphi_{\mathbf{S}}^{1} = \sum_{\mathbf{D}\setminus\mathbf{S}} \varphi_{\mathbf{D}}^{1} \quad \text{and} \quad \varphi_{\mathbf{C}}^{1} = \varphi_{\mathbf{C}}^{0} \frac{\varphi_{\mathbf{S}}^{1}}{\varphi_{\mathbf{S}}^{0}} .$$

It turns out that still **S** is consistent with **C**:

(6.30) $$\varphi_{\mathbf{S}}^{1} = \varphi_{\mathbf{S}}^{0} \frac{\varphi_{\mathbf{S}}^{1}}{\varphi_{\mathbf{S}}^{0}} = (\sum_{\mathbf{C}\setminus\mathbf{S}} \varphi_{\mathbf{C}}^{0}) \frac{\varphi_{\mathbf{S}}^{1}}{\varphi_{\mathbf{S}}^{0}} = \sum_{\mathbf{C}\setminus\mathbf{S}} (\varphi_{\mathbf{C}}^{0} \frac{\varphi_{\mathbf{S}}^{1}}{\varphi_{\mathbf{S}}^{0}}) = \sum_{\mathbf{C}\setminus\mathbf{S}} \varphi_{\mathbf{C}}^{1} .$$

In sum, we see that if a connection **C – S – D** is hit by two passes, one for each direction, **S** will be consistent with **C** and **D**. In order to ensure consistency for all connections in the tree, we must make sure that after some connection **C – S – D** has been passed back and forth, neither **C** nor **D** take part in any further passes, as this might again disrupt the already achieved consistency. The following global scheduling scheme assures this condition:

1. To start, single out any clique **C** and call it the "center".
2. In a first phase ("collect evidence" in the Huang/Darwiche paper), carry out all passes that are oriented towards the center. Carry out these passes in any order such that a pass "leaves" some node on some connection only after all *other* "incoming" connections have been used for passes.
3. In a second phase ("distribute evidence"), carry out all passes that are oriented away from the center. Observe the same order constraint as in the first phase.

Figure 6.10 shows a possible global scheduling for our example join tree.

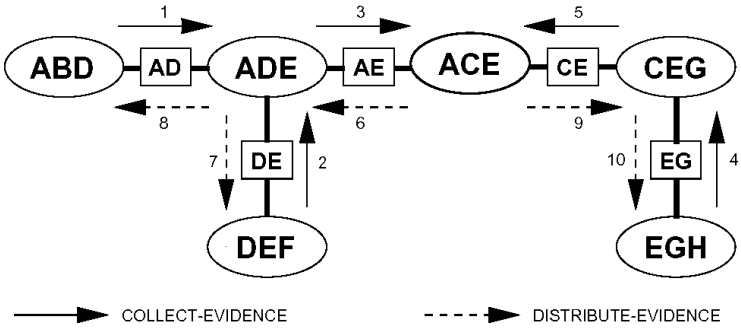


Figure 6.10 A scheduling for the global propagation of message passing. The center is ACE. (Figure taken from the Huang/Darwiche paper)

6.5.2 Computing $P(X)$

Having a join tree $\mathcal{T}$ for $P(\mathbf{X})$, computing the marginal distribution $P(X)$ of any of the variables is a simple two-step procedure:

1. Identify a clique or sepset **K** that contains X.
2. Obtain $P(X)$ by marginalization:

(6.31) $$P(X) = \sum_{\mathbf{K}\setminus\{X\}} \varphi_{\mathbf{K}} .$$

This directly follows from (6.20).

6.5.3 Computing $P(Z \mid e_1, ..., e_k)$

We have now arrived, after a long algorithmic journey, in the destination harbour: computing those quantities which really interest us, namely, conditional distributions of the kind $P(Z \mid e_1, ..., e_k)$. Here $Z, E_1, ..., E_k \in \mathbf{X}$, and $e_1, ..., e_k = \mathbf{e}$ are particular values for $E_1, ..., E_k = \mathbf{E}$. We call the set of concrete observations **e** *evidence*.

Let $e_1, ..., e_k = \mathbf{e}$ be an evidence, that is, values $e_1, ..., e_k$ have been observed, measured ore otherwise been ascertained. To feed in this information into $\mathcal{T}$, we introduce a new kind of belief potentials called (in this context) *likelihoods*. For each $E \in \mathbf{E}$, we define the likelihood Λ_E: image$(E) \to \{0,1\}$ of E as

(6.32) $$\Lambda_E(e) = 1 \quad \text{if } e \text{ is the observed value of } E$$
$$\Lambda_E(e) = 0 \quad \text{for all other } e \in \text{image}(E)$$

That is, the likelihood is a single-point distribution in the observed value e.

In order to compute $P(Z \mid e_1, ..., e_k) = P(Z \mid \mathbf{e})$, we have to go through the routine of initializing $\mathcal{T}$ and making it consistent via global message propagation again, using an augmented version of the method described in Subsection 6.5.1. Here is how:

1. **Initialization**: exactly like in Subsection 6.5.1.1. This yields belief potentials $\varphi_{\mathbf{C}}, \varphi_{\mathbf{S}}$ for all cliques and sepsets.
2. **Observation entry**: for each $E \in \mathbf{E}$, do the following:

   (a) Identify a clique $\mathbf{C}_E$ that contains E.
   (b) Update $\varphi_{\mathbf{C}_E} = \varphi_{\mathbf{C}_E} \Lambda_E$.

   Note that (b) simply resets $\varphi_{\mathbf{C}_E}$ to zero for all arguments that have a different value for E than the observed one. It is easy to see that with the new potentials, the tree globally encodes $P(\mathbf{X})\,\mathbf{1}_{\mathbf{e}}$, where $\mathbf{1}_{\mathbf{e}}$: $\otimes X_i \to \{0,1\}$ is the indicator function of the set $\{(x_1,...,x_n) \subseteq image(\otimes X_i) \mid x_i = e_j \text{ for } X_i = E_j, j = 1,...,k\}$.

(6.33) $$\frac{\prod_i \varphi_{\mathbf{C}_i}}{\prod_j \varphi_{\mathbf{S}_j}} = P(\mathbf{X})\Lambda_{E_1} \cdots \Lambda_{E_k} = P(\mathbf{X})\mathbf{1}_{\mathbf{e}} =: P(\mathbf{X},\mathbf{e})$$

   Note furthermore that $\sum_{\mathbf{X}\setminus\mathbf{E}} P(\mathbf{X},\mathbf{e}) = P(\mathbf{e})$.

3. **Global propagation**: exactly like in Subsections 6.5.1.2 and 6.5.1.3, but starting from the updated potentials obtained by observation entry. After this is completed, the join tree is locally consistent, that is, (6.18) holds. Furthermore, each clique or sepset **K** has a potential satisfying

(6.34) $$\varphi_{\mathbf{K}} = P(\mathbf{K}, \mathbf{e}) = P(\mathbf{K})\mathbf{1}_{\mathbf{e}}.$$

4. **Normalization**: In order to compute $P(Z \mid \mathbf{e})$, determine a clique $\mathbf{C}_Z$ that contains Z. When we marginalize this clique's potential to Z, we obtain the probability of Z *and* $\mathbf{e}$:

(6.35) $$\sum_{\mathbf{C}_Z \setminus \{Z\}} \varphi_{\mathbf{C}_Z} = P(Z, \mathbf{e}).$$

Our goal is to compute $P(Z \mid \mathbf{e})$, the distribution of Z *given* $\mathbf{e}$. We obtain this by normalizing $P(Z, \mathbf{e})$ as follows:

(6.36) $$P(Z \mid \mathbf{e}) = \frac{P(Z, \mathbf{e})}{P(\mathbf{e})} = \frac{P(Z, \mathbf{e})}{\sum_Z P(Z, \mathbf{e})}.$$

6.6 Creating BNs in the first place... where do the probabilities come from?

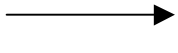
We have seen how we can calculate statistical inferences on a *given* BN. But who gives it to us, and how?

A common approach, especially in AI, is to simply distil the conditional probability tables from your intuition – or by interviewing an expert of the target domain about which values s/he finds appropriate in the light of professional experience. This is a very valuable approach, because domain experts typically have quite clear conceptions about local conditional probabilities that connect a few variables. You might ask, why use a BN if such experts are available in the first place? The answer is that while humans might have a good insight on local interdependencies between a few variables, they are psychologically and intellectually poorly equipped to use this local knowledge for drawing statistically sound inference that connect variables that are further apart in the independence network. In fact, humans tend to make gross, systematic errors in such tasks, and BN-based *decision support systems* are in practice extremely useful even if they are constructed solely from "local intuitions".

In other cases, empirical observations are available that allow one to *estimate* the conditional probability tables from data. For instance, the table (6.2) could have been estimated from counts # of observed outcomes in the following, obvious way:

X_2	X_3	#($X_4 = 0$)	#($X_4 = 1$)
0	0	7	0
0	1	2	18
1	0	1	9
1	1	2	198

frequency counts



X_2	X_3	$P(X_4 = 0)$	$P(X_4 = 1)$
0	0	1.0	0.0
0	1	0.1	0.9
1	0	0.1	0.9
1	1	0.01	0.99

maximum likelihood probability estimates

Figure 6.11 Example of estimating a probability table from frequency counts.

Notice that each row in such a table is estimated independently from the other rows. The task of estimating such a row is technically the same as estimating a discrete distribution from data. We have discussed maximum likelihood and Bayesian mean posterior estimators for this task in Section 2. The latter approach is often important in practice because firstly, often the empirical sample sizes are small and second, high-quality expert opinion is available that serves as a valuable Bayesian prior information, greatly enhancing the quality of the final model.

A situation that is also common in practice arises when a BN is ready and in place, but additional information comes in at a later point and should somehow be used to modify the existing BN. We have already encountered an extreme case of such "later information", namely, the evidence **e** resulting from some observation. It was used to modify the belief potentials of the join tree. "Later information" can take other forms, for instance noisy observations. Such *soft evidence* can be fed into an existing BN, or rather its join tree representation, by exactly the same mechanism described in Subsection 6.5.2 for the "hard" observation evidence, if instead of a point distribution Λ_E: image(E) $\rightarrow$ {0,1} a general distribution Λ_E: image(E) $\rightarrow$ [0,1] is employed.

