Introduction to Bayesian Learning

Aaron Hertzmann
University of Toronto

Course Notes

Version of: September 15, 2004

Links, course slides, and updated course notes:
http://www.dgp.toronto.edu/~hertzman/ibl2004/

Contents

4

Chapter 1

Introduction

We live in an age of widespread exploration of art and communication using computer graphics and animation. Filmmakers, scientists, graphic designers, fine artists, and game designers, are finding new ways to communicate and new kinds of media to create. Computers and rendering software are now quite powerful. Arguably, the largest barrier to using digital media is not technological limitations, but the tedious effort required to create digital worlds and digital life. Suppose you wish to simulate life on the streets of New York in the 1930's, or a mysterious alien society. Someone has to actually create all the 3D models and textures for the world, physics for the objects, and behaviors or animations for the characters. Although tools exist for all of these tasks, the sheer scale of even the most prosaic world can require months or years of labor. An alternative approach is to create these models from existing data, either designed by artists or captured from the world. However, creating complex models from real-world data is often quite difficult — it is often difficult to describe the fitting problem mathematically, especially when trying to estimate high-level models that produce the data through indirect or complex mechanisms. For example, extracting character shape or behavior from raw video sequences requires accounting for many sources of variability and noise.

Fortunately, a methodology known as **Bayesian reasoning** provides a unified and natural approach to many difficult data-modeling problems. Bayesian reasoning is, at heart, a model for **logic in the presence of uncertainty.** Bayesian methods match human intuition very closely, and even provides a promising model for low-level neurological processes (such as human vision). The mathematical foundations of Bayesian reasoning are at least 100 years old, and have become widely-used in many areas of science and engineering, such as astronomy, geology, and electrical engineering. Closer to home, Bayesian learning techniques are prominant in fields such as computer vision, bioinformatics, and natural language processing (including, but not limited to, spam filtering).

In these course notes, I will first argue that fitting models from data can be very useful for computer graphics, and that Bayesian machine learning can provide powerful tools. I will attempt to address some of the common concerns of this approach, and discuss the pros and cons of Bayesian modeling, and briefly discuss the relation to non-Bayesian machine learning. I will also provide a brief tutorial on probabilistic reasoning.

Bayesian reasoning provides three main benefits:

1. Principled modeling of uncertainty
2. General purpose models for unstructured data

3. Effective algorithms for data fitting and analysis under uncertainty

I will give simple but detailed examples later on. Of the existing graphics research that uses machine learning, most of them use existing these methods as "black boxes." I advocate modeling the entire system within a Bayesian framework, which requires more understanding of Bayesian learning, but yields much more powerful and effective algorithms.

There are also many useful non-probabilistic techniques in the learning literature as well. I put more emphasis on probabilistic methods, since I believe these have the most value for graphics.

Why data-driven graphics? Consider the problem of creating motions for a character in a movie. You could create the motions procedurally, i.e. by designing some algorithm that synthesizes motions. However, synthetic motions typically look very artificial and lack the essential style of a character, and pure procedural synthesis is rarely used for animation in practice. More commonly, one animates characters "by hand," or captures motions from an actor in a studio. These "pure data" approaches give the highest quality motions, but at substantial cost in time and the efforts of artists or actors. Moreover, there is little flexibility: if you discover that you did not get just the right motions in the studio, then you have to go back and capture more. The situation is worse for a video game, where one must capture all motions that might conceivably be needed.

Learning techniques promise the best of both worlds: starting from some captured data, we can procedurally synthesize more data in the style of the original. Moreover, we can constrain the synthetic data, for example, according to the requirements of an artist. Of course, we must begin with some data produced by an artist or a capture session. But now, we can do much more with this data than just simple playback. For these problems, machine learning offers an attractive set of tools for modeling the patterns of data.

Data-driven techniques have shown a small but steadily increasing presence in graphics research. Principal components analysis and basic clustering algorithms are becoming almost *de rigueur* at SIGGRAPH. Most significantly, the recent proliferation of papers on texture synthesis and motion texture suggests a growing acceptance of learning techniques. However, the acceptance of these works relies largely on their accessibility — one does not need to know much about machine learning to implement texture synthesis. In this article, I will argue that graphics can benefit more deeply from the learning literature.

1.1 Learning and graphics today

These notes are written primarily for computer graphics researchers and practitioners developing new algorithms. There seems to be some resistance to machine learning in the graphics community. For example, one researcher said to me a few years ago:

> Graphics researchers are already starting to grumble about "machine learning doesn't really work, maybe only on the two examples shown in the paper." ... Without commenting on whether this might actually be true or not, I just want to note that I've heard increasing amounts of this sentiment.

On the other hand, the graphics community is very diverse, and cannot be summarized by a single sensibility. A computer vision researcher with an electrical engineering background doubts there is any real controversy:

> I am surprised that you think there's so much resistance to machine learning (even the term)! ... Certainly in EE or other branches of CS (e.g. database query, robotics, etc.), "machine learning" is almost a classical term, and covers an amazing amount of territory. No one would be shocked by someone using "machine learning" in any of the literature I read ...

In fact, Bayesian methods are quite standard in computer vision, going back to the early work of Geman and Geman [1984] and Szeliski [1989]. However, my general impression of the attitude of the graphics community (which could be wrong) is of a mixture of curiosity with deep skepticism. In my opinion, insofar as there is any resistance, this resistance stems from misconceptions about learning and its application. At one time I expected "learning" to be a magical black box that discovers the meaning of some raw data, and I suspect that others expect the same. This promise naturally breeds skepticism. Although this black box does not currently exist, machine learning research has been very fertile in many domains, even without solving the AI problem. It is a truism that artificial intelligence research can never become successful, because its successes are not viewed as AI. Recent successes include work in bioinformatics, data mining, robotics, computer vision, spam filters, and medical diagnoses. For the reader who is bothered by the term "machine learning," I suggest mentally substituting the phrase "statistical data fitting" instead.

I truly believe that current machine learning research and neuroscience research are on the verge of understanding how the brain works. However, this is still conjecture, and one does not need to believe this to see the power of Bayesian methods.

Moreover, data-fitting techniques are widely used in graphics — whether one is fitting a 3D surface to a point cloud obtained from a laser range scanner, or fitting a spline to a user-drawn curve, or fitting a Mixtures-of-Gaussians (MoG) model to motion capture data, one is fitting a structured model to observed data. It should be noted that the MoG model is a direct generalization of vector quantization, which is already widely used in graphics. Similarly, one may think of a Hidden Markov Model (HMM) as a probabilistic generalization of vector quantization that models temporal coherence.

One may also object to learning techniques because they take away control from the artist — but this is really a complaint about all procedural techniques. In my opinion, the goal of procedural techniques is not to replace the artist, but to provide effective high-level tools. Data-driven methods give the artist the ability to build from captured data, and the ability to design styles by example rather than by setting thousands of parameters manually.

1.2 What is machine learning?

For the purposes of computer graphics, machine learning should really be viewed as a set of techniques for leveraging data. Given some data, we can model the process that generated the data. Then, we can make more data that is consistent with this process, possibly with new, user-defined constraints.

In learning, we combine our prior knowledge of the problem with the information in the training data; the model that we fit should be carefully chosen. On one hand, trying to model everything about the world — such as the exact shape and dynamics of the muscle tissue in a human actor and the actor's complete mental state — would be hopeless. Instead, we must fit simpler models of observed data, say, of the movements of markers or handles; the parameters of this model will rarely have any direct interpretation in terms of physical parameters. On the other hand, choosing features that are too general may make learning require

far too much data. For example, Blanz and Vetter [1999] modeled the distribution of possible faces and expressions with a Gaussian probability density function. Such a weak model allowed them to model patterns in the data without requiring explicit *a priori* understanding of them. They can then generate new faces and expressions by sampling from this density, or by estimating the most likely pose that matches some input photograph. However, they did need to represent face data using training data in correspondence; directly learning a model from range data not in correspondence would not be likely to work very well at all. At present, learning algorithms do not perform magic: you must know something about the problem you want to model. As a rule of thumb, the less information you specify in advance, the more training data you will need in order to train a good model. It is very difficult to get good results without having some high-level understanding of how the model operates. The main benefit is that we can still get good results with fairly high-level models.

1.3 What does learning have to offer?

The idea of driving graphics from data is hardly new, and one can build some models from data without knowing anything about machine learning. One could argue that the word "learning" should be avoided in computer graphics since it leads to the sort of unrealistic expectations mentioned above. However, I believe that using the tools, terminology, and experience of the machine learning community offer many benefits to computer graphics research and practice. By employing existing ideas and techniques, we get the benefit of the collective experience of the researchers who studied these problems in the past. Otherwise, we will waste substantial effort reinventing the wheel. The literature provides many intellectual tools that can be applied again and again. For example, the authors of the Composable Controllers paper from SIGGRAPH 2001 [2001] sought an algorithm to classify data based on some training examples. Rather than attempting to solve the classification problem from scratch, the authors simply used Support Vector Machines (SVMs), a state-of-the-art classification procedure that has consistently outperformed competing methods (including neural networks, and, in this case, nearest-neighbors). In fact, they did not even have to implement an SVM classifier; instead, they downloaded one from the web. This kind of reuse illustrates how accessing existing research can save us from having to reinvent (and reimplement) the wheel. Moreover, it is unlikely that anyone would casually invent a technique as effective as SVMs in the course of conducting a larger project.

In general, the machine learning literature and community have much to offer graphics:

Problem taxonomy. The literature makes a distinction between types of problems, such as density estimation, classification, regression, and reinforcement learning. See Section **??** for more detail. Understanding these distinctions helps one understand a new problem and relate it to existing approaches. For example, the authors of the Video Textures [Schödl et al. 2000] identified their synthesis problem as a reinforcement learning problem, which allowed them to immediately draw on existing solutions rather than to attempt to solve the problem from scratch.

General-purpose models. Machine learning researchers have developed many models for learning structure in arbitrary data. For many fitting problems, it is likely that one of these methods will be useful, either as a complete model or as a starting point for a problem-specific model. Many of these methods are outlined in the next section.

Reasoning with probabilities. One of the major trends in learning research is to reason with probabilities, in order to model the uncertainty present in all of our models and data. Probabilistic modeling provides a very powerful, general-purpose tool for expressing relative certainty in our understanding of the world. Often, one source of information will be more reliable than another, and we must weigh the reliability of data along with the data itself when making estimates or decisions; probability theory provides a principled mechanism for reasoning with uncertainty, learning from data, and generating new data (e.g. by sampling from a learned model). Machine learning researchers have developed (or adapted from other disciplines) many powerful tools for statistical reasoning, such as Expectation-Maximization, Belief Propagation, Markov Chain Monte Carlo methods, and Particle Filtering. Although probabilistic reasoning is not necessary for every problem (and it will always be dependent on some *a priori* assumptions that we make about the world), it has been shown to be a very powerful tool in many situations. Some cognitive science researchers even believe that the human brain can be viewed as performing probabilistic inference, at least in low-level processes [Rao et al. 2002].

A few papers in graphics have used techniques from learning in interesting ways. Of these few papers, most of them use an existing learning technique as a "black box" subroutine. While these uses are exciting, we have yet to see much work that does not just reuse models but tightly fits them into a graphics system. In contrast, the interaction of learning and computer vision is much more mature. In much computer vision research, there is no "learning subroutine," but a unified system that completely models the process being analyzed. For example, Jojic and Frey's video processing algorithms extract sprites and solve for all relevant parameters in a unified probabilistic framework [2001]; similarly, our recent work on modeling non-rigid shape from video does not require any tuning parameters [Torresani and Hertzmann 2004].

Incidentally, probabilistic methods can be useful in graphics entirely separate from data-driven techniques, as argued convincingly by several authors [Barzel et al. 1996; Chenney and Forsyth 2000; Perlin 1985; Perlin and Goldberg 1996]. For digital actors and behaviors, it is important that the animation is not the same every time. Probabilistic models allow multiple solutions to a problem, and can model random subtleties for which an exact model is impractical. (Probabilistic methods have long been used in global illumination, but only as part of numerical integration techniques, not to represent uncertainty in the scene).

Some probabilistic methods yields simple least-squares formulations. In the past, this has been a source of contention in the computer vision community. Some people argue that, if one can pose the problem simply as a simple least-squares fitting problem (as one does with regression methods, such as radial basis functions and neural networks) then there is no need for probabilistic methods. On a theoretical level, it is in fact the case that least-squares fitting itself is a statistical method, assuming a specific noise distribution [Jaynes 2003; Sorenson 1970]. On a more practical level, I would certainly agree that a simple least-squares formulation may sometimes be adequate; however, if the problem involves weighing between multiple terms, thresholding, and/or estimating terms that have very different meanings, then generally a probabilistic technique will be necessary in order to fit these parameters. Posing the probabilistic model explicitly can lead to insights about how better to handle unknowns that would otherwise be treated in a clumsy, *ad hoc* manner. For example, linear regression can be posed in a least-squares setting, and there is no real need to state an explicit noise model. Linear regression where both variables are corrupted by noise, linear regression that is robust to outliers, and linear fitting with missing data each would require either (a) parameter tuning by the user, or (b) probabilistic methods that can learn the noise and outlier models.

Note that there is occasionally some confusion in that the probabilistic methods mentioned here are *not* necessarily randomized algorithms. Probabilistic methods model uncertainty, but often involve deterministic

operations. Randomized algorithms may be used in optimization, and random sampling may be used to draw from a distribution.

Learning all the parameters. Most computer graphics systems (including many current data-driven algorithms) have many parameters to tune. (This fact that is often mentioned in paper reviews; it is a very safe thing to comment on). Bayesian reasoning provides ways to fit energy functions to data, even energy functions that are too complicated to fit by hand. Moreover, I will go on a limb here and say that machine learning systems can learn *all* of the parameters of a model. There are a few caveats: you must choose a model that is suitably powerful for the problem you wish to solve, there must be enough training data, and you must be willing to perform a potentially slow optimization procedure. Probabilistic modeling provides a principled problem formulation for learning all the parameters, although optimizing the resulting objective function may be difficult for certain types of parameters. However, there is flexibility — a good model with few parameters needs less training data and time than a weak model with many parameters. In practice, one will generally specify a few parameters of the model that are difficult to learn (e.g. model dimensionality), and have the algorithm learn the rest (including noise values, outlier thresholds, data labeling, and so on). Of course, if more user control is desired, then one may allow some of the parameters to be specified by hand.

For many graphics applications, the learning process may be viewed as **learning the objective function** for procedural synthesis. Objective functions and energy functions are widely used throughout computer graphics; one typically synthesizes data by optimizing an energy function subject to some constraints. For example, geometric models are often created by optimizing a specific objective function (sometimes implicitly). Instead of designing this objective function by hand, we could use machine learning methods to create the objective function from data — or, more precisely, to fit the parameters of an objective function. Synthesis is then a matter of optimizing with respect to this objective. In a sense, the learned objective function measures the similarity of the synthesized motion to the examples, but in a much more general way. See Section **??** for a detailed example.

1.4 Frequentist statistics

Bayesian reasoning is one of the main approaches to statistics, the other is known as Frequentist (or "orthodox") statistics. Frequentist statistics defines probabilities in terms of frequencies of repeated events, and is presently more prevalent among many of the natural sciences than Bayesian methods. Frequentist statistics is more commonly taught in undergraduate science programs. (If you learn statistics in an undergrad statistics class or in a biology, chemistry, or medicine program, you probably learn frequentist statistics; if you learn statistics in an electrical engineering program, then you probably learn some form of Bayesian statistics.)

In contrast to the Bayesian view, frequentist statistics defines probabilities in terms of repeated events. For example, suppose we flip a coin many times; what proportion of those trials will land heads? In the frequentist view, the probability of heads is defined as the limit of this ratio as the number of trials goes to infinity; one generally assumes absolute certainty about the other variables in the experiment. This definition of probability has had success in areas where repeated trials are possible, such as in biology and chemistry, where one can perform thousands of repeated tests on chemicals or plants. However, in cases

where one cannot repeat trials, the frequentist view is useless for modeling uncertainty. For example, we make judgements based on meeting someone for the first time despite not having thousands of interactions; similarly, in graphics, we would like to synthesize data from small amounts of user input.

There is often confusion in the distinction between Bayesian and Frequentist statistics, since they yield the same estimates in some very simple cases. However, in most nontrivial examples, frequentist methods provide relatively little value — I will give many examples of dealing with unreliable data, and small amounts of data, in which Frequentist methods are unusable. For an entertaining (though one-sided) history of the debates between Bayesian and Frequentist statistics, see Jaynes [2003] (Chapter 16). Minka [2001] and Jaynes [2003] give concrete examples of simple cases in which frequentist methods give nonsensical answers (the pathologies that I describe about estimators in Chapter **??** apply to Frequentist methods in general).

Although Bayesian methods are very strong in the learning literature, frequentist methods remain widely studied, usually under the name Computational Learnining Theory. One of the most successful results in this field is the Support Vector Machine (SVM) architecture[1]. However, these methods are primarily concered with classification, which, in my opinion, is of much less interest for graphics applications than is density modeling.

In many of the sciences, I am of the impression that frequentist methods remain dominant, as evidenced by the preeminence of frequentist significance testing. However, this is slowly changing.

If, like many people, you always found probability and statistics to be a dry and tedious subject; I offer my own experience. When I took undergraduate statistics (which focused primarily on frequentist methods), I found it to be dry and tedious, concerned mainly with memorizing dozens of seemingly-arbitrary estimators and signficance tests. Much later, when I began to study on my own the principles of Bayesian methods, everything became very simple and clear, founded on a few simple, intuitive rules.

1.5 Different uses of the word "Bayesian"

The word "Bayesian" seems to mean slightly different things to different communities. I follow the common usage of it as being a philosophy for probabilistic reasoning, which is most in line with the usage in the machine learning and statistics community. Here is a discussion of some of the technical differences, which can safely be skipped (until you start to hear conflicting usages).

In computer vision paper titles, the word typically refers to the use of a probabilistic formulation (e.g., [Szeliski 1989]), in which MAP/ML estimation is then performed (Chapter **??**). However, estimation is not strictly Bayesian [Neal 1996], and many papers in the machine learning literature use the word "Bayesian" to specifically refer to algorithms that perform "Bayesian prediction," as discussed in Chapter **??**, as opposed to estimation. On the other hand, the electrical engineering community uses the word Bayesian to mean something very different, specifically the use of specific forms of the MAP rule (e.g., [Poor 1994]). In statistics, it is sometimes claimed (incorrectly) that the defining feature of Bayesian methods that distinguishes them from other methods is the use of priors.

[1] Although there are Bayesian derivations of SVMs [Chu et al. 2003; Sollich 2002; Tipping 2001] which provide many advantages over the frequentist version.

1.6 About these notes

These notes are not finished, and may contain some errors. I may revise and expand these notes in the future. Please send me feedback if you find them useful (as this will help motivate me to revise them), or if you have suggestions.

Throughout these notes, I will emphasize the main points that you should remember:

> **Key point:**
>
> Main principles are highlighted in boxes.

Optional sections (sections that provide additional background or examples and can be safely skipped) are marked with a star ($\star$).

Chapter 2

Overview: learning problems and motivating examples

Let's begin by looking at a few typical data-fitting problems, in order to provide concrete motivation and goals for the following discussion.

2.1 Supervised learning

One common problem is known as **regression**, in which we are given N pairs of training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, and we assume that there exists a functional mapping f, so that $\mathbf{y} = f(\mathbf{x})$. This is a standard problem in numerical analysis and statistics. A common approach is to assume some functional form for f, such as a linear combination of K basis functions $B_i(\mathbf{x})$:

$$f(\mathbf{x}; \mathbf{w}) = \sum_{k=1}^{K} w_k B_k(\mathbf{x}) \tag{2.1}$$

The curve is parameterized by weights $\mathbf{w} = [w_1, ..., w_K]$ that must be estimated from the data, and the basis function $B_i(\mathbf{x})$ may be, for example, polynomials or spline basis functions. Once we have the weights $\mathbf{w}$, we can generate a new $\mathbf{y}$ for any new vector $\mathbf{x}$. How do we estimate $\mathbf{w}$ from the training data? If the measurements are free from noise, and the correct basis functions are known, then the weights may be found by closed-form solution. However, this is almost never the case is real-world data modeling problems.

