Informative Features in Vision and Learning

by

Archisman Rudra

A dissertation submitted in partial fulfillment of the requirements for the degree

of Doctor of Philosophy

Department of Computer Science

Graduate School of Arts and Science

New York University

January 2002

Davi Geiger

(Dissertation Advisor)

© Archisman Rudra

All Rights Reserved 2002

To dadu and dida

Acknowledgments

Many people helped me in completing this thesis. Bas Rokers, then at Rutgers and now at UCLA, performed the memory experiments. He and Zili Liu helped us understand the data and form hypotheses. As important as Bas' academic help was his friendship; and the noontime games of soccer. I am very grateful to him.

I would also like to express my deep gratitude to Davi Geiger and David Jacobs, for being great advisors and wonderful teachers, as well as for helping me go through the "third year blues".

I would like to thank Bud Mishra for being supportive while I wrote this thesis, his careful readings of the draft thesis, as well as detailed technical discussions.

I am also indebted to my fellow students at NYU: Gediminas Adomivicius, Henning Biermann, Robert Buff, Raoul-Sam Daruwala, Deepak Goyal, Hiroshi Ishikawa, Ian Jermyn, Fabian Monrose, Niranjan Nilakantan, Jong Oh, Ken Pao, Laxmi Parida, Toto Paxia, David Tanzer, Daniela Tulone as well as my current colleagues: Marco Antoniotti, Will Casey, Vera Cherepenski, Valerio Luccio, Marc Rejali, Joey Zhou for their friendship and camaraderie.

I would also like to thank my parents for their constant support and encouragement; through undergraduate as well as graduate school.

Most of all I would like to thank my closest friend and wife Mousumi for suffering

through the writing of the thesis and for keeping up my morale through everything.

Abstract

We explore the role of features in solving problems in computer vision and learning. Features captures important domain-dependent knowledge and are fundamental in simplifying problems. Our goal is to consider the universal features of the problem concerned, and not just particular algorithms used in its solution. Such an approach reveals only the fundamental difficulties of any problem. For most problems we will face a host of other specialized concerns. Therefore, we consider simplified problems which captures the essence of our approach.

This thesis consists of two parts. First, we explore means of discovering features. We come up with an information theoretic criterion to identify features which has deep connections to statistical estimation theory. We consider features to be "nice" representations of objects. We find that, ideally, a feature space representation of on image is the most concise representation of an image which captures all available information in it. In practice, however, we are satisfied with an approximation to it. Therefore, we explore a few such approximations and explain their connection to the information-theoretic approach. We look at the algorithms which implement these approximation and look at their generalizations in the related field of stereo vision.

Using features, whether they come from some feature-discovery algorithm or are hand crafted, is usually an ad hoc process which depends on the actual problem, and the exact representation of features. This diversity mostly arises from the multitude of ways features capture information. In the second part of this thesis, we come up

with an architecture which lets us use features in a very flexible way, in the context of content-addressable memories. We apply this approach to two radically different domains, face images and English words. We also look at human performance in reconstructing words from fragments, which give us some information about the memory subsystem in human beings.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Object representation is a problem that arises in many applications. We frequently deal with a collection of objects. An exact representation of these objects may be fairly complex. On the other hand, the way we use them is sometimes approximate and thus we may not need all the information carried by an exact representation. Let us consider a concrete example to understand the issues that might arise.

In computer vision and in speech understanding, the raw data is usually an array or arrays (both one-dimensional and 2-dimensional) of *intensities*. However, these numbers are not of primary interest. In the case of vision, our interest is to extract the scene from the image or set of images. In speech, our goal is to understand the words. It is useful, for this purpose, to represent the underlying signal in a highly compressed form: in terms of the so called features.

We have an intuitive understanding of what we mean by a feature. In many

cases of practical interest, we can even identify useful features by which we can represent the signal. For example, if the image is that of a set of polygons, an important set of features might be the corner points and the edges. We get an enormous compression by representing images in this way. Furthermore, we get a representation which is little affected by small amounts of noise in the image.

Even if our underlying image is not composed of a set of polygons, the so-called *natural image*, edges and corners constitute a good set of features in practice. Depending on the scale we view the image at, various textures have also been used. Thus, in our discussion, we will use the word feature very broadly to mean some quantity computed from the image which

1. lets us get a "good enough" approximation to it.

2. actually compresses the image so that the representation of the image using features is smaller than the naive representation.

The point of view we take is that features are not specific to the image per se, but rather to the task we use the image in. Thus, if we need the image in any algorithm, we can get a feature space representation of the image suitable for that algorithm. This representation depends on what information the image brings into the algorithm, so that when we cull away the inessential part of the image, we are left with a compact representation that is adapted to the problem at hand.

In practice, though, features are typically chosen in an ad hoc manner. In the first part of the thesis, we will explore how our point of view leads to a more systematic approach to feature selection.

Our criterion is applicable for a large variety of estimation tasks, particularly function estimation. However, we find that formulating a similar criterion for feature discovery can be difficult for some tasks. Thus, while we can find features for function estimation in a fairly direct way, a similar approach fails for epipolar line stereo. For stereo, we are left to fall back upon a quick and approximate criterion which also works for the function estimation problem.

1.2 Related Work

There has been a lot of work on finding features for specific problem domains. Maximum entropy approaches had been used earlier in the context of natural language processing by Berger, della Pietra and della Pietra ([22]) and subsequently been used extensively in that community. Texture modeling has also been a particularly rich area. Zhu et al ([125], [126]) have shown how to build a *Markov Random Field(MRF)* model for textures. Our work is closest to this in spirit in that both are based on looking at the behavior of entropy of different distributions on the objects under question.

To explain this approach, we assume that we are given a bank of K linear or non-linear filters $\{\phi^\alpha\}_{\alpha\in\{1,\ldots,K\}}$. Zhu et al assumes that the filter bank is such that all processing and understanding of images depends only on the response of the filters in the bank. Thus, if two images have the same response under this set of filters, they will be indistinguishable. Suppose also that $f(I)$ is the *unknown* probability distribution of the images I. The goal is to pick a "good" probability distribution on the set of images such that the expected values of the filter responses under

this estimated probability distribution is the same as the expected values under the actual probability distribution f. Thus, we consider approximating f from the set of probability distributions over the set of images I:

$$\Omega = \{p \mid \mathbf{E}_p(\phi^\alpha(I)) = \mathbf{E}_f(\phi^\alpha(I)) \quad \forall \alpha \in \{1, 2, \ldots, K\}\}$$

Among this set Ω, we consider those probability distributions p which locally maximize the entropy. These are of the form

$$p(I) = \frac{1}{Z} \exp\left[\sum_\alpha \lambda^\alpha \phi^\alpha(I)\right],$$

where the Lagrange multipliers $\Lambda = \{\lambda^\alpha\}$ are chosen so that the ϕ^α-s have the correct expected response.

Zhu et al advocate a "minimax" entropy approach for estimating densities.

For every subset of features

$$\{\phi^\alpha \mid \alpha \in S\}, \quad S \subseteq \{1, 2, \ldots, K\},$$

let us define the set of probability distributions that match the corresponding subset of expectations values:

$$\Omega_S = \{p \mid \mathbf{E}_p(\phi^\alpha(I)) = \mathbf{E}_f(\phi^\alpha(I)) \quad \forall \alpha \in S\}, \quad S \subseteq \{1, 2, \ldots, K\}$$

and choose the corresponding maximum entropy distribution:

$$\begin{aligned} p_S(I) &= \operatorname*{argmax}_{p \in \Omega_S} \mathbf{H}(p) \\ &= \frac{1}{Z} \exp\left[\sum_{\alpha \in S} \lambda^\alpha \phi^\alpha(I)\right], \end{aligned}$$

where as before the λ^{α}-s are chosen so that the expectations of the ϕ-s match those obtained from the distribution f.

The minimax approach of Zhu et al chooses the feature set which minimizes this maximum entropy. Of course, a minimum over all subsets occurs when the chosen subset is empty. Thus, to correct against this deficiency; as well as to achieve capacity control, this work restricts the minimization to subsets of a specific size k. Thus, feature selection is achieved by means of the equation:

$$S_k^* = \underset{|S|=k}{\operatorname{argmin}} \max_{p \in \Omega_S} \mathbf{H}(p)$$

The maximum entropy principle used in these papers lead to the use of exponential distributions for the estimated density. For the connection between exponential distributions and the maximum entropy principle, see e.g. the book by Cover and Thomas[31].

While conceptually clean, this approach suffered from a slow speed of learning. This deficiency follows from the *Gibb's sampling* algorithm used in this approach for learning the expectations of the filter responses which drive this algorithm.

Furthermore, the problem is formulated in a way that precludes the use of any knowledge we might have of the process generating the image. Thus, for many practical cases, we might view the generative process as being composed of items generated from some prior and then being corrupted by noise. As we will see later, knowing something about the noise process lets us design very efficient algorithms for feature selection.

As far as texture synthesis is concerned there has been more recent work to

synthesize more natural looking texture faster.(DeBonet [34], Efros [39], Levoy [120])

Sampling techniques to approximate distributions has a long history after Geman and Geman[48] used it in the context of image restoration. Their paper ([48]) also introduced the idea of using a Gibb's distribution to solve problems of optimization. Earlier Kirkpatrick [73] had introduced ideas from statistical physics to approximately solve combinatorial optimization problems.

Our techniques are also similar to finding best representation from an overcomplete set of basis functions. Indeed this problem is a slight generalization of the examples we treat here.

1.3 Overview of the Thesis

The thesis can roughly divided into two parts. In the first part we look at feature-discovery from an information theoretic viewpoint. Then, we change tack and look at how the use of features can actually help in solving problems.

1.3.1 Feature Discovery

We first review the relevant concepts from *statistical estimation theory* in chapter 2. A reader familiar with basic statistics can skip this material. There are plenty of textbooks which explain it in greater detail as well. For further references, one is encouraged to consult one such (e.g. Wilk's "Mathematical Statistics" [121]). Another source which explains the role of information theory in statistical estimation is the book by Kullback ([77]).

We view images as being generated from some random process. We use Shannon's information theory [108] to quantify the effectiveness of any function as a feature. We then link this to ideas from statistical estimation theory. This naturally leads to an information-theoretic criterion for feature selection. This is enunciated and explained in chapter 3.

Our criterion explains features in a way that is mathematically intuitive, and which matches our natural ideas of what constitutes a feature. In chapter 4 we apply our ideas to the problem of discovering features for the task of one-dimensional signal estimation, as well as for stereo. However, in spite of its mathematical elegance, discovering features in this way is algorithmically inefficient. Therefore we present an alternate criterion which approximates our original approach but leads to a quick discovery of features.

Later in the chapter we describe the stereo problem and discuss what it means to have features for this problem. We will find that by the very topology of the problem, it is difficult to have a very clean formulation of our information-theoretic criterion in this domain. However, we find that it is quite straightforward to apply the approximate criterion in this case as well.

1.3.2 Uniform and Efficient Use of Features

In all of this, our focus had been information-theoretic. We thus assume that we are able to efficiently extract all the information which a feature carries. Note that this is not the same as saying that we can do feature-discovery efficiently. The information-theoretic point of view assumes that the feature space representation

of an image carries all the information present in the image which is relevant for the task at hand. However, the algorithm needed to extract this information from the feature space representation can be arbitrarily complex.

In the second half of the thesis we examine how we can use the information contained in features in a systematic way. We do not really propose any measure of complexity for the extraction process. Rather, we look at the very specific problem of associative access from a memory of items. In spite of its seeming specificity, this problem lets us capture the essential core of the problem of efficient use of features: how to use the information latent in a feature space representation.

When we actually use any particular class of features, we are faced with the problem that features can potentially capture any kind of information. The challenge is to be able to utilize this information in a uniform way. Thus, we are not really concerned whether the actual features we use came from running some kind of feature-discovery algorithm, as in the first part of the thesis, or were basically hand-crafted for the particular problem at hand. All we really need is that the feature space provides a good match between the statistical properties of the items in memory as well as the properties of the cues that are likely to be presented to the memory system.

We describe such a high-level architecture in chapter 5. This is an iterative algorithm which lets us use the information from a vast class of feature space constructions in an essentially uniform way. The dependence on problem domain is captured through the choice of the feature space, while the way this space is used is kept invariant. When we implement such an architecture, we will need to

apply various optimizations to the recall process which might depend on the specific spaces involved.

Such an implementation is described in chapter 6 where we describe a memory system for storing registered images. We choose a linear feature space, so that the set of features span a low-dimensional linear subspace of the set of all possible images. Such subspaces are constructed in practice by means of the eigenvalue decomposition of some correlation matrix, and the technique goes by the name of *Principle Component Analysis* or *Karhunen-Loeve Analysis*. This choice of feature space leads to interesting implementation of the preceding ideas. By means of successive improvements in our basic iterative algorithm, we are led to the development of an iterative algorithm inside the feature space, thus leading to enormous savings in efficiency. We test the algorithm on a database of registered face images.

A radically different class of algorithms is obtained when we choose a discrete domain. A good example is the memory for storing English words. In this example, much of the information is stored in the statistical properties of the words and their substrings: syllables, trigrams etc. we describe such a memory system in chapter 7. We define several optimized versions of the general algorithm and compare their performances. We find that a simpler version of these algorithms is very similar to the *Baum-Welch* learning algorithm from the theory of *Hidden Markov Models*.

We also study human performance in this domain. While some parts of human memory performance can be explained by the details of actual structures of human memory; many of the characteristics can be inferred from statistical and information-theoretic structures of naturally occurring words. Our goal in this

chapter is to find out some of those information-theoretic bounds of human memory performance.

Chapter 2

A Quick Tour of Statistics

In the first part of our thesis, we look at the information-theoretic underpinnings of the feature-discovery process. The connection of feature-discovery with information-theory lies through the area of statistical parameter estimation. In this chapter, we recapitulate some of the concepts from this field.

2.1 Parameters and Statistics

Our approach is based on the statistical estimation of parameters. The general problem is as follows: let $\{x_1, x_2, \ldots, x_n\}$ be a set of n independent samples from some probability distribution $P(x;\tau)$. The distribution depends on some unknown parameter τ (which might be a vector parameter, i.e. there are several scalar parameters). Our task is to estimate the actual value of τ.

This is one of the central problems of statistics. In this section we will introduce some notions from statistical estimation theory. This material can be found in any

introductory textbook. We mainly concentrate on introducing the terms and ideas that will be of use subsequently; and give simple examples to illustrate the concepts.

To make the ideas clear, let us start with a simple example. Suppose we are dealing with the one dimensional normal distribution with known variance σ but unknown mean μ. Thus the *probability density function (pdf)* is of the form

$$P(x;\mu) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{2.1}$$

Each of $x_i, i = 1, 2, \ldots, x_n$ are random variables which follow this pdf. Our goal is to estimate the mean μ. We have the natural estimator

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

At this point we will not discuss the effectiveness of estimators; our purpose is just to give the definitions. Thus, we notice that $\bar{x}$ is actually a function of the n sample values x_i and so is a random variable in its own right. Any such function of the sample values is called a *statistic*. Thus $\bar{x}$ is a statistic. The value we are trying to estimate, μ, actually serves to specify the probability distribution. Such a quantity is called a *parameter*. To emphasize, statistics are estimators for parameters.

2.1.1 Classes of Estimators

We will need a ready vocabulary to identify various classes of estimators. For simplicity, we will introduce these classes here.

Let $x_1, x_2, \ldots, x_n$ be the n sample points.

Definition 2.1.1 (Unbiased Estimator) *An estimator $t(x_1, x_2, \ldots, x_n)$ for a parameter τ is called* unbiased *if the expectation of t equals τ. In symbols*

$$\mathbf{E}(t) = \tau$$

where the expectation is taken over the joint distribution of the n sample points x_i. If $P(x;\tau)$ is the dependence of the probability distribution on τ, then the above criterion can be written as an integral:

$$\int \int \cdots \int dx_n \, dx_{n-1} \cdots dx_1 \, t(x_1, \ldots, x_n) P(x_1, x_2, \ldots, x_n; \tau) = \tau$$

Using the linearity of expectation, we can easily verify that $\bar{x}$ is an unbiased estimator of the mean μ. To get an example of a biased estimator, we turn to the "obvious" estimator for variance:

$$s = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

If the samples are independent, we can show that this is a biased estimator. The corresponding unbiased estimator is given by

$$s_{unbiased} = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

Note that as $n \to \infty$, $s/s_{unbiased} \to 1$. Thus we say that s is an *asymptotically unbiased* estimator of variance. Here is the formal definition:

Definition 2.1.2 (Asymptotic Unbiasedness) *Let $t(x_1, x_2, \ldots, x_n)$ be an estimator for τ for all natural numbers n. t is said to be* asymptotically unbiased *if, for all values of τ, $\mathbf{E}(t) \to \tau$ as $n \to \infty$.*

Best Unbiased Estimators: Cramer-Rao Inequality

Unbiasedness of a statistic guarantees that on the average the estimate obtained from a sample gives the right answer. However, we usually need something stronger than that. We want the variability in the estimator to be small as well. As is customary, we use the variance of the statistic to measure its variability.

Thus, let t be an unbiased estimator for τ and let us look at the variance, $\mathbf{E}[(t-\tau)^2]$. Clearly, the smaller the variance, the better is the estimator t. However, for the given parameter τ there is an absolute lower bound for the variance of any unbiased estimator t. This is the celebrated *Cramer-Rao Inequality* stated below.

Theorem 2.1.1 (Cramer-Rao Inequality) *Let t be an unbiased estimator of τ. Then*

$$\mathbf{Var}(t) \;\geq\; \frac{1}{J(\tau)} \;,$$

where

$$J(\tau) \;=\; \mathbf{E}[(\frac{\partial f(x,\tau)}{\partial \tau})^2] \;=\; \mathbf{E}[-(\frac{\partial^2 f(x,\tau)}{\partial \tau^2})] \;,$$

and f is the pdf.

The proof basically looks at the random variable $v = \frac{\partial \ln f(x,\tau)}{\partial \tau}$ whose expectation is zero; and applies the *Cauchy-Schwartz inequality* to the pair of random variables v and t. The equality in the second line is an easy algebraic manipulation.

Definition 2.1.3 (Efficient Estimator) *Efficiency e of an estimator t is a measure of how close it approaches the ideal Cramer-Rao bound. Quantitatively,*

$$e \;=\; \frac{1}{\mathbf{Var}[t]J(\tau)}$$

An estimator is said to be efficient *if its efficiency is 1. An estimator is* asymptotically efficient *if* $e \to 1$ *as* $n \to \infty$. *Here* n *is the number of sample points.*

2.2 Bayes' Theorem and Bayesian Estimation of Density

Parametric statistics works by assuming a certain parametric form for the distribution function (or for the pdf) and estimating the values of the parameters from statistics computed from sample values. If we have certain ideas about the probability distribution of the values of the parameters themselves, we can use Bayes' theorem to obtain what is known as a Bayesian estimator.

Bayes' theorem lets one calculate one conditional probability, knowing another. It states that

$$\begin{aligned} \mathbf{Pr}(A = a|B = b) &= \frac{\mathbf{Pr}(B = b \mid A = a)\ \mathbf{Pr}(A = a)}{\mathbf{Pr}(B = b)} && (2.2) \\ &= \frac{\mathbf{Pr}(B = b \mid A = a)\ \mathbf{Pr}(A = a)}{\sum_{a \in A} \mathbf{Pr}(B = b \mid A = a)\ \mathbf{Pr}(A = a)} && (2.3) \end{aligned}$$

A Bayesian estimator of a parameter τ can be obtained when we have a knowledge of the prior distribution of τ. We think of the parametric distribution as a conditional $\mathbf{Pr}(x|\tau)$. We then use Bayes' theorem to calculate $\mathbf{Pr}(\tau|x)$ and use the computed distribution to somehow estimate the parameter τ. A Bayesian estimator provides an estimate which is good "on average".

2.3 Distances Between distributions: Kullback-Leibler Distance and Mutual Information

Bayes' theorem (equation 2.2 or 2.3) lets one estimate one distribution from sample data. When we are estimating parameters, we can use some simple distance function like the absolute value to measure the quality of the estimation. However, when we are trying to estimate whole probability distributions we need some distance measures between them.

One such measure is the L^2 norm : $|P_1 - P_2|_2^2 = \int (P_1(x) - P_2(x))^2 \, dx$.

We will use this measure when convenient. More frequently, however, we will use an information theoretic distance measure: the *Kullback Leibler Distance*, also called the *Relative Entropy*, defined below:

$$D(P_1 \,||\, P_2) \;=\; \int P_1(x) \log \frac{P_1(x)}{P_2(x)} \, dx \tag{2.4}$$

The distance is actually defined if P_2 is non-zero in the support of P_1, except for some set of measure 0. When the distributions in question are discrete, the integrals are, of course, replaced by summations. Note, that D is *not* symmetric in P_1 and P_2.

The connections to information theory are many. Let us give just one here. Let us be working in a finite domain, so the integrals are all finite sums. Let P_2 be the uniform distribution, U: $U(i) = 1/N$, where N is the size of the domain. Then, for any other distribution P,

$$D(P \,||\, U) \;=\; \log N \;-\; \mathbf{H}(P)$$

where $\mathbf{H}$ is the *Shannon entropy*.

From the properties of the Kullback-Leibler distance, we know that it is non-negative. Thus, the maximum value of the entropy of a distribution on N items is $\log N$.

2.3.1 Significance of the Kullback Leibler Distance

Here we will give a quick sketch of why Kullback-Leibler distance is a significant measure of distance between probability distributions.

Shannon [108] first made the connection between the entropy of a distribution P and the best possible compression achievable for a source S drawn from P. Essentially, $\mathbf{H}(P)$ is the minimum value of the average number of bits that we need to encode symbols from S. The Huffman code [58] treats the case when the symbols are drawn independently from some alphabet satisfying a probability distribution P. It generates a coding scheme which asymptotically approaches the Shannon bound.

However, we need to know the form of the distribution P. If we erroneously assume that the distribution of the symbols is Q, whereas the true distribution is P, then the best possible code we can design cannot achieve an average code length of $\mathbf{H}(P)$. Instead, we achieve a bound of $\mathbf{H}(P) + \mathbf{D}(P \mid\mid Q)$.

Thus, we will think of the *K-L distance* as the error made when we falsely assume a probability distribution other than the real distribution. This is reflected in other ways as well. For example, suppose we try to estimate the underlying distribution by drawing independent samples from it. Let the possible empirical distributions

be denoted by P_i. Then log probability of P_i is close to the K-L distance between P_i and Q.

2.3.2 Mutual Information

We get a special case of the preceding discussion when we consider the K-L distance between the joint probability distribution $\mathbf{Pr}(X, Y)$ of two random variables X and Y and the joint distribution obtained under the assumption of independence of X and Y: $\mathbf{Pr}(X)\mathbf{Pr}(Y)$. This particular value is called the *mutual information* between X and Y and is denoted by $\mathbf{M}(X, Y)$:

$$\begin{aligned} \mathbf{M}(X,Y) &= \mathbf{D}(\mathbf{Pr}(X,Y) \,||\, \mathbf{Pr}(X)\mathbf{Pr}(Y)) && (2.5) \\ &= \int_{X,Y} \mathbf{Pr}(X,Y) \log \frac{\mathbf{Pr}(X,Y)}{\mathbf{Pr}(X)\mathbf{Pr}(Y)} && (2.6) \end{aligned}$$

Indeed $\mathbf{M}(X, Y)$ measures the degree of independence between X and Y. It vanishes when they are independent and is otherwise positive.

2.4 Sufficient Statistics

We now introduce the idea of a *sufficient statistic*. Let us look at the formula for $\bar{x}$ as an estimator of the mean μ:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Thus the estimator t of a parameter τ is some function $t(x_1, x_2, \ldots, x_n)$ of the sample points $x_1, x_2, \ldots, x_n$. Clearly, one desirable property of the function t is that it should not depend on the order of the variables x_i. (This is true for most

sampling schemes.) Furthermore, we need to give a separate function whenever n, the number of sample points, changes.

These considerations hold good even in the more general case of estimating the probability distribution of X from the sample points x_i. In this case the estimator, π, of the probability distribution, $\mathbf{Pr}(x)$, can be written in the form $\pi(x;\ x_1, x_2, \ldots, x_n)$. The comments made above for t hold good for π as well.

It is therefore desirable to be able to express the estimated probability distribution in terms of a finite number k, not depending upon the sample size n, of sample characteristics:

$$t_i \ = \ f_i(x_1, x_2, \ldots, x_n), \quad i = 1, 2, \ldots, k$$

such that dependence of the estimated distribution on the sample points factor through the t_i.

The notion we want to capture is that all the statistical information about τ that is contained in the sample $x_1, x_2, \ldots, x_n$ can be described by giving the k numbers

$$\begin{aligned} t_1 &= f_1(x_1, x_2, \ldots, x_n) \\ t_2 &= f_2(x_1, x_2, \ldots, x_n) \\ &\vdots \\ t_k &= f_k(x_1, x_2, \ldots, x_n). \end{aligned}$$

This can be ensured by demanding that the conditional distribution $\mathbf{Pr}(x_1, x_2, \ldots, x_n \,|\, t_1, t_2, \ldots t_k)$ be independent of the parameter τ. Writing the

conditional probability as a ratio of joint probabilities, our condition is

$$\frac{\mathbf{Pr}(x, t(x))}{\mathbf{Pr}(t(x))} \quad \text{is independent of } \tau, \tag{2.7}$$

where we have written x for the complete set of samples $x_1, x_2, \ldots x_n$ and likewise for the t's.

We now note that since the t's are functions of the x's, their values are uniquely known when the values of x's are known as well. Thus, $\mathbf{Pr}(x, t(x)) = \mathbf{Pr}(x)$. We write P for the quantity in 2.7. Making the dependence on τ explicit, and expanding out the subscripts in the x's and t's, we get the classical definition of a *Sufficient Statistic*:

Definition 2.4.1 (Sufficient Statistic) *The set of functions (statistics)* $t_i = f_i(x_1, x_2, \ldots, x_n)$ *are said to be* sufficient *for the class of densities* $\mathbf{Pr}(x; \tau)$*, indexed by the parameter* τ*, if, for all* τ*, the joint density* $\mathbf{Pr}(x_1, x_2, \ldots, x_n; \tau)$ *of the sample can be represented in the form*

$$\mathbf{Pr}(x_1, x_2, \ldots, x_n; \tau) = \mathbf{Pr}(t_1, t_2, \ldots, t_k; \tau) P(x_1, x_2, \ldots, x_n) \tag{2.8}$$

In other words, the joint density $\mathbf{Pr}(x_1, x_2, \ldots, x_n; \tau)$ is decomposed into a product of two terms. One term does not depend on the parameter(s) τ of the distribution and is the conditional distribution of the sample given the values of the statistics. The other is simply the distribution of $t_1, t_2, \ldots, t_k$, and depends on τ, but (naturally) not on the sample values $x_1, x_2, \ldots, x_n$.

2.4.1 Discussion

The parameter(s) τ serves to pick the correct probability distribution on the random variable X. Thus, we can think of τ to be parameterizing a family of probability distributions for X, which we will denote by $\mathbf{Pr}_\tau$. In this notation equation 2.8 states that $\mathbf{Pr}_\tau(x)/\mathbf{Pr}_\tau(t(x)) \equiv \mathbf{Pr}_\tau(x \mid t(x))$ is independent of τ. Using Jensen's inequality, we can show that

Theorem 2.4.1 *If τ_1 and τ_2 are two possible values of the parameter specifying the probability distribution of X, and t is any function, then*

$$\mathbf{D}(\mathbf{Pr}_{\tau_1}(x) \mid\mid \mathbf{Pr}_{\tau_2}(x)) \geq \mathbf{D}(\mathbf{Pr}_{\tau_1}(t(x)) \mid\mid \mathbf{Pr}_{\tau_2}(t(x))) \tag{2.9}$$

with equality when $\mathbf{Pr}_\tau(x)/\mathbf{Pr}_\tau(t(x))$ is independent of τ. (See, for example Kullback [77].)

We can think of the quantity $\mathbf{D}(\mathbf{Pr}_{\tau_1} \mid\mid \mathbf{Pr}_{\tau_2})$ as a measure of distinguishability of τ_2 from τ_1. To help us explain this notion further, let us assume that the true value of the parameter τ is τ_1. Then, the above KL-divergence is the difference in average log-likelihood values computed according to the distributions $\mathbf{Pr}_{\tau_1}$ and $\mathbf{Pr}_{\tau_2}$ respectively. It thus is a measure of our ability of distinguishing the incorrect value τ_2 from the correct value τ_1 if we estimate parameters using the maximum likelihood method.

Thus, for most statistics we lose some ability to distinguish different parameter values when we use the values of some statistic, instead of the original sample values. Only if we use a sufficient statistic do we not lose any distinguishing power.

This is the sense in which sufficient statistics capture the entirety of the dependence of the parameter value of a probability distribution on the sample points.

Algorithmic View of the Information Content

We can also explore a sufficient statistic's information capturing power in a more algorithmic way. To understand this, we think of the hypothetical experimenter $\mathcal{C}$ who knows the values of the sample points $x_1, x_2, \ldots, x_n$ versus another experimenter $\mathcal{T}$ who is shown only the values of the sufficient statistic t. We show that we can simulate the computations of $\mathcal{S}$ using the operations of $\mathcal{T}$, so that the probability distribution of the final result is the same, no matter what the actual parameter values for τ are. This shows that a sufficient statistic does not lose any information.

We will not give a formal proof, but will confine ourselves to an illustrative example. Thus, suppose $\mathcal{S}$ computes the numerical function f. Then, if $\mathcal{T}$ produces values from the conditional distribution $\mathbf{Pr}_\tau(f(x)|t(x))$, the probability values stay the same. For more complicated cases, where the operations of $\mathcal{S}$ are described by some algorithm expressed in some programming language, we replace all function evaluations with sampling from the above probability distribution. This gives a probabilistic simulation of the computations of $\mathcal{S}$ using computations (and input data) of $\mathcal{T}$.

If we insist that the estimator for $\mathcal{T}$ be deterministic, the above simulation would not work. However, if we confine ourselves to the class of unbiased estimators, we can still simulate the computation of $\mathcal{S}$ using operations from $\mathcal{T}$. Thus we can start with an unbiased estimator, r for $\mathcal{S}$ and get an estimator for $\mathcal{T}$ by using $r' =$

$\mathbf{E}[r(x)|t(x)]$. Using the Radon-Nikodym theorem, it can be shown that $\mathbf{Var}(r') \leq \mathbf{Var}(r)$. See, for example, the paper by Halmos and Savage ([53]). Thus, sufficient statistics are efficient for the class of unbiased estimators.

2.4.2 Markov Chains and the Data Processing Inequality

There is another information-theoretic way of viewing sufficient statistics. This is using the idea of *Markov Chains*

Definition 2.4.2 *Let X, Y and Z be three random variables with the property that*

$$\mathbf{Pr}(\ z\ |y,x)\ =\ \mathbf{Pr}(z|y)$$

i.e. Z is conditionally independent of X, given Y. Or, in other words, all the dependence of Z on X is what comes through Y. Then these 3 random variables is said to form a Markov Chain*; and is denoted as $X \longrightarrow Y \longrightarrow Z$. In general, if we have a sequence of variables, $X_1, X_2, \ldots, X_i, \ldots$, such that the distribution of X_i given all the previous ones is the same as its distribution given only X_{i-1}; then that sequence is called a* Markov Chain.

The name honors the Russian mathematician Markov who studied the letters of the alphabet arising in any natural text as a Markov Chain with conditional probabilities controlling the transition from one letter to another.

Theorem 2.4.2 (Data Processing Inequality) *If X, Y and Z form a Markov Chain then $\mathbf{M}(X;Y) \geq \mathbf{M}(X;Z)$*

Note that if Z is a function $f(Y)$ of Y, then the corresponding sequence is always a Markov Chain.

Let τ be the relevant population parameter, or parameters; X be the random variable denoting the sample (the whole sample, as a very big random variable); and $t(X)$ be the vector of statistics calculated from X. Then, the sequence τ, X, $t(X)$ forms a Markov Chain in the sense that the value of $t(X)$ depends only on the value of X and not on the actual parameter value τ. We would like to apply the Data Processing Inequality to this "chain". In the Bayesian framework this is acceptable, as we have a prior distribution on τ and the possible $\mathbf{Pr}_\tau$-s are viewed as conditional distributions of X given τ. Otherwise, our definitions as well as deductions will hold for any prior distribution on τ. In other words, we will not change the priors.

The equality case of the Data Processing Inequality:

$$\mathbf{M}(\tau; X) \;=\; \mathbf{M}(\tau; t(X))$$

occurs when t is a sufficient statistic.

Note, that the Data Processing Inequality is an "average" result. t is specified in advance, and does not depend on the X we actually see. The result is averaged over the possible sample and parameter values. In fact, the data processing inequality tells us that no matter what statistic we use, we cannot get more information about τ than the quantity $\mathbf{M}(\tau; X)$, which depends on both the process of sampling, as well as the distribution of τ.

2.5 Summary

In this chapter we introduced the basic concepts from statistical estimation theory. In particular, we introduced the notion of a parameter and a statistic to estimate that parameter. We also defined an unbiased estimator; and, using the Cramer-Rao inequality, defined the efficiency of an estimator.

We also defined some of the information-theoretic quantities:

- *Entropy*
- *Kullback-Leibler distance*
- *Mutual Information*

We also explained the connection between the KL distance between two distributions and the error made in confusing one for the other in the setting of maximum likelihood estimation.

We then introduced the central statistical notion that we will need, that of a sufficient statistic. A sufficient statistic for a class $\mathcal{C}$ of distributions carries exactly the information needed to distinguish a single distribution $P \in \mathcal{C}$ from the other distributions in $\mathcal{C}$.

None of this is new but this introduction serves as a jumping off point to the next chapter where we will explore the connection between features and sufficient statistics. This connection is one of the original contributions of this thesis.

Chapter 3

Features and Sufficient Statistics

At this point we are ready to describe the connection between informative features and sufficient statistics. To understand this connection better, we first discuss in a very general setting what features are, and what role they play in any image processing application.

3.1 What are Features

In any vision task, we have one or more images as inputs. In the subsequent discussion, let us assume, for simplicity, that we have a single image I. The way this image is used can be summarized by some algorithm $\mathcal{A}$ which takes in the image I as input and produces some output or possibly affects the world in some way.

The essential idea is the following: if we can capture the information present in the image I that is relevant to the algorithm $\mathcal{A}$ then this information represents

the complete way the image I can affect the running of $\mathcal{A}$ and thus is a concise but complete representation of the image I. Here we take the philosophical position that if two (possibly completely different) representations affect the algorithm $\mathcal{A}$ in the same way, they represent the same object, at least as far as $\mathcal{A}$ is concerned. In our example, we are thinking of $\mathcal{A}$ as the algorithm which represents how I is used.

The main idea is that the input image (or images) in a computer vision problem contains a lot of extraneous (junk) information. The task of most algorithms is to get rid of the irrelevant information and represent the input, either explicitly or implicitly, as the collection of only the relevant information. This is the feature-space representation.

Clearly, which part of the information in I is relevant to our task depends on the exact details of the algorithm $\mathcal{A}$. Analyzing this dependence is very complicated and we will not attempt it in this thesis. Rather, we concentrate on features that are more *generic*.

3.1.1 Model Estimation and Generic Features

To make progress we assume that the image I is generated from some model by letting it interact with some noise. We thus assume that the noise does not carry any useful information, though we do not assume that it is additive.

Thus, let $\mathcal{M}$ be the set of possible models. We could assume there is a prior distribution defined on the $\mathcal{M}$, which would then determine the likelihood of each model. This would place us in a Bayesian setting. However, our subsequent discussion is independent of this decision. What we really assume is that for each model

$m \in \mathcal{M}$, there is a probability distribution for the possible values of the image I. In the Bayesian case, this distribution is the conditional distribution $\mathbf{Pr}(I|m)$. This collection of probability distributions essentially determines the properties of the noise that combined with the model to generate I.

In this setting, let us look at any image processing algorithm $\mathcal{A}$ that uses the image I as input. No matter how we use I, it is clear that the quantity that really matters is the model m that generated I. This is almost true by definition : the model carries all the information about the world needed to generate I, the noise carries no useful information. Thus, we can factorize the action of $\mathcal{A}$ into two parts:

1. First, from I, estimate the model $m \in \mathcal{M}$.

2. Then, knowing m, simulate the actions of $\mathcal{A}$, by simply generating the image I according to the appropriate probability distribution and then running A on it.

Thus, if we have features which capture all the information about m that is present in I; these will lead to improvement in performance for all other algorithms as well. In this sense, we can say that the features for model estimation are generic.

3.1.2 Model Estimation as a Fundamental Task: An Example

We now look at a concrete example where we can apply the preceding framework of parametric estimation. We show that many fundamental problems in *computer vision* can be reformulated as problems of estimating the parameters of a parametric family of probability distribution.

Eigenfaces

Perhaps the best illustration of this point of view is the eigenface representation of face images. Turk and Pentland [115] applied Principal Component Analysis (PCA) to a set of face images, suitably registered, and showed that keeping the face images corresponding to a few eigenvalues leads to good learning performance. Earlier Kirby and Sirovich ([72]) used the eigenvalue decomposition approach in a similar domain and named the obtained eigenvectors *eigenfaces*.

Typically a few tens of eigenfaces can be used. What this showed was that the space of suitably registered and scaled face images approximately followed some normal distribution. The PCA analysis lets one estimate the parameters of this distribution. Thus the task of learning features in the set of face images reduces to estimating parameters in some (high dimensional) parametric family of probability distributions.

3.2 Different Kinds of Models

The example of eigenfaces was special, in that a uniform model of moderate complexity was enough to explain the whole image. Thus, we pictured the whole image to be a random Gaussian vector distributed in a very flat ellipse (i.e. only a few principal components explain most of the variance). This is just one of several kinds of models that arise in a framework of parametric estimation of model parameters.

3.2.1 Homogeneous Models

In this approach, the whole image I is a (high dimensional) random variable generated from some probability distribution. Thus in the eigenface example we viewed the registered face image as a normal distribution with some unknown mean and correlation matrix. In many applications we do have strong grounds for positing that the probability distribution of images belongs to some (particular) parametric family.

3.2.2 Gluing together Images Generated from Parametric Families

However, sometimes we do not have strong grounds to posit nice parametric distributions for the whole image. In these cases it is often preferable to assume parametric distributions for smaller parts of the image and somehow glue these distributions together. We need to make some probabilistic assumption to justify these "gluing" operation. The simplest kind of assumption is to impose some kind of independence of images on disjoint domains. This leads to very simple algorithms, but fails to capture any dependency between images in disjoint regions. We have one example of dependency in the case of face images. If we detect an eye in one location, we have a strong expectation of detecting another eye close to it in the correct (horizontal) orientation.

We can also use more complicated assumptions. One popular compromise between complexity of the gluing process and the class of dependencies captured by it is to assume that the underlying process is *Markovian*. This means that given any simply connected domain, the inside of the domain is conditionally independent of

the outside, given the pixels on the boundary. In the one dimensional case, this leads to an optimization problem solvable using a simple dynamic programming approach. This algorithm is called the *Viterbi algorithm* in the *Hidden Markov Model* literature.

Of course, in the formal sense, the second alternative is merely a special case of the first. We choose to delineate them because the algorithms that result from the first approach are more homogeneous. The second approach leads to more flexible architecture for estimation. However, the number of parameters we need to estimate increases. This could lead to non-robust estimates.

3.3 Model Estimation Problem & Its Features

After these preliminaries, let us get back to the discussion of what constitutes a good feature. As explained in subsection 3.1.1 we will concentrate on finding functions ϕ of the image which are good features for the model estimation task. Thus, ϕ is a function (or set of functions) such that the input image I carries as much information about the particular model that generated it, as does $\phi(I)$.

To understand what this means we first define explicitly what the model estimation problem is. In the subsequent discussion, we assume the image I to be the observed value of some random variable. [1] As in subsection 3.1.1, I is generated from another random variable, m, the model. The models m come from a set $\mathcal{M}$, and occur according to a given probability distribution $\mathbf{Pr}(m)$. In a Bayesian

[1] Unless there is some chance of misunderstanding we will refer to the random variable as well as a particular realization by the same symbol I.

framework, $\mathbf{Pr}(m)$ is called a prior probability distribution.The prior probability $\mathbf{Pr}(m)$ tells us about the model we assume for the world.

Furthermore, for every $m \in \mathcal{M}$, we are given a conditional distribution, $\mathbf{Pr}(I|m)$ over the possible values of the image I.This "generative model" gives the characteristics of the noise that is encountered in measuring the world.

Statement of the Model Estimation Problem: Given I, estimate $m \in \mathcal{M}$.

3.3.1 The Obvious Approach: MAP

The most obvious way to estimate the model is the *Maximum A Posteriori Probability (MAP)* method. We can, in principle, compute the probability of model, m, given the image, I using the Bayes' formula, equation 2.3:

$$\mathbf{Pr}(m|I) = \frac{\mathbf{Pr}(I|m)\ \mathbf{Pr}(m)}{\sum_m \mathbf{Pr}(I|m)\ \mathbf{Pr}(m)}$$

In the MAP approach we choose $m \in \mathcal{M}$ that maximizes this "posterior" probability. The denominator in the right hand side of this equation is simply the probability, $\mathbf{Pr}(I)$, of the variable I. For the MAP estimator, we can ignore the denominator, as it does not depend on m.

We can reformulate the above maximization problem as a cost minimization problem if we take the negative logarithm of both sides:

$$-\log \mathbf{Pr}(m|I) \ = \ (-\log \mathbf{Pr}(I|m)) \ + \ (-\log \mathbf{Pr}(m)) \ + \ F(I), \qquad (3.1)$$

where $F(I)$ is a function of I and does not depend on m. Ignoring this term in the cost function, we get the following cost function for our model selection problem:

$$\mathcal{C}_I(m) \equiv (-\log \mathbf{Pr}(I|m)) + (-\log \mathbf{Pr}(m)) \tag{3.2}$$

$$= \mathcal{D}_I(m) + \mathcal{P}(m) \tag{3.3}$$

where $\mathcal{D}$ is the "data" term and $\mathcal{P}$ is the "prior" cost.

Thus, the cost function for the MAP estimation procedure consists of two terms, one (the data) depending on the image I, the other (the prior) independent of I. In the next subsection we will assume that our models have a little bit more structure and see what these cost functions look like in practice.

3.3.2 A More Detailed Look at MAP Estimation

In many problems we minimize a cost function which is a sum of parts dependent on the local value of image with a part which connects neighboring pixel values. This is the case with MAP estimation of models, if the set $\mathcal{M}$ of models has some structure. Thus, as happens very frequently, we assume that the models m are also vectors in the same space as the image I; and that the image I is formed by a noise process which acts independently on each pixel. Thus, if we denote the i-th pixel of I by I_i and of m by m_i; and if we are given the conditional distributions $\mathbf{Pr}(I_i|m_i)$, then the generative model states that

$$\mathbf{Pr}(I|m) = \prod_i \mathbf{Pr}(I_i|m_i)$$

Under these conditions, the data term in the cost function is a sum of local costs:

$$\begin{aligned}\mathcal{D}_I(m) &\equiv -\log \mathbf{Pr}(I \mid m) \\ &= \sum_i -\log \mathbf{Pr}(I_i \mid m_i) \\ &= \sum_i \mathcal{L}_{I_i}(m_i)\end{aligned}$$

The prior model usually is a kind of continuity condition; stating that pixels near each other should have values that are close to each other. This is usually formulated using a neighborhood structure which determines when two pixels i and j are neighbors and a prior cost function (as in equation 3.3) of the form

$$\mathcal{P}(m) = \sum_{i,j \text{ neighboring}} \mathcal{T}(m_i, m_j),$$

where $\mathcal{T}$ is some distance function measuring the distance between m_i and m_j.

Thus, we try to minimize a cost function of the form:

$$\mathcal{C}_I(m) = \sum_i \mathcal{L}_{I_i}(m_i) + \sum_{i,j \text{ neighboring}} \mathcal{T}(m_i, m_j)$$

where $\mathcal{C}$ is the total cost, $\mathcal{L}$ is the local cost and $\mathcal{T}$ is the joint cost of two neighboring regions; and the subscripts denote which input variable the corresponding cost depends on. [2]

There are several observations we can make at this point. Firstly, cost functions of this form arise in many problems of computer vision and learning, most notably in the context of smoothing. In many such cases there is no explicit probabilistic model. However, we can always build a probabilistic model which corresponds to

[2] $\mathcal{T}$ is supposed to stand for transition cost.

these cost functions. In the constructed model, $\mathcal{T}$ will be viewed as a prior (cost) on the set of models, and $\mathcal{L}$ as the data cost.

Secondly, the index variables i and j which we use in the expression for the cost function need not be pixel locations. In many cases I_i (or m_i) is the output of some low pass filter centered about each point of the image. In fact, we can choose them to be responses to filters taken from some filter bank. This is equivalent to looking at the image in some transformed space (feature space). This is very helpful in the context of synthesis of textures (See Zhu et al [125], Efros et al [39], or DeBonet [34]).

In most of these cases, we still have a notion of the neighborhood structure which is related to the neighborhood structure in the underlying image, but in principle, we can have a completely different notion of neighborhood. This neighborhood structure certainly affects the algorithms we use to calculate expectation values and estimate parameters. Later we will look at these problems in the case of epipolar line stereo, where we will see a novel neighborhood structure.

Sampling the Image

Actual estimation of densities in this framework usually involves calculating the expectations of various functions (such as local variances) of the distribution.

There are two approaches to estimating densities using cost functions of this form, depending on how these expectations are estimated (or calculated):

1. We can try to compute the expectations directly, either numerically or symbolically.

2. Or, we can estimate the necessary quantities by sampling from the image itself. Thus, to estimate the background illumination of a scene, we might choose, say, a hundred pixels randomly, and use the average intensity at these points as the value of the background illumination.

The first approach depends crucially on our ability to compute the expectations; or even some good approximation to them. This usually strongly depends on the underlying topology, as well as the specific properties of the cost functions in use (like convexity etc.)

In the second approach, the problem is that if we view the whole image as a single random variable, we have just one sample from which to estimate the quantities. The property that we need is *ergodicity*:

3.3.3 Ergodic Processes

Definition 3.3.1 *A random field (or process) is said to be* ergodic *if its ensemble averages equal its space (or time) averages.*

This means that to compute the mean values at a particular region of the field, we can use two approaches.

1. We can use many instances of the field and compute the mean values using this set of instances.

2. We can use a single instance of the process and average over different regions.

In an ergodic process these two different methods are guaranteed to give the same result.