However we have only scratched the surface yet. It is very often the case that for some of the variables, neither empirical observations nor an expert's opinion is available, either because simply the observations have not been attempted or because these quantities are in principle unobservable. Such unobservable variables are called *hidden variables*. To get an impression of the nature and virtues of hidden variables, consider the following BN:

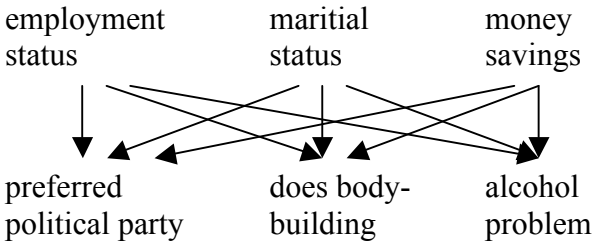


Figure 6.12 A BN for use by a social worker (apologies to professionals in the field)

Now, the maritial status is hardly *causal* for political preferences. According to some (folk) psychology, the following BN would more appropriately capture the causal logics connecting our variables:

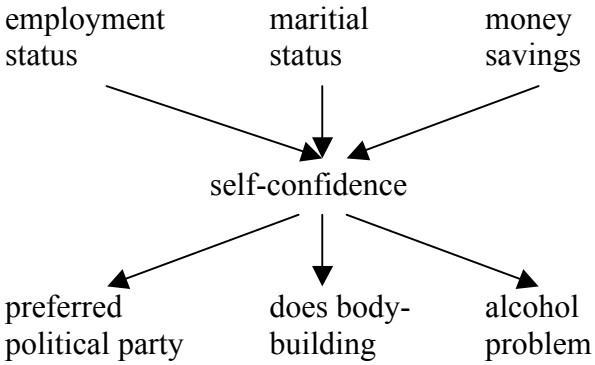


Figure 6.13 A BN for use by a psychologically and statistically enlightened social worker

However, nobody has yet found a convincing way to directly measure self-confidence – it is an *explanatory concept*, and becomes a hidden variable in a statistical context. While all other variables in this BN can be readily measured, self-confidence can't. Yet the augmented BN is in an intuitive sense more valuable as a model of a piece of reality than the first one, because it tries to reveal a causal mechanism whereas the former one only superficially connects variables by arrows that can hardly be understood.

Besides being intellectually more pleasing, the second BN offers substantial computational savings: its join tree (construct it!) is much more leightweight than the first BN's, so statistical inference algorithms will run much faster.

Generalizing from this simplistic example, it should be clear that hidden variables are a great asset in modelling reality. But – they are hidden, which means that the requisite probability tables cannot be directly estimated from empirical data.

Fortunately, there exists a powerful, general-purpose algorithm that allows one to estimate the probability tables for hidden variables *from the observation statistics of the observable variables*. This is the "expectation-maximization" (EM) algorithm, whose discovery in 1977 by Dempster, Laird and Rubin[39] marks a breakthrough in statistics and machine learning. More precisely, this algorithm achieves the following. Let θ denote the parameters necessary to specify all the tables that make up a BN. For discrete-valued variables, θ is the set of all probability values in the conditional probability tables – including the tables of the hidden variables. Let D denote the available empirical data (of course, D says nothing about the hidden variables). Then the EM algorithm finds a (locally optimal) solution to the problem

$$\theta_{opt} = \arg\max_{\theta} P(D \mid \theta), \tag{6.37}$$

that is, a maximum likelihood estimate of θ. Other algorithms serving the same purpose are not known. The EM algorithm is computationally demanding and not trivial to set up, but opens a door to the unobservable. Here I only want to avert you to the existence of this algorithm[40], which is of fundamental importance for learning BNs and many other statistical models, notably hidden Markov models, the backbone of much of automated speech processing and biosequence analysis.

6.7 Concluding remarks

This completes our brief sightseeing tour in the land of Bayesian Networks. Here is a summary of the ab-so-lu-te-ly not-to-forget basic messages:

[39] Dempster, A.P. , Laird, N.M. and Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). Journal of the Royal Statistical Society series B 39, 1-38

[40] I treat this algorithm in some of my Machine Learning lectures.

1. One approach (arguably the most powerful and general) to model complex real-world systems is to describe them by a set **X** of observables and investigate the joint distribution $P(\mathbf{X})$.
2. $P(\mathbf{X})$ is typically an ultra-complex entity whose complexity exponentially explodes with the number of observables. One way to reduce this complexity[41] is to exploit statistical independence relationships between the variables. Such independence relations can conveniently be expressed in directed or undirected graphs, that is, BNs or UGMs, which are collectively called graphical models.
3. Most tasks of reasoning about $P(\mathbf{X})$ reduce to calculating conditional probabilities $P(Y \mid \mathbf{e})$.
4. The most common way to calculate precise values $P(Y \mid \mathbf{e})$ is the join tree method. Here from a given BN or UGM first a join tree graph is constructed, from which $P(Y \mid \mathbf{e})$ is obtained by an initialization / observation entry / propagation / normalization algorithm.
5. The probabilities constituting the local tables of a BN are often set by hand on an intuitive basis. Equally often they are estimated from empirical data. Hand-setting and estimation from data can be mixed, using Bayesian methods. For more advanced BNs incorporating hidden variables, the EM algorithm has to be used.

Graphical models constitute a major, modern field of research in Artificial Intelligence / knowledge representation and machine learning, and a one-year full-scale graduate course (or an entire academic or engineering carreer, for that matter) could easily be filled with them. I hope that you have gotten a fair first impression nonetheless.

[41] Another fundamental way to reduce complexity, particularly popular in physics and neural modelling, is to take a *mean field approach*, exploiting that in many physical systems, some variable Y approximately depends just on the *average* value of other variables **Z**. Incidentally, mean field methods can sometimes be exploited for inference in BNs.

7 Fuzzy Logic

7.1 Introduction

Fuzzy logic (FL) is an approach to cast imprecise expert knowledge into a mathematical format that lies somewhere between propositional logic (to enable chains of inference), Bayesian statistics (to enable "soft" interaction between concepts), and neural networks (some of which can be seen as special cases of FL systems and vice versa). In contrast to either of these three, FL is refreshingly simple and easy to work with. The approach dates back to a paper by Lotfi Zadeh[42], who worked in control theory. The area of control is still the dominant application arena for FL, and FL controllers are at work in devices as costly as the space shuttle (for docking control) or as mundane as camera autofocus controllers. FL based algorithms are also used in time series prediction, scheduling and optimization, and signal processing. For a long time, FL was considered by engineering snobs as disreputable – it has definitely a boorish "hands-on" flavour and lacks an elegant mathematical underpinning. But, especially Japanese high-tech firms that didn't care about academic attitudes and used FL very successfully. After some time, even the stronghold of traditional electrical engineering, the IEEE, had to establish a FL society (now immersed in the IEEE Computational Intelligence Society, http://ieee-cis.org/) and a journal (the IEEE Transactions on Fuzzy Systems, http://ieee-cis.org/pubs/tfs/), and sponsored a series of FL conferences.

There are a number of FL tutorials on the Web. I will rely on one written by J. M. Mendel[43], which is quite detailed and easy to read. An even easier to read tutorial, albeit with a smaller coverage, has been supplied by J. Jantzen[44]. I really would not need to write my own account of the material, but I will do so nonetheless, because I find that the original FL literature (including the tutorials) suffers too much from the desire to make the field respectable, leading to unneccessary (in my view) efforts to connect FL to logics proper, which obscures the simplicity of the approach.

In order to understand the original motivation for FL, we must briefly dive into the world of control engineering. "Control" is a broad concept and generally refers to the task to compute time-dependent input $u(t)$ to a system such that some objective for the system output $y(t)$ is reached or maintained. For instance, $u(t)$ may be the voltage applied to an electric motor such that the rpm $y(t)$ is kept at a constant target level even in the presence of varying loads. The algorithm/device that actually computes and administers the control input $u(t)$ is called the *controller*, and the system that one wishes to control is customarily called the *plant*. There are various types of objectives that one may wish to achieve. The most important ones are listed here in order of increasing difficulty:

- **stabilize** the plant's output $y(t)$ at a constant reference value r_0 (the controller is then called a stabilizer);
- let the plant's output $y(t)$ **track** a time-dependent reference trajectory $r(t)$ – you are doing this when steering a car!;
- **optimize** the integral or max of some cost function $c(y(t))$ within a certain time horizon – for instance, buy and sell stock such that the expected profit in the next two years is

[42] L. A. Zadeh, *Fuzzy Sets*, Information and Control **8**, 338-353, 1965
[43] Jerry M. Mendel, *Fuzzy Logic Systems for Engineering: a Tutorial. IEEE Proc.*, vol. 83, no. 2, pp. 345-377, March 1995. http://sipi.usc.edu/~mendel/publications/FLS_Engr_Tutorial_Errata.pdf
[44] Jan Jantzen, *Tutorial on Fuzzy Logic*. http://www.iau.dtu.dk/~jj/pubs/logic.pdf

maximized, or turn the knobs and valves of a chemical reaction plant such that the quality of the output product is best. This is the field of *optimal control*.

There are two major classes of control strategies. In *feedback* (or *closed-loop*) control, the controller is guided by information it gets from monitoring the plant's output $y(t)$. In its classical simplest form, all that a feedback controller gets as input is the current *error* $\varepsilon(t) = y(t) - r(t)$. Figure 6.1 **a** shows an elementary feedback controller. In *direct* (or *open-loop*) control, the controller receives (only) the reference signal $r(t)$ as its input (Fig. 6.1 **b**). Both approaches have complementary merits. Feedback control is never able to achieve its (tracking) objective perfectly, because a feedback controller always needs some error to build up before it can adapt the control $u(t)$. On the positive side, a feedback controller is able to compensate disturbances (e.g., if you externally brage an electric motor, the error will grow and the feedback controller will increase voltage input to make the motor return to the desired rpm). A direct controller can, in principle, achieve 100% perfect control – that is, zero error; but only as long as there are no disturbances. Therefore it is not surprising that in practice one often combines aspects of open-loop and closed-loop control. This is however never trivial to achieve, and there is a plethora of different control architectures whose block diagrams may contain a host of "boxes".

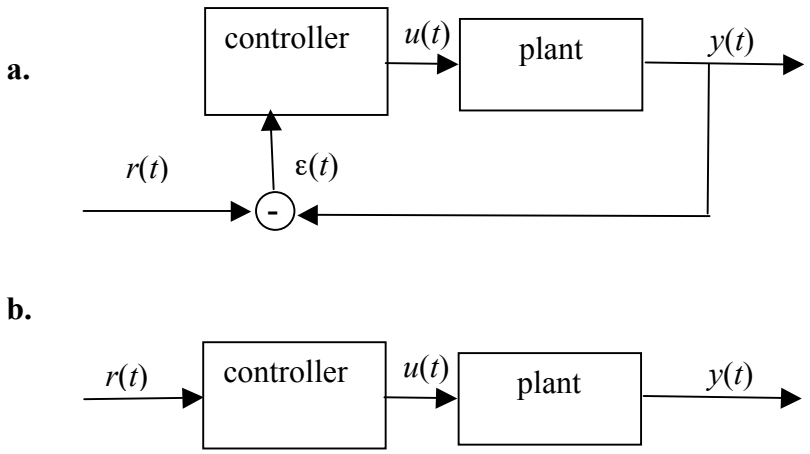


Figure 7.1 An elementary feedback controller (**a**) and a direct controller (**b**), shown for the case of tracking control.

Now consider a sequence of plants of increasing complexity: a thermostat – an electric motor – a robot arm – a legged robot – a chemical reaction with dozens of substances involved – an assembly line – a power plant – a nationwide power distribution system – an economical system. Classical (linear) control theory has been developed for well over a hundred years and today offers rigorous methods to design controllers for plants up the complexity of, say, a robot arm. Making a legged robot walk is something for which classical control methods so far offer only marginally satisfactory solutions, and for anything beyond that there exists only one class of controllers that is up to the respective task – human operators. To be more precise, *highly experienced* human operators.

