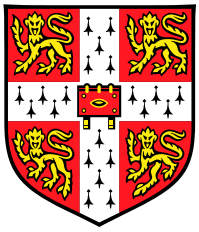
Adaptive Computer Interfaces

David J. Ward
Churchill College
Cambridge

A dissertation submitted in candidature for the degree of Doctor of Philosophy,
University of Cambridge

Inference Group
Cavendish Laboratory
University of Cambridge



November 2001

Declaration

I hereby declare that my dissertation entitled "Adaptive Computer Interfaces" is not substantially the same as any that I have submitted for a degree or diploma or other qualification at any other University.

I further state that no part of my dissertation has already been or is being concurrently submitted for any such degree or diploma or other qualification.

Except where explicit reference is made to the work of others, this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration. This dissertation does not exceed sixty thousand words in length.

Date: ..... Signed: .....

David Ward
Churchill College
Cambridge
November 30th, 2001

Abstract

Existing devices for communicating information from people to computers are either bulky, slow to use, or unreliable. The conventional keyboard requires the user to make one or two gestures per character (one for lower-case and two for upper-case characters). All ten human digits are used, and a large 12-inch by 4-inch physical device is required; only one of the ten digits is used per gesture.

Dasher is a new interface incorporating language modelling and driven by continuous two-dimensional gestures, *e.g.* a mouse, touchscreen, or eye-tracker. Tests have shown that this device can be used to enter text at a rate of up to 39 words per minute during a dictation task, compared with typical ten-finger keyboard typing of 40-60 words per minute.

We also show that Dasher can be used as a mobile text entry device. On a hand-held personal organiser, users enter text with a stylus and a touchscreen.

Dasher can be used with an eye-tracker as a hands free text entry system. Expert writing speed is currently 20 words per minute. We hope that this system will be useful to people with limited access to a keyboard.

The writing speed of Dasher depends of the quality of the language model. The Dirichlet Language Model, Boltzmann machines and neural networks are all investigated as better language models for use in Dasher.

Dasher is still under development; we think it shows promise as a keyboard–less text entry system both in its absolute writing speed and ease of use.

Acknowledgements

During the past three years in Cambridge, I have benefited from discussions with many people. I am thankful to Alan Blackwell, Robert Chappell, Phil Cowans, Matthew Davey, Russell Goyder, Sanjoy Mahajan, James Miskin, Stephen Oliver, Edward Ratzer, T. Tokieda and Sebastian Wills.

I especially thank my supervisor, David MacKay, for his advice, criticism, and source of new ideas.

Dasher was conceived by David MacKay and Mike Lewicki and the first prototype would have not been possible without Ousterhout's Tcl language. Thanks to the experimental subjects who took part in the evaluation of Dasher. Thanks to T. Matsumoto and T. Fujiyoshi for advice with the Japanese version.

This work was supported by the Engineering and Physical Sciences Research Council, the Cavendish Laboratory and the Inference Group.

This thesis is dedicated to Martin, Ann, Richard and Sarah.

Contents

Chapter 1

The Information Content of Natural Language

Entropy is a measure of the average information content of an outcome. If plain text is optimally compressed into binary digits, the entropy H of the language is the average number of binary digits required per letter of the original text.

1.1 Experiments with Human Guessing

In a famous experiment in 1951 [86] Shannon estimated the entropy of English to be about 1 bit per character. Shannon pioneered the technique of eliciting human knowledge of language by asking human subjects to predict the next element of text.

The procedure Shannon used was to show subjects text and ask them to guess the next letter. If they were wrong, they were told so and asked to guess again, until they guessed the correct letter. A typical result from this experiment is shown below, where • indicates a space and subscripts show the number of guesses.

$$T_1 H_1 E_1 R_5 E_1 \bullet_1 I_2 S_1 \bullet_1 N_2 O_1 \bullet_1 R_{15} E_1 V_{17} E_1 R_1 S_1 E_2 \bullet_1 O_3 N_2 \bullet_1 A_2 \bullet_2 M_7 O_1 T_1 O_1 R_1 C_4 Y_1 C_1 Y_1 E_1$$

He proved that if the probability of taking r guesses until the correct letter is guessed is p_r, then the entropy, H (in bits per character) is bounded in the following way:

$$\sum_r r(p_r - p_{r+1}) \log_2 r < H < \sum_r p_r \log_2 \frac{1}{p_r}. \tag{1.1}$$

Shannon chose 100 random samples taken from Dumas Malone's *Jefferson the Virginian*. From the number of guesses made by each subject, he derived estimates of the lower and upper bounds equal to 0.6 and 1.3 bits per character.

Cover and King [26] noted that Shannon's procedure gave only partial information subjects predictive probability distribution. They performed a similar experiment in which subjects assigned probabilities directly, by gambling. Cover and King derived an upper bound of 1.25

Upper bound on entropy (bits per character)	Reference	Model
$0.6^* - 1.3$	Shannon [86]	Experiments with human subjects
1.2	Rosenfeld [81]	Maximum entropy language model
1.25	Cover and King [26]	Experiments with human subjects
1.5	Tilbourg [94]	n-grams
2.14	Shannon [86]	Word frequencies
2.39	Witten, Moffat, Bell [100]	PPMC (context length of 5)
2.48	Moffat [69]	PPMC (context length of 3)
3.1	Shannon [86]	Character trigrams
3.32	Shannon [86]	Character bigrams
4.03	Shannon [86]	Character frequencies
4.06	Bell, Cleary & Witten [4]	Unix compress
5	Church and Mercer [21]	Huffman encoding of each symbol

Figure 1.1: Estimates of the entropy of English. * - lower bound.

bits per character.

1.2 Statistical Analysis of Letter and Word Frequencies

Shannon also made estimates based on the statistical analysis of letter and word frequency data. He showed that the upper bound estimate of the entropy decreased from 4.0 bits per character using letter frequencies, down to 3.1 bits per character for character trigrams. His estimate based on word frequencies was 2.1 bits per character. These data, together with other estimates of the entropy of English, are shown in figure 1.1.

1.3 Modelling Marginal Frequencies

Consider the task of modelling discrete data, a string of N symbols, $s_1, s_2, s_3, \ldots, s_N$ from an alphabet of size $\mathcal{A}$. For the time being, we make the assumption that the symbols are independent. We define the marginal count F_i to be the number of times the symbol i occurs in the string. The maximum likelihood estimate of the probability of character i is

$$P(i) = \frac{F_i}{\sum_{j=1}^{\mathcal{A}} F_j}. \tag{1.2}$$

The maximum likelihood estimate will generate a zero probability for any symbol whose count is zero.

Arithmetic coding allows the encoding of messages into a number of bits equal to the log probability assigned by the model [70]. A zero probability prevents arithmetic coding from working correctly; the 'optimum' code length for a zero probability string is infinite.

1.3.1 Laplace's law of succession

A solution was formulated by Laplace and is known as Laplace's Law of Succession [56]:

$$P(i) = \frac{F_i + 1}{\mathcal{A} + \sum_{j=1}^{\mathcal{A}} F_j}. \tag{1.3}$$

Laplace proved that this is the correct Bayesian prediction if we put a uniform prior on the probabilities.

Laplace's law of succession has been widely criticised for assigning too much or too little probability to novel events. This has led many influential statisticians to reject Bayesian inference altogether [31, 32, 48]. It is only fair to point out, as Laplace did, that the uniform prior does not take into account any comprehensive prior knowledge of the problem in question.

1.3.2 Lidstone's Law

A commonly adopted alternative to Laplace's Law is Lidstone's Law of succession [58]:

$$P(i) = \frac{F_i + \lambda}{\lambda\mathcal{A} + \sum_{j=1}^{\mathcal{A}} F_j}, \tag{1.4}$$

where λ is a positive constant. This class of estimate is prominent in the information theory literature [25, 35, 54], and the statistical language modelling community [51]. We observe that $\lambda = 0$ corresponds to the maximum likelihood estimate and $\lambda = 1$ gives Laplace's law of succession. The most widely advocated single value for λ is $1/2$ [49, 78], but the reasons are diverse and unsatisfactory.

1.3.3 A Bayesian viewpoint - Dirichlet distributions

Dirichlet distributions are particularly convenient prior distributions to work with where count data is involved. The Dirichlet distribution is discussed by MacKay and Peto [62], and is reviewed here.

The Dirichlet distribution for a probability vector $\mathbf{p}$ with $\mathcal{A}$ components is parameterized by a measure (a positive vector) which I will write here as $\mathbf{u} = \alpha\mathbf{m}$, where $\mathbf{m}$ is a normalized measure over the $\mathcal{A}$ components ($\sum m_i = 1$), and α is positive:

$$P(\mathbf{p}|\alpha\mathbf{m}) = \frac{1}{Z(\alpha\mathbf{m})} \prod_{i=1}^{\mathcal{A}} p^{\alpha m_i - 1} \delta\left(\textstyle\sum_i p_i - 1\right) \equiv \text{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\alpha\mathbf{m}). \tag{1.5}$$

The function $\delta(x)$ is the Dirac delta function which restricts the distribution to the simplex such that $\sum_i p_i = 1$. The normalizing constant is

$$Z(\alpha\mathbf{m}) = \prod_i \Gamma(\alpha m_i) \,/\Gamma(\alpha) \tag{1.6}$$

The Gamma function, $\Gamma(\alpha)$ is described in appendix B. The vector $\mathbf{m}$ is the mean of the probability distribution:

$$\mathbf{m} = \int d^{\mathcal{A}}\mathbf{p}\, \mathrm{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\alpha\mathbf{m})\mathbf{p} \tag{1.7}$$

The effect of α can be visualized by drawing samples from the Dirichlet distribution, with $\mathbf{m}$ set to the uniform vector, $m_i = 1/\mathcal{A}$ and making a Zipf plot. In figure 1.2, we plot p_i versus rank on a logarithmic scale. A large value of α produces a distribution over $\mathbf{p}$ which is sharply peaked around $\mathbf{m}$. As α is decreased, the distribution becomes broader; as α approaches zero, the distribution spreads into the corners of the simplex. For α close to zero, a typical sample $\mathbf{p}$ has nearly all its mass on just one component p_i.

If we observe samples from $\mathbf{p}$ and obtain counts $\mathbf{F} = (F_1, F_2, \ldots F_{\mathcal{A}})$ of the possible outcomes i, then we use Bayes' theorem to infer the posterior probability of $\mathbf{p}$, given the counts $\mathbf{F}$:

$$\begin{aligned} P(\mathbf{p}|\mathbf{F}, \alpha\mathbf{m}, \mathcal{H}_D) &= \frac{P(\mathbf{F}|\mathbf{p}, \mathcal{H}_D)P(\mathbf{p}|\alpha\mathbf{m}, \mathcal{H}_D)}{P(\mathbf{F}|\alpha\mathbf{m}, \mathcal{H}_D)} && (1.8)\\ &= \frac{\prod_i p_i^{F_i} \prod_i p_i{}^{\alpha m_i - 1} \delta(\sum_i p_i - 1)/Z(\alpha\mathbf{m})}{P(\mathbf{F}|\alpha\mathbf{m})} && (1.9)\\ &= \frac{\prod_i p_i^{F_i + \alpha m_i - 1} \delta(\sum_i p_i - 1)}{P(\mathbf{F}|\alpha\mathbf{m})Z(\alpha\mathbf{m})} && (1.10)\\ &= \mathrm{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\mathbf{F} + \alpha\mathbf{m}). && (1.11) \end{aligned}$$

The posterior probability of $\mathbf{p}$ is another Dirichlet distribution. The predictive distribution given the data $\mathbf{F}$ is then:

$$P(i|\mathbf{F}, \alpha\mathbf{m}, \mathcal{H}_D) = \int \mathrm{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\mathbf{F} + \alpha\mathbf{m})\, \mathbf{p}\ d^{\mathcal{A}}\mathbf{p} = \frac{F_i + \alpha m_i}{\sum_{i'} F_{i'} + \alpha m_{i'}}. \tag{1.12}$$

Note that the term αm_i appears as an effective initial count in bin i. The value of α defines the number of samples from $\mathbf{p}$ that are required in order that the data dominate over the prior.

In addition, if we set $\alpha m_i = 1$, the prior $P(\mathbf{p})$ is uniform and we have Laplace's law of succession.

1.4 Basic Language Modelling

Speech recognition, automatic translation and text compression all depend on a language model that assigns probabilities to word or character sequences. The automatic translation system implemented by IBM used a trigram model for words with impressive results [16]. Trigram models are also used in speech recognition systems [3]. Text compression is similar prediction task in which we assign a probability distribution to all possible sequences of symbols.

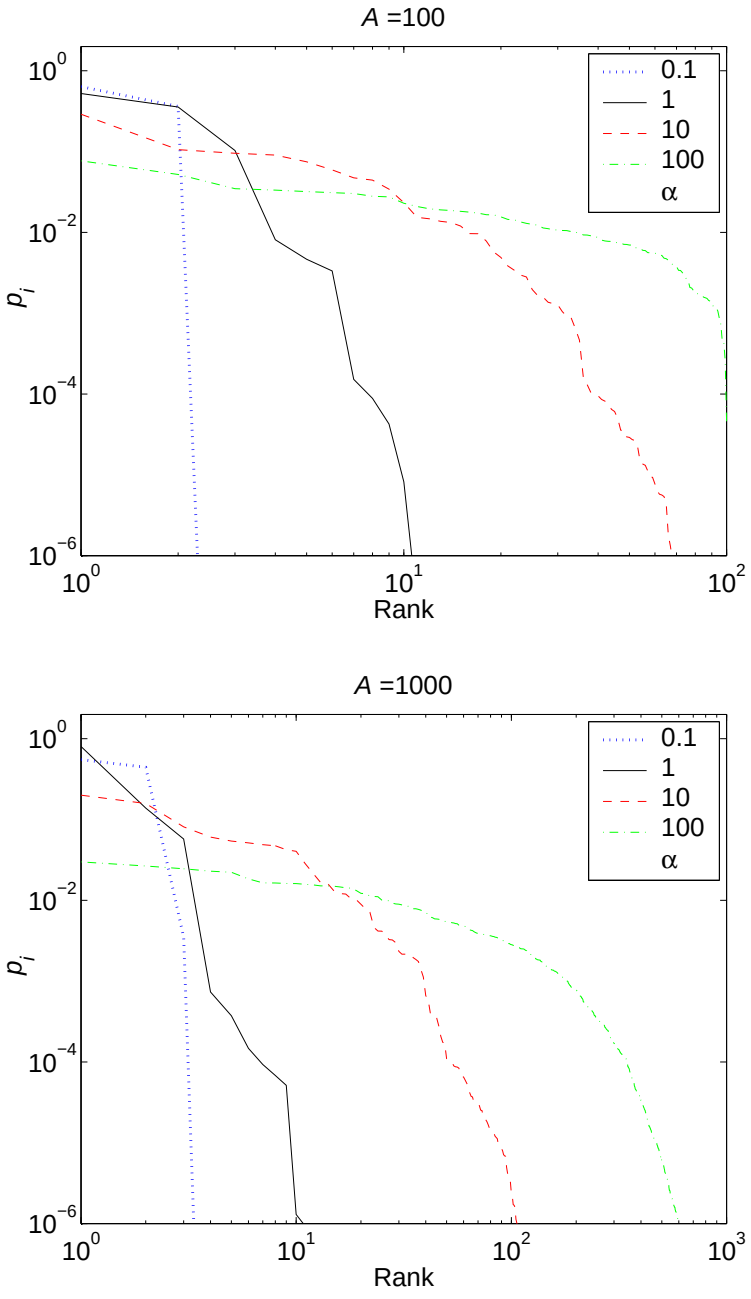


Figure 1.2: Zipf plots for random samples from Dirichlet distributions with various values of $\alpha = 0.1, \ldots, 100$. For each given $\mathcal{A}$ and α, a sample from the Dirichlet distribution was generated. The Zipf plot shows the probabilities p_i, ranked by magnitude, versus their rank.

The language model most frequently used by present–day speech recognizers is based on the very simple approximation that the probability of a word depends only on the last two words.

$$P(\mathbf{W}) = \prod_{i=1}^{N} P(w_i|w_{i-2}, w_{i-1}) \tag{1.13}$$

Language models of this form are called trigram models. The simplest way to estimate the conditional probability is by

$$P(w_i|w_{i-2}, w_{i-1}) \approx f(w_i|w_{i-2}, w_{i-1}) \equiv \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}. \tag{1.14}$$

$C(w_1, w_2, \ldots, w_n)$ is the number of times we observe the string $w_1, w_2, \ldots, w_n$ in the training text. Equation 1.14 can lead to poor predictions, since many possible word trigrams are never encountered, even in a very large corpora of training text.

The smoothed trigram language model combines trigram, bigram and unigram relative frequencies.

$$P(w_i|w_{i-2}, w_{i-1}) = \lambda_3 f(w_3|w_1, w_2) + \lambda_2 f(w_3|w_2) + \lambda_1 f(w_3) \tag{1.15}$$

where the non–negative weights satisfy $\lambda_1 + \lambda_2 + \lambda_3 = 1$. The combination of different order models is usually known as smoothing. A cross-validation procedure called 'deleted interpolation' is used to set the values λ_i [3, 50]. This procedure involves dividing the data into a number of blocks, computing the predictions for each block using the other blocks as training data. The values of λ_i are adjusted to optimize predictive performance.

Other methods of smoothing include Katz smoothing [52], Kneser-Nay smoothing [53] and Witten-Bell smoothing [4, 99]. Many of these methods are based on the Good-Turing estimate [39]. A review of smoothing techniques can be found in [20].

MacKay and Peto [62] demonstrated a Bayesian language model whose predictions are similar to smoothing. The Dirichlet model is not identical to smoothing; it does away with cross-validation and therefore makes full use of the data while requiring fewer computational resources. We develop the Dirichlet language model in chapter 5.

1.5 Prediction by Partial Match

The state of the art in text compression for over 15 years has been the PPM text compression algorithm. The original algorithm was first published in 1984 by Cleary and Witten [23]. Several improvements were described by Moffat, who implemented the benchmark version, PPMC [69]. PPMC achieves superior results to many other compression methods, despite attempts to better it.

Other methods such as those based on Ziv-Lempel coding [102, 103] are more commonly used in practice (for example, in the popular gzip compressor). Compression performance generally falls below that of PPM [4], but Ziv-Lempel coding is usually faster.

Finite–context models use the preceding few symbols (the context) to estimate the probability of the next one. It is desirable to have a long context, to take into account as much relevant information as possible when making predictions. However, when using a long context, most of the contexts observed in a novel data set will never have been seen in the training text.

PPM is finite–context statistical modelling technique that can be viewed as blending together several fixed–order context models to predict the next symbol. The length of the context is usually fixed, although recent PPM models by Bloom [12] have employed adaptive length contexts.

1.5.1 Blending

The order of a finite–context model is the length of the context used to make predictions. An order 1 model corresponds to a bigram model; an order 0 model uses the marginal frequencies of symbols. We also introduce an 'order -1' model which has a uniform distribution over symbols.

Let $P_o(\phi)$ be the adaptive probability assigned to a symbol ϕ by the finite–context model of order o. If the weight given to the model of order o is w_o and the maximum order used is m, then the blended probability $P(\phi)$ is given by

$$P(\phi) = \sum_{o=-1}^{m} w_o P_o(\phi). \tag{1.16}$$

The weights are normalized to sum to one.

1.5.2 Escape probabilities

In the PPM algorithm, the weights in equation 1.16 are determined indirectly with escape probabilities. Each model divides the probability space into two; one part for its own predictions, and the second for the predictions of lower order models. The escape probability is the proportion allocated to lower order models.

The escape mechanism is equivalent to the blending mechanism. Denoting the probability of an escape at level o by e_o, equivalent weights can be calculated from the escape probabilities by

$$w_o = (1 - e_o) \prod_{i=o+1}^{l} e_i, \ -1 \leq o < l \tag{1.17}$$

$$w_l = 1 - e_l \tag{1.18}$$

where l is the highest order making a non-zero prediction. The advantage of expressing things in terms of escape probabilities is that it is more practical to implement the escape mechanism than weight blending. For example, when using arithmetic coding we encode a special *escape*

symbol every time the model escapes to a lower order context.

Let $c_o(\phi)$ be the number of times that the symbol ϕ occurs in the current context of order o. Denote by C_o the total number of times the context has been seen; that is,

$$C_o = \sum_{\phi \subset A} c_o(\phi). \tag{1.19}$$

Many different escape mechanisms have been suggested; most are adhoc. We describe the most important four.

Method A

Method A is based on Laplace's law of succession, described in section 1.3.1. We allocate one additional count over and above the number of times the context has been seen to allow for occurrences of new symbols. The escape probability is

$$e_o = \frac{1}{1 + C_o}. \tag{1.20}$$

The probability assigned by each model is proportional to each symbol's relative frequency,

$$p_o(\phi) = \frac{c_o(\phi)}{C_o}. \tag{1.21}$$

Method B

Method B classifies a symbol as novel even if it has occurred once before. Let n_o be the number of different symbols that have occurred in some context of order o. The escape probability is defined as

$$e_o = \frac{n_o}{C_o}. \tag{1.22}$$

The probability assigned by each model is

$$Wp_o(\phi) = \begin{cases} \frac{c_o(\phi)-1}{C_o - n_o} & c(\phi) > 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.23}$$

Method C

Method C was proposed by Moffat [69] and uses the number of times a novel character has occurred:

$$e_o = \frac{n_o}{1 + C_o}, \tag{1.24}$$

$$p_o(\phi) = \frac{c_o(\phi)}{C_o}. \tag{1.25}$$

Method C is implemented by maintaining an escape count and a symbol count. When a new symbol is received, we add one to the escape count and one to the symbol count. If a symbol

has already been seen in a context, we just add one to the symbol count.

Method D

Method D is a minor modification of method C:

$$e_o = \frac{n_o}{2C_o}, \tag{1.26}$$

$$p_o(\phi) = \begin{cases} \frac{c_o(\phi)-\frac{1}{2}}{C_o-\frac{n_o}{2}} & c(\phi) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1.27}$$

In method D, we also maintain an escape count and a symbol count. However, when encountering a new symbol, we add 1/2 to both the escape count and the symbol count. If a symbol has already been seen, we add 1 to the symbol count.

Experiments with compressing files show that method D is slightly better than method C. Both methods perform better than A and B.

1.5.3 Approximate blending

In a fully blended model, the probability of a symbol includes predictions from several different contexts. Full blending is rarely a practical technique for finite–context modelling because it is so slow.

Exclusion

The escape mechanism can be used as a basis for an approximate blending technique called *exclusion*. When coding the symbol ϕ using context models with a maximum order of m, the order-m model is first consulted. If it predicts ϕ with a non–zero probability, it is used to code ϕ. Otherwise, the escape symbol is encoded, and the next shortest context attempts to predict ϕ. The context of order -1 ensures success.

The exclusion method is so named because it excludes lower–order predictions from the final probability of a symbol. Experiments to assess the impact of exclusion indicate that compression is only a few percent worse than for a fully blended model. Furthermore, execution is much faster and implementation is simplified considerably.

Update exclusions

In a fully blended model, we update the counts in all the models after each symbol is coded. However, when exclusions are used, only one context is used to predict the symbol.

The fact that only one context is used to predict the symbol suggests a modification to the method of updating the models, called update exclusion. The count for a predicted symbol is not incremented if it is already predicted by higher order context. In other words, a symbol

is only counted in the context used to predict it. This generally improves performance by a few percent.

1.5.4 Recent improvements to PPM

Charles Bloom [12] has recently proposed some improvements to the PPM algorithm; his variant is called PPMZ. Compression is improved by about 4% over PPMD. We summarize the improvements below.

Unbounded length deterministic contexts

Instead of using a maximum order, we match the highest order context found in training set.

Secondary escape estimation

The escape estimation scheme is adaptive, rather than fixed.

Local order estimation

A major problem with PPM is that different files have different optimal highest orders. A local order estimation (LOE) scheme is incorporated to decide which order to use for each encoded symbol.

1.6 Compression Results

1.6.1 Canterbury Corpus

The Canterbury Corpus (`http://corpus.canterbury.ac.nz/`) is a benchmark to enable researchers to evaluate lossless compression methods. The files selected were based on their ability to provide representative performance results. In figure 1.3, we summarize results from the Canterbury Corpus. Descriptions of each compression method can be found in appendix A.

1.6.2 Archive Comparison Test

The Archive Comparison Test (`http://compression.ca/`) is a set of benchmarks designed to show the state of the art in lossless data compression. While the Canterbury Corpus is primarily intended for testing new algorithms, the Archive Comparison Test is aimed at commercial systems. In figure 1.4, we summarize results from the Archive Comparison Text. Descriptions of each compressor can be found in appendix A.

Method	Compressed length (bits per byte)
PPMD5	2.11
BZIP	2.15
SZIP	2.16
PPMC	2.19
BZIP2	2.23
BRED	2.38
DMC	2.39
GZIP	2.53
compress	3.31
LZRW1	4.18
pack	4.53
cat	8.00

Figure 1.3: Canterbury Corpus compression results (Dec 2000). Figures quoted are compressed length in bits per character. The algorithms and are sorted by average compression ratio on the corpus.

Name	Compress time (seconds)	Extract time (seconds)	Compressed length (bits per byte)
RK 1.04.	131.6	132.5	1.42
SBC 0.860	24.1	10.1	1.51
RKIVE 1.92b1	234.7	211.5	1.56
ACB 2.00c	332.4	333.4	1.56
BOA 0.58b	119.3	123.0	1.57
PPMD vE	11.9	13.5	1.65
SZIP 1.11	42.2	10.6	1.65
BZIP2 0.9.5d	23.6	6.4	1.79
RAR 2.80	49.7	2.4	1.99
ACE 1.2b	50.1	1.0	2.27
PPMC3h	14.4	16.0	2.42
PKZIP 2.6.02	8.6	1.2	2.55
WINZIP 7.0	16.3	1.7	2.56
GZIP 1.2.4	20.5	6.8	2.56
TAR 3.21g	1.4	0.4	8.01

Figure 1.4: The Archive Comparison Test, Jun 2001. A survey of the performance of commercial text compression software. Details on each of these packages can be found in appendix A.

1.7 The State of the Art in Language Modelling

To date, the best language model is a maximum entropy language model by Rosenfeld [81]. Maximum entropy models are exponential models which satisfy given constraints. For example, if we want to predict a word w_i, given a history or context $h_i = \{w_1, w_2, \ldots, w_{i-1}\}$, then the conditional probability of w_i is

$$P(w_i|h_j) = \frac{1}{Z(h_j)} \exp\left[\sum_k \lambda_k f_k(w_i, h_j)\right] \tag{1.28}$$

where $f_k(w_i, h_j)$ is a constraint, and $Z(h_j)$ is a normalization factor. The maximum likelihood parameters λ_k are usually derived using the Generalized Iterative Scaling Algorithm [27]. The language model achieved about 1.2 bits per character on hundreds of megabytes of newspaper articles.

A survey of human language technology describing various applications of these models is given in Cole [24]. Some other types of language models have been based on context–free grammars [19], decision trees [2], local rules [15], collocation matrices [34] and neural networks [83].

Chapter 2

Data Entry

Gesture-based text entry systems involve two significant trade-offs: between potential efficiency and training time, and between device size and the number of accessible characters. Research on keyboards for conventional typing by Norman [77] has addressed efficiency versus training, while Sears [85] reports the effect of keyboard size. In conventional keyboards, efficiency depends on relative key size (according to Fitts' law [66]) and on the mapping between movement over the keyboard configuration and English letter sequences. Learning speed appears to be a function of similarity to the QWERTY layout [41], which has almost ubiquitous familiarity, even in non-typists [77].

The difficulty of resolving these trade-offs has led to the predominance of the QWERTY keyboard, even in the face of improvements such as Dvorak [57], alphabetic [77], or chord keyboards [40].

2.1 The Information Content of Text and of Hand Movements

Conventional keyboards require the user to make one or two gestures per character (one for lower-case and two for upper-case characters). Each gesture is a selection of 1 from about 80 keys, so the keyboard has the capacity to read about $\log_2(80) = 6.3$ bits per gesture. However, the entropy of English text is roughly 1 bit per character [87], so existing keyboards are inefficient by a factor of six. This inefficiency manifests itself in the fact that when typing, people make errors that are clearly not English.

The keyboard is inefficient for two reasons. First, text is usually highly redundant, yet users are forced to type most or all of the characters in a document. Second, keyboards register only contact events, whereas humans are capable of fine motor movements. Human motor capabilities dictate an upper limit for finger tapping frequency, and hence the information rate achievable from successive key presses, no matter how small the distance between keys. Continuous pointing gestures have the potential to convey more information through high-resolution measurement of position.

According to MacKenzie's analysis of Fitts' experiments measuring one-dimensional pointing speed and accuracy, the rate at which information can be conveyed by one-dimensional

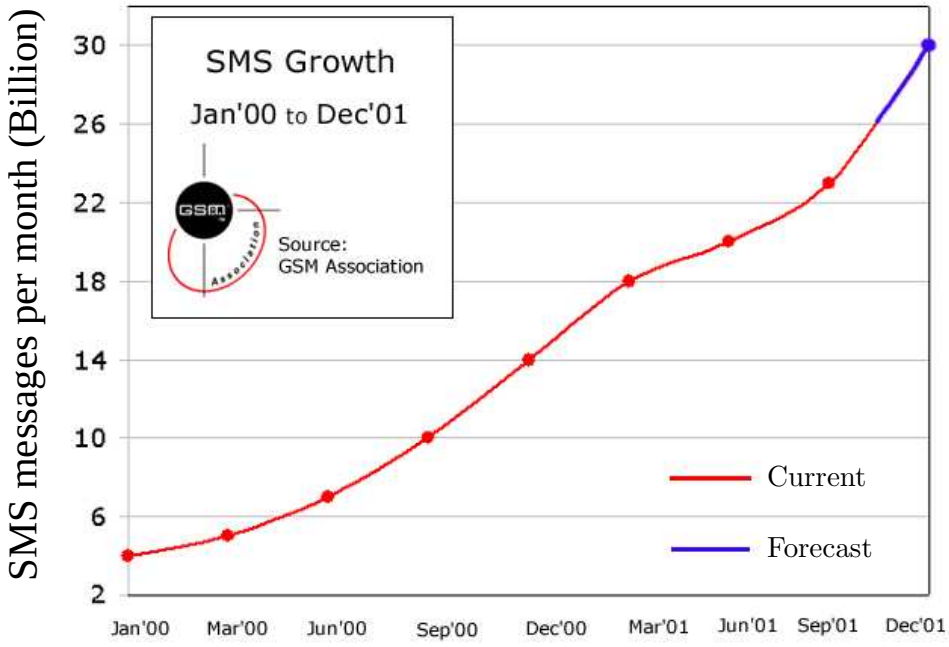


Figure 2.1: Growth in text messaging from 2000-2001. Source: GSM Association (figure used with permission).

pointing is about 8.2 bits per second [65].

So using just one finger in a one-dimensional pointing environment, it should be possible to write English at a rate of 8.2 characters per second. This character rate corresponds to about 100 words per minute. How close we can get to this figure depends on the quality of our model of English, and on whether we can map finger gestures to text in a user friendly way.

2.2 Mobile Text Entry

2.2.1 The personal digital assistant (PDA)

PDAs are small hand-held mobile devices that provide portable computing capabilities for individual use. Calendars, note taking, address books, task lists, and email are usually standard features.

Since their introduction in the early 1990s, pen-based systems have faltered in their user acceptance. Claims of unconstrained, cursive handwritten input did not meet the expectations of a demanding user community. Today however, handwriting recognition products are much improved [10, 11] and pen-based PDAs are increasing in popularity.

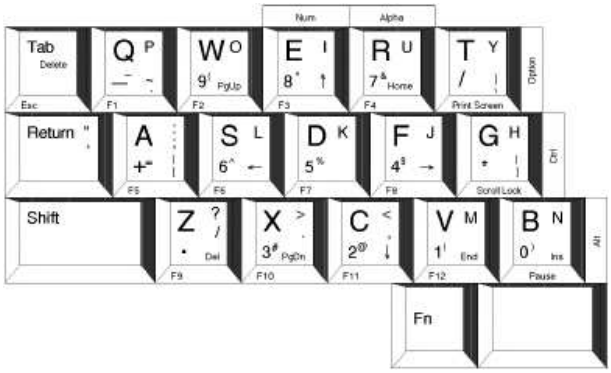


Figure 2.2: Half-QWERTY keyboard. When a key is depressed, the character in the upper left of the key is entered. When preceded by holding down the space bar, the character in the lower right is entered.

2.2.2 SMS messaging

SMS (short messaging service) is a popular messaging service and is incorporated into most mobile phones. Figure 2.1 shows the growth over the last two years. SMS messaging possesses several appealing features; it is inexpensive, unobtrusive, and communication is asynchronous.

Overwhelmingly, SMS messages are entered using the telephone keypad and the multi-tap input technique [29, 90], although other techniques are emerging such as Tegic's T9 (www.tegic.com) and Eatoni's WordWise (www.eatoni.com). Evidently, the appeal of text messaging is so strong that users are willing to put up with the poor performance of the 12-key phone keypad.

2.2.3 Mobile text entry solutions

In contrast to the predominance of the QWERTY keyboard in the conventional keyboard market, no single mobile text entry system is dominant. Approaches to text entry in hand-held devices can be grouped into three broad categories; miniature and rearranged keyboards; gestural alphabets; and dynamic selection techniques.

Miniature and rearranged keyboards

Miniature and rearranged keyboards present as much of the alphabet as possible while retaining sufficiently large keys. The number of keys is often reduced, and some selection mechanism is used to toggle between alternate sets. The 'Half-QWERTY' device shown in figure 2.2 uses half of the QWERTY keyboard. When a key is depressed, the character in the upper left of the key is entered. When a key press is preceded by holding down the space bar, the character in the upper right of the key is entered. Evaluation showed typing speeds of 34.7 words per minute were achieved after ten sessions, each lasting 50 minutes [68]. The 'Fitaly' keyboard (figure 2.2.3) arranges keys in a compact square optimized for minimal hand travel

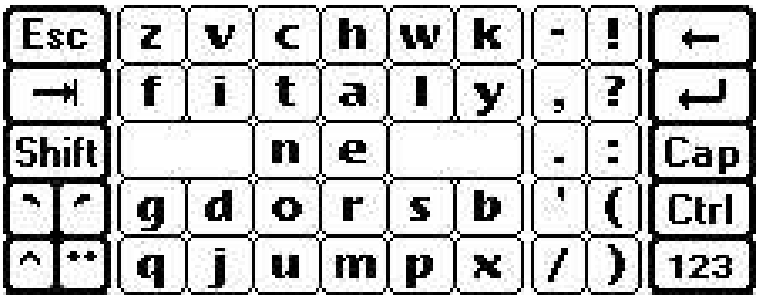


Figure 2.3: Fitaly keyboard.

Figure 2.4: The 12-key telephone keypad.

with one-finger typing [64].

T9 and multi-map text input systems use the conventional mapping of the 26 alphabetic characters onto a 12-key telephone keypad, shown in figure 2.4. Multi-tapping requires multiple key presses per letter. For example, 1 press of the '2' key types the letter 'a'; 3 presses of the '6' key types the letter 'o'. In T9, the user presses one key per letter and a dictionary helps to disambiguate alternative readings [90]. Both Fitaly and T9 are available as alternative text entry systems for the Palm Pilot. An experiment with 20 subjects compared novice and expert mobile phone text users [46]. When writing chat messages, experts achieved 26 words per minute with T9, versus 11 words per minute for multi-tap. Novices achieved 11 words per minute with T9 versus 10 words per minute with multi-tap. T9 demands visual attention while multi-tap does not.

The OPTI keyboard – a mathematically optimized keyboard has been proposed in which keys are arranged on a hexagonal grid [44]. While the predicted text entry speed is 42 to 44 words per minute, a recent evaluation [101] reports writing speeds of 8.9 to 9.7 words per

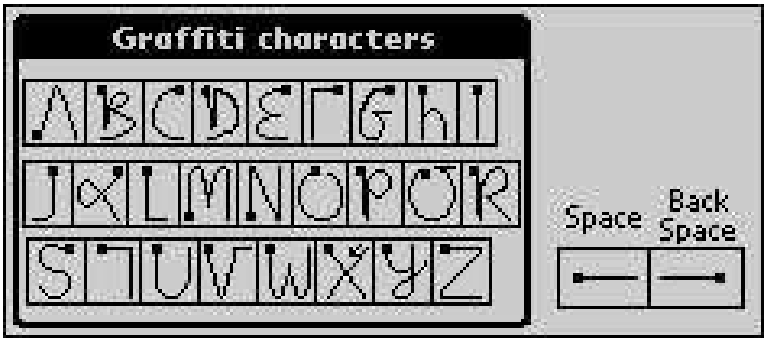


Figure 2.5: Graffiti alphabet.

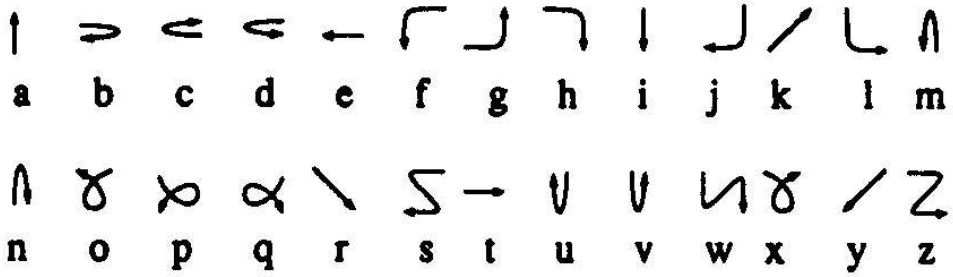


Figure 2.6: The Unistroke alphabet.

minute.

Sears [85] has evaluated the effect of using reduced sized QWERTY keyboards on a one-finger touch screen. Typing speed for experts after 30 minutes was 32.5 words per minute with a large keyboard (246 mm wide), dropping to 21.1 words per minute on a very small one (68 mm wide).

It is also possible to reduce keyboard size with 'chorded' schemes having fewer keys, but in which multiple keys are pressed at one time. Performance of the ternary chorded keyboard (TCK) [55] was about 16 words per minute after 600 minutes training.

Gestural alphabets

Most hand-held pen devices are supplied with software to recognize handwritten characters, possibly using a special alphabet. Early device such as the Apple Newton attempted, with limited success, to recognize users' natural handwriting - and hence required no learning on the part of the user. Current systems – Graffiti [67] for the Palm Pilot (figure 2.5), and Jot for Windows CE (also available on the Pilot) – improve efficiency while maintaining some resemblance to alphabetic shapes to assist learning. More efficient alphabets such as Goldberg's Unistrokes [38], map simple gestures to common characters regardless of mnemonic similarity (figure 2.6). The inventors of Unistrokes estimate a rate of 40.8 words per minute,

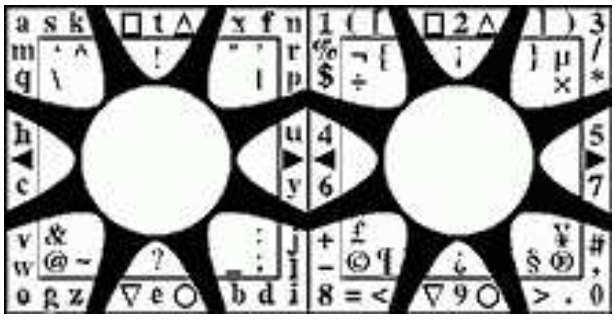


Figure 2.7: Quikwriting. Two of of the four character sets are shown above. Special characters allow the user to toggle between the sets.

without any empirical testing.

Dynamic selection

Dynamic selection techniques share characteristics of miniature keyboards and gestural alphabets. The user moves the pen in the direction of a selection region, which either selects a character or reveals a further set of alternatives. Examples include T-Cube [95] and Quikwriting [79] (figure 2.7). Quikwriting is commercially available for the Palm Pilot.