Sampling the image to produce estimates for the probability distribution has been used by many authors. Notably Amit et al ([2]) first used it to detect spatial relation in the context of character recognition.

3.3.4 Features for the Model Estimation Problem

We now look for functions of the image that are good features for the model estimation problem. Following the discussion in subsection 3.1.1, we look for functions which do not lose any relevant information. There are two ways we can capture this idea:

1. Firstly, recalling the definition in section 2.4, we see that we are seeking a statistic sufficient for estimating the model.

2. Alternately, we could think in terms of distances between different conditional distributions. To formalize this second approach, we define a feature to be any (possibly probabilistic) function of the image, I. We concern ourselves with sets of features, ordered by inclusion. The sets of features, S, that are *useful* are distinguished by the following property :

   "The conditional distribution of the model m given I, $\mathbf{Pr}(m \mid I)$, is close, as a distribution, to the conditional distribution of m given the feature set, S, $\mathbf{Pr}(m \mid S)$".

   In other words, though we lose some data in going from I to S, what we lose does not contribute any useful information. The notion of distance we use in the above criterion is the KL divergence. Thus, let us denote the

conditional probability distribution, $\mathbf{Pr}(m|I)$ by $\mathbf{Pr}_1$; and the conditional probability distribution function $\mathbf{Pr}(m|S)$ by $\mathbf{Pr}_2$. Then we want to minimize the quantity $\mathbf{D}(\mathbf{Pr}_1 \,||\, \mathbf{Pr}_2)$.

It is remarkable that these two notions of features are actually very intimately related. To understand this connection, let us assume, for simplicity, that we are dealing with a single feature, not a set. The consideration that follows generalizes easily to the more general case. Let the feature be a function, $\phi(I)$ of the image. As before, and to keep later notation simple, we assume that the distribution function, $\mathbf{Pr}(m|I)$ is denoted $\mathbf{Pr}_1$; and the distribution, $\mathbf{Pr}(m|\phi(I))$ is denoted $\mathbf{Pr}_2$. Then we notice that the triple $m \;\rightarrow\; I \;\rightarrow\; \phi(I)$ forms a *Markov chain*. Then, we have:

Theorem 3.3.1 (Sufficiency & KL-distance)

$$\mathbf{E}_I[\mathbf{D}(\mathbf{Pr}_1 \,||\, \mathbf{Pr}_2)] \;=\; \mathbf{M}(m;I) - \mathbf{M}(m;\phi(I)) \tag{3.4}$$

Proof

Markov property (definition 2.4.2) states that
$\mathbf{Pr}(m,I,\phi(I)) = \mathbf{Pr}(I,\phi(I))\mathbf{Pr}(m|I,\phi(I)) = \mathbf{Pr}(I,\phi(I))\mathbf{Pr}(m|I)$.
Furthermore, if ϕ is a function, $\mathbf{Pr}(I,\phi(I)) = \mathbf{Pr}(I)$. Therefore,

$$\begin{aligned}\mathbf{M}(m;I) - \mathbf{M}(m;\phi(I)) \;&=\; \sum_{I,m} \mathbf{Pr}(m,I)\log\frac{\mathbf{Pr}(m,I)}{\mathbf{Pr}(m)\mathbf{Pr}(I)} \\ &-\; \sum_{\phi(I),m} \mathbf{Pr}(m,\phi(I))\log\frac{\mathbf{Pr}(m,\phi(I))}{\mathbf{Pr}(m)\mathbf{Pr}(\phi(I))}]\end{aligned}$$

Simplifying,

$$\begin{aligned}
\mathbf{M}(m;I) - \mathbf{M}(m;\phi(I)) &= \sum_{I,m} \mathbf{Pr}(m,I) \log \frac{\mathbf{Pr}(m|I)}{\mathbf{Pr}(m)} \\
&- \sum_{\phi(I),m} \mathbf{Pr}(m,\phi(I)) \log \frac{\mathbf{Pr}(m|\phi(I))}{\mathbf{Pr}(m)}]
\end{aligned}$$

$$\begin{aligned}
&= \sum_{I,\phi(I),f} \mathbf{Pr}(m,I,\phi(I))[\log \frac{\mathbf{Pr}(m|I)}{\mathbf{Pr}(m)} - \log \frac{\mathbf{Pr}(m|\phi(I))}{\mathbf{Pr}(m)}] \\
&= \sum_{I,\phi(I)} \mathbf{Pr}(I,\phi(I)) \sum_{m} \mathbf{Pr}(m|I,\phi(I)) \log[\frac{\mathbf{Pr}(m|I)}{\mathbf{Pr}(m)} \frac{\mathbf{Pr}(m)}{\mathbf{Pr}(m|\phi(I))}] \\
&= \sum_{I} \mathbf{Pr}(I) \sum_{m} \mathbf{Pr}(m|I) \log \frac{\mathbf{Pr}(m|I)}{\mathbf{Pr}(m|\phi(I))} \\
&= E_I[D(\mathbf{Pr}(m|I)||\mathbf{Pr}(m|\phi(I)))] \qquad (3.5)
\end{aligned}$$

■

If we consider I to have a family of distributions depending on the parameter m, then, as we saw earlier, the vanishing of the above quantity is the condition for ϕ to be sufficient for this family of probability distributions on I. Thus the notion of sufficient statistic exactly captures what it means for a function to be a (useful) feature.

The left side of equation 3.4 is an expectation of a positive quantity, while its right hand side is non-negative, by the Data Processing Inequality, Theorem 2.4.2. Thus the vanishing of the right hand side, implies that $\mathbf{D}(\mathbf{Pr}_1 \mid\mid \mathbf{Pr}_2) = 0$ for almost all I. Thus, the different concepts of sufficiency tie together as does the concept of a feature.

3.4 Feature Discovery

We saw in the last section that the set of "nice" features and statistics sufficient for the class of models share some intuitively justifiable properties. The criterion we arrived at was this:

Criterion 3.4.1 (Features and Sufficiency) *A feature ϕ is informative (for the task of estimating the model) if ϕ is a sufficient statistic. Thus ϕ would satisfy all of the following (equivalent) conditions:*

Conditional Density: *The conditional density,* $\mathbf{Pr}(I \mid \phi(I), m)$ *is independent of the model* m.

Mutual Information: $\mathbf{M}(f; I) = \mathbf{M}(f; \phi(I))$

Kullback-Leibler Divergence: $\mathbf{D}(\mathbf{Pr}_1 \mid\mid \mathbf{Pr}_2) = 0$ *for almost all* I, *where* $\mathbf{Pr}_1$ *is the conditional* $\mathbf{Pr}(m \mid I)$ *and* $\mathbf{Pr}_2$ *is the conditional* $\mathbf{Pr}(m \mid \phi(I))$.

As the third condition illustrates clearly, the connection between features and sufficient statistics is one that involves the probability distribution on the set of all possible images. Thus, we seek a function ϕ whose form does not depend on I, but which nevertheless satisfies the condition on the KL divergences for almost all I. This is a very strong condition on ϕ and a distribution does not admit a sufficient statistic unless it is of a very special form. In fact we have the following theorem, which is fairly standard. (See for example Koopman [75], Pitman [95], Wilks [121] page 393, Linnik [78] page 39.)

Theorem 3.4.2 *Let $x_1, x_2, \ldots, x_n$ be n (not necessarily independent) samples from a parametric distribution with parameter τ and let $t(x_1, x_2, \ldots, x_n)$ be a sufficient statistic estimating τ. Let the joint distribution of the x_i-s be denoted as $f_n(x_1, x_2, \ldots, x_n; \tau)$. Then f_n is necessarily of the form:*

$$f_n(x_1, x_2, \ldots, x_n; \tau) \;=\; \exp\left[K_1(\tau)g(t) \;+\; K_2(\tau) \;+\; h_n(x_1, x_2, \ldots, x_n)\right], \qquad (3.6)$$

where $K_1(\tau)$ and $K_2(\tau)$ do not depend on x_i's, h_n does not depend on τ and $g(t)$ depends only on t and not on τ.

Thus a sufficient statistic exists only for an exponential family. In our setting, where features are sufficient statistics, this theorem implies that the conditional probability distribution $\mathbf{Pr}(I \mid m)$ of I given m will be an exponential distribution where the response of the feature detector enters in the exponent.

This is a strong restriction, but in practice this is a minor problem. We actually end up maximizing the mutual information $\mathbf{M}(m; \phi(I))$ between the model and the feature. Thus, if we do not start with a family of distributions which admit a sufficient statistic, we will be led to a feature-set which is as close to a sufficient statistic as possible in an information-theoretic sense.

However, there is a more practical problem with a naive implementation of our approach. As pointed out earlier, sufficiency is a condition involving the probability distribution on the set of all images I. Therefore a naive calculation of the expectation needs to iterate over all possible values of I. This is too large a space to search.

If we were intent on building features for smaller patches of the image, we could sample over these patches and use the sample values to calculate the expectations. This sampling approach would depend on the ergodicity of the image generation process.

In this thesis however, we explore ways of using a single image to approximate the expectation.

3.4.1 Minimizing KL-Divergence to Approximate Sufficiency

Equation 3.4 connects the average value of the KL divergence of the two conditionals $\mathbf{Pr}(m|I)$ and $\mathbf{Pr}(f \mid \phi(I))$ with the difference between the corresponding mutual informations: $\mathbf{M}(f;I) - \mathbf{M}(f;\phi(I))$.

We use this to motivate our approach. Thus, instead of minimizing the difference on the right hand side of that equation, we will (approximately) minimize the quantity

$$\mathbf{D}(\mathbf{Pr}(m|I) \,||\, \mathbf{Pr}(m|\phi(I)))$$

There is an intuitive justification for this approach as well. As we saw earlier, the quantity in question, $\mathbf{D}(\mathbf{Pr}(m|I) \,|\, \mathbf{Pr}(m|\phi(I)))$, is a measure of how much the actual conditional $\mathbf{Pr}(m|I)$ differs from the approximate conditional $\mathbf{Pr}(m|\phi(I))$. More precisely, this is the error made in a maximum-likelihood estimation of the model m if the feature is used instead of the image. Thus, if we do not concern ourselves with the distribution of the image I but concentrate instead on the specific instance of I that we have, minimizing the KL divergence $\mathbf{D}(\mathbf{Pr}(m|I) \,||\, \mathbf{Pr}(m|\phi(I)))$ is the right thing to do.

3.5 Drawbacks of the Approach

The main problem that occurs when we use this approach is that we have too few data points. Optimizing the KL divergence for the given image I might lead to a ϕ which is incorrect for most other realizations of I. This phenomenon is called *over-fitting*. The process that leads to over-fitting is fairly general and is not peculiar to the particular cost function (KL divergence) we minimize. To understand it let us look carefully at the possible sources of error in estimation.

Thus, we have some possible set of models $\mathcal{M}$ and the possible set of features $\mathcal{P}$. Thus, $m \in \mathcal{M}$ and $\phi \in \mathcal{P}$. We have some notion of complexity of the classes $\mathcal{M}$ and $\mathcal{P}$. Thus, for one-dimensional signals, the class of step functions is (conceivably) less complex than the class of functions which look like ramps bounded by horizontal lines.[3] Over-fitting occurs when the complexity of the class $\mathcal{P}$ exceeds that of the class $\mathcal{M}$. Thus, given some unlikely value of I, our algorithm will choose some ϕ which significantly reduces $\mathbf{D}(\mathbf{Pr}(m|I) \,||\, \mathbf{Pr}(m|\phi(I)))$ for that particular I but performs badly on the others. This is possible because the greater complexity of $\mathcal{P}$ means we can choose ϕ to explain variation which actually arose in the "noise" process: $\mathbf{Pr}(I \,|\, m)$.

Thus, over-fitting reveals itself in an estimate of the model m which is bounded away from the true model with high probability. As explained in the "risk minimization" literature (see Vapnik [117] for example) we can modify the basic optimization

[3]Note the "conceivably". The formal definition of complexity involves some semi-norm defined over the corresponding domains. This might not correspond to our intuitive ideas of what is more complex.

framework to guard against over-fitting. This is the regularization approach and we will discuss it a little later.

3.5.1 Guarding Against Over-fitting

There are basically two approaches we can take to guard against over-fitting. These are somewhat related, but there are enough points of difference that we describe them separately.

Choosing a Less Complex Class of Models

In the first approach we artificially reduce the complexity (capacity) of the class of features $\mathcal{P}$. Thus, in the case of estimating the model for one-dimensional signals for example, we can choose to detect only step edges, and ramps. This means for any given image, we allow slightly higher error probability in fitting to the sample, while getting better guarantees on the accuracy of the model estimated.

It is possible to iterate this process. Suppose we can decompose the given class $\mathcal{P}$ in an increasing sequence $\mathcal{P}_1 \subseteq \mathcal{P}_2 \subseteq \cdots \subseteq \mathcal{P}_s$, with $\mathcal{P}_s = \mathcal{P}$ and of increasing complexity. Suppose, also, that the choice of the best feature in the class $\mathcal{P}_i$ somehow affects the estimate in subsequent classes; for example by serving as the starting point in the next iteration scheme. Since we start with a class of lower complexity, our probability of learning a "wrong" model is much reduced. This is the basis of the *Structural Risk Minimization* (SRM) approach.(again, see Vapnik [117]).

The choice of the restricted class of features is somewhat arbitrary. This is

mitigated in the second approach for reducing over-fitting.

Regularization Methods

In this approach we modify the cost function.Instead of minimizing the KL divergence, we minimize a cost function of the form

$$\mathbf{D}(\mathbf{Pr}(m|I)\,||\,\mathbf{Pr}(m|\phi(I)))\ +\ \gamma\Omega(\phi), \tag{3.7}$$

where $\Omega(\phi)$ is a regularizer for ϕ and γ is a positive number which we let approach 0. Under sufficiently general conditions, it is possible to select a sequence of values for γ which approach 0 and such that the regularized solutions approach the correct solution and such that the regularized problem is well-posed (in the sense of Tikhonov, [113]). This kind of cost function is extremely common in many optimization tasks mainly because of its good generalization behavior.

The SRM approach sketched in the previous subsection frequently leads to an optimization problem of this structure. In some sense what is being attempted is an estimation of the quantity

$$\mathbf{E}[\mathbf{D}(\mathbf{Pr}(m|I)\,||\,\mathbf{Pr}(m|\phi(I)))]$$

in terms of a single sample value $\mathbf{D}(\mathbf{Pr}(m|I)\,||\,\mathbf{Pr}(m|\phi(I)))$. Thus for each restricted complexity class $\mathcal{P}_i$ we find an upper bound for the expectation. This upper bound is a sum of the sampled cost and a term representing the cost penalty incurred for a inferring an incorrect model. The SRM approach proceeds by minimizing this upper bound on "risk" for each class $\mathcal{P}_i$. Each of these modified cost functions is thus of the form 3.7.

In our particular case the regularized cost function mitigates two problems. Firstly, it penalizes the divergence as well as the complexity of the resulting feature. Thus, it leads to simpler features and guards against over-fitting. For our particular problem, there is another problem. We are looking for a function ϕ which will serve as the feature and for which the conditional distribution $\mathbf{Pr}(m \mid \phi(I))$ is as close to the "real" conditional distribution $\mathbf{Pr}(m \mid I)$ as possible. However, there is no guarantee that we have a unique solution. Indeed, the identity function which does not change the image at all (or any invertible transformation) could serve as well. Using a regularization term lets us put a penalty on these functions as well.

Thus, we are led to the following characterization for features:

A feature is obtained by minimizing a regularized KL-divergence cost function, the expectation of the unregularized version of which gives a measure of the sufficiency of the statistic.

3.6 Summary

In this chapter we covered the meat of the first part of this thesis. We characterized features as regularized versions of an approximation to sufficient statistics. This is a novel way of looking at features which incorporate within it many ad hoc methods of feature discovery.

If we had computational power, or if the space of images was small enough, we could calculate the sufficient statistics and that would give us features as well. In the next chapter we investigate how this theory can be applied to discover features for one-dimensional signals and for stereo.

Chapter 4

Examples of Feature Discovery

4.1 One Dimensional Signals

We now try to apply the preceding, rather general, considerations to more concrete examples. The first example we treat is in the domain of one dimensional signals. Later on in this chapter, we will consider the problem of discovering features for stereo.

Thus, consider a one dimensional signal, I, that is the result of a stochastic image-formation process. The value I assumes at a particular location can be either discrete or continuous. We assume that we have a setup similar to the one described in chapter 3. Thus, we assume that I depends on the precise state, f, of an environment (or the model).[1] The signal (image), accordingly, contains information about the environmental state f, possibly corrupted by noise. We

[1] In chapter 3 we denoted the model by m. In this chapter, we use f to denote the model to remind us that we are really estimating functions corrupted by noise.

wish to choose feature vectors $\phi(I)$ derived from the image that summarize this information concerning the environment. We are not otherwise interested in the contents of I and wish to discard any information concerning the image that does not depend on the environmental state f. Thus, the generative process has exactly the same structure as in chapter 3 and all we need to specify are the actual forms of the prior probability distribution and the generative model.

4.2 Prior Probability

The first probability distribution we need is the prior probability. We choose this prior so as to make continuous functions more likely. Thus we suppose that we are given a cost function which penalizes against discontinuities. In keeping with the notation from last chapter, we choose this of the form:

$$\mathcal{P}(f) = \sum_i F_i(f_{i-1}, f_i) \tag{4.1}$$

The functions F_i are functions which penalize for local variation in f at pixel i. This was denoted $\mathcal{T}$ in the last chapter.

From this cost function, and following Blake and Zisserman [24], Geman and Geman [48] Mumford and Shah [85], we construct the prior probability as a Gibb's distribution using the cost function as energy:

$$\mathbf{Pr}(f) \ \propto \ \exp(-\sum_i F_i(f_{i-1}, f_i))$$

or normalizing, we get for the prior probability of f

$$\mathbf{Pr}(f) \ = \ (1/Z_p)\exp(-\sum_i F_i(f_{i-1}, f_i)) \tag{4.2}$$

where Z is the partition function defined by

$$Z_p = \sum_{\{f'\}} \exp(-\sum_i F_i(f'_{i-1}, f'_i)).$$

Depending on the situation, this summation could be replaced by a repeated integration.

To complete the description of the prior probability, we need to specify the local discontinuity costs F_i. These we choose to be thresholded quadratic functions:

$$F_i(f_i, f_{i-1}) = \begin{cases} (f_i - f_{i-1})^2 & \text{if } |f_i - f_{i-1}| \leq \theta \\ \theta^2 & \text{otherwise} \end{cases} \tag{4.3}$$

where θ is some positive parameter denoting the threshold.

Using a thresholded discontinuity cost function lets us penalize small discontinuities, while ensuring that we do not make step edges etc. completely impossible.

4.3 The Generative Process

We now need to choose the generative process. This will be formulated by specifying the conditional probability distribution $\mathbf{Pr}(I \mid f)$. In the following discussion, we will consider a single image, I, which is defined on points $i = 0 \ldots N-1$. As explained in chapter 3 we start by defining the form of a data cost function, and use the corresponding Gibb's distribution as the required conditional distribution. Thus to start the ball rolling we define the data cost function between the model f and the image I as:

$$\mathcal{D}(f, I) \equiv \sum_i C(f_i, I_i)$$

where C is some cost (loss) function. We choose the quadratic loss function:

$$C(f_i, I_i) \equiv (f_i - I_i)^2$$

Thus, we get for the data cost, the following expression:

$$\mathcal{D}(f, I) \equiv \sum_i (f_i - I_i)^2 \tag{4.4}$$

Following the recipe given above, we form the conditional probability as the Gibb's distribution:

$$\begin{aligned} \mathbf{Pr}(I \mid f) &= (1/Z_D(f)) \exp(\mathcal{D}(f, I)) \\ &= (1/Z_D(f)) \exp(-\sum_i C(f_i, I_i)) \end{aligned} \tag{4.5}$$

where Z_D is the "data" partition function:

$$\begin{aligned} Z_D(f) &= \sum_{\{I\}} \exp(-\sum_i C(f_i, I_i)) \\ &= \prod_i \sum_{\{I_i\}} \exp(-C(f_i, I_i)) \end{aligned} \tag{4.6}$$

Again, depending on the situation, we might have to replace this summation by a repeated integration. The expression for the partition function indicates that there is a possible dependence on f. We have indicated this in the formula above.

Now we note that the various I_i-s are conditionally independent given f_i. Thus, we can express the distribution of I given f as a product of the distributions for I_i given f_i. Tolerating a slight abuse of notation, this enables us to factorize the

above conditional as

$$\begin{aligned}\mathbf{Pr}(I|f) &= \prod_i \frac{\exp(-C(f_i, I_i))}{\sum_{\{I_i\}} \exp(-C(f_i, I_i))} \\ &= \prod_i \frac{\exp(-C(f_i, I_i))}{Z_D(f_i)} \end{aligned} \tag{4.7}$$

where the partition function is dependent on f_i.

Now we make an assumption which is true for most reasonable situations. We assume that the I_i-s have infinite range and the cost function C is symmetric in that it satisfies

$$\sum_{I_i} \exp(-C(f, I_i)) \quad \text{is independent of } f.$$

Here the summation sign is to be replaced by an integral if I_i is a continuous variable. This is certainly true of the quadratic loss function we have chosen, or indeed any loss function of the form $C(f, I_i) = g(f - I_i)$ where g is an even function.

Under this condition, the normalization constant (partition function) $Z(f_i)$ for each i; which is nominally a function of f_i, is actually independent of it and is a constant. We also notice that the denominator of the expression for $\mathbf{Pr}(f)$ is a constant as well.

Then, using Bayes' theorem,

$$\begin{aligned}\mathbf{Pr}(f|I) &= \frac{\mathbf{Pr}(f)\mathbf{Pr}(I|f)}{\mathbf{Pr}(I)} \\ &= \frac{\mathbf{Pr}(f)\mathbf{Pr}(I|f)}{\sum_f (\mathbf{Pr}(f)\mathbf{Pr}(I|f))} \\ &= \frac{\exp(-\sum_i (C(f_i, I_i) + F_i(f_{i-1}, f_i)))}{\sum_{\{f\}} \exp(-\sum_i (C(f_i, I_i) + F_i(f_{i-1}, f_i)))} \end{aligned}$$

For reasons that will become clear later, we denote this a posteriori probability as P_0.

4.4 Possible Features

Now we discuss the choice of features. To motivate this discussion, let us first consider the process of reconstruction of the image given a few isolated points. The best approach that works, and is consistent with our prior and generative models, is a dynamic programming algorithm that minimizes a cost function comprising of two terms:

1. the prior cost
2. the cost C at the points where the value of I_i is known.

Indeed the same algorithm would be used to calculate the MAP estimate of the model f given the image I. This can be seen immediately from the form of the posterior distribution calculated above:

$$\mathbf{Pr}(f \mid I) \;=\; \frac{\exp(-\sum_i (C(f_i, I_i) + F_i(f_{i-1}, f_i)))}{\sum_{\{f\}} \exp(-\sum_i (C(f_i, I_i) + F_i(f_{i-1}, f_i)))}$$

and so calculating the MAP estimate for f is equivalent to the minimization of the cost function:

$$\sum_i (C(f_i, I_i) + F_i(f_{i-1}, f_i))$$

In keeping with this idea, we view features as specific pixels so that completion using a dynamic programming algorithm to obtain the underlying model f yields a solution close to that obtained using the original image I. One might argue that picking single pixels as features makes one ignore a lot of structure. However, using sufficiency as a criterion forces the feature selection algorithm to look beyond the immediate pixel value and consider the values of the image in the neighborhood as well. Furthermore, this approach can easily be modified to operate on the coefficients of some expansion of the image in some over-complete basis. Such an approach would be very close in spirit to algorithms which use basis pursuit to pick a sparse representation of vectors.

We view the model f as coming from a class $\mathcal{F}$ such that the particular element $f \in \mathcal{F}$ is determined by some (small) set of location parameters. This is not an absolute guarantee, but rather is implied by the structure of the prior distribution of the model. Models with many discontinuities incur higher costs, and once the position of the discontinuities are located, the prior cost implies we can approximately reconstruct the model f using the dynamic programming approach.

Given the image I our approach will be to keep a small set of locations whose pixel values serve to specify which element $f \in \mathcal{F}$ generated I. We reject the other pixels in I. Thus, we want to find out the "minimum" set of pixels, such that if we keep the values of I on these pixels, the prior cost function ensures our ability to recreate the model f.

4.4.1 s vector

Analytically, the best way to achieve our stated goal (of finding the best subset of pixels) is to define a 0-1 vector, $\{s_i\}$, where the value s_i corresponds to the i'th pixel and indicates whether this pixel is among the feature-set:

$$s_i = \begin{cases} 1 & \text{if } i\text{-th pixel is omitted from the feature} \\ 0 & \text{if } i\text{-th pixel is included among the feature} \end{cases} \tag{4.8}$$

After the definition of this vector, our model estimation boils down to the solution of the following minimization problem using dynamic programming:

$$\begin{aligned} f^* &= \underset{f}{\text{argmin}} \left[\sum_i \left(C(f_i, I_i)(1 - s_i) + F_i(f_{i-1}, f_i) \right) \right] \\ &= \underset{f}{\text{argmin}} \left[\sum_i \left((f_i - I_i)^2 (1 - s_i) + F_i(f_{i-1}, f_i) \right) \right] \end{aligned} \tag{4.9}$$

The problem of feature selection for the model estimation problem would be solved when we can estimate the vector s.

To put this within our probabilistic framework, we construct a family of (probabilistic) functions $\{\phi_s\}$ which depends on the vector parameter s. We will consider this the set of possible features and the feature selection problem consists in picking a member from this family, or, equivalently, in picking a s.

$\phi_s(I)$ simply adds independent Gaussian noises at each pixel location. The variance of the noise added at pixel location i depends on the value s_i and is given by:

$$\sigma_i^2 = \frac{s_i}{2(1 - s_i)}$$

Thus, integrating out I, we get for the conditional distribution $\mathbf{Pr}(\phi_s \mid f)$,

$$\mathbf{Pr}(\phi_s \mid f) = (1/Z) \exp[-(f_i - \phi_i)^2(1 - s_i)]$$

The proof of this simply notes that

1. variance of the sum of two independent Gaussian distributions is the sum of the variances; and

2. $$\frac{1}{2} + \frac{s_i}{2(1 - s_i)} = \frac{1}{2(1 - s_i)}$$

Following the same calculation as in the previous section, we calculate the conditional distribution $\mathbf{Pr}(f \mid \phi_s)$ as

$$P_s(f) \equiv \mathbf{Pr}(f \mid \phi_s) = (1/Z_s) \exp(-\mathcal{E}(f)) \tag{4.10}$$

where $\mathcal{E}(f)$ is the cost (energy) function given by

$$\mathcal{E}(f) = \sum_i (f_i - \phi_i)^2(1 - s_i) + F_i(f_i, f_{i-1}) \tag{4.11}$$

For any s, $P_s(f)$ is a probability distribution on functions f. The proportionality constant Z_s in the definition of P_s is called the *partition function* in statistical physics. We also note that for $s = 0$ (equality of vectors), we get back the conditional $\mathbf{Pr}(f|I)$. This is the case where we keep all the data.

4.5 Feature Discovery Using Sufficiency

At this point, discovering features is simply a matter of choosing the right s. The idea is to get an s with as many entries 1 as possible, so as to get as few features as possible, while still being close to the probability distribution. Therefore, in accordance with our discussion in the general case, we seek to minimize the following *regularized cost function* for s :

$$\mathcal{C}(s) = \mathbf{D}(P_0||P_s) - \lambda \sum_{i=1}^{N} s_i \tag{4.12}$$

where, $\mathbf{D}$, as before, denotes the *Kullback Leibler distance* between the two distributions P_0 and P_s and λ is a non-negative number.

The vector s is a 0-1 vector. But we simplify the problem for ease of solution. Thus we assume that $0 \leq s_i \leq 1$. We will compute the optimal s under this assumption. Then we will threshold the resulting s to get a 0-1 vector.

We next show that the final cost function (on s) $\mathcal{C}$ is a concave function of its argument s. To see this, we calculate the second derivative $\frac{\partial^2 \mathcal{C}}{\partial s_i \partial s_j}$

$$\frac{\partial \mathcal{C}}{\partial s_i} = (\mathbf{E}_{P_s}[(f_i - I_i)^2] - \mathbf{E}_{P_0}[(f_i - I_i)^2]) - \lambda \tag{4.13}$$

$$\frac{\partial^2 \mathcal{C}}{\partial s_i \partial s_j} = (\mathbf{E}_{P_s}[(f_i - I_i)^2 (f_j - I_j)^2] - \mathbf{E}_{P_s}[(f_i - I_i)^2]\mathbf{E}_{P_s}[(f_j - I_j)^2]) \tag{4.14}$$

where $\mathbf{E}_{P_s}$ denotes expectation taken with respect to the distribution P_s. Thus, because the Hessian matrix is a correlation matrix, its eigenvalues are non-negative and it is positive semi-definite. Consequently, $\mathcal{C}$ is concave in s.

We use the path following method (also known as the *homotopy* method) of calculating the optimal s vector. We know that $s = 0$, for $\lambda = 0$. Then, taking

derivatives with respect to λ of the relations for local optimum, we get the following equation:

$$\sum_j \frac{\partial^2 \mathcal{C}}{\partial s_i \partial s_j} \frac{\partial s_j}{\partial \lambda} - 1 = 0 \tag{4.15}$$

We use this scheme to follow the change of the optimal s with λ.

4.5.1 Reconstructing the Original Image

Now that we know how to calculate the probabilities for f_i to take a specific value, we can estimate the actual function (model) from the feature space representation. This estimation is essentially a dynamic programming problem to calculate the MAP estimate of the model f.

We can also use the mean value of f as our estimate. First we calculate the probability distribution of f_i, for every i using the algorithm described in the next section. This lets us calculate $\mathbf{E}[f_i]$ by a direct summation.

4.6 Calculating Expectations of Quantities Depending on Paths

To use equation 4.15 to solve for s, we need to calculate the quantities $(\mathbf{E}_{P_s}[(f_i - I_i)^2(f_j - I_j)^2] - \mathbf{E}_{P_s}[(f_i - I_i)^2]\mathbf{E}_{P_s}[(f_j - I_j)^2])$. Alternatively, we need to be able to calculate the quantity $(\mathbf{E}_{P_s}[(f_i - I_i)^2] - \mathbf{E}_{P_0}[(f_i - I_i)^2])$ and approximate the second derivative in equation 4.15 by finite differencing. Let us see how these can be computed for the case of one-dimensional signals. This computation is standard

and occurs as a sub-computation in Baum-Welch algorithm. However, we present it here to contrast with the complexity of the algorithm in the case of the stereo problem, presented later in the chapter.

We first note that we can calculate the probability distribution for f_i, which is the i-th component of the approximating vector. The calculation entails computing two quantities, α and β, which are $N \times 256$ arrays.[2]. In what follows, it is best to think of the approximating functions as paths in a $N \times 256$ array. Thus, the function f corresponds to the path $\{(i, f(i)) | 1 \leq i \leq N\}$. We also call the quantity $\exp(-(f_i - I_i)^2 (i - s_i))$ the exponentiated local cost at point (i, f_i) of the array; and the quantity $\exp(-F_i(f_i - f_{i-1}))$ the exponentiated transition cost from the point $(i-1, f_{i-1})$ to the point (i, f_i). We denote (exponentiated) local cost and transition cost by $\mathcal{L}(p)$ and $\mathcal{T}(p_1 \longrightarrow p_2)$ respectively, where p is an arbitrary point and p_1 and p_2 are an arbitrary pair of adjacent points. For convenience, we also define *path-products (PP)*. By a path in this context we include partial paths as well. Thus a path is a set of the form $\{(i, f(i)) | j \leq i \leq k\}$, where j and k are arbitrary integers between 1 and N; and f is an arbitrary (approximating) function. $\Pi(p)$ is defined to be the product of the (exponentiated) local costs and transition costs that lie on the path. Thus, if the path p consists of the sequence $p_1 p_2 p_3 \ldots p_k$ then

$$\Pi(p) = \mathcal{L}(p_1)\mathcal{T}(p_1 \longrightarrow p_2)\mathcal{L}(p_2) \ldots \mathcal{T}(p_{k-1} \longrightarrow p_k)\mathcal{L}(p_k) \tag{4.16}$$

We also extend the definition of $\mathcal{T}$ between arbitrary points p_1 and p_2, with

[2]The names of these variables are pretty standard in *Hidden Markov Model* literature

p_1 occurring to the left of p_2. It is just the sum of all the path-products of the (sub)-paths that start at p_1 and end at p_2. We use the same notation to denote either.

With this notation, we define α and β as follows:

$$\alpha(i,j) = \frac{\sum_k \mathcal{T}((1,k) \longrightarrow (i,j))}{\sum_j \sum_k \mathcal{T}((1,k) \longrightarrow (i,j))} \tag{4.17}$$

$$\beta(i,j) = \frac{\sum_k \mathcal{T}((i,j) \longrightarrow (N,k))}{\sum_j \sum_k \mathcal{T}((i,j) \longrightarrow (N,k))} \tag{4.18}$$

Thus, $\alpha(i,j)$ is the probability of $f(i) = j$, in the restricted probability space of functions over the (smaller) interval $[1 \ldots i]$. Similarly, $\beta(i,j)$ is the same probability but in the space of functions over $[i \ldots N]$.

Now, the calculation of the probability of a particular point being on the approximating function is given by

$$\begin{aligned} \mathbf{Pr}(f_i = j) &= \frac{\sum_{(i,j) \in p} \Pi(p)}{\sum_p \Pi(p)} && (4.19) \\ &= \frac{\sum_{(i,j) \in p} \Pi(p)}{\sum_j \sum_{(i,j) \in p} \Pi(p)} && (4.20) \\ &= \frac{\alpha(i,j)\beta(i,j)/\mathcal{L}(i,j)}{\sum_j \alpha(i,j)\beta(i,j)/\mathcal{L}(i,j)} && (4.21) \end{aligned}$$

The variable p in equations 4.19 and 4.20 iterate over complete paths which pass through the point (i,j) (i.e. functions f' such that $f'(i) = j$)

Thus, we can compute the one point marginals with a straightforward loop once we know the values of α and β. A more complicated equation gives us the 2-point probability distributions, but for their computation we need to compute the path-product (Π) values between every two pixels, not just from the two ends.

Calculation of α and β is iterative as well:

$$\alpha(i,j) = \frac{\mathcal{L}(i,j)\sum_k \alpha(i-1,k)\mathcal{T}(i-1,k \longrightarrow i,j)}{\sum_j \mathcal{L}(i,j)\sum_k \alpha(i-1,k)\mathcal{T}(i-1,k \longrightarrow i,j)} \tag{4.22}$$

$$\beta(i,j) = \frac{\mathcal{L}(i,j)\sum_k \beta(i+1,k)\mathcal{T}(i,j \longrightarrow i+1,k)}{\sum_j \mathcal{L}(i,j)\sum_k \beta(i-1,k)\mathcal{T}(i,j \longrightarrow i+1,k)} \tag{4.23}$$

4.7 A Summary of Feature Discovery

Let us briefly summarize the various steps of feature discovery.

1. Set up the path following equation 4.15
2. Calculate the *Hessian matrix* in equation 4.15 as follows. Calculate the first derivatives of the regularized cost using the one-point marginal distributions using the current s value. Finite difference with the previous vector of first derivatives.
3. Calculate the new s values by solving for $\frac{\partial s}{\partial \lambda}$
4. Find the ultimate s vector. The pixels i where s_i is small are the feature locations.

The iteration converges quite rapidly. However, to get an intuitive understanding of the solution it generates, we will give an alternative approach.

4.8 A Condition Depending on Local Variances

We have already seen that we can consider the features to be the pixels i where the value of the s vector are low (say, less than a certain threshold). However, the computation of s is an iterative process. If D is the image depth (number of gray values) and N is the total number of pixels, a naive calculation of α and β and the one point probability distribution takes $\Theta(D^2N)$ steps. Thus, at each step of the iteration, we compute a $D \times D$ matrix in $\Theta(D^2N)$ time and invert it . While we can do better than this, e.g. by stopping computations of α and β too far away from the I_i, the process is still expensive.

Furthermore, the above description does not give us a good idea of what the discovered features are.

To find an alternate method of feature discovery, we consider the variances of the one point distributions f_i of f.

Let us first examine the case of the step edge to understand what is going on. From the Figures 4.1 and 4.3 we see that that the image has the step between pixels 32 and 33. Qualitatively, it is easiest to see why the variance is higher at these points. At pixel location 32 there are two competing pulls. The local cost of any particular f favors its making the jump and following the image exactly, whereas the transition cost tells it to continue at the same value it is at. Thus the probability distribution of f_{33} is more spread out. The same thing happens at pixel location 33.

Thus the variance is higher at locations where there are jumps in the image and lower at smoother locations. Because of discontinuities introduced by the noisy

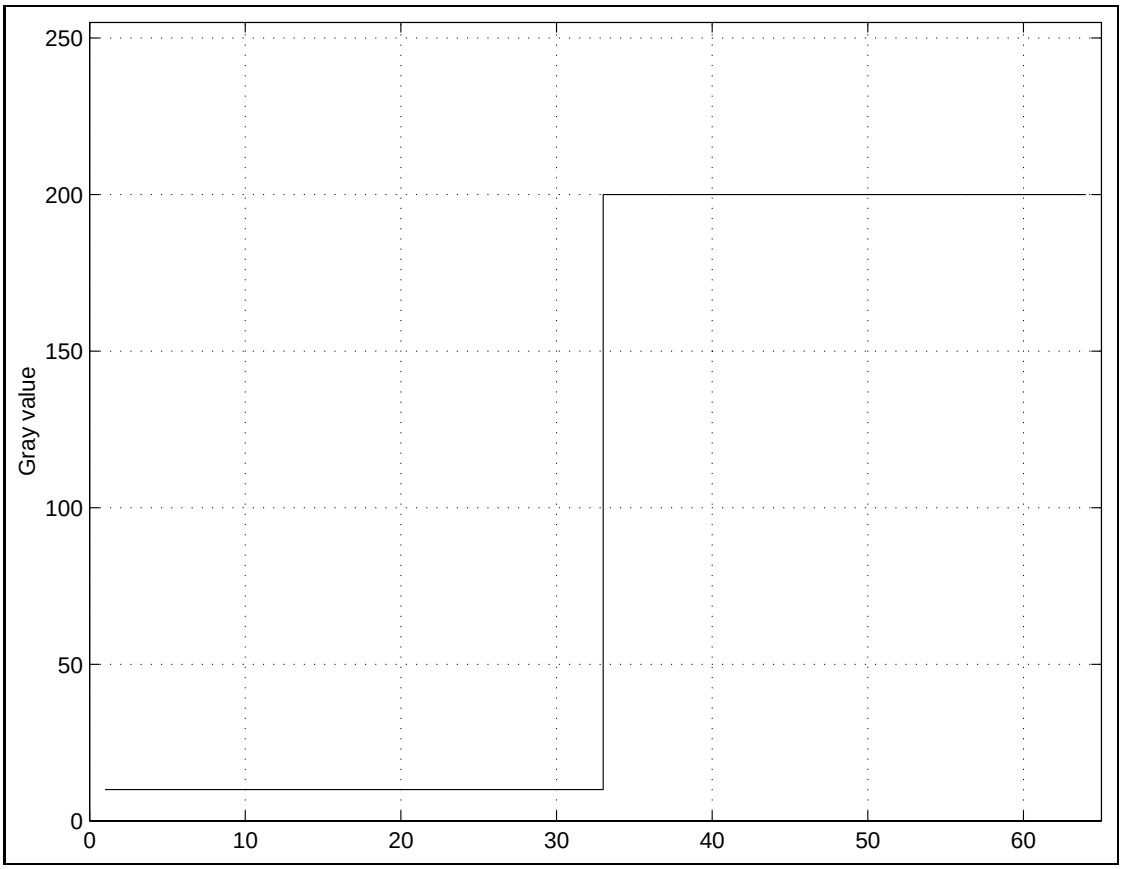

Figure 4.1: A Step Edge

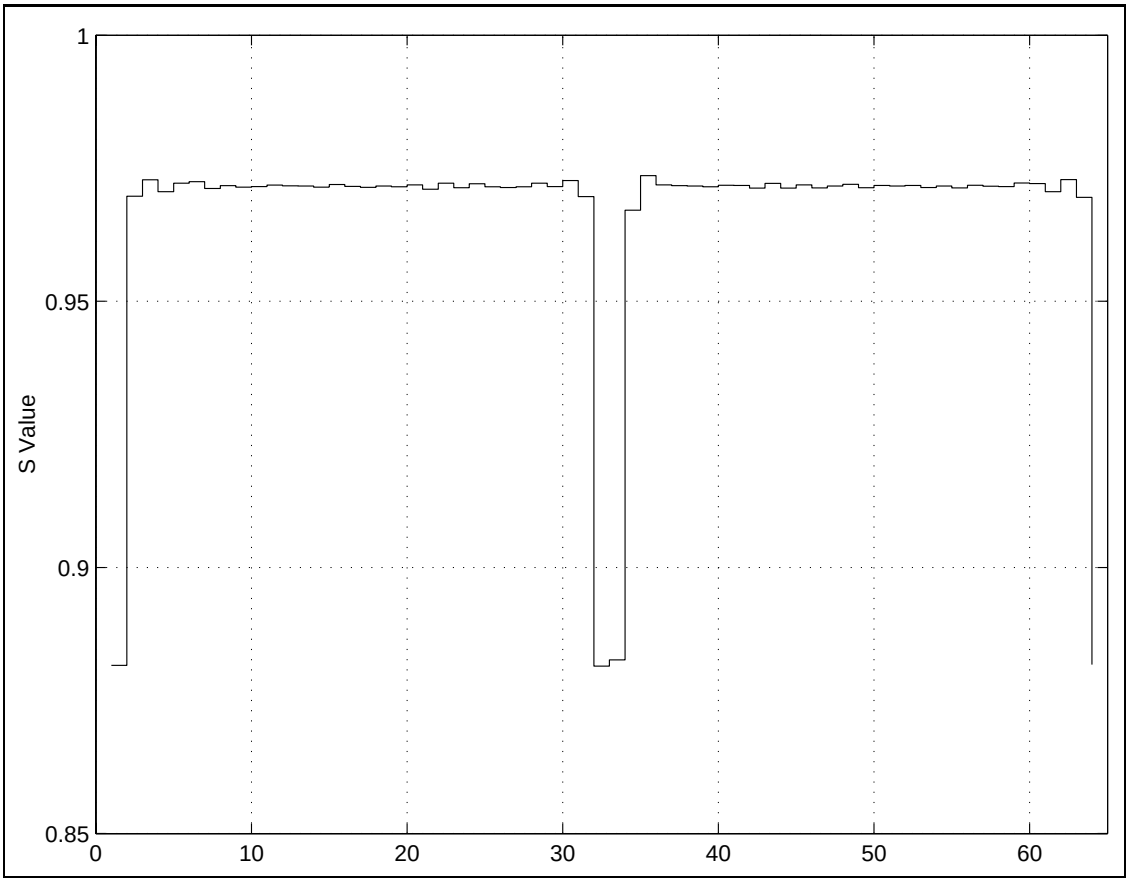

Figure 4.2: S Vector for the Step Edge

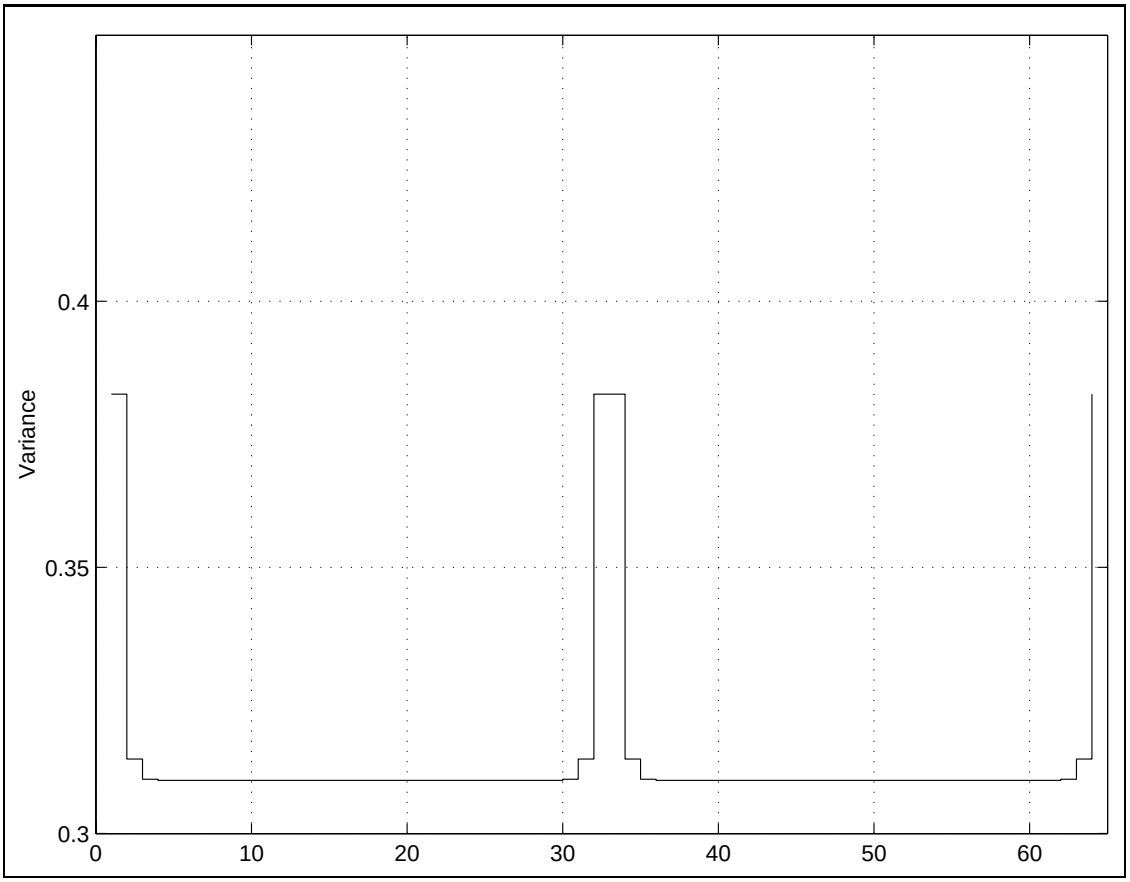

Figure 4.3: Variance for the Step Edge

nature of the generative process, there are other places where this method detects a transition between two levels. However, these artifacts of noise are easy to detect as these are isolated locations where the variance is high and we can use some simple smoothing technique to eliminate this noise.