So, what makes the task of controlling, say, a chemical destillation process, so difficult for an algorithm, and apparently feasible for a human expert? There are many reasons, for instance

- There are many different "knobs" to turn – the input $u(t)$ is actually a high-dimensional vector $\mathbf{u}(t)$, and to make matters worse, the different knobs may be of mathematically different nature. Some may be continuous, others discrete numerical, yet others symbolic ("add catalyst A" vs. "add catalyst B"). Generally, mathematics has difficulties with heterogeneous data structures. Humans somehow can do it quite well.
- There are delays involved. If the heating energy in a chemical reactor is increased, then the actual increase in temperature will lag behind, and the influence of the increased temperature on the chemical products forming will even lag further. Controlling systems with various delays is mathematically very challenging. Human experts acquire a "feeling" for delays, they are able to predict the delayed effects of current input. In fact, the ability of humans to instinctively learn predictive models is breathtaking, and human experts are distinguished from human novices very much by virtue of the former's better predictive models.
- Many of the variables that one would wish to know for improving control are unobservable. For instance, in order to tune an ongoing chemical reaction, it would clearly help to monitor the various by-products that are forming in the reactor – but this is often infeasible because a chemical analysis would be too slow or too expensive. Human experts can typically quite nicely estimate the current values of "hidden" variables.

All in all, it seems a clever idea to investigate the way how human experts determine their control actions, and mimick this if possible. This is how fuzzy logic came into being. FL is an attempt to copy the reasoning processes of human experts into an algorithmical framework.

You may say, also the classical logic-based methods approaches of Artificial Intelligence, or Bayesian statistics are attempts to cast human, intelligent reasoning into a formalism. But there is an important philosophical difference between classical AI and Bayesian statistics on the one side, and FL on the other. The former two are *normative*: they cast into formalism ways of how humans *should* reason, namely, reason *correctly* with respect to the rigorous laws of logics or statistics. In contrast, FL is *descriptive*: this approach "simply" wishes to mimick what humans actually do, regardless of whether these actions are mathematically sound in any sense or can be derived from some stringent set of axioms. So it comes as no surprise that FL has no set of axioms and lacks a mathematical theory. This in turn makes that FL comes in a huge number of different variants because each FL developer is essentially free to redefine FL concepts according to his/her intuitions and objectives.

The latter point makes teaching and learning FL a bit difficult. There is no standard FL system on the market. I will simply adhere to the FL flavour(s) used in the tutorials cited earlier, pointing out alternatives here and there.

7.2 Fuzzy sets

The starting point for FL is the observation that human control engineers, when asked about why they do what, typically come up with answers like this:

(7.1) "If the pressure is too low, I turn the pressure valve a bit left."

Zadeh started out from sentences like this one and thought out a way of how they can be cast into formal terms. His first observation was that humans use *linguistic terms* (that is, everyday language expressions) instead of numerical values. The engineer would not say, "If the

pressure is 2.43, I turn the pressure valve by –13 degrees". The engineer, according to FL, treats `pressure` as a *linguistic variable*, not a numerical variable. A numerical variable has numbers as possible values; a linguistic variable has linguistic terms as possible values. What these possible values are depends on the particular engineer and is essentially an arbirtrary choice. For instance, the linguistic variable `pressure` might have possible values (= linguistic terms) {`dangerously low`, `much too low`, `too low`, `OK`, `too high`, `much too high`, `dangerously high`}.

Now, the control engineer of course has access to real numbers. In a situation where s/he says (7.1), s/he might just be reading a *numerical value* of $x = 2.43$ from a pressure meter. And of course s/he knows that pressure might range from 0 to 12 (after which the plant would explode...). But whenever a numerical value is observed, the first thing done by the human expert is a classification and naming by one of the linguistic terms reserved for pressure. This assignment depends on the numerical value in a *fuzzy* way. The numerical range of `too low` might be anything between $x = 2.1$ and $x = 3.5$, with $x = 2.43$ at the low end of this category. Assuming that the category `much too low` spans numerical values from 1.1 to 2.5, the value of 2.43 might also be regarded as a case of `much too low`, – but rather, it is just `too low`, isn't it? So we see, following Zadeh and FL, that assigning a numerical value to a linguistic term is a matter of degree. We might say, a value of 2.43 is `too low` by a degree of 0.6, whereas it is `much too low` by a degree of only 0.2. Such degrees are typically measured by a *membership function* which ranges from 0 to 1. For instance, the membership function $\mu_{\texttt{too low}}$ could look like in Figure 7.2:

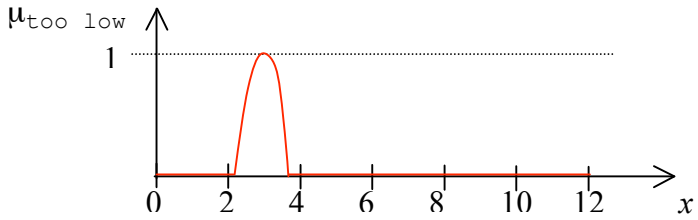


Figure 7.2 A membership function for the linguistic term `too low`.

A value of $\mu_{\texttt{too low}}(x) = 1$ indicates that x is, one might say, "maximally typical" for the linguistic term `too low`. A value of $\mu_{\texttt{too low}}(x) = 0$ means that x is not considered as `too low` at all. The x-range where $\mu_{\texttt{too low}}(x) > 0$ is called the *support* of the linguistic term `too low`. In our example, the support of `too low` is [2.1, 3.5].

If $\mu_{\texttt{A}}$ is a membership function of some linguistic term `A` with support $S_{\texttt{A}} \subseteq \mathbb{R}$, (almost always $S_{\texttt{A}}$ is a finite interval), then the graph set $\{(x, \mu_{\texttt{A}}(x) \mid x \in S_{\texttt{A}}\}$ is called the *fuzzy set* associated with `A`. Fuzzy logic adepts spend a lot of time explaining that this generalizes the notion of a standard (sub-)set, which would be obtained if $\mu_{\texttt{A}}$ is binary with values either 0 or 1. In classical set theory, a value x either belongs to `A` (then $\mu_{\texttt{A}}(x) = 1$) or doesn't (then $\mu_{\texttt{A}}(x) = 0$). Thus fuzzy sets can be seen as "sets with gradual membership".

In Figure 7.3, the complete account of the linguistic variable `pressure` is given, by plotting the fuzzy sets for all its linguistic terms.

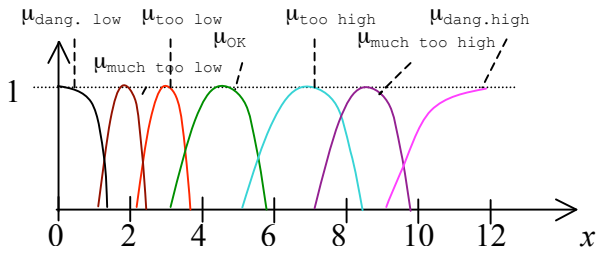


Figure 7.3 All the fuzzy sets associated with the linguistic variable `pressure`.

Importantly, the action taken by the engineer depends on which of the possible linguistic terms the numerical value is assigned to – likely, if 2.43 would be considered (and named) `much too low`, a more decisive action than turning the valve "a bit left" would have been taken.

There is another linguistic variable and linguistic term involved in (7.1), which I mention for completeness. Namely, the linguistic variable `valve action` – you may give it another name – is assigned to a value of `a bit left`. Other linguistic terms for this variable might include `fully open` or `don't change`. The numerical range for `valve action` would presumably differ from that of `pressure`, say, it is [−1,1].

It might appear at first sight that the gradual membership function is related to a probability – after all, both range between 0 and 1 and somehow describe certainty of judgement. A nice example in the Mendel tutorial highlights that probability and fuzzyness are quite different things. Imagine that a hiker in a desert, almost dried out and short of dying, finds two bottles. Bottle A is labelled, "the contents of this bottle is drinkable with a probability of 0.9", the other bottle B is labelled "$\mu_{\texttt{drinkable}}$(*contents of this bottle*) = 0.9". Which bottle would you drink from? So take these lecture notes along on your next desert hike...

Let's orderly summarize the basic concepts of fuzzy sets, and introduce the FL terminology in a summary fashion:

- A physical/numerical quantity x, like pressure, can take in a given (control) context a range of possible values. In our example $x \in [0,12]$. This range of possible numerical realizations is called the *universe of discourse*. A given control task may involve different quantitites of interest and thus different universes of discourse. The universe of discourse is a classical set. We consider only universes of discourse which are subsets of the reals; FL can also accomodate universes of discourses that are arbitrary classical sets, but this is rarely seen in control applications and we will ignore this possibility.
- When a human expert reasons/talks about such a quantity, s/he treats it not as a numerical variable but as a *linguistic variable* that can take a few *linguistic terms* as values.
- A particular numerical value x is gradually assigned to one or more linguistic terms by virtue of the *membership functions* in whose domains x happens to fall. In the example from Figure 7.3, a value of x = 2.43 would be assigned to `much too low` with a membership degree of $\mu_{\texttt{much too low}}(2.43) = 0.2$ and to `too low` with a membership degree of $\mu_{\texttt{too low}}(2.43) = 0.6$. Standardly, membership functions have a maximum value of 1.0, however this is not required.

- A *fuzzy set* of a linguistic term A is identified with the set $\{(x, \mu_A(x) \mid x \in S_A\}$, where S_A is the support of μ_A. If the membership function μ_A is binary, a fuzzy set can be seen as a classical set.

There are interesting and useful things one can do with fuzzy sets. Consider the following variant of (7.1):

(7.2) "If the pressure is really too low, I turn the pressure valve about a quarter left."

Here the value `too low` of the linguistic variable `pressure` has been modified by "really", and the value `a quarter left` of `valve action` has been modified by "about". Such linguistic modifications play a formidable role in human reasoning. They can be accomodated in FL by the use of operators that are applied to membership functions. Sometimes such operators are called *hedges*. For instance, the modification "really" *concentrates* the membership function $\mu_{\texttt{too low}}$ around its maximum. A membership function $\mu_{\texttt{really too low}}$, compared to $\mu_{\texttt{too low}}$, should assign to x's at the outskirts of the support of $\mu_{\texttt{too low}}$ a lower value than in the original $\mu_{\texttt{too low}}$. Conversely, a modification by "about" makes a linguistic term even more fuzzy than it originally is – it should act as a *dilation*. The standard way to implement concentration and dilation hedges is to use exponentiation by 2 and 1/2:

(7.3) $\text{concentrated}(\mu_A) = \mu_A^{\,2}$, $\text{diluted}(\mu_A) = \mu_A^{\,1/2}$.

Concentration becomes manifest by words like "very", "strongly", "really" etc., dilution by works like "somehow", "about", "more or less", etc. Other powers than 2 or 1/2 can also be used – it is up to the FL modeller to choose.