In computer graphics (and many other fields), the most common way to estimate the weights is by **least-squares fitting**, in which we define an objective function:

$$E_1(\mathbf{w}) = \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \mathbf{w})||^2 \tag{2.2}$$

By choosing $\mathbf{w}$ to minimize this objective function, we will get a function f that fits the data very well. Unfortunately, there is no guarantee that we will get a good curve f — as illustrated in Figure 2.1(a), the curve might fit the data very well, but look extremely convoluted elsewhere. In general, this problem is

known as **overfitting** — overfitting occurs when the model *fits the training data very well, but does not generalize well to new data*. Intuitively, we might say that the result is undesirable because we think that curve should be smooth. We can make this assumption explicit by restricting the fitting to a small number of smooth basis functions; however, this very much limits the types of curves we can fit. Alternatively, we can add a second penalty term to the objective function:

$$E_2(\mathbf{w}) = c_0 \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \mathbf{w})||^2 + c_1 \int ||\nabla f||^2 d\mathbf{x} \tag{2.3}$$

The second term penalizes highly curved functions. This this objective function can sometimes be optimized analytically; alternatively, we can replace the smoothness term with a numerical approximation:

$$E_3(\mathbf{w}) = c_0 \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \mathbf{w})||^2 + c_1 \sum_{j=1}^{J} ||\nabla f|_{\mathbf{x}_j}||^2 \tag{2.4}$$

where the smoothness is evaluated at J sample point. The gradients $\nabla f|_{\mathbf{x}_j}$ may be estimated numerically or analytically. This may require a large number of sample points $\mathbf{x}_j$. Alternatively, we can observe that, if the basis functions are smooth, then then highly-curved functions can only be obtained with weights $\mathbf{w}$ that have large absolute values. Hence, an alternative choice of objective function is:

$$E_3(\mathbf{w}) = c_0 \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \mathbf{w})||^2 + c_1 ||\mathbf{w}||^2 \tag{2.5}$$

The $||\mathbf{w}||^2$ term is called a **weight decay** term. Regardless of the formulation that we choose, we still need to make a few important choices:

- What functional form of f should we use?
- If we use basis functions $B_i(\mathbf{x})$, how many should we use?
- What weights c_0 and c_1 should we use?

As shown in Figure 2.1, the fitting result will still be very sensitive to how we make these choices. Moreover, the best choices for these questions will be different for different problems and even for different data sets. As we shall see later in these notes, Bayesian methods give principled and effective methods for making all of the choices. In fact, we shall see that the best methods eliminate these choices entirely. Figure 2.2 illustrates a Bayesian regression algorithm that automatically estimates all unknown parameters and yields excellent fits. Although this is a relatively simple 1D example, Bayesian methods scale to problem with much larger sets of unknown parameters, in which manual tuning is exceedingly difficult to do well.

An additional question that we might ask is: why least-squares fitting? Why not use, for example, an L1-norm, or L3-norm, or some other objective function? While there is practical motivation for the L2-norm (namely, its reltaive ease of use and success in a wide variety of applications), one must look to statistical models to find a principled justification for the L2-norm, and to gain intuitions as to when it should or should not be applicable. In fact, the theory of least-squares estimation was invented by Gauss and Legendre based on probabilistc models [Sorenson 1970].

The regression problem is a form of **supervised** learning, because training pairs $(\mathbf{x}, \mathbf{y})$ are provided. An important special case of supervised learning is **classification**, in which we wish to separate the data into two discrete classes, usually represented by $\mathbf{y} = -1$ and $\mathbf{y} = 1$.

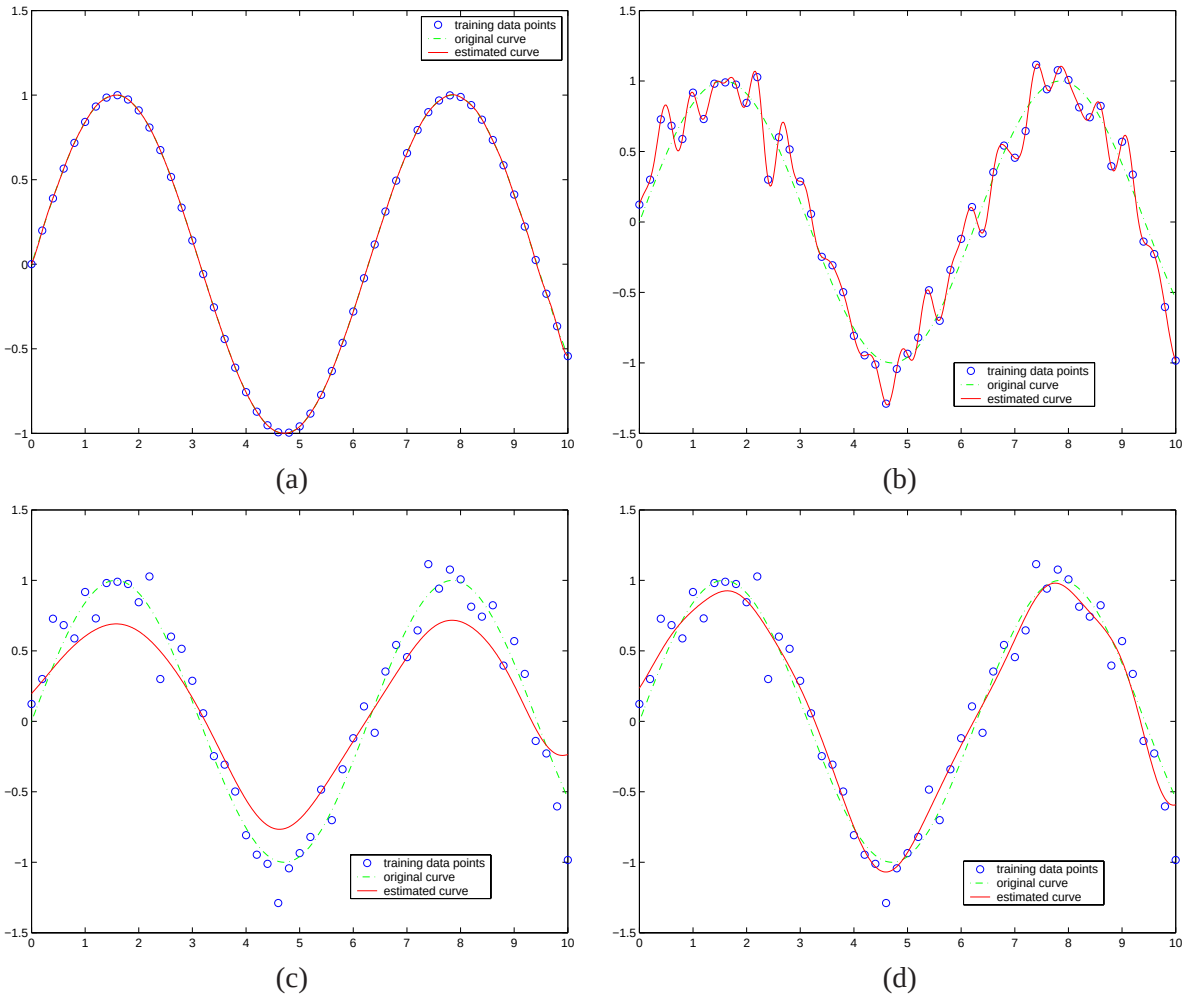

Figure 2.1: Least-squares curve fitting. (a) Point data (blue circles) was taken from a sine curve, and a curve was fit to the points by a least-squares fit (i.e., optimizing Equation 2.2). The horizontal axis is $\mathbf{x}$, the vertical axis is $\mathbf{y}$, and the red curve is the estimated $f(\mathbf{x})$. In this case, the fit is essentially perfect. The curve representation is a sum of Gaussian basis functions. (b) Overfitting. Random noise was added to the data points, and the curve was fit again. The curve exactly fits the data points, which does not reproduce the original curve (a green, dashed line) very well. (c) Underfitting. Adding a smoothness term makes the resulting curve too smooth. (In this case, weight decay was used, along with reducing the number of basis functions). (d) Reducing the strength of the smoothness term yields a better fit, but requires careful manual tuning, and may be very difficult to get right.

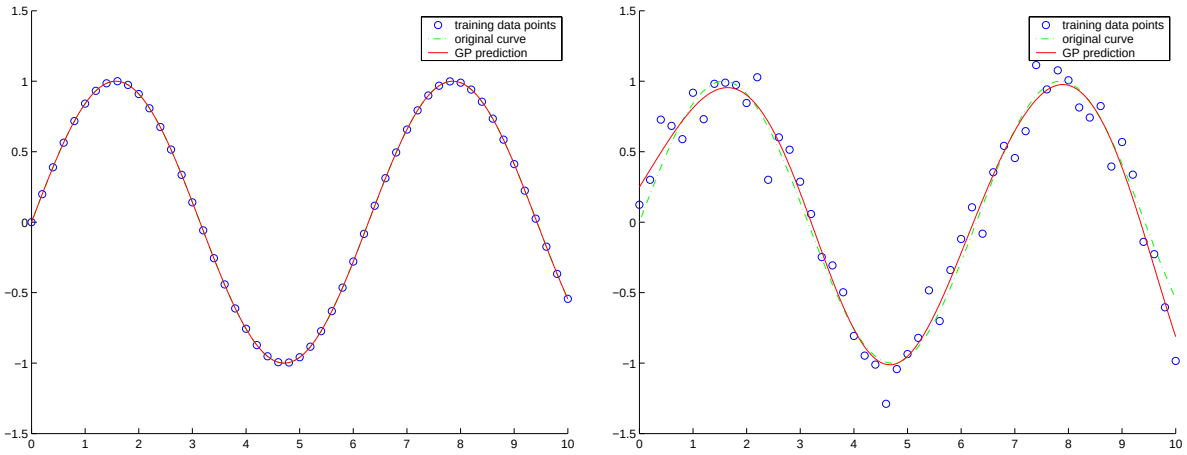

Figure 2.2: Regression with Gaussian Processes (GPs), described in more detail in Chapter **??**. Unlike the least-squares method, no manual parameter tuning is required — smoothness and noise terms are estimated automatically. (a) GP fit to the same data as in Figure 2.2(a). As before, the fit is exact. (b) GP fit to the same data as in Figure 2.2(b,c,d). The GP curve is very close to the original curve.

2.2 Unsupervised learning

Another important branch in learning is **unsupervised** learning, in which case we are only given the training data $\{\mathbf{y}_i\}$, and we wish to generalize from it, i.e., describe which values of $\{\mathbf{y}_i\}$ are likely, and which ones are unlikely. For example, suppose we flip a coin 100 times — how likely are heads versus how likely are tails?

A more practical example comes from our recent work in learning distributions over body poses [Grochow et al. 2004]. In this case, the training data consists of a set of 3D body poses, each one represented by a vector of joint angles $\{\mathbf{y}_i\}$. Our goal is to learn which poses are most "likely" (Figure 2.3). Once we have a likelihood function over poses, we can use it for inference tasks, such as filling in poses from incomplete data.

As we shall see, the same basic principles will be used in both the supervised and unsupervised cases. (There is no real conceptual divide between supervised and unsupervised learning; like much terminology in learning, the two terms are rather loosely defined).

TODO: graphics/vision examples: animation, faces, bodies, marker matching, ...

2.3 Reinforcement learning

The third major branch in learning is **reinforcement learning**, learning decision-making policies for agents. See [Kaelbling et al. 1996; Sutton and Barto 1998] for more on reinforcement learning.

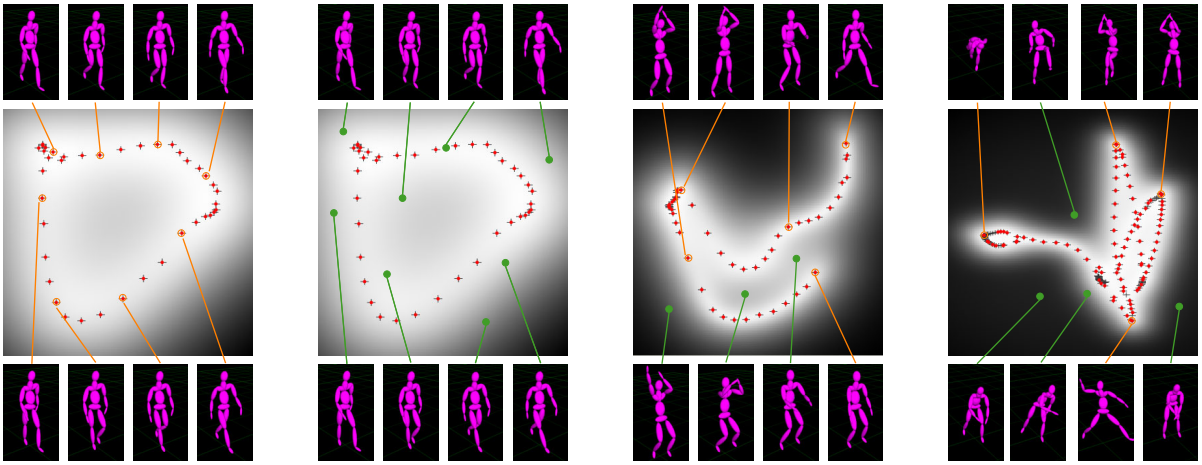

Figure 2.3: Unsupervised learning from different motion capture sequences: a walk cycle, a jump shot, and a baseball pitch (from [Grochow et al. 2004]). Each red dot corresponds to a full-body pose. The grayscale plot corresponds to a likelihood function over poses — poses near the original training data are most likely, but the likelihood function is smooth. (In this case, the 2D parameterization was also learned by the algorithm).

14

Chapter 3

Fundamentals of Bayesian reasoning

"Probability theory is nothing but common sense reduced to calculation." — Pierre-Simon Laplace [1814]

Bayesian probability theory addresses the following fundamental question: *how do we reason?* Reasoning is central to many areas of human endeavor, including philosophy (what is the best way to make decisions?), cognitive science (how does the mind work?), artificial intelligence (how do we build reasoning machines?), and science (how do we test and develop theories based on experimental data?). In nearly all real-world situations, our data and knowledge about the world is incomplete, indirect, and noisy; hence, uncertainty must be a fundamental part of our decision-making process. Bayesian reasoning provides a formal and consistent way to reasoning in the presence of uncertainty; probability theory is an embodiment of common sense reasoning.

In this section, I will give an overview of some basic concepts of probabilistic reasoning and learning. I will then show a few basic examples to illustrate these concepts.

3.1 Classical logic

Perhaps the most famous attempt to describe a formal system of reasoning is classical logic, originally developed by Aristotle. In classical logic, we have a number of statements that may be true or false, and we have a set of rules which allow us to determine the truth or falsity of new statements. For example, suppose we introduce two statements, named $\mathbf{A}$ and $\mathbf{B}$:

$\mathbf{A} \equiv$ "My car was stolen"
$\mathbf{B} \equiv$ "My car is not in the parking spot where I remember leaving it"

Moreover, let us assert the rule "$\mathbf{A}$ implies $\mathbf{B}$", which we will write as $\mathbf{A} \rightarrow \mathbf{B}$. Then, if $\mathbf{A}$ is known to be true, we may deduce logically that $\mathbf{B}$ must also be true (if my car is stolen then it won't be in the parking spot where I left it). Alternatively, if I find my car where I left it ("$\mathbf{A}$ is false," written $\bar{\mathbf{A}}$), then I may infer that it was not stolen ($\bar{\mathbf{B}}$) by the contrapositive $\bar{\mathbf{B}} \rightarrow \bar{\mathbf{A}}$.

Classical logic provides a model of how humans might reason, and a model of how we might build an "intelligent" computer. Unfortunately, classical logic has a significant shortcoming: it assumes that all

knowledge is absolute. Logic requires that we know some facts about the world with absolute certainty, and, then, we may deduce only those facts which must follow with aboslute certainty.

In the real world, there are almost no facts that we know with absolute certainty — most of what we know about the world we acquire indirectly, though our five senses, or from dialogue with other people. In other words, most of what we know about the world is **uncertain**.

For example, suppose I discover that my car is not where I remember left it ($\mathbf{B}$). Does this mean that it was stolen? No, there are many other explanations — maybe I remember wrong or maybe it was towed. However, the knowledge of $\mathbf{B}$ makes $\mathbf{A}$ more *plausible* — even though I do not know it to be stolen, it becomes more likely a scenario than before. The actual degree of plausibility depends on other contextual information — did I park it in a safe neighborhood, did I park it in a handicapped zone, etc.

Predicting the weather is another task that requires reasoning with uncertain information. While we can make some predictions with great confidence (e.g. we can reliably predict that it will not snow in June, north of the equator), we are often faced with much more difficult questions (will it rain today?) which we must infer from unreliable sources of information (e.g., the weather report, clouds in the sky, yesterday's weather, etc.). In the end, we usually cannot determine for certain whether it will rain, but we do get a degree of certainty upon which to base decisions and decide whether or not to carry an umbrella.

Another important example of uncertain reasoning occurs whenever you meet someone new — at this time, you immediately make hundreds of inferences (mostly unconscious) about who this person is and what their emotions and goals are. You make these decisions based on the person's appearance, they way they are dressed, their facial expressions, their actions, the context in which you meet, and what you have learned from previous experience with other people. Of course, you have no conclusive basis for forming opinions (e.g., the panhandler you meet on the street may be a method actor preparing for a role). However, we need to be able to make judgements about other people based on incomplete information; otherwise, normal interpersonal interaction would be impossible (e.g., how do you really *know* that everyone isn't out to get you?).

3.2 Towards a logic of uncertainty

What we need is a way of discussing not just true or false statements, but statements that have different levels of certainty, statements in which we have varying degrees of **belief.** In addition to defining such statements, we would like to be able to use our beliefs to reason about the world and interpret it. As we gain new information, our beliefs should change to reflect our greater knowledge. For example, for any two propositions $\mathbf{A}$ and $\mathbf{B}$ (that may be true or false), if $\mathbf{A} \rightarrow \mathbf{B}$, then strong belief in $\mathbf{A}$ should increase our belief in $\mathbf{B}$. Moreover, strong belief in $\mathbf{B}$ may sometimes increase our belief in $\mathbf{A}$ as well.

Let us now imagine devising a set of rules for reasoning with uncertainty. We'll define $B(\mathbf{A})$ to be our "belief" in $\mathbf{A}$, defined as some value that expresses our certainty that $\mathbf{A}$ is true, and $B(\mathbf{A}|\mathbf{B})$ to be what our certainty would be that $\mathbf{A}$ is true, *if* we knew $\mathbf{B}$ to be true. $B(\mathbf{A} \wedge \mathbf{B})$ will denote our certainty that both $\mathbf{A}$ and $\mathbf{B}$ are true.

Additionally, we would like our reasoning system to obey rules of common sense. For example, we would like our logical system to reduce to classical logic in the special case where all propositions are known with certainty (e.g., if $\mathbf{A} \rightarrow \mathbf{B}$, then absolute certainty in $\mathbf{A}$ would lead to absolute certainty in $\mathbf{B}$).

Such a system has been devised, in fact, by Richard T. Cox [1946]. Specifically. he sought a system for

beliefs that would qualitatively match common sense, but also be consistent (i.e., any two derivations for the certainty of a statement should yield the same value). From these desires, he asserted the following rules, known as the **Cox Axioms**:

1. Degrees of certainty can be ordered. If $B(\mathbf{A}) > B(\mathbf{B})$, and $B(\mathbf{B}) > B(\mathbf{C})$, then $B(\mathbf{A}) > B(\mathbf{C})$. A direct consequence of this assumption is that beliefs can be mapped onto the real numbers.

2. There exists a function f that maps the certainty of a statement to its negation. In other words, given $B(\mathbf{A})$, you can compute $B(\bar{\mathbf{A}}) = f(B(\mathbf{A}))$.

3. The degree of belief $B(\mathbf{A} \wedge \mathbf{B})$ is related to the conditional belief $B(\mathbf{A}|\mathbf{B})$ and $B(\mathbf{B})$ by some function g. Specifically, there exists a function g such that $B(\mathbf{A} \wedge \mathbf{B}) = g(B(\mathbf{A}|\mathbf{B}), B(\mathbf{B}))$.