The FOCL (Fluctuating Optimal Character Layout) keyboard rearranges keys after each keystroke to place the most probable next letters near the centre. Results with the FOCL keyboard were around 10 words per minute after 10 sessions of 15 minutes [7].

These dynamic selection devices are subject to a trade-off between efficiency and ease of learning. The arrangement of nested regions or sets of alternatives must favour the selection of common characters, while still being learnable.

In chapter 4, we introduce Dasher, a novel data entry interface. Dasher offers an alternative dynamic selection technique which is easier to learn, and improves efficiency by dynamically setting the size of the selection targets according to a language model. The system can be used with arbitrary alphabets; special characters and capitalization can be included without modification.

2.3 Text Entry for Disabled Users

2.3.1 Introduction

The standard PC keyboard is designed to be used with two hands. It favours right handed people (the numeric keypad is on the right), and can be 'over-sensitive' so that it is easy to type a string of letters if a key is held down for slightly too long.

Figure 2.8: A chorded keyboard.

2.3.2 Modifying the keyboard response

There are a number of software programs that can be used to modify the way the keyboard behaves to suit a user's needs. Some of the main functions are:

- Sticky keys allow single finger typists to operate shift, control and alt keys. The modifier key is effectively held down until the next key is pressed. So to type '`The`' the keystrokes would be: `shift t h e`.
- Modifying the key repeat rate, which is the length of time a key needs to be held down before another character is outputted.
- 'Mouse keys' allow the mouse pointer to be moved around using the numeric keypad keys.

2.3.3 Large keyboards

Large keyboards can help in situations where it is difficult to locate a normal sized key-top accurately. The larger size gives more area to 'aim at'. If a user is unable to fully support the weight of his arm, then the keys are often sunk beneath the surface of the keyboard.

2.3.4 Small keyboards

Small keyboards can be more easily positioned and are often suited to single handed users. These keyboards can fit between the arms of a standard wheelchair, for example.

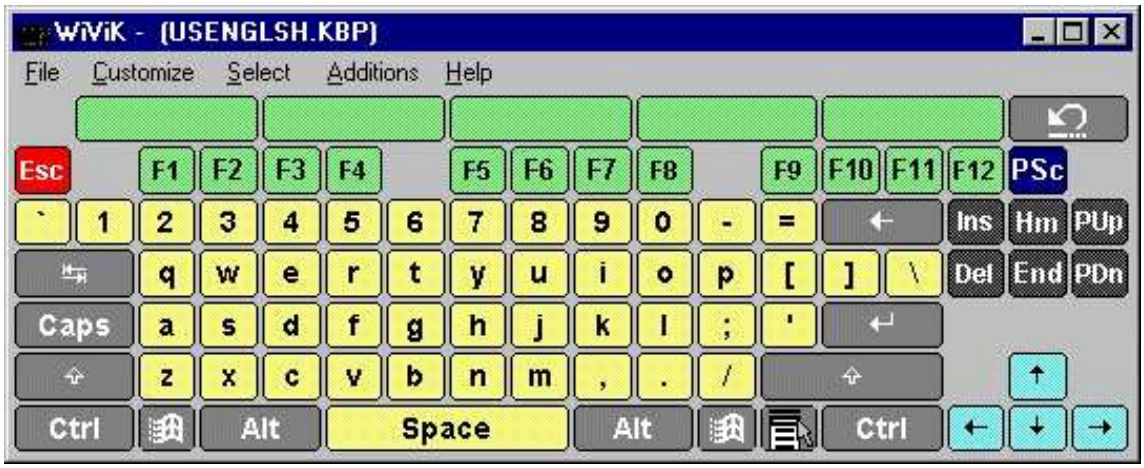


Figure 2.9: WiViK2 on-screen keyboard.

2.3.5 Specialized keyboards

Chorded keyboards (figure 2.8) have only a few keys and are operated by pressing multiple keys at a time. They generally work well for single handed users who possess independent movement of each finger, but learning is slow.

2.3.6 Soft keyboards

WiViK is an on-screen keyboard for designed specifically for persons with physical disabilities [88]. It allows any pointing device to be used to enter text into any application. The pointing device could be a mouse, touchpad, trackerball, joystick, or eye-tracker.

2.3.7 Speech recognition

Speech recognition systems are almost as fast as conventional keyboards [47], but are not always socially acceptable and users may also suffer from voice fatigue.

2.3.8 Eye-tracking

A straightforward way to implement text input by gaze-tracking is to display a QWERTY keyboard on the computer screen and use dwell time as a selection technique. For example, the WiViK keyboard can be used in this way. We describe eye-tracking in chapter 3.

A character is produced when the gaze of the user stays on a key for a predetermined amount of time. The length of the dwell time is a trade-off between rapid input and erroneous input. If the dwell time is too short, the system will generate input when the user is looking at or scanning the keyboard without the intention of selecting a key. If the dwell time is too long, the user gets frustrated because it takes so long for the system to respond when he or she is typing.

This basic input technique can be improved in many ways. One improvement is the use of various word completion schemes to reduce the number of key presses needed for writing. Another way to improve the interface is to replace the dwell time protocol by using a physical switch, such as blink of an eye, physical button, or EMG activity measured from a face muscle.

The Quick Glace system is an eye-tracker manufactured by Eyetech Digital Systems. The designers estimate a maximum writing speed of 1 character per second [18] (12 words per minute) when using their eyetracker with the WiViK keyboard. Thus on-screen keyboards are a painfully slow method for text input using eye gaze.

Chapter 3

Eye-tracking

3.1 Introduction

As early as 1936, Mowrer [71] used electrodes next to the eye to record the orientation of the eye in the head, and thus the direction of gaze.

Ware and Mikaelian [98] observed that single selection tasks could be performed more quickly with eye movement than a mouse. Before the user operates any mechanical pointing device, he or she usually looks at the destination to which he wishes to move. Thus eye movement is available as an indication of the user's goal before he or she actuates any other input device. A more recent experiment by Sibert and Jacob [89] confirms these results.

3.2 Eye-tracking Techniques

3.2.1 A technique based on electric skin potential

The electro-oculography tracking technique [37, 71] uses the electrostatic field that rotates along with the eye. By recording quite small differences in the skin potential around the eye, the position of the eye can be detected. This technique does not require a clear view of the eye, so this method has a large dynamic range of approximately $\pm 70^{\circ}$.

3.2.2 Techniques based on contact lenses

If the users wear a special contact lens, it is possible to make quite accurate recordings of the direction of gaze. By engraving one or more plane mirror surfaces on the lens, reflections of light beams can be used to calculate the position of the eye. Alternatively, a tiny induction coil can be implanted into the lens; the exact positioning of the lens can be recorded through wires attached to the lenses, and the use of high-frequency electro-magnetic fields placed around the user's head.

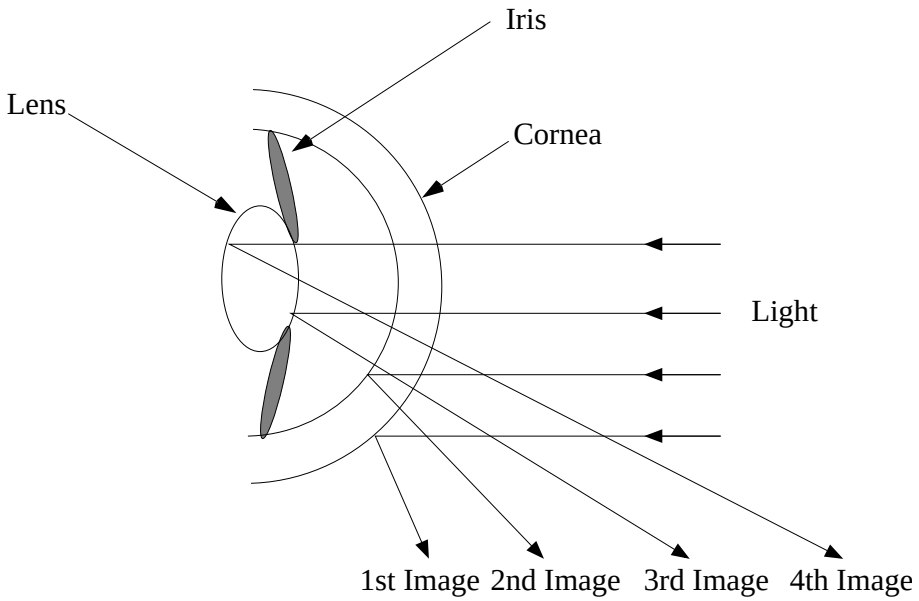


Figure 3.1: The four Purkinje images are reflections of incoming light on the boundaries of the lens and cornea.

3.2.3 Techniques based on reflected light

Limbus tracking (visible light)

The limbus is the boundary between the white sclera and the darker iris. The position of the limbus is used to determine the direction of eye gaze. The head is usually held quite still or the apparatus is fixed to the user's head.

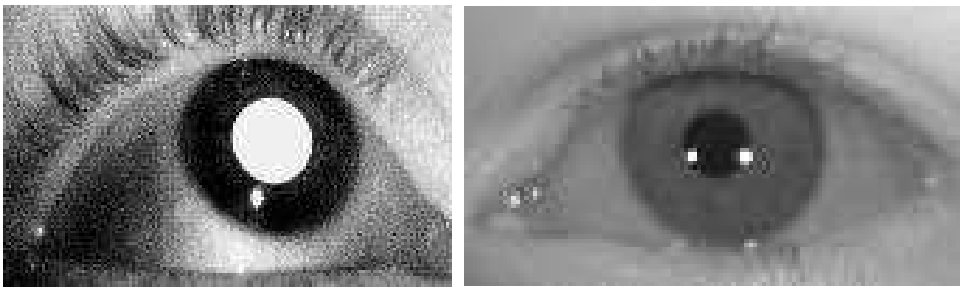
According to Scott and Findlay [84], limbus tracking is more suitable for precise horizontal tracking only. The covering of the top and bottom of the limbus by the eyelids means that vertical movement is harder to measure.

Pupil Tracking (visible light)

The boundary between the pupil and the iris is used to determine the direction of eye gaze. Once again, the apparatus is usually held still in relation to the head. The pupil is far less covered by the eyelids than the limbus, so a higher vertical resolution is achieved than with limbus tracking. The disadvantage of pupil tracking is that the difference in contrast is lower between the pupil and iris than between the iris and sclera, thus making the border detection more difficult.

Corneal Reflection (infrared)

When infrared light is shone into the user's eye, several reflections occur on the boundaries of the lens and cornea. The reflections which give rise to the so-called Purkinje images are shown in figure 3.1. The first Purkinje image (also called the glint), together with the reflection of



(a) (b)

Figure 3.2: (a) Bright-eye and corneal reflection with an on-axis infra-red source. (b) Dark-pupil and corneal reflection from two diffuse infra-red sources.

light off the retina (bright-eye or red-eye) appear as two bright spots under an infra-red camera (see figure 3.2(a)). When the eye is panned horizontally or vertically, the relative positioning of the glint and the centre of the bright-eye can be used to determine the direction of gaze.

In other eye-trackers, more diffuse infra-red light from one or more sources is used to illuminate the users eye (see figure 3.2(b)). Light reflected off the cornea still appears as a bright spot, but the pupil appears as a darker area. The relative positions of corneal reflections and the pupil are used to determine the direction of gaze.

According to Scott and Findlay [84], the range over which the direction of gaze can be tracked by simple algorithms is ± 12–15°. Larger eye movement causes the glint to appear outside of the spherical part of the cornea, thus requiring more complex geometrical calculations.

Purkinje Image Tracking

The first and fourth Purkinje images can be used for tracking the direction of gaze by the Dual Purkinje Image technique [72], which uses the relative positions of these reflections to calculate the direction of gaze. The Dual Purkinje Image technique is generally more accurate than other techniques with a spatial resolution about 1 minute of an arc. The disadvantage of this technique is that the fourth Purkinje image is rather weak, so the surrounding lighting must be carefully controlled.

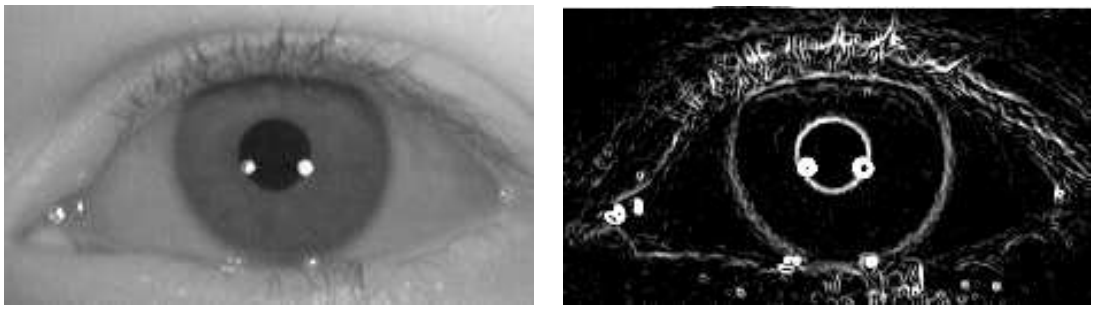
3.3 Computer Vision

The majority of commercial eye-trackers process video camera images to determine the direction of eye gaze. We describe the main image processing techniques used in these systems.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Figure 3.3: The two convolutional kernels used in the Sobel filter. Left - x gradient kernel. Right - y gradient kernel.



(a) (b)

Figure 3.4: (a) Example input to a Sobel Filter. (b) Output from Sobel filter.

3.3.1 The Sobel filter

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency. It is typically used to perform edge detection. The operator consists of a pair of 3×3 convolution kernels, shown in figure 3.3. The kernels are applied separately to the input image to produce measurements of the gradient components, $G_x(x, y)$ and $G_y(x, y)$. A Sobel filter takes the magnitude of the gradient at each pixel, $G(x, y) = \sqrt{|G_x(x, y)|^2 + |G_y(x, y)|^2}$. Figure 3.4 shows example output from a Sobel filter.

3.3.2 The Hough transform

The Hough transform [43, 45] is a technique which can be used to isolate features of a particular shape within an image. The basic Hough transform requires that the desired features be specified in some parametric form and is most commonly used for the detection of lines, circles and ellipses.

Assume that we have an image that is nearly all black, and contains a small fraction of white pixels, or features, which nearly all to lie on a parameterized curve. For example a circle, whose parameters are not known. These features could be selected from an image by thresholding the output from a Sobel filter.

How can we locate the assumed circle? From a Bayesian perspective, each white pixel gives evidence in favour of some hypothesized locations of the circle, and against other hypotheses; each black pixel also provides evidence against some locations, and weak evidence in favour of others. The Hough transform approximates the ideal likelihood computation by visiting each white pixel and enumerating all the hypothesized locations with which that pixel is compatible. For example, if the circle is known to have radius 5, then one white pixel is compatible with the centre of all the circles being at distance of 5 pixels.

The transform is implemented by quantizing the Hough parameter space into finite intervals or accumulator cells. Each feature is transformed into a discretized curve and the accumulator cells which lie along this curve are incremented. The peak in the accumulator identifies the maximum likelihood parameters of the assumed curve.

3.3.3 Demonstration of the circular Hough transform

We show the application of the circular Hough transform to an image of the eye in figure 3.5(a). The task is to locate the positions of the pupil and the 2 corneal reflections. For the purpose of this demonstration, we use the true values of the radii; a pupil radius of 16 pixels and a corneal reflection radius of 3 pixels.

The features were obtained with the use of a Sobel filter; the output of which is shown in figure 3.5(b). All pixels with an intensity greater than 10% were selected as features for the Hough transform.

In figure 3.5(c) we show the Hough transform for a radius of 15 pixels. The circle corresponding to the maximum intensity is shown in figure 3.5(d). In figure 3.5(e) we show the

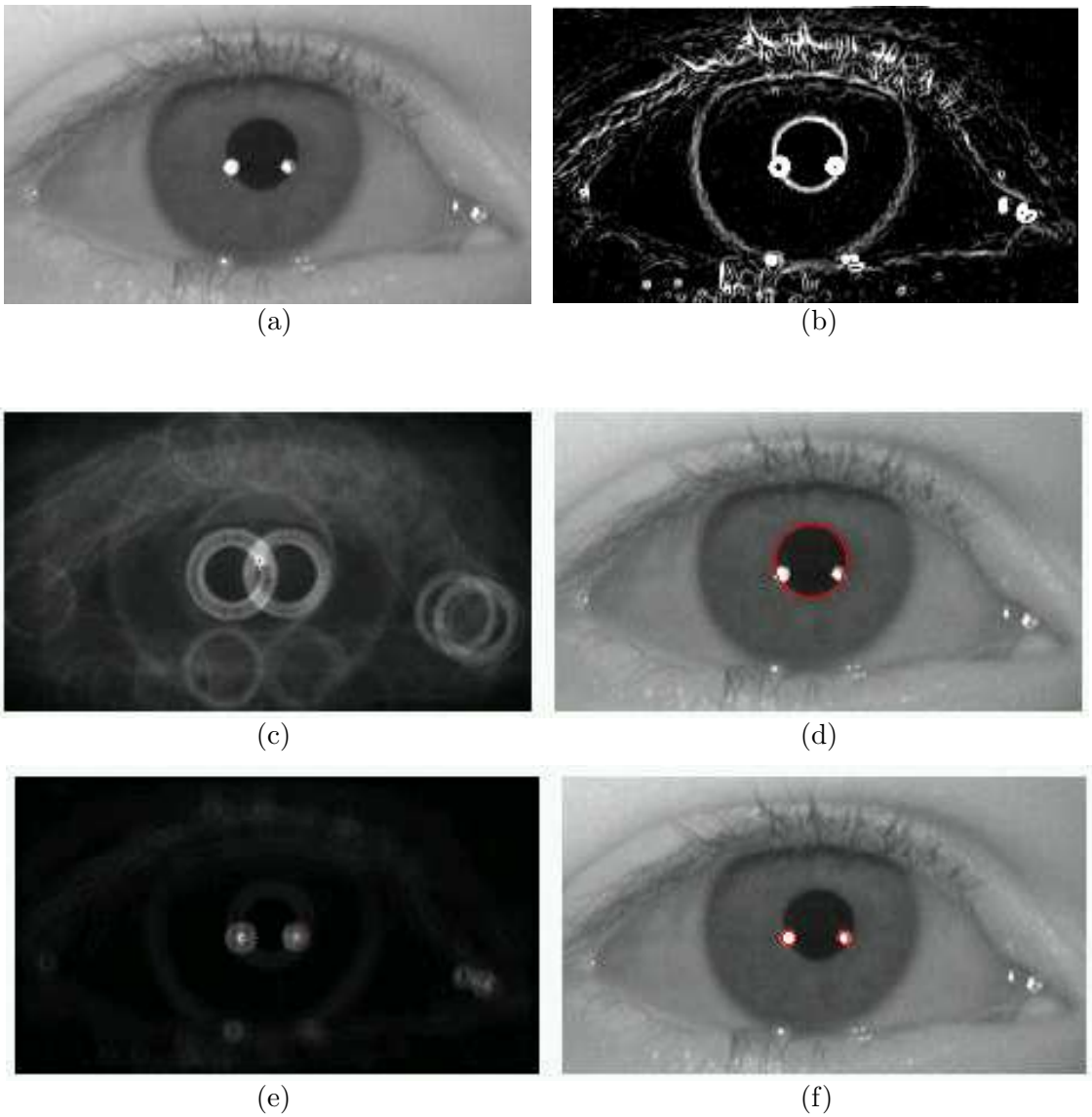


Figure 3.5: Demonstration of the circular Hough transform on an eye image. (a) Original image of the eye. (b) Output from Sobel Filter. (c) Hough transform with radius of 15 pixels. (d) Red circle corresponds to the peak in the Hough transform. (e) Hough transform with radius of 3 pixels. (f) Red circles correspond to the 2 largest peaks in the Hough transform.

Hough transform for a radius of 3 pixels. The circles corresponding to the two largest maxima are shown in figure 3.5(e).

3.4 Practical Considerations for Eye-Tracking

3.4.1 The 'Midas Touch' problem

Eye movements, like other passive, non-command inputs (e.g. hand gestures and speech) are often non-intentional or subconscious. Therefore, they must be interpreted carefully to avoid unwanted responses.

The problem with a simple implementation of an eye-tracking interface is that people are not used to operating devices with their eyes. They expect to be able to look at an item without any action occurring. Of the ways of managing this problem, the use of a dwell threshold is the most common [33, 91].

3.4.2 Screen cursor

Another interface design issue is whether the system should provide a screen cursor that follows the user's eye position. If the calibration and response speed of an eye-tracker is perfect, feedback would not be necessary, since a person knows exactly where he or she is looking. In addition the image of a cursor would become stationary on the user's retina and thus disappear from perception.

However, all eye-tracking systems are subject to calibration error, and a screen cursor will be slightly offset from where the user is actually looking. If the user's gaze is drawn to the cursor, this will lead to further displacement, creating a positive feedback loop. Despite this behaviour, it is often useful to display a screen cursor.

3.5 Quick Glance

The Quick Glace system is an eye-tracker manufactured by Eyetech Digital Systems. It is designed as a mouse replacement. Mouse clicks are emulated by a slow blink, an eye dwell, or a hardware switch.

Figure 3.6 shows the Quick Glance system. It consists of a camera which is mounted on a computer monitor with two infra-red emitters on either side. The LED's output infra-red with a wavelength of about 880nm. The camera is focused on one eye and it is connected to a PC via a PCI card.

The Quick Glance software processes the camera image to determine where the user is looking (figure 3.7). Once the system is calibrated (section 3.5.2), the eye gaze can be computed from the location of the pupil, and the two corneal reflections. The cursor is then placed at the calculated gaze point. The sampling rate is 30 Hz and the claimed accuracy is $\pm 1^{\circ}$.

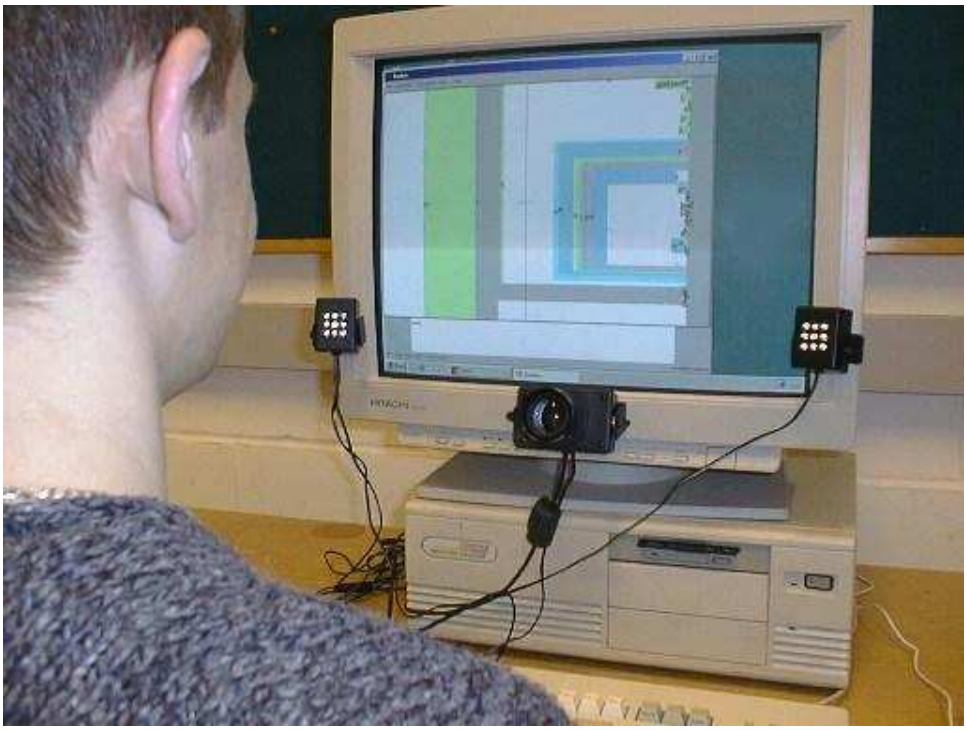


Figure 3.6: Quick Glance eye-tracker. A standard monitor, two infra-red emitters, and an infra-red camera are shown. The camera is focused on one eye, and connects to a PC via a PCI card.

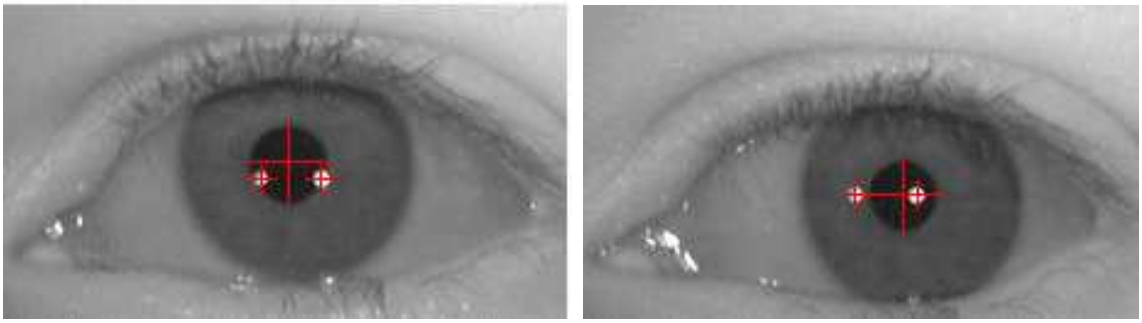


Figure 3.7: Processed images from the infra-red camera. Left: the user is looking forwards. Right: the user is looking down and to one side. The red crosses show the positions of the pupil and corneal reflections.

3.5.1 Head movement

Since the camera's position is fixed, the eye must be kept within the camera's field of view; about 4.5 cm width by 3.5 cm height at a working distance of 55 cm from the screen. A chair with a headrest makes this fairly easy to achieve.

3.5.2 Calibration procedure

The software displays 16 targets on the screen, which the users looks at in succession. The calibration can be saved and used for subsequent sessions, but we found it necessary to recalibrate at the start of each session.

Chapter 4

Dasher - a Data Entry Interface Using Continuous Gestures and Language Models

4.1 Introduction

Existing devices for communicating information from people to computers are either bulky, slow to use, or unreliable. Dasher is a new interface incorporating language modelling and driven by continuous two-dimensional gestures, *e.g.* a mouse, touch screen, or eye-tracker. Tests have shown that experienced users can enter text at a rate of up to 39 words per minute during a dictation task, compared with typical ten-finger keyboard typing of 40–60 words per minute.

Although the interface is slower than a conventional keyboard, it is small and simple, and can be used on personal data assistants and by motion-impaired computer users.

Dasher has a website, `http://www.inference.phy.cam.ac.uk/djw30/dasher/`, which has a non-interactive demo. Dasher can be downloaded for Microsoft Windows and UNIX platforms.

4.2 Entering Text

Dasher is intended for writing short documents, *e.g.* memos and emails. We first describe how Dasher is used to enter the word '`the`'. Figure 4.1(a) shows the initial configuration, with an alphabet of 27 characters displayed alphabetically in a column. There are 26 lower case letters and the symbol '`_`' represents a space. The user writes the first letter by making a gesture towards the letter's rectangle. The trails show the user moving the mouse towards the letter '`t`'.

In a continuous motion, the point of view zooms towards the point indicated by the mouse (figure 4.1(b)). As the rectangles get larger, possible extensions of the written string appear

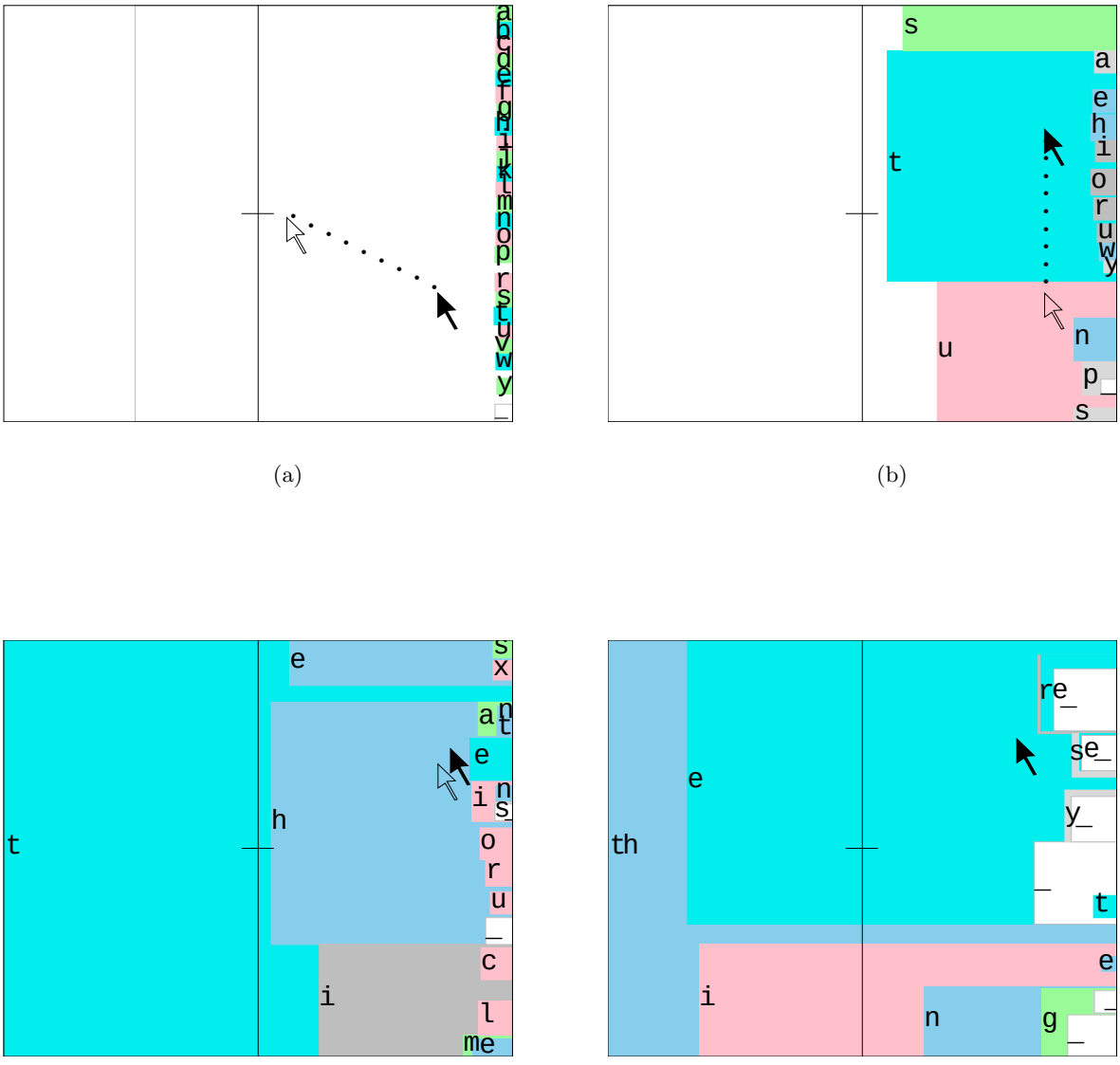


Figure 4.1: (a) Initial configuration. (b),(c),(d) Three successive screen shots from Dasher showing the string '`the`' being entered. The symbol '`_`' represents a space.

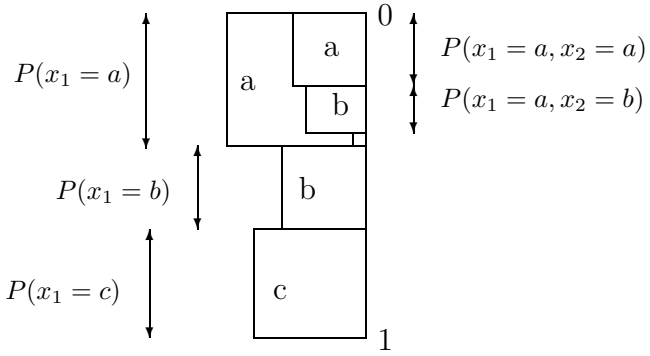


Figure 4.2: Arithmetic coding with an alphabet of {a,b,c}.

within the rectangle that we are moving towards. So if we are moving into the '`t`', rectangles corresponding to '`ta`','`tb`','`tc`',...,'`tz`' appear in a vertical line like the first line. The heights of the rectangles correspond to the probabilities of these strings according to the language model. In English, '`ta`' is quite probable, '`tb`' is less so, '`th`' is very probable. So it is easy to gesture our point of view into '`th`' (figure 4.1(c)), and from there into '`the`' (figure 4.1(d)).

4.3 How the Probabilistic Model Determines the Screen Layout

In a given context, we display the alphabet of possible continuations as a column of characters as shown in figure 4.2. The division of the right-hand vertical is analogous to arithmetic coding [4]. Let our alphabet be $A_X = \{a_1, a_2, \ldots, a_I\}$ (in figure 4.2 we show a smaller alphabet of {a,b,c}). We divide the real line [0,1) into I intervals of lengths equal to the probabilities $P(x_i = a_i)$. We subdivide the interval a_i into intervals denoted $a_i a_1, a_i a_2, a_i a_3, \ldots, a_i a_I$, such that the length of the interval $a_i a_j$ is

$$P(x_1 = a_i, x_2 = a_j) = P(x_1 = a_i)P(x_2 = a_j | x_1 = a_i). \tag{4.1}$$

The language model described in section 4.6 assigns the probabilities. The interval [0,1) can be divided into a sequence of intervals corresponding to all possible finite length strings $x_1 x_2 \ldots x_n$, such that the length of an interval is equal to the probability of the string given our model.

This sequence of intervals corresponds to the alphabetically ordered sequence of books in Borges' 'Library of Babel' [13], with the size of each book being proportional to the probability of its contents under the language model. The user writes by gliding into the library and selecting the desired book.

In figure 4.2, the rectangles are shown as squares, but we can use rectangles with any aspect ratio.

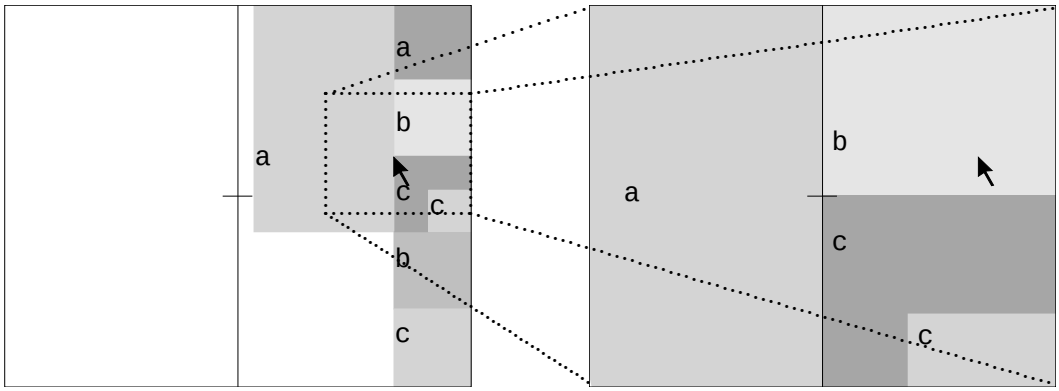


Figure 4.3: Dynamics of the interface, assuming the user holds the pointer at a constant screen location. After S frame updates, the point of view has zoomed in so that the point originally under the mouse is now under the cross-hair. We call S the Steps parameter.

4.4 Dynamics of the Interface

4.4.1 The Steps parameter, S

The pointing device controls a continuous zooming-in of the user's point of view. The interface zooms in so that the place under the pointer passes through the cross-hair S frame updates later (figure 4.3). We call S the 'Steps parameter'.

The left-right coordinate controls the rate of zooming-in, and the vertical coordinate determines the point on the right-hand vertical that is being zoomed into. The intuitive idea is that the user should point where they want to go.

4.4.2 Dynamical equations

We use two coordinate systems to describe the dynamics of the interface. The first coordinate system specifies the *visible interval* on the real line (introduced in section 4.3). The second are the *screen coordinates*.

In the initial configuration, the visible interval is [0,1). The visible interval after i screen updates can be specified by the centre of the interval, C_i, and the height, H_i (figure 4.4). In the initial configuration,

$$C_0 = 0.5, \tag{4.2}$$

$$H_0 = 1. \tag{4.3}$$

Figure 4.5 shows the screen coordinate system. The position of the pointer, (m_x,m_y), is specified in screen coordinates. The position of the cross-hair is $(k_x,0.5)$. The following dynamical equations give the next interval, (C_{i+1}, H_{i+1}), in terms of the current interval,

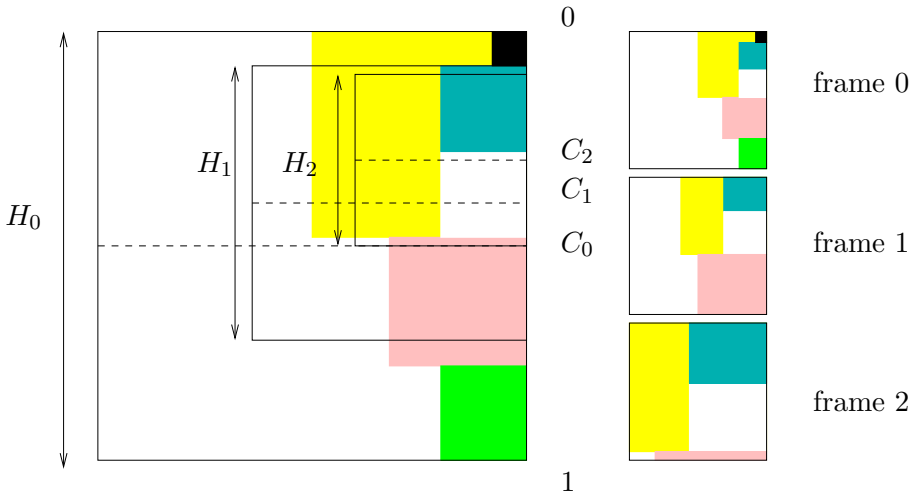


Figure 4.4: Dynamics of the interface. The left diagram shows a view of the whole interval. The point of view is specified by a visible interval on the real line. The interval has a centre, C_i and a height, H_i. The left figure shows three successive intervals and the right figure shows the user's view.

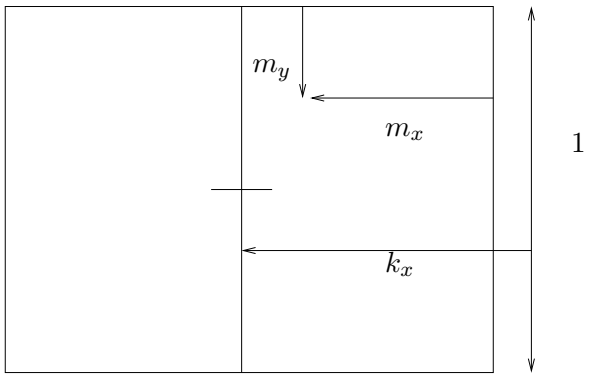


Figure 4.5: Screen coordinates. The height of the display is one screen unit.

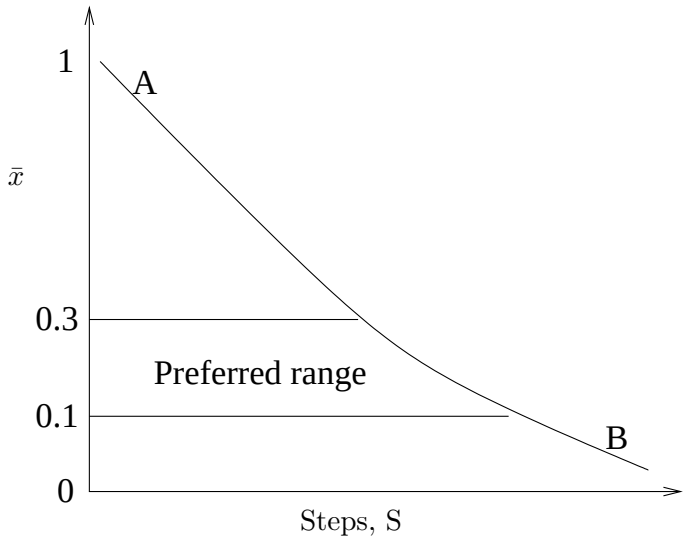


Figure 4.6: Mean horizontal pointer position, $\bar{x}$. The graph qualitatively shows a typical user's response, in terms of $\bar{x}$, to different values of S when they are using Dasher to write.