In FL, essentially everything can be decided by the modelling engineer – it appears to me that only the terminology is about fixed (see how I use a dilation here?), but not what the terminology refers to. Specifically, the shape of membership functions is not restricted by any theoretical constraints. The most commonly used format appears to be triangles (because they are the simplest to program and fastest to calculate), followed by Gaussians (because they are smooth) which are set to 0 outside a certain interval to yield a finite support. More recently, *B-splines* are apparently gaining in favour, especially in certain high-precision applications of FL in the area of time series prediction. I take this opportunity to explain B-splines, because they are generally very useful for interpolation and function approximation.

B-splines are generalization of Bézier curves (hence the "B"), which you probably know from drawing programs. B-splines are used to create smooth interpolation curves between control points in $\mathbb{R}^n$. Check out http://mathworld.wolfram.com/B-Spline.html for a concise intro. Here we are interested not in the B-splines themselves but only in the 1-dimensional *basis functions* $N_k^j : \mathbb{R} \to \mathbb{R}$ they are made of, where j refers to the placement of the basis function on the real line and k determines the *order* of the function.

In order to create N_k^j, one first has to fix $k + 1$ *knots* $\lambda_0 < \lambda_1 < \ldots < \lambda_k$ on the real line. Define the j-th interval $I_j = [\lambda_{j-1}, \lambda_j)$. Then N_k^j is defined recursively by

$$N_1^j(x) = \begin{cases} 1, \text{if } x \in I_j \\ 0, \text{else} \end{cases}$$

(7.4)

$$N_k^j(x) = \left(\frac{x - \lambda_{j-k}}{\lambda_{j-1} - \lambda_{j-k}} \right) N_{k-1}^{j-1}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}} \right) N_{k-1}^j(x)$$

An inspection of (7.4) reveals that N_1^j is a zero-th order polynomial (namely, a constant) on each interval $I_{j'}$. Because in the recursion equation, a k-th order basis function is essentially created by multiplying x into k-1-th order basis functions with some coefficients, it is clear that a B-spline basis function N_k^j of order k is piecewise polynomial of order $k - 1$ on its support. Figure 7.3 shows some B-spline basis functions of different order.

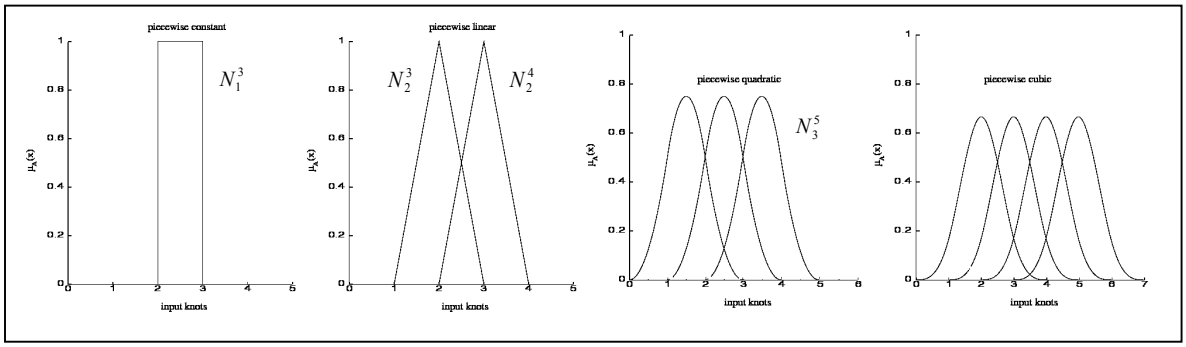


Figure 7.3 B-spline basis functions of orders 1 to 4, with equidistant knots λ_j placed at the integers[45].

B-spline basis functions have a number of pleasant properties which explain why they are so often used:

- They are fast to compute, using the regression equations (7.4).
- k-th order basis functions are k-2 times differentiable.
- For a sequence of knots ... $\lambda_i < \lambda_{i+1} <$ ... and a fixed order k, the N_k^i form a *partition of unity*, that is, for any x, $\sum_{i=-\infty}^{\infty} N_k^i(x) = 1$.

So far we have considered continuous-valued unverses of discourse. The membership functions are then continuous functions $\mu : \mathbb{R} \to \mathbb{R}$. In many practical applications, one also meets *discrete* universes , either because the "piece of world" one is modelling is inherently discrete (for instance, the customers in a customer database) or because one prefers to discretize a continuous domain for reasons of computational efficiency. A membership function on a discrete universe $= \{x_1, ..., x_N\}$ of size N is then simply an N-dimensional vector

[45] Figure taken (and redrawn with corrections) from: Zhang, J., Knoll, A., Renners, I., *Efficient Learning of Non-Uniform B-Splines for Modelling and Control*. Proc. Int. Conf. of Computational Intelligence for Modelling, Control and Automation, Vienna 1999.

(7.5) $\mu \cong (\mu(x_1),...,\mu(x_N))$. [46]

Discrete membership functions are also mostly normalized by convention to a range of [0,1], but it makes no sense to require smoothness or any other geometric property for the shape of the function – discrete membership functions typically are not intended to be "bell-shaped" or "triangular".

7.3 Fuzzy logic basics

Human experts can combine concepts into new ones. In the classical world of logics and set theory, this is reflected on the syntactical side by applying the logical connectives ∧, ∨, ¬ to propositions, and on the semantic side by applying the corresponding set-theoretic operations ∩, ∪, ^{c} (the latter denoting the complement operation). For instance, the concept expressed by the first-order logic formula "`Car` x ∧ `Expensive` x" is semantically interpreted (in the extensional semantics that has become standard for logic) by $\texttt{Car}^{\mathcal{A}} \cap \texttt{Expensive}^{\mathcal{A}}$, where $\texttt{X}^{\mathcal{A}}$ denotes the set by which a predicate `X` is interpreted in a model $\mathcal{A}$ -- in plain English, here this refers to the set of all expensive cars.

Now let us consider two linguistic terms `A` and `B` represented by membership functions μ_A and μ_B on a shared universe of discourse. For instance, think of `A` as the value `OK` for the linguistic variable `pressure` over the universe [0,12] and of `B` as the value `high` of the linguistic variable `mechanical strain` over the same universe (assuming that pressure determines mechanical strain). Then an engineer might mumble something like, "If the pressure is OK and the strain is high, I would suggest to tune down the pressure valve a bit". Just like in classical logic, the "and" operation creates a new predicate from two which is semantically represented by the set-theoretic cut, we desire a "fuzzy and" operation that defines a new linguistic term `OK pressure and high strain` through a new membership function $\mu_{\texttt{OK pressure and high strain}} =: \mu_{\texttt{OK} \cap \texttt{high}}$. One design constraint that has guided fuzzy logicians for coming up with a recipe to define $\mu_{\texttt{A} \cap \texttt{B}}$ from μ_A and μ_B is that in the limiting case of binary (= classical) membership functions μ_A and μ_B, the fuzzy definition should coincide with the classical one, that is, for binary μ_A and μ_B it should hold that

$$(7.6) \qquad \mu_{A \cap B}(x) = \begin{cases} 1, \text{ if } \mu_A(x) = \mu_B(x) = 1 \\ 0, \text{ all other cases} \end{cases}$$

Fuzzy sets with binary membership functions, which thus can be regarded as classical sets, are called *crisp* sets in FL.

Now it turns out that the requirement (7.6) vastly underspecifies admissible "fuzzy and" operations. There are arbitrarily many, more or less reasonable numerical operators on membership functions that in the limiting binary case behave like (7.6). The two most commonly employed instantiations of a "fuzzy and" are

$$(7.7) \qquad \mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) \qquad \text{and}$$

[46] In the tutorial paper by Mendel, for such objects the rather weird notation $\mu \cong \sum_{x \in \text{suppport}(\mu)} \mu(x)/x$ is used.

(7.8) $$\mu_{A\cap B}(x) = \mu_A(x)\,\mu_B(x).$$

Both are examples of a larger family of "fuzzy and" operations that have become known as *t-norm operators*[47], and which often are generically denoted by the star symbol ★:

(7.9) $$\mu_{A\cap B}(x) = \mu_A(x) \;\bigstar\; \mu_B(x).$$

T-norm operators are all operators that conform to the following definition:

Definition 7.1. An operator T: [0, 1] × [0, 1] → [0, 1] is a T-norm operator if it satisfies the following conditions:

1) $T(0, 0) = 0$; $T(a, 1) = a$ ("boundary" condition)
2) $a \le c$ and $b \le d \Rightarrow T(a,b) \le T(c,d)$ (monotonicity)
3) $T(a, b) = T(b, a)$ (communtativity)
4) $T(a, T(b, c)) = T(T(a, b), c)$ (associativity)

We encounter here for the first time a most disturbing fact about FL: the almost complete freedom to design concrete specs for even the most elementary operations. This results from the fact that the limiting, crisp set case imposes too weak restrictions to guide the design, and that no axiomatic "master theory" exists that could replace the "distant mirror", classical logic.

Likewise, for the classical "or / set union" operation, the limiting constraint for crisp sets

(7.10) $$\mu_{A\cup B}(x) = \begin{cases} 1, \text{ if } \mu_A(x) = 1 \text{ or } \mu_B(x) = 1 \\ 0, \text{ all other cases} \end{cases}$$

admits many fuzzy versions. Common examples are

(7.11) $$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x)) \quad \text{and}$$
(7.12) $$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\,\mu_B(x).$$

The second is the operator originally proposed by Zadeh. Again, some requirements on reasonable "fuzzy or" operations have been distilled into the concept of *t-conorm* operators (also known as *s-norm* operators); if one wishes to refer to any such operation generically, the symbol ⊕ is often used:

(7.13) $$\mu_{A\cup B}(x) = \mu_A(x) \oplus \mu_B(x).$$

T-conorm-operators are defined exactly like *T*-norm operators except for the boundary condition, which here becomes

1') $T(1, 1) = 1$; $T(a, 0) = a$ ("boundary" condition for *T*-conorms)

[47] The "*t*" comes from "triangular". Introduced in a paper by R. R. Yager and D. P. Filev: *Template-based fuzzy systems modeling*. J. Intell. and Fuzzy Systems **2**, 1994, 39-54.

The logical "not / set complement" operation is however more standardized; in all texts I saw its fuzzy version is implemented by

(7.14) $$\mu_{A^c}(x) = 1 - \mu_A(x).$$

What about an engineer's statement like the following?