The amazing fact is that these simple assumptions are enough to derive a complete system for reasoning with uncertainty: this is probability theory. Moreover, this system is unique — any reasoning system that is consistent with the above axioms must be equivalent to probability theory; any system that violates them must also violate common sense reasoning. If you have trouble believing any of this, then I highly encourage reading through the first two chapters of [Jaynes 2003].

Key point:

Probability theory is a quantative expression of common-sense reasoning.

Henceforth, I will refer to a belief as a probability, and proceed to enumerate the rules that can be derived from the Cox Axioms.

3.3 Basic definitions and rules

We can now state the basic rules of probability theory, all of which can be derived from the Cox Axioms.

- The probability of a statement $\mathbf{A}$ — denoted $P(\mathbf{A})$ — is a real number between 0 and 1, inclusive. $P(\mathbf{A}) = 1$ indicates absolute certainty that $\mathbf{A}$ is true, $P(\mathbf{A}) = 0$ indicates absolute certainty that $\mathbf{A}$ is false, and values between 0 and 1 correspond to varying degrees of certainty.

- The **joint probability** of two statements $\mathbf{A}$ and $\mathbf{B}$ — denoted $P(\mathbf{A}, \mathbf{B})$ — is the probability that both statements are true. (i.e., the probability that the statement "$\mathbf{A} \wedge \mathbf{B}$" is true). (Clearly, $P(\mathbf{A}, \mathbf{B}) = P(\mathbf{B}, \mathbf{A})$.)

- The **conditional probability** of $\mathbf{A}$ given $\mathbf{B}$ — denoted $P(\mathbf{A}|\mathbf{B})$ — is the probability that we would assign to $\mathbf{A}$ being true, **if** we knew $\mathbf{B}$ to be true. The conditional probability is defined as $P(\mathbf{A}|\mathbf{B}) = P(\mathbf{A}, \mathbf{B})/P(\mathbf{B})$. **TODO: intuition**

- **The Product Rule:**

$$P(\mathbf{A}, \mathbf{B}) = P(\mathbf{A}|\mathbf{B})P(\mathbf{B}) \tag{3.1}$$

In other words, the probability that $\mathbf{A}$ and $\mathbf{B}$ are both true is given by the probability that $\mathbf{B}$ is true, multiplied by the probability we would assign to $\mathbf{A}$ if we knew $\mathbf{B}$ to be true. Similarly, $P(\mathbf{A}, \mathbf{B}) = P(\mathbf{B}|\mathbf{A})P(\mathbf{A})$. This rule follows directly from the definition of conditional probability.

- **The Sum Rule:**

$$P(\mathbf{A}) + P(\bar{\mathbf{A}}) = 1 \tag{3.2}$$

In other words, the probability of a statement and its complement must sum to 1. In other words, our certainty that $\mathbf{A}$ is true is in inverse proportion to our certainty that it is not true. A consequence: given a set of mutually-exclusive statements $\mathbf{A}_i$, exactly one of which must be true, we have

$$\sum_i P(\mathbf{A}_i) = 1 \tag{3.3}$$

- All of the above rules can be made conditional on additional information. For example, given an additional statement $\mathbf{C}$, we can write the Sum Rule as:

$$\sum_i P(\mathbf{A}_i|\mathbf{C}) = 1 \tag{3.4}$$

and the Product Rule as

$$P(\mathbf{A}, \mathbf{B}|\mathbf{C}) = P(\mathbf{A}|\mathbf{B}, \mathbf{C})P(\mathbf{B}|\mathbf{C}) \tag{3.5}$$

(Note that, to condition on $\mathbf{C}$, I didn't add any more vertical bars (|); the expression $P(\mathbf{A}|\mathbf{B}|\mathbf{C})$ is undefined.)

From these rules, we further derive many more expressions to relate probabilities. For example, one important operation is called **marginalization:**

$$P(\mathbf{B}) = \sum_i P(\mathbf{A}_i, \mathbf{B}) \tag{3.6}$$

if $\mathbf{A}_i$ are mutually-exclusive statements, of which exactly one must be true. In the simplest case — where the statement $\mathbf{A}$ may be true or false — we can derive:

$$P(\mathbf{B}) = P(\mathbf{A}, \mathbf{B}) + P(\bar{\mathbf{A}}, \mathbf{B}) \tag{3.7}$$

The derivation of this formula is straightforward, using the basic rules of probability theory:

$$\begin{aligned}
P(\mathbf{A}) + P(\bar{\mathbf{A}}) &= 1, && \text{Sum rule} && (3.8)\\
P(\mathbf{A}|\mathbf{B}) + P(\bar{\mathbf{A}}|\mathbf{B}) &= 1, && \text{Conditioning} && (3.9)\\
P(\mathbf{A}|\mathbf{B})P(\mathbf{B}) + P(\bar{\mathbf{A}}|\mathbf{B})P(\mathbf{B}) &= P(\mathbf{B}), && \text{Algebra} && (3.10)\\
P(\mathbf{A}, \mathbf{B}) + P(\bar{\mathbf{A}}, \mathbf{B}) &= P(\mathbf{B}), && \text{Product rule} && (3.11)
\end{aligned}$$

TODO: intuition Marginalization gives us a useful way to compute the probability of a statement $\mathbf{B}$ that is intertwined with many other uncertain statements.

Another useful concept is the notion of **independence**. Two statements are independent if and only if $P(\mathbf{A}, \mathbf{B}) = P(\mathbf{A})P(\mathbf{B})$. If $\mathbf{A}$ and $\mathbf{B}$ are independent, then it follows that $P(\mathbf{A}|\mathbf{B}) = P(\mathbf{A})$ (by combining

the Product Rule with the defintion of independence). Intuitively, this means that, whether or not $\mathbf{B}$ is true tells you nothing about whether $\mathbf{A}$ is true.

In the rest of these notes, I will always use probabilities as statements about variables. For example, suppose we have a variable x that indicates whether there are one, two, or three people in a room (i.e., the only possibilities are $x = 1$, $x = 2$, $x = 3$). Then, by the sum rule, we can derive $P(x = 1) + P(x = 2) + P(x = 3) = 1$. I will also use probabilities to describe the range of a real variable. For example, $P(y < 5)$ is the probability that the variable y is less than 5.

To summarize:

Key point:

The basic rules of probability theory:
- $P(\mathbf{A}) \in [0...1]$
- **Product rule:** $P(\mathbf{A}, \mathbf{B}) = P(\mathbf{A}|\mathbf{B})P(\mathbf{B})$
- **Sum rule:** $P(\mathbf{A}) + P(\bar{\mathbf{A}}) = 1$
- Two statements $\mathbf{A}$ and $\mathbf{B}$ are **independent** iff: $P(\mathbf{A}, \mathbf{B}) = P(\mathbf{A})P(\mathbf{B})$
- **Marginalizing:** $P(\mathbf{B}) = \sum_i P(\mathbf{A}_i, \mathbf{B})$
- Any basic rule can be made conditional on additional information.

For example, it follows from the product rule that $P(\mathbf{A}, \mathbf{B}|\mathbf{C}) = P(\mathbf{A}|\mathbf{B}, \mathbf{C})P(\mathbf{B}|\mathbf{C})$

Once we have these rules — and a suitable model — we can derive *any* probability that we want. With some experience, you should be able to derive any desired probability (e.g., $P(A|C)$) given a basic model

Key point:

Get very familiar with the rules of probability theory (e.g., by doing the exercises) — manipulating these rules needs to be second nature.

TODO: examples

We will put these rules to use in many concrete examples in the next chapters.

3.4 Other interpretations of probability theory ($\star$)

It may seem rather vague to call a probability a "certainty," and, there are a number of other useful ways to think about probabilities. In practice, we might wish to use the term in more concrete ways. One common use of probability functions is for random sampling, for example, to create virtual characters that behave in non-deterministic ways. For example, we might create a character that randomly decides what action to take, based on the character's environment. In this case, we would describe this behavior as a probability of the action, conditioned on the environment. We might even imagine the world to be a random-number generator, so that when you flip a coin or count the pedestrians passing by, the results are generated by some hypothetical randomized machine. However, the definition of probability as a certainty is the most general and powerful. For example, if we flip a coin, the process may not actually be random, as it is determined by Newtonian mechanics, the physics of the forces you put on the coin, and the interaction of the coin with the

air around it. With enough knowledge, we could treat this as a deterministic system and predict exactly the coin's fall (randomness at the level of subatomic particles notwithstanding). However, in practice, no one does this — even though the process is not random, we nonetheless have uncertainty about its behavior.

TODO: gambling (de Finetti), dutch book theorem, frequentist thry, when is a probability a frequency?, possible worlds, Venn diagrams, Kolmogorov

3.5 Is probability theory an accurate model for human reasoning? ($\star$)

TODO: humans are not very good at reasoning probabilistically; humans make irrational decisions and evaluations; p.t. represents a sort of common sense that we'd like to believe in; doesn't model instinct, evolution, emotion

TODO: probabilistic models of the brain; neural processing

TODO: 2. Highly recommended for further reading on Bayesian inference, rational decision making, and the question 'does ordinary human reasoning obey Bayesian probability?' (to which the answer is 'often not!') @bookTversky1982, title=Judgment under Uncertainty: Heuristics and Biases, editor=Daniel Kahneman and Paul Slovic and Amos Tversky, year=1982, annote=544 pages, ISBN=0521284147

3.6 Other reasoning systems ($\star$)

While probability theory has arguably been the most successful system for reasoning with uncertainty, there are a number of other systems that can be created, for example, by making less restrictive assumptions [Friedman and Halpern 1995; Halpern 2003]. **TODO: more about plausibility vs. probability?**

3.7 Examples

(To be written.)

3.8 Exercises

1. Derive a formula for $P(\mathbf{A})$, assuming you know: $P(\mathbf{A}|\mathbf{B}_1, \mathbf{C})$ and $P(\mathbf{A}|\mathbf{B}_2, \mathbf{C})$ and $P(\mathbf{B}_1|\mathbf{C}) + P(\mathbf{B}_2|\mathbf{C}) = 1$.

2. In classical logic, knowing $\mathbf{A}$ is true and that $\mathbf{A} \rightarrow \mathbf{B}$, we can deduce $\mathbf{B}$. Show that the same can conclusion can be made using probability theory. First, write down the probabilities that correspond to the statements "$\mathbf{A}$ is true" and "$\mathbf{A} \rightarrow \mathbf{B}$". Then, solve for the probability of $\mathbf{B}$ using the basic rules of probability theory.

Chapter 4

Discrete distributions: flipping lots of coins

It it convenient to describe systems in terms of variables. For example, to describe the weather, we might define a discrete variable $\mathbf{w}$ that can take on two values sunny or rainy, and then try to determine $P(\mathbf{w} = \mathsf{sunny})$, i.e., the probability that it will be sunny today. In this chapter, we consider **discrete distributions,** that is, probabilities over discrete variables, in more detail.

As a concrete example, let's flip a coin. Let $\mathbf{c}$ be a variable that indicates the result of the flip: $\mathbf{c} = \mathsf{heads}$ if the coin lands on its head, and $\mathbf{c} = \mathsf{tails}$ otherwise. In this chapter and the rest of these notes, I will use probabilities specifically to refer to values of variables, e.g., $P(\mathbf{c} = \mathsf{heads})$ is the probability that the coin lands heads.

What is the probability that the coin lands heads? This probability should be some real number θ, $0 \leq \theta \leq 1$. For most coins, we would say $\theta = .5$. What does this number mean? The number θ is a representation of our belief about the possible values of $\mathbf{c}$. Some examples:

$\theta = 0$	we are absolutely certain the coin will land tails
$\theta = 1/3$	we believe that tails is twice as likely as heads
$\theta = 1/2$	we believe heads and tails are equally likely
$\theta = 1$	we are absolutely certain the coin will land heads

Formally, we denote the probability of the coin coming up heads as $P(\mathbf{c} = \mathsf{heads})$, so $P(\mathbf{c} = \mathsf{heads}) = \theta$. In general, we denote the probability of a specific event event as $P(\mathsf{event})$. By the Sum Rule, we know $P(\mathbf{c} = \mathsf{heads}) + P(\mathbf{c} = \mathsf{tails}) = 1$, and thus $P(\mathbf{c} = \mathsf{tails}) = 1 - \theta$.

TODO: examples, exercises

Once we flip the coin and observe the result, then we can be pretty sure that we know the value of $\mathbf{c}$; there is no practical need to model the uncertainty in this measurement. However, suppose we do not observe the coin flip, but instead hear about it from a friend, who may be forgetful or untrustworthy. Let $\mathbf{f}$ be a variable indicating how the friend claims the coin landed, i.e. $\mathbf{f} = \mathsf{heads}$ means the friend says that the coin came up heads. Suppose the friend says the coin landed heads — do we believe him, and, if so, with how much certainty? As we shall see, probabilistic reasoning obtains quantitative values that, qualitatively, matches our common sense very effectively.

Suppose we know something about our friend's behavior. We can represent our beliefs with the following probabilities, for example, $P(\mathbf{f} = \mathsf{heads} | \mathbf{c} = \mathsf{heads})$ represents our belief that the friend says "heads"

when the the coin landed heads. Because the friend can only say one thing, we can apply the Sum Rule to get:

$$P(\mathbf{f} = \mathsf{heads}|\mathbf{c} = \mathsf{heads}) + P(\mathbf{f} = \mathsf{tails}|\mathbf{c} = \mathsf{heads}) = 1 \quad (4.1)$$

$$P(\mathbf{f} = \mathsf{heads}|\mathbf{c} = \mathsf{tails}) + P(\mathbf{f} = \mathsf{tails}|\mathbf{c} = \mathsf{tails}) = 1 \quad (4.2)$$

If our friend always tells the truth, then we know $P(\mathbf{f} = \mathsf{heads}|\mathbf{c} = \mathsf{heads}) = 1$ and $P(\mathbf{f} = \mathsf{tails}|\mathbf{c} = \mathsf{heads}) = 0$. If our friend *usually* lies, then, for example, we might have $P(\mathbf{f} = \mathsf{heads}|\mathbf{c} = \mathsf{heads}) = .3$

4.1 Multinomial distributions

A multinomial distribution is one in which a discrete variable can take on multiple values. For example, rolling a die can yield one of six values, each with probability 1/6 (assuming the die is fair).

TODO: expected values; lottery example

Chapter 5

Continuous distributions

In graphics and vision, we are usually focused on continuous (real-valued) variables, such as images, 3D shapes, and character poses. In either case, we would represent the data as a real-value vector $\mathbf{x} = [x_1, x_2, ..., x_n]^T$, where the components are, for example, pixel intensities, 3D positions, or joint angles. Most of the intuitions from discrete variables transfer directly to the continuous case, although there are some subtleties.

We describe the probabilities of a real-valued scalar variable x with a Probability Distribution Function (PDF), written $p(x)$. Any real-valued function $p(x)$ that satisfies:

$$p(x) \geq 0 \qquad \text{for all } x \tag{5.1}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{5.2}$$

is a valid PDF. I will use the convention of upper-case P for discrete probabilities, and lower-case p for PDFs.

The PDF tells us the probability that the variable x falls within a given range:

$$P(x_0 \leq x \leq x_1) = \int_{x_0}^{x_1} p(x)dx \tag{5.3}$$

This can be visualized by plotting the curve $p(x)$ — in order to determine the probability that x falls within a range can be computed by computing the area under the curve for that range. **TODO: plot, and illustrate as a sequence of histograms**

The PDF can be thought of as the infinite limit of a discrete distribution, i.e., a discrete distribution with an infinite number of possible outcomes. Specifically, suppose we create a discrete distribution with N possible outcomes, each corresponding to a range on the real number line. Then, suppose we increase N towards infinity, so that each outcome shrinks to a single real number; a PDF is defined as the limiting case of this discrete distribution.

There is an important subtlety here: a PDF is *not* a probability. There is no requirement that $p(x) \leq 1$. Moreover, the probability that x attains any specific value is always zero, e.g. $P(x = 5) = \int_5^5 p(x)dx = 0$ for any PDF $p(x)$. People (myself included) are sometimes sloppy in referring to $p(x)$ as a probability, but it is not a probability — is a a function that can be used in computing probabilities.

Joint distributions are defined in a natural way. For two variables x and y, the joint PDF $p(x, y)$ defines the probability that (x, y) lies in a given domain $\mathcal{D}$:

$$P((x, y) \in \mathcal{D}) = \int_{(x,y)\in\mathcal{D}} p(x, y)d\mathbf{x} \tag{5.4}$$

For example, the probability that a 2D coordinate (x, y) lies in the domain $(0 \leq x \leq 1, 0 \leq y \leq 1)$ is $\int_{0\leq x\leq 1} \int_{0\leq y\leq 1} p(x, y)dxdy$. The PDF over a vector may also be written as a joint PDF of its variables. For example, for a 2D-vector $\mathbf{a} = [x, y]^T$, the PDF $p(\mathbf{a})$ is equivalent to the PDF $p(x, y)$.

Conditional distributions are defined as well: $p(x|\mathbf{A})$ is the PDF over x, if the statement $\mathbf{A}$ is true. This statement may be an expression on a continuous value, e.g. "$y = 5$." As a short-hand, we can write $p(x|y)$, which provides a PDF for x for every value of y. (It must be the case that $\int p(x|y)dx = 1$, since $p(x|y)$ is a PDF over values of x.)

In general, all of the rules for manipulating discrete distributions apply as well to continuous distributions:

Key point:

Probability rules for PDFs:
- $p(x) \geq 0$, for all x
- $\int_{-\infty}^{\infty} p(x)dx = 1$
- $P(x_0 \leq x \leq x_1) = \int_{x_0}^{x_1} p(x)dx$
- **Sum rule:** $\int_{-\infty}^{\infty} p(x)dx = 1$
- **Product rule:** $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$.
- **Marginalization:** $p(y) = \int_{-\infty}^{\infty} p(x, y)dx$
- We can also add conditional information, e.g. $p(y|z) = \int_{-\infty}^{\infty} p(x, y|z)da$
- **Independence:** Variables x and y are independent if: $p(x, y) = p(x)p(y)$.

TODO: figures galore

In the next sections, we'll consider two of the simplest types of PDFs: uniform distributions and Gaussian distributions.

5.1 Uniform distributions

The simplest PDF is the **uniform distribution**. Intuitively, this distribution states that all values within a given range $[x_0, x_1]$ are equally likely. **TODO: plot** Formally, the uniform distribution is:

$$p(x) = \begin{cases} \frac{1}{x_1 - x_0} & \text{if } x_0 \leq x \leq x_1 \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

It is easy to see that this is a valid PDF (because $p(x) > 0$ and $\int p(x)dx = 1$).

We can also write this distribution with this alternative notation:

$$x|x_0, x_1 \quad \sim \quad \mathcal{U}(x_0, x_1) \tag{5.6}$$

Equations 5.5 and 5.6 are equivalent. This expression simply says: x is distributed uniformly in the range x_0 and x_1, and it is impossible that x lies outside of that range.

5.2 Gaussian distributions

Arguably the single most important PDF is the Gaussian probability distribution function (PDF). The simplest case is a Gaussian PDF over a scalar value x, in which case the PDF is:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{5.7}$$

The Gaussian has two parameters: the mean μ, and the variance σ^2. The mean specifies the center of the distribution, and the variance tells us how "spread-out" the PDF is.

For a d-dimensional vector $\mathbf{x}$, the Gaussian is written

$$p(\mathbf{x}|\mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\mathbf{\Sigma}|}} e^{-(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x}-\mu)/2} \tag{5.8}$$

where μ is the mean vector, and $\mathbf{\Sigma}$ is the $d \times d$ covariance matrix. An important special case is when the covariance matrix is diagonal, and can be written $\mathbf{\Sigma} = \sigma^2 \mathbf{I}$. In this case, the PDF reduces to:

$$p(\mathbf{x}|\mu, \sigma^2) = \frac{1}{\sqrt{(2\pi)^d \sigma^{2d}}} e^{-\frac{1}{2\sigma^2}||\mathbf{x}-\mu||^2} \tag{5.9}$$

This is called a spherical covariance matrix.