(C_i, H_i):

$$r = \frac{m_x}{k_x}, \tag{4.4}$$

$$r' = r^{1/S}, \tag{4.5}$$

$$H_{i+1} = r' H_i, \tag{4.6}$$

$$C_{i+1} = C_i + H_i(0.5 - m_y)\frac{1 - r'}{1 - r}. \tag{4.7}$$

4.4.3 Setting the Steps parameter, S

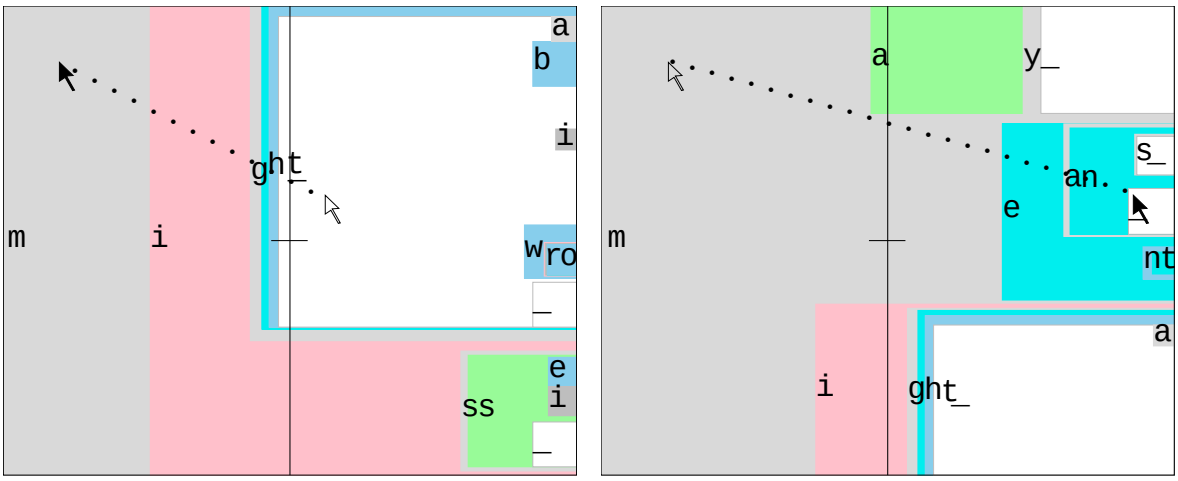
The Steps parameter determines how responsive the interface is; the smaller it is, the faster the point of view can be changed. Inexperienced users usually require a relatively large value of S, say 70 (assuming a frame rate of around 40 frames per second). A high value of S makes the system relatively unresponsive to large or inaccurate movements. As users improve, S can be decreased, enabling higher writing speeds.

However, when measuring their writing speed, we decided not to allow users to set S, otherwise their performance would be subject to their ability to choose this parameter as well as their skill in using the interface.

The rate of zooming-in or out depends on the frame rate so the appropriate value of S is system dependent. We designed an on-line algorithm to set S to a suitable value. Let x be the normalized distance of the pointer from the right hand side of the screen,

$$x = \frac{m_x}{k_x}. \tag{4.8}$$

This normalization ensures that x is scale invariant. Figure 4.6 qualitatively shows a typical



(a) (b)

Figure 4.7: Correcting mistakes. The user has written '`might` instead of '`mean`'. In figure 4.7(a), the user points at '`m`' to go back. In figure 4.7(b), the user points at '`mean`'. The open arrowhead indicated the start of the motion; the closed arrowhead the end.

user's response, in terms of $\bar{x}$ to different values of S while they are writing with Dasher. At point A on the graph, S is relatively small. The interface is too responsive and the user keeps the pointer close to the cross-hair, $x = 1$. At B, S is relatively large. The interface is not responsive enough; the user is able to hold the pointer near the right hand vertical, $x = 0$.

Experience shows that the best writing speeds are obtained in the range $0.1 < \bar{x} < 0.3$, where $\bar{x}$ is the average over the last N frames. Therefore, we used the following update rule for S;

$$\text{if} \quad \bar{x} > 0.3, \quad S \rightarrow S + 1; \tag{4.9}$$

$$\text{if} \quad \bar{x} < 0.1, \quad S \rightarrow S - 1. \tag{4.10}$$

We can alter the time-scale of adaptation by adjusting the size of the sample, N. In practice, the time-scale was around 15 seconds.

4.5 Correcting Mistakes

Inexperienced users of Dasher often inadvertently follow a path other than that corresponding to the desired text. This generally happens when the desired string contains a high probability string followed by a low probability string. The user may react too late, inadvertently accepting a high probability string in the neighbourhood of the desired text. After entering

a high-probability neighbour, the desired text may not even be visible.

There are a number of ways in which we could handle corrections. Currently, the dynamics allow the user to 'back up' by pointing to the left of the cross-hair. The higher the leftward offset from the cross-hair, the faster the zooming-out.

In figure 4.7, the user has written the string '`might`' instead of '`mean`'. To alter this string, the user points to the left side of the display. Then the user points at the desired string, '`mean`'. An alternative mechanism, not yet tested, is to use an extra character to indicate that a misspelling has occurred.

4.6 The Language Model

When choosing a language model for Dasher we considered two qualities: how well the language model compresses text and the time taken to compute the probability of a character. Dasher must calculate many probabilities each time the screen is updated.

The model in Dasher versions 1.0-1.6 is based on a popular text compression algorithm called prediction by partial match (PPM), reviewed in section 1.5. PPM is a context-based algorithm; it uses the preceding characters to predict the next one. The maximum size of the context is called the order of the model. We use a variant of the algorithm called PPMD5 [93], which is fifth order, and can compress most English text to around 2 bits per character. There are slightly better algorithms [36] but PPMD5 is simple and fast.

Given a context, we compute the probabilities of all the symbols in the alphabet. The conditional probabilities determine the intervals in equation 4.1.

When using Dasher, it is difficult to select a character that has a very small probability. Therefore we add a small fixed probability δ to the probability of every character and then renormalize the probabilities. In Dasher 1.6, the default value of δ is 0.002.

As the user enters text the characters can be fed back into the PPM algorithm, hence adjusting the future probabilities in accordance with that user's vocabulary.

4.7 Benefits of Continuous Gestures and Language Modelling

Dasher is driven by continuous gestures, so inaccurate gestures can be compensated for by later gestures, the way a driver keeps a car on the road. When using a conventional keyboard we select one character per gesture but in Dasher some gestures select more than one character so Dasher has the potential to convey information at higher rates than a keyboard. Figure 4.8 shows completions of the string '`object`'. Grammatically correct continuations can easily be selected with a single gesture.

The language model make spelling mistakes less likely, as will be demonstrated in section 4.10.

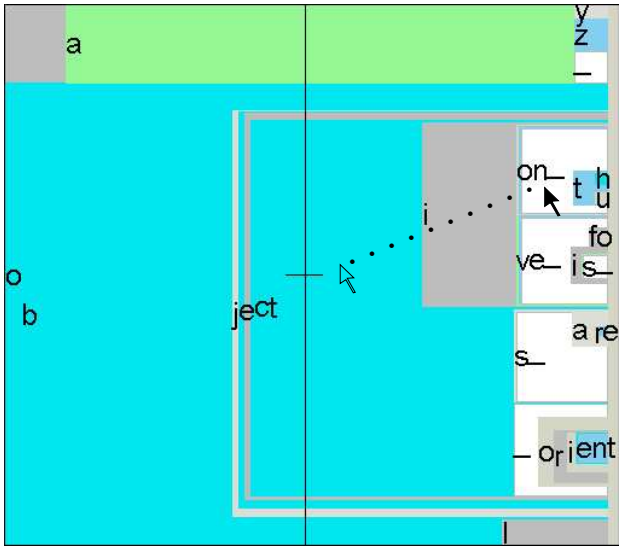


Figure 4.8: A single gesture can select several characters. Continuations of the string '`object`' include '`objection`', '`objects`', '`object_and`' and '`object_of`'.

4.8 Horizontally Modified Display

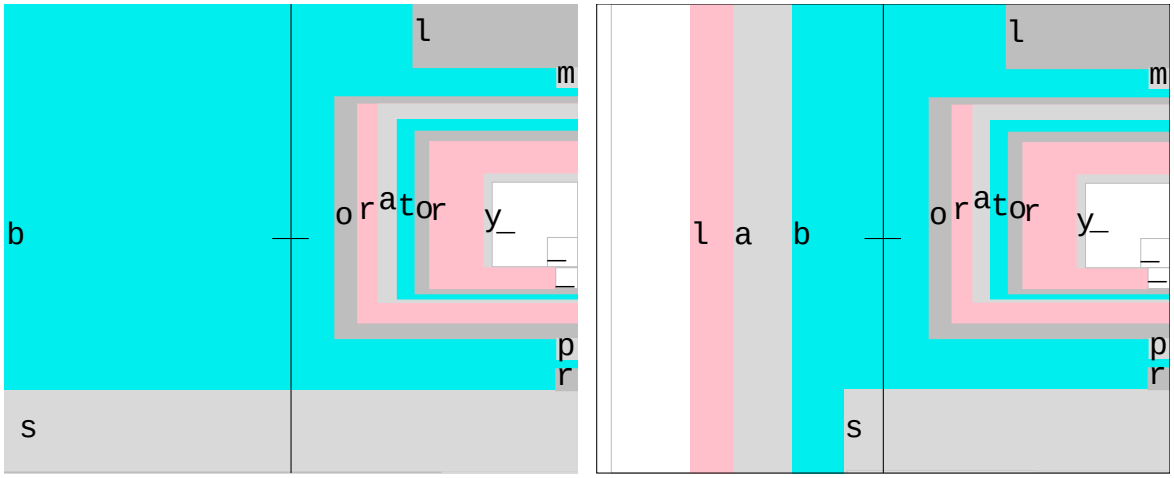
When testing Dasher version 1.0, many users found that they were not able to see an adequate history and that the maximum possible rate of back-tracking was too slow. In figure 4.9 for example, the user writes '`laboratory`' and the letters '`la`' pass out of the display before finishing the word.

In version 1.3, we put the horizontal coordinates of the squares through a non-linear mapping (figure 4.10) before rendering them on the screen (figure 4.9(b)). The mapping is linear on the right hand side, and logarithmic on the left hand side:

$$x' = \left\{ \begin{array}{ll} Cx & x \leq B \\ C\left[A \log\left[\frac{x+A-B}{A}\right] + B\right] & x > B \end{array} \right\} \tag{4.11}$$

The function is specified by 3 parameters, A, B, and C. A specifies the degree of non-linearity, B specifies the point at which the non-linearity begins, and C gives the aspect ratio. The function was constructed to be continuous in x' and its first derivative. A, B, and C were set by hand, but can be adjusted. In Dasher 1.6, the default values are $A = 0.01$, $B = 0.5$ and $C = 1$.

When using the modified display, the place under the pointer will still pass through the cross-hair in S frame updates, as in the original display. With the modified aspect ratio, it will take less frame updates to zoom out to the '`l`' of '`laboratory`' when pointing at the same screen coordinates. Therefore, the modified display gives a higher rate of zooming-out and it is easier for users to go back and correct mistakes.



(a) (b)

Figure 4.9: The user writes '`laboratory`'. (a) Square aspect ratio. (b) Modified display obtained by a non-linear mapping of the horizontal coordinates.

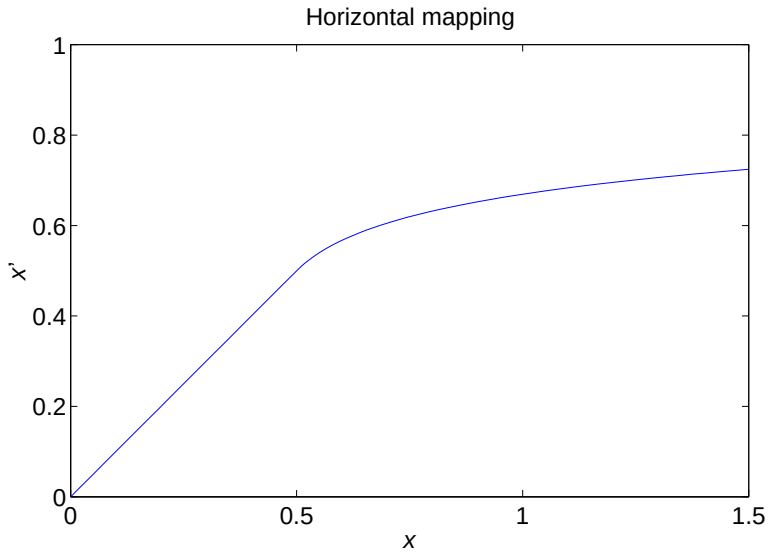


Figure 4.10: Horizontal mapping from old coordinates, x, to new coordinates x'. Both are distances from the right hand side of the display.

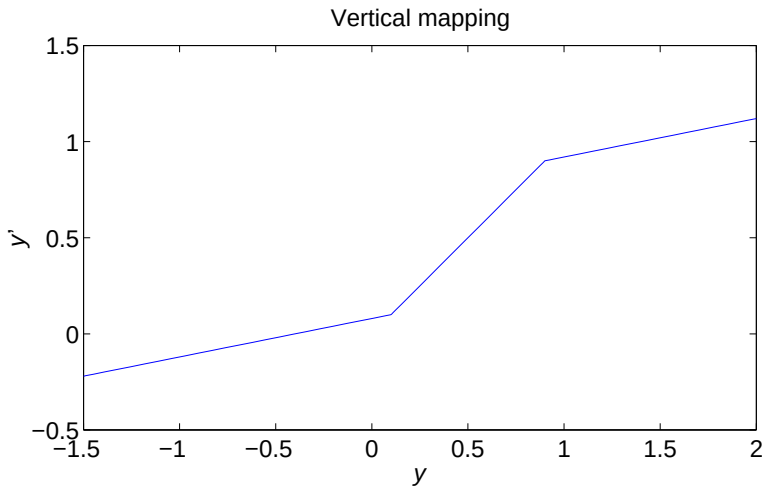


Figure 4.11: Vertical mapping from old coordinates, y, to new coordinates y'.

4.9 Vertically Modified Display

In Dasher 1.5 onwards, we put the vertical coordinates through a non-linear transformation as shown in figure 4.12. Its functional form is specified by two parameters, D, and E:

$$y' = \left\{ \begin{array}{ll} D + (y - D)E & y < D \\ y & D \leq y \leq 1 - D \\ 1 - D + (y - 1 + D)E & y > 1 - D \end{array} \right\} \tag{4.12}$$

D controls the position where the gradient changes. E gives the gradient at the edge of the display. Both parameters were set by hand. In Dasher 1.6, the default values are $D = 0.9$ and $E = 0.1$.

This vertically modified display increases the maximum speed of vertical scrolling, and gives users more time to react before a desired letter disappears out of view.

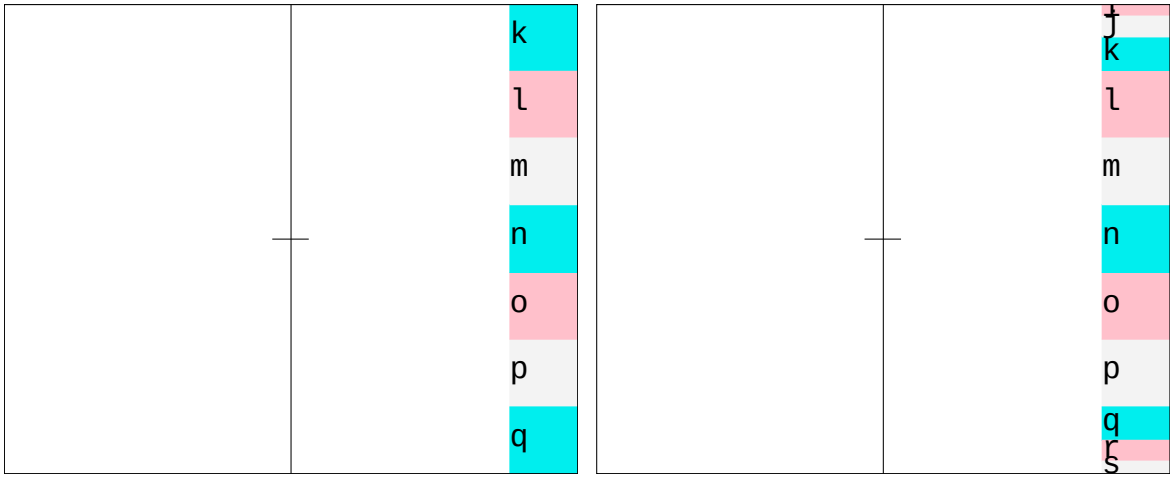
4.10 Empirical Evaluation

The evaluation below was carried out on Dasher version 1.3 and was initially presented at UIST 2000 [97].

4.10.1 Evaluation approach

An objective of this research was to assess Dasher in a realistic text entry task. Dasher requires uninterrupted visual attention, so we measured the writing speed for dictated text.

Dasher has been used with a variety of position controllers. For the experiment, we originally hoped to use a stylus on a touch screen the size of a hand-held computer. However, our original touch screen required a contact force of 2 ounces. This is difficult to maintain



(a) (b)

Figure 4.12: Figure to demonstrate vertically modified display. (a) Square aspect ratio. (b) Modified display obtained by a non-linear mapping of the vertical coordinates.

when using a continuous gesture interface such as Dasher, so we used a standard mouse and computer monitor in the experiment described below.

4.10.2 Pilot experiment

We carried out a pilot experiment in which users transcribed text passages spoken by a speech synthesizer. The speed of dictation was dynamically controlled to stay just ahead of the words being entered by the user.

The results of the three-subject pilot experiment were used estimate the length of training that would be required to observe significant learning. Subjects found the speech synthesizer hard to understand, so we used recorded human speech for the main experiment.

4.10.3 Method

Subjects

The main evaluation experiment involved ten experimental subjects, recruited from the students and staff of the Cavendish Laboratory. All subjects had vision corrected to normal and spoke English as their first language. Subjects were paid for participating in the experiment. None of the subjects had previous experience with Dasher.

Task

The experimental task was to enter text dictated from Jane Austen's *Emma*. We selected 18 extracts at random from the total 883kB of text in *Emma* to form the test set. The remaining 866kB of text was used to train the language model.

Austen has a distinctive writing style. This allowed us to test Dasher in a reasonable simulation of typical usage where the language model would have been trained on previous prose written by the same user.

Apparatus

Platform	Pentium II 300 MHz running Linux 2.0.38
Monitor	38cm LCD display
Input Device	mouse

The dictated passages were recorded as a series of audio files, with each file containing a short phrase. The text entered by the subject was monitored so that as he or she wrote the penultimate word in each phrase, the next audio file was played automatically. This allowed subjects to enter text continuously, with dictation proceeding at a comfortable speed. A simple synchronization algorithm compensated for mis-spelt words when identifying the end of a dictation phrase. Subjects could press a key with their free hand to repeat the last phrase if it was forgotten or misheard. There was no limit to the number of repeats.

Procedure

Subjects completed 6 experimental sessions of approximately 20 minutes each. Each session consisted of three exercises. Subjects first took dictation using Dasher for 5 minutes. They then took dictation using the keyboard (conventional typing) for 2 minutes. Finally they took dictation using Dasher for another 5 minutes. Sessions were generally spaced at daily intervals, with no more than two on any day and no more than three days between sessions. Subjects were instructed at the start of each exercise to write as fast as possible, and were told that they could correct simple mistakes within the current word. They were told not to correct mistakes in previous words.

Configuration

Dasher was configured with the following parameters:

Dasher version	1.6.4
Display	600×500 pixels, measuring 12cm×10cm
Character set size	27 (lower case and space)

The δ parameter, (defined in section 4.6), specifies a lower limit to the probability of a character. For each exercise, the Steps parameter S (defined in section 4.4) was initialized to 60. Definitions of the parameters A, B and C can be found in section 4.8. Definitions of the parameters D and E can be found in section 4.9.

Analysis

The control software counted the numbers of characters entered in each period of dictation. We used a common convention to obtain writing speeds in words per minute by dividing the character rate by 5 [63]. The software also counted word-level errors; a deletion, insertion, or misspelling of a word counted as an error. The Viterbi algorithm [5, 96] was used to determine the minimum number of errors which, when applied to the target text, produced the text entered by the user.

Data on character entry speed and proportion of errors were collected for the twelve periods of Dasher use, and for the six periods of conventional typing. The interpolated conventional typing tests allowed us to estimate what proportion of this practice effect could be attributed to practice with the experimental dictation software, as opposed to practice with the Dasher text entry method.

The keyboard task should be viewed as a control condition for the experimental procedure, rather than a serious attempt to measure keyboard speeds under realistic conditions of use.

4.10.4 Results

Text entry speeds

All subjects successfully completed the 6 sessions (figure 4.13). The dictation texts were used in the same order for all subjects. As a result, variation in vocabulary between the texts resulted in consistently lower speeds for some exercises. For example, exercise 8 included a particularly uncommon word which caused users to write more slowly. As a result, the raw data shown in figures 4.13, 4.16, and 4.17 include dips in writing speed during those exercises.

To investigate these correlations further, we plot information rate rather than writing speed in figure 4.14. Uncommon words have a higher information content, so information rate is relatively continuous despite discontinuities in writing speed.

Writing errors

We the percentage of words that were written differently from the test text. The rate of keyboard errors in figure 4.15 show that users took a little while to get used to the dictation system (assuming that their typineir typing skill did not improve during the experiment). However, after the third keyboard exercise, the errors in both Dasher and keyboard exercises change very little.

Towards the end of the evaluation the average error rate is 3% when using Dasher and 7% when using the keyboard. In 10 exercises, Dasher users made no errors while keyboard users invariably made errors.

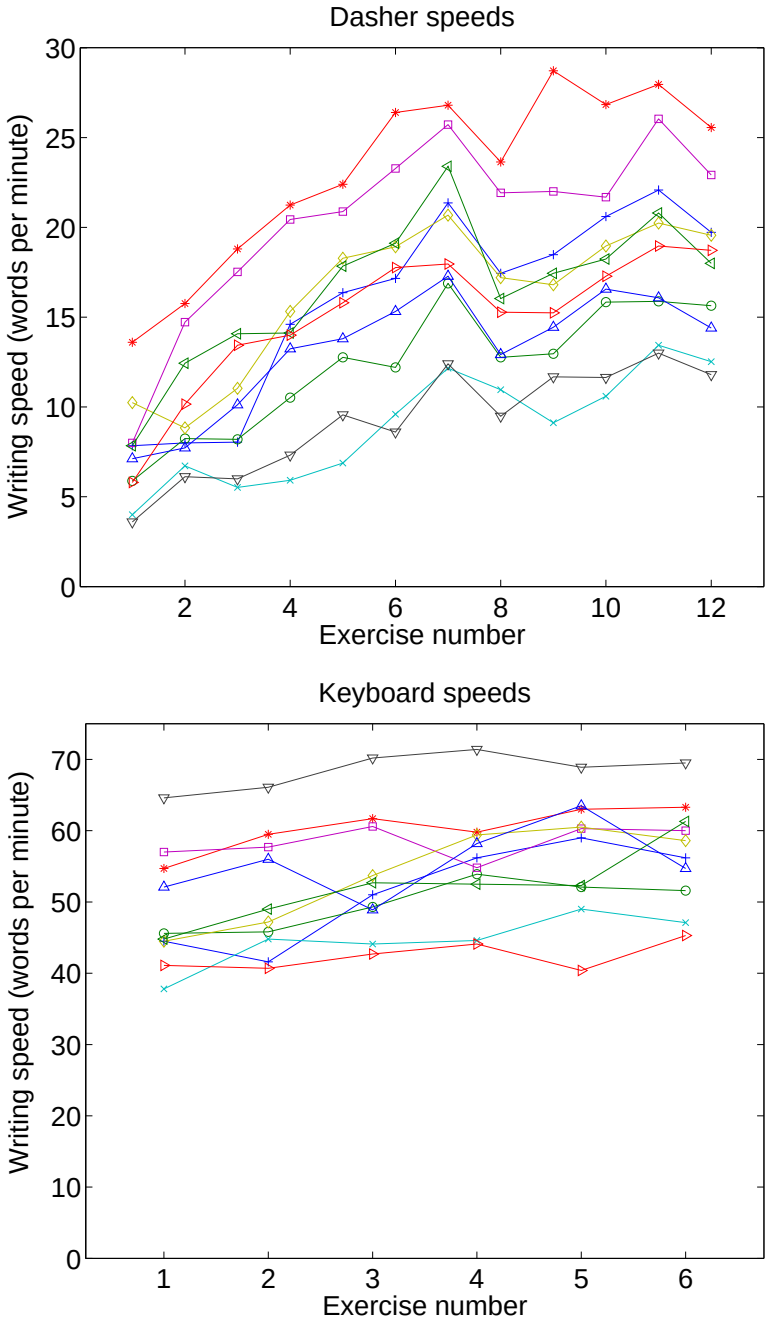


Figure 4.13: Effect of practice on writing speeds for ten subjects. Each Dasher exercise was 5 minutes long. The keyboard exercises were 2 minutes long.

Figure 4.14: Dasher information rate. Each exercise was 5 minutes long.

The power law of practice

The power law of practice [76] predicts that the writing speed dN/dt should increase with training time t as

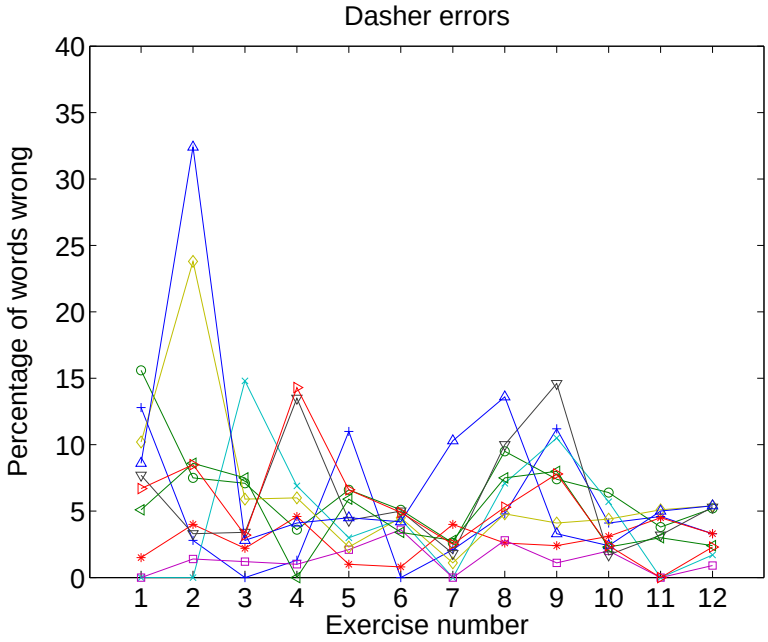
$$\log\left[\frac{dN}{dt}\right] = \alpha + \beta \log(t), \tag{4.13}$$

where α and β are constants. From figure 4.16 we can see that the data give a reasonably good fit to the power law. The gradient $\beta = 0.32 \pm 0.05$ (related to the learning rate) is similar for all 10 subjects, but $\alpha = 3.22 \pm 0.45$ (the log writing speed after one minute) varies more among subjects. The users with faster initial writing speeds were generally the fastest after an hour's training. Consequently, we can estimate a user's future writing speed with reasonable accuracy, based on a small amount of use, say 15 minutes.

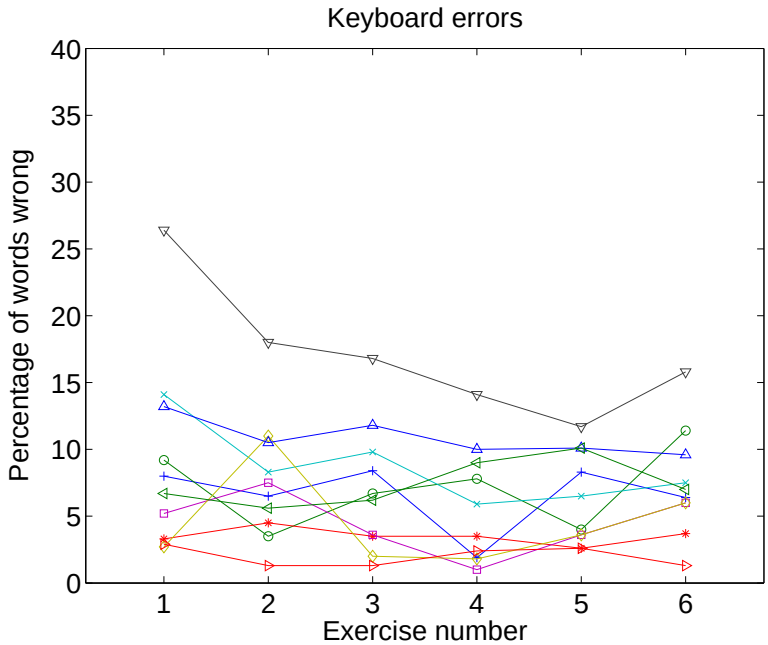
4.10.5 Comparison to other devices

Where previous researchers have reported improvement of performance over time, we can compare Dasher with other devices. We have extracted data from a number of earlier published evaluations of text entry interfaces and compare the results to Dasher in figure 4.17.

In a relatively short training time, writing performance with Dasher already exceeded that of many methods. If we extrapolate a power law fit (a straight line on figure 4.17), it seems that with further practice, Dasher performance may also equal the fastest method



(a)



(b)

Figure 4.15: Writing Errors

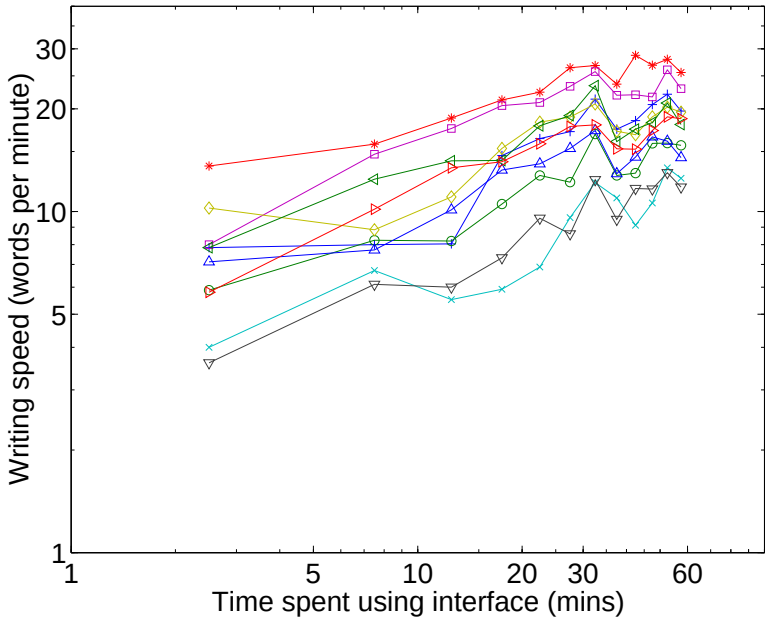


Figure 4.16: Logarithmic plot to test the Power Law of Practice. Dasher results are plotted for all ten subjects

using QWERTY keyboard derivatives.

4.11 Input Devices

Dasher could be used with any 2-D pointing device, for example, a mouse, touch screen, joystick, eye-tracker, or trackerball. We describe how they could be used with Dasher.

4.11.1 Mouse, eye-tracker and non-centring trackerballs

The challenge here is to provide a suitable method of starting and stopping the interface. The dynamics described in section 4.4 allow the user to stop writing by returning to the cross-hair. However, it is awkward for a user to have to align the pointer perfectly when they finish writing. In Dasher 1.0, we used the space bar or left mouse button to start and stop the interface.

When using the start/stop mode of operation, users must remember to switch off the interface before interacting with another application, or when leaving the computer. In Dasher 1.5, we implemented an automatic system to overcome this problem. When the system is idle and the user moves into the Dasher window, the interface starts. If the user doesn't move the mouse for a preset time, Dasher goes idle. If the user points out of the window and stays there for a preset time, then the interface stops.

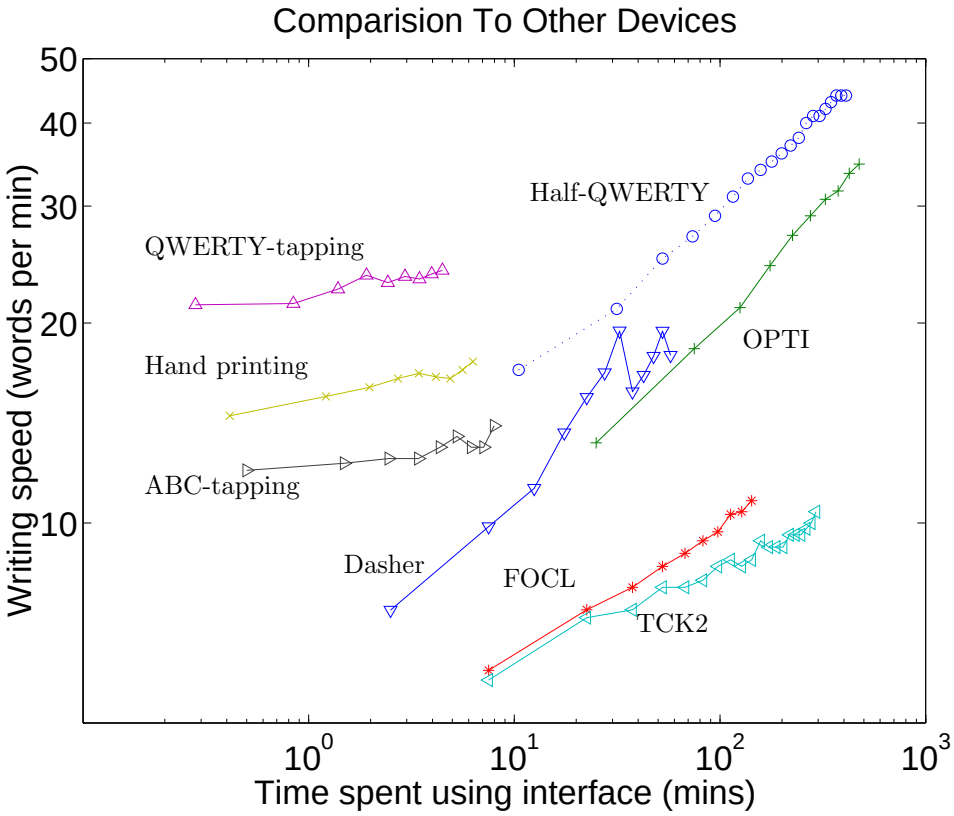


Figure 4.17: Comparison of Dasher to other devices. Hand printing required the user to print letters on a touch screen. In QWERTY-tapping, a stylus selects characters from an on-screen QWERTY keyboard. ABC-tapping uses a stylus with letters in alphabetical order. Half-QWERTY uses half a QWERTY keyboard, with the other half accessible with a special key. FOCL is a keyboard designed with a probabilistic character layout strategy. TCK2 is a version of the ternary chorded keyboard. OPTI is a soft-keyboard with an optimized layout.

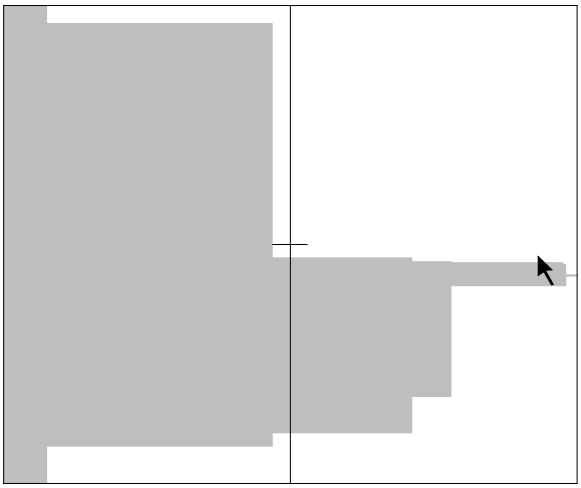


Figure 4.18: Modified version of Dasher to measure writing speed without visual search

4.11.2 Touch screen

A touch screen registers contact with the surface and we can take advantage of this behaviour in the following way. When the pen is up, Dasher does nothing. When the pen is down, Dasher updates the screen according to the usual dynamics, described in 4.4.

4.11.3 Self-centring trackerballs and joysticks

In this situation, we set up Dasher so that when the device is in the central position, the pointer is over the cross-hair. In this configuration, the interface will only operate while the device is being used.

4.12 Information Rate of Dasher

4.12.1 Current information rate of Dasher

I have used Dasher for substantially longer than the experimental training period; a total of about 10 hour's use, mostly during development testing rather than sustained text entry. During a number of trials with the experimental dictation text from Jane Austen's *Emma*, I achieved an average speed of 170 characters per minute (34 words per minute).

The experimental texts have an information content of 1.7 bits per character with respect to the model. Expert performance of 170 characters per minute therefore corresponds to an information rate of 4.8 bits per second.

4.12.2 Potential information rate of Dasher

Apart from the language model, two factors might limit the rate at which a user can enter information with Dasher. First, steering Dasher in the required direction involves visual

motor control similar to pole-balancing; the timescale of the eye-to-pointer feedback loop imposes a maximum writing speed. Second, a limit in Dasher might be the time required for the user to search among the presented strings. We performed an experiment to estimate the maximum writing speed of Dasher when this visual search is not required, so only the first factor applies. The required sequence of squares was highlighted in a strongly-contrasting colour (figure 4.18), and the I user guided Dasher along this sequence as fast as possible. The required sequence of rectangles and their sizes were identical to those displayed when writing the experimental text from *Emma*. The information content of the required writing path is therefore identical to that of the experimental text.

When operating Dasher as shown in figure 4.18, I was able to 'write' at a rate of 228 characters per minute. This rate corresponds to an information rate of 6.5 bits per second which is 1.3 times faster than the dictation speed when visual search is required.

Earlier we cited an estimate of 8.2 bits per second for the rate at which information can be conveyed by one-dimensional pointing. Although Dasher is a two-dimensional interface, we only use the vertical dimension to select letters. The horizontal co-ordinate determines the rate of zooming in and out.