(7.15) "If the pressure is too high and the temperature is far above norm, then I turn the heating rigorously down"

Here the antecedent apparently refers to two different universes of discourse, one of pressure (ranging in our example in [0,12]) and another of temperature (which might range in [0,100]). In order to connect the FL modelling of such statements back to classical logic, it is convenient to work with a variant of classical logic known as *many-sorted logic* or simply *sorted logic*. In the ACS 1 lecture we treated only *one-sorted* or *unsorted* predicate logic. In one-sorted logic, interpretations $\mathcal{A}$ have a single carrier A, such that written they look like $\mathcal{A} = (A, ...)$, where the "..." should be filled by the symbols from the respective symbol set. By contrast, sorted predicate logic has interpretations that have several sets as carriers and in writing look like $\mathcal{A} = (A, B, ..., Z, ...)$, where the carrier sets $A, B, ..., Z$ are called the *sorts* of the interpretation. Many-sorted logics are useful if a piece of world is to be modelled that clearly contains things of different type, for instance, people, animals, and fruit. In one-sorted logics, there would only be one category of elements of the world (you might call it "objects"), and the fact that indeed there are people, animals and fruit would be captured by introducing unary predicates `people`, `animal` and `fruit`. Instead, one can immediately agree on the convention that the world is made from different (disjoint) sets P, A, and F; then if $p \in P$ it is immediately clear that p is a person and not an animal. The predicates `people`, `animal` and `fruit` become superfluous. This convenience is however bought at a price: the predicate and function symbols that one wishes to use must be *sorted*, that is, together with each predicate it must be specified of which sort are its arguments. For instance, a unary predicate `grows_on_trees` x could only accept arguments of sort F (from the fruit set). By and large, unsorted and sorted logics can express the same things: the sorts of a sorted logic can be captured in an unsorted logic by unary predicates, and an unsorted logic is simply a special case of a sorted logic.
Returning to the statement (7.15), the "and" in the antecedent connects two linguistic terms over different universes, or as the classical logician would say, predications of different sort. Our original "fuzzy and" definitions (7.7) and (7.8) do not capture this case. They can be extended as follows. Let `A` be a linguistic term over a universe of discourse X, `B` be a linguistic term over a universe of discourse Y, and μ_A and μ_B be two associated membership functions. The the statement template,

(7.16) "x is `A` and y is `B`"

has pairs $(x, y) \in X \times Y$ as arguments. The membership function for the statement (7.16) is therefore defined over $X \times Y$, and in analogy with (7.7) and (7.8) reads

(7.17) $$\mu_{A\cap B}(x,y) = \min(\mu_A(x), \mu_B(y))$$ or

(7.18) $$\mu_{A\cap B}(x,y) = \mu_A(x)\,\mu_B(y).$$

The "fuzzy or" generalizes to the multiple-sorted case in the same way. It should also be clear how the "fuzzy and" and "fuzzy or" generalize to a larger number of arguments, e.g., "x is `A` and y is `B` and z is `C`" etc (exercise on homework sheet).

Another elementary item from classical logics that has been imported into FL is *relations*. We treat only binary relations; n-ary relations can be treated in an obvious way. In classical unsorted logic, a binary relational expression like `smaller` x y is semantically represented in an interpretation $\mathcal{A} = (A, \texttt{smaller}^{\mathcal{A}})$ by a subset of $A \times A$:

(7.19) $$\texttt{smaller}^{\mathcal{A}} \subseteq A \times A,$$

or in classical sorted logic, a binary relation like `drives` (were elements of sort P [for `person`] drive elements of sort V [for `vehicle`]) would be a subset of $P \times V$:

(7.20) $$\texttt{drives}^{\mathcal{A}} \subseteq P \times V.$$

Note that subset relations of the kind $X \subseteq Y \times Z$ can be expressed by the indicator function

(7.21) $$\mathbf{1}_X : \quad Y \times Z \to \{0,1\}, \quad (y,z) \mapsto \begin{cases} 1, & \text{if } (y,z) \in X \\ 0, & \text{if } (y,z) \notin X \end{cases}$$

Now, a fuzzy binary relation simply generalizes the binary indicator function to a membership function with values in [0,1], that is, pairs (x, y) can belong to the relation by varying degree. For example, a fuzzy binary relation on a shared universe A would be represented by

(7.22) $$\mu_{\texttt{smaller}}: A \times A \to [0,1];$$

a fuzzy binary relation on pairs that come from different universes would be exemplified by

(7.23) $$\mu_{\texttt{drives}}: P \times V \to [0,1].$$

Finally, we turn to *implications*. They are of particular interest to the FL community because control engineers typically describe their precious know-how in IF-THEN clauses – as in our examples (7.1), (7.2), (7.15). The most natural way to handle implications would be to copy the classical definition of "$\to$" in terms of $\wedge$, $\vee$, $\neg$:

(7.24) $$\varphi \to \psi \quad :\Leftrightarrow \quad \neg\varphi \vee \psi\,.$$

Since we already know how to implement "fuzzy or" and "fuzzy negation", this would straightforwardly lead to

(7.25) $$\mu_{A \to B}(x,y) = \mu_{A^c \cup B}(x,y) = (1 - \mu_A(x)) \oplus \mu_B(y)\,.$$

Concretely, if we use the max operator for $\oplus$, this would spell out as

(7.26) $$\mu_{A \to B}(x,y) = \max(1 - \mu_A(x), \mu_B(y))\,.$$

Unfortunately, this apparently obvious way to proceed has proven to be a stumbling block for FL. An implication defined as in (7.25), (7.26) does not satisfy some simple logical

tautological laws which are valid for two-valued logic and that fuzzycists wanted to keep. For instance, the Boolean tautology $(\psi \wedge \varphi) \rightarrow \psi$ is not a tautology in FL under (7.26). It is, e.g., easily checked that $\mu_{(A \wedge B) \rightarrow A}(0.5,0.5) = 0.5$, while an outcome of 1.0 would be wanted from a tautology. The desire to keep as many as possible of the classical laws of logic intact has spurred fuzziologists to find other definitions for implication than (7.25). One example from a multitude of proposed constructions[48] is the *Gödel implication* $\varphi \rightarrow \psi \;\; :\Leftrightarrow \;\; (\varphi \leq \psi) \vee \psi$. It is easy to check (homework exercise) that this implication indeed makes $(\psi \wedge \varphi) \rightarrow \psi$ a tautology, and preserves some other classical tautologies as such. However, no fuzzy implication found so far preserves all the classical tautologies.

Now let us consider when and how IF-THEN rules are actually "fired" in a control context. Consider again the statement (7.1), here repeated for convenience:

(7.1) "If the pressure is too low, I turn the pressure valve a bit left."

This rule lies dormant in the brain of the control engineer and nothing happens until the pressure gauge shows a value that can be classified as too low – then the rule becomes triggered, and the engineer will turn the valve a bit left. However, *how much exactly* the valve is turned left depends on *by which degree*, *exactly*, the pressure can be judged as too low. Here we come to the heart of FL: it allows us to fine-tune the output of (fuzzy-)logical reasoning. We will consider a number of different cases before we present the general rule of computing the action consequence of fuzzy rules.

- Example 1: Assume that the gauge shows a value of 2.2. According to the membership function $\mu_{\texttt{too low}}$ from Figure 7.2, a pressure of $x = 2.2$ would only be marginally judged "too low", because $\mu_{\texttt{too low}}(2.2)$ is only slightly above zero (let's say, $\mu_{\texttt{too low}}(2.2) = 0.1$). Somehow this should make the rule (7.1) fire only weakly, resulting in only a slight activation of the desire to turn the valve a bit left. ($\mu_{\texttt{much too low}}(2.2) > \mu_{\texttt{too low}}(2.2)$, so we would expect that another rule, not given here, for the case of much too low pressure fires more strongly than (7.1), presumably yielding a stronger left-turn of the valve).
- Example 2: Assume that the gauge shows a value of 3.0, right in the center of the $\mu_{\texttt{too low}}$. Now the rule (7.1) should fire with maximal strength, fully commanding the action taken by the engineer (assuming there are no other rules with an antecedent using $\mu_{\texttt{too low}}$).
- Example 3: Assume that the engineer does not read the pressure gauge, but his apprentice tells him in words, "the pressure is too low". Then again, the rule (7.1) should become activated – although it is not intuitively quite clear whether it should take command with the same strength as in the second case, or with more or with less strength.
- Example 4: Assume the apprentice says, "the pressure is much too low". Then presumably there is some rule in the engineer's brain that has the antecedent "IF the pressure is much too low, ...", and this rule would very much determine the action taken. But, the rule (7.1) would *also* be activated a little bit, because the membership functions $\mu_{\texttt{too low}}$ and $\mu_{\texttt{much too low}}$ have overlapping support, and a much too low pressure is at the same time also a too low pressure, albeit to a small degree only.

[48] A paper by Kiszka, Kochanska and Sliwinska (*The Influence of Some Fuzzy Implication Operators on the Accuracy of a Fuzzy Model*, Fuzzy Sets and Systems **15**, 111-128 and 223-240, 1985) lists 72 alternative implication definitions to choose from [cited after the Jantzen tutorial)

- Example 5: Assume that the engineer knows also the following rule:

(7.27) "If the temperature is less then 40°, then the pressure is likely to be much too low,"

and assume further that he reads a temperature of 35° from the temperature meter. Then the temperature reading will trigger rule (7.27), which in turn will trigger our old rule (7.1), because the consequence of (7.27) matches (to the degree that `too low` and `much too low` overlap). Here we see a *chaining* of rules.

In all examples, there is some empirical information provided that triggers the rule(s). In cases 3 and 4, this information is just $\mu_{\texttt{too low}}$ and $\mu_{\texttt{much too low}}$, respectively. The observations $x = 2.2$ and $x = 3.0$ from the first two cases can likewise be coded by *singleton membership functions*

$$\mu_{x_0}(x) \mapsto \begin{cases} 1, \text{ if } x = x_0 \\ 0, \text{if } x \neq x_0 \end{cases} \tag{7.28}$$

that is, in our case by $\mu_{2.2}$ and $\mu_{3.0}$; in the last case, μ_{35} will be the initial trigger information.

FL has a generic way to accomodate all these examples and many more. To motivate it, notice that the IF-THEN rule (7.1) is represented by a membership function of the kind $\mu_{\texttt{A}\to\texttt{B}}(x,y)$, that is, the membership function of a binary fuzzy relation – regardless of which concrete instantiation we use for the implication. Likewise, the triggering information ($\mu_{\texttt{too low}}$, $\mu_{\texttt{much too low}}$, $\mu_{2.2}$, $\mu_{3.0}$, or μ_{35}) is the membership function of a unary fuzzy relation (that is, a fuzzy predicate, that is, a linguistic term). Notice furthermore that in all cases 1—5 we saw a *chaining* of arguments happen. This is clear in case 5, where the consequence of one rule triggers another rule whose antecedent it matches. In the other examples, we have a chaining of an empirical observation which "fires" a rule whose antecedent it matches. In all cases we thus witness a chaining of membership functions, which we will denote by $\circ$ (following the Jantzen tutorial). The format of the chaining is

- Example 3: $\mu_{\texttt{A}} \circ \mu_{\texttt{A}\to\texttt{B}}(x,y)$ – here the trigger $\texttt{A} = \mu_{\texttt{too low}}$ is identical to the antecedent of the implication (7.1).
- Examples 1, 2, 4: $\mu_{\texttt{A}'} \circ \mu_{\texttt{A}\to\texttt{B}}(x,y)$ – here the triggers $\texttt{A}' = \mu_{2.2}$, $\mu_{3.0}$, $\mu_{\texttt{too low}}$ are not identical to the antecedent $\texttt{A} = \mu_{\texttt{too low}}$ of the rule, but they are related to it by sharing the same universe of discourse.
- Example 5: here we see the chaining pattern $\mu_{\texttt{A}'} \circ \mu_{\texttt{A}\to\texttt{B}}(x,y) \circ \mu_{\texttt{B}'\to\texttt{C}}(y,z)$.