I will also write Gaussian distributions using the notation:

$$p(\mathbf{x}|\mu, \mathbf{\Sigma}) = \mathcal{N}(\mathbf{x}|\mu; \mathbf{\Sigma}) \tag{5.10}$$

or

$$\mathbf{x}|\mu, \mathbf{\Sigma} \sim \mathcal{N}(\mu; \mathbf{\Sigma}) \tag{5.11}$$

Equations 5.8, 5.11, and 5.10 are three ways of writing the same thing.

The covariance matrix $\mathbf{\Sigma}$ of a Gaussian must be symmetric and positive definite — this is equivalent to requiring that $|\mathbf{\Sigma}| > 0$. Otherwise, the formula does not correspond to a valid PDF, since Equation 5.8 is no longer real-valued if $|\mathbf{\Sigma}| \leq 0$.

There are many good reasons why Gaussians are widely-used; see [Bishop 1995], Section 2.1.2, and [Jaynes 2003]. Moreover, even if the data is decidedly non-Gaussian, we will often use the Gaussian as a basic building-block for describing a more sophisticated distribution.

TODO: figures, figures, figures

5.3 Expectation, mean, variance

Some very brief definitions of ways to describe a PDF:

Given a function $\phi(\mathbf{x})$ of an unknown variable $\mathbf{x}$, the **expected value** of the function with repect to a PDF $p(\mathbf{x})$ is defined as:

$$E_{p(\mathbf{x})}[\phi(\mathbf{x})] \equiv \int \phi(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{5.12}$$

Intuitively, this is the value that we roughly "expect" $\mathbf{x}$ to have.

The mean μ of a distribution $p(\mathbf{x})$ is the expected value of $\mathbf{x}$:

$$\mu = E_{p(\mathbf{x})}[\mathbf{x}] = \int \mathbf{x} p(\mathbf{x}) d\mathbf{x} \tag{5.13}$$

The variance of a scalar variable x is the expected deviation from the mean:

$$E_{p(x)}[(x-\mu)^2] = \int (x-\mu)^2 p(x) d\mathbf{x} \tag{5.14}$$

The variance of a distribution gives us a tells us how uncertain, or "spread-out" the distribution is (Figure **??**). A very narrow distribution would have, on average, small values of $(x-\mu)^2$.

The **covariance** of a vector $\mathbf{x}$ is a matrix:

$$\mathbf{S} = E_{p(\mathbf{x})}[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^T] = \int (\mathbf{x}-\mu)(\mathbf{x}-\mu)^T p(x) d\mathbf{x} \tag{5.15}$$

By inspection, we can see that the diagonal entries of the covariance matrix are the variances of the individual entries of the vector:

$$\mathbf{S}_{ii} = E_{p(\mathbf{x})}[(x_i - \mu_i)^2] \tag{5.16}$$

The off-diagonal terms are covariances:

$$\mathbf{S}_{ij} = E_{p(x)}[(x_i - \mu_i)(x_j - \mu_j)] \tag{5.17}$$

which tells us the two variables x_i and x_j are. If the covariance is a large positive number, then we expect x_i to be larger than μ_i when x_j is larger than μ_j; if the covariance is zero, then knowing $x_i > \mu_i$ does not tell us whether $x_j > \mu_j$.

The mean of a collection of N data points $\{\mathbf{x}_i\}$ is their average: $\bar{\mathbf{x}} = \frac{1}{N}\sum_i \mathbf{x}_i$; the covariance of a set of data points is: $\frac{1}{N}\sum_i (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$. The covariance of the data points tells us how "spread-out" the data points are.

The mean of a uniform distribution $\mathcal{U}(x_0, x_1)$ is $(x_1 + x_0)/2$. The variance is $(x_1 - x_0)^2/12$.

The mean and covariance of a Gaussian distribution are its mean and covariance parameters μ and $\mathbf{\Sigma}$, respectively. (i.e., $E_{\mathcal{N}(\mathbf{x}|\mu;\mathbf{\Sigma})}[\mathbf{x}] = \mu$).

The covariance of a data set is always non-negative definite: let $\mathbf{v}$ be a vector with the same dimensionality of the data; then $\frac{1}{N}\sum_i \mathbf{v}^T(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\mathbf{v} = \frac{1}{N}\sum_i (\mathbf{v}^T(\mathbf{x}_i - \bar{\mathbf{x}}))^2$. The square of a scalar must be non-negative. The covariance of the data set is not necessarily full-rank, however, depending on the spread of the data.

TODO: examples TODO: manipulation, e.g. gaussian dist. plus gaussian noise

5.4 Exercises

1. Prove that the covariance matrix for any distribution is non-negative definite. (Note that, as a consequence, the variance of any distribution is non-negative; zero variance is a delta-function).

Chapter 6

Inference, estimation, and prediction

The central task in Bayesian reasoning is **inference:** reasoning about what we don't know, given what we know. When we make inferences about the nature of the world, this is **learning**, and this is what allows us to benefit from experience and adapt to new conditions. Although we humans make inferences and learn about the world all the time, inference and learning are arguably some of the most difficult tasks to formulate mathematically. Fortunately, Bayesian probability theory provides effective, elegant, and precise quantitative tools for reasoning about the world.

In this chapter, we explore what it means to make quantitative inferences about the world. In Bayesian reasoning, one can determine probabilities about the unknown variables given measurements. However, quite often we are also interested in **estimating** parameters, i.e., determining a single estimate of unknown values, and discarding uncertainty. Parameter estimation is most common tasks performed in statistics. However, a central theme of this chapter is that parameter estimation is not strictly justified by probability theory, and, consequently, can cause problems that would not occur in a "pure" Bayesian solution. In addition to developing the basics of inference, this chapter explores exactly where these problems come from, when they do or do not matter, and considers a number of concrete examples. The alternative is to perform **prediction,** in which we evaluate new data while accounting for all uncertainty in our models; prediction is almost always more accurate than estimation, but may be much more complicated to perform.

6.1 Overview: Learning a binomial distribution

For a simple example, we return to coin-flipping. We flip a coin N times, with the result of the i-th flip denoted by a variable $\mathbf{c}_i$: "$\mathbf{c}_i = \mathsf{heads}$" means that the i-th flip came up heads. The probability that the coin lands heads on any given trial is given by a parameter θ. We have no prior knowledge as to the value of θ, and so our prior distribution on θ is uniform.[1] In other words, we describe θ as coming from a uniform distribution from 0 to 1; we believe that all values of θ are equally likely if we have not seen any data. We assume that the individual coin flips are independent, i.e., $P(\mathbf{c}_{1:N}|\theta) = \prod_i p(\mathbf{c}_i|\theta)$. (The notation "$\mathbf{c}_{1:N}$"

[1] We would usually expect a coin to be fair, i.e., the prior distribution for θ is peaked near 0.5.

indicates the set of data values $\{\mathbf{c}_1, ..., \mathbf{c}_N\}$.) We can summarize this model as follows:

Model: Coin-Flipping		
θ	$\sim$	$\mathcal{U}(0,1)$
$P(\mathbf{c} = \text{heads})$	$=$	θ
$P(\mathbf{c}_{1:N}\|\theta)$	$=$	$\prod_i p(\mathbf{c}_i\|\theta)$

(6.1)

Suppose we wish to learn about a coin by flipping it 1000 times and observing the results $\mathbf{c}_{1:1000}$, where the coin landed heads 750 times? What is our belief about θ, given this data? We now need to solve for $p(\theta|\mathbf{c}_{1:1000})$, i.e., our belief about θ *after* seeing the 1000 coin flips. To do this, we apply the basic rules of probability theory, beginning with the Product Rule:

$$P(\mathbf{c}_{1:1000}, \theta) = P(\mathbf{c}_{1:1000}|\theta)p(\theta) = p(\theta|\mathbf{c}_{1:1000})P(\mathbf{c}_{1:1000}) \tag{6.2}$$

Solving for the desired quantity gives:

$$p(\theta|\mathbf{c}_{1:1000}) \quad = \quad \frac{P(\mathbf{c}_{1:1000}|\theta)p(\theta)}{P(\mathbf{c}_{1:1000})} \tag{6.3}$$

The numerator may be written using

$$P(\mathbf{c}_{1:1000}|\theta) = \prod_i P(\mathbf{c}_i|\theta) = \theta^{750}(1-\theta)^{1000-750} \tag{6.4}$$

The denominator may be solved for by the marginalization rule:

$$P(\mathbf{c}_{1:1000}) \quad = \quad \int_0^1 P(\mathbf{c}_{1:1000}, \theta)d\theta = \int_0^1 \theta^{750}(1-\theta)^{1000-750}d\theta = Z \tag{6.5}$$

where Z is a constant. **TODO: figure out** Z Hence, the final probability distribution is:

$$p(\theta|\mathbf{c}_{1:1000}) \quad = \quad \theta^{750}(1-\theta)^{1000-750}/Z \tag{6.6}$$

which is plotted in Figure 6.1. This form gives a probability distribution over θ that expresses our belief about θ *after* we've flipped the coin 1000 times.

Suppose we just take the peak of this distribution; from the graph, it can be seen that the peak is at $\theta = .75$. This makes sense: if a coin lands heads 75% of the time, then we would probably estimate that it will land heads 75% of the time of the future. More generally, suppose the coin lands heads H times out of N flips; we can compute the peak of the distribution as follows:

$$\arg\max_\theta p(\theta|\mathbf{c}_{1:N}) = H/N \tag{6.7}$$

(Deriving this formula is given as an exercise at the end of the chapter.)

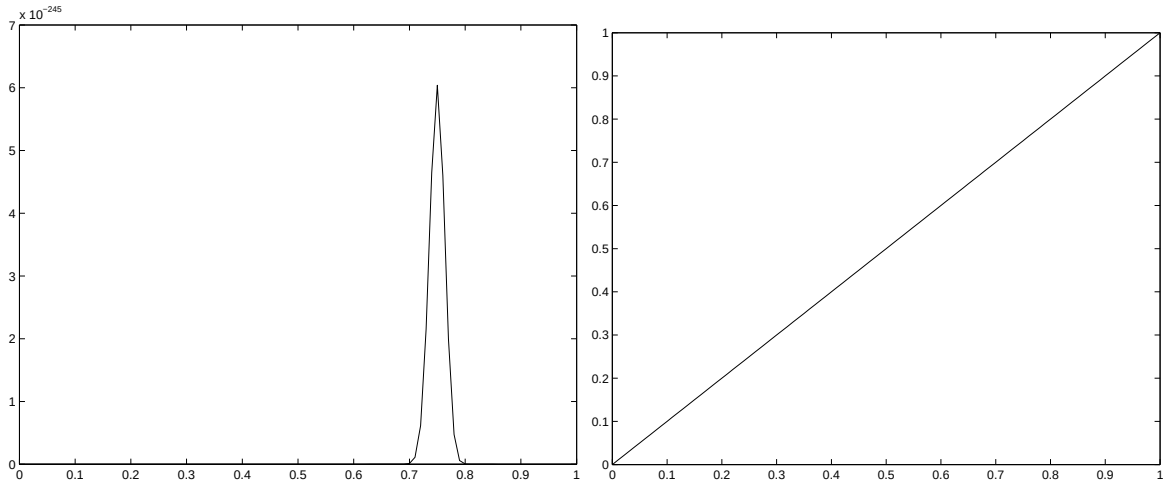

Figure 6.1: *Left:* Posterior probability of θ from 1000 coin flips, of which 750 landed heads. *Right:* Posterior probability of θ from one coin flip that landed heads. **Note: scale factor is not correct, needs to be fixed**

6.1.1 Bayes' Rule

In general, we have a model of the world described by some unknown variables model, and we observe some data data; our goal is to determine model from the data. (In coin-flip example, the model consisted of the variable θ, and the data consisted of the results of N coin flips.) We describe the probability model as $p(\mathsf{data}|\mathsf{model})$ — if we knew model, then this model will tell us what data we expect. Furthermore, we must have some prior beliefs as to what model is ($p(\mathsf{model})$), even if these beliefs are completely non-committal (e.g., a uniform distribution). Given the data, what do we know about model?

Applying the product rule as before gives:

$$p(\mathsf{data}, \mathsf{model}) = p(\mathsf{data}|\mathsf{model})p(\mathsf{model}) = p(\mathsf{model}|\mathsf{data})p(\mathsf{data}) \tag{6.8}$$

Solving for the desired distribution, gives **Bayes' Rule:**

Key point:

Bayes' Rule:

$$p(\mathsf{model}|\mathsf{data}) = \frac{p(\mathsf{data}|\mathsf{model})p(\mathsf{model})}{p(\mathsf{data})}$$

The different terms in Bayes' Rule are used so often that they all have names:

$$\underbrace{p(\mathsf{model}|\mathsf{data})}_{\text{posterior}} = \frac{\overbrace{P(\mathsf{data}|\mathsf{model})}^{\text{likelihood}}\overbrace{p(\mathsf{model})}^{\text{prior}}}{\underbrace{p(\mathsf{data})}_{\text{evidence}}} \tag{6.9}$$

TODO: explain intuition in terms of coin flips

- The **likelihood** distribution describes the likelihood of data given model — it reflects our assumptions about how the data **c** was generated. This is typically a generative model (Chapter 9).
- The **prior distribution** describes our assumptions about model before observing the data data.
- The **posterior distribution** describes our knowledge of model, incorporating both the data and the prior.
- The **evidence** is of somewhat more esoteric value; it is useful in model comparison [MacKay 2003].

6.2 Parameter estimation

Quite often, we are interested in getting a single estimate of the value of an unknown parameter, even if this means discarding all uncertainty. This is called **estimation:** determining the values of some unknown variables from observed data. In this chapter, I will outline the problem, and describe two of the main ways to do this, namely, Maximum A Posteriori (MAP), and Maximum Likelihood (ML). Estimation is the most common form of learning — given some data from the world, we wish to "learn" how the world behaves, which we will describe in terms of a set of unknown variables.

Strictly speaking, parameter estimation is not justified by Bayesian probability theory, and can lead to a number of problems, such as overfitting and nonsensical results in extreme cases. Nonetheless, it is widely used in many problems.

6.2.1 MAP and ML estimation

We can now define the MAP learning principle: choose the parameter value θ that maximize the posterior, i.e.,

$$\hat{\theta} = \arg\max_{\theta} p(\theta|\mathcal{D}) \tag{6.10}$$

$$= \arg\max_{\theta} P(\mathcal{D}|\theta)p(\theta) \tag{6.11}$$

Note that we don't need to be able to evaluate the evidence term $p(\mathcal{D})$ for MAP learning, since there are no θ terms in it.

Very often, we will assume that we have no prior assumptions about the value of θ, which we express as a **uniform prior**: $p(\theta)$ is a uniform distribution over some suitably large range. In this case, the $p(\theta)$ term can also be ignored from MAP learning, and we are left with only maximizing the likelihood. Hence, the **Maximum Likelihood** (ML) learning principle, which is a special case of MAP learning:

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}|\theta) \tag{6.12}$$

It often turns out that it is more convenient to minimize the negative-log of the objective function. Becuase "$-\ln$" is a monotonic decreasing function, we can pose MAP estimation as:

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}|\theta)p(\theta) \tag{6.13}$$

$$= \arg\min_{\theta} -\ln\left(P(\mathcal{D}|\theta)p(\theta)\right) \tag{6.14}$$

$$= \arg\min_{\theta} -\ln P(\mathcal{D}|\theta) - \ln p(\theta) \tag{6.15}$$

In the case of coin-flipping, we have obtained an intuitive result for estimating θ (i.e., the proportion of heads to total flips). However, the power of the approach presented here is that we can easily generalize to more difficult situations, when our observations our noisy, or where there are multiple sources of information:

- Suppose we wish to estimate θ based only on what our friend tells us about 100 coin flips, rather than observing them directly. In this case, we do not observe $\mathbf{c}_i$ directly, but indirectly through the friend. Again, we can solve for the optimal θ by plugging in the elements of the model to $p(\theta|\mathbf{f}_1, ..., \mathbf{f}_{100})$ (where $\mathbf{f}_i$ is what our friend says about the i-th flip) and optimizing. In this case, our final uncertainty about θ will be increased if our friend is not always reliable, and may be skewed towards $\theta = 1$ or $\theta = 0$, if we believe the friend has a preference for lying one way or another.

- Suppose we get two different sources of information about each coin flip; perhaps our friend tells us something about every coin flip, and another, more reliable friend tells us something about just a few of those coin flips. We can merge these two sources of information — and will have less uncertainty for data where we get the more reliable information — to estimate θ.

6.2.2 Overfitting and underfitting

(To be written.)

TODO: The Bias-Variance Tradeoff

6.2.3 Parameterization dependence in MAP estimation ($\star$)

(To be written.)

6.2.4 Other estimation principles ($\star$)

TODO: revise and discuss loss functions

Formally, an estimator is a function of data that outputs estimated parameter values. In general, there is no "right" way to select a single estimate of a variable, although the MAP and ML principles are widely used. I recommend using MAP/ML, if one must choose an estimator. However, there are many other choices. Here is a quick summary of a few other estimators.

- **Unbiased estimators.** Suppose we randomly sample many data points from a problem governed by an unknown parameter θ, and then we form an estimate $\hat{\theta}$ of the parameter. This estimate is a function of the data points that are observed. The **bias** of this estimator is how far off the estimate of θ will be from the true value, *averaged over all possible data sets.* An **unbiased estimate** is one in which the average estimate of θ will match the true values. However, the unbiased estimator gives no guarantees than any individual estimate will be accurate — this is the well-known "bias-variance tradeoff."

- **Median likelihood.** Another appealing estimator is to select the *median* of the posterior distribution $p(\theta|\mathcal{D})$. Specifically, the estimate $\hat{\theta}$ is chosen so that

$$P(\theta < \hat{\theta}|\mathcal{D}) = 1/2 \tag{6.16}$$

where $P(\theta < \hat{\theta}|\mathcal{D}) = \int_{-\infty}^{\hat{\theta}} p(\theta|\mathcal{D})d\theta$. In other words, the estimate splits the posterior PDF "in half."

The appeal of this estimator is that the estimate is in "the middle" of the PDF of likely values. However, there are a number of undesirable pathologies to this estimator. For example, it is possible that the median estimate is extremely unlikely; for example, in the case where the posterior PDF has two very likely values of θ, but all values in-between are very unlikely. Additionally, the median estimate is sensitive to changes in the posterior PDF very far from the median — if a mode of the PDF far away from the estimate is moved, then the median also moves. Also, deriving the median estimator may be difficult for many models.

- **Designing estimators with loss functions.** The standard approach in statistical signal processing is to select a **loss function** for estimators, and select the estimator that minimizes the loss function (see a signal processing textbook for details). By different choices of loss functions, it is possible to derive many of the previously-described estimators, and the advantages and problems of the estimator will depend on the choice of loss function.

6.3 Bayesian prediction

Although parameter estimation methods (such as MAP and least-squares) are in wide use, they are not justified by Bayesian probability theory. These estimators have theoretical deficiencies, and, moreover, there are practical situations where using them can lead to very bad results. Throughout these notes, we have seen examples where overfitting causes problems. The root of the problem is that estimating a parameter value means discarding all uncertainty in that estimate, i.e., replacing the posterior distribution $p(\theta|\mathcal{D})$ with a single estimate $\hat{\theta}$. Instead, you should always keep around the uncertainty when making predictions; which I will refer to as **Bayesian prediction**. In practice, these methods generally produce better result than based on estimation, sometimes much better [Neal 1996; Rasmussen 1997]. The disadvantage of Bayesian prediction is that it is slower than estimation, and often intractable (thus requiring numerical approximations). In this chapter, I will describe the problems with MAP and ML estimation, describe why Bayesian prediction avoids these problems, and discuss when estimation might be an acceptable approach.