Explanation in terms of Channel Capacity

There is another explanation of the effectiveness of the variance as a quick estimator for the feature in terms of Shannon's theory of communication channels.

Consider the Markov chain $f \rightarrow I \rightarrow I_i$. We have made the case above that for I_i to be a feature, we want it to be a sufficient statistic. The condition for that is that $I(f;I) = I(f;I_i)$. Under normal circumstances, we will not be able to achieve this equality. So, we do the next best possible thing. We try to make the right hand side as large as possible, keeping in mind, of course, that it cannot exceed the left hand side because of Markovian property of the chain. Thus, we ask under which circumstances can we make the mutual information, $I(f;I_i) = I(f_i;I_i)$ large. But of course this second quantity cannot be larger than its largest possible value over all probability distributions. This supremum is known as the channel capacity; and by a standard result it is bounded above by the quantity

$$\frac{1}{2}\log(1 + \frac{Var(f_i)}{Var(N_i)})$$

where N_i is the local noise process at pixel i.

$Var(N_i)$ is a constant for the particular model chosen. Thus we seek to keep points with high variance of f_i. Note that this criterion makes no mention of the parameters s_i.

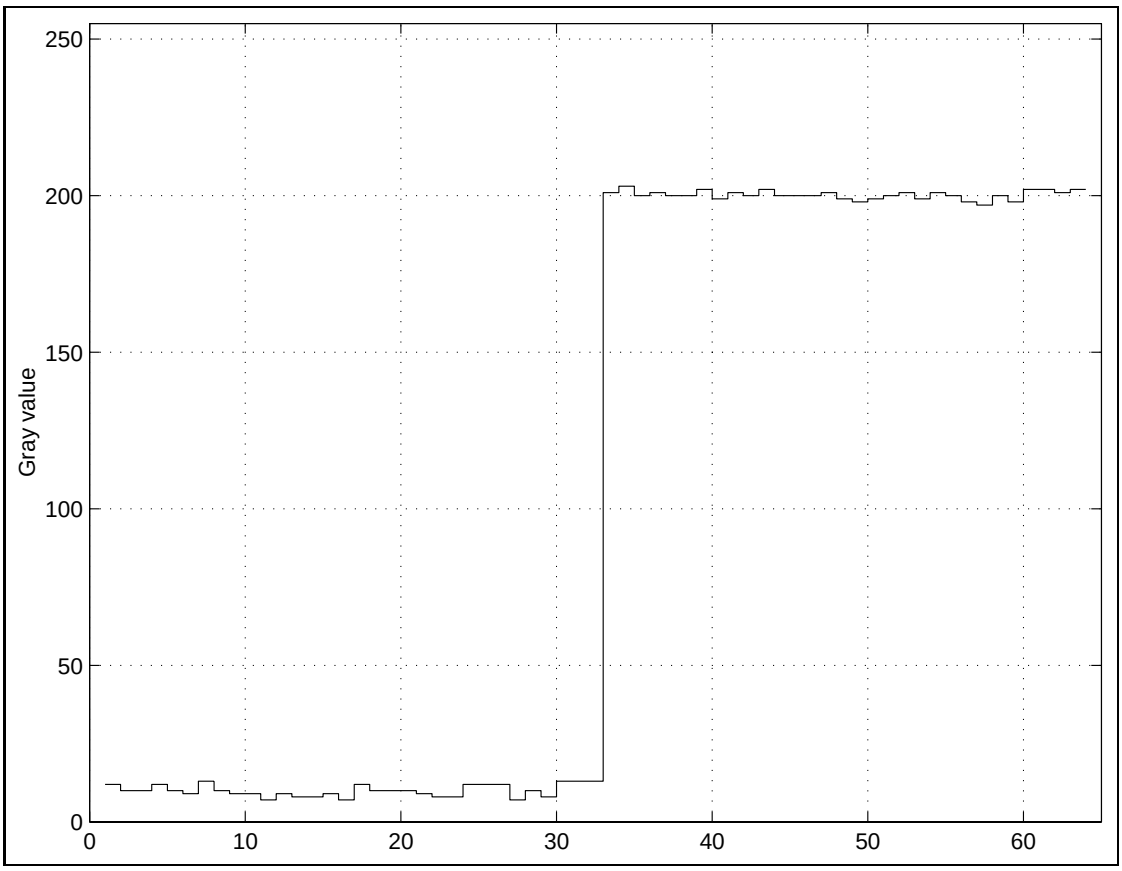

Figure 4.4: Step Edge with Noise

Thus we see that the *variance* provides a good computational tool to determine the sufficiency of a pixel (feature). We will see that the same theme recurs in the case of two dimensional functions.

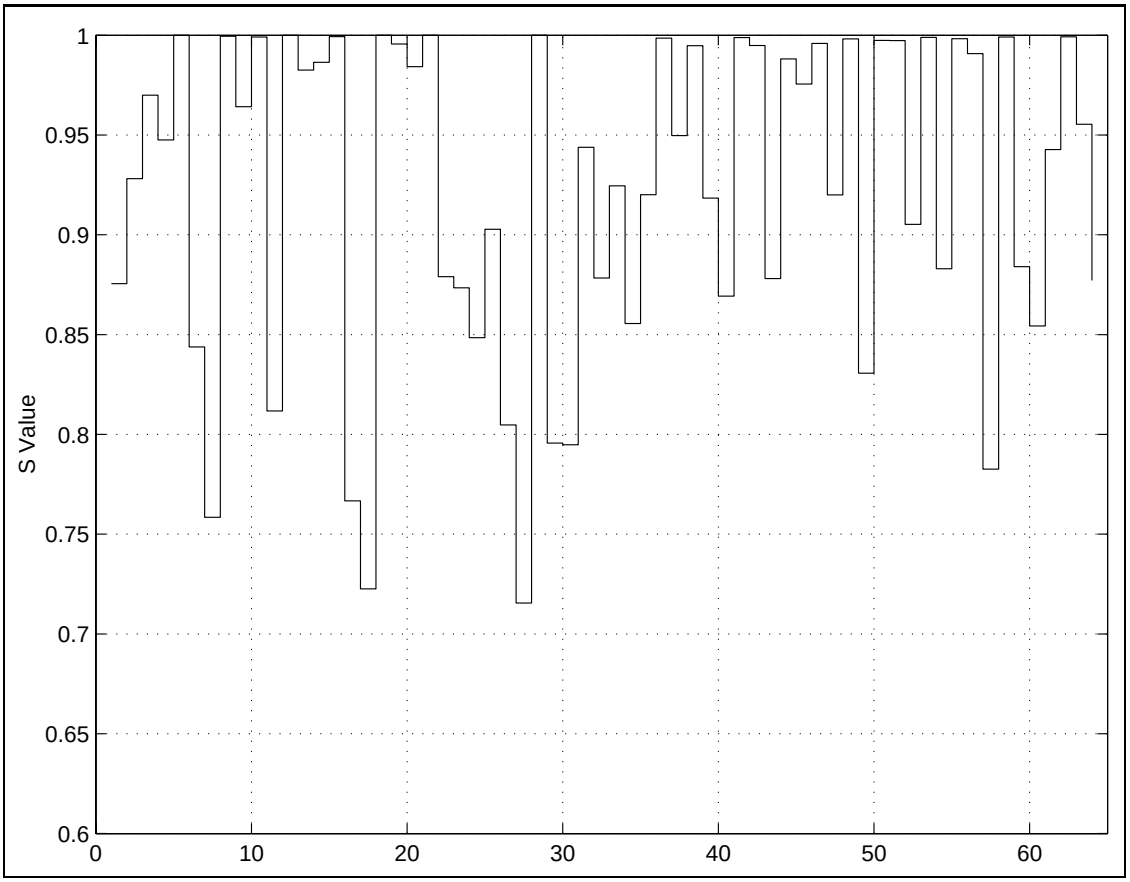

Figure 4.5: S Vector for the Noisy Step Edge

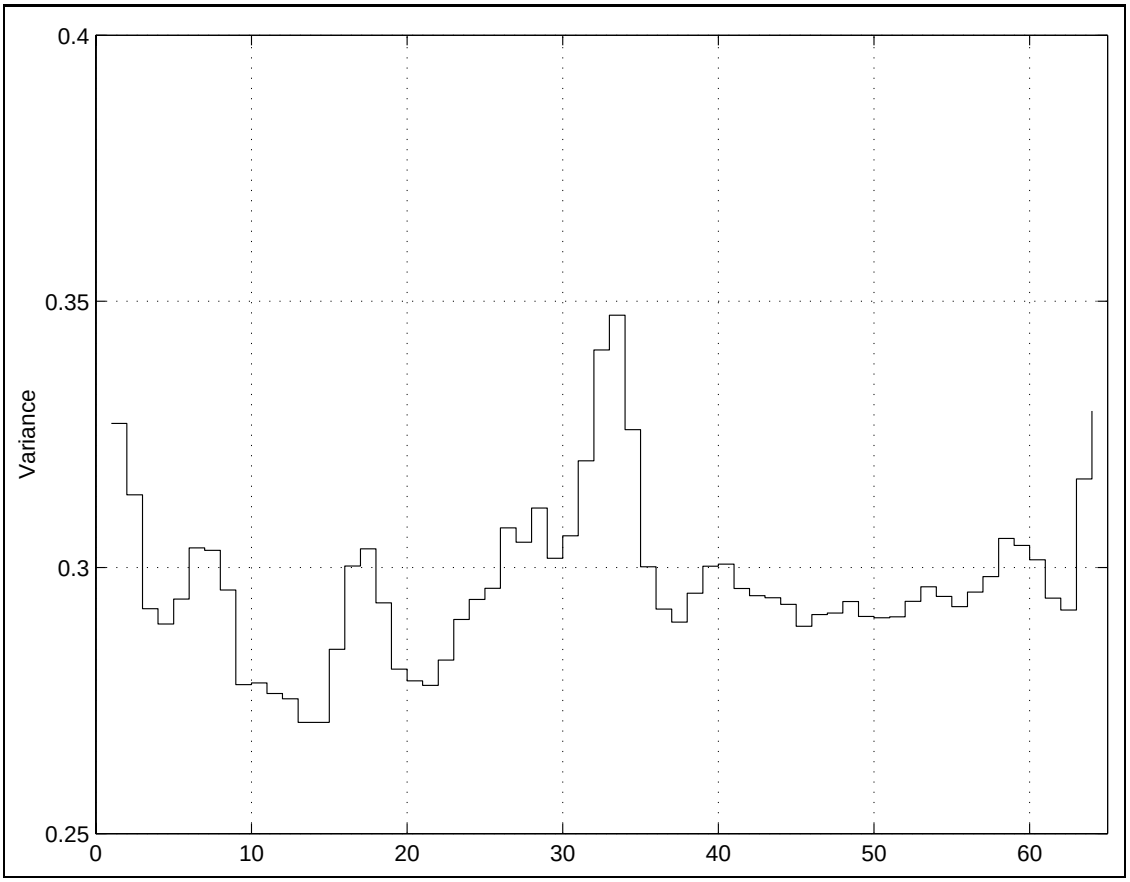

Figure 4.6: Filtered Variance for the Noisy Step Edge

4.9 Results for one dimensional signals (s-values and variances for special kinds of images)

We now look at how our procedure applies to some concrete examples. We first consider the case of special one-dimensional signals. For each of these classes we will calculate the features obtained by solving the path-following equation 4.15 where the Hessian matrix is calculated by calculating the first derivative $\frac{\partial \mathcal{C}}{\partial s_i}$ using the equation 4.13, and using a finite difference method to calculate the second derivative.

It is also possible to apply equation 4.14 to directly calculate the second derivative, but calculating the correlation on the right hand side of that equation is computationally expensive.

We look at successively more complex kinds of signals to see which points are considered important by our algorithm. First we solve the path-following equation, equation 4.15 to solve for the vector parameters s. The pixels where these values are small are the locations that are considered important.

We show the calculated s values for the step edge of figure 4.1 in figure 4.2. As expected, we note that the values of s are lower at the location where the step is. This indicates that if we start with the pixel values at these locations, and fill in the other locations using dynamic programming, we will get a graph very close to the one we started from. For this input, we also calculate the pixel-wise variances, as explained in section 4.8. These values are plotted in figure 4.3. As expected, the variances are higher in the region around the step.

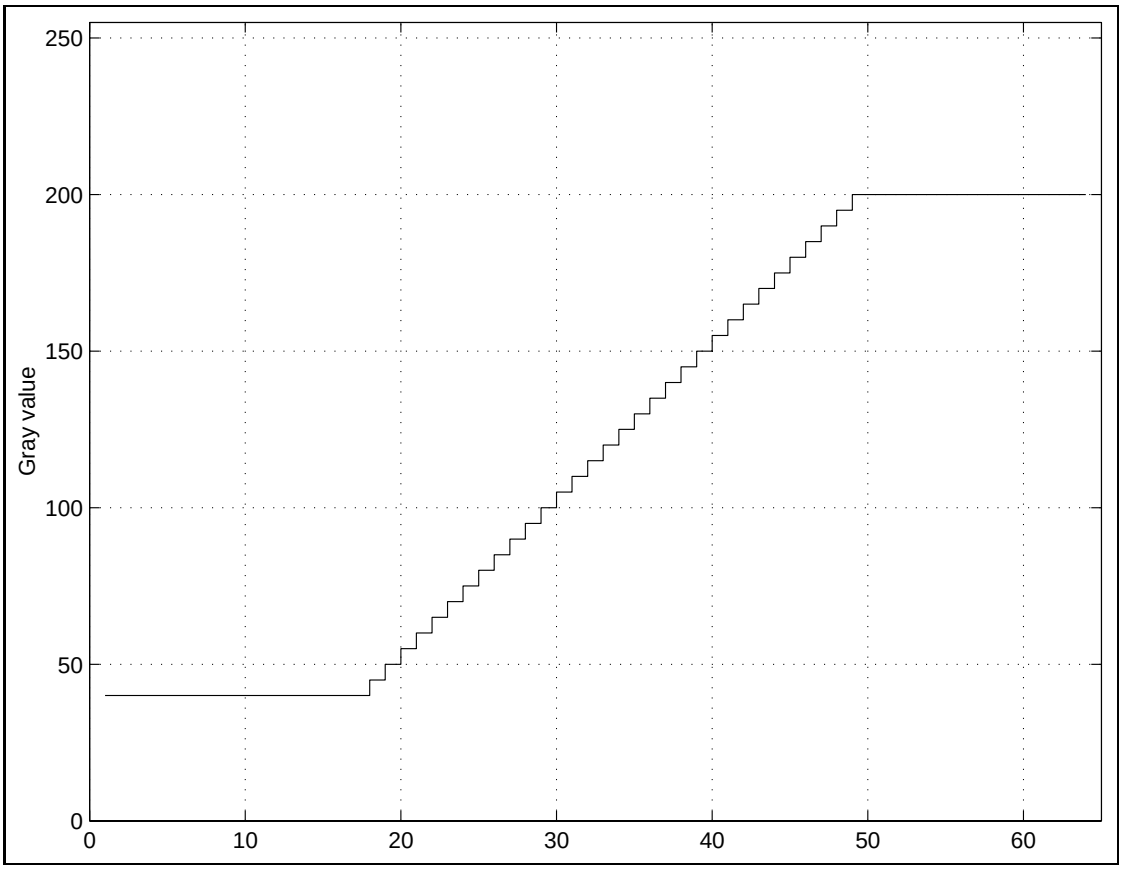

Figure 4.7: A Ramp Signal

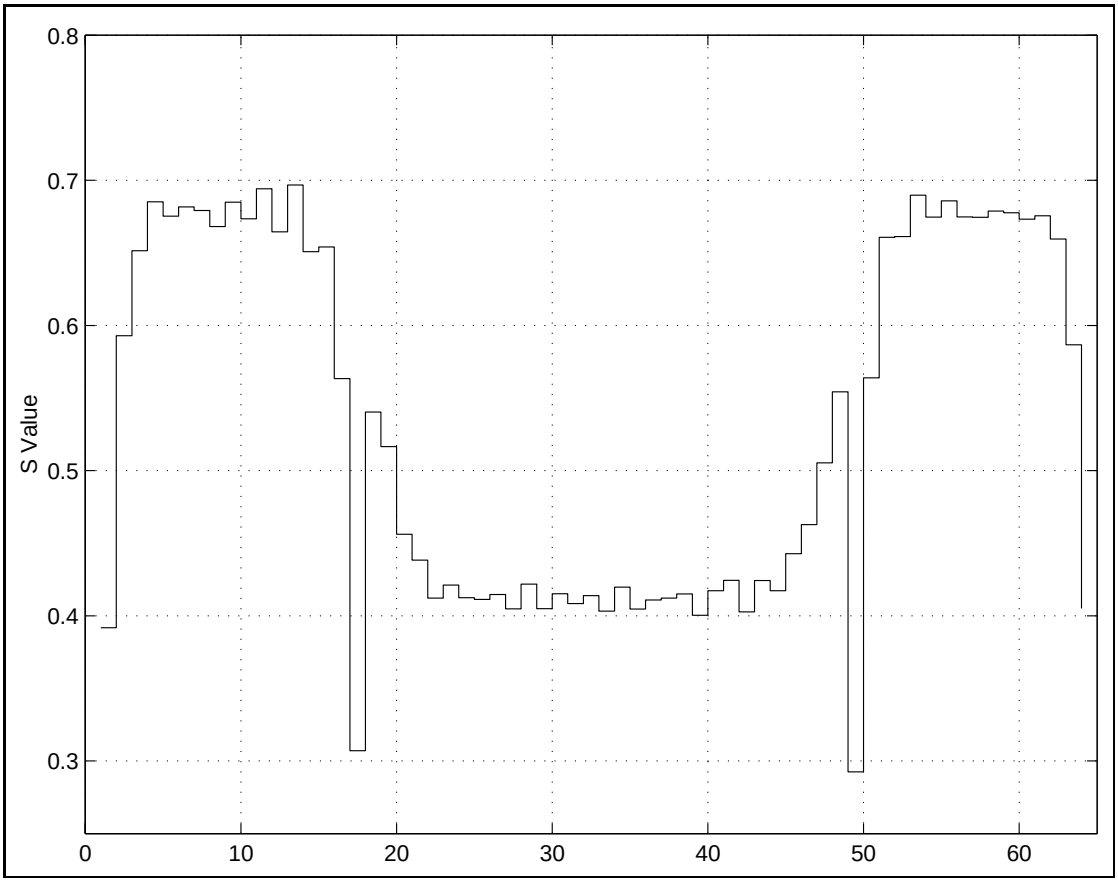

Figure 4.8: S Vector for the Ramp

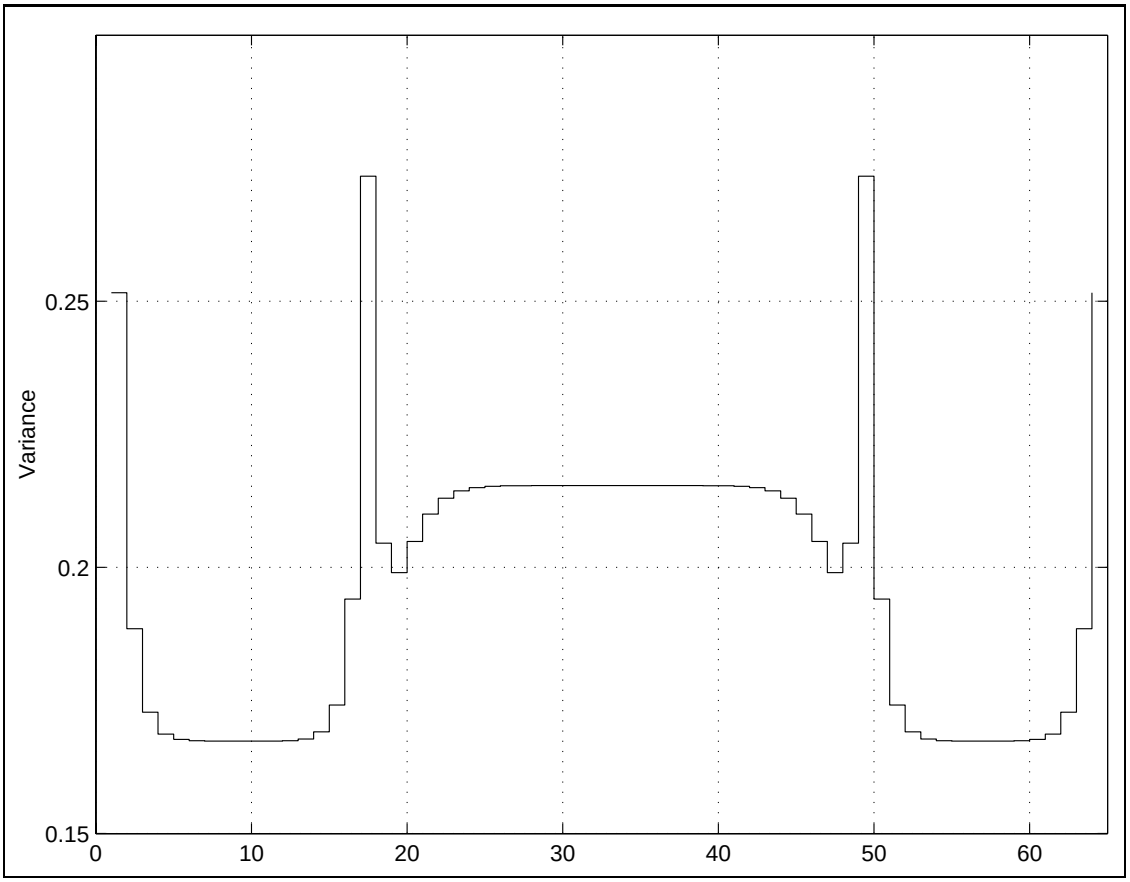

Figure 4.9: Variance for the Ramp

When we add noise to the step edge, we find a qualitative change. In figure 4.4 we show the input signal. figures 4.5 and 4.6 show the calculated s values and the variances respectively. We do find lower values of s around the step, but we have other areas where s is low as well. This makes sense as we need more information to distinguish between the gray-value changes that are due to noise and those that are due to actual presence of features. The variance measure, however, turns out to be more robust. Addition of noise does increase the variance in places, but these tend to be isolated peaks. Thus, we can remove these by a simple smoothing operation. The plot of the smoothed variance in figure 4.6

We treat ramp signals similarly, and plot the input, the s values as well as the variances in figures 4.7, 4.8 and 4.9 respectively. We find that the starting and the terminating point of the ramp are selected as features. Furthermore, the points inside the slope are slightly better as features than points outside.

4.10 Finding Features for the Stereo Correspondence Problem

We will explore how we can apply similar considerations to the problem of computing stereo correspondence. Stereo vision is the ability of humans and other animals to detect depth of field (i.e. to detect how far away an object is relative to the other objects). The position of two eyes creates two slightly images to form on the two retinas. The apparent displacement of nearby objects is larger than the apparent displacements of further objects. Thus, detection of distance is reduced to finding

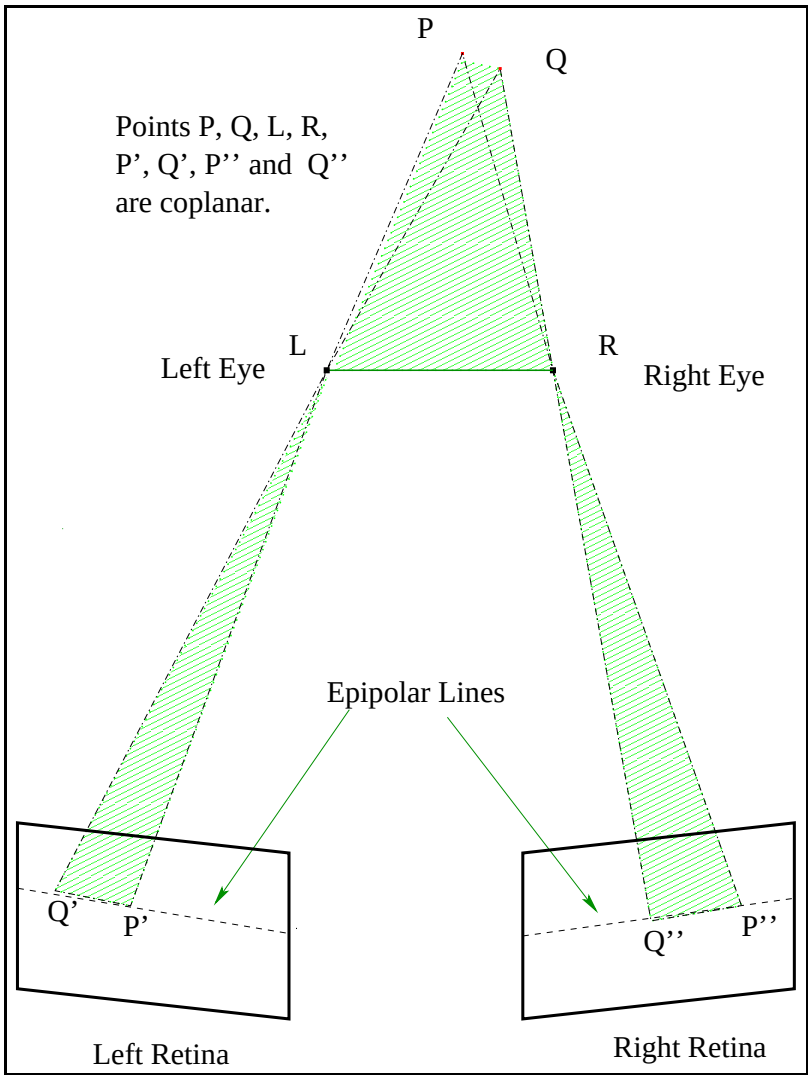

Figure 4.10: Geometry of Epipolar Lines

matching fragments in the two images and calculating their displacements.

Before we plunge into a description of how we can find features which help us in stereo reconstruction; let us first explain the approaches that has been taken to solve this problem. The problem involves a fairly heavy use of projective geometry and we will first explain this. Once we understand the geometrical foundations, we can go on to explain the algorithms for computing stereo correspondence, and review some of the literature.

4.11 Geometry of Stereo: Matching on Epipolar Line

Let us understand the geometry of stereo reconstruction. In figure 4.10, L and R are the left and the right eyes respectively and we also see the corresponding retinal planes. Given any point P in space, we consider the place formed by the three points P, L and R. This plane intersects the two retinal planes in two lines l' and l''. These lines correspond to each other, in the sense that the two image points P' and P'' of P lie on them. For any other point Q lying in the same plane (formed by P, L and R), the images Q' and Q'' of Q in the two retinal planes, lie in the same lines l' and l''.

Thus, the (unknown) geometry of the three-dimensional scene and its location with respect to the foci and the two retinas, generate partitions of the two retinal images into corresponding pairs of lines, one in each image. If the straight lines l and l' correspond to each other under this correspondence, then the point corresponding to $P \in l$ will be a point in l' (if it exists at all) and conversely. Thus, we can try independently matching up pairs of corresponding lines and will automatically get

the global solution for stereo Such pairs of corresponding lines are called epipolar lines.

Identifying epipolar lines simplifies the stereo problem immensely. If we know that l' and l'' are corresponding epipolar lines, we know for certain that the point corresponding to $P \in l'$ will be a point in l'' (if it exists at all) and conversely. Thus, we can try matching up pairs of corresponding epipolar line and will automatically get the global solution for stereo. Thus computing the match can proceed independently for each such pair.

4.11.1 A Simple Example

In figure 4.11 we show a schematic of stereo vision. The scene consists of a background object and a foreground object. We show a pair of corresponding epipolar lines on the two retinal planes. These correspond to the line PQ on the foreground, and the dotted line $ABCD$ on the background object. From the figure it is clear that the left eye cannot see anything between B and D, while the right eye cannot see anything between A and C. Thus, the part of the background between B and C is completely covered by the foreground object. We also see a point T on the background which is completely visible from both the eyes.

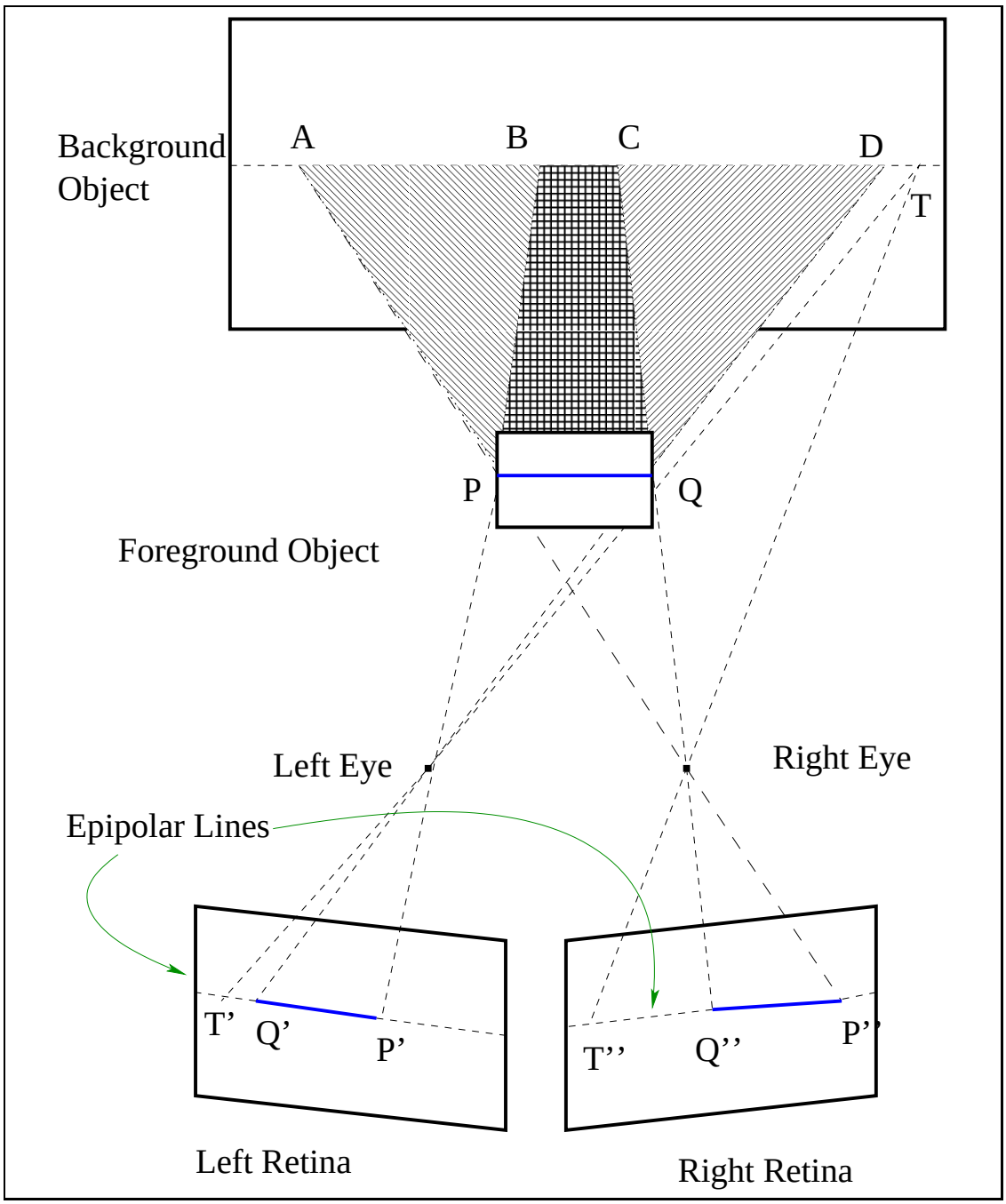

Figure 4.11: Geometry of Stereo

4.12 Algorithms for Stereo Correspondence

Prior Work

A number of researchers have come up with a dynamic programming approach to calculate stereo correspondence, and compute the disparity function. Some of the early theories were motivated by theories of the human visual system, which, in their turn were inspired by experiments with Julesz random dot stereo-grams. Some of these are described in book by Sperling ([111]) and Julesz ([67]).

Many researchers have proposed using dynamic programming to calculate the disparity map. Ohta and Kanade ([88]) proposed using dynamic programming to detect epipolar lines as well as to perform matching within the epipolar lines. Kanade and Okutomi ([69]) proposed using adaptive windows for the local disparity values. Other work in this direction includes Pollard's use of disparity gradients (See [96]).

A lot of work has gone into choosing the correct cost function for dynamic programming as well. While the exact form of this cost function does not change our description of the feature selection algorithm, its analytical properties (particularly convexity properties) affect the quality of the features we derive. Thus, Geiger et. al. ([46]) described a cost function that is concave for small values of its arguments, and this is the one we use for our experiments. Other cost functions have been described by Yuille et. al. ([124]). Other cost functions have been suggested which can be optimized using graphical algorithms (e.g. minimum cut algorithms). Ishikawa et.al. ([59]) as well as Boykov et. al. ([26]) proposed cost functions and

minimized them by reducing the problem to a graph algorithm. Earlier, Roy and Cox ([103]) proposed a maximum flow formulation for the stereo correspondence problem. Marr and Poggio ([82, 83]), gave a bayesian formulation to the stereo correspondence problem. The role of features like junctions and edges in stereo has been pointed out by many researchers. See, for example, Anderson ([4]) and Malik ([80]).

A Minimalistic Dynamic Programming Algorithm for Computing Stereo Correspondence

We will not concern ourselves with how epipolar lines can be detected, but rather assume from the outset that we are given a pair of epipolar lines which correspond to each other. Let us consider a point sufficiently toward the edge of the retinal plane, so that we can be sure that there are no occlusions in that area and so that we can determine two points, one on each line which correspond to each other via the stereo correspondence. For example, in figure 4.11 we have the points T' and T''.

We travel along the two epipolar lines l_1 and l_2, matching points as we go. Let the variable s_1 measure the arc length along l_1 and s_2 along l_2.

We will assume that the objects are sufficiently far away from the eye so that differences in arc length accurately reflect differences in distance from the eyes.

To understand the topology of the collection of possible matchings, we draw a graph with two axes, the horizontal axis being the distance along the right image, and the vertical axis plotting the distance along the left image. Any possible match

between these two images appears as a path in this graph.

We look at a simple example in figure 4.12. The left image has a red patch 4 pixels from the left; while the right image has the same patch two pixels from the left. We draw the two images on the corresponding axes. Any match corresponds to a line drawn in this graph. We have to allow for jumps, either horizontal, or vertical. These are the places where the disparity changes; i.e. where there is a partial occlusion. Since partial occlusions are caused by differences in depth of the objects in the three dimensional scene, the points of disparity change are the ones that we concentrate on.

In figure 4.12 we have drawn a particular match in green. It is clear from the figure that at any point of the match, one of three things can happen to the succeeding point:

1. The next point has both the left coordinate as well as the right coordinate, the immediate successor of those of the current point. This happens when the disparity does not change. Thus (in this particular match) the constructed depth in the three dimensional scene does not change.

2. The next point shows a vertical jump. Thus, on the right image the corresponding points are immediate neighbors, but there is a gap between them in the left image. Thus, the disparity changes. There is some pixels on the left image that the right eye cannot see, which means that the reconstructed surface comes closer to the viewer.

3. This case is the exact opposite of the previous one. The disparity changes in

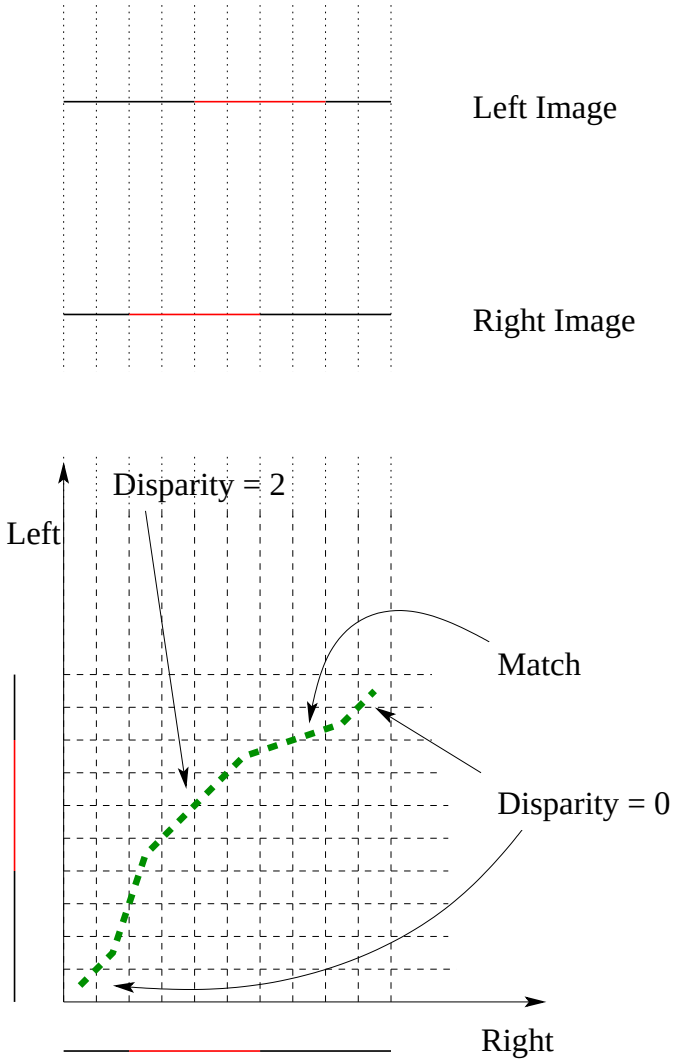

Figure 4.12: Two simple images and the l-r diagram showing their match

the opposite direction.

To make the ideas concrete let us give some formal definitions.

Definition 4.12.1 *A* match *is a sequence of pairs of points, one in each image, such that at least one component of consecutive pairs consists of consecutive points in the corresponding image.*

In this definition, we consider only epipolar lines, and so images are one dimensional. Let the (images of) the two epipolar lines be I_1 and I_2 respectively. Say, I_1 is the left image and I_2 is the right image. For points on the same image, we denote the relation of adjacency by $\sim$. Then, a match is a sequence

$$\{ \ (p_i, q_i) \mid p_i \in I_1, \ q_i \in I_2, \ ((p_i \sim p_{i-1}) \vee (q_i \sim q_{i-1}))\} \tag{4.24}$$

Definition 4.12.2 Disparity *at any point* $P = (p_1, p_2)$ *is defined as* $disp(P) = p_2 - p_1$.

We will associate a cost function with every match. In keeping with cost functions proposed for stereo correspondence problem by earlier literature, we want this cost to be small in two cases

1. If the corresponding pixels in the two images are similar. This is the data term.

2. If the match does not "stretch" too much. In other words, if the disparity does not vary too much from point to point. This is the prior.

Thus, the cost of a match, M, is

$$\mathcal{E}(M) \ = \ \mathcal{E}_{DATA}(M) \ + \ \mathcal{E}_{PRIOR}(M) \tag{4.25}$$

In what follows, the specific forms of the two terms do not matter too much. We only assume that they are of the following forms, where the match M is $\{(p_i, q_i) \mid i = 1, 2, \ldots, n\}$.

$$\mathcal{E}_{DATA}(M) = \sum_{i=1}^{n} F_1(I_1(p_i), I_2(q_i)) \tag{4.26}$$

$$\mathcal{E}_{PRIOR}(M) = \sum_{i=2}^{n} F_2(disp(p_i, q_i) - disp(p_{i-1}, q_{i-1})) \tag{4.27}$$

where F_1 and F_2 are functions and F_2 is symmetric about the origin. The function F_1 is a similarity measure which tells us how alike points $p_i \in I_1$ and $q_i \in I_2$ are as pixels.[3] Thus, instead of being a simple function of the pixel values, it can look at a window around the two points and determine how alike the parts of the images in the window are. F_1 is small (positive, ideally zero) when the two points are identical. Similarly, F_2 is a function, which is really a function of the absolute value of its argument. The value of F_2 increases as the absolute value of its argument increases. For simplicity, we restrict the possible values the disparity can take to be between $-W_{fd}$ and W_{fd}, where W_{fd} is a positive (integral) parameter.

The dynamic programming algorithm which minimizes this cost function is fairly straightforward, and resembles the viterbi algorithm from HMM literature.

[3]To be more realistic, we should include the neighboring pixels (even ones off the epipolar lines) in the $\mathcal{E}_{DATA}$. This is one way we can enforce (for example) continuity conditions stating that the computed disparity values are continuous functions of position.

4.13 Feature Selection for Stereo Correspondence

Now we note that this basic approach to stereo correspondence can be much improved if we incorporate information from features in our dynamic programming search. Many approaches to such feature-based stereo has been described in the literature. However, these approaches all use features which are chosen in an ad hoc manner. We now look at how our information theoretic approach towards feature selection can be used in this setting.

Thus, as in the case of one-dimensional functions before, we define a probability distribution on the set of allowed matches:

$$P(M) \propto \exp(-\mathcal{E}(M))$$

and as above, we normalize the probability to sum to 1 :

$$P(M) = \frac{\exp(-\mathcal{E}(M))}{Z} \tag{4.28}$$

where Z, the partition function is defined as

$$Z = \sum_{\text{M is a match}} \exp(-\mathcal{E}(M))$$

In the case of one-dimensional signals, we defined the variables s_i which denoted the presence or absence of a feature at location i, and used this to modify the posterior cost function, as well as the posterior probability distribution. Furthermore, we could interpret this modified probability distribution as the posterior distribution of the model given the image. Such an encoding is problematic in the case of stereo.

First of all, we would like to select subsets of both the left as well as the right epipolar lines as features. One possible course of action would be to set up pairs of variables s_i and t_j to encode the fact that the i'th pixel of the left image and the j'th pixel of the right image belong to the feature. We could then use these variables to modify the cost function in a similar manner. However, interpreting this modified cost function as the posterior probability of the model given the value of the feature is problematic.

4.13.1 Features for Stereo

The prime probabilistic structure in this formulation of epipolar line stereo is the probabilistic structure of the matches. Though there are particular pairs of points (one in the left image, the other in the right) which are more likely to be matched to one another, these probability values are never high enough that we can say with certainty that that particular match is (almost) certain. Many of these *"points of concentration"* are simply the locations of edges where the tendency of the matches to concentrate is driven by the local structure in each image. While these features might be important as generic features, we will argue that not all of these can be considered as good features for the stereo correspondence problem.

The primary task of the stereo correspondence algorithm is the computation of the disparity map. an image feature helps in this algorithm if it helps resolve any ambiguity in the disparity map at that point. Thus, in regions of the image which can be unambiguously placed in either the foreground or the background, features of the individual images will not occur as features for the stereo correspondence

problem. The ability to judge the pertinence of image features for the task of stereo correspondence calculation thus cuts down the number of unnecessary features and leads to a more compact input to the dynamic programming problem.

4.13.2 Feature Selection for Stereo

Now that we know what a good feature selection algorithm should generate, let us look at how we can approach feature selection. As we indicated earlier, it is difficult to even formulate a (regularized) KL-distance measure that will help us in this task. However, we can fall back upon the approximate approach we discovered for the case of one-dimensional signal. There we discovered (empirically) that regions of high variance indicate the presence of features.

We can apply that criterion in this case. Thus we are led to a feature selection algorithm that starts with a probability distribution on the set of matches given by equation 4.28 and calculates the probabilities of matches for every possible pair of pixels, one on the left image and the other on the right image. From this marginal distribution, we compute, for every pixel on the left image, the variance of the distribution of matching right pixel, attaching a fixed cost to the case where there is no match. The locations on the left image where this variance is high is considered a feature for stereo. Similarly, we compute these distributions for the right image and get corresponding features.

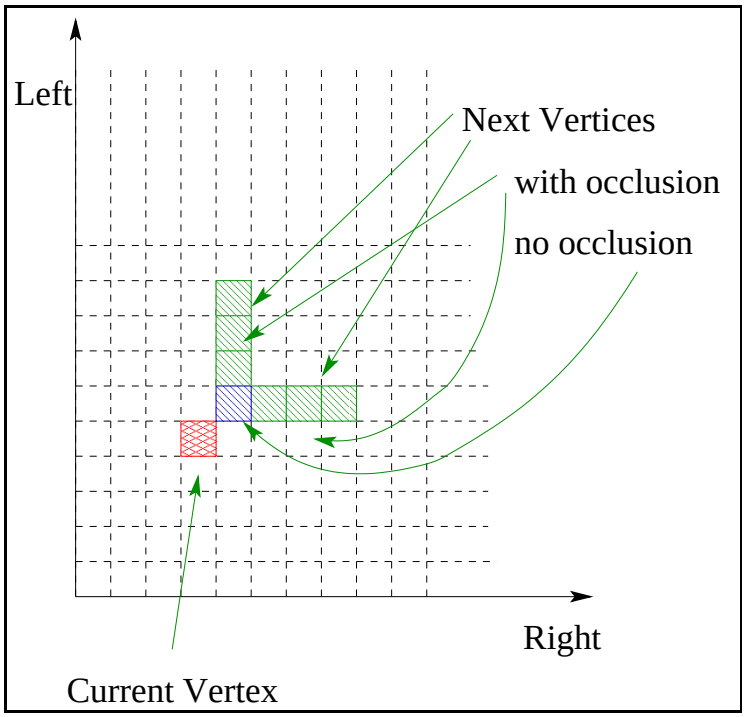

Figure 4.13: Neighborhood structure of an l-r diagram

4.14 Computations

Now that we have an idea of what we want to achieve, we describe our algorithm in greater detail. The most difficult step of the algorithm is the computation of the marginal described above. We now describe the computation of this marginal. We will achieve somewhat more, as we will end up computing not only the probabilities of particular matches, but also the transition probabilities between them.

4.14.1 Neighborhood Structure of *l-r* Diagrams and Computation of Path Probabilities

As before, we work in the space of pairs of points, the first from the left image, I_1, the second from the right image, I_2. We can think of this space as an $N \times N$ square, where N is the number of pixels in each epipolar line. We call this square variously *l-r square*, *l-r space*, or *l-r diagram*. The two components will be called the *l-component* and the *r-component* respectively. In this picture, a match is a sequence of points of the *l-r space*, such that for any two consecutive points of the sequence, either the l-components or the r-components are consecutive (or both). We will refer to a match as a *path*.

The structure of neighborhood of a point is determined by the definition of a match. Equation 4.24 specifies what a match can look like in the stereo problem and we use this to define our neighborhood structure. We show this in figure 4.13 where the green and the blue cells are the neighbors (in one direction) of the red cell. The blue cell corresponds to the case where there is no further change in disparity. If the match goes from the red cell to any of the green cells the disparity

changes.

The particular neighborhood topology forces us to change our algorithms for calculating the path probabilities. Since we can compute the costs fairly easily, we can compute the path-products (Π) quite simply by exponentiation. However, computation of the probabilities involves our being able to normalize these products in a consistent way.