Chaining of membership functions of relations (unary, binary or *n*-ary) is the general-purpose mechanism of FL to combine information in, one might say, "chains of reasoning". I will present the definition of two special cases here, (a) chaining membership functions of two binary relations that share the "middle" universe of discourse and (b) chaining a unary with a binary membership function where the universe of discourse of the unary relation coincides with the first universe of discourse of the binary one:

Definition 7.2. **(a)** Let $\mu_{\texttt{R}}: X \times Y \to [0,1]$ and $\mu_{\texttt{S}}: Y \times Z \to [0,1]$ be two membership functions of binary relations `R` and `S`, where the second universe of discourse Y of the first relation is the same as the first universe of discourse of the second relation. Then

(7.29) $$\mu_{R\circ S}: \quad X \times Z \to [0,1], \quad (x,z) \mapsto \sup_{y \in Y} \mu_R(x,y) \star \mu_S(y,z)$$

defines the *composition* (*chaining*) of μ_R with μ_S.
(b) Let $\mu_A: X \to [0,1]$ and $\mu_S: X \times Y \to [0,1]$ be a unary and a binary membership function. Then

(7.30) $$\mu_{A\circ S}: \quad Y \to [0,1], \quad (y) \mapsto \sup_{x \in X} \mu_A(x) \star \mu_S(x,y)$$

defines the composition of μ_A with μ_S.

Notice that this definition is open w.r.t. the concrete choice of *t-norm* ★ – the min and the algebraic product t-norm are mostly used. Notice furthermore that the "sup" can be replaced by "max" for discrete membership functions. Definition 7.2 can be generalized to arbitrary *n*- and *m*-ary relations `R` and `S`, with no further constraints on the "middle" universes of discourse (homework exercise).

For discrete membership functions μ_R and μ_S the composition (7.29) can be implemented in a way that resembles matrix multiplication. If $|X| = k$, $|Y| = m$, $|Z| = n$, then μ_R is represented by a $k \times m$ table (`R`) = $(\mu_R(i,j))_{i=1,\ldots,k;\, j=1,\ldots,m}$ and μ_S by a similar $m \times n$ table (`S`). The result table (`R`∘`S`) = $(\mu_{R\circ S}(i,j))_{i=1,\ldots,k;\, j=1,\ldots,n}$ is then obtained by computing the "matrix product" of (`R`) and (`S`), where the ordinary multiplication is replaced by ★ and the ordinary summation by the max operator. Likewise, the composition (7.30) can be implemented by a "vector-matrix product" of a $1 \times m$ matrix (`A`) with an $m \times n$ matrix (`S`), resulting in a $1 \times n$ matrix (`A`∘`S`). Computing (7.30) becomes particularly simple when μ_A is a singleton membership function, with a value of 1 in the *j* th position of the row vector (`A`). Then the "vector-matrix multiplication" (`A`∘`S`) boils down to an element-wise application of the max operation to the *j*-th row in (`S`).

Now the stage is prepared for demonstrating what computations would be used in FL to compute a membership function for the target term "`valve action`" of our examples 1 through 5. Let us first consider example 1. Using (7.30) with the singleton $\mu_{2.2}$ for the trigger which is composed with $\mu_{\texttt{too low}}$, the min for ★, and (7.26) for $\mu_{\text{too low}\to\text{a bit left}}$, we obtain

(7.31)
$$\begin{aligned}\mu_{2.2\circ(\text{too low}\to\text{a bit left})}(y) &= \sup_{x\in[0,12]} \min(\mu_{2.2}(x), \mu_{\text{too low}\to\text{a bit left}}(x,y)) = \mu_{\text{too low}\to\text{a bit left}}(2.2,y) \\ &= \max(1-\mu_{\text{too low}}(2.2),\ \mu_{\text{a bit left}}(y)) = \max(0.9,\ \mu_{\text{a bit left}}(y))\end{aligned}$$

OOOps!!! something seems to have gone wrong here, because $\max(0.9, \mu_{\text{a bit left}}(y))$ is certainly always at least 0.9! For a grahical representation of this disturbing finding, assume that the universe of discourse for `valve action` is $Y = [-180, 180]$ degrees, and that $\mu_{\text{a bit left}}(y)$ is centered at –13 degrees. Figure 7.4 illustrates $\mu_{\text{a bit left}}$ and $\mu_{2.2\circ(\text{too low}\to\text{a bit left})}$.

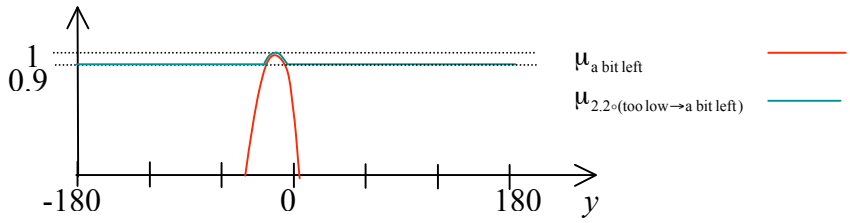


Figure 7.4 A somewhat unsatisfactory computation of a control command $\mu_{2.2\circ(\text{too low}\to\text{a bit left})}$ upon an input of pressure $p = 2.2$.

The membership function $\mu_{2.2\circ(\text{too low}\to\text{a bit left})}$, which should represent the linguistic term for the desired control action after an input of $\mu_{2.2}$, is equal to 0.9 everywhere except for a slight bump at the crest of $\mu_{\text{a bit left}}$. Thus the control system would suggest, if it could speak, an advice that goes like this: "oh my, doesn't really matter what action you take – turn the valve by – 180 degrees or by + 180, looks both very good to me, maybe you should try –13, that looks a little bit better even, but really, it doesn't make a difference...."

This effect would not vanish if we would be using a different choice than max and min for ★ and ⊕, and it also doesn't go away if we use the inputs from the other examples in our list. If you analyse where it comes from, you will see that the undesired, everywhere high values of the command output membership function are due to the nature of (classical) logical implication. The truth table of $\varphi \to \psi$ assigns a 1 to almost every input combination, especially also to those where the truth value of φ is 0:

(7.32)

φ	ψ	$\varphi \to \psi$
0	0	1
0	1	1
1	0	0
1	1	1

Since we tried to model the fuzzy implication after the classical one, we also inherited this "almost everywhere high output value" behaviour. In control applications, the IF-THEN statements of engineers are *simply not the classical logical implications*! A control engineer does not intend to "fire" the THEN part if there is no IF-input, but this is what classical implication does.

In this situation, fuzzy control engineers do a very wild thing: they discard the guidance from classical logic and define their own implication (indeed called "engineering implication" in the Mendel tutorial!), which is as simple as can be:

(7.33) $\mu_{\mathbb{A}\to\mathbb{B}}(x,y) = \mu_{\mathbb{A}}(x) \bigstar \mu_{\mathbb{B}}(y)$,

that is, they use the "fuzzy and" for the implication. If the min t-norm is used for ★, then

(7.34) $\mu_{A\to B}(x,y) = \min(\mu_A(x),\mu_B(y))$

is called the *Mamdani implication* after the person who first dared to propose this trick. But honestly, didn't you secretly always think that

(7.35)

φ	ψ	$\varphi \to \psi$
0	0	0
0	1	0
1	0	0
1	1	1

is the proper interpretation of the intuitive "IF-THEN"? and isn't this also the way how IF-THEN is handled in programming languages? After all, if the IF-premise of an IF-THEN statement in a computer program is not matched, nothing happens. If there is no cause, there should be no effect... So, in a sense, the engineering implication only puts our naive understanding into effect again.

As a final twist, consider the case of IF-THEN rules with several antecedents connected by the word "and" and a single consequence, as in (7.15). The generic format of such rules is

(7.36) "If $\mathtt{y}_1 = \mathtt{Y}_1$ and ... and $\mathtt{y}_m = \mathtt{Y}_m$, then $u = \mathtt{U}$."

Here $\mathtt{Y}_1$, ..., $\mathtt{Y}_m$, $\mathtt{U}$ are linguistic terms of linguistic variables $\mathtt{y}_1$, ..., $\mathtt{y}_m$ and u; we assume they have universes of discourse Y_1, ..., Y_m, U, respectively. Note that if we wish to have several consequences in a rule, we can simply create several single-consequence rules which taken together are equivalent to the single multiple-consequence rule.

A rule of this kind is represented by an m+1-ary membership function

(7.37) $\mu_{(\mathtt{Y}_1\cap\ldots\cap\mathtt{Y}_m)\to\mathtt{U}} : Y_1 \times \ldots \times Y_m \times U \to [0,1]$.

If the engineering implication is used, then $\mu_{(\mathtt{Y}_1\cap\ldots\cap\mathtt{Y}_m)\to\mathtt{U}}$ is obtained from the membership functions of the occurring linguistic terms by

(7.38) $\mu_{(\mathtt{Y}_1\cap\ldots\cap\mathtt{Y}_m)\to\mathtt{U}} = (\mu_{\mathtt{Y}_1} \star \ldots \star \mu_{\mathtt{Y}_m}) \star \mu_{\mathtt{U}} = \mu_{\mathtt{Y}_1} \star \ldots \star \mu_{\mathtt{Y}_m} \star \mu_{\mathtt{U}}$.

If convenient, the antecedent arguments can be lumped into cartesian product linguistic variables $\mathbf{y} = \mathtt{y}_1 \times \ldots \times \mathtt{y}_m$, $\mathbf{Y} = \mathtt{Y}_1 \times \ldots \times \mathtt{Y}_m$ on a universe of discourse $\mathbf{Y} = Y_1 \times \ldots \times Y_m$, obtaining the notation

(7.39) $\mu_{\mathbf{Y}\to\mathtt{U}} = \mu_{\mathbf{Y}} \star \mu_{\mathtt{U}}$.

7.4 Fuzzy Control

Now we can assemble everything together to obtain a complete FL control "box", which receives as input (numerical) measurements $\mathbf{y} = (y_1, \ldots, y_n)$, processes these inputs by applying

some FL operations on them, and returns a numerical output vector $\mathbf{u} = (u_1, ..., u_k)$ which contains the control actions that are administered to the plant. Figure 7.5 shows a block diagram of such a FL control box.

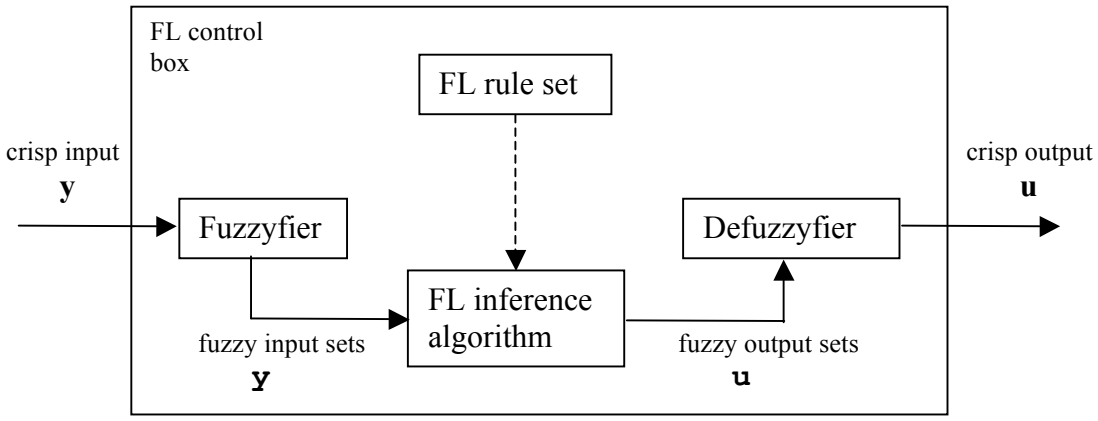


Figure 7.5 Schema of a FL control box.