6.3.1 Coin-flipping revisited

Let us now reconsider the simple case of flipping a coin, for which we do not know its bias θ (i.e., the likelihood that the coin lands heads). We will assume a uniform prior probability over θ, meaning that we have no assumptions about what θ might be[2]:

Model: Coin-flipping		
θ	$\sim$	$\mathcal{U}(0,1)$
$P(\mathbf{c} = \mathsf{heads}\|\theta)$	$=$	θ

(6.17)

where $\mathbf{c}$ is a binary variable describing how the coin lands.

[2] In reality, most people would generally assume that a coin is roughly fair (i.e., θ is near .5).

Suppose we flip the coin once, and it lands heads. What is θ? The posterior distribution is straightforward to compute:[3]

$$p(\theta|\mathbf{c} = \mathsf{heads}) = \frac{P(\mathbf{c} = \mathsf{heads}|\theta)p(\theta)}{P(\mathbf{c} = \mathsf{heads})} \quad (6.19)$$

$$= 2\theta, \text{ for } 0 \leq \theta \leq 1 \quad (6.20)$$

Suppose we now estimate θ by MAP; maximizing the posterior gives:

$$\hat{\theta} = 1 \quad (6.21)$$

No one in their right mind would do this — it makes no sense to conclude, as the result of a single coin-flip, that the coin always lands heads! Why did this happen?

Let us consider the posterior PDF (Equation 6.20), plotted in Figure 6.1. This PDF states that larger values of θ are considered more likely (and $\theta = 0$ is impossible, since we saw the coin land heads). However, this distribution is very "spread-out" — $\theta = 1$ is not dramatically more likely than $\theta = .9$, or even $\theta = .5$. There is not enough evidence here to make a conclusive decision about the value of θ. This is an illustration of a general principle:

Key point:

Parameter estimation leads to overfitting.

One could argue that this is an artificial problem — we only need to gather "enough" data so that we can get a reliable estimate. But how do we decide what is enough? It would be much more desirable to have a procedure that seamlessly works well for any amount of data. Moreover, as we consider more complex models and complex data sets, it will become more and more difficult to define exactly what it means to "gather enough data," and degeneracies will abound.

6.3.2 Bayesian prediction

Suppose we are about to flip the coin a second time. How likely is to land heads? Denoting the results of the first and second flips as variables $\mathbf{c}_1$ and $\mathbf{c}_2$, we can apply the rules of probability to obtain:

$$p(\mathbf{c}_2 = \mathsf{heads}|\mathbf{c}_1 = \mathsf{heads}) = \int_0^1 p(\mathbf{c}_2 = \mathsf{heads}, \theta|\mathbf{c}_1 = \mathsf{heads})d\theta \quad (6.22)$$

$$= \int_0^1 p(\mathbf{c}_2 = \mathsf{heads}|\theta, \mathbf{c}_1 = \mathsf{heads})p(\theta|\mathbf{c}_1 = \mathsf{heads})d\theta \quad (6.23)$$

[3] The denominator is expanded using the basic rules as:

$$P(\mathbf{c} = \mathsf{heads}) = \int_0^1 P(\mathbf{c} = \mathsf{heads}, \theta)d\theta = \int_0^1 P(\mathbf{c} = \mathsf{heads}|\theta)p(\theta)d\theta = \int_0^1 \theta d\theta = 1/2 \quad (6.18)$$

This means that, if we have a uniform prior over θ, then we assign probability $1/2$ to the coin landing heads, which makes sense. If we have no opinion about θ, then we have no opinion about $\mathbf{c}$ either.

$$= \int_0^1 p(\mathbf{c}_2 = \text{heads}|\theta)\frac{p(\mathbf{c}_1 = \text{heads}|\theta)p(\theta)}{p(\mathbf{c}_1 = \text{heads})}d\theta \quad (6.24)$$

$$= \int_0^1 2\theta^2 d\theta = 2/3 \quad (6.25)$$

This is a much more reasonable result — if we see the coin land heads once, we think that it's twice as likely to land heads rather than tails the second time, but we do not feel strongly about it. Since we have used only the rules of probability theory to derive this result — and not the MAP heuristic — we get a common-sense result based on our assumptions and the data.

In general, given a model parameterized by parameters θ, and some observed data $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, we assign the following PDF to some new data $\mathbf{x}_{N+1}$:

$$p(\mathbf{x}_{N+1}|\mathcal{D}) = \int p(\mathbf{x}_{N+1}|\theta)p(\theta|\mathcal{D})d\theta \quad (6.26)$$

$$= \int p(\mathbf{x}_{N+1}|\theta)\frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}d\theta \quad (6.27)$$

$$\propto \int p(\mathbf{x}_{N+1}|\theta)p(\mathcal{D}|\theta)p(\theta)d\theta \quad (6.28)$$

which is derived from the rules of probability as above. This is called **Bayesian Model Averaging**, or **Bayesian prediction**. We predict new values of $\mathbf{x}$ by averaging over all possible values of θ, but giving higher weight to values of θ that are more likely, according to the data and our prior beliefs. Since it is derived from the laws of probability, this is the ideal way to make predictions.

The power of Bayesian prediction — and of maintaing PDFs for parameters rather than single estimates — has been abundantly demonstrated in practice, including such methods as Kalman filtering and Particle filtering, Latent Dirichlet Allocation (state-of-the-art in text classification), and so on. In the NIPS 2003 feature selection challenge [**?**], many competitive learning algorithms were tested on a set of standard data sets, and a Bayesian technique — averaging over a vast space of possible models — gave the best performance. In the next section, we'll see these methods applied in the context of regression.

Another theoretical problem of parameter estimation methods is that they are sensitive to parameterization; it can be shown that, by reparameterizing the likelihood function that we can make the MAP estimate into anything we like! In contrast, the Bayesian prediction is invariant to parameterization.

The main disadvantage of Bayesian prediction and model averaging is that the integrals involved are often intractable; even if they are tractable, they will be more computationally intensive than a parameter estimation algorithm. This means that numerical approximations are often required (including MAP as one form of approximation, in which the posterior is replaced with a single estimate). However, it is very important to understand the tradeoffs when making these approximations — parameter estimation is not necessarily the best choice to make, and is frequently a very bad one. It is quite remarkable the number of problems that are introduced by parameter estimation.

Key point:

Parameter estimation is easier than Bayesian prediction. It works well when the posterior is "peaked;" i.e., there is "enough" data to resolve any uncertainty.

6.3.3 Overfitting revisited

The integral required for Bayesian prediction is often difficult to compute. Suppose we approximate the posterior distribution $p(\theta|\mathcal{D})$ by a delta function $\delta_{\hat{\theta}}(\theta)$ around a parameter estimate $\hat{\theta}$; then we have:

$$p(\mathbf{x}_{N+1}|\mathcal{D}) \approx \int p(\mathbf{x}_{N+1}|\theta)\delta_{\hat{\theta}}(\theta)d\theta \tag{6.29}$$

$$= p(\mathbf{x}_{N+1}|\hat{\theta}) \tag{6.30}$$

In other words, *parameter estimation is an approximation to Bayesian prediction*. This tells us when parameter estimation is reasonable — if we can approximate the posterior with a delta-function, then we should get good predictions. For example, if we flip a coin n times, and get h heads, then we expect that next flip to land heads with probability $(h+1)/(n+2)$. For a large number of flips, the MAP estimate $\hat{\theta} = h/n$ is very nearly the same. This is because the posterior distribution becomes very peaked around the MAP estimate (Figure **??**). However, since the posterior distribution is not a delta-function, we are still discarding some information when using the MAP estimate, and assuming absolute certainty about the value of θ.

TODO: the following are the same problem: overfitting, can't estimate 100 unknowns from 99 constraints, 98 constraints is unstable, discarding uncertainty in the estimate; parameter counting in general

(To be written.)

Key point:

When estimating parameters, marginalize out as many unknowns as possible to get the parameters you want.

6.3.4 Estimating a uniform distribution ($\star$)

Suppose you live in a place where all automobile license plates are numbered sequentially, starting at #1, so that the 50th car has license #50. If you watch cars for a day, can you estimate how many cars there are?

Assuming that every car you see is sampled uniformly from all cars, the generative model is:

Model: License plates

$$\begin{aligned} M &\sim \mathcal{U}(0, L) \\ \ell &\sim \mathcal{U}(1, M) \end{aligned} \tag{6.31}$$

where M is the total number of cars (also, the number of the largest license plate), ℓ is the license plate of a car randomly chosen from among all cars, and L is a very large number: the largest number of cars you imagine that there possibly could be.

Suppose we observe a set of cars $\mathcal{D} = \{\ell_1, ..., \ell_N\}$. Let $X = \max_i(\ell_i)$ be the largest license plate number observed. Then posterior distribution for M is:

$$p(M|\mathcal{D}) = \frac{p(\mathcal{D}|M)p(M)}{p(\mathcal{D})} \tag{6.32}$$

$$= \frac{\prod_i \mathcal{U}(\ell_i|1,M)\mathcal{U}(M|0,L)}{\sum_{k=0}^{L}\prod_i \mathcal{U}(\ell_i|1,k)\mathcal{U}(k|0,L)} \tag{6.33}$$

$$= \begin{cases} \frac{M^{-N}}{\sum_{k=X}^{L} k^{-N}} & \text{if } X \le M \le L \\ 0 & \text{otherwise} \end{cases} \tag{6.34}$$

What is the MAP estimate of M? Clearly, the posterior is maximized by setting

$$\hat{M} = X \tag{6.35}$$

In other words, if the largest license plate number was #500, then we believe that there are 500 cars — assuming a larger number of cars would decrease the likelihood of the data we saw. This is not a very intuitive result — generally, most people would assume that the total number of cars is at least a bit larger than the largest number that we saw.[4] In the extreme case, if we only look at one car — and it has license plate #30 — it would be crazy to then decide that there must be exactly 30 cars. Of course, one could choose another estimator other than MAP that might give a larger estimate. Even then, once you estimate $\hat{M}$, you are saying that the probability of seeing a license plate numbered larger than that estimate is 0 — it is absolutely impossible that any more cars exist.

The posterior $p(M|\mathcal{D})$ is reasonable — it's just unreasonable to estimate a single value from it. However, we can perform Bayesian prediction: what car numbers are we most likely to see next?

$$p(\ell_{N+1}|\mathcal{D}) = \sum_{M=1}^{L} p(\ell_{N+1}, M|\mathcal{D}) \tag{6.36}$$

$$= \sum_{M=1}^{L} p(\ell_{N+1}|M)p(M|\mathcal{D}) \tag{6.37}$$

$$= \frac{\sum_{M=\max(X,\ell_{N+1})}^{L} M^{-(N+1)}}{\sum_{k=X}^{L} k^{-N}} \tag{6.38}$$

This is a much more sensible prediction — license plates $\le X$ are all equally likely, and larger numbers are possible, but less likely. **TODO: plot**

Note that, while the size of the prior L did not affect the maximum likelihood estimate, it does affect the posterior distribution and the Bayesian prediction. In general, we need to be more careful to construct reasonable priors for Bayesian methods, even if the prior is meant to be "non-informative," expressing no prior opinion about the unknown values. However, it may be possible to consider the prediction in the limit as $L \to \infty$, if the series converges.

6.3.5 When is estimation safe?

(To be written.)

[4] Nonetheless, this is the solution that we were given in my undergraduate statistics course.

6.4 Learning Gaussians

We now consider the problem of learning a Gaussian distribution from K training samples $\{\mathbf{y}_i\}$. Maximum likelihood learning of the the parameters μ and $\mathbf{\Sigma}$ entails maximizing the likelihood:

$$p(\mu, \mathbf{\Sigma}|\{\mathbf{y}_i\}) \quad \propto \quad p(\{\mathbf{y}_i\}|\mu, \mathbf{\Sigma}) \tag{6.39}$$

which follows from Bayes Rule. Since we assume that the data points come from a Gaussian:

$$\begin{aligned} p(\{\mathbf{y}_i\}|\mu, \mathbf{\Sigma}) &= \prod_{i=1}^{K} p(\{\mathbf{y}_i\}|\mu, \mathbf{\Sigma}) && (6.40) \\ &= \prod_{i=1}^{K} \frac{1}{\sqrt{(2\pi)^N |\mathbf{\Sigma}|}} e^{-(\mathbf{y}_i-\mu)^T \mathbf{\Sigma}^{-1} (\mathbf{y}_i-\mu)/2} && (6.41) \end{aligned}$$

where M is the dimensionality of the data $\mathbf{y}_i$. It is somewhat more convenient to minimize the negative log-likelihood:

$$\begin{aligned} L(\mu, \mathbf{\Sigma}) &\equiv -\ln p(\{\mathbf{y}_i\}|\mu, \mathbf{\Sigma}) && (6.42) \\ &= -\sum_i \ln p(\mathbf{y}_i|\mu, \mathbf{\Sigma}) && (6.43) \\ &= \sum_i (\mathbf{y}_i - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{y}_i - \mu)/2 + \frac{K}{2} \ln |\mathbf{\Sigma}| + \frac{KM}{2} \ln(2\pi) && (6.44) \end{aligned}$$

Solving for μ and $\mathbf{\Sigma}$ by setting $\partial L(\mu, \mathbf{\Sigma})/\partial \mu = 0$ and $\partial L(\mu, \mathbf{\Sigma})/\partial \mathbf{\Sigma} = 0$ (subject to the constraint that $\mathbf{\Sigma}$ is symmetric) gives the maximum likelihood estimates[5]:

$$\begin{aligned} \hat{\mu} &= \frac{1}{K} \sum_i \mathbf{y}_i && (6.45) \\ \hat{\mathbf{\Sigma}} &= \frac{1}{K} \sum_i (\mathbf{y}_i - \hat{\mu})(\mathbf{y}_i - \hat{\mu})^T && (6.46) \end{aligned}$$

The ML estimates make intuitive sense: we estimate the Gaussian's mean to be the mean of the data, and the Gaussian's covariance to be the covariance of the data. Maximum likelihood estimates usually make sense intuitively. This is very helpful when debugging your math — you can sometimes find bugs in derivations simply because the ML estimates did not look right.

6.4.1 Overfitting and regularization for Gaussians

It sometimes happens that the estimated covariance matrix $\hat{\mathbf{\Sigma}}$ is not full rank, or, more perversely, has negative eigenvalues. This can happen for two reasons: first, a small data set may not effectively capture all the variation in the model (and thus be overfit), and, second, numerical instability can mangle small

[5] A good exercise is to derive this formula in the scalar case. Deriving it in the matrix case requires using some matrix differentials; see [Magnus and Neudecker 1999; Minka 2000a].

eigenvalues. One way to understand the problem with small data sets is that there are $d^2/2$ unknowns in the covariance matrix, and that, for large d, many training data points are needed to estimate all these unknowns.

There are many ways to fix this problem. The simplest way to fix this problem is as follows. Compute the eigenvector decomposition of the estimated covariance matrix:

$$\hat{\mathbf{\Sigma}} = \mathbf{A}\mathbf{\Lambda}\mathbf{A}^T \tag{6.47}$$

where $\mathbf{A}$ is the eigenvector matrix, and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, ..., \lambda_d)$ are the eigenvalues. Then, simply threshold the eigenvalues by some threshold T: $\lambda_i = \max(\lambda_i, T)$. I normally use $T = \sqrt{\epsilon}$ where $\epsilon = 2 \cdot 10^{-16}$ is approximately floating-point precision. The covariance matrix can then be reconstructed using the new eigenvalues, using Equation 6.47.

A more principled approach would be to express prior assumptions about the covariance using a prior over $\mathbf{\Sigma}$. Or, we can use a PCA model instead of the Gaussian, which will represent the Gaussian with less parameters, as described in Chapter 7. We can use a Bayesian prediction method (Chapter **??**) which avoids overfitting without extra assumptions, but this is overkill for this particular problem.)

There is another form of overfitting at work in maximum likelihood estimation of a Gaussian — in general, a better estimate of the covariance is to replace the $1/N$ factor with a factor of $1/(N-1)$. This will be discussed in more detail in Section **??**. (For most data sets, the difference between $1/N$ and $1/(N-1)$ will be insignificant).

6.5 Decision theory and making choices

TODO: choosing actions (e.g., gambling), loss functions
(To be written.)

6.6 Summary

(To be written.)

6.7 Exercises

1. Derive Equation 6.7. (Hint: it might be slightly easier to maximize the negative log-posterior).

Chapter 7

Linear models: Linear regression, PCA, factor analysis

Armed with the Gaussian distribution, we now consider linear models, including the supervised case (linear regression), and the unsupervised case (factor analysis and PCA). We begin with linear regression.

7.1 Linear regression in 1D

In linear regression, we assume that there are two sets of variables: input variables x, and output variables y. We first consider the case in which both values are scalar. We assume that the output variables y are produced by a linear function of the input variables, plus Gaussian noise:

Model: 1D Linear regression		
n	$\sim$	$\mathcal{N}(0; \sigma^2)$
y	$=$	$ax + b + n$

(7.1)

This is equivalent to writing $p(y|x, a, b, \sigma^2) = \mathcal{N}(y|ax + b; \sigma^2)$. **TODO: figure** We additionally assume uniform priors over the parameters a, b, and σ^2 (not written in the model above, for brevity). The estimation problem is to solve for these parameters, given K pairs training data $\mathcal{D} = \{(x_1, y_1), ..., (x_K, y_K)\}$. Estimating the parameters by maximum likelihood entails maximizing:

$$p(\mathcal{D}|a, b, \sigma^2) = \prod_{i=1}^{K} p(y_i, x_i|a, b, \sigma^2) \tag{7.2}$$

$$= \prod_{i=1}^{K} p(y_i|a, b, \sigma^2)p(x_i|a, b, \sigma^2) \tag{7.3}$$

Equivalently, we can minimize

$$L(a, b, \sigma^2) = -\ln p(\mathcal{D}|a, b, \sigma^2) \tag{7.4}$$

$$= -\sum_{i=1}^{K} \ln p(y_i|a, b, \sigma^2) - \sum_{i=1}^{K} \ln p(x_i|a, b, \sigma^2) \tag{7.5}$$

because "$-\ln$" is a monotonically-decreasing function.

We assume that the values of x_i are independent from the unknowns, so we can drop the second term.[1] Expanding gives:

$$L(a, b, \sigma^2) = \sum_{i=1}^{K} \frac{1}{2\sigma^2}(y_i - (ax_i + b))^2 + \frac{N}{2}\ln\sigma^2 \tag{7.6}$$

In order to optimize this, we solve the system of equations $\partial L/\partial a = 0$, $\partial L/\partial b = 0$, $\partial L/\partial \sigma^2 = 0$, which gives the estimators:

$$\hat{a} = \frac{\sum_{i=1}^{K}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{K}(x_i - \bar{x})^2} \tag{7.7}$$

$$\hat{b} = \bar{y} - \hat{a}\bar{x} \tag{7.8}$$

$$\hat{\sigma^2} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\hat{a}x_i + \hat{b}))^2 \tag{7.9}$$

where $\bar{x} = \frac{1}{N}\sum_i x_i$ and $\bar{y} = \frac{1}{N}\sum_i y_i$. These estimators should make sense: $\hat{b}$ is the difference between the averages x and the average y; $\hat{a}$ is the ratio of the average $y - \bar{y}$ over the average $x - \bar{x}$, and $\hat{\sigma^2}$ is the average deviation from the actual y_i from the "predicted" value $\hat{a}x_i + \hat{b}$.

Note that the first term in the objective function is essentially a "least-squares" objective function. This illustrates that

Key point:

Least-squares fitting is a MAP estimation rule. It suffers from the same overfitting problems as MAP and maximum likelihood.

For example, if we fit a line to 100 data points, we would generally expect the fit to be very certain. However, if all 100 data points are identical to each other, then the fit will not be reliable (because the posterior distribution is very ambiguous).