We now show how a generalization of the earlier approach lets us do this computation. The first part of the computation is similar to one described in [92]. There the computation of α and β is called belief propagation. In our work, we use these values in a novel way to compute vertex and edge probabilities for an arbitrary acyclic graph.

For every point in the *l-r space*, we want to calculate the probability that a path goes through that point. As in the case of one-dimensional signals, here too do we have a concept of *path-product*, denoted by Π.

Notation

To simplify notation, we denote the the sum of the path-products of all paths between two points p and q in the *l-r square* by $p \longrightarrow q$. We also consider paths that are of a fixed length t. Thus $p \xrightarrow{t} q$ denotes the sum of path-products of all length-t sub-paths between p and q; and $p \xrightarrow{t} q \longrightarrow r$ denotes the same sum over all sub-paths from p to r that go through q and which takes t steps between p and q. We also have notation for denoting that a path takes at most t steps between p and q: $p \xrightarrow{\leq t} q$. In the above, we have to make a consistent decision as to whether

we include the local costs at the end points in the sum of path-products. Also, by sub-path, we mean that we take just the costs that lie between the relevant points.

4.14.2 Probability Distribution on the Paths

The probability distributions on paths we consider here are similar to *Gibbs Distributions* that arise in *Markov Random Fields (MRF)* . However, the difference is that, unlike in the case of MRF-s, we do not have a well defined boundary structure such that all the influence of the "rest" of the universe can be factored through the boundary.

To define the probability distribution, assume there is a multiplicative factor $\Pi(v)$ associated with each vertex v, and a multiplicative factor $\Pi(e)$ associated with each edge e. If e connects vertices v_1 and v_2, we will denote this last factor by $\Pi(v_1, v_2)$. In our example of stereo,

$$\begin{aligned} \Pi(v) &= \exp[-F_1(p,q)] \\ \Pi(v_1,v_2) &= \exp[-F_2(disp(p_2,q_2) - disp(p_1,q_1))] \end{aligned}$$

where F_1 and F_2 are as in equations 4.26 and 4.27 and the vertices v v_1 and v_2 correspond to matching p, p_1 and p_2 respectively with q, q_1 and q_2.

Now we can define the probability distribution on the set of paths from the source vertex s to the destination vertex d.

In our case of stereo, the source and destination vertices correspond to finding corresponding points on the epipolar line which are on opposite ends of the retinal planes and which are sufficiently far away from center of the three dimensional scene that we know their correspondence exactly.

Let p be the path consisting of the vertices $s = v_0, v_1, v_2, \ldots, v_r = d$. Let e_i be the edge joining v_i to v_{i+1}. Then the probability $\mathbf{Pr}(p)$ of the path p satisfies

$$\mathbf{Pr}(p) \propto \Pi(v_0)\Pi(e_0)\Pi(v_1)\cdots\Pi(e_{r-1})\Pi(v_r)$$

To get actual probabilities, we have to normalize the right hand side so that they sum to 1 when summed over all the allowed paths. To simplify the notation, we will call the quantity on the right hand side the *path product* for the path p. We will denote the path product of p by $\Pi(p)$.

$$\begin{aligned} \Pi(p) &= \Pi(v_0)\Pi(e_0)\Pi(v_1)\cdots\Pi(e_{r-1})\Pi(v_r) \\ \mathbf{Pr}(p) &= \frac{\Pi(p)}{\sum_p \Pi(p)} \end{aligned}$$

where the summation in the denominator is over all the paths of the graph which start at s, the source, and end in d, the destination; i.e. the legal paths. We note that this is the same expression (in a different notation) as that given in equation 4.28.

We can do this computation explicitly, but it suffers from the twin problems of speed and accuracy. Generation of each path is expensive; and often there are overflows or underflows in intermediate results, and we end up with $0/0$ or ∞/∞ forms.

Notice that when we express the multiplicative factors (the Π's) as negative exponentials of costs (or energies), we get the standard form used in *Gibbs distributions*. The situation is much more complicated because of the topology of the

underlying graph is not that of a lattice.

1. Paths between any two vertices might have varying lengths.

2. Given any vertex, there might not be a (small) set of vertices such that any path which includes the given vertex must include a vertex of the set. i.e. we might not be able to define boundaries. (**Loss of Markov property**)

Below we present an algorithm which is applicable to any finite acyclic graph.

4.15 Formal Calculation of Probabilities

For every vertex of the graph, we want to calculate the probability that a path goes through that vertex. To simplify notation, we denote the the sum of the path-products of all paths between two vertices v_1 and v_2 of the graph by $v_1 \longrightarrow v_2$.

$$(v_1 \longrightarrow v_2) = \sum_{p \text{ goes from } v_1 \text{ to } v_2} \Pi(p) \tag{4.29}$$

We also consider paths that are of a fixed length t. Thus $v_1 \stackrel{t}{\longrightarrow} v_2$ denotes the sum of path-products of all $t - length$ sub-paths between v_1 and v_2; and $v_1 \stackrel{t}{\longrightarrow} v_2 \longrightarrow v_3$ denotes the same sum over all sub-paths from v_1 to v_3 that go through v_2 and which takes t steps between v_1 and v_2. We also have notation for denoting that a path takes at most t steps between v_1 and v_2: $v_1 \stackrel{\leq t}{\longrightarrow} v_2$. In the above, we have to make a consistent decision as to whether we include the factors at the terminating vertices in the sum of path-products.

The naive calculation of the marginals at the vertices create intermediate sums that are too large. To prevent this from happening, we define our final probability in terms of intermediate quantities which are themselves other (conditional) probabilities. Thus, we avoid the problem of handling too large numbers. Intuitively, this is similar to carrying various scale of quantities along the computation, and changing the current scale whenever the value of a path-probability becomes too large or too small.

With this notation, we can define α and β. These are functions of two variables, v, which varies over the set of vertices, and t, which varies over the set of positive integers. This generalizes the α and β variables of one dimensional models (and HMM-s) in that for those there was no dependence on t.

The variable t counts the path-lengths of various paths from the source s to the variable vertex v. Thus, we should work with graphs which has an upper bound on the maximum value of t. We are also fine if the edge factors and vertex factors are small enough that too many cycles become improbable and so we can fix a cut-off for t.

Then, we define eight quantities:

$$\begin{aligned}
\alpha[t][v] &= \frac{(s \xrightarrow{t} v)}{\sum_v (s \xrightarrow{t} v)} \\
\alpha'[t][v] &= \frac{(s \xrightarrow{\leq t} v)}{\sum_v (s \xrightarrow{\leq t} v)} \\
\beta[t][v] &= \frac{(v \xrightarrow{t} d)}{\sum_v (v \xrightarrow{t} d)} \\
\beta'[t][v] &= \frac{(v \xrightarrow{\leq t} d)}{\sum_v (v \xrightarrow{\leq t} d)}
\end{aligned}$$

We have encountered (special cases of) α and β before. The primed quantities α' and β' are normalized value which measure path-products of partial paths of length bounded by t. Not all of these quantities are algebraically necessary, but we use one or the other depending on which particular value leads to a 0/0 form.

$$\begin{aligned}
\gamma[t] &= \frac{\sum_p (s \xrightarrow{t} p)}{\sum_p (s \xrightarrow{t-1} p)} \\
\gamma'[t] &= \frac{\sum_p (s \xrightarrow{\leq t} p)}{\sum_p (s \xrightarrow{\leq t-1} p)} \\
\delta[t] &= \frac{\sum_p (p \xrightarrow{t} d)}{\sum_p (p \xrightarrow{t-1} d)} \\
\delta'[t] &= \frac{\sum_p (p \xrightarrow{\leq t} d)}{\sum_p (p \xrightarrow{\leq t-1} d)}
\end{aligned}$$

The quantities γ, δ and their primed counterparts are the ratios of the normalization constants involved in the computation of α and β for successive values of t (and similarly for the primed versions).

$$
\begin{aligned}
\alpha[t][v] &\leftarrow \sum_{v'} \alpha[t-1][v'] \ (v' \longrightarrow v) \\
\gamma[t-1] &\leftarrow \sum_{v} \alpha[t][v] \\
\alpha[t][v] &\leftarrow \alpha[t][v] \ / \ \gamma[t-1]
\end{aligned}
$$

$$
\begin{aligned}
\alpha'[t][v] &\leftarrow \begin{cases} \sum_{v'} \alpha'[t-1][v'] \ (v' \longrightarrow v) & \text{if } v \neq s \\ \alpha'[t-1][v] & \text{else} \end{cases} \\
\gamma'[t-1] &\leftarrow \sum_{v} \alpha'[t][v] \\
\alpha'[t][v] &\leftarrow \alpha'[t][v] \ / \ \gamma'[t-1]
\end{aligned}
$$

$$
\begin{aligned}
\beta[t][v] &\leftarrow \sum_{v'} \beta[t-1][v'] \ (v \longrightarrow v') \\
\delta[t-1] &\leftarrow \sum_{v} \beta[t][v] \\
\beta[t][v] &\leftarrow \beta[t][v] \ / \ \delta[t-1]
\end{aligned}
$$

$$
\begin{aligned}
\beta'[t][v] &\leftarrow \begin{cases} \sum_{v'} \beta'[t-1][v'] \ (v \longrightarrow v') & \text{if } v \neq d \\ \beta'[t-1][v] & \text{else} \end{cases} \\
\delta'[t-1] &\leftarrow \sum_{v} \beta'[t][v] \\
\beta'[t][v] &\leftarrow \beta'[t][v] \ / \ \delta'[t-1]
\end{aligned}
$$

Figure 4.14: Recursive Computation of α, β and other quantities

Probabilistic Interpretation and Recursive Computation

Let us discuss the probabilistic interpretation of these quantities. We will just discuss the meaning of α and γ. To do that, first assume that in the original definition of the probability of a path, instead of constraining the paths to begin at s and end at d, we put the weaker constraint: begin at s, as before, but end at any vertex. Thus the probability space is changed. (Similarly β is defined in yet another probability space in which d is kept fixed, but the restriction on the source is lifted.) Then, in this new probability space, $\alpha[t][v]$ is the probability that a path ends at v, conditioned on the fact that it is of length t. $\gamma[t]$ is the normalization constant that occurs when we try to calculate the $\alpha[t]$s from the $\alpha[t-1]$s.

The recursive calculation is shown in figure 4.14. Note that, in the figure, the $\leftarrow$ sign actually means assignment. Thus the quantity on the left hand side denotes the new value of that quantity, while if it occurs on the right hand side of the same assignment, it denotes the old value.

By an accident, we end up calculating the values for t, from the values for $t-1$, thus t denotes both path lengths as well as "time".

First we calculate αs and γs recursively as also βs and δs as well as their primed counterparts. Notice how γ fits in as the normalization constant for α.

Connection to Pearl's Belief Propagation Algorithm

The variables α and β we defined above are related to the belief values as defined in Pearl's belief propagation algorithm (see [92]). We could interpret our $\alpha[t][v]$ as the (forward) belief value at the vertex v at time step t. Pearl was interested in the

final values of the beliefs. However, we use the time dependence to calculate the actual probabilities of vertices and edges.

4.15.1 Computation of Vertex and Edge Probabilities

We compute the vertex and probabilities according to the following strategy. For every t' we calculate the conditional probability of a vertex conditioned on the (complete) path lengths being t'. Then if we can compute the probabilities of the path lengths, we have completed our task. Now the conditional probability itself is expressed as a sum of conditionals, one for each distance the vertex v can be away from s, the starting vertex.

We use the γ and δ values for calculating the probability distribution of "time", i.e. the probabilities that a path is of a particular length. We first calculate the cumulative probability function, $\mathbf{Pr}(\text{path has length} \leq t)$. When t is the maximum value for number of steps, this probability is 1. We calculate the ratios of successive probabilities:

$$\begin{aligned} \frac{\mathbf{Pr}(\text{path has length} \leq t)}{\mathbf{Pr}(\text{path has length} \leq t+1)} &= \frac{(s \stackrel{\leq t}{\longrightarrow} d)}{(s \stackrel{\leq t+1}{\longrightarrow} d)} \\ &= \frac{\alpha'[t][d]}{\alpha'[t+1][d]} \frac{1}{\gamma'[t]} \end{aligned}$$

Once the cumulative probabilities are calculated, the actual probability distribution for time can be calculated by differencing.

We can thus implement our strategy. We denote by $\mathbf{Pr}(t)$ the probability that a path is of length exactly t. Then, to calculate the probability that a point, p, is on a path, we first calculate, for all values of t, the probability that p is at t steps

from s, given that the paths are of length t'. Call this last probability $\mathbf{Pr}[v][t][t']$. Then

$$\mathbf{Pr}[v][t][t'] = \frac{\alpha[t][v]\ \beta[t'-t][v]}{\sum_p \alpha[t][v]\ \beta[t'-t][v]}$$

gives the conditionals mentioned above.

Then, we can calculate the probability the probability that a point v belongs to a path by a simple summation:

$$\mathbf{Pr}(v) = \sum_{t'} \sum_{t} \mathbf{Pr}[v][t][t']\ \mathbf{Pr}(t')$$

In this way, we can compute the various point probabilities involved.

4.15.2 Edge Probabilities

We can also calculate the edge (or transition) probabilities using a similar strategy. The formulas are more complicated, but the idea is the same one of breaking up the probability as a conditional probabilities over the total path length, and calculating each conditional by a sum of separate conditionals. Each term in this sum depends on the position of our edge of interest in terms of the number of steps from the start vertex.

Thus, the relevant quantity to calculate in terms of α and β is

$$\mathbf{Pr}'[v][v'][t][t'] = \frac{\alpha[t][v](v \longrightarrow v')\beta[t'-t-1][v']}{\sum_{\{v,v'\}} \alpha[t][v](v \longrightarrow v')\beta[t'-t-1][v']}$$

This gives the probability of an edge at t steps from s and going from v to v', given that the total path-length is t'.

Thus, we get the probability of the edge by summing these quantities weighted by $\mathbf{Pr}(t')$, the probability of a path being of length t'.

$$\mathbf{Pr}(v, v') \;=\; \sum_{t'} (\sum_{t} \mathbf{Pr}'[v][v'][t][t'])\mathbf{Pr}(t')$$

4.16 Results for stereo: variances

We now look at how our feature selection algorithm performs. The test set we run our algorithm on must have certain characteristics.

1. First of all, we should have relatively large areas of disparity discontinuities, indicating the presence of half-occluded regions.

2. There should be irrelevant features in the unoccluded part of the image which our algorithm should reject.

3. Clearly the parts of the image that are interesting lie near the occlusion boundaries. If there are features in this region, which decrease the uncertainty of locating the occlusion boundary, then the problem is simplified.

The input we use is chosen so as to create an adversarial situation, where the above criteria are satisfied. Figure 4.15 shows the left and the right image. This is a fairly complicated image with illusory contours. The three dimensional scene which generated this image consisted of a white cross in a black box. Floating in

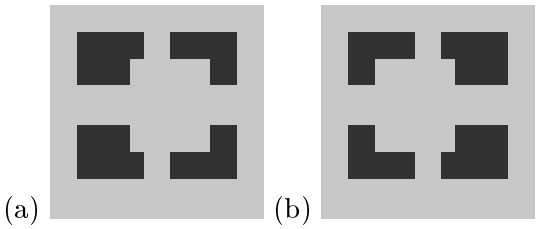

Figure 4.15: Input image: box in front of a cross: (a) left, (b) right. Notice the displacement of the box

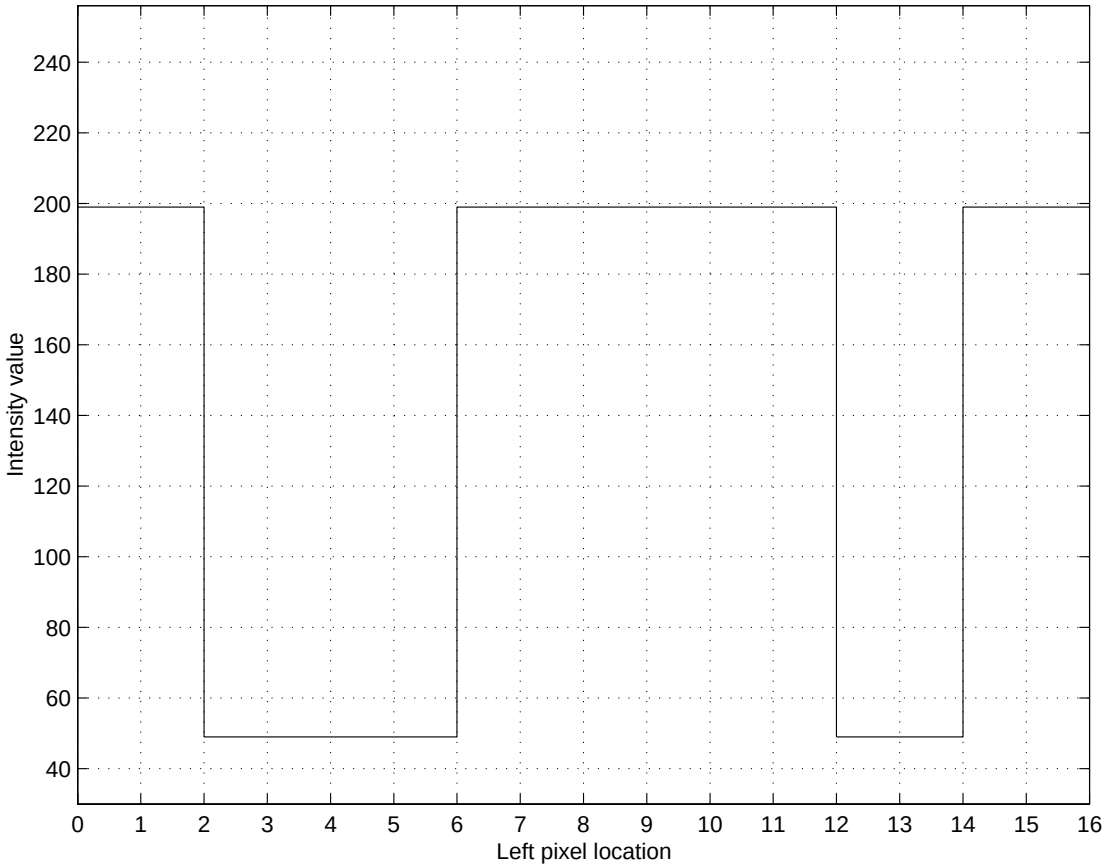

Figure 4.16: Intensity values along the epipolar line: left image

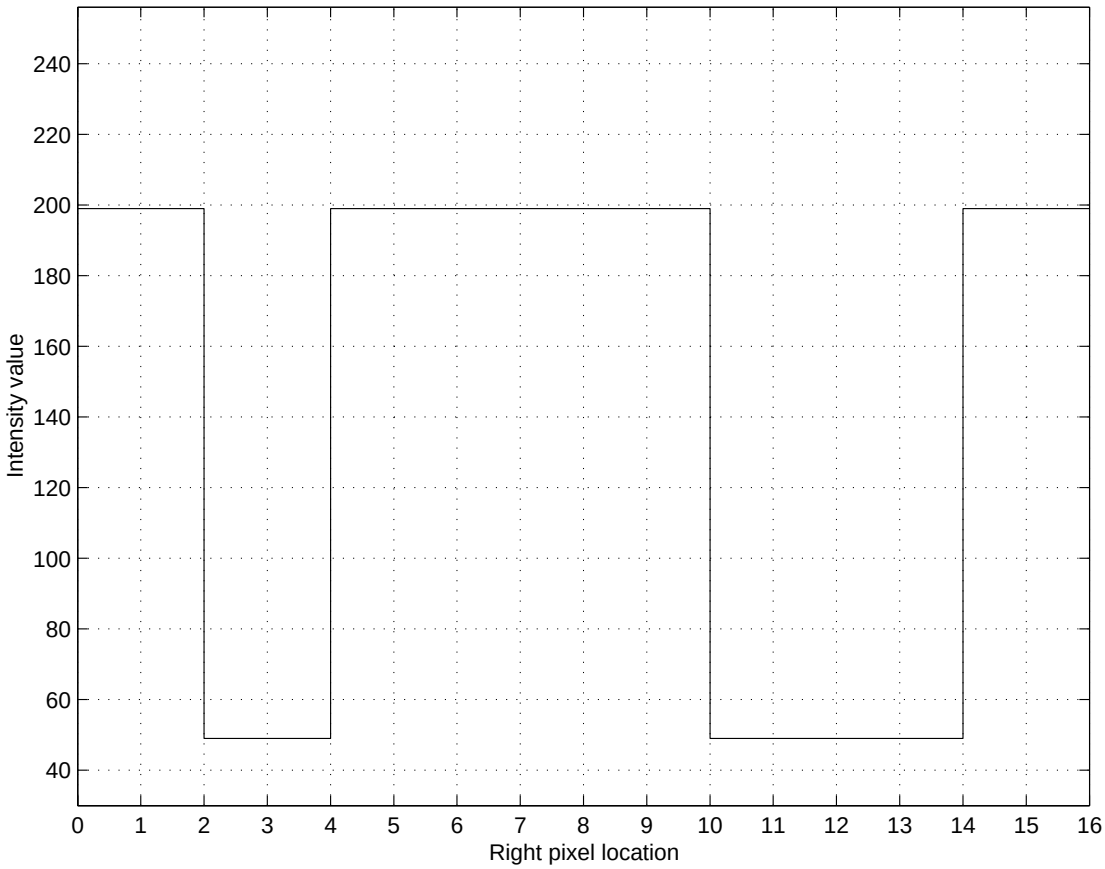

Figure 4.17: Intensity values along the epipolar line: right image

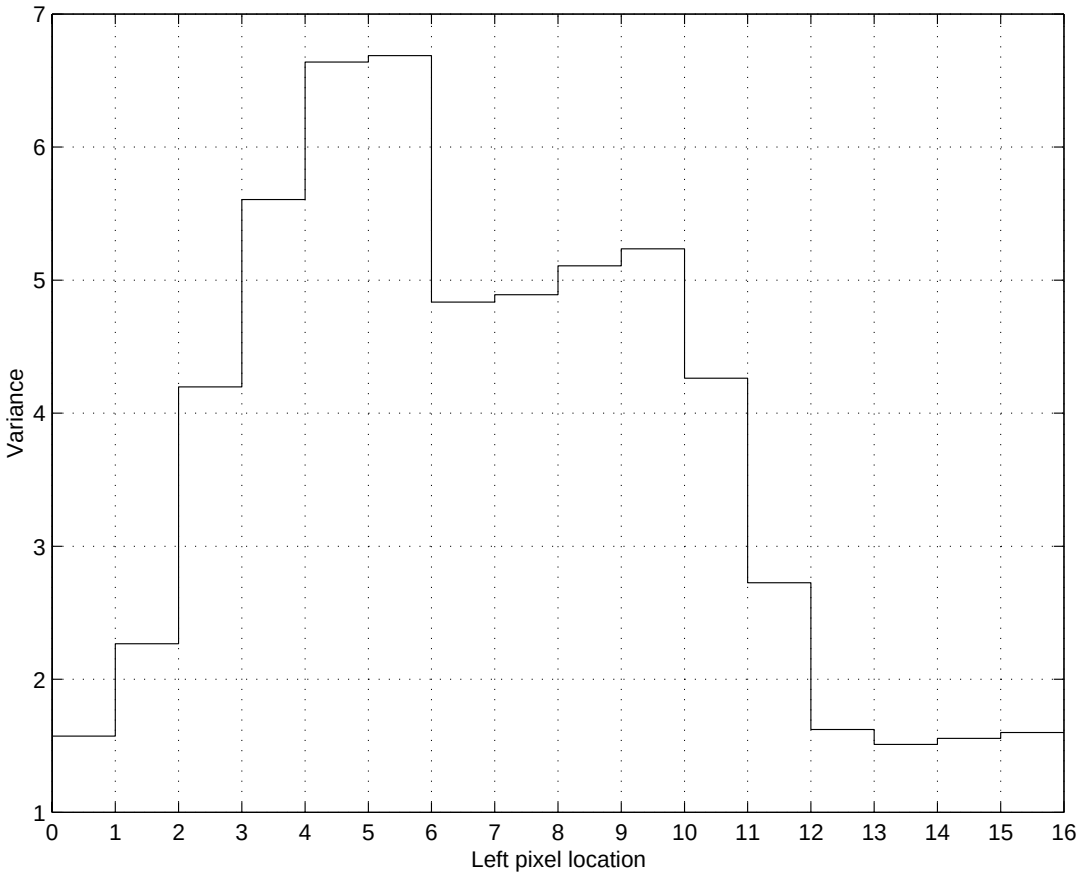

Figure 4.18: Variance Plots: left image

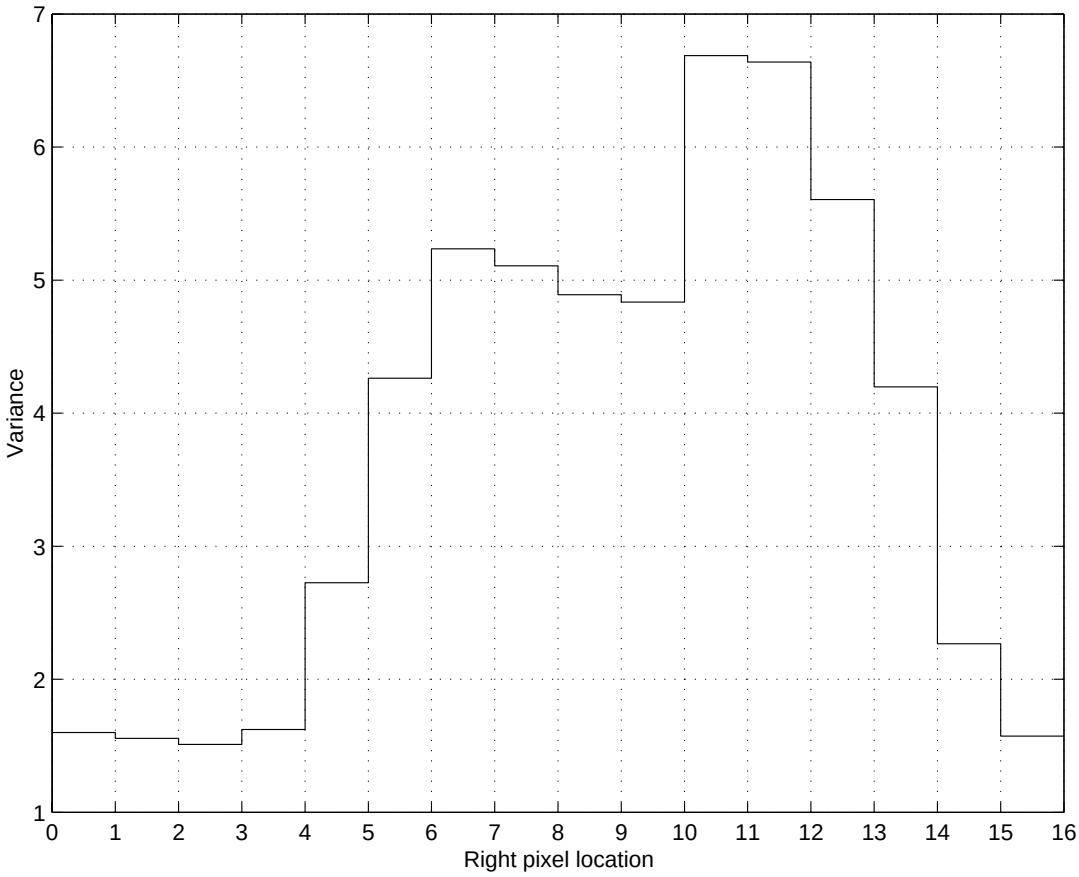

Figure 4.19: Variance Plots: left image

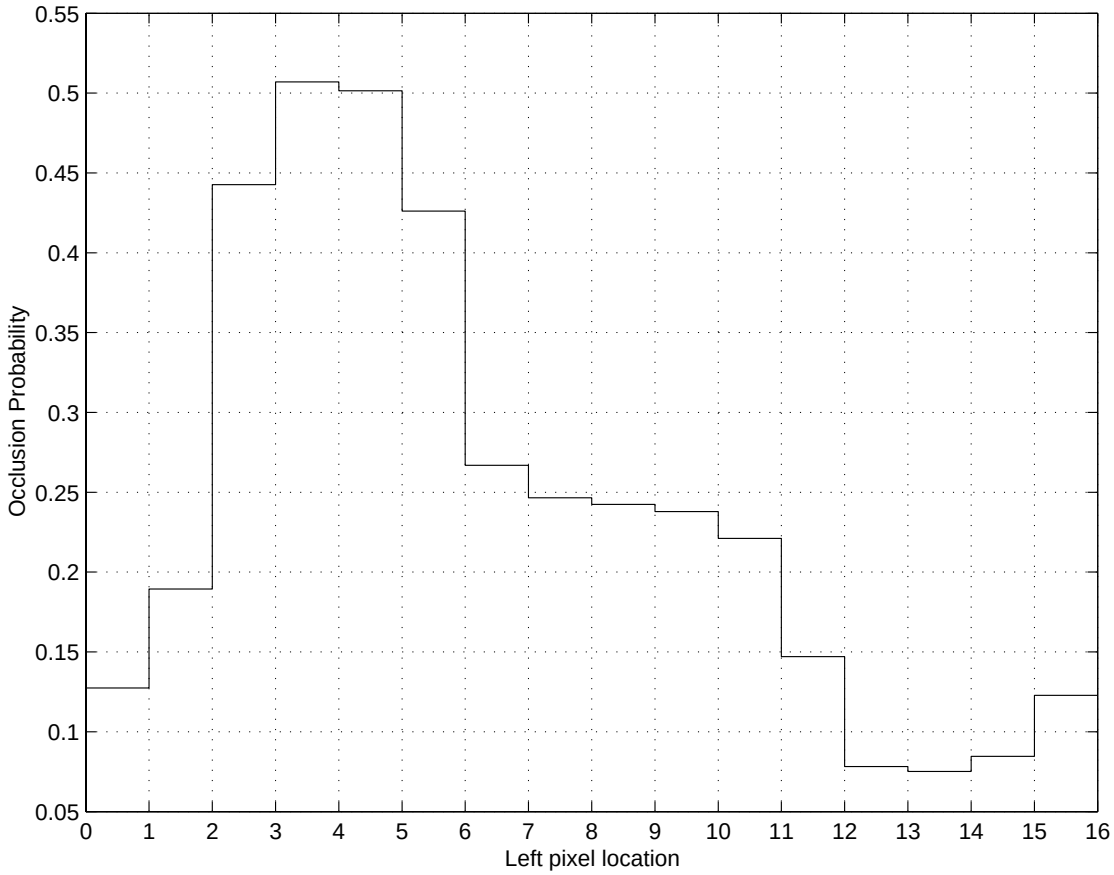

Figure 4.20: Plot of occlusion probability: probability that a pixel location on the left image is invisible from the right

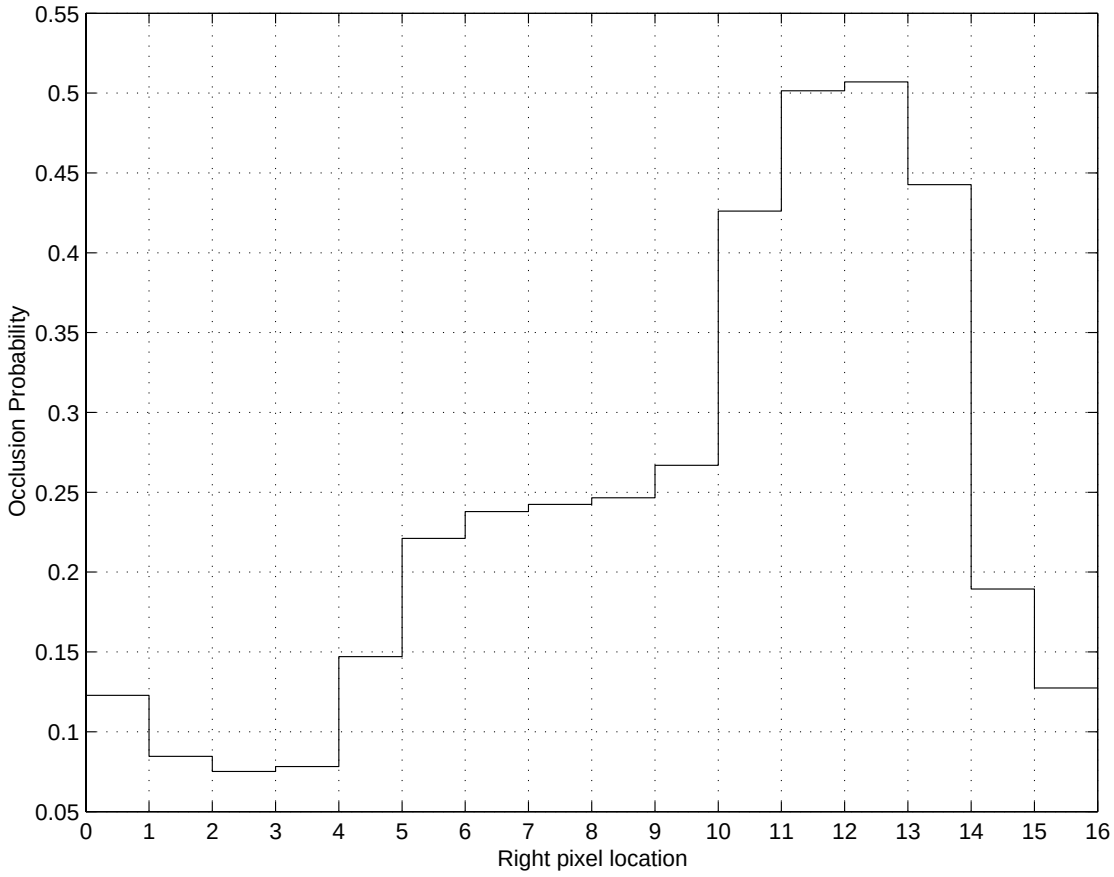

Figure 4.21: Plot of occlusion probability: probability that a pixel location on the right image is invisible from the left

front of the cross is the image of a white box. There is an illusory contour because the intensity of the box and that of the cross coincide.

We concentrate on the part of the image that does not have the illusory contour, but still has non-zero disparity gradient. This would be the narrow horizontal strip near the top and bottom of the picture which consists of a light region followed by a dark region followed, in that order by a light, a dark and a light region. We indicate the intensity profiles along the epipolar lines in figures 4.16 and 4.17. The 5th and 6th pixels on the left image are invisible from the right. Similarly, the 11th and 12th pixels on the right image are invisible from the left. Clearly, for this pair of input images the leftmost edge (between the 2nd and 3rd pixels) and the rightmost edge (between the 14th and 15th pixels) are non-essential.

We show the computed variance values in figures 4.18 and 4.19. The local maxima in the variance plots coincide with pixels which are invisible from the other side. Furthermore, the irrelevant edges are not picked up. Thus, the variance approach for feature discovery yields features which satisfy the criteria we delineated above.

We also plot the occlusion probability in figures 4.20 and 4.21. We see from these plots that the occluded regions are correctly identified as such.

4.17 Discussion and Summary

At the end of this chapter, we come to the end of the first part of our thesis. In this part we looked at a principled method of selecting features. This criterion was the same as the notion of sufficient statistics. This connection between sufficient

statistics and informative features is new and is the central idea behind feature discovery.

In this chapter we looked at two concrete problems to see how the abstract formulation could be applied. In the first example, that of estimating the model for one-dimensional signals, we had a feature space which could be parameterized very simply using a single vector. The feature selection task is therefore easy to formulate and solve.

However, this approach leads to slow algorithms for feature selection due to slow convergence and the existence of local optima. We found that an approximate concept using variances led to very fast algorithms which performed well.

The set of features for the problem of stereo correspondence calculation does not have a nicely parameterization. For this reason, it is difficult to directly apply the general technique in this case. However, we can apply the variance method fairly simply, and this leads to an algorithm which avoids unnecessary features which might be otherwise present in the input, but which are not relevant for the task of computing the stereo correspondence.

While the specific cost functions involved in one-dimensional signal estimation are hardly new, the specific form of the cost function used for feature selection (as opposed to signal estimation) which results from our general approach is new. The approximate approach using variances is new as well. The same is true about our variance based approach to discovery of features for stereo.

In the succeeding chapters we investigate systematic ways in which features (whether optimal or not) can be used to yield information contained in them.

Chapter 5

Using Features: Associative Memory Problems

5.1 Introduction: The Generic Use of Features

In the previous chapters we have looked at an ab initio approach to feature-selection. We posited different tasks had different features, depending on how much task-relevant information a particular feature carries. Naively, this approach leads to completely different features for different tasks. However, we suggested that features for problems such as the model-selection problem are generic, in the sense that a feature for any other task is a function of a feature of the model-selection problem. Of course, not all features for model-selection need be relevant for every task. We saw an example of this phenomenon when we considered feature-selection for the stereo correspondence problem. In this problem, edges in the input in regions of no great change in disparity can be considered irrelevant because we can

infer the general shape of the disparity map (locally) even if we did not know of the presence of the edge.

In this and succeeding chapters, we stop looking at feature selection and, instead, look at how features can be used efficiently. The features we consider in this discussion could be discovered using any method. We even allow ad hoc features which are known to be useful in any particular problem domain. Thus, this is really an analysis of how features code the information they carry. However, even though we do not formalize the conditions we impose on our features, we deal with a fairly restricted class of features. To get an informal idea of what these limits are, we will consider problems of a specific form. As our discussion will indicate, however, we do not really lose any generality by doing this.

5.1.1 Fragment Completion

The class of task we consider can be described as *fragment completion*. Simply stated, it is the following:

Given a set M of items (the memory), how can we efficiently find a specific member of M from partial information about it.

The kind of information we will be given are fragments. Thus, if our memory is a dictionary of English words, we will be given a cue where some positions are blank and some are filled in, and our goal would be to fill in the blanks to get a valid English word, viz. one in the dictionary. For example, we could start with the cue "s__a_e" and complete it to get the word "sesame". In fact, in chapter 7 we consider such a memory of words.

Fragment completion for memories of words are special, in that the parts that are known are known exactly. This is not the case for memories which store images. Thus, given a fragment of an image, we can only demand that the completed item be approximately equal to the input cue over the region the fragment covers. In chapter 6 we look at an example of such a situation.

Now that we have an inkling of the kinds of problems at which we are looking, let us discuss the role features play in this kind of task. Most current approaches to detection of objects consist in calculating a feature vector for the input cue, and comparing this with the (pre-computed) feature vectors of the items in memory.

For many simple situations, where not too many features are occluded, this algorithm performs quite well. However, the drawback of this kind of approach is that if the input fragment covers too many features, our algorithm can be led astray.

We explicitly address the problem of retrieving items in memory from partial information. In particular, we focus on associative (or content-addressable) memory tasks. This is in contrast to the techniques described above, which try to match whole images without significant occlusion. In contrast, we actually look at the case where there is significant occlusion in the input image.

Furthermore, when we deal with more general kinds of features, for example, the features discovered using some approach as in the first part of the thesis, it is not clear that an approach of this kind is fruitful. For one thing, there might not be a clear notion of a position of the feature; or we might not be able to calculate the features for subdomains of the original image.

Much more importantly, our goal is to discover algorithms that manage to accomplish this task, without looking at all the items. Thus, perforce, we are restricted to looking at local characteristics of the set of items and use this knowledge, somehow, to aid in fragment completion. Sometimes this will prove impossible. In these cases we try to minimize the number of lookup steps we perform (though not explicitly).

The reason why we do not want to look at all the memory items, at least at the time we answer the queries, will become clear in the next subsection where we discuss the genericity of the class of problems we are considering. From a more practical point of view, such an algorithm will let us handle huge memories, which might be otherwise impossible to store in compute memory.

We allow ourselves the luxury of pre-treating the memory elements during which we try to learn the local characteristics of the items in memory. Our ability to do such a learning step is closely related to the question of a "local / global" decomposition of problems.

The (meta-)algorithm we come up with can be applied to this more general classes of features,[1] while still retaining a similarity to the above "minimum distance" algorithm which works in simple situations.

Our approach does not involve looking at the items (images). Rather, we learn statistical properties of the items in memory and use this knowledge to design an efficient look-up algorithm which makes very few mistakes.

[1] provided we can define certain "matching" functions

5.1.2 Without Loss of Generality?

However, the question that remains to be answered is whether the kind of fragment completion tasks we look at is general enough to shed light on other problems involving the use of features. We do not offer a definitive answer to this question. There are, however, some hints that we might not lose too much generality in considering this specific a problem.

Let us first look (again) at the way we argued that features for the model estimation problem are generic. The gist of that argument was that in a situation where the input is generated from a model, the conditional distribution $\mathbf{Pr}(I \mid m)$ of the input given the model contributes no useful information. In other words, it is pure noise. Thus, any useful information contained in the input (image) is also contained in the model, and thus if we can reliably estimate the model m , we can use m instead.

This line of argument suggests that model estimation is the whole reason features exist. While this is not entirely true (as we saw for stereo), it is a useful picture. At this point, we can construct an artificial scenario, where the models are the items in memory and the generative process consists of taking a fragment of an item. This process, at least in many situations, does not add any extra information; and thus the condition on $\mathbf{Pr}(I \mid m)$ is satisfied for it.[2]

[2]We ignore cases like stereo, where occlusion actually yields information about the three-dimensional position of objects. The argument we outline is, in any case, intended to be an indication that our problem has a good deal of applicability, even when the underlying problem is not one of content-addressable memories.

5.1.3 Incorporating Ad Hoc Features

The other issue we have to treat is what kind of features can we use for associative-memory problems of this kind? Our (meta-) algorithm is actually fairly general in this respect. We describe an architecture, which itself is pretty invariant, but which lets us incorporate diverse kinds of features; provided we define certain "matching functions". The effectiveness of the algorithm of course depends on the suitability of the given features for the given memory.

At this point, we do not consider addition and deletion of items to and from the memory. Thus, the specific set of items in the memory is static. Our algorithm will still work, however, so long we do not add or delete too many items, or if the new items are not too different statistically.

5.2 An Iterative Framework for Memory

Thus, as we explained in the introduction, we look at the problem of retrieving a particular item from a memory, given a fragment of it. In this chapter, we describe high-level algorithms for this problem. In later chapters, we look at specific problem domains, and see how we can instantiate our algorithm for these domains.

In chapter 6 we do this in the context of face recognition. Thus, we are given a library of face images. Given an image of a fragment of a face, possibly with slight changes in viewpoint and other noise, our task is to determine which face image in the library is the best completion of the input image. For faces, at least, there is a naive algorithm to do this: we look at each library image and compute some

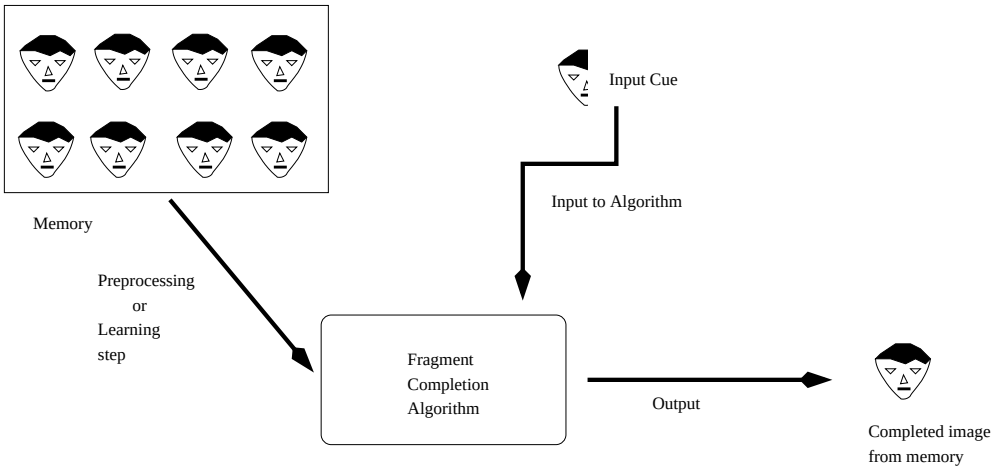

Figure 5.1: The Fragment Completion Task

distance function from the input image, choosing the library image closest according to this distance. For most problem domains, there are analogous methods which involve looking up the library of items.

Although these tasks are very challenging computationally, people perform them quite well, and in different problem domains. Frequently, partial information triggers a complete memory. At the same time this process is not perfect, as we will see from human performance on completion of word cues, described in a later chapter.

An Example

To explain our computational model, consider the example in which one stores a set of face images in some database. The query (or cue) is a partial image, say a horizontal band. The position of this band can change, but for each query it is known, i.e. we do not have to estimate this position. Our goal is to find the image

in the database of which the query is a sub-image. Figure 5.1 shows a schematic diagram.

Let us first assume that all the images in memory are normalized so that the image sizes are the same for all the images in memory. Thus we deal with $m \times n$ dimensional images. The set of all possible $m \times n$ images constitute a mn dimensional vector space. Let us consider the subset of this space which consists of the images actually stored in the database. We denote this subset by $\mathcal{M}$. Our input is a partially occluded image. It defines an (axially aligned) affine subspace of $\mathcal{M}$ on which all completions of the input image lie. We will denote this coordinate subspace by $\mathcal{I}$. Of course, in the presence of noise, the completions will not lie exactly on the subspace $\mathcal{I}$ but, rather, will be close to it. Our goal is to find the nearest match between $\mathcal{M}$ and $\mathcal{I}$, i.e., the completion of the query that best matches an item in memory.

Another Example

In a later chapter we will consider the example of a memory system storing English words. In that example one wishes to find a word to match the input query, say "_e_o__it___". Here a "_" indicates an unknown character. Thus, we consider cases where the positions of the unknown characters are known. We can think of the input space, $\mathcal{I}$, as the space of all possible strings that are consistent with this query. $\mathcal{M}$ denotes the set of all strings in memory.

5.2.1 Abstract Statement of the Associative Memory Problem

We now state the problem of associative memory in terms of the two spaces $\mathcal{M}$ and $\mathcal{I}$. In the most general case, $\mathcal{M}$ denotes the set of items in the memory. This set might have some other additional structure which we might exploit in specific cases. $\mathcal{I}$ denotes the set of possible completions of the given input. Thus $\mathcal{I}$ depends on the specific input given to the associative memory.