We now describe the components of such a control box in turn.

7.4.1 The FL rule set

The rule set is a collection of rules of the form (7.39) $\mu_{\mathbf{Y}\to\mathbf{U}} = \mu_{\mathbf{Y}} \bigstar \mu_{\mathbf{U}}$. These rules represent the "knowledge base" of the controller. There are two major ways of how such a rule set can be created:

1. Interview a human expert how s/he goes about a particular control task and code this expert's statement in FL rules. This approach reflects the original intentions behind FL.
2. Learn the rules from data. To achieve this, the actions of a (human or mechanical) controller are monitored over time and rules are extracted by methods of machine learning. We will not further consider this option here, but I want to note that this approach of learning rules (and membership functions!) from observed data is popular and widely treated in the FL literature.

Both approaches can be combined in various ways, for instance by first extracting knowledge from an expert, yielding a coarse first version of a control system, which is then refined by automated learning methods whereby the numerical parameters of the rule base (e.g., support and shape of membership functions) are optimized.

The rule base is the core of a FL control system, and we have learnt in the previous sections the most important things we need to know about a rule base. I will round off the picture by going through a standard example, which is often used in the control literature as a benchmark control problem, not only for FL based control systems but for nonlinear control methods in general – the *truck backing up* (TBU) control task.

In one of its simplest forms,the TBU task refers to a truck that has to be maneuvred backwards into a driving lane. The objective is to steer it such that it drives along the center line of the lane. The initial position of the truck can be as awkward as you can imagine

(possibly requiring difficult initial maneuvres with a lot of back-and-forth driving), but is often constrained to a more harmless initial configuration from which it is possible to achieve a tracking of the centerline without changing into forward gear.

Figure 7.6 shows the relevant variables for this task, featuring a rather easy starting position. The lane *x*-position x has a numerical range from 0 to 20, with 10 being the target value of driving in the middle of the lane. Angles φ and θ denote the truck heading and wheel steering angle, which range in [-90, 270] and [-45,45] degrees, respectively. The control task is to produce a time series θ(*t*) such that the truck backs up the lane on the centerline, that is, to stabilize the system at a state of $x = 10$, $\varphi = 90°$. The input to the controller is the current state $(x(t), \varphi(t))$.

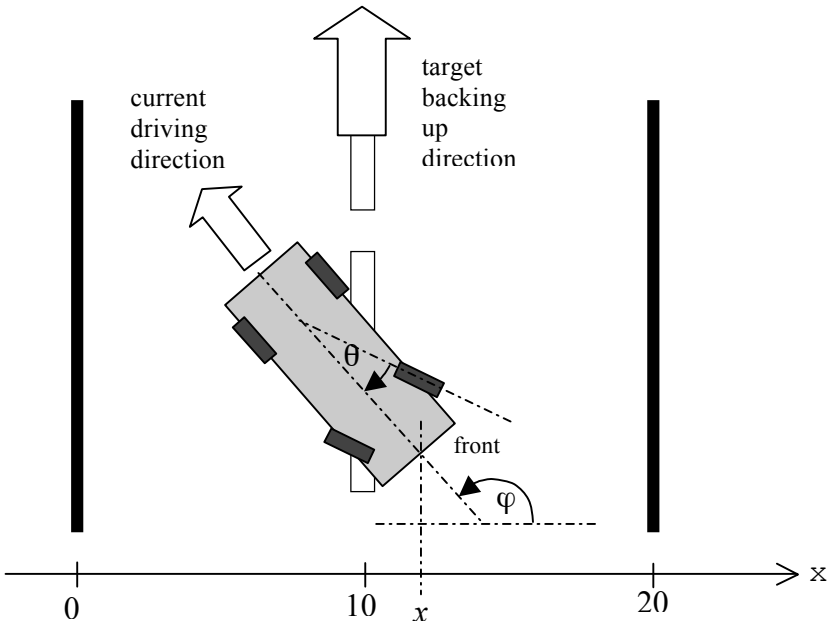


Figure 7.6 Setup of the truck backing up task (adapted from the Mendel tutorial)

Here is a short account of one possible way to design a rule set that covers the essential maneuvres (I follow the Mendel tutorial). The linguistic variables x and heading φ are modelled by linguistic terms as detailed out in Figure 7.7.

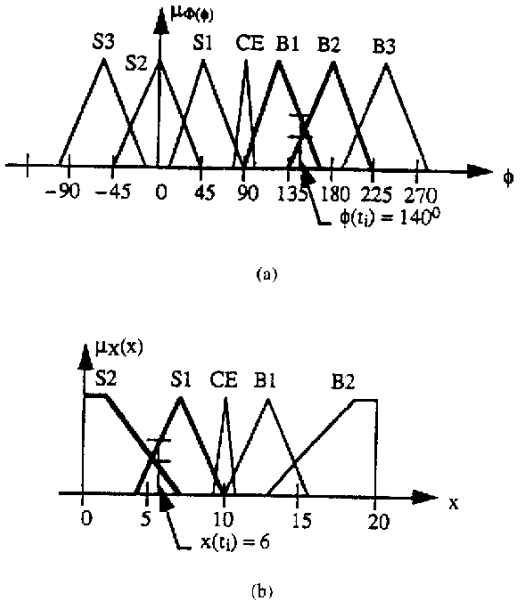


Figure 7.7 Membership functions for the truck backing up problem. (Taken from the Mendel tutorial). S, CE, B stand for Small, Center, Big. Membership functions for steering angle (not shown) use 7 triangles similar to the ones shown here for φ.

In order to create rules, the truck driving expert is asked what steering angle he would use for any of the reasonable value combinations of the state variables x and heading. One such rule (corresponding to the situation shown in Fig. 7.6) might read like this:

(7.40) "If the lane position is a bit bigger than on target [B1], and if the heading is also a bit bigger than 90° [B1], I would steer by a small positive angle [B1]."

Not all combinations of state variable values would be expected to occur in practice. The driver might choose the 27 combinations shown in Figure 7.8, which gives a condensed account of the entire rule base (cells show rule consequences).

φ \ x	S2	S1	CE	B1	B2
S3	S2	S3			
S2	S2	S3	S3	S3	
S1	B1	S1	S2	S3	S2
C3	B2	B2	CE	S2	S2
B1	B2	B3	B2	B1	S1
B2		B3	B3	B3	B2
B3				B3	B2

Figure 7.8 The entire rule base in a table representation. Each filled cell contains a rule consequence and corresponds to one rule. The cells enclosed in a heavier square indicate rules that at some time might be fired together, when the input is $\varphi(t) = 140$ and $x(t) = 6$ (compare highlighted triangles in Figure 7.7).

Notice that this example is rather special in some respects. First, a compact table representation as in Figure 7.8 is only possible with one or two input variables. Second, this control problem is quite "homogeneous" in the sense that the rules nicely cover a two-dimensional region of possible input value combinations. In more involved control tasks with dozens of inputs, the space of input value combinations necessarily will be filled only very sparsely with rules, because (i) only a small subset of input combinations will be deemed relevant by the expert, (ii) the number of possible input combinations explodes with the number of input variables, so it is simply not feasible to cover all combinations.

7.4.2 The fuzzyfier

A fuzzy rule needs a fuzzy set as an empirical input in order to be "fired" according to the composition mechanism exemplified in (7.31). However, the real-life input to a control box is not a fuzzy set but a "crisp" numerical input vector $\mathbf{y}(t)$ This must be translated into a fuzzy set first by assigning to it a membership function $\mu_{\mathbf{y}(t)}$. This is called *fuzzyfication.*

The standard choice is to use the singleton membership function

$$\mu_{\mathbf{y}(t)}(\mathbf{y}) = \begin{cases} 1, \text{ if } \mathbf{y}(t) = \mathbf{y} \\ 0, \text{ else.} \end{cases} \tag{7.41}$$

If the observation/measurement process that produces the numerical input $\mathbf{y}$ is considered noisy, another common choice for the fuzzyfication is to create $\mu_{\mathbf{y}(t)}$ as a truncated multi-dimensional Gaussian centered on $\mathbf{y}(t)$. Its covariance matrix should reflect the noise model that one assumes.

7.4.3 The inference algorithm

After fuzzyfication, for each input value $y(t)$ a membership function $\mu_{\mathrm{Y}(t)}$ is available inside the control box. If $\mu_{\mathbf{Y}^i \to \mathrm{U}^i} = \mu_{\mathbf{Y}^i} \star \mu_{\mathrm{U}^i}$ is the i-th rule of the rule set, where $\mathbf{Y}^i = \mathrm{Y}^i_1 \times ... \times \mathrm{Y}^i_m$ with domains of discourse $Y^i_1 \times ... \times Y^i_m$, then this rule can use the input membership functions $\mu_{\mathrm{Y}^i_1(t)},...,\mu_{\mathrm{Y}^i_m(t)}$ to fire according to

$$\begin{aligned} \mu_{\mathbf{Y}^i(t) \circ (\mathbf{Y}^i \to \mathrm{U}^i)}(u) &= \mu_{(\mathrm{Y}^i_1(t) \cap ... \cap \mathrm{Y}^i_m(t)) \circ ((\mathrm{Y}^i_1 \cap ... \cap \mathrm{Y}^i_m) \to \mathrm{U}^i)}(u) \\ &= \sup_{y_1 \in Y^i_1, ..., y_m \in Y^i_m} \mu_{\mathrm{Y}^i_1(t)}(y_1) \star ... \star \mu_{\mathrm{Y}^i_m(t)}(y_m) \star \mu_{\mathrm{Y}^i_1}(y_1) \star ... \star \mu_{\mathrm{Y}^i_m}(y_m) \star \mu_{\mathrm{U}^i}(u) \\ &=: \mu_{\mathrm{U}^i(t)}(u), \end{aligned} \tag{7.42}$$

yielding an output membership function $\mu_{\mathrm{U}^i(t)}$ on the universe of discourse U^i.

Now our rule base will typically contain several rules $\mu_{\mathbf{Y}^{i_1} \to \mathrm{U}^{i_1}},...,\mu_{\mathbf{Y}^{i_\kappa} \to \mathrm{U}^{i_\kappa}}$ with output terms $\mathrm{U}^{i_1},...,\mathrm{U}^{i_\kappa}$ on a shared universe of discourse U. In our trucker back upper task, for instance, *all*

rules share the same output universe. Most of the rules sharing a particular output universe will typically yield an all-zero membership function $\mu_{U^i(t)}$, because this already happens if one of the starred terms in (7.42) is zero, which is more likely than not. However, a small number of rules sharing an output universe U will yield non-zero $\mu_{U^i(t)}$. For instance, in the input situation highlighted in Figs. 7.7 and 7.8 there are 3 (out of 27) rules that yield a nonzero output $\mu_{U^i(t)}$. How is the information from these combined?