4.13 Time-Delay Model

We now present a quantitative model of the information rate limit imposed by the visual-motor feedback loop. A person is trying to track an object at screen coordinates $y(t)$ with a pointer at $u(t)$. The object runs away from the pointer in accordance with:

$$\frac{dy}{dt} = l\,(y - u)\,, \tag{4.14}$$

where l is the exponential growth rate of the deviation. We model the best performing person as a tracker with a delay:

$$u(t) = y(t - \tau). \tag{4.15}$$

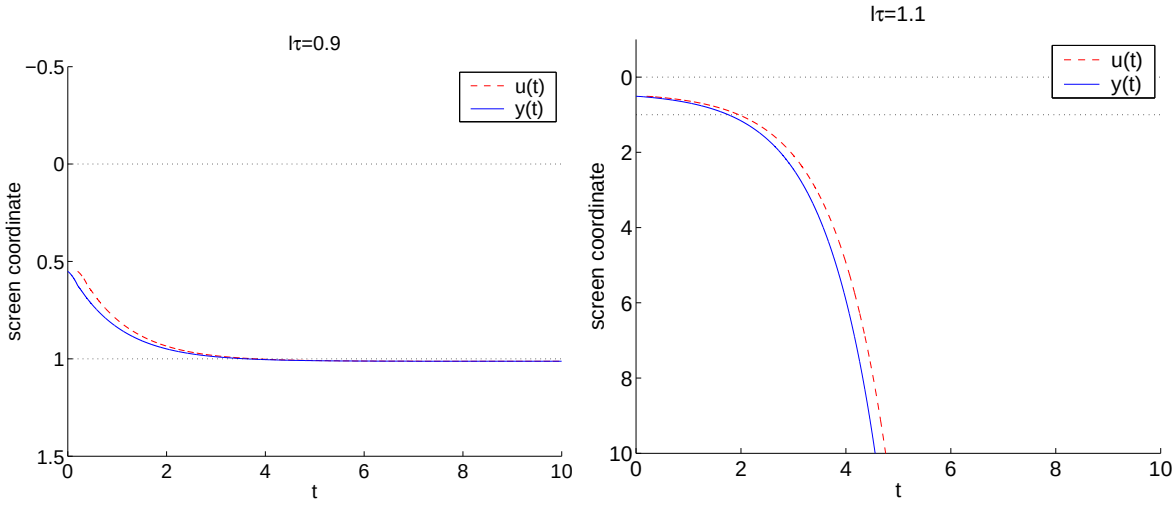
Substituting for $u(t)$ in equation 4.14,

$$\frac{dy}{dt} - ly(t) + ly(t - \tau) = 0. \tag{4.16}$$

This is a type of delay differential equation with well known properties [6]. Analysis by Laplace transform shows that $y(t)$ is stable if and only if

$$l\tau < 1. \tag{4.17}$$

The tracking problem is identical to using Dasher, where l is the expansion rate of Dasher, assumed constant. If the visible interval at time $t = 0$ has size 1, then after time t, the visible



(a) (b)

Figure 4.19: Numerical demonstration of stability condition, $l\tau < 1$. (a) Stability with $l\tau = 0.9$. (b) Instability with $l\tau = 1.1$. Dotted lines show the edges of Dasher's display.

interval has size

$$p = \exp(-lt). \tag{4.18}$$

The information content is defined as $-\log_2(p)$, so the information rate of Dasher is

$$\frac{-\log_2(p)}{t} = \frac{l}{\log_e 2} \quad \text{bits per second.} \tag{4.19}$$

Applying the stability condition $l\tau < 1$, we find an expression for the maximum information rate, M.

$$M = \frac{1}{\tau \log_e 2} \quad \text{bits per second.} \tag{4.20}$$

If we assume an eye-to-hand reaction time of 175 milliseconds [80], then the maximum information rate is $M = 8.2$ bits per second. This theoretical figure is a little larger than our measured information rate of 6.5 bits per second for our most expert user (section 4.12.2).

In Dasher version 1.0, users could drive the interface arbitrarily fast by holding the mouse near the right hand vertical. From the analysis of the time-delay model, it would seem appropriate to set an upper limit on the rate of expansion. In version 1.6, there is an option to set the maximum information rate. The default value is 8 bits per second.

4.13.1 Numerical solutions

To explore the stability of the system in more detail, we computed numerical solutions to the delay differential equation. We used the trapezoidal Runge-Kutta method (see appendix D).

Figure 4.19 shows two numerical solutions to the differential equation. Figure 4.19(a)

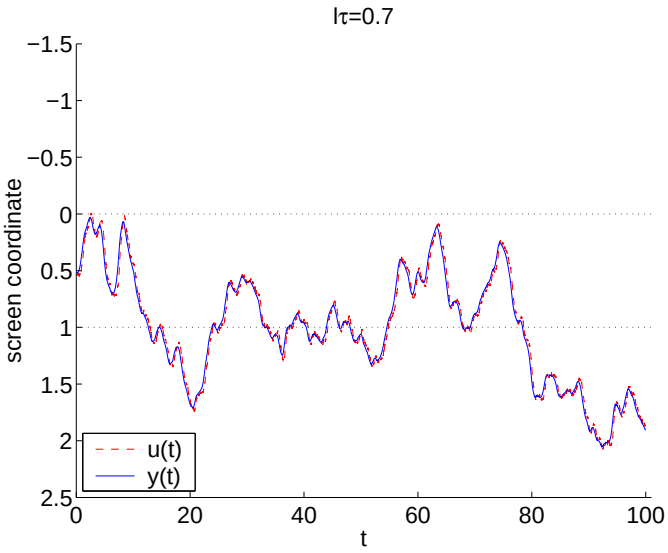


Figure 4.20: Numerical simulation of tracking with human error. Both y and u move outside of the Dasher display, which is shown by dotted lines.

shows stability with $l\tau = 0.9$ and figure 4.19(b) shows instability with $l\tau = 1.1$. These results agree with the stability condition, equation 4.17.

Human error

A human tracker is not a perfect tracker with delay, as assumed by equation 4.15. If the position of the pointer is modified by an error, denoted by $e(t)$, then

$$u(t) = y(t - \tau) + e(t). \tag{4.21}$$

We will assume that the error is random, with some correlation time, τ. One method of generating an error function is by using the following iterative method:

$$e_{t+1} = \lambda e_t + (1 - \lambda)\eta_t, \qquad t > 0, \quad 0 < \lambda < 1 \tag{4.22}$$

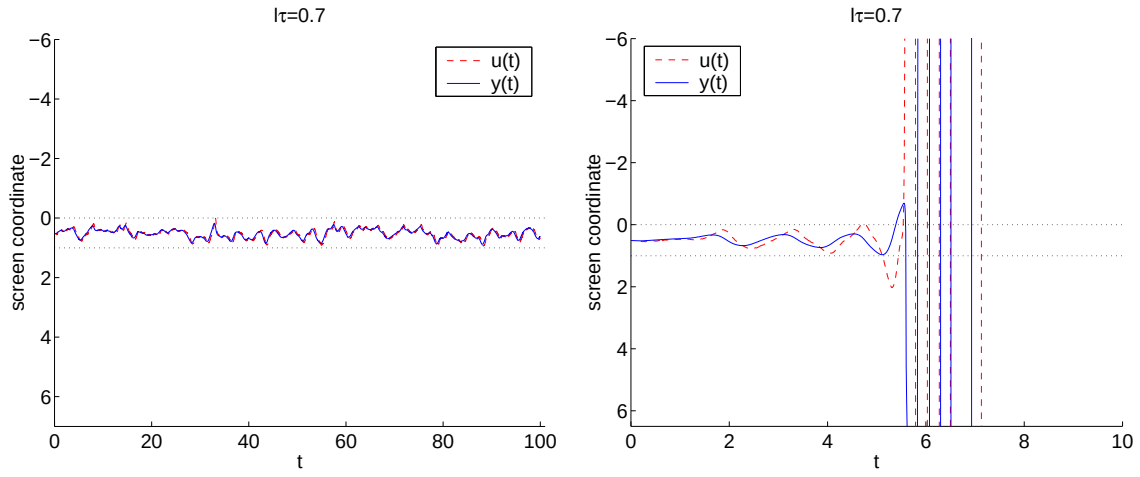
The N Gaussian deviates, $\{\eta_t\}_{t=0}^{N}$ have mean zero and variance $\langle \eta^2 \rangle$ and we define $e_0 = \eta_0$. The result is a correlated error function with time constant

$$\tau = \frac{\Delta t}{1 - \lambda}, \tag{4.23}$$

where Δt is the size of the time step in the numerical solution.

Figure 4.20 shows a numerical solution for $l\tau = 0.7$ and a noise level of $\langle \eta^2 \rangle = 0.5$. The graph indicates that $y(t)$ performs a random walk with displacement $\propto \sqrt{t}$. The Dasher display is of finite size so we predict that the target will eventually move off one of the edges.

When performing the evaluation of Dasher we observed short bursts of rapid writing,



(a) (b)

Figure 4.21: Numerical simulation of tracking with human error and quadratic correction to vertical mouse coordinate. (a) When $\alpha = 0.5$ both $y(t)$ and $u(t)$ are confined. (b) When $\alpha = 5$, the $u(t)$ falls out of phase with $y(t)$, resulting in oscillatory behaviour.

separated by periods in which the user had to slow down. Sometimes, users couldn't find the next letter, but there were also occasions when the point of interest, $y(t)$ drifted slowly off the top or bottom of the display. Experienced users learnt to prevent this by over-correcting towards the top and bottom of the display, thereby returning $y(t)$ to the centre of view. However, we would prefer to solve this problem by modifying the interface.

Object tracking with spatial confinement

When using Dasher, the target text must be confined to the visible display. In our time-delay model, this corresponds to upper and lower limits to the value of $y(t)$. Therefore, we investigate methods of maintaining $y(t)$ between finite limits.

One solution to the problem is to over-correct as the target approaches the limit. Rather than have the user over-correct, we change the effective position of the pointer. We could add a linear correction to the y coordinate. However, such a correction would alter the 'point to where you want to go' behaviour everywhere on the screen. With a small quadratic correction the dynamics are only significantly affected towards the edges of the display. Let m_y be the true vertical coordinate of the pointer, and let m_y' be the effective coordinate which is used in the dynamical equations 4.7:

$$m_y{}' = m_y + \alpha \frac{(m_y - 0.5)^3}{|m_y - 0.5|}, \qquad \alpha > 0 \tag{4.24}$$

The parameter α determines the degree of non-linearity. In figure 4.21 we show the effect

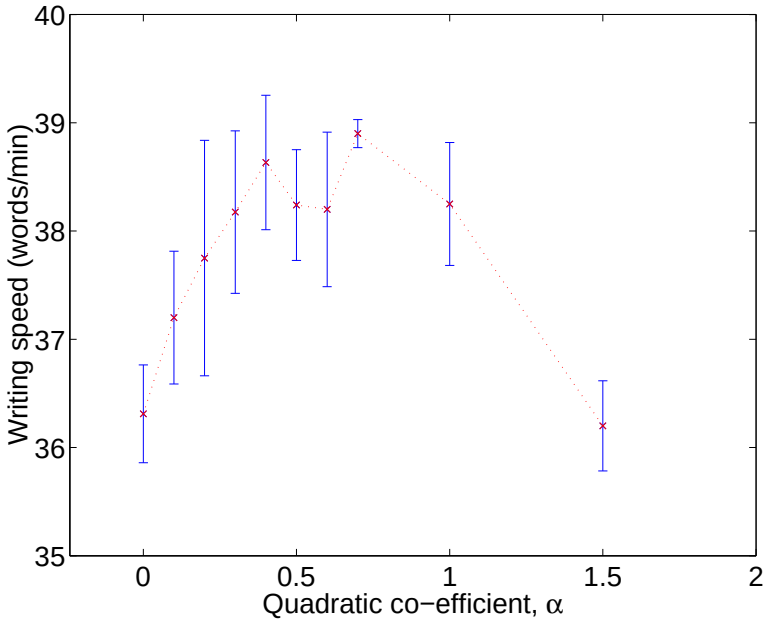


Figure 4.22: Writing speed with quadratic correction to the vertical co-ordinate. Error bars are the standard deviation of the mean.

of the quadratic correction. When setting $\alpha = 0.5$ (figure 4.21(a)), the random walk behaviour is suppressed and y and u are confined to the display. However, a larger value of α can lead to instability (4.21(b)). Encouraged by the suppression of the random walk in figure 4.21(a), we implemented the quadratic correction into Dasher version 1.6. We asked a user to experiment with different values of α. He found that the Dasher was only usable when $0 < \alpha < 2$.

I took part in a number of experimental sessions; each consisted of taking 5 minutes' dictation using Dasher. A standard mouse was used as the pointing device. Each time, I selected a random α (in between 0 and 2) and a random dictation text, to avoid a preference for any single value of α. In total, 5 dictations were carried out on each value of α. We plot the mean writing speeds with error bars in figure 4.22. The results show the benefit of using the quadratic correction, with a 7% increase in writing speed when $\alpha = 0.5$.

4.14 Alternative Character Sets

It is easy to incorporate any character set into Dasher. We give two examples.

4.14.1 Capitalization

We can extend the character set introduced in section 4.2 to 53 symbols by adding capital letters (figure 4.23). Here, we show Dasher 1.5 with the lower and upper case letter interleaved, but we could place the capital letters after the lower case letters. Notice how the language model predicts capitals where they are probable (after '`Mr_`') and lower case letters after

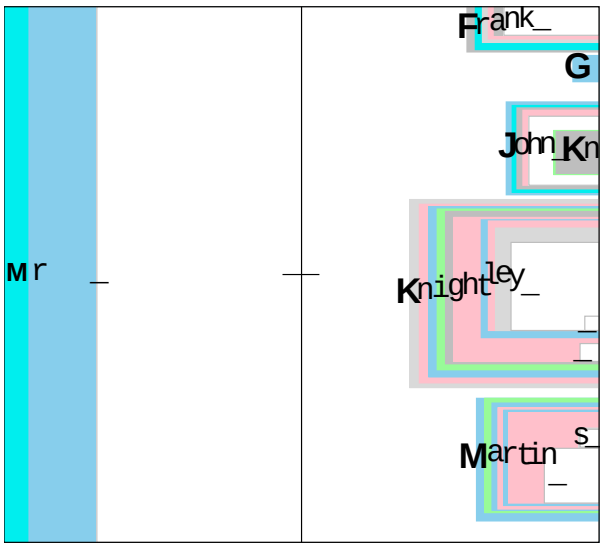


Figure 4.23: Dasher with capital letters.

'Mr_K','Mr_J', etc.

An expert user took dictation and wrote at 42 words per minute with lower case and 41 words per minute with both upper and lower case. It appears that the performance of Dasher is not significantly altered, even though size of the character set has doubled.

4.14.2 Japanese

Japanese is written in three different scripts.

- **Kanji** are Chinese characters or ideograms. Each character represents a word.
- **Hiragana** is a system in which each symbol represents a spoken syllable.
- **Katakana** is a phonetic alphabet, similar to Hiragana, but used for writing words of non-Japanese origin.

There are thousands of Kanji symbols, so for our first prototype, we decided to use the Hiragana character set. In the future we plan to make a combined Hiragana-Kanji system in which Kanji characters are selected by their relative probability as the completion of a phonetic Hiragana spelling. Hiragana consists of 83 symbols, shown in figure 4.24. The symbols are listed in their conventional order for computers.

While Hiragana keyboards are available, many Japanese users type on conventional QWERTY keyboards. Each Hiragana symbol has a Romanisation, *e.g.* ka, ki, ku. Typically, two key presses are required to enter one Hiragana symbol. We carried out a small survey of Japanese people to find out the typing speed for Hiragana entered with a QWERTY keyboard. A typical typing rate is about 60 to 120 Hiragana per minute. We trained a language model on 200kB of Japanese text, and estimated the information content of Hiragana to be

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x8290																あ
0x82A0	あ	ぃ	い	ぅ	う	ぇ	え	ぉ	お	か	が	き	ぎ	く	ぐ	け
0x82B0	げ	こ	ご	さ	ざ	し	じ	す	ず	せ	ぜ	そ	ぞ	た	だ	ち
0x82C0	ぢ	っ	つ	づ	て	で	と	ど	な	に	ぬ	ね	の	は	ば	ぱ
0x82D0	ひ	び	ぴ	ふ	ぶ	ぷ	へ	べ	ぺ	ほ	ぼ	ぽ	ま	み	む	め
0x82E0	も	ゃ	や	ゅ	ゆ	ょ	よ	ら	り	る	れ	ろ	ゎ	わ	ゐ	ゑ
0x82F0	を	ん														

Figure 4.24: Shift-JIS character set for Hiragana.

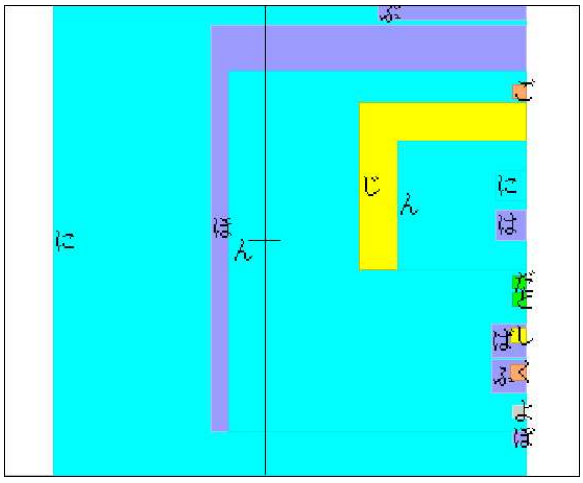


Figure 4.25: JDasher - Hiragana character set.

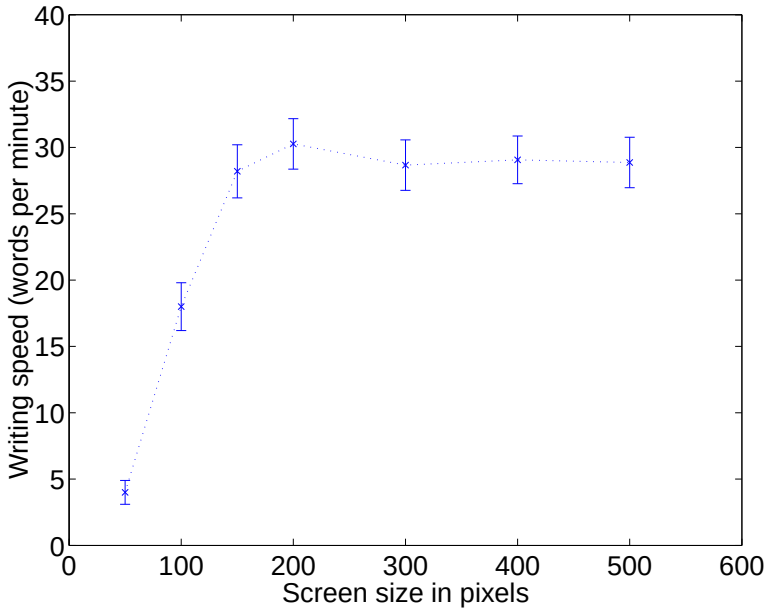


Figure 4.26: Touch screen writing: effect of screen size. Error bars are the standard deviation of the mean writing speed.

2.5 bits per character. The current rate of writing with Dasher is about 4.8 bits per second, so we might expect a writing speed of 2 Hiragana per second or 120 Hiragana per minute.

The Hiragana version, 'JDasher', shown in figure 4.25 is incorporated into Dasher versions 1.5 onwards. The character set can be split into 10 classes, based on sound. For example, the characters (ka, ki, ku, ke, ko) and (sa, shi, su, se, so) form two different classes. In Dasher, we gave each class a fixed colour to aid the user's search.

We used two orderings; the first ordering is identical to that shown in figure 4.24. In an alternative character set, we separated the diacritical marks from the base characters. Evaluation of JDasher is continuing.

4.15 Mobile Text Entry

4.15.1 Viability of Dasher as a hand-held text entry device

We used a variable sized window on a 38 cm LCD touch screen to simulate the use of Dasher on a Personal Data Assistant (PDA). This touch screen was not ideal, as it requires an unreasonably large 2 ounces of force.

The size of the output display was varied from 50×50 pixels (1.6cm by 1.6 cm) to 600×600 pixels (20cm by 20cm). Our screen had 30 pixels per cm, similar to the 27 pixels per cm on the Handspring Visor, a Palm compatible PDA. For each size, an experienced user performed 5 dictation tasks. In each session of 5 minutes' writing, a random screen size and dictation text was selected. We plot the mean writing speeds, with error bars, in figure 4.26.

The writing speed decreases rapidly when the screen size is decreased below 150×150 pixels. Increasing the screen size beyond 200×200 pixels does not appear to be advantageous. The results of the evaluation encouraged us to implement Dasher on a PDA. We constructed a shortlist of requirements for the target platform:

- Screen of at least 150 × 150 pixels.
- Colour screen preferred.
- Sensitive touch screen.
- Fast processor.

Initially, we considered the Palm Pilot platform. However, we found out that animated objects appeared blurred and therefore the platform is unsuitable for an animated version of Dasher.

4.15.2 Dasher 2.0

The first hand-held matching our list of specifications was the Compaq iPAQ:

- 206 MHz Intel StrongARM RISC processor.
- TFT LCD display, 240 × 320 pixels, 4096 colours.
- Touch screen.
- Operating system: Windows Pocket PC (CE 3.0).
- 32MB memory.
- Overall dimensions: 13.0cm×8.3cm×1.6cm.

We therefore ported Dasher to the Pocket PC (`http://www.pocketpc.com`) operating system. Dasher 2.0 for Pocket PC can be downloaded from

`http://www.inference.phy.cam.ac.uk/djw30/dasher/` .

Two screenshots of Dasher version 2.0 are shown in figure 4.27. The alphabet consisted of lower case letters, upper case letters, digits, and some punctuation. The ordering is shown in figure 4.28. Strongly contrasting coloured boxes were placed around capital letters, digits, and punctuation symbols to make these groups easier to find (figure 4.29).

Input method

The Compaq iPAQ has a touch screen and the stylus interacts with Dasher in the following way. When the pen is up, Dasher is idle. When the pen is down, Dasher updates the screen according to the usual dynamics, described in 4.4. This behaviour is simple and intuitive.

Figure 4.27: Screenshots of Dasher 2.0 running on a Compaq iPAQ H3630.

Lower case	abcdefghijklmnopqrstuvwxyz
Upper case	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Digits	1234567890
Punctuation	. newline
Space	_

Figure 4.28: Character set order in Dasher 2.0

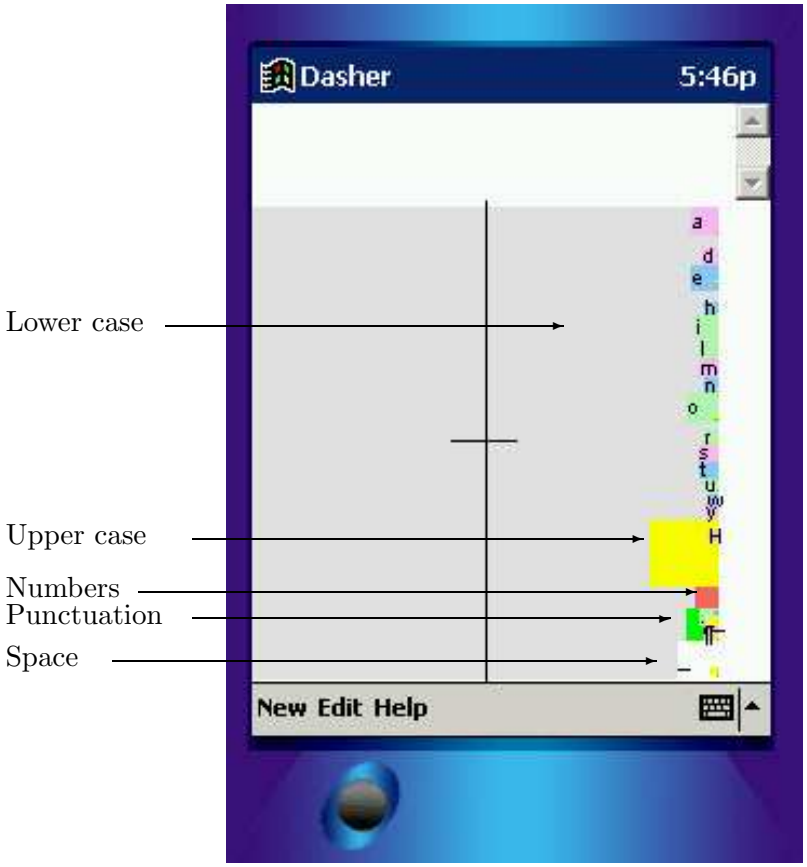


Figure 4.29: Character set of Dasher 2.0 running on a Compaq iPAQ H3630.

Fonts

In the desktop version of Dasher, we had quite a large display of around 400×400 pixels and a fixed font size was used. The Compaq iPAQ has a smaller resolution of 240×320, of which the Dasher display occupies 240×220 pixels. We therefore reduced the font size on the right-hand side of the display. This enables Dasher to display characters in rectangles which are just a few pixels in height.

Language model

The current implementation of PPM was originally designed to run on a desktop computer and was optimized for speed rather than memory consumption. The data structure created from a 1MB training file, for example, occupies 8MB of memory.

The Compaq iPAQ has 32MB of RAM, while other brands of Pocket PC's have 16MB. To reduce memory consumption, we reduced the order of the model to 4. The resulting performance of the language model is about 10% worse in compression, while memory consumption is typically halved.

Teahan [92] found that trie-based implementations of PPM typically grow in size with the square root of the training text. In a later paper, [22], he discusses methods of reducing the memory consumption of PPM. He reported that most models consumed less memory than the original text. In future, we will implement a more compact version of PPM.

Using Dasher as a text entry device

In previous versions of Dasher, text was transferred to the edit box as it passed the crosshair, but no editing was possible. To make Dasher 2.0 a practical text entry device, users are allowed to alter text in the edit box. When the user moves the cursor to a new location inside the text box, the new context is retrieved from the characters to the left of the cursor. The Dasher display updates accordingly, as shown in figure 4.30. In addition, text in the edit box highlighted by dragging the stylus is replaced when the user starts writing with Dasher.

Evaluation

We performed a small evaluation of Dasher and two built-in input methods. Two users took part in the evaluation and had no experience with the input methods. The Pocket PC operating system includes an on-screen QWERTY keyboard, occupying the bottom 1/3 of the screen. There is also a letter recognizer, which is similar to Jot (see section 2.2.3). Each session consisted of 5 minutes' writing. Users were allowed to write on any subject, but were asked to be consistent in style throughout the experiment. There were 12 sessions in total, giving one hour's use on each system. Writing speeds and errors are shown in figure 4.31. A logarithmic plot of writing speed is also shown.

The results show that Dasher outperforms the letter recognizer. Although Dasher was not as fast as the on-screen keyboard during the period of the evaluation, the logarithmic plot

(a) (b)

Figure 4.30: Improved text editing capability. (a) User has written some text. (b) User taps after the 'H' of 'How' and the display updates to reflect the new context. Words such as 'Hartfield', 'He', 'His', and 'How' are now quite probable.

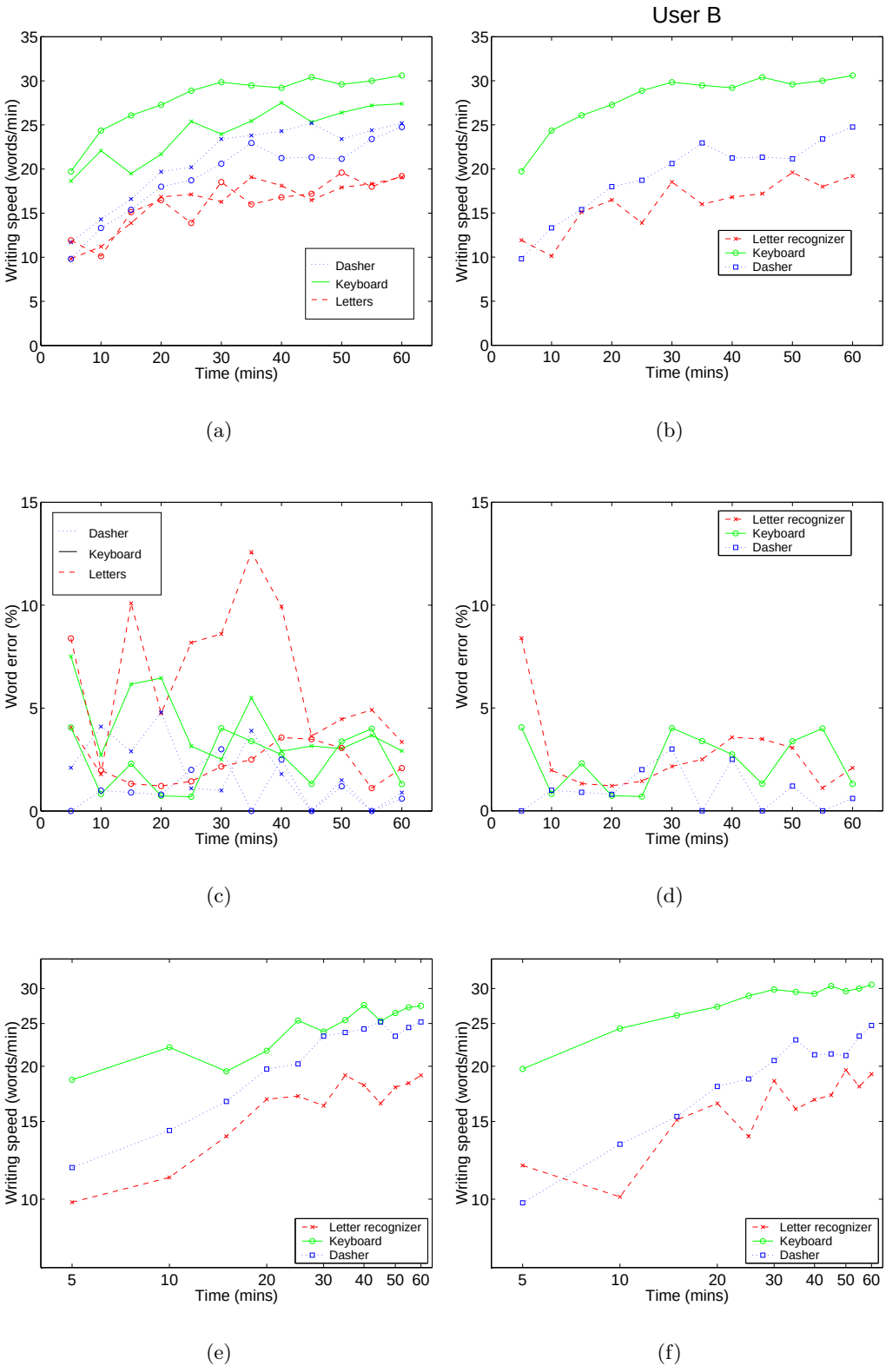


Figure 4.31: Writing speeds for two users on the iPAQ. Results are shown for Dasher, the on-screen QWERTY keyboard, and the letter recognizer. (a) (b) Writing speeds, (c) (d) writing errors and (e) (f) logarithmic plot of writing speeds.

shows that Dasher has a faster learning rate. Further evaluation is necessary to determine whether Dasher will outperform the keyboard. Both users demonstrated that Dasher has a superior error rate.

Expert writing speed

On the desktop version of Dasher, I achieved a writing speed of 39 words per minute during a dictation task (figure 4.21). In a similar task on the Compaq iPAQ, I achieved a writing speed of 29 words per minute. I felt that the frame rate of 10 frames per second was limiting my maximum speed.

Conclusion

Dasher is operated with a single, continuous stream of motor input. The combination of stylus and touch screen is highly suited to Dasher. The user simply points at the string of letters that they want to select. We had hoped that this mode of operation would prove to be faster than a mouse, where the user is using an indirect method of pointing. Although the language model on the Compaq iPAQ was inferior to the desktop version, the current frame rate of 10 frames per second appeared to be limiting the expert writing speed.

A more memory-efficient implementation of the PPM data structure would allow the use of the same language model as the desktop version. We could also improve the language model by using the built-in dictionary or by including our own.

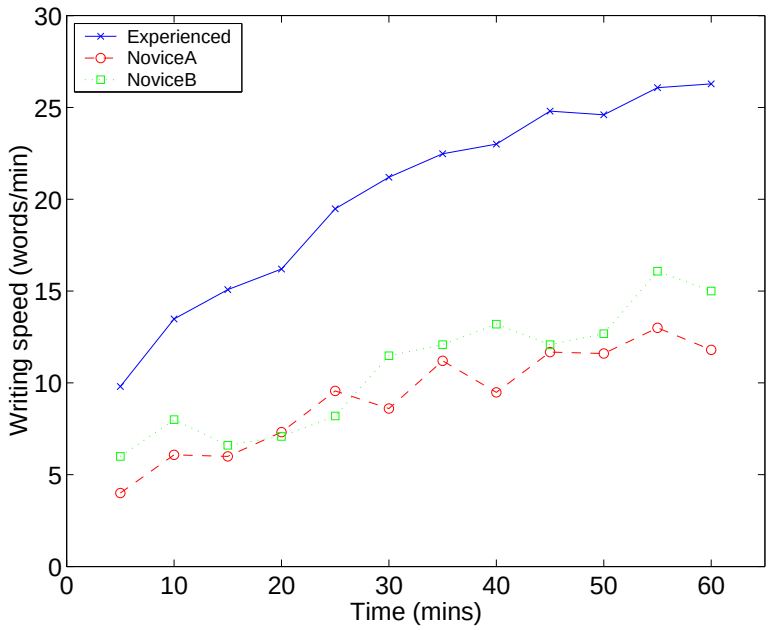
The rendering on the Pocket PC currently uses computationally expensive operating system calls. For example, copying the buffer to the screen takes 50ms. The Pocket PC has a Game API (`http://www.microsoft.com/mobile/developer/`) which allows access to the display memory. Using the Game API will substantially increase the frame rate.

4.16 Eye-tracking

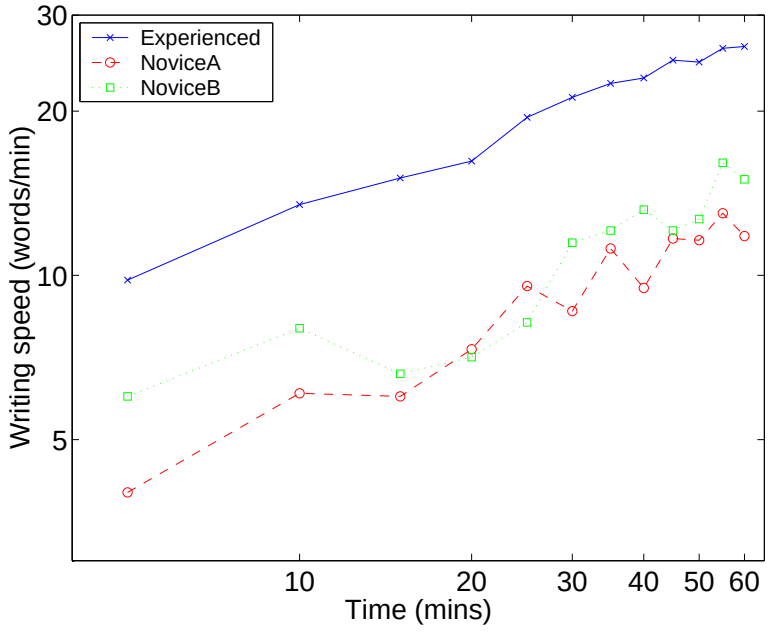
I have developed a system which incorporates an eye-tracking device as an alternative input device to Dasher, enabling hands-free text entry. I currently use the Quick Glance eye-tracker, developed by EyeTech Digital Systems (see section 3.5). This system could be especially beneficial to users with limited hand movement. When Dasher is used with an eye-tracker, users simply look at the string they want to write. We hope that this will be easier and more natural to use than eye-tracking interfaces in which the users must consciously direct their gaze toward action targets, for example Salvucci's Intelligent Gaze-Added Interfaces [82].

4.16.1 Evaluation

I performed a small evaluation of Dasher with the eye-tracker as the input. Three users took part in the evaluation; all had no experience of using the eye-tracker. Two were novice users of Dasher and I took part as an experienced user of Dasher. Each user performed 12



(a)



(b)

Figure 4.32: Eye-tracking results. (a) Writing speeds. (b) Logarithmic plot of writing speeds. All 3 users had no prior experience of using the eye-tracker. The experienced user had used Dasher for a considerable amount of time. The two novice users had no prior experience of using Dasher.

experimental sessions; each consisted of taking 5 minutes' dictation of text from Emma by Jane Austen. I dictated to both novice users and novice user 'A' dictated to me. Writing speeds are shown in figure 4.32.

4.17 Discussion

4.17.1 Attention demand

It is clear that Dasher requires sustained visual attention from the user. This requirement is in common with word completion systems and T9, for example, but in contrast to conventional keyboards which can be used without visual attention after sufficient training.

4.17.2 Potential writing speed

Expert writing performance of 39 words per minute was obtained on the desktop version of Dasher (figure 4.21). The performance of Dasher is determined by two factors. The first is how well the language model can compress the text being entered. Shannon [87] estimated the entropy of English to be 1 bit per character but PPM typically compresses to 2 bits per character. A perfect language model would increase the speed of Dasher by a factor of 2 which would result in a writing speed similar to that of a QWERTY keyboard.

The second factor is the user interface, which can currently convey 4.8 bits per second to the computer (section 4.12.1). It might be possible to improve Dasher to give an information rate of 8.2 bits per second; the maximum rate at which information can be conveyed by one-dimensional pointing. We would expect the writing speed to increase proportionally.

4.18 Conclusion

Dasher is a novel text entry interface which exploits the information redundancy of language, and the human capacity to convey high information rates through fine motor control.

The operation of Dasher is simple, and immediately evident to new users. Furthermore Dasher has a rapid learning rate that is comparable to alternative text entry methods.

We think Dasher shows promise as a keyboard-less text entry system both in its writing speed and its ease of use.

Chapter 5

Extension of the Dirichlet Language Model

The second half of this thesis is concerned with language modelling. The aim is to develop language models which could be used with Dasher.

The Dirichlet language model, described by MacKay and Peto [62], is a Bayesian language model whose predictions are similar to 'smoothing'. In this chapter, we summarize the language model and demonstrate a new method of combining similar contexts.

5.1 The Dirichlet Language Model

A string of T words $D = w_1, w_2, \ldots w_T$, is observed. Define the marginal count F_i to be the number of times word i occurs. Define the conditional count $F_{i|j}$ to be the number of times word j is immediately followed by word i.

A bigram model for W distinct words is described by a matrix of parameters Q:

$$q_{i|j} = P(w_t{=}i|w_{t-1}{=}j). \tag{5.1}$$

A single row of Q, the probability vector for transitions from state j, is denoted by $\mathbf{q_{|j}}$. The parameters are never perfectly known. The task is to infer them from the data, D.

5.1.1 The inferences

Model $\mathcal{H}$ is a specification of the model parameters, the way that the likelihood depends on the parameters, and a prior probability on the parameters.

A: Infer the parameters given the data

Bayes' theorem gives us the posterior probability of the parameters Q given the data D in terms of the likelihood function $P(D|Q,H)$ and the prior distribution $P(Q|H)$.

$$P(Q|D,\mathcal{H}) = \frac{P(D|Q,\mathcal{H})P(Q|\mathcal{H})}{P(D|\mathcal{H})} \tag{5.2}$$

B: Predict the next word given the context

To obtain the probability of w_t given w_{t-1} and the data, we marginalize over the unknown parameters Q.

$$\begin{aligned} P(w_t|w_{t-1},D,\mathcal{H}) &= \int P(w_t|w_{t-1},Q,D,\mathcal{H})P(Q|D,\mathcal{H})\, d^kQ \\ &= \int q_{w_t|w_{t-1}} P(Q|D,\mathcal{H})\, d^kQ \end{aligned} \tag{5.3}$$

5.1.2 The likelihood function

The likelihood is independent of the assumptions $\mathcal{H}$ that define the rest of the model.

$$\begin{aligned} P(D|Q,\mathcal{H}) &= \prod_t q_{w_t|w_{t-1}} \\ &= \prod_j \prod_i q_{i|j}^{F_{i|j}} \end{aligned} \tag{5.4}$$

5.1.3 Definition of the hierarchical model $\mathcal{H}_D$

We introduce an *unknown* measure on the words, $\mathbf{u} = \alpha\mathbf{m}$, and define a separable prior, given $\alpha\mathbf{m}$, on the vectors $\mathbf{q}_{|j}$ that make up Q:

$$P(Q|\alpha\mathbf{m},\mathcal{H}_D) = \prod_j \text{Dirichlet}^{(\mathcal{A})}(\mathbf{q}_{|j}|\alpha\mathbf{m}) \tag{5.5}$$

The Dirichlet distribution was introduced in chapter 1.