Our task is to find the element in $\mathcal{M}$ which is closest to $\mathcal{I}$ in the set theoretic sense. Thus, we consider $\mathcal{M}$ and $\mathcal{I}$ as subsets of some metric space with distance function d and define distances between a point p and a point-set S as

$$d(p, S) = \inf_{q \in S} d(p, q) \tag{5.1}$$

Thus, in the associative memory problem, we seek the element $m \in \mathcal{M}$ such that $d(m, \mathcal{I})$ is minimized:

$$\begin{aligned} m &= \operatorname*{argmin}_{m \in \mathcal{M}} d(m, \mathcal{I}) && (5.2) \\ &= \operatorname*{argmin}_{m \in \mathcal{M}} \inf_{i \in \mathcal{I}} d(m, i) && (5.3) \end{aligned}$$

Clearly, we need to exploit whatever additional structure is available to us. In the case of face recognition, we use the fact that the elements in the memory as well as the input cue lie in a linear Euclidean space, and we can use all the machinery of such spaces. Thus we can do, in particular, principle component analysis.

In the case of word memories, which we consider in the next chapter, our additional structure consists of the monoid structure of word concatenation.

5.2.2 A Slight Generalization: The Probabilistic Setting

Under certain situations, the above setup is inadequate. In many cases, where the memory elements do not have too much internal structure, it is sometimes advisable to add more structure to the space $\mathcal{M}$ in order to obtain better algorithms. One such desirable extra structure is convexity. Let us explain how we can implement this.

Let us suppose that the set of items we wish to store is

$$\mathcal{M}_1 \;=\; \{m_1, m_2, \ldots, m_n\}$$

We choose the space $\mathcal{M}$ to be the space of probability distributions over the set $\mathcal{M}_1$, whose elements $\hat{m}$ are assignments of probability values for each m_i :

$$\mathcal{M} \;=\; \{(p_1, p_2, \ldots, p_n) \,|\, p_1 + p_2 + \cdots + p_n = 1, \text{ and } p_i \geq 0 \text{ for all } i.\}$$

We still have to make some kind of choice for a distance function between elements of $\mathcal{I}$ and elements of $\mathcal{M}$, i.e. probability distributions over $\mathcal{M}_1$. This can be accomplished in many cases. At the very least we can consider the distance function $\hat{d}(\hat{m}, I) = \sum_j p_j d(m_j, I)$. This has the advantage that the result of the above minimization using this distance function actually leads to an element $m_k \in \mathcal{M}_1$ which minimizes the original distance function, but an approximate minimization can sometimes make sense. We will use such a probabilistic memory when we consider the example of face memories.

Convexity of the Input Space $\mathcal{I}$

We can use a similar trick to make the input space convex as well. Thus we define $\mathcal{I}$ to be the set of probability distributions on some set M depending on the input cue. One possible choice for M would be the items in memory which complete the input cue. However, under certain situations, computing this set is difficult. Fortunately however, we usually have some property P on a larger set that we can verify very easily. Thus, for the memory of English words, we can easily verify that a certain string of letters has the correct length and matches the input cue; however, verifying it is a valid word is more difficult. Under these circumstances, we can choose M to be the set of structures satisfying the property P.

5.3 Practical Considerations: Incorporating Domain Specific Knowledge

As equations 5.2 and 5.3 show, we can consider the problem of associative memory to be an instance of the problem of finding nearest neighbors. If the sets $\mathcal{I}$ and $\mathcal{M}$ associated in this procedure are convex then there is an iterative algorithm that is guaranteed to find the nearest neighbors. This algorithm proceeds by repeatedly minimizing the distance function. This leads to an algorithm for finding the distance between the two sets $\mathcal{M}$ and $\mathcal{I}$.

In this formulation, the memory space $\mathcal{M}$ is the one defined by the elements in the memory, whereas the space $\mathcal{I}$ is defined by the specific input cue presented. In our setup, we always treat the $\mathcal{M}$ as a constant space. This is because, we will run

our *learning algorithm* after the memory elements are given. The output of this process is a machine (or algorithm) which takes in cues and generates some $m \in \mathcal{M}$ which is a "good" completion of the cue.

However, iterative algorithm for computing distances cannot be used if the spaces involved are not convex. We can remedy this drawback by choosing convex sets for $\mathcal{M}$ and $\mathcal{I}$, as indicated in the previous section. However, there is still no guarantee about the speed of convergence. Indeed, the iterative method of finding the nearest neighbors is usually very slow, unless it is speeded up by means of domain dependent tricks. As we would like our memory look-up task to be quick, this is a big problem.

Furthermore, we cannot improve our lookup speeds unless we consider domain dependent characteristics. The spaces involved in this formulation always have fairly large dimensions, at least for applications in computer vision. Thus we look for fast algorithms for the nearest neighbor problem in high dimensional spaces. Among the best performing algorithms is an ϵ-approximate (probabilistic) algorithm due to Kleinberg ([74]) which provides a query time of $O(n + d \log^3 n)$ where n is the number of items and d is the dimension of the space. This algorithm needs a pre-processing time of approximately $O(d^2 n)$ and storage requirements of approximately $O(dn)$. Even these fairly reasonable complexity bounds could be very slow when both n and d are large. Below, we describe an architecture which lets us capture the domain dependent information in a uniform way.

Many of the speed improvements are obtained by precomputing whatever we can. However, the space $\mathcal{I}$ is itself determined by the input. For this reason, we

cannot precompute it. This makes our algorithm slow in situations where there is a lot of heterogeneity in the set of memory elements as well as in the set of possible input queries.

5.4 Nearest Neighbor Search in Feature Space

In order to obtain a fast and accurate lookup algorithm without too expensive a pre-processing step, we incorporate features in our model. So far, our model had used the two sets $\mathcal{M}$ and $\mathcal{I}$, and so, was similar in spirit to a two level (neural) network. We now imagine there is a third hidden layer between these two. This is the set $\mathcal{F}$ which denote the set of features peculiar to the elements in the memory.

We will present a framework for performing memory tasks in which these features are used to mediate the matching process between items in memory and an input query. Features therefore allow approximate matching while requiring much less processing, and much less communication between potentially complex inputs and potentially huge memories.

To do this efficiently, we demand that the feature space description of any particular element in the database be simple enough; and, as in chapter 3, capture sufficient information about the possible values of the database elements from an input query. Furthermore, the features should be close enough to the input space so as to be able to distinguish the possible inputs.

We first present our generic algorithm to do associative memory recall using features. This will make clear what requirements we demand of our feature space $\mathcal{F}$. The algorithm itself is novel, but combines ideas from many known algorithms

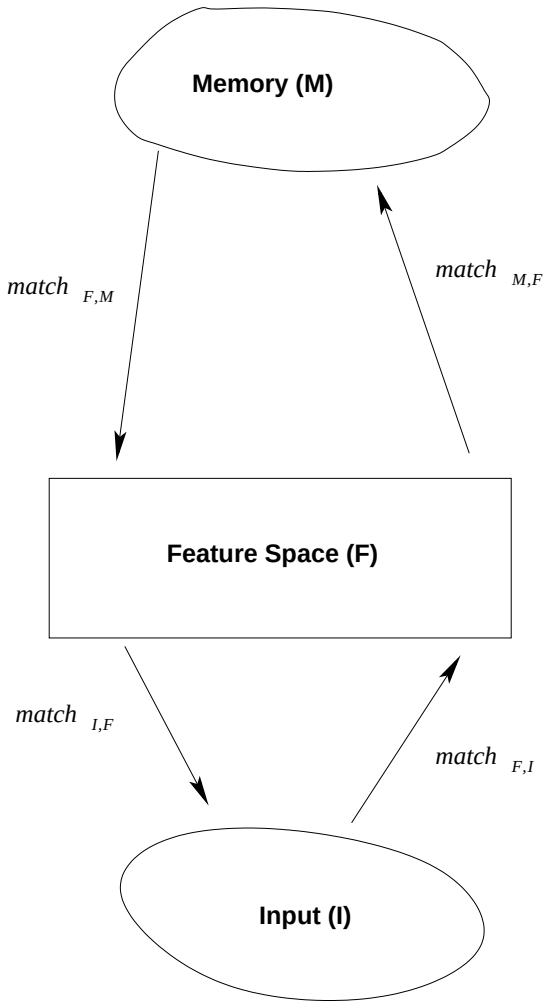

Figure 5.2: Relevant spaces and matching functions between them.

in an integrated fashion.

We use the matching functions: $match_{\mathcal{I},\mathcal{F}}, match_{\mathcal{F},\mathcal{I}}, match_{\mathcal{F},\mathcal{M}}, match_{\mathcal{M},\mathcal{F}}$. $match_{A,B}$ takes an element in B and generates the element in A that "best matches" it. The notion of a "best match" will depend on which pair of spaces we are talking about.

After some domain-specific initialization, we apply the iteration:

$$f' \leftarrow match_{\mathcal{F},\mathcal{I}}(c), \quad c \in \mathcal{I} \tag{5.4}$$

$$m \leftarrow match_{\mathcal{M},\mathcal{F}}(f') \tag{5.5}$$

$$f'' \leftarrow match_{\mathcal{F},\mathcal{M}}(m) \tag{5.6}$$

$$c \leftarrow match_{\mathcal{I},\mathcal{F}}(f'') \tag{5.7}$$

This algorithm can be seen as an approximation to a standard iterative approach to finding the nearest neighbors between two sets. Its success depends on the feature space, $\mathcal{F}$.

Explanation

Figure 5.2 shows a schematic diagram of one iterative step. Let us discuss what happens in various situations to get a feeling for why this iterative scheme might be useful.

In the first place, if the feature space $\mathcal{F}$ (denoted by the rectangle in the figure) is identical with the memory $\mathcal{M}$, we obtain the algorithm for distance computation by repeated minimization. We note that this algorithm can get trapped in local minima if the sets $\mathcal{M}$ and $\mathcal{I}$ are not convex. Now let us imagine that the space is

slowly changed from the initial value $\mathcal{I}$. Thus, the elements of $\mathcal{M}$ are not too far from $\mathcal{F}$. Further, assume that the projection of $\mathcal{M}$ into $\mathcal{F}$ is contained in a convex region. Under this situation, we can picture the role of $\mathcal{F}$ in the following way:

1. The matching function from $\mathcal{I}$ to $\mathcal{F}$ produces an element which is almost a member of $\mathcal{M}$.

2. At this point the matching function to go to $\mathcal{M}$ and back to $\mathcal{F}$ produces a perturbation, that acts as a noise to prevent the iteration from getting captured in a local minimum.

3. Thus, this iterative scheme generates nice results without too many iterations.

Choice of the Feature Space $\mathcal{F}$

The choice of the feature space is driven by two (sometimes conflicting) considerations:

1. For efficiency it must be much simpler to compare $\mathcal{F}$ to $\mathcal{I}$ and $\mathcal{M}$ than it is to compare $\mathcal{I}$ and $\mathcal{M}$ directly.

2. For accuracy, the feature space must be able to approximate $\mathcal{M}$ and $\mathcal{I}$ reasonably well.

Notice that the feature space $\mathcal{F}$ does not depend on the input. [3] Thus, we can precompute the matching functions between $\mathcal{F}$ and $\mathcal{M}$. This will lead to enormous efficiencies in the case of face image databases, as we will see.

[3] Though the particular features in an input will of course be dependent on the exact input query.

5.5 Summary

At this point, we have a novel and simple high level algorithm which we claim lets us easily incorporate features in a nearest neighbor algorithm to solve the problem of associative memory. In this thesis we do not give a formal proof of this claim. However, we back up this claim by using this architecture to design associative memories in various problem domains. In chapter 6 we use linear features to obtain an associative memory for faces. In chapter 7 we look at word memories. We instantiate our algorithm and create a generalization of the well known Baum Welch iterative algorithm. We also look at human performance in this area, and explore various theories in the light of these experiments.

In all these constructions of associative memory, we will start with the skeletal algorithm formed by blindly instantiating the parameters. After that we will apply domain-specific optimizations to this basic algorithm.

Chapter 6

Linear Feature Spaces: Associative Memory of Faces

We now apply the approach (described in chapter 5) to recognize partially occluded images of faces. Human beings learn to recognize faces over a year in early childhood. This, together with an ability to recognize facial expressions, play a vital role in our social (and political) life. The success of our approach depends critically on the proper choice of the feature space (and the matching functions). Thus, we first review the kind sof approaches people have used in this problem, and what this reveals about the statistical structure of the problem.

6.1 Prior Work on Face Recognition

There are a few distinct notions that reappear in in various guises in most works involving face recognition.

Early approaches to automatic recognition of faces in images involved searching for various specialized geometrical shapes, such as eyes, nose, mouth etc. These approaches differ in the way this information are used.

6.1.1 Feature-Based Approaches

A face may be recognized[1] even when the internal details of the individual features are unknown. This usually leads to a decrease in the dimension of the input data. The hope is that the remaining information is information about local geometry at a very coarse level. The success of this approach depends on accurate extraction of the features. See Goldstein ([49]), Kaya ([71]), Bledsoe ([25]) and Kanade ([68]) for examples of this approach.

6.1.2 Template Matching Based Approach

In its original form, template matching proceeds by matching the image (represented as a 2-dimensional matrix) to a single template for the whole face using some metric. More sophisticated techniques are usually needed. We can do template matching after preprocessing the images. We can also use multiple templates to take into account multiple viewpoints etc. We can even use multiple templates for each viewpoint each covering a smaller area of the face. See for example Baron ([11]) and Bischel ([23])

[1] i.e., the presence of a face may be detected

Deformable Templates

A more complex approach is to use a deformable template, where we allow the template to be deformed before matching; but penalizing the deformation by putting a cost to it. This is equivalent to having a prior model on the possible deformations that can occur. Burr ([29]), Buhmann ([28]), and Yuille ([123]) all describe algorithms which can be called *deformable template matching.*

For a comparative study of these approaches see Brunelli and Poggio ([27]).

6.1.3 Linear Approaches to Face Recognition

Given a quadratic cost function, the optimal linear method for reducing redundancy of different factors is the *Principal Component Analysis*(PCA). In computer vision literature, this is also known as the Karhunen-Loéve transform. In practice, the method involves computing the eigenvectors of the correlation matrix. For this reason we group such "best linear" approaches together as eigenvector approaches. This class of approaches actually constitute a subclass of template matching, since here we end up matching templates in a transformed space.

We can use a intensity-based approach to obtain data for an eigenvector approach. Thus we compute quantities from the actual intensity values and use these in recognition. One approach is to seek representations of images in terms of some smaller basis set. Image bases obtained using PCA have been used in face recognition tasks. Turk and Pentland ([115]) as well as Kirby and Sirovich ([72]) independently advocated this approach.

There have been correlation based approaches to image recognition as well. Most

of these need some kind of dimension reduction to work in a realistic situation, and the PCA decomposition can be seen as one such technique.

Low Level Representations not Depending on Intensity

Intensity values represent only one way of representing images. Other representations have been considered in the literature as well. These representations tend to extract as much local information as possible, leaving the higher level algorithms the task of extracting longer range dependences.

Edelman et al ([38]) considered zeroth-order Gaussian kernels; whereas Buhmann ([28]) used a representation where the images were captured using the responses of various Gabor filters at every point (the so called "jets").

6.1.4 Higher Level Processing

The original approach used in Turk and Pentland's paper ([115]) uses a PCA decomposition of the set of images in memory, and thus this space changes when new items are added to the memory or when old items are removed. Some other representations work for large classes of natural as well as artificial images. Rao and Ballard ([100]) for example, used derivatives of Gaussian filters at various scales as their basis elements . Other approaches involve computing the PCA locally at various scales or views and then combining the different results to get the final answer; instead of forming an omnibus "face space". Pentland, Mohaddam and Starner gave such an approach in [93].

A slightly different approach advocates using a *Linear Discriminant Analysis*

approach to form what are known as *fisherfaces* (instead of *eigenfaces*) in our face recognition task. Linear discriminant analysis leads to optimal linear classifiers to separate two sets. Belhumeur et al ([18]) uses this approach to distinguish between different face images.

In an example illustrating how these different approaches fit together, Buhmann ([28]) represented images using the aforementioned "Gabor jets" and used graph matching as a higher level algorithm to merge the local information.

6.2 Linear Features for Image Retrieval

Following the approach of using linear algorithms for face recognition, we now describe a memory for images in which the feature space, $\mathcal{F}$, is a linear subspace of the space of all possible images. We assume that a set of n images has been stored in memory, so that each $m \in \mathcal{M}$ is an image, thought of as a vector in a d-dimensional space. The query, c is a partly occluded image. We assume we know the locus of the occlusion, so c is a d-dimensional vector in which some of the coordinates are unknown.

We assume that the memory elements come from some multivariate Gaussian distribution concentrated near a small subspace in the space of all possible images. This is the case, for example, for faces. See, for example Turk and Pentland ([115]) and Sanger ([104]).

The parameters defining this subspace can be either known from prior work, or can be found by sampling from the distribution. In any case, we assume that this is known in advance before the recall task is attempted.

Under this assumption, it makes sense to use this smaller subspace as the feature space $\mathcal{F}$. As explained above, we can either use precomputed parameter values for $\mathcal{F}$ or compute it by performing *Principal Component Analysis* (PCA) on $\mathcal{M}$, choosing only the first p principal components. PCA has been widely used to select features spaces in the past . Here, however, we use the space produced by PCA as part of our iterative scheme for associative memory.

We use the iterative algorithm of Section 5.4. Since the cue, c has some fixed coordinates and some coordinates that can take on any values, the images consistent with c form an affine subspace[2]of the space of all possible images. This is the space $\mathcal{I}$.

If c matches an item in memory exactly, then $\mathcal{I}$ and $\mathcal{M}$ intersect. If not, the element of $\mathcal{M}$ closest to $\mathcal{I}$ is the image in memory that best matches c.

We choose $match_{\mathcal{F},\mathcal{I}}, match_{\mathcal{F},\mathcal{M}}$ and $match_{\mathcal{I},\mathcal{F}}$ to be the orthogonal projection to the respective spaces, $\mathcal{F}$ and $\mathcal{I}$. $match_{\mathcal{M},\mathcal{F}}$ takes an element $f \in \mathcal{F}$ and outputs the element $m \in \mathcal{M}$ nearest f. This leads to the following iterative loop:

{**Input:** $C \in \mathcal{I}$}

{Denote the Orthogonal Projection to an affine subspace S by Π_S}

loop

$F_1 \leftarrow \Pi_{\mathcal{F}}\, C$

$M \leftarrow \text{argmin}_{M' \in \mathcal{M}}\, d(F_1, M')$

$F_2 \leftarrow \Pi_{\mathcal{F}}\, M$

$C \leftarrow \Pi_{\mathcal{I}}\, F_2$

[2]An affine subspace is a regular subspace of a vector space, only translated by some vector.

end loop

6.2.1 Eigenfaces versus Iterative Approach

Before we discuss in detail the various optimizations we can perform on this skeletal algorithm, let us explain what our iterative scheme buys us over a regular eigenface-like approach. Essentially, the iterative schemes scores in terms of robustness to occlusion. In other words, the eigenface approaches start losing accuracy when more and more of the face is occluded and, thus more features are hidden. This can be corrected for to a certain extent, but these are very ad hoc methods which depends on innate knowledge of relative position of features. Our iterative approach performs well even when a large fraction of the features are occluded.

6.2.2 Offline Computations

In actual computation, we exploit the fact that many of the relevant spaces are in fact known before we know the input. In particular, we can compute the feature space $\mathcal{F}$ and the matrix for the orthogonal projection operator $\Pi_{\mathcal{F}}$.

If we choose to calculate the space $\mathcal{F}$ using PCA on the possible memory elements, then we have at our disposal one parameter to control the capacity of the intermediate feature space: the dimension of $\mathcal{F}$, or, equivalently, the number of principal components we keep. Let us call this number p.

Let d be the dimension of the space of the images. If our images are $r \times s$ in dimension, then $d = rs$. We choose p to be significantly less than d. For specific classes of images we can actually do this without sacrificing performance.

The only other projection involved in our algorithm is the projection $\Pi_{\mathcal{I}}$ onto the image space $\mathcal{I}$. We can also calculate this projection once we know the input query. It can, in fact, be described very simply. Let us suppose that we are trying to calculate the projection of the image M. Let us also suppose that the original input query was the partial image C, which was defined at pixels $p_1, p_2, \ldots, p_s$. Let the value of C at pixel p_i be C_{p_i}. Then, we can compute the image of M by assigning the value C_{p_i} to the pixel location p_i. We show this in algorithmic terms:

{**Computation of the Orthogonal Projection** $\Pi_{\mathcal{I}}$}

{**Input:** Image M, not necessarily one of the memory elements}

{**Input:** Input query C, which is a partial image}

{Let $p_1, p_2, \ldots, p_s$ be the pixel locations which are present in C}

{**Output:** $I \in \mathcal{I}$ which is the image of M under the orthogonal projection $\Pi_{\mathcal{I}}$}

$I \quad \leftarrow \quad M$

$I(p_1, p_2, \ldots, p_s) \quad \leftarrow \quad C$

Thus, we can easily implement this projection using a vector assignment.

6.2.3 Doing Nearest Neighbor Search in Feature Space

We can vastly improve our iterative search if we concentrate on the matching process we have not considered so far: $match_{\mathcal{M},\mathcal{F}}$.

This matching function is actually a nearest neighbor search. Thus, for $F \in \mathcal{F}$,

$$match_{\mathcal{M},\mathcal{F}}(F) = \underset{M' \in \mathcal{M}}{\operatorname{argmin}}\, d(F, M') \tag{6.1}$$

A naive calculation of this step would proceed by calculating the distance of each memory element M from F, and then picking the minimum.

Calculation of each distance takes a time proportional to the dimension d. Thus the naive computation of the projection $match_{\mathcal{M},\mathcal{F}}$ takes $\Theta(nd)$ time.

We can do better than this. Notice that F is a point on the feature space $\mathcal{F}$. Thus the square of the distance between it and any memory element M is the sum of two parts:

1. the square of the distance between F and $\Pi_{\mathcal{F}}(M)$, the foot of the perpendicular from M to the subspace $\mathcal{F}$; and

2. the square of the distance between M and $\Pi_{\mathcal{F}}(M)$.

Since the second part is a constant independent of F, we can precompute these n numbers, one for each memory element in $\mathcal{M}$. Similarly, we can also precompute the respective feet of perpendiculars: $\Pi_{\mathcal{F}}(M)$.

Then, for each F, we just need to compute the first part, which takes $\Theta(p)$ time; and thus the complete matching function can be computed in $\Theta(np)$ time. Since p can be hundreds of times smaller than d, this is a sizable saving in time.

We thus precompute the distances of each memory element from the feature space $\mathcal{F}$ and record the coordinates of the corresponding feet of the perpendicular in the coordinate system of that space.

Let us compute the complexity of each iteration. The projection operations: $match_{\mathcal{I},\mathcal{F}}$ and $match_{\mathcal{F},\mathcal{I}}$ (steps 5.7 and 5.4 in the generic iteration scheme), consisting of projections onto known spaces, can be done in $\Theta(dp)$ steps per iteration.

The argmin operation $match_{\mathcal{M},\mathcal{F}}$ can be done $\Theta(np)$ time. The last step $match_{\mathcal{F},\mathcal{M}}$ is another projection and takes $\Theta(dp)$ time.

Thus, our total time per iteration is $\Theta((n+d)p)$.

6.3 Further Algorithmic Improvements

We can get further improvements in the complexity, if we work completely in the feature space. Thus, much of the complexity derives from having matrices which has d rows or columns, or, equivalently, the high dimension of the space of images.

If our feature space is of sufficiently small dimension, we can speed up each iteration by a large factor.

6.3.1 Eliminating the Image Space $\mathcal{I}$

We can combine the projection to and from the input space $\mathcal{I}$ to compute $match_{\mathcal{F},\mathcal{I}} \circ match_{\mathcal{I},\mathcal{F}}$. This is easily seen to be a transformation of the feature space $\mathcal{F}$. We will see below that it is a fixed affine transformation T_a of $\mathcal{F}$; depending on the input query. Thus, we can calculate it in the *Principal Component* coordinate system, to generate a small matrix, and reduce the time complexity of this part of the iteration to $\Theta(p^2)$ down from $\Theta(dp)$.

Overall, if the algorithm performs k iterations it requires $O(dp^2 + kp(n+p))$ time, which can be much less than the $O(nd)$ time required for brute-force matching when the number of images and number of their pixels is large. Ordinarily, p is in the order of 10's and p is usually less than 5.

Let us now calculate the composition of the two projections $match_{\mathcal{I},\mathcal{F}}$ ($\Pi_{\mathcal{I}}$) and

$match_{\mathcal{F},\mathcal{I}}$ ($\Pi_{\mathcal{F}}$). We will first state the result as a theorem, and the proof of the theorem will give the explicit form of the composition.

Theorem 6.3.1 *The composition* $match_{\mathcal{F},\mathcal{I}} \circ match_{\mathcal{I},\mathcal{F}}$ *is an affine transformation of the feature space* $\mathcal{F}$*; i.e. it is of the form* $T_a\alpha = A\alpha + b$ *for* $\alpha \in \mathcal{F}$ *and* A *is a* $p \times p$ *matrix and* b *is a* p *dimensional vector, where* $\dim \mathcal{F} = p$.

Proof

Let $F_1, F_2, \ldots, F_p$ be the principal components generating the feature space $\mathcal{F}$. They form an orthonormal basis for $\mathcal{F}$. Let $F_{i,j}$ be the i-th component of F_j. From the dimensions of the spaces involved, we see that F forms a $d \times p$ matrix. Since the F-s form a basis for $\mathcal{F}$, any vector of $\mathcal{F}$ can be written as a linear combination $\sum_j \alpha_j F_j$. Thus, we can represent any vector of $\mathcal{F}$ by a p dimensional column vector of the α-s. We will calculate the effect of applying the composition $match_{\mathcal{F},\mathcal{I}} \circ match_{\mathcal{I},\mathcal{F}}$ on such a column vector.

Let C be the partial query image which is used to form the image space $\mathcal{I}$. We can assume without the loss of generality that the first r components of C are known. Then a projection into $\mathcal{I}$ simply overwrites these r components with the corresponding coordinate value from C.

Thus, a vector

$$\alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{pmatrix} \quad \in \mathcal{F}$$

corresponds to the vector $\sum_j \alpha_j F_j$ and therefore its projection to $\mathcal{I}$ is the vector

$$I = \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_r \\ \sum_j \alpha_j F_{(r+1),j} \\ \sum_j \alpha_j F_{(r+2),j} \\ \vdots \\ \sum_j \alpha_j F_{d,j} \end{pmatrix} \in \mathcal{I}$$

We just need to find the projection of I into $\mathcal{F}$ to find the image of α under the composition of the two projections. This is easy. Since the F-s form an orthonormal basis, to find the coefficients of the projection, we just need to calculate the inner products of I with the various F-s. Thus, if α' is the projection expressed in the coordinate system formed by the F-s,

$$\begin{aligned} \alpha'_i &= C_1 F_{1,i} + C_2 F_{2,i} + \cdots + C_r F_{r,i} \\ &\quad + \left(\sum_j \alpha_j F_{(r+1),j} \right) F_{(r+1),i} + \left(\sum_j \alpha_j F_{(r+2),j} \right) F_{(r+2),i} \cdots \left(\sum_j \alpha_j F_{d,j} \right) F_{d,i} \end{aligned}$$

.

Rewriting this more compactly, we have

$$\begin{aligned}\alpha'_i &= \sum_{j=1}^{r} C_j F_{j,i} + \sum_{k=r+1}^{d} \left(\sum_{j=1}^{p} \alpha_j F_{k,j} \right) F_{k,i} \\ &= \sum_{j=1}^{r} C_j F_{j,i} + \sum_{j=i}^{p} \alpha_j \left(\sum_{k=r+1}^{d} F_{k,j} F_{k,i} \right)\end{aligned}$$

Thus, if we choose

$$(A)_{i,j} = \sum_{k=r+1}^{d} F_{k,j} F_{k,i}$$

and

$$(b)_i = \sum_{j=1}^{r} C_j F_{j,i}$$

then

$$\alpha' = A\alpha + b$$

where the last is a matrix equation.

■

We note that the matrix A depends only on the positions of the unknown pixels in the cue but not on the known values, whereas b depends on the actual values. Both the expressions for A and b look like truncated inner products. Looking at the expression for A, we can see that its computation takes time $O(dp^2)$, because the summation takes time $O(d)$ and there are p^2 elements.

6.4 **A Probabilistic** argmin

At this point, we have eliminated the image space completely; except from the initialization steps. This let us cut down our time complexity by a factor of d/p. If we can compose the other two matching functions: One taking us from $\mathcal{F}$ to $\mathcal{M}$ and the other from $\mathcal{M}$ to $\mathcal{F}$ then we can do the entire iteration in the feature space with a concomitant speed-up.

This is actually easy. From what has been already described, we can calculate the distance of any memory element from a point on $\mathcal{F}$ using the feature space coordinate system with an additional number equal to the distance of the memory element from the feature space. If, further, we store the projections $\Pi_{\mathcal{F}} M$ of elements $M \in \mathcal{M}$ in the feature space coordinate system; then we can use this vector for the minimum distance memory element to get the image of $F \in \mathcal{F}$ under the composition $match_{\mathcal{M},\mathcal{F}} \circ match_{\mathcal{F},\mathcal{M}}$.

However, there is a problem with the quality of the solution generated by this algorithm under certain conditions.When the images in $\mathcal{M}$ form a sparse set it is possible for local minima to emerge in the iterative algorithm that are far from the global minima. To deal with this problem, we heuristically modify the matching function $match_{\mathcal{M},\mathcal{F}}$.

Thus, instead of finding the nearest memory element, we find a weighted average of the memory elements, where closer elements have bigger weights. In practice, we use a weight proportional to $\exp(-d_i^2/2\sigma^2)$, where d_i is the distance to the i'th memory element. The limit $\sigma \to 0$ corresponds to choosing the nearest neighbor. We actually decrease σ as the iteration proceeds. The intuitive justification for this

is that as the iteration proceeds, we are likely to be at the domain of attraction of the global minimum, and having a small σ speeds up the iteration.

Similar ideas motivate simulated annealing procedures in heuristic search. ([73, 33, 48])Thus, higher values of σ correspond to higher temperatures and the position of the current approximation in $\mathcal{F}$ can change very easily. As the iteration proceeds we decrease the temperature (σ) to 0 and the ease of change decreases as well.

Thus we have the algorithm for associative recall for images.

{**Input:** $C \in \mathcal{I}$}

{Denote the Orthogonal Projection to an affine subspace S by Π_S}

{Let A and b be matrices defined in the proof of Theorem 6.3.1}

{Let $M_1, M_2, \ldots, M_n$ be the elements in the memory $\mathcal{M}$}

$F \leftarrow$ Some initial point in $\mathcal{F}$ depending on C

loop

{Project to $\mathcal{I}$ and back}

$F \leftarrow AF + b$

{Probabilistic argmin and project back to $\mathcal{F}$}

{**WeightedMean** denotes the weighted mean with changing parameters described above.}

$F \leftarrow \Pi_{\mathcal{F}}$ (**WeightedMean**$(F;\ \ M_1, M_2, \ldots, M_n)$)

end loop

6.5 Choice of Initial Point for the Iteration

At this point, we still need to figure out how to start the iteration. The most obvious way would be to zero out the unknown pixels in the input query and use its projection on $\mathcal{F}$ as the starting point of the iteration. However, this does not lead to very good solutions, because it does not capture any information at all about the position in $\mathcal{I}$ where the projections of the memory elements lie. Alternately, we can start with a random position in $\mathcal{I}$. This suffers from the same defect as before.

We can try to capture more information about the memory elements by starting from the projection of the mean of the memory images. There are several alternatives here. We can just project the mean image onto $\mathcal{F}$ and use that as the starting point. Alternately, we can first project the mean image to $\mathcal{I}$ and then to $\mathcal{F}$.

However, we will describe a much better starting point which captures a lot of information about the final result. To understand the choice of the initial point, let us go back to the affine transformation T_a on $\mathcal{F}$ which is the composition of the two projections $match_{\mathcal{I},\mathcal{F}}$ and $match_{\mathcal{F},\mathcal{I}}$ according to Theorem 6.3.1.

We get a starting point for iteration which captures information about the rough position that the typical image occupies in $\mathcal{F}$. Our estimate does not capture information about the particular set of images in memory. Also, we try to capture some dependence on the initial cue.

The algorithm described in Section 6.4 does two operations on the current estimate in feature space $\mathcal{F}$:

1. Affine transformation T_a

2. Probabilistic argmin

If we discard the probabilistic part of this iteration, we are left with the repeated iteration

$$F \leftarrow AF + b \tag{6.2}$$

This terminates when the vector F stabilizes. We choose this final value of F as our initial point. Thus we solve the equation

$$F = AF + b \tag{6.3}$$

Simplifying, we get

$$F_{initial} = (I - A)^{-1} b \tag{6.4}$$

If $(I - A)$ is not invertible, we use some pseudo-inverse.

Notice that if our query had been a complete image, we would have got the projection of it as our starting point. More generally, we can state the following:

$F_{initial}$ is the projection to $\mathcal{F}$ of the completion of C that is closest to $\mathcal{F}$.

This follows easily if we consider that ignoring the memory $\mathcal{M}$ in the iterative algorithm leads to an iterative computation of the distance between $\mathcal{I}$ and $\mathcal{F}$.

Thus, if one considers $\mathcal{F}$ to cantain "face-like" images, then one is computing the most "face-like" completion of the query C, and computing this in the feature space co-ordinate system. This choice of initial point gives a good final image, and cuts down on the number of iterations needed for the iteration to converge.

We also make one final modification. Instead of successively applying the affine transformation (from Theorem 6.3.1) and the probabilistic argmin in each step of

the iteration, we take a convex combination of them; with the proportions of each changing as the iteration proceeds as in any simulated annealing algorithm.

Thus, our complete algorithm is:

{**Input:** $C \in \mathcal{I}$}

{Denote the Orthogonal Projection to an affine subspace S by Π_S}

{Let A and b be matrices defined in the proof of Theorem 6.3.1}

{Let $M_1, M_2, \ldots, M_n$ be the elements in the memory $\mathcal{M}$}

$F \leftarrow (I - A)^{-1} b$

loop

{Project to $\mathcal{I}$ and back}

$F' \leftarrow AF + b$

{Probabilistic argmin and project back to $\mathcal{F}$}

{**WeightedMean** denotes the weighted mean with changing parameters described above.}

$F \leftarrow \Pi_{\mathcal{F}}$ (**WeightedMean**$(F, F'; \quad M_1, M_2, \ldots, M_n)$)

end loop

6.6 Experimental Results

To test our iterative algorithm for identifying an item in the memory from partial information, we applied it to a small memory of 79 face images. These were frontal pictures with no change in viewpoint. and lighting. The input cue was obtained from a random memory element by taking a horizontal strip of it and corrupting it by noise.

6.6.1 The Images in Memory

To obviate issues regarding registration of images, we marked out some hand-picked positions on each image. These are points on the image on easily identifiable landmarks like the tops and sides of eye-brows, various positions around the nose and mouth etc.

Figure 6.5 shows a typical face with the landmarks. We store the registered faces in memory. The corresponding registered and normalized face is shown in figure 6.6. The registration and normalization is designed to make the dimensions of the images the same, and at the same geometric location.

The locations of the landmarks are such as to exclude the forehead and the head, and our registration and normalization process excludes the hairline and most of the forehead. Our algorithm is thus insensitive to hair-style changes.

6.6.2 The Input Query

We use an occluded image of the face as an input query. We use a strip with about 50% of the area as our input. We test our algorithm for erroneous recall, and also for the average number of iterations needed for convergence. We look at these results as the dimension of the feature space increase. The dimension of the feature space is a good measure of the complexity of the intermediate features.

We test for robustness by adding Gaussian noise to the input query. The noise is of standard deviation $\sigma = 50$, which is 20% of the full range of gray values. Figure 6.7 shows a sample query, without the added noise, and Figure 6.8 shows one with the added noise.

We tested our iterative algorithm on a set of 79 registered face images. We covered each face by half to get the initial queries, C. We checked that the face that was matched to C was correct. We did this for each choice of the dimension of $\mathcal{F}$, the feature space. We also measured the average number of iterations needed to converge.

6.6.3 Results

We essentially get error free recall starting from about a 15 dimensional feature space. The number of iterations are usually between 4 and 5, and the average over the all the images is less than 6 for all useful values of the parameter ($\dim \mathcal{F}$). We do have some tradeoffs regarding the allowable error rate, the dimension $\dim \mathcal{F}$, and the number of iterations. We can trade off the number of iterations against the error rate as well as against the dimension. We can also fine tune the error rate by using a different annealing schedule for the probabilistic argmin described in section 6.5.

The graphs in Figures 6.1 and 6.3 show the variation of the %-age errors with the dimension of $\mathcal{F}$, $\dim \mathcal{F}$; in the noiseless and the noisy case respectively. Similarly, the graphs in Figures 6.2 and 6.4 plot the number of iterations to convergence.

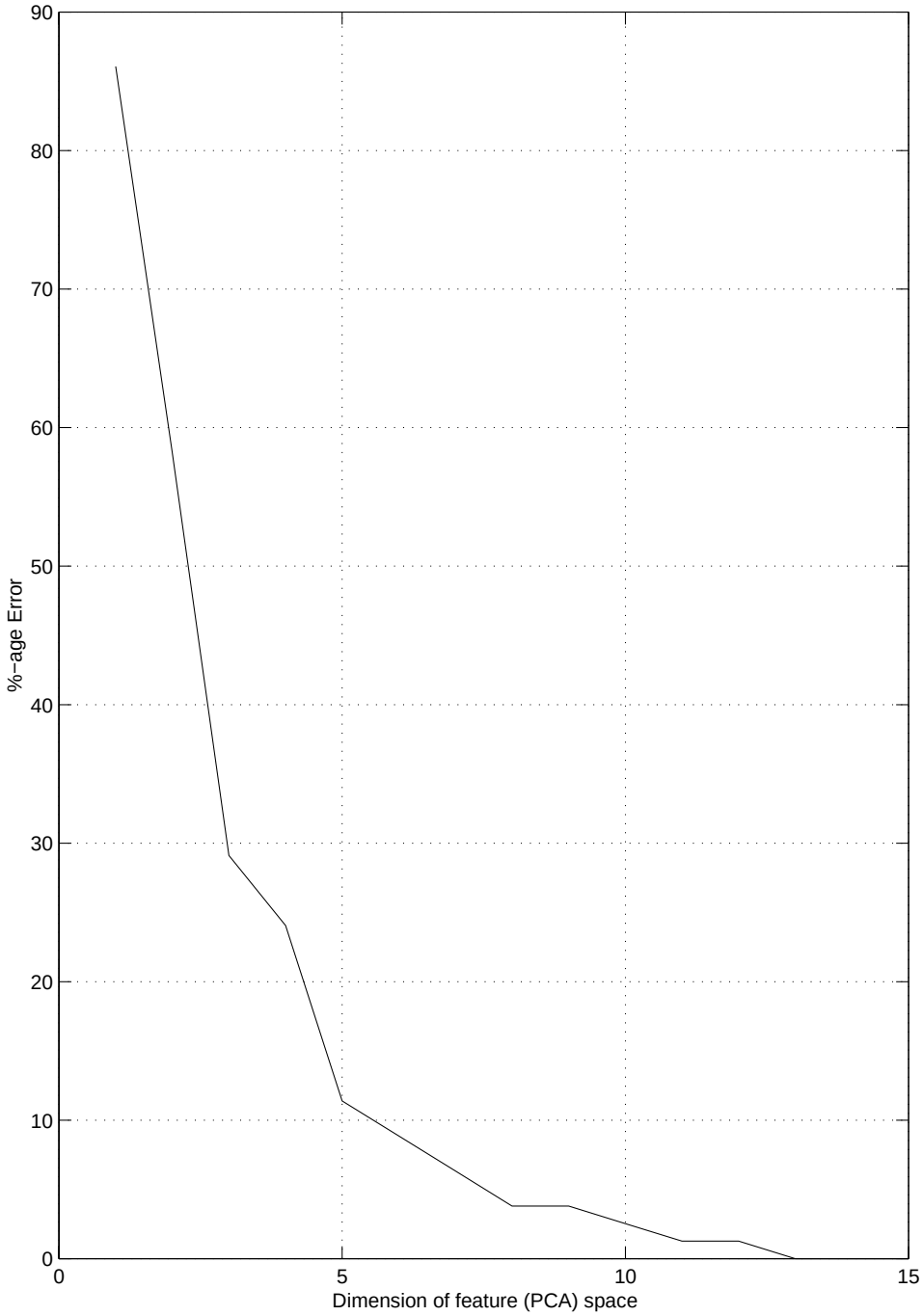

Figure 6.1: %-age errors against PCA dimension: Noiseless Case

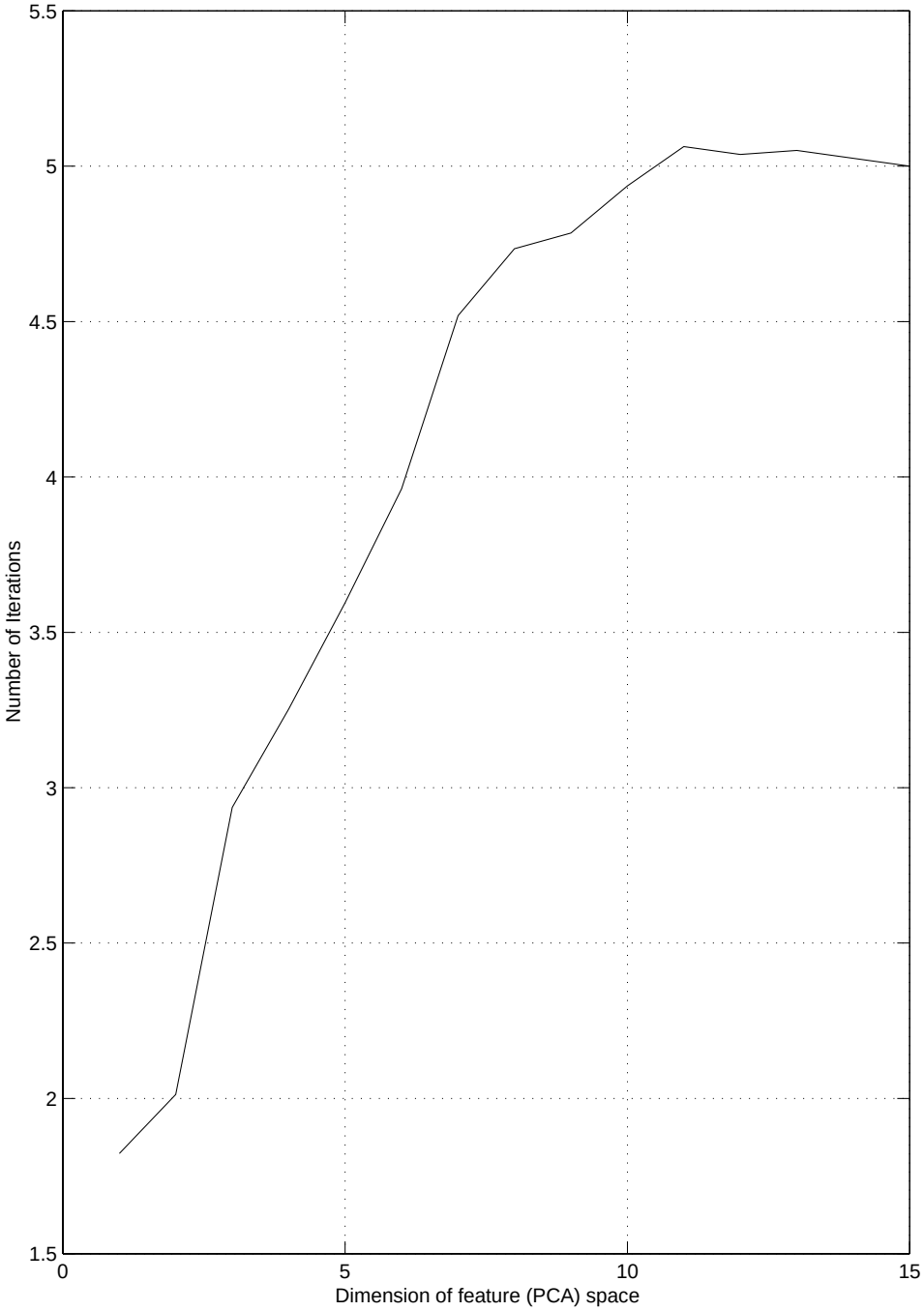

Figure 6.2: Number of iterations for different $\dim \mathcal{F}$: Noiseless Case

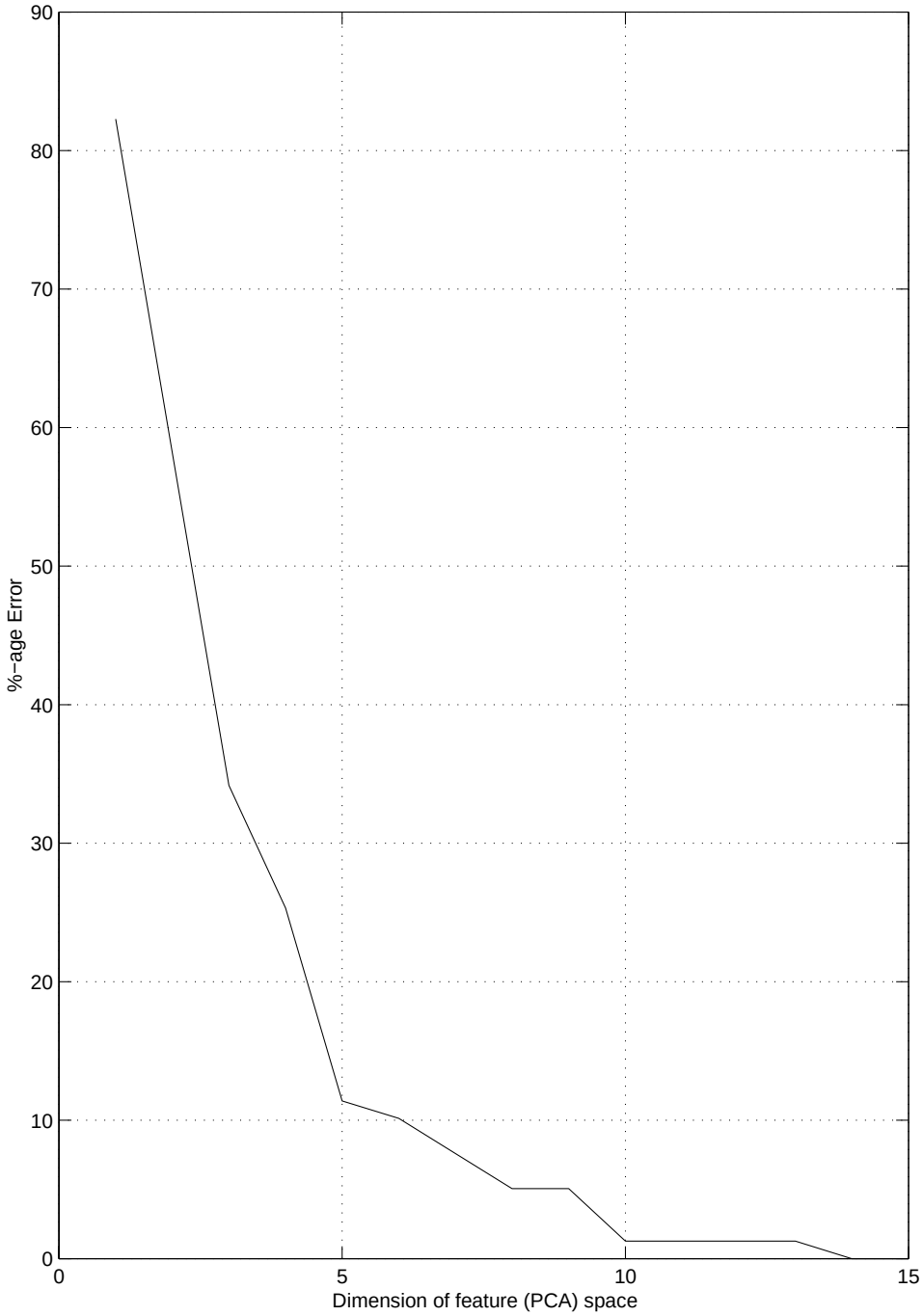

Figure 6.3: %-age errors against PCA dimension: Noisy Case, Gaussian noise, $\sigma = 50$