Again, there is nothing by way of theory which guides us to find the "right" way to combine the outputs of several rules that fired with nonzero output. According to the Mendel tutorial, one of the following methods is often used:

1. In order to combine the nonzero outputs $\mu_{U^{i_1}(t)},...,\mu_{U^{i_\kappa}(t)}$ into a single joint membership function $\mu_{U(t)}$, use the *t*-conorm:

   (7.43) $\mu_{U(t)} = \mu_{U^{i_1}(t)} \oplus ... \oplus \mu_{U^{i_\kappa}(t)}$.

2. Or, simply use straightforward algebraic addition:

   (7.44) $\mu_{U(t)} = \mu_{U^{i_1}(t)} + ... + \mu_{U^{i_\kappa}(t)}$

3. Or, use a weighted algebraic summation:

   (7.45) $\mu_{U(t)} = a_1 \mu_{U^{i_1}(t)} + ... + a_\kappa \mu_{U^{i_\kappa}(t)}$,

   where the weights are either set by hand, reflecting different degrees of confidence in the concerned rules, or else are trained from data with methods borrowed from neural network learning[49].

In the second and third case, notice that the resultant $\mu_{U(t)}$ does not range in [0,1] – the subsequent defuzzyfication must take care of that anomaly.

The combination method for joining the contributions from several fired rules is the core of the inference algorithm. In whatever way this is realized, the result is a single output membership function $\mu_{U(t)}$ for every output variable.

7.4.4 The defuzzifier

The final output of the control box cannot be membership functions $\mu_{U(t)}$ but must be "crisp" values $u(t)$ for the output variables, because only such values can be fed into the plant. This requires a *defuzzification* mechanism that transforms a 1-dimensional membership function $\mu_{U(t)}$ into a single real number $u(t)$. Again, we are confronted with a number of options. The following methods are often used:

[49] There are some control architectures that use a combination of a FL part, as outlined here, and a trainable neural network superstructure – these are *fuzzy neural networks*, on which some literature exists.

1. *Maximum defuzzifier*: $u(t) = \max_{u \in U} \mu_{U(t)}(u)$. This is a simple and natural choice, but it fails if $\mu_{U(t)}$ has no single maximum. This is likely to happen if some of the membership functions used in the rule base have trapezoid shape (like S2 in Fig. 7.7 b). An alternative that avoids this pitfall is the
2. *Mean of Maxima defuzzifier*: $u(t) = mean\{u \in U \mid u = \max_{v \in U} \mu_{U(t)}(v)\}$, or the
3. *Centroid defuzzifier*, which returns the center of gravity of $\mu_{U(t)}$:

$$u(t) = \frac{\int_U v \mu_{U(t)}(v)\, dv}{\int_U \mu_{U(t)}(v)\, dv} \tag{7.46}$$

The Mendel tutorial describes some more defuzzifiers.

7.5 Concluding remarks

We have gone through a condensed presentation of the essentials of FL. I would like to point out some omissions:

- FL is used in other contexts besides control engineering. One is the representation and learning of nonlinear input-output mappings for purposes of time series prediction. Here the methods resemble very much what we have seen in our little tutorial. Another field for FL is expert system like applications. Then the rules contained in the rule set of a "knowledge base" form more complicated interdependence patterns that are best described by (chaining) graphs, very much like what we have seen in Bayesian networks. The inference algorithms become more involved. It seems to me however that such applications are no longer pursued with the vigour that was originally applied to this field in the 1980'ies.
- Sometimes it is claimed that FL is useful for "reverse engineering" from numerical control systems or nonlinear functions back into a format that can be understood by humans. Consider some nonlinear function that can be empirically observed but is unintelligible to humans. One example would be to monitor a human control expert's actions, without interviewing him. Using machine learning methods, from the observed input-output data a FL representation with synthetically created membership functions can be distilled. These membership functions can then (hopefully) be interpreted in retrospect by a human looker-on. This is not as silly as it might at first appear because it is very difficult for the human expert (who produced the observable control action in the first place) to give a clear linguistic account of why he does what and when. Circumventing the interviewing stage by machine learning techniques is an interesting approach to the *knowledge acquisition* problem. It is also claimed that control (or other) systems based on FL rules are more easily checked for soundness, and are better maintainable, than purely "black box" numerical systems.
- An interesting field of investigations that receives constant (but not overwhelming) attention is the combination of FL rules with neural networks – *fuzzy neural networks.*

I would like to conclude with some take-home messages:

- FL is a method for formalizing "soft" expert knowledge and make it amenable to algorithmic inferences, mostly in control engineering applications.
- Fuzzy logicians spend considerable effort on justifying FL as a generalization of classical logic (and classical set theory). However, the classical theories enormously underspecify FL approaches, and there is currently no axiomatic basis for FL that can fill the justification gap. Thus FL has a very heuristic flavour.
- The most important concepts of FL are linguistic variables, linguistic terms, membership functions, rules, and the composition of relations, the latter giving rise to a mechanism to chain rules or "fire" them through observed input.
- Virtually all concrete instantiations of the basic FL concepts admit a host of alternatives. Thus there is no such thing as *the* FL system. In the Mendel tutorial, a quick estimate informs us that even if only the most common choices for the single operations are taken into account, by their combination one can easily create as many as 2^{15} different "fuzzy logic systems".

Appendix A. The Dirac delta function

We have used the Dirac delta δ variously in this lecture. It is a strange beast and has to be used carefully. It was originally introduced in physics in the early 20th century as a symbolic aid to come to terms with probability "point" distributions that concentrate the entire probability mass in the origin. Physicists are reckless warriors and they worked with the Dirac delta function as if it were a pdf. From this perspective, the defining property of δ is that

$$\text{(A.1)} \quad \int_{\Re^n} f(\mathbf{x})\delta(\mathbf{x})\, d\mathbf{x} = f(\mathbf{0}) .$$

However, this is a riddlesome definition, because it cannot be identified with any "classical" function $\delta: \mathbb{R}^n \to \mathbb{R}$, which is what other pdfs can. Therefore, one has to be careful when handling δ as a pdf; not everything that one can do with standard pdfs works with δ.

There appear two rigorous accounts of what type of mathematical thing δ actually is.

For mathematicians, the world is clear: they simply don't use δ. When mathematicians want to deal with probability distributions that are concentrated in the origin (or some other single point $\mathbf{x}_0$), they use a particular probability measure, the *point measure*, which is often denoted by $\varepsilon_{\mathbf{x}_0}$. It has the defining property that for all $A \in \mathcal{B}$, where $\mathcal{B}$ is the Borel σ-field on $\mathbb{R}^n$, it holds that $\varepsilon_{\mathbf{x}_0}(A) = 1$ iff $\mathbf{x}_0 \in A$, else $\varepsilon_{\mathbf{x}_0}(A) = 0$. Quoting (and translating) from my favourite textbook on mathematical probability theory, Heinz Bauer's *Measure and Integration Theory* (IRC QC20.7.M43 B4813 2001):

"Physicists occasionally work with such a "symbolic density" δ_x and call it the Dirac-function at x. The correct mathematical object however is $\varepsilon_{\mathbf{x}}$."

The price that mathematicians pay for their aristocratic attitude is that they can't tap from the notational conveniences of using pdfs, nor from the intuitions that are connected to pdfs (remark: in equation A.1, the Dirac delta appears like a pdf).

The other rigorous account is to view δ as a *distribution*. Here we meet a deplorable double use of a technical term. In probability theory, a distribution is the induced probability measure of a random variable in its measure space. Another (related but mathematically quite different) usage of the term has emerged in theoretical physics, mathematics and engineering in the last century, where in the *theory of distributions* the term *distribution* refers to mathematical objects that can be understood likewise as generalizations of (ordinary) functions and (ordinary) pdfs. A very good introduction to distributions in this sense is given in the Wikipedia article on DISTRIBUTIONS (MATHEMATICS). For simplicity, I copy here from that article (the remainder of this appendix is taken almost verbatim from Wikipedia).

In mathematical analysis, distributions (also known as generalized functions) are objects which generalize functions and probability distributions. They extend the concept of derivative to all integrable functions and beyond, and are used to formulate generalized solutions of partial differential equations. They are important in physics and engineering where many non-continuous problems naturally lead to differential equations whose solutions are distributions, such as the Dirac delta distribution.

The basic idea is to identify functions with abstract linear functionals on a space of unproblematic test functions (conventional and well-behaved functions). Operators on distributions can be understood by moving them to the test function.

For example, if

$$f : \mathbb{R} \to \mathbb{R}$$

is a locally integrable function, and

$$\varphi : \mathbb{R} \to \mathbb{R}$$

is a smooth (that is, infinitely differentiable) function with compact support (so, identically zero outside of some bounded set), then we set

$$\langle f, \varphi \rangle = \int_{\Re} f\varphi \, dx \, .$$

This is a real number which linearly and continuously depends on φ. One can therefore think of the function f as a continuous linear functional on the space which consists of all the "test functions" φ.

Similarly, if P is a probability distribution on the reals and φ is a test function, then

$$\langle P, \varphi \rangle = \int_{\Re} \varphi \, dP$$

is a real number that continuously and linearly depends on φ: probability distributions can thus also be viewed as continuous linear functionals on the space of test functions. This notion of "continuous linear functional on the space of test functions" is therefore used as the definition of a distribution. Such distributions may be multiplied with real numbers and can be added together, so they form a real vector space.

To define the derivative of a distribution, we first consider the case of a differentiable and integrable function $f : \mathbb{R} \to \mathbb{R}$. If φ is a test function, then we have

$$\int_{\Re} f' \varphi \, dx = -\int_{\Re} f\varphi' \, dx \, ,$$

using integration by parts (note that φ is zero outside of a bounded set and that therefore no boundary values have to be taken into account). This suggests that if S is a distribution, we should define its derivative S' by

$$\langle S', \varphi \rangle = -\langle S, \varphi' \rangle .$$

It turns out that this is the proper definition; it extends the ordinary definition of derivative, every distribution becomes infinitely differentiable and the usual properties of derivatives hold.

Example: The Dirac delta (so-called Dirac delta function) is the distribution defined by

$$\langle \delta, \varphi \rangle = \varphi(0).$$

It is the derivative of the Heaviside step function: For any test function φ,

$$\langle H', \varphi \rangle = -\langle H, \varphi' \rangle = -\int_{-\infty}^{\infty} H(x)\varphi'(x)dx = -\int_{0}^{\infty} \varphi'(x)dx = \varphi(0) = \langle \delta, \varphi \rangle,$$

so $H' = \delta$.

References

Bishop, Christopher M. (1995): Neural Networks for Pattern Recognition. Oxford Univ. Press, 1995. IRC: QA76.87 .B574 1995

Bishop, Christopher M. (2006): Pattern Recognition and Machine Learning. Springer Verlag, New York 2006

Farhang-Boroujeny, B. (1999): Adaptive Filters: Theory and Applications. Wiley and Sons 1999 (IRC: TK7872.F5 F37 1998)