7.1.1 Regression in higher dimensions

We assume that the M-dimensional output variables $\mathbf{y}$ are produced by a linear function of the N-dimensional input $\mathbf{x}$ variables, plus additional noise:

$$\underbrace{\mathbf{y}}_{M\times 1} = \underbrace{\mathbf{A}}_{M\times N}\underbrace{\mathbf{x}}_{N\times 1} + \underbrace{\mathbf{b}}_{M\times 1} + \underbrace{\mathbf{n}}_{M\times 1} \tag{7.10}$$

The linear transformation is defined by the matrix $\mathbf{A}$ and the vector $\mathbf{b}$. These matrices define a hyperplane in the M-dimensional space, with basis vectors corresponding to the columns of $\mathbf{A}$. The vector $\mathbf{n}$ contains

[1] For linear regression, we do not need to assume a distribution over the x values at all, since (a) they are given with the problem, and (b) none of the unknown parameters describe how the x's are sampled (i.e., we're not trying to learn a distribution over x). We will later consider the unsupervised case of estimating x, for which will will need to assume some distribution over x.

noise — each component of the vector is sampled from a zero-mean Gaussian distribution with variance σ^2. This is equivalent to $\mathbf{n}$ being sampled from a multivariate Gaussian with mean zero and covariance matrix $\sigma^2\mathbf{I}$. The complete model is:

Model: Linear regression		
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \sigma^2\mathbf{I})$
$\mathbf{y}$	$=$	$\mathbf{A}\mathbf{x} + \mathbf{b} + \mathbf{n}$

(7.11)

Alternatively, we may write the complete model as:

$$p(\mathbf{y}|\mathbf{A}, \mathbf{b}, \mathbf{x}, \sigma^2) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}; \sigma^2\mathbf{I}) \tag{7.12}$$

In other words, if we know all of the other parameters of the model, then the PDF of $\mathbf{y}$ given the unknowns is a Gaussian with mean $\mathbf{A}\mathbf{x} + \mathbf{b}$ and covariance $\sigma^2\mathbf{I}$.

The estimation problem in linear regression is to estimate the model parameters $\mathbf{A}$, $\mathbf{b}$, and σ^2 from data. Given K training data pairs $\{(\mathbf{x}_i, \mathbf{y}_i)\}$, how do we estimate the unknowns? First, let us write out the joint likelihood of the training data; since they are independent (given the model parameters), we have:

$$p(\{(\mathbf{x}_i, \mathbf{y}_i)\}|\mathbf{A}, \mathbf{b}, \sigma^2) = \prod_{i=1}^{K} p(\mathbf{x}_i, \mathbf{y}_i|\mathbf{A}, \mathbf{b}, \sigma^2) \tag{7.13}$$

$$= \prod_{i=1}^{K} p(\mathbf{y}_i|\mathbf{A}, \mathbf{b}, \sigma^2, \mathbf{x}_i)p(\mathbf{x}_i) \tag{7.14}$$

Assuming uniform priors over the model parameters, our goal is to maximize the likelihood (Equation 7.14). This is equivalent to minimizing the negative log of the likelihood:

$$\mathcal{L}(\mathbf{A}, \mathbf{b}, \sigma^2) = -\ln \prod_{i=1}^{K} p(\mathbf{y}_i|\mathbf{A}, \mathbf{b}, \sigma^2, \mathbf{x}_i)p(\mathbf{x}_i) \tag{7.15}$$

$$= -\sum_{i=1}^{K} \ln p(\mathbf{y}_i|\mathbf{A}, \mathbf{b}, \sigma^2, \mathbf{x}_i) - \ln p(\mathbf{x}_i) \tag{7.16}$$

We assume that the second term is constant with respect to the unknowns, i.e the values of $\mathbf{x}$ do not depend on the model parameters. Dropping these terms and subsituting in the Gaussian model gives:

$$\mathcal{L}(\mathbf{A}, \mathbf{b}, \sigma^2) = \sum_{i=1}^{K} \frac{1}{2\sigma^2}||\mathbf{y}_i - (\mathbf{A}\mathbf{x}_i + \mathbf{b})||^2 + \frac{K}{2}\ln((2\pi)^M\sigma^{2M}) \tag{7.17}$$

To solve for the maximum likelihood model parameters, we must minimize this expression. Fortunately, this may be computed in closed form, by solving the simultaneous system of equations $\frac{\partial L}{\partial \mathbf{A}} = 0$, $\frac{\partial L}{\partial \mathbf{b}} = 0$,, and $\frac{\partial L}{\partial \sigma^2} = 0$. Doing a little bit of algebra, we obtain the estimators:

$$\hat{\mathbf{A}} = \left(\sum_i (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\right)\left(\sum_i (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T\right)^{-1} \tag{7.18}$$

$$\hat{\mathbf{b}} = \bar{\mathbf{y}} - \hat{\mathbf{A}}\bar{\mathbf{x}} \tag{7.19}$$

$$\hat{\sigma}^2 = \frac{1}{KM}\sum_{i=1}^{K} ||\mathbf{y}_i - (\hat{\mathbf{A}}\mathbf{x}_i + \hat{\mathbf{b}})||^2 \tag{7.20}$$

where $\bar{\mathbf{x}} = \frac{1}{K}\sum_i \mathbf{x}_i$ and $\bar{\mathbf{y}} = \frac{1}{K}\sum_i \mathbf{y}_i$. **TODO: double-check this** These estimates have intuitive interpretations; the linear estimators $\hat{\mathbf{A}}$ and $\hat{\mathbf{b}}$ is a standard result from linear algebra, and the variance $\hat{\sigma}^2$ measures the variance in the residuals $(\mathbf{y}_i - (\hat{\mathbf{A}}\mathbf{x}_i + \hat{\mathbf{b}}))$.

Once we have estimated these parameters, we can apply regression to new input values $\mathbf{x}$, or we can randomly sample new values for given values of $\mathbf{x}$. For example, given a new $\mathbf{x}$, what is the most likely value of the corresponding $\mathbf{y}$? Since the noise is zero-mean, the answer is $\hat{\mathbf{A}}\mathbf{x} + \hat{\mathbf{b}}$.

7.2 Unsupervised linear models

What if we do not know the values of $\mathbf{x}_i$ in advance, i.e., the problem is unsupervised? Depending on the noise models we choose — assuming we restrict ourselves to Gaussians — then we obtain one of the two popular models: Principal Components Analysis (PCA), or Factor Analysis (FA).

Again, we assume that there is a linear relationship between the data: $\mathbf{y}_i = \mathbf{A}\mathbf{x}_i + \mathbf{b} + \mathbf{n}$, where $\mathbf{n}$ denotes Gaussian noise. However, our goal is find all of the parameters of the hyperplane $\mathbf{A}$, $\mathbf{b}$, as well as the $\mathbf{x}_i$ values and the noise parameters. In other words, we are fitting a hyperplane to a scattered set of data points $\mathbf{y}_i$. The $\mathbf{x}$ values are called **latent parameters** because there is one for each data point, but they are not observed with the data. The space of latent parameters is the **latent space**.

TODO: figure, 1D example, 2D example

Note that the columns matrix $\mathbf{A}$ provides a basis for points in the high-dimensional space, and the elements of $\mathbf{x}$ correspond to coordinates in this space: $\mathbf{y} = \sum_j \mathbf{a}_j x_j + \mathbf{b}$, where $\mathbf{a}_j$ is a column of $\mathbf{A}$, and x_j is the j-th element of a vector $\mathbf{x}$.

7.2.1 Conventional PCA as hyperplane estimation

Conventional PCA is widely used in graphics, vision, and many other areas. There are several different derivations for PCA; here is one that I find simplest[2]. We seek the maximum likelihood estimates of the hyperplane parameters *and* the unknown $\mathbf{x}_i$ values. Moreover, we assume that the $\mathbf{x}_i$ values are uniformly distributed[3] , and that the noise is zero-mean Gaussian with variance σ^2. Hence, the complete model is

Model: Conventional PCA		
$\mathbf{x}$	$\sim$	$\mathcal{U}$
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \sigma^2\mathbf{I})$
$\mathbf{y}$	$=$	$\mathbf{A}\mathbf{x} + \mathbf{b} + \mathbf{n}$

(7.21)

TODO: state dimensionality. In this model, all data points lie near a hyperplane $\mathbf{A}\mathbf{x} + \mathbf{b}$ in d-dimensional space. In order to make the representation unambiguous[4], we assume that the basis is orthonormal: $\mathbf{A}^T\mathbf{A} = \mathbf{I}$. We assume uniform priors over $\mathbf{x}$, $\mathbf{A}$, $\mathbf{b}$, and σ^2 as well. Note that this model is identical to linear

[2] I have not found in the literature a definition of PCA in these terms. The following sections in this chapter describe the more usual formulations.

[3] Technically, we must specify a finite domain D over which the $\mathbf{x}_i$ values are distributed; we cannot have a uniform distribution over an infinite domain. We finesse this point by assuming that the distribution is uniform over a domain so large that it contains any values we might ever possibly observe. For example, we can assume a uniform distribution over the numbers within double-precision floating point.

[4] Specifically, we can rescale and/or rotate $\mathbf{x}$ and $\mathbf{A}$, and get the same data and the same likelihood.

regression model in the previous section, except that we have had to express priors over $\mathbf{x}$, and constrain $\mathbf{A}$. (If you find this constraint to be inelegant, then you may prefer the other formulations later in this chapter).

Given the data $\mathcal{D} = \{\mathbf{y}_i\}$, we can write down the likelihood of the model parameters:

$$\begin{aligned} p(\mathbf{A}, \mathbf{b}, \sigma^2, \{\mathbf{x}_i\}|\mathcal{D}) &\propto p(\mathcal{D}|\mathbf{A}, \mathbf{b}, \sigma^2, \{\mathbf{x}_i\}) & (7.22) \\ &= \prod_i \mathcal{N}(\mathbf{y}_i|\mathbf{A}\mathbf{x}_i + \mathbf{b}; \sigma^2\mathbf{I}) & (7.23) \end{aligned}$$

We then wish to maximize this, or equivalently, minimize the negative log-probability:

$$\mathcal{L}(\mathbf{A}, \mathbf{b}, \sigma^2, \{\mathbf{x}_i\}) = \sum_i \frac{1}{2\sigma^2}||\mathbf{y}_i - (\mathbf{A}\mathbf{x}_i + \mathbf{b})||^2 + \frac{K}{2}\log((2\pi)^M\sigma^{2M}) \quad (7.24)$$

We can then estimate the parameters of this function minimizing this expression [Bishop 1995] in closed-form:

$$\hat{\mathbf{b}} = \frac{1}{N}\sum_j \mathbf{y}_j \quad (7.25)$$

$$\hat{\sigma^2} = \frac{1}{N}\sum_i ||\mathbf{y}_i - (\hat{\mathbf{A}}\mathbf{x}_i + \hat{\mathbf{b}})||^2 \quad (7.26)$$

and $\hat{\mathbf{A}}$ is a matrix of the first N eigenvectors of the data covariance matrix $\frac{1}{N}\sum_i(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$. We can then solve for the maximum likelihood value of each $\mathbf{x}_i$ by solving $\partial\mathcal{L}/\partial\mathbf{x}_i = 0$:

$$\hat{\mathbf{x}}_i = \hat{\mathbf{A}}^T(\mathbf{y}_i - \hat{\mathbf{b}}) \quad (7.27)$$

since $\hat{\mathbf{A}}^T\hat{\mathbf{A}} = \mathbf{I}$.

7.2.2 Conventional PCA as data compression

PCA is sometimes motivated by a very different goal: lossy data compression. Suppose we wish to transmit a collection of vector $\{\mathbf{y}_i\}$ over a network, but the vectors are very large. We could encode these vectors using a matrix $\mathbf{A}$ and some vectors $\mathbf{b}$ and $\{\mathbf{x}_i\}$ instead, so that the receiver reconstructs the original vectors as $\mathbf{y}'_i = \mathbf{A}\mathbf{x} + \mathbf{b}$. If the dimensionality of $\mathbf{x}$ is much less than the dimensionality of $\mathbf{y}$, or there are many vectors, than this encoded representation will be cheaper to transmit than the original vectors.

We would like to choose the encoding to minimize the distortion of the compressed data:

$$E(\mathbf{A}, \mathbf{b}, \{\mathbf{x}_i\}) = \sum_i ||\mathbf{y}_i - (\mathbf{A}\mathbf{x}_i + \mathbf{b})||^2 \quad (7.28)$$

This is effectively the same objective function as in the previous section, and is solved in the same way. PCA has widely-used for compression in the graphics literature (e.g., [Hertzmann et al. 2001; Lengyel 1999; Matusik et al. 2002]).

Suppose that, once we've built the model, we will receive new $\mathbf{y}$ vectors to be transmitted. A PCA model that fits the original data well may not compress new data well, unless (a) the new data distribution is accurately described by a hyperplane, and (b) the original data is sufficient for estimating this hyperplane. Even if we use a linear model to compress the model, we might wish to estimate some other PDF for the data, and then optimize the compression scheme to minimize the expected distortion with respect to the PDF.

7.2.3 Conventional PCA as variance maximization ($\star$)

The classical definition of PCA is somewhat different from the above definitions, in that there is no model assumed of the data. Instead, we seek to replace the $\mathbf{y}_i$ variables with lower-dimensional variables $\mathbf{x}_i = \mathbf{A}^T(\mathbf{y}_i - \mathbf{b})$, in such a manner that the data covariance $\sum_i \mathbf{x}_i \mathbf{x}_i^T$ is maximized. This goal yields an objective function identical to the ones above. The idea is that the low-dimensional features should capture as much variation of the high-dimensional features as possible. (In the opposite extreme, minimizing the variance would lead to a data set with no variation: $\mathbf{x}_i = 0$.)

One possible advantage of this approach is that it is "model-free:" we do not assume that the data comes from some model (such as a hyperplane) — in this view, PCA is a data-reduction procedure, not a parameter estimation algorithm. Joliffe [**?**] stresses repeatedly that the original and correct definition of PCA is model-free variance maximization. However, a quick look through the literature indicates that this view is not universally shared in the machine learning community. Personally, I find the model-based view to be most useful, both for understanding the algorithm (it is hard to argue for variance maximization as a general principle of learning) and for building generalizations (such as in synthesizing new data).

7.2.4 Pros and cons of conventional PCA

There are three possible reasons to use PCA:

- You believe that the model is appropriate, i.e., that the data lies uniformly distributed on an infinite hyperplane.
- You wish to compress data for transmission or storage.
- You wish to preprocess a data set for efficiency, before applying a more expensive algorithm.

The first two cases are straightforward. The third case is an instance of **dimensionality reduction**. Many algorithms perform poorly on data sets with very high dimensionality (both in terms of speed, and, for MAP-based methods, in terms of overfitting). Since PCA is very fast and easy to compute, one can replace the original $\mathbf{y}$ data values with the $\mathbf{x}$ values, and then apply the "real" model to the low-dimensional $\mathbf{x}$ values. For example, in Style Machines [Brand and Hertzmann 2000], MAP estimation of our model was too expensive and unreliable using the original high-dimensional body pose representation. Hence, we reduced the data to 10 dimensions and then fit our model in this reduced space. We found that 10 dimensions was sufficient to keep around the variation in human figure motion[5].

PCA has also been used as the complete model for data for analysis and synthesis. However, unless your data really fits the hyperplane model, this is probably not the best model to choose. There are a number of ways to look at this. The main issue is that PCA does not really "learn anything" about the distribution of the $\mathbf{x}$ values; if one is going to learn a model, it seems inadequate to first fit a hyperplane, and then learn nothing about the reduced representation $\mathbf{x}$. A consequence is that the model is very sensitive to the choice of dimensionality — if we choose a dimensionality for $\mathbf{x}$ that is very large, then the model does not constrain the $\mathbf{y}$ values very much. If we choose the dimensionality of $\mathbf{x}$ to be the same as the value of $\mathbf{y}$, then the

[5] I have since found 10 dimensions to be sufficient on other motion capture datasets with other joint angle parameterizations, and other researchers have independently found 10 to be sufficient. This is an intriguing coincidence, although I would expect the number to be larger for much larger data sets.

model is entirely vacuous. On the other hand, if we choose the dimensionality to be very small, then we may lose a lot of degrees of variation in the data set. In practice, it may be hard to appropriately choose the dimensionality.

The alternative would be to fit the hyperplane, but also learn a distribution over the $\mathbf{x}$ values — this is what probabilistic PCA does. For data modeling problems, I would always recommend using some form of probabilistic PCA (as described in the next sections) over conventional PCA.

7.2.5 Probabilistic PCA

In probabilistic PCA (PPCA) [Roweis 1998; Tipping and Bishop], we assume that the latent parameters come from a Gaussian distribution:

Model: Probabilistic PCA		
$\mathbf{x}$	$\sim$	$\mathcal{N}(0; \mathbf{I})$
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \sigma^2\mathbf{I})$
$\mathbf{y}$	$=$	$\mathbf{A}\mathbf{x} + \mathbf{b} + \mathbf{n}$

(7.29)

Unlike conventional PCA, we do not assume that the matrix $\mathbf{A}$ is orthonormal.

TODO: visualization

Another way to understand PPCA is as follows. Marginalizing out the $\mathbf{x}_i$ latent parameters from the model yields the following equivalent formulation:

Model: Probabilistic PCA		
$\mathbf{y}$	$\sim$	$\mathcal{N}(\mathbf{b}; \mathbf{A}\mathbf{A}^T + \sigma^2\mathbf{I})$

(7.30)

In other words, fitting PPCA is equivalent to fitting a Gaussian distribution, but with the covariance matrix in a special form $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^T + \sigma^2\mathbf{I}$. This covariance matrix is full-rank, but has less parameters than a full-rank matrix — depending on the latent dimensionality, potentially a lot less.

Estimation in this case is somewhat different than for conventional PCA. The PDF of the unknowns given the training data $\mathcal{D} = \{\mathbf{y}_i\}$ is:

$$p(\mathbf{A}, \mathbf{b}, \sigma^2, \{\mathbf{x}_i\}|\{\mathbf{y}_i\}) \quad \propto \quad \prod_i \Big(\mathcal{N}(\mathbf{y}_i|\mathbf{A}\mathbf{x}_i + \mathbf{b}; \sigma^2\mathbf{I})\mathcal{N}(\mathbf{x}_i|0; \mathbf{I})\Big) \tag{7.31}$$

Now, suppose we were to estimate some values for the parameters $\mathbf{A}$, $\mathbf{b}$, σ^2, and $\{\mathbf{y}_i\}$. Then for any scale factor $s < 1$, we could rescale the data as $\mathbf{x}'_i = s\mathbf{x}_i$, and $\mathbf{A}' = \mathbf{A}/s$, and thereby get a model with lower likelihood. Doing so gives us very poor model in which the $\mathbf{x}_i$ values attain infinitesimal values.

What went wrong? The problem is overfitting — we are trying to estimate too many parameters from not enough data points[6] (this problem will be discussed in more generality in Chapter **??**). Instead, what we can do is estimate just the $\mathbf{A}$, $\mathbf{b}$, and σ^2 parameters *without* estimating the $\{\mathbf{x}_i\}$ values, by maximizing:

$$p(\mathbf{A}, \mathbf{b}, \sigma^2|\{\mathbf{y}_i\}) \quad \propto \quad \prod_i p(\mathbf{y}_i|\mathbf{A}, \mathbf{b}, \sigma^2) \tag{7.32}$$

[6] Thanks to Sam Roweis for explaining this to me.

$$= \prod_i \int p(\mathbf{x}_i, \mathbf{y}_i | \mathbf{A}, \mathbf{b}, \sigma^2) d\mathbf{x}_i \tag{7.33}$$

$$= \prod_i \int p(\mathbf{y}_i | \mathbf{x}_i, \mathbf{A}, \mathbf{b}, \sigma^2) p(\mathbf{x}_i) d\mathbf{x}_i \tag{7.34}$$

$$= \prod_i \int \mathcal{N}(\mathbf{y}_i | \mathbf{A}\mathbf{x}_i + \mathbf{b}; \sigma^2 \mathbf{I}) \mathcal{N}(\mathbf{x}_i | 0; \mathbf{I}) d\mathbf{x}_i \tag{7.35}$$

$$= \prod_i \mathcal{N}(\mathbf{y}_i | \mathbf{b}; \mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}) \tag{7.36}$$

TODO: double-check this

In this case, we marginalize out the unknown $\mathbf{x}_i$ values (using the standard rules of probability), and thus we have a problem with less unknowns that can be maximized robustly.