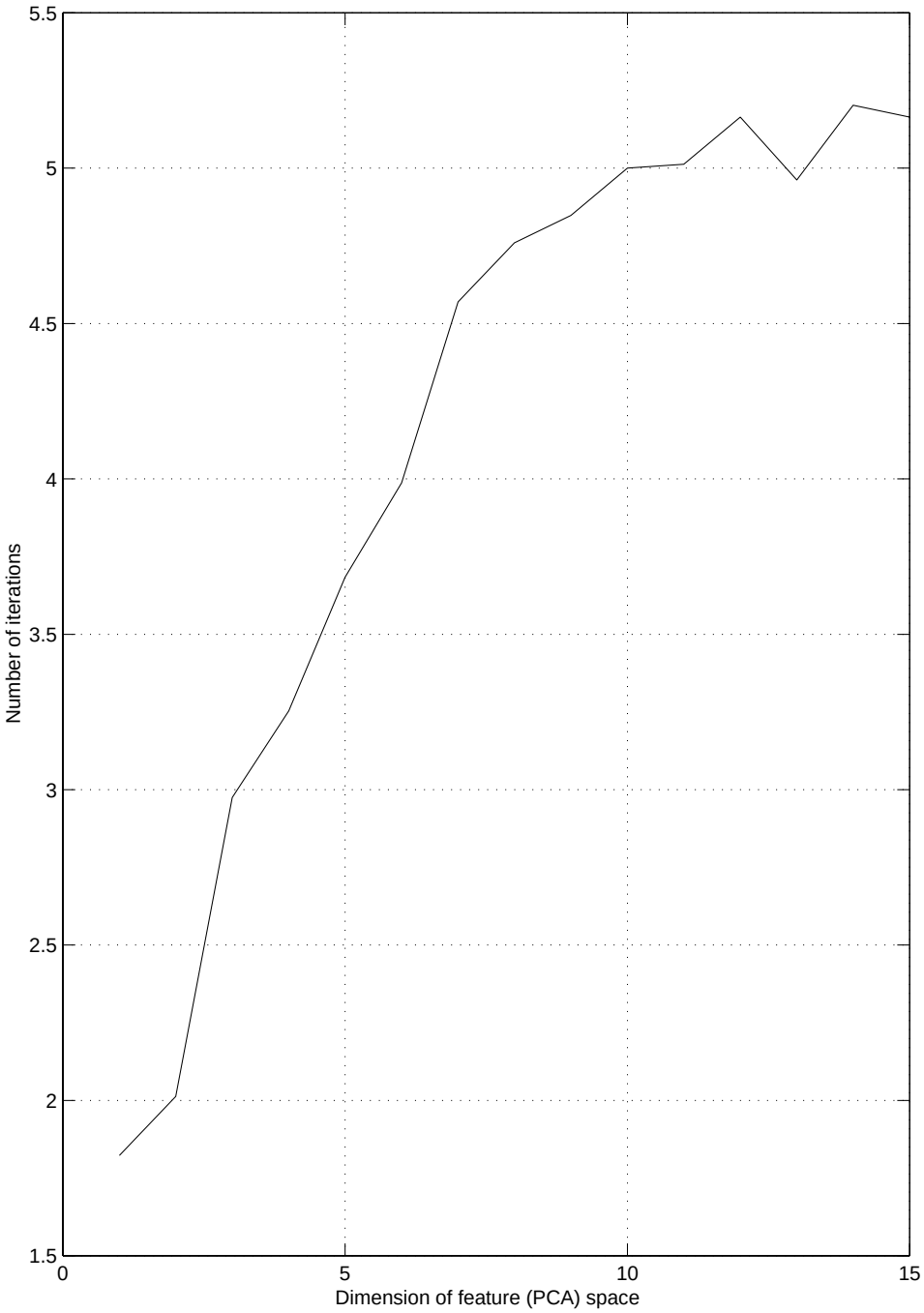

Figure 6.4: Number of iterations for different $\dim \mathcal{F}$: Noisy Case, Gaussian noise, $\sigma = 50$

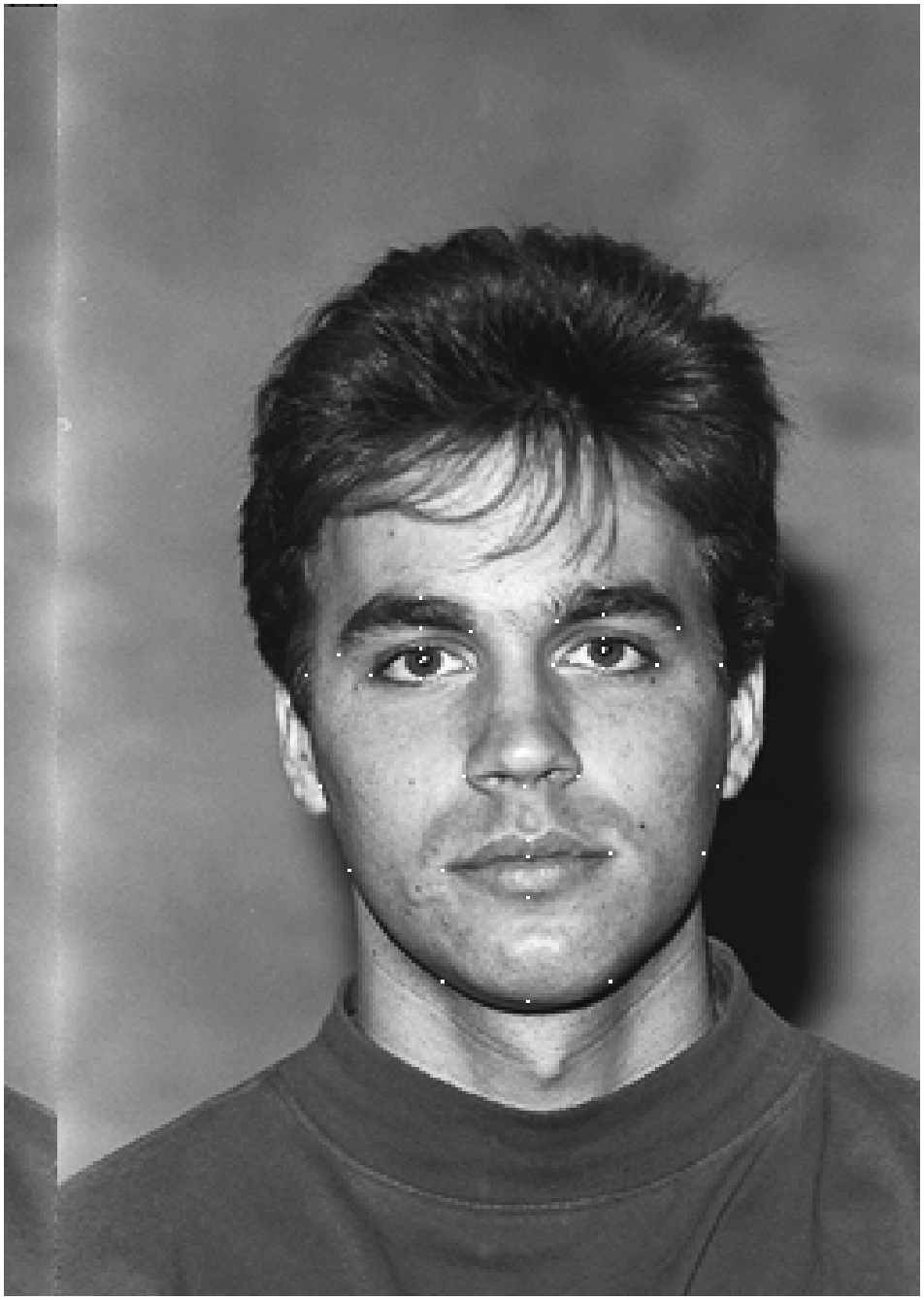

Figure 6.5: Sample face used in the experiment: full face with landmarks.

Figure 6.6: Sample face used in the experiment: registered part face stored in memory

Figure 6.7: Sample face used in the experiment: occluded face

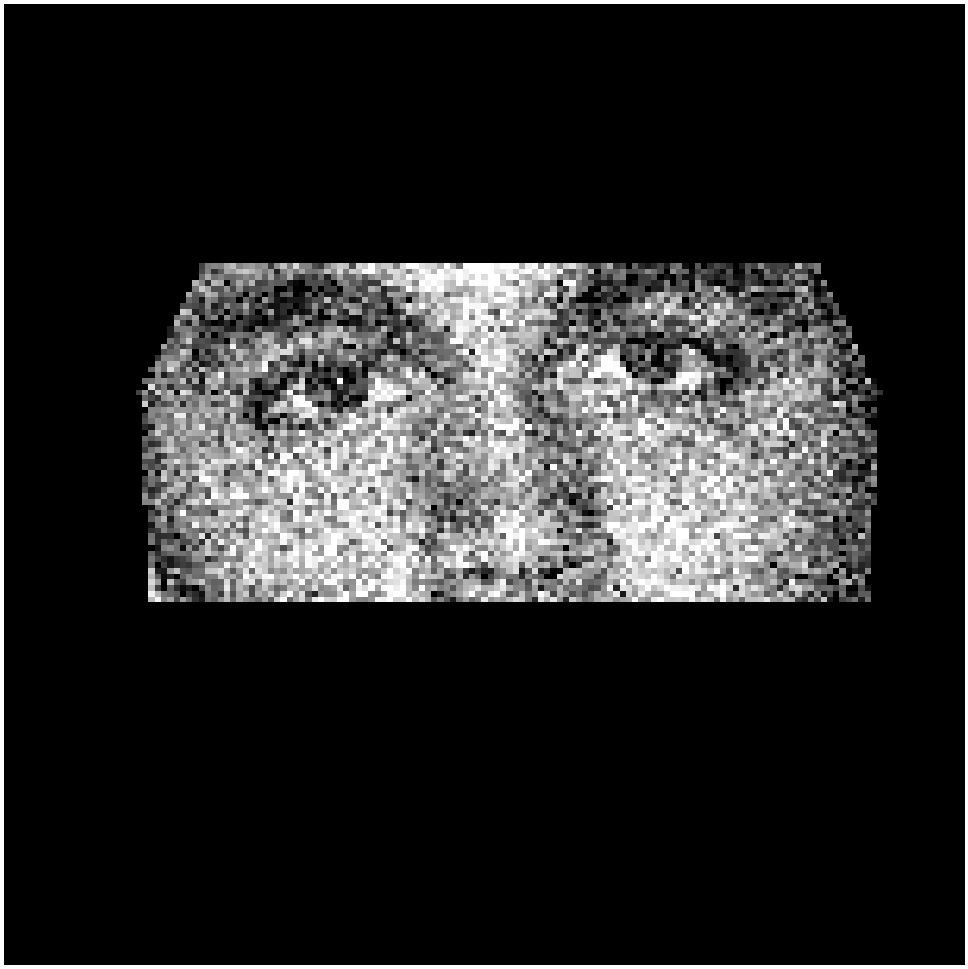

Figure 6.8: Sample face used in the experiment: occluded face with added Gaussian noise.

6.7 Discussion

A Simple Case

To get some intuitions about this algorithm, we analyze a simple case. Suppose the images in memory densely fill a convex subset of a low-dimensional space. In this case, $\mathcal{M} \subset \mathcal{F}$, and so the projection going from $\mathcal{M}$ to the feature space $\mathcal{F}$ of the iteration has no effect. The projection going from $\mathcal{F}$ to $\mathcal{I}$ and back projects a point from $\mathcal{M}$ first to the nearest point in $\mathcal{I}$, and then back to the nearest point in $\mathcal{M}$ (since $\mathcal{M}$ densely covers part of $\mathcal{F}$).

The algorithm, then, amounts to projecting between two convex sets, $\mathcal{I}$ and $\mathcal{M}$, at each step mapping a point in one set to the nearest point in the other set. This is a standard algorithm that converges to the points in $\mathcal{I}$ and $\mathcal{M}$ that are closest to each other.

More Generally

A bit more generally, if the images in $\mathcal{M}$ do not lie perfectly on a low-dimensional space but lie near one, and if the cue has a small amount of noise in addition to occlusion, we can show that the algorithm will converge to an image that is either the correct one, or very similar to it. Thus, intuitively, our approach approximates a standard algorithm for finding the nearest points between two convex sets by using an intermediate, lower-dimensional space to speed up the process of finding nearest neighbors.

This ties in with the idea that considering $\mathcal{M}$ to be a probability distribution

helps eliminate spurious local minima.

Other Approaches

The algorithm is also straight-forward to encode in a neural net. Although it is beyond our present scope to discuss the best way of doing that, we note that such an implementation seems to require far less than $O(nd)$ connections. Baum, Moody and Wilczek[17], for example, describe a neural net for associative memory that performs perfectly, but requires $O(nd)$ connections.

We can briefly contrast our algorithm with current approaches to performing image matching in the presence of occlusion. These typically rely on computing descriptions of local pieces of the image, perhaps using the output of multi-scale filters (e.g., Rao and Ballard[98]). Then some robust voting or matching scheme is used so that if some local features are occluded, the correct match can still be found.

We view this type of approach as complementary to our own. It is a feed-forward approach in which the image is projected onto a feature set, then projected into the space of previously stored images; this is similar to performing two steps of our iterative approach. This prior work primarily focuses on the question of what can be an effective feature set for image matching; we focus on demonstrating some advantages of an iterative projection algorithm that can work with a range of possible feature sets. These approaches can be combined in the future by applying our algorithm to different sorts of features, such as the output of multi-scale filters or some steerable filter. Rao and Ballard[99] presents a different way of doing this.

Conclusion

Thus, to tie up, we can say the following. Referring back to the skeletal iteration scheme in section 5.4, we see that combining the two steps 5.7 and 5.4; as well as the steps 5.5 and 5.6; result in an iteration scheme that stays entirely in the feature space $\mathcal{F}$; and thus can be done efficiently.

These considerations also let us select a starting point of the iteration which leads to a quick convergence to a good solution. We have thus achieved our goal of efficiently doing associative recall from a memory of images.

Other approaches handle occlusion by relying on local features, such as the output of multi-scale filters (e.g., Rao and Ballard[98]). Our approach is complementary to these; our iterative matching method that can be used with any appropriate feature set.

Chapter 7

Associative Memory for Words: Playing Hangman and Superghost

7.1 Introduction

In chapter 5 we described a three level iterative architecture for associative memory, where the set of features mediated the matching between the input and the item in memory. This architecture was captured in the very high level description of the algorithm for associative recall described in section 5.4. In this chapter, we use this iterative architecture to create an associative memory for words. This is an expanded version of the paper [60].

7.2 Versions of the Problem: Hangman versus Superghost

To help ground our discussion, let us describe the kinds of problems we are trying to solve. Thus, we are given a memory $\mathcal{M}$ which consists of English words, say all words in a dictionary. As in the previous chapter, we will assume this memory to be static: we do not add new words to the dictionary, nor do we remove any word.

Our task is to search for a matching item in the dictionary given a fragment of it. Thus, in a hypothetical experiment, our memory would be presented with some input cue, say '__tr_lo__', and it would be expected to fill in the blanks to get a valid word.[1] This is very similar to the computer game hangman. This particular task is directly analogous to the face recognition architecture we considered in the last chapter, in that the position and number of the unknown letters is known when the cue is presented.

A slightly more difficult version of the task occurs in the game *superghost*, where the lengths of the blanks are unknown. Thus, our cue would be a string like '*p*l*c*' and 'palace', 'place', 'simplicity' and 'application' are all valid answers. We could also consider cues like '*pl*c*', for which 'palace' would be invalid, but for which the others remain valid.

We thus seek a good feature space which make it possible to solve this learning task in an efficient manner. We also describe psychological experiments which attempt to understand how humans solve similar problems.

[1] In our example, one valid completion is "astrology".

7.3 Prior Work on Models for Memory (Psycho-physical)

There has been a great deal of prior work on associative memory. Many recurrent neural networks have been proposed as models (Anderson[3] contains a review). Of these, perhaps most similar in spirit to our approach is the bidirectional models of Kosko ([76]) and Sommer and Palm ([110]). Our approach differs quite a bit from these, though, in that we maintain a complete memory of items, and use features as an efficient intermediate step in matching. Our use of features is perhaps more related to feed-forward neural nets, and especially the "information bottleneck" approach of Tishby, Pereira and Bialek[114]. Our work differs from feed-forward methods in many ways, especially in that our method is iterative, and uses features symmetrically to relate the memory to input in both directions. Many other models iteratively combine top-down and bottom-up information (e.g., Hinton et al[54], Rao and Ballard[99]). The structure of our algorithm is particularly related to methods like the Wake-Sleep algorithm of Hinton et al, although we differ from these in many ways, especially our use of this for associative memory with complete memory of stored items.

In spirit our approach is also similar to that of Ullman's[116] approach in visual object recognition, which combines a bottom-up perceptual organization with top-down knowledge, although the way in which we relate top-down and bottom-up information is quite different. Also, related are Markov models for illusory contours as explored by Mumford ([84]) and Williams and Jacobs([122]).

7.3.1 Other Memory Models in Psychological Literature

Other models for content-addressable memory has been proposed in the psychological literature. Many of these can be described as a form of convolutional memory. (See, for example Murdoch [86] and Eich [40]) In these memories, the current state of the memory is stored as a vector in some space and the information corresponding to an input is retrieved by computing the convolution between the input vector and the state vector.

A competing model, the matrix memories, was put forward by Pike ([94]), where the state was represented as a matrix, and the retrieval operation consisted of multiplying this matrix with the input vector. Because of the larger size of the state, matrix memories can have more complicated effects. Thus, matrix memories tend to model order effects found in human memory much better than the convolutional memories.

7.4 Possibilities for Feature Space

For simplicity, we will treat the case of hangman. Let us recall the iterative algorithm from section 5.4:

{**Input:** $C \in \mathcal{I}$}

loop

$F_1 \leftarrow match_{\mathcal{F},\mathcal{I}}\,(C)$

$M \leftarrow match_{\mathcal{M},\mathcal{F}}\,(F_1)$

$F_2 \leftarrow match_{\mathcal{F},\mathcal{M}}\,(M)$

$C \leftarrow match_{\mathcal{I},\mathcal{F}}(F_2)$

end loop

In this algorithm, $\mathcal{M}$ denotes the memory. $\mathcal{I}$ is the representation of all possible completions of the input cue. We include all possible completions of the input in $\mathcal{I}$, even the ones that are not valid words. Thus for the cue 'p _ l _ c _', we include obviously nonsense words like 'pdlbce' in $\mathcal{I}$. Our goal is to construct some feature space $\mathcal{F}$ and corresponding matching functions, so that the above scheme lets us quickly obtain a completion which is incorrect with a low probability.

To motivate our choice of the feature space let us look at a toy model for a matrix memory for associative recall of words. Thus, given a dictionary of words $\{w_i\}_{i \in D}$, all of the same length l,we form the state matrix with $|D|$ rows and l columns. Suppose, further, that we are in a binary world where the alphabet consists of just two symbols: $+1$ and -1. Thus, each w_i is a l length row vector of $+1$ and -1's. The inner product of any w_i with itself is l; for any other vector w (whether in the memory or not) the inner product of w_i and w is less than l. Thus given a partial cue v [2], its product with the state matrix will result in a vector whose largest entries are those corresponding to words in the dictionary which are closest to v in the Hamming distance sense.

This is a memory which performs perfectly according to a reasonable metric, but uses too much space ($\Theta(l|D|)$). Our goal would be to cut down on the space usage. We note that if our memory elements are completely heterogeneous, i.e. there is no statistical regularity among the possible memory elements, then this is

[2]We assume that its unknown positions are filled with 0's.

the absolute best we can do. The only way we can achieve further compression is by utilizing its statistical features.

Since we are looking at memories for English words, it is natural to see what are their natural statistical characteristics. There has been a lot of study on this subject. Shannon ([108]) described various order Markov models for formation of English words. The book by Cover and Thomas ([31]) gives the results of generating English words according to Markov models of various orders.

It is clear that we need to work in a probabilistic setting. We explained in section 5.2.2 how we can formulate this generalization. Thus, we consider $\mathcal{M}$ to be the set of probability distributions over the memory items, and $\mathcal{I}$ to be the set of probability distributions over the set of completions of the input cue.

This slight generalization can be recast in the old language where all these spaces, $\mathcal{M}$, $\mathcal{F}$ and $\mathcal{I}$, are probabilistic in the above sense, i.e., their elements are probability distributions over the whole set. We can think of the elements in the old (non-probabilistic) sense as distributions which are delta functions.

This generalization gives us an added flexibility to handle algorithms dealing with probability distributions. We will see that this leads to algorithms for associative recall which are quick and which work correctly most of the time.

It is clear that n-grams for various values of n encode quite a lot of information. For this reason, we choose our feature space $\mathcal{F}$ to be the set of probability distributions on the set of n-grams for some specific n. In practice, we will stick to trigrams and bigrams. In a later section we will see that the case for general n can be useful as well.

7.5 The Iterative Framework

To derive the algorithm for associative recall with (probability distributions on) n-grams as intermediate features, we will follow the toy matrix model which we considered in the last section.

This particular matrix multiplication model is just a formal way of counting the number of matching letters between the input cue and each element in memory. The obvious generalization of this approach, is to count the number of matching n-grams. The problem, of course, is that the n-grams also depend on (unknown) letters appearing in the blank positions. Thus, we need a method to estimate the likely n-grams that might appear in a completion of the input query.

It is here, that we can utilize known facts about statistics of n-grams in natural languages. We can use a probabilistic approach to estimate the unknown letters in the cue. More precisely, we use some probabilistic assumptions (which may not actually be true, but simplifies the estimation) to estimate distributions on the n-grams.

The approach is a reformulation of Pearl's ([92]) *belief propagation* algorithm and has been itself rediscovered a number of times in various areas. In the *hidden Markov model* literature, it is known as the *forward-backward* algorithm, because of the order in which the quantities are computed. We used a more general version of the algorithm in the chapter on features (chapter 3) when we treated the problem of finding features for the stereo matching problem.

The distribution on trigrams (or n-grams) represent our knowledge of the statistical regularities of the problem and provide the handle by which we can reduce

the complexity of the associative recall problem.

To formulate our iterative scheme, we need to define the four matching functions between the three spaces defined so far. We look at our iterative algorithm which successively refines its estimate of which word is a possible completion of the input cue.

Thus, we start with a probability distribution on the dictionary (memory) elements which does not depend at all on the input cue. This initial distribution could be the uniform distribution on the words. If we want to be a little more informative, we could even start with the probability distribution on words occurring in a sufficiently representative corpus.

As the iteration proceeds we have more and more refined distributions on the memory elements, which ultimately converge to a single element with overwhelming certainty.

7.5.1 The Matching Functions

To complete our iterative scheme, we need to define the matching functions: $match_{\mathcal{I},\mathcal{F}}$, $match_{\mathcal{F},\mathcal{M}}$, $match_{\mathcal{M},\mathcal{F}}$ and $match_{\mathcal{F},\mathcal{I}}$ which are illustrated in figure 5.2 of chapter 5.

The Matching Function: $match_{\mathcal{F},\mathcal{M}}$

Let us assume, for concreteness, that we use the set of distributions on trigrams as our feature space. We first notice that there is a natural function from $\mathcal{M}$, the set of probability distributions on the memory elements, to $\mathcal{F}$, the set of probability distributions on the trigrams. This projection function merely constructs a

distribution on the set of trigrams from a given distribution on the set of words. The constructed distribution assigns a probability to any particular trigram which is proportional to the sum of probabilities of the words that trigram occurs in, the sum being weighted by the number of times the trigram occurs in that word. This is the naive algorithm to construct a Markov model from a given distribution on the set of words.

We choose our matching function $match_{\mathcal{F},\mathcal{M}}$ to be the projection function described above.

More formally, let $\{w_i\}_{i \in D}$ be the set of words in memory. For any word w, let $\Delta(w)$ be the set of trigrams occurring in the word w and let $\mathbf{Count}(t, w)$ be the number of times the trigram t occurs in the word w.

Then, given any probability distribution $\mathbf{Pr}_{\mathcal{M}} \in \mathcal{M}$, $match_{\mathcal{F},\mathcal{M}}$ creates the probability distribution $match_{\mathcal{F},\mathcal{M}}(\mathbf{Pr}_{\mathcal{M}}) \equiv \mathbf{Pr}_{\mathcal{F}}$ defined by

$$\mathbf{Pr}_{\mathcal{F}}(t) = \mathbf{Normalize}\left(\sum_{w \in D} \mathbf{Count}(t, w)\mathbf{Pr}_{\mathcal{M}}(w)\right) \quad \text{for all trigrams } t \qquad (7.1)$$

where the notation $\mathbf{Normalize}(\cdot)$ denotes that its argument is normalized so that its sum over all trigrams t is 1.

The Matching Function: $match_{\mathcal{M},\mathcal{F}}$

To go the other way, from $\mathcal{F}$ to $\mathcal{M}$, the toy matrix memory model gives some help. Thus, following that model, we assume that we have the information about which trigrams occur in every word; though not their positions. Thus, for the word

'testing', we record connections from the trigrams 'tes', 'est', 'sti', 'tin' and 'ing' to the word 'testing', without recording the positions in any way.

Then, given a distribution on the trigrams, we compute a distribution on the set of words, by a breaking the word up into its constituent trigrams, and assuming each trigram to be independent of the others. Thus, if a word w has the set of trigrams $\Delta(w)$, and we are given a probability distribution $\mathbf{Pr}_{\mathcal{F}}$ on the set of trigrams, then $match_{\mathcal{M},\mathcal{F}}$ constructs a probability distribution $match_{\mathcal{M},\mathcal{F}}(\mathbf{Pr}_{\mathcal{F}}) \equiv \mathbf{Pr}_{\mathcal{M}}$ on the set of words in the memory defined by

$$\mathbf{Pr}_{\mathcal{M}}(w) = \mathbf{Normalize}\left(\prod_{t\in\Delta(w)} (\mathbf{Pr}_{\mathcal{F}}(t))^{\mathbf{Count}(t,w)}\right) \quad \forall w \in D \tag{7.2}$$

where D denotes the set of words in the memory (D stands for *dictionary*).

The Matching Function: $match_{\mathcal{F},\mathcal{I}}$

We now describe the matching functions between the feature space $\mathcal{F}$ and the input space $\mathcal{I}$. We recall that the input space $\mathcal{I}$ consists of all probability distributions on the set of possible completions of the input cue c.

Thus, an element of $\mathcal{I}$ is basically a probability distribution over the set of (syntactic) words which complete the input cue. For every position in the hangman cue this probability distribution generates a probability distribution on the set of trigrams. We will abuse our notation and denote by $\mathcal{I}$ such a a set of probability distributions on the set of trigrams, one for each blank position of the cue, which arise from a distribution on the set of completions of the input cue in this way.

With this understood, we can immediately define $match_{\mathcal{F},\mathcal{I}}$ to be the function which marginalizes the distribution on trigrams by forgetting the position where the trigram occurs.

The Matching Function: $match_{\mathcal{I},\mathcal{F}}$

We can also define last remaining matching function very easily. We consider the input to be formed from an array of trigrams. $match_{\mathcal{I},\mathcal{F}}$ operates by starting with the probability distribution on trigram specified by some element $\mathbf{Pr}_{\mathcal{F}}$ of $\mathcal{F}$ and using Pearl's ([92]) belief propagation algorithm on the input cue using the trigram probabilities according to $\mathbf{Pr}_{\mathcal{F}}$ as the transition probabilities (and initial probabilities).

The result is a structure which codes the trigram probability distributions at every (unknown) position of the input cue. This is the element of the input space ($\mathcal{I}$) that is the image of $\mathbf{Pr}_{\mathcal{F}}$ under $match_{\mathcal{I},\mathcal{F}}$.

7.5.2 Getting Everything Together

With all the matching functions in place we can state the full iterative algorithm from section 5.4 after we substitute these matching functions.

{**Input:** C is a cue.}

$I \leftarrow$ "natural" trigram probabilities for every blank trigram position in C

loop

$\{F_1$ is assigned the trigram distribution corresponding to $I\}$

$F_1 \leftarrow \textbf{Marginalize}(I)$

$\{M$ is assigned the distribution on the memory elements that assume all trigrams occur independently.$\}$

$M \leftarrow \mathbf{Pr}_{\mathcal{M}}, \quad \text{where } \mathbf{Pr}_{\mathcal{M}}(w) = \textbf{Normalize}\left(\prod_{t \in \Delta(w)} (I(t))^{\textbf{Count}(t,w)}\right)$

$\{F_2$ computes the trigram distribution corresponding to $M\}$

$F_2 \leftarrow \mathbf{Pr}_{\mathcal{F}}, \quad \text{where } \mathbf{Pr}_{\mathcal{F}}(t) = \textbf{Normalize}\left(\sum_{w \in D} \textbf{Count}(t,w) M(w)\right)$

{Use belief propagation on F_2 to fill in I}

$I \leftarrow \textbf{Belief\_Propagation}\,(F_2)$

end loop

In the next few sections we will investigate the relation of this iterative algorithm with other natural approaches one can pursue in this problem. We will find that this algorithm generalizes these approaches in a very natural way.

7.6 Superghost Queries

At this point, our iterative algorithm can handle hangman queries. The fundamental step of this approach was a use of the belief propagation algorithm to estimate the trigram probability at locations where the letters were not known. This belief propagation algorithm needs the number and positions of the blanks to known beforehand. This knowledge is lost in superghost.

We take the *ad hoc* approach that for every '*' in a superghost cue, there can only be a small number of possible blank positions that '*' can represent. We put a prior model on this distribution of number of blanks and solve the hangman problem for each such expansion. Our prior model is a very simple truncated exponential distribution on the number of blanks.

Experimental evidence suggests that among many factors which affect human memory performance is the effect of the starting letter. Thus given a cue, people tend to start searching for words which has the same word stem as the given cue. We will explore the existing psychological literature in a later section. We remark at this point, that our computations have a similar characteristic as well. If we have a sequence of successive blanks, the distributions near the edges of this sequence are more peaked than those near the middle. Of course, in the computational setting this does not lead to an increase in the running time of the estimation algorithm.

7.7 Some Initial Complexity Considerations

At this stage we try to see how well our iterative scheme behaves in terms of space and time complexity and which operations are bottlenecks to performance in this memory architecture.

The main storage needed is to record which trigrams occur in which word in memory. We need $\Theta(D)$ space to store this information. Asymptotically this is better than the $\Theta(lD)$ space requirement for a naive storage of D words, each length l. For realistic dictionaries for English, however, the space requirement for storing this information is about the same as needed to store the dictionary naively. The architecture is still useful for storing strings of discrete symbols where there is not much topological structure on the symbols to aid in recognition, but where there are strong local correlations between nearby symbols.

Let us also look at the time taken for the recall algorithm to run. The estimation step using the belief propagation algorithm takes time independent of the size of the dictionary. The operations which contribute most to the time complexity are those that need an iteration over the dictionary elements.

For an implementation of our iteration scheme on a serial computer, the time-consuming steps are the two matching steps: $match_{\mathcal{M},\mathcal{F}}$ and $match_{\mathcal{F},\mathcal{M}}$. Both of these need an iteration over all the dictionary elements and are thus expensive.

On a neural network architecture, however, the matching function from $\mathcal{F}$ to $\mathcal{M}$ can be implemented very quickly, if we have an appropriate network architecture.

7.8 From Features to Retrieval

At this point, our iterative algorithm generates very good completions when our features are trigrams (almost always perfect); and makes a few errors when we use bigrams as features. So performance-wise it matches the naive algorithm which goes through each element of the memory and checks if it matches the input cue.

As we indicated in the last section, however, it still needs to iterate over all the dictionary elements at some point. To understand how much extra power this look-up gives, we need to see how a completely local algorithm performs. Thus, in this section we explore how well we perform when we do not have access to any global information.

7.8.1 Dynamic Programming Formulation: Viterbi Algorithm

Then, let $\{t_i\}$ be the set of $26^3 = 17576$ possible trigrams. Most of these occur very infrequently (and thus can be ignored, as we shall see). In practice, if we keep about 3000 most frequent fragments, we get practically the same performance.

In any case, let us denote the proportion of times the trigram t_i occurs in natural language as p_i. With these notations in place, let us describe the cost function we put on the set of all possible completions of a given cue.

For a particular input cue c, let $\Gamma(c)$ denote the set of possible completions of c. Each such completion is an element of the set $\mathcal{I}$ defined in a previous chapter and does not contain any unknown positions. Let t be any such completion and let $\Delta(t)$ be the set of trigrams in the "word" t. Then the cost associated to a particular

completion is

$$\mathcal{C}(t) = \sum_{t_j \in \Delta(t)} (-\log p_j) \quad \text{for all } t \in \Gamma(c) \tag{7.3}$$

The idea in the trigram estimation process is to find the completion which minimizes this cost function. Thus, our goal is to solve the optimization problem:

$$\begin{aligned} t^*(c) &= \operatorname*{argmin}_{t \in \Gamma(c)} \mathcal{C}(t) \\ &= \operatorname*{argmin}_{t \in \Gamma(c)} \sum_{t_j \in \Delta(t)} (-\log p_j) \end{aligned} \tag{7.4}$$

The 1 dimensional structure of the words lets us solve this optimization very easily by means of the *viterbi algorithm*, first described by Viterbi ([119]) which is essentially the dynamic programming approach to solving a minimum path problem with non-negative costs. A more general description was earlier given by Bellman ([20]). Shortest path algorithms has been described by Dijkstra ([37])and by Bellman and Ford.([42, 21]).

The viterbi algorithm is a mainstay of estimation in *Markov Chains* and the related *Hidden Markov Models*.

The reason viterbi algorithm can be used in this minimization is the following. Suppose we have a sequence of blanks following some known letters: 't e _ _ _ _'. The dynamic programming minimization process works by filling in the blanks by all possible letters starting from the leftmost blank. Filling in a new letter generates a new trigram. The trigrams are arranged in successive positions depending on the position of their last letter. The algorithm proceeds by considering the possible trigrams in successive positions. For any position, for all possible trigrams, we keep

track of the minimum cost completion which ends in that trigram. Knowing this information, it is possible to build up the same information for the next layer of trigrams. The linear order of the trigrams, which follow from the linear order of the blanks in the word, lets us use the efficient dynamic programming approach.

Viterbi algorithm can actually solve the minimum cost completion problem for a more general class of cost functions where the cost of adding a new trigram depends on the trigram occurring before it. Thus this is a Markov model in the trigram representation. This generalization will become important when we consider HMM models for memory.

In this more general setting, our cost function gets modified as follows. First we are given some transition probability between different trigrams at adjacent positions. Let us denote the trigram occurring at location i of some word (or fragment of word) w by $\Lambda_w(i)$. Then, if, as before, $\{t_i\}$ is the set of possible trigrams, with the associated proportions $p_i = \mathbf{Pr}(t_i)$, we define the additional quantities $a_{i,j}$ denoting the conditional probability $\mathbf{Pr}(\Lambda_w(n+1) = t_j \mid \Lambda_w(n) = t_i)$ which we assume independent of w and n. Thus we are assuming that the successive trigrams in any word are generated from some first order Markov model. The definition of $a_{i,j}$ as a conditional probability imposes the normalization condition:

$$\sum_j a_{i,j} = 1 \tag{7.5}$$

We can estimate the quantities p_i and $a_{i,j}$ by sampling from the dictionary itself.

As before, we denote by $\Gamma(c)$ the set of possible completions of c. The difference is, we look at arrays of trigrams of any particular completion instead of just the set

of trigrams. Thus, for any completion $t \in \Gamma(c)$, let us denote the array of trigrams generated from t by the symbol Π_t. The i-th element of this array would be the trigram $\Pi_t(i)$. Let this trigram be denoted by t_{τ_i}, and therefore, the associated proportion is p_{τ_i}. Then the modified cost function of any completion $t \in \Gamma(c)$ of c is

$$\mathcal{C}_{generalized}(t) = (-\log p_{\tau_1}) + \sum_{n=2}^{l-2}(-\log\,(a_{\tau_{n-1},\tau_n})) \tag{7.6}$$

and the corresponding minimization problem is

$$\begin{aligned} t^*(c) &= \underset{t\in\Gamma(c)}{\operatorname{argmin}}\,\mathcal{C}_{generalized}(t) \\ &= \underset{t\in\Gamma(c)}{\operatorname{argmin}}\,[(-\log p_{\tau_1}) + \sum_{n=2}^{l-2}(-\log\,(a_{\tau_{n-1},\tau_n}))] \end{aligned} \tag{7.7}$$

Thus, the viterbi algorithm lets us estimate a completion of the cue, and hence the set of trigrams (more generally, n-grams) in it. Thus, it constitutes one choice for the matching function $match_{\mathcal{F},\mathcal{I}}$.

7.9 The Iterative Framework: A Local Version

The viterbi algorithm of the last section lets one start with a probability distribution on the trigrams (and a consistent set of conditional probabilities), and lets one complete a partial cue to a completed form. This completed cue will possibly not be a valid English word, but this is the best possible completion of the cue given only the local information inherent in the probability values of the trigrams.

In this section, we will explore how far we can go when we confine ourselves entirely to local computations, without any look-up of the dictionary at all. One approach would be to use the viterbi algorithm to estimate the "best" possible completion of the input cue.

However, if we start with the p_i's and $a_{i,j}$'s initialized to the corresponding relative frequency values found in English text, the quality of reconstruction is poor. This fact is a reflection of why need the look-up of the dictionary in the iterative algorithm. However, in a sense, the viterbi algorithm does provide an alternative to the look-up step. Given a probability distribution on the set of trigrams, it lets us form an estimate of a memory element. We can thus use it as a kind of local matching function from $\mathcal{F}$ to $\mathcal{M}$.

However, since we do not have a local equivalent of $match_{\mathcal{F},\mathcal{M}}$, we can only use the viterbi algorithm in the last iteration.

To see what this algorithm looks like without the look-ups of dictionary, let us write down the iterative scheme in this case.

We actually start the iteration with the generic distribution on the trigrams consisting of their relative frequencies of occurrence.

$\{I$ stores the position-wise probability distributions on trigrams.$\}$

$\{T$ stores probability distributions on trigrams.$\}$

loop

$I \leftarrow$ **Belief_Propagation** (T)

$T \leftarrow$ **Marginalize** (I)

end loop

This is exactly the *Baum Welch* learning algorithm for estimating transition probabilities in *hidden Markov models (HMM)*. In the next section we give a quick introduction of HMM's and cover the Baum-Welch algorithm.

7.10 Hidden Markov Models

In this section we give a brief introduction to HMM's. We will see that the *state estimation problem* for an HMM with known parameters can be solved using the viterbi algorithm of subsection 7.8.1. We will formulate the viterbi algorithm formally in this section, as well as explain the Baum-Welch re-estimation procedure for parameter learning.

The theory of hidden Markov model has been around for many years. The basic theory was expounded by Baum and others in a series of papers ([12, 13, 14, 15, 16]). This was implemented in a speech understanding setting by Baker ([8]) and Jelinek et al ([61, 6, 63, 62, 10, 64, 7]) See Rabiner([97]) for an introduction. Hidden Markov models have connections to lots of similar models which treat linear systems with Gaussian noise: Kalman filters, PCA etc. For a unified treatment, see Roweis and Ghahramani([102]). The connection between HMM's and Kalman filters was earlier elucidated by Digalakis et al ([36]).

Let us first consider a system which is described at any (discrete) time as being in one of a set of N distinct states: $\{S_1, S_2, \ldots, S_N\}$ and which undergoes state transitions at regularly spaces discrete time points. We denote the time instances when the state change occurs by $t = 1, 2, \ldots$, and we denote the state at time t by q_t.

We suppose that the state transition process is a stochastic process. To describe such a process in full detail would normally require the specification of the full conditional probabilities: $\mathbf{Pr}(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \dots)$. However, we assume that the state changes follow a first order Markov model, and so this particular conditional is actually equal to the same probability conditioned on the immediately preceding time step:

$$a_{i,j} = \mathbf{Pr}(q_t = S_j | q_{t-1} = S_i)$$

We assume that the chain is stationary, i.e. the above quantity is independent of time t.

The quantities $a_{i,j}$ are non-negative and satisfy the normalization condition given in equation 7.5:

$$a_{i,j} \geq 0, \tag{7.8}$$

$$\sum_j a_{i,j} = 1, \quad \forall i \in \{1, 2, \dots, N\} \tag{7.9}$$

The above process could be called observable since the sequence of states in any realization of it become known. In a realization of a *hidden Markov model*, the actual state sequence remains unknown. Instead, there is a probabilistic function defined on the states, whose value is observed.

More formally, an HMM is characterized by

1. N, the number of states, and the set of states $S = \{S_1, S_2, \dots, S_N\}$. As for Markov models, we denote the state at time t by q_t.

2. M, the number of observation symbols. We denote the individual symbols by $V = \{v_1, v_2, \dots, v_M\}$ and the observed symbol at time t by o_t.

3. The $N \times N$ state transition matrix $A = \{a_{i,j}\}$ defined, as above, by

$$a_{i,j} = \mathbf{Pr}(q_t = S_j | q_{t-1} = S_i), \quad 1 \leq i, j \leq N$$

4. The observable symbol probability distribution, specified by the $N \times M$ matrix $B = \{b_j(k)\}$:

$$b_j(k) = \mathbf{Pr}(o_t = v_k | q_t = S_j) \quad 1 \leq j \leq N, 1 \leq k \leq M. \tag{7.10}$$

5. The initial state distribution $\pi = \{\pi_j\}$ with

$$\pi_j = \mathbf{Pr}(q_1 = S_j) \quad 1 \leq j \leq N. \tag{7.11}$$

N, and M are usually omitted. The matrix and vector parameters are combined together to form the parameters of the hidden Markov model. This will be denoted by

$$\lambda = (A, B, \pi)$$

7.10.1 Three Fundamental Problems Connected with Hidden Markov Models

Before the HMM approach can become useful,we need to be able to solve three basic problems:

1. **The evaluation problem:** Given an HMM λ and a sequence of observations $O = o_1, o_2, \ldots, o_T$ representing a realization of the HMM, how can we we (efficiently) compute the observation probability $\mathbf{Pr}(O \mid \lambda) = \mathbf{Pr}(o_1, o_2, \ldots, o_T \mid \lambda)$.

2. **The decoding problem:** Given an HMM with model parameters λ and a sequence of observations $O = o_1, o_2, \ldots, o_T$ representing a realization of the HMM, how do we calculate a corresponding sequence of states: $Q = q_1, q_2, \ldots, q_T$ which is the most likely sequence to have produced the observed sequence.

3. **The learning problem:** Given an HMM, and a sequence of observations $O = o_1, o_2, \ldots, o_T$, how do we adjust the model parameters $\lambda = (A, B, \pi)$ so as to maximize $\mathbf{Pr}(O \mid \lambda)$.

7.11 Quick Overview of the Solutions of the Three Fundamental Problems

We will give a quick overview of the standard solutions of these problems and their complexities.

7.11.1 Evaluation Problem

The naive solution of the evaluation problem would be to write the required probability as a integral over all possible state sequences. Given, any such, $Q = q_1, q_2, \ldots, q_T$, the probability of the observation sequence is:

$$\mathbf{Pr}(O \mid Q, \lambda) \;=\; \prod_{t=1}^{T} \mathbf{Pr}(o_t \mid q_t, \lambda) \;=\; \prod_{t=1}^{T} b_{q_t}(o_t)$$

Since the probability of any such state sequence is

$$\pi_{q_1} a_{q_1,q_2} a_{q_2,q_3} \cdots a_{q_{T-1},q_T}$$

we can calculate the probability of the observations as

$$\begin{aligned}\mathbf{Pr}(O \mid \lambda) &= \sum_{Q} \mathbf{Pr}(O \mid Q, \lambda)\mathbf{Pr}(Q \mid \lambda) && (7.12)\\ &= \sum_{q_1, q_2, \cdots, q_T} \pi_{q_1} b_{q_1}(o_1) a_{q_1,q_2} b_{q_2}(o_2) a_{q_2,q_3} \cdots a_{q_{T-1},q_T} b_{q_T}(o_T) && (7.13)\end{aligned}$$

However, the complexity of this algorithm is prohibitive: $\Theta(TN^T)$.

The standard way to efficiently solve the evaluation problem is through the *forward* algorithm:

We define the forward variables $\alpha_t(i)$ to be the probabilities of partial observation sequences when the terminating state is known to be S_i:

$$\alpha_t(i) = \mathbf{Pr}(o_1, o_2, \ldots, o_t, q_t = S_i \mid \lambda) \tag{7.14}$$

A little thought convinces one that these variables satisfy the recurrence

$$\alpha_{t+1}(j) = b_j(o_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{i,j} \tag{7.15}$$

with the initial condition

$$\alpha_1(j) = \pi_j b_j(o_1) \tag{7.16}$$

Thus, the whole α array can be computed in $\Theta(TN^2)$ time, and we can get the probability values $\mathbf{Pr}(O \mid \lambda)$ by forming the marginal distribution on α_T:

$$\mathbf{Pr}(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Similarly, we can form the backward probabilities

$$\beta_t(i) = \mathbf{Pr}(o_{t+1}, o_{t+2}, \ldots, o_T \,|\, q_t = S_i, \lambda)$$

and calculate these quantities using the recursion:

$$\beta_T(i) = 1 \tag{7.17}$$

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j)\, a_{i,j}\, b_j(o_{t+1}) \tag{7.18}$$

We will use these quantities in solving the decoding and learning problems as well.