Inferring the parameters, Q

We assume we know the hyperparameters $\mathbf{m}$ and α and infer a posterior for Q.

$$\begin{aligned} P(Q|D,\alpha\mathbf{m},\mathcal{H}_D) &= \frac{P(D|Q,\mathcal{H}_D)P(Q|\alpha\mathbf{m},\mathcal{H}_D)}{P(D|\alpha\mathbf{m},\mathcal{H}_D)} \\ &= \prod_j P(\mathbf{q}_{|j}|D,\alpha\mathbf{m},\mathcal{H}_D) \end{aligned} \tag{5.6}$$

The posterior distribution of each conditional probability vector is another Dirichlet distribution:

$$P(\mathbf{q}_{|j}|\mathbf{F}_{|j}, \alpha\mathbf{m}, \mathcal{H}_D) = \frac{P(\mathbf{F}_j|\mathbf{q}_{|j}, \mathcal{H}_D)P(\mathbf{q}_j|\alpha\mathbf{m}, \mathcal{H}_D)}{P(\mathbf{F}_j|\alpha\mathbf{m}, \mathcal{H}_D)} \tag{5.7}$$

$$= \frac{\prod_i q_{i|j}^{F_{i|j}} \prod_i q_{i|j}^{\alpha m_i - 1} \delta(\sum_i q_{i|j} - 1)/Z(\alpha\mathbf{m})}{P(\mathbf{F}_{|j}|\alpha\mathbf{m})} \tag{5.8}$$

$$= \frac{\prod_i q_{i|j}^{F_{i|j} + \alpha m_i - 1} \delta(\sum_i q_{i|j} - 1)}{P(\mathbf{F}_{|j}|\alpha\mathbf{m}) Z(\alpha\mathbf{m})} \tag{5.9}$$

$$= \text{Dirichlet}^{(\mathcal{A})}(\mathbf{q}_{|j}|\mathbf{F}_{|j} + \alpha\mathbf{m}). \tag{5.10}$$

The predictive distribution given the data $\mathbf{F}_{|j}$ is then:

$$P(i|j, D, \alpha\mathbf{m}, \mathcal{H}_D) = \int \text{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\mathbf{F}_{|j} + \alpha\mathbf{m})\, \mathbf{p}\; d^{\mathcal{A}}\mathbf{p} = \frac{F_{i|j} + \alpha m_i}{\sum_{i'} F_{i'|j} + \alpha m_{i'}} \tag{5.11}$$

Inferring the hyperparameters

We infer the hyperparameters given the data. The posterior for $\alpha\mathbf{m}$ is, by Bayes' theorem.

$$P(\alpha\mathbf{m}|D, \mathcal{H}_D) = \frac{P(D|\alpha\mathbf{m}, \mathcal{H}_D)P(\alpha\mathbf{m}|\mathcal{H}_D)}{P(D|\mathcal{H}_D)}. \tag{5.12}$$

Instead of marginalizing over the hyperparameters, we optimize them. Assuming a non-informative prior, the posterior probability maximum is found by maximizing the evidence $P(D|\alpha\mathbf{m}, \mathcal{H}_D)$. The evidence is obtained from equations 5.6 and 5.10.

$$P(D|\alpha\mathbf{m}) = \prod_j P(\mathbf{F}_{|j}|\alpha\mathbf{m}) \tag{5.13}$$

$$= \prod_j \left[\frac{Z(\mathbf{F}_{|j} + \alpha\mathbf{m})}{Z(\alpha\mathbf{m})}\right] \tag{5.14}$$

$$= \prod_j \left(\frac{\prod_i \Gamma(F_{i|j} + \alpha m_i)}{\Gamma(F_j + \alpha)} \frac{\Gamma(\alpha)}{\prod_i \Gamma(\alpha m_i)}\right) \tag{5.15}$$

By making some approximations it is possible to derive an explicit optimization algorithm [62].

5.2 Context Redundancy

The Dirichlet language model assumes that the counts for each context are generated from a unique probability vector, drawn from the Dirichlet distribution. An alternative hypothesis is that there are clusters of identical contexts, each described by the same probability vector. By combining the counts of such contexts, the evidence is increased.

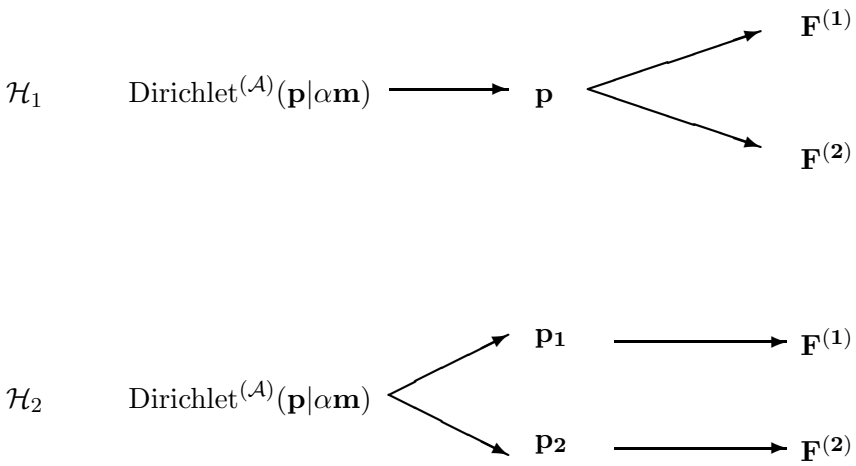


Figure 5.1: Two contexts generate counts $\mathbf{F^{(1)}}$ and $\mathbf{F^{(2)}}$. We show two hypotheses for generating these counts from a Dirichlet distribution.

For example, we may have observed the bigrams 'Tuesday morning', 'Tuesday afternoon' and 'Thursday morning', but not 'Thursday afternoon'. In the context 'Thursday' we only predict the word 'afternoon' with the frequency at which it occurs in the training text. If we were to infer that the contexts 'Tuesday' and 'Thursday' are identical, then combining them would mean that 'afternoon' is more probable in the context 'Thursday'.

5.2.1 Two hypotheses

Let $\mathbf{F^{(1)}}$ and $\mathbf{F^{(2)}}$ be conditional counts for any two contexts. The first hypothesis, $\mathcal{H}_1$, is that a single probability vector $\mathbf{p}$ was drawn from the Dirichlet distribution. The vector $\mathbf{p}$ then generated counts $\mathbf{F^{(1)}}$ and $\mathbf{F^{(2)}}$. The second hypothesis, $\mathcal{H}_2$, is that two different probability vectors $\mathbf{p}_1$ and $\mathbf{p}_2$ were drawn from the Dirichlet distribution. Each probability vector generates counts $\mathbf{F^{(1)}}$ and $\mathbf{F^{(2)}}$ respectively.

We define $\mathbf{F} = \mathbf{F^{(1)}} + \mathbf{F^{(2)}}$. We use Bayes' theorem to calculate the ratio of the posterior probabilities of the two hypotheses:

$$\frac{P(H_1|\{F\})}{P(H_2|\{F\})} = \frac{P(\{F\}|H_1)P(H_1)}{P(\{F\}|H_2)P(H_2)}. \tag{5.16}$$

The evidence for H_1 is

$$\begin{aligned} P(\{F\}|H_1) &= \int d\mathbf{p}\, P(\{F\}|\mathbf{p}, H_1)P(\mathbf{p}) \\ &= \frac{1}{Z(\mathbf{u})} \int d\mathbf{p} \prod_i {p_i}^{F_i^{(1)}+F_i^{(2)}+u_i-1} \\ &= \frac{Z(\mathbf{F}+\mathbf{u})}{Z(\mathbf{u})} \end{aligned} \tag{5.17}$$

The evidence for H_2 is

$$P(\{F\}|H_2) = P(\mathbf{F^{(1)}}|H_2)P(\mathbf{F^{(2)}}|H_2) \tag{5.18}$$

Now using equation 5.17

$$P(\{F\}|H_2) = \frac{Z(\mathbf{F^{(1)}} + \mathbf{u})Z(\mathbf{F^{(2)}} + \mathbf{u})}{[Z(\mathbf{u})]^2} \tag{5.19}$$

If we let the prior probability ratio of the two hypotheses be one, then

$$\frac{P(H_1|\{F\})}{P(H_2|\{F\})} = \frac{Z(\mathbf{u} + \mathbf{F})Z(\mathbf{u})}{Z(\mathbf{F}^{(1)} + \mathbf{u})Z(\mathbf{F}^{(2)} + \mathbf{u})} \tag{5.20}$$

5.2.2 Grouping the contexts

We would like to find the most probable clustering of the contexts. From a computational point of view, considering all possible clusterings is not possible. A simple heuristic approach is to compute the ratio 5.20 for every pair of contexts. If H_1 is more likely, then we could replace the two contexts with a new one with conditional counts $\mathbf{F} = \mathbf{F^{(1)}} + \mathbf{F^{(2)}}$. If H_2 is more likely, then we do nothing. After we have compared every pair of contexts and found new contexts, we can repeat the procedure to build larger clusters. The procedure terminates as soon as we find that $\mathcal{H}_2$ is more likely when considering every pair of contexts. Note that this is a heuristic procedure, as the final clustering depends on the order in which we compare the posterior probabilities.

5.2.3 Efficient computation using a hash function

Each attempt to cluster is an $O(W^2)$ task, where W is the number of words. We use a hash function to reduce this computational cost.

Take N ($N \ll W$) vectors $\{\mathbf{h}_i\}$, of dimension W, where the components are all i.i.d normal deviates. Compute the hash function

$$\phi_i = H(\mathbf{F}.\mathbf{h}_i), \tag{5.21}$$

where H is the Heaviside Function applied to each component ($H(x) = 1$ if $x > 0, H(x) = 0$ otherwise). ϕ has dimension N, so the total number of bins is 2^N. If the counts $\{\mathbf{F}\}$ are distributed evenly in this space, then there will be $W/2^N$ contexts in each bin.

The hash function is deliberately chosen so that similar contexts are likely to hash to the same value. Rather than test every pair for our clustering condition, we only test pairs with the same hash function. If we assume that the computational cost of calculating the posterior probabilities is far greater than calculating the hash functions, then the complexity could be as low as $O(W^2/2^N)$. The true complexity will depend on the number that end up in the same bin.

To avoid problems where similar contexts fall into different bins, we periodically generate a new set of hash vectors $\{\mathbf{h}_i\}$.

5.2.4 Demonstration

To demonstrate the method of grouping contexts, we chose text from the Gutenberg project. All text was converted to lower case, we replaced all sequences of punctuation with a single punctuation word '.', and all sequences of digits were replaced by the word '@'.

We took Darwin and Modern Science by A.C. Seward (270,000 words long) as our training text, with a vocabulary of 28,400 words. To reduce the computational cost, we reduced the vocabulary to the most frequent 5000 words, ignoring bigrams containing a word outside of the vocabulary.

Test texts were also prepared by removing bigrams containing a word outside the vocabulary generated from the training text.

Bigram model A was the original Dirichlet language model. First, we found the most probable hyperparameters by maximizing equation 5.15. Our predictive distribution is given by equation 5.11.

We took model A and applied the clustering algorithm to give a pruned model, which we call model B. Each iteration consisted of hashing the conditional counts and attempting to cluster every pair in the same bin. We then re–estimated the hyperparameters. We continued until the algorithm failed to pair any further contexts. The final model contained 1969 of the original 5000 contexts. Some of the clusters are shown:

- pass passing passed
- february january february july june march aged august september december october april
- fourth twentieth third seventh
- look looked looking stared
- market ministry country ireland labour locks guns
- committed condemned failed forward done harm admired attained sustained
- shall will might should may must would could

Note that the word 'may' appears in a group of auxiliary verbs and not with the calendar months.

Model comparison

For simplicity, we ignored the zero frequency problem; any bigrams containing words outside of the vocabulary were skipped when computing predictive probability on the test texts.

The following table shows the predictive probability measured on several texts, including the training text.

	Predictive probability / bits per word		
Text	Model A	Model B	% Change
Training Text	5.90	6.13	+3.9
T. Hardy: Madding Crowd	7.98	7.92	-0.7
Sent-mail	8.20	8.14	-0.7
Tales of Tom Swift	7.70	7.64	-0.7

5.3 Conclusion

We have overviewed the Dirichlet language model, described by MacKay and Peto [62]. We described a new Bayesian pruning algorithm, which successfully reduces the number of contexts in the language model. The combination of contexts reveals words with similar contextual meaning.

The new model slightly outperforms the original on English text, while requiring fewer parameters and hence less memory.

Chapter 6

Neural Networks for Modelling Discrete Data

The multilayer perceptron is a feedforward network; it has input neurons, hidden neurons and output neurons. The hidden neurons are often arranged in a sequence of layers. Feedforward networks define a non-linear mapping from an input vector $\mathbf{x}$ to an output vector $\mathbf{y} = (\mathbf{x}; \mathbf{w})$; they can be trained to perform regression and classification tasks.

In this chapter we show how to model discrete data with a neural network. We explore the limits of traditional, non-Bayesian training. Then we show how to perform Bayesian learning by Monte Carlo methods.

6.1 Classification Networks

Supervised neural networks are given data in the form of inputs and targets, the targets being a specification by a teacher of what the neural network's response to the input should be. The purpose of training a classification network is to infer the classes of unseen data points.

We assume we have a data set $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ with labels $\{t^{(n)}\}_{n=1}^{N}$. The labels take discrete values. The mapping for a network with one hidden layer may have the form:

$$\text{Hidden layer:} \quad a_j^{(1)} = \sum_l w_{jl}^{(1)} x_l + \theta_j^{(1)}; \quad h_j = f^{(1)}(a_j^{(1)}) \tag{6.1}$$

$$\text{Output layer:} \quad a_i^{(2)} = \sum_j w_{ij}^{(2)} h_j + \theta_i^{(2)}; \quad y_i = f^{(2)}(\{a_j^{(2)}\}) \tag{6.2}$$

The value of output unit i is interpreted as the probability that the object $\mathbf{x}$ is in class i;

$$y_i(\mathbf{w}; \mathbf{x}) = P(t = i | \mathbf{x}, \mathbf{w}). \tag{6.3}$$

In classification networks, we often use the softmax function [14] at the output layer:

$$y_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}. \tag{6.4}$$

The softmax activation function ensures that the outputs are positive and that they sum to 1.

6.2 Neural Network Learning

Given examples of a relationship between an input vector $\mathbf{x}$ and a target t, we would like the neural network to learn a model of this relationship. A successfully trained network will, for any given $\mathbf{x}$, give an output y that is close (in some sense) to the target value t. Traditional training of the network involves searching in the weight space for a value of $\mathbf{w}$ that produces a function that fits the provided training data well.

6.2.1 Bayesian learning

Infer the parameters given the data

The network is parameterized by weights and biases, collectively denoted by $\mathbf{w}$. A prior for the network parameters is defined, which may depend on a number of hyperparameters, $\boldsymbol{\alpha}$. The prior for the parameters is written as $P(\mathbf{w}|\boldsymbol{\alpha})$ and the prior on the hyperparameters is written as $P(\boldsymbol{\alpha})$. We use Bayes' theorem to infer the parameters;

$$P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H}) = \frac{P(D|\mathbf{w}, \boldsymbol{\alpha}, \mathcal{H})P(\mathbf{w}, \boldsymbol{\alpha}|\mathcal{H})}{P(D|\mathcal{H})} \tag{6.5}$$

$$= \frac{P(D|\mathbf{w}, \mathcal{H})P(\mathbf{w}|\boldsymbol{\alpha}, \mathcal{H})P(\boldsymbol{\alpha}|\mathcal{H})}{P(D|\mathcal{H})}. \tag{6.6}$$

The likelihood is

$$P(D|\mathbf{w}, \mathcal{H}) = \prod_n P(t^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}, \mathcal{H}). \tag{6.7}$$

Predict the next class label for a given input

To obtain the probability of $t^{(n+1)}$ given $\mathbf{x}^{(n+1)}$ and the data D, we marginalize over the unknown parameters $\mathbf{w}$ and $\boldsymbol{\alpha}$.

$$P(t^{(n+1)}|\mathbf{x}^{(n+1)}, D, \mathcal{H}) = \int d\mathbf{w}\, d\boldsymbol{\alpha}\, P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H})P(t^{(n+1)}|\mathbf{x}^{(n+1)}, \mathbf{w}, \mathcal{H}) \tag{6.8}$$

The distribution inside the integral, $P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H})$ depends upon the likelihood function and the priors.

Methods

There are few cases in which equation 6.6 can be computed analytically. The traditional method of training a neural network is find the most probable parameters and use these for prediction. We shall discuss this method in section 6.2.2.

The two main approaches to implementing Bayesian inference for neural networks are the Monte Carlo methods developed by Neal [74] and Gaussian approximation methods developed by MacKay [59]. We investigate Monte Carlo methods in section 6.6.

6.2.2 Non-Bayesian learning

Training by MAP (maximum a-posteriori probability)

If we know the values of the hyperparameters, $\boldsymbol{\alpha}$, then the posterior probability of the parameters $\mathbf{w}$ is

$$P(\mathbf{w}|D, \alpha, \mathcal{H}) = \frac{P(D|\mathbf{w}, \mathcal{H})P(\mathbf{w}|\boldsymbol{\alpha}, \mathcal{H})}{P(D|\mathcal{H})}. \tag{6.9}$$

Rather than learning an ensemble of plausible parameters, we choose the value of $\mathbf{w}$ that maximizes the posterior density. Equivalently, we minimize the negative log of the posterior:

$$M(\mathbf{w}) \quad = \quad -\log[P(D|\mathbf{w}, \mathcal{H})P(\mathbf{w}|\boldsymbol{\alpha})]. \tag{6.10}$$

For simplicity, we assume a Gaussian prior on the parameters parameterized by a single hyperparameter, α

$$P(\mathbf{w}|\boldsymbol{\alpha}) \propto \prod_i \exp\left[-\frac{\alpha w_i^2}{2}\right] \tag{6.11}$$

Our objective function is therefore

$$M(\mathbf{w}) = -\sum_n \log y_{t^{(n)}}(\mathbf{x}^{(n)}; \mathbf{w}) + \frac{\alpha}{2}\sum_i w_i^2 + \text{constants} \tag{6.12}$$

The value of $\mathbf{w}$ that minimizes $M(\mathbf{w})$ is the most probable value, denoted by $\mathbf{w}_{\text{MP}}$. It is common practice to use this optimum for prediction

$$P(t^{(n+1)}|\mathbf{x}^{(n+1)}, D, \mathcal{H}) = P(t^{(n+1)}|\mathbf{x}^{(n+1)}, \mathbf{w}_{\text{MP}}, \mathcal{H}). \tag{6.13}$$

Cross-validation

In general, we do not know the values of the hyperparameters. The usual method of determining the hyperparameters in a non-Bayesian framework is by cross-validation. We set aside a part of the training test as a validation set. The hyperparameters are optimized to give the best performance on the validation set. Once we have found these 'optimum' parameters, the network is trained on the whole of the data set.

As the amount of data increases, the posterior ensemble becomes increasingly concentrated

around the most probable value $\mathbf{w}_{\mathrm{MP}}$. In the case of a large amount of data, the MAP predictions will be good.

6.3 Automatic Relevance Determination

In a finite data set, there will be random correlations between any irrelevant inputs and the targets. In this case, the neural network does not set the weights connecting to these inputs to zero. Thus these inputs hurt the model's performance, particularly in the case of a large number of inputs and little data.

Rather than having to choose the correct number of inputs, we would like a model in which we can include all the inputs which we think might be important. We then automatically infer which inputs are relevant. A simple way to do this is to introduce multiple regularization constants, one associated with each class of weights. For example, a weight class might consist of all the weights connecting one or several inputs to the hidden units.

In the automatic relevance determination model, the parameters are divided into classes c, with independent hyperparameters, α_c. Assuming a Gaussian prior for each class, we define

$$E_{W(c)} = \sum_{i \in c} w_i^2/2 \tag{6.14}$$

so that the prior on the parameters can be written as

$$P(\{w_i\}|\{\alpha_c\}, \mathcal{H}_{ARD}) = \frac{1}{\prod_c Z_{W(c)}} \exp\left(-\sum_c \alpha_c E_{W(c)}\right), \tag{6.15}$$

where $Z_{W(c)}$ is the normalization constant for class c. We discuss the inference of the hyperparameters $\{\alpha_c\}$ in sections 6.5.3 and 6.7.1.

6.4 Modelling Discrete Data

We want to model the data $D = \{t^{(n)}\}_{n=1}^{L}$, a sequence of L symbols from an alphabet of size A. The probability of the data can be written in terms of conditional probabilities

$$P(D) = \prod_{n=1}^{L} P(t^{(n)}|\{t^{(i)}\}_{i=1}^{n-1}). \tag{6.16}$$

We use a neural network to model the probability of the next symbol given the previous ones. It is convenient to map the 'context' $\{t^{(i)}\}_{i=1}^{n-1}$ onto a binary vector $\mathbf{x}$.

$$\mathbf{x}^{(n)} = f(\{t^{(i)}\}_{i=1}^{n-1}) \tag{6.17}$$

For example, in a bigram model, $\mathbf{x}$ could be a unary encoding of the previous character.

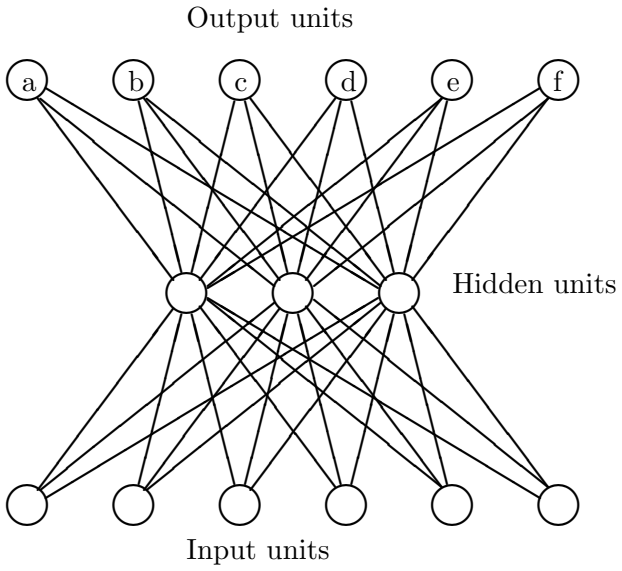


Figure 6.1: Neural Network architecture for a bigram neural network.

A feed-forward network defines a mapping from input $\mathbf{x}$ to an output $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w})$, such that

$$P(t^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}) = y_{t^{(n)}}(\mathbf{x}^{(n)}; \mathbf{w}) \tag{6.18}$$

The parameters of the network, $\mathbf{w}$, are specified by two matrices, $\mathbf{C}$ and $\mathbf{G}$. The matrix of weights, $\mathbf{C}$, maps input vectors $\mathbf{x}$ to hidden units $\mathbf{h}$

$$h_j = \tanh\left[\sum_{i=1}^{M} x_i C_{ij} + C_{0j}\right] \tag{6.19}$$

The matrix $\mathbf{C}$ includes biases on each hidden unit. A matrix $\mathbf{G}$, maps the hidden units to outputs $\mathbf{y}$ through a softmax activation function.

$$a_i = \sum_{j=1}^{H} h_j G_{ji} + G_{0i} \tag{6.20}$$

$$y_i = \frac{e^{a_i}}{\sum_{j=1}^{A} e^{a_j}} \tag{6.21}$$

The matrix $\mathbf{G}$ includes biases on each output unit. The motivation behind using such a network is parameter sharing for similar contexts. The rows of $\mathbf{G}$ are prototype contexts and the entries of $\mathbf{C}$ map the actual context onto these prototype contexts.

Under this assumption, $\mathcal{H}_N$, we can write the likelihood as

$$P(D|\mathbf{w}, \mathcal{H}_N) = \prod_{n=1}^{L} P(t^{(n)}|\mathbf{x}^{(n)}, \mathbf{w}, \mathcal{H}_N) \quad (6.22)$$

$$= \prod_{n=1}^{L} y_{t^{(n)}}(\mathbf{x}^{(n)}; \mathbf{w}), \quad (6.23)$$

where $\mathbf{w} = \{\mathbf{C}, \mathbf{G}\}$.

6.5 Implementing MAP Models

Before considering a Bayesian model, we explore the limits of training by MAP.

6.5.1 A bigram model of English

We start with a simple bigram character model for English. The alphabet is of size $A = 27$ (a-z and space). The neural network input $\mathbf{x}^{(n)}$ is a unary code of the previous character $t^{(n-1)}$:

$$x_i^{(n)} = \delta_{t^{(n-1)}i} \text{ for } n > 1, \quad (6.24)$$

$$x_i^{(n)} = 0 \text{ for } n = 1. \quad (6.25)$$

The text of *Emma* by Jane Austen was used to train the model. The text was first converted to an alphabet of size 27. Upper case was converted to lower case and non-alphabetical characters were replaced by the space character. Finally, all repeated spaces were removed. The resulting text was divided into two pieces, 500kB for training and 300kB for testing the performance of the model.

Qualitative results

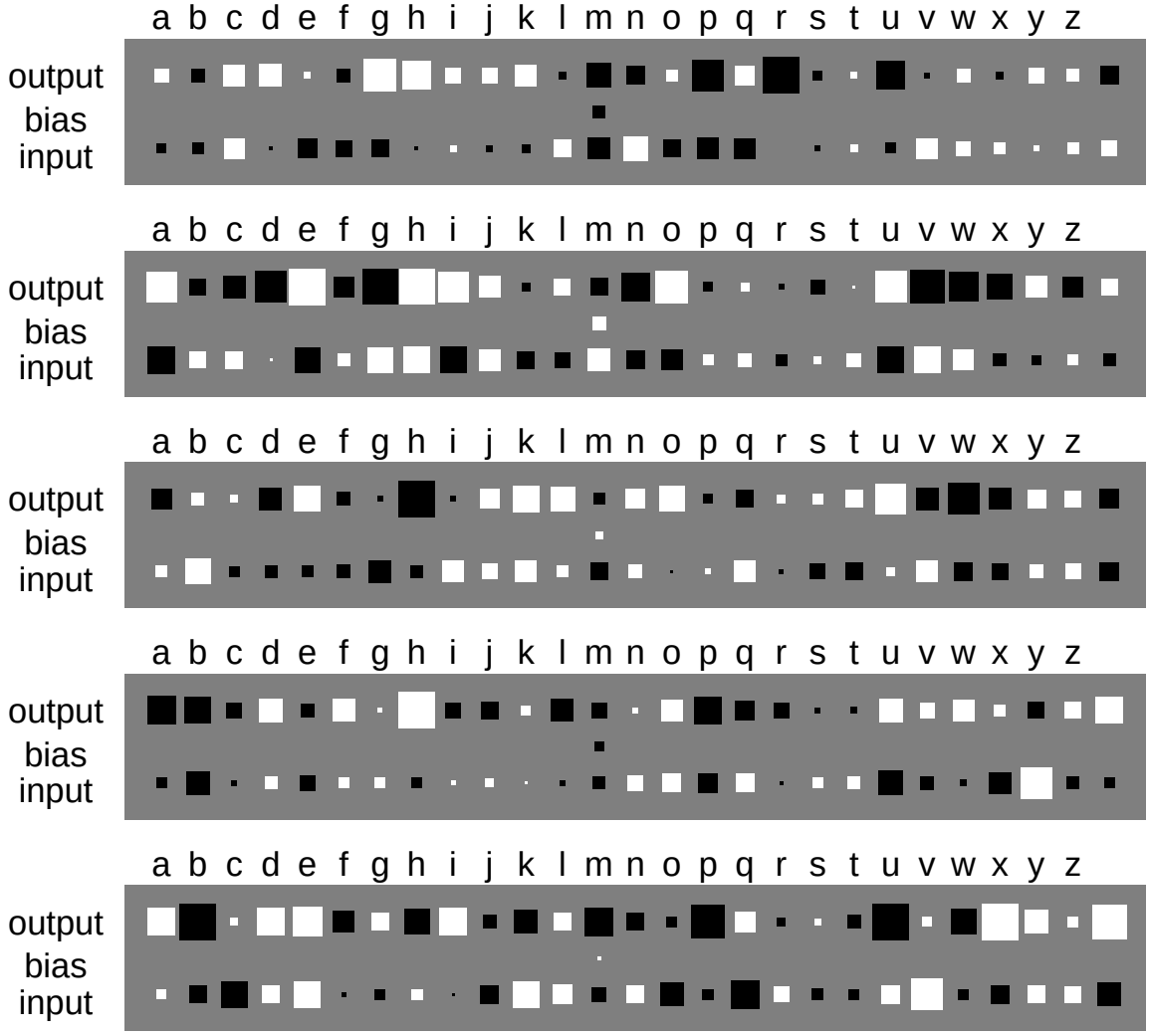
In figure 6.2 we show the neural network weights when a neural network with $H = 5$ hidden units has been trained on 500kB of data. A single hyperparameter, $\alpha = 5$, was set by hand. Figure 6.3(a) shows the predictions of the neural network. For comparison we also compute the maximum likelihood distribution:

$$P(x_i|x_{i-1}) = \frac{F_{i|i-1}}{F_j} \quad (6.26)$$

where $F_{i|j}$ is the number of times that letter i follows j, and F_j is the number of times that the character j appears. The maximum likelihood distribution is shown in figure 6.3(b) .

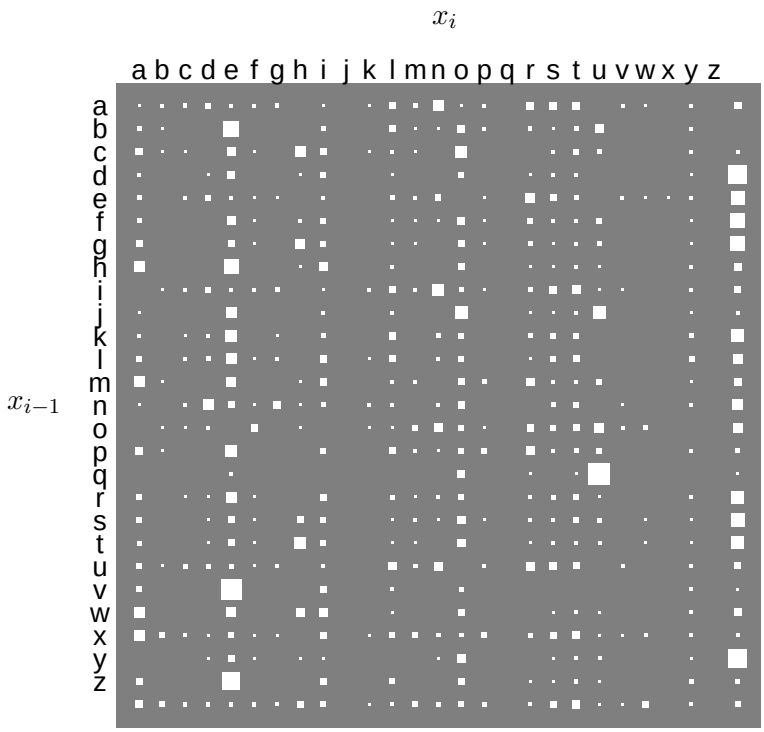
In figure 6.4, we show the difference between two two distributions. The maximum likelihood model is specified by $27 \times 27 = 729$ parameters, but the neural network is capable of reproducing a similar distribution with only $27 \times 6 + 5 \times 28 = 302$ parameters.

(a)

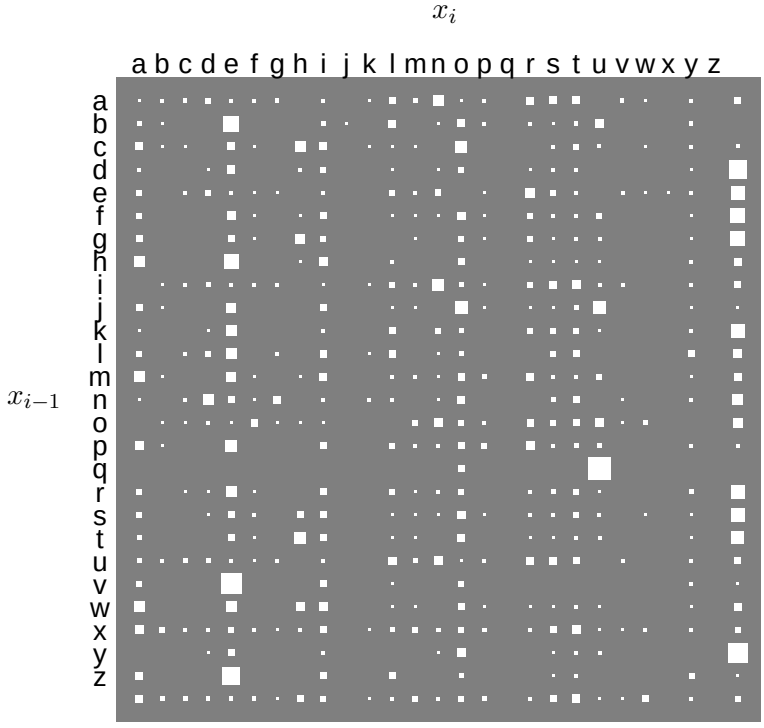


(b)

Figure 6.2: Trained bigram neural network with $H = 5$ hidden units. There is a single hyperparameter, $\alpha = 5$. Positive values are white; negative values are black, with area proportional to the magnitude of the weight. Figure (a) shows the biases to the output units. In figure (b), there is a Hinton diagram for each hidden unit. The top row shows the weights to output; the bottom row shows the weights connecting to the input units. The value in middle row is the bias on the hidden unit.



(a)



(b)

Figure 6.3: Bigram conditional probability distributions, $P(x_i|x_{i-1})$ of Emma training text. (a) Reconstructed distribution from neural network with $H = 5$. (b) Maximum likelihood bigram distribution generated from frequency counts. The area of each box is proportional to $P(x_i|x_{i-1})$.

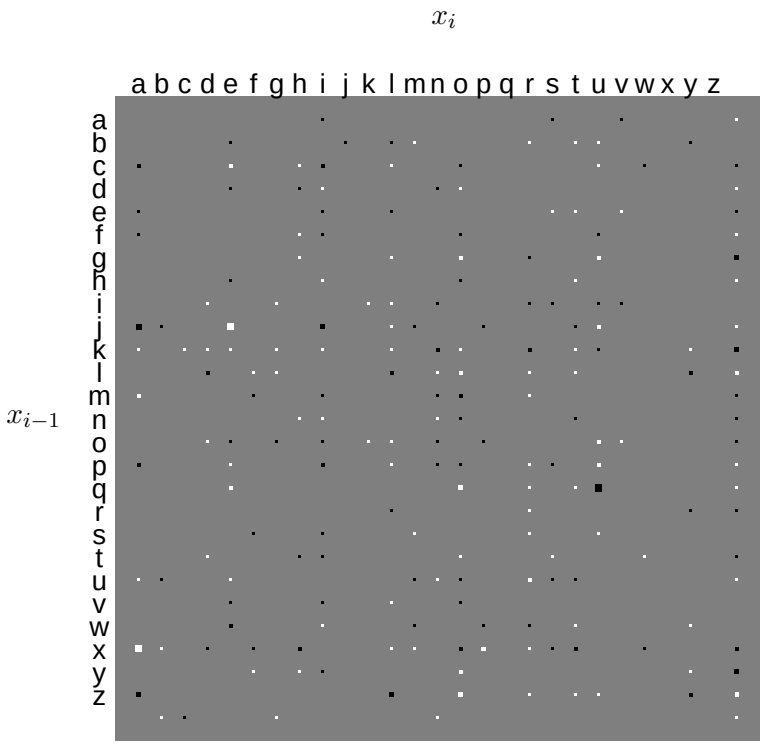


Figure 6.4: Differences between the bigram conditional probability distributions, shown in figures 6.3(a) and (b). Differences are shown on the same scale as figure 6.3. The area of each box is proportional to $P(x_i|x_{i-1})$. Positive values are shown in white and negative values are black. The largest difference is 0.09.

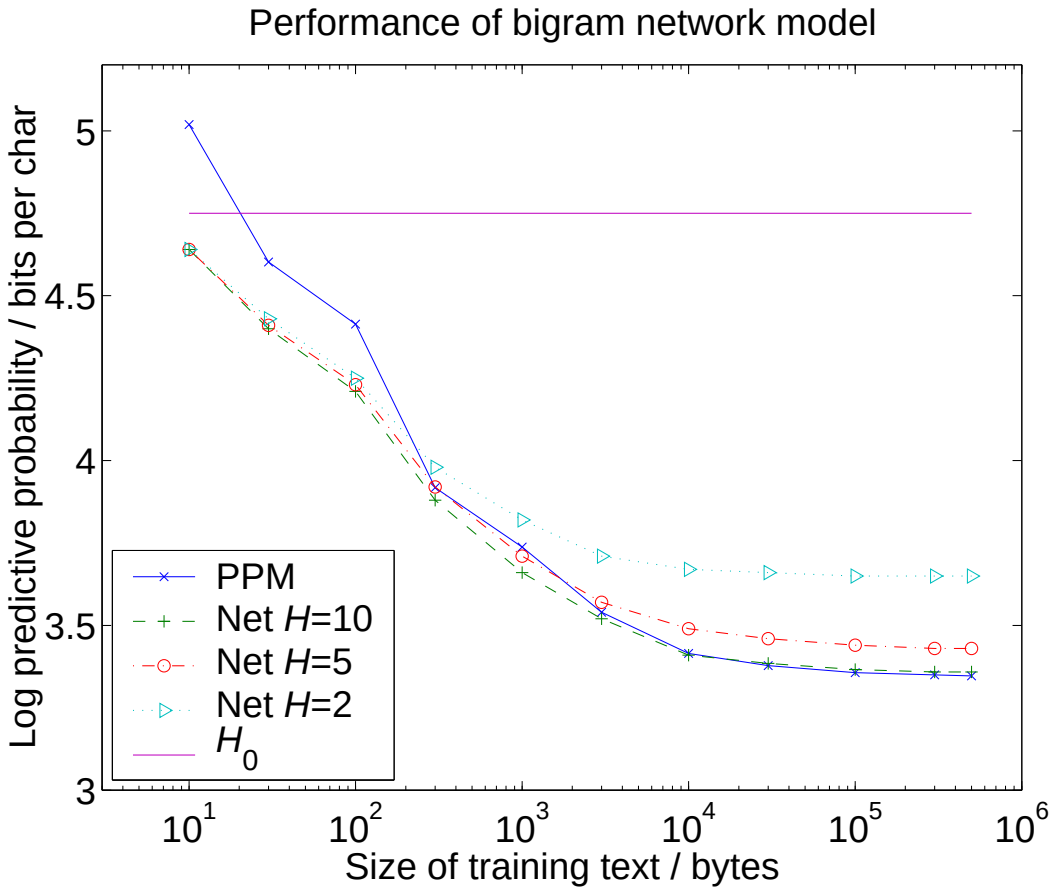


Figure 6.5: Predictive performance of bigram neural network with $H = 2, 5, 10$ hidden units. Results are also shown for PPMD5, and the uniform model H_0.

Predictive results

Several bigram neural networks with $H = 2, 5, 10$ and 27 inputs were trained on different amounts of data, ranging from 10B to 500kB. We plot the predictive probability of the test set in figure 6.5. For comparison, we show results for PPMD5, a model used in text compression and reviewed in section 1.5. We also introduce the uniform model, H_0, which assigns equal probability to each output symbol.

The results show that all neural networks outperform PPMD5 with small amounts of data. However, as the amount of data increases, the neural networks with larger numbers of hidden units perform better. The asymptotic performance of the best neural network is almost as good as PPMD5, so it would appear that 10 hidden units (577 parameters) are sufficient to build a good bigram model.

We also observe that neural networks always perform better than the uniform model H_0. In the case of a small amount of data, PPMD5 performs worse that the uniform model.