For example, **(To be written.)TODO: detailed examples where PPCA is superior**

7.2.6 Factor analysis

The most general unsupervised linear-Gaussian model is factor analysis, in which, we assume that the $\mathbf{x}_i$ values come from a Gaussian distribution with covariance matrix $\mathbf{R}$, and that the noise vectors $\mathbf{n}$ come from a Gaussian with covariance $\mathbf{Q}$:

Model: Factor analysis		
$\mathbf{x}$	$\sim$	$\mathcal{N}(0; \mathbf{R})$
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \mathbf{Q})$
$\mathbf{y}$	$=$	$\mathbf{A}\mathbf{x} + \mathbf{b} + \mathbf{n}$

(7.37)

It should be clear that the PPCA model in the previous section is a special case of this, in which the noise is spherical (i.e. $\mathbf{Q} = \sigma^2 \mathbf{I}$), and $\mathbf{R} = \mathbf{I}$ as well.

(To be written.)

TODO: FA is a Gaussian

TODO: ambiguities

TODO: learning the scale

TODO: references to learning algorithms

7.2.7 How many dimensions should we choose?

In general, we do not know in advance the dimensionality of the low-dimensional $\mathbf{x}$ space should be. There are a number of simple heuristics commonly in use. These heuristics are used especially when PCA is used for dimension reduction, in this case, there may not be much penalty for being somewhat conservative and choosing a large number of dimensions. If PCA is the entire probability model, then choose too many dimensions could make the model overly "flexible," especially for conventional PCA. This is much less of an issue for probabilistic PCA and factor analysis, since they model Gaussian distributions.

Heuristics for choosing the number of dimensions are primarily based on inspecting the eigenvalues $\mathbf{\Lambda}$, since they measure the variance of the data in each of the $\mathbf{x}$ coordinates. Here are two simple heuristics:

- Plot the eigenvalues. Quite often, you might see an "elbow" or bend in the curve, suggesting that the dimensions past the elbow are just noise. Truncate at the elbow, i.e., if the elbow is the J-th dimension, then keep J PCA dimensions.

- Pick a ratio r of the amount of variance that you want to keep, e.g., 99%, and then choose the smallest number J of eigenvalues so that $\sum_{j=1}^{J} \lambda_j / \sum_{j=1}^{M} \lambda_j \geq r$, assuming that the eigenvalues are sorted in decreasing order.

There is a solution to estimating the correct number of dimensions, namely, by computing the maximum likelihood estimate of the number of dimensions. This computation is complex to derive; however, Minka has shown that this estimate is very effective and fast [2000b]. An even more general option is to perform Bayesian prediction using a PCA model (Chapter **??**), thereby choosing an "effective" dimensionality [Bishop 1998].

7.2.8 PCA as approximating a Gaussian

We can also arrive at an algorithm similar to conventional PCA by approximating a Gaussian distribution. As described in Chapter 11, Blanz and Vetter [1999], fit a Gaussian distribution to a collection of faces $\{\mathbf{y}_i\}$. However, due to the extremely-high dimensionality of faces, estimating a full covariance matrix would have been impractical (since it would have millions of entries, and would require millions of faces to be full-rank). Instead, they used the following PCA-like approximation. Recall, from Equation 6.44, that the negative log-likelihood of a set of data points $\mathcal{D} = \{\mathbf{y}_i\}$ according to a Gaussian is:

$$L(\mu, \mathbf{\Sigma}) = \sum_i (\mathbf{y}_i - \mu)^T \mathbf{\Sigma}^{-1} (\mathbf{y}_i - \mu)/2 + \frac{K}{2} \ln |\mathbf{\Sigma}| + \frac{KM}{2} \ln(2\pi) \tag{7.38}$$

where μ and $\mathbf{\Sigma}$ are the mean and the variance of the Gaussian, respectively. To estimate these parameters, we would normally use the data mean ($\bar{\mathbf{y}} = \sum_i \mathbf{y}_i / K$) and the data covariance ($\mathbf{S} = \sum_i (\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T / K$). However, suppose we desire a low-rank approximation to the covariance that will be cheaper to compute. It can be shown[7] that the low-rank estimator closest to the data covariance is given by the reduced eigenvalue decomposition:

$$\hat{\mathbf{\Sigma}} = \mathbf{A} \mathbf{\Lambda} \mathbf{A}^T \tag{7.39}$$

where $\mathbf{A}$ is an $M \times N$ matrix containing the first N eigenvectors of $\mathbf{S}$, and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1^2, ..., \lambda_N^2)$ is a matrix of the largest N eigenvalues of $\mathbf{S}$. $\mathbf{A}$ is orthonormal, so that $\mathbf{A}^T \mathbf{A} = \mathbf{I}$.

Given the estimated mean and covariance, suppose we wish to evaluate the likelihood of some new face $\mathbf{y}$. The exponent of the Gaussian can be rewritten as follows:

$$(\mathbf{y} - \hat{\mu})^T \hat{\mathbf{\Sigma}}^{-1} (\mathbf{y} - \hat{\mu}) = (\mathbf{y} - \hat{\mu})^T (\mathbf{A} \mathbf{\Lambda} \mathbf{A}^T)^{-1} (\mathbf{y} - \hat{\mu}) \tag{7.40}$$

$$= (\mathbf{y} - \hat{\mu})^T \mathbf{A} \mathbf{\Lambda}^{-1} \mathbf{A}^T (\mathbf{y} - \hat{\mu}) \tag{7.41}$$

$$= (\mathbf{A}^T (\mathbf{y} - \hat{\mu}))^T \mathbf{\Lambda}^{-1} (\mathbf{A}^T (\mathbf{y} - \hat{\mu})) \tag{7.42}$$

[7] Specifically, the optimal low-rank approximation to a matrix $\mathbf{S}$ — according to Frobenius norm — is given by the reduced SVD of a matrix. Moreover, because the data covariance matrix is necessarily symmetric and positive definite, the SVD is identical to the eigenvalue decomposition.

If we define $\mathbf{x} = \mathbf{A}^T(\mathbf{y} - \hat{\mu})$, then the above expression reduces to:

$$\mathbf{x}^T \mathbf{\Lambda}^{-1} \mathbf{x} = \sum_j \frac{x_i^2}{\lambda_i^2} \tag{7.43}$$

Note that the $\mathbf{x}$ values are exactly the low-dimensional coordinates computed by conventional PCA, and $\mathbf{A}$ and $\hat{\mu}$ is the same hyperplane as computed by PCA[8]. The model is a spherical Gaussian with respect to the $\mathbf{x}$ coordinates: $\mathbf{x} \sim \mathcal{N}(0; \mathbf{\Lambda})$.

The power of this approximation is that the Gaussian can be modeled in terms of the low-dimensional $\mathbf{x}$ coordinates. However, this approximation restricts points to precisely lie on the hyperplane.

[8] Aside from a scale factor of $1/K$ used when computing the covariance matrix.

Chapter 8

Non-linear regression: splines, RBFs, neural networks

We now return to the non-linear regression problem introduced in Chapter 2. In non-linear regression, our goal is to estimate a non-linear mapping $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$, where $\mathbf{x}$ is an input vector, $\mathbf{y}$ is an output vector, and $\mathbf{w}$ are parameters of the mapping. The learning problem is to estimate the parameters $\mathbf{w}$, from training data $\{\mathbf{x}_i, \mathbf{y}_i\}$; the prediction problem is to the values of $\mathbf{y}$ given a new value of $\mathbf{y}$. For now, I will use a basis function representation for f:

$$f(\mathbf{x}; \mathbf{w}) \quad = \quad \sum_{\ell=1}^{L} \mathbf{w}_\ell B_\ell(\mathbf{x}) \tag{8.1}$$

where $B_k(\mathbf{x})$ is the k-the basis function.

This problem can be placed in a Bayesian framework in the following way [Szeliski 1989]. We assume a model in which output vectors are produced as follows:

Model: Non-linear regression		
$\mathbf{w}$	$\sim$	$\mathcal{N}(0; \sigma_w^2 \mathbf{I})$
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \sigma^2 \mathbf{I})$
$\mathbf{y}$	$=$	$f(\mathbf{x}; \mathbf{w}) + \mathbf{n}$

(8.2)

where $\mathbf{w}$ encapsulates all curve parameters as a vector (e.g., $\mathbf{w} = [\mathbf{w}_1^T, ..., \mathbf{w}_L^T]^T$). In other words, we assume a Gaussian prior distribution over the weight vectors (i.e., smaller weights are more likely). We that the outputs are produced by applying the non-linear function f and then adding Gaussian noise. The prior on $\mathbf{w}$ is called a **weight decay prior**. Alternatively, we could use a smoothness prior over the function f, as discussed below.

Given a collection of data $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, and known values for the model parameters σ^2 and σ_w^2, we can estimate the model weights $\mathbf{w}$ by MAP, by maximizing:

$$p(\mathbf{w}|\mathcal{D}, \sigma^2, \sigma_w^2) \quad = \quad \frac{p(\mathcal{D}|\mathbf{w}, \sigma^2, \sigma_w^2) p(\mathbf{w}|\sigma^2, \sigma_w^2)}{p(\mathcal{D}|\sigma^2, \sigma_w^2)} \tag{8.3}$$

Because $\mathbf{w}$ and σ^2 are independent, we have $p(\mathbf{w}|\sigma^2, \sigma_w^2) = p(\mathbf{w}|\sigma_w^2)$. Additionally, we can disregard the denominator $p(\mathcal{D}|\sigma^2, \sigma_w^2)$, since it does not depend on $\mathbf{w}$. Maximizing the log-probability is equivalent to minimizing the negative log-probability:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= -\ln p(\mathbf{w}|\mathcal{D}, \sigma^2, \sigma_w^2) && (8.4)\\
&= -\ln \prod_i p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) - \ln p(\mathbf{w}|\sigma_w^2) && (8.5)\\
&= \sum_i \left(\frac{1}{2\sigma^2} ||\mathbf{y}_i - f(\mathbf{x}_i)||^2 + \frac{M}{2} \ln 2\pi\sigma^2 \right) + \frac{1}{2\sigma_w^2} ||\mathbf{w}||^2 + \frac{LM}{2} \ln 2\pi\sigma_w^2 && (8.6)
\end{aligned}$$

Notice that this objective function is equivalent to the least-squares objective function in Equation 2.5. Moreover, if, instead of the weight decay prior, we choose instead a smoothness prior (e.g., so that $p(\mathbf{w}|\sigma_w^2) \propto \exp(-\int ||\nabla f||^2 d\mathbf{x})$), then we would get Equation 2.4. In other words, *least-squares estimation is a MAP estimation principle.* This is the Bayesian derivation of least-squares estimation.

TODO: fix dimensionality terms

TODO: fill in gaps, define variables, define dimensionality

The above derivation sometimes leads people who work with least-squares methods to think that "Bayesian methods are just the same thing that you would do anyway." In the rest of this chapter and these notes, some of the power of the Bayesian approach should be become clear.

One point to note is that the only principled derivation of least-squares fitting is statistical (although one does not need to be Bayesian to obtain it). Without a statistical framework, it is unclear where the least-squares rule comes from, or why specifically the L2-norm should be used. From the above derivation, we can see that there is an assumption of Gaussian noise — if the noise is very non-Gaussian, then we would not expect least-squares to work well. A specific example of when we would not want to use least-squares (or Gaussian noise) is data that is corrupted with outliers. In this case, heavy-tailed distributions are more appropriate; the study of heavy-tailed distributions is called robust statistics.

On a more practical level, the Bayesian framework allows us to estimate the smoothness parameters. For example, suppose we know $\mathbf{w}$, and we wish to estimate the unknown variances σ^2 and σ_w^2, assuming uniform priors over these variables. Writing out the posterior distribution $p(\sigma^2, \sigma_w^2|\mathcal{D}, \mathbf{w})$ gives an objective that is equivalent to Equation 8.6, and the MAP estimates can be obtained in closed-form by solving $\partial\mathcal{L}/\partial\sigma^2 = 0$, $\partial\mathcal{L}/\partial\sigma_w^2 = 0$ in closed-form:

$$\begin{aligned}
\hat{\sigma}^2 &= \frac{1}{K} ||\mathbf{y}_i - f(\mathbf{x}_i)||^2 && (8.7)\\
\hat{\sigma_w^2} &= \frac{1}{LM} ||\mathbf{w}||^2 && (8.8)
\end{aligned}$$

TODO: double-check and fix dimension variables

In fact, we can even estimate all of the curve parameters simultaneously, i.e., estimate $\mathbf{w}$, σ^2, and σ_w^2. Again, it turns out that maximizing the posterior is equivalent to minimizing $\mathcal{L}$ above.

Since these are MAP estimation rules, overfitting may be a signficant concern; all of the problems of overfitting curves as discussed in Chapter 2 apply. This will become even more of a concern if we estimate the variances and the weights simultaneously. Furthermore, the fitting may still be sensitive to the number of basis functions used. The next chapter will discuss how to avoid all these problems.

8.1 Radial Basis Functions

Instead of using the weight decay prior, let us consider a prior that directly applies a penalty to the shape of the function. Moreover, we seek to directly estimate the function f. Similar to Equation 2.3, we use the objective function

$$E(f) = c_{fit} \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i)||^2 + c_{smooth} E_{smooth}(f) \tag{8.9}$$

By now, it should be clear that this corresponds to the negative-log-posterior of a function f, in which the negative-log-posterior is represented by the smoothness term. For example, we might choose $E_{smooth}(f) = \int ||\nabla f||^2 d\mathbf{x}$ or. $E_{smooth}(f) = \int ||\nabla^2 f||^2 d\mathbf{x}$.

Poggio and Girosi [1990] have shown that specific forms of this objective function can be optimized directly, without assuming a known functional form for f. Specifically, the solution has the following form:

$$f(\mathbf{x}) \quad = \quad \sum_i \mathbf{w}_i G(||\mathbf{x} - \mathbf{x}_i||) \tag{8.10}$$

where $G(r)$ is called a **radial basis function**, and $\mathbf{w}_i$ are vector-valued weights. This is simply another basis function representation, with a specific type of basis function. The optimal form of G depends on the smoothness term in Equation 8.9; a good choice is a Gaussian:

$$G(r) \quad = \quad e^{-\frac{|r|^2}{2\sigma^2}} \tag{8.11}$$

where σ^2 is a constant that must be determined by the user or by heuristics. Note that the $\mathbf{x}_i$ values correspond to each of the original training data points: we simply place a basis function around each training point. The number of basis functions is equal to the number of training data points.

Given the training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_1, \mathbf{y}_1)\}$ and the basis functions, we can solve for the weights. Our goal is to solve for weights $\mathbf{w}_i$ subject to the constraint $\mathbf{y}_j = f(\mathbf{x}_j) = \sum_i \mathbf{w}_i G(||\mathbf{x}_j - \mathbf{x}_i||)$ for all j. We can write these constraints in matrix form, defining the matrix $\mathbf{G}$ so that $\mathbf{G}_{i,j} = G(||\mathbf{x}_i - \mathbf{x}_j||)$, the matrix $\mathbf{Y} = [\mathbf{y}_1, ..., \mathbf{y}_N]$, and the matrix $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_N]$. Then the constraint is equivalent to:

$$\mathbf{Y} \quad = \quad \mathbf{WG} \tag{8.12}$$

and so the weights may be obtained in closed-form:

$$\mathbf{W} \quad = \quad \mathbf{YG}^{-1} \tag{8.13}$$

Michelli's theorem [] shows that $\mathbf{G}$ is invertible, provided all $\mathbf{x}_i$ points are distinct.

This approximation method in general is called **Radial Basis Functions** (RBFs), and was first introduced by Michelli [] as a scattered data interpolation procedure, and later adopted within machine learning.

RBFs typically give smooth and reasonable curves that exactly interpolate the data. However, they require that one define a suitable shape for the RBFs (equivalently, a suitable smoothness function); for example, if the $\mathbf{x}$ vectors contain both joint angles and absolute positions, then these two quantities may vary at quite different scales; we might want a more general basis function that measures Mahalanobis distance (e.g., of the form $e^{-\mathbf{x}_i^T \mathbf{A} \mathbf{x}}$ for some matrix $\mathbf{A}$). In general, there will be parameters inside the basis functions that are difficult to optimize without overfitting.

RBFs may be somewhat expensive if the training set is large, since one must keep around the entire training set to perform regression. An alternative is to choose a reduced set of basis functions, and then solve for the weights in a least-squares sense. In other words, this is a choice of basis functions with less basis functions than data points, and the weights are obtained by minimizing the usual least-squares objective function. The centers of the basis functions can be selected by heuristics such as clustering, or they can be optimized, but the algorithm may not be very robust [Bishop 1995].

8.2 Neural networks

Neural networks are widely used for solving regression problems. Their name reflects their origin in early theories of neuroscience, but they are no longer viewed as realistic models of the human brain. In my opinion, their mystique and hype overshadows the fact that they represent just another parametric functional form for data-fitting, albeit a useful one.

The simplest neural for regression is the "single-layer perceptron," which, in fact, is exactly identical to linear regression.

Neural networks become more interesting when there are multiple "layers." In fact, such "multi-layer perceptrons" have some advantages over basis-function representations. For example, RBFs scale very poorly with the number of input dimensions, whereas neural networks should theoretically be able to handle the extra dimensionality easily. A weight decay prior can be used to prevent overfitting, to a limited degree. Neural networks become much more useful with Bayesian prediction (Section **??**).

Neural networks are discussed in detail by Bishop [1995].

8.3 Problems with non-linear regression methods

While the above methods are widely used in practice they still suffer some of the limitations described in Chapter 2, namely, that there are a number of parameters to set — the strength and form of the smoothness prior, the noise variance, the number of basis functions, and so on. MAP estimation can estimate all of these unknowns, e.g., the smoothness weights can be estimated as described above; see [MacKay 1992] on correctly estimating the number of basis functions. However, in my opinion, the methodology of Gaussian Process regression (Chapter **??** is far more powerful and effective than any of the methods that I've described in this chapter, and avoids all of these difficulties. (Effeciency is somewhat of a concern, although acceleration techniques exist).

8.4 Unsupervised learning: Non-linear dimensionality reduction

In the previous chapter, we developed Principle Components Analysis (PCA), which is based on a linear mapping $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ in which the $\mathbf{x}$'s are unknown. PCA and PPCA are useful as dimension reduction algorithms and for learning PDFs, but are limited to data can is approximately linear. The next question to ask is: can we do the same thing with a non-linear mapping? The answer is yes.

Specifically, if we assume a non-linear mapping $\mathbf{y} = f(\mathbf{x}; \mathbf{w})$, it is possible to simultaneously learn the weights $\mathbf{w}$ and the low-dimensional coordinates $\mathbf{x}$; this provides a way of learning non-linear dimensionality

reduction, and non-Gaussian PDFs. Two examples are the Generative Topographic Mapping [Bishop et al. 1998] (which uses an RBF mapping), and non-linear autoencoders [DeMers and Cuttrell 1992; Kramer 1991] (which use a neural network mapping). In animation, Grzeszczuk [July 1998] use a neural network to fit a low-dimensional descriptions of an animal control systems, optimized for a specific motion task. Non-linear dimensionality reduction continues to be a very active research area.

TODO: mention principal curves

54

Chapter 9

Generative models and graphical models

TODO: absorb this chapter into inference

In all of inference, we use the following general strategy:

- First, we define a **generative model** (or "likelihood function") of how we believe observations are created.
- Second, we define a problem of interest, namely to estimate the probability of some unknown quantity, given known observations.
- Finally, we compute the desired value numerically.

The "generative model" is a probabilistic model in which every variable is sampled from some distribution. For example, one model of coin flipping from the previous chapter was:

Model: Coin-Flipping		
θ	$\sim$	$\mathcal{U}(0,1)$
$P(\mathbf{c} = \mathsf{heads})$	$=$	θ
$p(\mathbf{f} = \mathsf{heads} \mid \mathbf{c} = \mathsf{heads})$	$=$	$.7$
$p(\mathbf{f} = \mathsf{heads} \mid \mathbf{c} = \mathsf{tails})$	$=$	$.3$

(9.1)

Once we've defined the generative model,[1] we can define any inference problem (what is the probability of θ given many coin flips? what is the probability of given θ and $\mathbf{c}$?). We can then derive this probability by applying the basic rules of probability (such as the Sum Rule and Product Rule), rewriting the unknown in terms of known quantities (the measurements and the model). Morover, we use the same model for both learning and synthesis — if we learn (or design a probability model) for motion, then we can generate new motions by sampling from the model, or computing the most likely most from new constraints.