7.11.2 Decoding Problem

Here we want to find the most likely state sequence for a given model λ and an observation sequence $O = o_1, o_2, \ldots, o_T$. The naive approach of calculating the most probable state at every time instance t and concatenating these values together may lead to state sequences which are impossible. Thus, for the model $\lambda = (A, B, \pi)$, and for i and j such that $a_{i,j} = 0$, but there might be successive time instances t and $t+1$ such that S_i is the most probable state at t and S_j is the most probable state at $t+1$. However, such a sequence cannot actually occur because $a_{i,j} = 0$.

To get a reasonable sequence of states, we estimate the state sequence by maximizing $\mathbf{Pr}(Q\,|O\,\lambda)$. By Bayes' theorem, this is equivalent to maximizing $\mathbf{Pr}(Q, O \,|\, \lambda)$. After we write down the algebraic expressions of the quantities involved, it will be clear that we are dealing with a generalization of equation 7.7:

$$
\begin{aligned}
Q^* &= \operatorname*{argmax}_{q_1,q_2,\cdots,q_T} \mathbf{Pr}(q_1, q_2, \ldots, q_T \,;\, o_1, o_2, \ldots, o_T \mid \lambda) \qquad (7.19)\\
&= \operatorname*{argmax}_{q_T} \operatorname*{argmax}_{q_1,q_2,\cdots,q_{T-1}} \mathbf{Pr}(q_1, q_2, \ldots, q_{T-1}, q_T \,;\, o_1, o_2, \ldots, o_{T-1}, o_T \mid \lambda)
\end{aligned}
$$

The second line suggests we use the following quantities to do the actual maximization:

$$
\delta_t(i) = \max_{q_1,q_2,\cdots,q_{t-1}} \mathbf{Pr}\left[\, q_1, q_2, \ldots, q_{t-1}, q_t = S_i, o_1, o_2, \ldots, o_{t-1}, o_t \mid \lambda\right] \qquad (7.20)
$$

This satisfies recurrence relations similar to α:

$$
\begin{aligned}
\delta_1(i) &= \pi_i b_i(o_1) \quad 1 \le i \le N \qquad (7.21)\\
\delta_{t+1}(j) &= b_j(o_{t+1} \left[\max_{1\le i\le N} \delta_t(i) a_{i,j}\right] \qquad (7.22)
\end{aligned}
$$

This is the celebrated *viterbi algorithm* (See Viterbi [119] and Forney[43]). To get the actual state sequence, we need to keep back pointers at every state. We solve

$$
j^* = \operatorname*{argmax}_j \delta_T(j)
$$

and follow the back pointers to get the sequence of states.

7.11.3 Learning Problem

The learning problem is by far the most complicated of the three basic problems and there is no known way to *analytically* solve it. However, an iterative procedure

based on the *EM* family of algorithms was given by Baum and his colleagues ([12, 13, 14, 15, 16], see also Dempster et al [35]). This is the *Baum Welch* procedure, also called the *forward backward* iteration.

To derive the Baum-Welch re-estimation, we first define the auxiliary variables $\xi_t(i,j)$:

$$\xi_t(i,j) = \mathbf{Pr}(q_t = S_i, q_{t+1} = S_j \,|\, O, \lambda)$$

and the variables $\gamma_t(i)$ representing probabilities of a particular state at a particular time:

$$\gamma_t(i) = \mathbf{Pr}(q_t = S_i \,|\, O, \lambda)$$

Of course $\sum_j \xi_t(i,j) = \gamma_t(i)$.

We can compute both these variables in terms of the α's and the β's as follows:

$$\xi_t(i,j) = \frac{\alpha_t(i)\; a_{i,j}\; b_j(o_{t+1})\; \beta_{t+1}(j)}{\sum_{i,j} [\alpha_t(i)\; a_{i,j}\; b_j(o_{t+1})\; \beta_{t+1}(j)]} \tag{7.23}$$

$$\gamma_t(i) = \frac{\alpha_t(i)\; \beta_t(i)}{\sum_i [\alpha_t(i)\; \beta_t(i)]} \tag{7.24}$$

Baum Welch iteration works by starting from some particular parameter values $\lambda = (A, B, \pi)$ and computing new values $\lambda' = (A', B', \pi')$ which improve the match with the known observation sequence. The iteration actually chooses λ' to maximize the following quantity considered as a function of λ':

$$Q(\lambda, \lambda') = \sum_Q \mathbf{Pr}(Q \,|\, O, \lambda) \log [\mathbf{Pr}(Q, O \,|\, \lambda')]$$

It has been shown (Baker [8] and Baum and Sell [14]) that choosing this values of λ' ensures that the posterior probability improves:

$$\mathbf{Pr}(O \mid \lambda') \geq \mathbf{Pr}(O \mid \lambda)$$

The actual formulas for re-estimation follow if we try to estimate the various probabilities using counting arguments. Thus, $\pi'(i)$ would be the expected proportion of the state S_i at time $t = 1$. This turns out to exactly equal to $\gamma_1(i)$. Similarly, writing down fairly intuitive expressions for $a'_{i,j}$ and $b'_j(k)$, we get

$$\pi'(i) = \gamma_1(i) \tag{7.25}$$

$$a'_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{7.26}$$

$$b'_j(k) = \frac{\sum_{t,o_t=v_k} \gamma_t(j)}{\sum_t \gamma_t(j)} \tag{7.27}$$

Note that the normalization conditions are automatically satisfied.

7.12 Using HMM's

We now explore how we can use HMM's in a trigram (more generally n-gram) setting to iteratively refine our distribution over the set of trigrams. We had claimed in section 7.9 that eliminating global lookups from our iterative algorithm leads to a local lookup which is the same as the Baum-Welch re-estimation over the trigrams.

To see that this is really the case, we need to specify our HMM in greater detail. We will describe the state space and the observation symbols but leave the actual

numerical parameters (π, A and B) unspecified.

The first model we consider is a normal Markov model: the matrix B is the identity matrix. Thus, the set of states is the set of trigrams. The only constraint we put is that trigrams between which there is a non-zero transition probability must be consistent. Thus if t_1 = "tes" and t_2 = "rsl", then we demand that $a_{t_1,t_2} = 0$, since no word can have these two trigrams at adjacent positions. A look at the re-estimation formulas convinces one that this condition remains valid for all subsequent estimates of a_{t_1,t_2}.

Thus, given an input cue, say 't e _ _ c _ s t', we determine the number of trigrams (6 in this example). The first trigram clearly has to start with 't e'. We start with some consistent transition probability distribution over the trigrams (say one estimated from some corpus) and use equations 7.23 and 7.24 to calculate the quantities ξ and γ for the cue. After these are computed we use the re-estimation equations 7.25, 7.26 and 7.27 to get the next set of values for the parameters. This exactly follows the steps of the iterative algorithm in section 7.9.

7.13 Modifications: Adding Observation Noise

The difficulty with this approach is that for many cues, the Markov model on trigrams is not a very good estimate of the word formation process. The concrete way this manifests itself is in insufficient number of instances of any particular trigram. Thus, we need to somehow include more trigrams than we can see in any one run.

This is a well-known problem for HMM's in a discrete domain. Jelinek and

Mercer[65], for example, describe a modified architecture where one can automatically interpolate between the desired model (for which there is insufficient data) and a smaller model for which there is sufficient data. The smaller model serves as a regularizer. The interpolated model effectively uses an extra state to probabilistically determine whether to follow the larger model or the smaller model. The probability of the transition is learnt using the Baum-Welch iteration. See also Rabiner ([97]).

In our case, we chose a domain specific modification of our model, so that the evaluation of the state probabilities yield non-zero values for more states. Our approach is to model the word formation process as a hidden Markov model. We keep the state space to be the set of trigrams, as before. The observation symbols are trigrams as well. The noise model, given by the matrix B is such that there is a small probability for the vowels to change to another vowel, and consonants to change to another consonant. At any trigram, we allow only a single letter change. This ensures that over the whole word, the changes are well separated. We assume that the completion of the input cue is the observation sequence of the model thus obtained.

7.14 Boot-strapping: Can We Eliminate Dictionary Lookups?

In section 7.9 we described a completely local version of the iterative algorithm which eliminates dictionary lookups entirely. We found there that this algorithm is essentially a Baum-Welch re-estimation algorithm for the n-gram frequencies. As we shall see, the performance of this algorithm is weaker than the algorithm

with dictionary lookups. We now investigate whether there could be ways we can eliminate global lookups, but still incorporate some measure of global information in our search.

In a Markov, or a hidden Markov model, which was the basis of our local search algorithm, one correlates the letters at various locations by extending the "influence" of each letter on its immediate neighbors. However, there is a stronger dependence between far away letters than can be "explained" by the Markov model. This is the reason why the lookup step improves performance of the iterative algorithm by a large amount.

Thus, a lookup enables one to capture long range dependences that are too complicated to be captured by the simpler HMM. This also indicates that as we increase the size of our neighborhood, we will be able to capture more and more of this "extra" dependence. This is indeed true: at least for smaller neighborhood sizes. Thus, the complete algorithm (with lookups) performs almost perfectly when we work with trigrams, but start showing errors when we use bigrams instead.

However, the increase in modeling power is counterbalanced by fewer examples to learn the distribution from. Thus, if most of the words in our dictionary is between 6 and 9 letters long , having a HMM with neighborhood size of 7 defeats the whole purpose of using a local model to save us the cost of a lookup. Our local models start to become as large as the dictionary.We thus need a way to use a larger model, but inferring its parameters from progressively smaller models, and not directly from the dictionary.

Below, we describe a simple scheme which lets us estimate the transition prob-

abilities of a HMM on $(n+1)$-grams, given the probability values for an HMM on n-grams.

Of course, one course would be to start with a uniform distribution over the $(n+1)$-grams, but this would neglect the gains made by using the simpler model. We thus look for a different technique.

Thus, we start with a the quantities $\pi(t_i)$ and a_{t_i,t_j} where t_i and t_j are n-grams. Our noise model is described in algorithmic terms: a product of models of error on individual letters with a forced cutoff of at most one change over the whole n-gram. We thus just need the noise model on the letters; and this we keep the same as we go from n-grams to $(n+1)$-grams.

The t_i and t_j in a_{t_i,t_j} are compatible if this quantity is non-zero. From now on we will focus exclusively on such pairs. Thus, t_i and t_j are sequences of n letters such that the last $(n-1)$ letters of t_i are identical (as a sequence) to the first $(n-1)$ letters of t_j. Let us assume we are trying to estimate the transition matrix $\overline{A}$ for a $(n+1)$ order HMM from this data. Thus, in this higher order HMM, we will have transition between two $(n+1)$-grams t_k and t_l only if t_k and t_l are consistent. Thus let t_k be the $(n+1)$-gram $a_1a_2\cdots a_na_{n+1}$ and t_l be the transition matrix $a_2a_3\cdots a_{n+1}a_{n+2}$. Then the entry $\overline{A}_{t_k,t_l}$ captures the conditional probability of a_{n+2} given the letters $a_1, a_2, \ldots, a_{n+1}$. Since we are trying to estimate this probability using the values for the n-th order model, the obvious step would be to calculate this probability under the assumptions of the smaller order model. In this smaller model, $\overline{a}_{t_k,t_l}$ does not depend on the letter a_1 at all, and is the transition probability between the two n-grams formed from t_l : $t_{first} = a_2a_3\cdots a_{n+1}$ and $t_{second} = a_3a_4\cdots a_{n+2}$. Thus we

put

$$\overline{a}_{t_k,t_l} \;=\; a_{t_{first},\, t_{second}}$$

We can also form an estimate for the $(n+1)$-gram probabilities $\overline{\pi}$: We note that this probability is essentially the probability of two consistent n-grams occurring in adjacent positions and we can use $\xi_t(t_i, t_j)$ computed earlier to estimate these:

$$\overline{\pi}(t) = \sum_t \xi_t(t_{first}, t_{second})$$

where the notations t_{first} and t_{second} have the same meaning as before. We call this procedure of estimating larger HMM models from smaller models *bootstrapping*.

7.15 The Ground Truth: Human Performance

We now leave aside our computational architectures of memory and look at human memory organization. Human memory is strongly domain dependent. To see how closely our computational models match human memory, we compare human performance in recalling words from partial cues with our algorithm. These experiments described were performed by Bas Rokers, now at Psychology department of University of California at Los Angeles. Bas also contributed a lot to clarify the roles of the various factors we discuss here: frequency, adjacency and redundancy.

7.15.1 Prior Psychological Studies

In the psychological literature a distinction is being made between word-stem completion and word-fragment completion. In the former a number of letters are given beginning with the first letter(s) of the word. In the latter, the string of letters giving may begin at any other point in the word. Thus a word stem could be the cue "f l o _ _ _", because it can be completed to the word "flower" where the cue occurs at the beginning of the completed word. Word fragment describes a situation where the cue could occur anywhere in the completed word. It could describe either a hangman cue, where the subject knows the length of the completed word and the positions where the letters of the cue occur, as well as superghost cue, where neither is known.

Much of the hypotheses and experimental results in this field come from researchers interested in memory. Psychologists differentiate between two types of memory: *explicit* memory and *implicit* memory. Roughly, an explicit memory system helps in retrieval of information about some experience, while an implicit memory system facilitates a (memory) task in the absence of conscious recollection. There is a general consensus that these different types of memory arise from different systems or mechanisms in the human brain.The book by Shacter and Tulving ([107]) looks at these issues in a greater detail.

Priming

Partly because of this focus, a lot of studies in literature look at *priming effects* in recollection tasks.[3] Priming is an effect where recognition of items is facilitated when a particular item has been encountered before, even when there is not enough time for conscious recognition. It is best explained in a situation of word recognition studies. Thus, the basic task might be to identify a word when shown a fragment very fast. Performance increases on words which have been encountered before.

A lot of psychological literature is concerned with the effects of priming on word completion (for example Graf and Shacter, [50]). However, studies by Hintzman and Hartry ([56]) and by Olofsson and Nyberg ([91]) indicate that priming is usually able to account for only about 5% of the variance in a typical word completion task.

Characteristics of Human Performance

Performance for word-fragment completion has been found lower than word-stem completion (Olofsson & Nyberg, [90]). Additionally words, for which the ending of a word is given, show performance closer to word-stem completion than to word-fragment completion (Olofsson & Nyberg, [91]).

A lot of studies of performance are driven by theories about mental organization for words. It had been suggested earlier that words can be encoded as combinations of syllables, or orthographically, i.e. as likely combinations of letters. Seidenberg ([105]) studies trigrams as sub-lexical structures. This paper indicates that assuming orthographic encoding is in most cases sufficient to describe word completion

[3] Priming effects are the archetypal implicit memory situation.

performance. Srinivas, Roediger and Rajaram ([112]) also come to a similar conclusion.

Specifically, because of orthographic redundancy, low probability trigrams facilitate word completion compared to high probability trigrams. They also found that trigrams where letters are adjacent facilitate word completion more than dispersed ones. And finally they confirmed the effect that specification of word beginning or ending, facilitates word finding over specification of the middle part (see also Olofsson & Nyberg, [90, 91]).

7.15.2 Factors Affecting Recall

Word Length

We also look at how various factors affect human performance. The most obvious, and the primary factor is the length of the cue. Naively, one would expect that the longer the cue, the more information there is about the completed word, and thus the easier the task is. This is certainly true for other fragment completion tasks such as face recognition. The more information is available about the face, the better is the recognition.

However, we find a U-shaped performance profile for word recognition tasks. As in face recognition, recognition is pretty straightforward when most letters are given, or little occlusion is present. However, when only a small amount of letters is given it is easy to generate a word that matches.

However, the evidence is contradictory. Olofsson and Nyberg([90]) failed to find a difference between the performance between two letter and three letter cues. In

this work, the completed words were of length between five and eight letters. However, since the cues were chosen to have a unique completion, these were necessarily cues that were very unfamiliar. Thus the low frequency of completion might have led to this result (11 out of 25 were completed).

It might be that face recognition and word completion are fundamentally different tasks. There are many correct words to generate when only few letters are given. Conversely, there are very few ways to reconstruct a face, to match recollection.

Experiment Design can Skew Results

Other factors can affect the experimental results as well. In particular, we need to be careful that the effect we measure comes about because of innate mechanisms of human memory and not due to some experimental artifact.

Most researchers use a criterion for word completion which corresponds closely to the one used for face recognition. In case of priming experiments, the only correct completion of a fragment is considered to be the one that has been primed before. Even if by itself the word is an allowable completion of the fragment, it is scored as an error (Srinivas et al, [112]). Other researchers eliminate this problem by having a computer generate unique fragments from a large corpus of words (e.g. Olofsson & Nyberg, [90]). That is, the fragment is a unique identifier of a single word within the corpus. This way one is guaranteed not to mix correct, but undesired completions, with genuine errors. However, this leads to fragments with very low probabilities of completion.

Familiarity

One obvious factor affecting performance is the familiarity of the fragment given as the input cue. The more familiar the fragment, the more easily one would expect the subject to be able to complete it. In fact we do observe this phenomenon, most clearly in short cues. Two letter cues, for example, are relatively easy to solve. Indeed, subjects get most of these nearly instantaneously. Olofsson ([89]) found some weak evidence that word familiarity affects persistence of priming over long intervals of time.

We can measure familiarity of a cue by the frequency of its occurrence. The actual algorithm is described below.

7.15.3 Experimental Methods and Results

Methods

In our first experiment frequencies of fragments of lengths ranging from two to eight characters was determined by checking them against a large corpus of words and their frequencies in a large number of texts. This resulted in a large list of fragments, starting at aa and ending at zzzzzzzz and their respective frequencies in the corpus. Any fragment with an occurrence frequency less than 1000 was discarded from the corpus. The occurrence frequency was calculated by summing the frequencies in the corpus of all words that were a valid completion of the fragment.

Presentation of a cue was determined by random selection of the remaining fragments. In the control condition no effort was being made to ensure a uniform distribution based on fragment frequency. In the experimental condition, fragments

were drawn from the list in a way that ensured uniform distribution over frequencies. Fragments of length two usually have a much larger frequency of occurrence in a given corpus than fragments of length eight. This means that the fragments the subject sees on screen are either infrequent short fragments or relatively frequent long fragments (in comparison the other fragments of the same length).

A fragment was presented on a computer screen with spaces interspersed, indicating the possibility of insertion. The subject was required to enter a word that would fit the fragment. A subject was given 60 seconds to produce a completion, but had the possibility to give up. For each session fifty fragments were presented, with an equal amount of fragments from each length.

Reaction time was recorded by measuring the difference from the fragment first appearing on screen until the subject typed the first character of a matching word. Words provided by the subject were checked for internal consistency, e.g. if they were a valid completion of the fragment. Next they were checked for external consistency, e.g. whether the completion was a valid word. To this end words were checked against The American Heritage Dictionary of the English Language ([1]). If there were obvious spelling errors that did not affect a valid completion of the fragment, the spelling was manually corrected before checking against the dictionary.

31 subjects completed the control condition and 31 completed the experimental condition. The subjects were undergraduate students at Rutgers University, participating in the experiment for partial credit. Total time spent on the task varied from 15 minutes to close to one hour.

Results and Discussions

For each graph we plot the number of fragments completed divided by the number of fragments presented. Error bars are calculated as $\sqrt{(p-p^2)/n}$, where p is the percent correct in the sample, and n is the number of trials. This assumes a Bernoulli distribution for correctness of cues with a probability p of being correct. More precise results can be obtained by accounting for between-subject variance, but roughly the same results hold.

Figure 7.1 shows the unadorned data. Note the U-shaped curve. One possible explanation for this shape is that familiarity of the cue, as measured by its occurrence frequency is very high for smaller cues, simply because much more words complete these. As the cue size gets larger, the task again gets simpler, probably due to some other factor.

If this model is correct, when we correct for the frequency, we will get a monotonically increasing graph. In figure 7.2 we plot the results when we picked cues ensuring that their occurrence frequency stayed within a band. As expected, the overall performance is poorer. The graph is flatter as well, which we expected from the above argument. However, there is still a U-shape visible. There is however, not enough data to be sure if this U-shape is significant. i.e. we cannot tell if the difference in performance between the middle and the left end is significant.

To explain the rise in performance as the fragment length increases, we follow two threads. The main idea is to calculate some quantity which approximates the level of difficulty of solving a cue.

In figure 7.3 we plot the results grouped according to *redundancy*. This is a rough

measure of how important each letter is in finding a correct answer to the overall question. More accurately, redundancy of a cue is the average of the redundancy of every position of the cue. Redundancy of a cue at a certain position is the conditional probability that a word satisfies the cue at that position, conditioned on the event that it satisfies the rest of the cue.

Thus, for the superghost cue "p*l*e", the redundancy at the location "l" is the conditional probability that a word satisfies "p*l*e" conditioned on the event that it satisfies "p*e". Thus, it is the ratio of the total frequency of the words satisfying "p*l*e" and the total frequency of the words satisfying "p*e". To compute the redundancy of the cue "p*l*e", we compute the redundancies at "p", "l" and "e" and take the average.

In all cases where there is a significant difference, greater redundancy leads to better performance. In almost all cases, when we control for redundancy performance decreases with length. We will discuss the implications of these experiments after describing corresponding experiments with our model.

7.16 Computational Experiments

Let us now look at how our algorithms perform in practice and compare it to human performance we noted in the previous section.

We have run experiments to compare the performance of this model to that of human subjects. For simplicity, we used a memory of 6,040 words, each with eight characters. In the first experiment we selected cues of varying length that match between four and twenty-two words in the dictionary. Figure 7.4 shows

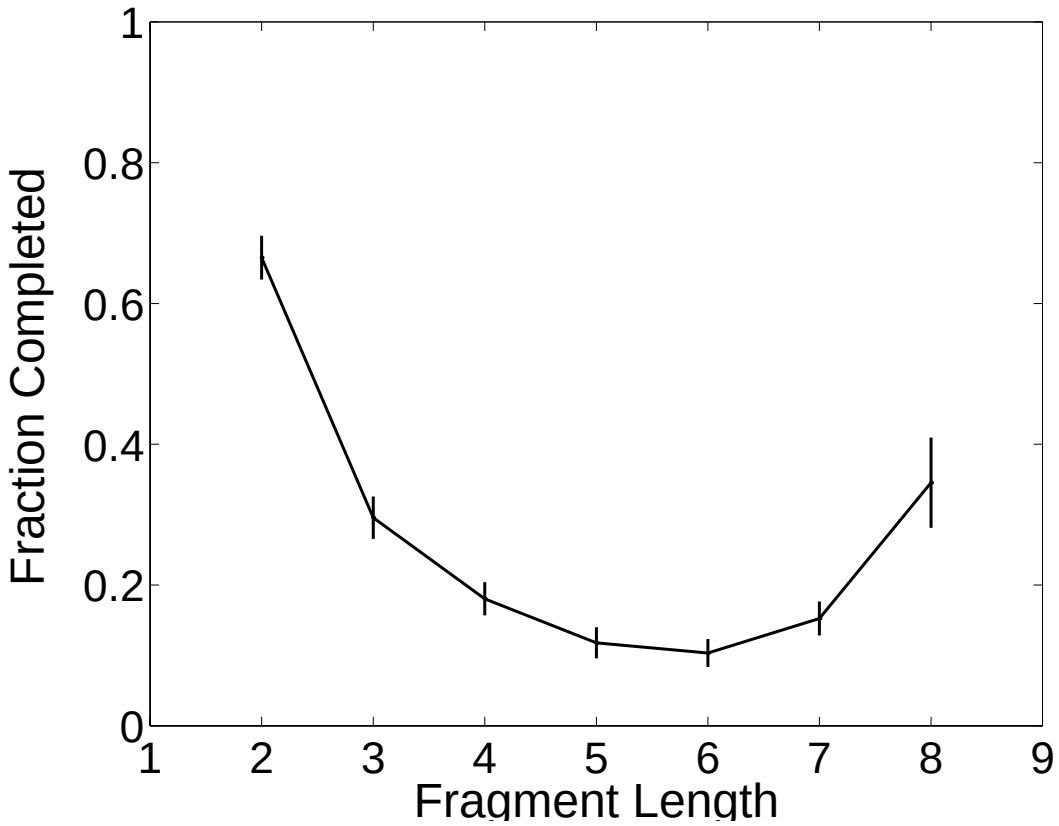

Figure 7.1: Human Performance: Fragment completion as a function of fragment length for randomly chosen cues.

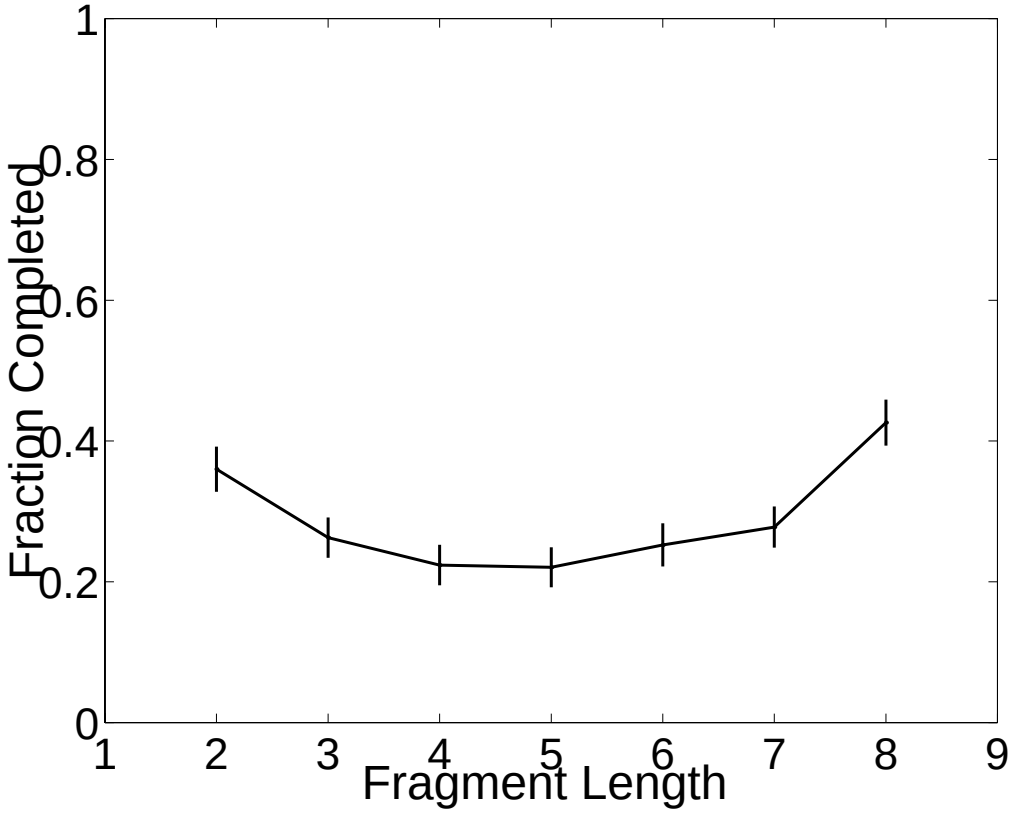

Figure 7.2: Human Performance: Fragment completion as a function of fragment length for cues of equal frequency.

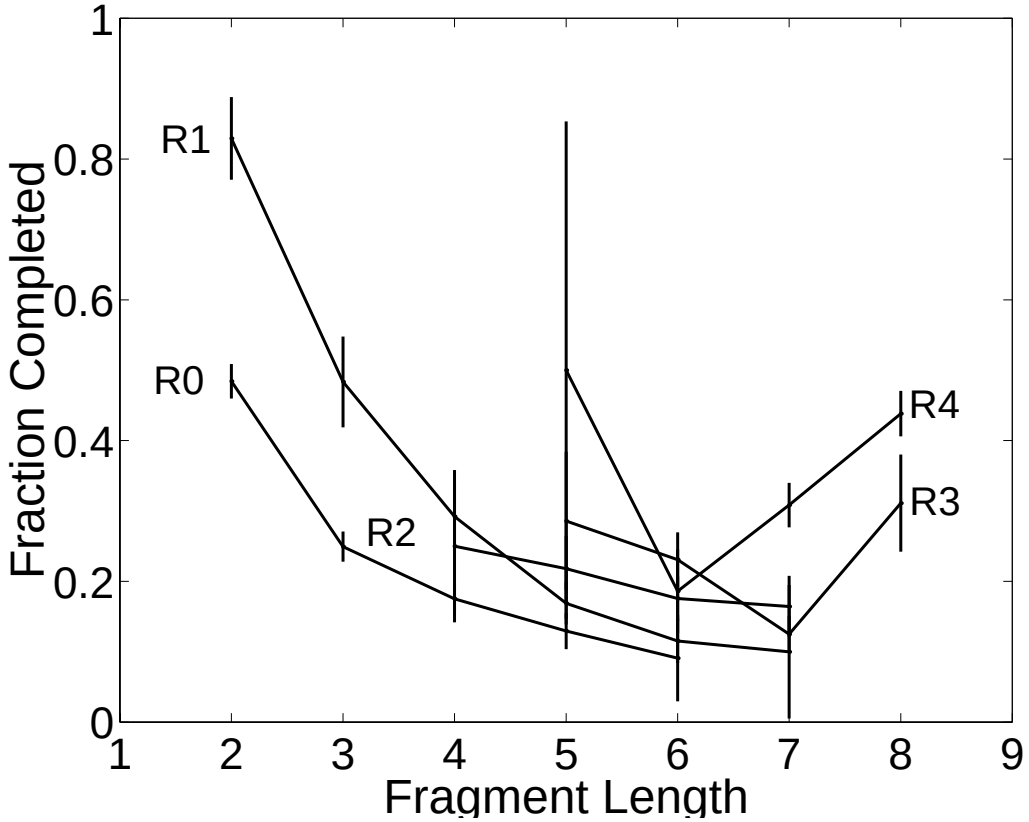

Figure 7.3: Human Performance: Fragment completion as a function of fragment length: the equal frequency cues are divided into five groups, from least redundancy (R0) to most (R4) .

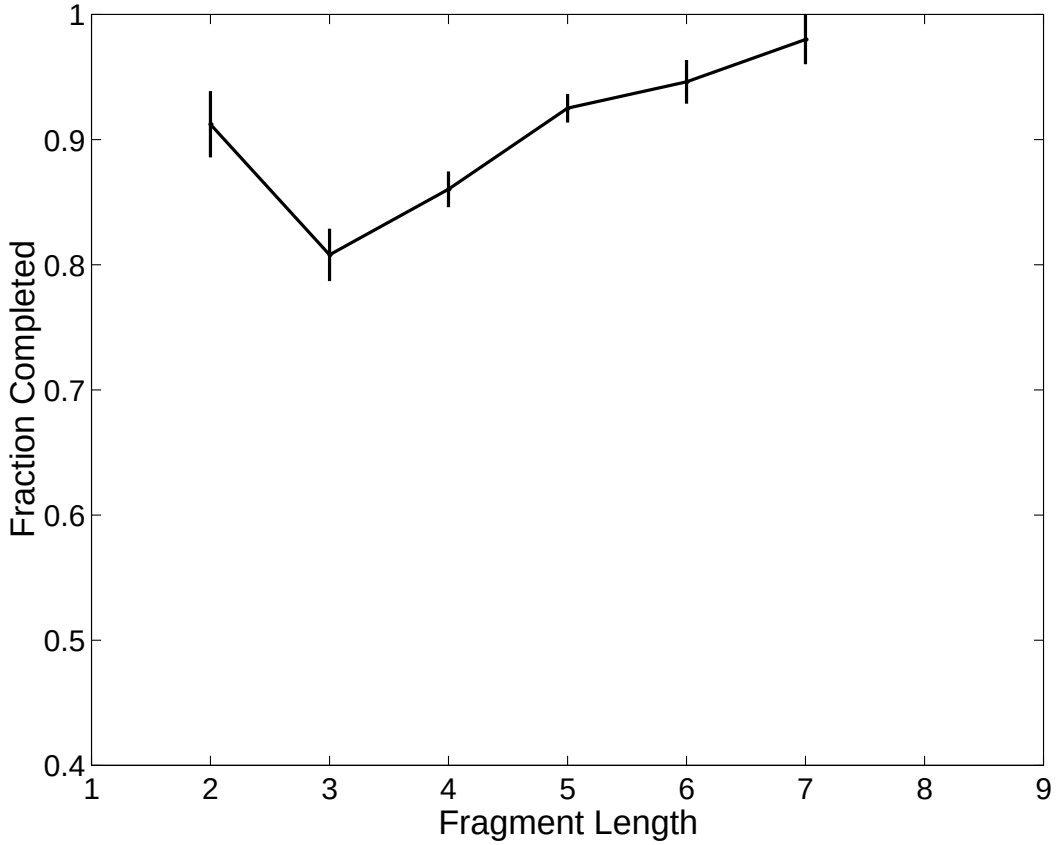

Figure 7.4: Computational performance as a function of cue length, for cues of frequency between 4 and 22.

the percentage of queries the algorithm correctly answered, for cues of lengths two to seven. Error bars show the standard deviation of these numbers, estimated assuming that the correctness of each query is a Bernoulli distribution, assuming the true probability is approximately the percentage of the sample. This figure shows a U-shaped performance curve qualitatively similar to that displayed by human subjects.

We also ran these experiments using cues that matched one to three words. These very low frequency cues did not display this U-shaped behavior. We have not yet been able to examine the question of whether human subjects display similar performance on very low frequency cues.

Next, we divided the cues into five groups of equal size, according to their redundancy. In Figure 7.5 we plot performance for each of these groups. We can see that performance drops monotonically with cue length, when we control for redundancy in this way.

In a final experiment, we simulated the conditions described in Olofsson and Nyberg[90] comparing word stem and word fragment completion. We used a modified algorithm that handled cues in which the number of missing letters can be specified. We found that like the subjects in these experiments, our algorithm performed better at word stem completion. For example, the algorithm correctly answered 87% of cues when the first three letters were given as a cue, but only 69% of cues in which three disconnected letters were provided as a cue.

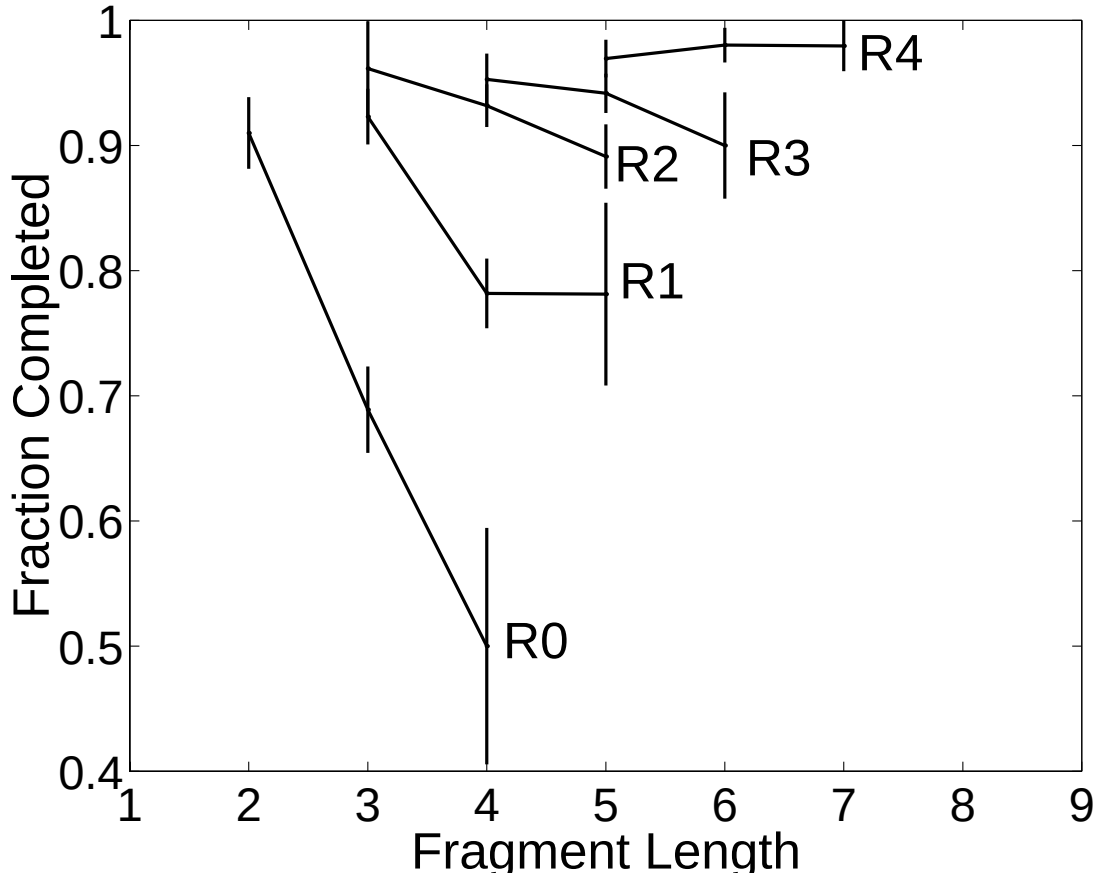

Figure 7.5: Computational performance as a function of cue length, for cues of frequency between 4 and 22. Here, the cues are divided into six groups, according to their redundancy. R0 contains the least redundant cues, R4 contains the most redundant.

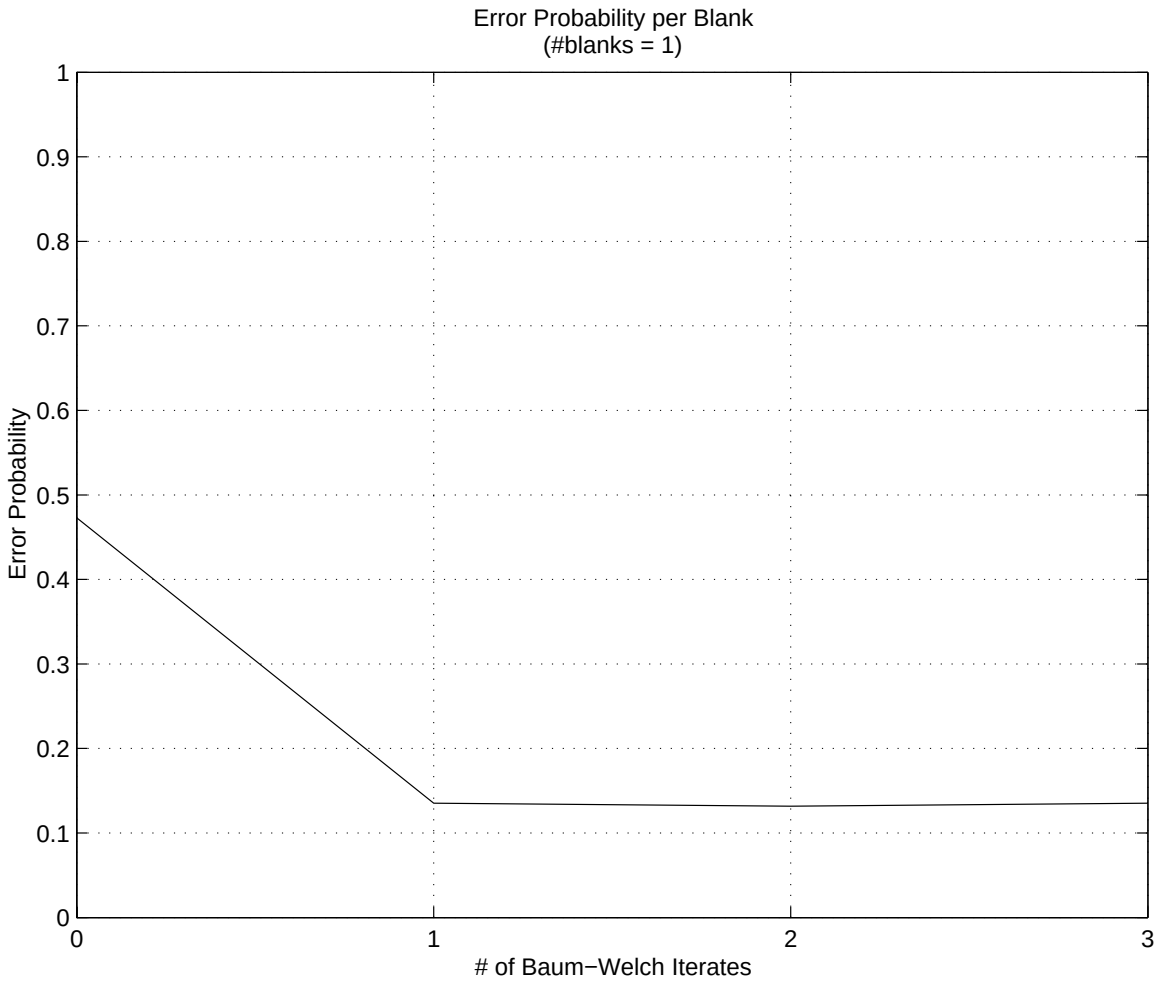

Figure 7.6: Computational performance of the local algorithm: Error Rate per blank for cues with one blank

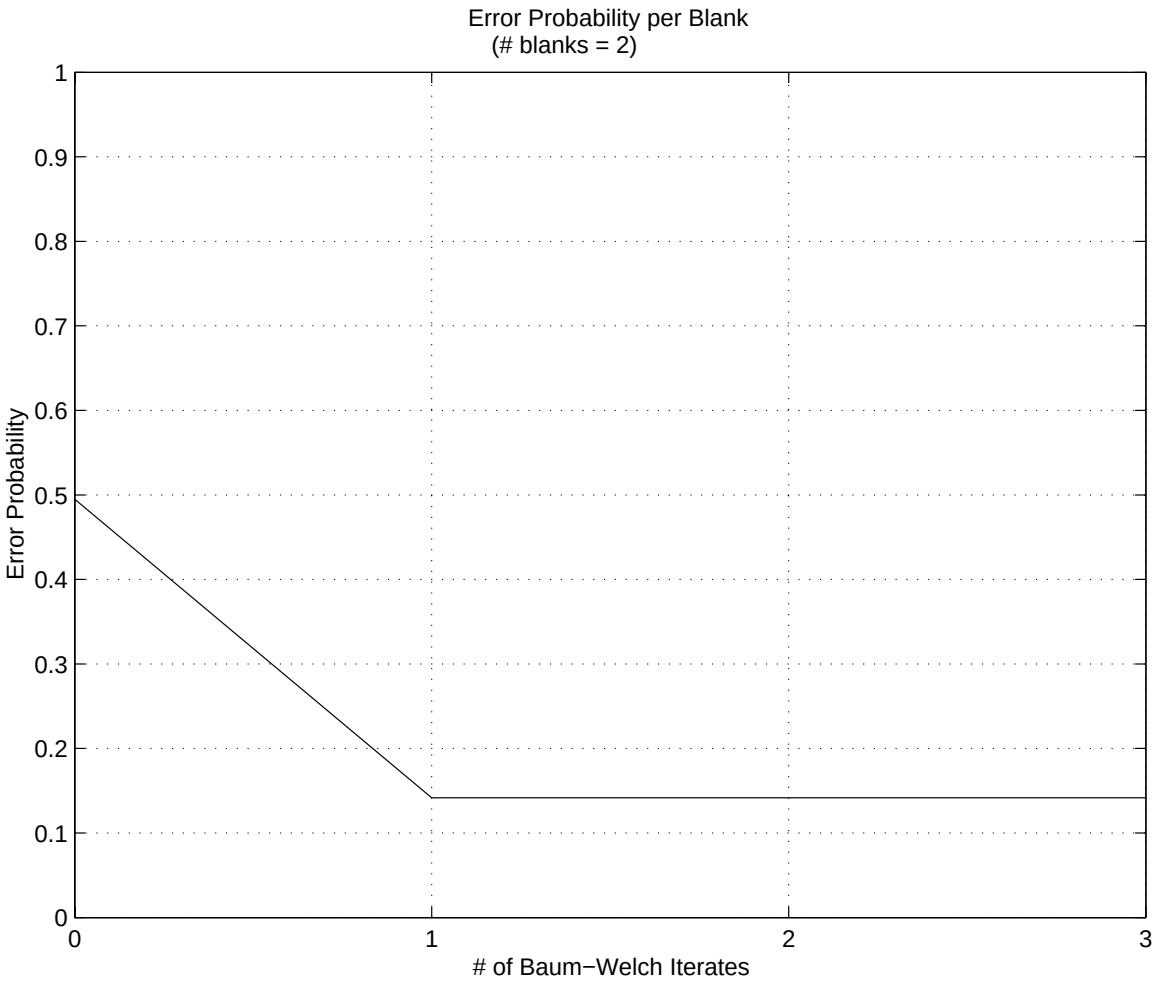

Figure 7.7: Computational performance of the local algorithm: Error Rate per blank for cues with two blanks