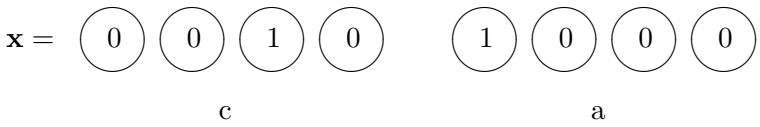


Figure 6.6: We use a unary code to map input symbols to the inputs of the neural network. In the example above, we show the mapping with alphabet size $A = 4$ and order $O = 2$.

6.5.2 Higher order models

To build a higher order model, we need to find a suitable mapping, f, from the history to the neural network inputs:

$$\mathbf{x}^{(n)} = f(\{t^{(i)}\}_{i=1}^{n-1}) \tag{6.27}$$

We could concatenate unary encodings of the previous O characters, resulting in a network with OA inputs. Figure 6.6 demonstrates this mapping. Higher order correlations would be captured through the non-linearity of the hidden units.

Results

Figure 6.7 shows the results of a neural network with order 2. Again, the neural network outperforms PPMD5 in the case of small and moderate amounts of data. It appears that about 50 hidden neurons are sufficient to capture most of the trigram statistics.

Beyond order 2 models

The results for neural networks of order 1 and 2 are promising. However, figure 6.8 shows that increasing the order further does not always work. If $\alpha = 5$, the neural networks with order greater than two perform worse than the order 2 model. Increasing the hyperparameter, α, reduces overfitting, but higher order models never outperform the order 2 model with $\alpha = 5$.

In the case of 1kB of training text, we expect many of the weights connecting the higher order input units to be irrelevant, while the lower order inputs are relevant. We therefore consider an automatic relevance determination model as introduced in section 6.3.

6.5.3 Approximate optimization of hyperparameters

We use a cheap and cheerful method, motivated by Gaussian approximations [60]. The following rule is used to update the hyperparameters:

$$\alpha_c = f\frac{k_c}{\sum_i w_i^2}, \tag{6.28}$$

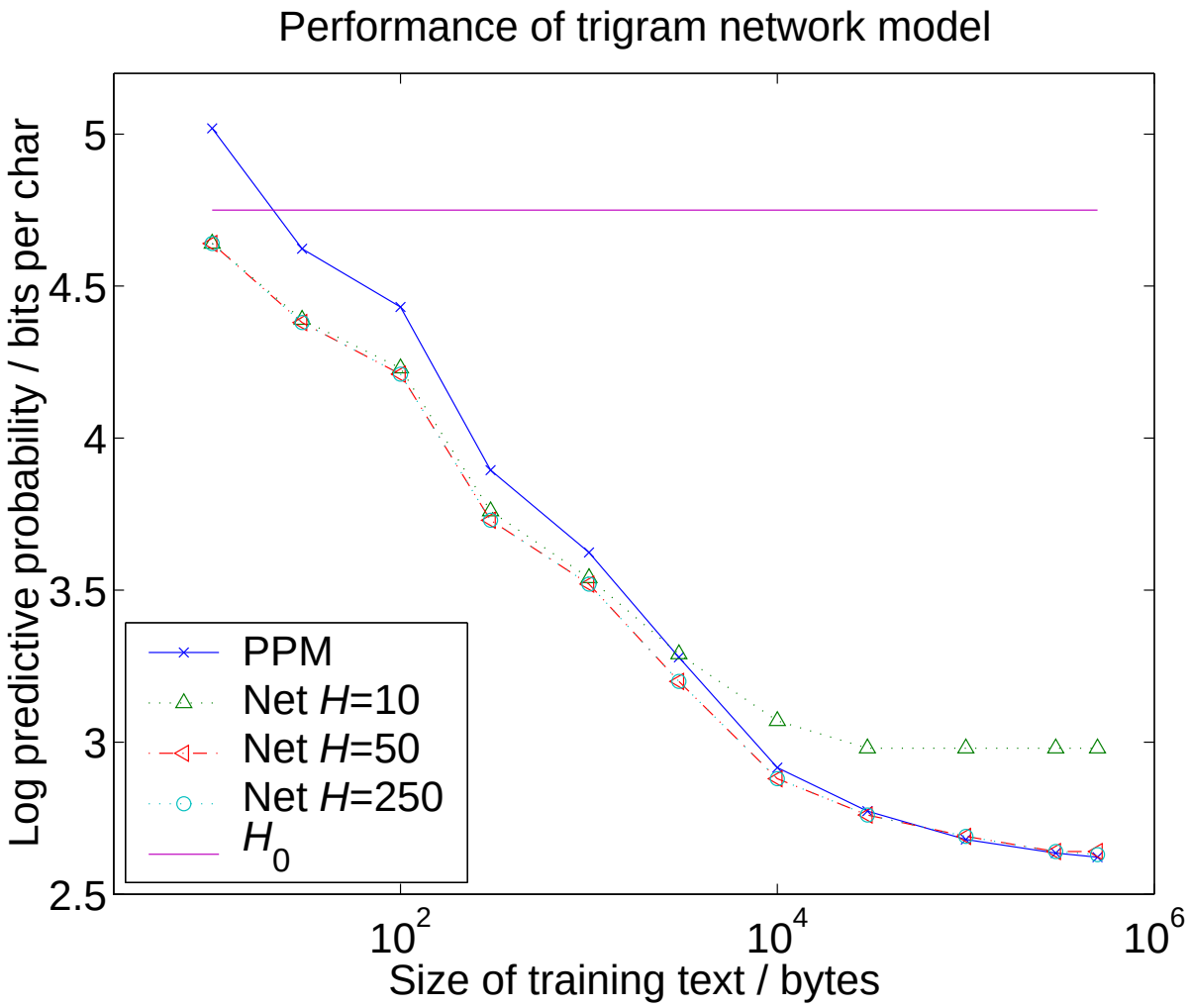
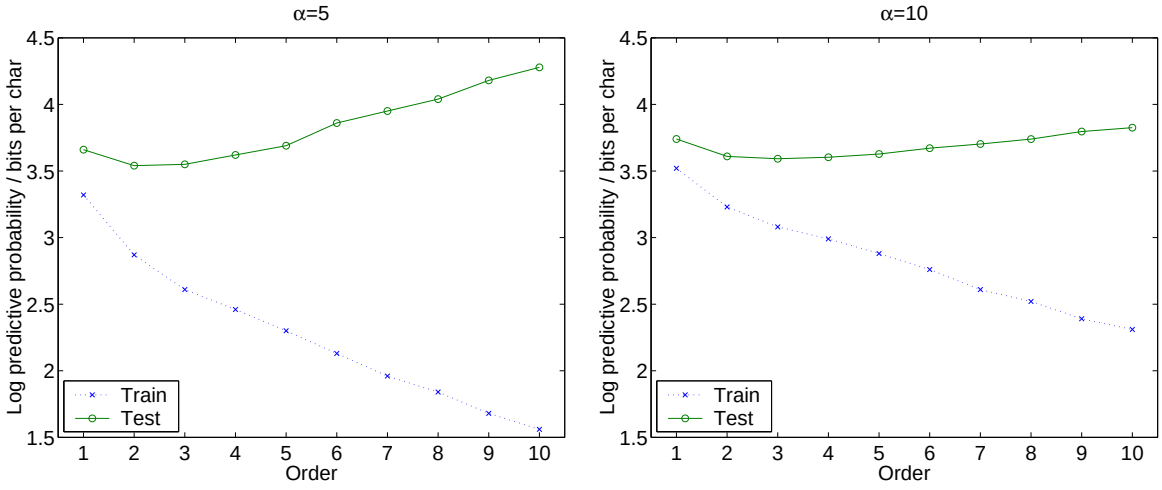


Figure 6.7: Predictive performance of trigram neural network with $H = 10, 50, 250$. Results are also shown for PPMD5, and the uniform model H_0.



(a) (b) (c) (d)

Figure 6.8: Performance of higher order models with 1kB of training text and $H = 10$ hidden units. All models of order greater than 2 perform worse than the order 2 model with $\alpha = 5$.

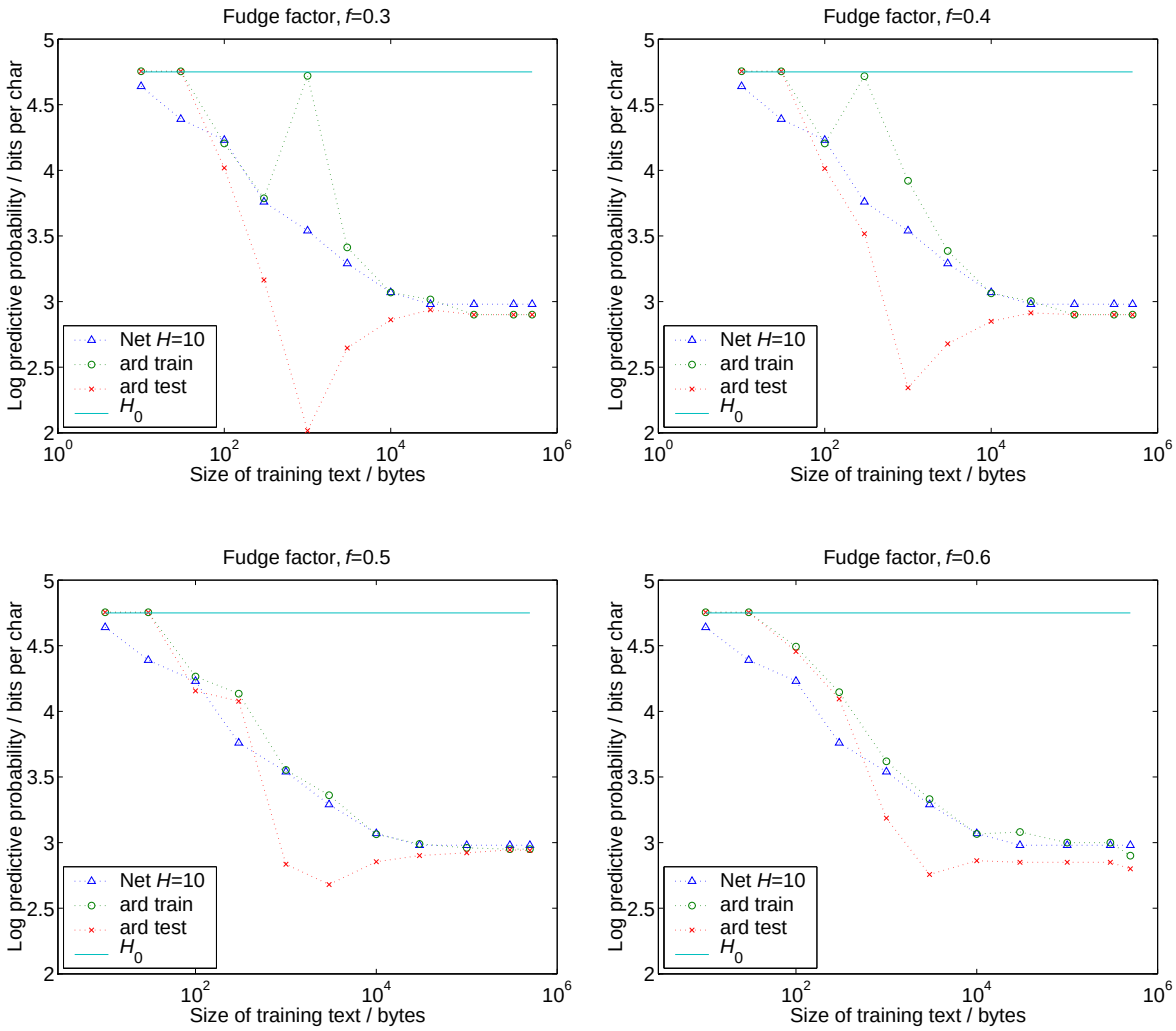


Figure 6.9: Performance of automatic relevance determination model with order, $O = 2$, compared to identical network with $\alpha = 0.5$. All models have 10 hidden units.

where k_c is the number of parameters in class c and f is a 'fudge factor' to imitate the effect of integrating over $\mathbf{w}$.

6.5.4 Applying automatic relevance determination to higher order models

To assess the performance of the approximate implementation of automatic relevance determination, we aim to reproduce the results of the order 2 network. Several neural networks with $H = 10$, $O = 2$ and $\alpha = 0.5$ were initialized. Before applying automatic relevance determination, each network was optimized by a conjugate gradient minimizer. Next, we alternated between applying update rule (6.28) and 20 iterations of the minimizer. This procedure was terminated if the optimizer's stopping criterion was met on the 1st of the 20 iterations.

Figure 6.9 shows results with several different settings of the 'fudge factor', f. If the size of the training set is 1kB, then setting f to a value of 0.3 or 0.4 gives a model which

badly overfits the data. Setting f to a larger value of 0.5 or 0.6 reduces overfitting, but neither models perform better than fixing $\alpha = 0.5$. It would appear that this approximate implementation of automatic relevance determination is not very good.

6.6 Monte Carlo Inference

The objective of Bayesian learning is to produce predictions for test cases as in equation 6.8. This requires that we evaluate the expectation of a function with respect to the posterior distribution for model parameters. Writing this posterior for the parameters as $P(\theta|D)$, the expectation of a function $R(\theta)$ is

$$E(R) = \int R(\theta)P(\theta|D)d\theta. \tag{6.29}$$

Expectations can be estimated by the Monte Carlo method, using sample values from R:

$$E(R) \approx \sum_{i=1}^{N} \frac{1}{N} R(\theta^{(i)}) \tag{6.30}$$

where $\theta^{(i)}$ are samples from the distribution $P(\theta|D)$. Generating independent samples might be difficult, but equation 6.30 still gives an unbiased estimate of $E(R)$ even when $\theta^{(i)}$ are dependent. As N increases, our estimate will converge to the true value.

If we can sample from the posterior distribution, then it is easy to make predictions. So, how do we generate samples from the posterior?

6.7 Hybrid Monte Carlo

The hybrid Monte Carlo method [73, 75] is a sampling method that makes use of gradient information to reduce random walk behaviour. The target probability distribution can often be written as

$$P(\mathbf{x}) = \frac{\exp\left[-E(\mathbf{x})\right]}{Z}. \tag{6.31}$$

We introduce a 'momentum' variable $\mathbf{p}$ which has the same dimensionality of $\mathbf{x}$. The canonical distribution over the 'phase space' of $\mathbf{x}$ and $\mathbf{p}$ is defined to be

$$P_H(\mathbf{x}, \mathbf{p}) \propto \exp\left[-H(\mathbf{x}, \mathbf{p})\right], \tag{6.32}$$

where H is the Hamiltonian

$$H(\mathbf{x}, \mathbf{p}) = E(\mathbf{x}) + K(\mathbf{p}), \tag{6.33}$$

and $K(\mathbf{p}) = \mathbf{p}^{\mathrm{T}}\mathbf{p}/2$ is a kinetic energy. In distribution 6.32, $\mathbf{p}$ and $\mathbf{x}$ are independent so the marginal distribution of $\mathbf{x}$ is the same as in equation 6.31, from which we wish to sample. We can therefore define a Markov chain that converges to P_H and discard the values of $\mathbf{p}$.

The first proposal draws a new momentum from the Gaussian density $P_K = \exp[-K(\mathbf{p})]/Z_K$. In the second dynamical proposal, we simulate the dynamics in accordance with the equations

$$\dot{\mathbf{x}} = \mathbf{p} \tag{6.34}$$

$$\dot{\mathbf{p}} = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}. \tag{6.35}$$

If the simulation is perfect, then the proposals are accepted every time, because the energy H is a constant of the motion. We shall use finite step sizes to simulate the dynamics, so some of the proposals will be rejected (see section 6.7.1).

6.7.1 Procedure

Hybrid Monte Carlo consists of two alternating steps.

Dynamical transition

Given values of a step size, ϵ and the number of leapfrog steps, L, a dynamical transition is performed as follows.

- Starting from the current state $(q, p) = (q(0), p(0))$, perform L leapfrog steps with step size ϵ, resulting in state $(q(\epsilon L), p(\epsilon L))$

- Regard $(q^\star, p^\star) = (q(\epsilon L), p(\epsilon L))$ as a candidate for the next state, accepting it with probability

$$\min\left[1, \exp\left(-\left(H\left(q^\star, p^\star\right) - H\left(q, p\right)\right)\right)\right] \tag{6.36}$$

  and otherwise leaving the state unchanged.

Gibbs sampling for hyperparameters

The posterior distribution of the parameters in class c, given the hyperparameter for class c is

$$P(w_{c1}, w_{c2}, \ldots, w_{ck}|\alpha_c) = (2\pi)^{-k/2} {\alpha_c}^{k/2} \exp\left[-\alpha_c \sum_i w_{ci}^2/2\right] \tag{6.37}$$

We put a Gamma prior (see appendix B) on the hyperparameters, with a mean μ_c, and a shape parameter specified by a_c, with density

$$P(\alpha_c) = \frac{1}{\Gamma(a_c/2)} \left[\frac{a_c}{2\mu_c}\right]^{a_c/2} {\alpha_c}^{a_c/2-1} \exp\left(-\alpha_c a_c/2\mu_c\right). \tag{6.38}$$

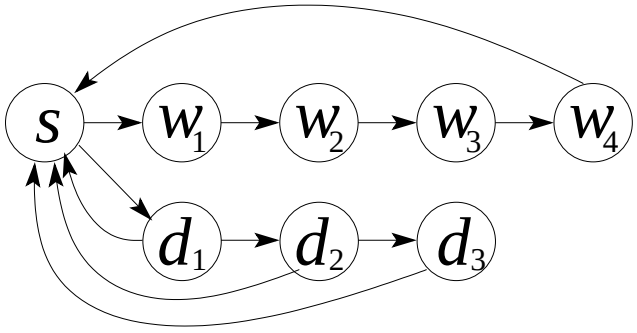


Figure 6.10: State diagram of a toy model. State s emits a space. States w_i emit symbols from {abcde}. States d_i emit symbols from {01234}.

The conditional distributions for α_c given the parameters $\{w_{ci}\}$ are determined by Bayes' theorem:

$$P(\alpha_c|w_{c1}, w_{c2}, \ldots, w_{ck}) = \frac{P(w_{c1}, w_{c2}, \ldots, w_{ck}|\alpha_c)P(\alpha_c)}{P(w_{c1}, w_{c2}, \ldots, w_{ck})} \tag{6.39}$$

$$\propto \alpha^{(a_c+k)/2-1} \exp\left[-\frac{\alpha_c}{2}\left(a_c/\mu_c + \sum w_{ci}^2\right)\right] \tag{6.40}$$

The conditional distribution 6.40 is used for Gibbs sampling, since given $w_{c1}, w_{c2}, \ldots, w_{ck}$, the value of α_c is independent of the other parameters, hyperparameters, and the data. Efficient methods of generating Gamma-distributed random variates are known [30].

6.8 Implementation of Monte Carlo Inference

6.8.1 A toy model

The hidden Markov model shown in figure 6.10 defines a toy model for a language with 11 symbols in the alphabet — $\{\text{space}, a, b, c, d, e, 1, 2, 3, 4, 5\}$. Each of the 8 states outputs a symbol according to the matrix Q. Each row of Q represents a state; the entries give the probability of emitting each symbol. Therefore, state s emits a space, states w_i emit letters {abcde}, and states d_i emit numbers {12345}.

$$Q = \begin{array}{c} \begin{array}{ccccccccccc} \text{space} & \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3/9 & 3/9 & 1/9 & 1/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 3/9 & 3/9 & 1/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 1/9 & 3/9 & 3/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 1/9 & 1/9 & 3/9 & 3/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{pmatrix} \end{array} \begin{array}{c} s \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ d_1 \\ d_2 \\ d_3 \end{array} \tag{6.41}$$

The state transition probabilities are given by the matrix P.

$$P = \begin{array}{c} \begin{array}{cccccccc} s & w_1 & w_2 & w_3 & w_4 & d_1 & d_2 & d_3 \end{array} \\ \begin{pmatrix} 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} s \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ d_1 \\ d_2 \\ d_3 \end{array} \tag{6.42}$$

Here are some data generated from this model.

```
14 3 bbde aacd 1 0 edcb 2 1 102 bace 2 3 322 140 4 bceb 21 20 341 21 33

acdd 243 1 bcdc dbdd 003 1 bacb 310 4 43 020 312 0 4 3 0 bbab aaee adcd
```

Given a sample from the model, let the state sequence be S and the data be X. The *mutual information* between S and X can be written in terms of conditional entropies.

$$H(X;S) = H(S) - H(S|X) \tag{6.43}$$

$$H(X;S) = H(X) - H(X|S) \tag{6.44}$$

In this toy model, the state sequence can always be deduced from the data (unless it is very short). In other words, $H(S|X) = 0$. Therefore,

$$H(X) = H(X|S) + H(S) \tag{6.45}$$

The two terms on the right hand side of equation 6.45 can be determined from matrices P and Q (see appendix C). In our toy model, the entropy is 2.07 bits per character.

6.8.2 Procedure

We modelled the toy data with a neural network of order 4. As in previous models, the inputs were unary encodings of previous characters.

We divided the weights into 7 classes. A separate class was given to each order of input. There was a class for the biases on the hidden units, a class for the weights connecting hidden units to the outputs, and a class for the biases on the output units.

Initially, we set all the hyperparameters, α, to 1, and the weights were drawn from their prior distributions.

We put broad Gamma priors on the hyperparameters, with mean $\mu = 1$ and shape parameter $a = 0.001$.

We set the number of leapfrog steps, L, by trial and error. A simple heuristic was applied to guess a suitable value of ϵ. During the 'burn-in' period, ϵ was modified to maintain an acceptance rate of around 30–90% . After equilibrium had been reached, we fixed the step-size and began to sample from the posterior distribution of parameters. About $N = 12$ samples proved to be adequate.

To obtain the predictive probability of $t^{(n+1)}$ given $\mathbf{x}^{(n+1)}$ and the data D, we use samples from the posterior distribution:

$$P(t^{(n+1)}|\mathbf{x}^{(n+1)}, D, \mathcal{H}) = \int d\mathbf{w}\, d\boldsymbol{\alpha}\, P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H}) P(t^{(n+1)}|\mathbf{x}^{(n+1)}, \mathbf{w}, \mathcal{H}) \quad (6.46)$$

$$\approx \frac{1}{N} \sum_{i=1}^{N} P(t^{(n+1)}|\mathbf{x}^{(n+1)}, \mathbf{w}^{(i)}, \mathcal{H}) \quad (6.47)$$

6.8.3 Results

We ran the procedure on different lengths of training text, ranging from 10 bytes to 300kB. The neural networks took about a day to train on a 1GHz desktop computer. In figure 6.11 we show the predictive probability of the models against 500kB of text text, computed from equation 6.47. The results show that the neural network easily outperforms PPM, even with 300kB of text.

6.9 Independent Data of Fixed Length

A neural network is also a suitable model for independent samples of fixed length categorical data, as in Bengio and Bengio [8]. For example, figure 6.12 shows descriptions of mushrooms taken from the UCI machine learning repository [9], which is available from `http://www1.ics.uci.edu/~mlearn/MLRepository.html`.

In [8], training was performed by MAP, and the hyperparameters were optimized by cross-validation. Initially, we hoped to to train a Bayesian neural network on the same data and compare results. However, we found large differences between the training and test sets. We

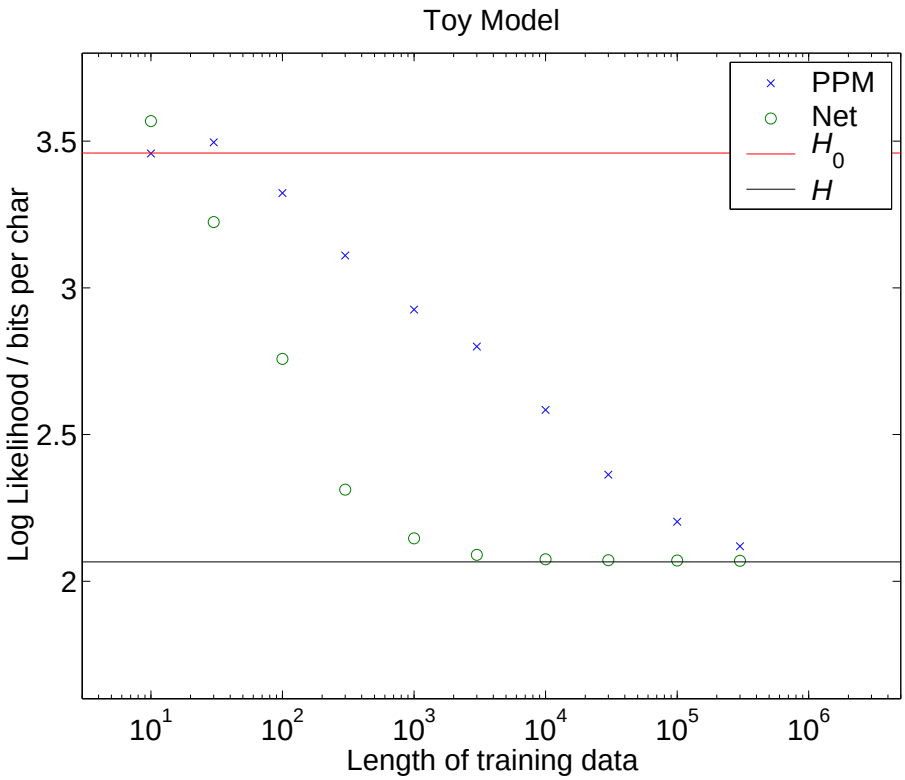


Figure 6.11: Performance of neural network on toy data. Comparison with PPMD5 shown.

S_1	S_2	S_3	S_4	S_5	S_6	S_7	
convex	smooth	white	bruises	almond	free	crowded	...
convex	scaly	yellow	bruises	almond	free	crowded	...
flat	scaly	white	bruises	pungent	free	close	...

Figure 6.12: Mushroom data, taken from the UCI dataset. 7 of the 23 features are shown in the table.

therefore decided to randomize the order of the data before constructing training and test sets.

6.9.1 The model

Each sample $S = S_1S_2S_3, \ldots, S_F$ consists of F categorical features. The variable S_i takes values from a finite alphabet $\{a_{ij}\}_{j=1}^{A_i}$. A_i is the number of possible values that the variable S_i can take. The values may differ across features, as in the mushroom example above. However, in the case of modelling DNA sequences, all variables would take one of 4 values, A, C, G, or T. The aim is to build a probabilistic model for the samples, $P(S)$.

The distribution of samples, $P(S)$, can be written as a product of conditional probabilities:

$$\begin{aligned} P(S) &= P(S_1S_2S_3, \ldots, S_F) && (6.48) \\ &= \prod_i^F P(S_i|S_1, S_2, \ldots, S_{i-1}) && (6.49) \end{aligned}$$

A neural network is used to compute the conditional distributions,

$$P(S_i = a_{ij}|S_1, S_2, \ldots, S_{i-1}, \mathcal{H}_{NN}) = y_{ij}\left(\mathbf{w}; \mathbf{x}\right) \tag{6.50}$$

The inputs $\mathbf{x}$ are functions of the feature variables S_i such that

$$x_{ij} = \begin{cases} 1 & \text{if } S_i = a_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{6.51}$$

The parameters of the network, $\mathbf{w}$, are specified by two matrices, $\mathbf{C}$ and $\mathbf{G}$. The matrix of weights, $\mathbf{C}$, maps input vectors $\mathbf{x}$ to hidden units $\mathbf{h}$:

$$h_{ij} = \tanh\left[\sum_{k=1}^{i}\sum_{l=1}^{A_i} x_{kl}C_{klj}\right], \tag{6.52}$$

A matrix $\mathbf{G}$, maps the hidden units to outputs $\mathbf{y}$ through a softmax activation function:

$$a_{ij} = \sum_{l=1}^{i-1}\sum_{k=1}^{H} h_{lk}G_{lkj}, \tag{6.53}$$

$$y_{ij} = \frac{\exp\left(a_{ij}\right)}{\sum_{k=1}^{A_i}\exp\left(a_{ik}\right)}. \tag{6.54}$$

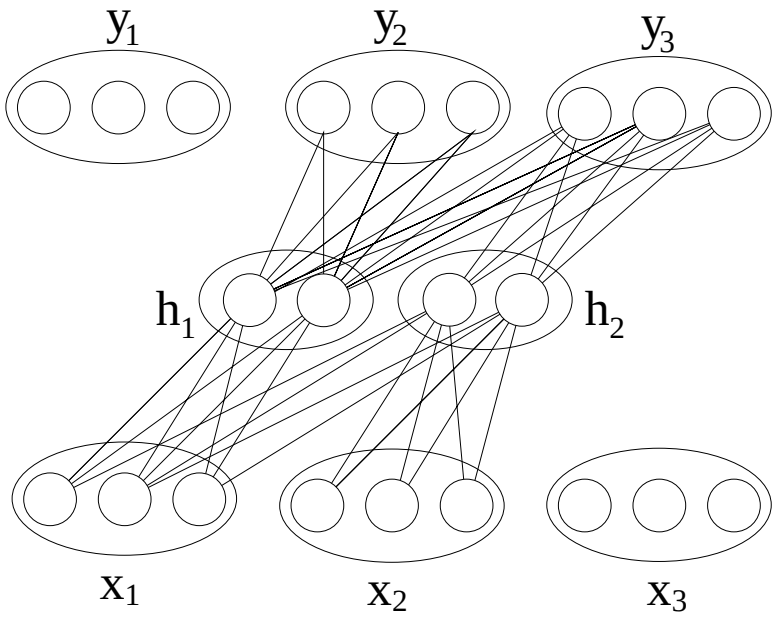


Figure 6.13: Neural network architecture for modelling fixed length categorical data

6.9.2 Inference

The samples are independent, so the likelihood is just the product of the probabilities of the individual samples:

$$P(D|\mathbf{w}, \mathcal{H}_{NN}) = \prod_{j=1}^{N} P(S^{(j)}|\mathbf{w}, \mathcal{H}_{NN}), \tag{6.55}$$

where $\mathbf{w} = \{\mathbf{C}, \mathbf{G}\}$.

Infer the parameters, given the data

$$P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H}_{NN}) = \frac{P(D|\mathbf{w}, \boldsymbol{\alpha}, \mathcal{H}_{NN})P(\mathbf{w}, \boldsymbol{\alpha}|\mathcal{H}_{NN})}{P(D|\mathcal{H}_{NN})} \tag{6.56}$$

$$= \frac{P(D|\mathbf{w}, \mathcal{H}_{NN})P(\mathbf{w}|\boldsymbol{\alpha}, \mathcal{H}_{NN})P(\boldsymbol{\alpha}|\mathcal{H}_{NN})}{P(D|\mathcal{H}_{NN})}. \tag{6.57}$$

Calculate the probability of a novel sample

To obtain the probability of a new sample, $S^{(n+1)}$, given the data D, we marginalize over the unknown parameters $\mathbf{w}$ and $\boldsymbol{\alpha}$. We used hybrid Monte Carlo as described in section 6.6 to approximate the integral with a finite sum, using samples from the posterior, $\{\mathbf{w}^{(i)}\}_{i=1}^{N}$.

$$P(S^{(n+1)}|D, \mathcal{H}) = \int d\mathbf{w}\, d\boldsymbol{\alpha} P(\mathbf{w}, \boldsymbol{\alpha}|D, \mathcal{H}_{NN})P(S^{(n+1)}|\mathbf{w}, \mathcal{H}_{NN}) \tag{6.58}$$

$$\approx \frac{1}{N} \sum_{i} P(S^{(n+1)}|\mathbf{w}^{(i)}, \mathcal{H}_{NN}) \tag{6.59}$$

6.9.3 Benchmark models

To provide a benchmark for the performance on the neural network, we used some simple models.

Uniform model

We assume a uniform distribution on each feature. Therefore, the uniform model assigns negative log probability

$$-\log P(S|\mathcal{H}_0) = \sum_{i=1}^{F} \log A_i \tag{6.60}$$

to each sample. Note that this probability is independent of S.

Naive Bayes model

A 'naive Bayes model' assumes that the distribution over features is separable,

$$P(S|\mathcal{H}_{NB}) = \prod_i P(S_i|\mathcal{H}_{NB}) \tag{6.61}$$

We estimated the probability $P(S_i)$ by maximum likelihood:

$$P(S_i = a_j|\mathcal{H}_{NB}) = \frac{F_i(a_{ij})}{\sum_k F_i(a_{ik})}, \tag{6.62}$$

where $F_i(a_{ij})$ is the number of times that feature i took the value a_{ij} in the training set.

6.9.4 Results on UCI datasets

We selected two data sets from the UCI machine learning repository. In each case, we trained a Bayesian neural network by Monte Carlo methods. The hybrid Monte Carlo method was described in section 6.7. The predictive probability was computed from equation 6.59.

Mushroom data

Each mushroom sample consists of 23 features, each taking in between 1 and 12 values. Some example data are shown in figure 6.12. The data set consisted of 8124 mushrooms; we randomized the order of the samples and split the data in half to give 4062 samples in each of the training and test sets. The results are shown in figure 6.14.

DNA data

Each DNA strand consisted of 60 features, each taking one of the four values, A, C, G, or T. We show 10 of the 60 features below.

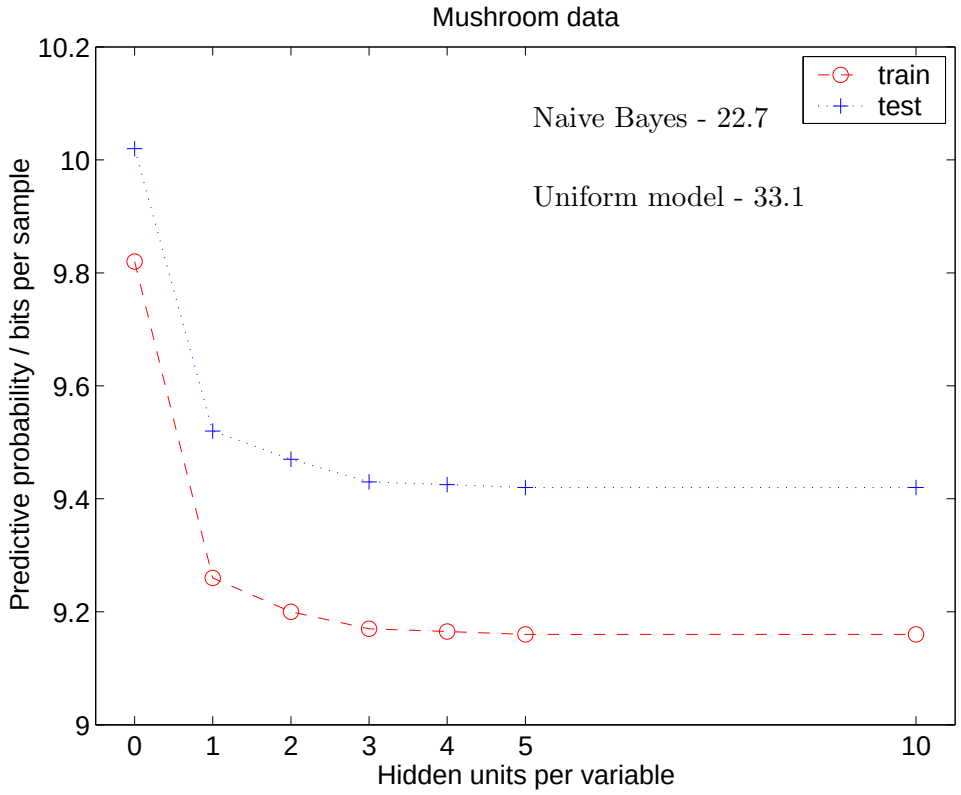


Figure 6.14: Mushroom results. Training and test results plotted. Uniform and naive Bayes figures are also shown.

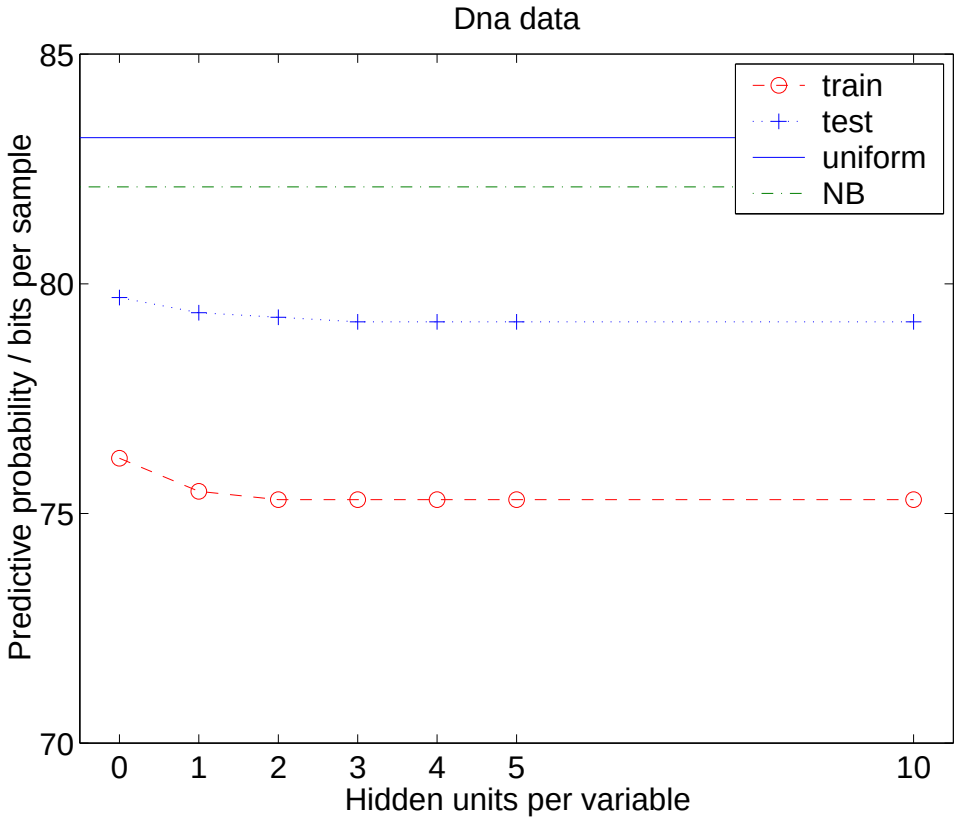


Figure 6.15: DNA results. Training and test results. Uniform and naive Bayes figures are also plotted.

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	
T	A	A	G	G	A	G	A	A	G	...
G	G	T	T	C	A	G	C	A	A	...
G	G	G	T	T	G	G	T	T	C	...
C	A	C	A	A	A	A	C	C	T	...

The data set consisted of 3176 strands; we randomized the order of the samples and split it to give 2000 in the training set and 1186 in the test set. The results are shown in figure 6.15.

6.10 Conclusion

We have shown that neural networks can be used as language models. A bigram model was used to demonstrate parameter sharing by different contexts. With small to moderate amounts of data, the bigram and trigram networks performed better than PPM, the algorithm used in state of the art text compression programs.

When using the most probable parameters for prediction (MAP), the results are highly sensitive to changes in hyperparameters. We attempted to solve the overfitting/underfitting problems by automatic relevance determination. We found that the simple implementation motivated by Gaussian approximations [60] is sensitive to the 'fudge factor'.

We have also shown that the neural network requires less storage space than conventional statistical models such as PPM. Statistical models such as PPM have to store frequency counts in all contexts that occurred.

We showed that a Bayesian approach with automatic relevance determination was more successful. However, the computational requirements prohibit the development of larger language models.

We have also shown that a neural network can be used to model fixed length sequences of discrete data. When training by Bayesian methods, we don't need to optimize number of hidden units; we just need a sufficient number to capture the dependencies of the variables.

Chapter 7

Generative Modelling

7.1 Introduction

Supervised neural networks such as multi–layer–networks are well established as probabilistic models for regression and classification, both of which are conditional modelling tasks. No modelling of the density over input variables is performed.

In generative modelling tasks, a density over all observable quantities is constructed. Previous work by MacKay [61] has shown how to use a multi–layer perceptron as a generative model for discrete data. Here, we shall use a type of Boltzmann machine to model discrete data.