Explicitly defining a model separate from the problem statement and the algorithm is very important to keeping the algorithm clear. Usually, the model and problem statement is very simple, but the algorithm may be fairly complex. If you understand the model and the problem statement, then the details of the algorithm are much less important.

[1] Note that, in learning, the term "generative model" has a somewhat different meaning that it has had in graphics.

9.1 Graphical models

Graphical models are a convenient way to illustrate generative models (e.g., [Jordan 1998]). **(To be written.)**

56

Chapter 10

Gaussian Processes

TODO: decide what to put in this chapter

10.1 Gaussian Process regression

Let us now reconsider the problem of non-linear regression introduced in Chapter 2. We return to the problem on non-linear regression, in which we assume that there exists a functional mapping f between input vectors $\mathbf{x}$ and output vectors $\mathbf{y}$:

Model: Non-linear regression		
$\mathbf{w}$	$\sim$	$\mathcal{N}(0; \sigma_w^2 \mathbf{I})$
$\mathbf{n}$	$\sim$	$\mathcal{N}(0; \sigma^2 \mathbf{I})$
$\mathbf{y}$	$=$	$f(\mathbf{x}; \mathbf{w}) + \mathbf{n}$

(10.1)

where $\mathbf{w}$ are parameters of the mapping. Given some training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, we can then predict the output $\mathbf{y}$ for a given new value $\mathbf{x}$:

$$p(\mathbf{y}|\mathbf{x}, \mathcal{D}, \sigma_w^2, \sigma^2) \quad = \quad \int p(\mathbf{y}|\mathbf{w}, \sigma^2) p(\mathbf{w}|, \mathcal{D}\sigma_w^2) d\mathbf{w} \tag{10.2}$$

assuming for now that we know the variances σ_w^2 and σ^2. If we need to estimate a single value of $\mathbf{y}$ given $\mathbf{x}$, we could choose the maximum of the posterior distribution. It so happens that, for many choices of the mapping f — such as a linear combination of spline or Gaussian basis functions — this prediction can be computed in closed-form.

What is really remarkable is that, if we represent f as a linear combination of basis function, is that we can make predictions even in the limiting case of an *infinite* number of basis functions. These predictions can be done in closed form. Furthermore, we can reliable estimate the parameters σ_w^2 and σ^2 by maximum likelihood, provided that we have a reasonable quantity of data. As a result, we get a regression algorithm that subsumes many existing models (including B-splines, RBFs, two-layer neural networks, and Brownian motion), does not require any parameter tuning (e.g., choosing the number of basis functions or the variances, and and produces (in my experience) better results than any of the above methods. This model is called Gaussian Process regression, and was introduced by O'Hagan [1978] and Williams and Rasmussen

[1996]. See [MacKay 2003] for a detailed tutorial, and [Seeger 2004] for a survey of advanced topics related to GPs.

As before, we may also consider the "unsupervised" case, i.e., when the $\mathbf{x}$'s are unknown in advance — this yields a very effective non-linear dimensionality reduction technique [Lawrence 2003]. We have extended this in our recent work on learning for inverse kinematics [Grochow et al. 2004].

58

Chapter 11

Application: Statistical shape and appearance models

TODO: eigenfaces, moghaddam, girosi-poggi, cootes-taylor, blanz-vetter, allen, isard-blake, torresani-hertzmann

11.1 Shape and appearance models

(To be written.)

11.1.1 Face recognition with "eigenfaces"

(To be written.)(sirovich-kirby, turk-pentland, moghaddam)

11.1.2 Tracking and face detection with active contours

(To be written.)(cootes-taylor, isard-blake)

11.1.3 Face and body interpolation

(To be written.)(girosi-poggio, rose et al)

11.1.4 Unsupervised 3D face and body modeling

As an example, consider the head-shape modeling described by Blanz and Vetter [1999]. In this case, a single person's head is represented by a parameter vector $\mathbf{x}$, containing the 3D positions of a set of facial features and the colors of a texture map. Blanz and Vetter assumed that human head shapes and textures are "generated" by random sampling from a Gaussian PDF[1]. We are given a set of N head shapes, and would like to learn the parameters of the Gaussian (μ, ϕ). As described in the Chapter **??**, learning according the

[1] Due to the large size of the data vectors, Blanz and Vetter use PCA to represent the Gaussian PDF as described in Section 7.2.8.

Maximum A Posteriori principle requires computing the values of μ and ϕ that maximize $p(\mu, \phi|\mathbf{X})$ with respect to these variables. In other words, given the data $\mathbf{X}$, we would like to compute the parameters of the Gaussian that is *most likely* to have generated the data. We will further assume uniform priors. In this case, the MAP estimate is equivalent

The $L(\mu, \phi)$ can be viewed as an energy function to be optimized for μ and ϕ. Inspecting the terms of $L(\mu, \phi)$ can be enlightening. The first term measures the fit of the data to the model. Note that the first and second terms must be balanced to optimize ϕ: the first term prefers large covariances ($\phi \to \infty$), whereas the second term penalizes increasing ϕ. The second term can be thought of as a penalty for learning too "vague" a model — such a penalty is built-in to Bayesian learning methods in general [MacKay 2003]. We did not have to manually specify this penalty term — it is a consequence of the fact that the likelihood function is required to be a normalized PDF.

Once we have estimates of μ and ϕ, we have a description of how "likely" any given face model is. For example, suppose we wish to estimate a face shape from a given image of someone's face. We assume that the face was created by selecting some viewing and lighting parameters $\mathbf{V}$, rendering an image $\mathbf{I}$ of the face $\mathbf{x}$, and adding zero-mean Gaussian noise with variance σ^2. In other words,

$$p(\mathbf{I}|\mathbf{x}, \mathbf{V}, \sigma^2) = \prod_{(x,y)} \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}(\mathbf{I}(x,y) - \mathbf{I}_{rendered}(x,y,\mathbf{x}))^2}$$

where $\mathbf{I}_{rendered}(x, y, \mathbf{x}, \mathbf{V})$ represents a rendering of the face $\mathbf{x}$ at pixel (x, y) with pose and lighting parameters $\mathbf{V}$.

To solve for the head shape and pose, we wish to estimate unknowns $\mathbf{V}$ and $\mathbf{x}$ by maximizing $p(\mathbf{x}, \mathbf{V}|I, \mu, \phi)$. Assuming uniform priors on $\mathbf{V}$, and assuming that $\mathbf{x}$ and $\mathbf{V}$ are independent, we have

$$p(\mathbf{x}, \mathbf{V}|\mathbf{I}, \hat{\mu}, \hat{\phi}, \sigma^2) = \frac{p(\mathbf{I}|\mathbf{x}, \mathbf{V}, \sigma^2)p(\mathbf{x}|\hat{\mu}, \hat{\phi})p(\mathbf{V})}{p(\mathbf{I})} \tag{11.1}$$

Again, maximizing this is equivalent to minimizing the negative log-likelihood :

$$L(\mathbf{x}, \mathbf{V}) = -\ln p(\mathbf{x}, \mathbf{V}|\mathbf{I}, \hat{\mu}, \hat{\phi}, \sigma^2) \tag{11.2}$$

$$= (\mathbf{x} - \hat{\mu})^T \hat{\phi}^{-1} (\mathbf{x} - \hat{\mu})/2 + \frac{1}{2\sigma^2} \sum_{(x,y)} (\mathbf{I}(x, y) - \mathbf{I}_{rendered}(x, y, \mathbf{x}, \mathbf{V}))^2 \tag{11.3}$$

(Terms that are constant with respect to $\mathbf{x}$ and $\mathbf{V}$ have been dropped.) Observe that this energy function over head shapes includes two terms: a facial "prior" that measures how "face-like" our 3D reconstruction is, and an image-fitting term. Although we could have arrived at this energy function without thinking in probabilistic terms, the advantage of the Bayesian approach is that we learned the μ and ϕ parameters of the energy function from data — we did not have to tune those parameters. Optimizing this objective form is more difficult than the previous one, and requires an iterative solver [Blanz and Vetter 1999; Romdhani et al. 2002].

It would also be straightforward to learn the σ^2 parameter from one or more images by optimizing $p(\mathbf{x}, \mathbf{V}, \sigma^2|\mathbf{I})$. Writing out the terms of the negative log-likelihood in this case, we get an optimization similar to the above fitting, but, with σ^2 as a free parameter, and a penalty term proportional to $\ln \sigma^2$ that

penalizes large image noise estimates. If we do this, there are now *no parameters to tune in the energy function; all parameters are learned from the input data*. This is one of the many benefits of the Bayesian approach — without this approach, you might come up with an optimization similar to the above, but you would have the tedious and difficult task of manually tuning the parameters μ, ϕ and σ^2. (Again, the caveats regarding the need for adequate initialization in the training, and the need for adequate training data apply.)

Not only do they require less user effort, automatically-learned parameters often perform better in vision tasks than hand-tuned parameters, since setting such parameters by hand can be very difficult. The quadratic error function $L(\mathbf{x}, \mathbf{V})$ for faces may have thousands of parameters, which would be impractical to set by hand. For example, in a recent project on non-rigid modeling from video, we used maximum likelihood to estimate 3D shapes, non-rigid motion, image noise, outlier likelihoods, and visibility from an image sequence [Torresani and Hertzmann 2004]. We found that learning these parameters always gave better results than the manually-specified initial values.

Note that the probabilistic model is more expressive than an energy function, since it can be used to randomly generate data as well. For example, we can randomly sample heads by sampling from $p(\mathbf{x}|\hat{\mu}, \hat{\phi})$. Given some user-specified constaints on a shape model, we can sample heads that satisfy that constraint.

62

Chapter 12

Summary and Conclusions

TODO: More stuff goes here. Move caveats to the intro

12.1 How to design learning algorithms

(To be written.)

12.2 Caveats

A few words of caution:

- You can't learn something from nothing. Your algorithm only "knows" what (a) you tell it explicitly, and (b) its models can deduce from the data. If you learn a Gaussian PDF over XYZ marker data, you will not get a very useful model of human motion. You will get a much better model working from joint angles.
- As tempting as it is to use a learning algorithm as a "black box," the more you understand about the inner workings of your model, the better. If you understand your formalisms well, you can predict when it will and won't work, and can debug it much more effectively. Of course, if you find that some method works as a "black box" for a given problem, then it is still useful.
- On a related note, it always pays to understand your assumptions and to make them explicit. For example, human faces are not *really* generated by random sampling from a Gaussian distribution — this model is an approximation to the real process, and should be understood as such. (Making your assumptions explicit and discussing the advantages and disadvantages is also very important in communicating your results to other people.)
- There are times when trying to define formal problem statements and to properly optimize objective functions or posterior likelihoods can impede creativity. The graphics community has a long history of being driven by clever, powerful hacks. Sometimes, clever hacks end up being more useful than their formal counterparts. On the other hand, clever hacks can sometimes lead to deeper understanding of a problem, and more formal and principled generalizations that would not be possible with hacks alone.

12.3 Research problems

The future is bright for applying learning to graphics problems, both in research and applications to industry and art. In the future, I expect that we will see more examples of directly modeling in a Bayesian setting in graphics. I expect that some of the major themes of research will be:

- Designing good models and algorithms for various concepts used in graphics
- Novel synthesis and sampling algorithms for learned models
- Discovering which models work will for which problems
- Integrating learned models and learning algorithms with user interfaces
- Providing artistic control in a system that uses learning in some components
- Interactive and real-time learning and synthesis

64

Chapter 13

Further reading

Here are some of the books related to machine learning that I've found most helpful. Some of these books are available in entirety on the authors' home pages.

- *Information Theory, Inference, and Learning Algorithms,* by David MacKay [2003]. A thorough and up-to-date study of the main topics of modern machine learning and information theory, viewed from a Bayesian perspective. Very highly recommended.
- *Probability Theory: The Logic of Science,* by Edwin T. Jaynes, [2003]. A detailed development of probability theory from first princples, written by one of the main advocates of Bayesian methods in the sciences. This book was published posthumously. The first two chapters provide an excellent description of the philosophical background of the Bayesian worldview.
- *Neural Networks for Pattern Recognition,* by Christopher M. Bishop, [1995]. This is an extremely clear and well-written overview of the major topics in learning, including neural networks, mixture models, interpolation, and a number of other topics. Although this book is becoming a bit dated, I have found it immensely useful, and I recommend it highly as a general-purpose introduction to learning.
- *Computer Vision: A Modern Approach,* by David Forsyth and Jean Ponce, [2003]. A good overview of computer vision, including a number of sections devoted to learning methods in computer vision and image-based rendering.
- *Elements of Information Theory,* by Thomas M. Cover and Joy Thomas [1991]. The bible of information theory.
- *Pattern Classification,* by Richard Duda, Peter Hart, and David Stork [2001]. An overview of statistical learning, with a strong emphasis on classification problems.
- *Fundamentals of Statistical Signal Processing,* by Steven M. Kay. An introduction to statistical signal processing, and solutions to many estimation problems.

One of the best ways to learn about recent topics is to read the proceedings of the main learning conferences, including NIPS, ICML, and UAI. The NIPS and ICML proceedings are available for free online.

66

Bibliography

BARZEL, R., HUGHES, J. F., AND WOOD, D. 1996. Plausible motion simulation for computer animation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*.

BISHOP, C. M., SVENSÉN, M., AND WILLIAMS, C. K. I. 1998. GTM: The Generative Topographic Mapping. *Neural Computation 10*, 1, 215–234.

BISHOP, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press.

BISHOP, C. M. 1998. Bayesian PCA. In *Proc. NIPS 11*, 382–388.

BLANZ, V., AND VETTER, T. 1999. A Morphable Model for the Synthesis of 3D Faces. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 187–194.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. *Proceedings of SIGGRAPH 2000* (July), 183–192.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH 2000*, 219–228.

CHU, W., KEERTHI, S. S., AND ONG, C. J. 2003. Bayesian trigonometric support vector classifier. *Neural Computation 15*, 9, 2227–2254.

COVER, T. M., AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley-Interscience.

COX, R. T. 1946. Probability, frequency, and reasonable expectation. *American J. of Physics 14*, 1, 1–13.

DEMERS, D., AND CUTTRELL, G. 1992. Non-Linear Dimensionality Reduction. In *Proc. NIPS 5*, MIT Press.

DUDA, R. O., HART, P. E., AND STORK, D. G. 2001. *Pattern Classification*, 2nd ed. Wiley-Interscience.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable Controllers for Physics-Based Character Animation. In *Proceedings of SIGGRAPH 2001*.

FORSYTH, D. A., AND PONCE, J. 2003. *Computer Vision: A Modern Approach*. Prentice Hall.

FRIEDMAN, N., AND HALPERN, J. Y. 1995. Plausibility meaures: a user's manual. In *Proc. UAI*, 175–184.

GEMAN, S., AND GEMAN, D. 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. Pattern Anal. Machine Intell. 6*, 6 (Nov.).

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-Based Inverse Kinematics. *ACM Transactions on Graphics* (Aug.). To appear.

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. July 1998. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. *Proceedings of SIGGRAPH 98*, 9–20.

HALPERN, J. Y. 2003. *Reasoning About Uncertainty*. MIT Press.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image Analogies. *Proceedings of SIGGRAPH 2001*, 327–340.

JAYNES, E. T. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press. In press; http://omega.math.albany.edu:8008/JaynesBook.html.

JOJIC, N., AND FREY, B. 2001. Learning Flexible Sprites in Video Layers. In *Proc. CVPR 2001*.

JORDAN, M. I., Ed. 1998. *Learning in Graphical Models*. MIT Press.

KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. W. 1996. Reinforcement Learning: A Survey. *J. of Artificial Intelligence Research 4*, 237–285.

KRAMER, M. A. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal 37*, 2, 223–243.

LAPLACE, P.-S. 1814. *A Philosophical Essay on Probabilities*. Dover Publications.

LAWRENCE, N. D. 2003. Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data. *Proc. NIPS 16*.

LENGYEL, J. E. 1999. Compression of time-dependent geometry. In *1999 ACM Symposium on Interactive 3D Graphics*, 89–96.

MACKAY, D. J. C. 1992. Bayesian Interpolation. *Neural Computation 4*, 3, 415–447.

MACKAY, D. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

MAGNUS, J. R., AND NEUDECKER, H. 1999. *Matrix Differential Calculus: with Applications in Statistics and Econometrics*. Wiley Series in Probability and Statistics.

MATUSIK, W., PFISTER, H., NGAN, A., BEARDSLEY, P., ZIEGLER, R., AND MCMILLAN, L. 2002. Image-based 3d photography using opacity hulls. *ACM Transactions on Graphics 21*, 3 (July), 427–437.

MICHELLI, C. A. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation 2*, 11–22.

MINKA, T., 2000. Old and New Matrix Algebra Useful for Statistics. MIT Media Lab note.

MINKA, T. P. 2000. Automatic Choice of Dimensionality for PCA. In *Proc. NIPS 13*, MIT Press.

MINKA, T., 2001. Pathologies of Orthodox Statistics. Unpublished note. http://www.stat.cmu.edu/∼minka/papers/pathologies.html.

NEAL, R. M. 1996. *Bayesian Learning for Neural Networks*. No. 118 in Lecture Notes in Statistics. Springer-Verlag.

O'HAGAN, A. 1978. Curve Fitting and Optimal Design for Prediction. *J. of the Royal Statistical Society, ser. B 40*, 1–42.

PERLIN, K., AND GOLDBERG, A. 1996. IMPROV: A System for Scripting Interactive Actors in Virtual Worlds. In *Proceedings of SIGGRAPH 96*, 205–216.

PERLIN, K. 1985. An Image Synthesizer. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3 (July), 287–296.

POGGIO, T., AND GIROSI, F. 1990. Networks for Approximation and Learning. *Proceedings of the IEEE 78*, 9 (Sept.).

POOR, H. V. 1994. *An Introduction to Signal Detection and Estimation*, 2nd ed. Springer Texts in Electrical Engineering. Springer-Verlag.

RAO, R. P. N., OLSHAUSEN, B. A., AND LEWICKI, M. S., Eds. 2002. *Probabilistic Models of the Brain: Perception and Neural Function*. MIT Press.

RASMUSSEN, C. E. 1997. *Evaluation of Gaussian processes and other methods for non-linear regression*. PhD thesis, University of Toronto.

ROMDHANI, S., BLANZ, V., AND VETTER, T. 2002. Face Identification by Fitting a 3D Morphable Model using Linear Shape and Texture Error Functions. In *Proc. ECCV 2002*, 3–19.

ROWEIS, S. T. 1998. EM algorithms for PCA and SPCA. In *Proc. NIPS 10*, 626–632.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video Textures. *Proceedings of SIGGRAPH 2000* (July), 489–498.

SEEGER, M. 2004. Gaussian Processes for Machine Learning. *International Journal of Neural Systems 14*, 2, 1–38.

SOLLICH, P. 2002. Bayesian methods for Support Vector Machines: Evidence and predictive class probabilities. *Machine Learning 46*, 1, 21–52.

SORENSON, H. W. 1970. Least-Squares estimation: from Gauss to Kalman. *IEEE Spectrum 7*, 63–68.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

SZELISKI, R. 1989. *Bayesian Modeling of Uncertainty in Low-Level Vision*. Kluwer Academic Publishers.

TIPPING, M. E., AND BISHOP, C. M. Probabilistic principal component analysis. *J. of the Royal Statistical Society, Ser. B 61*, 3, 611–622.

TIPPING, M. E. 2001. Sparse Bayesian learning and the relevance vector machine. *J. of Machine Learning Research 1* (June), 211–244.

TORRESANI, L., AND HERTZMANN, A. 2004. Automatic Non-Rigid 3D Modeling from Video. In *Proc. ECCV 2004*. To appear.

WILLIAMS, C. K. I., AND RASMUSSEN, C. E. 1996. Gaussian Processes for Regression. In *Proc. NIPS 8*, MIT Press.