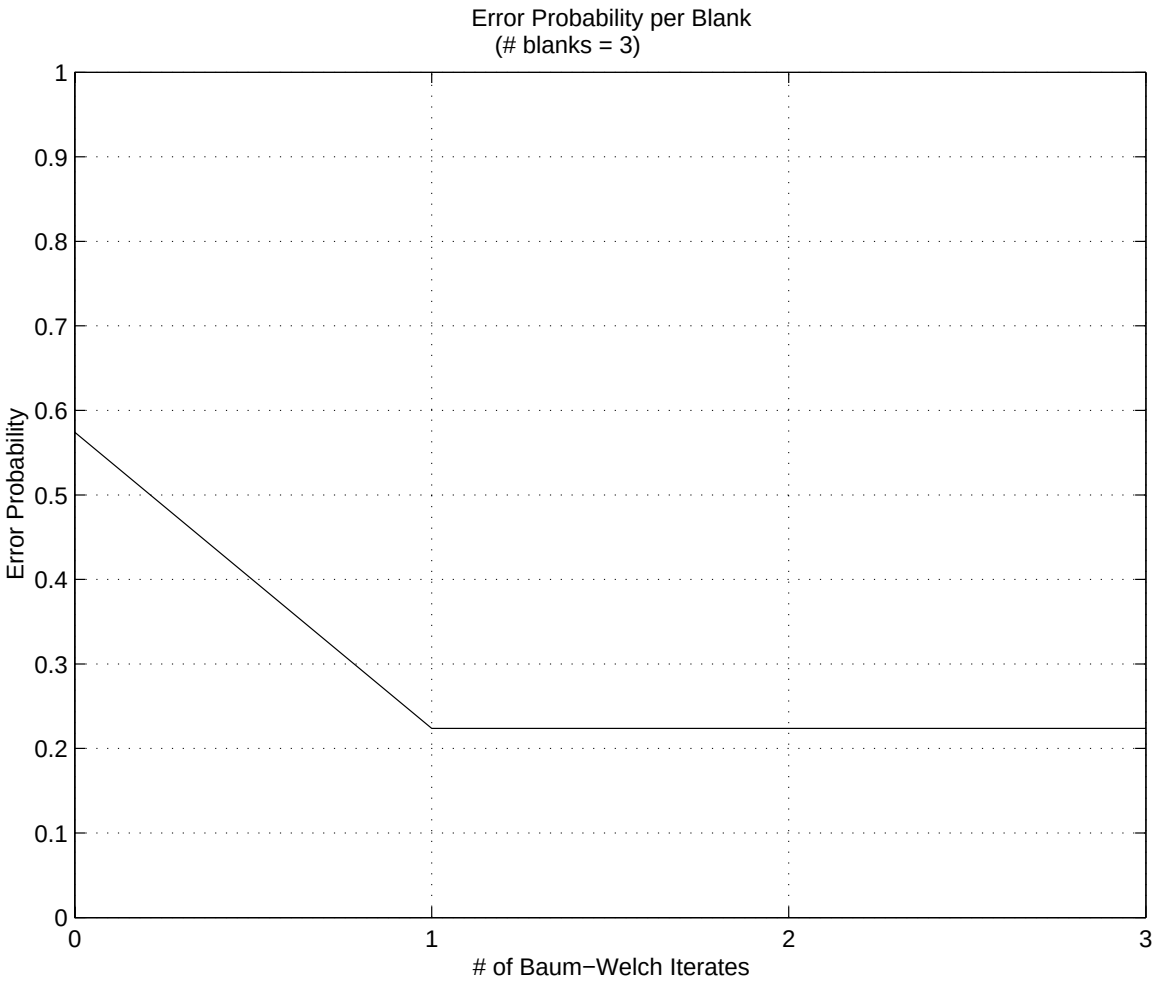

Figure 7.8: Computational performance of the local algorithm: Error Rate per blank for cues with three blanks

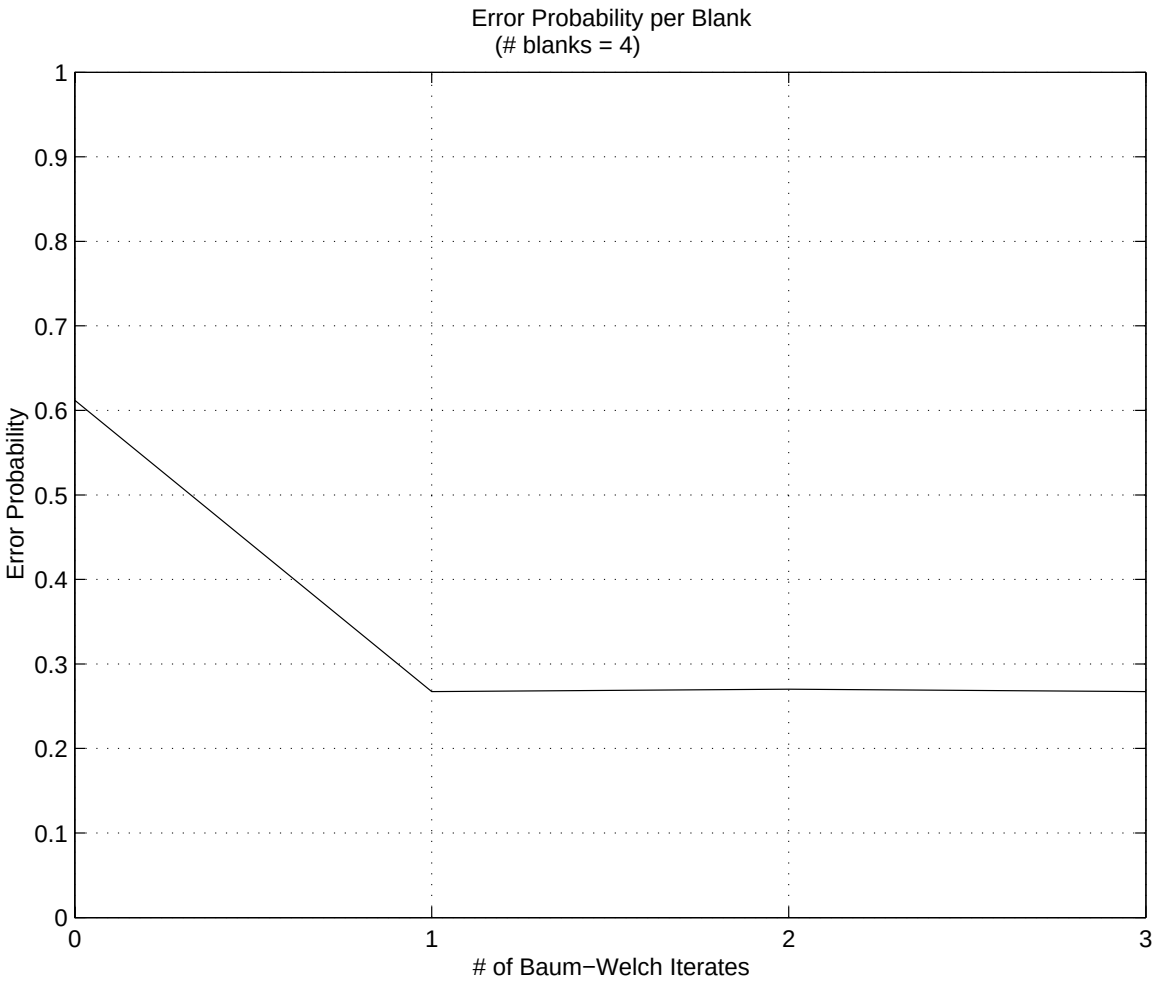

Figure 7.9: Computational performance of the local algorithm: Error Rate per blank for cues with four blanks

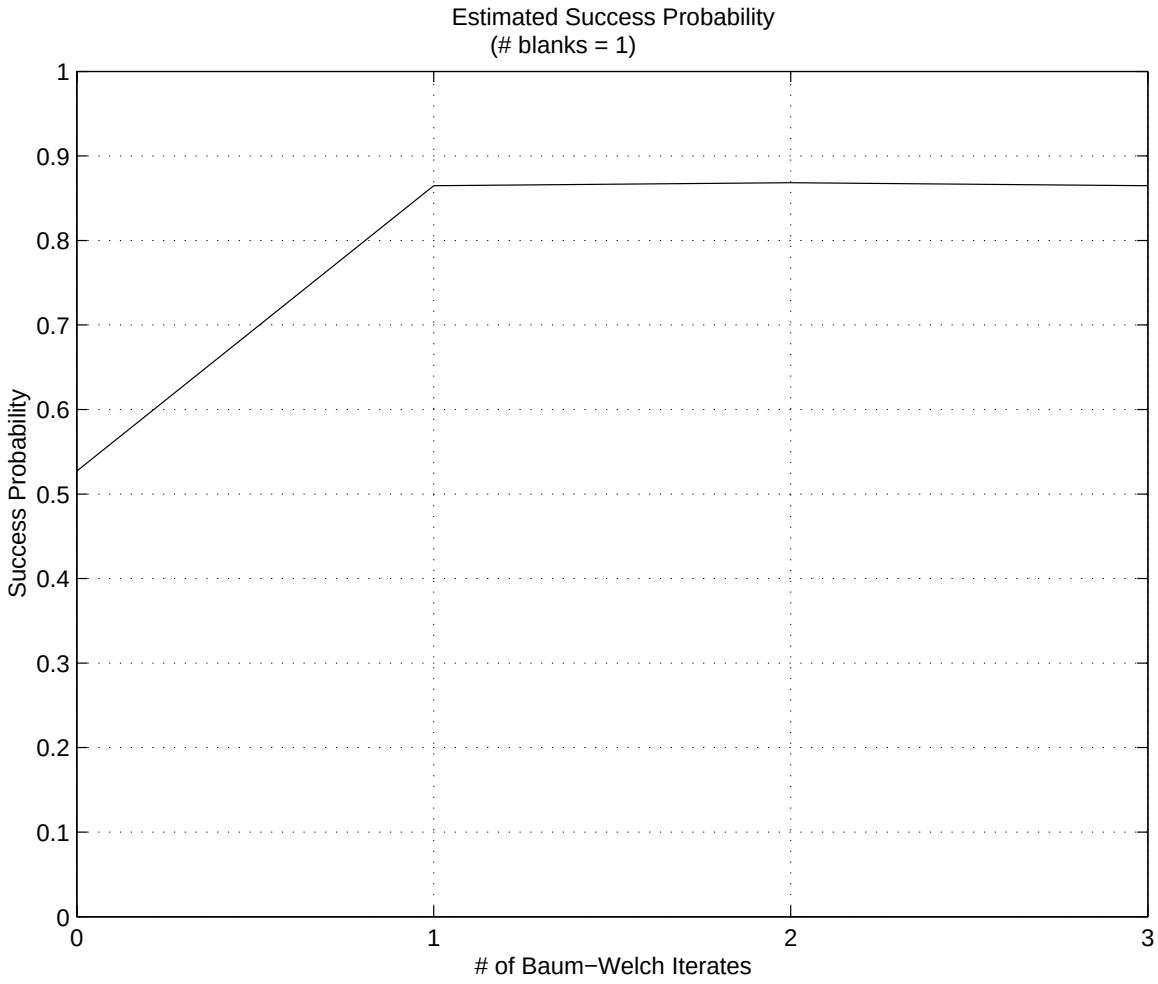

Figure 7.10: Computational performance of the local algorithm: Fraction of Correct Solutions for cues with a single blank

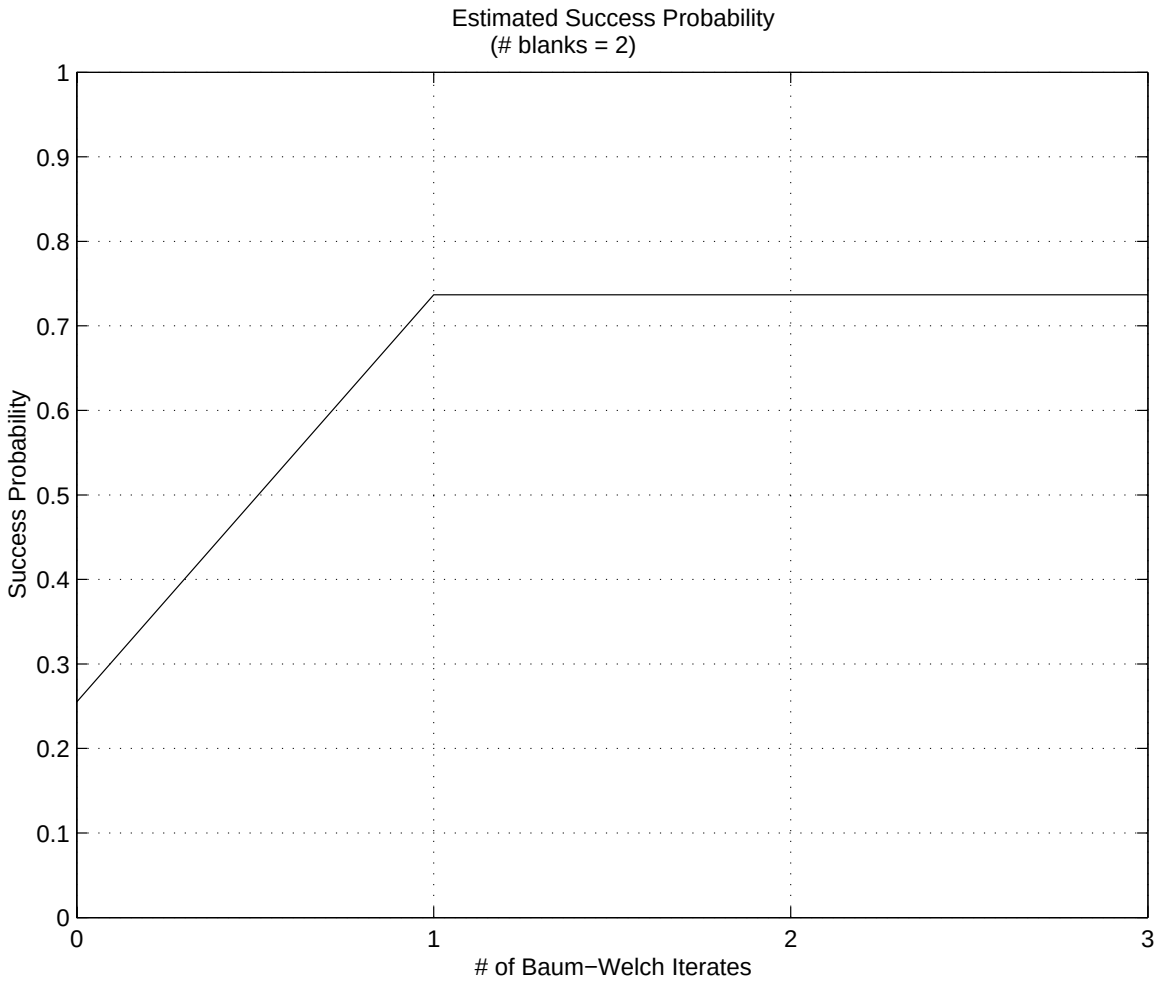

Figure 7.11: Computational performance of the local algorithm: Fraction of Correct Solutions for cues with two blanks

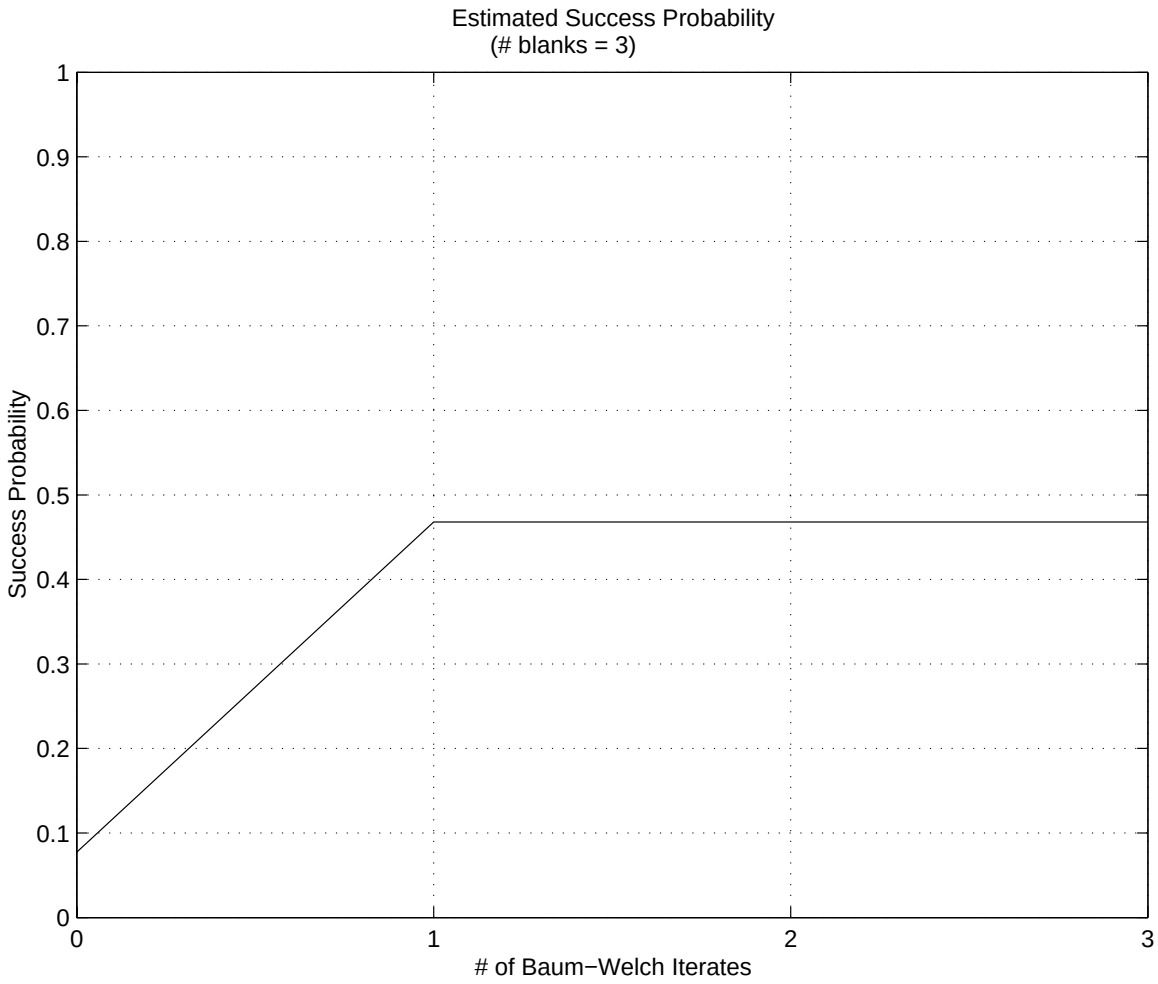

Figure 7.12: Computational performance of the local algorithm: Fraction of Correct Solutions for cues with three blanks

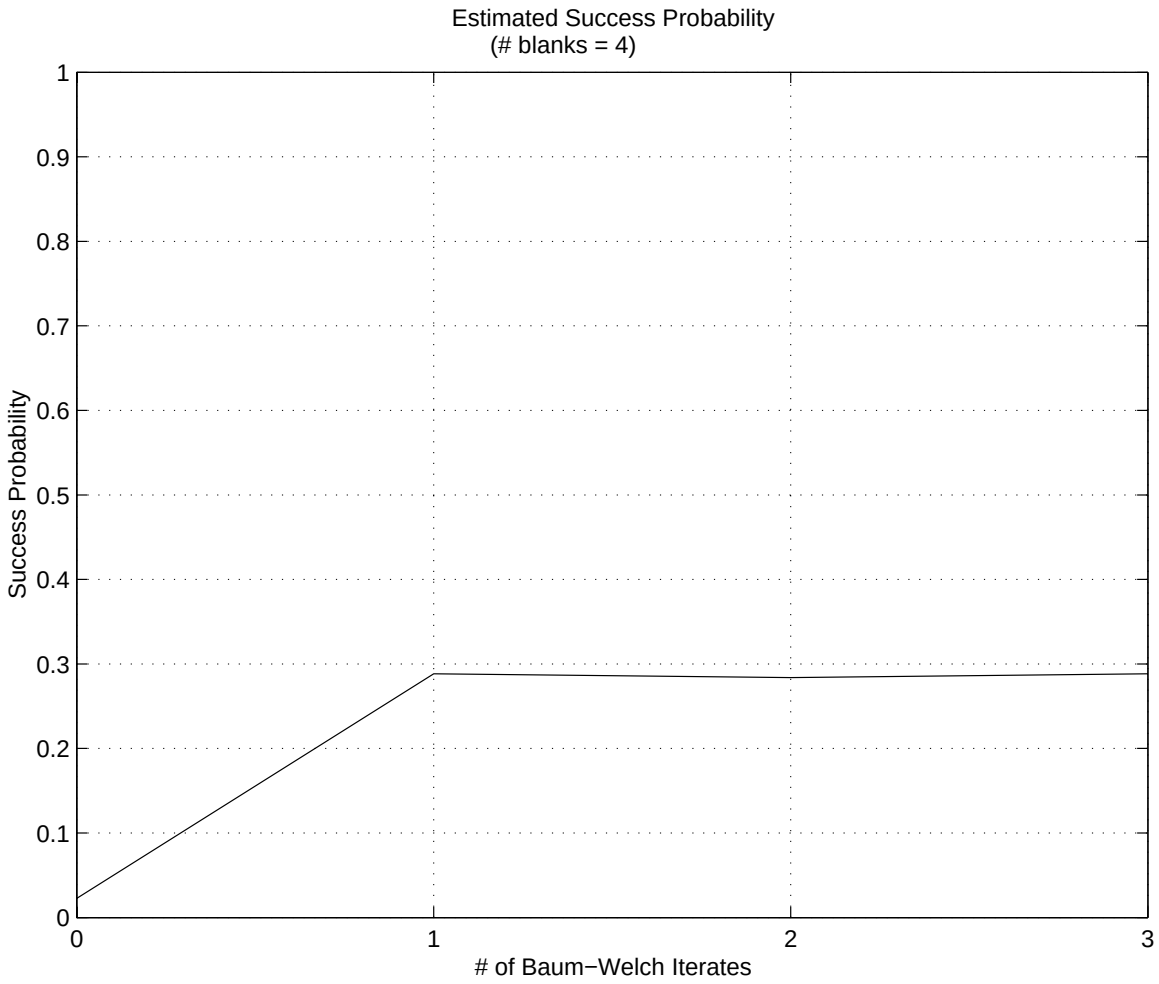

Figure 7.13: Computational performance of the local algorithm: Fraction of Correct Solutions for cues with four blanks

7.16.1 Performance of the Lookup-Free Algorithm

We also look at how much information we glean from the lookup stage of the iterative algorithm. We confine ourselves to hangman clues for simplicity, especially, as we find that elimination of lookups degrade performance by a large amount.

To get a measure of how well the HMM algorithm fills in the blanks, we calculate fraction of errors in filling in blanks. These are shown in figures 7.6 through 7.9 as the number of blanks increases from one to four. The size of the cue is a constant eight. For each value of the number of blanks, we test the performace of the algorithm after various iterations of the Baum-Welch re-estimation scheme. For all practical purposes, we achieve convergence after just one Baum-Welch iteration.

The cues are obtained by starting with a dictionary with all words occurring with a frequency at least 500, and picking all words of length eight. For each word of this list, we generate a random cue with the appropriate number of blanks. For ease of comparison, we present the same cues no matter how many Baum-Welch iteration steps we perform.

To test correctness of the response, we compare the response of the algorithm with the original word and see if they match and, if they do not, compute the mismatch.

To compare these results with the earlier experiments, we assume that errors are made independently in each blank position of the word. Thus, to estimate the probability of a correct response, from the probability p of an erroneous blank, for a cue with n blanks, we use the formula

$$(1-p)^n$$

We plot these values in figures 7.10 through 7.13. Comparison with figure 7.4 shows that there is a large degradation in performance. Thus, we conclude that some sort of global lookup step is essential for this sort of memory task.

We have also investigated whether bootstrapping approaches, such as the one described in section 7.14, enable us to eliminate this global operation. Unfortunately, we found that adding bootstrapping does not change the performance of an HMM which has already converged. This could be a characteristic of the problem domain of words in a large English dictionary. It is possible, that in other problem domains with more information at intermediate scales, a bootstrapping approach will lead to some improvements.

7.17 Discussion

Our experiments indicate two main effects in word memory. First, that performance improves with the redundancy of cues. Second, when we control for this, performance drops with cue length. Since redundancy tends to increase with cue length, this creates two conflicting tendencies that result in a U-shaped memory curve. We conjecture that these factors may be present in many memory tasks, leading to U-shaped memory curves in a number of domains.

In our model, the fact that performance drops with cue length is a result of our use of a simple feature set to mediate matching the cue to words in memory.

This means that not all the information present in the cue is conveyed to items in memory. When the length of a cue increases, but its redundancy remains low, all the information in the cue remains important in getting a correct answer, but the amount of information in the cue increases, making it harder to capture it all with a limited feature set. This can account for the performance of our model; similar mechanisms may account for human performance as well. On the other hand, the extent to which redundancy grows with cue length is really a product of the specific words in memory and the cues chosen. Therefore, the exact shape of the performance curve will also depend on these factors.

Finally, we also point out that our measure of redundancy is rather crude. In particular, it tends to saturate. So if we add a letter to a cue that is already highly redundant, the new letter may not be needed to find a correct answer, but that is not reflected in the new cue having still higher redundancy. This may account for the fact that there is some increase in human performance when length increases for highly redundant cues.

Chapter 8

Conclusion

In this thesis, we investigated the role of features in solving complex tasks. In the first part of the thesis we looked at the feature discovery problem. There, we came up with a novel description of features and illustrated this approach in the context of estimation of one-dimensional signals as well as the case of estimating the disparity map for epipolar line stereo. Our general approach lets us come up with feature selection algorithms for these problems which yield good features. Furthermore, we discovered an approximate method for accomplishing the same task which is extremely quick and which can be easily applied.

In the second part of the thesis we investigated the use of features in different situations. For this purpose we investigated the problem of designing an efficient memory system for retrieving items from partial descriptions of them. We described a new high-level algorithm which lets one capture domain specific information in a uniform way. We applied this algorithm to create memories for faces as well as for words. In both these applications, we showed that it is best to start from a blind

implementation of our high-level algorithm and optimize this algorithm to obtain a more efficient algorithm. The actual optimizations applied might depend on the problem domain, and the complexities of the various matching functions, but the optimized algorithm behaves identically as the high-level algorithm we started out with.

Bibliography

[1] The American Heritage Dictionary of the English Language, Third Edition, Houghton Mifflin Company, 1996.

[2] Y. Amit, D. Geman and K. Wilder, *"Recognizing Shapes from Simple Queries about Geometry"* Tech Report, Univ. of Massachusetts, 1995.

[3] J. Anderson, *An Introduction to Neural Networks*, MIT Press, Cambridge MA, 1995.

[4] B. Anderson, *"The Role of Partial Occlusion in Stereopsis"*, Nature, vol. 367, pp. 365–368, 1994.

[5] N. Ayache, *Artificial Vision for Mobile Robots*, MIT Press, Cambridge, MA, 1991.

[6] L.R. Bahl and F. Jelinek, *"Decoding for Channels with Insertions, Deletions and Substitutions with Application to Speech Recognition"*, IEEE Trans. Informat. Theory, vol. IT-21, pp. 404–411, 1975.

[7] L.R. Bahl, F. Jelinek and R.L. Mercer, *"A Maximum Likelihood Approach to*

Continuous Speech Recognition", IEEE Trans. on PAMI, vol. PAMI-5, pp. 179–190, 1983.

[8] J.K. Baker, *"The Dragon System: an Overview"*, IEEE Trans. Acoust. Speech Signal Processing, vol. ASSP-23. no. 1, pp. 24–29, Feb. 1975.

[9] H. H. Baker and T. O. Binford, *"Depth from Edge and Intensity Based Stereo"* in *Proc. of 7th IJCAI*, vol. 2, pp. 631–636, 1981.

[10] R. Bakis, *"Continuous Speech Word Recognition via centisecond Acoustic States"*, Proc. ASA Meeting, (Washington, DC), Apr. 1976.

[11] R.J. Baron, *"Mechanisms of Human Facial Recognition"* International Journal of Man Machine Studies vol 15, p. 137–178, 1981

[12] L.E. Baum and T. Petrie, *"Statistical Inference for Probabilistic Functions of Finite State Markov Chains"*, Ann. Math. Stat., vol. 37, pp. 1554–1563, 1966.

[13] L.E. Baum and J.A. Egon, *"An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology"*, Bull. of Amer. Meteorological Soc., vol. 73, pp. 360–363, 1967.

[14] L.E. Baum and G.R. Sell, *"Growth Functions for Transformations on Manifolds"*, Pacific Jour. Math., vol. 27, no. 2, pp. 211–227, 1968.

[15] L.E. Baum, T. Petrie, G. Soules and N. Weiss, *"A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains"*, Ann. Math. Stat., vol. 41, no. 1, pp. 164–171, 1970.

[16] L.E. Baum, *"An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes"*, Inequalities, vol. 3, pp. 1–8, 1972.

[17] L. E. Baum, J. Moody and F. Wilczek, *"Internal Representations for Associative Memory"*, Biological Cybernetics, **59**, pp. 217–228, 1988.

[18] P. N. Belhumeur, J. P. Hespanha and D. J. Kriegman, *"Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection"*, Proc. ECCV, pp. 45–58, 1996.

[19] P. N. Belhumeur and D. Mumford, *"A Bayesian Treatment of the Stereo Correspondence Problem using Half-occluded Regions"*, Proceedings of IEEE conference on CVPR, pp. 506–512, 1992.

[20] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957

[21] R. Bellman, *"On a Routing Problem"*, Quaterly of Applied Mathematics, vol. 16, pp. 87–90, 1958.

[22] A.L. Berger, S.A. della Pietra and V.J. della Pietra, *"A Maximum Entropy Approach to Natural Language Processing"*, Computational Linguistics, vol. 22 no. 1, pp. 39–71, March 1996.

[23] M. Bischel, *"Strategies of Robust object Recognition for the Identification of Human Faces"* Ph.D. thesis, Eidgenossischen Technischen Hoschule, Zurich, 1991

[24] A. Blake and A. Zisserman. *Visual Reconstruction.* MIT Press, Cambridge, Mass, 1987.

[25] W. W. Bledsoe, *"Man-machine Facial Recognition"*, Tech Report PRI:22, Panoramic Research Inc., Palo Alto, CA 1966

[26] Y. Boykov, O. Veksler and R. Zabih, *"Fast Approximate Energy Minimization via Graph Cuts"*, in *International Conference on Computer Vision*, pp. 377–384, 1999.

[27] R. Brunelli and T. Poggio, *"Face Recognition: Features versus Templates"*, IEEE Trans on PAMI vol. 15, no. 10, pp. 1042–1052, 1993.

[28] J. Buhmann, J. Lange and C. von der Malsburg, *"Distortion Invariant Object Recognition by Matching Hierarchically Labelled Graphs"* in Proc. IJCNN, 1989, pp. 151–159.

[29] D.J. Burr, *"Elastic Matching of Line Drawings"*, IEEE Trans on PAMI, vol 3, no. 6, pp. 708–713, 1981.

[30] J.F Canny. *"A computational approach to edge detection"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679–698, 1986.

[31] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley Interscience, New York, 1991.

[32] B. Cernushi-Frias, D. B. Cooper, Y. P. Huang and P. Belhumeur, *"Towards a Model-based Bayesian Theory for Estimating and Recognizing Parameterized*

3D Objects using Two or More Images Taken from Different Positions", IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI, vol. 11, pp. 1028–1052, 1989.

[33] V. Cerny *"A Thermodynamical Approach to the Travelling Salesman Problem: an Efficient Simulation Algorithm"* in Preprint Inst. Phys. & Biophys., Comenius Univ., Bratislava, 1982

[34] J.S. DeBonet *"Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images"*, in *ACM SIGGRAPH*, pages 361–368, Aug 1997

[35] A.P. Dempster, N.M. Laird and D.B. Rubin, *"Maximum Likelihood from Incomplete Data via the EM Algorithm"*, Journ. Royal Statistical Society, vol. 39, no. 1, pp. 1–38, 1977.

[36] V. Digalakis, J.R. Rohlicek and M. Ostendorf, *"ML Estimation of a Stochastic Linear System with the EM Algorithm and its Application to Speech Recognition"*, IEEE Trans. on Speech and Audio Processing, vol. 1, no. 4, pp. 431–442, 1993.

[37] E. Dijkstra, *"A Note on Two Problems in Connexion with Graphs"*, NumerischeMathematik vol, 1, pp. 269–271, 1959.

[38] S. Edelman, D. Reisfeld and Y. Yeshurun, *"Learning to Recognize Faces from Examples"*, in *Proc. of ECCV*, pp. 787–791, 1992

[39] A.A. Efros and T.K. Leung *Texture Synthesis by Non-parametric Sampling*

in *International Conference on Computer Vision*, vol 2, pages 1033–1038, Sep 1999

[40] J. M. Eich, *"A Composite Holographic Associative Recall Memory"*, Psychological Review, vol. 89, no. 6, pp. 627–661, Nov. 1982.

[41] D. Field, A. Hayes and R. Hess *Contour Integration by Human Visual System: evidence for a local "Association Field"* in *Vision Res.* Vol 33, No 2, pp 173 – 193, 1993.

[42] L. R. Ford Jr., *"Network Flow Theory"*, Paper P-923, RAND Corporation, Santa Monica, California, 1956.

[43] G.D. Forney, *"The Viterbi Algorithm"*, Proc. IEEE, vol. 61, pp. 268–278, 1973.

[44] D. Geiger and F. Girosi. *"parallel and deterministic algorithms for mrfs: surface reconstruction"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 5, pp. 401–412, May 1991.

[45] D. Geiger and J. E. Kogler. *"Scaling Images and Image Feature via the Renormalization Group"*, in *Proc. IEEE Conf. on Computer Vision & Pattern Recognition* , New York, NY, 1993.

[46] D. Geiger and B. Ladendorf and A. Yuille, *"Binocular stereo with occlusion"*, in *2nd ECCV-Lecture Notes on Computer Science*, Springer-Verlag, May 1992.

[47] D. Geiger, A. Rudra and L. Maloney, *"Features as Sufficient Statistics"*, in Advances in Neural Information Systems, 10, M.I. Jordan, M.J.Kearns and S.A. Solla eds., MIT Press, 1998.

[48] S. Geman and D. Geman *"Stochastic Relaxation, Gibbs distributions, and the Bayesian Restoration of Images"* in *IEEE Trans. on PAMI*, PAMI-6, pp. 721–741, 1984.

[49] A.J. Goldstein, L.D. Harmon and A.B. Lesk, *"Identification of Human Faces"*, Proc. IEEE, vol 59, p. 748, 1971

[50] P. Graf and D. L. Shacter, *"Implicit and Explicit Memory for New Associations in Normal and Amnesic Subjects"*, Journal of Experimental Psychology: Learning, Memory and Cognition, vol. 11, n0. 3, pp. 501-518, 1985.

[51] W. E. L. Grimson. *From Images to Surfaces.* MIT Press, Cambridge, Mass., 1981.

[52] S. Grossberg and E. Mingola *"Neural Dynamics of Prceptual Grouping: Textures, boundaries and emergent segmentations"* in *Perception and Psychophysics*, 38(2) 141 – 171, 1985.

[53] P.R. Halmos and L.J. Savage, *"Application of the Radon-Nikodym Theorem to the Theory of Sufficient Statistics"*, Annals of Mathematical Statistics, vol. 20, no. 2, pp. 225–241, June 1949.

[54] G. Hinton, P. Dayan, B. Frey and R. Neal, *"The 'Wake-Sleep' Algorithm for Unsupervised Neural Networks"*, Science, vol. **268**, pp. 1158–1161, 1995.

[55] G. Hinton and Z. Ghahramani. *"Generative Models for Discovering Sparse Distributed Representations"*, in *Phil. Trans. of the Royal Society* **B**, 1997.

[56] D.L. Hintzman and A.L. Hartry, *"Item Effects in Recognition and Fragment Completion: Contingency Relations Vary for Different Sets of Words"*, Journal of Experimental Psychology: Learning, Memory and Cognition, vol. 17, pp. 341–345, 1990.

[57] J. Hopfield, *"Neural networks and Physical Systems with Emergent Collective Computational Abilities"*, Proc. of the Nat. Acad. of Science, vol. 79, pp. 2554–2558, 1982.

[58] D. A. Huffman, *"A method for the construction of minimum-redundancy codes"*, Proceedings of the IRE, vol. 40 no. 9, pp. 1098–1101, 1952.

[59] H. Ishikawa and D. Geiger, *"Occlusions, Discontinuities, and Epipolar Lines in Stereo"*, in *Fifth European Conference on Computer Vision*, pp. 232–248, 1998.

[60] D. Jacobs, B. Rokers, A. Rudra and Z. Liu, *"Fragment Completion in Humans and Machines"*, in *Advances in Neural Information Processing Systems, 14*, MIT Press, Cambridge, MA, to appear (2003).

[61] F. Jelinek, *"A Fast Sequential Decoding Algorithm using a Stack"*, IBM J. Res. Develop., vol. 13, pp. 675–685, 1969.

[62] F. Jelinek, *"Continuous Speech Recognition by Statistical Methods"*, Proc. IEEE, vol. 64, pp. 532–536, 1976.

[63] F. Jelinek, L.R. Bahl and R.L. Mercer, *"Design of a Linguistic Statistical*

Decoder for the Recognition of Continuous Speech", IEEE Trans. Informat. Theory, vol. IT-21, pp. 250–265, 1975.

[64] F. Jelinek, L.R. Bahl and R.L. Mercer, *"Continuous Speech Recognition: Statistical Methods"*, in *Handbook of Statistics*, P.R. Krishnaiad, Ed. Amsterdam, the Netherlands: North-Holland, 1982.

[65] F. Jelinek and R.L. Mercer, *"Interpolated Estimation of Markov Source Parameters from Sparse Data"*, in *Pattern Recognition in Practice*, E.S Gelesma and L.N. Kanal, Eds., Amsterdam, The Netherlands: North-Holland, pp. 381–397, 1980.

[66] G.V. Jones, *"Fragment and Schema Models for Recall"*, Memory and Cognition, vol. 12, no. 3, pp. 250–263, 1984.

[67] B. Julesz, *Foundations of Cyclopean Perception*, University of Chicago Press, Chicago, 1971.

[68] T. Kanade, *"Picture Processing by Computer Complex and Recognition of Human Faces"*, Tech Report, Kyoto University, Dept. of Information Science, 1973.

[69] T. Kanade and M. Okutomi, *"A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiments"*, in *Proc. Image Understanding Workshop, DARPA*, Pennsylvania, Sept. 1990.

[70] M. Kass, A. Witkin and D. Terzopoulos, *"Snakes: Active Contour Models"*, Proc. First Intl. Conf. on Comp. Vision, pp. 259–268, England, 1987

[71] Y. Kaya and K. Kobayashi, *"A Basic Study on Human Face Recognition"*, in *Frontiers of Pattern Recognition* (S. Watanabe ed.), p. 265, 1972

[72] M. Kirby and L. Sirovich, *"Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces"* IEEE Trans. on PAMI, vol. 12, no. 1, Jan 1990.

[73] S. Kirkpatrick, C.D. Gellatt, Jr. and M.P. Vecchi, *"Optimization by Simulated Annealing"*, IBM T.J. Watson Research Center, Yorktown Heights, NY, 1982

[74] J. M. Kleinberg, *"Two Algorithms for Nearest Neighbor Search in High Dimensions"*, in *ACM Symposium on Theory of Computing*, pp. 599–608, 1997.

[75] B.O. Koopman, *"On Distributions Admitting a Sufficient Statistic"*, Trans. of Amer. Math. Soc., vol. 39, pp. 399–409, 1936.

[76] B. Kosko, *"Adaptive Bidirectional Associative Memory"*, Applied Optics, vol. 26, no.23, pp. 4947–4960, 1987.

[77] S. Kullback, *"Information Theory and Statistics"* Wiley, New York, 1959.

[78] J.V. Linnik, *"Statistical Problems with Nuisance Parameters"*, Translations of Mathematical Monographs, vol. 20, American Mathematical Society, 1966.

[79] R. Linsker, *"Self-Organization in a Perceptual Network"*, Computer, pp. 105–117, March 1988,

[80] J. Malik, *"On Binocularly Viewed Occlusion Junctions"*, in *Fourth European*

Conference on Computer Vision, vol. 1, pp. 167–174, Springer Verlag, Cambridge, UK, 1996.

[81] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, W.H. Freeman and Company, San Francisco, 1982.

[82] D. Marr and T. Poggio, *"Cooperative computation of stereo disparity"*, Science, vol. 194 pp. 283–287, 1976.

[83] D. Marr and T. Poggio, *"A computational theory of human stereo vision"*, Proceedings of the Royal Society of London B, vol. 204, pp. 301–328, 1979.

[84] D. Mumford *"Elastica and Computer Vision"*, in *Algebraic Geometry and Applicaltions*, ed. Chandrajit Bajaj, New York, Springer-Verlag, 1994.

[85] D. Mumford and J. Shah, *"Boundary detection by minimizing functionals, i."*, in *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, San Francisco, CA, 1985.

[86] B. B. Murdoch Jr., *" A Theory for the Storage and Retrieval of Item and Associated Information"*, Psychological Review, vol. 89, no. 6, pp. 609–626, Nov. 1982.

[87] K. Nakayama and S. Shimojo, *"Da Vinci Stereopsis: Depth and Subjective Occluding Contours from Unpaired Image Points"*, Vision Research, vol. 30, pp. 1811–1825, 1990.

[88] Y. Ohta and T. Kanade, *"Stereo by Intra- and Inter-scanline Search using Dynamic Programming"*, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 7, no. 2, pp. 139–154, 1985.

[89] U. Olofsson, *"Retention Interval, Response Competition and Word Familiarity Effects in Primed Fragment Completion"*, European Journal of Cognitive Psychology, vol. 7, no. 2, pp. 131–143, 1995.

[90] U.Olofsson and L. Nyberg, *"Swedish Norms for Completion of Word Stems and Unique Word Fragments"*, Scandinavian Journal of Psychology, vol. 33, no. 2, pp. 108–116, 1992.

[91] U.Olofsson and L. Nyberg, *"Determinants of Word Fragment Completion"*, Scandinavian Journal of Psychology, vol. 36, no. 1, pp. 59–64, 1995.

[92] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufman, 1988.

[93] A. Pentland, B. Moghaddam and T. Starner, *"View-based and Modular Eigenspaces for Face Recognition"*, in *Proc. Computer Vision and Pattern Recognition*, pp.84–91, 1994.

[94] R. Pike, *"Comparison of Convolutional and Matrix Distributed Memory Systems for Associative Recall and Recognition"*, Psychological Review, vol. 91, no. 3, pp. 281–294, July 1984.

[95] E.J.G. Pitman, *"Sufficient Statistics and Intrinsic Accuracy"*, Proc. Camb. Phil. Soc., vol. 32, pp. 567–579, 1936.

[96] S. B. Pollard, J. E. W. Mayhew and J. P. Frisby, *"Disparity Gradients and Stereo Correspondences"*, Perception, 1987.

[97] L.R. Rabiner, *"A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"*, Proc. IEEE, vol. 77, no. 2, pp. 257–286, 1989.

[98] R. Rao and D. Ballard, 1995. *"Object Indexing Using an Iconic Sparse Distributed Memory"*, IEEE Int. Conf. on Comp. Vis. , pp. 24–31.

[99] R. Rao and D. Ballard, *"Dynamic Model of Visual Recognition Predicts Neural Response Properties in the Visual Cortex"*, Neural Computation, vol. 9, no. 4, pp. 721–763, 1997.

[100] R. Rao and D. Ballard, *"Natural Basis Functions and Topographic Memory for Face Recognition"*, in *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 10–17, 1995.

[101] R.H. Ross and G.H. Bower, *"Comparisons of Models of Associative Recall"*, Memory and Cognition, vol. 9, no. 1, pp. 1–16, 1981.

[102] S. Roweis and Z. Ghahramani, *"A Unifying Review of Linear Gaussian Models"*, Neural Computation, vol. 11, pp. 305–345, 1999.

[103] S. Roy and I. Cox, *"A Maximum Flow Formulation of the N-camera Stereo Correspondence Problem"*, in *Proc. Intl. Conf. on Computer Vision, ICCV '98*, Bombay, India, 1998.

[104] T. Sanger, *"Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network"*, Neural Networks, vol. 2, pp. 459–473, 1989.

[105] M.S. Seidenberg, *"Sublexical Structures in Visual Word Recognition: Access Units or Orthographic Redundancy?"*, in *Attention and Performance XII*, M. Coltheart, Ed., Hillsdale, NJ, Erlbaum, pp. 245–263, 1987.

[106] T. Sejnowski, *"Computational Models and the Development of Topographic Projections"*, Trends Neuroscience, vol.10, pp. 304–305, 1987.

[107] D.L. Shacter and E. Tulving, *Memory Systems*, MIT Press, Cambridge, MA, 1994.

[108] C. E. Shannon *A Mathematical Theory of Communication* in *Bell Sys. Tech. Journal*, 27: 379–423, 623–656, 1948

[109] J. Shi and J. Malik *"Normalized Cuts and Image Segmentation"*, in *Conf. in Comp. Vision and Pattern Recog*, pp 731–737, San Juan, 1997

[110] F. T. Sommer, and G. Palm, *"Bidirectional Retrieval from Associative Memory"*, in *Neural Information Processing Systems, 1997*, pp. 675 – 681.

[111] G. Sperling, *"Binocular Vision: A Physical and Neural Theory"*, American Journal of Psychology, vol. 83, pp. 461–534, 1967.

[112] K. Srinivas, H.L. Roediger (3rd) and S. Rajaram, *"The Role of Syllablic and Orthographic Properties of Letter Cues in Solving Word Fragments"*, Memory and Cognition, vol. 20, no. 3, pp. 219–230, 1992.

[113] A. N. Tikhonov, *"Solution of Incorrectly Formulated Problems and the Regularization Method"*, Soviet Math., vol. 4, pp. 1035–1038, 1963.

[114] N. Tishby, F. Pereira and W. Bialek, *"The Information Bottleneck Method"*, 37th Allerton Conference on Communication, Control, and Computing, 1999.

[115] M. Turk, and A. Pentland, *"Eigenfaces for Recognition"*, Journal of Cognitive Neuroscience, vol. 3, pp. 71–86, 1991.

[116] S. Ullman, *High-level Vision*, MIT Press, Cambridge, MA. 1996.

[117] V. Vapnik, *Estimation of Dependences Based on Empirical Data*, Translated by Samuel Kotz, Springer Series in Statistics, Springer-Verlag, 1982

[118] P. A. Viola, *"Alignment by Maximization of Muutal Information"*, MIT Artificial Intelligence Laboratory, *Ph.D. thesis*, Cambridge, MA, March 1995.

[119] A. J. Viterbi, *"Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm"*, IEEE Trans. on Information Theory, vol. 13, pp. 260–269, 1967.

[120] L. Wei and M. Levoy *"Fast Texture Synthesis using Tree-structured Vector Quantization"*, in *ACM SIGGRAPH*, pages 479–488, Jul 2000.

[121] S. S. Wilks, *Mathematical Statistics*, Princeton Univ. Press, 1943.

[122] L. Williams and D. Jacobs *"Stochastic Completion Fields: A Neural Model of Illusory Contour Shape and Salience"*, in *Neural Computation*, vol. 9, no. 4, pp. 837–858.

[123] A.L. Yuille, *"Deformable Templates for Face Recognition"* J. Cognitive Neuroscience, vol. 3, no. 1, pp. 59–70, 1991.

[124] A. Yuille, D. Geiger and H. Bulthoff, *"Stereo, Mean Field Theory and Psychophysics"*, in *1st ECCV*, pp. 73–82, Antibes, France, Apr. 1990, Springer-Verlag.

[125] S.C. Zhu, Y. N. Wu and D. Mumford *"FRAME: Filters, Random Fields, and Minimax Entropy, Towards a Unified Theory of Texture Modeling"*, in *Proc. CVPR*, San Francisco, 1996.

[126] S.C. Zhu, Y. N. Wu and D. Mumford *"Filters, Random Fields, and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling"*, in *International Journal of Computer Vision*, vol. 27, no. 2, pp. 107–126, 1998.