7.2 The Simple Boltzmann Machine

Consider a binary vector $\mathbf{x}$ and a symmetric matrix of weights $\mathbf{w}$. We define a generative model for $\mathbf{x}$ by:

$$P(\mathbf{x}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp\left[\frac{1}{2}\mathbf{x}^T\mathbf{w}\mathbf{x}\right], \tag{7.1}$$

where

$$Z(\mathbf{w}) = \sum_{\mathbf{x}} \exp\left[\frac{1}{2}\mathbf{x}^T\mathbf{w}\mathbf{x}\right]. \tag{7.2}$$

To implement the probability distribution, we perform Gibbs sampling, where each variable draws from its conditional distribution given the state of the other variables.

Given a set of examples $\left\{\mathbf{x}^{(n)}\right\}_{n=1}^{N}$ from the real world, we might want to adjust the weights such that the generative model is well matched to those examples. A learning algorithm can be derived by calculating the posterior probability of the weights given the data:

$$P(\mathbf{w}|\{\mathbf{x}^{(n)}\}_{n=1}^{N}\}) = \frac{\left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)}|\mathbf{w})\right] P(\mathbf{w})}{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}\})}. \tag{7.3}$$

We concentrate on the first term in the numerator, the likelihood, and derive a maximum

likelihood algorithm. It is convenient to differentiate the logarithm of the likelihood:

$$\log \left[\prod_{n=1}^{N} P(\mathbf{x}^{(n)}|\mathbf{w}) \right] = \sum_{n=1}^{N} \left[\frac{1}{2} \mathbf{x}^{(n)\mathrm{T}} \mathbf{w} \mathbf{x}^{(n)} - \log Z(\mathbf{w}) \right]. \tag{7.4}$$

We differentiate with respect to w_{ij}, bearing in mind that $\mathbf{w}$ is defined to be symmetric with $w_{ji} = w_{ij}$:

$$\frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}\}|\mathbf{w}) = \sum_{n=1}^{N} \left[x_i^{(n)} x_j^{(n)} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{w})} \right] \tag{7.5}$$

$$= N \left[\langle x_i x_j \rangle_{\mathrm{Data}} - \langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{w})} \right]. \tag{7.6}$$

This gradient is thus proportional to a difference of two terms. The first term is the *empirical* correlation between x_i and x_j,

$$\langle x_i x_j \rangle_{\mathrm{Data}} \equiv \frac{1}{N} \sum_{n=1}^{N} \left[x_i^{(n)} x_j^{(n)} \right], \tag{7.7}$$

and the second term is the correlation between x_i and x_j under the current model,

$$\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{w})} \equiv \sum_{\mathbf{x}} x_i x_j P(\mathbf{x}|\mathbf{w}). \tag{7.8}$$

The first correlation $\langle x_i x_j \rangle_{\mathrm{Data}}$ is readily evaluated – it is just the empirical correlation between the activities in the real world. The second correlation, $\langle x_i x_j \rangle_{P(\mathbf{x}|\mathbf{w})}$, is not so easy to evaluate, but it can be estimated by Monte Carlo methods, that is, by observing the average value of $x_i x_j$ while the activity rule of the Boltzmann machine is iterated.

7.3 Boltzmann Machine with Hidden Units

A simple Boltzmann machine is only capable of capturing second order statistics. However, if we add hidden neurons it is possible to capture higher order features through these latent variables. We consider the architecture shown in figure 7.1, where there are no hidden-hidden or visible-visible unit connections. Boltzmann machines with this architecture are usually called restricted Boltzmann machines. We denote the states of the visible units by $\mathbf{x}$ and the states of the hidden units by $\mathbf{h}$. The generative model has the same form as a simple Boltzmann machine:

$$P(\mathbf{x}, \mathbf{h}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp\left[\mathbf{h}^{\mathrm{T}} \mathbf{w} \mathbf{x}\right], \tag{7.9}$$

where

$$Z(\mathbf{w}) = \sum_{\mathbf{x},\mathbf{h}} \exp\left[\mathbf{h}^{\mathrm{T}} \mathbf{w} \mathbf{x}\right]. \tag{7.10}$$

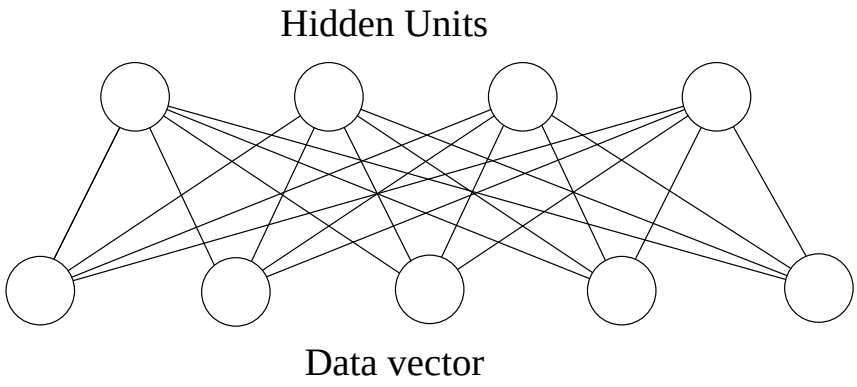


Figure 7.1: Restricted Boltzmann machine with hidden units. There are no hidden-hidden or visible-visible unit connections.

To get the generative model for $\mathbf{x}$, we marginalize over hidden units;

$$P(\mathbf{x}|\mathbf{w}) = \sum_{\mathbf{h}} P(\mathbf{x}, \mathbf{h}|\mathbf{w}) = \sum_{\mathbf{h}} \frac{1}{Z(\mathbf{w})} \exp\left[\mathbf{h}^{\mathrm{T}}\mathbf{w}\mathbf{x}\right], \tag{7.11}$$

Differentiating the log likelihood as before, we find that the derivative with respect to any weight w_{ij} is the difference between a 'waking' term and a 'sleeping' term,

$$\frac{\partial}{\partial w_{ij}} \log P(\{\mathbf{x}^{(n)}\}_1^N|\mathbf{w}) \quad = \quad \sum_n \left[\langle h_i x_j \rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{w})} - \langle h_i x_j \rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{w})}\right]. \tag{7.12}$$

The first term $\langle h_i x_j \rangle_{P(\mathbf{h}|\mathbf{x}^{(n)},\mathbf{w})}$ is the correlation between h_i and x_j if the Boltzmann machine is simulated with the visible variables clamped to $\mathbf{x}^{(n)}$ and the hidden variables freely sampling from their conditional distributions. The states of the hidden units are conditionally independent given the data, so it is possible to compute this term exactly.

The second term $\langle h_i x_j \rangle_{P(\mathbf{x},\mathbf{h}|\mathbf{w})}$ is the correlation between h_i and x_j when the Boltzmann machine generates samples from its model's distribution. This term is usually computed by Monte Carlo methods.

7.4 Products of Experts

A common way to combine several different generative models is a mixture model, where the distribution is a weighted sum of individual distributions. Combining models in this way is attractive because they can be fitted to data with the expectation-maximization (EM) algorithm [28].

However, mixture models are very inefficient in high–dimensional spaces. For example, text may be written in English or French. In both languages, the use of lower and upper case is very similar. A mixture model would require four components to capture all four

combinations of these binary attributes. However, there are only two independent degrees of freedom in this problem.

Hinton [42] showed that an alternative way to combine many individual expert models is to multiply the probabilities and renormalize. These products of experts have the advantage that they can produce sharper distributions than the individual expert models.

Consider individual expert models for which is a possible to compute the derivative of the log probability with respect to its parameters. Combine M expert models as follows.

$$P(\mathbf{d}|w_1, \ldots, w_M) = \frac{\prod_m P_m(\mathbf{d}|w_m)}{\sum_{\mathbf{c}_i} \prod_m P_m(\mathbf{c}_i|w_m)} \tag{7.13}$$

where $\mathbf{d}$ is a data vector in discrete space, w_m are all the parameters of an individual model m, and $P_m(\mathbf{d}|w_m)$ is the probability of $\mathbf{d}$ under model m.

7.4.1 Learning products of experts

To fit a product of experts, we need to compute the derivative of the log probability of each observed vector, $\mathbf{d}$, with respect to the parameters of each model.

$$\frac{\partial \log P(\mathbf{d}|w_1, \ldots, w_N)}{\partial w_m} = \frac{\partial \log P_m(\mathbf{d}|w_m)}{\partial w_m} - \sum_{\mathbf{c}} P(\mathbf{c}|w_1, \ldots, w_N) \frac{\partial \log P_m(\mathbf{c}|w_m)}{\partial w_m} \tag{7.14}$$

Assuming that each of the individual experts has a tractable derivative, the most difficult part of computing equation 7.14 is the second term. The sum over all possible data vectors is usually approximated by a Monte Carlo method:

$$\sum_{\mathbf{c}} P(\mathbf{c}|w_1, \ldots, w_N) \frac{\partial \log P_m(\mathbf{c}|w_m)}{\partial w_m} \approx \frac{1}{S} \sum_{j=1}^{S} \frac{\partial \log P_m(\mathbf{c}^{(j)}|w_m)}{\partial w_m} \tag{7.15}$$

where $\{\mathbf{c}^{(j)}\}_{j=1}^{S}$ are samples from the generative distribution, $P(\mathbf{c}|w_1, \ldots, w_N)$. For discrete data, it is possible to use rejection sampling. However, we only accept data points when all of the experts agree, so it is very inefficient. Gibbs sampling is usually more efficient; each variable draws from its posterior distribution given the state of the other variables.

7.4.2 Products of experts and Boltzmann machines

Recall the generative distribution, equation 7.11.

$$P(\mathbf{x}|\mathbf{w}) = \sum_{\mathbf{h}} \frac{1}{Z(\mathbf{w})} \exp\left[\mathbf{h}^{\mathrm{T}} \mathbf{w} \mathbf{x}\right] \tag{7.16}$$

$$= \frac{1}{Z(\mathbf{w})} \sum_{\mathbf{h}} \prod_{ij} \exp\left[h_j w_{ji} x_i\right] \tag{7.17}$$

Let $q_j = \sum_i w_{ji} x_i$, then

$$P(\mathbf{x}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \sum_{\mathbf{h}} \prod_j \exp\left(h_j q_j\right) \tag{7.18}$$

$$= \frac{1}{Z(\mathbf{w})} \prod_j \left[1 + \exp\left(q_j\right)\right] \tag{7.19}$$

$$= \frac{1}{Z(\mathbf{w})} \prod_j \left[1 + \exp\left[\sum_i w_{ji} x_i\right]\right] \tag{7.20}$$

Let p_j be the un-normalized probability distribution for what we shall call an 'expert':

$$p_j(\mathbf{x}|\mathbf{w}) = 1 + \exp\left[\sum_i w_{ji} x_i\right]. \tag{7.21}$$

We can then write the probability of a data vector in the following way:

$$P(\mathbf{x}|\mathbf{w}) = \frac{\prod_j p_j(\mathbf{x}|\mathbf{w})}{\sum_{\mathbf{x}} \prod_j p_j(\mathbf{x}|\mathbf{w})}. \tag{7.22}$$

Now it can be seen that the restricted Boltzmann machine is a product of experts.

7.4.3 The one-step learning algorithm

Hinton suggested a simple and effective alternative to maximum likelihood learning which eliminates almost all of the computation required to get samples from the equilibrium distribution. Rather than sampling from the fantasy data distribution $P(\mathbf{c}|w_1, \ldots, w_N)$, we sample over one–step reconstructions, which we denote by $\mathbf{r}$. One–step reconstructions are generated by performing Gibbs sampling on the experts, then generating data from the current states of the experts.

$$\frac{\partial \log P(\mathbf{d}|w_1, \ldots, w_N)}{\partial w_m} = \frac{\partial \log P_m(\mathbf{d}|w_m)}{\partial w_m} - \sum_{\mathbf{c}} P(\mathbf{c}|w_1, \ldots, w_N) \frac{\partial \log P_m(\mathbf{c}|w_m)}{\partial w_m} \tag{7.23}$$

$$\approx \frac{\partial \log P_m(\mathbf{d}|w_m)}{\partial w_m} - \frac{1}{S} \sum_{j=1}^{S} \frac{\partial \log P_m(\mathbf{r}^{(j)}|w_m)}{\partial w_m} \tag{7.24}$$

where $\{\mathbf{r}^{(j)}\}_{j=1}^{S}$ are one-step reconstructions. In practice Hinton has been found that just one reconstructed data point $\mathbf{r}$ does remarkably well:

$$\frac{\partial \log P(\mathbf{d}|w_1, \ldots, w_N)}{\partial w_m} \approx \frac{\partial \log P_m(\mathbf{d}|w_m)}{\partial w_m} - \frac{\partial \log P_m(\mathbf{r}|w_m)}{\partial w_m}. \tag{7.25}$$

7.5 Language Modelling with a Boltzmann Machine

The task is to model the date $D = \{t^{(n)}\}_{n=1}^{L}$, a sequence of L symbols from alphabet size $\mathcal{A}$. The restricted Boltzmann machine, described in section 7.3, has no hidden-hidden or visible-visible connections. We shall model the data with a restricted Boltzmann machine, but further restrict the values of the weights by making the weights translationally invariant.

7.5.1 The model

Each symbol is represented by a *unary code* - a binary vector of $\mathcal{A}$ bits with only one bit set to one. We can concatenate such vectors, with a single bias, to produce a single data vector. For example, the data $\mathbf{x} = (abaaba)$, with $\mathcal{A} = 2$ is represented by:

$$\mathbf{x} = (1 \;\; 10 \;\; 01 \;\; 10 \;\; 10 \;\; 01 \;\; 10) \tag{7.26}$$

We create a Boltzmann with the building block, shown in figure (figure 7.2(a)). Each line represents a connection from a visible unit to a hidden unit. The building block contains all of the free parameters of the model. There are $A \times O \times H + H$ parameters in total, including the bias on each of the hidden units. For example, the model shown in figure 7.2(a) has $(2 \times 3 \times 2 + 2) = 14$ parameters.

The full model, shown in figure 7.2(b) is created by replicating the building block along the data vector. The result is a restricted Boltzmann machine with tied weights.

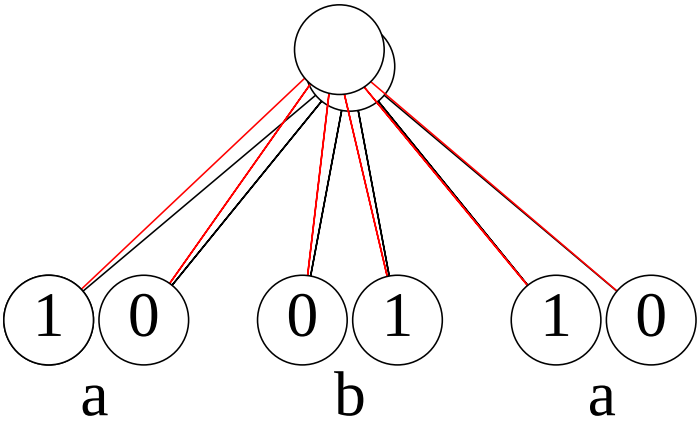
7.5.2 The probability distribution

It is useful to define all of the subsequences of length O, $\{\mathbf{y}^{(n)}\}_{n=1}^{N}$, which make up the data vector $\mathbf{x}$, of length L. We concatenate a bias and the unary codes of O data symbols in each subsequence. The data vector in equation 7.26 contains 5 subsequences:

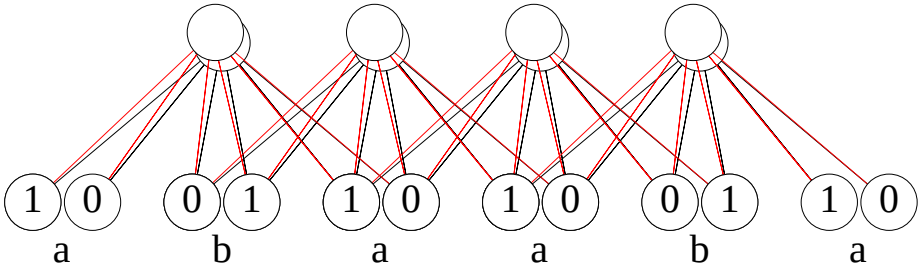
$$\begin{aligned}
\mathbf{y}^{(1)} &= (ab) = (1 \;\; 10 \;\; 01) \\
\mathbf{y}^{(2)} &= (ba) = (1 \;\; 01 \;\; 10) \\
\mathbf{y}^{(3)} &= (aa) = (1 \;\; 10 \;\; 10) \\
\mathbf{y}^{(4)} &= (ab) = (1 \;\; 10 \;\; 01) \\
\mathbf{y}^{(5)} &= (ba) = (1 \;\; 01 \;\; 10)
\end{aligned}$$

It follows that the number of subsequences is $N = L + O - 1$. Note that the subsequences are not independent. Each subsequence $\mathbf{y}^{(n)}$ has an associated hidden unit vector $\mathbf{h}^{(n)}$ of length H. The joint probability distribution of data $\mathbf{x}$ and the hidden units $\mathbf{h}$ is:

$$P(\mathbf{x}, \mathbf{h}|\mathbf{w}) = \frac{e^{\sum_{ijn} h_j^{(n)} w_{ji} y_i^{(n)}}}{Z(\mathbf{w})}, \tag{7.27}$$



(a)



(b)

Figure 7.2: (a) Building block for the language model. (b) Full model created by translation of building block. Shown with alphabet size $\mathcal{A} = 2$, order $O = 3$ and $H = 2$ hidden units.

where $Z(\mathbf{w})$ is the normalization factor

$$Z(\mathbf{w}) = \sum_{\mathbf{x},\mathbf{h}} P(\mathbf{x}, \mathbf{h}|\mathbf{w}). \tag{7.28}$$

Now, marginalising over the hidden variables

$$\begin{aligned} P(\mathbf{x}|\mathbf{w}) &= \sum_{\mathbf{h}} P(\mathbf{x}, \mathbf{h}|\mathbf{w}) && (7.29)\\ &= \sum_{\mathbf{h}} \frac{\prod_{jn} e^{\sum_i h_j^{(n)} w_{ji} y_i^{(n)}}}{Z(\mathbf{w})} && (7.30)\\ &= \frac{\prod_{jn}(1 + e^{\sum_i w_{ji} y_i^{(n)}})}{Z(\mathbf{w})} && (7.31) \end{aligned}$$

To perform learning, we will require the derivative of equation 7.31.

$$\frac{\partial \log P(\mathbf{x}|\mathbf{w})}{\partial w_{ji}} = \sum_n \left[\langle y_i^{(n)} h_j^{(n)} \rangle_{P(\mathbf{h}|\mathbf{x},\mathbf{w})} - \langle y_i^{(n)} h_j^{(n)} \rangle_{P(\mathbf{h},\mathbf{x}|\mathbf{w})} \right] \tag{7.32}$$

where $\langle y_i^{(n)} h_j^{(n)} \rangle_{P(\mathbf{h}|\mathbf{x},\mathbf{w})}$ is the expected value of $y_i^{(n)} h_j^{(n)}$ when $\mathbf{x}$ is clamped on the visible units. $\langle y_i^{(n)} h_j^{(n)} \rangle_{P(\mathbf{h},\mathbf{x}|\mathbf{w})}$ is the expected value of $y_i^{(n)} h_j^{(n)}$ with alternating Gibbs sampling.

7.5.3 Inference

Applying the one step learning algorithm

Let Q^0 be the data distribution, and Q^1 be the distribution over the one-step reconstructions of the data vectors, which are generated by one full step of Gibbs sampling:

$$\frac{\partial \log P(D|\mathbf{w})}{\partial w_{ji}} \approx \sum_n \left[\langle y_i^{(n)} h_j^{(n)} \rangle_{Q^0} - \langle y_i^{(n)} h_j^{(n)} \rangle_{Q^1} \right]. \tag{7.33}$$

As we discussed in section 7.4.3, the one-step learning algorithm works very well in practice even when a single reconstruction is used, rather than the full distribution:

$$\frac{\partial \log P(D|\mathbf{w})}{\partial w_{ji}} \approx \sum_n \left[\langle y_i^{(n)} h_j^{(n)} \rangle_{\mathbf{d}} - \langle y_i^{(n)} h_j^{(n)} \rangle_{\mathbf{r}} \right], \tag{7.34}$$

where $\mathbf{r}$ is a single reconstruction generated by sampling over hidden units and then visible units.

Training by MAP

If we know the values of the hyperparameters, $\boldsymbol{\alpha}$, then the posterior probability of the parameters $\mathbf{w}$ is

$$P(\mathbf{w}|D,\mathcal{H}) = \frac{P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\boldsymbol{\alpha},\mathcal{H})}{P(D|\mathcal{H})}. \tag{7.35}$$

Rather than learning an ensemble of plausible parameters, we choose the value of $\mathbf{w}$ that maximizes the posterior density. Equivalently, we minimize the negative log of the posterior:

$$M(\mathbf{w}) \quad = \quad -\log[P(D|\mathbf{w},\mathcal{H})P(\mathbf{w}|\boldsymbol{\alpha})]. \tag{7.36}$$

For simplicity, we assume a Gaussian prior on the parameters parameterized by a single hyperparameter, α

$$P(\mathbf{w}|\boldsymbol{\alpha}) \propto \prod_i \exp\left[-\frac{\alpha w_i^2}{2}\right] \tag{7.37}$$

Our objective function is therefore

$$M(\mathbf{w}) = -\sum_n \log P(D|\mathbf{w}) + \frac{\alpha}{2}\sum_i w_i^2 + \text{constants}. \tag{7.38}$$

The value of $\mathbf{w}$ that minimizes $M(\mathbf{w})$ is the most probable value, denoted by $\mathbf{w}_{\text{MP}}$. We shall use this optimum for classification (section 7.6) and prediction (section 7.7).

7.6 Classification

The denominator in equation 7.27, Z, is computationally hard to compute, unless H, O and $\mathcal{A}$ are very small. If we only want to compare the ratio of probabilities of two different data vectors under the model, then we do not need to compute Z.

If we have two sources of data, we learn two different models. After learning, test data are presented to both models. Each computes the un-normalized log probability or score:

$$S \quad = \quad \log\left[\prod_m P_m(\mathbf{x}|\mathbf{W})\right] \tag{7.39}$$

$$\quad = \quad \log P(\mathbf{x}|\mathbf{W}) + \log Z \tag{7.40}$$

In the next section, we show how this score can be used to perform classification on two data sets.

7.6.1 Toy language revisited

We return to the toy problem introduced in chapter 6, shown again in figure 7.3. Data from source A are generated from the original toy model. Source B is obtained by reversing data from source A. Examples of data from both sources are shown below.

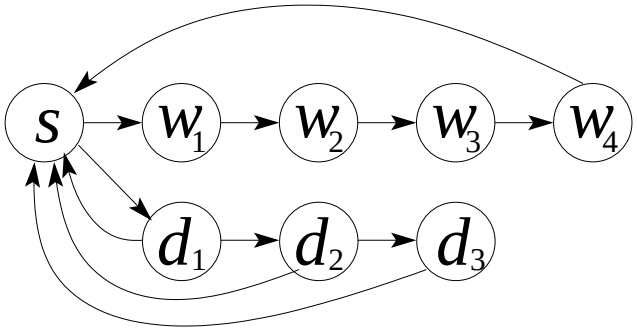


Figure 7.3: Toy problem

symbol	encoding
space	10000000000
a	01000000000
b	00100000000
c	00010000000
d	00001000000
e	00000100000
1	00000010000
2	00000001000
3	00000000100
4	00000000010
5	00000000001

Figure 7.4: Unary code for toy alphabet

```
Source A: abcc 1 3 aecc becd 5 aade bbca beee cbaa 5 accb abdd bbdd aecd
daba 422 1 bdde 5 bcaa 3 cddd 123 4 12 2 bcdb babd eadc 4 5 23 4 2 acdb

Source B: dceb ddab 2 533 3 ecbb 1 3 dccb 4 2 edcb 254 5 ecbe 34 3 33 55
23 bcbb adba dbda 2 25 2 ebcb deea bcda cdbb beba acbc 11 adbb ebde dcbe
```

The distributions of output symbols in states w_1, w_2, w_3 and w_4 are not symmetric under reversal, so it is possible to distinguish between the sources. Two models were trained separately on samples from sources A and B, each of length $L = 1000$. Each model had order $O = 5$ and $H = 5$ hidden units. The unary code for the visible units is shown in figure 7.4.

We performed gradient descent with momentum for 2000 iterations. We set the hyperparameter, α, to a value of 1. Hinton diagrams for the two trained Boltzmann machines are shown in figure 7.5.

Each model was then presented with 500 samples of length M from sources A and B. In figure 7.6 we show the log scores of the samples for a number of different sample lengths, M. For $M \geq 300$, good separation is obtained with a linear discriminator.

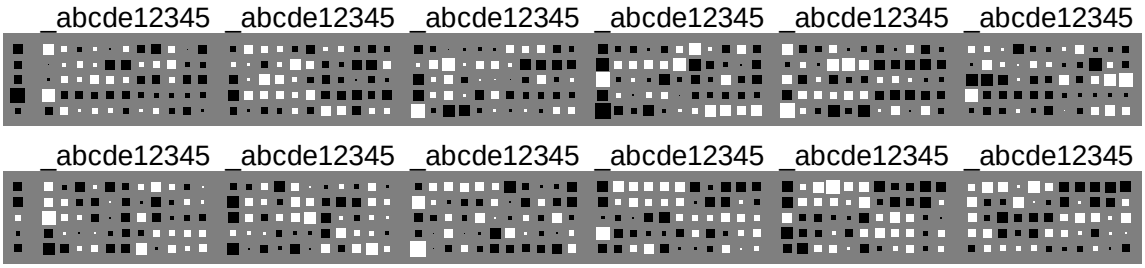


Figure 7.5: Hinton diagrams for models trained on source A (top) and source B (bottom). Each model had order $O = 6$ and $H = 6$ hidden units. Both samples were of length $L = 1000$, and α was set to 1. Each row represents a hidden unit; the biases are in the left–hand column.

7.7 Prediction Task

Given a history, we want to predict the following symbol by computing

$$P(t^{(i)}|t^{(1)}, t^{(2)}, \ldots, t^{(i-1)}). \tag{7.41}$$

It is not sufficient to create a Boltzmann machine of length i, because the variable $t^{(i)}$ depends on the values of future variables $t^{(j)}$ $(j > i)$ through hidden units. Therefore, we create a semi–infinite Boltzmann machine as shown in figure 7.7. We ignore past hidden and visible variables, which have no effect on the distribution of future variables.

We clamp the visible units to the desired context and perform alternate Gibbs sampling between the hidden and other visible units. Each time we sample the visible units, we compute $P(t^{(i)}|\mathbf{h}^{(g)})$, where g is the number of Gibbs performed so far.

Our predictive distribution is the mean of this quantity:

$$\begin{aligned} P(t^{(i)}|t^{(1)}, t^{(2)}, \ldots, t^{(i-1)}) &= P(t^{(i)}|t^{(i-1)}, t^{(i-2)}, \ldots, t^{(i-O+1)}) & (7.42) \\ &= \frac{1}{G}\sum_{g=1}^{G} P(t^{(i)}|\mathbf{h}^{(g)}), & (7.43) \end{aligned}$$

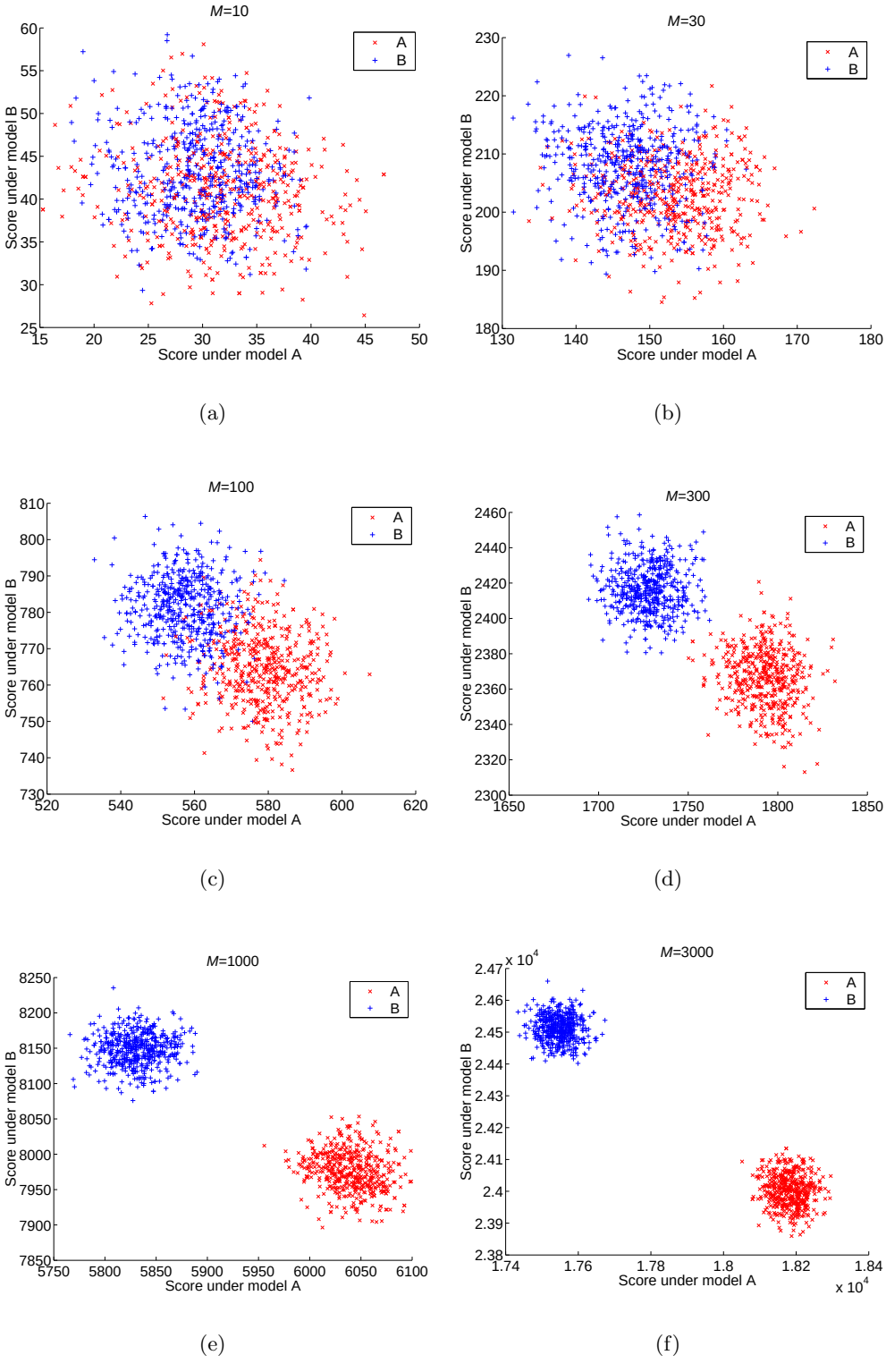
where G is the total number of Gibbs samples on the visible units.

7.7.1 Procedure

Several Boltzmann machines with $H = 5$ were trained on different amounts of data, ranging from 10 bytes to 300kB. We tried several different values of the hyperparameter, α, ranging from $\alpha = 1$ to $\alpha = 100$.

The length of the simulated Boltzmann machine

When computing the log predictive probability of the test text, we have to terminate the Boltzmann machine on the right–hand side. We aim to choose a length L large enough so



(a) (b) (c) (d) (e) (f)

Figure 7.6: Results of classification problem. Two models have been trained on samples of length $L = 1000$ from sources A and B. Scores for test data from sources A and B are shown. Each point represents a sample (of length M) from the labelled source.

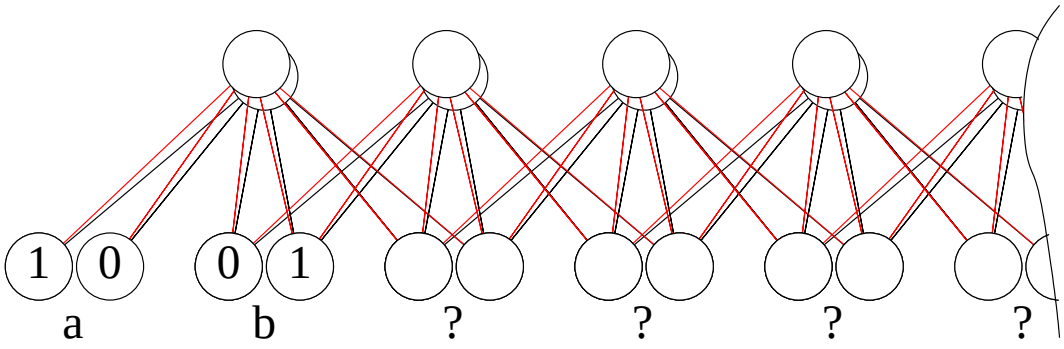


Figure 7.7: Prediction. We clamp the visible units to the desired context and perform alternate Gibbs sampling between the hidden and visible units. The context shown is '`ab`'.

that truncating the Boltzmann machine has negligible effect on the distribution of $t^{(i)}$.

We trained a single Boltzmann machine of $H = 5$ on data of length $10,000$. We then simulated a number of Boltzmann machines with different lengths and plotted the predictive distributions in figure 7.8. The graphs show that the length should be at least 15-20 visible units.

Number of Gibbs samples

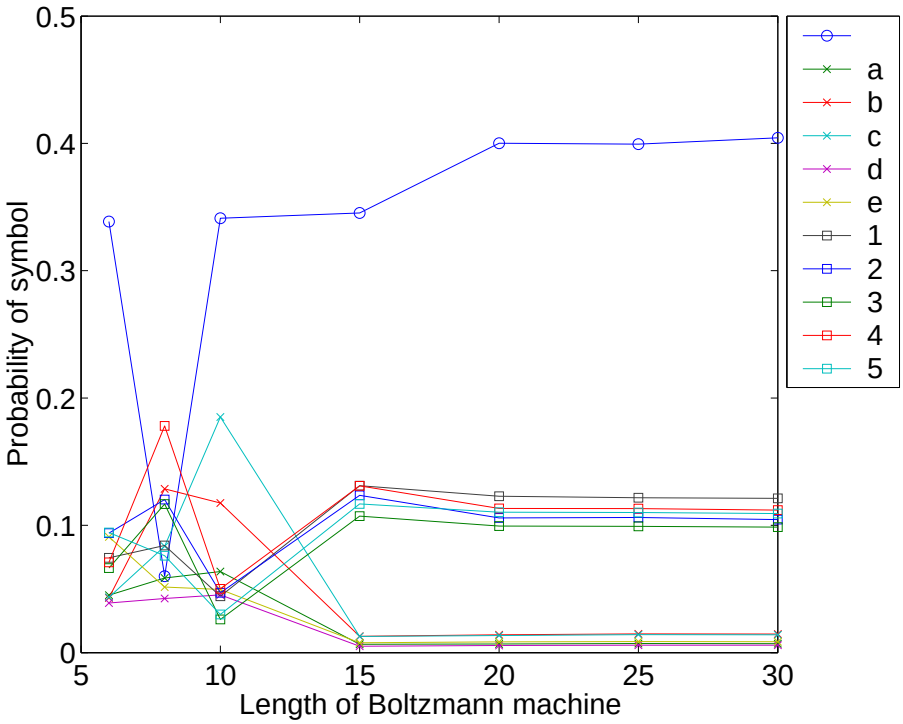
We need to choose G large enough to get a good sample from the distribution. We took the trained Boltzmann machine and varied the number of Gibbs samples. Figure 7.9 shows that the convergence is very slow.

7.7.2 Results

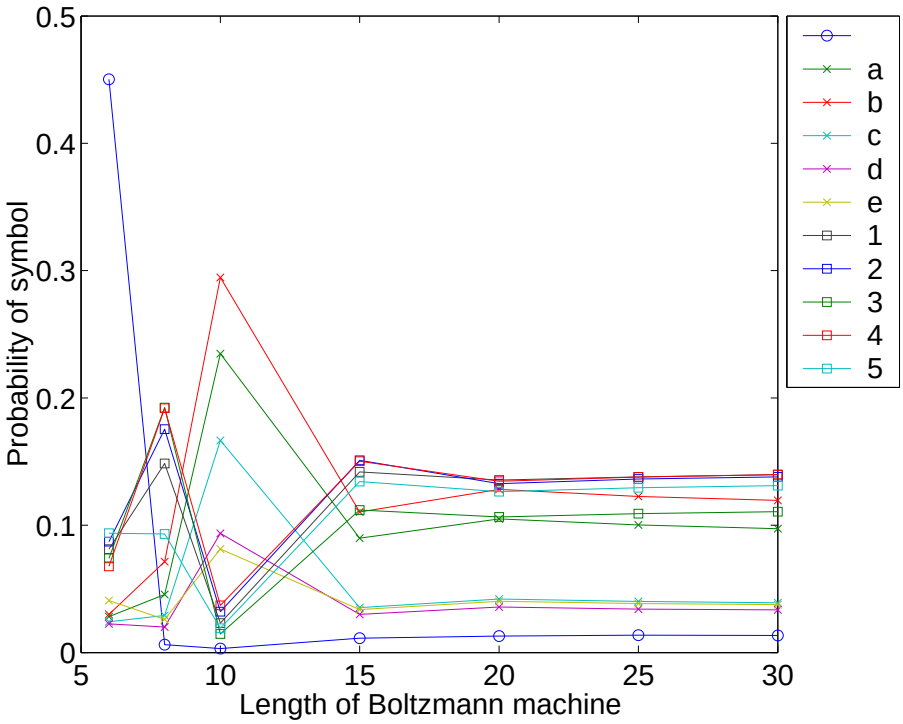
We tried several different values of the hyperparameter, α and plot the predictive probability of the test and training sets in figure 7.10. When calculating the predictive distribution, we used $G = 10^5$ and $L = 20$. For comparison, we show results for PPMD5, a model used in text compression and reviewed in section 1.5. The uniform model, H_0, which assigns equal probability to each output symbol. The entropy, calculated in section 6.8.1 is also shown. In figure 7.11, we summarize performance on the test sets.

The results show that the Boltzmann machine with $\alpha = 20$ gives the best overall performance. Although the Boltzmann machine outperforms PPM over a large range of data lengths, the performance does not appear to converge to the entropy of the source as we increase the size of the training set.

To verify that the number of hidden units was not a limiting factor, we also tried Boltzmann machines with more hidden units (figure 7.12). The results confirm that increasing the number of hidden units beyond 4 does not make a significant difference.

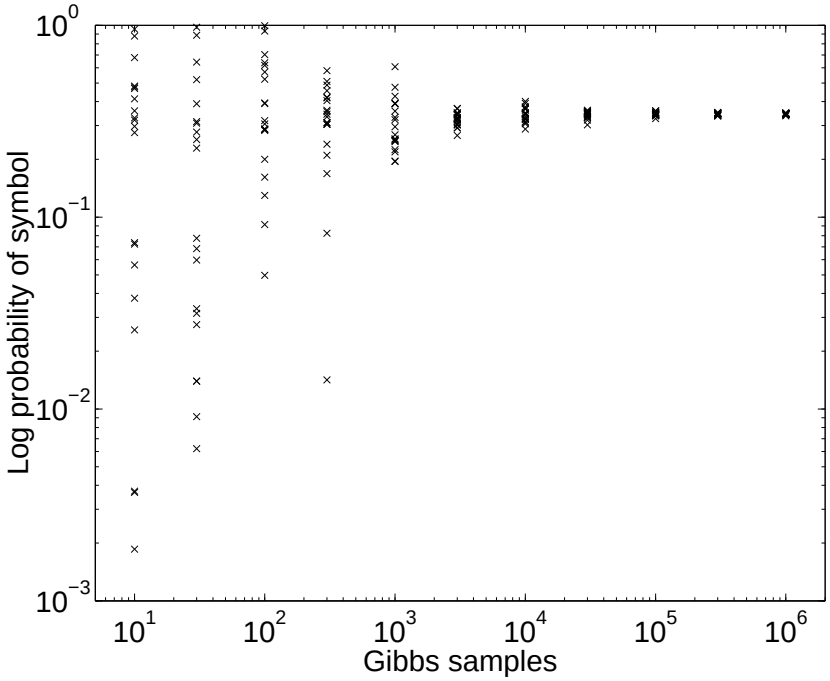


(a)

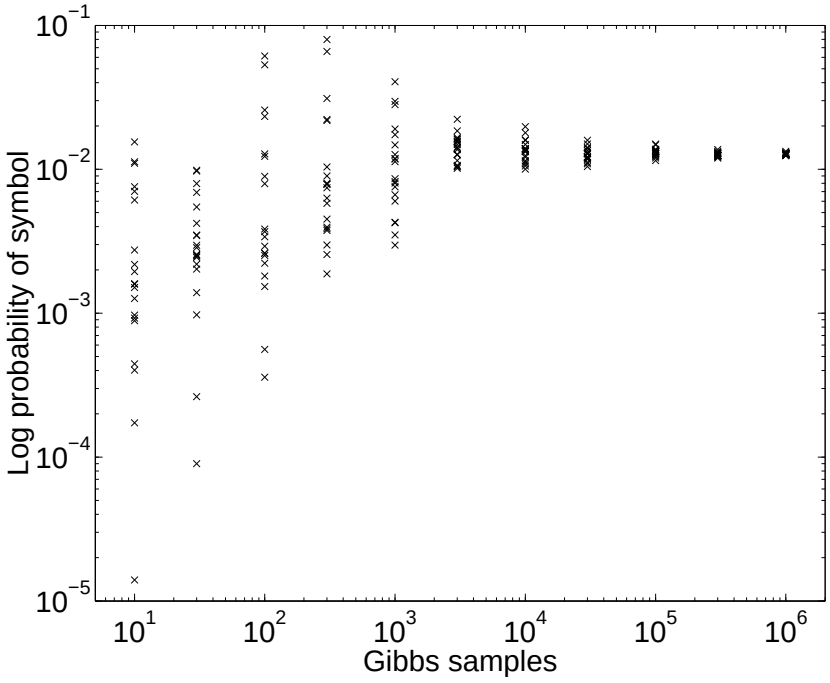


(b)

Figure 7.8: Length of the simulated Boltzmann machine. In each plot, we took a random context and plotted the probability of each symbol as a function of the length of the Boltzmann machine.

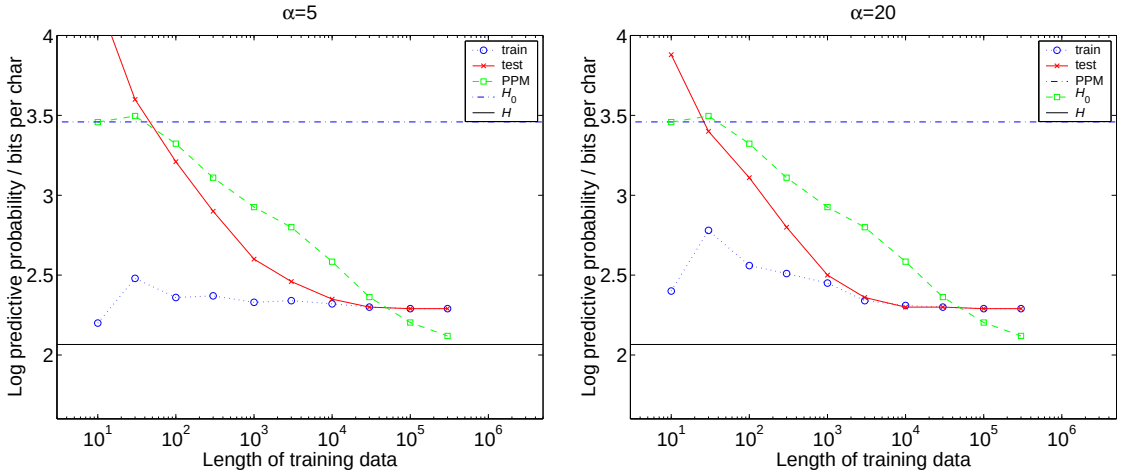


(a)

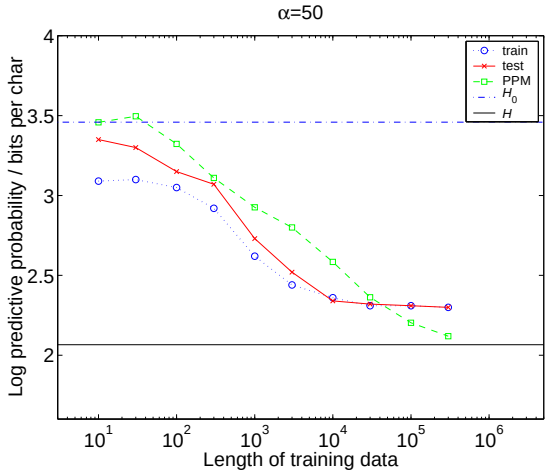


(b)

Figure 7.9: Number of Gibbs Samples. In each plot, we selected a random symbol and a random context from the test text. For each setting of the number of Gibbs samples, we computed the probability of the selected symbol 20 times, each time with a different random seed. The Boltzmann machine had $H = 5$ and was trained on text of length $10,000$.



(a) (b)



(c)

Figure 7.10: Prediction task. Log predictive probability of test and training texts of various lengths. We also show results from PPM, the uniform model H_0, and the entropy, H.

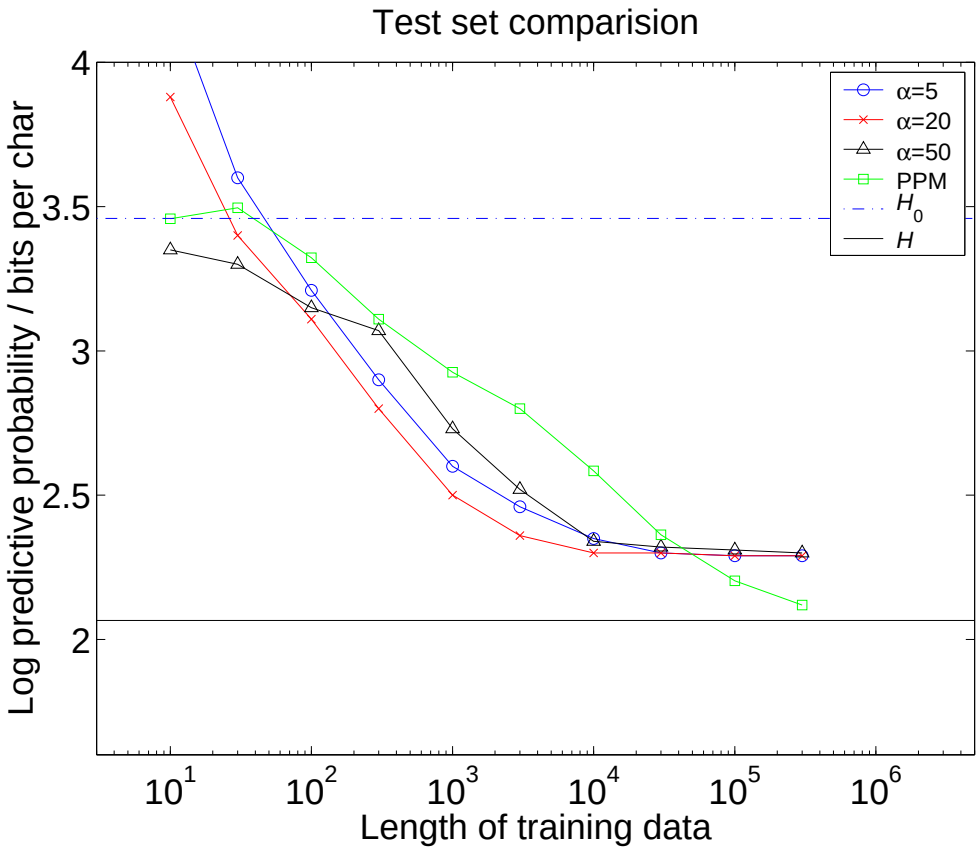


Figure 7.11: Prediction task. Log predictive probability of test texts of various lengths. We also show results from PPM, the uniform model H_0, and the entropy, H.

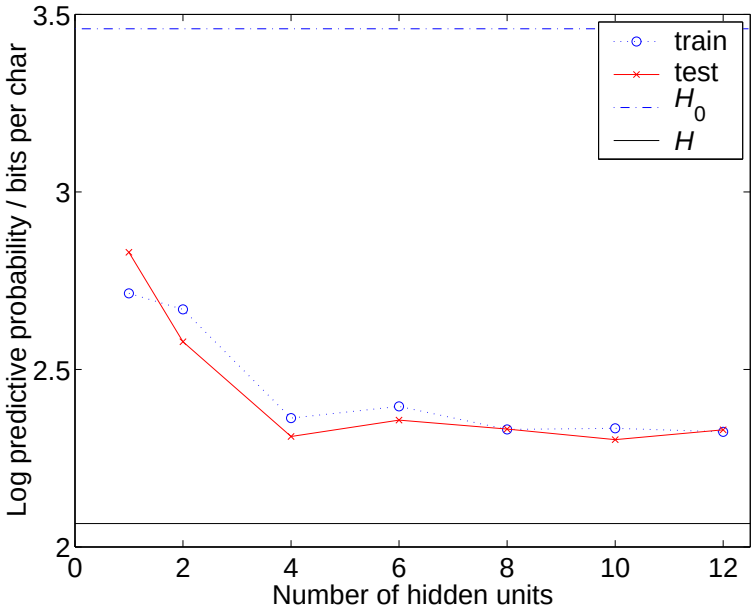


Figure 7.12: Boltzmann machines with different numbers of hidden units. The length of the training text was $L = 10,000$. The hyperparameter, $\alpha = 20$.

7.8 Conclusion

We have shown that the Boltzmann machine is capable of modelling discrete data. In a classification task, the Boltzmann machine discriminates between two similar sources.

In a prediction task, the Boltzmann machine outperforms PPM over a range of training text lengths. However, unlike PPM and the neural network in chapter 6, the performance did not approach the entropy of the toy model. We also found that prediction was computationally expensive to simulate.

Chapter 8

Conclusion

Existing devices for communicating information to computers are either bulky, slow to use, or unreliable. The conventional keyboard requires the user to make one or two gestures per character (one for lower–case and two for upper–case characters). All ten human digits are used, and a large 12–inch by 4–inch physical device is required; only one of the ten digits is used per gesture. The keyboard itself has the capacity to read about $\log_2(80) = 6.3$ bits per gesture, since each gesture is a selection of one key, and there are about 80 keys. But the entropy of English text is roughly 1 bit per character. So existing keyboards are inefficient by a factor of six.

Dasher is a novel text entry interface which exploits the information redundancy of language, and the human capacity to convey high information rates through fine motor control. The operation of Dasher is simple, and immediately evident to new users. Furthermore Dasher has a rapid learning rate that is comparable to alternative text entry methods.

Expert writing performance of 39 words per minute was obtained on the desktop version of Dasher during a dictation task. The performance of Dasher is determined by two factors. The first factor is the user interface, which can currently convey 4.8 bits per second to the computer. In this thesis, we cited an estimate of 8.2 bits per second for human pointing performance.

The second factor is how well the language model can compress the text being entered. The language model currently used in Dasher is PPM (prediction by partial match); variants of PPM have been state of the art in text compression for the last 15 years. PPM typically compresses text to about 2 bits per character. Since the entropy of English is about 1 bit per character, a perfect language model would increase the speed of Dasher by a factor of 2, which would result in a writing speed similar to that of a QWERTY keyboard.

The second half of this thesis was concerned with language modelling. It was shown that the performance of the Dirichlet language model could be improved by combining similar contexts. Feedforward neural networks can also be used to model discrete data. On a toy problem, we showed that a neural network outperforms PPM. However, computational demands prohibit the implementation of a language model with the neural network that we

described. We also demonstrated that discrete data can be modelled with generative models such as Boltzmann machines. We used Hinton's one-step learning algorithm, avoiding the usual computational cost associated with traditional methods.

We hoped to develop a better language model for Dasher, but none of these ideas gave worthwhile gains. Future directions could include improved PPM modes and maximum entropy models.

Dasher is still under development; we think it shows promise as a keyboard less text entry system both in its absolute writing speed and ease of use.

Appendix A

Compression Algorithms

A.1 ACB

Author: George Buyanovsky (April 1997). Algorithm: Associative Coding Algorithm.

A.2 ACE

Author: Marcel Lemke (April 1997). Algorithms: LZ77 + Huffman.

A.3 BOA

Author: Ian Sutton. Algorithm: PPM (Prediction by Partial Match) variant.

A.4 BRED

Author: David Wheeler. Burrows Wheeler transform [17].

A.5 BZIP2

Algorithm: Burrows Wheeler transform.

A.6 cat

This is the Unix program that copies its input to the output. It isn't a compression method but provides a benchmark of 8 bits per character. It also highlights how the compression time per character is higher for smaller files due to startup overheads, and indicates how consistent the speed measurements are (the encode and decode times should be the same).

A.7 compress

Standard Unix compress Utility, Revision 4.0 (30 July 1985).

A.8 DMC

This program implements Dynamic Markov Compression (DMC) as described in Data Compression using Dynamic Markov Modelling by Gordon Cormack and Nigel Horspool (in Computer Journal 30:6, December 1987).

A.9 gzip

gzip version 1.2.4 (18 Aug 93). Algorithms: Shannon-Fano, Huffman.

A.10 LZRW1

Author: Alistair Moffat (1990). Code from Ross Williams, comp.compression article, with local adaptations by Alistair Moffat.

A.11 Pack

Unix pack. Uses Huffman coding on a byte-by byte basis.

A.12 PKZIP

Algorithms: LZH, LZW, Shannon-Fano, Huffman.

A.13 PPMC

Author: Alistair Moffat (July 1987). Multi order character PPM text compression. Escape method C. The PPM method was originally developed John Cleary and Ian Witten.

A.14 PPMD

Author: Alistair Moffat (1993). Multi order character PPM text compression. Escape method D.

A.15 RAR

Author: Eugene Roshal (Mar 2001). Algorithms: LZ77 variant + Huffman.

A.16 RK

Author: Malcolm Taylor (Feb 1998). Algorithms: Reduced-offset LZ, PPMZ.

A.17 RKIVE

Author: Malcolm Taylor (Oct 2000). Algorithms: Reduced-offset LZ, PPMZ.

A.18 SBC

Author: Sami J. Mkinen (May 2001). Algorithms: Burrows-Wheeler Transform.

A.19 SZIP

Author: Michael Schindler (1997). szip, a block-sorting file compressor.

A.20 Winzip

Algorithms: LZH, LZW, Shannon-Fano, Huffman.

Appendix B

Probability Distributions

B.1 Gaussian Distribution

A Gaussian with mean μ and variance σ^2 has distribution

$$P(x|\mathcal{H}) = \mathcal{G}\left(x|\mu,\sigma^2\right) \tag{B.1}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{x^2}{2\sigma^2}\right]. \tag{B.2}$$

B.2 Gamma Distribution

The Gamma distribution is defined by shape parameter a, and scale parameter b.

$$P(x|\mathcal{H}) = \text{Gamma}(x|a,b) \tag{B.3}$$

$$= \frac{1}{\Gamma(a)}b^a x^{a-1}\exp\left(-bx\right) \tag{B.4}$$

The Gamma function $\Gamma(a)$ is the normalization factor of the Gamma distribution with $b = 1$,

$$\Gamma(a) = \int_0^\infty x^{a-1}\exp(-x)\,dx. \tag{B.5}$$

The Gamma distribution has mean a/b.

B.3 Dirichlet Distribution

The Dirichlet distribution [1] for a probability vector $\mathbf{p}$ with $\mathcal{A}$ components is parameterized by a measure $\mathbf{u}$ (a vector with all co-efficients $u_i > 0$) which we shall write as $u = \alpha\mathbf{m}$, where

$\mathbf{m}$ is a normalized measure over the $\mathcal{A}$ components ($\sum m_i = 1$), and α is a positive scalar:

$$\begin{aligned} P(\mathbf{p}|\alpha\mathbf{m}) &= \frac{1}{Z(\alpha\mathbf{m})} \prod_{i=1}^{\mathcal{A}} p_i^{\alpha m_i - 1} \delta\left(\textstyle\sum_i p_i - 1\right) \\ &\equiv \text{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\alpha\mathbf{m}) \end{aligned} \tag{B.6}$$

The vector $\mathbf{m}$ is the mean of the distribution. The Dirac delta function $\delta(x)$ restricts the distribution such that $\mathbf{p}$ is normalized, $\sum_i p_i = 1$.

The normalizing constant of the Dirichlet distribution is:

$$Z(\alpha\mathbf{m}) = \prod_i \Gamma(\alpha m_i) / \Gamma\left(\alpha\right). \tag{B.7}$$

The vector $\mathbf{m}$ is the mean of the probability distribution:

$$\mathbf{m} = \int \text{Dirichlet}^{(\mathcal{A})}(\mathbf{p}|\alpha\mathbf{m})\, \mathbf{p}\, d^{\mathcal{A}}\mathbf{p}. \tag{B.8}$$

Appendix C

Markov Models

C.1 Markov Chains

Let $\mathbf{s} = s_1, s_2, \ldots, s_n$ be a sequence of random variables taking their values from a finite alphabet $\mathcal{A} = a_1, a_2, \ldots, a_c$. Applying Bayes' rule

$$P(s_1, s_2, \ldots, s_n) = \prod_{i}^{N} P(s_i | s_1, s_2, \ldots, s_{i-1}) \tag{C.1}$$

The random variable are said to form a Markov chain, however, if

$$P(s_i | s_1, s_2, \ldots, s_{i-1}) = P(s_i | s_{i-1}) \text{ for all values of } i \tag{C.2}$$

As a consequence, for Markov chains,

$$P(s_1, s_2, \ldots, s_n) = \prod_{i}^{N} P(s_i | s_{i-1}) \tag{C.3}$$

C.2 Hidden Markov Models

In a Hidden Markov Model, the state sequence $\mathbf{s} = \{s_i\}$ of the Markov chain is not observable. Rather each state emits a symbol from an output alphabet.

We introduce the following formalism.

1. $\{s_i\}$, the state sequence of a hidden Markov chain.
2. $\mathcal{A} = \{a_1, a_2, \ldots, a_N\}$, state space of a Markov chain, $s_i \in \mathcal{A}$.
3. A state transition probability distribution $P(s_i = a_k | s_{i-1} = a_j) = P_{jk}$
4. $\mathbf{x} = \{x_i\}$, observation sequence.
5. $\mathcal{B} = \{b_1, b_2, \ldots, b_N\}$, output alphabet.
6. An output probability distribution associated with each state $P(x_i = b_k | s_i = a_j) = Q_{jk}$.

C.3 Entropy of an HMM

Let S be the ensemble of state sequences and X be the ensemble of the observed sequence. The *mutual information* between the S and X can be written in terms of conditional entropies.

$$H(X;S) = H(S) - H(S|X) \tag{C.4}$$
$$H(X;S) = H(X) - H(X|S) \tag{C.5}$$

Equating these two terms, we get an expression for the entropy of the output

$$H(X) = H(X|S) + H(S) - H(S|X) \tag{C.6}$$

C.3.1 Special case: $H(S|X) = 0$

We focus on the special case in which the state sequence can always be deduced from the data. In other words, $H(S|X) = 0$. From equation C.6, we have:

$$H(X) = H(X|S) + H(S) \tag{C.7}$$

We need to deduce the equilibrium distribution of the states, $P(s_i)$. The probabilities must sum to 1:

$$\sum_{i=1}^{|\mathcal{A}|} P(s_i) = 1 \tag{C.8}$$

Imposing the steady–state condition on the distribution gives $|\mathcal{A}| - 1$ independent equations:

$$P(s_i) = \sum_{j=1}^{|\mathcal{A}|} P(s_j) P_{ji} \tag{C.9}$$

Putting C.8 and C.9 together, we have $|\mathcal{A}|$ independent linear equations with $|\mathcal{A}|$ unknowns. These can be solved for $P(s_i)$ by matrix inversion. Now we proceed by calculating the the entropy of X.

$$H(X|S) = \sum_{i=1}^{|\mathcal{A}|} P(s_i) H(X_j | s_j = a_i) \tag{C.10}$$
$$= \sum_{i=1}^{|\mathcal{A}|} P(s_i) \sum_k \frac{Q_{ik}}{\log(Q_{ik})} \tag{C.11}$$

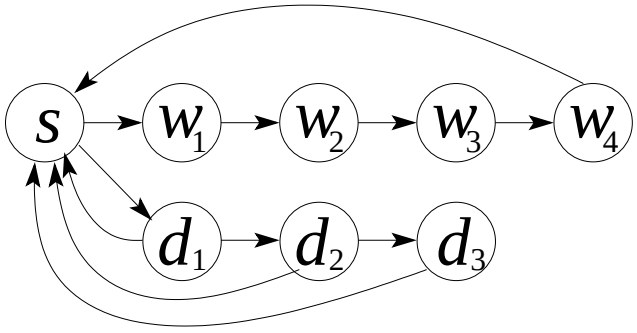


Figure C.1: State diagram. State s emits a space. States w_i emit symbols from {abcde}. States d_i emit symbols from {01234}.

$$H(S) = \sum_{i=1}^{|\mathcal{A}|} P(s_i) H(S_{j+1}|s_j = a_i) \tag{C.12}$$

$$= \sum_{i=1}^{|\mathcal{A}|} P(s_i) \sum_k \frac{P_{ik}}{\log(P_{ik})} \tag{C.13}$$

C.3.2 Demonstration

The Markov model shown in figure C defines a toy model. There are 11 symbols in the alphabet $\mathcal{B} = \{\text{space}, a, b, c, d, e, 1, 2, 3, 4, 5\}$ and each of the 8 states $\mathcal{A} = \{s, w_1, w_2, w_3, w_4, d_1, d_2, d_3\}$ outputs a symbol according to the matrix Q. Each row of Q represents a state; the entries give the probability of emitting each symbol.

$$Q = \begin{array}{c} \begin{array}{ccccccccccc} \text{space} & \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{ccccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3/9 & 3/9 & 1/9 & 1/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 3/9 & 3/9 & 1/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 1/9 & 3/9 & 3/9 & 1/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/9 & 1/9 & 1/9 & 3/9 & 3/9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{array} \right) \begin{array}{c} s \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ d_1 \\ d_2 \\ d_3 \end{array} \end{array} \tag{C.14}$$

The state transition probabilities are given by the matrix P.

$$P = \begin{array}{c} \begin{array}{cccccccc} s & w_1 & w_2 & w_3 & w_4 & d_1 & d_2 & d_3 \end{array} \\ \left(\begin{array}{cccccccc} 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \begin{array}{c} s \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ d_1 \\ d_2 \\ d_3 \end{array} \end{array} \tag{C.15}$$

We solve the simultaneous equations, C.8 and C.9, to give the steady–state distribution:

$$\{P(s_i)\} = \left(\begin{array}{cccccccc} 0.258 & 0.129 & 0.129 & 0.129 & 0.129 & 0.129 & 0.065 & 0.032 \end{array} \right). \tag{C.16}$$

Finally, we use equations C.7, C.11 and C.13 to calculate the entropy:

$$\begin{aligned} H(X|S) &= 1.62 && \text{(C.17)} \\ H(S) &= 0.45 && \text{(C.18)} \\ H(X) &= H(X|S) + H(S) && \text{(C.19)} \\ &= 2.07 \text{ bits per symbol.} && \text{(C.20)} \end{aligned}$$

Appendix D

Numerical Methods for Solving Differential Equations

Consider a differential equation of the form

$$y'(x) = f\left(y(x), x\right), \qquad y(x_0) = y_0 \tag{D.1}$$

In the following methods, we introduce a small interval of length h, and approximate $y(x)$ at a series of points x_i, such that

$$x_i = x_o + hi, \qquad i = 0, 1, 2, \ldots \tag{D.2}$$

D.1 Euler Method

The most natural interpretation of a differential equation and its solution is in terms of a distance and velocity. If x is interpreted as 'time' and $y(x)$ as the position, then $f(y(x), x)$ is the velocity.

Let y_1 denote an approximate solution at $x_1 = x_o + h$, then

$$y_i = y_{i-1} + hf(y_{i-1}, x_{i-1}), \qquad i = 1, 2, \ldots \tag{D.3}$$

It is expected that the total error is proportional to h, so this is a first–order method.

D.2 Runge-Kutta Methods

Runge-Kutta uses an approximation of f at the mid-point of the interval, instead of the left–hand end. This approximation is obtained by taking a temporary step to the desired point.

D.2.1 Trapezoidal method

$$y_i^* = y_{i-1} + hf(y_{i-1}, x_{i-1}) \tag{D.4}$$

$$y_i = y_{i-1} + \frac{h}{2}\left(f(y_{i-1}, x_{i-1}) + f(y_i^*, x_{i-1} + h)\right) \tag{D.5}$$

This approximation is second–order.

Bibliography

[1] C. E. Antoniak. Mixtures of Dirichlet processes with applications to nonparametric problems. *Annals of Statistics*, 2:1152–1174, 1974.

[2] L. R. Bahl, P. F. Brown, P. V. de Souza, and R. L. Mercer. A tree-based langauge model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008, 1989.

[3] L. R. Bahl, P.F. Brown, P.V. de Souza, R. L. Mercer, and D. Nahamoo. A fast algorithm for deleted interpolation. In *Proc. Eurospeech '91 Genoa*, pages 1209–1212, 1991.

[4] T.C. Bell, J.G. Cleary, and I.H. Witten. *Text Compression*. Prentice Hall, NJ, 1990.

[5] R. Bellman. *Dynamic Programming*. Princeton University Press, Boston, MA, 1957.

[6] Richard Bellman and Kenneth L. Cooke. *Difference Differential Equations*. Academic Press, NY, 1963.

[7] T. Bellman and I.S. MacKenzie. A probabilistic character layout strategy for mobile text entry. In *Proc. Graphics Interface '98*, pages 168–176, 1998.

[8] S. Bengio and Y. Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. *IEEE Transaction on Neural Networks special issue on data mining and knowledge discovery*, 11:550–557, 2000.

[9] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[10] C.H. Blickenstorfer. Handwriting recognition is alive and well. *Pen Computing Magazine*, pages 30–33, August 1996.

[11] C.H. Blickenstorfer. A new look at handwriting recognition. *Pen Computing Magazine*, pages 76–81, April 1997.

[12] Charles Bloom. Solving the problems of context modeling, 1998. `http://www.cbloom.com/src/ppmz.html`.

[13] Jorge Luis Borges. *The Library of Babel*. David R. Godine, 2000.

[14] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman-Soulie and J. Hérault, editors, *Neuro-computing: algorithms, architectures and applications*. Springer–Verlag, 1989.

[15] E. Brill. *A Corpus-Based Approach to Language Learning*. PhD thesis, Philadelpha, PA, 1993.

[16] Peter F. Brown, Stephen A. DellaPietra, Vincent J. DellaPietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, June 1993.

[17] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, 1994.

[18] Robert Chappell. Personal communication. 2001.

[19] S. Chen. *Building Probabilistic Models for Natural Language*. PhD thesis, Harvard University, USA., 1996.

[20] Stanley F. Chen and Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco, 1996. Morgan Kaufmann Publishers.

[21] Kenneth W. Church and Robert L. Mercer. Introduction to the special issue on computational linguistics using large corpora. *Computational Linguistics*, 19(1):1–24, 1993.

[22] J. Cleary and W. Teahan. An open interface for probabilistic models of text. In *DCC'99*. IEEE Computer Society Press, 1999.

[23] J.G. Cleary and I.H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans on Communications*, 32:396–402, 1984.

[24] R. Cole, J. Mariani, H. Uszkoreit, G. Varile, A. Zaenen, and A. Zampolli. *Survey of the State of the Art in Human Language Technology*. Cambridge University Press, 1998.

[25] T. Cover. Admissibility properties of Gilbert's encoding for unknown source probabilities. *IEEE Transaction on Information Theory*, 18(1):216–217, 1972.

[26] T. Cover and R. King. A Convergent Gambling Estimate of the Entropy of English. *IEEE Transactions on Information Theory*, 24(4):413–421, 1978.

[27] J. N. Darroch and D. Ratcliff. Generalized Iterative Scaling for Log-Linear Models. *The Annals of Mathematical Statistics*, 43:1470–1480, 1972.

[28] N. M. Dempster, A.P. Laird , and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statist. Soc. B*, 39:185–197, 1977.

[29] M.C. Detweiler, R.M. Shumacher, and N. L. Gattuso. Alphabetic input on a telephone keypad. In *Proceedings of the Human Factors Society 34th Annual Meeting*, pages 212–216, Santa Monica, 1990. Human Factors Society.

[30] L. Devroye. *Non-uniform Random Variate Generation*. Springer–Verlag, New York, 1986.

[31] W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley and Sons, New York, 1968. 3rd edition, volume 1.

[32] R. A. Fisher. *Statistical Methods for Research Workers*. Hafner, New York, 1958.

[33] L. Frey, K. White, and T. Hutchinson. Eye-gaze word processing. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:944–950, 1990.

[34] R. Garside, G. Leech, and G. Sampson. *The Computational Analysis of English*. Longman, London, 1987.

[35] E. Gilbert. Coding based on inaccurate source probabilities. *IEEE Transactions on Information Theory*, 17(3):304–314, 1971.

[36] G. Gilchrist. Archive Comparison Test, 2000. `http://www.geocities.com/SiliconValley/Park/4264/act.html`.

[37] J. Gips, P. Olivieri, and J. Tecce. Direct control of the computer through electrodes placed around the eyes. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4):630–635, 1993. In Fifth International Conference on Human-Computer Interaction, (HCI International '93).

[38] D. Goldberg and C. Richardson. Touch-typing with a stylus. In *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems.*, pages 168–176, New York, 1993. ACM.

[39] I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264, 1953.

[40] D. Gopher and D. Raij. Typing with a two-handed chord keyboard: Will the QWERTY become obsolete? *IEEE Transactions on Systems, Man, and Cybernetics*, 18:601–609, 1988.

[41] M.E. Gordon, H.G. Henry, and D.P. Massengill. Studies in typewriter keyboard modification: I. effects of amount of change, finger load, and copy content on accuracy and speed. *Journal of Applied Psychology*, 60:220–226, 1975.

[42] Geoffrey Hinton. Products of Experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks [ICANN 99]*, volume 1, pages 1–6, 1999.

[43] P. Hough. Method and means for recognizing complex patterns. US Patent 3,069,654, December 18, 1962.

[44] M. Hunter, S. Zhai, and B. A. Smith. Physics-based graphical keyboard design. In *Proceedings of CHI'2000.*, volume 2, pages 157–158, 2000.

[45] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116, 1988.

[46] C. L. James and K. M. Reischel. Text input for mobile devices: Comparing model prediction to actual performance. In *ACM Conference on Human Factors in Computing Systems (CHI 2001)*, pages 365–371, New York, 2001. ACM.

[47] Diane Jecker. Benchmark tests: Speech recognition. *PC Magazine*, November 1999.

[48] H. Jeffreys. *Theory of Probability.* Oxford Univ. Press, 1939. 3rd edition reprinted 1985.

[49] H. Jeffreys. An invariant form for the prior probability in estimation problems. *Proc. of the Royal Soc. of London A*, 186:453–454, 1946.

[50] F. Jelinek and R. L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern recognition in practice*, pages 381–402. North–Holland publishing company, 1980.

[51] Frederick Jelinek, Robert L. Mercer, and Salim Roukos. Principles of lexical language modeling for speech recognition. In Sakaoki Furui and M. Mohan Sondhi, editors, *Advances in Speech Signal Processing*, pages 651–699, New York, 1992. Marcel Decker, Inc.

[52] S. Katz. Estimation of probabilities for sparse data for the language model component of a speech recogniser. *I.E.E.E. Transactions on Acoustics, Speech and Signal Processing*, 35(3):400 – 401, March 1987.

[53] R. Kneser and H. Ney. Improved backing-off for M-gram language modeling. In *Proc. ICASSP '95*, pages 181–184, Detroit, MI, 1995.

[54] R. Krichevskii. Optimal source-coding based on observation. *Probl. Inform. Trans.*, 11(1):37–42, 1975.

[55] K.H.E. Kroemer. Performance on a prototype keyboard with ternary chorded keys. *Applied Ergonomics*, 23(2):83–90, 1992.

[56] P.-S. Laplace. *Philosophical Essay on Probabilities.* Springer-Verlag, New York, 1995. Originally published 1825.

[57] J. R. Lewis, K. M. Potosnak, and R.L Magyar. Keys and keyboards. In M. Helander, T. K. Landauer, and P. V. Prabhu, editors, *Handbook of human-computer interaction*, chapter 54. North-Holland (Elsevier), Amsterdam, 1997.

[58] G. Lidstone. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192, 1920.

[59] D. J. C. MacKay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991.

[60] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[61] D. J. C. MacKay. Density networks and their application to protein modelling. In J. Skilling and S. Sibisi, editors, *Maximum Entropy and Bayesian Methods, Cambridge 1994*, pages 259–268, Dordrecht, 1996. Kluwer.

[62] D. J. C. MacKay and L. Peto. A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(3):1–19, 1995.

[63] I. S. MacKenzie. A Note on Calculating Text Entry Speed. `http://www.yorku.ca/mack/RN-TextEntrySpeed.html`.

[64] I. S. MacKenzie, S. X. Zhang, and R. W. Soukoreff. Text entry using soft keyboards. *Behaviour & Information Technology*, 18:235–244, 1999.

[65] I.S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human Computer Interaction*, 7:91–139, 1992.

[66] I.S. MacKenzie. Movement time prediction in human-computer interfaces. In J. Grudin R. M. Baecker, W. A. S. Buxton and S. Greenberg, editors, *Readings in Human-Computer Interaction (2nd ed.)*, pages 483–493, Los Altos, CA, 1992. Kaufmann.

[67] I.S. MacKenzie and S.X. Zhang. The immediate usability of Graffiti. In *Proceedings of Graphics Interface '97*, pages 129–137, Toronto, 1997. Canadian Information Processing Society.

[68] E. Matias, I.S. MacKenzie, and W. Buxton. Half-QWERTY: A one-handed keyboard facilitating skill transfer from QWERTY. In *Proceedings of the INTERCHI '93 Conference on Human Factors in Computing Systems.*, pages 88–94, New York, 1993. ACM.

[69] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38, 1990.

[70] A. Moffat, R. M. Neal, and I. H. Witten. Arithmetic coding revisited. In J. A. Storer and M. Cohn, editors, *Proceedings of the Data Compression Conference 1995*, pages 202–111, Los Alamitos: CA, 1995. IEEE Computer Society Press.

[71] O.H. Mowrer, T.C. Ruch, and N.E. Miller. The corneo-retinal potential difference as basis of the galvanometric method of recording eye movements. *Am J Physiol*, 114:423–428, 1936.

[72] P.U. Müller, D. Cavegn, G. d'Ydewalle, and R. Groner. A comparison of a new limbus tracker, corneal reflection technique, purkinje eye tracking and electro-oculography. In G. d'Ydewalle and J. V. Rensbergen, editors, *Perception and Cognition - Advances in Eye Movement Research*, pages 393–401. Elsevier Science Publishers, 1993.

[73] R. M. Neal. Bayesian learning via stochastic dynamics. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 475–482, San Mateo, California, 1993. Morgan Kaufmann.

[74] R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Dept. of Computer Science, Univ. of Toronto, 1995.

[75] R. M. Neal. *Bayesian Learning for Neural Networks*. Number 118 in Lecture Notes in Statistics. Springer, New York, 1996.

[76] A. Newell and P.S. Rosenbloom. Mechanisms of skill acquisition and the power law of learning. In J. R. Anderson, editor, *Cognitive Skills and their Acquisition*, pages 1–55. L. Erlbaum Associates, NJ, 1981.

[77] D.A. Norman and D. Fisher. Why alphabetic keyboards are not easy to use: Keyboard layout doesn't much matter. *Human Factors*, 24:509–519, 1982.

[78] W. Perks. Some observations on inverse probability. *Journal of the Institute of Actuaries*, 73:285–312, 1947.

[79] K. Perlin. Quikwriting: continuous stylus-based text entry. In *Proceedings, ACM Symposium on User Interface Software and Technology*, pages 215–216, 1998.

[80] D. A. Rosenbaum. *Human Motor Control*. Academic Press, San Diego, CA, 1991.

[81] R. Rosenfeld. A Maximum Entropy Approach to Adaptive Statistical Language Modelling. *Computer, Speech and Language*, 10:187–288, 1996.

[82] D. D. Salcucci and J. R. Anderson. Intelligent gaze-added interface. In *Proceedings of CHI 2000*, pages 273–280, New York, 2000. ACM.

[83] H. Schmid. Part-of-speech tagging with neural networks. In *COLING'94*, pages 172–176, 1994.

[84] D. Scott and J. M. Findlay. Visual search, eye movements and display units. Technical report, before 1993.

[85] A. Sears, D. Revis, J. Swatski, R. Crittenden, and B. Shneiderman. Investigating touchscreen typing: the effect of keyboard size on typing speed. *Behaviour and Information Technology*, 12:17–22, 1993.

[86] C. E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, January 1951.

[87] C. E. Shannon. *Collected Papers*. IEEE Press, New York, 1993. Edited by N. J. A. Sloane and A. D. Wyner.

[88] F. Shein, G. Hamann, N. Brownlow, J. Treviranus, M. M. Milner, and P. Parnes. WiViK: A Visual Keyboard for Windows 3.0. In *In Proceedings of the 14th Annual Conference of RESNA*, pages 160–162, 1991.

[89] Linda E. Sibert and Robert J.K. Jacob. Evaluation of Eye Gaze Interaction. In *ACM Conference on Human Factors in Computing Systems (CHI 2000)*, pages 281–288, 2000.

[90] M. Silfverberg, I. S. MacKenzie, and P. Korhonen. Predicting text entry speed on mobile phones. In *Proceedings of CHI 2000*, pages 9–16, 2000.

[91] D. Stampe and E. Reingold. Selection by looking: A novel computer interface and its application to psychological research. In J.M. Findlay, R. Walker, and R. W. Kentridge, editors, *Eye Movement Research: Mechanisms, Processes, and Applications*, pages 467–478. Elsevier Science Publishing, New York, 1995.

[92] W. Teahan. *Modelling English Text*. PhD thesis, Univ. of Waikato, N.Z., 1997.

[93] W.J. Teahan. Probability estimation for PPM. In *Proceedings NZCSRSC'95*, 1995. Available from `http://www.cs.waikato.ac.nz/~wjt/ papers/NZCSRSC.ps.gz`.

[94] H. Tilbourg. *An introduction to cryptology*. Kluwer Academic Publishers, 1988.

[95] D. Venolia and F. Neiberg. T-Cube: A fast, self-disclosing pen-based alphabet. In *Proceedings CHI '94*, pages 265–270, 1994.

[96] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–269, 1967.

[97] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay. Dasher - A Data Entry Interface Using Continuous Gestures and Language Models. In *Proceedings of UIST 2000*, pages 129–137, 2000.

[98] Colin Ware and Harutune H. Mikaelian. An evaluation of an eye tracker as a device for computer input. In J. M. Carroll and P. P. Tanner, editors, *Proceedings of Human Factors in Computing Systems and Graphics Interface '87*, pages 183–188, 1987.

[99] I. Witten and T. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37:1085–1094, 1991.

[100] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann, San Francisco, CA, second edition, 1999.

[101] S. Zhai and B.A. Smith. Alphabetically biased virtual keyboards are easier to use - layout does matter. In *ACM Conference on Human Factors in Computing Systems (CHI 2001)*, pages 321–322, New York, 2001. ACM Press.

[102] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, IT-24:530–536, 1978.

[103] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.